DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

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

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

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

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

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

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

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

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

Notes on the Essay Data

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

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

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

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

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

In [1]:

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

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

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

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

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

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

from tqdm import tqdm
import os

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

!pip install -U -q PyDrive
```

```
done
```

In [2]:

```
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
# Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

# links to google drive
link='https://drive.google.com/open?id=18VAiuw3vfETGcuJOdicvkgQT0pSxF7Wy'
link3='https://drive.google.com/open?id=1Z6bjXmyCaoEzXYo_tRDwLTsfeA2F3K3j'
flufff, id2 = link3.split('=')
print (id2) # Verify that you have everything after '='
downloaded = drive.CreateFile({'id':id2})
downloaded.GetContentFile('glove_vectors')
```

```
1Z6bjXmyCaoEzXYo_tRDwLTsfeA2F3K3j
```

1.1 Reading Data

In [3]:

```
fluff, id = link.split('=')
print (id) # Verify that you have everything after '='


# for project data
downloaded = drive.CreateFile({'id':id})
downloaded.GetContentFile('train_data.csv')
project_data = pd.read_csv('train_data.csv',nrows=50000)

print(project_data.shape)
```

```
#------------------------------------------------------------------------------------------------------


link1='https://drive.google.com/open?id=11uHEj9KOgWD9SU-CPgKyb6VrWqVos4uV'
print('\n-----------------------------------------------------------------')

# for resource data
fluff1, idi = link1.split('=')
print (idi) # Verify that you have everything after '='
downloaded = drive.CreateFile({'id':idi})
downloaded.GetContentFile('resources .csv')
resource_data = pd.read_csv('resources .csv')


print(resource_data .head(3))
```

```
18VAiuw3vfETGcuJOdicvkgQT0pSxF7Wy
(50000, 17)

-----------------------------------------------------------------
11uHEj9KOgWD9SU-CPgKyb6VrWqVos4uV
        id                                          description  quantity   price
0  p233245  LC652 - Lakeshore Double-Space Mobile Drying Rack         1  149.00
1  p069063         Bouncy Bands for Desks (Blue support pipes)         3   14.95
2  p069063  Cory Stories: A Kid's Book About Living With Adhd         1    8.45
```

In [4]:

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

print(resource_data.shape)
print(resource_data.columns.values)
```

```
Number of data points in train data (50000, 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']
(1541272, 4)
['id' 'description' 'quantity' 'price']
```

In [5]:

```
#sort the datapoints by date  <-

# 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)# we drop the col
project_data.sort_values(by=['Date'], inplace=True)# sort the values y date


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


project_data.head(2)
```

Out[5]:

Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_s

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_s
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2	
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Grades 3-5	L

1.3 Text preprocessing

In [0]:

```
# 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 [7]:

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

Out[7]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_s
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2	
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Grades 3-5	L

In [0]:

```
# 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 [0]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\
```

```
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "d
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

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

Preprocessing of project_subject_categories

In [0]:

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

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

from collections import Counter
```

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

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

Preprocessing of project_subject_subcategories

In [0]:

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

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

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

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

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

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

Preprocessing of project_grade_category

In [12]:

```
print(project_data['project_grade_category'][:3])#   we have to remove the graddes from every row
```

```
473      Grades PreK-2
41558       Grades 3-5
29891       Grades 3-5
Name: project_grade_category, dtype: object
```

In [0]:

```
d= list(project_data['project_grade_category'].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
```

```
grade_cat_list = []
for i in d:
    # consider we have text like this:
    for j in i.split(' '): #    # split by spae
        j=j.replace('Grades','')# clean grades from the row
    grade_cat_list.append(j.strip())



project_data['clean_grade'] = grade_cat_list
project_data.drop(['project_grade_category'], 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_grade'].values:
    my_counter.update(word.split())

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

Assignment 7: Support Vector Machines(SVM)

1. **[Task-1] Apply Support Vector Machines(SGDClassifier with hinge loss: Linear SVM) on these feature sets**
    - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_eassay (BOW)
    - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)
    - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)
    - Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)
2. **The hyper paramter tuning (best alpha in range [10^-4 to 10^4], and the best penalty among 'l1', 'l2')**
    - Find the best hyper parameter which will give the maximum AUC value
    - Find the best hyper paramter using k-fold cross validation or simple cross validation data
    - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning
3. **Representation of results**
    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure.
    - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
    - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.
4. **[Task-2] Apply the Support Vector Machines on these features by finding the best hyper paramter as suggested in step 2 and step 3**
    - Consider these set of features Set 5 :
        - **school_state** : categorical data
        - **clean_categories** : categorical data
        - **clean_subcategories** : categorical data
        - **project_grade_category** :categorical data
        - **teacher_prefix** : categorical data
        - **quantity** : numerical data
        - **teacher_number_of_previously_posted_projects** : numerical data
        - **price** : numerical data
        - **sentiment score's of each of the essay** : numerical data
        - **number of words in the title** : numerical data
        - **number of words in the combine essays** : numerical data
        - **Apply TruncatedSVD on TfidfVectorizer of essay text, choose the number of components (`n_components`) using elbow method** : numerical data
    - **Conclusion**
        - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

please refer to this prettytable library.....

Note: Data Leakage

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

2. Preparing our data for the models

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

In [0]:

```
#Splitting Data into train and Test sklearn https://scikit-
learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(project_data,
                                                    project_data['project_is_approved'],
                                                      stratify=  project_data['project_is_approved'],
                                                      test_size = 0.33
                                                     )
```

In [0]:

```
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,  stratify= y_train,
                                                test_size = 0.33)
```

In [16]:

```
print(y_train.value_counts())
print(y_test.value_counts())
print(y_cv.value_counts())
# huge imbalance
```

```
1    18982
0     3463
Name: project_is_approved, dtype: int64
1    13954
0     2546
Name: project_is_approved, dtype: int64
1    9350
0    1705
Name: project_is_approved, dtype: int64
```

In [0]:

```
#droping the y labels
#https://stackoverflow.com/questions/13411544/delete-column-from-pandas-dataframe-by-column-name
#x_train =
X_train.drop(["project_is_approved"], axis = 1, inplace = True)
#x_test =
X_test.drop(["project_is_approved"], axis = 1, inplace = True)
#x_cv =
X_cv.drop(["project_is_approved"], axis = 1, inplace = True)
```

Text preprocessing of train,test and cv

In [ ]:

In [18]:

```
#Proprocessing for essay
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays_train = []
# tqdm is for printing the status bar
for sentance in tqdm(X_train['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_train.append(sent.lower().strip())
```

```
100%|██████████| 22445/22445 [00:11<00:00, 1917.29it/s]
```

In [19]:

```
#Proprocessing for essay
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays_test = []
# tqdm is for printing the status bar
for sentance in tqdm(X_test['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_test.append(sent.lower().strip())
```

```
100%|██████████| 16500/16500 [00:08<00:00, 1910.00it/s]
```

In [20]:

```
#Proprocessing for essay
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays_cv = []
# tqdm is for printing the status bar
for sentance in tqdm(X_cv['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_cv.append(sent.lower().strip())
```

```
100%|██████████| 11055/11055 [00:05<00:00, 1877.71it/s]
```

In [21]:

```
#Proprocessing for essay
# Combining all the above stundents
from tqdm import tqdm
preprocessed_titles_cv = []
# tqdm is for printing the status bar
for sentance in tqdm(X_cv['project_title'].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_titles_cv.append(sent.lower().strip())
```

```
100%|██████████| 11055/11055 [00:00<00:00, 33827.33it/s]
```

In [22]:

```
#Proprocessing for essay
# Combining all the above stundents
from tqdm import tqdm
preprocessed_titles_train = []
# tqdm is for printing the status bar
for sentance in tqdm(X_train['project_title'].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_titles_train.append(sent.lower().strip())
```

```
100%|██████████| 22445/22445 [00:00<00:00, 34045.76it/s]
```

In [23]:

```
#Proprocessing for essay
# Combining all the above stundents
from tqdm import tqdm
preprocessed_titles_test = []
# tqdm is for printing the status bar
for sentance in tqdm(X_test['project_title'].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_titles_test.append(sent.lower().strip())
```

```
100%|██████████| 16500/16500 [00:00<00:00, 34258.32it/s]
```

2.2 Make Data Model Ready: encoding numerical, categorical features

1. vectorize categorical data

1.*project*_subject_categories convert categorical to vectors*

In [24]:

```
# convert train,cv and test data of clean_categories into vectors


# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer1 = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False,
binary=True)
vectorizer1.fit(X_train['clean_categories'].values)

# firstly convert fit the train data into the vectoriaer then it learn hte vocablery

# we use the fitted CountVectorizer to convert the text to vector
X_train_cat = vectorizer1.transform(X_train['clean_categories'].values)
X_cv_cat = vectorizer1.transform(X_cv['clean_categories'].values)
X_test_cat = vectorizer1.transform(X_test['clean_categories'].values)
```

```
print(vectorizer1.get_feature_names())
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
```

In [25]:

```
print("After vectorizations")
print(X_train_cat.shape, y_train.shape)
print(X_cv_cat.shape, y_cv.shape)
print(X_test_cat.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 9) (22445,)
(11055, 9) (11055,)
(16500, 9) (16500,)
====================================================================================================
```

