DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

Feature	Description
`project_id`	A unique identifier for the proposed project. **Example:** `p036502`
`project_title`	Title of the project. **Examples:** • `Art Will Make You Happy!` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targeted. One of the following enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories for the project from the following enumerated list of values: • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:** • `Music & The Arts` • `Literacy & Language, Math & Science`
`school_state`	State where school is located (Two-letter U.S. postal code). **Example:** `WY`
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the project. **Examples:** • `Literacy`

Feature	Description
	• `Literature & Writing, Social Sciences`
`project_resource_summary`	An explanation of the resources needed for the project. **Example:** • `My students need hands on literacy materials to manage sensory needs!`
`project_essay_1`	First application essay*
`project_essay_2`	Second application essay*
`project_essay_3`	Third application essay*
`project_essay_4`	Fourth application essay*
`project_submitted_datetime`	Datetime when project application was submitted. **Example:** `2016-04-28 12:43:56.245`
`teacher_id`	A unique identifier for the teacher of the proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c15c56`
`teacher_prefix`	Teacher's title. One of the following enumerated values: • `nan` • `Dr.` • `Mr.` • `Mrs.` • `Ms.` • `Teacher.`
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted by the same teacher. **Example:** `2`

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

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

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

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

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

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

Notes on the Essay Data

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

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

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

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

your neighborhood, and your school are all helpful.

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

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

In [2]:

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

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

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

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

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

from tqdm import tqdm
import os

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

1.1 Reading Data

In [12]:

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

In [ ]:

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

In [ ]:

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

1.2 preprocessing of project_subject_categories

In [4]:

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

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

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

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

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

1.3 preprocessing of `project_subject_subcategories`

In [5]:

```
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 'Teacher_prefix'

In [6]:

```
teacher_pre = []
for prefix in project_data['teacher_prefix'].values:
    if prefix==prefix:
        prefix = re.sub('[^A-Za-z0-9]','',prefix).lower()
        teacher_pre.append(prefix)
    else:
        teacher_pre.append(prefix)

project_data['teacher_prefix'] = teacher_pre
```

Preprocessing of project_grade_category

In [7]:

```
project_grade_cat = []
for grade in project_data['project_grade_category'].values:
    grade = grade.replace('-','_').lower()
    project_grade_cat.append(grade)
project_data['project_grade_category'] = project_grade_cat
```

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 [ ]:

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

In [ ]:

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

In [ ]:

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

In [9]:

```
# 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 [ ]:

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

In [ ]:

```
# \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 [ ]:

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

In [10]:

```
# 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"]
```

Stratified Distribution between Train-Test-Cv(64-20-16)

In [11]:

```
from sklearn.model_selection import train_test_split as tts
X_train,X_test,y_train,y_test = tts(project_data,project_data['project_is_approved'],test_size =
0.2, stratify = project_data['project_is_approved'])
```

In [13]:

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

```
(87398, 17)
(21850, 17)
```

1.4 Preprocessing of Essay on Trainig data

In [14]:

```
# 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 not in stopwords)
    preprocessed_essays_train.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████| 87398/87398 [02:
11<00:00, 662.11it/s]
```

1.4 Preprocessing of Essay on Test data

In [15]:

```
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 not in stopwords)
    preprocessed_essays_test.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████| 21850/21850 [00:
35<00:00, 621.53it/s]
```

In [ ]:

```
# after preprocesing
preprocessed_essays[20000]
```

1.4 Preprocessing of `project_title`

1.4 Preprocessing of Title on Trainig data

In [17]:

```
# similarly you can preprocess the titles also
preprocessed_titles_train =[]
for title in tqdm(X_train['project_title'].values):
        des = decontracted(title)
        des = des.replace("\\r",' ')
        des = des.replace('\\"',' ')
        des = des.replace('\\n',' ')
        des = re.sub('[^A-Za-z0-9]+',' ',des)
        des = ' '.join(e for e in des.split() if e not in stopwords)
        preprocessed_titles_train.append(des.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████| 87398/87398
[00:07<00:00, 12318.82it/s]
```

1.4 Preprocessing ofTitle on Test data

In [18]:

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

```
100%|████████████████████████████████████████████████████████████████| 21850/21850
[00:01<00:00, 13702.89it/s]
```

1.5 Preparing data for models

In [ ]:

```
project_data.columns
```

we are going to consider

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

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

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

1.5.1 Vectorizing Categorical data

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

<font color = 'red'> One hot encoding of categories column in train,test,and cv data

In [19]:

```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True
)
vectorizer.fit(X_train['clean_categories'].values)
categories_one_hot_train = vectorizer.transform(preprocessed_essays_train)
categories_one_hot_test = vectorizer.transform(preprocessed_essays_test)
#categories_one_hot_cv = vectorizer.transform(preprocessed_essays_cv)
print(vectorizer.get_feature_names())
print("Shape of Train matrix after one hot encodig ",categories_one_hot_train.shape)
print("Shape of Test matrix after one hot encodig ",categories_one_hot_test.shape)
#print("Shape of CV matrix after one hot encodig ",categories_one_hot_cv.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of Train matrix after one hot encodig  (87398, 9)
Shape of Test matrix after one hot encodig  (21850, 9)
```

<font color = 'red'> One hot encoding of sub categories column in train,test,and cv data

In [20]:

```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=
True)
vectorizer.fit(X_train['clean_subcategories'].values)
sub_categories_one_hot_train = vectorizer.transform(preprocessed_titles_train)
sub_categories_one_hot_test = vectorizer.transform(preprocessed_titles_test)
#sub_categories_one_hot_cv = vectorizer.transform(preprocessed_titles_cv)
print(vectorizer.get_feature_names())
print("Shape of Train matrix after one hot encodig ",sub_categories_one_hot_train.shape)
print("Shape of Test matrix after one hot encodig ",sub_categories_one_hot_test.shape)
#print("Shape of CV matrix after one hot encodig ",sub_categories_one_hot_cv.shape)
```

```
['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']
Shape of Train matrix after one hot encodig  (87398, 30)
Shape of Test matrix after one hot encodig  (21850, 30)
```

<font color = 'red'> One hot encoding of teacher prefix column in train,test,and cv data

In [21]:

```
#https://stackoverflow.com/questions/11620914/removing-nan-values-from-an-array
#https://stackoverflow.com/questions/39303912/tfidfvectorizer-in-scikit-learn-valueerror-np-nan-is
-an-invalid-document
```

```
vectorizer = CountVectorizer(vocabulary=list(filter(lambda v:v==v,project_data['teacher_prefix'].un
ique())),lowercase = False,binary = True)
vectorizer = vectorizer.fit(X_train['teacher_prefix'].values.astype('U'))
prefix_one_hot_train = vectorizer.transform(X_train['teacher_prefix'].values.astype('U'))
#prefix_one_hot_cv = vectorizer.transform(X_cv['teacher_prefix'].values.astype('U'))
prefix_one_hot_test = vectorizer.transform(X_test['teacher_prefix'].values.astype('U'))
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encoding ", prefix_one_hot_train.shape)
#print("Shape of matrix after one hot encoding ", prefix_one_hot_cv.shape)
print("Shape of matrix after one hot encoding ", prefix_one_hot_test.shape)
```

```
['Mrs.', 'Mr.', 'Ms.', 'Teacher', 'Dr.']
Shape of matrix after one hot encoding  (87398, 5)
Shape of matrix after one hot encoding  (21850, 5)
```

<font color = 'red'> One hot encoding of project grade column in train,test,and cv data

In [22]:

```
vectorizer = CountVectorizer(vocabulary=list(filter(lambda
v:v==v,project_data['project_grade_category'].unique())),lowercase = False,binary = True)
vectorizer = vectorizer.fit(X_train['project_grade_category'].values.astype('U'))
project_grade_one_hot_train = vectorizer.transform(X_train['project_grade_category'].values.astype
('U'))
#project_grade_one_hot_cv =
vectorizer.transform(X_cv['project_grade_category'].values.astype('U'))
project_grade_one_hot_test = vectorizer.transform(X_test['project_grade_category'].values.astype('U
'))
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encoding ", project_grade_one_hot_train.shape)
#print("Shape of matrix after one hot encoding ", project_grade_one_hot_cv.shape)
print("Shape of matrix after one hot encoding ", project_grade_one_hot_test.shape)
```

```
['Grades PreK-2', 'Grades 6-8', 'Grades 3-5', 'Grades 9-12']
Shape of matrix after one hot encoding  (87398, 4)
Shape of matrix after one hot encoding  (21850, 4)
```

<font color = 'red'> One hot encoding of project grade column in train,test,and cv data

In [23]:

```
vectorizer = CountVectorizer(vocabulary=list(project_data['school_state'].unique()),lowercase =
False,binary = True)
vectorizer.fit(X_train['school_state'].values)
state_one_hot_train = vectorizer.transform(X_train['school_state'].values)
state_one_hot_test =  vectorizer.transform(X_test['school_state'].values)
#state_one_hot_cv = vectorizer.transform(X_cv['school_state'].values)
print(vectorizer.get_feature_names())
print("Shape of Train matrix after one hot encoding ", state_one_hot_train.shape)
print("Shape of Test matrix after one hot encoding ", state_one_hot_test.shape)
#print("Shape of cv matrix after one hot encoding ", state_one_hot_cv.shape)
```

```
['IN', 'FL', 'AZ', 'KY', 'TX', 'CT', 'GA', 'SC', 'NC', 'CA', 'NY', 'OK', 'MA', 'NV', 'OH', 'PA', 'A
L', 'LA', 'VA', 'AR', 'WA', 'WV', 'ID', 'TN', 'MS', 'CO', 'UT', 'IL', 'MI', 'HI', 'IA', 'RI', 'NJ',
'MO', 'DE', 'MN', 'ME', 'WY', 'ND', 'OR', 'AK', 'MD', 'WI', 'SD', 'NE', 'NM', 'DC', 'KS', 'MT', 'NH
', 'VT']
Shape of Train matrix after one hot encoding  (87398, 51)
Shape of Test matrix after one hot encoding  (21850, 51)
```

Essay and Title Words Count

Train data

In [24]:

```
essay_word_counter_train = []
title_word_counter_train = []
for sent in preprocessed_essays_train:
    count = len(set(sent.split()))
    essay_word_counter_train.append(count)
for title in preprocessed_titles_train:
    count = len(set(title.split()))
    title_word_counter_train.append(count)
X_train['Essay_word_count'] = essay_word_counter_train
X_train['Title_word_count'] = title_word_counter_train
```

Test Data

In [25]:

```
essay_word_counter_test = []
title_word_counter_test = []
for sent in preprocessed_essays_test:
    count = len(set(sent.split()))
    essay_word_counter_test.append(count)
for title in preprocessed_titles_test:
    count = len(set(title.split()))
    title_word_counter_test.append(count)
X_test['Essay_word_count'] = essay_word_counter_test
X_test['Title_word_count'] = title_word_counter_test
```

1.5.2 Vectorizing Text data

Bag of words - Essays and Titles

Train data-Essay

In [26]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
#training
vectorizer = CountVectorizer(min_df=10,max_features=5000)
essay_bow_train = vectorizer.fit_transform(preprocessed_essays_train[0:45000])
print("Shape of matrix after one hot encodig ",essay_bow_train.shape)
```

```
Shape of matrix after one hot encodig  (45000, 5000)
```

Test data-Essay

In [27]:

```
#test
essay_bow_test = vectorizer.transform(preprocessed_essays_test[0:15000])
print("Shape of matrix after one hot encodig ",essay_bow_test.shape)
```

```
Shape of matrix after one hot encodig  (15000, 5000)
```

Train data-Title

In [28]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
#training
vectorizer = CountVectorizer(min_df=10,max_features=5000)
title_bow_train = vectorizer.fit_transform(preprocessed_titles_train[0:45000])
print("Shape of matrix after one hot encodig ",title_bow_train.shape)
```

```
Shape of matrix after one hot encodig  (45000, 1977)
```

Test data - title

In [29]:

```
#test
title_bow_test = vectorizer.transform(preprocessed_titles_test[0:15000])
print("Shape of matrix after one hot encodig ",title_bow_test.shape)
```

```
Shape of matrix after one hot encodig  (15000, 1977)
```

TFIDF - Essays and Titles

<font color = 'red'>Essay on Train-Test_cv dataset

In [30]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10,max_features=5000)
essay_tfidf_train = vectorizer.fit_transform(preprocessed_essays_train[0:45000])
print("Shape of matrix after one hot encodig ",essay_tfidf_train.shape)
```

```
Shape of matrix after one hot encodig  (45000, 5000)
```

In [31]:

```
essay_tfidf_test = vectorizer.transform(preprocessed_essays_test[0:15000])
print("Shape of matrix after one hot encoding ",essay_tfidf_test.shape)
```

```
Shape of matrix after one hot encoding  (15000, 5000)
```

<font color = 'red'>Title on Train-Test_cv dataset

In [32]:

```
vectorizer = TfidfVectorizer(min_df = 10,max_features=5000)
title_tfidf_train = vectorizer.fit_transform(preprocessed_titles_train[0:45000])
print("Shape of matrix after one hot encding ",title_tfidf_train.shape)
```

```
Shape of matrix after one hot encding  (45000, 1977)
```

In [33]:

```
title_tfidf_test = vectorizer.transform(preprocessed_titles_test[0:15000])
```

```
print("Shape of matrix after one hot encding ",title_tfidf_test.shape)
```

```
Shape of matrix after one hot encding  (15000, 1977)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [ ]:

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

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

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

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

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

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

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

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


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

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


'''
```

In [32]:

```
# 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 cile
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

avg w2v vectors on Preprocessed Essays - Training data

In [33]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_essays_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_train): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_essays_train.append(vector)

print(len(avg_w2v_vectors_essays_train))
print(len(avg_w2v_vectors_essays_train[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 87398/87398
[00:41<00:00, 2082.95it/s]
```

```
87398
300
```

avg w2v vectors on Preprocessed Essays - Test data

In [34]:

```
avg_w2v_vectors_essays_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_test): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_essays_test.append(vector)

print(len(avg_w2v_vectors_essays_test))
print(len(avg_w2v_vectors_essays_test[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 21850/21850
[00:12<00:00, 1783.60it/s]
```

```
21850
300
```

avg w2v vectors on Preprocessed Titles - Training data

In [35]:

```
#compute avg w2v for each title
avg_w2V_vectors_title_train =[]
for title in tqdm(preprocessed_titles_train):
    vector_title = np.zeros(300)
    cnt_words = 0
    for word in title.split():
        if word in glove_words:
            vector_title+=model[word]
            cnt_words+=1
    if cnt_words!=0:
        vector_title/=cnt_words
```

```
        vector_title/=cnt_words
    avg_w2V_vectors_title_train.append(vector_title)
print(len(avg_w2V_vectors_title_train))
print(len(avg_w2V_vectors_title_train[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 87398/87398
[00:02<00:00, 37734.88it/s]
```

```
87398
300
```

avg w2v vectors on Preprocessed Titles - Test data

In [36]:

```
#compute avg w2v for each title
avg_w2V_vectors_title_test =[]
for title in tqdm(preprocessed_titles_test):
    vector_title = np.zeros(300)
    cnt_words = 0
    for word in title.split():
        if word in glove_words:
            vector_title+=model[word]
            cnt_words+=1
    if cnt_words!=0:
        vector_title/=cnt_words
    avg_w2V_vectors_title_test.append(vector_title)
print(len(avg_w2V_vectors_title_test))
print(len(avg_w2V_vectors_title_test[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 21850/21850
[00:00<00:00, 44301.18it/s]
```

```
21850
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [37]:

```
# 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())
```

tfidf w2v vectors on Preprocessed Essay - Training data

In [38]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_train): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
```

```
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors.append(vector)

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

```
100%|██████████| 87398/87398 [05:49<00:00, 250.35it/s]
```

```
87398
300
```

tfidf w2v vectors on Preprocessed Essay - Testing data

In [39]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_test): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_test.append(vector)

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

```
100%|██████████| 21850/21850 [01:12<00:00, 300.24it/s]
```

```
21850
300
```

tfidf of Title

In [40]:

```
# Similarly you can vectorize for title also
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_titles_train)
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

tfidf w2v vectors on Preprocessed Titles - Training data

In [41]:

```
tfidf_w2v_vectors_title_train= []
for title in tqdm(preprocessed_titles_train):
    vector = np.zeros(300)
    tf_idf_wgt = 0
    for word in title.split():
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word]
```

```
            tf_idf = dictionary[word]*(title.count(word)/len(title.split()))
            vector += (vec*tf_idf)
            tf_idf_weight+=tf_idf
    if tf_idf_weight!=0:
        vector/=tf_idf_weight
    tfidf_w2v_vectors_title_train.append(vector)
print(len(tfidf_w2v_vectors_title_train))
print(len(tfidf_w2v_vectors_title_train[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 87398/87398
[00:04<00:00, 20469.98it/s]
```

```
87398
300
```

tfidf w2v vectors on Preprocessed Titles - Test data

In [42]:

```
# Similarly you can vectorize for title also
tfidf_w2v_vectors_title_test= []
for title in tqdm(preprocessed_titles_test):
    vector = np.zeros(300)
    tf_idf_wgt = 0
    for word in title.split():
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word]
            tf_idf = dictionary[word]*(title.count(word)/len(title.split()))
            vector += (vec*tf_idf)
            tf_idf_weight+=tf_idf
    if tf_idf_weight!=0:
        vector/=tf_idf_weight
    tfidf_w2v_vectors_title_test.append(vector)
print(len(tfidf_w2v_vectors_title_test))
print(len(tfidf_w2v_vectors_title_test[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 21850/21850
[00:01<00:00, 18118.31it/s]
```

```
21850
300
```

1.5.3 Vectorizing Numerical features

In [35]:

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

Price

In [36]:

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

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

price_scalar = StandardScaler()
```

```
price_scalar.fit(X_train['price'][0:45000].values.reshape(-1,1)) # finding the mean and standard de
viation of this data
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
price_standardized_train = price_scalar.transform(X_train['price'][0:45000].values.reshape(-1, 1))
#price_standardized_cv = price_scalar.transform(X_cv['price'][0:12000].values.reshape(-1,1))
price_standardized_test = price_scalar.transform(X_test['price'][0:15000].values.reshape(-1,1))
```

```
Mean : 298.3524622222222, Standard deviation : 384.5664192046699
```

Quantity

In [37]:

```
# standardized quantity columns
quantity_scaler = StandardScaler()
quantity_scaler.fit(X_train['quantity'][0:45000].values.reshape(-1,1))
print(f"Mean :{quantity_scaler.mean_[0]},Standard Deviation :{np.sqrt(quantity_scaler.var_[0])}")
quantity_standardized_train = quantity_scaler.transform(X_train['quantity']
[0:45000].values.reshape(-1,1))
#quantity_standardized_cv = quantity_scaler.transform(X_cv['quantity'][0:12000].values.reshape(-1,
1))
quantity_standardized_test = quantity_scaler.transform(X_test['quantity'][0:15000].values.reshape(-
1,1))
```

```
Mean :17.005133333333333,Standard Deviation :26.059351835633468
```

No.of previously done Project

In [38]:

```
#standardized projects proposed by teachers
project_scaler = StandardScaler()
project_scaler.fit(X_train['teacher_number_of_previously_posted_projects'][0:45000].values.reshape(
-1,1))
print(f"Mean :{project_scaler.mean_[0]},Standard Deviation :{np.sqrt(project_scaler.var_[0])}")
project_standardized_train =
project_scaler.transform(X_train['teacher_number_of_previously_posted_projects']
[0:45000].values.reshape(-1,1))
#project_standardized_cv =
project_scaler.transform(X_cv['teacher_number_of_previously_posted_projects']
[0:12000].values.reshape(-1,1))
project_standardized_test =
project_scaler.transform(X_test['teacher_number_of_previously_posted_projects'][0:15000].values.res
hape(-1,1))
```

```
Mean :11.109755555555555,Standard Deviation :27.528632737291755
```

Essay Count

In [39]:

```
#standardized Essay Count
Essay_count_scaler = StandardScaler()
Essay_count_scaler.fit(X_train['Essay_word_count'][0:45000].values.reshape(-1,1))
print(f"Mean :{Essay_count_scaler.mean_[0]},Standard Deviation :
{np.sqrt(Essay_count_scaler.var_[0])}")
Essay_count_standardized_train = Essay_count_scaler.transform(X_train['Essay_word_count'][:45000].
values.reshape(-1,1))
Essay_count_standardized_test = Essay_count_scaler.transform(X_test['Essay_word_count']
[:15000].values.reshape(-1,1))
#Essay_count_standardized_cv = Essay_count_scaler.transform(X_cv['Essay_word_count']
[:45000].values.reshape(-1,1))
```

```
Mean :109.62097777777778,Standard Deviation :26.10350738331187
```

Title Count

In [40]:

```
#standardized Title Count
title_count_scaler = StandardScaler()
title_count_scaler.fit(X_train['Title_word_count'][0:45000].values.reshape(-1,1))
print(f"Mean :{title_count_scaler.mean_[0]},Standard Deviation :
{np.sqrt(title_count_scaler.var_[0])}")
title_count_standardized_train = title_count_scaler.transform(X_train['Title_word_count'][:45000].
values.reshape(-1,1))
title_count_standardized_test = title_count_scaler.transform(X_test['Title_word_count']
[:15000].values.reshape(-1,1))
#title_count_standardized_cv = title_count_scaler.transform(X_cv['Title_word_count']
[:45000].values.reshape(-1,1))
```

```
Mean :4.2408,Standard Deviation :1.7353750231898326
```

1.5.4 Merging all the above features

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

Computing Sentiment Scores

In [ ]:

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

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

sid = SentimentIntensityAnalyzer()

for_sentiment = 'a person is a person no matter how small dr seuss i teach the smallest students w
ith the biggest enthusiasm \
for learning my students learn in many different ways using all of our senses and multiple intelli
gences i use a wide range\
of techniques to help all my students succeed students in my class come from a variety of differen
t backgrounds which makes\
for wonderful sharing of experiences and cultures including native americans our school is a carin
g community of successful \
learners which can be seen through collaborative student project based learning in and out of the
classroom kindergarteners \
in my class love to work with hands on materials and have many different opportunities to practice
a skill before it is\
mastered having the social skills to work cooperatively with friends is a crucial aspect of the ki
ndergarten curriculum\
montana is the perfect place to learn about agriculture and nutrition my students love to role pla
y in our pretend kitchen\
in the early childhood classroom i have had several kids ask me can we try cooking with real food
i will take their idea \
and create common core cooking lessons where we learn important math and writing concepts while co
oking delicious healthy \
food for snack time my students will have a grounded appreciation for the work that went into maki
ng the food and knowledge \
of where the ingredients came from as well as how it is healthy for their bodies this project woul
d expand our learning of \
nutrition and agricultural cooking recipes by having us peel our own apples to make homemade apple
sauce make our own bread \
and mix up healthy plants from our classroom garden in the spring we will also create our own cook
books to be printed and \
shared with families students will gain math and literature skills as well as a life long enjoymen
t for healthy cooking \
nannan'
ss = sid.polarity_scores(for_sentiment)

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

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

Assignment 7: SVM

1. **[Task-1] Apply Support Vector Machines(SGDClassifier with hinge loss: Linear SVM) on these feature sets**

   - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_eassay (BOW)
   - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)
   - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)
   - Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)

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

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

3. **Representation of results**

   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure.
   - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
   - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.

4. **[Task-2] Apply the Support Vector Machines on these features by finding the best hyper paramter as suggested in step 2 and step 3**

   - Consider these set of features Set 5 :
     - **school_state** : categorical data
     - **clean_categories** : categorical data
     - **clean_subcategories** : categorical data
     - **project_grade_category** :categorical data
     - **teacher_prefix** : categorical data
     - **quantity** : numerical data
     - **teacher_number_of_previously_posted_projects** : numerical data
     - **price** : numerical data
     - **sentiment score's of each of the essay** : numerical data
     - **number of words in the title** : numerical data
     - **number of words in the combine essays** : numerical data
     - **Apply TruncatedSVD on TfidfVectorizer of essay text, choose the number of components (`n_components`) using elbow method** : numerical data

   - **Conclusion**
     - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

Note: Data Leakage

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

2. Support Vector Machines

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

In [ ]:

```
# 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
```

2.2 Make Data Model Ready: encoding numerical, categorical features

In [ ]:

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

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

2.3 Make Data Model Ready: encoding eassay, and project_title

In [ ]:

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

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

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

Apply Support Vector Machines on different kind of featurization as mentioned in the instructions
For Every model that you work on make sure you do the step 2 and step 3 of instrucations

In [ ]:

```
# 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
```

categorical, numerical features + project_title(BOW) + preprocessed_eassay (BOW)

In [59]:

```
# Please write all the code with proper documentation
#from xgboost import XGBClassifier
#import xgboost as xgb
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
from scipy import sparse
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_tr =
hstack((categories_one_hot_train[:45000],sub_categories_one_hot_train[:45000],prefix_one_hot_train
[:45000],
project_grade_one_hot_train[:45000],state_one_hot_train[:45000],sparse.csr_matrix(price_standardize
d_train[:45000]),
        sparse.csr_matrix(quantity_standardized_train[:45000]),sparse.csr_matrix(project_standardiz
ed_train[:45000]),
               sparse.csr_matrix(Essay_count_standardized_train[:45000]),sparse.csr_matrix(title_co
unt_standardized_train[:45000])
               ,essay_bow_train,title_bow_train)).tocsr()


X_ts =
hstack((categories_one_hot_test[:15000],sub_categories_one_hot_test[:15000],prefix_one_hot_test[:1
5000],
project_grade_one_hot_test[:15000],state_one_hot_test[:15000],sparse.csr_matrix(price_standardized_
test[:15000]),

sparse.csr_matrix(quantity_standardized_test[:15000]),sparse.csr_matrix(project_standardized_test[
:15000]),
        sparse.csr_matrix(Essay_count_standardized_test[:15000]),sparse.csr_matrix(title_count_stan
dardized_test[:15000])
             , essay_bow_test,title_bow_test)).tocsr()
```

In [50]:

```
# batch wise prediction
def proba_predict(model , data):
    y_pred_data = []
    n_loop = data.shape[0] - data.shape[0]%1000
    # here 1000 represents batch_size
    for i in range(0,n_loop,1000):
        y_pred_data.extend(model.predict_proba(data[i:i+1000])[:,1])
    if data.shape[0]%1000!=0:
        y_pred_data.extend(model.predict_proba(data[n_loop:])[:,1])
    return(y_pred_data)
```

Finding best Alpha value with best penalty among 'l1' and 'l2'

In [51]:

```
import matplotlib.pyplot as plt
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV
from sklearn.metrics import roc_auc_score,roc_curve,f1_score,auc
```

In [62]:

```
parameters={'alpha' : [1e-4,1e-3,1e-2,1e-1,1,1e+1,1e+2,1e+3,1e+4]}
model = SGDClassifier(loss = 'hinge',penalty='l2')
clf = GridSearchCV(model,param_grid=parameters,cv=5,scoring='roc_auc')
clf.fit(X_tr,y_train[:45000])
```

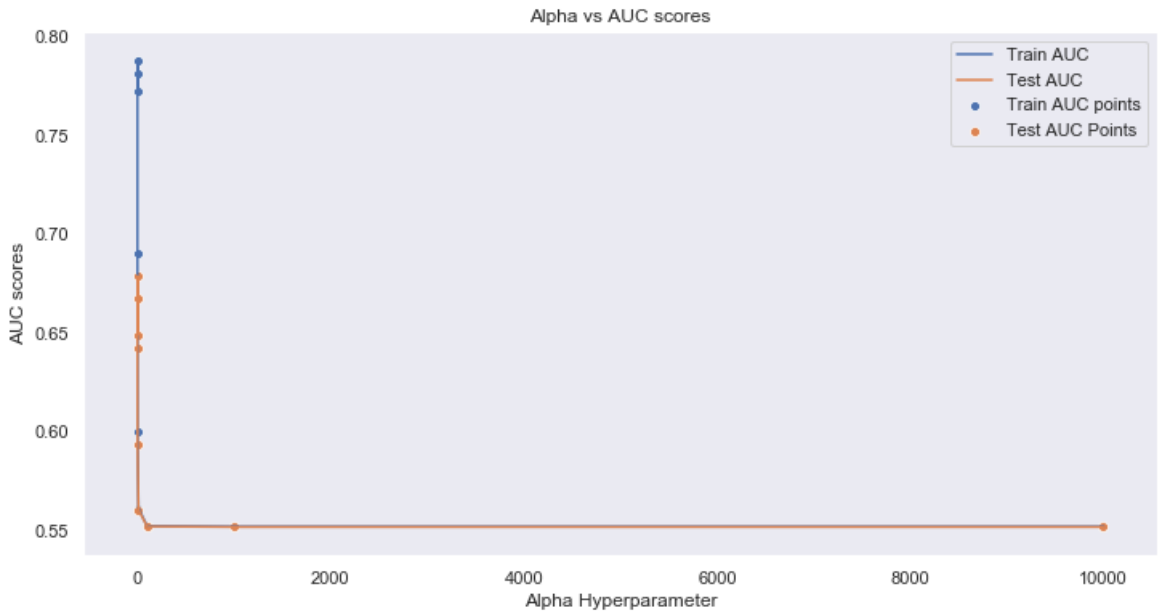
Out[62]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=SGDClassifier(alpha=0.0001, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
```

```
        l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
        n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
        power_t=0.5, random_state=None, shuffle=True, tol=None,
        validation_fraction=0.1, verbose=0, warm_start=False),
        fit_params=None, iid='warn', n_jobs=None,
        param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10.0, 100.0, 1000.0, 10000.0]},
        pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
        scoring='roc_auc', verbose=0)
```

In [65]:

```
train_auc = clf.cv_results_['mean_train_score']
test_auc = clf.cv_results_['mean_test_score']
plt.figure(figsize = (12,6))
sns.lineplot(parameters['alpha'],train_auc,label = "Train AUC")
sns.lineplot(parameters['alpha'],test_auc,label = "Test AUC")
sns.scatterplot(parameters['alpha'],train_auc,label = 'Train AUC points')
sns.scatterplot(parameters['alpha'],test_auc,label = 'Test AUC Points')
sns.set()
plt.legend()
plt.xlabel("Alpha Hyperparameter")
plt.ylabel("AUC scores")
plt.title("Alpha vs AUC scores")
plt.grid()
plt.show()
```



From the above it's not clear which alpha estimator is good for the model

as we can see larger alpha value has no or less std or variance and Hence

we look out alpha parameters with small values

In [68]:

```
parameters={'alpha' :[1e-4,1e-3,1e-2,1e-1,1,1e+1] }
model = SGDClassifier(loss = 'hinge',penalty='l2')
clf = GridSearchCV(model,param_grid=parameters,cv=5,scoring='roc_auc')
clf.fit(X_tr,y_train[:45000])
```

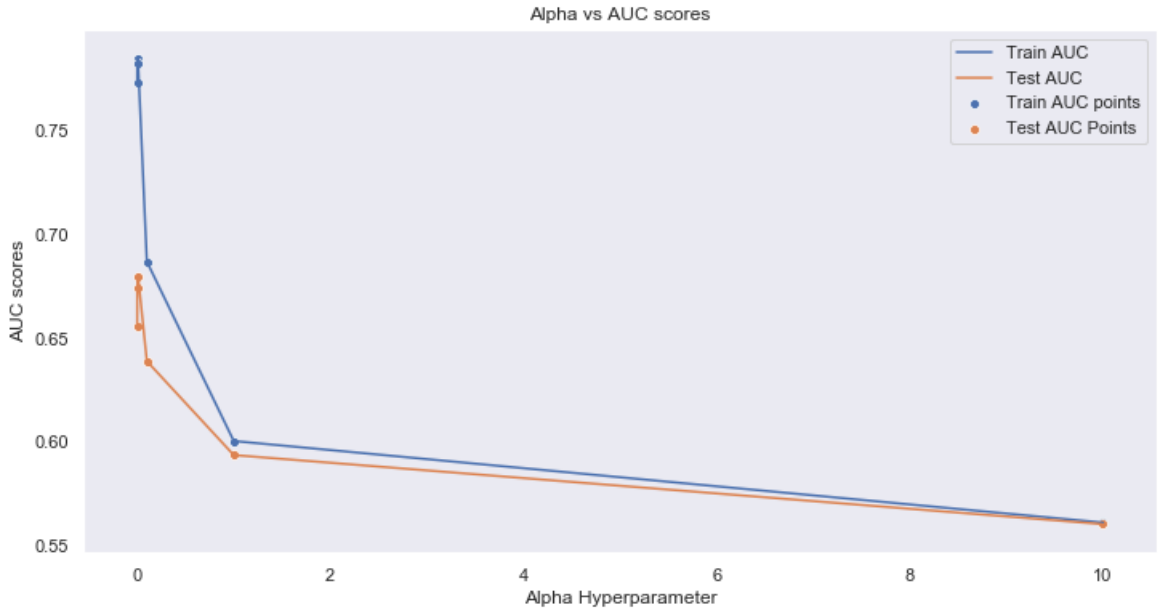
Out[68]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=SGDClassifier(alpha=0.0001, average=False, class_weight=None,
```

```
        early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
        l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
        n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
        power_t=0.5, random_state=None, shuffle=True, tol=None,
        validation_fraction=0.1, verbose=0, warm_start=False),
        fit_params=None, iid='warn', n_jobs=None,
        param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10.0]},
        pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
        scoring='roc_auc', verbose=0)
```

In [69]:

```
train_auc = clf.cv_results_['mean_train_score']
test_auc = clf.cv_results_['mean_test_score']
plt.figure(figsize = (12,6))
sns.lineplot(parameters['alpha'],train_auc,label = "Train AUC")
sns.lineplot(parameters['alpha'],test_auc,label = "Test AUC")
sns.scatterplot(parameters['alpha'],train_auc,label = 'Train AUC points')
sns.scatterplot(parameters['alpha'],test_auc,label = 'Test AUC Points')
sns.set()
plt.legend()
plt.xlabel("Alpha Hyperparameter")
plt.ylabel("AUC scores")
plt.title("Alpha vs AUC scores")
plt.grid()
plt.show()
```



We are getting our elbow graph so we tried to Zoom in further for value less than 1.0 using l2 penalty

In [70]:

```
parameters={'alpha' :[0.01,0.05,0.1,0.2,0.5,0.8,1.0] }
model = SGDClassifier(loss = 'hinge',penalty='l2')
clf = GridSearchCV(model,param_grid=parameters,cv=5,scoring='roc_auc')
clf.fit(X_tr,y_train[:45000])
```

Out[70]:

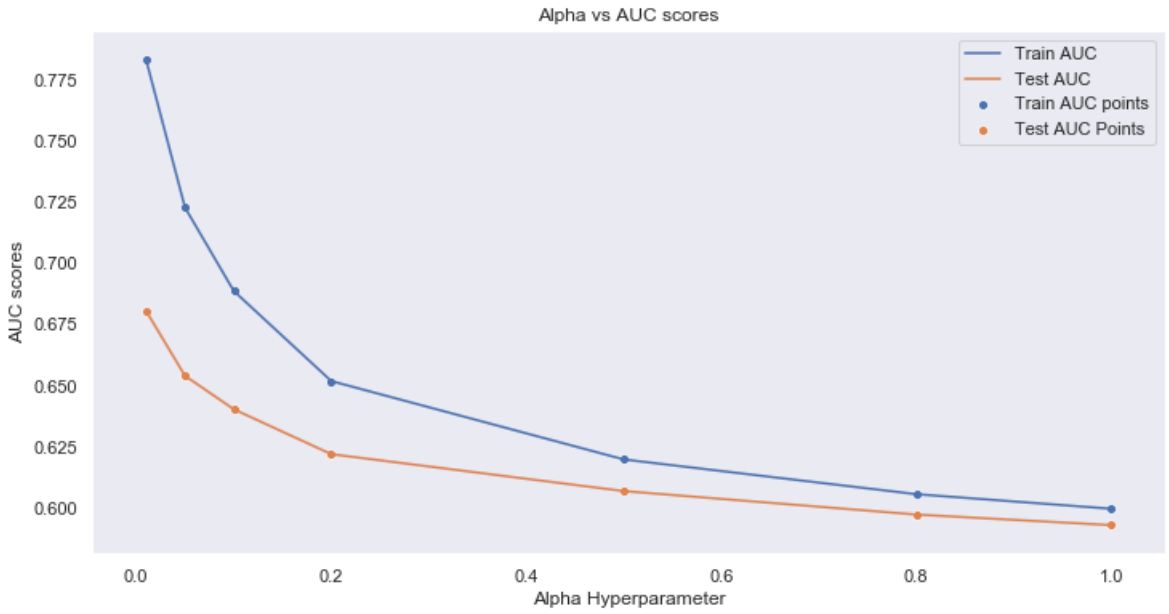
```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=SGDClassifier(alpha=0.0001, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'alpha': [0.01, 0.05, 0.1, 0.2, 0.5, 0.8, 1.0]},
```

```
        pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
        scoring='roc_auc', verbose=0)
```

Elbow Graph

In [71]:

```
train_auc = clf.cv_results_['mean_train_score']
test_auc = clf.cv_results_['mean_test_score']
plt.figure(figsize = (12,6))
sns.lineplot(parameters['alpha'],train_auc,label = "Train AUC")
sns.lineplot(parameters['alpha'],test_auc,label = "Test AUC")
sns.scatterplot(parameters['alpha'],train_auc,label = 'Train AUC points')
sns.scatterplot(parameters['alpha'],test_auc,label = 'Test AUC Points')
sns.set()
plt.legend()
plt.xlabel("Alpha Hyperparameter")
plt.ylabel("AUC scores")
plt.title("Alpha vs AUC scores")
plt.grid()
plt.show()
```



We are getting our elbow graph so we tried to Zoom in further for value less than 1.0 using l1 penalty

In [73]:

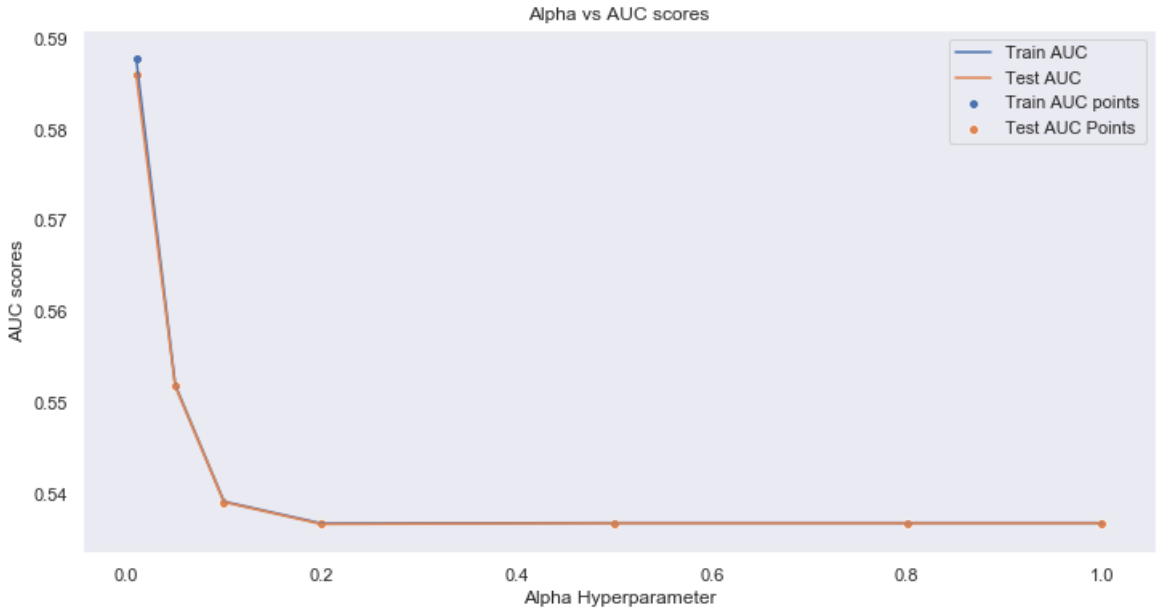
```
parameters={'alpha' :[0.01,0.05,0.1,0.2,0.5,0.8,1.0] }
model = SGDClassifier(loss = 'hinge',penalty='l1')
clf = GridSearchCV(model,param_grid=parameters,cv=5,scoring='roc_auc')
clf.fit(X_tr,y_train[:45000])
```

Out[73]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=SGDClassifier(alpha=0.0001, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l1',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'alpha': [0.01, 0.05, 0.1, 0.2, 0.5, 0.8, 1.0]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [74]:

```
train_auc = clf.cv_results_['mean_train_score']
test_auc = clf.cv_results_['mean_test_score']
plt.figure(figsize = (12,6))
sns.lineplot(parameters['alpha'],train_auc,label = "Train AUC")
sns.lineplot(parameters['alpha'],test_auc,label = "Test AUC")
sns.scatterplot(parameters['alpha'],train_auc,label = 'Train AUC points')
sns.scatterplot(parameters['alpha'],test_auc,label = 'Test AUC Points')
sns.set()
plt.legend()
plt.xlabel("Alpha Hyperparameter")
plt.ylabel("AUC scores")
plt.title("Alpha vs AUC scores")
plt.grid()
plt.show()
```



Best Fit

In [86]:

```
model = SGDClassifier(loss = 'hinge',alpha=0.1,penalty='l2')
model.fit(X_tr,y_train[:45000])
```

Out[86]:

```
SGDClassifier(alpha=0.1, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False)
```

ROC_AUC Curve

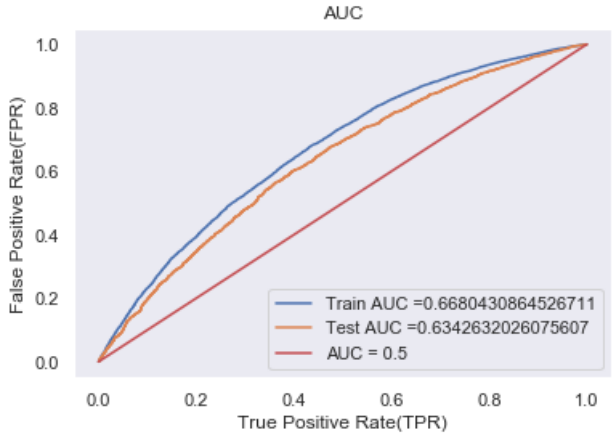
In [91]:

```
y_train_pred = model.decision_function(X_tr)
y_test_pred = model.decision_function(X_ts)

fpr_train,tpr_train,thres_train = roc_curve(y_train[:45000], y_train_pred)
fpr_test,tpr_test,thres_test = roc_curve(y_test[:15000], y_test_pred)

plt.plot(fpr_train, tpr_train, label="Train AUC ="+str(auc(fpr_train, tpr_train)))
plt.plot(fpr_test, tpr_test, label="Test AUC ="+str(auc(fpr_test, tpr_test)))
plt.plot(np.linspace(0,1,600),np.linspace(0,1,600),label = "AUC = 0.5",color = "r")
plt.legend()
```

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



In [66]:

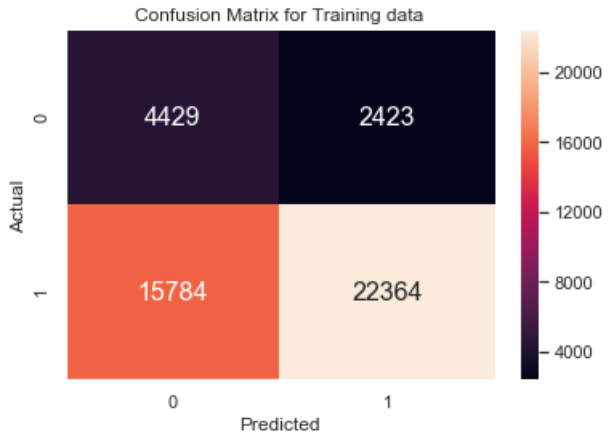
```
def pred_using_threshold(proba,thresh,tpr,fpr):
    flag = thresh[np.argmax(fpr*(1-tpr))]
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(flag,3))
    pred_auc = []
    for i in proba:
        if i>=flag:
            pred_auc.append(1)
        else:
            pred_auc.append(0)
    return pred_auc
```

Confusion Matrix on training data with alpha = 0.1

In [94]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
ax = sns.heatmap(confusion_matrix(y_train[:45000],pred_using_threshold(y_train_pred,thres_train,tpr_train,fpr_train)),
                 annot=True,annot_kws={"size": 16}, fmt='g')
ax.set_title("Confusion Matrix for Training data")
ax.set_xlabel("Predicted")
ax.set_ylabel("Actual")
sns.despine()
```

```
the maximum value of tpr*(1-fpr) 0.3850291825899937 for threshold 1.085
```

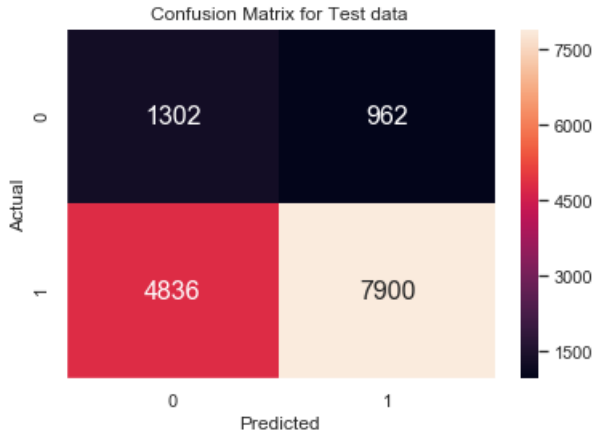


Confusion Matrix on test data with alpha = 0.1

In [97]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
ax =
sns.heatmap(confusion_matrix(y_test[:15000],pred_using_threshold(y_test_pred,thres_test,tpr_test,f
pr_test)),
                annot=True,annot_kws={"size": 16}, fmt='g')
ax.set_title("Confusion Matrix for Test data")
ax.set_xlabel("Predicted")
ax.set_ylabel("Actual")
sns.despine()
```

```
the maximum value of tpr*(1-fpr) 0.36208045805440636 for threshold 1.075
```



categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)

In [41]:

```
# Please write all the code with proper documentation
#from xgboost import XGBClassifier
#import xgboost as xgb
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
from scipy import sparse
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_tr =
hstack((categories_one_hot_train[:45000],sub_categories_one_hot_train[:45000],prefix_one_hot_train
[:45000],
project_grade_one_hot_train[:45000],state_one_hot_train[:45000],sparse.csr_matrix(price_standardize
d_train[:45000]),
        sparse.csr_matrix(quantity_standardized_train[:45000]),sparse.csr_matrix(project_standardiz
ed_train[:45000]),
               sparse.csr_matrix(Essay_count_standardized_train[:45000]),sparse.csr_matrix(title_co
unt_standardized_train[:45000])
               ,essay_tfidf_train,title_tfidf_train)).tocsr()


X_ts =
hstack((categories_one_hot_test[:15000],sub_categories_one_hot_test[:15000],prefix_one_hot_test[:1
5000],
project_grade_one_hot_test[:15000],state_one_hot_test[:15000],sparse.csr_matrix(price_standardized_
test[:15000]),

sparse.csr_matrix(quantity_standardized_test[:15000]),sparse.csr_matrix(project_standardized_test[
:15000]),
        sparse.csr_matrix(Essay_count_standardized_test[:15000]),sparse.csr_matrix(title_count_stan
dardized_test[:15000]) ,essay_tfidf_test,title_tfidf_test)).tocsr()
```

Finding best Alpha value with best penalty among 'l1' and 'l2'

In [99]:

```
import matplotlib.pyplot as plt
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV
from sklearn.metrics import roc_auc_score,roc_curve,f1_score,auc
```

In [100]:

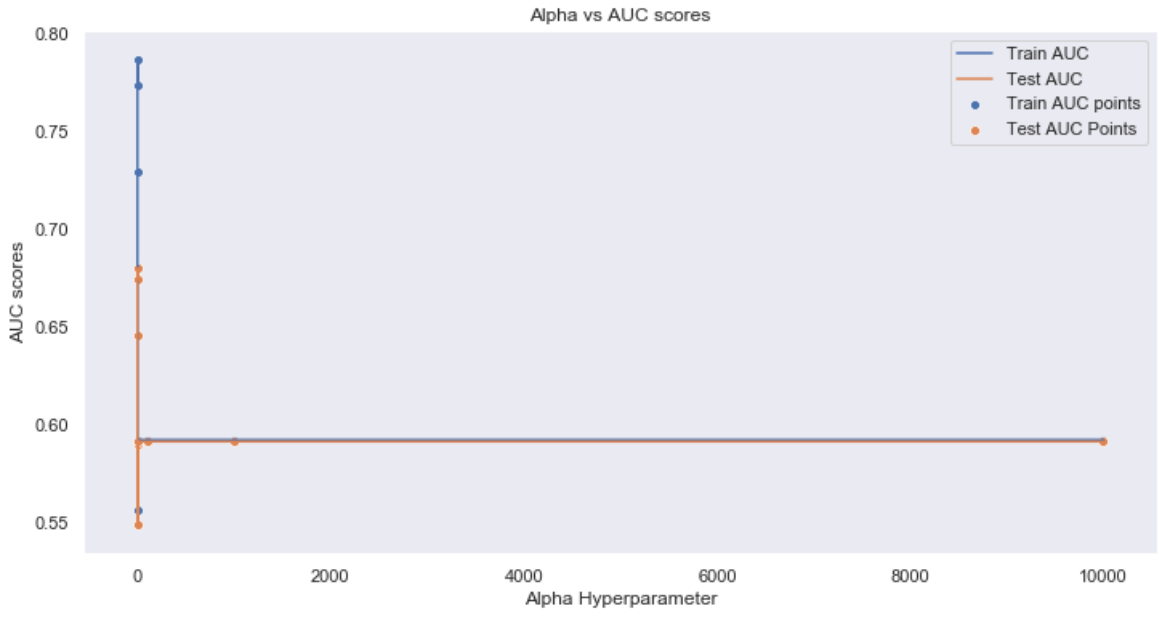
```
parameters={'alpha' : [1e-4,1e-3,1e-2,1e-1,1,1e+1,1e+2,1e+3,1e+4]}
model = SGDClassifier(loss = 'hinge',penalty='l2')
clf = GridSearchCV(model,param_grid=parameters,cv=5,scoring='roc_auc')
clf.fit(X_tr,y_train[:45000])
```

Out[100]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=SGDClassifier(alpha=0.0001, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10.0, 100.0, 1000.0, 10000.0]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [101]:

```
train_auc = clf.cv_results_['mean_train_score']
test_auc = clf.cv_results_['mean_test_score']
plt.figure(figsize = (12,6))
sns.lineplot(parameters['alpha'],train_auc,label = "Train AUC")
sns.lineplot(parameters['alpha'],test_auc,label = "Test AUC")
sns.scatterplot(parameters['alpha'],train_auc,label = 'Train AUC points')
sns.scatterplot(parameters['alpha'],test_auc,label = 'Test AUC Points')
sns.set()
plt.legend()
plt.xlabel("Alpha Hyperparameter")
plt.ylabel("AUC scores")
plt.title("Alpha vs AUC scores")
plt.grid()
plt.show()
```



From the above it's not clear which alpha estimator is good for the model

as we can see larger alpha value has no or less std or variance and Hence

we look out alpha parameters with small values

In [102]:

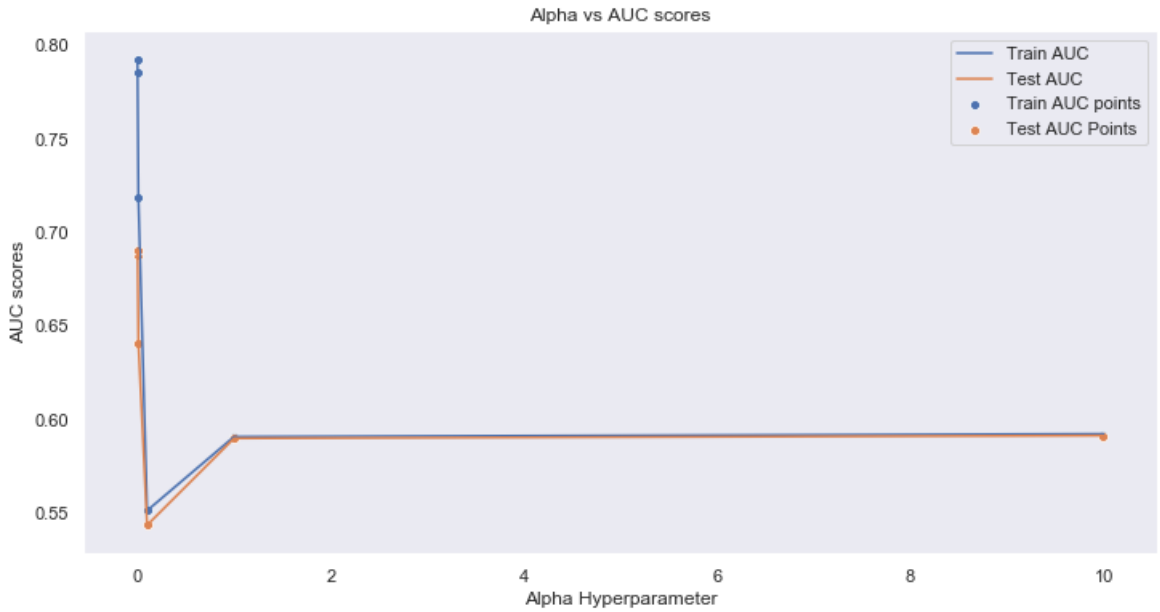
```
parameters={'alpha' :[1e-4,1e-3,1e-2,1e-1,1,1e+1] }
model = SGDClassifier(loss = 'hinge',penalty='l2')
clf = GridSearchCV(model,param_grid=parameters,cv=5,scoring='roc_auc')
clf.fit(X_tr,y_train[:45000])
```

Out[102]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=SGDClassifier(alpha=0.0001, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10.0]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [103]:

```
train_auc = clf.cv_results_['mean_train_score']
test_auc = clf.cv_results_['mean_test_score']
plt.figure(figsize = (12,6))
sns.lineplot(parameters['alpha'],train_auc,label = "Train AUC")
sns.lineplot(parameters['alpha'],test_auc,label = "Test AUC")
sns.scatterplot(parameters['alpha'],train_auc,label = 'Train AUC points')
sns.scatterplot(parameters['alpha'],test_auc,label = 'Test AUC Points')
sns.set()
plt.legend()
plt.xlabel("Alpha Hyperparameter")
plt.ylabel("AUC scores")
plt.title("Alpha vs AUC scores")
plt.grid()
plt.show()
```



We are getting our elbow graph so we tried to Zoom in further for value less than 1.0 using l2 penalty

In [104]:

```
parameters={'alpha' :[0.01,0.05,0.1,0.2,0.5,0.8,1.0] }
model = SGDClassifier(loss = 'hinge',penalty='l2')
clf = GridSearchCV(model,param_grid=parameters,cv=5,scoring='roc_auc')
clf.fit(X_tr,y_train[:45000])
```

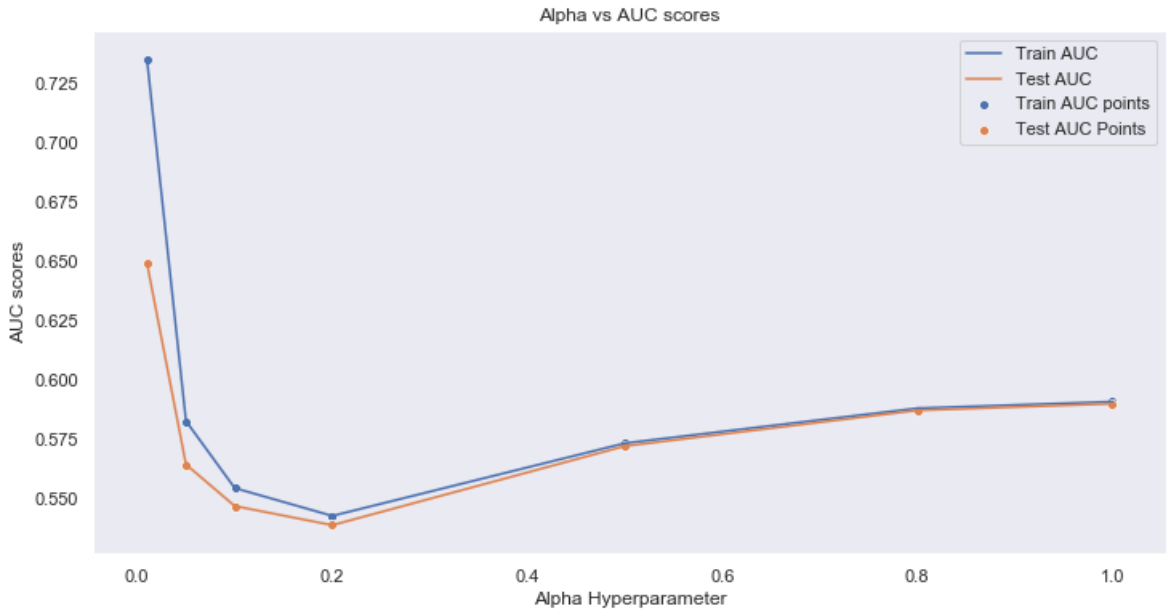
Out[104]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=SGDClassifier(alpha=0.0001, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'alpha': [0.01, 0.05, 0.1, 0.2, 0.5, 0.8, 1.0]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

Elbow Graph

In [105]:

```
train_auc = clf.cv_results_['mean_train_score']
test_auc = clf.cv_results_['mean_test_score']
plt.figure(figsize = (12,6))
sns.lineplot(parameters['alpha'],train_auc,label = "Train AUC")
sns.lineplot(parameters['alpha'],test_auc,label = "Test AUC")
sns.scatterplot(parameters['alpha'],train_auc,label = 'Train AUC points')
sns.scatterplot(parameters['alpha'],test_auc,label = 'Test AUC Points')
sns.set()
plt.legend()
plt.xlabel("Alpha Hyperparameter")
plt.ylabel("AUC scores")
plt.title("Alpha vs AUC scores")
plt.grid()
plt.show()
```



We are getting our elbow graph so we tried to Zoom in further for value less than 1.0 using l1 penalty

value less than 1.0 using l1 penalty

In [106]:

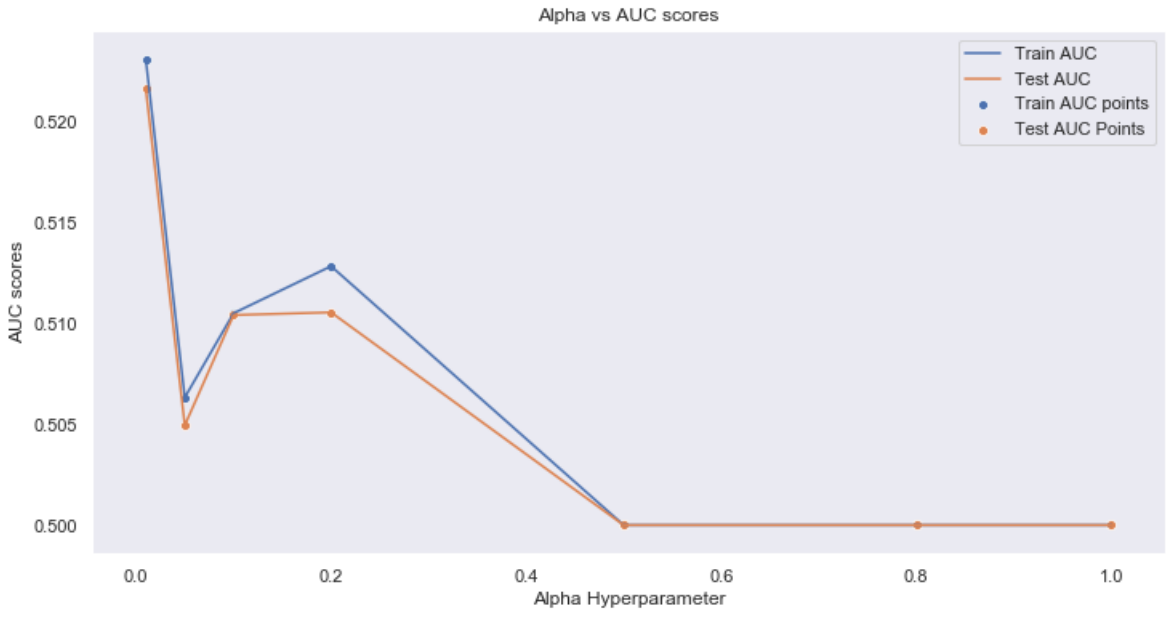
```
parameters={'alpha' :[0.01,0.05,0.1,0.2,0.5,0.8,1.0] }
model = SGDClassifier(loss = 'hinge',penalty='l1')
clf = GridSearchCV(model,param_grid=parameters,cv=5,scoring='roc_auc')
clf.fit(X_tr,y_train[:45000])
```

Out[106]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=SGDClassifier(alpha=0.0001, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l1',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'alpha': [0.01, 0.05, 0.1, 0.2, 0.5, 0.8, 1.0]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [107]:

```
train_auc = clf.cv_results_['mean_train_score']
test_auc = clf.cv_results_['mean_test_score']
plt.figure(figsize = (12,6))
sns.lineplot(parameters['alpha'],train_auc,label = "Train AUC")
sns.lineplot(parameters['alpha'],test_auc,label = "Test AUC")
sns.scatterplot(parameters['alpha'],train_auc,label = 'Train AUC points')
sns.scatterplot(parameters['alpha'],test_auc,label = 'Test AUC Points')
sns.set()
plt.legend()
plt.xlabel("Alpha Hyperparameter")
plt.ylabel("AUC scores")
plt.title("Alpha vs AUC scores")
plt.grid()
plt.show()
```



In [112]:

```
model = SGDClassifier(loss = 'hinge',alpha=0.01,penalty='l2')
model.fit(X_tr,y_train[:45000])
```

Out[112]:

```
SGDClassifier(alpha=0.01, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
```

```
power_t=0.5, random_state=None, shuffle=True, tol=None,
validation_fraction=0.1, verbose=0, warm_start=False)
```

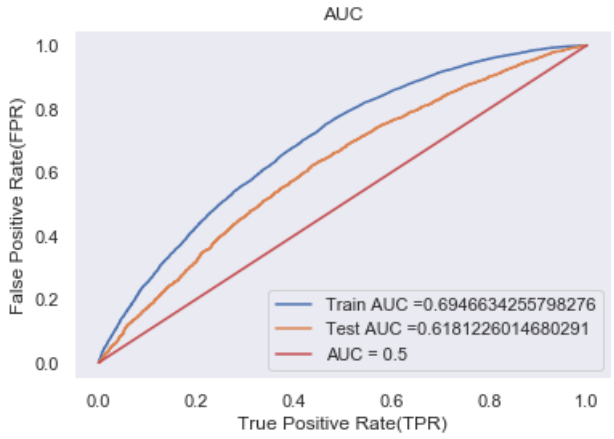
ROC_AUC Curve

In [113]:

```
y_train_pred = model.decision_function(X_tr)
y_test_pred = model.decision_function(X_ts)

fpr_train,tpr_train,thres_train = roc_curve(y_train[:45000], y_train_pred)
fpr_test,tpr_test,thres_test = roc_curve(y_test[:15000], y_test_pred)

plt.plot(fpr_train, tpr_train, label="Train AUC ="+str(auc(fpr_train, tpr_train)))
plt.plot(fpr_test, tpr_test, label="Test AUC ="+str(auc(fpr_test, tpr_test)))
plt.plot(np.linspace(0,1,600),np.linspace(0,1,600),label = "AUC = 0.5",color = "r")
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```

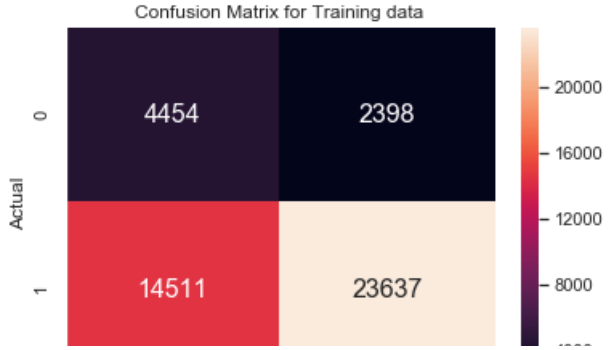


Confusion Matrix on training data with alpha = 0.01

In [114]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
ax = sns.heatmap(confusion_matrix(y_train[:45000],pred_using_threshold(y_train_pred,thres_train,tp
r_train,fpr_train)),
                 annot=True,annot_kws={"size": 16}, fmt='g')
ax.set_title("Confusion Matrix for Training data")
ax.set_xlabel("Predicted")
ax.set_ylabel("Actual")
sns.despine()
```

```
the maximum value of tpr*(1-fpr) 0.4091811496943633 for threshold 1.004
```

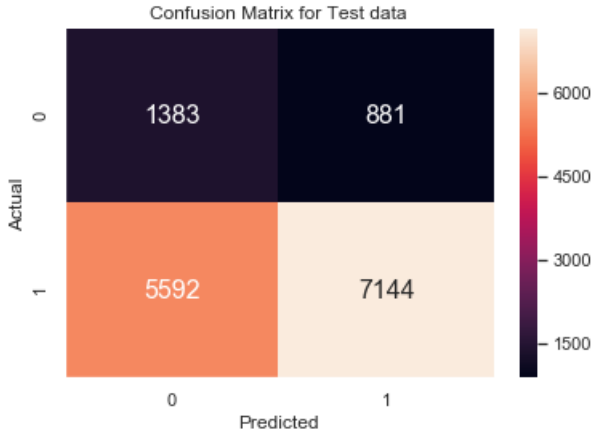


Confusion Matrix on test data with alpha = 0.01

In [115]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
ax =
sns.heatmap(confusion_matrix(y_test[:15000],pred_using_threshold(y_test_pred,thres_test,tpr_test,f
pr_test)),
                annot=True,annot_kws={"size": 16}, fmt='g')
ax.set_title("Confusion Matrix for Test data")
ax.set_xlabel("Predicted")
ax.set_ylabel("Actual")
sns.despine()
```

```
the maximum value of tpr*(1-fpr) 0.34973023798320224 for threshold 1.005
```



categorical, numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)

In [120]:

```
# Please write all the code with proper documentation
#from xgboost import XGBClassifier
#import xgboost as xgb
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
from scipy import sparse
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_tr =
hstack((categories_one_hot_train[:45000],sub_categories_one_hot_train[:45000],prefix_one_hot_train
[:45000],
project_grade_one_hot_train[:45000],state_one_hot_train[:45000],sparse.csr_matrix(price_standardize
d_train[:45000]),
        sparse.csr_matrix(quantity_standardized_train[:45000]),sparse.csr_matrix(project_standardiz
ed_train[:45000]),
               sparse.csr_matrix(Essay_count_standardized_train[:45000]),sparse.csr_matrix(title_co
unt_standardized_train[:45000]),
               avg_w2V_vectors_title_train[:45000],avg_w2v_vectors_essays_train[:45000])).tocsr()


X_ts =
hstack((categories_one_hot_test[:15000],sub_categories_one_hot_test[:15000],prefix_one_hot_test[:1
5000],
project_grade_one_hot_test[:15000],state_one_hot_test[:15000],sparse.csr_matrix(price_standardized_
test[:15000]),

sparse.csr_matrix(quantity_standardized_test[:15000]),sparse.csr_matrix(project_standardized_test[
```

```
:15000]),
        sparse.csr_matrix(Essay_count_standardized_test[:15000]),sparse.csr_matrix(title_count_stan
dardized_test[:15000]) ,
               avg_w2V_vectors_title_test[:15000],avg_w2v_vectors_essays_test[:15000])).tocsr()
```

Finding best Alpha value with best penalty among 'l1' and 'l2'

In [117]:

```
import matplotlib.pyplot as plt
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV
from sklearn.metrics import roc_auc_score,roc_curve,f1_score,auc
```

In [121]:

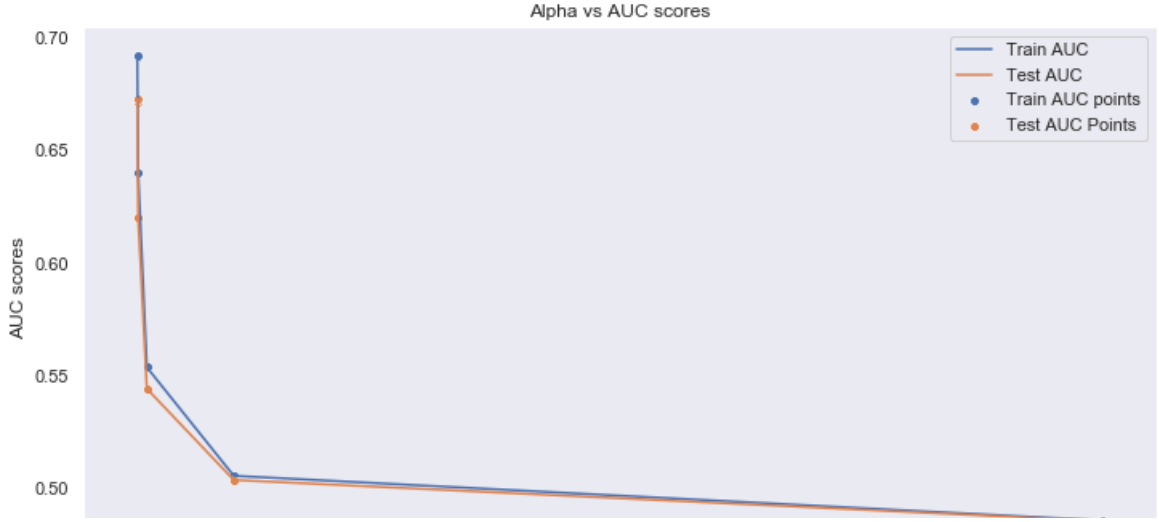
```
parameters={'alpha' :[1e-4,1e-3,1e-2,1e-1,1,1e+1] }
model = SGDClassifier(loss = 'hinge',penalty='l2')
clf = GridSearchCV(model,param_grid=parameters,cv=5,scoring='roc_auc')
clf.fit(X_tr,y_train[:45000])
```

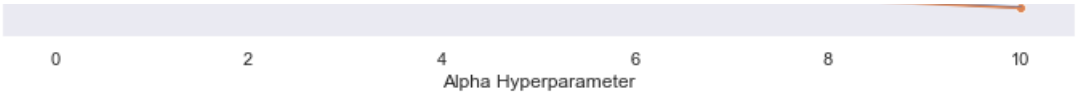
Out[121]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=SGDClassifier(alpha=0.0001, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10.0]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [122]:

```
train_auc = clf.cv_results_['mean_train_score']
test_auc = clf.cv_results_['mean_test_score']
plt.figure(figsize = (12,6))
sns.lineplot(parameters['alpha'],train_auc,label = "Train AUC")
sns.lineplot(parameters['alpha'],test_auc,label = "Test AUC")
sns.scatterplot(parameters['alpha'],train_auc,label = 'Train AUC points')
sns.scatterplot(parameters['alpha'],test_auc,label = 'Test AUC Points')
sns.set()
plt.legend()
plt.xlabel("Alpha Hyperparameter")
plt.ylabel("AUC scores")
plt.title("Alpha vs AUC scores")
plt.grid()
plt.show()
```





We are getting our elbow graph so we tried to Zoom in further for value less than 1.0 using l2 penalty

In [123]:

```
parameters={'alpha' :[0.01,0.05,0.1,0.2,0.5,0.8,1.0] }
model = SGDClassifier(loss = 'hinge',penalty='l2')
clf = GridSearchCV(model,param_grid=parameters,cv=5,scoring='roc_auc')
clf.fit(X_tr,y_train[:45000])
```

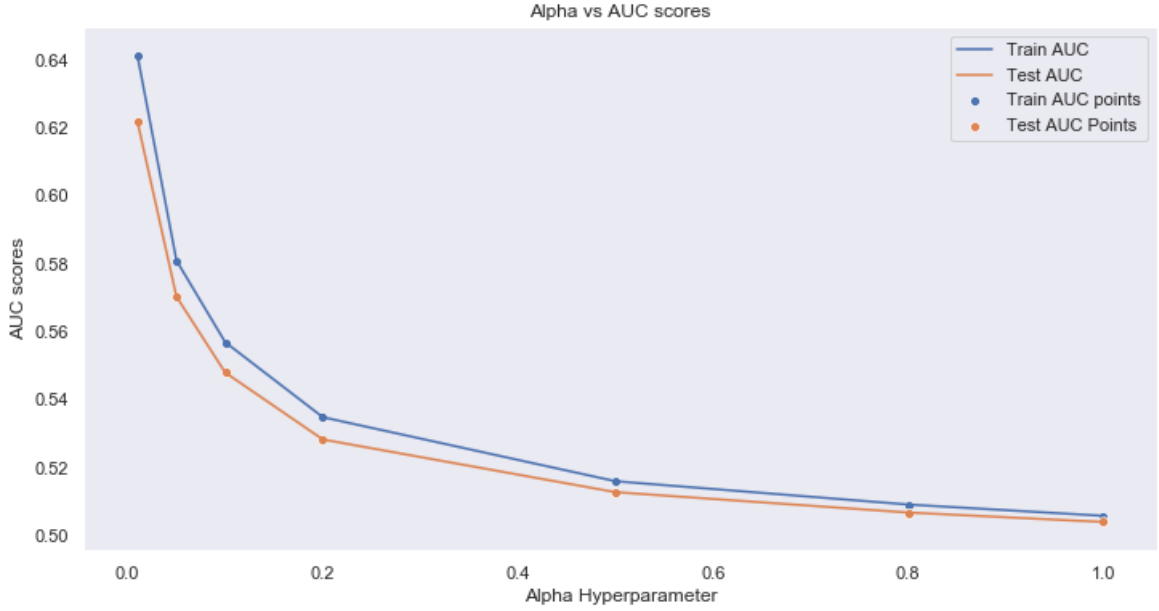
Out[123]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=SGDClassifier(alpha=0.0001, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'alpha': [0.01, 0.05, 0.1, 0.2, 0.5, 0.8, 1.0]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

Elbow Curve

In [124]:

```
train_auc = clf.cv_results_['mean_train_score']
test_auc = clf.cv_results_['mean_test_score']
plt.figure(figsize = (12,6))
sns.lineplot(parameters['alpha'],train_auc,label = "Train AUC")
sns.lineplot(parameters['alpha'],test_auc,label = "Test AUC")
sns.scatterplot(parameters['alpha'],train_auc,label = 'Train AUC points')
sns.scatterplot(parameters['alpha'],test_auc,label = 'Test AUC Points')
sns.set()
plt.legend()
plt.xlabel("Alpha Hyperparameter")
plt.ylabel("AUC scores")
plt.title("Alpha vs AUC scores")
plt.grid()
plt.show()
```



We are getting our elbow graph so we tried to Zoom in further for value less than 1.0 using l1 penalty

In [125]:

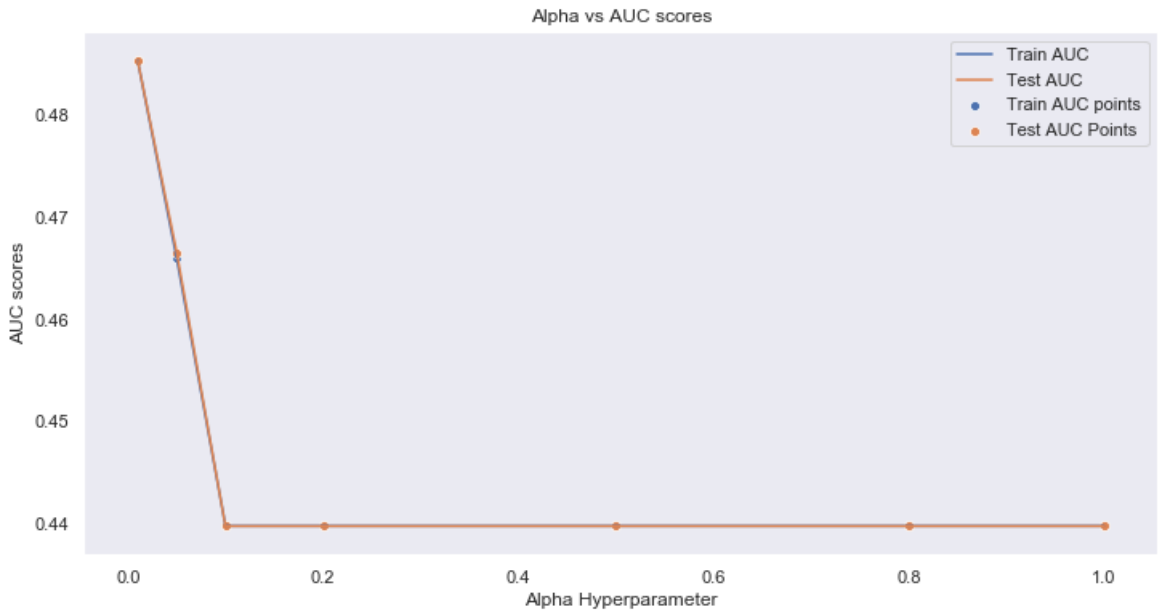
```
parameters={'alpha' :[0.01,0.05,0.1,0.2,0.5,0.8,1.0] }
model = SGDClassifier(loss = 'hinge',penalty='l1')
clf = GridSearchCV(model,param_grid=parameters,cv=5,scoring='roc_auc')
clf.fit(X_tr,y_train[:45000])
```

Out[125]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=SGDClassifier(alpha=0.0001, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l1',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'alpha': [0.01, 0.05, 0.1, 0.2, 0.5, 0.8, 1.0]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [126]:

```
train_auc = clf.cv_results_['mean_train_score']
test_auc = clf.cv_results_['mean_test_score']
plt.figure(figsize = (12,6))
sns.lineplot(parameters['alpha'],train_auc,label = "Train AUC")
sns.lineplot(parameters['alpha'],test_auc,label = "Test AUC")
sns.scatterplot(parameters['alpha'],train_auc,label = 'Train AUC points')
sns.scatterplot(parameters['alpha'],test_auc,label = 'Test AUC Points')
sns.set()
plt.legend()
plt.xlabel("Alpha Hyperparameter")
plt.ylabel("AUC scores")
plt.title("Alpha vs AUC scores")
plt.grid()
plt.show()
```



Finding Best estimator alpha with penalty = 'l2'

In [127]:

```
model = SGDClassifier(loss = 'hinge',alpha=0.01,penalty='l2')
model.fit(X_tr,y_train[:45000])
```

Out[127]:

```
SGDClassifier(alpha=0.01, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False)
```

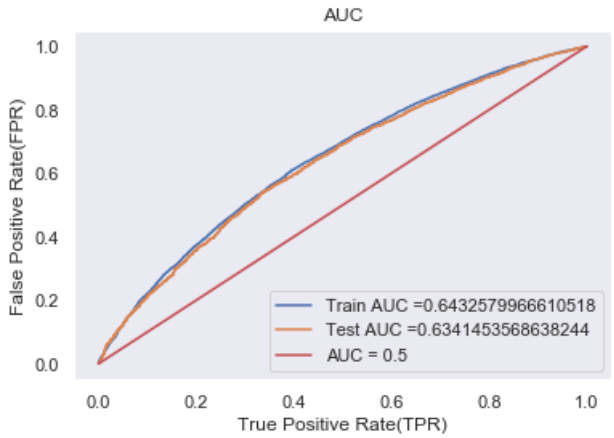
ROC_AUC Curve

In [128]:

```
y_train_pred = model.decision_function(X_tr)
y_test_pred = model.decision_function(X_ts)

fpr_train,tpr_train,thres_train = roc_curve(y_train[:45000], y_train_pred)
fpr_test,tpr_test,thres_test = roc_curve(y_test[:15000], y_test_pred)

plt.plot(fpr_train, tpr_train, label="Train AUC ="+str(auc(fpr_train, tpr_train)))
plt.plot(fpr_test, tpr_test, label="Test AUC ="+str(auc(fpr_test, tpr_test)))
plt.plot(np.linspace(0,1,600),np.linspace(0,1,600),label = "AUC = 0.5",color = "r")
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



In [137]:

```
model = SGDClassifier(loss = 'hinge',alpha=0.012,penalty='l2')
model.fit(X_tr,y_train[:45000])
```

Out[137]:

```
SGDClassifier(alpha=0.012, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False)
```

ROC_AUC Curve

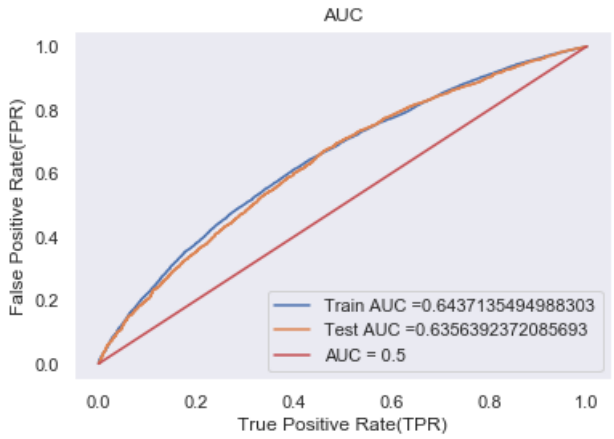
In [138]:

```
y_train_pred = model.decision_function(X_tr)
```

```
y_test_pred = model.decision_function(X_ts)

fpr_train,tpr_train,thres_train = roc_curve(y_train[:45000], y_train_pred)
fpr_test,tpr_test,thres_test = roc_curve(y_test[:15000], y_test_pred)

plt.plot(fpr_train, tpr_train, label="Train AUC ="+str(auc(fpr_train, tpr_train)))
plt.plot(fpr_test, tpr_test, label="Test AUC ="+str(auc(fpr_test, tpr_test)))
plt.plot(np.linspace(0,1,600),np.linspace(0,1,600),label = "AUC = 0.5",color = "r")
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```

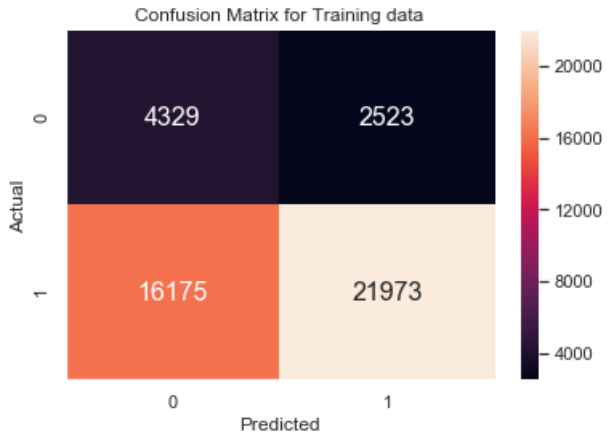


Confusion Matrix Using Training Data at alpha 0.012

In [139]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
ax = sns.heatmap(confusion_matrix(y_train[:45000],pred_using_threshold(y_train_pred,thres_train,tpr_train,fpr_train)),
                 annot=True,annot_kws={"size": 16}, fmt='g')
ax.set_title("Confusion Matrix for Training data")
ax.set_xlabel("Predicted")
ax.set_ylabel("Actual")
sns.despine()
```

```
the maximum value of tpr*(1-fpr) 0.36728580565653873 for threshold 1.04
```

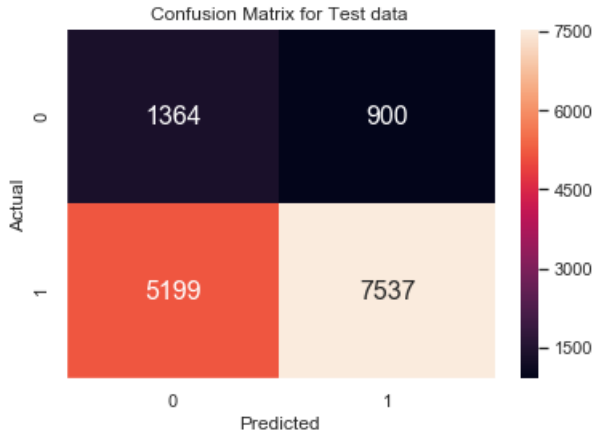


Confusion Matrix Using Test Data at alpha 0.012

In [140]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
ax =
sns.heatmap(confusion_matrix(y_test[:15000],pred_using_threshold(y_test_pred,thres_test,tpr_test,f
pr_test)),
                  annot=True,annot_kws={"size": 16}, fmt='g')
ax.set_title("Confusion Matrix for Test data")
ax.set_xlabel("Predicted")
ax.set_ylabel("Actual")
sns.despine()
```

```
the maximum value of tpr*(1-fpr) 0.36148831613899896 for threshold 1.038
```



categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)

In [141]:

```
# Please write all the code with proper documentation
#from xgboost import XGBClassifier
#import xgboost as xgb
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
from scipy import sparse
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_tr =
hstack((categories_one_hot_train[:45000],sub_categories_one_hot_train[:45000],prefix_one_hot_train
[:45000],
project_grade_one_hot_train[:45000],state_one_hot_train[:45000],sparse.csr_matrix(price_standardize
d_train[:45000]),
        sparse.csr_matrix(quantity_standardized_train[:45000]),sparse.csr_matrix(project_standardiz
ed_train[:45000]),
               sparse.csr_matrix(Essay_count_standardized_train[:45000]),sparse.csr_matrix(title_cc
unt_standardized_train[:45000]),
               tfidf_w2v_vectors_title_train[:45000],tfidf_w2v_vectors[:45000])).tocsr()


X_ts =
hstack((categories_one_hot_test[:15000],sub_categories_one_hot_test[:15000],prefix_one_hot_test[:1
5000],
project_grade_one_hot_test[:15000],state_one_hot_test[:15000],sparse.csr_matrix(price_standardized_
test[:15000]),

sparse.csr_matrix(quantity_standardized_test[:15000]),sparse.csr_matrix(project_standardized_test[
:15000]),
        sparse.csr_matrix(Essay_count_standardized_test[:15000]),sparse.csr_matrix(title_count_stan
dardized_test[:15000]),
               tfidf_w2v_vectors_title_test[:15000],tfidf_w2v_vectors_test[:15000])).tocsr()
```

Finding best Alpha value with best penalty among 'l1' and 'l2'

In [142]:

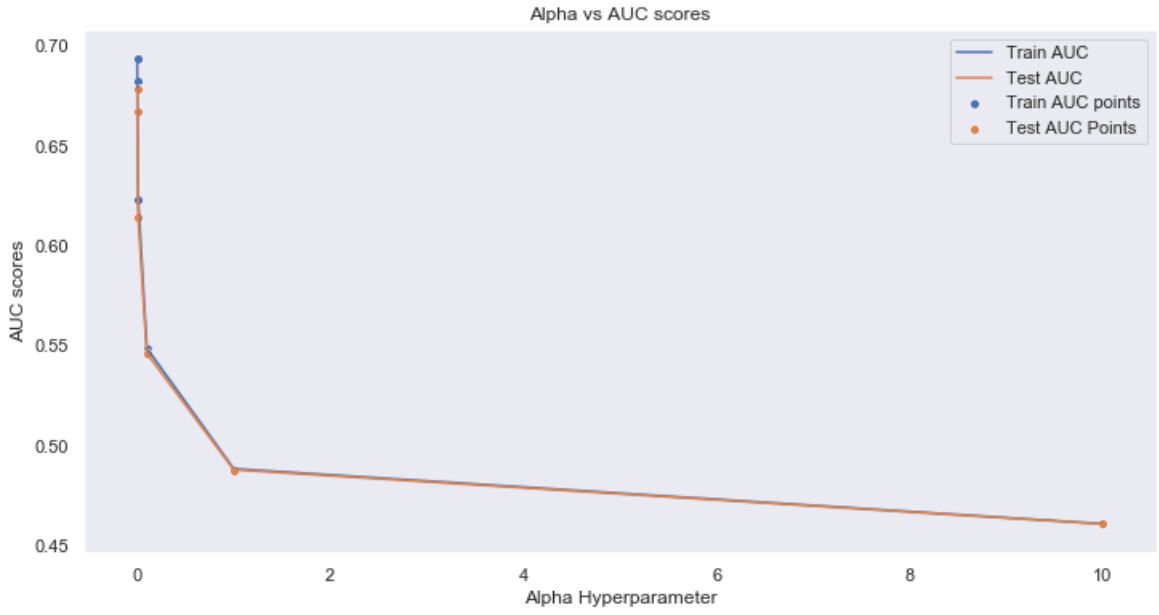
```
parameters={'alpha' :[1e-4,1e-3,1e-2,1e-1,1,1e+1] }
model = SGDClassifier(loss = 'hinge',penalty='l2')
clf = GridSearchCV(model,param_grid=parameters,cv=5,scoring='roc_auc')
clf.fit(X_tr,y_train[:45000])
```

Out[142]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=SGDClassifier(alpha=0.0001, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10.0]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [143]:

```
train_auc = clf.cv_results_['mean_train_score']
test_auc = clf.cv_results_['mean_test_score']
plt.figure(figsize = (12,6))
sns.lineplot(parameters['alpha'],train_auc,label = "Train AUC")
sns.lineplot(parameters['alpha'],test_auc,label = "Test AUC")
sns.scatterplot(parameters['alpha'],train_auc,label = 'Train AUC points')
sns.scatterplot(parameters['alpha'],test_auc,label = 'Test AUC Points')
sns.set()
plt.legend()
plt.xlabel("Alpha Hyperparameter")
plt.ylabel("AUC scores")
plt.title("Alpha vs AUC scores")
plt.grid()
plt.show()
```



We are getting our elbow graph so we tried to Zoom in further for value less than 1.0 using l2 penalty

In [144]:

```
parameters={'alpha' :[0.01,0.05,0.1,0.2,0.5,0.8,1.0] }
model = SGDClassifier(loss = 'hinge',penalty='l1')
clf = GridSearchCV(model,param_grid=parameters,cv=5,scoring='roc_auc')
clf.fit(X_tr,y_train[:45000])
```

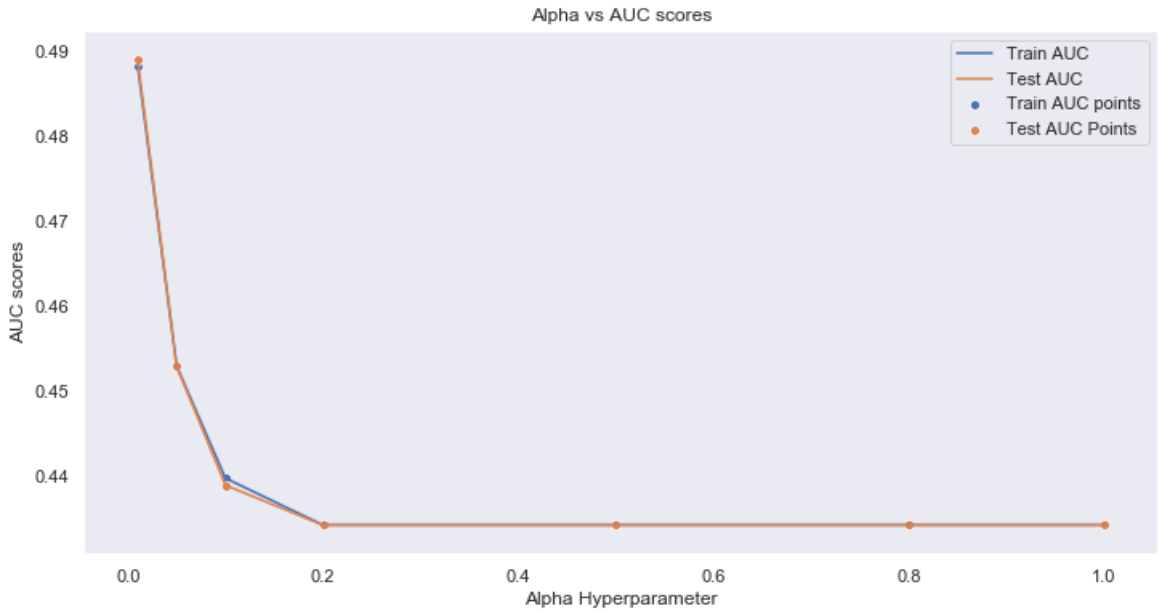
Out[144]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=SGDClassifier(alpha=0.0001, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l1',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'alpha': [0.01, 0.05, 0.1, 0.2, 0.5, 0.8, 1.0]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

Elbow Curve

In [145]:

```
train_auc = clf.cv_results_['mean_train_score']
test_auc = clf.cv_results_['mean_test_score']
plt.figure(figsize = (12,6))
sns.lineplot(parameters['alpha'],train_auc,label = "Train AUC")
sns.lineplot(parameters['alpha'],test_auc,label = "Test AUC")
sns.scatterplot(parameters['alpha'],train_auc,label = 'Train AUC points')
sns.scatterplot(parameters['alpha'],test_auc,label = 'Test AUC Points')
sns.set()
plt.legend()
plt.xlabel("Alpha Hyperparameter")
plt.ylabel("AUC scores")
plt.title("Alpha vs AUC scores")
plt.grid()
plt.show()
```



Finding Best estimator alpha with penalty = 'l2'

In [160]:

```
model = SGDClassifier(loss = 'hinge',alpha=0.0003,penalty='l2')
model.fit(X_tr,y_train[:45000])
```

Out[160]:

```
SGDClassifier(alpha=0.0003, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False)
```

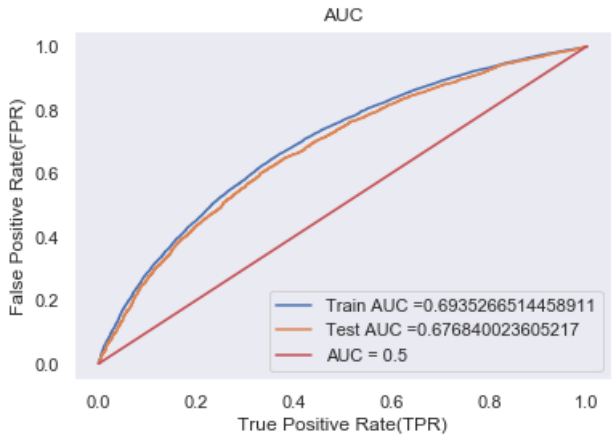
ROC_AUC Curve

In [161]:

```
y_train_pred = model.decision_function(X_tr)
y_test_pred = model.decision_function(X_ts)

fpr_train,tpr_train,thres_train = roc_curve(y_train[:45000], y_train_pred)
fpr_test,tpr_test,thres_test = roc_curve(y_test[:15000], y_test_pred)

plt.plot(fpr_train, tpr_train, label="Train AUC ="+str(auc(fpr_train, tpr_train)))
plt.plot(fpr_test, tpr_test, label="Test AUC ="+str(auc(fpr_test, tpr_test)))
plt.plot(np.linspace(0,1,600),np.linspace(0,1,600),label = "AUC = 0.5",color = "r")
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



In [168]:

```
model = SGDClassifier(loss = 'hinge',alpha=0.0002,penalty='l2')
model.fit(X_tr,y_train[:45000])
```

Out[168]:

```
SGDClassifier(alpha=0.0002, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False)
```

ROC_AUC CURVE

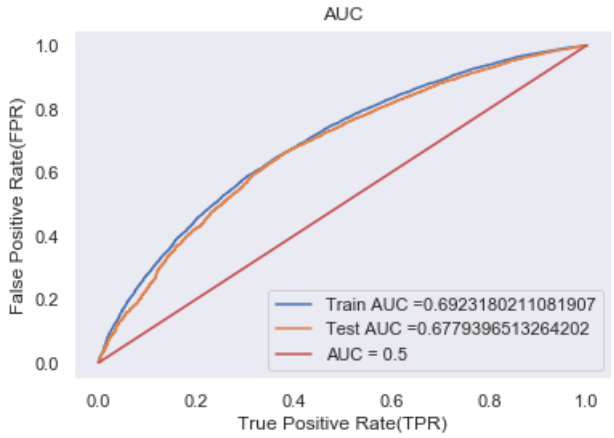
In [169]:

```
y_train_pred = model.decision_function(X_tr)
y_test_pred = model.decision_function(X_ts)

fpr_train,tpr_train,thres_train = roc_curve(y_train[:45000], y_train_pred)
fpr_test,tpr_test,thres_test = roc_curve(y_test[:15000], y_test_pred)

plt.plot(fpr_train, tpr_train, label="Train AUC ="+str(auc(fpr_train, tpr_train)))
plt.plot(fpr_test, tpr_test, label="Test AUC ="+str(auc(fpr_test, tpr_test)))
plt.plot(np.linspace(0,1,600),np.linspace(0,1,600),label = "AUC = 0.5",color = "r")
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
```

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



Model Fit with Penalty = 'l1'

In [164]:

```
model = SGDClassifier(loss = 'hinge',alpha=0.0002,penalty='l1')
model.fit(X_tr,y_train[:45000])
```

Out[164]:

```
SGDClassifier(alpha=0.0002, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l1',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False)
```

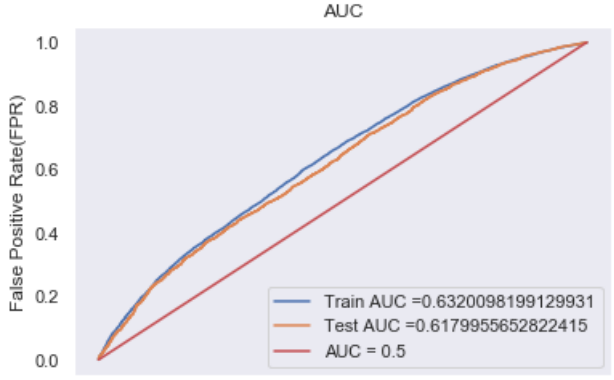
ROC_AUC Curve

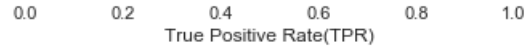
In [165]:

```
y_train_pred = model.decision_function(X_tr)
y_test_pred = model.decision_function(X_ts)

fpr_train,tpr_train,thres_train = roc_curve(y_train[:45000], y_train_pred)
fpr_test,tpr_test,thres_test = roc_curve(y_test[:15000], y_test_pred)

plt.plot(fpr_train, tpr_train, label="Train AUC ="+str(auc(fpr_train, tpr_train)))
plt.plot(fpr_test, tpr_test, label="Test AUC ="+str(auc(fpr_test, tpr_test)))
plt.plot(np.linspace(0,1,600),np.linspace(0,1,600),label = "AUC = 0.5",color = "r")
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



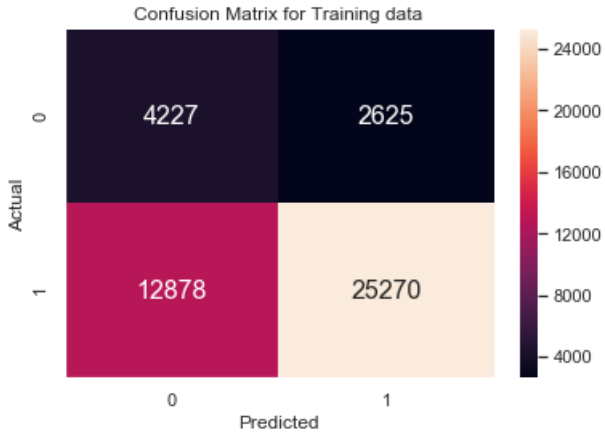


Confusion Matrix on training Data with alpha = 0.0002 and penalty = 'l2'

In [170]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
ax = sns.heatmap(confusion_matrix(y_train[:45000],pred_using_threshold(y_train_pred,thres_train,tpr_train,fpr_train)),
                 annot=True,annot_kws={"size": 16}, fmt='g')
ax.set_title("Confusion Matrix for Training data")
ax.set_xlabel("Predicted")
ax.set_ylabel("Actual")
sns.despine()
```

```
the maximum value of tpr*(1-fpr) 0.41158743826315436 for threshold 1.435
```

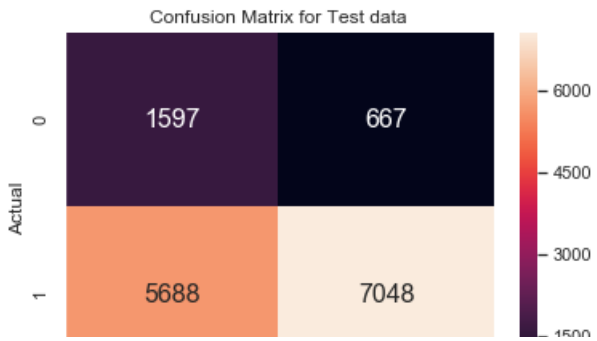


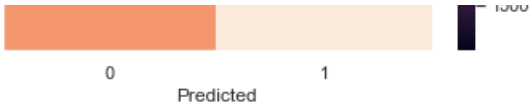
Confusion Matrix on test Data with alpha = 0.0002 and penalty = 'l2'

In [171]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
ax =
sns.heatmap(confusion_matrix(y_test[:15000],pred_using_threshold(y_test_pred,thres_test,tpr_test,fpr_test)),
                 annot=True,annot_kws={"size": 16}, fmt='g')
ax.set_title("Confusion Matrix for Test data")
ax.set_xlabel("Predicted")
ax.set_ylabel("Actual")
sns.despine()
```

```
the maximum value of tpr*(1-fpr) 0.41032459115364806 for threshold 1.574
```





2.5 Support Vector Machines with added Features `Set 5`

In [ ]:

```
# 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
```

Sentimental Analysis of Essay

In [29]:

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
senti = SentimentIntensityAnalyzer()
positive_tr,positive_ts,positive_cv=[],[],[]
negative_tr ,negative_ts,negative_cv= [],[],[]
neutral_tr,neutral_ts,neutral_cv = [],[],[]
comp_tr ,comp_ts,comp_cv= [],[],[]
```

In [31]:

```
for i in tqdm(X_train['essay']):
    positive_tr.append(senti.polarity_scores(i)['pos'])
    negative_tr.append(senti.polarity_scores(i)['neg'])
    neutral_tr.append(senti.polarity_scores(i)['neu'])
    comp_tr.append(senti.polarity_scores(i)['compound'])
```

```
100%|████████████████████████████████████████████████████████████████████████████| 87398/87398 [26
:27<00:00, 55.06it/s]
```

In [44]:

```
positive_tr = positive_tr[:87398]
negative_tr = negative_tr[:87398]
neutral_tr = neutral_tr[:87398]
comp_tr = comp_tr[:87398]
```

In [45]:

```
X_train['pos'] = positive_tr
X_train['neg'] = negative_tr
X_train['neu'] = neutral_tr
X_train['comp'] = comp_tr
```

In [46]:

```
for i in tqdm(X_test['essay']):
    positive_ts.append(senti.polarity_scores(i)['pos'])
    negative_ts.append(senti.polarity_scores(i)['neg'])
    neutral_ts.append(senti.polarity_scores(i)['neu'])
    comp_ts.append(senti.polarity_scores(i)['compound'])
```

```
100%|████████████████████████████████████████████████████████████████████████████| 21850/21850 [07
```

```
100%|████████████████████████████████████████████████████████████████████████████████| 21850/21850 [07
:15<00:00, 50.20it/s]
```

In [47]:

```
X_test['pos'] = positive_ts
X_test['neg'] = negative_ts
X_test['neu'] = neutral_ts
X_test['comp'] = comp_ts
```

Component Selection using TruncatedSVD on Tfidf Vectorizer

In [58]:

```
from sklearn.decomposition import TruncatedSVD
n_components = [2,5,10,50,100,200,300,500,1000,2000,4000]
var_sum = []
for c in n_components:
    truc_model = TruncatedSVD(n_components =c,random_state = 42,n_iter = 5 )
    truc_model.fit(essay_tfidf_train)
    var_sum.append(truc_model.explained_variance_ratio_.sum())
```

Out[58]:

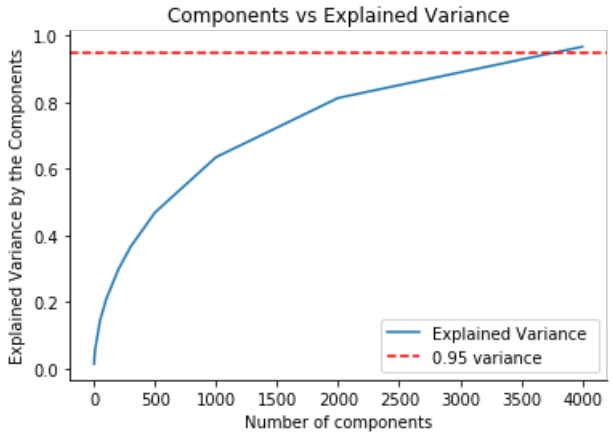
```
1
```

In [50]:

```
n_components = [2,5,10,50,100,200,300,500,1000,2000,4000]
```

In [56]:

```
plt.plot(n_components,var_sum,label = 'Explained Variance ')
plt.axhline(0.95,linestyle = '--',color = 'r',label = '0.95 variance')
plt.xlabel("Number of components")
plt.ylabel("Explained Variance by the Components")
plt.legend()
plt.title("Components vs Explained Variance")
sns.despine()
plt.show()
```



Train Data

In [38]:

```
from sklearn.decomposition import TruncatedSVD
svd_model = TruncatedSVD(n_components = 3500,n_iter = 3)
svd_model.fit(essay_tfidf_train)
svd_train = svd_model.transform(essay_tfidf_train)
```

Test Data

In [39]:

```
svd_test = svd_model.transform(essay_tfidf_test)
```

In [65]:

```
len(svd_train)
```

Out[65]:

45000

Using New feature

In [37]:

```
from scipy.sparse import hstack
from scipy import sparse
X_tr =
hstack((categories_one_hot_train[:45000],sub_categories_one_hot_train[:45000],prefix_one_hot_train
[:45000],
project_grade_one_hot_train[:45000],state_one_hot_train[:45000],sparse.csr_matrix(price_standardize
d_train[:45000]),
        sparse.csr_matrix(quantity_standardized_train[:45000]),sparse.csr_matrix(project_standardiz
ed_train[:45000]),
               sparse.csr_matrix(Essay_count_standardized_train[:45000]),sparse.csr_matrix(title_cc
unt_standardized_train[:45000])
      ,sparse.csr_matrix(np.array(positive_tr[:45000]).reshape(-1,1)),sparse.csr_matrix(np.array(ne
gative_tr[:45000]).reshape(-1,1)),sparse.csr_matrix(np.array(neutral_tr[:45000]).reshape(-1,1)),

sparse.csr_matrix(np.array(comp_tr[:45000]).reshape(-1,1)),sparse.csr_matrix(svd_train))).tocsr()



X_ts =
hstack((categories_one_hot_test[:15000],sub_categories_one_hot_test[:15000],prefix_one_hot_test[:1
5000],
project_grade_one_hot_test[:15000],state_one_hot_test[:15000],sparse.csr_matrix(price_standardized_
test[:15000]),

sparse.csr_matrix(quantity_standardized_test[:15000]),sparse.csr_matrix(project_standardized_test[
:15000]),
        sparse.csr_matrix(Essay_count_standardized_test[:15000]),sparse.csr_matrix(title_count_stan
dardized_test[:15000]),
       sparse.csr_matrix(np.array(positive_ts[:15000]).reshape(-1,1)),sparse.csr_matrix(np.array(ne
gative_ts[:15000]).reshape(-1,1)),
                sparse.csr_matrix(np.array(neutral_ts[:15000]).reshape(-1,1)),

sparse.csr_matrix(np.array(comp_ts[:15000]).reshape(-1,1)),sparse.csr_matrix(svd_test))).tocsr()
```

In [42]:

```
import matplotlib.pyplot as plt
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV
from sklearn.metrics import roc_auc_score,roc_curve,f1_score,auc
```

In [43]:

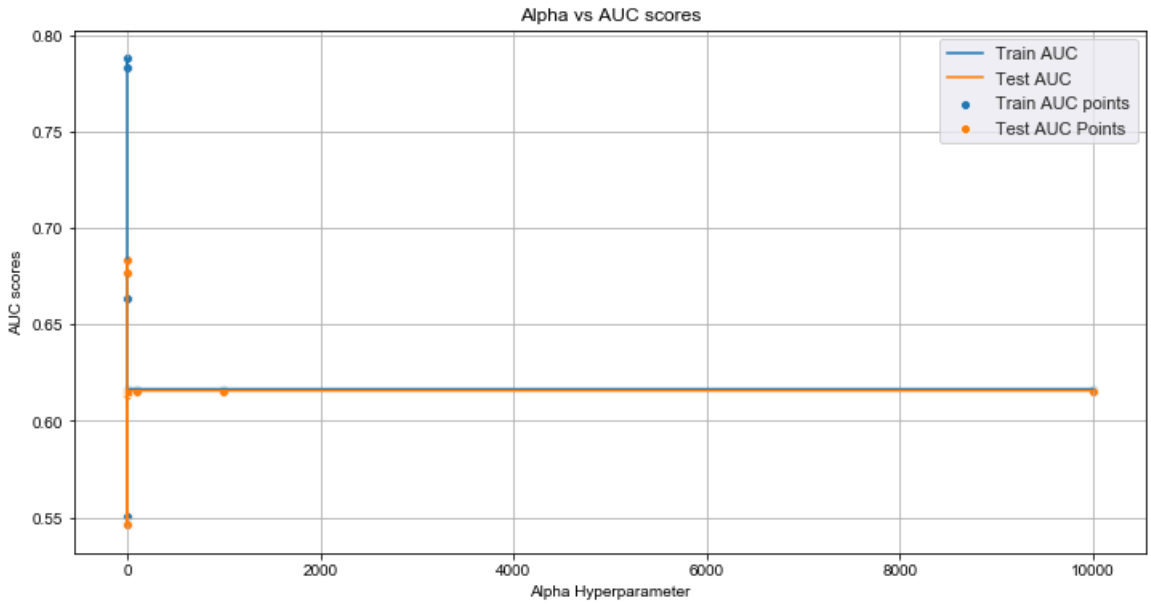
```
parameters={'alpha' :[1e-4,1e-3,1e-2,1e-1,1,1e+1,1e+2,1e+3,1e+4] }
model = SGDClassifier(loss = 'hinge',penalty='l2')
clf = GridSearchCV(model,param_grid=parameters,cv=5,scoring='roc_auc')
clf.fit(X_tr,y_train[:45000])
```

Out[43]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=SGDClassifier(alpha=0.0001, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10.0, 100.0, 1000.0, 10000.0]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [44]:

```
train_auc = clf.cv_results_['mean_train_score']
test_auc = clf.cv_results_['mean_test_score']
plt.figure(figsize = (12,6))
sns.lineplot(parameters['alpha'],train_auc,label = "Train AUC")
sns.lineplot(parameters['alpha'],test_auc,label = "Test AUC")
sns.scatterplot(parameters['alpha'],train_auc,label = 'Train AUC points')
sns.scatterplot(parameters['alpha'],test_auc,label = 'Test AUC Points')
sns.set()
plt.legend()
plt.xlabel("Alpha Hyperparameter")
plt.ylabel("AUC scores")
plt.title("Alpha vs AUC scores")
plt.grid()
plt.show()
```



Its hard to interpret the hyperparameter alpha from above graph so we dug in further into the smaller valus of alpha.

Finding best Alpha value with best penalty among 'l1' and 'l2'

In [60]:

```
parameters={'alpha' :[1e-4,1e-3,1e-2,1e-1,1,1e+1] }
model = SGDClassifier(loss = 'hinge',penalty='l2')
clf = GridSearchCV(model,param_grid=parameters,cv=5,scoring='roc_auc')
clf.fit(X_tr,y_train[:45000])
```

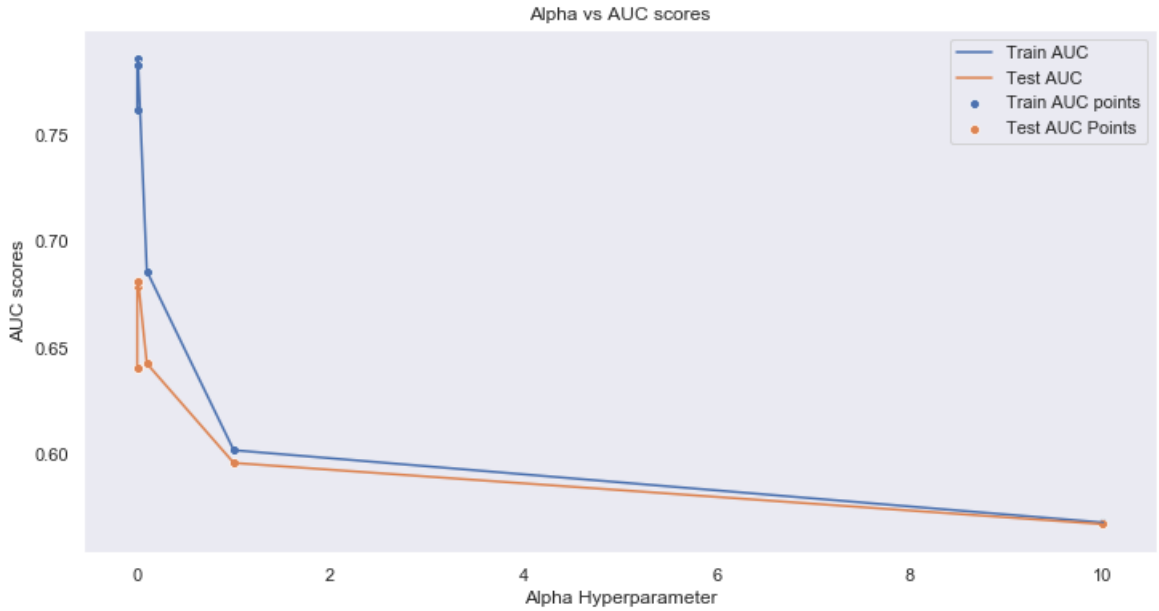
Out[60]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=SGDClassifier(alpha=0.0001, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
```

```
        n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
        power_t=0.5, random_state=None, shuffle=True, tol=None,
        validation_fraction=0.1, verbose=0, warm_start=False),
        fit_params=None, iid='warn', n_jobs=None,
        param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10.0]},
        pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
        scoring='roc_auc', verbose=0)
```

In [61]:

```
train_auc = clf.cv_results_['mean_train_score']
test_auc = clf.cv_results_['mean_test_score']
plt.figure(figsize = (12,6))
sns.lineplot(parameters['alpha'],train_auc,label = "Train AUC")
sns.lineplot(parameters['alpha'],test_auc,label = "Test AUC")
sns.scatterplot(parameters['alpha'],train_auc,label = 'Train AUC points')
sns.scatterplot(parameters['alpha'],test_auc,label = 'Test AUC Points')
sns.set()
plt.legend()
plt.xlabel("Alpha Hyperparameter")
plt.ylabel("AUC scores")
plt.title("Alpha vs AUC scores")
plt.grid()
plt.show()
```



We are getting our elbow graph so we tried to Zoom in further for value less than 1.0 using l2 penalty

In [64]:

```
parameters={'alpha' :[0.01,0.05,0.08,0.1,0.2,0.5,0.8,1.0] }
model = SGDClassifier(loss = 'hinge',penalty='l1')
clf = GridSearchCV(model,param_grid=parameters,cv=5,scoring='roc_auc')
clf.fit(X_tr,y_train[:45000])
```

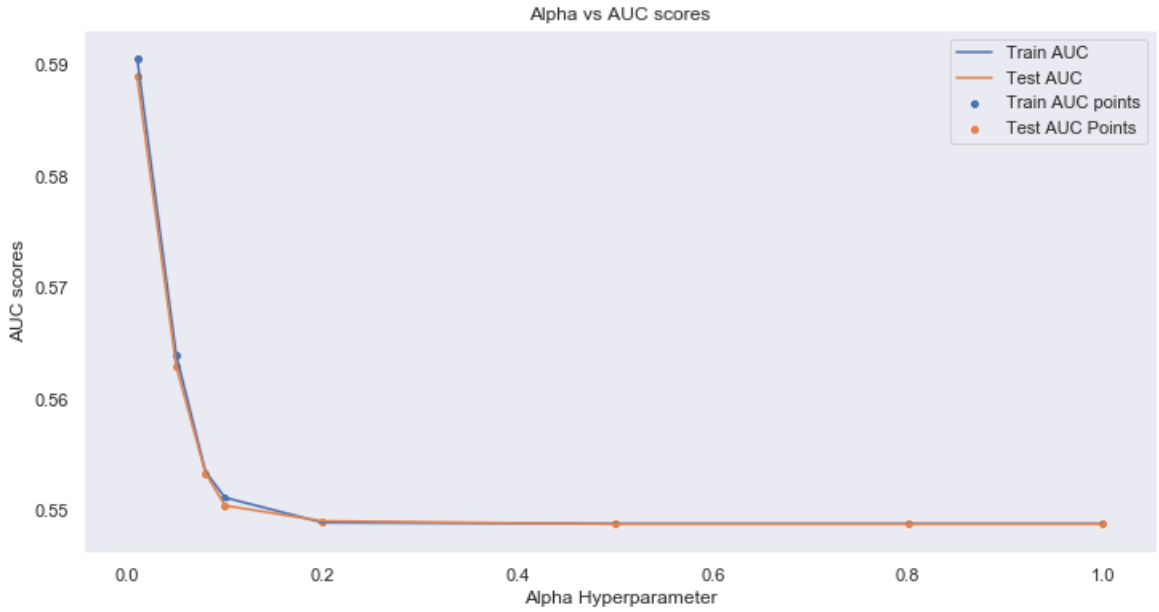
Out[64]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=SGDClassifier(alpha=0.0001, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l1',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'alpha': [0.01, 0.05, 0.08, 0.1, 0.2, 0.5, 0.8, 1.0]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

Elbow Curve

In [65]:

```
train_auc = clf.cv_results_['mean_train_score']
test_auc = clf.cv_results_['mean_test_score']
plt.figure(figsize = (12,6))
sns.lineplot(parameters['alpha'],train_auc,label = "Train AUC")
sns.lineplot(parameters['alpha'],test_auc,label = "Test AUC")
sns.scatterplot(parameters['alpha'],train_auc,label = 'Train AUC points')
sns.scatterplot(parameters['alpha'],test_auc,label = 'Test AUC Points')
sns.set()
plt.legend()
plt.xlabel("Alpha Hyperparameter")
plt.ylabel("AUC scores")
plt.title("Alpha vs AUC scores")
plt.grid()
plt.show()
```



Finding Best estimator alpha with penalty = 'l2'

In [57]:

```
model = SGDClassifier(loss = 'hinge',alpha=0.00012,penalty='l2')
model.fit(X_tr,y_train[:45000])
```

Out[57]:

```
SGDClassifier(alpha=0.00012, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False)
```

ROC_AUC Curve

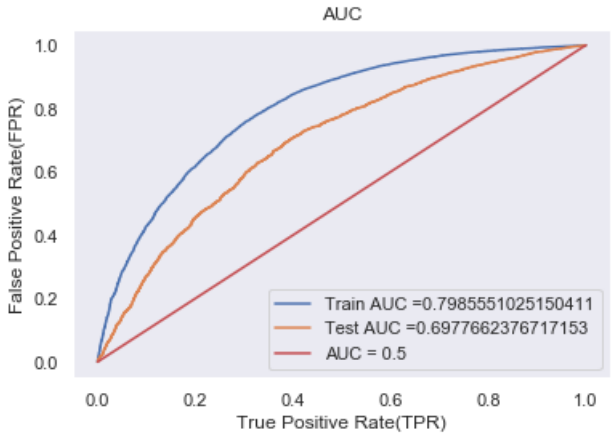
In [50]:

```
y_train_pred = model.decision_function(X_tr)
y_test_pred = model.decision_function(X_ts)

fpr_train,tpr_train,thres_train = roc_curve(y_train[:45000], y_train_pred)
```

```
fpr_test,tpr_test,thres_test = roc_curve(y_test[:15000], y_test_pred)

plt.plot(fpr_train, tpr_train, label="Train AUC ="+str(auc(fpr_train, tpr_train)))
plt.plot(fpr_test, tpr_test, label="Test AUC ="+str(auc(fpr_test, tpr_test)))
plt.plot(np.linspace(0,1,600),np.linspace(0,1,600),label = "AUC = 0.5",color = "r")
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



In [51]:

```
model = SGDClassifier(loss = 'hinge',alpha=0.00025,penalty='l2')
model.fit(X_tr,y_train[:45000])
```

Out[51]:

```
SGDClassifier(alpha=0.00025, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False)
```

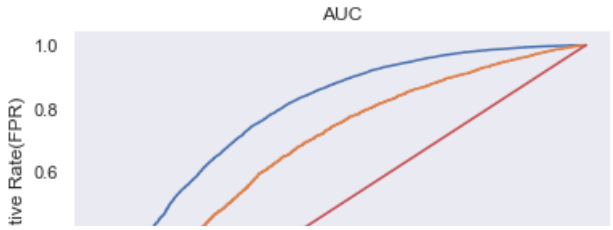
ROC_AUC Curve

In [52]:

```
y_train_pred = model.decision_function(X_tr)
y_test_pred = model.decision_function(X_ts)

fpr_train,tpr_train,thres_train = roc_curve(y_train[:45000], y_train_pred)
fpr_test,tpr_test,thres_test = roc_curve(y_test[:15000], y_test_pred)

plt.plot(fpr_train, tpr_train, label="Train AUC ="+str(auc(fpr_train, tpr_train)))
plt.plot(fpr_test, tpr_test, label="Test AUC ="+str(auc(fpr_test, tpr_test)))
plt.plot(np.linspace(0,1,600),np.linspace(0,1,600),label = "AUC = 0.5",color = "r")
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



Finding Best estimator alpha with penalty = 'l1'

In [55]:

```
model = SGDClassifier(loss = 'hinge',alpha=0.00012,penalty='l1')
model.fit(X_tr,y_train[:45000])
```

Out[55]:

```
SGDClassifier(alpha=0.00012, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l1',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False)
```

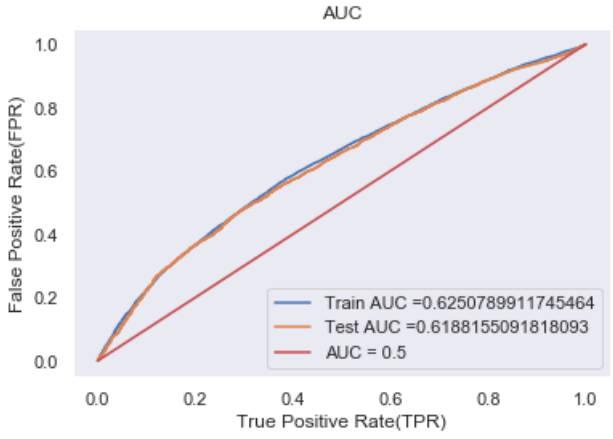
ROC_AUC Curve

In [56]:

```
y_train_pred = model.decision_function(X_tr)
y_test_pred = model.decision_function(X_ts)

fpr_train,tpr_train,thres_train = roc_curve(y_train[:45000], y_train_pred)
fpr_test,tpr_test,thres_test = roc_curve(y_test[:15000], y_test_pred)

plt.plot(fpr_train, tpr_train, label="Train AUC ="+str(auc(fpr_train, tpr_train)))
plt.plot(fpr_test, tpr_test, label="Test AUC ="+str(auc(fpr_test, tpr_test)))
plt.plot(np.linspace(0,1,600),np.linspace(0,1,600),label = "AUC = 0.5",color = "r")
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



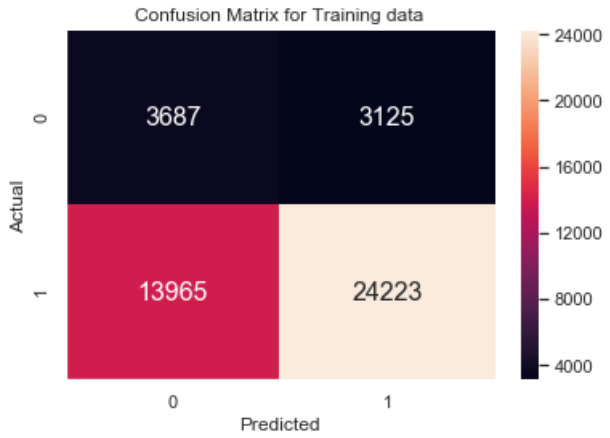
Confusion matrix using Training data at alpha = 0.00012

In [67]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
```

```
import seaborn as sns
ax = sns.heatmap(confusion_matrix(y_train[:45000],pred_using_threshold(y_train_pred,thres_train,tpr_train,fpr_train)),
                 annot=True,annot_kws={"size": 16}, fmt='g')
ax.set_title("Confusion Matrix for Training data")
ax.set_xlabel("Predicted")
ax.set_ylabel("Actual")
sns.despine()
```

```
the maximum value of tpr*(1-fpr) 0.3535713013855302 for threshold 1.02
```

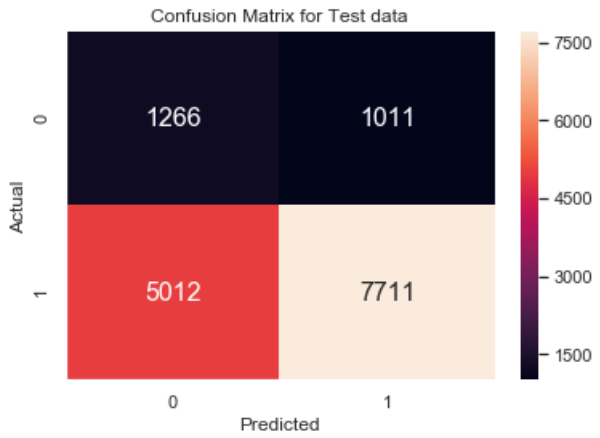


Confusion matrix using test data at alpha = 0.1

In [68]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
ax =
sns.heatmap(confusion_matrix(y_test[:15000],pred_using_threshold(y_test_pred,thres_test,tpr_test,fpr_test)),
                 annot=True,annot_kws={"size": 16}, fmt='g')
ax.set_title("Confusion Matrix for Test data")
ax.set_xlabel("Predicted")
ax.set_ylabel("Actual")
sns.despine()
```

```
the maximum value of tpr*(1-fpr) 0.3455029467967352 for threshold 1.025
```



3. Conclusion

In [69]:

```
# 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

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

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

x.add_row(["BOW", "SVM", 0.1, 0.634])
x.add_row(["TFIDF", "SVM", 0.01, 0.618])
x.add_row(["AVG W2V", "SVM", 0.01, 0.634])
x.add_row(["TFIDF W2V", "SVM", 0.0002, 0.677])
x.add_row(["With TruncatedSVD ", "SVM", 0.00012, 0.698])


print(x)
```

```
+--------------------+-------+-----------------------+-------+
|     Vectorizer     | Model | Alpha:Hyper Parameter |  AUC  |
+--------------------+-------+-----------------------+-------+
|        BOW         |  SVM  |          0.1          | 0.634 |
|       TFIDF        |  SVM  |          0.01         | 0.618 |
|      AVG W2V       |  SVM  |          0.01         | 0.634 |
|     TFIDF W2V      |  SVM  |         0.0002        | 0.677 |
| With TruncatedSVD  |  SVM  |        0.00012        | 0.698 |
+--------------------+-------+-----------------------+-------+
```