About the DonorsChoose Data Set

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

Feature	Description
`project_id`	A unique identifier for the proposed project. **Example:** `p036502`
`project_title`	Title of the project. **Examples:** • `Art Will Make You Happy!` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targeted. One of the following enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories for the project from the following enumerated list of values: • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:** • `Music & The Arts` • `Literacy & Language, Math & Science`
`school_state`	State where school is located (Two-letter U.S. postal code). **Example:** `WY`
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the project. **Examples:** • `Literacy` • `Literature & Writing, Social Sciences`
`project_resource_summary`	An explanation of the resources needed for the project. **Example:** • `My students need hands on literacy materials to manage sensory needs!`
`project_essay_1`	First application essay*
`project_essay_2`	Second application essay*
`project_essay_3`	Third application essay*
`project_essay_4`	Fourth application essay*
`project_submitted_datetime`	Datetime when project application was submitted. **Example:** `2016-04-28 12:43:56.245`
`teacher_id`	A unique identifier for the teacher of the proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c15c56`
`teacher_prefix`	Teacher's title. One of the following enumerated values: • `nan` • `Dr.` • `Mr.` • `Mrs.` • `Ms.` • `Teacher.`
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted by the same teacher. **Example:** `2`

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

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

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

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

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

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

Notes on the Essay Data

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

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

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

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

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

In [1]:

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

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

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

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

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

from tqdm import tqdm
```

```
import os

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


from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from  sklearn.model_selection import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
from  sklearn.model_selection import cross_validate
```

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

1.1 Reading Data

In [2]:

```
project_data = pd.read_csv('train_data.csv',nrows=50000)
```

In [3]:

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

```
Number of data points in train data (50000, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [4]:

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

In [5]:

```
print("Number of data points in train data", resource_data.shape)
print('-'*50)
print("The attributes of data :", resource_data.columns.values)
resource_data.head(2)
```

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

Out[5]:

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

In [6]:

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

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

# how to reorder columns pandas python: https://stackoverflow.com/a/13148611/4084039
project_data = project_data[cols]
project_data.head(2)
```

Out[6]:

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

1.2 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.com/a/47301924/4084039
# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
cat_list = []
for i in catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & H
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math &
Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

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

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

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

1.3 preprocessing of `project_subject_subcategories`

In [8]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039
# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
```

```
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python

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

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

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

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

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	Date	project_grade_category	project_t
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2	Flexi Seating Flexi Learn
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Grades 3-5	Going De The Ar In Thinki

In [11]:

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

```
I recently read an article about giving students a choice about how they learn. We already set goa
ls; why not let them choose where to sit, and give them options of what to sit on?I teach at a low
-income (Title 1) school. Every year, I have a class with a range of abilities, yet they are all t
he same age. They learn differently, and they have different interests. Some have ADHD, and some a
re fast learners. Yet they are eager and active learners that want and need to be able to move arc
```

re fast learners. Yet they are eager and active learners that want and need to be able to move aro
und the room, yet have a place that they can be comfortable to complete their work.We need a class
room rug that we can use as a class for reading time, and students can use during other learning t
imes. I have also requested four Kore Kids wobble chairs and four Back Jack padded portable chairs
so that students can still move during whole group lessons without disrupting the class. Having th
ese areas will provide these little ones with a way to wiggle while working.Benjamin Franklin once
said, \"Tell me and I forget, teach me and I may remember, involve me and I learn.\" I want these
children to be involved in their learning by having a choice on where to sit and how to learn, all
by giving them options for comfortable flexible seating.
==

In [12]:

```
# 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 [13]:

```
sent = decontracted(project_data['essay'].values[0])
print(sent)
print("="*50)
```

I recently read an article about giving students a choice about how they learn. We already set goa
ls; why not let them choose where to sit, and give them options of what to sit on?I teach at a low
-income (Title 1) school. Every year, I have a class with a range of abilities, yet they are all t
he same age. They learn differently, and they have different interests. Some have ADHD, and some a
re fast learners. Yet they are eager and active learners that want and need to be able to move aro
und the room, yet have a place that they can be comfortable to complete their work.We need a class
room rug that we can use as a class for reading time, and students can use during other learning t
imes. I have also requested four Kore Kids wobble chairs and four Back Jack padded portable chairs
so that students can still move during whole group lessons without disrupting the class. Having th
ese areas will provide these little ones with a way to wiggle while working.Benjamin Franklin once
said, \"Tell me and I forget, teach me and I may remember, involve me and I learn.\" I want these
children to be involved in their learning by having a choice on where to sit and how to learn, all
by giving them options for comfortable flexible seating.
==

In [14]:

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

I recently read an article about giving students a choice about how they learn. We already set goa
ls; why not let them choose where to sit, and give them options of what to sit on?I teach at a low
-income (Title 1) school. Every year, I have a class with a range of abilities, yet they are all t
he same age. They learn differently, and they have different interests. Some have ADHD, and some a
re fast learners. Yet they are eager and active learners that want and need to be able to move aro
und the room, yet have a place that they can be comfortable to complete their work.We need a class
room rug that we can use as a class for reading time, and students can use during other learning t
imes. I have also requested four Kore Kids wobble chairs and four Back Jack padded portable chairs
so that students can still move during whole group lessons without disrupting the class. Having th
ese areas will provide these little ones with a way to wiggle while working.Benjamin Franklin once
said, Tell me and I forget, teach me and I may remember, involve me and I learn. I want these ch
ildren to be involved in their learning by having a choice on where to sit and how to learn, all b

y giving them options for comfortable flexible seating.

In [15]:

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

```
I recently read an article about giving students a choice about how they learn We already set goal
s why not let them choose where to sit and give them options of what to sit on I teach at a low in
come Title 1 school Every year I have a class with a range of abilities yet they are all the same
age They learn differently and they have different interests Some have ADHD and some are fast lear
ners Yet they are eager and active learners that want and need to be able to move around the room
yet have a place that they can be comfortable to complete their work We need a classroom rug that
we can use as a class for reading time and students can use during other learning times I have als
o requested four Kore Kids wobble chairs and four Back Jack padded portable chairs so that student
s can still move during whole group lessons without disrupting the class Having these areas will
provide these little ones with a way to wiggle while working Benjamin Franklin once said Tell me a
nd I forget teach me and I may remember involve me and I learn I want these children to be involve
d in their learning by having a choice on where to sit and how to learn all by giving them options
for comfortable flexible seating
```

In [16]:

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

In [17]:

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

```
100%|████████████████████████████████████████████████████████████████████████████| 50000/50000 [01:
36<00:00, 517.34it/s]
```

In [18]:

```
# after preprocesing
preprocessed_essays[0]
```

Out[18]:

```
'recently read article giving students choice learn already set goals not let choose sit give opti
ons sit teach low income title 1 school every year class range abilities yet age learn differently
different interests adhd fast learners yet eager active learners want need able move around room y
et place comfortable complete work need classroom rug use class reading time students use learning
times also requested four kore kids wobble chairs four back jack padded portable chairs students s
till move whole group lessons without disrupting class areas provide little ones way wiggle
working benjamin franklin said tell forget teach may remember involve learn want children involved
learning choice sit learn giving options comfortable flexible seating'
```

1.4 Preprocessing of `project_title`

In [19]:

```
# preprocessing of project title
```

In [20]:

```
sent = decontracted(project_data['project_title'].values[0])
# \r \n \t remove from string python: http://texthandler.com/info/remove-line-breaks-python/
sent = sent.replace('\\r', ' ')
sent = sent.replace('\\"', ' ')
sent = sent.replace('\\n', ' ')
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
print(sent)
```

```
Flexible Seating for Flexible Learning
```

In [21]:

```
# Combining all the above statemennts
from tqdm import tqdm
preprocessed_project_title = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['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)
    preprocessed_project_title.append(sent.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████████████████| 50000/50000
[00:03<00:00, 12615.73it/s]
```

1.5 Preparing data for models

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

- 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

In [22]:

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

Out[22]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_t
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2	Flexi Seating Flexi Learn
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Grades 3-5	Going De The Ar In Thinki

In [23]:

```
#number of words in project titlefor set 5 new feature
```

In [24]:

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

```
100%|████████████████████████████████████████████████████████████████| 50000/50000
[00:01<00:00, 27087.01it/s]
```

In [25]:

```
#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%|████████████████████████████████████████████████████████████████| 50000/50000
[00:00<00:00, 240230.20it/s]
```

In [26]:

```
#number of words in project title for set 5 new feature
```

In [27]:

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

```
100%|████████████████████████████████████████████████████████████████| 50000/50000
[00:03<00:00, 15104.13it/s]
```

In [28]:

```
#Introducing New Features
essay_word_count = []
#for i in project_data['project_title']:
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%|████████████████████████████████████████████████████████████████| 50000/50000
[00:02<00:00, 18050.86it/s]
```

In [29]:

```
#split the data into train ,test and cross validation
```

In [30]:

```
y =project_data['project_is_approved'].values
project_data.drop(['project_is_approved'], axis=1, inplace=True)
print(project_data.shape)
```

```
(50000, 19)
```

In [31]:

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

Out[31]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_ti
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2	Flexi Seating Flexi Learn
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Grades 3-5	Going De The Ar In Thinki

Computing Sentiment Scores

In [32]:

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

# import nltk
nltk.download('vader_lexicon')

sid = SentimentIntensityAnalyzer()

for_sentiment = 'a person is a person no matter how small dr seuss i teach the smallest students w
ith the biggest enthusiasm'
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,
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\myuri\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

In [33]:

```
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%|██████████████████████████████████████████████████████████████████████████████| 50000/50000 [25
:35<00:00, 33.62it/s]
```

In [34]:

```
project_data['negitive'] = negitive
```

In [35]:

```
project_data['positive'] = positive
project_data['neutral'] = neutral
project_data['compound'] = compound
project_data.head(2)
```

Out[35]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_t
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2	Flexi Seating Flexi Learn
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Grades 3-5	Going De The Ar In Thinki

2 rows × 23 columns

In [36]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
#Splitting data into Train and cross validation
# split the data set into train and test
X_train, X_test, y_train, y_test = train_test_split(project_data, y, test_size=0.33,stratify=y)
# split the train data set into cross validation train and cross validation test
```

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

In [37]:

```
print(X_train.columns)
```

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', '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',
       'negitive', 'positive', 'neutral', 'compound'],
      dtype='object')
```