2.*project*_subject_subcategories convert categorical to vectors*

In [26]:

```
# convert train,cv and test data of clean_categories into vectors


# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer2 = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary
=True)
vectorizer2.fit(X_train['clean_subcategories'].values)

# firstly convert fit the train data into the vectoriaer then it learn hte vocablery

# we use the fitted CountVectorizer to convert the text to vector
X_train_subcat = vectorizer2.transform(X_train['clean_subcategories'].values)
X_cv_subcat = vectorizer2.transform(X_cv['clean_subcategories'].values)
X_test_subcat = vectorizer2.transform(X_test['clean_subcategories'].values)

print(vectorizer2.get_feature_names())
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL
', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
```

In [27]:

```
print("After vectorizations")
print(X_train_subcat.shape, y_train.shape)
print(X_cv_subcat.shape, y_cv.shape)
print(X_test_subcat.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 30) (22445,)
(11055, 30) (11055,)
(16500, 30) (16500,)
====================================================================================================
```

*3 school_state convert categorical to vectors**

In [28]:

```
# now time to cont the each words
from collections import Counter
my_counter = Counter()
for word in project_data['school_state'].values:
    my_counter.update(word.split())# count the words

school_state_dict = dict(my_counter)# store in dicionary
sorted_school_state_dict = dict(sorted(school_state_dict.items(), key=lambda kv: kv[1]))# sor it
print(sorted_school_state_dict)
```

```
{'VT': 32, 'WY': 51, 'ND': 63, 'MT': 106, 'RI': 126, 'NH': 141, 'SD': 142, 'NE': 144, 'AK': 153,
'DE': 155, 'WV': 218, 'ME': 222, 'NM': 236, 'HI': 239, 'DC': 247, 'KS': 285, 'ID': 302, 'IA': 306,
'AR': 446, 'CO': 538, 'MN': 556, 'OR': 577, 'MS': 598, 'KY': 614, 'NV': 665, 'MD': 668, 'TN': 774,
'CT': 774, 'AL': 790, 'UT': 792, 'WI': 833, 'VA': 916, 'AZ': 994, 'NJ': 1005, 'OK': 1074, 'MA': 107
6, 'LA': 1094, 'WA': 1103, 'MO': 1166, 'IN': 1171, 'OH': 1180, 'PA': 1419, 'MI': 1468, 'GA': 1828,
'SC': 1830, 'IL': 1967, 'NC': 2340, 'FL': 2839, 'TX': 3320, 'NY': 3393, 'CA': 7024}
```

In [29]:

```
# convert train,cv and test data of clean_categories into vectors


# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer3 = CountVectorizer(vocabulary=list(sorted_school_state_dict.keys()), lowercase=False, b
inary=True)
vectorizer3.fit(project_data['school_state'].values)

# firstly convert fit the train data into the vectoriaer then it learn hte vocablery

# we use the fitted CountVectorizer to convert the text to vector
X_train_school_state = vectorizer3.transform(X_train['school_state'].values)
X_cv_school_state = vectorizer3.transform(X_cv['school_state'].values)
X_test_school_state = vectorizer3.transform(X_test['school_state'].values)

print(vectorizer3.get_feature_names())
```

```
['VT', 'WY', 'ND', 'MT', 'RI', 'NH', 'SD', 'NE', 'AK', 'DE', 'WV', 'ME', 'NM', 'HI', 'DC', 'KS', 'I
D', 'IA', 'AR', 'CO', 'MN', 'OR', 'MS', 'KY', 'NV', 'MD', 'TN', 'CT', 'AL', 'UT', 'WI', 'VA', 'AZ',
'NJ', 'OK', 'MA', 'LA', 'WA', 'MO', 'IN', 'OH', 'PA', 'MI', 'GA', 'SC', 'IL', 'NC', 'FL', 'TX', 'NY
', 'CA']
```

In [30]:

```
print("After vectorizations")
print(X_train_school_state .shape, y_train.shape)
print(X_cv_school_state .shape, y_cv.shape)
print(X_test_school_state .shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 51) (22445,)
(11055, 51) (11055,)
(16500, 51) (16500,)
====================================================================================================
```

In [31]:

```
#https://stackoverflow.com/questions/42224700/attributeerror-float-object-has-no-attribute-split
project_data['clean_grade']=project_data['clean_grade'].fillna("")# fill the nulll values with
space

# convert train,cv and test data of clean_categories into vectors
```

```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer4 = CountVectorizer(vocabulary=list(sorted_project_grade_category_dict.keys()),
lowercase=False, binary=True)
vectorizer4.fit(project_data['clean_grade'].values)

# firstly convert fit the train data into the vectoriaer then it learn hte vocablery

# we use the fitted CountVectorizer to convert the text to vector
X_train_project_grade_category = vectorizer4.transform(X_train['clean_grade'].values)
X_cv_project_grade_category = vectorizer4.transform(X_cv['clean_grade'].values)
X_test_project_grade_category = vectorizer4.transform(X_test['clean_grade'].values)

print(vectorizer4.get_feature_names())
```

```
['9-12', '6-8', '3-5', 'PreK-2']
```

In [32]:

```
print("After vectorizations")
print(X_train_project_grade_category  .shape, y_train.shape)
print(X_cv_project_grade_category  .shape, y_cv.shape)
print(X_test_project_grade_category  .shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 4) (22445,)
(11055, 4) (11055,)
(16500, 4) (16500,)
====================================================================================================
```

In [0]:

```
#https://stackoverflow.com/questions/42224700/attributeerror-float-object-has-no-attribute-split
project_data['teacher_prefix']=project_data['teacher_prefix'].fillna(" ")# filll the null values
with space
my_counter = Counter()
for word in project_data['teacher_prefix'].values:
    my_counter.update(word.split())


# dict sort by value python: https://stackoverflow.com/a/613218/4084039
teacher_cat_dict = dict(my_counter)
sorted_teacher_prefix_dict = dict(sorted(teacher_cat_dict.items(), key=lambda kv: kv[1]))
```

In [34]:

```
# convert train,cv and test data of clean_categories into vectors


# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer5 = CountVectorizer(vocabulary=list(sorted_teacher_prefix_dict.keys()), lowercase=False,
binary=True)
vectorizer5.fit(project_data['teacher_prefix'].values.astype('U'))

# firstly convert fit the train data into the vectoriaer then it learn hte vocablery

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_prefix = vectorizer5.transform(X_train['teacher_prefix'].values.astype('U'))
X_cv_teacher_prefix= vectorizer5.transform(X_cv['teacher_prefix'].values.astype('U'))
X_test_teacher_prefix = vectorizer5.transform(X_test['teacher_prefix'].values.astype('U'))

print(vectorizer5.get_feature_names())
```

```
# when i executeed this error comes
#np.nan is an invalid document, expected byte or unicode string.
# then iconvert to unicode     just writ .astype('U') after the .values in fit and trainform
#https://stackoverflow.com/questions/39303912/tfidfvectorizer-in-scikit-learn-valueerror-np-nan-is
-an-invalid-document
```

```
['Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
```

In [35]:

```
print("After vectorizations")
print(X_train_teacher_prefix   .shape, y_train.shape)
print(X_cv_teacher_prefix   .shape, y_cv.shape)
print(X_test_teacher_prefix   .shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 5) (22445,)
(11055, 5) (11055,)
(16500, 5) (16500,)
====================================================================================================
```

2.3 Make Data Model Ready: encoding eassay, and project_title

Apply Baw featurezation *Title*

In [ ]:

```
X_train_essay=preprocessed_essays_train
X_cv_essay=preprocessed_essays_cv
X_test_essay=preprocessed_essays_test


X_train_title=preprocessed_titles_train
X_cv_title=preprocessed_titles_cv
X_test_title=preprocessed_titles_test

# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer6 = CountVectorizer(min_df=10,max_features=5000,ngram_range=(1, 2))# its a countvectors u
sed for convert text to vectors
vectorizer6.fit(X_train_essay)# that is learned from trainned  data



# we use the fitted CountVectorizer to convert the text to vector
X_train_bow = vectorizer6.transform(X_train_essay)
X_cv_bow = vectorizer6.transform(X_cv_essay)
X_test_bow = vectorizer6.transform(X_test_essay)



# print("After vectorizations")
# print(X_train_bow.shape, y_train.shape)
# print(X_cv_bow.shape, y_cv.shape)
# print(X_test_bow.shape, y_test.shape)
# print("="*100)
```

Apply Baw featurezation *essay*

In [37]:

```
vectorizer7 = CountVectorizer(min_df=10,max_features=5000,ngram_range=(1, 2))
vectorizer7.fit(X_train_title)# that is learned from trainned  data
```

```
vectorizer7.fit(X_train_title)# that is learned from trainned  data



# we use the fitted CountVectorizer to convert the text to vector
X_train_bow_title = vectorizer7.transform(X_train_title)
X_cv_bow_title= vectorizer7.transform(X_cv_title)
X_test_bow_title = vectorizer7.transform(X_test_title)



print("After vectorizations")
print(X_train_bow_title.shape, y_train.shape)
print(X_cv_bow_title.shape, y_cv.shape)
print(X_test_bow_title.shape, y_test.shape)
print("="*100)
# so the dimension of alll are the same by using first fit and then transform
```

```
After vectorizations
(22445, 1603) (22445,)
(11055, 1603) (11055,)
(16500, 1603) (16500,)
====================================================================================================
```

