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	
`project_id`	A unique identifier for the proposed project
`project_title`	Title of t • `Art Wil` •
`project_grade_category`	Grade level of students for which the project is targete • • • •

Feature	
project_subject_categories	One or more (comma-separated) subject categories f following enur • Li • Literacy & Languag
school_state	State where school is located (Two-(https://en.wikipedia.org/wiki/List_of_U.S._state_abbrevia
project_subject_subcategories	One or more (comma-separated) subject subcate • Literature & Writing
project_resource_summary	An explanation of the resources needed for t • My students need hands on literacy ma sen
project_essay_1	F
project_essay_2	Sec
project_essay_3	Tl
project_essay_4	Fou
project_submitted_datetime	Datetime when project application was submitted. **Ex**
teacher_id	A unique identifier for the teacher of the propos bdf8baa8fedef6b
teacher_prefix	Teacher's title. One of the following
teacher_number_of_previously_posted_projects	Number of project applications previously submitted

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

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

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

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

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

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

Notes on the Essay Data

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

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

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

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

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

In [1]:
```
import os
os.chdir("F:/f/MY____/AAIC/AssignmentS/DonorsChoose_Data")
os.getcwd()
```

Out[1]:
```
'F:\\f\\MY____\\AAIC\\AssignmentS\\DonorsChoose_Data'
```

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
```

```
C:\Users\Anvesh\Anaconda3\lib\site-packages\gensim\utils.py:1197: UserWarning:
detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

1.1 Reading Data

In [4]:
```
project_data = pd.read_csv('train_data.csv')
resource_data = pd.read_csv('resources.csv')
```

In [5]:
```
print("Number of data points in train data", project_data.shape)
print('-'*50)
print("The attributes of data :", project_data.columns.values)
```

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

In [6]:
```
print("Number of data points in train data", resource_data.shape)
print(resource_data.columns.values)
resource_data.head(2)
```

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

Out[6]:

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

1.2 preprocessing of project_subject_categories

In [7]:
```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.co

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-
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",
        if 'The' in j.split(): # this will split each of the catogory based on sp
            j=j.replace('The','') # if we have the words "The" we are going to re
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty)
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trail
        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 [8]:
```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.c

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

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",
        if 'The' in j.split(): # this will split each of the catogory based on sp
            j=j.replace('The','') # if we have the words "The" we are going to re
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty)
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trail
        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/22898595/4
my_counter = Counter()
for word in project_data['clean_subcategories'].values:
    my_counter.update(word.split())

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

1.3 Text preprocessing

In [9]:
```
# 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 [10]:
```
project_data.head(2)
```

Out[10]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_sul
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	20
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	20

In [11]:
```
# printing some random reviews
print(project_data['essay'].values[0])
print("="*50)
print(project_data['essay'].values[150])
print("="*50)
print(project_data['essay'].values[1000])
print("="*50)
print(project_data['essay'].values[20000])
print("="*50)
print(project_data['essay'].values[99999])
print("="*50)
```

```
My students are English learners that are working on English as their second or
third languages. We are a melting pot of refugees, immigrants, and native-born
Americans bringing the gift of language to our school. \r\n\r\n We have over 24
languages represented in our English Learner program with students at every lev
el of mastery.  We also have over 40 countries represented with the families wi
thin our school.  Each student brings a wealth of knowledge and experiences to
us that open our eyes to new cultures, beliefs, and respect.\"The limits of you
r language are the limits of your world.\"-Ludwig Wittgenstein  Our English lea
rner's have a strong support system at home that begs for more resources.  Many
times our parents are learning to read and speak English along side of their ch
ildren.  Sometimes this creates barriers for parents to be able to help their c
hild learn phonetics, letter recognition, and other reading skills.\r\n\r\nBy p
roviding these dvd's and players, students are able to continue their mastery o
f the English language even if no one at home is able to assist.  All families
with students within the Level 1 proficiency status, will be a offered to be a
part of this program.  These educational videos will be specially chosen by the
English Learner Teacher and will be sent home regularly to watch.  The videos a
re to help the child develop early reading skills.\r\n\r\nParents that do not h
ave access to a dvd player will have the opportunity to check out a dvd player
to use for the year.  The plan is to use these videos and educational dvd's for
the years to come for other EL students.\r\nnannan
==================================================
The 51 fifth grade students that will cycle through my classroom this year all
love learning, at least most of the time. At our school, 97.3% of the students
receive free or reduced price lunch. Of the 560 students, 97.3% are minority st
udents. \r\nThe school has a vibrant community that loves to get together and c
elebrate. Around Halloween there is a whole school parade to show off the beaut
iful costumes that students wear. On Cinco de Mayo we put on a big festival wit
h crafts made by the students, dances, and games. At the end of the year the sc
hool hosts a carnival to celebrate the hard work put in during the school year,
with a dunk tank being the most popular activity.My students will use these fiv
e brightly colored Hokki stools in place of regular, stationary, 4-legged chair
s. As I will only have a total of ten in the classroom and not enough for each
student to have an individual one, they will be used in a variety of ways. Duri
ng independent reading time they will be used as special chairs students will e
ach use on occasion. I will utilize them in place of chairs at my small group t
ables during math and reading times. The rest of the day they will be used by t
he students who need the highest amount of movement in their life in order to s
tay focused on school.\r\n\r\nWhenever asked what the classroom is missing, my
students always say more Hokki Stools. They can't get their fill of the 5 stool
s we already have. When the students are sitting in group with me on the Hokki
Stools, they are always moving, but at the same time doing their work. Anytime
the students get to pick where they can sit, the Hokki Stools are the first to
be taken. There are always students who head over to the kidney table to get on
e of the stools who are disappointed as there are not enough of them. \r\n\r\nW
```

e ask a lot of students to sit for 7 hours a day. The Hokki stools will be a co mpromise that allow my students to do desk work and move at the same time. Thes e stools will help students to meet their 60 minutes a day of movement by allow ing them to activate their core muscles for balance while they sit. For many of my students, these chairs will take away the barrier that exists in schools for a child who can't sit still.nannan

==========

How do you remember your days of school? Was it in a sterile environment with p lain walls, rows of desks, and a teacher in front of the room? A typical day in our room is nothing like that. I work hard to create a warm inviting themed roo m for my students look forward to coming to each day.\r\n\r\nMy class is made u p of 28 wonderfully unique boys and girls of mixed races in Arkansas.\r\nThey a ttend a Title I school, which means there is a high enough percentage of free a nd reduced-price lunch to qualify. Our school is an \"open classroom\" concept, which is very unique as there are no walls separating the classrooms. These 9 a nd 10 year-old students are very eager learners; they are like sponges, absorbi ng all the information and experiences and keep on wanting more.With these reso urces such as the comfy red throw pillows and the whimsical nautical hanging de cor and the blue fish nets, I will be able to help create the mood in our class room setting to be one of a themed nautical environment. Creating a classroom e nvironment is very important in the success in each and every child's educatio n. The nautical photo props will be used with each child as they step foot into our classroom for the first time on Meet the Teacher evening. I'll take picture s of each child with them, have them developed, and then hung in our classroom ready for their first day of 4th grade. This kind gesture will set the tone be fore even the first day of school! The nautical thank you cards will be used th roughout the year by the students as they create thank you cards to their team groups.\r\n\r\nYour generous donations will help me to help make our classroom a fun, inviting, learning environment from day one.\r\n\r\nIt costs lost of mon ey out of my own pocket on resources to get our classroom ready. Please conside r helping with this project to make our new school year a very successful one. Thank you!nannan

==========

My kindergarten students have varied disabilities ranging from speech and langu age delays, cognitive delays, gross/fine motor delays, to autism. They are eage r beavers and always strive to work their hardest working past their limitation s. \r\n\r\nThe materials we have are the ones I seek out for my students. I tea ch in a Title I school where most of the students receive free or reduced price lunch. Despite their disabilities and limitations, my students love coming to school and come eager to learn and explore.Have you ever felt like you had ants in your pants and you needed to groove and move as you were in a meeting? This is how my kids feel all the time. The want to be able to move as they learn or so they say.Wobble chairs are the answer and I love then because they develop t heir core, which enhances gross motor and in Turn fine motor skills. \r\nThey a lso want to learn through games, my kids don't want to sit and do worksheets. T hey want to learn to count by jumping and playing. Physical engagement is the k ey to our success. The number toss and color and shape mats can make that happe n. My students will forget they are doing work and just have the fun a 6 year o ld deserves.nannan

==========

The mediocre teacher tells. The good teacher explains. The superior teacher dem onstrates. The great teacher inspires. -William A. Ward\r\n\r\nMy school has 80 3 students which is makeup is 97.6% African-American, making up the largest seg ment of the student body. A typical school in Dallas is made up of 23.2% Africa n-American students. Most of the students are on free or reduced lunch. We are n't receiving doctors, lawyers, or engineers children from rich backgrounds or

```
neighborhoods. As an educator I am inspiring minds of young children and we foc
us not only on academics but one smart, effective, efficient, and disciplined s
tudents with good character.In our classroom we can utilize the Bluetooth for s
wift transitions during class. I use a speaker which doesn't amplify the sound
enough to receive the message. Due to the volume of my speaker my students ca
n't hear videos or books clearly and it isn't making the lessons as meaningful.
But with the bluetooth speaker my students will be able to hear and I can stop,
pause and replay it at any time.\r\nThe cart will allow me to have more room fo
r storage of things that are needed for the day and has an extra part to it I c
an use.  The table top chart has all of the letter, words and pictures for stud
ents to learn about different letters and it is more accessible.nannan
==================================================
```

Introducing New Features

[Task-2] Apply SVM on the below feature set Set 5 by finding the best hyper parameter GridSearch

Consider these set of features for Set 5 in Assignment:

categorical dataschool_state
clean_categories....clean_subcategories....project_grade_category....teacher_prefix

numerical data quantity....teacher_number_of_previously_posted_projects....price

New Features

sentiment score's of each of the essay : **numerical data**
number of words in the title : **numerical data**
number of words in the combine essays : **numerical data**

In [13]:
```
new_title = []
for i in tqdm(project_data['project_title']):
    j = decontracted(i)
    new_title.append(j)
```

```
100%|██████████| 109248/109248 [00:01<00:00, 76561.16it/s]
```

In [14]:
```
#Introducing New Features
title_word_count = []
#for i in project_data['project_title']:
for i in tqdm(new_title):
    j = len(i.split())
    title_word_count.append(j)
    #print(j)
project_data['title_word_count'] = title_word_count
```

```
100%|██████████| 109248/109248 [00:00<00:00, 557471.56it/s]
```

In [15]:
```
project_data.head(2)
```

Out[15]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_sul
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	20
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	20

In [16]:
```
new_essay = []
for i in tqdm(project_data['essay']):
    j = decontracted(i)
    new_essay.append(j)
```

```
100%|██████████| 1
09248/109248 [00:02<00:00, 42905.56it/s]
```

In [17]:
```
essay_word_count = []
for i in tqdm(new_essay):
    j = len(i.split())
    essay_word_count.append(j)
    #print(j)
project_data['essay_word_count'] = essay_word_count
```

```
100%|██████████| 1
09248/109248 [00:02<00:00, 39025.79it/s]
```

In [20]:
```
project_data.head(2)
```

Out[20]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_sul
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	20
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	20

Computing Sentiment Scores

In [94]:
```
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 sr
ss = sid.polarity_scores(for_sentiment)

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

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

```
neg: 0.109, neu: 0.693, pos: 0.198, compound: 0.2023,
```

In [95]:
```
SID = SentimentIntensityAnalyzer()
#There is NEGITIVE and POSITIVE and NEUTRAL and COMPUND SCORES
#http://www.nltk.org/howto/sentiment.html

negitive = []
positive = []
neutral = []
compound = []
for i in tqdm(project_data['essay']):
    j = SID.polarity_scores(i)['neg']
    k = SID.polarity_scores(i)['neu']
    l = SID.polarity_scores(i)['pos']
    m = SID.polarity_scores(i)['compound']
    negitive.append(j)
    positive.append(k)
    neutral.append(l)
    compound.append(m)
```

```
100%|
| 109248/109248 [32:01<00:00, 56.85it/s]
```

In [96]:
```
project_data['negitive'] = negitive
project_data['positive'] = positive
project_data['neutral'] = neutral
project_data['compound'] = compound
```

In [97]:
```
project_data.head(2)
```

Out[97]:

project_essay_1	project_essay_2	...	clean_subcategories	essay	title_word_count	essay_word_co
My students are English learners that are work...	\"The limits of your language are the limits o...	...	ESL Literacy	My students are English learners that are work...	7	2
Our students arrive to our school eager to lea...	The projector we need for our school is very c...	...	Civics_Government TeamSports	Our students arrive to our school eager to lea...	5	2

1.4 Train_Test_Split

In [98]:
```
#Train Test Split
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(project_data, project_data["
                                                    test_size = 0.33, stratify =
                                                    random_state = 42)
```

In [99]:
```
#Train CV Split
x_train, x_cv, y_train, y_cv = train_test_split(x_train, y_train, test_size = 0.
                                                random_state = 42)
```

In [100]:
```
print(x_test.columns)
print(x_train.columns)
#print(x_cv.columns)
#print(x_train.shape)
#print(x_test.shape)
#print(x_cv.shape)
```

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category', 'project_title',
       'project_essay_1', 'project_essay_2', 'project_essay_3',
       'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'essay', 'title_word_count',
       'essay_word_count', 'price', 'quantity', 'negitive', 'positive',
       'neutral', 'compound'],
      dtype='object')
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category', 'project_title',
       'project_essay_1', 'project_essay_2', 'project_essay_3',
       'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'essay', 'title_word_count',
       'essay_word_count', 'price', 'quantity', 'negitive', 'positive',
       'neutral', 'compound'],
      dtype='object')
```

In [21]:
```
#Dropping Class Label in train test and cv data
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)
```

In [22]:
```
print(x_train.columns)
```

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category', 'project_title',
       'project_essay_1', 'project_essay_2', 'project_essay_3',
       'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'clean_categories',
       'clean_subcategories', 'essay', 'title_word_count', 'essay_word_count'],
      dtype='object')
```

In [ ]:

Preparing Data for Models

In [23]:
```
# 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 [24]:
```
sent = decontracted(project_data['essay'].values[20000])
print(sent)
print("="*50)
```

```
My kindergarten students have varied disabilities ranging from speech and langu
age delays, cognitive delays, gross/fine motor delays, to autism. They are eage
r beavers and always strive to work their hardest working past their limitation
s. \r\n\r\nThe materials we have are the ones I seek out for my students. I tea
ch in a Title I school where most of the students receive free or reduced price
lunch.  Despite their disabilities and limitations, my students love coming to
school and come eager to learn and explore.Have you ever felt like you had ants
in your pants and you needed to groove and move as you were in a meeting? This
is how my kids feel all the time. The want to be able to move as they learn or
so they say.Wobble chairs are the answer and I love then because they develop t
heir core, which enhances gross motor and in Turn fine motor skills. \r\nThey a
lso want to learn through games, my kids do not want to sit and do worksheets.
They want to learn to count by jumping and playing. Physical engagement is the
key to our success. The number toss and color and shape mats can make that happ
en. My students will forget they are doing work and just have the fun a 6 year
old deserves.nannan
==================================================
```

In [25]:
```
# \r \n \t remove from string python: http://texthandler.com/info/remove-line-bre
sent = sent.replace('\\r', ' ')
sent = sent.replace('\\"', ' ')
sent = sent.replace('\\n', ' ')
print(sent)
```