In [38]:

```
print(X_train.shape)
print(X_test.shape)
print(X_cv.shape)
```

```
(22445, 23)
(16500, 23)
(11055, 23)
```

In [39]:

```
print(y_train.shape)
print(y_test.shape)
print(y_cv.shape)
```

```
(22445,)
(16500,)
(11055,)
```

In [40]:

```
print(X_train['essay'].values[0])
```

```
I am now a third-year head coach for the Girls Basketball Team. As a young coach I have learned mo
re than I could have dreamed in just a short amount of time. I truly love coaching and love the im
pact I get to make on a daily basis. These girls are so special and have so much potential. It is
exciting to see them come together and find joy in working towards a common goal. We do not have a
thletic funding so all the fundraising falls on our backs as coaches.Our goal this coming year is
to make school history and take our girls basketball team to state. It has never been accomplished
at this school and we believe that we have the tools to do so this year. \r\nWe are looking to get
a few training materials that will help them reach this lofty goal and for them to keep motivation
as we move forward. Having big cones to use in drills, getting a few more basketballs, and getting
a few items to give to girls as rewards for their efforts is a desire we have for our girls.nannan
```

In [41]:

```
#preprocessing of train ,cross validation and test essay data
```

In [42]:

```
#preprocess the X_train essay
```

In [43]:

```
# Combining all the above statemennts
from tqdm import tqdm
preprocessed_essay_train_data = []
# tqdm is for printing the status bar
for sentance in tqdm(X_train['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
```

```
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_essay_train_data.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████████████████| 22445/22445 [00:
37<00:00, 602.88it/s]
```

In [44]:

```
#preprocess the X_cv essay
```

In [45]:

```
# Combining all the above statemennts
from tqdm import tqdm
preprocessed_essay_cv_data = []
# 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)
    preprocessed_essay_cv_data.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████████████████| 11055/11055 [00:
17<00:00, 625.41it/s]
```

In [46]:

```
#preprocess the X_test essay
```

In [47]:

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

```
100%|████████████████████████████████████████████████████████████████████████████| 16500/16500 [00:
30<00:00, 536.39it/s]
```

In [48]:

```
#preprocessing of x_train,x_cv and x_test of project title
```

In [49]:

```
# Combining all the above statemennts of x_train
from tqdm import tqdm
train_preprocessed_project_title = []
# 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_project_title.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████████████████| 22445/22445
[00:03<00:00, 7311.46it/s]
```

In [50]:

```
# Combining all the above statemennts x_cv
from tqdm import tqdm
cv_preprocessed_project_title = []
# 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_project_title.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████████████████| 11055/11055
[00:01<00:00, 7651.92it/s]
```

In [51]:

```
# Combining all the above statemennts x_cv
from tqdm import tqdm
test_preprocessed_project_title = []
# 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_project_title.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████████████████| 16500/16500
[00:02<00:00, 8027.94it/s]
```

1.5.1 Vectorizing Categorical data

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

In [52]:

```
#vectorisation of clean categories
```

In [53]:

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

train_categories_one_hot=vect_categories.transform(X_train['clean_categories'].values)
cv_categories_one_hot=vect_categories.transform(X_cv['clean_categories'].values)
test_categories_one_hot=vect_categories.transform(X_test['clean_categories'].values)

print(vect_categories.get_feature_names())
```

```
print(vect_categories.get_feature_names())
print("Shape of train matrix after one hot encodig ",train_categories_one_hot.shape)
print("Shape of train matrix after one hot encodig ",cv_categories_one_hot.shape)
print("Shape of train matrix after one hot encodig ",test_categories_one_hot.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of train matrix after one hot encodig  (22445, 9)
Shape of train matrix after one hot encodig  (11055, 9)
Shape of train matrix after one hot encodig  (16500, 9)
```

In [54]:

```
#vectorisation of clean subcategories
```

In [55]:

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

train_subcategories_one_hot=vectorizer_subcategories.transform(X_train['clean_subcategories'].valu
es)
cv_subcategories_one_hot=vectorizer_subcategories.transform(X_cv['clean_subcategories'].values)
test_subcategories_one_hot=vectorizer_subcategories.transform(X_test['clean_subcategories'].values
)

print(vectorizer_subcategories.get_feature_names())
print("Shape of train matrix after one hot encodig ",train_subcategories_one_hot.shape)
print("Shape of train matrix after one hot encodig ",cv_subcategories_one_hot.shape)
print("Shape of train matrix after one hot encodig ",test_subcategories_one_hot.shape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL
', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of train matrix after one hot encodig  (22445, 30)
Shape of train matrix after one hot encodig  (11055, 30)
Shape of train matrix after one hot encodig  (16500, 30)
```

In [56]:

```
# Build the data matrix using these features-- school_state : categorical data (one hot encoding)
##Encoding for school state
```

In [57]:

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

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


from sklearn.feature_extraction.text import CountVectorizer
vectorizer_state = CountVectorizer(vocabulary=list(sorted_cat_dict_state.keys()), lowercase=False,
binary=True)
vectorizer_state.fit(project_data['school_state'].values)

train_state_one_hot=vectorizer_state.transform(X_train['school_state'].values)
cv_state_one_hot=vectorizer_state.transform(X_cv['school_state'].values)
test_state_one_hot=vectorizer_state.transform(X_test['school_state'].values)

print(vectorizer_state.get_feature_names())
```

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

```
['VT', 'WY', 'ND', 'MT', 'RI', 'NH', 'SD', 'NE', 'AK', 'DE', 'WV', 'ME', 'NM', 'HI', 'DC', 'KS', 'I
D', 'IA', 'AR', 'CO', 'MN', 'OR', 'MS', 'KY', 'NV', 'MD', 'TN', 'CT', 'AL', 'UT', 'WI', 'VA', 'AZ',
'NJ', 'OK', 'MA', 'LA', 'WA', 'MO', 'IN', 'OH', 'PA', 'MI', 'GA', 'SC', 'IL', 'NC', 'FL', 'TX', 'NY
', 'CA']
Shape of train matrix after one hot encodig  (22445, 51)
Shape of train matrix after one hot encodig  (11055, 51)
Shape of train matrix after one hot encodig  (16500, 51)
```

In [58]:

```
#Encoding for project_grade_category
```

In [59]:

```
project_data.project_grade_category = project_data.project_grade_category.str.replace('\s+', '_')
project_data.project_grade_category = project_data.project_grade_category.str.replace('-', '_')
project_data['project_grade_category'].value_counts()
```

Out[59]:

```
Grades_PreK_2    20316
Grades_3_5       16968
Grades_6_8        7750
Grades_9_12       4966
Name: project_grade_category, dtype: int64
```

In [60]:

```
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039

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

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

# we use count vectorizer to convert the values into one hot encoded features
from sklearn.feature_extraction.text import CountVectorizer
vectorizer_grade_cat = CountVectorizer(vocabulary=list(sorted_cat_dict_grade.keys()), lowercase=Fa
lse, binary=True)
vectorizer_grade_cat.fit(project_data['project_grade_category'].values)

train_grade_one_hot=vectorizer_grade_cat.transform(X_train['project_grade_category'].values)
cv_grade_one_hot=vectorizer_grade_cat.transform(X_cv['project_grade_category'].values)
test_grade_one_hot=vectorizer_grade_cat.transform(X_test['project_grade_category'].values)

print(vectorizer_grade_cat.get_feature_names())
print("Shape of train matrix after one hot encodig ",train_grade_one_hot.shape)
print("Shape of train matrix after one hot encodig ",cv_grade_one_hot.shape)
print("Shape of train matrix after one hot encodig ",test_grade_one_hot.shape)
```

```
{'Grades_9_12': 4966, 'Grades_6_8': 7750, 'Grades_3_5': 16968, 'Grades_PreK_2': 20316}
['Grades_9_12', 'Grades_6_8', 'Grades_3_5', 'Grades_PreK_2']
Shape of train matrix after one hot encodig  (22445, 4)
Shape of train matrix after one hot encodig  (11055, 4)
Shape of train matrix after one hot encodig  (16500, 4)
```

In [61]:

```
#Encoding for teacher_prefix
```

In [62]:

```
project_data.teacher_prefix= project_data.teacher_prefix.str.replace('\s+', '_')
project_data.teacher_prefix= project_data.teacher_prefix.str.replace('-', '_')
project_data['teacher_prefix'].value_counts()
```

Out[62]:

```
Mrs.       26140
Ms.        17936
Mr.         4859
Teacher     1061
Dr.            2
Name: teacher_prefix, dtype: int64
```

In [63]:

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

In [64]:

```
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039

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

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


# we use count vectorizer to convert the values into one hot encoded features
from sklearn.feature_extraction.text import CountVectorizer
vectorizer_teacher_prefix = CountVectorizer(vocabulary=list(sorted_cat_dict_prefix.keys()), lowerc
ase=False, binary=True)
vectorizer_teacher_prefix.fit(project_data['teacher_prefix'].values.astype('U'))

train_teacher_prefix_one_hot=vectorizer_teacher_prefix.transform(X_train['teacher_prefix'].values.
astype('U'))
cv_teacher_prefix_one_hot=vectorizer_teacher_prefix.transform(X_cv['teacher_prefix'].values.astype
('U'))
test_teacher_prefix_one_hot=vectorizer_teacher_prefix.transform(X_test['teacher_prefix'].values.as
type('U'))

print(vectorizer_teacher_prefix.get_feature_names())
print("Shape of train matrix after one hot encodig ",train_teacher_prefix_one_hot.shape)
print("Shape of train matrix after one hot encodig ",cv_teacher_prefix_one_hot.shape)
print("Shape of train matrix after one hot encodig ",test_teacher_prefix_one_hot.shape)
```

```
['Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
Shape of train matrix after one hot encodig  (22445, 5)
Shape of train matrix after one hot encodig  (11055, 5)
Shape of train matrix after one hot encodig  (16500, 5)
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [65]:

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

text_bow_essays_train = vectorizer_essay_bow.transform(preprocessed_essay_train_data)
print("Shape of matrix after one hot encodig ",text_bow_essays_train.shape)
```

```
Shape of matrix after one hot encodig  (22445, 30572)
```

In [66]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
text_bow_essays_cv = vectorizer_essay_bow.transform(preprocessed_essay_cv_data)
print("Shape of matrix after one hot encodig ",text_bow_essays_cv.shape)
```

```
Shape of matrix after one hot encodig  (11055, 30572)
```