Applly tf-idf featureization *titles*

In [38]:

```
#for titles
from sklearn.feature_extraction.text import TfidfVectorizer
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer8 = TfidfVectorizer(min_df=10,max_features=5000,ngram_range=(1, 2))# its a countvectors u
sed for convert text to vectors
vectorizer8.fit(X_train_title)# that is learned from trainned  data


# we use the fitted CountVectorizer to convert the text to vector
X_train_tf_title = vectorizer8.transform(X_train_title)
X_cv_tf_title= vectorizer8.transform(X_cv_title)
X_test_tf_title = vectorizer8.transform(X_test_title)



print("After vectorizations")
print(X_train_tf_title.shape, y_train.shape)
print(X_cv_tf_title.shape, y_cv.shape)
print(X_test_tf_title.shape, y_test.shape)
print("="*100)
# so the dimension of alll are the same by using first fit and then transform
```

```
After vectorizations
(22445, 1603) (22445,)
(11055, 1603) (11055,)
(16500, 1603) (16500,)
====================================================================================================
```

Applly tf-idf featureization *Essays*

In [39]:

```
#for essay
from sklearn.feature_extraction.text import TfidfVectorizer
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer9 = TfidfVectorizer(min_df=10,max_features=5000,ngram_range=(1, 2))# its a countvectors u
sed for convert text to vectors
```

```
vectorizer9.fit(X_train_essay)# that is learned from trainned  data



# we use the fitted CountVectorizer to convert the text to vector
X_train_tf_essay = vectorizer9.transform(X_train_essay)
X_cv_tf_essay= vectorizer9.transform(X_cv_essay)
X_test_tf_essay = vectorizer9.transform(X_test_essay)



print("After vectorizations")
print(X_train_tf_essay.shape, y_train.shape)
print(X_cv_tf_essay.shape, y_cv.shape)
print(X_test_tf_essay.shape, y_test.shape)
print("="*100)
# so the dimension of alll are the same by using first fit and then transform
```

```
After vectorizations
(22445, 5000) (22445,)
(11055, 5000) (11055,)
(16500, 5000) (16500,)
====================================================================================================
```

Using Pretrained Models: Avg W2V

In [0]:

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

In [0]:

```
#for essay
# average Word2Vec
# compute average word2vec for each review.
def func(wordlist):


  train_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
  for sentence in tqdm(wordlist): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length    # we are taking the 300
dimensions  very large
    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
    train_avg_w2v_vectors.append(vector)

  print(len(train_avg_w2v_vectors))
  print(len(train_avg_w2v_vectors[0]))
  return train_avg_w2v_vectors
```

In [42]:

```
train_avg_w2v_vectors=func(preprocessed_essays_train)
test_avg_w2v_vectors=func(preprocessed_essays_test)
cv_avg_w2v_vectors=func(preprocessed_essays_cv)
#for titles

cv_avg_w2v_vectors_title=func(preprocessed_titles_cv)
test_avg_w2v_vectors_title=func(preprocessed_titles_test)
train_avg_w2v_vectors_title=func(preprocessed_titles_train)
```

```
100%|██████████| 22445/22445 [00:05<00:00, 4049.95it/s]
  3%|▎         | 417/16500 [00:00<00:03, 4168.17it/s]
```

```
22445
300
```

```
100%|██████████| 16500/16500 [00:04<00:00, 4024.96it/s]
  4%|▍         | 400/11055 [00:00<00:02, 3994.66it/s]
```

```
16500
300
```

```
100%|██████████| 11055/11055 [00:02<00:00, 4002.83it/s]
100%|██████████| 11055/11055 [00:00<00:00, 70270.14it/s]
  0%|          | 0/16500 [00:00<?, ?it/s]
```

```
11055
300
11055
300
```

```
100%|██████████| 16500/16500 [00:00<00:00, 70858.14it/s]
 34%|███▍      | 7737/22445 [00:00<00:00, 77364.91it/s]
```

```
16500
300
```

```
100%|██████████| 22445/22445 [00:00<00:00, 75916.98it/s]
```

```
22445
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [0]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_essays_train)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [0]:

```
# average Word2Vec
# compute average word2vec for each review.
def tf_idf_done(word_list):

  train_title_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this
list
  for sentence in tqdm(word_list): # 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():#.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    train_title_tfidf_w2v_vectors.append(vector)
```

```
  print(len(train_title_tfidf_w2v_vectors))
  print(len(train_title_tfidf_w2v_vectors[0]))
  return train_title_tfidf_w2v_vectors
```

In [45]:

```
#train_title_tfidf_w2v_vectors=tf_idf_done(tf_idf_train_title)
#train_title_tfidf_w2v_vector
train_tfidf_w2v_vectors=tf_idf_done(preprocessed_essays_train)
test_tfidf_w2v_vectors=tf_idf_done(preprocessed_essays_test)
cv_tfidf_w2v_vectors=tf_idf_done(preprocessed_essays_cv)

#train_title_tfidf_w2v_vectors=tf_idf_done(tf_idf_train_title)
#train_title_tfidf_w2v_v   cdaaector
train_title_tfidf_w2v_vectors=tf_idf_done(preprocessed_titles_train)
test_title_tfidf_w2v_vectors=tf_idf_done(preprocessed_titles_test)
cv_title_tfidf_w2v_vectors=tf_idf_done(preprocessed_titles_cv)
```

```
100%|██████████| 22445/22445 [00:35<00:00, 629.78it/s]
  0%|          | 70/16500 [00:00<00:23, 696.62it/s]

22445
300

100%|██████████| 16500/16500 [00:25<00:00, 653.55it/s]
  1%|          | 58/11055 [00:00<00:18, 579.92it/s]

16500
300

100%|██████████| 11055/11055 [00:16<00:00, 653.20it/s]
 16%|█▌        | 3591/22445 [00:00<00:00, 35905.84it/s]

11055
300

100%|██████████| 22445/22445 [00:00<00:00, 35996.37it/s]
 15%|█▍        | 2474/16500 [00:00<00:00, 24734.89it/s]

22445
300

100%|██████████| 16500/16500 [00:00<00:00, 31334.89it/s]
 32%|███▏      | 3483/11055 [00:00<00:00, 34817.66it/s]

16500
300

100%|██████████| 11055/11055 [00:00<00:00, 34017.70it/s]

11055
300
```

1.5.3 Vectorizing Numerical features¶

In [46]:

```
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')
print(price_data.head(2))


# we also have to do this in tran test and cv
```

```
# we also have to do this in train,test and cv
# so also merge the resource data with the trian,cv and test

X_train = pd.merge(X_train, price_data, on = "id", how = "left")
#print(x_train.columns)
X_test = pd.merge(X_test, price_data, on = "id", how = "left")
X_cv = pd.merge(X_cv, price_data, on = "id", how = "left")
```

```
        id   price  quantity
0  p000001  459.56         7
1  p000002  515.89        21
```

Standadized price for the train,test and cv

In [0]:

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


price_scalar =  StandardScaler()

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

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



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



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

Stadadized Previous_year_tecaher_projects train,test and cv

In [0]:

```
# previous_year_projects
price_scalar.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1)) # fi
nding the mean and standard deviation of this data
#print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
train_prev_proj_standar =
price_scalar.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,
1))
# Now standardize the data with above maen and variance.
test_prev_proj_standar =
price_scalar.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1)
)
# Now standardize the data with above maen and variance.
cv_prev_proj_standar = price_scalar.transform(X_cv['teacher_number_of_previously_posted_projects']
.values.reshape(-1, 1))
```

Standaized the Quantity column of the train,test and cv

In [0]:

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

# Now standardize the data with above maen and variance.
train_qnty_standar = price_scalar.transform(X_train['quantity'].values.reshape(-1, 1))

# Now standardize the data with above maen and variance.
cv_qnty_standar = price_scalar.transform(X_cv['quantity'].values.reshape(-1, 1))


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

Merge all features whchh we clean till now**

Prepare for set 1:

In [50]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_set1_train = hstack((X_train_bow_title,X_train_bow,# all bows
                       X_train_teacher_prefix,X_train_cat,X_train_subcat
,X_train_project_grade_category,X_train_school_state,#all categorials
                       train_qnty_standar,train_price_standar,train_prev_proj_standar))# all
numericals


print(X_set1_train.shape, y_train.shape)
```

```
(22445, 6705) (22445,)
```

In [51]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_set1_cv = hstack((X_cv_bow_title,X_cv_bow,
                       X_cv_teacher_prefix,X_cv_cat,X_cv_subcat,
                       X_cv_project_grade_category,X_cv_school_state,
cv_qnty_standar,cv_price_standar,cv_prev_proj_standar))


print(X_set1_cv.shape, y_cv.shape)
```

```
(11055, 6705) (11055,)
```

In [52]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_set1_test = hstack((X_test_bow_title,X_test_bow,
                       X_test_teacher_prefix,X_test_cat,X_test_subcat,
                       X_test_project_grade_category,X_test_school_state,
                       test_qnty_standar,test_price_standar,test_prev_proj_standar))


print(X_set1_test.shape, y_test.shape)
```

```
(16500, 6705) (16500,)
```

Prepare for set 2:

In [53]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_set2_train = hstack((X_train_tf_essay,X_train_tf_title,
                       X_train_teacher_prefix,X_train_cat,X_train_subcat,
                       X_train_project_grade_category,X_train_school_state,
                        train_qnty_standar,train_price_standar,train_prev_proj_standar))


print(X_set2_train.shape, y_train.shape)
```

```
(22445, 6705) (22445,)
```

In [54]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_set2_cv = hstack((X_cv_tf_essay,X_cv_tf_title,
                       X_cv_teacher_prefix,X_cv_cat,X_cv_subcat,
                       X_cv_project_grade_category,X_cv_school_state,
                    cv_qnty_standar,cv_price_standar,cv_prev_proj_standar))


print(X_set2_cv.shape, y_cv.shape)
```

```
(11055, 6705) (11055,)
```

In [55]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_set2_test = hstack((X_test_tf_essay,X_test_tf_title,
                       X_test_teacher_prefix,X_test_cat,X_test_subcat,
                       X_test_project_grade_category,X_test_school_state,
                      test_qnty_standar,test_price_standar,test_prev_proj_standar))


print(X_set2_test.shape, y_test.shape)
```

```
(16500, 6705) (16500,)
```

Prepare for set 3:

In [56]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_set3_train = hstack((train_avg_w2v_vectors,train_avg_w2v_vectors_title,train_prev_proj_standar,t
rain_price_standar,train_qnty_standar,
                       X_train_teacher_prefix,X_train_cat,X_train_subcat,
                       X_train_project_grade_category,X_train_school_state))


print(X_set3_train.shape, y_train.shape)
```

```
(22445, 702) (22445,)
```

In [57]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_set3_cv =
```

```
hstack((cv_avg_w2v_vectors,cv_avg_w2v_vectors_title,cv_prev_proj_standar,cv_price_standar,cv_qnty_s
tandar,
                       X_cv_teacher_prefix,X_cv_cat,X_cv_subcat,
                       X_cv_project_grade_category,X_cv_school_state))


print(X_set3_cv.shape, y_cv.shape)
```

```
(11055, 702) (11055,)
```

In [58]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_set3_test =
hstack((test_avg_w2v_vectors,test_avg_w2v_vectors_title,test_prev_proj_standar,test_price_standar,
test_qnty_standar,
                       X_test_teacher_prefix,X_test_cat,X_test_subcat,
                       X_test_project_grade_category,X_test_school_state))


print(X_set3_test.shape, y_test.shape)
```

```
(16500, 702) (16500,)
```

Prepare for set 4:

In [59]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_set4_train =
hstack((train_tfidf_w2v_vectors,train_title_tfidf_w2v_vectors,train_prev_proj_standar,train_price_s
tandar,train_qnty_standar,
                       X_train_teacher_prefix,X_train_cat,X_train_subcat,
                       X_train_project_grade_category,X_train_school_state))


print(X_set4_train.shape, y_train.shape)
```

```
(22445, 702) (22445,)
```

In [60]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_set4_cv =
hstack((cv_tfidf_w2v_vectors,cv_title_tfidf_w2v_vectors,cv_prev_proj_standar,cv_price_standar,cv_q
nty_standar,
                       X_cv_teacher_prefix,X_cv_cat,X_cv_subcat,
                       X_cv_project_grade_category,X_cv_school_state))


print(X_set4_cv.shape, y_cv.shape)
```

```
(11055, 702) (11055,)
```

In [61]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_set4_test = hstack((test_title_tfidf_w2v_vectors,test_tfidf_w2v_vectors,test_prev_proj_standar,t
est_price_standar,test_qnty_standar,
                       X_test_teacher_prefix,X_test_cat,X_test_subcat,
                       X_test_project_grade_category,X_test_school_state))


print(X_set4_test.shape, y_test.shape)
```

```
(16500, 702) (16500,)
```

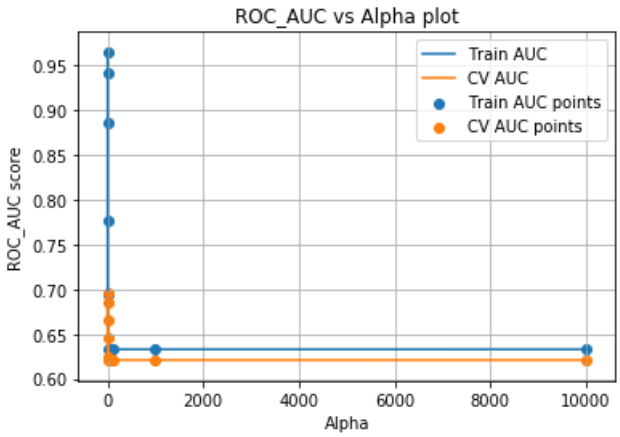
Applying. SVM section

2.4.1 Applying SVM on BOW, SET 1

we are using "l2" Regulrizer

In [63]:

```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
#from sklearn.datasets import *
from sklearn import linear_model
from sklearn.linear_model import SGDClassifier
from sklearn import svm
# hyperparameter tuning  with l2 reg
parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}

sd = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced')
classifier = GridSearchCV(sd, parameters, cv= 5, scoring='roc_auc',return_train_score=True)

classifier.fit(X_set1_train, y_train)

train_auc = classifier.cv_results_['mean_train_score']
cv_auc= classifier.cv_results_['mean_test_score']



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

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


plt.legend()
plt.xlabel("Alpha")
plt.ylabel("ROC_AUC score")
plt.title("ROC_AUC vs Alpha plot")
plt.grid()
plt.show()
```

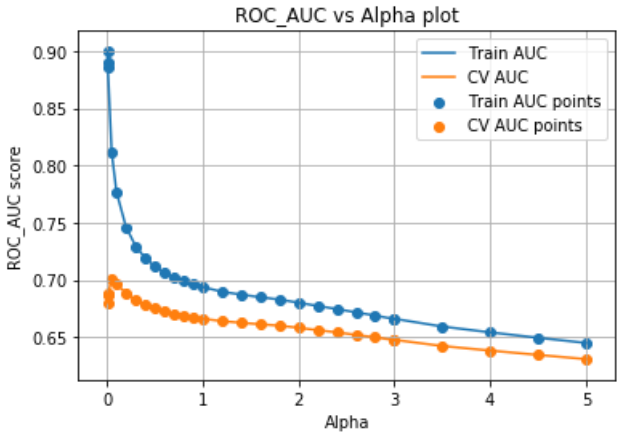


In Above plot, we can't able to find the best Hyperparamter Now i am reducing the range of 'ALPHA' for rest of my models

In [64]:

```
# hyperparameter tuning  with l2 reg
parameters = {'alpha':[0.007,0.009,0.01,0.05,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,1.2,1.4,1.6,1.8
```

```
parameters = {'alpha':[0.007,0.009,0.01,0.05,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,1.2,1.4,1.6,1.8,
2,2.2,2.4,2.6,2.8,3,3,3.5,4,4.5,5]}
sd = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced')
classifier = GridSearchCV(sd, parameters, cv= 5, scoring='roc_auc',return_train_score=True)

classifier.fit(X_set1_train, y_train)


train_auc = classifier.cv_results_['mean_train_score']
cv_auc= classifier.cv_results_['mean_test_score']



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

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




plt.legend()
plt.xlabel("Alpha")
plt.ylabel("ROC_AUC score")
plt.title("ROC_AUC vs Alpha plot")
plt.grid()
plt.show()
```

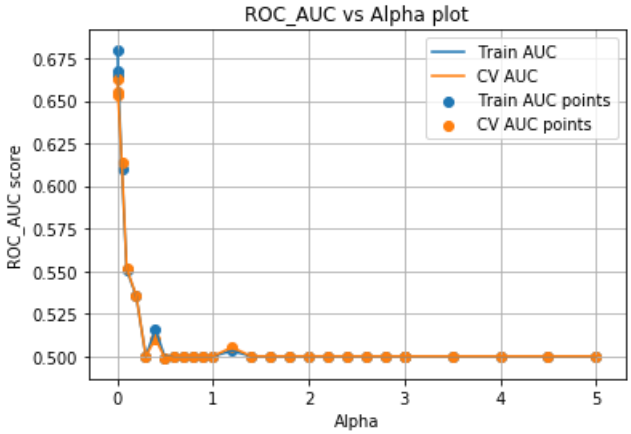


With l1 Regularization

In [65]:

```
# hyperparameter tuning  with l1 reg
parameters = {'alpha':[0.007,0.009,0.01,0.05,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,1.2,1.4,1.6,1.8,
2,2.2,2.4,2.6,2.8,3,3,3.5,4,4.5,5]}
sd = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced')
classifier = GridSearchCV(sd, parameters, cv= 5, scoring='roc_auc',return_train_score=True)

classifier.fit(X_set1_train, y_train)


train_auc = classifier.cv_results_['mean_train_score']
cv_auc= classifier.cv_results_['mean_test_score']