```
My kindergarten students have varied disabilities ranging from speech and langu
age delays, cognitive delays, gross/fine motor delays, to autism. They are eage
r beavers and always strive to work their hardest working past their limitation
s.     The materials we have are the ones I seek out for my students. I teach i
n a Title I school where most of the students receive free or reduced price lun
ch.  Despite their disabilities and limitations, my students love coming to sch
ool and come eager to learn and explore.Have you ever felt like you had ants in
your pants and you needed to groove and move as you were in a meeting? This is
how my kids feel all the time. The want to be able to move as they learn or so
they say.Wobble chairs are the answer and I love then because they develop thei
r core, which enhances gross motor and in Turn fine motor skills.   They also w
ant to learn through games, my kids do not want to sit and do worksheets. They
want to learn to count by jumping and playing. Physical engagement is the key t
o our success. The number toss and color and shape mats can make that happen. M
y students will forget they are doing work and just have the fun a 6 year old d
eserves.nannan
```

In [26]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
print(sent)
```

```
My kindergarten students have varied disabilities ranging from speech and langu
age delays cognitive delays gross fine motor delays to autism They are eager be
avers and always strive to work their hardest working past their limitations Th
e materials we have are the ones I seek out for my students I teach in a Title
I school where most of the students receive free or reduced price lunch Despite
their disabilities and limitations my students love coming to school and come e
ager to learn and explore Have you ever felt like you had ants in your pants an
d you needed to groove and move as you were in a meeting This is how my kids fe
el all the time The want to be able to move as they learn or so they say Wobble
chairs are the answer and I love then because they develop their core which enh
ances gross motor and in Turn fine motor skills They also want to learn through
games my kids do not want to sit and do worksheets They want to learn to count
by jumping and playing Physical engagement is the key to our success The number
toss and color and shape mats can make that happen My students will forget they
are doing work and just have the fun a 6 year old deserves nannan
```

In [27]:
```
# 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'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he',
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itse
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'tha
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'ha
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 't
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'of
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than'
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should'v
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "d
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma'
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn
            'won', "won't", 'wouldn', "wouldn't"]
```

Project_Essays preprocessing

In [28]:
```
#train_preprocessed_essays
# Combining all the above stundents
from tqdm import tqdm
train_preprocessed_essays = []
# 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)
    train_preprocessed_essays.append(sent.lower().strip())
```

```
100%|
| 49041/49041 [00:41<00:00, 1179.92it/s]
```

In [29]:
```
train_preprocessed_essays[10]
```

Out[29]:
```
'keeping visual arts classrooms passion i want share classroom in room add mean
s appliance door decor providing quality supplies foster appreciation desire pu
rsue art career i work title one school suburban neighborhood we provide many m
agnet programs including pbi positive behavior intervention preschool el err i
worked many students potential become artists i would like provide experiences
quality materials uniform colors brushes not shed bristles chance paint real ca
nvas i hardly wait share opportunity paint room 30 students not opportunity mix
colors create real canvas we start outdoor landscapes sky water reflection silh
ouetted trees begin three colors red white black mix need provide contrast high
lights 3 colors we cover vocabulary words tints shades hue line shape foregroun
d mid ground background practice plain construction paper create final product
canvas we begin implementation performance based art report card next year to p
rovide instruction color mixing techniques students could realize final project
real canvas would positive impact students may grow become design engineers arc
hitects fashion designers tomorrow'
```

In [30]:
```
# test_preprocessed_essay
from tqdm import tqdm
test_preprocessed_essays = []
# 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)
    test_preprocessed_essays.append(sent.lower().strip())
```

```
100%|
| 36052/36052 [00:31<00:00, 1151.48it/s]
```

In [31]:
```
test_preprocessed_essays[10]
```

Out[31]:
```
'hey i astronaut look says one kindergarten students no look i engineer says an
other another 5 year old shouts i love technology want computer person i grow t
his life eyes 5 6 year old public school boys girls they love learning explorin
g pretending want grow the children come variety ethnic social cultural economi
c backgrounds but children one thing common love learning especially math scien
ce our motto grown anything want learning anything possible just witnessing des
ire learn math science rewarding my students curious world need know technology
help identify real world become scientist math teacher engineer my students lov
e hands activities motivated technology they love learning new things enthusias
m radiates class this ipad case make huge difference way students learning math
science it let customize learning according students levels level level level i
t allow children opportunity understand real world live the children use free s
cience app brain pop jr see scientist work lab astronauts go outer space develo
p computer programs help keep earth free pollution they use free math app sushi
monster strengthen reasoning skills learn strategies solving math problems unde
rstanding patterns world this ipad allow show students stem activites used diff
erentiate instruction students bring real world classroom they learn work addit
ion math problems 5 3 2 10 seeing robot build tower learn planet mars water see
ing pictures form hubble telescope learn ipad show patterns exist throughout wo
rld this ipad used kindergarten math science exploration center the children pi
ck ipad go free stem app like brain pop jr using fingers activate game stimulat
e enrich learning math science nannan'
```

In [32]:
```
# CV_preprocessed_essays
from tqdm import tqdm
cv_preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(x_cv['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    cv_preprocessed_essays.append(sent.lower().strip())
```

```
100%|
| 24155/24155 [00:21<00:00, 1148.30it/s]
```

In [33]:
```
cv_preprocessed_essays[10]
```

Out[33]:
```
'when asked music means student told music makes feel happy it changed i not an
ything express feelings i this hope every student joins class my biggest challe
nge providing everything need learn i blessed able work wonderful students they
hunger music insatiable each day i able go work inspired dedication hard work s
tudents put everything our school made mission build culture embraces musical l
earning give every student opportunity participate music our school low income
area best provide much needed supplies possible music program keeps growing str
uggling keep there nothing worse hearing violins played tune it difficult begin
ning violin students tune strings not always hear differences sound this means
teacher often ends tuning instruments taking valuable learning time if students
pitch pipes would able play individual notes match violin strings sounds this m
eans could also fix violins get tune practicing home resulting better practice
improvement individual playing nannan'
```

Project_Titles_preprocessing

In [34]:
```
# train_preprocessed_title
from tqdm import tqdm
train_preprocessed_titles = []
# tqdm is for printing the status bar
for sentance in tqdm(x_train['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    train_preprocessed_titles.append(sent.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████████████████|
49041/49041 [00:01<00:00, 26875.36it/s]
```

In [35]:
```
train_preprocessed_titles[10]
```

Out[35]:
```
'we all have rembrandt inside us'
```

In [36]:
```
# Test_preprocessed_essays
from tqdm import tqdm
test_preprocessed_titles = []
# tqdm is for printing the status bar
for sentance in tqdm(x_test['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    test_preprocessed_titles.append(sent.lower().strip())
```

```
100%|██████████| 
36052/36052 [00:01<00:00, 26984.53it/s]
```

In [37]:
```
test_preprocessed_titles[10]
```

Out[37]:
```
'ipads ipads exploring'
```

In [38]:
```
# CV_preprocessed_titles
from tqdm import tqdm
cv_preprocessed_titles = []
# tqdm is for printing the status bar
for sentance in tqdm(x_cv['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    cv_preprocessed_titles.append(sent.lower().strip())
```

```
100%|██████████| 
24155/24155 [00:00<00:00, 26552.02it/s]
```

In [39]:
```
# after preprocesing
cv_preprocessed_titles[10]
```

Out[39]:
```
'help us play our violins in tune'
```

In [40]:
```
project_data.columns
```

Out[40]:
```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category', 'project_title',
       'project_essay_1', 'project_essay_2', 'project_essay_3',
       'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'essay', 'title_word_count',
       'essay_word_count'],
      dtype='object')
```

we are going to consider

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

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

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

1.5.1 Vectorizing Categorical data

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

In [41]:
```
# we use count vectorizer to convert the values into one
# Vectorizing Clean Categories
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=

vectorizer.fit(x_train['clean_categories'].values)
train_categories_one_hot = vectorizer.transform(x_train['clean_categories'].valu
test_categories_one_hot = vectorizer.transform(x_test['clean_categories'].values
cv_categories_one_hot = vectorizer.transform(x_cv['clean_categories'].values)

print(vectorizer.get_feature_names())
print("Shape of  Train matrix after one hot encodig ",train_categories_one_hot.sh
print("Shape of  Test matrix after one hot encodig ",test_categories_one_hot.sha
print("Shape of  cv  matrix after one hot encodig ",cv_categories_one_hot.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'S
pecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of  Train matrix after one hot encodig  (49041, 9)
Shape of  Test matrix after one hot encodig  (36052, 9)
Shape of  cv  matrix after one hot encodig  (24155, 9)
```

In [42]:
```
# we use count vectorizer to convert the values into one
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowerc
vectorizer.fit(x_train["clean_subcategories"].values)
train_sub_categories_one_hot = vectorizer.transform(x_train['clean_subcategories
test_sub_categories_one_hot = vectorizer.transform(x_test['clean_subcategories']
cv_sub_categories_one_hot = vectorizer.transform(x_cv['clean_subcategories'].valu

print(vectorizer.get_feature_names())
print("Shape of Train matrix after  Trainone hot encodig ",train_sub_categories_
print("Shape of test matrix after one hot encodig ",test_sub_categories_one_hot.
print("Shape of cv_ matrix after  one hot encodig ",cv_sub_categories_one_hot.sha
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Ex
tracurricular', 'Civics_Government', 'ForeignLanguages', 'NutritionEducation',
'Warmth', 'Care_Hunger', 'SocialSciences', 'PerformingArts', 'CharacterEducatio
n', 'TeamSports', 'Other', 'College_CareerPrep', 'Music', 'History_Geography',
'Health_LifeScience', 'EarlyDevelopment', 'ESL', 'Gym_Fitness', 'EnvironmentalS
cience', 'VisualArts', 'Health_Wellness', 'AppliedSciences', 'SpecialNeeds', 'L
iterature_Writing', 'Mathematics', 'Literacy']
Shape of Train matrix after  Trainone hot encodig  (49041, 30)
Shape of test matrix after one hot encodig  (36052, 30)
Shape of cv_ matrix after  one hot encodig  (24155, 30)
```

In [43]:
```
# you can do the similar thing with state, teacher_prefix and project_grade_categ
from collections import Counter
my_counter = Counter()
for word in project_data["school_state"].values:
    my_counter.update(word.split())
```

In [44]:
```
# dict sort by value python: https://stackoverflow.com/a/613218/4084039
state_cat_dict = dict(my_counter)
storted_state_cat_dict = dict(sorted(state_cat_dict.items(), key=lambda kv: kv[1
```

In [45]:
```
# Please do the similar feature encoding with state, teacher_prefix and project_g
#Using Count Vectorizer to convert the state value onto on hot encoded feature
vectorizer = CountVectorizer(vocabulary=list(storted_state_cat_dict.keys()), lowe
vectorizer.fit(project_data['school_state'].values)
print(vectorizer.get_feature_names())


train_state_one_hot = vectorizer.transform(x_train['school_state'].values)
test_state_one_hot = vectorizer.transform(x_test['school_state'].values)
cv_state_one_hot = vectorizer.transform(x_cv['school_state'].values)

print("Shape of Train matrix after one hot encodig ",train_state_one_hot.shape)
print("Shape of Test matrix after one hot encodig ",test_state_one_hot.shape)
print("Shape of CV matrix after one hot encodig ",cv_state_one_hot.shape)
```

```
['VT', 'WY', 'ND', 'MT', 'RI', 'SD', 'NE', 'DE', 'AK', 'NH', 'WV', 'ME', 'HI',
'DC', 'NM', 'KS', 'IA', 'ID', 'AR', 'CO', 'MN', 'OR', 'KY', 'MS', 'NV', 'MD',
'CT', 'TN', 'UT', 'AL', 'WI', 'VA', 'AZ', 'NJ', 'OK', 'WA', 'MA', 'LA', 'OH',
'MO', 'IN', 'PA', 'MI', 'SC', 'GA', 'IL', 'NC', 'FL', 'NY', 'TX', 'CA']
Shape of Train matrix after one hot encodig  (49041, 51)
Shape of Test matrix after one hot encodig  (36052, 51)
Shape of CV matrix after one hot encodig  (24155, 51)
```

In [46]:
```
#https://stackoverflow.com/questions/42224700/attributeerror-float-object-has-no
project_data['project_grade_category']=project_data['project_grade_category'].fi
my_counter = Counter()
for word in project_data['project_grade_category'].values:
    my_counter.update(word.split())
```

In [47]:
```
project_cat_dict = dict(my_counter)
sorted_project_cat_dict = dict(sorted(project_cat_dict.items(), key=lambda kv: k
```

In [48]:
```
# feature encoding for project_grade_category also
vectorizer = CountVectorizer(vocabulary=list(sorted_project_cat_dict.keys()), lo
vectorizer.fit(project_data['project_grade_category'].values)
print(vectorizer.get_feature_names())

train_grade_one_hot = vectorizer.transform(x_train['project_grade_category'].val
test_grade_one_hot = vectorizer.transform(x_test['project_grade_category'].value
cv_grade_one_hot = vectorizer.transform(x_cv['project_grade_category'].values)

print("Shape of Train matrix after one hot encodig ",train_grade_one_hot.shape)
print("Shape of test matrix after one hot encodig ",test_grade_one_hot.shape)
print("Shape of cv matrix after one hot encodig ",cv_grade_one_hot.shape)
```

```
['9-12', '6-8', '3-5', 'PreK-2', 'Grades']
Shape of Train matrix after one hot encodig  (49041, 5)
Shape of test matrix after one hot encodig  (36052, 5)
Shape of cv matrix after one hot encodig  (24155, 5)
```

In [49]:
```
#https://stackoverflow.com/questions/42224700/attributeerror-float-object-has-no
project_data['teacher_prefix']=project_data['teacher_prefix'].fillna(" ")
```

In [50]:
```
my_counter = Counter()
for word in project_data['teacher_prefix'].values:
    my_counter.update(word.split())
```

In [51]:
```
# dict sort by value python: https://stackoverflow.com/a/613218/4084039
teacher_cat_dict = dict(my_counter)
sorted_teacher_cat_dict = dict(sorted(teacher_cat_dict.items(), key=lambda kv: k
```

In [52]:
```
#Using Count Vectorizer to convert the teacher_prefix value onto on hot encoded f
#ValueError: np.nan is an invalid document, expected byte or unicode string.
#https://stackoverflow.com/questions/39303912/tfidfvectorizer-in-scikit-learn-va
vectorizer = CountVectorizer(vocabulary=list(sorted_teacher_cat_dict.keys()), lo
vectorizer.fit(project_data['teacher_prefix'].values.astype('U'))
print(vectorizer.get_feature_names())

train_teacher_one_hot = vectorizer.transform(x_train['teacher_prefix'].values.as
test_teacher_one_hot = vectorizer.transform(x_test['teacher_prefix'].values.asty
cv_teacher_one_hot = vectorizer.transform(x_cv['teacher_prefix'].values.astype('

print("Shape of Train matrix after one hot encodig ",train_teacher_one_hot.shape
print("Shape of Test matrix after one hot encodig ",test_teacher_one_hot.shape)
print("Shape of CV matrix after one hot encodig ",cv_teacher_one_hot.shape)
```

```
['Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
Shape of Train matrix after one hot encodig  (49041, 5)
Shape of Test matrix after one hot encodig  (36052, 5)
Shape of CV matrix after one hot encodig  (24155, 5)
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

Minimum frequency of words 10 (min_df = 10) of ALL features

In [53]:
```
# We are considering only the words which appeared in at least 10 documents(rows
vectorizer = CountVectorizer(min_df=10)
vectorizer.fit(train_preprocessed_essays)

train_text_bow = vectorizer.transform(train_preprocessed_essays)
print("Shape of matrix after one hot encodig ",train_text_bow.shape)
```

