

DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

| Feature | |
|-------------------------------------|---|
| <code>project_id</code> | A unique identifier for the proposed project |
| <code>project_title</code> | Title of the project |
| <code>project_grade_category</code> | Grade level of students for which the project is targeted |
| <code>teacher_prefix</code> | Prefix of the teacher's name |
| <code>teacher_email</code> | Email address of the teacher |
| <code>teacher_first_name</code> | First name of the teacher |
| <code>teacher_last_name</code> | Last name of the teacher |
| <code>teacher_state</code> | State where the teacher is located |
| <code>project_submittedлонг</code> | Date the project was submitted |
| <code>project_is_approved</code> | Whether the project was approved |

| Feature | |
|--|---|
| | One or more (comma-separated) subject categories following enum |
| project_subject_categories | <ul style="list-style-type: none"> • • • • • • • • |
| | Literacy & Language |
| school_state | <p>State where school is located (Two-letter state abbreviations)</p> |
| | One or more (comma-separated) subject subcategories |
| project_subject_subcategories | <ul style="list-style-type: none"> • • |
| | Literature & Writing |
| project_resource_summary | <p>An explanation of the resources needed for the project</p> <ul style="list-style-type: none"> • My students need hands on literacy materials. |
| project_essay_1 | F |
| project_essay_2 | Second |
| project_essay_3 | Third |
| project_essay_4 | Fourth |
| project_submitted_datetime | Datetime when project application was submitted. Example: 2018-01-12T12:00:00Z |
| teacher_id | A unique identifier for the teacher of the proposed project: bdf8baa8fedef6b |
| | Teacher's title. One of the following: |
| teacher_prefix | <ul style="list-style-type: none"> • • • • • |
| teacher_number_of_previously_posted_projects | Number of project applications previously submitted by the teacher. |

* 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 | Description 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 |
|----------------------------------|---|
| <code>project_is_approved</code> | A binary flag indicating whether DonorsChoose approved the project. A value of <code>0</code> indicates the project was not approved, and a value of <code>1</code> indicates the project was approved. |

Notes on the Essay Data

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

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

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

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

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

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

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

```
In [4]: %matplotlib inline
import warnings
warnings.filterwarnings("ignore")

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

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

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

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

from tqdm import tqdm
import os

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

1.1 Reading Data

```
In [5]: project_data = pd.read_csv('train_data.csv')
resource_data = pd.read_csv('resources.csv')
```

```
In [6]: print("Number of data points in train data", project_data.shape)
print('*'*50)
print("The attributes of data :", project_data.columns.values)
```

Number of data points in train data (109248, 17)

The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' '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 [7]: print("Number of data points in train data", resource_data.shape)
print(resource_data.columns.values)
resource_data.head(2)
```

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

Out[7]:

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

1.2 preprocessing of project_subject_categories

```
In [8]: categories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-list-of-strings-in-python

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-list-of-strings-in-python
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
cat_list = []
for i in categories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','):# it will split it in three parts ["Math & Science", "Warmth", "Care & Hunger"]
        if 'The' in j.split():# this will split each of the category based on space
            j=j.replace('The','') # if we have the words "The" we are going to remove it
            j = j.replace(' ', '') # we are placing all the ' ' (space) with '' (empty)
            temp+=j.strip()+" "# abc ".strip() will return "abc", remove the trailing space
            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 [9]: sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-list-in-python/23669025#23669025

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-list-in-python/23669025#23669025
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python/8270092#8270092

sub_cat_list = []
for i in sub_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','):# it will split it in three parts ["Math & Science", "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the category based on space
            j=j.replace('The','') # if we have the words "The" we are going to remove it
            j = j.replace(' ','') # we are placing all the ' '(space) with ''(empty)
            temp +=j.strip()+" "# abc .strip() will return "abc", remove the trailing space
            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/22898595/11459000
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]))
```

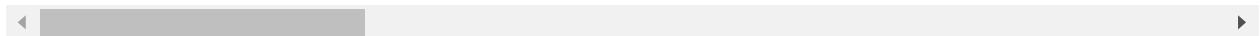
1.3 Text preprocessing

```
In [10]: # 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 [11]: `project_data.head(2)`

Out[11]:

| | Unnamed: 0 | id | | teacher_id | teacher_prefix | school_state | project_sul |
|---|---------------|---------|----------------------------------|------------|----------------|--------------|-------------|
| 0 | 160221 | p253737 | c90749f5d961ff158d4b4d1e7dc665fc | | Mrs. | IN | 20 |
| 1 | 140945 | p258326 | 897464ce9ddc600bcfd1151f324dd63a | | Mr. | FL | 20 |



```
In [12]: # printing some random reviews
print(project_data['essay'].values[0])
print("=*50)
print(project_data['essay'].values[150])
print("=*50)
print(project_data['essay'].values[1000])
print("=*50)
print(project_data['essay'].values[20000])
print("=*50)
print(project_data['essay'].values[99999])
print("=*50)
```

My students are English learners that are working on English as their second or third languages. We are a melting pot of refugees, immigrants, and native-born Americans bringing the gift of language to our school. \r\n\r\n We have over 24 languages represented in our English Learner program with students at every level of mastery. We also have over 40 countries represented with the families within our school. Each student brings a wealth of knowledge and experiences to us that open our eyes to new cultures, beliefs, and respect.\\"The limits of your language are the limits of your world.\\"-Ludwig Wittgenstein Our English learners have a strong support system at home that begs for more resources. Many times our parents are learning to read and speak English along side of their children. Sometimes this creates barriers for parents to be able to help their child learn phonetics, letter recognition, and other reading skills.\r\n\r\nBy providing these dvd's and players, students are able to continue their mastery of the English language even if no one at home is able to assist. All families with students within the Level 1 proficiency status, will be offered to be a part of this program. These educational videos will be specially chosen by the English Learner Teacher and will be sent home regularly to watch. The videos are to help the child develop early reading skills.\r\n\r\nParents that do not have access to a dvd player will have the opportunity to check out a dvd player to use for the year. The plan is to use these videos and educational dvd's for the years to come for other EL students.\r\n\r\n=====

The 51 fifth grade students that will cycle through my classroom this year all love learning, at least most of the time. At our school, 97.3% of the students receive free or reduced price lunch. Of the 560 students, 97.3% are minority students. \r\nThe school has a vibrant community that loves to get together and celebrate. Around Halloween there is a whole school parade to show off the beautiful costumes that students wear. On Cinco de Mayo we put on a big festival with crafts made by the students, dances, and games. At the end of the year the school hosts a carnival to celebrate the hard work put in during the school year, with a dunk tank being the most popular activity. My students will use these five brightly colored Hokki stools in place of regular, stationary, 4-legged chairs. As I will only have a total of ten in the classroom and not enough for each student to have an individual one, they will be used in a variety of ways. During independent reading time they will be used as special chairs students will each use on occasion. I will utilize them in place of chairs at my small group tables during math and reading times. The rest of the day they will be used by the students who need the highest amount of movement in their life in order to stay focused on school.\r\n\r\nWhenever asked what the classroom is missing, my students always say more Hokki Stools. They can't get their fill of the 5 stools we already have. When the students are sitting in group with me on the Hokki Stools, they are always moving, but at the same time doing their work. Anytime the students get to pick where they can sit, the Hokki Stools are the first to be taken. There are always students who head over to the kidney table to get one of the stools who are disappointed as there are not enough of them. \r\n\r\n

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

=====

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

=====

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

=====

The mediocre teacher tells. The good teacher explains. The superior teacher demonstrates. The great teacher inspires. -William A. Ward\r\n\r\nMy school has 803 students which is makeup is 97.6% African-American, making up the largest segment of the student body. A typical school in Dallas is made up of 23.2% African-American students. Most of the students are on free or reduced lunch. We aren't receiving doctors, lawyers, or engineers children from rich backgrounds or neighborhoods. As an educator I am inspiring minds of young children and we foc

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

Introducing New Features

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

Consider these set of features for Set 5 in Assignment:

categorical data school_state

clean_categories....clean_subcategories....project_grade_category....teacher_prefix

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

New Features

sentiment score's of each of the essay : **numerical data**

number of words in the title : **numerical data**

number of words in the combine essays : **numerical data**

```
In [15]: new_title = []
for i in tqdm(project_data['project_title']):
    j = decontracted(i)
    new_title.append(j)
```

100% |██████████| 1
09248/109248 [00:01<00:00, 72600.59it/s]

```
In [16]: #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% |██████████| 10
9248/109248 [00:00<00:00, 581198.63it/s]

```
In [17]: project_data.head(2)
```

Out[17]:

| | Unnamed: 0 | id | teacher_id | teacher_prefix | school_state | project_sul |
|---|---------------|---------|----------------------------------|----------------|--------------|-------------|
| 0 | 160221 | p253737 | c90749f5d961ff158d4b4d1e7dc665fc | Mrs. | IN | 20 |
| 1 | 140945 | p258326 | 897464ce9ddc600bcfd1151f324dd63a | Mr. | FL | 20 |

```
In [18]: new_essay = []
for i in tqdm(project_data['essay']):
    j = decontracted(i)
    new_essay.append(j)
```

100% |██████████| 1
09248/109248 [00:02<00:00, 40170.99it/s]

```
In [19]: essay_word_count = []
for i in tqdm(new_essay):
    j = len(i.split())
    essay_word_count.append(j)
    #print(j)
project_data['essay_word_count'] = essay_word_count
```

100% |██████████| 1
09248/109248 [00:02<00:00, 36864.07it/s]

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

Out[20]:

| | Unnamed: 0 | id | | teacher_id | teacher_prefix | school_state | project_sul |
|---|---------------|---------|----------------------------------|------------|----------------|--------------|-------------|
| 0 | 160221 | p253737 | c90749f5d961ff158d4b4d1e7dc665fc | | Mrs. | IN | 20 |
| 1 | 140945 | p258326 | 897464ce9ddc600bcfd1151f324dd63a | | Mr. | FL | 20 |

Computing Sentiment Scores

In [21]:

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

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

sid = SentimentIntensityAnalyzer()

for_sentiment = 'a person is a person no matter how small dr seuss i teach the ss = sid.polarity_scores(for_sentiment)

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

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

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

```
In [22]: SID = SentimentIntensityAnalyzer()
#There is NEGITIVE and POSITIVE and NEUTRAL and COMPOUND SCORES
#http://www.nltk.org/howto/sentiment.html

negative = []
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']
    negative.append(j)
    positive.append(k)
    neutral.append(l)
    compound.append(m)
```

100% |  | 109248/109248 [31:17<00:00, 58.20it/s]

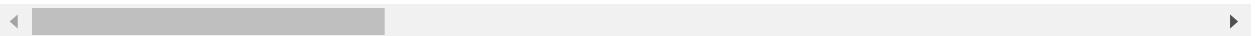
```
In [23]: project_data['negative'] = negative
project_data['positive'] = positive
project_data['neutral'] = neutral
project_data['compound'] = compound
```

```
In [24]: project_data.head(2)
```

Out[24]:

| | Unnamed: 0 | id | teacher_id | teacher_prefix | school_state | project_sul |
|---|---------------|---------|----------------------------------|----------------|--------------|-------------|
| 0 | 160221 | p253737 | c90749f5d961ff158d4b4d1e7dc665fc | Mrs. | IN | 20 |
| 1 | 140945 | p258326 | 897464ce9ddc600bcfd1151f324dd63a | Mr. | FL | 20 |

2 rows × 24 columns



1.4 Train_Test_Split

```
In [25]: #Train Test Split
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(project_data, project_data[""
    test_size = 0.33, stratify =
    random_state = 42)
```

```
In [26]: #Train CV Split
x_train, x_cv, y_train, y_cv = train_test_split(x_train, y_train, test_size = 0.33,
    random_state = 42)
```

```
In [27]: print(x_test.columns)
print(x_train.columns)
#print(x_cv.columns)
#print(x_train.shape)
#print(x_test.shape)
#print(x_cv.shape)
```

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

```
In [28]: #Dropping Class Label in train test and cv data
x_train.drop(["project_is_approved"], axis = 1, inplace = True)
x_test.drop(["project_is_approved"], axis = 1, inplace = True)
x_cv.drop(["project_is_approved"], axis = 1, inplace = True)
```

```
In [29]: print(x_train.columns)

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

```
In [ ]:
```

Preparing Data for Models

```
In [30]: # 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 [31]: sent = decontracted(project_data['essay'].values[20000])
print(sent)
print("*50)
```

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

=====

```
In [32]: # \r \n \t remove from string python: http://texthandler.com/info/remove-Line-br
sent = sent.replace('\r', ' ')
sent = sent.replace('\n', ' ')
sent = sent.replace('\t', ' ')
print(sent)
```

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

```
In [33]: #remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
print(sent)
```

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

```
In [34]: # https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you',
    "you'll", "you'd", "your", 'yours', 'yourself', 'yourselves', 'he',
    'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itse',
    'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'tha',
    'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'ha',
    'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because
    'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 't
    'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'of
    'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all
    'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than
    's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've
    've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "d
    "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma
    "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn
    'won', "won't", 'wouldn', "wouldn't"]
```

Project_Essays preprocessing

```
In [35]: #train_preprocessed_essays
# Combining all the above students
from tqdm import tqdm
train_preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(x_train['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\'', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    train_preprocessed_essays.append(sent.lower().strip())
```

100% |
| 49041/49041 [00:44<00:00, 1113.45it/s]

In [36]: `train_preprocessed_essays[10]`

Out[36]: 'keeping visual arts classrooms passion i want share classroom in room add means appliance door decor providing quality supplies foster appreciation desire pursue art career i work title one school suburban neighborhood we provide many magnet programs including pbi positive behavior intervention preschool el err i worked many students potential become artists i would like provide experiences quality materials uniform colors brushes not shed bristles chance paint real canvas i hardly wait share opportunity paint room 30 students not opportunity mix colors create real canvas we start outdoor landscapes sky water reflection silhouette trees begin three colors red white black mix need provide contrast high lights 3 colors we cover vocabulary words tints shades hue line shape foreground mid ground background practice plain construction paper create final product canvas we begin implementation performance based art report card next year to provide instruction color mixing techniques students could realize final project real canvas would positive impact students may grow become design engineers architects fashion designers tomorrow'

In [37]: `# test_preprocessed_essay`

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

100% |
| 36052/36052 [00:31<00:00, 1150.16it/s]

In [38]: `test_preprocessed_essays[10]`

Out[38]: 'hey i astronaut look says one kindergarten students no look i engineer says an other another 5 year old shouts i love technology want computer person i grow t his life eyes 5 6 year old public school boys girls they love learning explorin g pretending want grow the children come variety ethnic social cultural economi c backgrounds but children one thing common love learning especially math scienc e our motto grown anything want learning anything possible just witnessing desire learn math science rewarding my students curious world need know technology help identify real world become scientist math teacher engineer my students love hands activities motivated technology they love learning new things enthusias m radiates class this ipad case make huge difference way students learning math science it let customize learning according students levels level level level i t allow children opportunity understand real world live the children use free s cience app brain pop jr see scientist work lab astronauts go outer space develo p computer programs help keep earth free pollution they use free math app sushi monster strengthen reasoning skills learn strategies solving math problems unde rstanding patterns world this ipad allow show students stem activites used diff erentiate instruction students bring real world classroom they learn work addit ion math problems 5 3 2 10 seeing robot build tower learn planet mars water see ing pictures form hubble telescope learn ipad show patterns exist throughout wo rld this ipad used kindergarten math science exploration center the children pi ck ipad go free stem app like brain pop jr using fingers activate game stimulat e enrich learning math science nannan'

In [40]: `# CV_preprocessed_essays`

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

100% |██████████| 24155/24155 [00:20<00:00, 1206.97it/s]

In [41]: cv_preprocessed_essays[10]

Out[41]: 'when asked music means student told music makes feel happy it changed i not anything express feelings i this hope every student joins class my biggest challenge providing everything need learn i blessed able work wonderful students they hunger music insatiable each day i able go work inspired dedication hard work students put everything our school made mission build culture embraces musical learning give every student opportunity participate music our school low income area best provide much needed supplies possible music program keeps growing struggling keep there nothing worse hearing violins played tune it difficult beginning violin students tune strings not always hear differences sound this means teacher often ends tuning instruments taking valuable learning time if students pitch pipes would able play individual notes match violin strings sounds this means could also fix violins get tune practicing home resulting better practice improvement individual playing nannan'

Project_Titles_preprocessing

In [42]:

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

100% |██████████|
49041/49041 [00:01<00:00, 26159.29it/s]

In [43]: train_preprocessed_titles[10]

Out[43]: 'we all have rembrandt inside us'

```
In [44]: # Test_preprocessed_essays
from tqdm import tqdm
test_preprocessed_titles = []
# tqdm is for printing the status bar
for sentance in tqdm(x_test['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\'', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    test_preprocessed_titles.append(sent.lower().strip())
```

100% |
36052/36052 [00:01<00:00, 25590.95it/s]