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

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




plt.legend()
plt.xlabel("Alpha")
```

```
plt.ylabel("ROC_AUC score")
plt.title("ROC_AUC vs Alpha plot")
plt.grid()
plt.show()
```

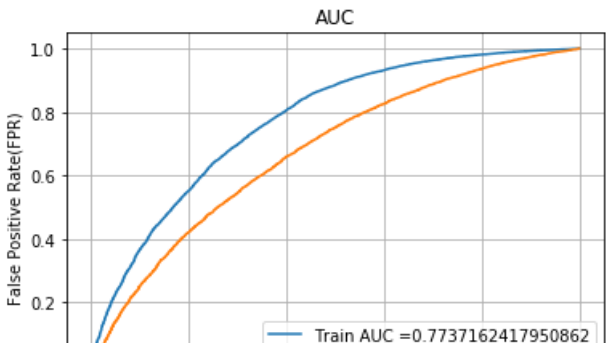


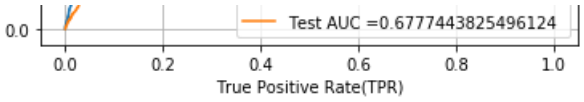
Observation: l2 regularizatin works better than l1 and best alpha is 0.01.

Fitting Model to Hyper-Parameter Curve

In [0]:

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


Classifier_bow = SGDClassifier(loss = 'hinge', penalty = 'l2', alpha = 0.01)
Classifier_bow.fit(X_set1_train ,y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

#https://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_model.SGl
sifier.decision_function

y_train_pred = Classifier_bow.decision_function(X_set1_train)
y_test_pred = Classifier_bow.decision_function(X_set1_test)

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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```





Confusion Matrix¶

In [0]:

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

    t = threshould[np.argmax(fpr*(1-tpr))]
    print("the maximum value of tpr*(1-fpr)", np.round(max(tpr*(1-fpr)),2) , "for threshold", np.ro
und(t,2))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

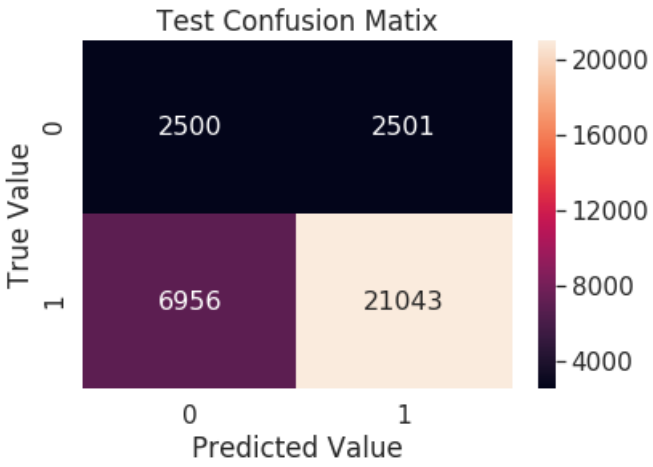
In [0]:

```
import seaborn as sea
#for test dara
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred, te_thresholds,
                                                                      test_fpr,test_fpr)),
range(2),range(2))
sea.set(font_scale=1.4)
sea.heatmap(train_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = 'd')
plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Test Confusion Matix")
```

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 1.05
```

Out[0]:

```
Text(0.5, 1.0, 'Test Confusion Matix')
```



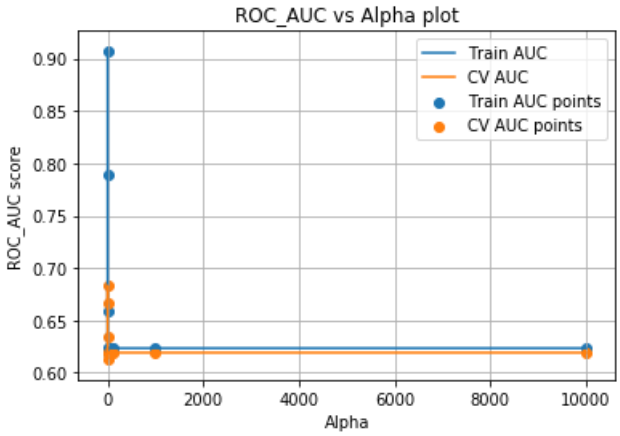
2.4.2 Applying SVM on TFIDF, SET 2

We use l2 regularizer

In [66]:

```
# hyperparameter tuning  with l2 reg
parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}
sd = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced')
classifier = GridSearchCV(sd, parameters, cv= 5, scoring='roc_auc',return_train_score=True)
```

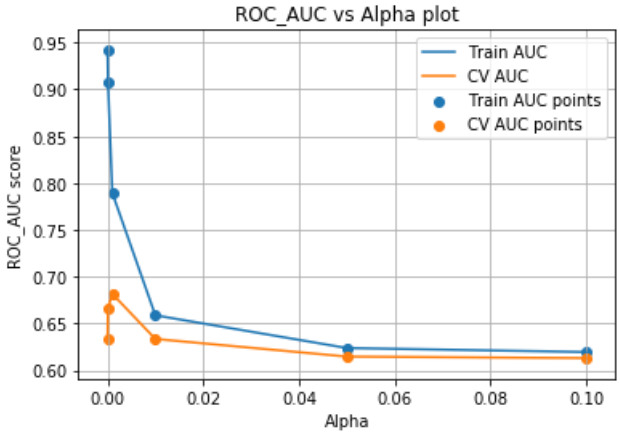
```
classifier.fit(X_set2_train, y_train)

#0.00001, 0.0001, 0.001,0.01, 0.05, 0.1
train_auc = classifier.cv_results_['mean_train_score']
cv_auc= classifier.cv_results_['mean_test_score']



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

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




plt.legend()
plt.xlabel("Alpha")
plt.ylabel("ROC_AUC score")
plt.title("ROC_AUC vs Alpha plot")
plt.grid()
plt.show()
```



Try by reducing the ranges of alpha

In [67]:

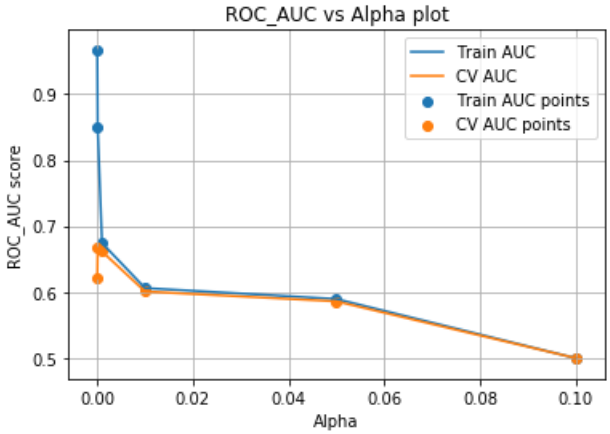
```
# hyperparameter tuning  with l2 reg          reduce the alpha values in list
parameters = {'alpha':[0.00001,0.0001, 0.001,0.01, 0.05, 0.1]}
sd = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced')
classifier = GridSearchCV(sd, parameters, cv= 5, scoring='roc_auc',return_train_score=True)

classifier.fit(X_set2_train, y_train)


train_auc = classifier.cv_results_['mean_train_score']
cv_auc= classifier.cv_results_['mean_test_score']



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

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




plt.legend()
plt.xlabel("Alpha")
plt.ylabel("ROC_AUC score")
plt.title("ROC_AUC vs Alpha plot")
plt.grid()
plt.show()
```



We use l1 regularizer

In [68]:

```
# hyperparameter tuning  with l2 reg          reduce the alpha values in list
parameters = {'alpha':[0.00001,0.0001, 0.001,0.01, 0.05, 0.1]}
sd = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced')
classifier = GridSearchCV(sd, parameters, cv= 5, scoring='roc_auc',return_train_score=True)

classifier.fit(X_set2_train, y_train)


train_auc = classifier.cv_results_['mean_train_score']
cv_auc= classifier.cv_results_['mean_test_score']



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

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




plt.legend()
plt.xlabel("Alpha")
plt.ylabel("ROC_AUC score")
plt.title("ROC_AUC vs Alpha plot")
plt.grid()
plt.show()
```

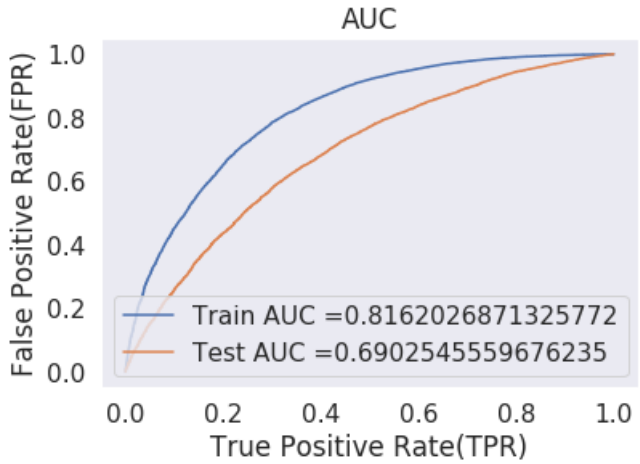


Observation: l1 regularizatin works better than l2 and best alpha is 0.001.