```
Shape of matrix after one hot encodig  (49041, 12132)
```

In [54]:
```
#Vectorizing Test Data
# you can vectorize the title also
# before you vectorize the title make sure you preprocess it
test_text_bow = vectorizer.transform(test_preprocessed_essays)
print("Shape of matrix after one hot encodig ",test_text_bow.shape)
```

```
Shape of matrix after one hot encodig  (36052, 12132)
```

In [55]:
```
# Vectrozing CV Data
cv_text_bow = vectorizer.transform(cv_preprocessed_essays)
print("Shape of matrix after one hot encodig ",cv_text_bow.shape)
```

```
Shape of matrix after one hot encodig  (24155, 12132)
```

Project_Title BOW

In [56]:
```
# We are considering only the words which appeared in at least 10 documents(rows
vectorizer = CountVectorizer(min_df=10)
vectorizer.fit(train_preprocessed_titles)

train_titles_bow = vectorizer.transform(train_preprocessed_titles)
print("Shape of matrix after one hot encodig ",train_titles_bow.shape)
```

```
Shape of matrix after one hot encodig  (49041, 2080)
```

In [57]:
```
#Vectorizing Test Data
test_titles_bow = vectorizer.transform(test_preprocessed_titles)
print("Shape of matrix after one hot encodig ",test_titles_bow.shape)
```

```
Shape of matrix after one hot encodig  (36052, 2080)
```

In [58]:
```
#Vectrizing CV Data
cv_titles_bow = vectorizer.transform(cv_preprocessed_titles)
print("Shape of matrix after one hot encodig ",cv_titles_bow.shape)
```

```
Shape of matrix after one hot encodig  (24155, 2080)
```

1.5.2.2 TFIDF vectorizer

Here also we are considering word frequency of 10 words(min_df = 10) of all features

In [59]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(train_preprocessed_essays)

train_text_tfidf = vectorizer.transform(train_preprocessed_essays)
print("Shape of matrix after one hot encodig ",train_text_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (49041, 12132)
```

In [60]:
```
test_text_tfidf = vectorizer.transform(test_preprocessed_essays)
print("Shape of matrix after one hot encodig ",test_text_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (36052, 12132)
```

In [61]:
```
cv_text_tfidf = vectorizer.transform(cv_preprocessed_essays)
print("Shape of matrix after one hot encodig ",cv_text_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (24155, 12132)
```

Project Titles

In [62]:
```
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(train_preprocessed_titles)

train_title_tfidf = vectorizer.transform(train_preprocessed_titles)
print("Shape of matrix after one hot encodig ",train_title_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (49041, 2080)
```

In [63]:
```
test_title_tfidf = vectorizer.transform(test_preprocessed_titles)
print("Shape of matrix after one hot encodig ",test_title_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (36052, 2080)
```

In [64]:
```
cv_title_tfidf = vectorizer.transform(cv_preprocessed_titles)
print("Shape of matrix after one hot encodig ",cv_title_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (24155, 2080)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [65]:
```
'''
# 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-

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


'''
```

Out[65]: '\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084

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

In [66]:
```
# stronging variables into pickle files python: http://www.jessicayung.com/how-t
# make sure you have the glove_vectors file
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [67]:
```
# average Word2Vec
# compute average word2vec for each review.
train_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in
for sentence in tqdm(train_preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_avg_w2v_vectors.append(vector)

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

```
100%|
| 49041/49041 [00:22<00:00, 2160.21it/s]

49041
300
```

In [68]:
```
# average Word2Vec
# compute average word2vec for each review.
test_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in t
for sentence in tqdm(test_preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_avg_w2v_vectors.append(vector)

print(len(test_avg_w2v_vectors))
print(len(test_avg_w2v_vectors[0]))
```

```
100%|
| 36052/36052 [00:19<00:00, 1861.39it/s]

36052
300
```

In [69]:
```
# average Word2Vec
# compute average word2vec for each review.
cv_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this
for sentence in tqdm(cv_preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    cv_avg_w2v_vectors.append(vector)

print(len(cv_avg_w2v_vectors))
print(len(cv_avg_w2v_vectors[0]))
```

```
100%|
| 24155/24155 [00:13<00:00, 1828.96it/s]

24155
300
```

AVG_W2V Project_Titles

In [70]:
```
# average Word2Vec
# compute average word2vec for each review.
train_title_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stor
for sentence in tqdm(train_preprocessed_titles): # 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
    train_title_avg_w2v_vectors.append(vector)

print(len(train_title_avg_w2v_vectors))
print(len(train_title_avg_w2v_vectors[0]))
```

```
100%|██████████| 
49041/49041 [00:01<00:00, 43739.54it/s]

49041
300
```

In [71]:
```
# average Word2Vec
# compute average word2vec for each review.
test_title_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is store
for sentence in tqdm(test_preprocessed_titles): # 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
    test_title_avg_w2v_vectors.append(vector)

print(len(test_title_avg_w2v_vectors))
print(len(test_title_avg_w2v_vectors[0]))
```

```
100%|██████████| 
36052/36052 [00:00<00:00, 39193.71it/s]

36052
300
```

In [72]:
```
# average Word2Vec
# compute average word2vec for each review.
cv_title_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored 
for sentence in tqdm(cv_preprocessed_titles): # 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
    cv_title_avg_w2v_vectors.append(vector)

print(len(cv_title_avg_w2v_vectors))
print(len(cv_title_avg_w2v_vectors[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████████|
24155/24155 [00:00<00:00, 39938.63it/s]

24155
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [73]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(train_preprocessed_essays)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [74]:
```
# average Word2Vec
# compute average word2vec for each review.
train_essay_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is st
for sentence in tqdm(train_preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf valu
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    train_essay_tfidf_w2v_vectors.append(vector)

print(len(train_essay_tfidf_w2v_vectors))
print(len(train_essay_tfidf_w2v_vectors[0]))
```

```
100%|
| 49041/49041 [02:38<00:00, 308.46it/s]

49041
300
```

In [75]:
```
# Similarly you can vectorize for title also
# average Word2Vec
# compute average word2vec for each review.
test_essay_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is sto
for sentence in tqdm(test_preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf valu
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    test_essay_tfidf_w2v_vectors.append(vector)

print(len(test_essay_tfidf_w2v_vectors))
print(len(test_essay_tfidf_w2v_vectors[0]))
```

```
100%|
| 36052/36052 [01:56<00:00, 310.64it/s]

36052
300
```

In [76]:
```
# average Word2Vec
# compute average word2vec for each review.
cv_essay_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored
for sentence in tqdm(cv_preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf valu
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    cv_essay_tfidf_w2v_vectors.append(vector)

print(len(cv_essay_tfidf_w2v_vectors))
print(len(cv_essay_tfidf_w2v_vectors[0]))
```

```
100%|
| 24155/24155 [01:23<00:00, 289.42it/s]

24155
300
```

Proect_titles

In [77]:
```
# average Word2Vec
# compute average word2vec for each review.
train_title_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is st
for sentence in tqdm(train_preprocessed_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf valu
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    train_title_tfidf_w2v_vectors.append(vector)

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

```
100%|
49041/49041 [00:02<00:00, 21141.79it/s]

49041
300
```

In [78]:
```
# average Word2Vec
# compute average word2vec for each review.
test_title_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is sto
for sentence in tqdm(test_preprocessed_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf val
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    test_title_tfidf_w2v_vectors.append(vector)

print(len(test_title_tfidf_w2v_vectors))
print(len(test_title_tfidf_w2v_vectors[0]))
```

```
100%|██████████| 
36052/36052 [00:01<00:00, 21199.42it/s]

36052
300
```

In [79]:
```
# average Word2Vec
# compute average word2vec for each review.
cv_title_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is store
for sentence in tqdm(cv_preprocessed_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf val
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    cv_title_tfidf_w2v_vectors.append(vector)

print(len(cv_title_tfidf_w2v_vectors))
print(len(cv_title_tfidf_w2v_vectors[0]))
```

```
100%|██████████| 
24155/24155 [00:01<00:00, 21981.52it/s]

24155
300
```

1.5.3 Vectorizing Numerical features

In [80]:
```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).
project_data = pd.merge(project_data, price_data, on='id', how='left')
print(price_data.head())
#print(project_data.columns)
print(x_train.columns)
```

```
        id    price  quantity
0  p000001   459.56         7
1  p000002   515.89        21
2  p000003   298.97         4
3  p000004  1113.69        98
4  p000005   485.99         8
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category', 'project_title',
       'project_essay_1', 'project_essay_2', 'project_essay_3',
       'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'clean_categories',
       'clean_subcategories', 'essay', 'title_word_count', 'essay_word_count'],
      dtype='object')
```

In [81]:
```
#   - quantity : numerical (optinal)
#   - teacher_number_of_previously_posted_projects : numerical
#   - price : numerical
x_train = pd.merge(x_train, price_data, on = "id", how = "left")
#print(x_train.columns)
x_test = pd.merge(x_test, price_data, on = "id", how = "left")
x_cv = pd.merge(x_cv, price_data, on = "id", how = "left")
```

Standardize Price

In [82]:
```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/skle
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.
# Reshape your data either using array.reshape(-1, 1)

price_scalar = StandardScaler()

price_scalar.fit(x_train['price'].values.reshape(-1,1)) # finding the mean and s
print(f"TRAIN -> Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(p
# Now standardize the data with above maen and variance.
train_price_standar = price_scalar.transform(x_train['price'].values.reshape(-1,

price_scalar.fit(x_test['price'].values.reshape(-1,1)) # finding the mean and st
print(f"TEST -> Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(pr
# Now standardize the data with above maen and variance.
test_price_standar = price_scalar.transform(x_test['price'].values.reshape(-1, 1

price_scalar.fit(x_cv['price'].values.reshape(-1,1)) # finding the mean and stan
print(f"CV -> Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(pric
# Now standardize the data with above maen and variance.
cv_price_standar = price_scalar.transform(x_cv['price'].values.reshape(-1, 1))
```

```
TRAIN -> Mean : 297.1335653840664, Standard deviation : 363.28872800136634
TEST -> Mean : 300.7768501054033, Standard deviation : 382.6699753150988
CV -> Mean : 296.1543266404471, Standard deviation : 352.5491742990371
```

In [83]:
```
print(train_price_standar.shape, y_train.shape)
print(test_price_standar.shape, y_test.shape)
print(cv_price_standar.shape, y_cv.shape)
```

```
(49041, 1) (49041,)
(36052, 1) (36052,)
(24155, 1) (24155,)
```

Standardize Teacher previously posted Projects

In [84]:
```
warnings.filterwarnings("ignore")
price_scalar.fit(x_train['teacher_number_of_previously_posted_projects'].values.
print(f"TRAIN -> Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(p
# Now standardize the data with above maen and variance.
train_prev_proj_standar = price_scalar.transform(x_train['teacher_number_of_prev

price_scalar.fit(x_test['teacher_number_of_previously_posted_projects'].values.r
print(f"TEST -> Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(pr
# Now standardize the data with above maen and variance.
test_prev_proj_standar = price_scalar.transform(x_test['teacher_number_of_previo

price_scalar.fit(x_cv['teacher_number_of_previously_posted_projects'].values.res
print(f"CV -> Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(pric
# Now standardize the data with above maen and variance.
cv_prev_proj_standar = price_scalar.transform(x_cv['teacher_number_of_previously_
```

```
TRAIN -> Mean : 11.093656328378296, Standard deviation : 27.627970646902444
TEST -> Mean : 11.299123488294686, Standard deviation : 28.185609983236137
CV -> Mean : 11.056137445663424, Standard deviation : 27.46187050275317
```

In [85]:
```
print(train_prev_proj_standar.shape, y_train.shape)
print(test_prev_proj_standar.shape, y_test.shape)
print(cv_prev_proj_standar.shape, y_cv.shape)
```

```
(49041, 1) (49041,)
(36052, 1) (36052,)
(24155, 1) (24155,)
```

Standardize Quantity

In [86]:
```
price_scalar.fit(x_train['quantity'].values.reshape(-1,1)) # finding the mean an
print(f"TRAIN -> Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(p
# Now standardize the data with above maen and variance.
train_quantity_standar = price_scalar.transform(x_train['quantity'].values.resha

price_scalar.fit(x_test['quantity'].values.reshape(-1,1)) # finding the mean and
print(f"TEST -> Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(pr
# Now standardize the data with above maen and variance.
test_quantity_standar = price_scalar.transform(x_test['quantity'].values.reshape

price_scalar.fit(x_cv['quantity'].values.reshape(-1,1)) # finding the mean and s
print(f"CV -> Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(pric
# Now standardize the data with above maen and variance.
cv_quantity_standar = price_scalar.transform(x_cv['quantity'].values.reshape(-1,
```

```
TRAIN -> Mean : 17.10701249974511, Standard deviation : 27.48278828673773
TEST -> Mean : 16.771219349828026, Standard deviation : 24.416340622026798
CV -> Mean : 16.968660732767542, Standard deviation : 26.023272927346344
```

In [87]:
```
print(train_quantity_standar.shape, y_train.shape)
print(test_quantity_standar.shape, y_test.shape)
print(cv_quantity_standar.shape, y_cv.shape)
```

```
(49041, 1) (49041,)
(36052, 1) (36052,)
(24155, 1) (24155,)
```

Standardize Title_word_count

In [88]:
```
title_scalar = StandardScaler()
title_scalar.fit(x_train['title_word_count'].values.reshape(-1,1)) # finding the
print(f"Mean : {title_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scal
train_title_word_count_standar = title_scalar.transform(x_train['title_word_coun

title_scalar.fit(x_test['title_word_count'].values.reshape(-1,1)) # finding the
print(f"Mean : {title_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scal
test_title_word_count_standar = title_scalar.transform(x_test['title_word_count'

title_scalar.fit(x_cv['title_word_count'].values.reshape(-1,1)) # finding the me
print(f"Mean : {title_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scal
cv_title_word_count_standar = title_scalar.transform(x_cv['quantity'].values.res

print(train_title_word_count_standar.shape, y_train.shape)
print(test_title_word_count_standar.shape, y_test.shape)
print(cv_title_word_count_standar.shape, y_cv.shape)
```

```
Mean : 5.203258497991476, Standard deviation : 26.023272927346344
Mean : 5.199628314656607, Standard deviation : 26.023272927346344
Mean : 5.200455392258331, Standard deviation : 26.023272927346344
(49041, 1) (49041,)
(36052, 1) (36052,)
(24155, 1) (24155,)
```

Standardize Essay_word_count

In [89]:
```
essay_scalar = StandardScaler()

essay_scalar.fit(x_train['essay_word_count'].values.reshape(-1,1)) # finding the
train_essay_word_count_standar = essay_scalar.transform(x_train['essay_word_coun

essay_scalar.fit(x_train['essay_word_count'].values.reshape(-1,1)) # finding the
test_essay_word_count_standar = essay_scalar.transform(x_test['essay_word_count'

essay_scalar.fit(x_cv['essay_word_count'].values.reshape(-1,1)) # finding the me
cv_essay_word_count_standar = essay_scalar.transform(x_cv['essay_word_count'].va

print(train_essay_word_count_standar.shape, y_train.shape)
print(test_essay_word_count_standar.shape, y_test.shape)
print(cv_essay_word_count_standar.shape, y_cv.shape)
```