In [67]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
text_bow_essays_test= vectorizer_essay_bow.transform(preprocessed_essay_test_data)
print("Shape of matrix after one hot encodig ",text_bow_essays_test.shape)
```

```
Shape of matrix after one hot encodig  (16500, 30572)
```

In [68]:

```
# you can vectorize the title also
# before you vectorize the title make sure you preprocess it
```

In [69]:

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

text_bow_title_train= vectorizer_bow_title.transform(train_preprocessed_project_title)
print("Shape of matrix after one hot encodig ",text_bow_title_train.shape)
```

```
Shape of matrix after one hot encodig  (22445, 8042)
```

In [70]:

```
text_bow_title_cv=vectorizer_bow_title.transform(cv_preprocessed_project_title)
print("Shape of matrix after one hot encodig ",text_bow_title_cv.shape)
```

```
Shape of matrix after one hot encodig  (11055, 8042)
```

In [71]:

```
text_bow_title_test= vectorizer_bow_title.transform(test_preprocessed_project_title)
print("Shape of matrix after one hot encodig ",text_bow_title_test.shape)
```

```
Shape of matrix after one hot encodig  (16500, 8042)
```

TFIDF Vectorizer on project_title

In [72]:

```
# Similarly you can vectorize for title also
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
vectorizer.fit(train_preprocessed_project_title)

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

```
Shape of matrix after one hot encodig  (22445, 8042)
```

In [73]:

```
# Similarly you can vectorize for title
```

```
# Similarly you can vectorize for title
text_tfidf_title_cv = vectorizer.transform(cv_preprocessed_project_title)
print("Shape of matrix after one hot encodig ",text_tfidf_title_cv.shape)
```

```
Shape of matrix after one hot encodig  (11055, 8042)
```

In [74]:

```
# Similarly you can vectorize for title also
text_tfidf_title_test = vectorizer.transform(test_preprocessed_project_title)
print("Shape of matrix after one hot encodig ",text_tfidf_title_test.shape)
```

```
Shape of matrix after one hot encodig  (16500, 8042)
```

TFIDF Vectorizer on preprocessed essay

In [75]:

```
# Similarly you can vectorize for title also
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
vectorizer.fit(preprocessed_essay_train_data)

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

```
Shape of matrix after one hot encodig  (22445, 30572)
```

In [76]:

```
# Similarly you can vectorize for title also
tfidf_essay_cv = vectorizer.transform(preprocessed_essay_cv_data)
print("Shape of matrix after one hot encodig ",tfidf_essay_cv.shape)
```

```
Shape of matrix after one hot encodig  (11055, 30572)
```

In [77]:

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

```
Shape of matrix after one hot encodig  (16500, 30572)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [78]:

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

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

Loading Glove Model
```

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

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

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

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

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

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


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

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


'''
```

Out[78]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039\ndef
loadGloveModel(gloveFile):\n    print ("Loading Glove Model")\n    f = open(gloveFile,\'r\',
encoding="utf8")\n    model = {}\n    for line in tqdm(f):\n        splitLine = line.split()\n
word = splitLine[0]\n        embedding = np.array([float(val) for val in splitLine[1:]])\n        m
odel[word] = embedding\n    print ("Done.",len(model)," words loaded!")\n    return model\nmodel =
loadGloveModel(\'glove.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    words.extend(i.split(\'
\'))\n\nfor i in preproced_titles:\n    words.extend(i.split(\' \'))\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(model.keys()).intersection(words)\nprint("The number of words tha
t are present in both glove vectors and our coupus",       len(inter_words),"
(",np.round(len(inter_words)/len(words)*100,3),"%)")\n\nwords_courpus = {}\nwords_glove =
set(model.keys())\nfor i in words:\n    if i in words_glove:\n        words_courpus[i] = model[i]\r
print("word 2 vec length", len(words_courpus))\n\n\n# stronging variables into pickle files python
: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/\n\nimport pic
kle\nwith open(\'glove_vectors\', \'wb\') as f:\n    pickle.dump(words_courpus, f)\n\n\n'
```

In [79]:

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

In [80]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_essay_train_data = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essay_train_data): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
```

```
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_essay_train_data.append(vector)

print(len(avg_w2v_essay_train_data))
print(len(avg_w2v_essay_train_data[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 22445/22445 [00:
25<00:00, 888.32it/s]
```

```
22445
300
```

In [81]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_essay_cv_data = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essay_cv_data): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_essay_cv_data.append(vector)

print(len(avg_w2v_essay_cv_data))
print(len(avg_w2v_essay_cv_data[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 11055/11055 [00:
11<00:00, 777.95it/s]
```

```
11055
300
```

In [82]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_essay_test_data = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essay_test_data): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_essay_test_data.append(vector)

print(len(avg_w2v_essay_test_data))
print(len(avg_w2v_essay_test_data[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 16500/16500 [00:
17<00:00, 947.39it/s]
```

```
16500
300
```

In [83]:

```
#Using Pretrained Models: Avg W2V for project title
```

In [84]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_project_title_train_data = []; # the avg-w2v for each sentence/review is stored in this li
st
for sentence in tqdm(train_preprocessed_project_title): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_project_title_train_data.append(vector)

print(len(avg_w2v_project_title_train_data))
print(len(avg_w2v_project_title_train_data[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 22445/22445
[00:01<00:00, 18685.32it/s]
```

```
22445
300
```

In [85]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_project_title_cv_data = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(cv_preprocessed_project_title): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_project_title_cv_data.append(vector)

print(len(avg_w2v_project_title_cv_data))
print(len(avg_w2v_project_title_cv_data[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 11055/11055
[00:00<00:00, 14547.63it/s]
```

```
11055
300
```

In [86]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_project_title_test_data = []; # the avg-w2v for each sentence/review is stored in this lis
t
for sentence in tqdm(test_preprocessed_project_title): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_project_title_test_data.append(vector)

print(len(avg_w2v_project_title_test_data))
print(len(avg_w2v_project_title_test_data[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 16500/16500
[00:00<00:00, 17965.68it/s]
```

```
16500
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [87]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_essay_train_data)
# 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 [88]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_essay_train_data = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essay_train_data): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_essay_train_data.append(vector)

print(len(tfidf_w2v_essay_train_data))
print(len(tfidf_w2v_essay_train_data[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 22445/22445 [02:
01<00:00, 184.81it/s]
```

```
22445
300
```

In [89]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_essay_cv_data = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essay_cv_data): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_essay_cv_data.append(vector)

print(len(tfidf_w2v_essay_cv_data))
print(len(tfidf_w2v_essay_cv_data[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████████| 11055/11055 [00:57<00:00, 192.97it/s]
```

```
11055
300
```

In [90]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_essay_test_data = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essay_test_data): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_essay_test_data.append(vector)

print(len(tfidf_w2v_essay_test_data))
print(len(tfidf_w2v_essay_test_data[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████████| 16500/16500 [01:24<00:00, 195.69it/s]
```

```
16500
300
```

Using Pretrained Models: TFIDF weighted W2V on project_title

In [91]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(train_preprocessed_project_title)
# 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 [92]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_train_project_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(train_preprocessed_project_title): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_train_project_title.append(vector)

print(len(tfidf_w2v_train_project_title))
```

```
print(len(tfidf_w2v_train_project_title[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 22445/22445
[00:02<00:00, 11102.67it/s]
```

```
22445
300
```

In [93]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_cv_project_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(cv_preprocessed_project_title): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_cv_project_title.append(vector)

print(len(tfidf_w2v_cv_project_title))
print(len(tfidf_w2v_cv_project_title[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 11055/11055
[00:00<00:00, 11380.56it/s]
```

```
11055
300
```

In [94]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_test_project_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(test_preprocessed_project_title): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_test_project_title.append(vector)

print(len(tfidf_w2v_test_project_title))
print(len(tfidf_w2v_test_project_title[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 16500/16500
[00:01<00:00, 11446.78it/s]
```

```
16500
300
```

In [95]:

```
print(X_train.columns)
```

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', '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',
       'negitive', 'positive', 'neutral', 'compound'],
      dtype='object')
```

In [96]:

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

In [97]:

```
x_train = pd.merge(X_train, price_data, on = "id", how = "left")
x_test = pd.merge(X_test, price_data, on = "id", how = "left")
x_cv = pd.merge(X_cv, price_data, on = "id", how = "left")
```

In [98]:

```
print(x_train.shape)
```

```
(22445, 25)
```

Vectorizing Numerical features price

In [99]:

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

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

price_scalar = StandardScaler()

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

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

```
Mean : 300.767233236801, Standard deviation : 376.49697505714016
```

Out[99]:

```
array([[-0.36878711],
       [-0.52807126],
       [-0.27030027],
       ...,
       [ 0.35026249],
       [-0.0054907 ],
       [ 0.04561196]])
```

In [100]:

```
price_scalar.fit(x_test['price'].values.reshape(-1,1)) # finding the mean and standard deviation
```

```
of this data
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")

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

```
Mean : 298.98534969696965, Standard deviation : 390.8851480602115
```

Out[100]:

```
array([[-0.08264665],
       [-0.43313324],
       [-0.2046518 ],
       ...,
       [ 0.51169673],
       [-0.49798605],
       [-0.38117424]])
```

In [101]:

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

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

```
Mean : 296.9430113071009, Standard deviation : 362.08457813276
```

Out[101]:

```
array([[-0.08264665],
       [-0.43313324],
       [-0.2046518 ],
       ...,
       [ 0.51169673],
       [-0.49798605],
       [-0.38117424]])
```

In [102]:

```
print(train_price_standar.shape, y_train.shape)
print(test_price_standar.shape, y_test.shape)
print(cv_price_standar.shape, y_cv.shape)
```

```
(22445, 1) (22445,)
(16500, 1) (16500,)
(11055, 1) (11055,)
```

Vectorizing teacher_number_of_previously_posted_projects

In [103]:

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

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

```
C:\Users\myuri\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

```
Mean : 11.273423925150368, Standard deviation : 28.703704478818878
```

```
C:\Users\myuri\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

Out[103]:

```
array([[-0.07920315],
       [-0.11404186],
       [-0.28823541],
       ...,
       [-0.39275153],
       [ 6.122087  ],
       [-0.21855799]])
```

In [104]:

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

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