Fit the best hyperparameter

In [0]:

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


Classifier_bow = SGDClassifier(loss = 'hinge', penalty = '12', alpha = 0.001)
Classifier_bow.fit(X_set2_train ,y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

#https://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_model.SGI
sifier.decision_function

y_train_pred = Classifier_bow.decision_function(X_set2_train)
y_test_pred = Classifier_bow.decision_function(X_set2_test)

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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



COnfusion matrix

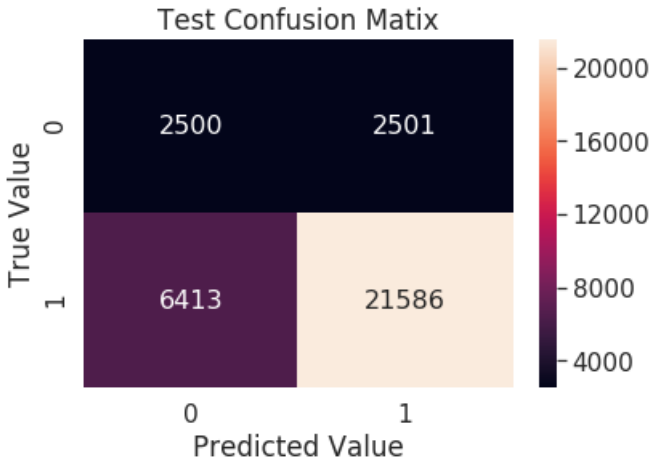
In [0]:

```
import seaborn as sea
#for test dara
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred,
te_thresholds,test_fpr,test_fpr)), range(2),range(2))
sea.set(font_scale=1.4)
sea.heatmap(train_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = 'd')
plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Test Confusion Matix")
```

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

Out[0]:

```
Text(0.5, 1.0, 'Test Confusion Matix')
```

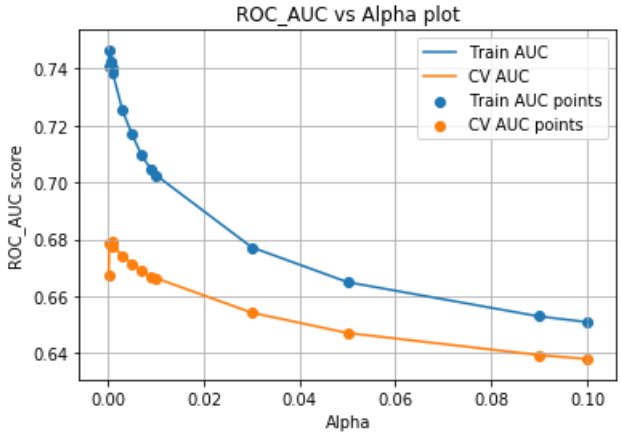


2.4.3 Applying SVM on AVG W2V, SET 3

We use l2 regularizer

In [69]:

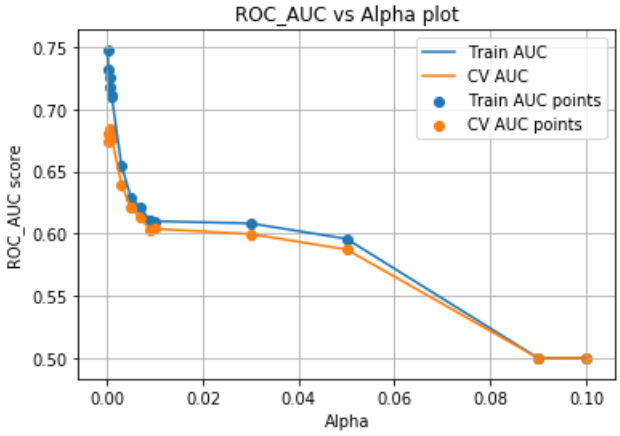
```
# hyperparameter tuning  with l2 reg
parameters = {'alpha':[0.0001,0.0003, 0.0005, 0.0007, 0.0009, 0.001, 0.003, 0.005, 0.007, 0.009, 0.
01, 0.03, 0.05, 0.09, 0.1]}
sd = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced')
classifier = GridSearchCV(sd, parameters, cv= 5, scoring='roc_auc',return_train_score=True)

classifier.fit(X_set3_train, y_train)

train_auc = classifier.cv_results_['mean_train_score']
cv_auc= classifier.cv_results_['mean_test_score']



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

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




plt.legend()
plt.xlabel("Alpha")
plt.ylabel("ROC_AUC score")
plt.title("ROC_AUC vs Alpha plot")
plt.grid()
plt.show()
```



We use l1 regularizer

In [70]:

```
# hyperparameter tuning  with l2 reg
parameters = {'alpha':[0.0001,0.0003, 0.0005, 0.0007, 0.0009, 0.001, 0.003, 0.005, 0.007, 0.009, 0.
01, 0.03, 0.05, 0.09, 0.1]}
sd = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced')
classifier = GridSearchCV(sd, parameters, cv= 5, scoring='roc_auc',return_train_score=True)

classifier.fit(X_set3_train, y_train)

train_auc = classifier.cv_results_['mean_train_score']
cv_auc= classifier.cv_results_['mean_test_score']



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

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




plt.legend()
plt.xlabel("Alpha")
plt.ylabel("ROC_AUC score")
plt.title("ROC_AUC vs Alpha plot")
plt.grid()
plt.show()
```



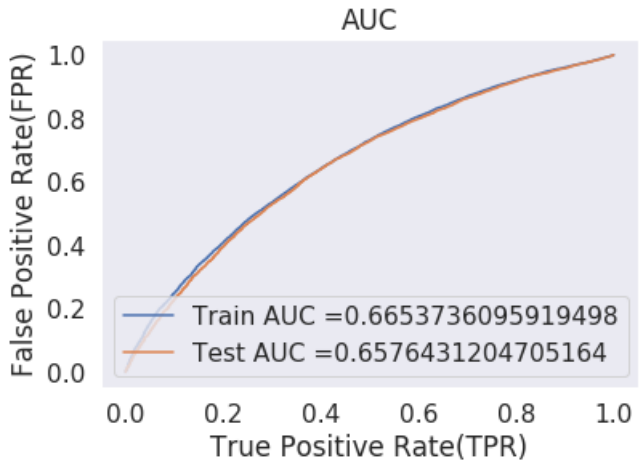
l1 regularizer performs better than l2, best alpha=0.0001

Fitting Model to Hyper-Parameter Curve:

In [0]:

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


Classifier_bow = SGDClassifier(loss = 'hinge', penalty = 'l1', alpha = 0.0001)
Classifier_bow.fit(X_set3_train ,y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

#https://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_model.SGI
sifier.decision_function

y_train_pred = Classifier_bow.decision_function(X_set3_train)
```

```
y_test_pred = Classifier_bow.decision_function(X_set3_test)

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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



confusion matrix test data

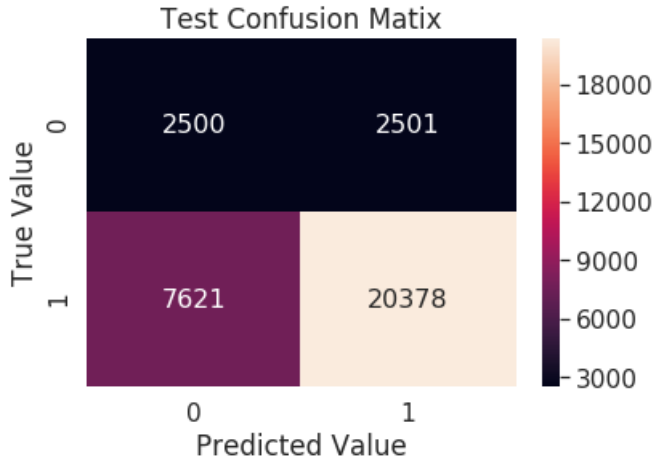
In [0]:

```
import seaborn as sea
#for test dara
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred,
te_thresholds,test_fpr,test_fpr)), range(2),range(2))
sea.set(font_scale=1.4)
sea.heatmap(train_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = 'd')
plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Test Confusion Matix")
```

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 1.05
```

Out[0]:

```
Text(0.5, 1.0, 'Test Confusion Matix')
```

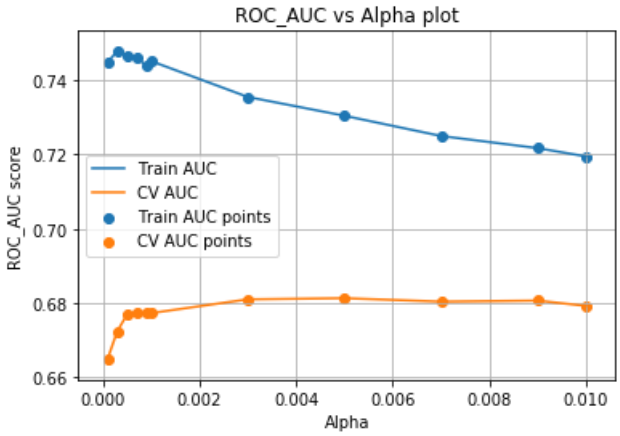


Applying SVM on td_idf W2V, SET 4

We use l2 regularizer

In [71]:

```
""#we are using L2 Regularizer
parameters = {'alpha':[0.0001,0.0003, 0.0005, 0.0007, 0.0009, 0.001, 0.003, 0.005, 0.007, 0.009, 0.
01]}

SV = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced',)
classifier = GridSearchCV(SV, parameters, cv= 5, scoring='roc_auc',return_train_score=True)

classifier.fit(X_set4_train, y_train)

train_auc= classifier.cv_results_['mean_train_score']
cv_auc = classifier.cv_results_['mean_test_score']