```
(49041, 1) (49041,)
(36052, 1) (36052,)
(24155, 1) (24155,)
```

Standardize Positive Intensity

In [101]:
```
essay_scalar.fit(x_train['positive'].values.reshape(-1,1)) # finding the mean an
train_positive_standar = essay_scalar.transform(x_train['positive'].values.resha

essay_scalar.fit(x_train['positive'].values.reshape(-1,1)) # finding the mean an
test_positive_standar = essay_scalar.transform(x_test['positive'].values.reshape

essay_scalar.fit(x_cv['positive'].values.reshape(-1,1)) # finding the mean and s
cv_positive_standar = essay_scalar.transform(x_cv['positive'].values.reshape(-1,

print(train_positive_standar.shape, y_train.shape)
print(test_positive_standar.shape, y_test.shape)
print(cv_positive_standar.shape, y_cv.shape)
```

```
(49041, 1) (49041,)
(36052, 1) (36052,)
(24155, 1) (24155,)
```

Standarsize Negitive Intensity

In [102]:
```
essay_scalar.fit(x_train['negitive'].values.reshape(-1,1)) # finding the mean an
train_negitive_standar = essay_scalar.transform(x_train['negitive'].values.resha

essay_scalar.fit(x_train['negitive'].values.reshape(-1,1)) # finding the mean an
test_negitive_standar = essay_scalar.transform(x_test['negitive'].values.reshape

essay_scalar.fit(x_cv['negitive'].values.reshape(-1,1)) # finding the mean and s
cv_negitive_standar = essay_scalar.transform(x_cv['negitive'].values.reshape(-1,

print(train_negitive_standar.shape, y_train.shape)
print(test_negitive_standar.shape, y_test.shape)
print(cv_negitive_standar.shape, y_cv.shape)
```

```
(49041, 1) (49041,)
(36052, 1) (36052,)
(24155, 1) (24155,)
```

Standardize Neutral Intensity

In [103]:
```
essay_scalar.fit(x_train['neutral'].values.reshape(-1,1)) # finding the mean and
train_neutral_standar = essay_scalar.transform(x_train['neutral'].values.reshape

essay_scalar.fit(x_train['neutral'].values.reshape(-1,1))
test_neutral_standar = essay_scalar.transform(x_test['neutral'].values.reshape(-

essay_scalar.fit(x_cv['neutral'].values.reshape(-1,1)) # finding the mean and st
cv_neutral_standar = essay_scalar.transform(x_cv['neutral'].values.reshape(-1, 1

print(train_neutral_standar.shape, y_train.shape)
print(test_neutral_standar.shape, y_test.shape)
print(cv_neutral_standar.shape, y_cv.shape)
```

```
(49041, 1) (49041,)
(36052, 1) (36052,)
(24155, 1) (24155,)
```

In [ ]:

Assignment 5: Support Vector Machines

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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) 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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.
      (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

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 (http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html) on TfidfVectorizer (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorize of essay text, choose the number of components (`n_components`) using elbow method (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/pca-code-example-using-non-visualization/)** : 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 (http://zetcode.com/python/prettytable/)

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. (https://soundcloud.com/applied-ai-course/leakage-bow-and-tfidf)

Support Vector Machine

2.1 SVM On Set-1

Merging all Categorical and Numerical _ SET-1 BOW Encoding

In [91]:
```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense
X_train1 = hstack((train_categories_one_hot,train_sub_categories_one_hot,train_s
                   train_teacher_one_hot,train_text_bow, train_titles_bow, train_
                   train_prev_proj_standar, train_price_standar, train_title_word
                   train_essay_word_count_standar, train_positive_standar, train_
                   train_neutral_standar)).tocsr()
print(X_train1.shape, y_train.shape)
print(type(X_train1))
```

```
(49041, 14320) (49041,)
<class 'scipy.sparse.csr.csr_matrix'>
```

In [92]:
```
X_test1 = hstack((test_categories_one_hot,test_sub_categories_one_hot,test_state_
                  test_teacher_one_hot,test_text_bow, test_titles_bow, test_quant
                  test_prev_proj_standar, test_price_standar, test_essay_word_cou
                  test_title_word_count_standar, test_positive_standar, test_neg
                  test_neutral_standar)).tocsr()
print(X_test1.shape, y_test.shape)
print(type(X_test1))
```

```
(36052, 14320) (36052,)
<class 'scipy.sparse.csr.csr_matrix'>
```

In [93]:
```
X_cv1 = hstack((cv_categories_one_hot, cv_sub_categories_one_hot, cv_state_one_ho
                  cv_teacher_one_hot, cv_text_bow, cv_titles_bow, cv_quantity_sta
                  cv_prev_proj_standar, cv_price_standar, cv_essay_word_count_sta
                  cv_title_word_count_standar, cv_positive_standar, cv_negitive_s
                  cv_neutral_standar)).tocsr()
print(X_cv1.shape, y_cv.shape)
print(type(X_cv1))
```

```
(24155, 14320) (24155,)
<class 'scipy.sparse.csr.csr_matrix'>
```

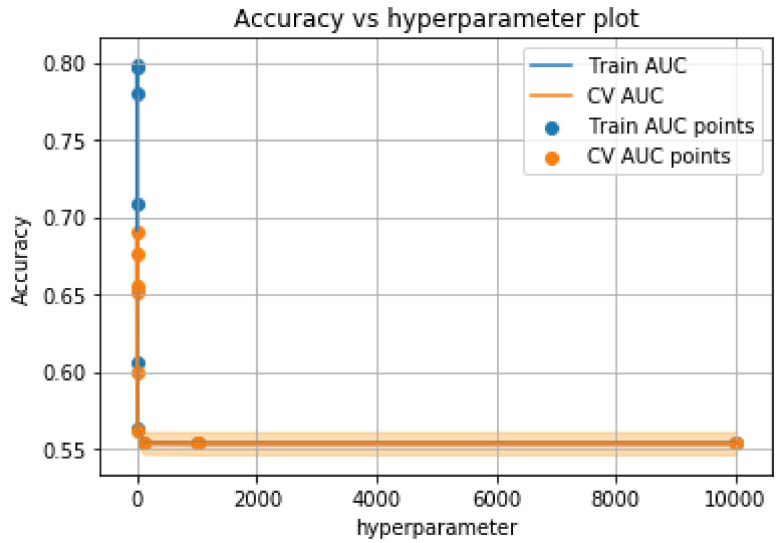
In [94]:
```
print(X_train1.shape, y_train.shape)
```

```
(49041, 14320) (49041,)
```

Hyperparameter Tunning

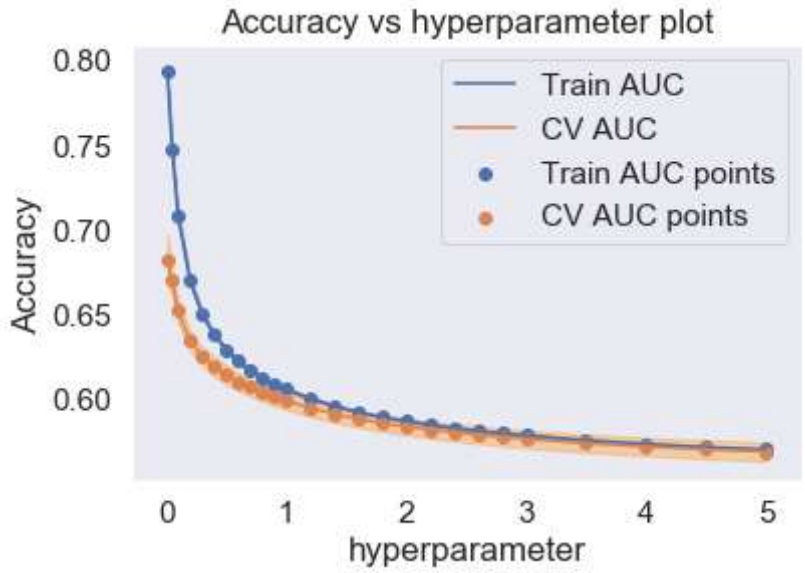
In [98]:
```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
#from sklearn.datasets import *
from sklearn import linear_model
from sklearn.linear_model import SGDClassifier
from sklearn import svm
```

In [103]:
```
parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**

SV = SGDClassifier(loss = 'hinge', penalty = 'l2')
classifier = GridSearchCV(SV, parameters, cv= 10, scoring='roc_auc')

classifier.fit(X_train1, y_train)

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

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

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

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


plt.legend()
plt.xlabel("hyperparameter")
plt.ylabel("Accuracy")
plt.title("Accuracy vs hyperparameter plot")
plt.grid()
plt.show()
```



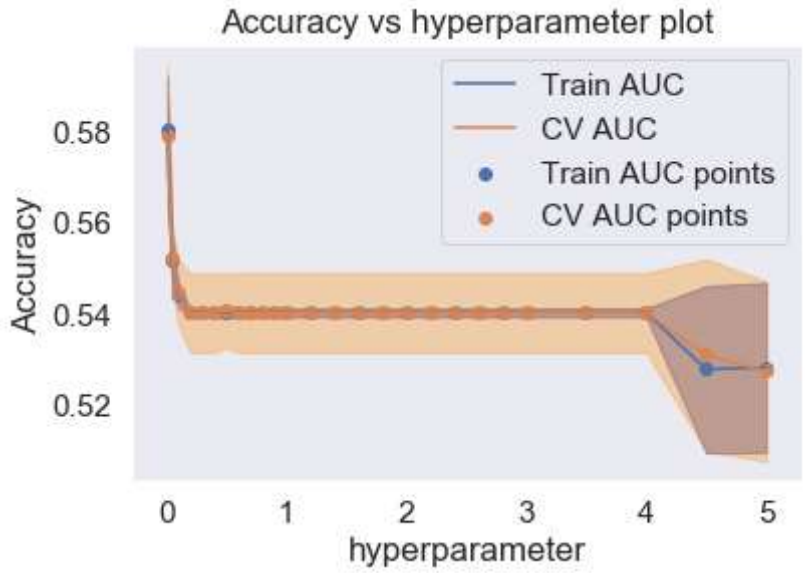
Here we can't able to find the best Hyperparamter Now i am reducing the range of 'ALPHA' for rest of my models

In [139]:
```
parameters = {'alpha':[0.01,0.05,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,1.2,1.4,1

SV = SGDClassifier(loss = 'hinge', penalty = 'l2')
classifier = GridSearchCV(SV, parameters, cv= 10, scoring='roc_auc')

classifier.fit(X_train1, y_train)

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

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
#https://stackoverflow.com/questions/48796282/how-to-visualize-dependence-of-mode
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_s

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


plt.legend()
plt.xlabel("hyperparameter")
plt.ylabel("Accuracy")
plt.title("Accuracy vs hyperparameter plot")
plt.grid()
plt.show()
```



To find the Best Penalty we are using "l1" Regulrizer on same range of "ALPHA" values

In [140]:
```
parameters = {'alpha':[0.01,0.05,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,1.2,1.4,1

SV = SGDClassifier(loss = 'hinge', penalty = 'l1')
classifier = GridSearchCV(SV, parameters, cv= 10, scoring='roc_auc')

classifier.fit(X_train1, y_train)

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

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
#https://stackoverflow.com/questions/48796282/how-to-visualize-dependence-of-mode
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_s

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


plt.legend()
plt.xlabel("hyperparameter")
plt.ylabel("Accuracy")
plt.title("Accuracy vs hyperparameter plot")
plt.grid()
plt.show()
```



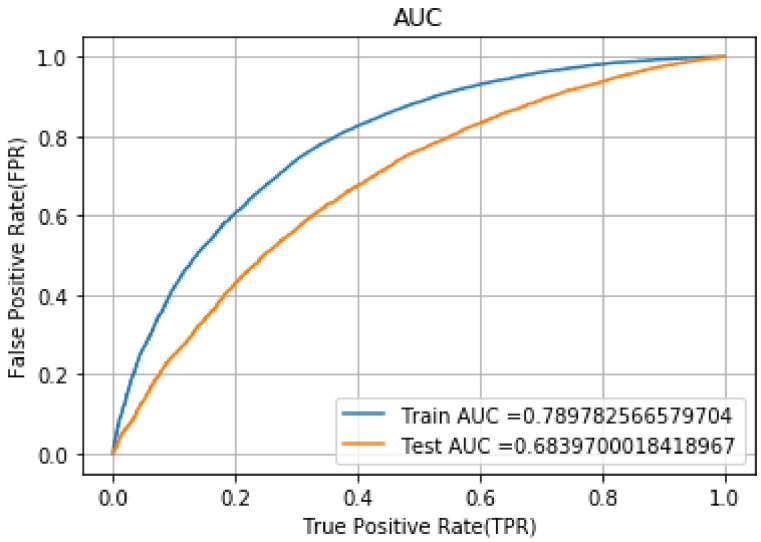
L1 Regularizer performs well on BOW encoded data than L2 regularizer

Trainng Model Using Best HyperParameter

In [132]:
```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability est
    # not the predicted outputs

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

    return y_data_pred
```

In [116]:
```
# The Hyperpaameter cannot be -VE since we taking the least values as Hyperparame
best_alpha_1 = 0.01
```

In [128]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.htr
from sklearn.metrics import roc_curve, auc
#default alpha = 0.0001
Classifier_bow = SGDClassifier(loss = 'hinge', penalty = 'l2', alpha = 0.001)

Classifier_bow.fit(X_train1, y_train)
#https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClass

y_train_pred = Classifier_bow.decision_function(X_train1)
y_test_pred = Classifier_bow.decision_function(X_test1)

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

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



Confusion Matrix

In [133]:
```
def predict(proba, threshould, fpr, tpr):

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

Train Data

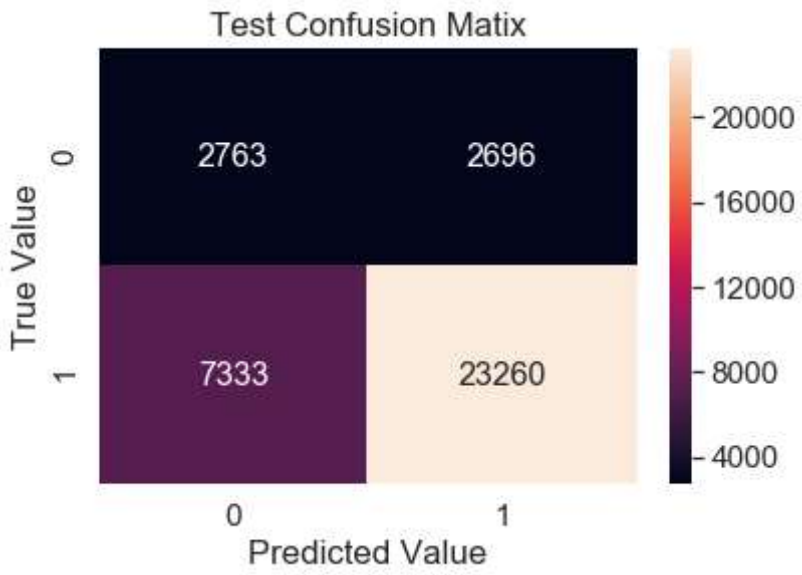
In [130]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sea
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr,
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2499999818661462 for threshold 1.103
[[ 3714  3712]
 [ 4761 36854]]
```

In [131]:
```
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pre
                                                                      test_fpr,t
sea.set(font_scale=1.4)
sea.heatmap(train_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = '
plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Test Confusion Matix")
```