```
C:\Users\myuri\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

```
Mean : 11.160545454545455, Standard deviation : 27.458989492421303
```

```
C:\Users\myuri\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

Out[104]:

```
array([[-0.40644414],
       [-0.26077236],
       [-0.40644414],
       ...,
       [-0.3700262 ],
       [-0.33360825],
       [-0.3700262 ]])
```

In [105]:

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

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

```
C:\Users\myuri\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

```
Mean : 11.332971506105835, Standard deviation : 28.063924142088887
```

```
C:\Users\myuri\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

Out[105]:

```
array([[-0.33256117],
       [-0.15439649],
       [-0.36819411],
       ...,
       [ 2.51807367],
       [-0.40382704],
       [-0.36819411]])
```

Standardize Quantity

In [106]:

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

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

```
C:\Users\myuri\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

```
Mean : 17.077656493651148, Standard deviation : 26.63703678030565
```

```
C:\Users\myuri\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

Out[106]:

```
array([[ 0.33496006],
       [-0.37833249],
       [-0.34079078],
       ...,
       [ 0.56021034],
       [ 0.03462636],
       [ 2.06187888]])
```

In [107]:

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

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

```
C:\Users\myuri\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

```
Mean : 17.122, Standard deviation : 26.63505939034626
```

```
C:\Users\myuri\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning:
```

```
Data with input dtype int64 was converted to float64 by StandardScaler.
```

Out[107]:

```
array([[ 5.70218364],
       [ 0.40840908],
       [-0.49265894],
       ...,
       [-0.60529244],
       [-0.38002543],
       [-0.56774794]])
```

In [108]:

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

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

```
C:\Users\myuri\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

```
Mean : 16.915422885572138, Standard deviation : 27.402449685971902
```

```
C:\Users\myuri\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

Out[108]:

```
array([[ 2.19267174],
       [-0.43483057],
       [ 0.47749662],
       ...,
       [ 0.40451044],
       [-0.39833748],
       [-0.58080292]])
```

In [109]:

```
print(train_quantity_standar.shape, y_train.shape)
print(test_quantity_standar.shape, y_test.shape)
print(cv_quantity_standar.shape, y_cv.shape)
```

```
(22445, 1) (22445,)
(16500, 1) (16500,)
(11055, 1) (11055,)
```

In [110]:

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

```
100%|██████████████████████████████████████████████████████████████████████████| 50000/50000
[00:01<00:00, 39997.95it/s]
```

In [111]:

```
#Introducing New Features
title_word_count = []
#for i in project_data['project_title']:
```

```
#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%|██████████████████████████████████████████████████████████████████████████| 50000/50000
[00:00<00:00, 431353.88it/s]
```

Standardize Title_word_count

In [112]:

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

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

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

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)
```

```
C:\Users\myuri\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

```
Mean : 5.211539318333704, Standard deviation : 27.402449685971902
```

```
C:\Users\myuri\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.

C:\Users\myuri\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

```
Mean : 5.199333333333334, Standard deviation : 27.402449685971902
```

```
C:\Users\myuri\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.

C:\Users\myuri\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

```
Mean : 5.24966078697422, Standard deviation : 27.402449685971902
```

```
C:\Users\myuri\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

```
(22445, 1) (22445,)
(16500, 1) (16500,)
(11055, 1) (11055,)
```

Standardize Title_word_count

In [113]:

```
essay_scalar = StandardScaler()

essay_scalar.fit(X_train['essay_word_count'].values.reshape(-1,1)) # finding the mean and standard deviation of this data
train_essay_word_count_standar = essay_scalar.transform(X_train['essay_word_count'].values.reshape(-1, 1))

essay_scalar.fit(X_train['essay_word_count'].values.reshape(-1,1)) # finding the mean and standard deviation of this data
test_essay_word_count_standar = essay_scalar.transform(X_test['essay_word_count'].values.reshape(-1, 1))

essay_scalar.fit(X_cv['essay_word_count'].values.reshape(-1,1)) # finding the mean and standard deviation of this data
cv_essay_word_count_standar = essay_scalar.transform(X_cv['essay_word_count'].values.reshape(-1, 1))

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)
```

```
C:\Users\myuri\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.

C:\Users\myuri\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.

C:\Users\myuri\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.

C:\Users\myuri\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.

C:\Users\myuri\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.

C:\Users\myuri\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

```
(22445, 1) (22445,)
(16500, 1) (16500,)
(11055, 1) (11055,)
```

Standardize POSITIVE

In [114]:

```
essay_scalar.fit(X_train['positive'].values.reshape(-1,1)) # finding the mean and standard deviation of this data
train_positive_standar = essay_scalar.transform(X_train['positive'].values.reshape(-1, 1))

essay_scalar.fit(X_train['positive'].values.reshape(-1,1)) # finding the mean and standard deviation of this data
```

```
test_positive_standar = essay_scalar.transform(X_test['positive'].values.reshape(-1, 1))

essay_scalar.fit(X_cv['positive'].values.reshape(-1,1)) # finding the mean and standard deviation
of this data
cv_positive_standar= essay_scalar.transform(X_cv['positive'].values.reshape(-1, 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)
```

```
(22445, 1) (22445,)
(16500, 1) (16500,)
(11055, 1) (11055,)
```

Standardize NEGATIVE

In [115]:

```
essay_scalar.fit(X_train['negitive'].values.reshape(-1,1)) # finding the mean and standard
deviation of this data
train_negative_standar = essay_scalar.transform(X_train['negitive'].values.reshape(-1, 1))

essay_scalar.fit(X_train['negitive'].values.reshape(-1,1)) # finding the mean and standard
deviation of this data
test_negative_standar = essay_scalar.transform(X_test['negitive'].values.reshape(-1, 1))

essay_scalar.fit(X_cv['negitive'].values.reshape(-1,1)) # finding the mean and standard deviation
of this data
cv_negative_standar = essay_scalar.transform(X_cv['negitive'].values.reshape(-1, 1))

print(train_negative_standar.shape, y_train.shape)
print(test_negative_standar.shape, y_test.shape)
print(cv_negative_standar.shape, y_cv.shape)
```

```
(22445, 1) (22445,)
(16500, 1) (16500,)
(11055, 1) (11055,)
```

Standardize neutral

In [116]:

```
essay_scalar.fit(X_train['neutral'].values.reshape(-1,1)) # finding the mean and standard
deviation of this data
train_neutral_standar = essay_scalar.transform(X_train['neutral'].values.reshape(-1, 1))

essay_scalar.fit(X_train['neutral'].values.reshape(-1,1)) # finding the mean and standard
deviation of this data
test_neutral_standar = essay_scalar.transform(X_test['neutral'].values.reshape(-1, 1))

essay_scalar.fit(X_cv['neutral'].values.reshape(-1,1)) # finding the mean and standard deviation o
f this data
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)
```

```
(22445, 1) (22445,)
(16500, 1) (16500,)
(11055, 1) (11055,)
```

Standardize compound

In [117]:

```
essay_scalar.fit(X_train['compound'].values.reshape(-1,1)) # finding the mean and standard
deviation of this data
train_compound_standar = essay_scalar.transform(X_train['compound'].values.reshape(-1, 1))

essay_scalar.fit(X_train['compound'].values.reshape(-1,1)) # finding the mean and standard
deviation of this data
test_compound_standar = essay_scalar.transform(X_test['compound'].values.reshape(-1, 1))

essay_scalar.fit(X_cv['compound'].values.reshape(-1,1)) # finding the mean and standard deviation
of this data
cv_compound_standar = essay_scalar.transform(X_cv['compound'].values.reshape(-1, 1))

print(train_compound_standar.shape, y_train.shape)
print(test_compound_standar.shape, y_test.shape)
print(cv_compound_standar.shape, y_cv.shape)
```

```
(22445, 1) (22445,)
(16500, 1) (16500,)
(11055, 1) (11055,)
```

1.5.4 Merging all the above features

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

In [118]:

```
# combine all the numerical  data together
```

In [119]:

```
#categrical data --category
print("Shape of train matrix after one hot encodig ",train_categories_one_hot.shape)
print("Shape of train matrix after one hot encodig ",cv_categories_one_hot.shape)
print("Shape of train matrix after one hot encodig ",test_categories_one_hot.shape)
```

```
Shape of train matrix after one hot encodig  (22445, 9)
Shape of train matrix after one hot encodig  (11055, 9)
Shape of train matrix after one hot encodig  (16500, 9)
```

In [120]:

```
#categrical data --subcategory
print("Shape of train matrix after one hot encodig ",train_subcategories_one_hot.shape)
print("Shape of train matrix after one hot encodig ",cv_subcategories_one_hot.shape)
print("Shape of train matrix after one hot encodig ",test_subcategories_one_hot.shape)
```

```
Shape of train matrix after one hot encodig  (22445, 30)
Shape of train matrix after one hot encodig  (11055, 30)
Shape of train matrix after one hot encodig  (16500, 30)
```

In [121]:

```
#category --state
print("Shape of train matrix after one hot encodig ",train_state_one_hot.shape)
print("Shape of train matrix after one hot encodig ",cv_state_one_hot.shape)
print("Shape of train matrix after one hot encodig ",test_state_one_hot.shape)
```

```
Shape of train matrix after one hot encodig  (22445, 51)
Shape of train matrix after one hot encodig  (11055, 51)
Shape of train matrix after one hot encodig  (16500, 51)
```

In [122]:

```
#category ----grade
print("Shape of train matrix after one hot encodig ",train_grade_one_hot.shape)
print("Shape of train matrix after one hot encodig ",cv_grade_one_hot.shape)
print("Shape of train matrix after one hot encodig ",test_grade_one_hot.shape)
```

```
Shape of train matrix after one hot encodig  (22445, 4)
Shape of train matrix after one hot encodig  (11055, 4)
Shape of train matrix after one hot encodig  (16500, 4)
```

In [123]:

```
#category ----teacher
print("Shape of train matrix after one hot encodig ",train_teacher_prefix_one_hot.shape)
print("Shape of train matrix after one hot encodig ",cv_teacher_prefix_one_hot.shape)
print("Shape of train matrix after one hot encodig ",test_teacher_prefix_one_hot.shape)
```

```
Shape of train matrix after one hot encodig  (22445, 5)
Shape of train matrix after one hot encodig  (11055, 5)
Shape of train matrix after one hot encodig  (16500, 5)
```