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

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



plt.legend()
plt.xlabel("Alpha")
plt.ylabel("ROC_AUC score")
plt.title("ROC_AUC vs Alpha plot")
plt.grid()
plt.show()
```

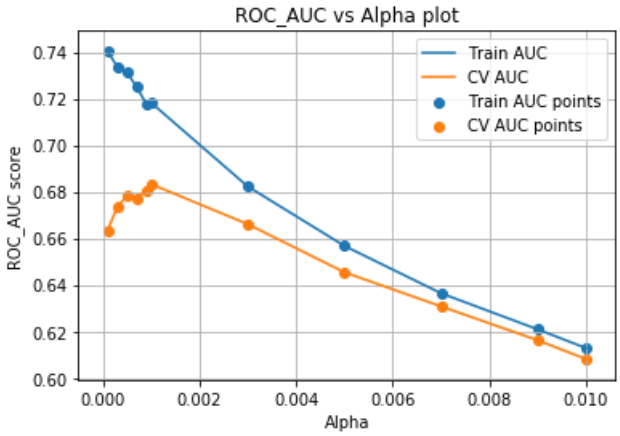


Now,We use l1 regularizer

In [72]:

```
""#we are using L2 Regularizer
parameters = {'alpha':[0.0001,0.0003, 0.0005, 0.0007, 0.0009, 0.001, 0.003, 0.005, 0.007, 0.009, 0.
01]}

SV = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced')
classifier = GridSearchCV(SV, parameters, cv= 5, scoring='roc_auc',return_train_score=True)

classifier.fit(X_set4_train, y_train)

train_auc= classifier.cv_results_['mean_train_score']
cv_auc = classifier.cv_results_['mean_test_score']

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

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

```
plt.legend()
plt.xlabel("Alpha")
plt.ylabel("ROC_AUC score")
plt.title("ROC_AUC vs Alpha plot")
plt.grid()
plt.show()
```

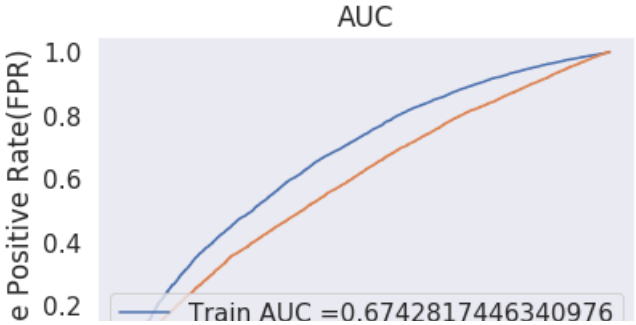


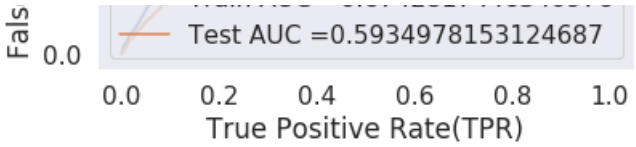
Obervation: l1 regularizer works better than l2

Fitting Model to Hyper-Parameter Curve:

In [0]:

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


Classifier_bow = SGDClassifier(loss = 'hinge', penalty = 'l1', alpha = 0.0001)
Classifier_bow.fit(X_set4_train ,y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

#https://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_model.SGl
sifier.decision_function

y_train_pred = Classifier_bow.decision_function(X_set4_train)
y_test_pred = Classifier_bow.decision_function(X_set4_test)

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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```





Confusion matrix

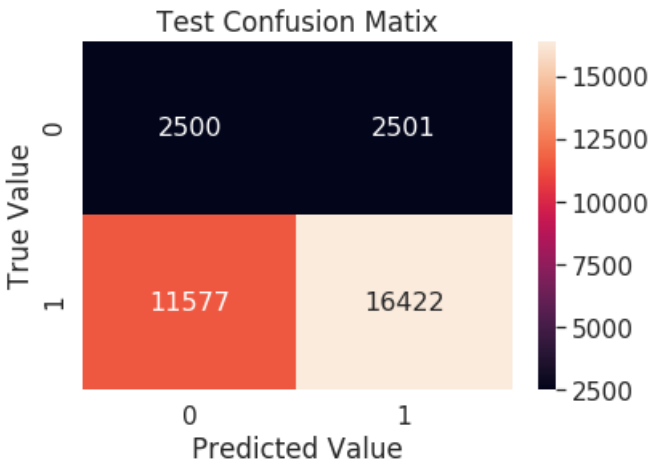
In [0]:

```
import seaborn as sea
#for test dara
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred,
te_thresholds,test_fpr,test_fpr)), range(2),range(2))
sea.set(font_scale=1.4)
sea.heatmap(train_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = 'd')
plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Test Confusion Matix")
```

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 1.02
```

Out[0]:

```
Text(0.5, 1.0, 'Test Confusion Matix')
```



Task 2:

Apply SVM on the set 5

In [0]:

```
# Now instead of bow,tf-df ,wordtovec and tfwor2v featurizers i use three new features
# 1.Sentiment scores of each's essay
# 2.Number of words in titles
# 3.Number of words in combined esssays

# then after apply logistic regression and by taking best hypermeter then i'll compare my results
```

New feature(No. of words in title)

In [80]:

```
X_cv.shape
```

Out[80]:

```
(11055, 19)
```

In [0]:

```
# For train data
title_length_train=[]
for i in range(0,22445):
  title_length_train.append(len(X_train["project_title"][i].split()))


title_length_train=np.array(title_length_train)

#for test data titles
title_length_test=[]
for i in range(0,16500):
  title_length_test.append(len(X_test["project_title"][i].split()))

title_length_test=np.array(title_length_test)

#for cv data titles

title_length_cv=[]
for i in range(0,11055):
  title_length_cv.append(len(X_cv["project_title"][i].split()))

title_length_cv=np.array(title_length_cv)
```

New feature(No. of words in combined essays)

In [0]:

```
#for test data esssay
essay_length_test=[]
for i in range(0,16500):
  essay_length_test.append(len(X_test["essay"][i].split()))

essay_length_test=np.array(essay_length_test)

#for cv data essay

essay_length_cv=[]
for i in range(0,11055,):
  essay_length_cv.append(len(X_cv["essay"][i].split()))

essay_length_cv=np.array(essay_length_cv)


#for train data essay

essay_length_train=[]
for i in range(0,22445):
  essay_length_train.append(len(X_train["essay"][i].split()))

essay_length_train=np.array(essay_length_train)
```

New feature(Sentiment scores of each combined essay's)

In [83]:

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
nltk.download('vader_lexicon')


#https://www.programcreek.com/python/example/100005/nltk.sentiment.vader.SentimentIntensityAnalyze

def analyze_sentiment(df):
    sentiments = []
    sid = SentimentIntensityAnalyzer()
    for i in tqdm(range(df.shape[0])):
```

```
        line = df['essay'][i]# take one essay
        sentiment = sid.polarity_scores(line)# calculate the sentiment
        sentiments.append([sentiment['neg'], sentiment['pos'],
                           sentiment['neu'], sentiment['compound']])# list of lists
    df[['neg', 'pos', 'neu', 'compound']] = pd.DataFrame(sentiments)
    df['Negative'] = df['compound'] < -0.1
    df['Positive'] = df['compound'] > 0.1
    return df
```

```
/usr/local/lib/python3.6/dist-packages/nltk/twitter/__init__.py:20: UserWarning:

The twython library has not been installed. Some functionality from the twitter package will not b
e available.
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
```

In [84]:

```
X_train=analyze_sentiment(X_train)
X_test=analyze_sentiment(X_test)
X_cv=analyze_sentiment(X_cv)
```

```
100%|██████████| 22445/22445 [01:06<00:00, 338.04it/s]
100%|██████████| 16500/16500 [00:49<00:00, 336.72it/s]
100%|██████████| 11055/11055 [00:31<00:00, 346.25it/s]
```

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

In [0]:

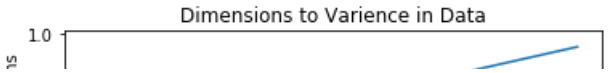
```
#Dimensions are very large so thats why i take less here.
X_train_tf_essay=X_train_tf_essay[:,0:3000]
X_cv_tf_essay=X_cv_tf_essay[:,0:3000]
X_test_tf_essay=X_test_tf_essay[:,0:3000]
```

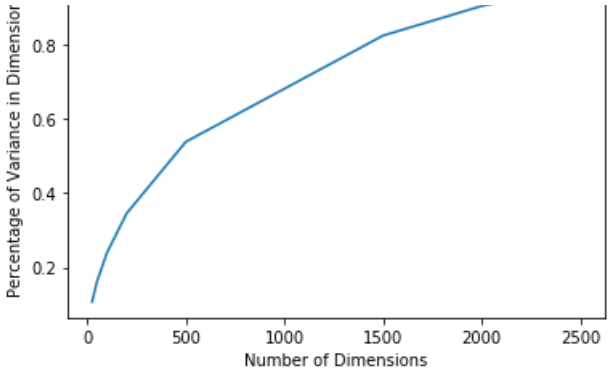
```
from sklearn.decomposition import TruncatedSVD
#https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html
#declaring index as Dimensions in train_text_tfidf
Di = [25,50,100,200,500,1500,2000,2500]
Varience_sum = []
for i in tqdm(Di):
    svd = TruncatedSVD(n_components = i, random_state = 42)
    svd.fit(X_train_tf_essay)
    Varience_sum.append(svd.explained_variance_ratio_.sum())
```

```
100%|██████████| 8/8 [08:47<00:00, 118.85s/it]
```

In [0]:

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





At 2000 dimensions we have Accuracy of greater than 90% so considering 2000 dimensions.