```
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 1.538
```

Out[131]:
```
Text(0.5, 1.0, 'Test Confusion Matix')
```



Test Data

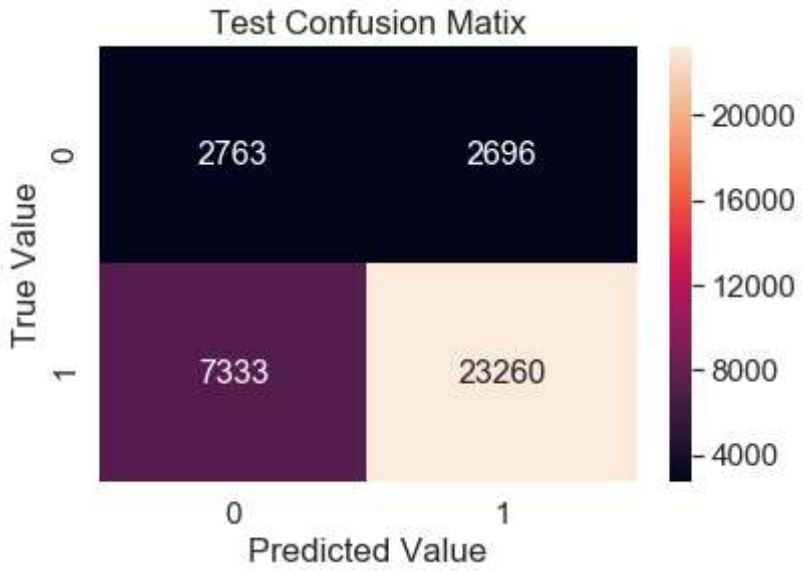
In [132]:
```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, tes
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 1.538
[[ 2763  2696]
 [ 7333 23260]]
```

In [133]:
```
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pre
                                                                       test_fpr,t
sea.set(font_scale=1.4)
sea.heatmap(train_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = '
plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Test Confusion Matix")
```

```
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 1.538
```

Out[133]: Text(0.5, 1.0, 'Test Confusion Matix')



2.2 Merging all Categorical and Numerical _ SET-2 TF-IDF Encoding

In [134]:
```
X_train2 = hstack((train_categories_one_hot, train_sub_categories_one_hot, train_
                   train_teacher_one_hot,train_text_tfidf, train_title_tfidf, tra
                   train_prev_proj_standar, train_price_standar, train_title_word_
                   train_essay_word_count_standar, train_positive_standar, train_
                   train_neutral_standar)).tocsr()
print(X_train2.shape, y_train.shape)
print(type(X_train2))
```

```
(49041, 14320) (49041,)
<class 'scipy.sparse.csr.csr_matrix'>
```

In [135]:
```
X_test2 = hstack((test_categories_one_hot,test_sub_categories_one_hot,test_state_
                  test_teacher_one_hot,test_text_tfidf, test_title_tfidf, test_q
                  test_prev_proj_standar, test_price_standar,test_title_word_cou
                  test_essay_word_count_standar, test_positive_standar, test_neg
                  test_neutral_standar)).tocsr()
print(X_test2.shape, y_test.shape)
print(type(X_test2))
```

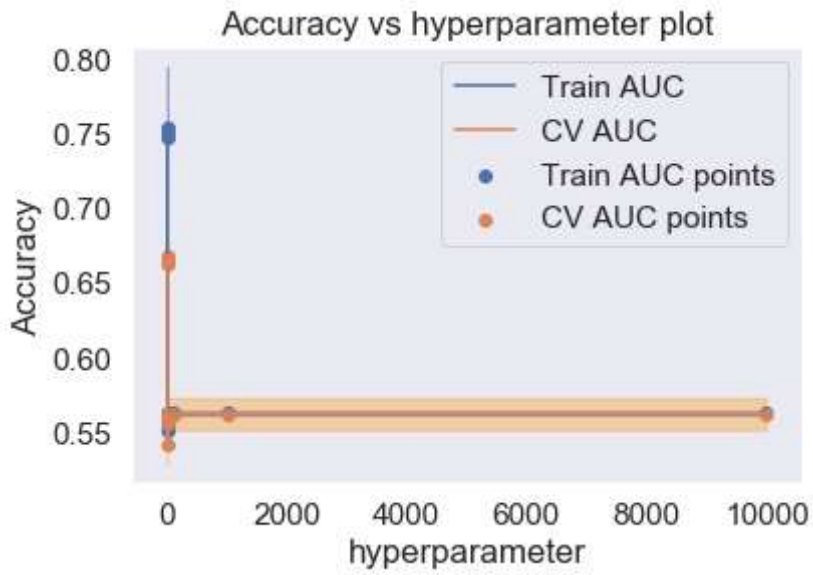
```
(36052, 14320) (36052,)
<class 'scipy.sparse.csr.csr_matrix'>
```

In [136]:
```
X_cv2 = hstack((cv_categories_one_hot,cv_sub_categories_one_hot,cv_state_one_hot
                  cv_teacher_one_hot,cv_text_tfidf, cv_title_tfidf, cv_quantity_
                  cv_prev_proj_standar, cv_price_standar, cv_title_word_count_st
                  cv_essay_word_count_standar, cv_positive_standar, cv_negitive_
                  cv_neutral_standar)).tocsr()
print(X_cv2.shape, y_cv.shape)
print(type(X_cv2))
```

```
(24155, 14320) (24155,)
<class 'scipy.sparse.csr.csr_matrix'>
```

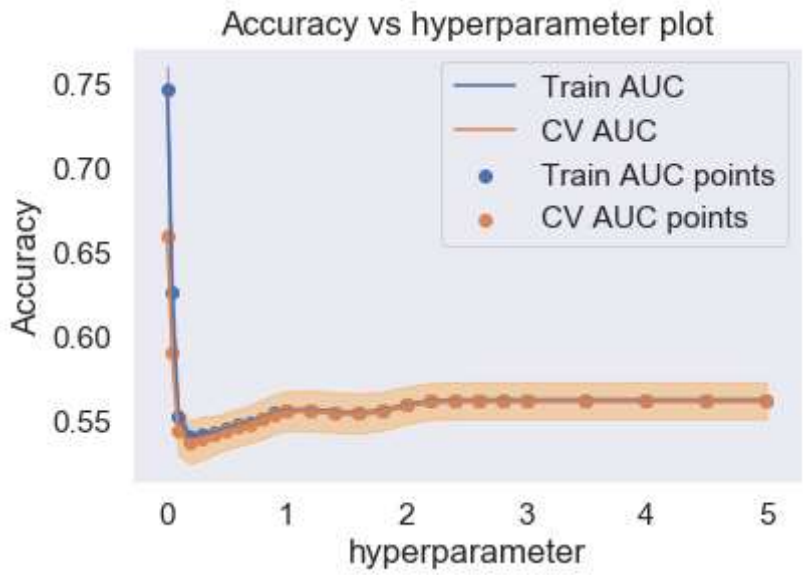
Hyperparameter Tunning

In [212]:
```
parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**

SV = SGDClassifier(loss = 'hinge', penalty = 'l2')
classifier = GridSearchCV(SV, parameters, cv= 10, scoring='roc_auc')

classifier.fit(X_train2, y_train)

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

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

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

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


plt.legend()
plt.xlabel("hyperparameter")
plt.ylabel("Accuracy")
plt.title("Accuracy vs hyperparameter plot")
plt.grid()
plt.show()
```



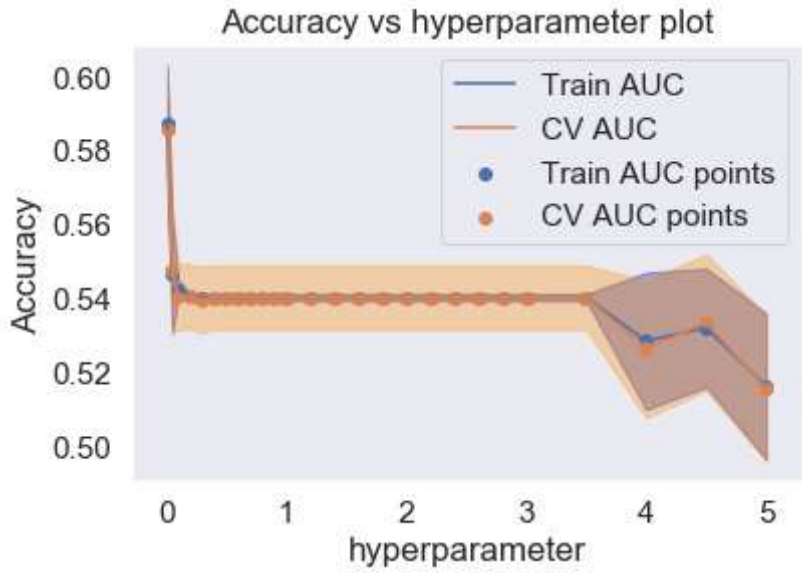
Again for TF-IDF encoding we cant able to find best "ALPHA" from above

In [141]:
```
#Reducing the range of alpha for L2 regularizer
parameters = {'alpha':[0.00001, 0.0001, 0.001,0.01, 0.05, 0.1]}

SV = SGDClassifier(loss = 'hinge', penalty = 'l2')
classifier = GridSearchCV(SV, parameters, cv= 10, scoring='roc_auc')

classifier.fit(X_train2, y_train)

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

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
#https://stackoverflow.com/questions/48796282/how-to-visualize-dependence-of-mode
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_s

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


plt.legend()
plt.xlabel("hyperparameter")
plt.ylabel("Accuracy")
plt.title("Accuracy vs hyperparameter plot")
plt.grid()
plt.show()
```



For best Penalty usin L1 Regularization on same range of alpha Values

In [142]:
```
parameters = {'alpha':[0.01,0.05,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,1.2,1.4,1

SV = SGDClassifier(loss = 'hinge', penalty = 'l1')
classifier = GridSearchCV(SV, parameters, cv= 10, scoring='roc_auc')

classifier.fit(X_train1, y_train)

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

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
#https://stackoverflow.com/questions/48796282/how-to-visualize-dependence-of-mode
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_s

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


plt.legend()
plt.xlabel("hyperparameter")
plt.ylabel("Accuracy")
plt.title("Accuracy vs hyperparameter plot")
plt.grid()
plt.show()
```

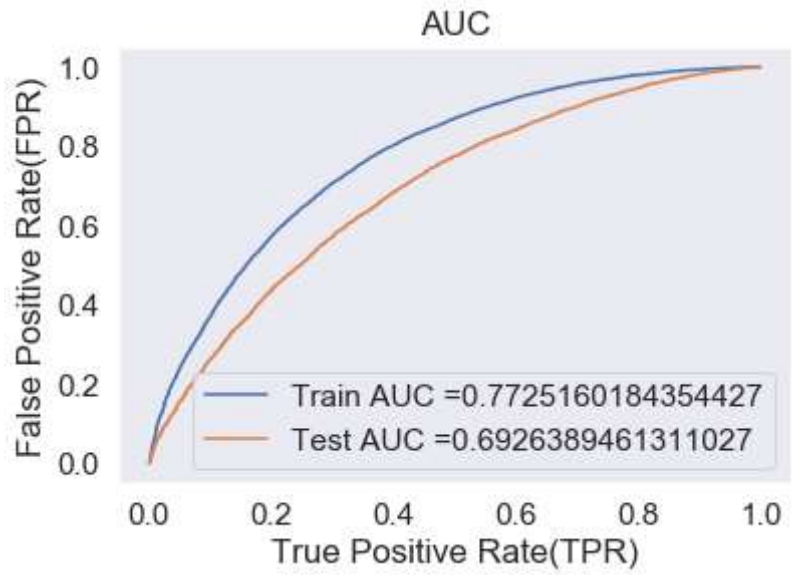


L1 Regularizer seems to perform well in TF-IDF encoded data. Since AUC curve for L1 regularizer is seems better than L2 reg

Training Model on best Hyperparameter

In [215]:
```
best_alpha_2 = 0.001
```

In [216]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.htr
from sklearn.metrics import roc_curve, auc

Classifier_tfidf = SGDClassifier(loss = 'hinge', penalty = 'l2', alpha = 0.001)

Classifier_tfidf.fit(X_train2, y_train)

y_train_pred = Classifier_tfidf.decision_function(X_train2)
y_test_pred = Classifier_tfidf.decision_function(X_test2)

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

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



Train Confusing Matrix

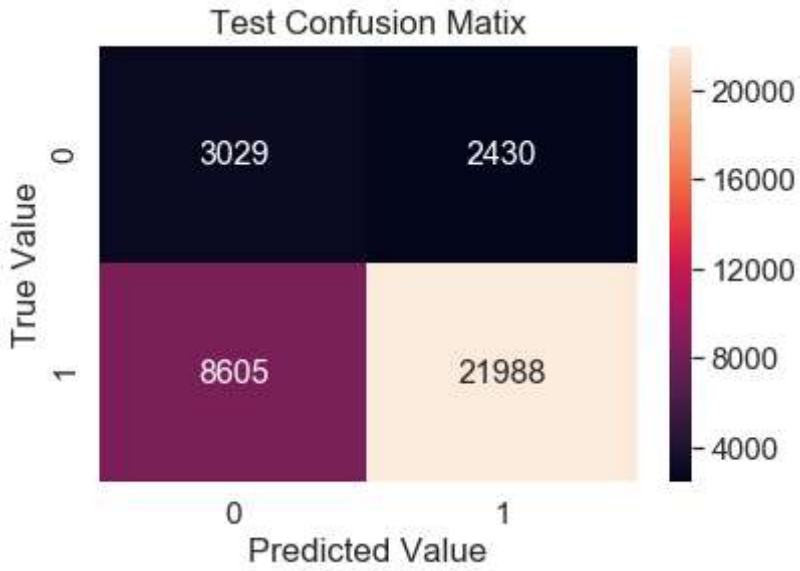
In [147]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sea
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr,
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.985
[[ 3713  3713]
 [ 6385 35230]]
```

In [148]:
```
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pre
                                                                       test_fpr,t
sea.set(font_scale=1.4)
sea.heatmap(train_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = '
plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Test Confusion Matix")
```

```
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 1.014
```

Out[148]:
```
Text(0.5, 1.0, 'Test Confusion Matix')
```



Test Confusion Matrix

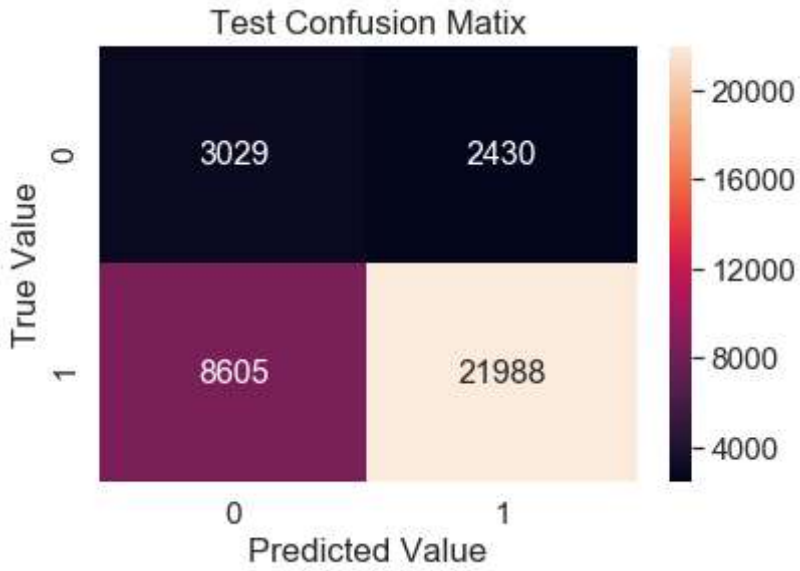
In [149]:
```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, tes
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 1.014
[[ 3029  2430]
 [ 8605 21988]]
```