```
In [45]: test_preprocessed_titles[10]
```

```
Out[45]: 'ipads ipads exploring'
```

```
In [46]: # CV_preprocessed_titles
from tqdm import tqdm
cv_preprocessed_titles = []
# tqdm is for printing the status bar
for sentance in tqdm(x_cv['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\'', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    cv_preprocessed_titles.append(sent.lower().strip())
```

100% |
24155/24155 [00:00<00:00, 24476.51it/s]

```
In [47]: # after preprocesing
cv_preprocessed_titles[10]
```

```
Out[47]: 'help us play our violins in tune'
```

```
In [48]: project_data.columns
```

```
Out[48]: Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category', 'project_title',
       'project_essay_1', 'project_essay_2', 'project_essay_3',
       'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'essay', 'title_word_count',
       'essay_word_count', 'negative', 'positive', 'neutral', 'compound'],
      dtype='object')
```

we are going to consider

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

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

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

1.5.1 Vectorizing Categorical data

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

```
In [49]: # we use count vectorizer to convert the values into one
# Vectorizing Clean Categories
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=True)

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

print(vectorizer.get_feature_names())
print("Shape of Train matrix after one hot encoding ",train_categories_one_hot.shape)
print("Shape of Test matrix after one hot encoding ",test_categories_one_hot.shape)
print("Shape of cv matrix after one hot encoding ",cv_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 encoding (49041, 9)
Shape of Test matrix after one hot encoding (36052, 9)
Shape of cv matrix after one hot encoding (24155, 9)
```

```
In [50]: # we use count vectorizer to convert the values into one
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=True)
vectorizer.fit(x_train["clean_subcategories"].values)
train_sub_categories_one_hot = vectorizer.transform(x_train['clean_subcategories'])
test_sub_categories_one_hot = vectorizer.transform(x_test['clean_subcategories'])
cv_sub_categories_one_hot = vectorizer.transform(x_cv['clean_subcategories'])

print(vectorizer.get_feature_names())
print("Shape of Train matrix after Trainone hot encodig ",train_sub_categories_one_hot.shape)
print("Shape of test matrix after one hot encodig ",test_sub_categories_one_hot.shape)
print("Shape of cv_ matrix after one hot encodig ",cv_sub_categories_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 Trainone hot encodig (49041, 30)
Shape of test matrix after one hot encodig (36052, 30)
Shape of cv_ matrix after one hot encodig (24155, 30)
```

```
In [51]: # you can do the similar thing with state, teacher_prefix and project_grade_categories
from collections import Counter
my_counter = Counter()
for word in project_data["school_state"].values:
    my_counter.update(word.split())
```

```
In [52]: # dict sort by value python: https://stackoverflow.com/a/613218/4084039
state_cat_dict = dict(my_counter)
storted_state_cat_dict = dict(sorted(state_cat_dict.items(), key=lambda kv: kv[1]))
```

```
In [53]: # Please do the similar feature encoding with state, teacher_prefix and project_grade_categories
#Using Count Vectorizer to convert the state value onto on hot encoded feature
vectorizer = CountVectorizer(vocabulary=list(storted_state_cat_dict.keys()), lowercase=True)
vectorizer.fit(project_data['school_state'].values)
print(vectorizer.get_feature_names())

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

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

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

```
In [54]: #https://stackoverflow.com/questions/42224700/attributeerror-float-object-has-no
project_data['project_grade_category']=project_data['project_grade_category'].fillna(" ")
my_counter = Counter()
for word in project_data['project_grade_category'].values:
    my_counter.update(word.split())
```

```
In [55]: project_cat_dict = dict(my_counter)
sorted_project_cat_dict = dict(sorted(project_cat_dict.items(), key=lambda kv: kv[1], reverse=True))
```

```
In [56]: # feature encoding for project_grade_category also
vectorizer = CountVectorizer(vocabulary=list(sorted_project_cat_dict.keys()), lowercase=False)
vectorizer.fit(project_data['project_grade_category'].values)
print(vectorizer.get_feature_names())

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

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

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

```
In [57]: #https://stackoverflow.com/questions/42224700/attributeerror-float-object-has-no
project_data['teacher_prefix']=project_data['teacher_prefix'].fillna(" ")
```

```
In [58]: my_counter = Counter()
for word in project_data['teacher_prefix'].values:
    my_counter.update(word.split())
```

```
In [59]: # dict sort by value python: https://stackoverflow.com/a/613218/4084039
teacher_cat_dict = dict(my_counter)
sorted_teacher_cat_dict = dict(sorted(teacher_cat_dict.items(), key=lambda kv: kv[1], reverse=True))
```

```
In [60]: #Using Count Vectorizer to convert the teacher_prefix value onto on hot encoded
#ValueError: np.nan is an invalid document, expected byte or unicode string.
#https://stackoverflow.com/questions/39303912/tfidfvectorizer-in-scikit-learn-valueerror-nan-is-an-invalid-document
vectorizer = CountVectorizer(vocabulary=list(sorted_teacher_cat_dict.keys()), lowercase=True)
vectorizer.fit(project_data['teacher_prefix'].values.astype('U'))
print(vectorizer.get_feature_names())

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

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

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

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

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

```
In [61]: # We are considering only the words which appeared in at least 10 documents(rows)
vectorizer = CountVectorizer(min_df=10)
vectorizer.fit(train_preprocessed_essays)

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

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

```
In [62]: #Vectorizing Test Data
# you can vectorize the title also
# before you vectorize the title make sure you preprocess it
test_text_bow = vectorizer.transform(test_preprocessed_essays)
print("Shape of matrix after one hot encoding ",test_text_bow.shape)
```

Shape of matrix after one hot encoding (36052, 12132)

```
In [63]: # Vectrozing CV Data
cv_text_bow = vectorizer.transform(cv_preprocessed_essays)
print("Shape of matrix after one hot encoding ",cv_text_bow.shape)

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

Project_Title BOW

```
In [64]: # We are considering only the words which appeared in at least 10 documents(rows)
vectorizer = CountVectorizer(min_df=10)
vectorizer.fit(train_preprocessed_titles)

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

Shape of matrix after one hot encoding (49041, 2080)

```
In [65]: #Vectorizing Test Data
test_titles_bow = vectorizer.transform(test_preprocessed_titles)
print("Shape of matrix after one hot encoding ",test_titles_bow.shape)
```

Shape of matrix after one hot encoding (36052, 2080)

```
In [66]: #Vectrizing CV Data
cv_titles_bow = vectorizer.transform(cv_preprocessed_titles)
print("Shape of matrix after one hot encoding ",cv_titles_bow.shape)
```

Shape of matrix after one hot encoding (24155, 2080)

1.5.2.2 TFIDF vectorizer

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

```
In [67]: from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(train_preprocessed_essays)

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

Shape of matrix after one hot encoding (49041, 12132)

```
In [68]: test_text_tfidf = vectorizer.transform(test_preprocessed_essays)
print("Shape of matrix after one hot encoding ",test_text_tfidf.shape)
```

Shape of matrix after one hot encoding (36052, 12132)

```
In [69]: cv_text_tfidf = vectorizer.transform(cv_preprocessed_essays)
print("Shape of matrix after one hot encoding ",cv_text_tfidf.shape)
```

Shape of matrix after one hot encoding (24155, 12132)

Project Titles

```
In [70]: vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(train_preprocessed_titles)

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

Shape of matrix after one hot encoding (49041, 2080)

```
In [71]: test_title_tfidf = vectorizer.transform(test_preprocessed_titles)
print("Shape of matrix after one hot encoding ",test_title_tfidf.shape)
```

Shape of matrix after one hot encoding (36052, 2080)

```
In [72]: cv_title_tfidf = vectorizer.transform(cv_preprocessed_titles)
print("Shape of matrix after one hot encoding ",cv_title_tfidf.shape)
```

Shape of matrix after one hot encoding (24155, 2080)

1.5.2.3 Using Pretrained Models: Avg W2V

In [73]:

```

...
# 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 preproc_texts:
    words.extend(i.split(' '))

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

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

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

# stronging variables into pickle files python: http://www.jessicayung.com/how-to

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

...

```

Out[73]: '\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084

```

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

```

In [74]:

```

# stronging variables into pickle files python: http://www.jessicayung.com/how-to-
# 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 [75]:

```

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

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

```

100% |
| 49041/49041 [00:23<00:00, 2066.51it/s]

49041

300

```
In [76]: # average Word2Vec
# compute average word2vec for each review.
test_avg_w2v_vectors = [] # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(test_preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero Length
    cnt_words = 0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_avg_w2v_vectors.append(vector)

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

100% |██████████|
| 36052/36052 [00:17<00:00, 2071.21it/s]

36052
300

```
In [77]: # average Word2Vec
# compute average word2vec for each review.
cv_avg_w2v_vectors = [] # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(cv_preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero Length
    cnt_words = 0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    cv_avg_w2v_vectors.append(vector)

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

100% |██████████|
| 24155/24155 [00:12<00:00, 1980.06it/s]

24155
300

AVG_W2V Project_Titles

```
In [78]: # average Word2Vec
# compute average word2vec for each review.
train_title_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(train_preprocessed_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero Length
    cnt_words = 0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_title_avg_w2v_vectors.append(vector)

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

100% |██████████|
49041/49041 [00:01<00:00, 38773.09it/s]

49041
300

```
In [79]: # average Word2Vec
# compute average word2vec for each review.
test_title_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(test_preprocessed_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero Length
    cnt_words = 0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_title_avg_w2v_vectors.append(vector)

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

100% |██████████|
36052/36052 [00:00<00:00, 39149.70it/s]

36052
300

```
In [80]: # average Word2Vec
# compute average word2vec for each review.
cv_title_avg_w2v_vectors = [] # the avg-w2v for each sentence/review is stored
for sentence in tqdm(cv_preprocessed_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero Length
    cnt_words = 0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    cv_title_avg_w2v_vectors.append(vector)

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

100% |██████████|
24155/24155 [00:00<00:00, 36827.38it/s]

24155
300

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

```
In [81]: # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(train_preprocessed_essays)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

```
In [82]: # average Word2Vec
# compute average word2vec for each review.
train_essay_tfidf_w2v_vectors = [] # the avg-w2v for each sentence/review is stored here
for sentence in tqdm(train_preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero Length
    tf_idf_weight = 0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    train_essay_tfidf_w2v_vectors.append(vector)

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

100% |██████████| 49041/49041 [02:47<00:00, 293.31it/s]

49041
300

```
In [83]: # Similarly you can vectorize for title also
# average Word2Vec
# compute average word2vec for each review.
test_essay_tfidf_w2v_vectors = [] # the avg-w2v for each sentence/review is stored here
for sentence in tqdm(test_preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero Length
    tf_idf_weight = 0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    test_essay_tfidf_w2v_vectors.append(vector)

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

100% |██████████| 36052/36052 [01:57<00:00, 307.65it/s]

36052
300

```
In [84]: # average Word2Vec
# compute average word2vec for each review.
cv_essay_tfidf_w2v_vectors = [] # the avg-w2v for each sentence/review is stored
for sentence in tqdm(cv_preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero Length
    tf_idf_weight = 0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    cv_essay_tfidf_w2v_vectors.append(vector)

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

100% |██████████| 24155/24155 [01:28<00:00, 273.91it/s]

24155
300

Project_titles

```
In [85]: # average Word2Vec
# compute average word2vec for each review.
train_title_tfidf_w2v_vectors = [] # the avg-w2v for each sentence/review is stored
for sentence in tqdm(train_preprocessed_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero Length
    tf_idf_weight = 0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    train_title_tfidf_w2v_vectors.append(vector)

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

100% |██████████| 49041/49041 [00:02<00:00, 19930.41it/s]

49041
300

```
In [86]: # average Word2Vec
# compute average word2vec for each review.
test_title_tfidf_w2v_vectors = [] # the avg-w2v for each sentence/review is stored here
for sentence in tqdm(test_preprocessed_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero Length
    tf_idf_weight = 0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    test_title_tfidf_w2v_vectors.append(vector)

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

100% |
36052/36052 [00:02<00:00, 17867.99it/s]

36052
300

```
In [87]: # average Word2Vec
# compute average word2vec for each review.
cv_title_tfidf_w2v_vectors = [] # the avg-w2v for each sentence/review is stored here
for sentence in tqdm(cv_preprocessed_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero Length
    tf_idf_weight = 0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    cv_title_tfidf_w2v_vectors.append(vector)

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

100% |
24155/24155 [00:01<00:00, 14606.30it/s]

24155
300

1.5.3 Vectorizing Numerical features

```
In [88]: price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
project_data = pd.merge(project_data, price_data, on='id', how='left')
print(price_data.head())
#print(project_data.columns)
print(x_train.columns)
```

| | id | price | quantity |
|---|---------|---------|----------|
| 0 | p000001 | 459.56 | 7 |
| 1 | p000002 | 515.89 | 21 |
| 2 | p000003 | 298.97 | 4 |
| 3 | p000004 | 1113.69 | 98 |
| 4 | p000005 | 485.99 | 8 |

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

```
In [89]: # - quantity : numerical (optional)
# - teacher_number_of_previously_posted_projects : numerical
# - price : numerical
x_train = pd.merge(x_train, price_data, on = "id", how = "left")
#print(x_train.columns)
x_test = pd.merge(x_test, price_data, on = "id", how = "left")
x_cv = pd.merge(x_cv, price_data, on = "id", how = "left")
```

Standardize Price

```
In [90]: # check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler

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

price_scalar = StandardScaler()

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

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

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

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

```
In [91]: print(train_price_standar.shape, y_train.shape)
print(test_price_standar.shape, y_test.shape)
print(cv_price_standar.shape, y_cv.shape)
```

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

Standardize Teacher previously posted Projects

```
In [92]: warnings.filterwarnings("ignore")
price_scalar.fit(x_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
print(f"TRAIN -> Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")
# Now standardize the data with above mean and variance.
train_prev_proj_standar = price_scalar.transform(x_train['teacher_number_of_previous_projects'])

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

price_scalar.fit(x_cv['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
print(f"CV -> Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")
# Now standardize the data with above mean and variance.
cv_prev_proj_standar = price_scalar.transform(x_cv['teacher_number_of_previous_projects'])

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

```
In [93]: print(train_prev_proj_standar.shape, y_train.shape)
print(test_prev_proj_standar.shape, y_test.shape)
print(cv_prev_proj_standar.shape, y_cv.shape)
```

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

Standardize Quantity

```
In [94]: price_scalar.fit(x_train['quantity'].values.reshape(-1,1)) # finding the mean and standard deviation
print(f"TRAIN -> Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")
# Now standardize the data with above mean and variance.
train_quantity_standar = price_scalar.transform(x_train['quantity'].values.reshape(-1,1))

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

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

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

```
In [95]: print(train_quantity_standar.shape, y_train.shape)
print(test_quantity_standar.shape, y_test.shape)
print(cv_quantity_standar.shape, y_cv.shape)

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

Standardize Title_word_count

```
In [96]: title_scalar = StandardScaler()
title_scalar.fit(x_train['title_word_count'].values.reshape(-1,1)) # finding the mean and standard deviation for training data
print(f"Mean : {title_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_)}")
train_title_word_count_standar = title_scalar.transform(x_train['title_word_count'])

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

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

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

Standardize Essay_word_count

```
In [97]: essay_scalar = StandardScaler()

essay_scalar.fit(x_train['essay_word_count'].values.reshape(-1,1)) # finding the mean and standard deviation for training set
train_essay_word_count_standar = essay_scalar.transform(x_train['essay_word_count'])

essay_scalar.fit(x_train['essay_word_count'].values.reshape(-1,1)) # finding the mean and standard deviation for test set
test_essay_word_count_standar = essay_scalar.transform(x_test['essay_word_count'])

essay_scalar.fit(x_cv['essay_word_count'].values.reshape(-1,1)) # finding the mean and standard deviation for cross-validation set
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)

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

Standardize Positive Intensity

```
In [98]: essay_scalar.fit(x_train['positive'].values.reshape(-1,1)) # finding the mean and standard deviation for training set
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 for test set
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 for cross-validation set
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)

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

Standarsize Negitive Intensity

In [99]:

```
essay_scalar.fit(x_train['negative'].values.reshape(-1,1)) # finding the mean and
train_negative_standar = essay_scalar.transform(x_train['negative'].values.reshape(-1,1))

essay_scalar.fit(x_train['negative'].values.reshape(-1,1)) # finding the mean and
test_negative_standar = essay_scalar.transform(x_test['negative'].values.reshape(-1,1))

essay_scalar.fit(x_cv['negative'].values.reshape(-1,1)) # finding the mean and std
cv_negative_standar = essay_scalar.transform(x_cv['negative'].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)
```

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

Standardize Neutral Intensity

In [100]:

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

essay_scalar.fit(x_train['neutral'].values.reshape(-1,1))
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 std
cv_neutral_standar = essay_scalar.transform(x_cv['neutral'].values.reshape(-1,1))

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

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

In []:

Assignment 5: Support Vector Machines

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

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

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

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

- Find the best hyper parameter which will give the maximum [AUC](#) (<https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/>) value
- Find the best hyper parameter using k-fold cross validation or simple cross validation data
- Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

3. Representation of results

- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure.

 Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.

 Along with plotting ROC curve, you need to print the [confusion matrix](#) (<https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/>) with predicted and original labels of test data points. Please visualize your confusion matrices using [seaborn heatmaps](#).

 (<https://seaborn.pydata.org/generated/seaborn.heatmap.html>)

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

- Consider these set of features **Set 5** :
 - **school_state** : categorical data
 - **clean_categories** : categorical data
 - **clean_subcategories** : categorical data
 - **project_grade_category** :categorical data
 - **teacher_prefix** : categorical data
 - **quantity** : numerical data
 - **teacher_number_of_previously_posted_projects** : numerical data
 - **price** : numerical data
 - **sentiment score's of each of the essay** : numerical data
 - **number of words in the title** : numerical data
 - **number of words in the combine essays** : numerical data
 - **Apply TruncatedSVD** (<http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html>) on **TfidfVectorizer** (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html) of essay text, choose the number of components ('n_components') using **elbow method** (<https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/pca-code-example-using-non-visualization/>) : numerical data

- Conclusion

- You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library [link](http://zetcode.com/python/prettytable/) (<http://zetcode.com/python/prettytable/>)



Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakage, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method `fit_transform()` on your train data, and apply the method `transform()` on cv/test data.
4. For more details please go through this [link](https://soundcloud.com/applied-ai-course/leakage-bow-and-tfidf). (<https://soundcloud.com/applied-ai-course/leakage-bow-and-tfidf>)

Support Vector Machine

2.1 SVM On Set-1

Merging all Categorical and Numerical _ SET-1 BOW Encoding

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

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

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

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

```
In [93]: X_cv1 = hstack((cv_categories_one_hot, cv_sub_categories_one_hot, cv_state_one_ho_
    cv_teacher_one_hot, cv_text_bow, cv_titles_bow, cv_quantity_stan_
    cv_prev_proj_standar, cv_price_standar, cv_essay_word_count_stan_
    cv_title_word_count_standar, cv_positive_standar, cv_negitive_
    cv_neutral_standar)).tocsr()
print(X_cv1.shape, y_cv.shape)
print(type(X_cv1))
```

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

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

(49041, 14320) (49041,)

Hyperparameter Tunning

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

```
In [103]: parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4, 10**5, 10**6, 10**7, 10**8, 10**9, 10**10]}

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

classifier.fit(X_train1, y_train)

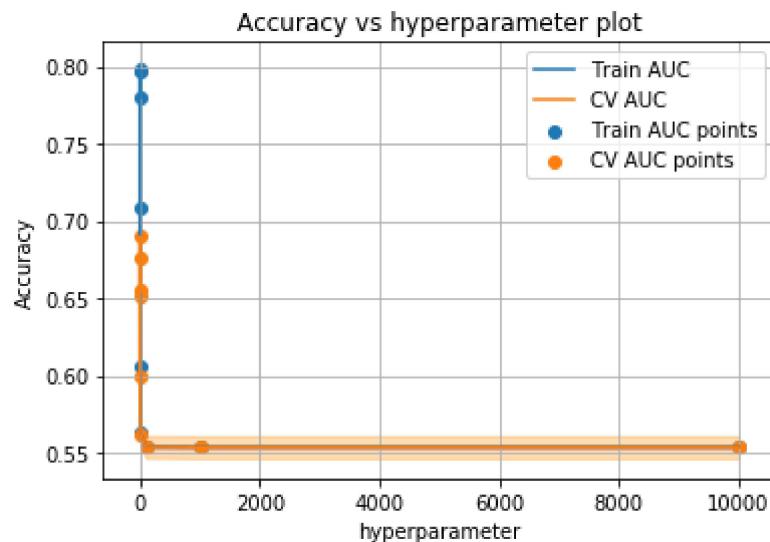
train_auc= classifier.cv_results_['mean_train_score']
train_auc_std= classifier.cv_results_['std_train_score']
cv_auc = classifier.cv_results_['mean_test_score']
cv_auc_std= classifier.cv_results_['std_test_score']

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc + train_auc_std, alpha=.5)

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std, cv_auc + cv_auc_std, alpha=.5)

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

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



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

```
In [139]: parameters = {'alpha':[0.01,0.05,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,1.2,1.4,1.5,1.6,1.7,1.8,1.9,2,2.2,2.4,2.6,2.8,3,3.2,3.4,3.6,3.8,4,4.2,4.4,4.6,4.8,5,5.2,5.4,5.6,5.8,6,6.2,6.4,6.6,6.8,7,7.2,7.4,7.6,7.8,8,8.2,8.4,8.6,8.8,9,9.2,9.4,9.6,9.8,10,10.2,10.4,10.6,10.8,11,11.2,11.4,11.6,11.8,12,12.2,12.4,12.6,12.8,13,13.2,13.4,13.6,13.8,14,14.2,14.4,14.6,14.8,15,15.2,15.4,15.6,15.8,16,16.2,16.4,16.6,16.8,17,17.2,17.4,17.6,17.8,18,18.2,18.4,18.6,18.8,19,19.2,19.4,19.6,19.8,20,20.2,20.4,20.6,20.8,21,21.2,21.4,21.6,21.8,22,22.2,22.4,22.6,22.8,23,23.2,23.4,23.6,23.8,24,24.2,24.4,24.6,24.8,25,25.2,25.4,25.6,25.8,26,26.2,26.4,26.6,26.8,27,27.2,27.4,27.6,27.8,28,28.2,28.4,28.6,28.8,29,29.2,29.4,29.6,29.8,30,30.2,30.4,30.6,30.8,31,31.2,31.4,31.6,31.8,32,32.2,32.4,32.6,32.8,33,33.2,33.4,33.6,33.8,34,34.2,34.4,34.6,34.8,35,35.2,35.4,35.6,35.8,36,36.2,36.4,36.6,36.8,37,37.2,37.4,37.6,37.8,38,38.2,38.4,38.6,38.8,39,39.2,39.4,39.6,39.8,40,40.2,40.4,40.6,40.8,41,41.2,41.4,41.6,41.8,42,42.2,42.4,42.6,42.8,43,43.2,43.4,43.6,43.8,44,44.2,44.4,44.6,44.8,45,45.2,45.4,45.6,45.8,46,46.2,46.4,46.6,46.8,47,47.2,47.4,47.6,47.8,48,48.2,48.4,48.6,48.8,49,49.2,49.4,49.6,49.8,50,50.2,50.4,50.6,50.8,51,51.2,51.4,51.6,51.8,52,52.2,52.4,52.6,52.8,53,53.2,53.4,53.6,53.8,54,54.2,54.4,54.6,54.8,55,55.2,55.4,55.6,55.8,56,56.2,56.4,56.6,56.8,57,57.2,57.4,57.6,57.8,58,58.2,58.4,58.6,58.8,59,59.2,59.4,59.6,59.8,60,60.2,60.4,60.6,60.8,61,61.2,61.4,61.6,61.8,62,62.2,62.4,62.6,62.8,63,63.2,63.4,63.6,63.8,64,64.2,64.4,64.6,64.8,65,65.2,65.4,65.6,65.8,66,66.2,66.4,66.6,66.8,67,67.2,67.4,67.6,67.8,68,68.2,68.4,68.6,68.8,69,69.2,69.4,69.6,69.8,70,70.2,70.4,70.6,70.8,71,71.2,71.4,71.6,71.8,72,72.2,72.4,72.6,72.8,73,73.2,73.4,73.6,73.8,74,74.2,74.4,74.6,74.8,75,75.2,75.4,75.6,75.8,76,76.2,76.4,76.6,76.8,77,77.2,77.4,77.6,77.8,78,78.2,78.4,78.6,78.8,79,79.2,79.4,79.6,79.8,80,80.2,80.4,80.6,80.8,81,81.2,81.4,81.6,81.8,82,82.2,82.4,82.6,82.8,83,83.2,83.4,83.6,83.8,84,84.2,84.4,84.6,84.8,85,85.2,85.4,85.6,85.8,86,86.2,86.4,86.6,86.8,87,87.2,87.4,87.6,87.8,88,88.2,88.4,88.6,88.8,89,89.2,89.4,89.6,89.8,90,90.2,90.4,90.6,90.8,91,91.2,91.4,91.6,91.8,92,92.2,92.4,92.6,92.8,93,93.2,93.4,93.6,93.8,94,94.2,94.4,94.6,94.8,95,95.2,95.4,95.6,95.8,96,96.2,96.4,96.6,96.8,97,97.2,97.4,97.6,97.8,98,98.2,98.4,98.6,98.8,99,99.2,99.4,99.6,99.8,100,100.2,100.4,100.6,100.8,101,101.2,101.4,101.6,101.8,102,102.2,102.4,102.6,102.8,103,103.2,103.4,103.6,103.8,104,104.2,104.4,104.6,104.8,105,105.2,105.4,105.6,105.8,106,106.2,106.4,106.6,106.8,107,107.2,107.4,107.6,107.8,108,108.2,108.4,108.6,108.8,109,109.2,109.4,109.6,109.8,110,110.2,110.4,110.6,110.8,111,111.2,111.4,111.6,111.8,112,112.2,112.4,112.6,112.8,113,113.2,113.4,113.6,113.8,114,114.2,114.4,114.6,114.8,115,115.2,115.4,115.6,115.8,116,116.2,116.4,116.6,116.8,117,117.2,117.4,117.6,117.8,118,118.2,118.4,118.6,118.8,119,119.2,119.4,119.6,119.8,120,120.2,120.4,120.6,120.8,121,121.2,121.4,121.6,121.8,122,122.2,122.4,122.6,122.8,123,123.2,123.4,123.6,123.8,124,124.2,124.4,124.6,124.8,125,125.2,125.4,125.6,125.8,126,126.2,126.4,126.6,126.8,127,127.2,127.4,127.6,127.8,128,128.2,128.4,128.6,128.8,129,129.2,129.4,129.6,129.8,130,130.2,130.4,130.6,130.8,131,131.2,131.4,131.6,131.8,132,132.2,132.4,132.6,132.8,133,133.2,133.4,133.6,133.8,134,134.2,134.4,134.6,134.8,135,135.2,135.4,135.6,135.8,136,136.2,136.4,136.6,136.8,137,137.2,137.4,137.6,137.8,138,138.2,138.4,138.6,138.8,139,139.2,139.4,139.6,139.8,140,140.2,140.4,140.6,140.8,141,141.2,141.4,141.6,141.8,142,142.2,142.4,142.6,142.8,143,143.2,143.4,143.6,143.8,144,144.2,144.4,144.6,144.8,145,145.2,145.4,145.6,145.8,146,146.2,146.4,146.6,146.8,147,147.2,147.4,147.6,147.8,148,148.2,148.4,148.6,148.8,149,149.2,149.4,149.6,149.8,150,150.2,150.4,150.6,150.8,151,151.2,151.4,151.6,151.8,152,152.2,152.4,152.6,152.8,153,153.2,153.4,153.6,153.8,154,154.2,154.4,154.6,154.8,155,155.2,155.4,155.6,155.8,156,156.2,156.4,156.6,156.8,157,157.2,157.4,157.6,157.8,158,158.2,158.4,158.6,158.8,159,159.2,159.4,159.6,159.8,160,160.2,160.4,160.6,160.8,161,161.2,161.4,161.6,161.8,162,162.2,162.4,162.6,162.8,163,163.2,163.4,163.6,163.8,164,164.2,164.4,164.6,164.8,165,165.2,165.4,165.6,165.8,166,166.2,166.4,166.6,166.8,167,167.2,167.4,167.6,167.8,168,168.2,168.4,168.6,168.8,169,169.2,169.4,169.6,169.8,170,170.2,170.4,170.6,170.8,171,171.2,171.4,171.6,171.8,172,172.2,172.4,172.6,172.8,173,173.2,173.4,173.6,173.8,174,174.2,174.4,174.6,174.8,175,175.2,175.4,175.6,175.8,176,176.2,176.4,176.6,176.8,177,177.2,177.4,177.6,177.8,178,178.2,178.4,178.6,178.8,179,179.2,179.4,179.6,179.8,180,180.2,180.4,180.6,180.8,181,181.2,181.4,181.6,181.8,182,182.2,182.4,182.6,182.8,183,183.2,183.4,183.6,183.8,184,184.2,184.4,184.6,184.8,185,185.2,185.4,185.6,185.8,186,186.2,186.4,186.6,186.8,187,187.2,187.4,187.6,187.8,188,188.2,188.4,188.6,188.8,189,189.2,189.4,189.6,189.8,190,190.2,190.4,190.6,190.8,191,191.2,191.4,191.6,191.8,192,192.2,192.4,192.6,192.8,193,193.2,193.4,193.6,193.8,194,194.2,194.4,194.6,194.8,195,195.2,195.4,195.6,195.8,196,196.2,196.4,196.6,196.8,197,197.2,197.4,197.6,197.8,198,198.2,198.4,198.6,198.8,199,199.2,199.4,199.6,199.8,200,200.2,200.4,200.6,200.8,201,201.2,201.4,201.6,201.8,202,202.2,202.4,202.6,202.8,203,203.2,203.4,203.6,203.8,204,204.2,204.4,204.6,204.8,205,205.2,205.4,205.6,205.8,206,206.2,206.4,206.6,206.8,207,207.2,207.4,207.6,207.8,208,208.2,208.4,208.6,208.8,209,209.2,209.4,209.6,209.8,210,210.2,210.4,210.6,210.8,211,211.2,211.4,211.6,211.8,212,212.2,212.4,212.6,212.8,213,213.2,213.4,213.6,213.8,214,214.2,214.4,214.6,214.8,215,215.2,215.4,215.6,215.8,216,216.2,216.4,216.6,216.8,217,217.2,217.4,217.6,217.8,218,218.2,218.4,218.6,218.8,219,219.2,219.4,219.6,219.8,220,220.2,220.4,220.6,220.8,221,221.2,221.4,221.6,221.8,222,222.2,222.4,222.6,222.8,223,223.2,223.4,223.6,223.8,224,224.2,224.4,224.6,224.8,225,225.2,225.4,225.6,225.8,226,226.2,226.4,226.6,226.8,227,227.2,227.4,227.6,227.8,228,228.2,228.4,228.6,228.8,229,229.2,229.4,229.6,229.8,230,230.2,230.4,230.6,230.8,231,231.2,231.4,231.6,231.8,232,232.2,232.4,232.6,232.8,233,233.2,233.4,233.6,233.8,234,234.2,234.4,234.6,234.8,235,235.2,235.4,235.6,235.8,236,236.2,236.4,236.6,236.8,237,237.2,237.4,237.6,237.8,238,238.2,238.4,238.6,238.8,239,239.2,239.4,239.6,239.8,240,240.2,240.4,240.6,240.8,241,241.2,241.4,241.6,241.8,242,242.2,242.4,242.6,242.8,243,243.2,243.4,243.6,243.8,244,244.2,244.4,244.6,244.8,245,245.2,245.4,245.6,245.8,246,246.2,246.4,246.6,246.8,247,247.2,247.4,247.6,247.8,248,248.2,248.4,248.6,248.8,249,249.2,249.4,249.6,249.8,250,250.2,250.4,250.6,250.8,251,251.2,251.4,251.6,251.8,252,252.2,252.4,252.6,252.8,253,253.2,253.4,253.6,253.8,254,254.2,254.4,254.6,254.8,255,255.2,255.4,255.6,255.8,256,256.2,256.4,256.6,256.8,257,257.2,257.4,257.6,257.8,258,258.2,258.4,258.6,258.8,259,259.2,259.4,259.6,259.8,260,260.2,260.4,260.6,260.8,261,261.2,261.4,261.6,261.8,262,262.2,262.4,262.6,262.8,263,263.2,263.4,263.6,263.8,264,264.2,264.4,264.6,264.8,265,265.2,265.4,265.6,265.8,266,266.2,266.4,266.6,266.8,267,267.2,267.4,267.6,267.8,268,268.2,268.4,268.6,268.8,269,269.2,269.4,269.6,269.8,270,270.2,270.4,270.6,270.8,271,271.2,271.4,271.6,271.8,272,272.2,272.4,272.6,272.8,273,273.2,273.4,273.6,273.8,274,274.2,274.4,274.6,274.8,275,275.2,275.4,275.6,275.8,276,276.2,276.4,276.6,276.8,277,277.2,277.4,277.6,277.8,278,278.2,278.4,278.6,278.8,279,279.2,279.4,279.6,279.8,280,280.2,280.4,280.6,280.8,281,281.2,281.4,281.6,281.8,282,282.2,282.4,282.6,282.8,283,283.2,283.4,283.6,283.8,284,284.2,284.4,284.6,284.8,285,285.2,285.4,285.6,285.8,286,286.2,286.4,286.6,286.8,287,287.2,287.4,287.6,287.8,288,288.2,288.4,288.6,288.8,289,289.2,289.4,289.6,289.8,290,290.2,290.4,290.6,290.8,291,291.2,291.4,291.6,291.8,292,292.2,292.4,292.6,292.8,293,293.2,293.4,293.6,293.8,294,294.2,294.4,294.6,294.8,295,295.2,295.4,295.6,295.8,296,296.2,296.4,296.6,296.8,297,297.2,297.4,297.6,297.8,298,298.2,298.4,298.6,298.8,299,299.2,299.4,299.6,299.8,300,300.2,300.4,300.6,300.8,301,301.2,301.4,301.6,301.8,302,302.2,302.4,302.6,302.8,303,303.2,303.4,303.6,303.8,304,304.2,304.4,304.6,304.8,305,305.2,305.4,305.6,305.8,306,306.2,306.4,306.6,306.8,307,307.2,307.4,307.6,307.8,308,308.2,308.4,308.6,308.8,309,309.2,309.4,309.6,309.8,310,310.2,310.4,310.6,310.8,311,311.2,311.4,311.6,311.8,312,312.2,312.4,312.6,312.8,313,313.2,313.4,313.6,313.8,314,314.2,314.4,314.6,314.8,315,315.2,315.4,315.6,315.8,316,316.2,316.4,316.6,316.8,317,317.2,317.4,317.6,317.8,318,318.2,318.4,318.6,318.8,319,319.2,319.4,319.6,319.8,320,320.2,320.4,320.6,320.8,321,321.2,321.4,321.6,321.8,322,322.2,322.4,322.6,322.8,323,323.2,323.4,323.6,323.8,324,324.2,324.4,324.6,324.8,325,325.2,325.4,325.6,325.8,326,326.2,326.4,326.6,326.8,327,327.2,327.4,327.6,327.8,328,328.2,328.4,328.6,328.8,329,329.2,329.4,329.6,329.8,330,330.2,330.4,330.6,330.8,331,331.2,331.4,331.6,331.8,332,332.2,332.4,332.6,332.8,333,333.2,333.4,333.6,333.8,334,334.2,334.4,334.6,334.8,335,335.2,335.4,335.6,335.8,336,336.2,336.4,336.6,336.8,337,337.2,337.4,337.6,337.8,338,338.2,338.4,338.6,338.8,339,339.2,339.4,339.6,339.8,340,340.2,340.4,340.6,340.8,341,341.2,341.4,341.6,341.8,342,342.2,342.4,342.6,342.8,343,343.2,343.4,343.6,343.8,344,344.2,344.4,344.6,344.8,345,345.2,345.4,345.6,345.8,346,346.2,346.4,346.6,346.8,347,347.2,347.4,347.6,347.8,348,348.2,348.4,348.6,348.8,349,349.2,349.4,349.6,349.8,350,350.2,350.4,350.6,350.8,351,351.2,351.4,351.6,351.8,352,352.2,352.4,352.6,352.8,353,353.2,353.4,353.6,353.8,354,354.2,354.4,354.6,354.8,355,355.2,355.4,355.6,355.8,356,356.2,356.4,356.6,356.8,357,357.2,357.4,357.6,357.8,358,358.2,358.4,358.6,358.8,359,359.2,359.4,359.6,359.8,360,360.2,360.4,360.6,360.8,361,361.2,361.4,361.6,361.8,362,362.2,362.4,362.6,362.8,363,363.2,363.4,363.6,363.8,364,364.2,364.4,364.6,364.8,365,365.2,365.4,365.6,365.8,366,366.2,366.4,366.6,366.8,367,367.2,367.4,367.6,367.8,368,368.2,368.4,368.6,368.8,369,369.2,369.4,369.6,369.8,370,370.2,370.4,370.6,370.8,371,371.2,371.4,371.6,371.8,372,372.2,372.4,372.6,372.8,373,373.2,373.4,373.6,373.8,374,374.2,374.4,374.6,374.8,375,375.2,375.4,375.6,375.8,376,376.2,376.4,376.6,376.8,377,377.2,377.4,377.6,377.8,378,378.2,378.4,378.6,378.8,379,379.2,379.4,379.6,379.8,380,380.2,380.4,380.6,380.8,381,381.2,381.4,381.6,381.8,382,382.2,382.4,382.6,382.8,383,383.2,383.4,383.6,383.8,384,384.2,384.4,384.6,384.8,385,385.2,385.4,385.6,385.8,386,386.2,386.4,386.6,386.8,387,387.2,387.4,387.6,387.8,388,388.2,388.4,388.6,388.8,389,389.2,389.4,389.6,389.8,390,390.2,390.4,390.6,390.8,391,391.2,391.4,391.6,391.8,392,392.2,392.4,392.6,392.8,393,393.2,393.4,393.6,393.8,394,394.2,394.4,394.6,394.8,395,395.2,395.4,395.6,395.8,396,396.2,396.4,396.6,396.8,397,397.2,397.4,397.6,397.8,398,398.2,398.4,398.6,398.8,399,399.2,399.4,399.6,399.8,400,400.2,400.4,400.6,400.8,401,401.2,401.4,401.6,401.8,402,402.2,402.4,402.6,402.8,403,403.2,403.4,403.6,403.8,404,404.2,404.4,404.6,404.8,405,405.2,405.4,405.6,405.8,406,406.2,406.4,406.6,406.8,407,407.2,407.4,407.6,407.8,408,408.2,408.4,408.6,408.8,409,409.2,409.4,409.6,409.8,410,410.2,410.4,410.6,410.8,411,411.2,411.4,411.6,411.8,412,412.2,412.4,412.6,412.8,413,413.2,413.4,413.6,413.8,414,414.2,414.4,414.6,414.8,415,415.2,415.4,415.6,415.8,416,416.2,416.4,416.6,416.8,417,417.2,417.4,417.6,417.8,418,418.2,418.4,418.6,418.8,419,419.2,419.4,419.6,419.8,420,420.2,420.4,420.6,420.8,421,4
```

```
In [140]: parameters = {'alpha':[0.01,0.05,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,1.2,1.4,1.6,1.8,2,2.2,2.4,2.6,2.8,3,3.2,3.4,3.6,3.8,4,4.2,4.4,4.6,4.8,5,5.2,5.4,5.6,5.8,6,6.2,6.4,6.6,6.8,7,7.2,7.4,7.6,7.8,8,8.2,8.4,8.6,8.8,9,9.2,9.4,9.6,9.8,10,10.2,10.4,10.6,10.8,11,11.2,11.4,11.6,11.8,12,12.2,12.4,12.6,12.8,13,13.2,13.4,13.6,13.8,14,14.2,14.4,14.6,14.8,15,15.2,15.4,15.6,15.8,16,16.2,16.4,16.6,16.8,17,17.2,17.4,17.6,17.8,18,18.2,18.4,18.6,18.8,19,19.2,19.4,19.6,19.8,20,20.2,20.4,20.6,20.8,21,21.2,21.4,21.6,21.8,22,22.2,22.4,22.6,22.8,23,23.2,23.4,23.6,23.8,24,24.2,24.4,24.6,24.8,25,25.2,25.4,25.6,25.8,26,26.2,26.4,26.6,26.8,27,27.2,27.4,27.6,27.8,28,28.2,28.4,28.6,28.8,29,29.2,29.4,29.6,29.8,30,30.2,30.4,30.6,30.8,31,31.2,31.4,31.6,31.8,32,32.2,32.4,32.6,32.8,33,33.2,33.4,33.6,33.8,34,34.2,34.4,34.6,34.8,35,35.2,35.4,35.6,35.8,36,36.2,36.4,36.6,36.8,37,37.2,37.4,37.6,37.8,38,38.2,38.4,38.6,38.8,39,39.2,39.4,39.6,39.8,40,40.2,40.4,40.6,40.8,41,41.2,41.4,41.6,41.8,42,42.2,42.4,42.6,42.8,43,43.2,43.4,43.6,43.8,44,44.2,44.4,44.6,44.8,45,45.2,45.4,45.6,45.8,46,46.2,46.4,46.6,46.8,47,47.2,47.4,47.6,47.8,48,48.2,48.4,48.6,48.8,49,49.2,49.4,49.6,49.8,50,50.2,50.4,50.6,50.8,51,51.2,51.4,51.6,51.8,52,52.2,52.4,52.6,52.8,53,53.2,53.4,53.6,53.8,54,54.2,54.4,54.6,54.8,55,55.2,55.4,55.6,55.8,56,56.2,56.4,56.6,56.8,57,57.2,57.4,57.6,57.8,58,58.2,58.4,58.6,58.8,59,59.2,59.4,59.6,59.8,60,60.2,60.4,60.6,60.8,61,61.2,61.4,61.6,61.8,62,62.2,62.4,62.6,62.8,63,63.2,63.4,63.6,63.8,64,64.2,64.4,64.6,64.8,65,65.2,65.4,65.6,65.8,66,66.2,66.4,66.6,66.8,67,67.2,67.4,67.6,67.8,68,68.2,68.4,68.6,68.8,69,69.2,69.4,69.6,69.8,70,70.2,70.4,70.6,70.8,71,71.2,71.4,71.6,71.8,72,72.2,72.4,72.6,72.8,73,73.2,73.4,73.6,73.8,74,74.2,74.4,74.6,74.8,75,75.2,75.4,75.6,75.8,76,76.2,76.4,76.6,76.8,77,77.2,77.4,77.6,77.8,78,78.2,78.4,78.6,78.8,79,79.2,79.4,79.6,79.8,80,80.2,80.4,80.6,80.8,81,81.2,81.4,81.6,81.8,82,82.2,82.4,82.6,82.8,83,83.2,83.4,83.6,83.8,84,84.2,84.4,84.6,84.8,85,85.2,85.4,85.6,85.8,86,86.2,86.4,86.6,86.8,87,87.2,87.4,87.6,87.8,88,88.2,88.4,88.6,88.8,89,89.2,89.4,89.6,89.8,90,90.2,90.4,90.6,90.8,91,91.2,91.4,91.6,91.8,92,92.2,92.4,92.6,92.8,93,93.2,93.4,93.6,93.8,94,94.2,94.4,94.6,94.8,95,95.2,95.4,95.6,95.8,96,96.2,96.4,96.6,96.8,97,97.2,97.4,97.6,97.8,98,98.2,98.4,98.6,98.8,99,99.2,99.4,99.6,99.8,100,100.2,100.4,100.6,100.8,101,101.2,101.4,101.6,101.8,102,102.2,102.4,102.6,102.8,103,103.2,103.4,103.6,103.8,104,104.2,104.4,104.6,104.8,105,105.2,105.4,105.6,105.8,106,106.2,106.4,106.6,106.8,107,107.2,107.4,107.6,107.8,108,108.2,108.4,108.6,108.8,109,109.2,109.4,109.6,109.8,110,110.2,110.4,110.6,110.8,111,111.2,111.4,111.6,111.8,112,112.2,112.4,112.6,112.8,113,113.2,113.4,113.6,113.8,114,114.2,114.4,114.6,114.8,115,115.2,115.4,115.6,115.8,116,116.2,116.4,116.6,116.8,117,117.2,117.4,117.6,117.8,118,118.2,118.4,118.6,118.8,119,119.2,119.4,119.6,119.8,120,120.2,120.4,120.6,120.8,121,121.2,121.4,121.6,121.8,122,122.2,122.4,122.6,122.8,123,123.2,123.4,123.6,123.8,124,124.2,124.4,124.6,124.8,125,125.2,125.4,125.6,125.8,126,126.2,126.4,126.6,126.8,127,127.2,127.4,127.6,127.8,128,128.2,128.4,128.6,128.8,129,129.2,129.4,129.6,129.8,130,130.2,130.4,130.6,130.8,131,131.2,131.4,131.6,131.8,132,132.2,132.4,132.6,132.8,133,133.2,133.4,133.6,133.8,134,134.2,134.4,134.6,134.8,135,135.2,135.4,135.6,135.8,136,136.2,136.4,136.6,136.8,137,137.2,137.4,137.6,137.8,138,138.2,138.4,138.6,138.8,139,139.2,139.4,139.6,139.8,140,140.2,140.4,140.6,140.8,141,141.2,141.4,141.6,141.8,142,142.2,142.4,142.6,142.8,143,143.2,143.4,143.6,143.8,144,144.2,144.4,144.6,144.8,145,145.2,145.4,145.6,145.8,146,146.2,146.4,146.6,146.8,147,147.2,147.4,147.6,147.8,148,148.2,148.4,148.6,148.8,149,149.2,149.4,149.6,149.8,150,150.2,150.4,150.6,150.8,151,151.2,151.4,151.6,151.8,152,152.2,152.4,152.6,152.8,153,153.2,153.4,153.6,153.8,154,154.2,154.4,154.6,154.8,155,155.2,155.4,155.6,155.8,156,156.2,156.4,156.6,156.8,157,157.2,157.4,157.6,157.8,158,158.2,158.4,158.6,158.8,159,159.2,159.4,159.6,159.8,160,160.2,160.4,160.6,160.8,161,161.2,161.4,161.6,161.8,162,162.2,162.4,162.6,162.8,163,163.2,163.4,163.6,163.8,164,164.2,164.4,164.6,164.8,165,165.2,165.4,165.6,165.8,166,166.2,166.4,166.6,166.8,167,167.2,167.4,167.6,167.8,168,168.2,168.4,168.6,168.8,169,169.2,169.4,169.6,169.8,170,170.2,170.4,170.6,170.8,171,171.2,171.4,171.6,171.8,172,172.2,172.4,172.6,172.8,173,173.2,173.4,173.6,173.8,174,174.2,174.4,174.6,174.8,175,175.2,175.4,175.6,175.8,176,176.2,176.4,176.6,176.8,177,177.2,177.4,177.6,177.8,178,178.2,178.4,178.6,178.8,179,179.2,179.4,179.6,179.8,180,180.2,180.4,180.6,180.8,181,181.2,181.4,181.6,181.8,182,182.2,182.4,182.6,182.8,183,183.2,183.4,183.6,183.8,184,184.2,184.4,184.6,184.8,185,185.2,185.4,185.6,185.8,186,186.2,186.4,186.6,186.8,187,187.2,187.4,187.6,187.8,188,188.2,188.4,188.6,188.8,189,189.2,189.4,189.6,189.8,190,190.2,190.4,190.6,190.8,191,191.2,191.4,191.6,191.8,192,192.2,192.4,192.6,192.8,193,193.2,193.4,193.6,193.8,194,194.2,194.4,194.6,194.8,195,195.2,195.4,195.6,195.8,196,196.2,196.4,196.6,196.8,197,197.2,197.4,197.6,197.8,198,198.2,198.4,198.6,198.8,199,199.2,199.4,199.6,199.8,200,200.2,200.4,200.6,200.8,201,201.2,201.4,201.6,201.8,202,202.2,202.4,202.6,202.8,203,203.2,203.4,203.6,203.8,204,204.2,204.4,204.6,204.8,205,205.2,205.4,205.6,205.8,206,206.2,206.4,206.6,206.8,207,207.2,207.4,207.6,207.8,208,208.2,208.4,208.6,208.8,209,209.2,209.4,209.6,209.8,210,210.2,210.4,210.6,210.8,211,211.2,211.4,211.6,211.8,212,212.2,212.4,212.6,212.8,213,213.2,213.4,213.6,213.8,214,214.2,214.4,214.6,214.8,215,215.2,215.4,215.6,215.8,216,216.2,216.4,216.6,216.8,217,217.2,217.4,217.6,217.8,218,218.2,218.4,218.6,218.8,219,219.2,219.4,219.6,219.8,220,220.2,220.4,220.6,220.8,221,221.2,221.4,221.6,221.8,222,222.2,222.4,222.6,222.8,223,223.2,223.4,223.6,223.8,224,224.2,224.4,224.6,224.8,225,225.2,225.4,225.6,225.8,226,226.2,226.4,226.6,226.8,227,227.2,227.4,227.6,227.8,228,228.2,228.4,228.6,228.8,229,229.2,229.4,229.6,229.8,230,230.2,230.4,230.6,230.8,231,231.2,231.4,231.6,231.8,232,232.2,232.4,232.6,232.8,233,233.2,233.4,233.6,233.8,234,234.2,234.4,234.6,234.8,235,235.2,235.4,235.6,235.8,236,236.2,236.4,236.6,236.8,237,237.2,237.4,237.6,237.8,238,238.2,238.4,238.6,238.8,239,239.2,239.4,239.6,239.8,240,240.2,240.4,240.6,240.8,241,241.2,241.4,241.6,241.8,242,242.2,242.4,242.6,242.8,243,243.2,243.4,243.6,243.8,244,244.2,244.4,244.6,244.8,245,245.2,245.4,245.6,245.8,246,246.2,246.4,246.6,246.8,247,247.2,247.4,247.6,247.8,248,248.2,248.4,248.6,248.8,249,249.2,249.4,249.6,249.8,250,250.2,250.4,250.6,250.8,251,251.2,251.4,251.6,251.8,252,252.2,252.4,252.6,252.8,253,253.2,253.4,253.6,253.8,254,254.2,254.4,254.6,254.8,255,255.2,255.4,255.6,255.8,256,256.2,256.4,256.6,256.8,257,257.2,257.4,257.6,257.8,258,258.2,258.4,258.6,258.8,259,259.2,259.4,259.6,259.8,260,260.2,260.4,260.6,260.8,261,261.2,261.4,261.6,261.8,262,262.2,262.4,262.6,262.8,263,263.2,263.4,263.6,263.8,264,264.2,264.4,264.6,264.8,265,265.2,265.4,265.6,265.8,266,266.2,266.4,266.6,266.8,267,267.2,267.4,267.6,267.8,268,268.2,268.4,268.6,268.8,269,269.2,269.4,269.6,269.8,270,270.2,270.4,270.6,270.8,271,271.2,271.4,271.6,271.8,272,272.2,272.4,272.6,272.8,273,273.2,273.4,273.6,273.8,274,274.2,274.4,274.6,274.8,275,275.2,275.4,275.6,275.8,276,276.2,276.4,276.6,276.8,277,277.2,277.4,277.6,277.8,278,278.2,278.4,278.6,278.8,279,279.2,279.4,279.6,279.8,280,280.2,280.4,280.6,280.8,281,281.2,281.4,281.6,281.8,282,282.2,282.4,282.6,282.8,283,283.2,283.4,283.6,283.8,284,284.2,284.4,284.6,284.8,285,285.2,285.4,285.6,285.8,286,286.2,286.4,286.6,286.8,287,287.2,287.4,287.6,287.8,288,288.2,288.4,288.6,288.8,289,289.2,289.4,289.6,289.8,290,290.2,290.4,290.6,290.8,291,291.2,291.4,291.6,291.8,292,292.2,292.4,292.6,292.8,293,293.2,293.4,293.6,293.8,294,294.2,294.4,294.6,294.8,295,295.2,295.4,295.6,295.8,296,296.2,296.4,296.6,296.8,297,297.2,297.4,297.6,297.8,298,298.2,298.4,298.6,298.8,299,299.2,299.4,299.6,299.8,300,300.2,300.4,300.6,300.8,301,301.2,301.4,301.6,301.8,302,302.2,302.4,302.6,302.8,303,303.2,303.4,303.6,303.8,304,304.2,304.4,304.6,304.8,305,305.2,305.4,305.6,305.8,306,306.2,306.4,306.6,306.8,307,307.2,307.4,307.6,307.8,308,308.2,308.4,308.6,308.8,309,309.2,309.4,309.6,309.8,310,310.2,310.4,310.6,310.8,311,311.2,311.4,311.6,311.8,312,312.2,312.4,312.6,312.8,313,313.2,313.4,313.6,313.8,314,314.2,314.4,314.6,314.8,315,315.2,315.4,315.6,315.8,316,316.2,316.4,316.6,316.8,317,317.2,317.4,317.6,317.8,318,318.2,318.4,318.6,318.8,319,319.2,319.4,319.6,319.8,320,320.2,320.4,320.6,320.8,321,321.2,321.4,321.6,321.8,322,322.2,322.4,322.6,322.8,323,323.2,323.4,323.6,323.8,324,324.2,324.4,324.6,324.8,325,325.2,325.4,325.6,325.8,326,326.2,326.4,326.6,326.8,327,327.2,327.4,327.6,327.8,328,328.2,328.4,328.6,328.8,329,329.2,329.4,329.6,329.8,330,330.2,330.4,330.6,330.8,331,331.2,331.4,331.6,331.8,332,332.2,332.4,332.6,332.8,333,333.2,333.4,333.6,333.8,334,334.2,334.4,334.6,334.8,335,335.2,335.4,335.6,335.8,336,336.2,336.4,336.6,336.8,337,337.2,337.4,337.6,337.8,338,338.2,338.4,338.6,338.8,339,339.2,339.4,339.6,339.8,340,340.2,340.4,340.6,340.8,341,341.2,341.4,341.6,341.8,342,342.2,342.4,342.6,342.8,343,343.2,343.4,343.6,343.8,344,344.2,344.4,344.6,344.8,345,345.2,345.4,345.6,345.8,346,346.2,346.4,346.6,346.8,347,347.2,347.4,347.6,347.8,348,348.2,348.4,348.6,348.8,349,349.2,349.4,349.6,349.8,350,350.2,350.4,350.6,350.8,351,351.2,351.4,351.6,351.8,352,352.2,352.4,352.6,352.8,353,353.2,353.4,353.6,353.8,354,354.2,354.4,354.6,354.8,355,355.2,355.4,355.6,355.8,356,356.2,356.4,356.6,356.8,357,357.2,357.4,357.6,357.8,358,358.2,358.4,358.6,358.8,359,359.2,359.4,359.6,359.8,360,360.2,360.4,360.6,360.8,361,361.2,361.4,361.6,361.8,362,362.2,362.4,362.6,362.8,363,363.2,363.4,363.6,363.8,364,364.2,364.4,364.6,364.8,365,365.2,365.4,365.6,365.8,366,366.2,366.4,366.6,366.8,367,367.2,367.4,367.6,367.8,368,368.2,368.4,368.6,368.8,369,369.2,369.4,369.6,369.8,370,370.2,370.4,370.6,370.8,371,371.2,371.4,371.6,371.8,372,372.2,372.4,372.6,372.8,373,373.2,373.4,373.6,373.8,374,374.2,374.4,374.6,374.8,375,375.2,375.4,375.6,375.8,376,376.2,376.4,376.6,376.8,377,377.2,377.4,377.6,377.8,378,378.2,378.4,378.6,378.8,379,379.2,379.4,379.6,379.8,380,380.2,380.4,380.6,380.8,381,381.2,381.4,381.6,381.8,382,382.2,382.4,382.6,382.8,383,383.2,383.4,383.6,383.8,384,384.2,384.4,384.6,384.8,385,385.2,385.4,385.6,385.8,386,386.2,386.4,386.6,386.8,387,387.2,387.4,387.6,387.8,388,388.2,388.4,388.6,388.8,389,389.2,389.4,389.6,389.8,390,390.2,390.4,390.6,390.8,391,391.2,391.4,391.6,391.8,392,392.2,392.4,392.6,392.8,393,393.2,393.4,393.6,393.8,394,394.2,394.4,394.6,394.8,395,395.2,395.4,395.6,395.8,396,396.2,396.4,396.6,396.8,397,397.2,397.4,397.6,397.8,398,398.2,398.4,398.6,398.8,399,399.2,399.4,399.6,399.8,400,400.2,400.4,400.6,400.8,401,401.2,401.4,401.6,401.8,402,402.2,402.4,402.6,402.8,403,403.2,403.4,403.6,403.8,404,404.2,404.4,404.6,404.8,405,405.2,405.4,405.6,405.8,406,406.2,406.4,406.6,406.8,407,407.2,407.4,407.6,407.8,408,408.2,408.4,408.6,408.8,409,409.2,409.4,409.6,409.8,410,410.2,410.4,410.6,410.8,411,411.2,411.4,411.6,411.8,412,412.2,412.4,412.6,412.8,413,413.2,413.4,413.6,413.8,414,414.2,414.4,414.6,414.8,415,415.2,415.4,415.6,415.8,416,416.2,416.4,416.6,416.8,417,417.2,417.4,417.6,417.8,418,418.2,418.4,418.6,418.8,419,419.2,419.4,419.6,419.8,420,420.2,420.4,420.6,420.8,421,421.2,421.4,4
```

```
In [115]: def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability est-
    # not the predicted outputs

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

    return y_data_pred
```