In [0]:

```
svd = TruncatedSVD(n_components= 2000)
svd.fit(X_train_tf_essay)

           #Transforms:

#Train SVD
X_train_tf_essay= svd.transform(X_train_tf_essay )
#print("Shape of SVD Train Matrix is ",X_train_tf_essay .shape)

#Test SVD
X_test_tf_essay = svd.transform(X_test_tf_essay )
#print("Shape of matrix after Decomposition ",X_test_tf_essay .shape)

#CV SVD
X_cv_tf_essay = svd.transform(X_cv_tf_essay )
#print("Shape of matrix after Decomposition ",X_cv_tf_essay .shape)
```

Combine all features:

In [0]:

```
#for train

pos=list(X_train['pos'])
pos=np.array(pos)
neg=list(X_train['neg'])
neg=np.array(neg)
com=list(X_train['compound'])
com=np.array(com)

# combine all
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_set5_train = hstack((
                       X_train_teacher_prefix,X_train_cat,X_train_subcat
,X_train_project_grade_category,X_train_school_state,
                       train_qnty_standar,train_price_standar,train_prev_proj_standar,
                       essay_length_train.reshape(-1,1),title_length_train.reshape(-1,1),
                       pos.reshape(-1,1),neg.reshape(-1,1),com.reshape(-1,1)))



# all categorials, all nurericals and text_tf_train
```

In [0]:

```
#for test
pos=list(X_test['pos'])
pos=np.array(pos)
```

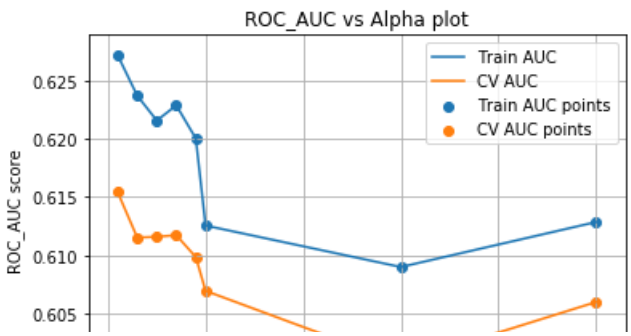
```
neg=list(X_test['neg'])
neg=np.array(neg)

com=list(X_test['compound'])
com=np.array(com)

# combine all
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_set5_test = hstack((
                      X_test_teacher_prefix,X_test_cat,X_test_subcat ,X_test_project_grade_category
,X_test_school_state,
                      test_qnty_standar,test_price_standar,test_prev_proj_standar,
                      essay_length_test.reshape(-1,1),title_length_test.reshape(-1,1),
                      pos.reshape(-1,1),neg.reshape(-1,1),com.reshape(-1,1),))

# all categorials, all nurericals and text_tf_test
```

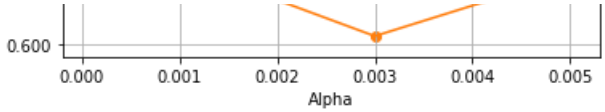
Applying SVM on SET 5

We use the l1 regularizer

In [87]:

```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
#from sklearn.datasets import *
from sklearn import linear_model
from sklearn.linear_model import SGDClassifier
from sklearn import svm
# hyperparameter tuning  with l2 reg
""#we are using L1 Regularizer
parameters = {'alpha':[0.0001,0.0003, 0.0005, 0.0007, 0.0009, 0.001, 0.003, 0.005]}
SV = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced',)
classifier = GridSearchCV(SV, parameters, cv= 3, scoring='roc_auc',return_train_score=True)

classifier.fit(X_set5_train, y_train)

train_auc= classifier.cv_results_['mean_train_score']
cv_auc = classifier.cv_results_['mean_test_score']

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

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



plt.legend()
plt.xlabel("Alpha")
plt.ylabel("ROC_AUC score")
plt.title("ROC_AUC vs Alpha plot")
plt.grid()
plt.show()
```

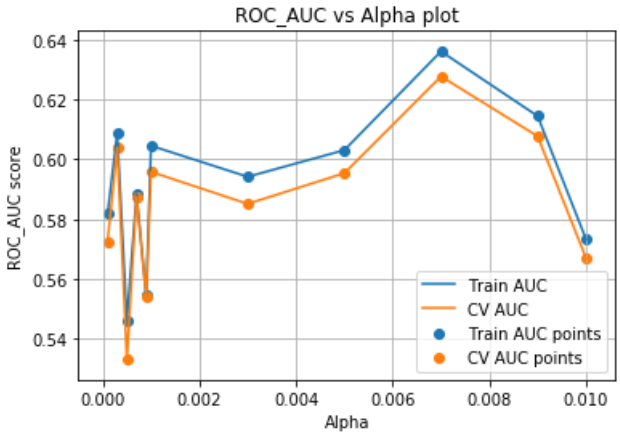




We use the l2 regularizer

In [88]:

```
""#we are using L2 Regularizer
parameters = {'alpha':[0.0001,0.0003, 0.0005, 0.0007, 0.0009, 0.001, 0.003, 0.005, 0.007, 0.009, 0.
01]}
SV = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced',)
classifier = GridSearchCV(SV, parameters, cv= 3, scoring='roc_auc',return_train_score=True)

classifier.fit(X_set5_train, y_train)

train_auc= classifier.cv_results_['mean_train_score']
cv_auc = classifier.cv_results_['mean_test_score']

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

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



plt.legend()
plt.xlabel("Alpha")
plt.ylabel("ROC_AUC score")
plt.title("ROC_AUC vs Alpha plot")
plt.grid()
plt.show()
```



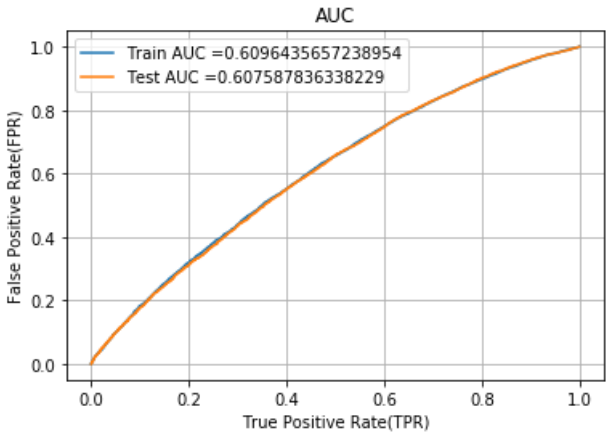
Observatoins: L1 regularizer gives better result

Now *train* with best hyperparameter

In [0]:

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


Classifier_bow = SGDClassifier(loss = 'hinge', penalty = 'l1', alpha = 0.0007)
Classifier_bow.fit(X_set5_train ,y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
```

```
# plot the predicted outputs

#https://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_model.SGD
sifier.decision_function

y_train_pred = Classifier_bow.decision_function(X_set5_train)
y_test_pred = Classifier_bow.decision_function(X_set5_test)

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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



Confusion matrix

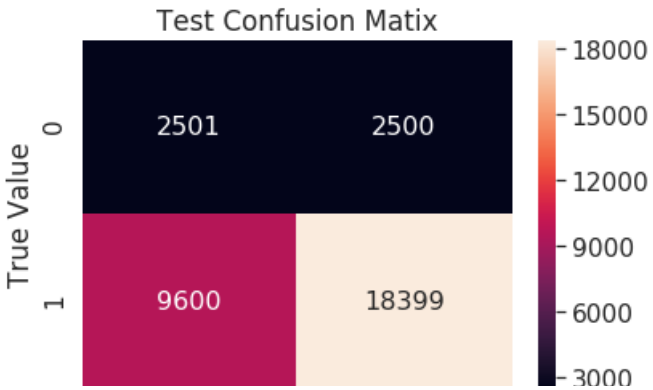
In [0]:

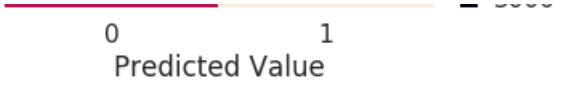
```
import seaborn as sea
#for test dara
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred,
te_thresholds,test_fpr,test_fpr)), range(2),range(2))
sea.set(font_scale=1.4)
sea.heatmap(train_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = 'd')
plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Test Confusion Matix")
```

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 3.21
```

Out[0]:

```
Text(0.5, 1.0, 'Test Confusion Matix')
```





3. Conclusions

In [90]:

```
# Please compare all your models using Prettytable library
# Please compare all your models using Prettytable library
#how to use pretty table http://zetcode.com/python/prettytable/
from prettytable import PrettyTable
tb = PrettyTable()
tb.field_names= ("Vectorizer", " Alpha ", "    AUC  ")
tb.add_row(["BOW ",                      0.01,        68])
tb.add_row(["Tf - Idf ",                        0.001,      70])
tb.add_row(["AVG - W2V",                     0.001,      66])
tb.add_row(["AVG - Tf -  Idf",                    0.001,     60])
tb.add_row(["SVD-Top 3000 Features",        0.007,  60])
print(tb.get_string(titles = "SVM- Observations")) #print(tb)
```

```
+-----------------------+---------+-----------+
|       Vectorizer      |  Alpha  |     AUC   |
+-----------------------+---------+-----------+
|          BOW          |   0.01  |     68    |
|       Tf - Idf        |  0.001  |     70    |
|       AVG - W2V       |  0.001  |     66    |
|    AVG - Tf -  Idf    |  0.001  |     60    |
| SVD-Top 3000 Features |  0.007  |     60    |
+-----------------------+---------+-----------+
```

In [0]:

```

```