In [150]:
```
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pre
                                                                       test_fpr,t
sea.set(font_scale=1.4)
sea.heatmap(train_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = '
plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Test Confusion Matix")
```

```
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 1.014
```

Out[150]:
```
Text(0.5, 1.0, 'Test Confusion Matix')
```



In [ ]:
```
```

2.3 Merging all Categorical and Numerical _ SET-3 AVG-W2V Encoding

In [217]:
```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense
X_train3 = hstack((train_categories_one_hot,train_sub_categories_one_hot,train_s
                   train_teacher_one_hot, train_title_avg_w2v_vectors, train_avg_
                   train_prev_proj_standar, train_price_standar, train_positive_s
                   train_neutral_standar, train_title_word_count_standar, train_e
print(X_train3.shape, y_train.shape)
print(type(X_train3))
```

```
(49041, 708) (49041,)
<class 'scipy.sparse.csr.csr_matrix'>
```

In [218]:
```
X_test3 = hstack((test_categories_one_hot,test_sub_categories_one_hot,test_state_
                  test_teacher_one_hot, test_title_avg_w2v_vectors, test_avg_w2v_
                  test_prev_proj_standar, test_price_standar, test_positive_stand
                  test_neutral_standar, test_title_word_count_standar, test_essay
print(X_test3.shape, y_test.shape)
print(type(X_test3))
```

```
(36052, 708) (36052,)
<class 'scipy.sparse.csr.csr_matrix'>
```

In [219]:
```
X_cv3 = hstack((cv_categories_one_hot, cv_sub_categories_one_hot, cv_state_one_ho
                  cv_teacher_one_hot, cv_title_avg_w2v_vectors, cv_avg_w2v_vector
                  cv_prev_proj_standar, cv_price_standar, cv_positive_standar, cv
                  cv_neutral_standar, cv_title_word_count_standar, cv_essay_word_
print(X_cv3.shape, y_cv.shape)
print(type(X_cv3))
```

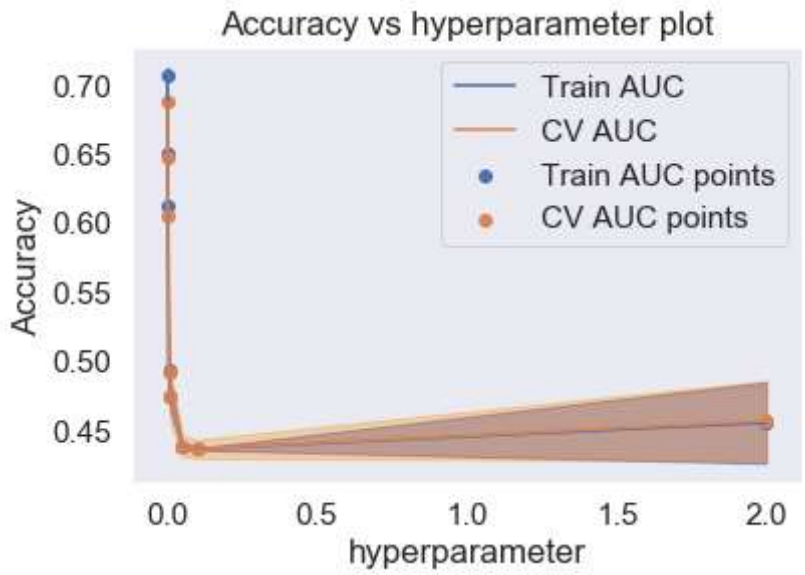
```
(24155, 708) (24155,)
<class 'scipy.sparse.csr.csr_matrix'>
```

In [170]:
```
print(X_train3.shape, y_train.shape)
```

```
(49041, 708) (49041,)
```

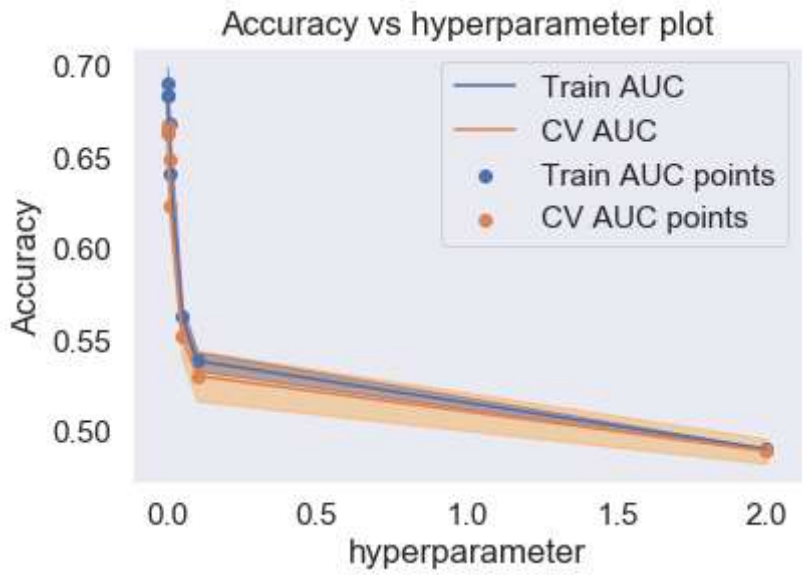
Hyperparameter Tunning

In [220]:
```
#we are using L1 Regularizer
parameters = {'alpha':[0.0001, 0.0005, 0.001, 0.005, 0.01,0.05,0.1, 2]}

SV = SGDClassifier(loss = 'hinge', penalty = 'l1')
classifier = GridSearchCV(SV, parameters, cv= 10, scoring='roc_auc')

classifier.fit(X_train3, y_train)

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

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
#https://stackoverflow.com/questions/48796282/how-to-visualize-dependence-of-mode
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_s

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


plt.legend()
plt.xlabel("hyperparameter")
plt.ylabel("Accuracy")
plt.title("Accuracy vs hyperparameter plot")
plt.grid()
plt.show()
```



we are using L2 Regularizer as we seen that L1 is Peforms badly

In [221]:
```
#we are using L2 Regularizer as we know that L1 is Peforms badly
parameters = {'alpha':[0.0001, 0.0005, 0.001, 0.005, 0.01,0.05,0.1, 2]}

SV = SGDClassifier(loss = 'hinge', penalty = 'l2')
classifier = GridSearchCV(SV, parameters, cv= 10, scoring='roc_auc')

classifier.fit(X_train3, y_train)

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

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
#https://stackoverflow.com/questions/48796282/how-to-visualize-dependence-of-mode
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_s

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


plt.legend()
plt.xlabel("hyperparameter")
plt.ylabel("Accuracy")
plt.title("Accuracy vs hyperparameter plot")
plt.grid()
plt.show()
```

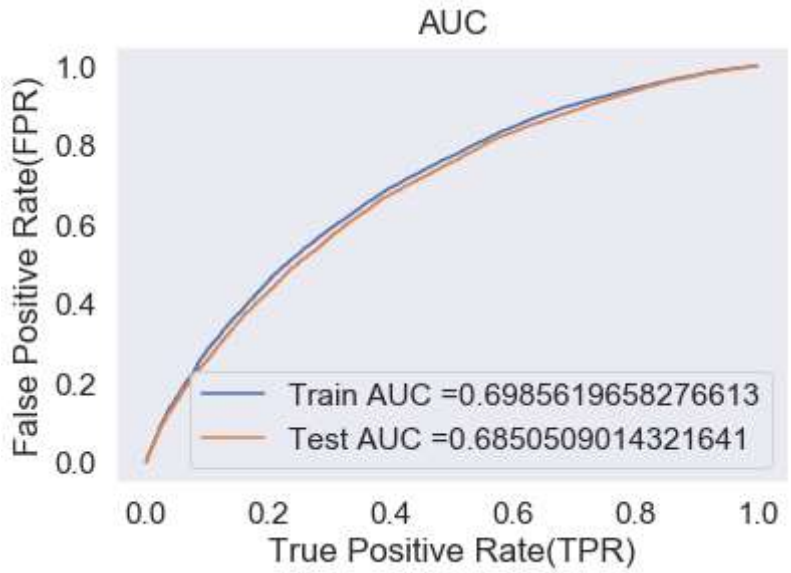


L2 Regularizer Seems better Auccaracy than L1 Regualrizer

Training Model Using best Hyperparameter

In [198]:
```
best_c_3 = 0.0005
```

In [176]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.htr
from sklearn.metrics import roc_curve, auc

Classifier_avgw2v = SGDClassifier(loss = 'hinge', penalty = 'l2', alpha = 0.0001

Classifier_avgw2v.fit(X_train3, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimate
# not the predicted outputs

y_train_pred = Classifier_avgw2v.decision_function(X_train3)
y_test_pred = Classifier_avgw2v.decision_function(X_test3)

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

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



Train Confusion Matrix

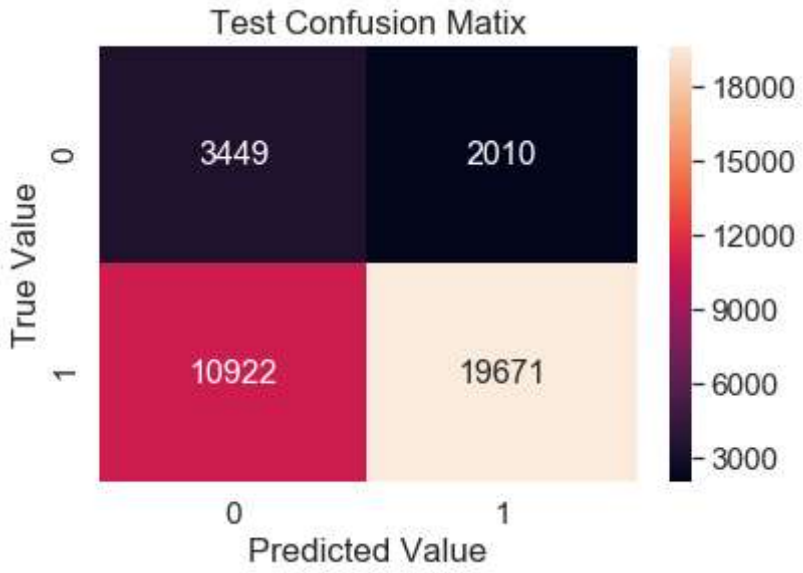
In [177]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sea
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr,
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 2.03
[[ 3713  3713]
 [ 9501 32114]]
```

In [178]:
```
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pre
                                                                      test_fpr,t
sea.set(font_scale=1.4)
sea.heatmap(train_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = '
plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Test Confusion Matix")
```

```
the maximum value of tpr*(1-fpr) 0.24999999161092995 for threshold 2.524
```

Out[178]:
```
Text(0.5, 1.0, 'Test Confusion Matix')
```



Test Confusion Matrix

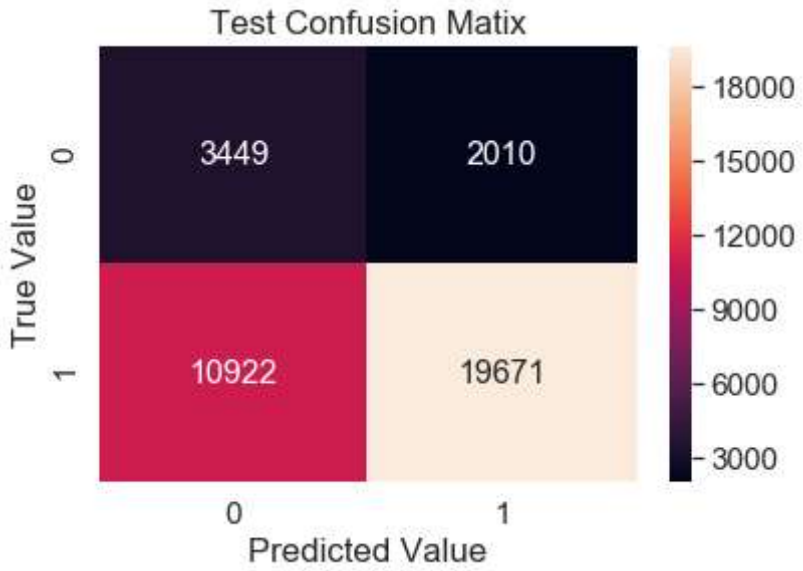
In [179]:
```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, tes
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092995 for threshold 2.524
[[ 3449  2010]
 [10922 19671]]
```

In [180]:
```
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pre
                                                                       test_fpr,t
sea.set(font_scale=1.4)
sea.heatmap(train_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = '
plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Test Confusion Matix")
```

```
the maximum value of tpr*(1-fpr) 0.24999999161092995 for threshold 2.524
```

Out[180]: Text(0.5, 1.0, 'Test Confusion Matix')



2.4 Merging all Categorical and Numerical _ SET-4 TFIDF-W2V Encoding

In [189]:
```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense
X_train3 = hstack((train_categories_one_hot,train_sub_categories_one_hot,train_s
                   train_teacher_one_hot, train_title_tfidf_w2v_vectors, train_e
                   train_quantity_standar, train_prev_proj_standar, train_price_
                   train_negitive_standar,  train_neutral_standar, train_title_w
                   train_essay_word_count_standar)).tocsr()
print(X_train3.shape, y_train.shape)
print(type(X_train3)) #train_title_tfidf_w2v_vectors train_essay_tfidf_w2v_vecto
```

```
(49041, 708) (49041,)
<class 'scipy.sparse.csr.csr_matrix'>
```

In [193]:
```
X_test3 = hstack((test_categories_one_hot,test_sub_categories_one_hot,test_state_
                  test_teacher_one_hot, test_title_tfidf_w2v_vectors, test_essay_
                  test_quantity_standar, test_prev_proj_standar, test_price_stan
                  test_negitive_standar, test_neutral_standar, test_title_word_c
                  test_essay_word_count_standar)).tocsr()
print(X_test3.shape, y_test.shape)
print(type(X_test3)) #train_title_tfidf_w2v_vectors train_essay_tfidf_w2v_vectors
```

```
(36052, 708) (36052,)
<class 'scipy.sparse.csr.csr_matrix'>
```

In [194]:
```
X_cv3 = hstack((cv_categories_one_hot, cv_sub_categories_one_hot, cv_state_one_h
                 cv_teacher_one_hot, cv_title_tfidf_w2v_vectors, cv_essay_tfid
                 cv_quantity_standar, cv_prev_proj_standar, cv_price_standar,c
                 cv_negitive_standar, cv_neutral_standar, cv_title_word_count_
                 cv_essay_word_count_standar)).tocsr()
print(X_cv3.shape, y_cv.shape)
print(type(X_cv3)) #train_title_tfidf_w2v_vectors train_essay_tfidf_w2v_vectors
```

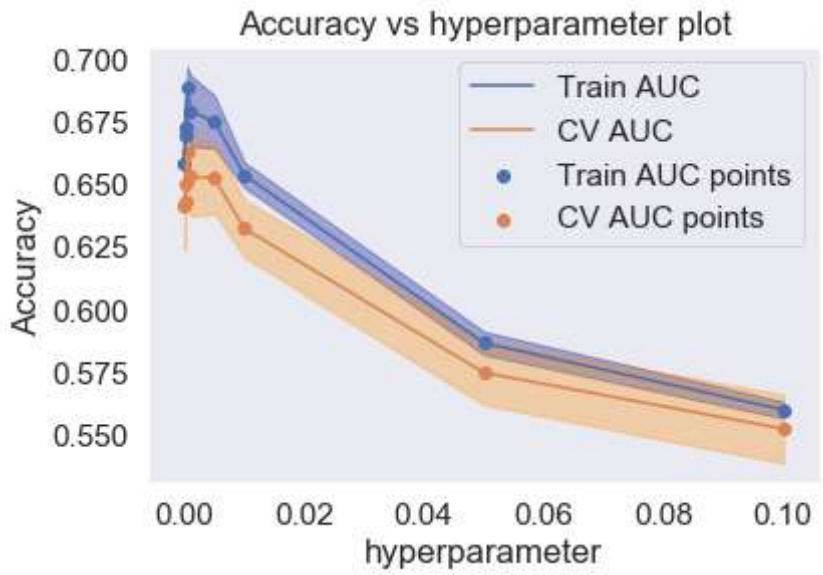
```
(24155, 708) (24155,)
<class 'scipy.sparse.csr.csr_matrix'>
```

In [195]:
```
print(X_train3.shape, y_train.shape)
```