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

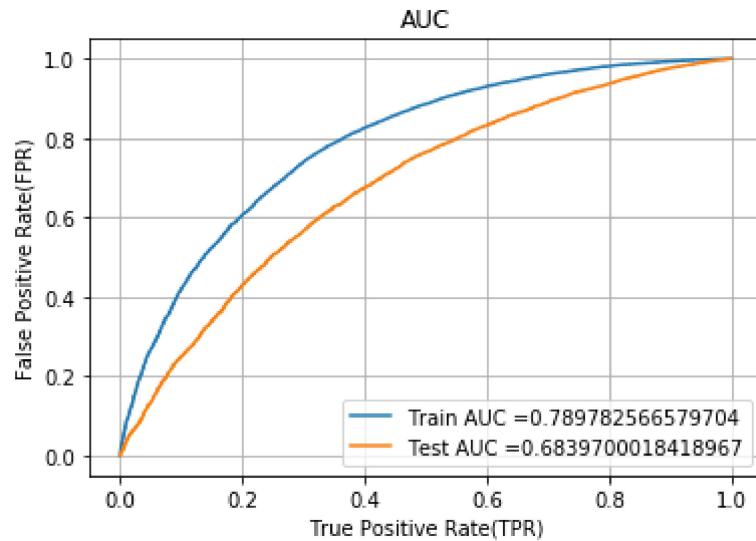
```
In [128]: # https://scikit-Learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html
from sklearn.metrics import roc_curve, auc
#default alpha = 0.0001
Classifier_bow = SGDClassifier(loss = 'hinge', penalty = 'l2', alpha = 0.001)

Classifier_bow.fit(X_train1, y_train)
#https://scikit-Learn.org/stable/modules/generated/skLearn.linear_model.SGDClassi

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

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

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



Confusion Matrix

```
In [129]: def predict(proba, threshold, fpr, tpr):

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

Train Data

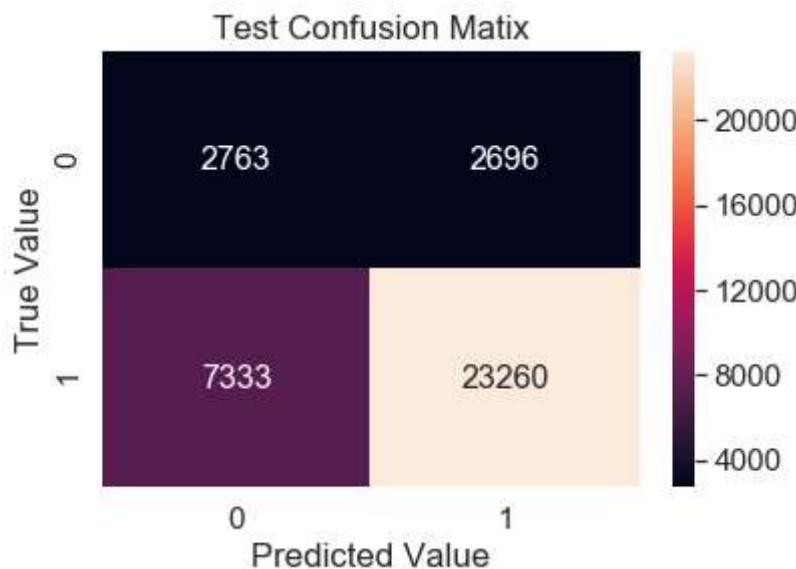
```
In [130]: from sklearn.metrics import confusion_matrix
import seaborn as sea
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, test_fpr, test_thresholds)))
```

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

```
In [131]: train_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred, tr_thresholds, test_fpr, test_thresholds)))
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) 0.24999999161092998 for threshold 1.538

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



Test Data

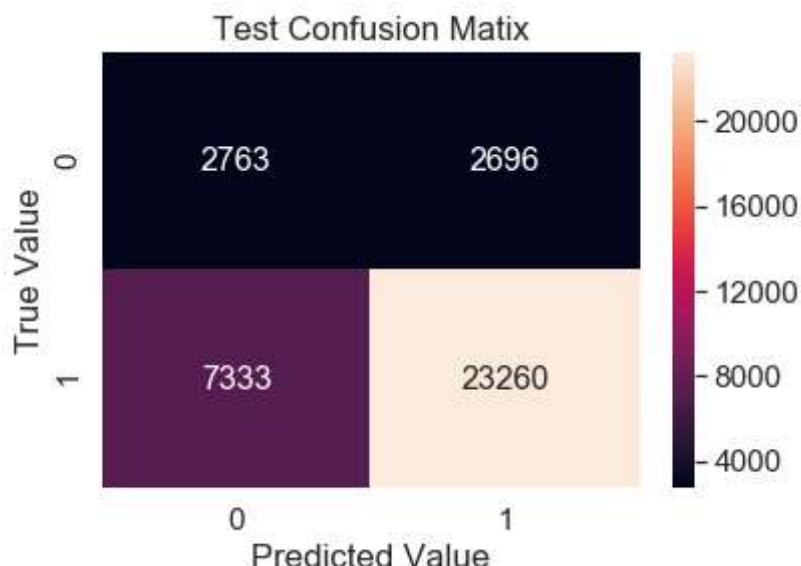
```
In [132]: print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_thresholds)))
```

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

```
In [133]: train_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred),test_fpr,test_tpr))
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) 0.24999999161092998 for threshold 1.538

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



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

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

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

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

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

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

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

Hyperparameter Tunning

```
In [212]: parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4, 10**5]}

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

classifier.fit(X_train2, y_train)

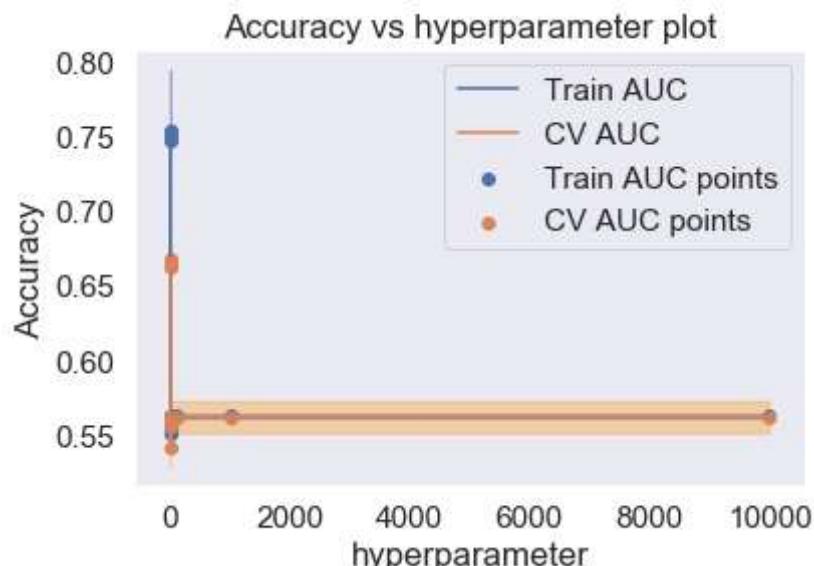
train_auc= classifier.cv_results_['mean_train_score']
train_auc_std= classifier.cv_results_['std_train_score']
cv_auc = classifier.cv_results_['mean_test_score']
cv_auc_std= classifier.cv_results_['std_test_score']

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc + train_auc_std, alpha=.5)

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std, cv_auc + cv_auc_std, alpha=.5)

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

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



Again for TF-IDF encoding we can't able to find best "ALPHA" from above

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

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

classifier.fit(X_train2, y_train)

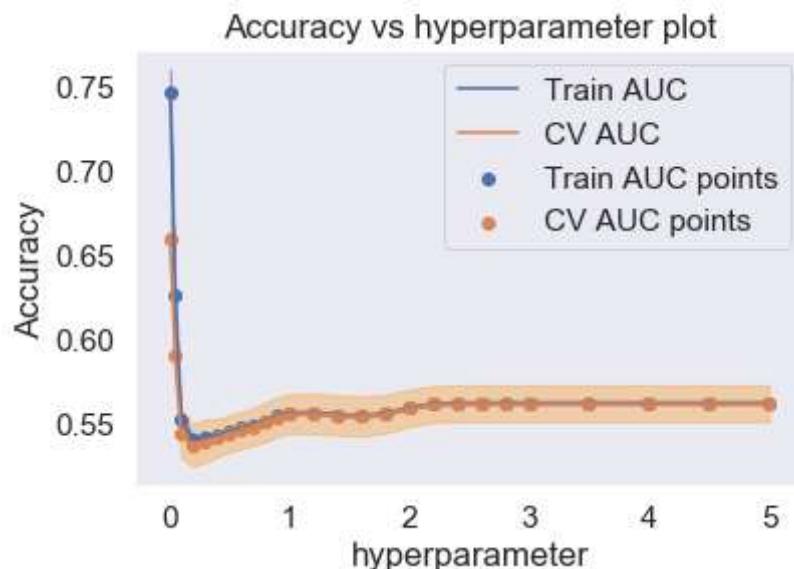
train_auc= classifier.cv_results_[ 'mean_train_score' ]
train_auc_std= classifier.cv_results_[ 'std_train_score' ]
cv_auc = classifier.cv_results_[ 'mean_test_score' ]
cv_auc_std= classifier.cv_results_[ 'std_test_score' ]

plt.plot(parameters[ 'alpha' ], train_auc, label='Train AUC')
#https://stackoverflow.com/questions/48796282/how-to-visualize-dependence-of-model-parameters-on-a-specific-metric
plt.gca().fill_between(parameters[ 'alpha' ],train_auc - train_auc_std,train_auc + train_auc_std, color="red", alpha=0.5)

plt.plot(parameters[ 'alpha' ], cv_auc, label='CV AUC')
plt.gca().fill_between(parameters[ 'alpha' ],cv_auc - cv_auc_std, cv_auc + cv_auc_std, color="blue", alpha=0.5)

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

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



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

```
In [142]: parameters = {'alpha':[0.01,0.05,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,1.2,1.4,1.6,1.8,2,2.2,2.4,2.6,2.8,3,3.2,3.4,3.6,3.8,4,4.2,4.4,4.6,4.8,5,5.2,5.4,5.6,5.8,6,6.2,6.4,6.6,6.8,7,7.2,7.4,7.6,7.8,8,8.2,8.4,8.6,8.8,9,9.2,9.4,9.6,9.8,10,10.2,10.4,10.6,10.8,11,11.2,11.4,11.6,11.8,12,12.2,12.4,12.6,12.8,13,13.2,13.4,13.6,13.8,14,14.2,14.4,14.6,14.8,15,15.2,15.4,15.6,15.8,16,16.2,16.4,16.6,16.8,17,17.2,17.4,17.6,17.8,18,18.2,18.4,18.6,18.8,19,19.2,19.4,19.6,19.8,20,20.2,20.4,20.6,20.8,21,21.2,21.4,21.6,21.8,22,22.2,22.4,22.6,22.8,23,23.2,23.4,23.6,23.8,24,24.2,24.4,24.6,24.8,25,25.2,25.4,25.6,25.8,26,26.2,26.4,26.6,26.8,27,27.2,27.4,27.6,27.8,28,28.2,28.4,28.6,28.8,29,29.2,29.4,29.6,29.8,30,30.2,30.4,30.6,30.8,31,31.2,31.4,31.6,31.8,32,32.2,32.4,32.6,32.8,33,33.2,33.4,33.6,33.8,34,34.2,34.4,34.6,34.8,35,35.2,35.4,35.6,35.8,36,36.2,36.4,36.6,36.8,37,37.2,37.4,37.6,37.8,38,38.2,38.4,38.6,38.8,39,39.2,39.4,39.6,39.8,40,40.2,40.4,40.6,40.8,41,41.2,41.4,41.6,41.8,42,42.2,42.4,42.6,42.8,43,43.2,43.4,43.6,43.8,44,44.2,44.4,44.6,44.8,45,45.2,45.4,45.6,45.8,46,46.2,46.4,46.6,46.8,47,47.2,47.4,47.6,47.8,48,48.2,48.4,48.6,48.8,49,49.2,49.4,49.6,49.8,50,50.2,50.4,50.6,50.8,51,51.2,51.4,51.6,51.8,52,52.2,52.4,52.6,52.8,53,53.2,53.4,53.6,53.8,54,54.2,54.4,54.6,54.8,55,55.2,55.4,55.6,55.8,56,56.2,56.4,56.6,56.8,57,57.2,57.4,57.6,57.8,58,58.2,58.4,58.6,58.8,59,59.2,59.4,59.6,59.8,60,60.2,60.4,60.6,60.8,61,61.2,61.4,61.6,61.8,62,62.2,62.4,62.6,62.8,63,63.2,63.4,63.6,63.8,64,64.2,64.4,64.6,64.8,65,65.2,65.4,65.6,65.8,66,66.2,66.4,66.6,66.8,67,67.2,67.4,67.6,67.8,68,68.2,68.4,68.6,68.8,69,69.2,69.4,69.6,69.8,70,70.2,70.4,70.6,70.8,71,71.2,71.4,71.6,71.8,72,72.2,72.4,72.6,72.8,73,73.2,73.4,73.6,73.8,74,74.2,74.4,74.6,74.8,75,75.2,75.4,75.6,75.8,76,76.2,76.4,76.6,76.8,77,77.2,77.4,77.6,77.8,78,78.2,78.4,78.6,78.8,79,79.2,79.4,79.6,79.8,80,80.2,80.4,80.6,80.8,81,81.2,81.4,81.6,81.8,82,82.2,82.4,82.6,82.8,83,83.2,83.4,83.6,83.8,84,84.2,84.4,84.6,84.8,85,85.2,85.4,85.6,85.8,86,86.2,86.4,86.6,86.8,87,87.2,87.4,87.6,87.8,88,88.2,88.4,88.6,88.8,89,89.2,89.4,89.6,89.8,90,90.2,90.4,90.6,90.8,91,91.2,91.4,91.6,91.8,92,92.2,92.4,92.6,92.8,93,93.2,93.4,93.6,93.8,94,94.2,94.4,94.6,94.8,95,95.2,95.4,95.6,95.8,96,96.2,96.4,96.6,96.8,97,97.2,97.4,97.6,97.8,98,98.2,98.4,98.6,98.8,99,99.2,99.4,99.6,99.8,100,100.2,100.4,100.6,100.8,101,101.2,101.4,101.6,101.8,102,102.2,102.4,102.6,102.8,103,103.2,103.4,103.6,103.8,104,104.2,104.4,104.6,104.8,105,105.2,105.4,105.6,105.8,106,106.2,106.4,106.6,106.8,107,107.2,107.4,107.6,107.8,108,108.2,108.4,108.6,108.8,109,109.2,109.4,109.6,109.8,110,110.2,110.4,110.6,110.8,111,111.2,111.4,111.6,111.8,112,112.2,112.4,112.6,112.8,113,113.2,113.4,113.6,113.8,114,114.2,114.4,114.6,114.8,115,115.2,115.4,115.6,115.8,116,116.2,116.4,116.6,116.8,117,117.2,117.4,117.6,117.8,118,118.2,118.4,118.6,118.8,119,119.2,119.4,119.6,119.8,120,120.2,120.4,120.6,120.8,121,121.2,121.4,121.6,121.8,122,122.2,122.4,122.6,122.8,123,123.2,123.4,123.6,123.8,124,124.2,124.4,124.6,124.8,125,125.2,125.4,125.6,125.8,126,126.2,126.4,126.6,126.8,127,127.2,127.4,127.6,127.8,128,128.2,128.4,128.6,128.8,129,129.2,129.4,129.6,129.8,130,130.2,130.4,130.6,130.8,131,131.2,131.4,131.6,131.8,132,132.2,132.4,132.6,132.8,133,133.2,133.4,133.6,133.8,134,134.2,134.4,134.6,134.8,135,135.2,135.4,135.6,135.8,136,136.2,136.4,136.6,136.8,137,137.2,137.4,137.6,137.8,138,138.2,138.4,138.6,138.8,139,139.2,139.4,139.6,139.8,140,140.2,140.4,140.6,140.8,141,141.2,141.4,141.6,141.8,142,142.2,142.4,142.6,142.8,143,143.2,143.4,143.6,143.8,144,144.2,144.4,144.6,144.8,145,145.2,145.4,145.6,145.8,146,146.2,146.4,146.6,146.8,147,147.2,147.4,147.6,147.8,148,148.2,148.4,148.6,148.8,149,149.2,149.4,149.6,149.8,150,150.2,150.4,150.6,150.8,151,151.2,151.4,151.6,151.8,152,152.2,152.4,152.6,152.8,153,153.2,153.4,153.6,153.8,154,154.2,154.4,154.6,154.8,155,155.2,155.4,155.6,155.8,156,156.2,156.4,156.6,156.8,157,157.2,157.4,157.6,157.8,158,158.2,158.4,158.6,158.8,159,159.2,159.4,159.6,159.8,160,160.2,160.4,160.6,160.8,161,161.2,161.4,161.6,161.8,162,162.2,162.4,162.6,162.8,163,163.2,163.4,163.6,163.8,164,164.2,164.4,164.6,164.8,165,165.2,165.4,165.6,165.8,166,166.2,166.4,166.6,166.8,167,167.2,167.4,167.6,167.8,168,168.2,168.4,168.6,168.8,169,169.2,169.4,169.6,169.8,170,170.2,170.4,170.6,170.8,171,171.2,171.4,171.6,171.8,172,172.2,172.4,172.6,172.8,173,173.2,173.4,173.6,173.8,174,174.2,174.4,174.6,174.8,175,175.2,175.4,175.6,175.8,176,176.2,176.4,176.6,176.8,177,177.2,177.4,177.6,177.8,178,178.2,178.4,178.6,178.8,179,179.2,179.4,179.6,179.8,180,180.2,180.4,180.6,180.8,181,181.2,181.4,181.6,181.8,182,182.2,182.4,182.6,182.8,183,183.2,183.4,183.6,183.8,184,184.2,184.4,184.6,184.8,185,185.2,185.4,185.6,185.8,186,186.2,186.4,186.6,186.8,187,187.2,187.4,187.6,187.8,188,188.2,188.4,188.6,188.8,189,189.2,189.4,189.6,189.8,190,190.2,190.4,190.6,190.8,191,191.2,191.4,191.6,191.8,192,192.2,192.4,192.6,192.8,193,193.2,193.4,193.6,193.8,194,194.2,194.4,194.6,194.8,195,195.2,195.4,195.6,195.8,196,196.2,196.4,196.6,196.8,197,197.2,197.4,197.6,197.8,198,198.2,198.4,198.6,198.8,199,199.2,199.4,199.6,199.8,200,200.2,200.4,200.6,200.8,201,201.2,201.4,201.6,201.8,202,202.2,202.4,202.6,202.8,203,203.2,203.4,203.6,203.8,204,204.2,204.4,204.6,204.8,205,205.2,205.4,205.6,205.8,206,206.2,206.4,206.6,206.8,207,207.2,207.4,207.6,207.8,208,208.2,208.4,208.6,208.8,209,209.2,209.4,209.6,209.8,210,210.2,210.4,210.6,210.8,211,211.2,211.4,211.6,211.8,212,212.2,212.4,212.6,212.8,213,213.2,213.4,213.6,213.8,214,214.2,214.4,214.6,214.8,215,215.2,215.4,215.6,215.8,216,216.2,216.4,216.6,216.8,217,217.2,217.4,217.6,217.8,218,218.2,218.4,218.6,218.8,219,219.2,219.4,219.6,219.8,220,220.2,220.4,220.6,220.8,221,221.2,221.4,221.6,221.8,222,222.2,222.4,222.6,222.8,223,223.2,223.4,223.6,223.8,224,224.2,224.4,224.6,224.8,225,225.2,225.4,225.6,225.8,226,226.2,226.4,226.6,226.8,227,227.2,227.4,227.6,227.8,228,228.2,228.4,228.6,228.8,229,229.2,229.4,229.6,229.8,230,230.2,230.4,230.6,230.8,231,231.2,231.4,231.6,231.8,232,232.2,232.4,232.6,232.8,233,233.2,233.4,233.6,233.8,234,234.2,234.4,234.6,234.8,235,235.2,235.4,235.6,235.8,236,236.2,236.4,236.6,236.8,237,237.2,237.4,237.6,237.8,238,238.2,238.4,238.6,238.8,239,239.2,239.4,239.6,239.8,240,240.2,240.4,240.6,240.8,241,241.2,241.4,241.6,241.8,242,242.2,242.4,242.6,242.8,243,243.2,243.4,243.6,243.8,244,244.2,244.4,244.6,244.8,245,245.2,245.4,245.6,245.8,246,246.2,246.4,246.6,246.8,247,247.2,247.4,247.6,247.8,248,248.2,248.4,248.6,248.8,249,249.2,249.4,249.6,249.8,250,250.2,250.4,250.6,250.8,251,251.2,251.4,251.6,251.8,252,252.2,252.4,252.6,252.8,253,253.2,253.4,253.6,253.8,254,254.2,254.4,254.6,254.8,255,255.2,255.4,255.6,255.8,256,256.2,256.4,256.6,256.8,257,257.2,257.4,257.6,257.8,258,258.2,258.4,258.6,258.8,259,259.2,259.4,259.6,259.8,260,260.2,260.4,260.6,260.8,261,261.2,261.4,261.6,261.8,262,262.2,262.4,262.6,262.8,263,263.2,263.4,263.6,263.8,264,264.2,264.4,264.6,264.8,265,265.2,265.4,265.6,265.8,266,266.2,266.4,266.6,266.8,267,267.2,267.4,267.6,267.8,268,268.2,268.4,268.6,268.8,269,269.2,269.4,269.6,269.8,270,270.2,270.4,270.6,270.8,271,271.2,271.4,271.6,271.8,272,272.2,272.4,272.6,272.8,273,273.2,273.4,273.6,273.8,274,274.2,274.4,274.6,274.8,275,275.2,275.4,275.6,275.8,276,276.2,276.4,276.6,276.8,277,277.2,277.4,277.6,277.8,278,278.2,278.4,278.6,278.8,279,279.2,279.4,279.6,279.8,280,280.2,280.4,280.6,280.8,281,281.2,281.4,281.6,281.8,282,282.2,282.4,282.6,282.8,283,283.2,283.4,283.6,283.8,284,284.2,284.4,284.6,284.8,285,285.2,285.4,285.6,285.8,286,286.2,286.4,286.6,286.8,287,287.2,287.4,287.6,287.8,288,288.2,288.4,288.6,288.8,289,289.2,289.4,289.6,289.8,290,290.2,290.4,290.6,290.8,291,291.2,291.4,291.6,291.8,292,292.2,292.4,292.6,292.8,293,293.2,293.4,293.6,293.8,294,294.2,294.4,294.6,294.8,295,295.2,295.4,295.6,295.8,296,296.2,296.4,296.6,296.8,297,297.2,297.4,297.6,297.8,298,298.2,298.4,298.6,298.8,299,299.2,299.4,299.6,299.8,300,300.2,300.4,300.6,300.8,301,301.2,301.4,301.6,301.8,302,302.2,302.4,302.6,302.8,303,303.2,303.4,303.6,303.8,304,304.2,304.4,304.6,304.8,305,305.2,305.4,305.6,305.8,306,306.2,306.4,306.6,306.8,307,307.2,307.4,307.6,307.8,308,308.2,308.4,308.6,308.8,309,309.2,309.4,309.6,309.8,310,310.2,310.4,310.6,310.8,311,311.2,311.4,311.6,311.8,312,312.2,312.4,312.6,312.8,313,313.2,313.4,313.6,313.8,314,314.2,314.4,314.6,314.8,315,315.2,315.4,315.6,315.8,316,316.2,316.4,316.6,316.8,317,317.2,317.4,317.6,317.8,318,318.2,318.4,318.6,318.8,319,319.2,319.4,319.6,319.8,320,320.2,320.4,320.6,320.8,321,321.2,321.4,321.6,321.8,322,322.2,322.4,322.6,322.8,323,323.2,323.4,323.6,323.8,324,324.2,324.4,324.6,324.8,325,325.2,325.4,325.6,325.8,326,326.2,326.4,326.6,326.8,327,327.2,327.4,327.6,327.8,328,328.2,328.4,328.6,328.8,329,329.2,329.4,329.6,329.8,330,330.2,330.4,330.6,330.8,331,331.2,331.4,331.6,331.8,332,332.2,332.4,332.6,332.8,333,333.2,333.4,333.6,333.8,334,334.2,334.4,334.6,334.8,335,335.2,335.4,335.6,335.8,336,336.2,336.4,336.6,336.8,337,337.2,337.4,337.6,337.8,338,338.2,338.4,338.6,338.8,339,339.2,339.4,339.6,339.8,340,340.2,340.4,340.6,340.8,341,341.2,341.4,341.6,341.8,342,342.2,342.4,342.6,342.8,343,343.2,343.4,343.6,343.8,344,344.2,344.4,344.6,344.8,345,345.2,345.4,345.6,345.8,346,346.2,346.4,346.6,346.8,347,347.2,347.4,347.6,347.8,348,348.2,348.4,348.6,348.8,349,349.2,349.4,349.6,349.8,350,350.2,350.4,350.6,350.8,351,351.2,351.4,351.6,351.8,352,352.2,352.4,352.6,352.8,353,353.2,353.4,353.6,353.8,354,354.2,354.4,354.6,354.8,355,355.2,355.4,355.6,355.8,356,356.2,356.4,356.6,356.8,357,357.2,357.4,357.6,357.8,358,358.2,358.4,358.6,358.8,359,359.2,359.4,359.6,359.8,360,360.2,360.4,360.6,360.8,361,361.2,361.4,361.6,361.8,362,362.2,362.4,362.6,362.8,363,363.2,363.4,363.6,363.8,364,364.2,364.4,364.6,364.8,365,365.2,365.4,365.6,365.8,366,366.2,366.4,366.6,366.8,367,367.2,367.4,367.6,367.8,368,368.2,368.4,368.6,368.8,369,369.2,369.4,369.6,369.8,370,370.2,370.4,370.6,370.8,371,371.2,371.4,371.6,371.8,372,372.2,372.4,372.6,372.8,373,373.2,373.4,373.6,373.8,374,374.2,374.4,374.6,374.8,375,375.2,375.4,375.6,375.8,376,376.2,376.4,376.6,376.8,377,377.2,377.4,377.6,377.8,378,378.2,378.4,378.6,378.8,379,379.2,379.4,379.6,379.8,380,380.2,380.4,380.6,380.8,381,381.2,381.4,381.6,381.8,382,382.2,382.4,382.6,382.8,383,383.2,383.4,383.6,383.8,384,384.2,384.4,384.6,384.8,385,385.2,385.4,385.6,385.8,386,386.2,386.4,386.6,386.8,387,387.2,387.4,387.6,387.8,388,388.2,388.4,388.6,388.8,389,389.2,389.4,389.6,389.8,390,390.2,390.4,390.6,390.8,391,391.2,391.4,391.6,391.8,392,392.2,392.4,392.6,392.8,393,393.2,393.4,393.6,393.8,394,394.2,394.4,394.6,394.8,395,395.2,395.4,395.6,395.8,396,396.2,396.4,396.6,396.8,397,397.2,397.4,397.6,397.8,398,398.2,398.4,398.6,398.8,399,399.2,399.4,399.6,399.8,400,400.2,400.4,400.6,400.8,401,401.2,401.4,401.6,401.8,402,402.2,402.4,402.6,402.8,403,403.2,403.4,403.6,403.8,404,404.2,404.4,404.6,404.8,405,405.2,405.4,405.6,405.8,406,406.2,406.4,406.6,406.8,407,407.2,407.4,407.6,407.8,408,408.2,408.4,408.6,408.8,409,409.2,409.4,409.6,409.8,410,410.2,410.4,410.6,410.8,411,411.2,411.4,411.6,411.8,412,412.2,412.4,412.6,412.8,413,413.2,413.4,413.6,413.8,414,414.2,414.4,414.6,414.8,415,415.2,415.4,415.6,415.8,416,416.2,416.4,416.6,416.8,417,417.2,417.4,417.6,417.8,418,418.2,418.4,418.6,418.8,419,419.2,419.4,419.6,419.8,420,420.2,420.4,420.6,420.8,421,421.2,421.4,4
```

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

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

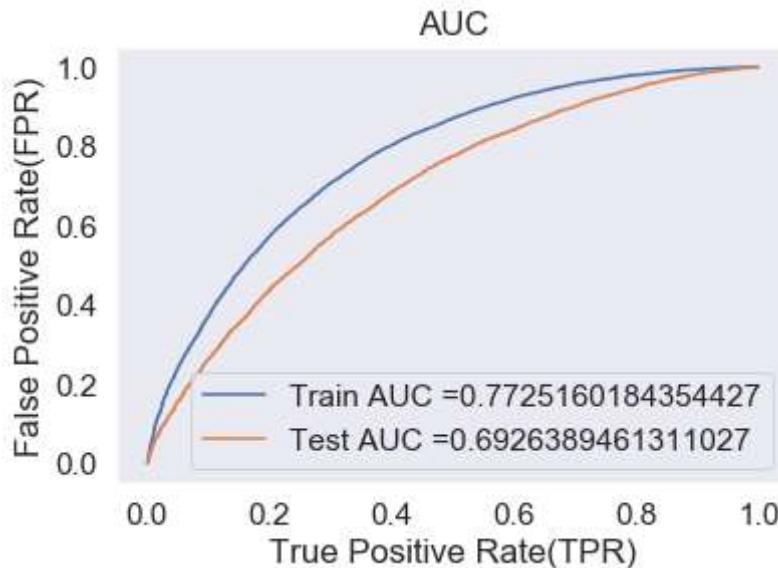
Classifier_tfidf = SGDClassifier(loss = 'hinge', penalty = 'l2', alpha = 0.001)

Classifier_tfidf.fit(X_train2, y_train)

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

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

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



Train Confusing Matrix

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

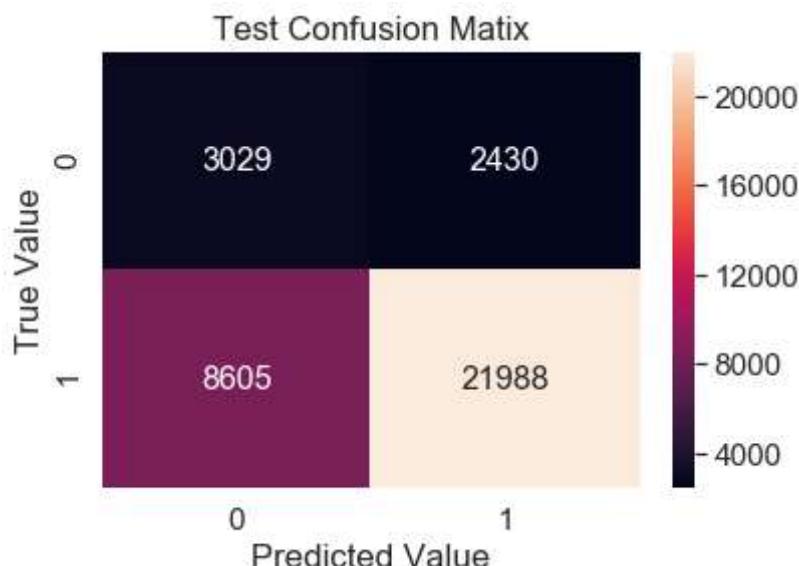
```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.985
[[ 3713  3713]
 [ 6385 35230]]
```

```
In [148]: train_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)), columns=['Predicted Value'], index=['True Value'])

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) 0.24999999161092998 for threshold 1.014

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



Test Confusion Matrix

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

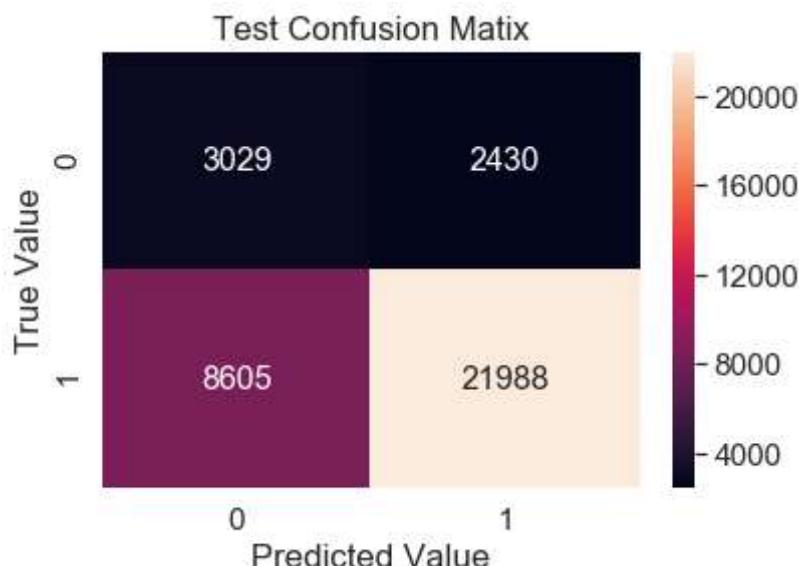
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 1.014

| | | Predicted Value |
|------------|------|-----------------|
| True Value | 0 | 1 |
| | 0 | 3029 |
| 1 | 8605 | 21988 |

```
In [150]: train_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred),test_fpr,test_tpr))
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) 0.24999999161092998 for threshold 1.014

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



In []:

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

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

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

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

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

```
In [219]: X_cv3 = hstack((cv_categories_one_hot, cv_sub_categories_one_hot, cv_state_one_hot_
    cv_teacher_one_hot, cv_title_avg_w2v_vectors, cv_avg_w2v_vectors,
    cv_prev_proj_standar, cv_price_standar, cv_positive_standar, cv_
    cv_neutral_standar, cv_title_word_count_standar, cv_essay_word_
print(X_cv3.shape, y_cv.shape)
print(type(X_cv3))
```

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

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

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

Hyperparameter Tuning

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

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

classifier.fit(X_train3, y_train)

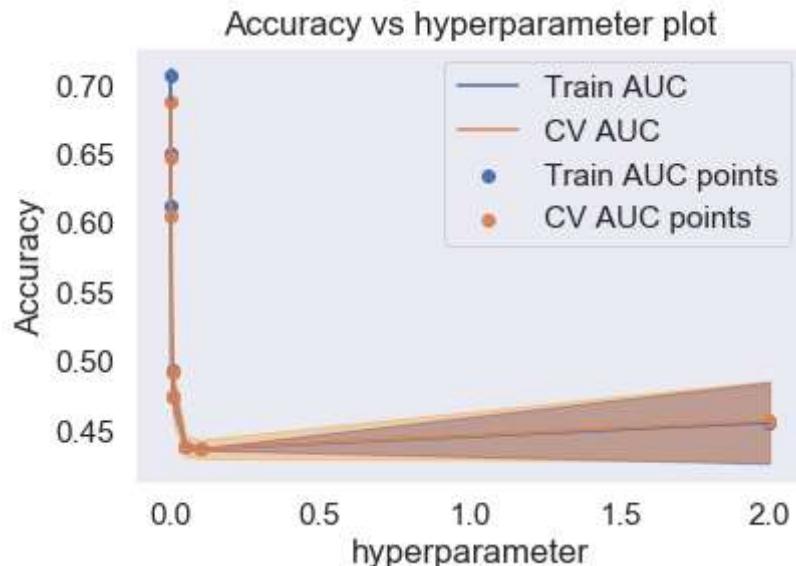
train_auc= classifier.cv_results_['mean_train_score']
train_auc_std= classifier.cv_results_['std_train_score']
cv_auc = classifier.cv_results_['mean_test_score']
cv_auc_std= classifier.cv_results_['std_test_score']

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
#https://stackoverflow.com/questions/48796282/how-to-visualize-dependence-of-model-parameters-on-a-specific-feature
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc + train_auc_std, color="red", alpha=0.5)

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std, cv_auc + cv_auc_std, color="blue", alpha=0.5)

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

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



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

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

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

classifier.fit(X_train3, y_train)

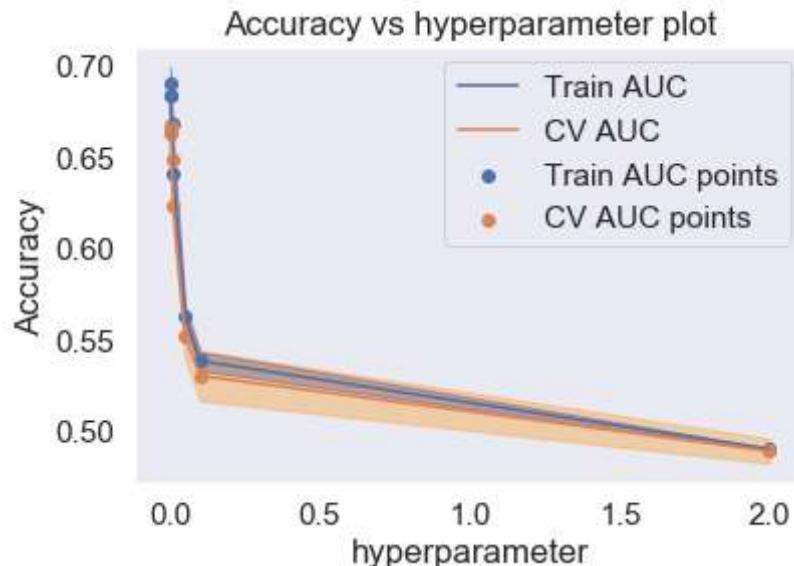
train_auc= classifier.cv_results_['mean_train_score']
train_auc_std= classifier.cv_results_['std_train_score']
cv_auc = classifier.cv_results_['mean_test_score']
cv_auc_std= classifier.cv_results_['std_test_score']

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
#https://stackoverflow.com/questions/48796282/how-to-visualize-dependence-of-mode
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc + train_auc_std, color="red", alpha=0.5)

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std, cv_auc + cv_auc_std, color="blue", alpha=0.5)

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

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



L2 Regularizer Seems better Auccaracy than L1 Regualrizer

Training Model Using best Hyperparameter

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

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

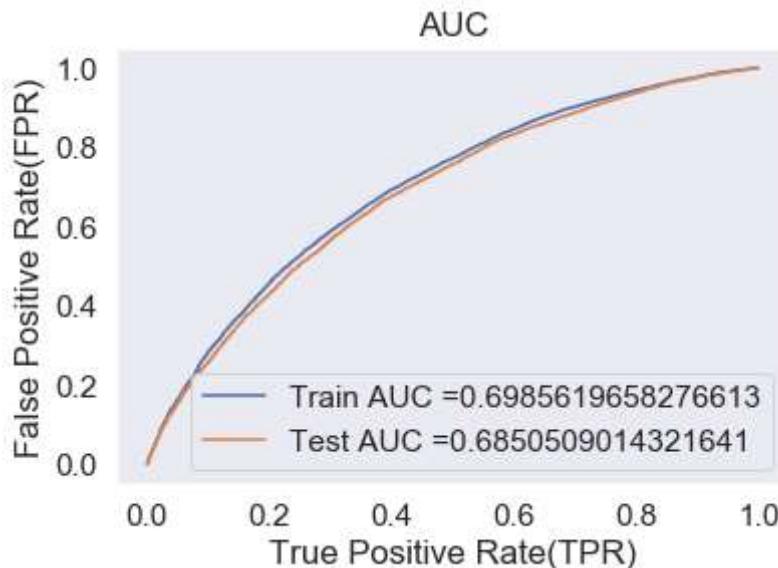
Classifier_avgw2v = SGDClassifier(loss = 'hinge', penalty = 'l2', alpha = 0.0001)

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

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

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

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



Train Confusion Matrix

```
In [177]: from sklearn.metrics import confusion_matrix
import seaborn as sea
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, test_fpr, test_thresholds), labels=[0, 1]))
```

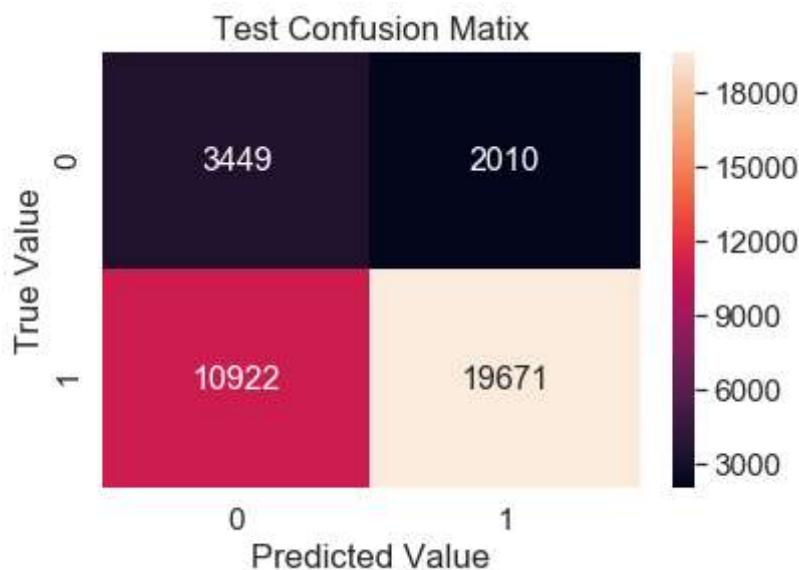
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 2.03
[[3713 3713]
[9501 32114]]

```
In [178]: train_confusion_matrix = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, train_fpr, test_fpr, test_thresholds), labels=[0, 1]), columns=['Predicted Value'], index=['True Value'])
```

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) 0.24999999161092995 for threshold 2.524

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



Test Confusion Matrix

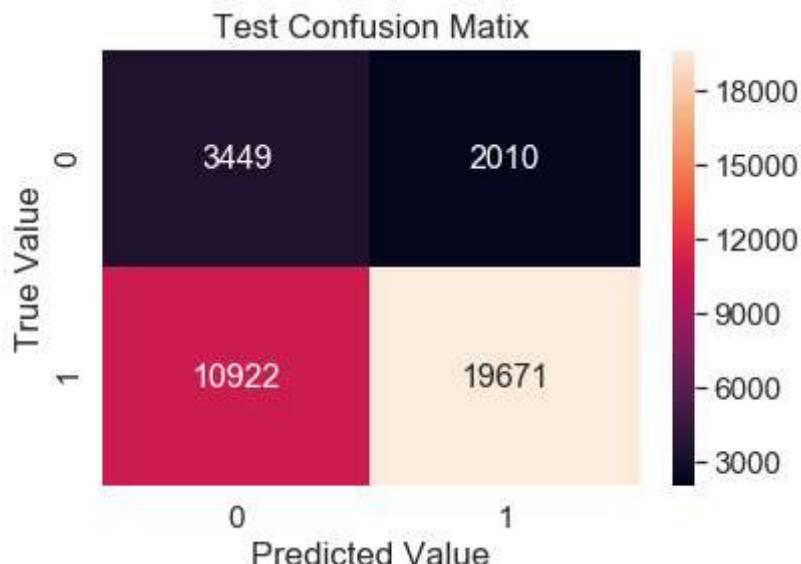
```
In [179]: print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_thresholds), labels=[0, 1]))
```

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

```
In [180]: train_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred),test_fpr,test_tpr))
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) 0.24999999161092995 for threshold 2.524

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



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

```
In [189]: from scipy.sparse import hstack
# with the same hstack function we are concatenating a sparse matrix and a dense
X_train3 = hstack((train_categories_one_hot,train_sub_categories_one_hot,train_size_standar,
                   train_teacher_one_hot, train_title_tfidf_w2v_vectors, train_email,
                   train_quantity_standar, train_prev_proj_standar, train_price_standar,
                   train_negitive_standar, train_neutral_standar, train_title_word_count_standar)).tocsr()
print(X_train3.shape, y_train.shape)
print(type(X_train3)) #train_title_tfidf_w2v_vectors train_email_tfidf_w2v_vector
```

(49041, 708) (49041,)

<class 'scipy.sparse.csr.csr_matrix'>

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

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

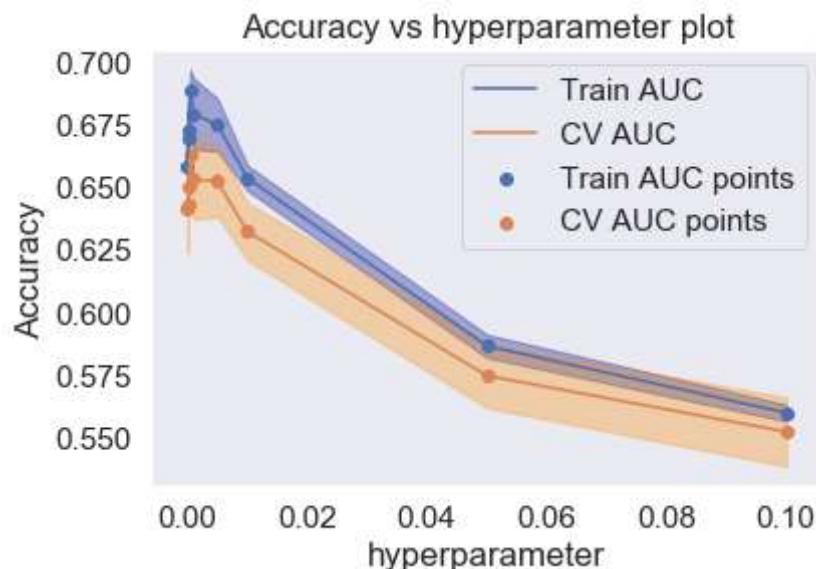
```
In [194]: X_cv3 = hstack((cv_categories_one_hot, cv_sub_categories_one_hot, cv_state_one_h_
    cv_teacher_one_hot, cv_title_tfidf_w2v_vectors, cv_essay_tfid_
    cv_quantity_standar, cv_prev_proj_standar, cv_price_standar,c_
    cv_negitive_standar, cv_neutral_standar, cv_title_word_count_
    cv_essay_word_count_standar)).tocsr()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

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

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

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

Hyperparameter Tunning



L1 Regularizer

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

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

classifier.fit(X_train3, y_train)

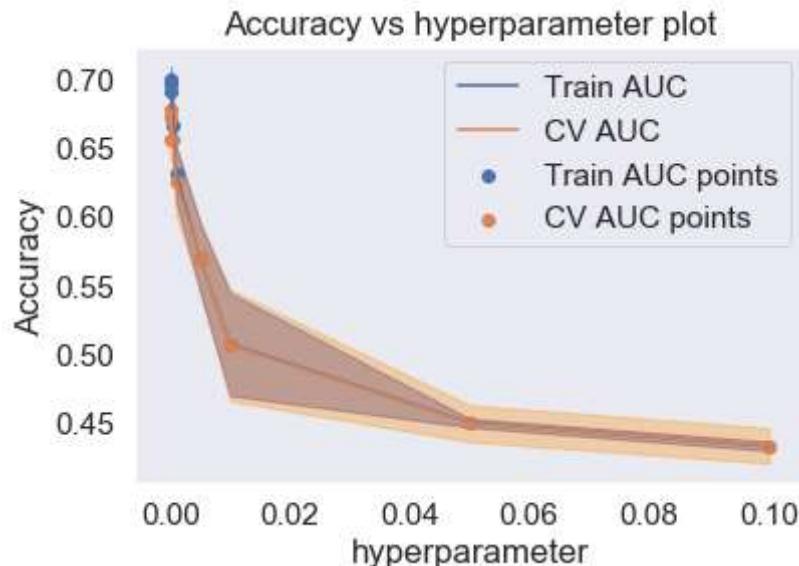
train_auc= classifier.cv_results_[ 'mean_train_score']
train_auc_std= classifier.cv_results_[ 'std_train_score']
cv_auc = classifier.cv_results_[ 'mean_test_score']
cv_auc_std= classifier.cv_results_[ 'std_test_score']

plt.plot(parameters[ 'alpha'], train_auc, label='Train AUC')
#https://stackoverflow.com/questions/48796282/how-to-visualize-dependence-of-mode
plt.gca().fill_between(parameters[ 'alpha'],train_auc - train_auc_std,train_auc + train_auc_std, color="red", alpha=0.5)

plt.plot(parameters[ 'alpha'], cv_auc, label='CV AUC')
plt.gca().fill_between(parameters[ 'alpha'],cv_auc - cv_auc_std, cv_auc + cv_auc_std, color="blue", alpha=0.5)

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

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



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

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

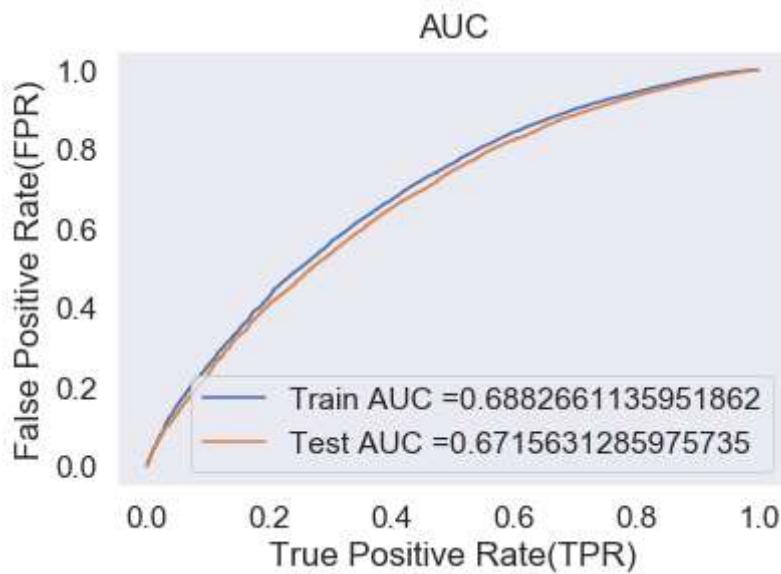
Classifier_tfidf2v = SGDClassifier(loss = 'hinge', penalty = 'l2', alpha = 0.0001,
                                    max_iter=1000, random_state=42)

Classifier_tfidf2v.fit(X_train3, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates
# not the predicted outputs

y_train_pred = Classifier_tfidf2v.decision_function(X_train3)
y_test_pred = Classifier_tfidf2v.decision_function(X_test3)

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

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



Train Confusion Matrix

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

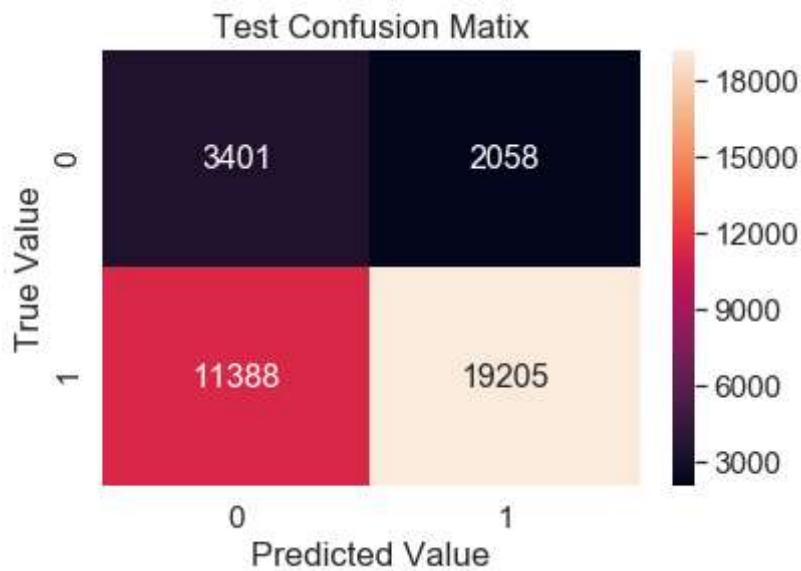
```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.746
[[ 3713  3713]
 [ 9845 31770]]
```

```
In [205]: train_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)), columns=['Predicted Value'], index=['True Value'])

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) 0.24999999161092998 for threshold 0.885

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



Test Confusion Matrix

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

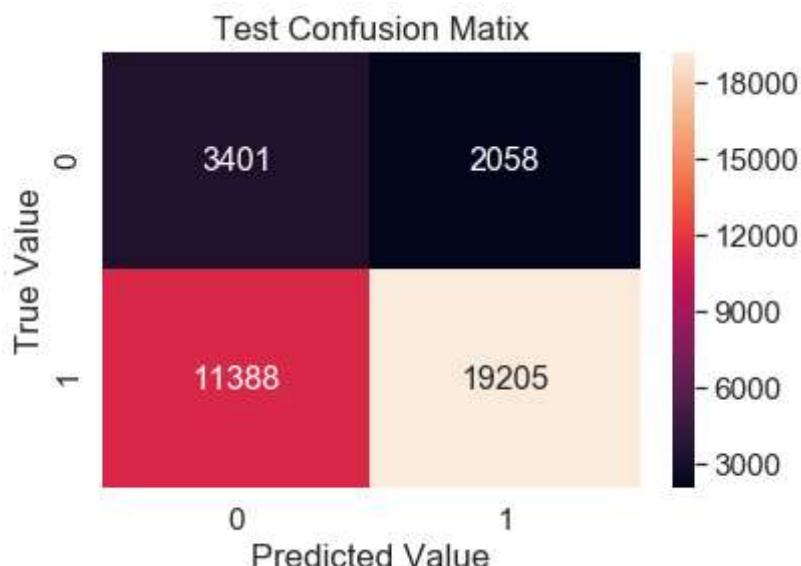
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.885

| | | Predicted Value 0 | Predicted Value 1 |
|--------------|--------------|-------------------|-------------------|
| True Value 0 | 3401 | 2058 | |
| | True Value 1 | 11388 | 19205 |

```
In [207]: train_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred),test_fpr,test_tpr))
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) 0.24999999161092998 for threshold 0.885

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



2.5 Apply Logistic Regression on Set-5

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

Consider these set of features for Set 5 in Assignment:

categorical data school_state

clean_categories....clean_subcategories....project_grade_category....teacher_prefix

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

New Features:-

sentiment score's of each of the essay : numerical data

number of words in the title : numerical data

number of words in the combine essays : numerical data

```
In [102]: from scipy.sparse import hstack
X_train3 = hstack((train_categories_one_hot, train_sub_categories_one_hot, train_
                   train_teacher_one_hot, train_quantity_standar, train_prev_pro_
                   train_title_word_count_standar, train_essay_word_count_standar_
                   train_negitive_standar, train_neutral_standar)).tocsr()
print(X_train3.shape, y_train.shape)
print(type(X_train3))
```

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

```
In [103]: X_test3 = hstack((test_categories_one_hot, test_sub_categories_one_hot, test_gra_
                           test_teacher_one_hot, test_quantity_standar, test_prev_proj_s_
                           test_title_word_count_standar, test_essay_word_count_standar,
                           test_negitive_standar, test_neutral_standar)).tocsr()
print(X_test3.shape, y_train.shape)
print(type(X_test3))
```

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

```
In [104]: X_cv3 = hstack((cv_categories_one_hot, cv_sub_categories_one_hot, cv_grade_one_ho_
                           cv_teacher_one_hot, cv_quantity_standar, cv_prev_proj_standar,
                           cv_title_word_count_standar, cv_essay_word_count_standar, cv_
                           cv_negitive_standar, cv_neutral_standar)).tocsr()
print(X_cv3.shape, y_train.shape)
print(type(X_cv3))
```

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

```
In [199]: y_trainn = y_train[0:24155,]
print(y_trainn.shape)
```

```
(24155,)
```

Hyperparameter Tunning

```
In [224]: parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4, 10**5, 10**6, 10**7, 10**8, 10**9, 10**10]}

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

classifier.fit(X_train3, y_train)

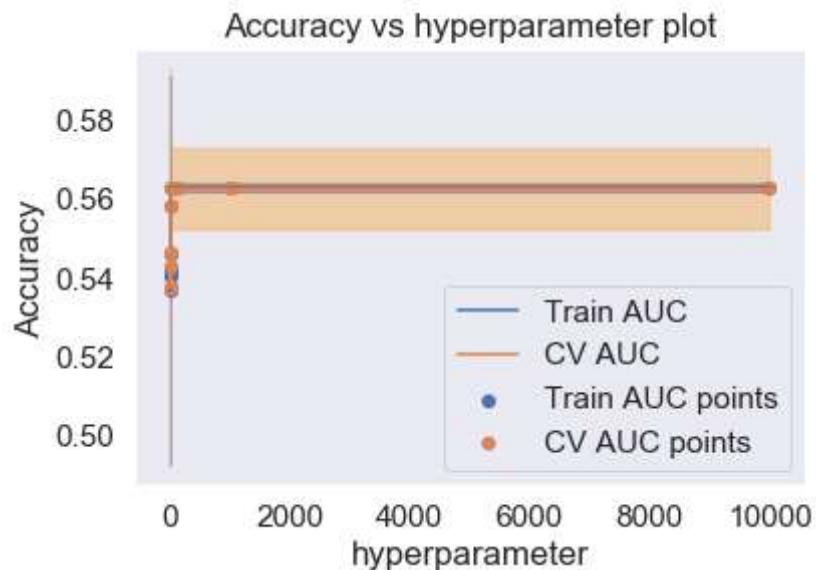
train_auc= classifier.cv_results_['mean_train_score']
train_auc_std= classifier.cv_results_['std_train_score']
cv_auc = classifier.cv_results_['mean_test_score']
cv_auc_std= classifier.cv_results_['std_test_score']

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc + train_auc_std, alpha=0.5)

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std, cv_auc + cv_auc_std, alpha=0.5)

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

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



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

```
In [30]: print(train_text_tfidf.shape)
#There are 12132 dimensions here

(49041, 12132)
```

```
In [ ]: from sklearn.decomposition import TruncatedSVD
#https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.Truncate
#declaring index as Dimensions in train_text_tfidf
Dim = [1000,2000,3000,4000,5000,6000,7000,8000,9000,10000]
Varience = []
for i in tqdm(Dim):
    svd = TruncatedSVD(n_components = i, random_state = 42)
    svd.fit(train_text_tfidf)
    Varience.append(svd.explained_variance_ratio_)

50%|██████████| 10/20 [00:00:00<00:00:00, 1096.14s/it]
```

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

```
In [ ]: #Train SVD
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components= 5000, n_iter=7, random_state=42)
svd.fit(train_text_tfidf)
svd_train = svd.transform(train_text_tfidf)
```

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

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

Grid Search

```
In [ ]: parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4, 10**5]}

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

classifier.fit(X_train2, y_train)

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

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc + train_auc_std, alpha=0.5)

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std, cv_auc + cv_auc_std, alpha=0.5)

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

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

In []:

```
In [204]: best_c_5 = 0.5
```

Train Confusion Matrix

```
In [ ]: 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,
```

```
In [ ]: train_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred),test_fpr,test_fnr))
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")
```

Test Confusion Matrix

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

```
In [ ]: train_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred)), columns = ["Predicted Value"], index = ["True Value"])
test_fpr, test_fnr, test_tpr, test_fnr = metrics.confusion_matrix(y_test, predict(y_test_pred))
train_fpr, train_fnr, train_tpr, train_fnr = metrics.confusion_matrix(y_train, predict(y_train_pred))

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")
```

3. Conclusion

```
In [3]: # Please compare all your models using Prettytable library
# http://zetcode.com/python/prettytable/
```

```
from prettytable import PrettyTable
TB = PrettyTable()
TB.field_names = ["Vectorizer", "C:Hyperparameter", "Regularizer", "Train_AUC", "Test_Auc"]
TB.title = "Support Vector Regression"
TB.add_row(["BOW-Model", 0.01, "L1", 0.78, 0.68])
TB.add_row(["TFIDF-Model", 0.001, "L1", 0.77, 0.69])
TB.add_row(["AvgW2v-Model", 0.0005, "L2", 0.69, 0.68])
TB.add_row(["Tf-Idf-Model", 0.0002, "L1", 0.68, 0.67])
TB.add_row(["NUM_Features-Model", 2000, "L2", 0.63, 0.61])
print(TB)
```

| Vectorizer | C:Hyperparameter | Regularizer | Train_AUC | Test_Auc |
|--------------------|------------------|-------------|-----------|----------|
| BOW-Model | 0.01 | L1 | 0.78 | 0.68 |
| TFIDF-Model | 0.001 | L1 | 0.77 | 0.69 |
| AvgW2v-Model | 0.0005 | L2 | 0.69 | 0.68 |
| Tf-Idf-Model | 0.0002 | L1 | 0.68 | 0.67 |
| NUM_Features-Model | 2000 | L2 | 0.63 | 0.61 |

```
In [ ]:
```