In [124]:

```
#bow essay
print("Shape of matrix after one hot encodig ",text_bow_essays_train.shape)
print("Shape of matrix after one hot encodig ",text_bow_essays_cv.shape)
print("Shape of matrix after one hot encodig ",text_bow_essays_test.shape)

#bow project title
print("Shape of matrix after one hot encodig ",text_bow_title_train.shape)
print("Shape of matrix after one hot encodig ",text_bow_title_cv.shape)
print("Shape of matrix after one hot encodig ",text_bow_title_test.shape)
```

```
Shape of matrix after one hot encodig  (22445, 30572)
Shape of matrix after one hot encodig  (11055, 30572)
Shape of matrix after one hot encodig  (16500, 30572)
Shape of matrix after one hot encodig  (22445, 8042)
Shape of matrix after one hot encodig  (11055, 8042)
Shape of matrix after one hot encodig  (16500, 8042)
```

In [125]:

```
#bow essay  tfidf
print("Shape of matrix after one hot encodig ",tfidf_essay_train.shape)
print("Shape of matrix after one hot encodig ",tfidf_essay_cv.shape)
print("Shape of matrix after one hot encodig ",tfidf_essay_test.shape)

#bow project title
print("Shape of matrix after one hot encodig ",text_tfidf_title_train.shape)
print("Shape of matrix after one hot encodig ",text_tfidf_title_cv.shape)
print("Shape of matrix after one hot encodig ",text_tfidf_title_test.shape)
```

```
Shape of matrix after one hot encodig  (22445, 30572)
Shape of matrix after one hot encodig  (11055, 30572)
Shape of matrix after one hot encodig  (16500, 30572)
Shape of matrix after one hot encodig  (22445, 8042)
Shape of matrix after one hot encodig  (11055, 8042)
Shape of matrix after one hot encodig  (16500, 8042)
```

Apply TruncatedSVD on TfidfVectorizer of essay text, choose the number of components (`n_components`)

In [143]:

```
from sklearn.decomposition import TruncatedSVD

svd=TruncatedSVD(algorithm='randomized', n_components=2, n_iter=7,random_state=42, tol=0.0)
Truncated_tfidf_essay_train=svd.fit_transform(tfidf_essay_train)
```

In [142]:

```
Truncated_tfidf_essay_cv=svd.fit_transform(tfidf_essay_cv)
```

In [141]:

```
Truncated_tfidf_essay_test=svd.fit_transform(tfidf_essay_test)
```

In [140]:

```
print(svd.explained_variance_ratio_)
print(svd.explained_variance_ratio_.sum())
print(svd.singular_values_)
```

```
[0.00413484 0.01009363]
0.014228471692014284
[36.38156785 12.40327304]
```

Assignment 7:SVM

[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)

The hyper paramter tuning (best alpha in range [10^-4 to 10^4], and the best penalty among 'l1', 'l2') Find the best hyper parameter which will give the maximum AUC value Find the best hyper paramter using k-fold cross validation or simple cross validation data Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

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

[Task-2] Apply the Support Vector Machines on these features by finding the best hyper paramter as suggested in step 2 and step 3 Consider these set of features Set 5 : school_state : categorical data clean_categories : categorical data clean_subcategories : categorical data project_grade_category :categorical data teacher_prefix : categorical data quantity : numerical data teacher_number_of_previously_posted_projects : numerical data price : numerical data sentiment score's of each of the essay : numerical data number of words in the title : numerical data number of words in the combine essays : numerical data Apply TruncatedSVD on TfidfVectorizer of essay text, choose the number of components (`n_components`) using elbow method : numerical data

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

Note: Data Leakage

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

Support Vector Machines

Set 1: categorical, numerical features + project_title(BOW) + preprocessed_eassay

In [127]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
```

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

In [164]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_train1=hstack(((train_categories_one_hot,train_subcategories_one_hot,train_state_one_hot,train_gr
ade_one_hot,
                 train_teacher_prefix_one_hot,text_bow_essays_train,text_bow_title_train,train_pri
ce_standar,
                 train_quantity_standar,train_prev_proj_standar))).tocsr()
X_train1.shape
```

Out[164]:

```
(22445, 38716)
```

In [165]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_cv1=hstack(((cv_categories_one_hot,cv_subcategories_one_hot,cv_state_one_hot,cv_grade_one_hot,
              cv_teacher_prefix_one_hot,text_bow_essays_cv,text_bow_title_cv,cv_price_standar,
                 cv_quantity_standar,cv_prev_proj_standar))).tocsr()
X_cv1.shape
```

Out[165]:

```
(11055, 38716)
```

In [166]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_test1=hstack(((test_categories_one_hot,test_subcategories_one_hot,test_state_one_hot,test_grade_c
ne_hot,
                 test_teacher_prefix_one_hot,text_bow_essays_test,text_bow_title_test,
                 test_price_standar,test_quantity_standar,test_prev_proj_standar
                ))).tocsr()
X_test1.shape
```

Out[166]:

```
(16500, 38716)
```

In [167]:

```
print(X_train1.shape,y_train.shape)
print(X_cv1.shape,y_cv.shape)
print(X_test1.shape,y_test.shape)
```

```
(22445, 38716) (22445,)
(11055, 38716) (11055,)
(16500, 38716) (16500,)
```

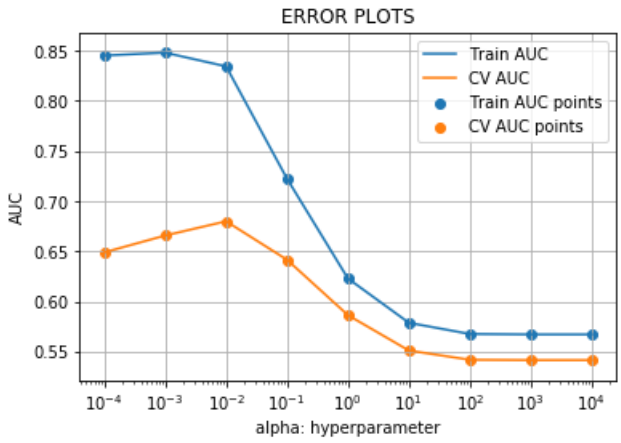
In [168]:

```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs

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

```
    # we will be predicting for the last data points
    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [169]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import SGDClassifier

train_auc = []
cv_auc = []
alpha= [10**-4, 10**-3, 10**-2, 10**-1, 1 , 10**1, 10**2, 10**3, 10**4]
for i in alpha:
    model = SGDClassifier(alpha=i,loss='hinge', penalty='l2',random_state=0)
    model.fit(X_train1, y_train)

    y_train_pred = model.decision_function(X_train1)
    y_cv_pred = model.decision_function(X_cv1)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

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

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

plt.legend()
plt.xscale('log')
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



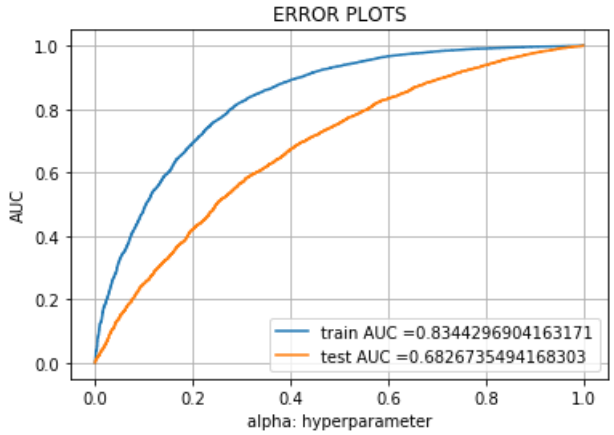
In [170]:

```
best_alpha=0.01
```

In [171]:

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

model=SGDClassifier(alpha=best_alpha,loss='hinge', penalty='l2',random_state=0)
model.fit(X_train1, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
```

```
y_train_pred = model.decision_function(X_train1)
y_test_pred = model.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("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [172]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):

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

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

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

In [173]:

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

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

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24999981238068972 for threshold 0.716
[[ 1733  1730]
 [ 1224 17758]]
```

In [174]:

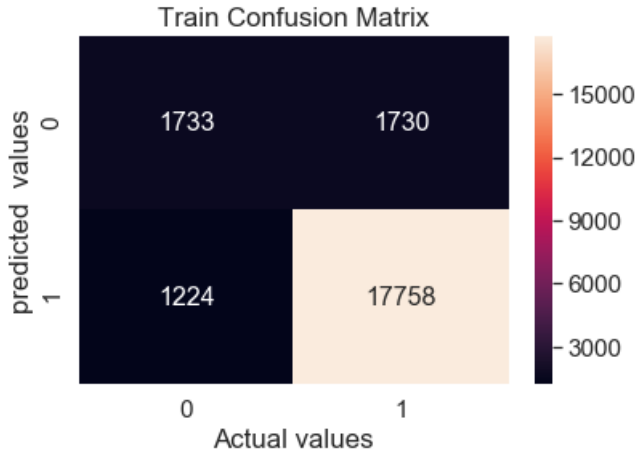
```
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred,
tr_thresholds, train_fpr, train_fpr)),
                                      range(2),range(2))
sns.set(font_scale=1.4) #for label size
```

```
sns.heatmap(train_confusion_matrix , annot = True, annot_kws={"size":16}, fmt = 'd')# font size
plt.xlabel('Actual values')
plt.ylabel('predicted  values')
plt.title('Train Confusion Matrix')
```

```
the maximum value of tpr*(1-fpr) 0.24999981238068972 for threshold 0.716
```

Out[174]:

```
Text(0.5, 1.0, 'Train Confusion Matrix')
```



In [175]:

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

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.86
[[ 1102  1444]
 [ 2664 11290]]
```

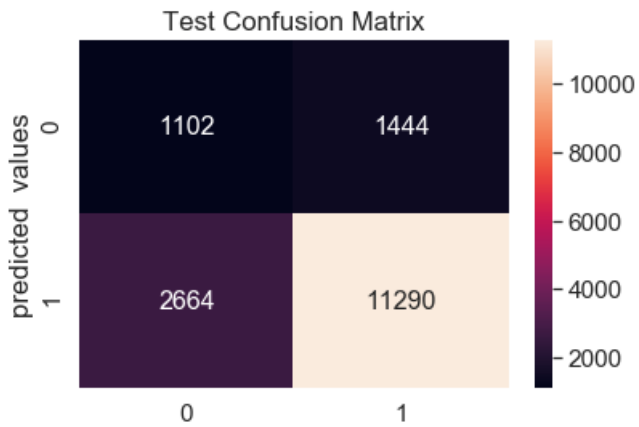
In [176]:

```
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds,
test_fpr, test_fpr)),
                                      range(2),range(2))
sns.set(font_scale=1.4) #for label size
sns.heatmap(train_confusion_matrix , annot = True, annot_kws={"size":16}, fmt = 'd')# font size
plt.xlabel('Actual values')
plt.ylabel('predicted  values')
plt.title('Test Confusion Matrix')
```

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.86
```

Out[176]:

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



Actual values

In [ ]:

```
# Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)
```

In [181]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_train2=hstack(((train_categories_one_hot,train_subcategories_one_hot,train_state_one_hot,train_gr
ade_one_hot,
                 train_teacher_prefix_one_hot,train_price_standar,train_quantity_standar,
                  train_prev_proj_standar
                ,tfidf_w2v_essay_train_data,tfidf_w2v_train_project_title))).tocsr()
X_train2.shape
```

Out[181]:

```
(22445, 702)
```

In [182]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_cv2=hstack(((cv_categories_one_hot,cv_subcategories_one_hot,cv_state_one_hot,cv_grade_one_hot,
cv_teacher_prefix_one_hot
               ,cv_price_standar,cv_quantity_standar,cv_prev_proj_standar,
               tfidf_w2v_essay_cv_data,tfidf_w2v_cv_project_title))).tocsr()
X_cv2.shape
```

Out[182]:

```
(11055, 702)
```

In [183]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_test2=hstack(((test_categories_one_hot,test_subcategories_one_hot,test_state_one_hot,test_grade_c
ne_hot,
                test_teacher_prefix_one_hot,
test_price_standar,test_quantity_standar,test_prev_proj_standar,
                 tfidf_w2v_essay_test_data,tfidf_w2v_test_project_title))).tocsr()
X_test2.shape
```

Out[183]:

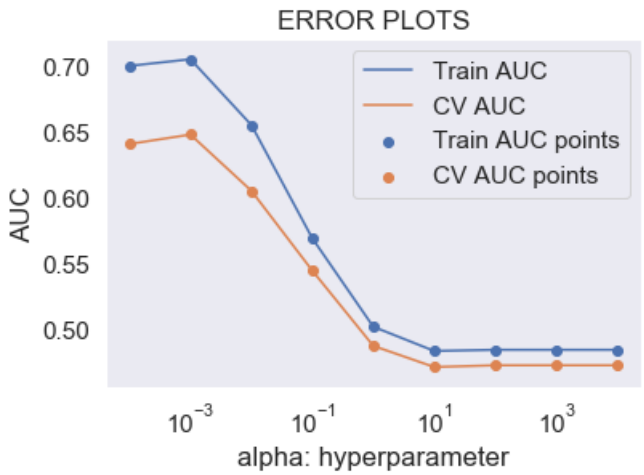
```
(16500, 702)
```

In [184]:

```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs

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

    return y_data_pred
```

In [185]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import SGDClassifier

train_auc = []
cv_auc = []
alpha= [10**-4, 10**-3, 10**-2, 10**-1, 1 , 10**1, 10**2, 10**3, 10**4]
for i in alpha:
    model = SGDClassifier(alpha=i,loss='hinge', penalty='l2',random_state=0)
    model.fit(X_train2, y_train)

    y_train_pred = model.decision_function(X_train2)
    y_cv_pred = model.decision_function(X_cv2)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

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

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

plt.legend()
plt.xscale('log')
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



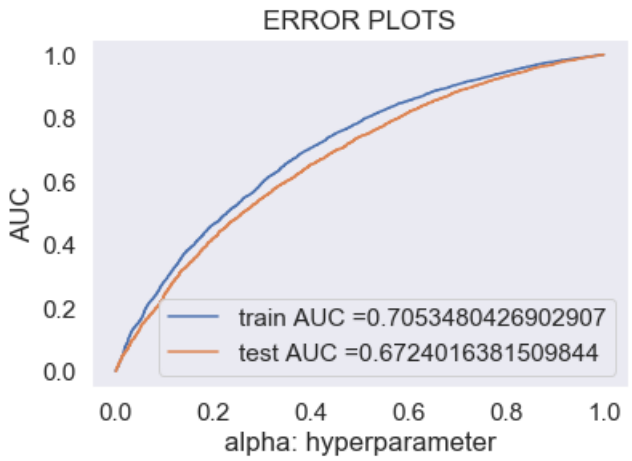
In [190]:

```
best_alpha=0.001
```

In [191]:

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

model=SGDClassifier(alpha=best_alpha,loss='hinge', penalty='l2',random_state=0)
model.fit(X_train2, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = model.decision_function(X_train2)
y_test_pred = model.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("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [192]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):

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

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

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

In [193]:

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

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

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24999997915341 for threshold 1.355
[[ 1732  1731]
 [ 4038 14944]]
```

In [194]:

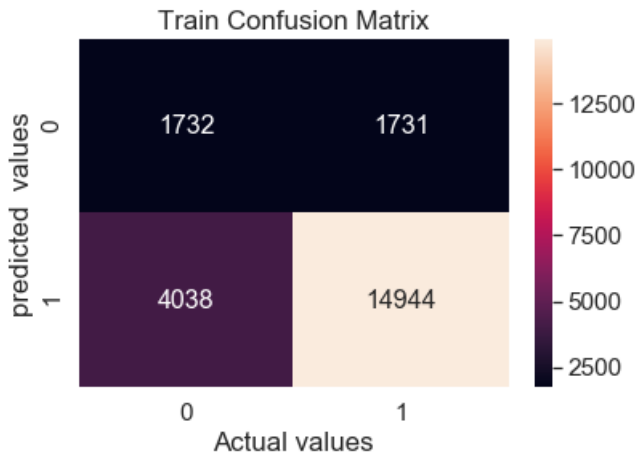
```
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred,
tr_thresholds, train_fpr, train_fpr)),
                                      range(2),range(2))
sns.set(font_scale=1.4) #for label size
sns.heatmap(train_confusion_matrix , annot = True, annot_kws={"size":16}, fmt = 'd')# font size
plt.xlabel('Actual values')
plt.ylabel('predicted  values')
```

```
plt.title('Train Confusion Matrix')
```

```
the maximum value of tpr*(1-fpr) 0.24999997915341 for threshold 1.355
```

Out[194]:

```
Text(0.5, 1.0, 'Train Confusion Matrix')
```



In [195]:

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

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 1.493
[[1446 1100]
 [4478 9476]]
```

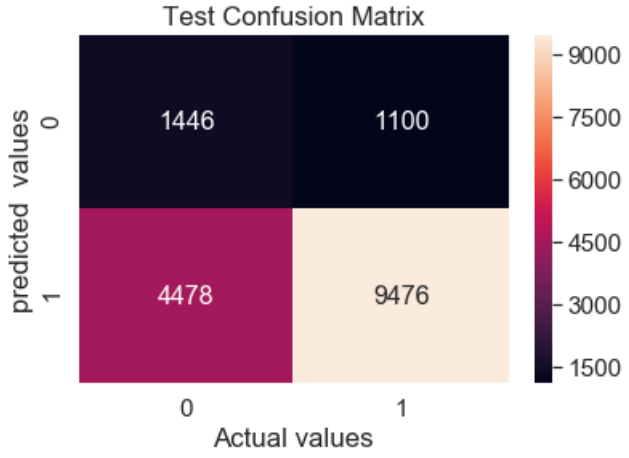
In [196]:

```
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds,
test_fpr, test_fpr)),
                                      range(2),range(2))
sns.set(font_scale=1.4) #for label size
sns.heatmap(train_confusion_matrix , annot = True, annot_kws={"size":16}, fmt = 'd')# font size
plt.xlabel('Actual values')
plt.ylabel('predicted  values')
plt.title('Test Confusion Matrix')
```

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 1.493
```

Out[196]:

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



Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)

In [197]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_train3=hstack(((train_categories_one_hot,train_subcategories_one_hot,train_state_one_hot,train_gr
ade_one_hot,
                 train_teacher_prefix_one_hot,train_price_standar,train_quantity_standar,
                 train_prev_proj_standar,
                 avg_w2v_essay_train_data,avg_w2v_project_title_train_data))).tocsr()
X_train3.shape
```

Out[197]:

```
(22445, 702)
```

In [198]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_cv3=hstack(((cv_categories_one_hot,cv_subcategories_one_hot,cv_state_one_hot,cv_grade_one_hot,cv_
teacher_prefix_one_hot,
               cv_price_standar,cv_quantity_standar,cv_prev_proj_standar,
               avg_w2v_essay_cv_data,avg_w2v_project_title_cv_data))).tocsr()
X_cv3.shape
```

Out[198]:

```
(11055, 702)
```

In [199]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_test3=hstack(((test_categories_one_hot,test_subcategories_one_hot,test_state_one_hot,test_grade_c
ne_hot,test_teacher_prefix_one_hot,
                 test_price_standar,test_quantity_standar,test_prev_proj_standar,
                 avg_w2v_essay_test_data,avg_w2v_project_title_test_data))).tocsr()
X_test3.shape
```

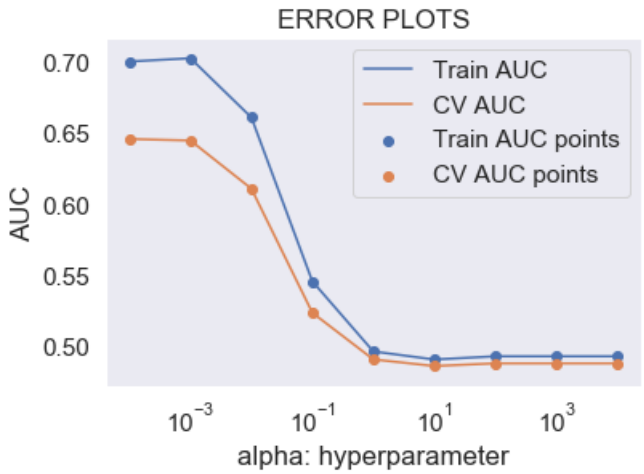
Out[199]:

```
(16500, 702)
```

In [200]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import SGDClassifier

train_auc = []
cv_auc = []
alpha= [10**-4, 10**-3, 10**-2, 10**-1, 1 , 10**1, 10**2, 10**3, 10**4]
for i in alpha:
    model = SGDClassifier(alpha=i,loss='hinge', penalty='l2',random_state=0)
    model.fit(X_train3, y_train)

    y_train_pred = model.decision_function(X_train3)
    y_cv_pred = model.decision_function(X_cv3)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(alpha, train_auc, label='Train AUC')
plt.plot(alpha, cv_auc, label='CV AUC')
```

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

plt.legend()
plt.xscale('log')
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

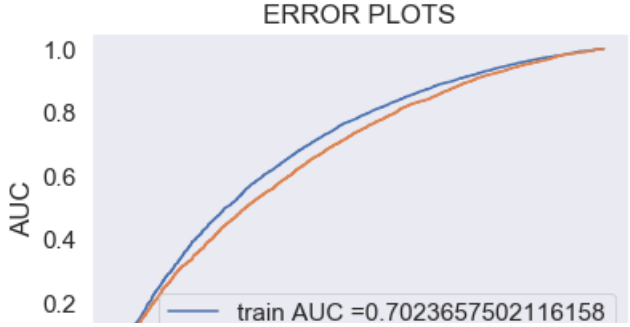


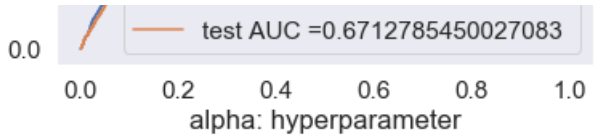
In [201]:

```
best_C =0.01
```

In [202]:

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

model=SGDClassifier(alpha=best_alpha,loss='hinge', penalty='l2',random_state=0)
model.fit(X_train3, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = model.decision_function(X_train3)
y_test_pred = model.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("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```





In [203]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):

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

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

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

In [204]:

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

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

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24999997915341 for threshold 1.275
[[ 1732  1731]
 [ 4068 14914]]
```

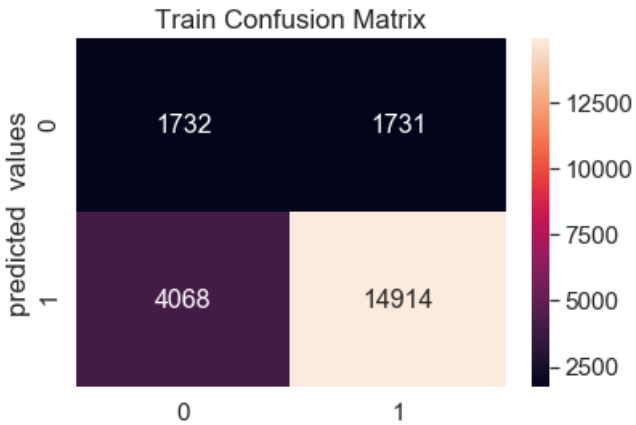
In [205]:

```
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred,
tr_thresholds, train_fpr, train_fpr)),
                                      range(2),range(2))
sns.set(font_scale=1.4) #for label size
sns.heatmap(train_confusion_matrix , annot = True, annot_kws={"size":16}, fmt = 'd')# font size
plt.xlabel('Actual values')
plt.ylabel('predicted  values')
plt.title('Train Confusion Matrix')
```

```
the maximum value of tpr*(1-fpr) 0.24999997915341 for threshold 1.275
```

Out[205]:

```
Text(0.5, 1.0, 'Train Confusion Matrix')
```



Actual values

In [206]:

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

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999984572938835 for threshold 1.403
[[1504 1042]
 [4749 9205]]
```

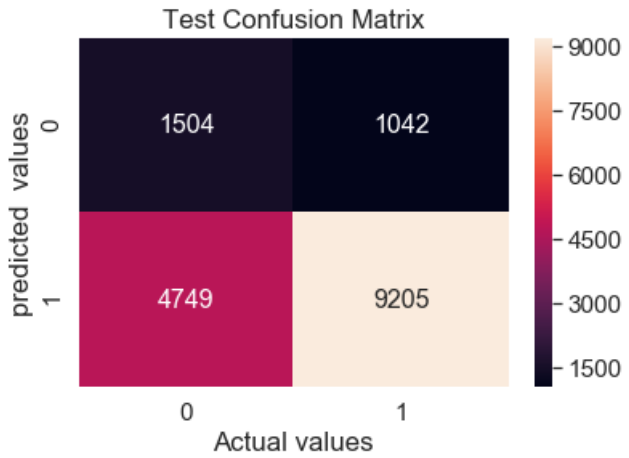
In [207]:

```
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds,
test_fpr, test_fpr)),
                                      range(2),range(2))
sns.set(font_scale=1.4) #for label size
sns.heatmap(train_confusion_matrix , annot = True, annot_kws={"size":16}, fmt = 'd')# font size
plt.xlabel('Actual values')
plt.ylabel('predicted  values')
plt.title('Test Confusion Matrix')
```

```
the maximum value of tpr*(1-fpr) 0.24999984572938835 for threshold 1.403
```

Out[207]:

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



Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)

In [208]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_train4=hstack(((train_categories_one_hot,train_subcategories_one_hot,train_state_one_hot,train_gr
ade_one_hot,
                  train_teacher_prefix_one_hot,train_price_standar,train_quantity_standar,
                 train_prev_proj_standar,
                 tfidf_w2v_essay_train_data,tfidf_w2v_train_project_title))).tocsr()
X_train4.shape
```

Out[208]:

```
(22445, 702)
```

In [209]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_cv4=hstack(((cv_categories_one_hot,cv_subcategories_one_hot,cv_state_one_hot,cv_grade_one_hot,
               cv_teacher_prefix_one_hot,cv_price_standar,cv_quantity_standar,
                  cv_prev_proj_standar,
               tfidf_w2v_essay_cv_data,tfidf_w2v_cv_project_title))).tocsr()
X_cv4.shape
```

Out[209]:

```
(11055, 702)
```

In [210]:

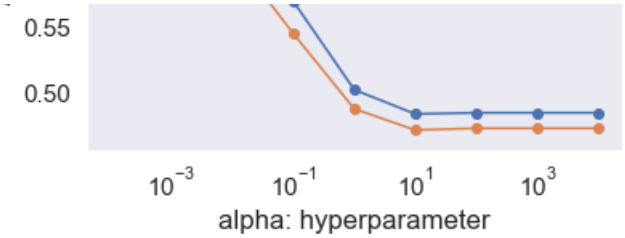
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_test4=hstack(((test_categories_one_hot,test_subcategories_one_hot,test_state_one_hot,test_grade_c
ne_hot,
                 test_teacher_prefix_one_hot,test_price_standar,test_quantity_standar,
                  test_prev_proj_standar,
                 tfidf_w2v_essay_test_data,tfidf_w2v_test_project_title))).tocsr()
X_test4.shape
```

Out[210]:

```
(16500, 702)
```

In [211]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import SGDClassifier

train_auc = []
cv_auc = []
alpha= [10**-4, 10**-3, 10**-2, 10**-1, 1 , 10**1, 10**2, 10**3, 10**4]
for i in alpha:
    model = SGDClassifier(alpha=i,loss='hinge', penalty='l2',random_state=0)
    model.fit(X_train4, y_train)

    y_train_pred = model.decision_function(X_train4)
    y_cv_pred = model.decision_function(X_cv4)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

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

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

plt.legend()
plt.xscale('log')
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

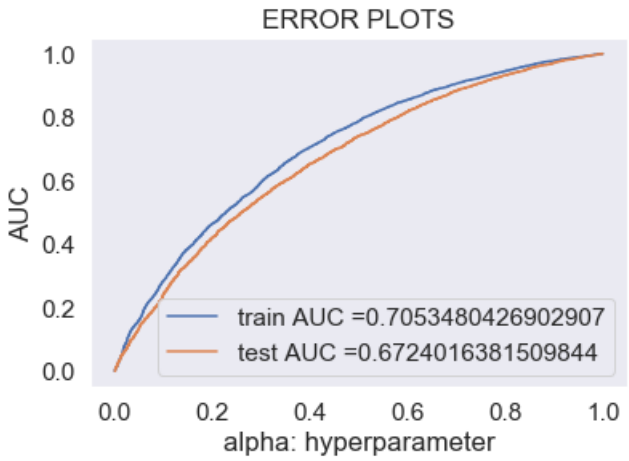


In [212]:

```
best_C =0.1
```

In [213]:

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

model=SGDClassifier(alpha=best_alpha,loss='hinge', penalty='l2',random_state=0)
model.fit(X_train4, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = model.decision_function(X_train4)
y_test_pred = model.decision_function(X_test4)

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("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [214]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):

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

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
```

```
        else:
            predictions.append(0)
    return predictions
```

In [215]:

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

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

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24999997915341 for threshold 1.355
[[ 1732  1731]
 [ 4038 14944]]
```

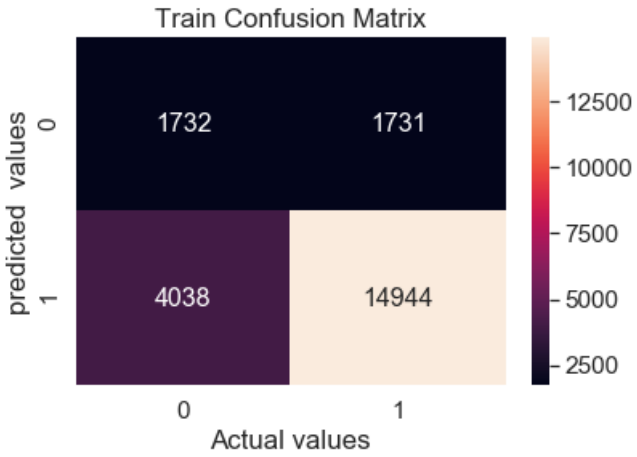
In [216]:

```
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred,
tr_thresholds, train_fpr, train_fpr)),
                                      range(2),range(2))
sns.set(font_scale=1.4) #for label size
sns.heatmap(train_confusion_matrix , annot = True, annot_kws={"size":16}, fmt = 'd')# font size
plt.xlabel('Actual values')
plt.ylabel('predicted  values')
plt.title('Train Confusion Matrix')
```

```
the maximum value of tpr*(1-fpr) 0.24999997915341 for threshold 1.355
```

Out[216]:

```
Text(0.5, 1.0, 'Train Confusion Matrix')
```



In [217]:

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

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 1.493
[[1446 1100]
 [4478 9476]]
```

In [218]:

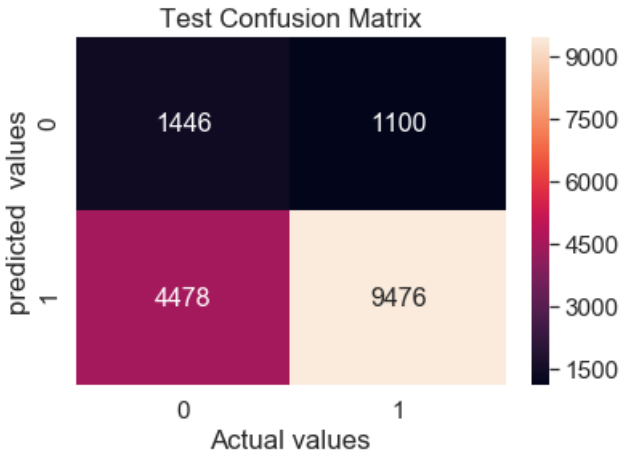
```
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds,
test_fpr, test_fpr)),
                                      range(2),range(2))
sns.set(font_scale=1.4) #for label size
sns.heatmap(train_confusion_matrix , annot = True, annot_kws={"size":16}, fmt = 'd')# font size
```

```
sns.heatmap(train_confusion_matrix , annot = True, annot_kws={"size":16}, fmt = "d");# font size
plt.xlabel('Actual values')
plt.ylabel('predicted  values')
plt.title('Test Confusion Matrix')
```

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 1.493
```

Out[218]:

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



Apply Logistic Regression on the below feature set Set 5 by finding the best hyper parameter

In [144]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_train5=
hstack((train_categories_one_hot,train_subcategories_one_hot,train_state_one_hot,train_grade_one_hc
t,
train_teacher_prefix_one_hot, train_quantity_standar, train_prev_proj_standar, train_price_standar
,train_positive_standar,
train_negative_standar,  train_neutral_standar,train_compound_standar,
train_title_word_count_standar,
train_essay_word_count_standar,Truncated_tfidf_essay_train)).tocsr()
print(X_train5.shape, y_train.shape)
print(type(X_train5))
```

```
(22445, 110) (22445,)
<class 'scipy.sparse.csr.csr_matrix'>
```

In [150]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_cv5=hstack((cv_categories_one_hot,cv_subcategories_one_hot,cv_state_one_hot,cv_grade_one_hot,
cv_teacher_prefix_one_hot, cv_quantity_standar, cv_prev_proj_standar, cv_price_standar,
cv_positive_standar,cv_negative_standar, cv_neutral_standar,cv_compound_standar,
cv_title_word_count_standar, cv_essay_word_count_standar,Truncated_tfidf_essay_cv)).tocsr()
print(X_cv5.shape,y_cv.shape)
print(type(X_cv5))
```

```
(11055, 110) (11055,)
<class 'scipy.sparse.csr.csr_matrix'>
```

In [151]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_test5=hstack((test_categories_one_hot,test_subcategories_one_hot,test_state_one_hot,test_grade_or
```

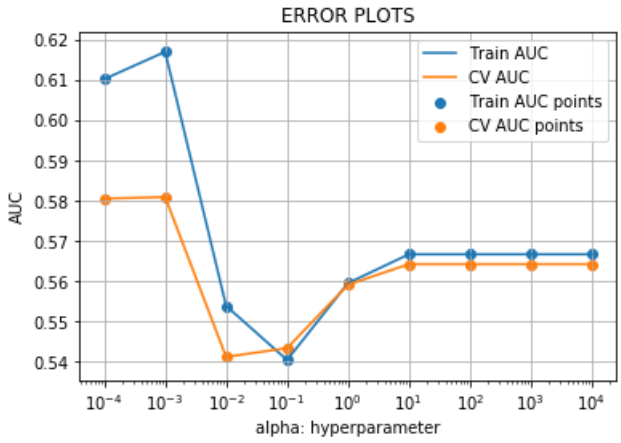
```
e_hot,
test_teacher_prefix_one_hot, test_quantity_standar, test_prev_proj_standar, test_price_standar,tes
t_positive_standar,
test_negative_standar,  test_neutral_standar,test_compound_standar, test_title_word_count_standar,

test_essay_word_count_standar,Truncated_tfidf_essay_test)).tocsr()
print(X_test5.shape,y_test.shape)
print(type(X_test5))
```

```
(16500, 110) (16500,)
<class 'scipy.sparse.csr.csr_matrix'>
```

Hyperparameter Tunning

In [152]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import SGDClassifier

train_auc = []
cv_auc = []
alpha= [10**-4, 10**-3, 10**-2, 10**-1, 1 , 10**1, 10**2, 10**3, 10**4]
for i in alpha:
    model = SGDClassifier(alpha=i,loss='hinge', penalty='l2',random_state=0)
    model.fit(X_train5, y_train)

    y_train_pred = model.decision_function(X_train5)
    y_cv_pred = model.decision_function(X_cv5)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

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

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

plt.legend()
plt.xscale('log')
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

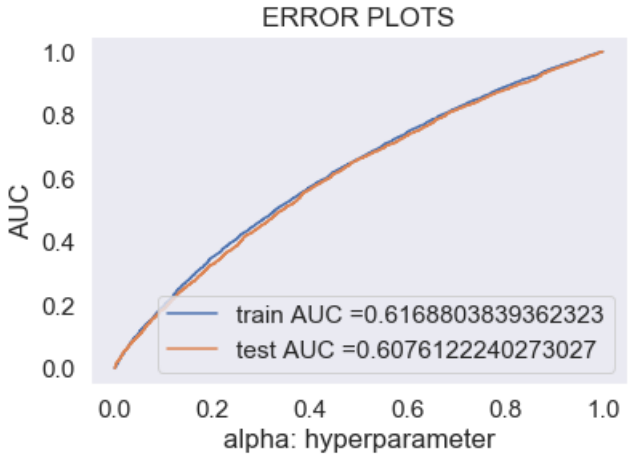


In [231]:

```
best_alpha =0.001
```

In [232]:

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

model=SGDClassifier(alpha=best_alpha,loss='hinge', penalty='l2',random_state=0)
model.fit(X_train5, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = model.decision_function(X_train5)
y_test_pred = model.decision_function(X_test5)

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("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



Train Confusion Matrix

In [223]:

```
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_fpr)))
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24999997915341 for threshold 1.034
[[ 1732  1731]
 [ 6417 12565]]
```

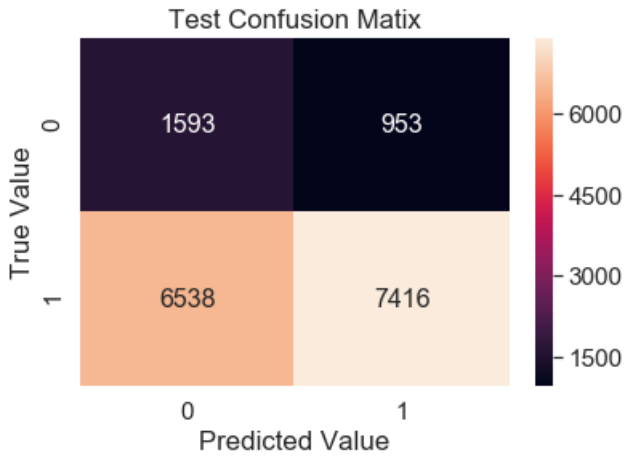
In [224]:

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

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 1.058
```

Out[224]:

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



Test Confusion Matrix

In [225]:

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

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 1.058
[[1593  953]
 [6538 7416]]
```

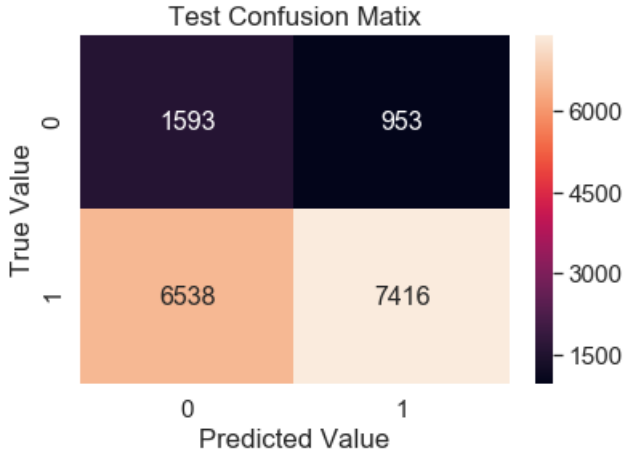
In [226]:

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

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 1.058
```

Out[226]:

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



2.2 Make Data Model Ready: encoding numerical, categorical features

In [ ]:

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

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

2.3 Make Data Model Ready: encoding eassay, and project_title

In [ ]:

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

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

2.4 Appling Logistic Regression on different kind of featurization as mentioned in the instructions

In [ ]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

2.5 Logistic Regression with added Features `Set 5

In [ ]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

Summary

In [234]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer","Model"," hyperParameter","AUC"]
x.add_row(["BOW","Auto","0.01","0.83"])
x.add_row(["TFIDF","Auto","0.001","0.70"])
x.add_row(["AVGW2V","Auto","0.01","0.70"])
x.add_row(["TFIF-2V","Auto","0.1","0.70"])
x.add_row(["SET5","Auto","0.001","0.61"])
print(x)
```

```
+------------+-------+-----------------+------+
| Vectorizer | Model |  hyperParameter | AUC  |
+------------+-------+-----------------+------+
|    BOW     |  Auto |       0.01      | 0.83 |
|   TFIDF    |  Auto |      0.001      | 0.70 |
|   AVGW2V   |  Auto |       0.01      | 0.70 |
|  TFIF-2V   |  Auto |       0.1       | 0.70 |
|    SET5    |  Auto |      0.001      | 0.61 |
+------------+-------+-----------------+------+
```

Observations

Accuracy has been decrased as we have reduced the dimesions of text features using truncatedsvd.

SVM Classifiers offer good accuracy and perform faster prediction compared to Naïve Bayes algorithm. They also use less memory because they use a subset of training points in the decision phase. SVM works well with a clear margin of separation and with high dimensional space.