```
(49041, 708) (49041,)
```

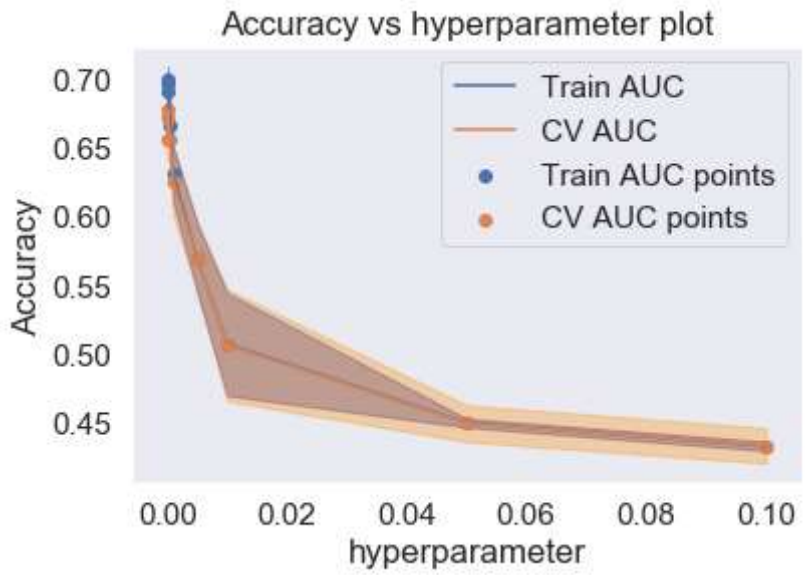
Hyperparameter Tunning

In [200]:
```
#we are using L2 Regularizer
parameters = {'alpha':[0.00001, 0.00005,0.0001, 0.0005, 0.001, 0.005, 0.01,0.05,(

SV = SGDClassifier(loss = 'hinge', penalty = 'l2')
classifier = GridSearchCV(SV, parameters, cv= 10, scoring='roc_auc')

classifier.fit(X_train3, y_train)

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

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
#https://stackoverflow.com/questions/48796282/how-to-visualize-dependence-of-mode
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_s

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


plt.legend()
plt.xlabel("hyperparameter")
plt.ylabel("Accuracy")
plt.title("Accuracy vs hyperparameter plot")
plt.grid()
plt.show()
```



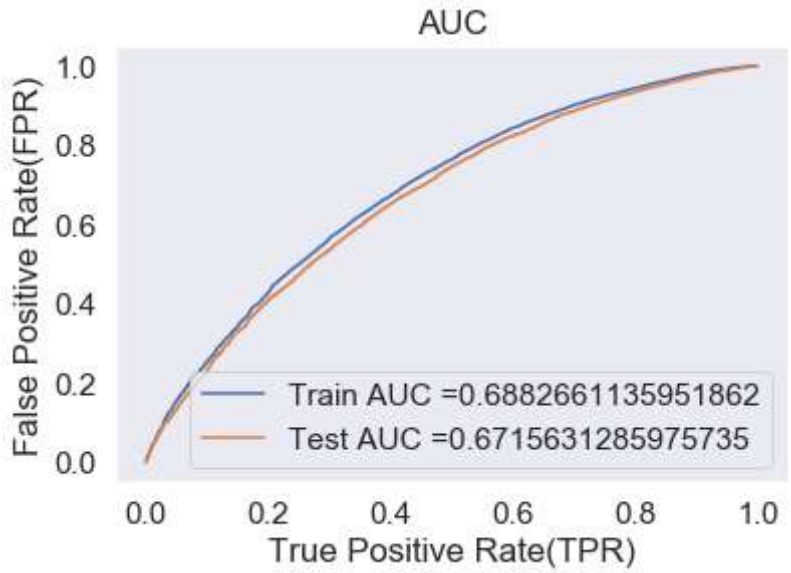
L1 Regularizer

In [201]:
```
#we are using L2 Regularizer as we know that L1 is Peforms badly
parameters = {'alpha':[0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01,0.05

SV = SGDClassifier(loss = 'hinge', penalty = 'l1')
classifier = GridSearchCV(SV, parameters, cv= 10, scoring='roc_auc')

classifier.fit(X_train3, y_train)

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

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
#https://stackoverflow.com/questions/48796282/how-to-visualize-dependence-of-mode
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_s

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


plt.legend()
plt.xlabel("hyperparameter")
plt.ylabel("Accuracy")
plt.title("Accuracy vs hyperparameter plot")
plt.grid()
plt.show()
```



In [202]:
```
best_c_4 = 0.0005
```

In [203]:
```
from sklearn.metrics import roc_curve, auc

Classifier_tfidfw2v = SGDClassifier(loss = 'hinge', penalty = 'l2', alpha = 0.00(

Classifier_tfidfw2v.fit(X_train3, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimate
# not the predicted outputs

y_train_pred = Classifier_tfidfw2v.decision_function(X_train3)
y_test_pred = Classifier_tfidfw2v.decision_function(X_test3)

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

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



Train Confusion Matrix

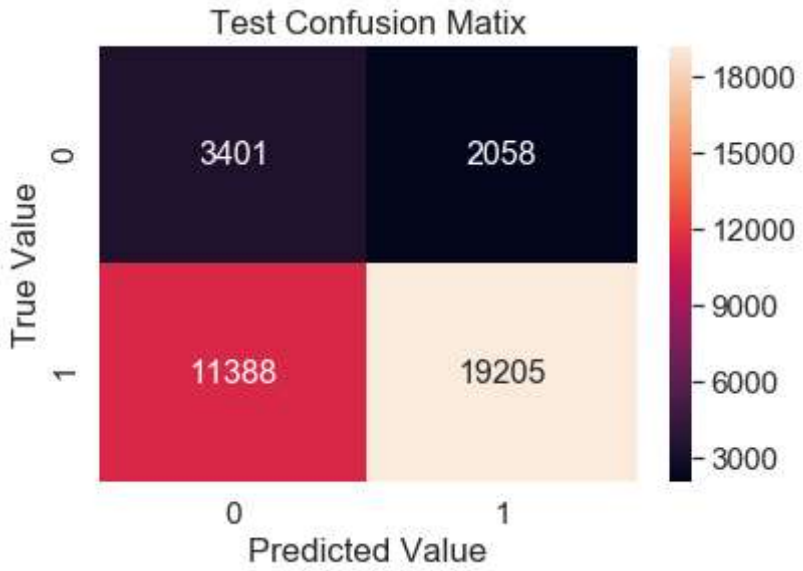
In [204]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sea
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr,
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.746
[[ 3713  3713]
 [ 9845 31770]]
```

In [205]:
```
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pre
                                                                       test_fpr,t
sea.set(font_scale=1.4)
sea.heatmap(train_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = '
plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Test Confusion Matix")
```

```
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.885
```

Out[205]:
```
Text(0.5, 1.0, 'Test Confusion Matix')
```



Test Confusion Matrix

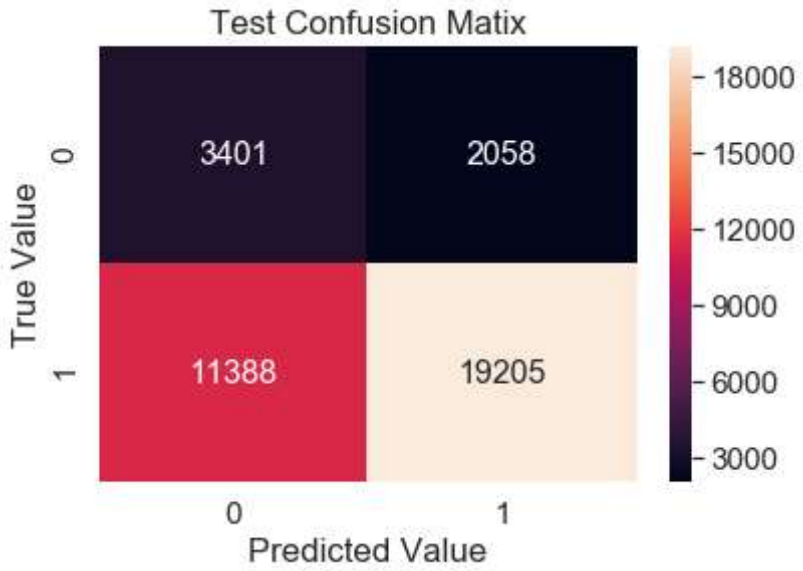
In [206]:
```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, tes
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.885
[[ 3401  2058]
 [11388 19205]]
```

In [207]:
```
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pre
                                                                       test_fpr,t
sea.set(font_scale=1.4)
sea.heatmap(train_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = '
plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Test Confusion Matix")
```

```
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.885
```

Out[207]:
```
Text(0.5, 1.0, 'Test Confusion Matix')
```



2.5 Apply Logistic Regression on Set-5

[Task-2] Apply Logistic Regression on the below feature set Set 5 by finding the best hyper parameter GridSearch

Consider these set of features for Set 5 in Assignment:

categorical dataschool_state clean_categories....clean_subcategories....project_grade_category....teacher_prefix

numerical data quantity....teacher_number_of_previously_posted_projects....price

New Features:-

sentiment score's of each of the essay : numerical data
number of words in the title : numerical data
number of words in the combine essays : numerical data

In [104]:
```
from scipy.sparse import hstack
X_train3 = hstack((train_categories_one_hot, train_sub_categories_one_hot, train_
                   train_teacher_one_hot, train_quantity_standar, train_prev_proj
                   train_title_word_count_standar, train_essay_word_count_standar
                   train_negitive_standar, train_neutral_standar)).tocsr()
print(X_train3.shape, y_train.shape)
print(type(X_train3))
```

```
(49041, 57) (49041,)
<class 'scipy.sparse.csr.csr_matrix'>
```

In [105]:
```
X_test3 = hstack((test_categories_one_hot, test_sub_categories_one_hot, test_grad
                  test_teacher_one_hot, test_quantity_standar, test_prev_proj_st
                  test_title_word_count_standar, test_essay_word_count_standar,
                  test_negitive_standar, test_neutral_standar)).tocsr()
print(X_test3.shape, y_train.shape)
print(type(X_test3))
```

```
(36052, 57) (49041,)
<class 'scipy.sparse.csr.csr_matrix'>
```

In [106]:
```
X_cv3 = hstack((cv_categories_one_hot, cv_sub_categories_one_hot, cv_grade_one_ho
                  cv_teacher_one_hot, cv_quantity_standar, cv_prev_proj_standar
                  cv_title_word_count_standar, cv_essay_word_count_standar, cv_
                  cv_negitive_standar, cv_neutral_standar)).tocsr()
print(X_cv3.shape, y_train.shape)
print(type(X_cv3))
```

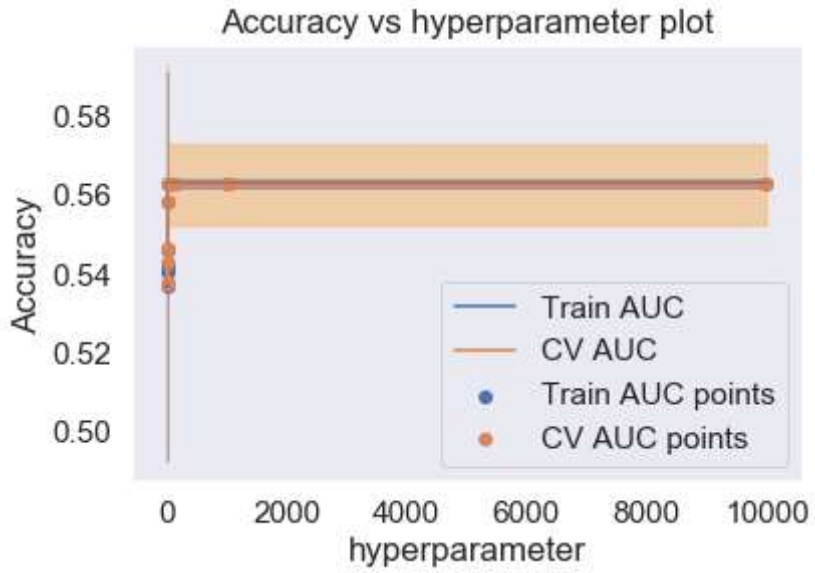
```
(24155, 57) (49041,)
<class 'scipy.sparse.csr.csr_matrix'>
```

In [107]:
```
y_trainn = y_train[0:24155,]
print(y_trainn.shape)
```

```
(24155,)
```

Hyperparameter Tunning

In [224]:
```
parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**

SV = SGDClassifier(loss = 'hinge', penalty = 'l2')
classifier = GridSearchCV(SV, parameters, cv= 10, scoring='roc_auc')

classifier.fit(X_train3, y_train)

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

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_s

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


plt.legend()
plt.xlabel("hyperparameter")
plt.ylabel("Accuracy")
plt.title("Accuracy vs hyperparameter plot")
plt.grid()
plt.show()
```



Apply TruncatedSVD on TfidfVectorizer of essay text, choose the number of components (n_components) using Elbow Method on Numerical data

In [61]:
```
#There are 12132 dimensions here
train_text_tfidf1 = train_text_tfidf[:,0:2999]
print(train_text_tfidf1.shape)

test_text_tfidf1 = test_text_tfidf[:,0:2999]
print(test_text_tfidf1.shape)

cv_text_tfidf1 = cv_text_tfidf[:,0:2999]
print(cv_text_tfidf1.shape)
```

```
(49041, 2999)
(36052, 2999)
(24155, 2999)
```

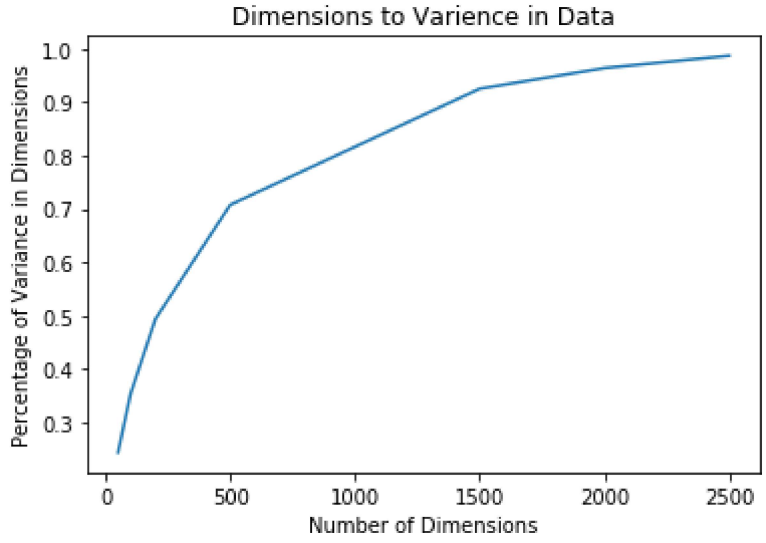
In [43]:
```
from sklearn.decomposition import TruncatedSVD
#https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.Truncate
#declaring index as Dimensions in train_text_tfidf
Dim = [50,100,200,500,1500,2000,2500]
Varience_sum = []
for i in tqdm(Dim):
    svd = TruncatedSVD(n_components = i, random_state = 42)
    svd.fit(train_text_tfidf1)
    Varience_sum.append(svd.explained_variance_ratio_.sum())
```

```
  0%|
| 0/7 [00:00<?, ?it/s]
 14%|
| 1/7 [00:02<00:14,  2.46s/it]
 29%|
| 2/7 [00:06<00:14,  2.98s/it]
 43%|
| 3/7 [00:14<00:17,  4.32s/it]
 57%|
| 4/7 [00:36<00:28,  9.63s/it]
 71%|
| 5/7 [01:53<00:59, 29.94s/it]
 86%|
| 6/7 [03:51<00:56, 56.46s/it]
100%|
| 7/7 [06:38<00:00, 89.40s/it]
```

In [44]:
```
Varience_sum
```

Out[44]:
```
[0.24436991118923532,
 0.353698559460497,
 0.49429668430423584,
 0.707750338643848,
 0.9250949868117755,
 0.9637410083977519,
 0.9868316170611472]
```

In [45]:
```
plt.xlabel("Number of Dimensions")
plt.ylabel("Percentage of Variance in Dimensions")
plt.title("Dimensions to Varience in Data")
plt.plot(Dim,Varience_sum)
plt.show()
```



At 2000 dimensions we are seeing Accuracy greater than 90% so considering 2000 dim

In [47]:
```
#Train SVD
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components= 2000, random_state=42)
svd.fit(train_text_tfidf1)
svd_train = svd.transform(train_text_tfidf1)
```

In [48]:
```
print("Shape of SVD Train Matrix is ", svd_train.shape)
```

```
Shape of SVD Train Matrix is  (49041, 2000)
```

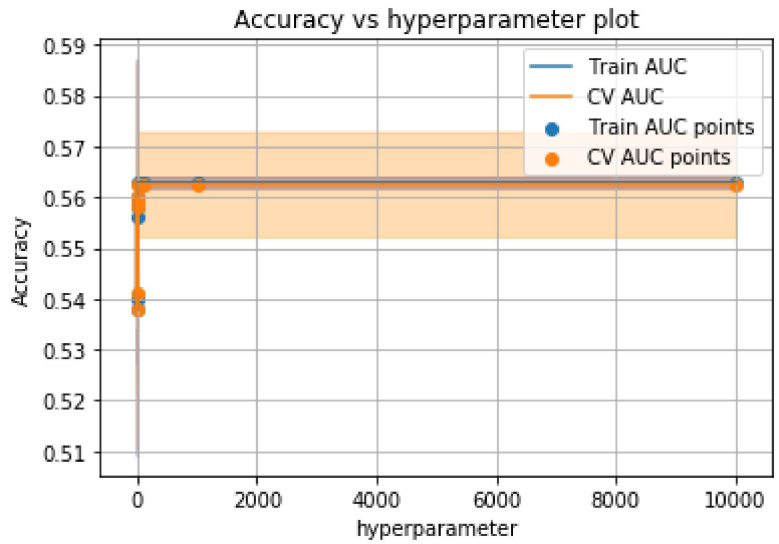
In [63]:
```
#Test SVD
svd_test = svd.transform(test_text_tfidf1)
print("Shape of matrix after Decomposition ",svd_test.shape)
```

```
Shape of matrix after Decomposition  (36052, 2000)
```

In [ ]:
```
#CV SVD
svd_cv = svd.transform()
print("Shape of matrix after Decomposition ",svd_cv.shape)
```

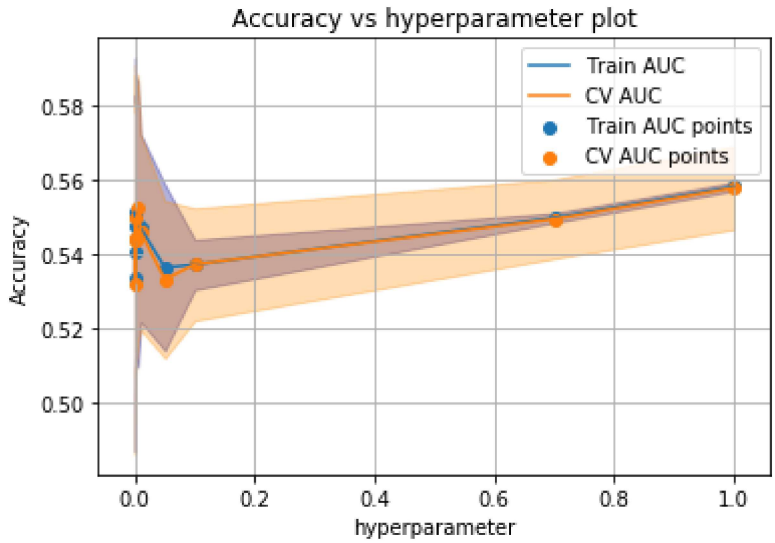
Grid Search

In [111]:
```
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import GridSearchCV
parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**

SV = SGDClassifier(loss = 'hinge', penalty = 'l2')
classifier = GridSearchCV(SV, parameters, cv= 10, scoring='roc_auc')

classifier.fit(X_train3, y_train)

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

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

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

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


plt.legend()
plt.xlabel("hyperparameter")
plt.ylabel("Accuracy")
plt.title("Accuracy vs hyperparameter plot")
plt.grid()
plt.show()
```

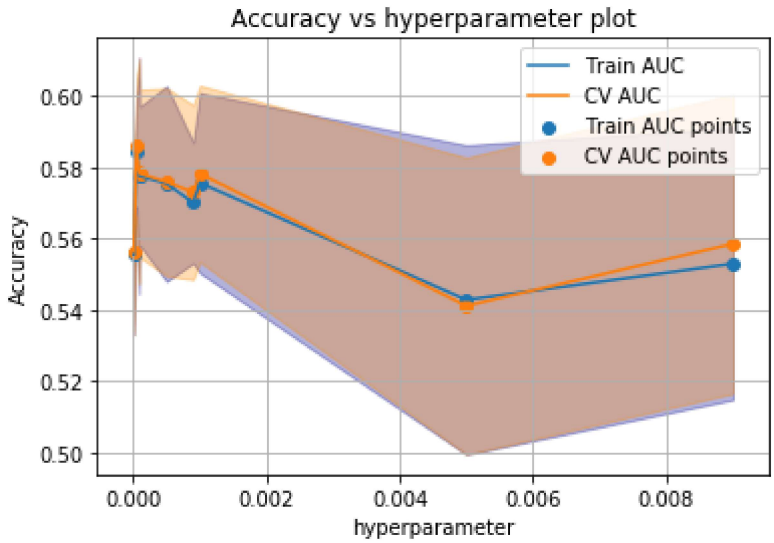


In [ ]:

In [112]:
```
#we are using L2 Regularizer
parameters = {'alpha':[0.00001, 0.00005,0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05

SV = SGDClassifier(loss = 'hinge', penalty = 'l2')
classifier = GridSearchCV(SV, parameters, cv= 10, scoring='roc_auc')

classifier.fit(X_train3, y_train)

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

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
#https://stackoverflow.com/questions/48796282/how-to-visualize-dependence-of-mode
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_s

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


plt.legend()
plt.xlabel("hyperparameter")
plt.ylabel("Accuracy")
plt.title("Accuracy vs hyperparameter plot")
plt.grid()
plt.show()
```



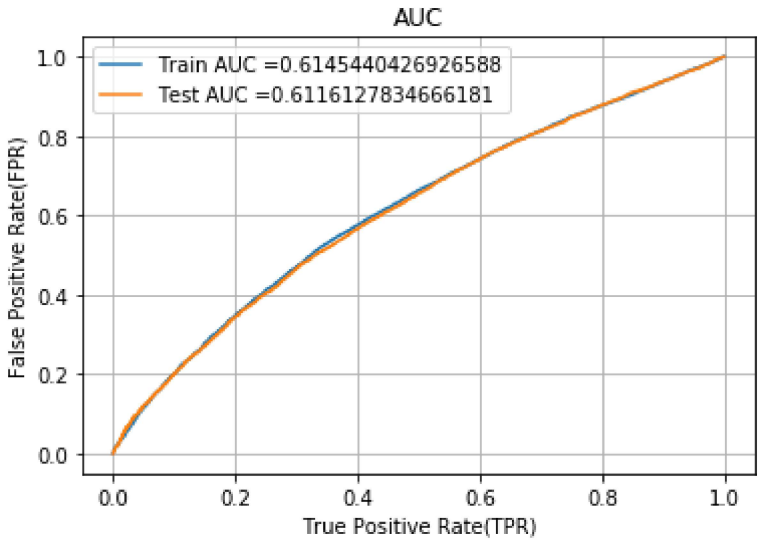
In [117]:
```
#we are using L2 Regularizer
parameters = {'alpha':[0.00001, 0.00005, 0.00009, 0.0001, 0.0005, 0.0009, 0.001,

SV = SGDClassifier(loss = 'hinge', penalty = 'l1')
classifier = GridSearchCV(SV, parameters, cv= 10, scoring='roc_auc')

classifier.fit(X_train3, y_train)

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

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
#https://stackoverflow.com/questions/48796282/how-to-visualize-dependence-of-mode
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_s

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


plt.legend()
plt.xlabel("hyperparameter")
plt.ylabel("Accuracy")
plt.title("Accuracy vs hyperparameter plot")
plt.grid()
plt.show()
```



Training Model using Best Hyper Parameter

Here the Best Hyperparameter Seems to be 0.0008

In [128]:
```
from sklearn.metrics import roc_curve, auc

Classifier_tfidfw2v = SGDClassifier(loss = 'hinge', penalty = 'l1', alpha = 0.00

Classifier_tfidfw2v.fit(X_train3, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimat
# not the predicted outputs

y_train_pred = Classifier_tfidfw2v.decision_function(X_train3)
y_test_pred = Classifier_tfidfw2v.decision_function(X_test3)

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

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



Train Confusion Matrix

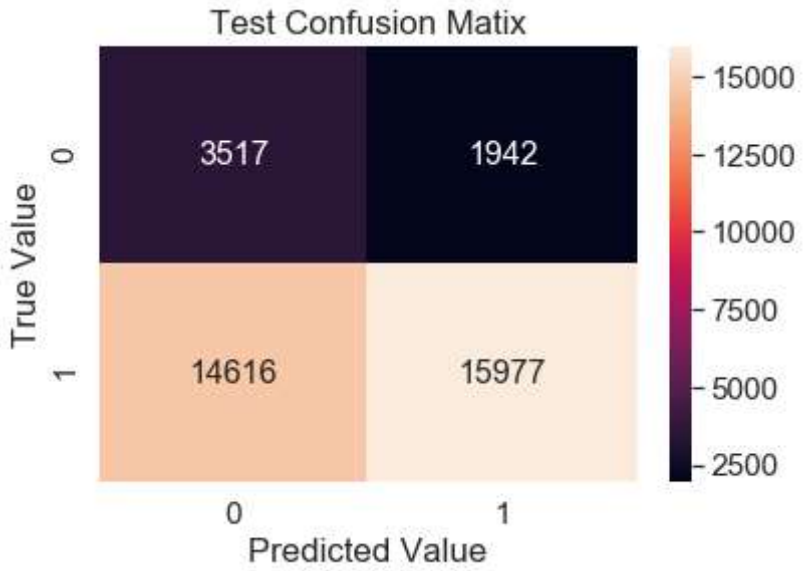
In [134]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sea
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr,
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 1.008
[[ 3713  3713]
 [14007 27608]]
```

In [135]:
```
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pre
                                                                       test_fpr,t
sea.set(font_scale=1.4)
sea.heatmap(train_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = '
plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Test Confusion Matix")
```

```
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 1.013
```

Out[135]:
```
Text(0.5, 1.0, 'Test Confusion Matix')
```



Test Confusion Matrix

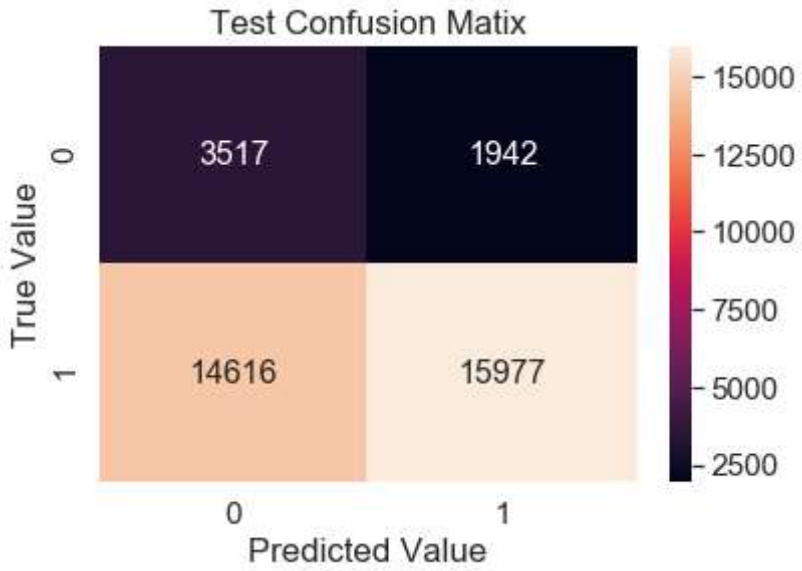
In [136]:
```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, tes
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 1.013
[[ 3517  1942]
 [14616 15977]]
```

In [137]:
```
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pre
                                                                       test_fpr,t
sea.set(font_scale=1.4)
sea.heatmap(train_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = '
plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Test Confusion Matix")
```

```
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 1.013
```

Out[137]: Text(0.5, 1.0, 'Test Confusion Matix')



3. Conclusion

In [141]:
```
# Please compare all your models using Prettytable library
# http://zetcode.com/python/prettytable/

from prettytable import PrettyTable
TB = PrettyTable()
TB.field_names = ["Vectorizer", "C:Hyperparameter", "Regularizer", "Train_AUC",
TB.title = "Support Vector Regression"
TB.add_row(["BOW-Model", 0.01, "L1", 0.78,0.68])
TB.add_row(["TFIDF-Model", 0.001, "L1", 0.77, 0.69])
TB.add_row(["AvgW2v-Model", 0.0005,"L2", 0.69, 0.68])
TB.add_row(["Tf-Idf-Model", 0.0002, "L1", 0.68, 0.67])
TB.add_row(["NUM_Features-Model",0.0008,"L1", 0.62, 0.61])
print(TB)
```

```
+--------------------+------------------+-------------+-----------+----------+
|     Vectorizer     | C:Hyperparameter | Regularizer | Train_AUC | Test_Auc |
+--------------------+------------------+-------------+-----------+----------+
|     BOW-Model      |       0.01       |      L1     |    0.78   |   0.68   |
|    TFIDF-Model     |      0.001       |      L1     |    0.77   |   0.69   |
|    AvgW2v-Model    |      0.0005      |      L2     |    0.69   |   0.68   |
|    Tf-Idf-Model    |      0.0002      |      L1     |    0.68   |   0.67   |
| NUM_Features-Model |      0.0008      |      L1     |    0.62   |   0.61   |
+--------------------+------------------+-------------+-----------+----------+
```

Observations:

As we have used all BOW, TF-IDF, AVG-W2V, TF-IDF weighted W2V encoding and in any of the encoding technique we didn't find best model which gives maximum accuracy. But from above models AvgW2v and tf-idf weighted w2v is performing best.

In [ ]: