```
%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
```

```
#getting the file from google drive (test data)
import gdown

url = 'https://drive.google.com/uc?id=1bDLwb_Vq7q2W9S89JB96PgmZG3LsLns9'
output = 'train.csv'
# https://drive.google.com/file/d/1bDLwb_Vq7q2W9S89JB96PgmZG3LsLns9/view?usp=sharing
gdown.download(url, output, quiet=False)
```

```
Downloading...
From: https://drive.google.com/uc?id=1bDLwb_Vq7q2W9S89JB96PgmZG3LsLns9
To: /content/train.csv
201MB [00:02, 77.1MB/s]
'train.csv'
```

```
#getting the data from google drive (resources data)resources data
import gdown
url = 'https://drive.google.com/uc?id=14OVXWu_SJU-lJD-jKMOCld14EZ21lYYe'
output = 'resources.csv'
gdown.download(url, output, quiet=False)

```

```
Downloading...
From: https://drive.google.com/uc?id=14OVXWu_SJU-lJD-jKMOCld14EZ21lYYe
To: /content/resources.csv
127MB [00:00, 73.4MB/s]
'resources.csv'
```

```
dft = pd.read_csv('train.csv',nrows=50000)
dfr = pd.read_csv('resources.csv')
```

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

```
Number of data points in train data (50000, 17)
```

```
# sort the datapoints by date and time column
# list comprehension python :# https://stackoverflow.com/a/2582163/4084039
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(dft.columns)]
#sort dataframe based on time  uisng pandas to_datetime function  : https://stackoverflow.com/a/49702492/4084039
dft['Date'] = pd.to_datetime(dft['project_submitted_datetime'])
dft.drop('project_submitted_datetime', axis=1, inplace=True)# we drop the col
dft.sort_values(by=['Date'], inplace=True)# sort the values y date
dft.head(2)
```

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_grade_category	projec
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	Grades PreK-2	
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	Grades 3-5	

1.1 Text preprocessing

```
# merge two column text dataframe:
dft["essay"] = dft["project_essay_1"].map(str) +\
               dft["project_essay_2"].map(str) + \
               dft["project_essay_3"].map(str) + \
               dft["project_essay_4"].map(str)
```

```
# 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
```

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list so as to get btter prediction : that is , no , not ,etc .
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

Preprocessing of the **project_subject_categories**

```
categories = list(dft['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47301924/4084039
# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
cat_list = []
for i in categories:
    temp = ""
    for j in i.split(','):
        if 'The' in j.split():
            j=j.replace('The','')
        j = j.replace(' ','')
        temp+=j.strip()+" "
        temp = temp.replace('&','_')
    cat_list.append(temp.strip())

```

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

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

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

```
{'Warmth': 643, 'Care_Hunger': 643, 'History_Civics': 2689, 'Music_Arts': 4699, 'AppliedLearning': 5569, 'SpecialNeeds'
```

preprocessing of the **project_subject_subcategories**

```
sub_catogories = list(dft['project_subject_subcategories'].values)
# # remove special characters from list of strings python: https://stackoverflow.com/a/47301924/4084039
# # https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# # https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# # https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python

sub_cat_list = []
for i in sub_catogories:
    temp = ""
    for j in i.split(','):
        if 'The' in j.split():
            j=j.replace('The','')
        j = j.replace(' ','')
        temp +=j.strip()+" "
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

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

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

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

```
{'AppliedSciences': 4901,
 'Care_Hunger': 643,
 'CharacterEducation': 958,
 'Civics_Government': 380,
 'College_CareerPrep': 1168,
 'CommunityService': 214,
 'ESL': 1999,
 'EarlyDevelopment': 1937,
 'Economics': 127,
 'EnvironmentalScience': 2533,
 'Extracurricular': 373,
 'FinancialLiteracy': 253,
 'ForeignLanguages': 388,
 'Gym_Fitness': 2068,
 'Health_LifeScience': 1876,
 'Health_Wellness': 4732,
 'History_Geography': 1433,
 'Literacy': 15611,
 'Literature_Writing': 10127,
 'Mathematics': 12832,
 'Music': 1432,
 'NutritionEducation': 617,
 'Other': 1128,
 'ParentInvolvement': 302,
 'PerformingArts': 910,
 'SocialSciences': 864,
 'SpecialNeeds': 6233,
 'TeamSports': 995,
 'VisualArts': 2865,
 'Warmth': 643}
```

preprocessing of the **project_grade_category**

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

```
dft['clean_grade'] = grade_cat_list
dft.drop(['project_grade_category'], axis=1, inplace=True)

my_counter = Counter()
for word in dft['clean_grade'].values:
    my_counter.update(word.split())
project_grade_category_dict= dict(my_counter)
sorted_project_grade_category_dict = dict(sorted(project_grade_category_dict.items(), key=lambda kv: kv[1]))
sorted_project_grade_category_dict
```

```
{'grades_3_5': 16968,
 'grades_6_8': 7750,
 'grades_9_12': 4966,
 'grades_prek_2': 20316}
```

Preparing data matrix for the models for Model

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(dft,dft['project_is_approved'],stratify= dft['project_is_approved'],test_size
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify= y_train,test_size = 0.33)
```

```
print("the shape of the Y_train data ", y_train.value_counts())
print("&"*25)
print("the shape of the Y_test data ",y_test.value_counts())
print("&"*25)
print("the shape of the Y_cv data ",y_cv.value_counts())
print("&"*25)
print("the above representaion shows a huge imbalance in data ")
```

```
the shape of the Y_train data  1    18982
0     3463
Name: project_is_approved, dtype: int64
&&&&&&&&&&&&&&&&&&&&&&&&&
the shape of the Y_test data  1    13954
0     2546
Name: project_is_approved, dtype: int64
&&&&&&&&&&&&&&&&&&&&&&&&&
the shape of the Y_cv data  1    9350
0    1705
Name: project_is_approved, dtype: int64
&&&&&&&&&&&&&&&&&&&&&&&&&
the above representaion shows a huge imbalance in data
```

```
# # Dropping the target variable coloums
X_train.drop(["project_is_approved"], axis = 1, inplace = True)
#x_test =
X_test.drop(["project_is_approved"], axis = 1, inplace = True)
#x_cv =
X_cv.drop(["project_is_approved"], axis = 1, inplace = True)

```

Pre-processing the Text Features ,

here we are independtenly doing the preprocessing as i observed that if we pre-processes and then divide the data , it is casuing in lower AUC score

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

```

```
100%|██████████| 22445/22445 [00:13<00:00, 1712.52it/s]
```

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

```
100%|██████████| 16500/16500 [00:10<00:00, 1540.34it/s]
```

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

```

```
100%|██████████| 11055/11055 [00:06<00:00, 1712.96it/s]
```

```
#Proprocessing for essay
# Combining all the above students
from tqdm import tqdm
preprocessed_titles_train = []
# tqdm is for printing the status bar
for sentance in tqdm(X_train['project_title'].values):
    sent = decontracted(sentance)
```

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

```

```
100%|██████████| 22445/22445 [00:00<00:00, 39737.28it/s]
```

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

```
100%|██████████| 11055/11055 [00:00<00:00, 39204.44it/s]
```

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

```
100%|██████████| 16500/16500 [00:00<00:00, 39630.83it/s]
```

Vectorizinig the Numerical and categorical data

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

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



from sklearn.feature_extraction.text import CountVectorizer
vectorizer1 = CountVectorizer(vocabulary=list(project_subject_categories_dict.keys()), lowercase=False,binary=True)
vectorizer1.fit(X_train['clean_categories'].values)
X_train_cat = vectorizer1.transform(X_train['clean_categories'].values)
X_cv_cat = vectorizer1.transform(X_cv['clean_categories'].values)
X_test_cat = vectorizer1.transform(X_test['clean_categories'].values)
print(vectorizer1.get_feature_names())
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds', 'Health_Sports', 'Math_Scie
```

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

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

```
my_counter = Counter()
for word in X_train['clean_subcategories'].values:
    my_counter.update(word.split())

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

from sklearn.feature_extraction.text import CountVectorizer
```

```
vectorizer2 = CountVectorizer(vocabulary=list(project_subject_subcategories_dict.keys()), lowercase=False, binary=True)
vectorizer2.fit(X_train['clean_subcategories'].values)
# firstly convert fit the train data into the vectoriaer then it learn hte vocablery
# we use the fitted CountVectorizer to convert the text to vector
X_train_subcat = vectorizer2.transform(X_train['clean_subcategories'].values)
X_cv_subcat = vectorizer2.transform(X_cv['clean_subcategories'].values)
X_test_subcat = vectorizer2.transform(X_test['clean_subcategories'].values)
print(vectorizer2.get_feature_names())

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

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Civics_Government', 'Extracurricular', 'Fo
After vectorizations
(22445, 30) (22445,)
(11055, 30) (11055,)
(16500, 30) (16500,)
====================================================================================================
```

```
from collections import Counter
my_counter = Counter()
for word in X_train['school_state'].values:
    my_counter.update(word.split())# count the words
school_state_dict = dict(my_counter)# store in dicionary
sorted_school_state_dict = dict(sorted(school_state_dict.items(), key=lambda kv: kv[1]))

from sklearn.feature_extraction.text import CountVectorizer
vectorizer3 = CountVectorizer(vocabulary=list(sorted_school_state_dict.keys()), lowercase=False, binary=True)
vectorizer3.fit(dft['school_state'].values)
# firstly convert fit the train data into the vector then it learn the vocablery
# we use the fitted CountVectorizer to convert the text to vector
X_train_school_state = vectorizer3.transform(X_train['school_state'].values)
X_cv_school_state = vectorizer3.transform(X_cv['school_state'].values)
X_test_school_state = vectorizer3.transform(X_test['school_state'].values)
print(vectorizer3.get_feature_names())
print("After vectorizations")
print(X_train_school_state .shape, y_train.shape)
print(X_cv_school_state .shape, y_cv.shape)
print(X_test_school_state .shape, y_test.shape)
print("="*100)
```

```
['VT', 'WY', 'ND', 'MT', 'NE', 'RI', 'SD', 'NH', 'AK', 'DE', 'WV', 'NM', 'ME', 'DC', 'HI', 'IA', 'ID', 'KS', 'AR', 'CO'
After vectorizations
(22445, 51) (22445,)
(11055, 51) (11055,)
(16500, 51) (16500,)
====================================================================================================
```

```
my_counter = Counter()
for word in X_train['clean_grade'].values:
    my_counter.update(word.split())
project_grade_category_dict= dict(my_counter)
sorted_project_grade_category_dict = dict(sorted(project_grade_category_dict.items(), key=lambda kv: kv[1]))
sorted_project_grade_category_dict


dft['clean_grade']=dft['clean_grade'].fillna("")# fill the null values with space
vectorizer4 = CountVectorizer(vocabulary=list(sorted_project_grade_category_dict.keys()),lowercase=False, binary=True)
vectorizer4.fit(dft['clean_grade'].values)
# firstly convert fit the train data into the vectoriaer then it learn hte vocablery
# we use the fitted CountVectorizer to convert the text to vector
X_train_project_grade_category = vectorizer4.transform(X_train['clean_grade'].values)
X_cv_project_grade_category = vectorizer4.transform(X_cv['clean_grade'].values)
X_test_project_grade_category = vectorizer4.transform(X_test['clean_grade'].values)
print(vectorizer4.get_feature_names())

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

```
['grades_9_12', 'grades_6_8', 'grades_3_5', 'grades_prek_2']
After vectorizations
(22445, 4) (22445,)
(11055, 4) (11055,)
(16500, 4) (16500,)
====================================================================================================
```

```
dft['teacher_prefix']=dft['teacher_prefix'].fillna(" ")# filll the null valueswith space
my_counter = Counter()
for word in dft['teacher_prefix'].values:
    my_counter.update(word.split())
teacher_cat_dict = dict(my_counter)
```

```
sorted_teacher_prefix_dict = dict(sorted(teacher_cat_dict.items(), key=lambda kv: kv[1]))

from sklearn.feature_extraction.text import CountVectorizer
vectorizer5 = CountVectorizer(vocabulary=list(sorted_teacher_prefix_dict.keys()), lowercase=False,binary=True)
vectorizer5.fit(dft['teacher_prefix'].values.astype('U'))
# firstly convert fit the train data into the vectorizer
# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_prefix = vectorizer5.transform(X_train['teacher_prefix'].values.astype('U'))
X_cv_teacher_prefix= vectorizer5.transform(X_cv['teacher_prefix'].values.astype('U'))
X_test_teacher_prefix = vectorizer5.transform(X_test['teacher_prefix'].values.astype('U'))
print(vectorizer5.get_feature_names())

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

```
['Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
After vectorizations
(22445, 5) (22445,)
(11055, 5) (11055,)
(16500, 5) (16500,)
====================================================================================================
```

```
# chainging the names
X_train_essay=preprocessed_essays_train
X_cv_essay=preprocessed_essays_cv
X_test_essay=preprocessed_essays_test
```

```
# Considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer6 = CountVectorizer(min_df=10,max_features=5000,ngram_range=(1, 2))
vectorizer6.fit(X_train_essay)# that is learned from trained data

# we use the fitted CountVectorizer to convert the text to vector
X_train_bow = vectorizer6.transform(X_train_essay)
X_cv_bow = vectorizer6.transform(X_cv_essay)
X_test_bow = vectorizer6.transform(X_test_essay)
print("After vectorizations")
print(X_train_bow.shape, y_train.shape)
print(X_cv_bow.shape, y_cv.shape)
print(X_test_bow.shape, y_test.shape)
print("="*100)
```

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

```
X_train_title=preprocessed_titles_train
X_cv_title=preprocessed_titles_cv
X_test_title=preprocessed_titles_test
```

```
#bow featurization title
vectorizer7 = CountVectorizer(min_df=10,max_features=5000,ngram_range=(1, 2))
vectorizer7.fit(X_train_title)# that is learned from trainned data
X_train_bow_title = vectorizer7.transform(X_train_title)
X_cv_bow_title= vectorizer7.transform(X_cv_title)
X_test_bow_title = vectorizer7.transform(X_test_title)
print("After vectorizations")
print(X_train_bow_title.shape, y_train.shape)
print(X_cv_bow_title.shape, y_cv.shape)
print(X_test_bow_title.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1609) (22445,)
(11055, 1609) (11055,)
(16500, 1609) (16500,)
==================================================================================================
```

TFIDF Vectorization of the data metrix

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

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

```
X_test_tf_title = vectorizer8.transform(X_test_title)
print("After vectorizations")
print(X_train_tf_title.shape, y_train.shape)
print(X_cv_tf_title.shape, y_cv.shape)
print(X_test_tf_title.shape, y_test.shape)
print("="*100)

```

```
After vectorizations
(22445, 1609) (22445,)
(11055, 1609) (11055,)
(16500, 1609) (16500,)
====================================================================================================
```

```
#for essay
from sklearn.feature_extraction.text import TfidfVectorizer
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer9 = TfidfVectorizer(min_df=10,max_features=5000,ngram_range=(1, 2))
vectorizer9.fit(X_train_essay)# that is learned from trained data
# we use the fitted CountVectorizer to convert the text to vector
X_train_tf_essay = vectorizer9.transform(X_train_essay)
X_cv_tf_essay= vectorizer9.transform(X_cv_essay)
X_test_tf_essay = vectorizer9.transform(X_test_essay)
print("After vectorizations")
print(X_train_tf_essay.shape, y_train.shape)
print(X_cv_tf_essay.shape, y_cv.shape)
print(X_test_tf_essay.shape, y_test.shape)
print("="*100)
```

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

AVG Word2Vec Vectorization using the pretrained models

```
import gdown

url = 'https://drive.google.com/uc?id=1MqUasf7jYoPbG35MJ28VQcOjjNp-ZDDp'
output = 'glove_vectors'
gdown.download(url, output, quiet=False)
```

```
Downloading...
From: https://drive.google.com/uc?id=1MqUasf7jYoPbG35MJ28VQcOjjNp-ZDDp
To: /content/glove_vectors
128MB [00:02, 52.4MB/s]
'glove_vectors'
```

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

```
def AVG_W2V(values):
# AVG W2V Vectorization.

  train_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
  for sentence in tqdm(values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length    # we are taking the 300 dimensions  very large
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_avg_w2v_vectors.append(vector)

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

```

```
train_avg_w2v_vectors=AVG_W2V(preprocessed_essays_train)
test_avg_w2v_vectors=AVG_W2V(preprocessed_essays_test)
cv_avg_w2v_vectors=AVG_W2V(preprocessed_essays_cv)
print("After vectorizations")
print(len(train_avg_w2v_vectors), y_train.shape)
print("="*100)
```

```
100%|██████████| 22445/22445 [00:06<00:00, 3582.45it/s]
  2%|▏         | 340/16500 [00:00<00:04, 3395.42it/s]22445
300
100%|██████████| 16500/16500 [00:04<00:00, 3546.40it/s]
  3%|▎         | 370/11055 [00:00<00:02, 3697.19it/s]16500
300
100%|██████████| 11055/11055 [00:03<00:00, 3603.87it/s]11055
300
After vectorizations
22445 (22445,)
==========================================================================================================
```

```
#  AVG W2V for preprocessed_titles
train_avg_w2v_vectors_title=AVG_W2V(preprocessed_titles_train)
test_avg_w2v_vectors_title=AVG_W2V(preprocessed_titles_test)
cv_avg_w2v_vectors_title=AVG_W2V(preprocessed_titles_cv)

```

```
100%|██████████| 22445/22445 [00:00<00:00, 70982.27it/s]
 36%|███▌      | 6005/16500 [00:00<00:00, 60049.20it/s]22445
300
100%|██████████| 16500/16500 [00:00<00:00, 65943.15it/s]
100%|██████████| 11055/11055 [00:00<00:00, 75073.64it/s]16500
300
11055
300
```

3.4 TFIDF weighted W2V vectorization using the Pretrained Models

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

```
def tf_idf(word_list):
```

```
    train_title_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
    for sentence in tqdm(word_list): # for each review/sentence
        vector = np.zeros(300) # as word vectors are of zero length
        tf_idf_weight =0; # num of words with a valid vector in the sentence/review
        for word in sentence.split():#.split(): # for each word in a review/sentence
            if (word in glove_words) and (word in tfidf_words):
              #vec = model.wv[word]
              vec = model[word] # getting the vector for each word
              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]))
    return train_title_tfidf_w2v_vectors

```

```
train_tfidf_w2v_vectors=tf_idf(preprocessed_essays_train)
test_tfidf_w2v_vectors=tf_idf(preprocessed_essays_test)
cv_tfidf_w2v_vectors=tf_idf(preprocessed_essays_cv)
```

```
100%|██████████| 22445/22445 [00:38<00:00, 583.23it/s]
  0%|          | 60/16500 [00:00<00:27, 594.22it/s]22445
300
100%|██████████| 16500/16500 [00:28<00:00, 586.50it/s]
  1%|          | 60/11055 [00:00<00:18, 595.49it/s]16500
300
100%|██████████| 11055/11055 [00:18<00:00, 584.14it/s]11055
300
```

```
train_title_tfidf_w2v_vectors=tf_idf(preprocessed_titles_train)
test_title_tfidf_w2v_vectors=tf_idf(preprocessed_titles_test)
cv_title_tfidf_w2v_vectors=tf_idf(preprocessed_titles_cv)
```

```
100%|██████████| 22445/22445 [00:00<00:00, 38113.87it/s]
 31%|███       | 5129/16500 [00:00<00:00, 26755.79it/s]22445
300
100%|██████████| 16500/16500 [00:00<00:00, 27463.81it/s]
 36%|███       | 3949/11055 [00:00<00:00, 39486.74it/s]16500
300
100%|██████████| 11055/11055 [00:00<00:00, 37477.29it/s]
11055
300
```

4. 1 Vectorization of the Numerical features

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

```
from sklearn.preprocessing import StandardScaler
# https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

scalar =  StandardScaler()
scalar.fit(X_train['price'].values.reshape(-1,1))

train_price_standar = scalar.transform(X_train['price'].values.reshape(-1, 1))
test_price_standar = scalar.transform(X_test['price'].values.reshape(-1, 1))
cv_price_standar = scalar.transform(X_cv['price'].values.reshape(-1, 1))
```

```
scalar.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
train_prev_proj_standar = scalar.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1))
test_prev_proj_standar = scalar.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1))
cv_prev_proj_standar = scalar.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1))
```

```
scalar.fit(X_train['quantity'].values.reshape(-1,1))
train_qnty_standar = scalar.transform(X_train['quantity'].values.reshape(-1, 1))
cv_qnty_standar = scalar.transform(X_cv['quantity'].values.reshape(-1, 1))
test_qnty_standar = scalar.transform(X_test['quantity'].values.reshape(-1, 1))
```

*Mergaing of all the Data matrix for diffrent sets of opeations *

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

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_set1_train = hstack((X_train_bow_title,X_train_bow,X_train_teacher_prefix,X_train_cat,X_train_subcat ,X_train_project_grade_cate
                       X_train_school_state,train_qnty_standar,train_price_standar,train_prev_proj_standar))
# printing the shape of X_set1_train data metrix
print(" printing the shape of X_set1_train data metrix",X_set1_train.shape, y_train.shape)

X_set1_cv = hstack((X_cv_bow_title,X_cv_bow,X_cv_teacher_prefix,X_cv_cat,X_cv_subcat,X_cv_project_grade_category,X_cv_school_state
                    cv_qnty_standar,cv_price_standar,cv_prev_proj_standar))
# printing the shape of X_set1_cv data metrix
print("printing the shape of X_set1_cv data metrix",X_set1_cv.shape, y_cv.shape)

X_set1_test = hstack((X_test_bow_title,X_test_bow,X_test_teacher_prefix,X_test_cat,X_test_subcat,X_test_project_grade_category,X_t
                      test_qnty_standar,test_price_standar,test_prev_proj_standar))

# printing the shape of X_set1_test data metrix
print("printing the shape of X_set1_test data metrix ",X_set1_test.shape, y_test.shape)

```

```
 printing the shape of X_set1_train data metrix (22445, 6711) (22445,)
printing the shape of X_set1_cv data metrix (11055, 6711) (11055,)
printing the shape of X_set1_test data metrix  (16500, 6711) (16500,)
```

```
X_set2_train = hstack((X_train_tf_essay,X_train_tf_title,X_train_teacher_prefix,X_train_cat,X_train_subcat,X_train_project_grade_c
                       train_qnty_standar,train_price_standar,train_prev_proj_standar))


print("printing the shape of X_set2_train data metrix",X_set2_train.shape, y_train.shape)

print("*"*50)
X_set2_cv = hstack((X_cv_tf_essay,X_cv_tf_title,X_cv_teacher_prefix,X_cv_cat,X_cv_subcat,X_cv_project_grade_category,X_cv_school_s
                    cv_qnty_standar,cv_price_standar,cv_prev_proj_standar))

print("printing the shape of X_set2_cv data metrix",X_set2_cv.shape, y_cv.shape)

print("*"*50)
X_set2_test = hstack((X_test_tf_essay,X_test_tf_title,X_test_teacher_prefix,X_test_cat,X_test_subcat, X_test_project_grade_categor
```

```
                    test_qnty_standar,test_price_standar,test_prev_proj_standar))

print("printing the shape of X_set2_test data metrix",X_set2_test.shape, y_test.shape)
```

```
printing the shape of X_set2_train data metrix (22445, 6711) (22445,)
**************************************************
printing the shape of X_set2_cv data metrix (11055, 6711) (11055,)
**************************************************
printing the shape of X_set2_test data metrix (16500, 6711) (16500,)
```

```
X_set3_train = hstack((train_avg_w2v_vectors,train_avg_w2v_vectors_title,train_prev_proj_standar,train_price_standar,train_qnty_st
                    X_train_teacher_prefix,X_train_cat,X_train_subcat,
                    X_train_project_grade_category,X_train_school_state))


print("printing the shape of X_set3_train data metrix",X_set3_train.shape, y_train.shape)
print("*"*50)

X_set3_cv = hstack((cv_avg_w2v_vectors,cv_avg_w2v_vectors_title,cv_prev_proj_standar,cv_price_standar,cv_qnty_standar,
                    X_cv_teacher_prefix,X_cv_cat,X_cv_subcat,
                    X_cv_project_grade_category,X_cv_school_state))


print("printing the shape of X_set3_cv data metrix",X_set3_cv.shape, y_cv.shape)
print("*"*50)

X_set3_test = hstack((test_avg_w2v_vectors,test_avg_w2v_vectors_title,test_prev_proj_standar,test_price_standar,test_qnty_standar,
                    X_test_teacher_prefix,X_test_cat,X_test_subcat,
                    X_test_project_grade_category,X_test_school_state))


print("printing the shape of X_set3_test data metrix",X_set3_test.shape, y_test.shape)
```

```
printing the shape of X_set3_train data metrix (22445, 702) (22445,)
**************************************************
printing the shape of X_set3_cv data metrix (11055, 702) (11055,)
**************************************************
printing the shape of X_set3_test data metrix (16500, 702) (16500,)
```

```
X_set4_train = hstack((train_tfidf_w2v_vectors,train_title_tfidf_w2v_vectors,train_prev_proj_standar,train_price_standar,train_qnt
                    X_train_teacher_prefix,X_train_cat,X_train_subcat,
```

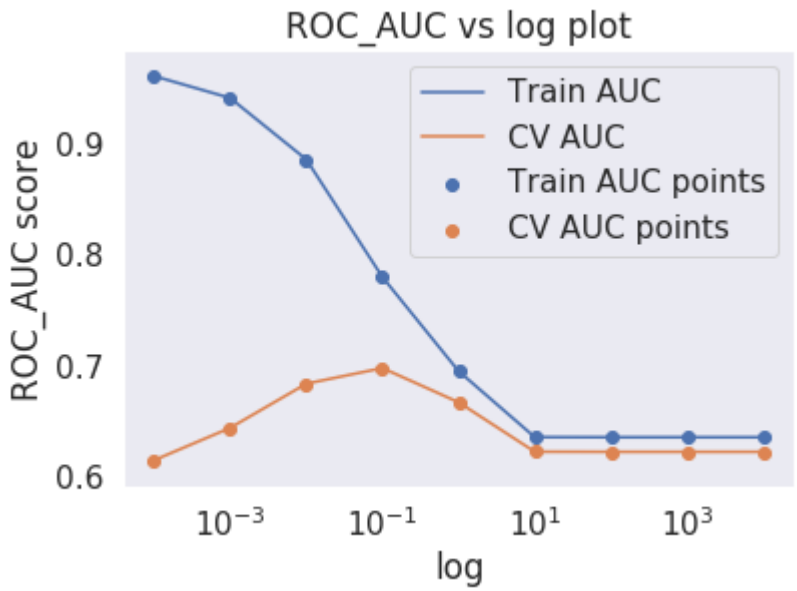
```
                      X_train_project_grade_category,X_train_school_state))


print("printing the shape of X_set4_train data metrix",X_set4_train.shape, y_train.shape)
print("*"*50)

X_set4_cv = hstack((cv_tfidf_w2v_vectors,cv_title_tfidf_w2v_vectors,cv_prev_proj_standar,cv_price_standar,cv_qnty_standar,
                      X_cv_teacher_prefix,X_cv_cat,X_cv_subcat,
                      X_cv_project_grade_category,X_cv_school_state))


print("printing the shape of X_set4_CV data metrix",X_set4_cv.shape, y_cv.shape)

print("*"*50)
X_set4_test = hstack((test_title_tfidf_w2v_vectors,test_tfidf_w2v_vectors,test_prev_proj_standar,test_price_standar,test_qnty_star
                      X_test_teacher_prefix,X_test_cat,X_test_subcat,
                      X_test_project_grade_category,X_test_school_state))


print("printing the shape of X_set4_test data metrix",X_set4_test.shape, y_test.shape)
```

```
printing the shape of X_set4_train data metrix (22445, 702) (22445,)
**************************************************
printing the shape of X_set4_CV data metrix (11055, 702) (11055,)
**************************************************
printing the shape of X_set4_test data metrix (16500, 702) (16500,)
```

Applying the SGDClassifier on SET:1

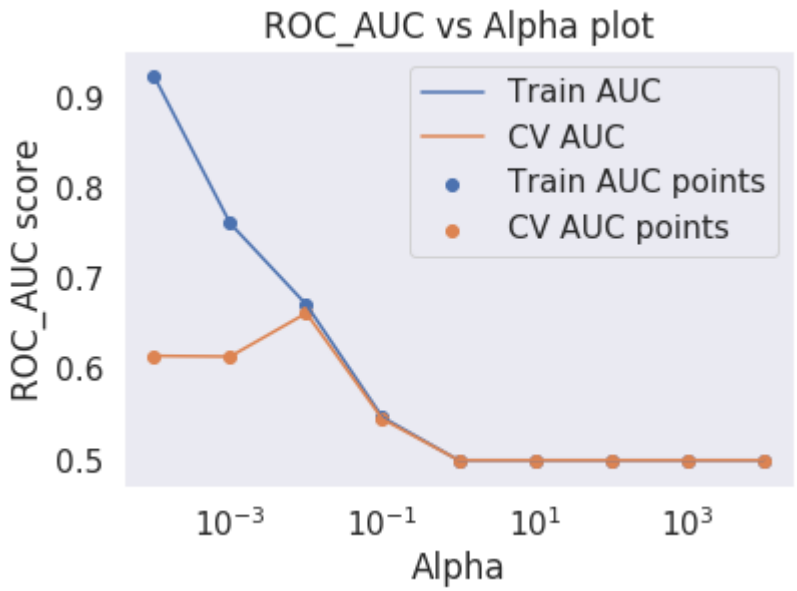
```
## By using "l2" Regulrizer
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import SGDClassifier

# hyperparameter tuning with l2 reg
parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}
sd = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced')
classifier = GridSearchCV(sd, parameters, cv= 5, scoring='roc_auc',return_train_score=True)
classifier.fit(X_set1_train, y_train)

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

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

```
plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("log")
plt.xscale('log')
plt.ylabel("ROC_AUC score")
plt.title("ROC_AUC vs log plot")
plt.grid()
plt.show()

```



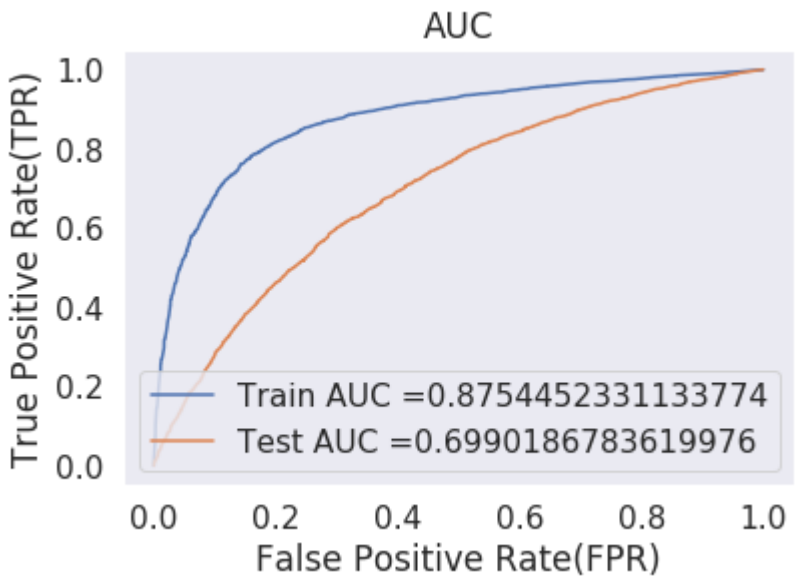

```
#By using  "l1" Regularization
import warnings
warnings.filterwarnings("ignore")

parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}
sd = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced')
classifier = GridSearchCV(sd, parameters, cv= 5, scoring='roc_auc',return_train_score=True)
classifier.fit(X_set1_train, y_train)
train_auc = classifier.cv_results_['mean_train_score']
cv_auc= classifier.cv_results_['mean_test_score']

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

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

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

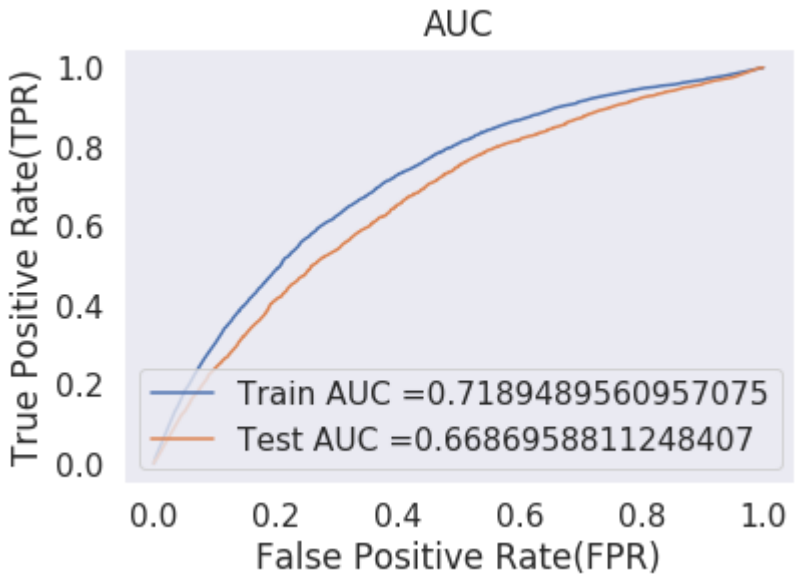



```
# using the L2 regularization
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import roc_curve, auc
Classifier_bow = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced', alpha = 10**-2)
Classifier_bow.fit(X_set1_train ,y_train)

clfcalibrated = CalibratedClassifierCV(Classifier_bow, cv=3 , method='isotonic')
clfcalibrated.fit(X_set1_train, y_train)

y_train_pred = clfcalibrated.predict_proba(X_set1_train)[:, 1]
y_test_pred1 = clfcalibrated.predict_proba(X_set1_test)[:, 1]
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_pred1)

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

```
# print(y_train_pred.shape)
# y_test_pred.shape
```



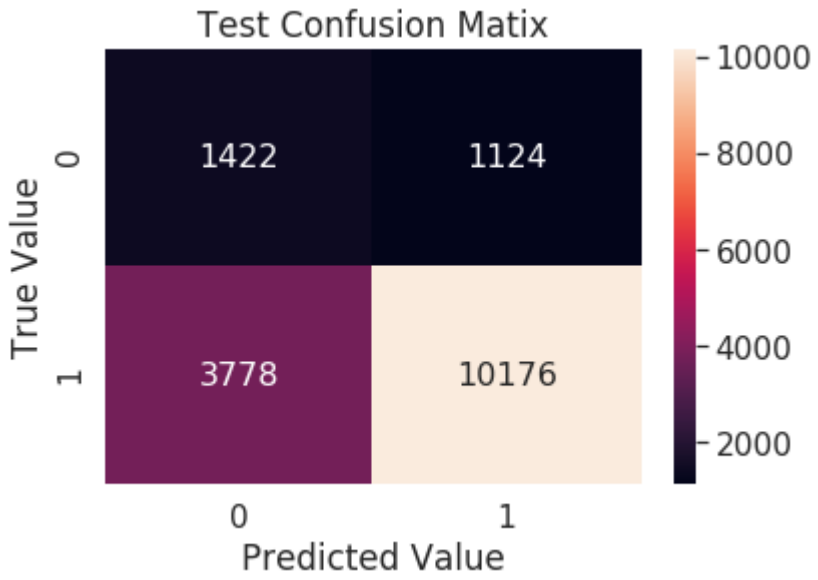

```
# using the L1 regularization
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import roc_curve, auc
Classifier_bow = SGDClassifier(loss = 'hinge', penalty = 'l1', alpha = 10**-3)
Classifier_bow.fit(X_set1_train ,y_train)

clfcalibrated = CalibratedClassifierCV(Classifier_bow, cv=3 , method='isotonic')
clfcalibrated.fit(X_set1_train, y_train)

y_train_pred = clfcalibrated.predict_proba(X_set1_train)[:, 1]
y_test_pred = clfcalibrated.predict_proba(X_set1_test)[:, 1]
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.ylabel("True Positive Rate(TPR)")
plt.xlabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
# print(y_train_pred.shape)
# y_test_pred.shape
```




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

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

```

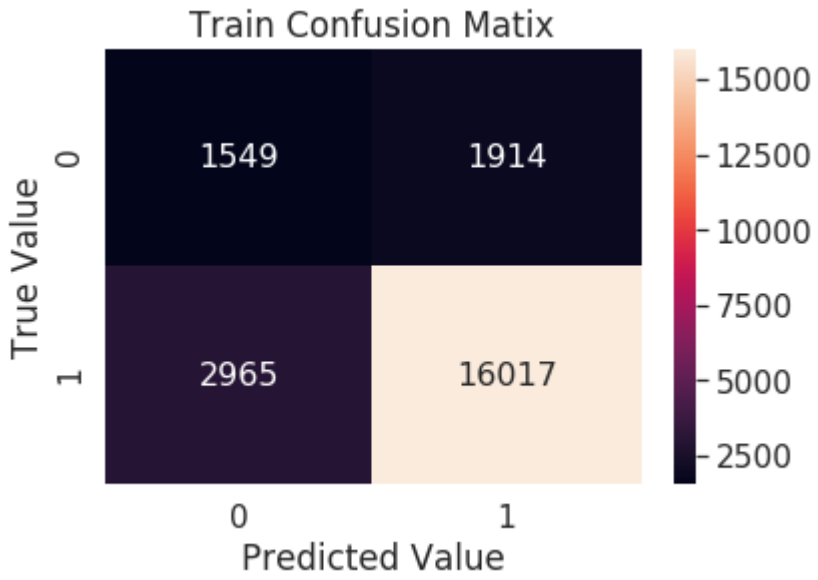
```
import seaborn as sea
test_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred1, te_thresholds,test_fpr,test_fpr)),range(2),rang
sea.set(font_scale=1.4)
sea.heatmap(test_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = 'd')
plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Test Confusion Matix")
```

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.82
Text(0.5, 1.0, 'Test Confusion Matix')
```




```
import seaborn as sea
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_train,predict(y_train_pred, te_thresholds,train_fpr,train_fpr)),range(2),
sea.set(font_scale=1.4)
sea.heatmap(train_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = 'd')
plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Train Confusion Matix")
```

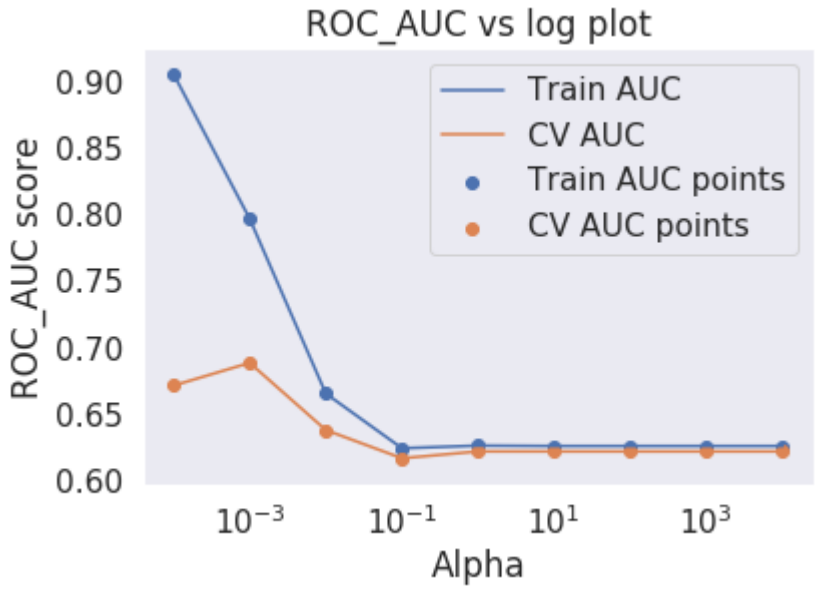
```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.8
Text(0.5, 1.0, 'Train Confusion Matix')
```



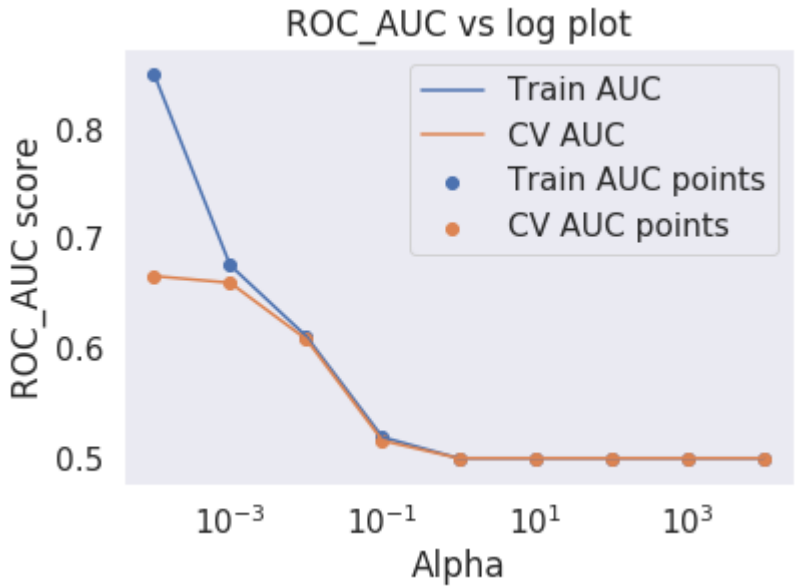
Applying the SGDClassifier on SET:2

```
#BY USING L2 REGULARISER
# hyperparameter tuning with l2 reg
parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}
sd = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced')
classifier = GridSearchCV(sd, parameters, cv= 5, scoring='roc_auc',return_train_score=True)
classifier.fit(X_set2_train, y_train)

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

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

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

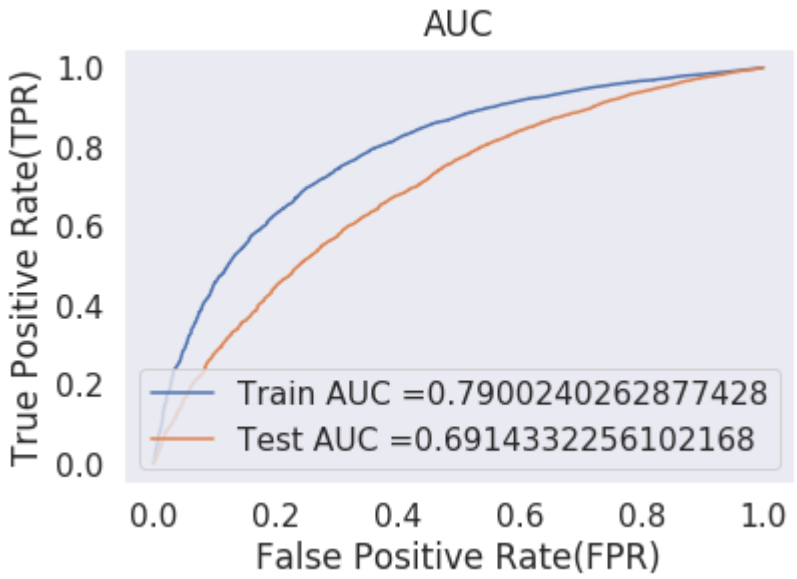
```
plt.show()
```



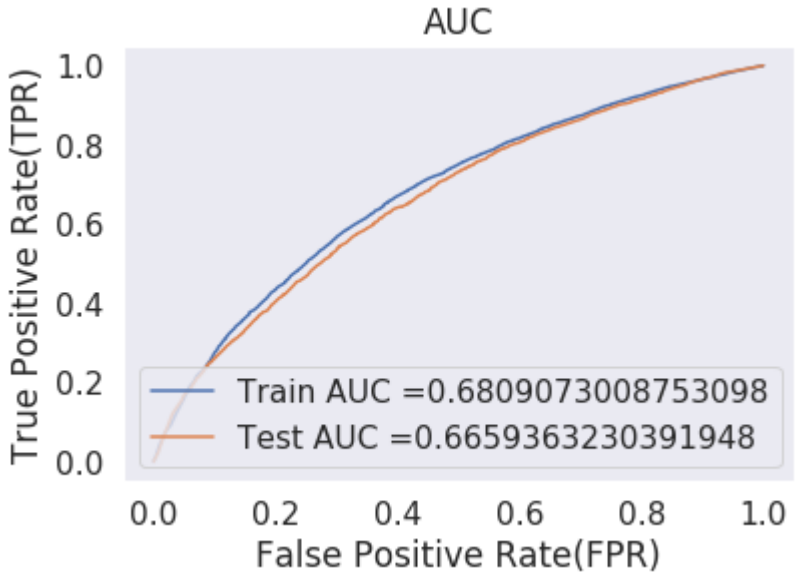

```
#BY USING "L1" REGULARISER
# hyperparameter tuning with l2 reg reduce the alpha values in list
parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}
sd = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced')
classifier = GridSearchCV(sd, parameters, cv= 5, scoring='roc_auc',return_train_score=True)
classifier.fit(X_set2_train, y_train)
train_auc = classifier.cv_results_['mean_train_score']
cv_auc= classifier.cv_results_['mean_test_score']
plt.plot(parameters['alpha'], train_auc, label='Train AUC')
plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
plt.legend()
plt.xscale('log')
plt.xlabel("Alpha")
plt.ylabel("ROC_AUC score")
plt.title("ROC_AUC vs log plot")
plt.grid()
plt.show()

```




```
# using the L2 regularization
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import roc_curve, auc
Classifier_bow = SGDClassifier(loss = 'hinge', penalty = 'l2',class_weight = 'balanced', alpha = 10**-3)
Classifier_bow.fit(X_set2_train ,y_train)

clfcalibrated = CalibratedClassifierCV(Classifier_bow, cv=3 , method='isotonic')
clfcalibrated.fit(X_set2_train, y_train)

y_train_pred2 = clfcalibrated.predict_proba(X_set2_train)[:, 1]
y_test_pred2 = clfcalibrated.predict_proba(X_set2_test)[:, 1]
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred2)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test , y_test_pred2)

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.ylabel("True Positive Rate(TPR)")
plt.xlabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
# print(y_train_pred.shape)
# y_test_pred.shape
```

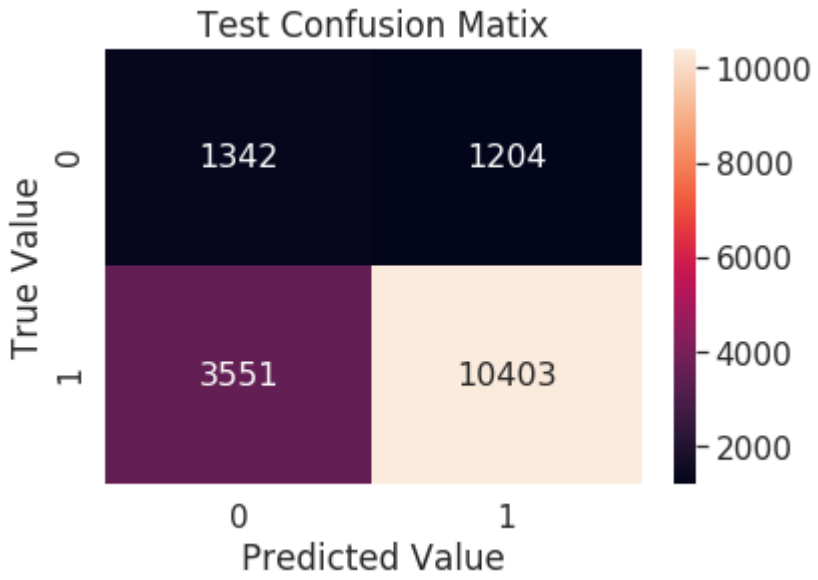



```
# using the L2 regularization
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import roc_curve, auc
Classifier_bow = SGDClassifier(loss = 'hinge', penalty = 'l1',class_weight = 'balanced', alpha = 10**-3)
Classifier_bow.fit(X_set2_train ,y_train)

clfcalibrated = CalibratedClassifierCV(Classifier_bow, cv=3 , method='isotonic')
clfcalibrated.fit(X_set2_train, y_train)

y_train_pred = clfcalibrated.predict_proba(X_set2_train)[:, 1]
y_test_pred = clfcalibrated.predict_proba(X_set2_test)[:, 1]
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.ylabel("True Positive Rate(TPR)")
plt.xlabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
# print(y_train_pred.shape)
# y_test_pred.shape
```



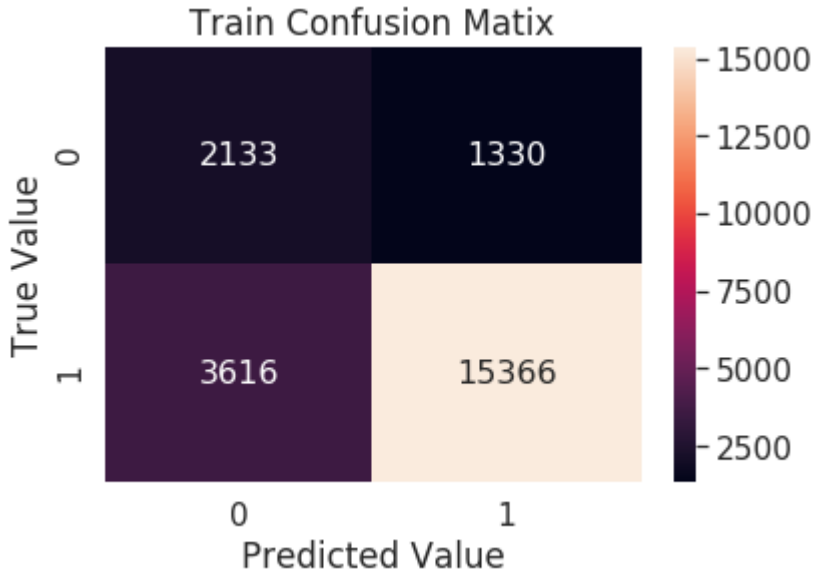

```
#CONFUSION MATRIX
import seaborn as sea
test_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred2,te_thresholds,test_fpr,test_fpr)), range(2),rang
sea.set(font_scale=1.4)
sea.heatmap(test_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = 'd')
plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Test Confusion Matix")

```

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.81
Text(0.5, 1.0, 'Test Confusion Matix')
```




```
import seaborn as sea
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_train,predict(y_train_pred2,te_thresholds,train_fpr,train_fpr)), range(2)
sea.set(font_scale=1.4)
sea.heatmap(train_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = 'd')
plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Train Confusion Matix")
```

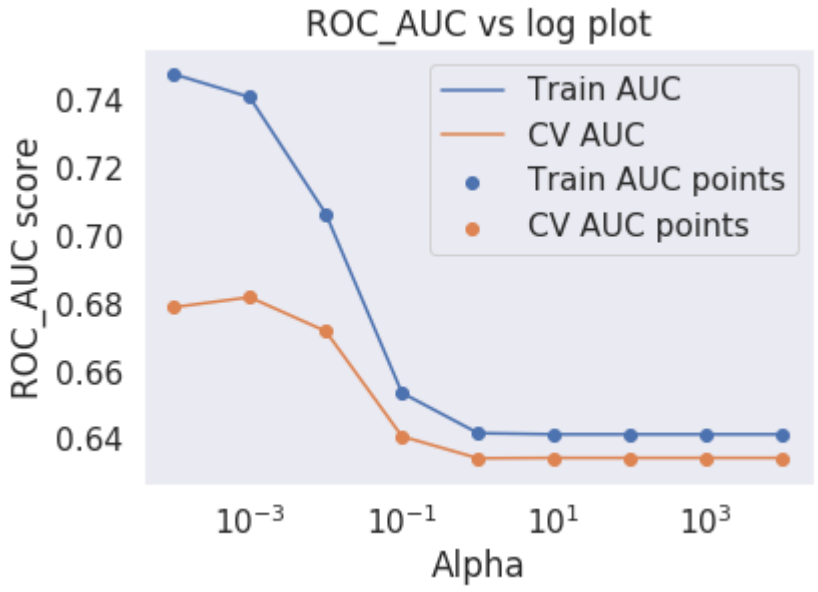
```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.79
Text(0.5, 1.0, 'Train Confusion Matix')
```



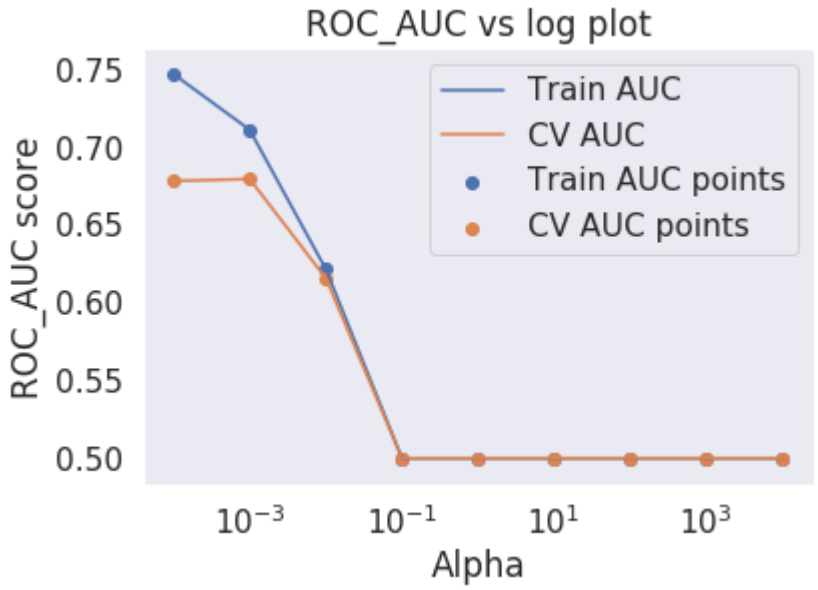
Applying the SGDClassifier on SET:3

```
#BY USING "L2" REGULARISER
# hyperparameter tuning with l2 reg
parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}
sd = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced')

classifier = GridSearchCV(sd, parameters, cv= 5, scoring='roc_auc',return_train_score=True)
classifier.fit(X_set3_train, y_train)

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

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
plt.legend()
plt.xscale('log')
plt.xlabel("Alpha")
plt.ylabel("ROC_AUC score")
plt.title("ROC_AUC vs log plot")
plt.grid()
```

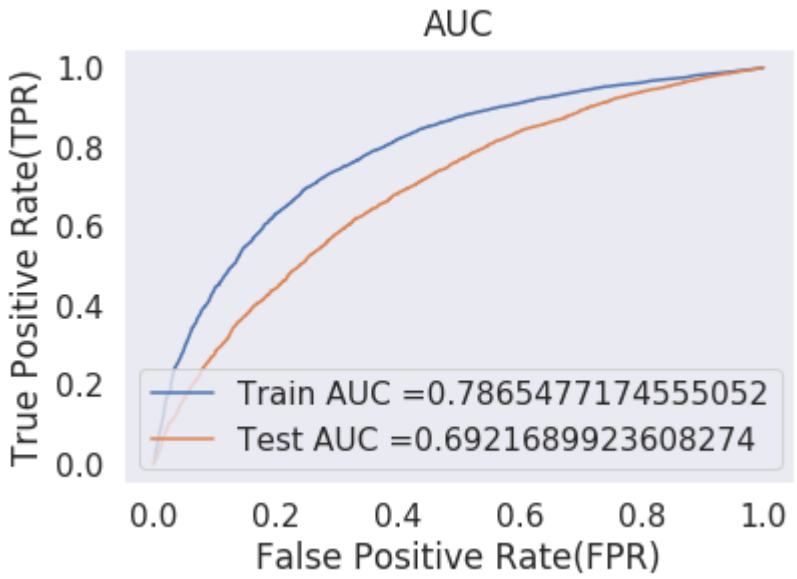
```
plt.show()
```



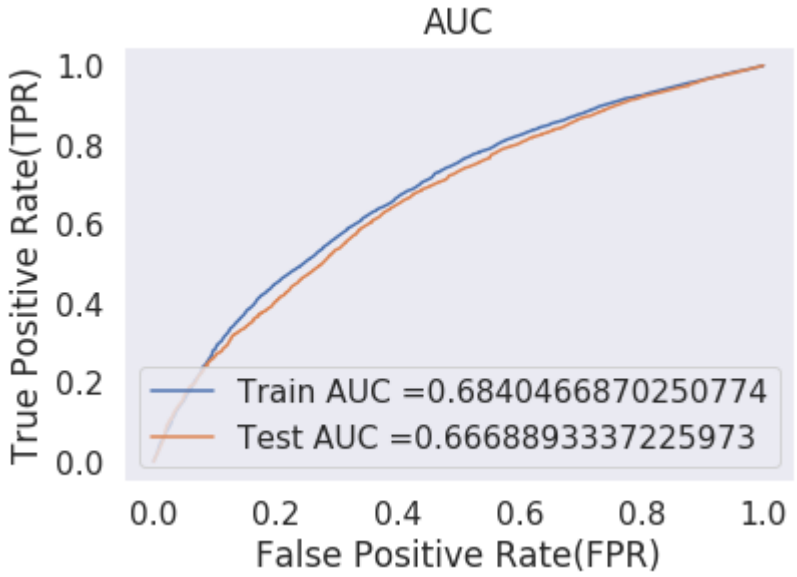

```
#BY USING "L1" REGULARISER
parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}
sd = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced')

classifier = GridSearchCV(sd, parameters, cv= 5, scoring='roc_auc',return_train_score=True)
classifier.fit(X_set3_train, y_train)

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

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
plt.legend()
plt.xscale('log')
plt.xlabel("Alpha")
plt.ylabel("ROC_AUC score")
plt.title("ROC_AUC vs log plot")
plt.grid()
plt.show()

```




```
# using the L2 regularization
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import roc_curve, auc
Classifier_bow = SGDClassifier(loss = 'hinge', penalty = 'l2',class_weight = 'balanced', alpha = 10**-3)
Classifier_bow.fit(X_set2_train ,y_train)

clfcalibrated = CalibratedClassifierCV(Classifier_bow, cv=3 , method='isotonic')
clfcalibrated.fit(X_set2_train, y_train)

y_train_pred3 = clfcalibrated.predict_proba(X_set2_train)[:, 1]
y_test_pred3 = clfcalibrated.predict_proba(X_set2_test)[:, 1]
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred3)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test , y_test_pred3)

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.ylabel("True Positive Rate(TPR)")
plt.xlabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
# print(y_train_pred.shape)
# y_test_pred.shape
```

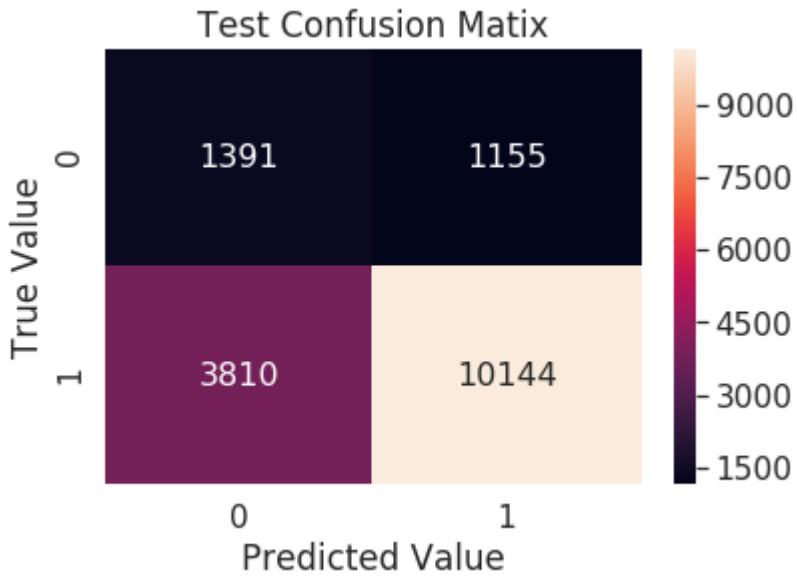



```
# using the L2 regularization
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import roc_curve, auc
Classifier_bow = SGDClassifier(loss = 'hinge',class_weight = 'balanced', penalty = 'l1', alpha = 10**-3)
Classifier_bow.fit(X_set2_train ,y_train)

clfcalibrated = CalibratedClassifierCV(Classifier_bow, cv=3 , method='isotonic')
clfcalibrated.fit(X_set2_train, y_train)

y_train_pred = clfcalibrated.predict_proba(X_set2_train)[:, 1]
y_test_pred = clfcalibrated.predict_proba(X_set2_test)[:, 1]
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.ylabel("True Positive Rate(TPR)")
plt.xlabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
# print(y_train_pred.shape)
# y_test_pred.shape
```



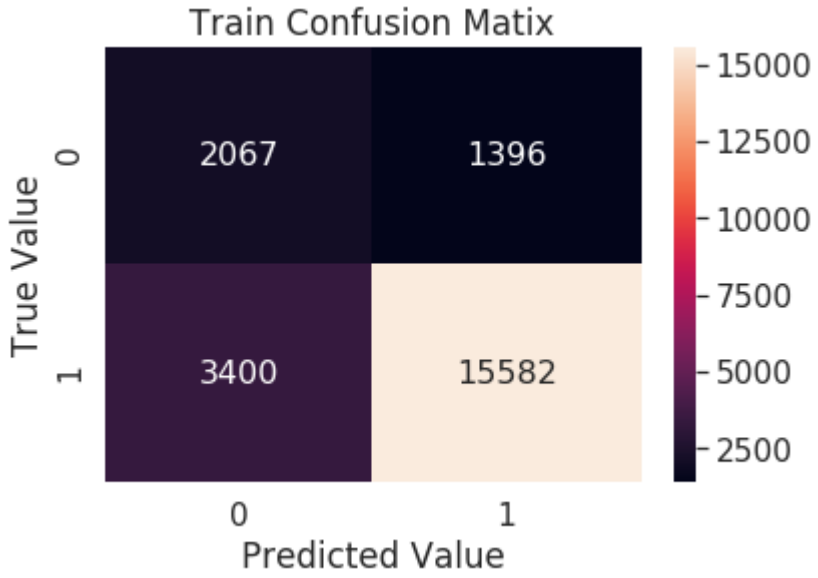

```
#CONFUSION MATRIX as per L2 Regularization
import seaborn as sea
test_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred3,te_thresholds,test_fpr,test_fpr)), range(2),rang
sea.set(font_scale=1.4)
sea.heatmap(test_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = 'd')
plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Test Confusion Matix")
```

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.81
Text(0.5, 1.0, 'Test Confusion Matix')
```




```
#CONFUSION MATRIX as per L2 Regularization
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_train,predict(y_train_pred3,te_thresholds,train_fpr,train_fpr)), range(2)
sea.set(font_scale=1.4)
sea.heatmap(train_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = 'd')
plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Train Confusion Matix")
```

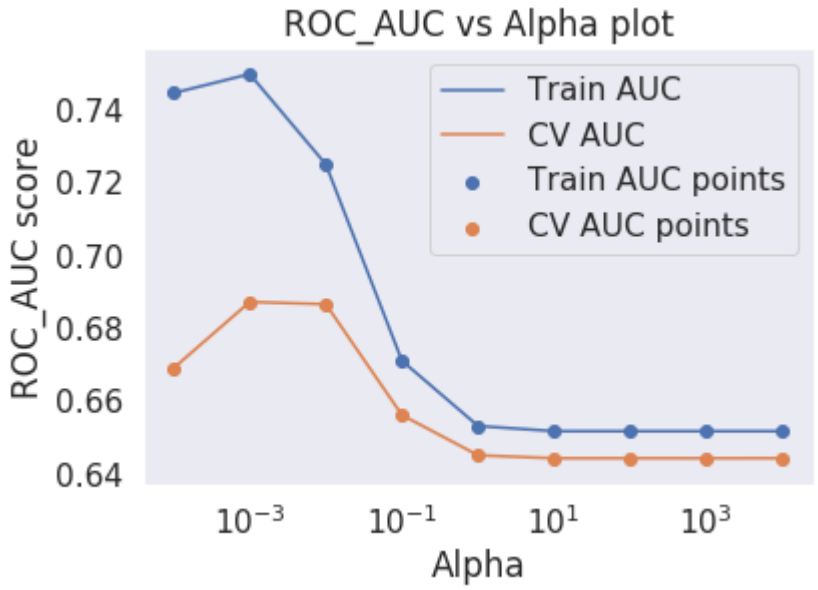
```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.79
Text(0.5, 1.0, 'Train Confusion Matix')
```



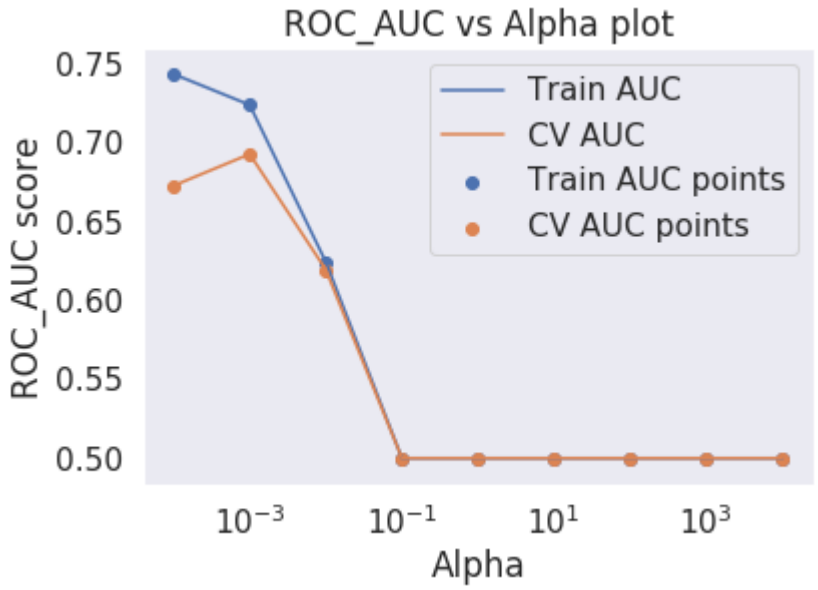
Applying the SGDClassifier on SET:4

```
#BY USING "l2" REGULARISER

parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}
SV = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced',)

classifier = GridSearchCV(SV, parameters, cv= 5, scoring='roc_auc',return_train_score=True)
classifier.fit(X_set4_train, y_train)

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

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
plt.legend()
plt.xscale('log')
plt.xlabel("Alpha")
plt.ylabel("ROC_AUC score")
plt.title("ROC_AUC vs Alpha plot")
plt.grid()
```

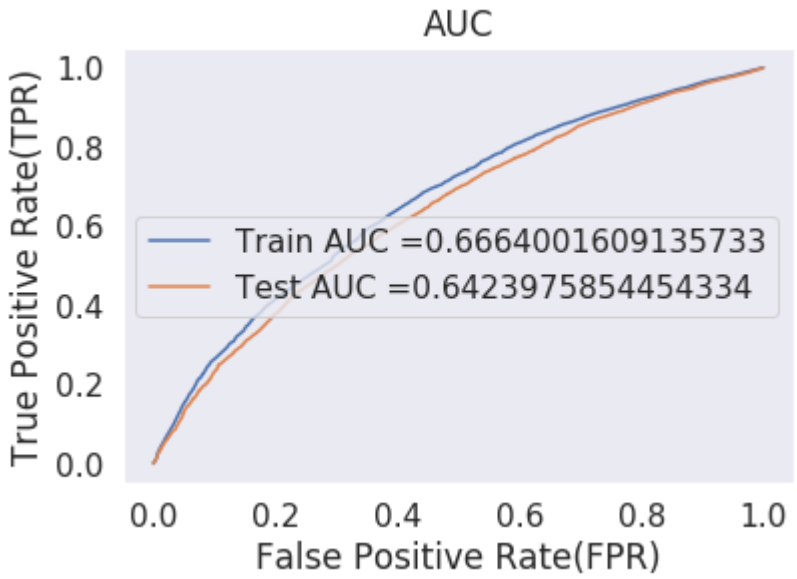
```
plt.show()
```



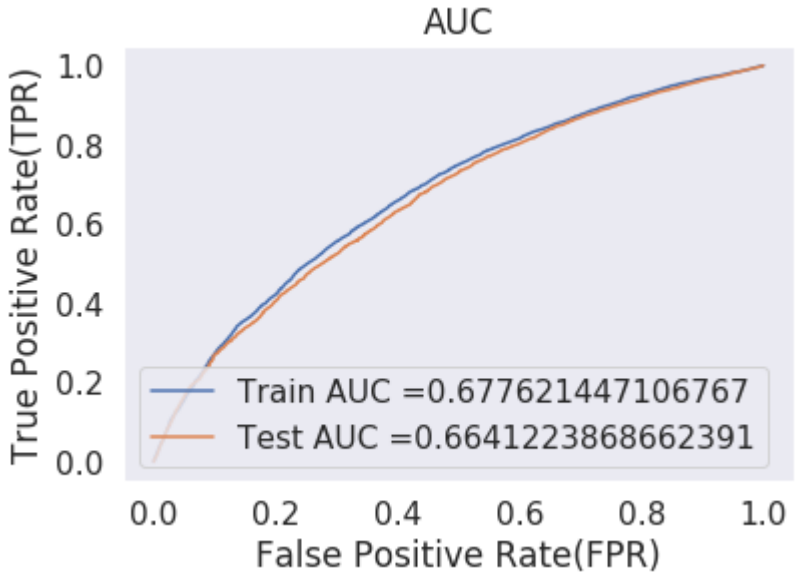

```
""#BY USING "L1" REGULARIZER

parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}
SV = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced')

classifier = GridSearchCV(SV, parameters, cv= 5, scoring='roc_auc',return_train_score=True)
classifier.fit(X_set4_train, y_train)

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

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
plt.legend()
plt.xscale('log')
plt.xlabel("Alpha")
plt.ylabel("ROC_AUC score")
plt.title("ROC_AUC vs Alpha plot")
plt.grid()
plt.show()
```




```
# using the L2 regularization
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import roc_curve, auc
Classifier_bow = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced' ,alpha = 10**-2)
Classifier_bow.fit(X_set2_train ,y_train)

clfcalibrated = CalibratedClassifierCV(Classifier_bow, cv=3 , method='isotonic')
clfcalibrated.fit(X_set2_train, y_train)

y_train_pred4 = clfcalibrated.predict_proba(X_set2_train)[:, 1]
y_test_pred4 = clfcalibrated.predict_proba(X_set2_test)[:, 1]
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred4)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test , y_test_pred4)

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.ylabel("True Positive Rate(TPR)")
plt.xlabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
# print(y_train_pred.shape)
# y_test_pred.shape
```

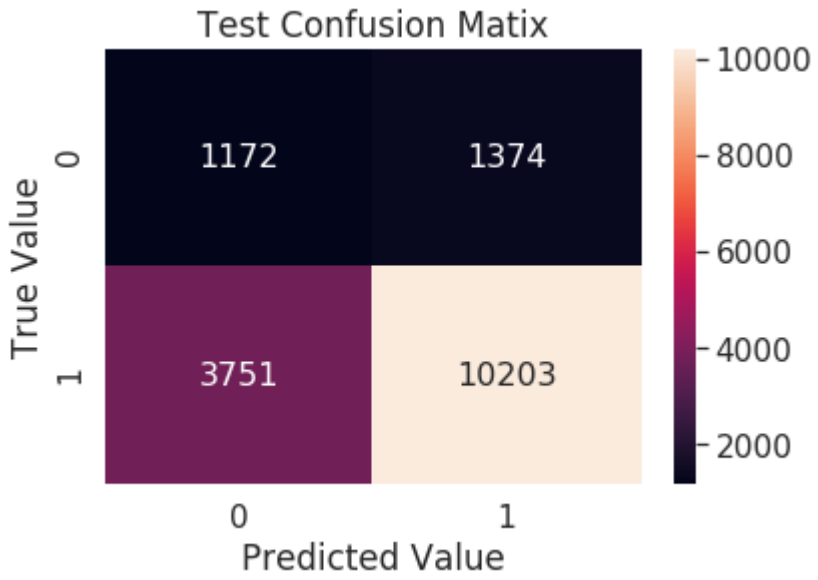



```
# using the L2 regularization
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import roc_curve, auc
Classifier_bow = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced' ,alpha = 10**-3)
Classifier_bow.fit(X_set2_train ,y_train)

clfcalibrated = CalibratedClassifierCV(Classifier_bow, cv=3 , method='isotonic')
clfcalibrated.fit(X_set2_train, y_train)

y_train_pred = clfcalibrated.predict_proba(X_set2_train)[:, 1]
y_test_pred = clfcalibrated.predict_proba(X_set2_test)[:, 1]
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.ylabel("True Positive Rate(TPR)")
plt.xlabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
# print(y_train_pred.shape)
# y_test_pred.shape
```



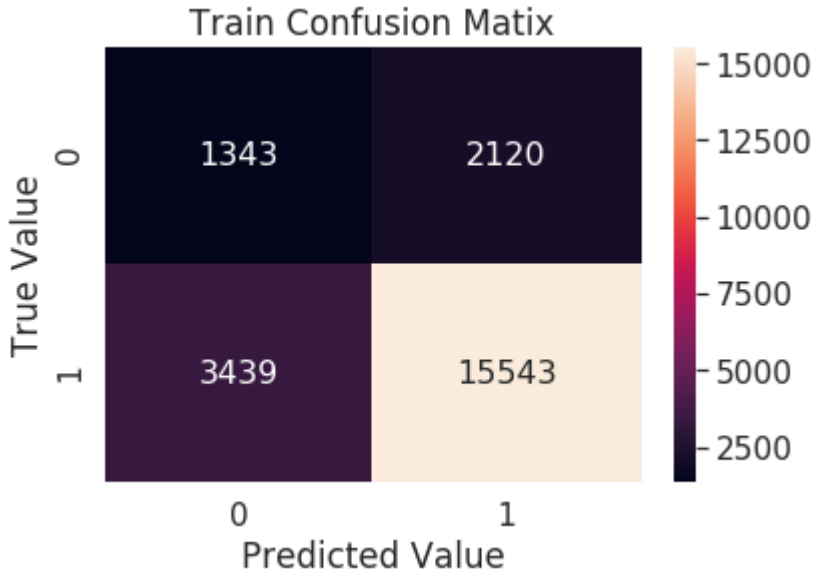

```
#CONFUSION MATRIX
import seaborn as sea
test_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred4,te_thresholds,test_fpr,test_fpr)), range(2),rang
sea.set(font_scale=1.4)
sea.heatmap(test_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = 'd')
plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Test Confusion Matix")

```

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.82
Text(0.5, 1.0, 'Test Confusion Matix')
```




```
#CONFUSION MATRIX
import seaborn as sea
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_train,predict(y_train_pred4,te_thresholds,train_fpr,train_fpr)), range(2)
sea.set(font_scale=1.4)
sea.heatmap(train_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = 'd')
plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Train Confusion Matix")

```

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.79
Text(0.5, 1.0, 'Train Confusion Matix')
```



Applying the SGDClassifier on SET:5

1. school_state : categorical data
2. clean_categories : categorical data
3. clean_subcategories : categorical data
4. project_grade_category :categorical data
5. teacher_prefix : categorical data
6. quantity : numerical data
7. teacher_number_of_previously_posted_projects : numerical data
8. price : numerical data
9. sentiment score's of each of the essay : numerical data
10. number of words in the title : numerical data
11. number of words in the combine essays : numerical data
12. Apply TruncatedSVD on TfidfVectorizer of essay text, choose the number of components (`n_components`) using elbow method : numerical data

```
print(X_train.shape)
```

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

```
(22445, 19)
(16500, 19)
(11055, 19)
```

Numeber of words in title feature number 10 in above descritpion

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

title_length_train=np.array(title_length_train)

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

title_length_test=np.array(title_length_test)
#for cv data titles
title_length_cv=[]
for i in range(0,11055):
  title_length_cv.append(len(X_cv["project_title"][i].split()))

title_length_cv=np.array(title_length_cv)
```

Number of words in combined essays

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

essay_length_test=np.array(essay_length_test)

#for cv data essay

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

essay_length_cv=np.array(essay_length_cv)
```

```

#for train data essay

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

essay_length_train=np.array(essay_length_train)
```

Sentiment scores of each combined essay's

```
"""
The code is taken from
https://www.nltk.org/_modules/nltk/sentiment/vader.html
https://www.programcreek.com/python/example/100005/nltk.sentiment.vader.SentimentIntensityAnalyzer
https://www.nltk.org/api/nltk.sentiment.html

VADER sentiment analysis tools:

Hutto, C.J. & Gilbert, E.E. (2014). VADER: A Parsimonious Rule-based Model for
Sentiment Analysis of Social Media Text. Eighth International Conference on
Weblogs and Social Media (ICWSM-14). Ann Arbor, MI, June 2014.
"""
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
nltk.download('vader_lexicon')
#https://www.programcreek.com/python/example/100005/nltk.sentiment.vader.SentimentIntensityAnalyzer
def analyze_sentiment(df):
    sentiments = []
    sid = SentimentIntensityAnalyzer()
    for i in tqdm(range(df.shape[0])):
        line = df['essay'][i]  # take one essay
        sentiment = sid.polarity_scores(line)# calculate the sentiment
        sentiments.append([sentiment['neg'], sentiment['pos'],
        sentiment['neu'], sentiment['compound']])# list of lists
    df[['neg', 'pos', 'neu', 'compound']] = pd.DataFrame(sentiments)
    df['Negative'] = df['compound'] < -0.1
    df['Positive'] = df['compound'] > 0.1
    return df
```

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

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

```
X_cv=analyze_sentiment(X_cv)
```

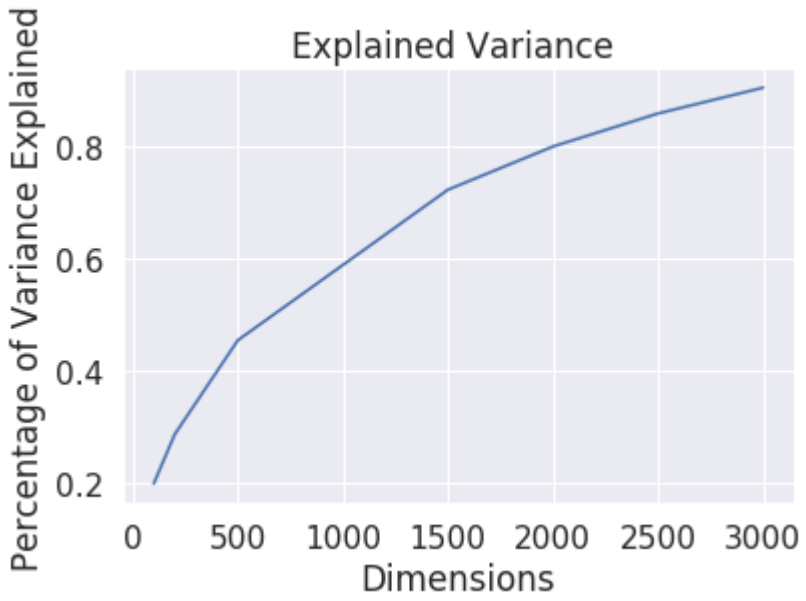
```
100%|██████████| 22445/22445 [01:00<00:00, 371.76it/s]
100%|██████████| 16500/16500 [00:43<00:00, 378.97it/s]
100%|██████████| 11055/11055 [00:29<00:00, 374.51it/s]
```

TruncatedSVD on TfidfVectorizer of essay text

```
# considering only 5000 Points
X_train_tf_essay=X_train_tf_essay[:,0:5000]
X_cv_tf_essay=X_cv_tf_essay[:,0:5000]
X_test_tf_essay=X_test_tf_essay[:,0:5000]
from sklearn.decomposition import TruncatedSVD
#https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html
Dimensions = [100,200,500,1500,2000,2500,3000]
Varience_Explained = []
for i in tqdm(Dimensions):
    svd = TruncatedSVD(n_components = i, random_state = 42)
    svd.fit(X_train_tf_essay)
    Varience_Explained.append(svd.explained_variance_ratio_.sum())
```

```
100%|██████████| 7/7 [11:33<00:00, 146.09s/it]
```

```
plt.xlabel("Dimensions")
plt.ylabel("Percentage of Variance Explained")
plt.title("Explained Variance ")
plt.plot(Dimensions,Varience_Explained)
plt.show()
```




```
# from the above graph we can see that 90 % of the varince is explained by 3000 diemnsions
svd = TruncatedSVD(n_components= 3000)
svd.fit(X_train_tf_essay)
#Transforms:
#Train SVD
X_train_tf_essay= svd.transform(X_train_tf_essay )
#Test SVD
X_test_tf_essay = svd.transform(X_test_tf_essay )
#CV SVD
X_cv_tf_essay =  svd.transform(X_cv_tf_essay )
```

```
#for train
pos=list(X_train['pos'])
pos=np.array(pos)
neg=list(X_train['neg'])
neg=np.array(neg)
com=list(X_train['compound'])
com=np.array(com)
# combine all
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_set5_train = hstack((X_train_teacher_prefix,X_train_cat,X_train_subcat,X_train_project_grade_category,X_train_school_state,trai
                       essay_length_train.reshape(-1,1),title_length_train.reshape(-1,1),
                       pos.reshape(-1,1),neg.reshape(-1,1),com.reshape(-1,1)))
```

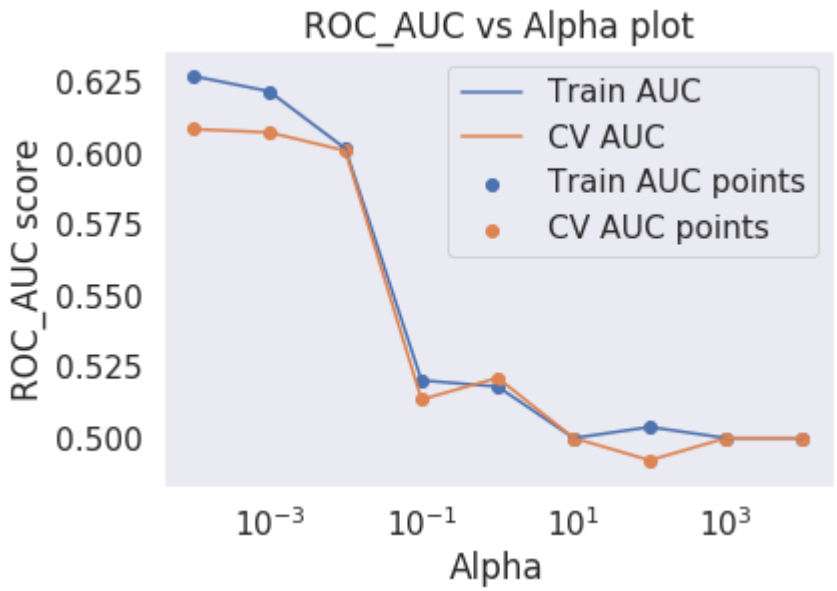
```
14
```

```
#for test
pos=list(X_test['pos'])
pos=np.array(pos)
neg=list(X_test['neg'])
neg=np.array(neg)
com=list(X_test['compound'])
com=np.array(com)
# combine all
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_set5_test = hstack((X_test_teacher_prefix,X_test_cat,X_test_subcat ,X_test_project_grade_category,X_test_school_state,
                      test_qnty_standar,test_price_standar,test_prev_proj_standar,
                      essay_length_test.reshape(-1,1),title_length_test.reshape(-1,1),
                      pos.reshape(-1,1),neg.reshape(-1,1),com.reshape(-1,1),))
```

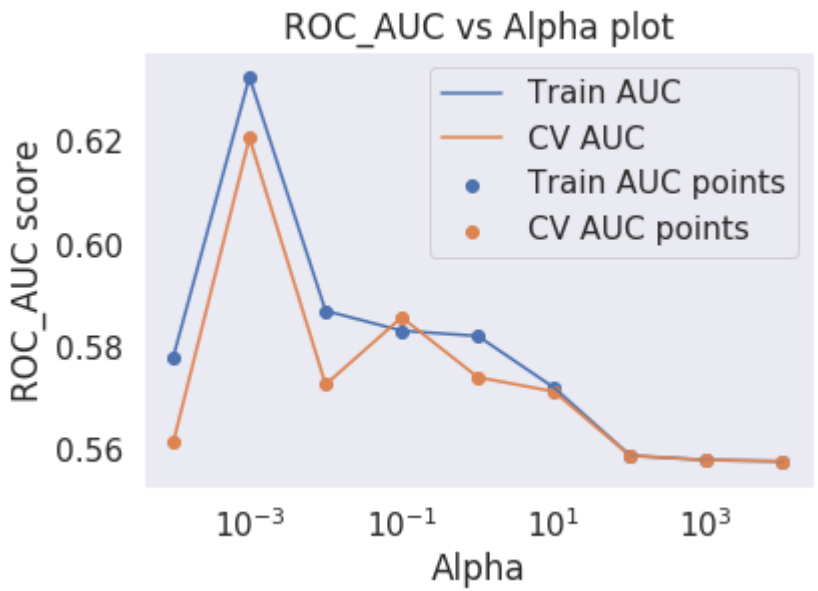
Applying SGDClassifier on SET 5

```
#BY USING L1 RGULARISER
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
#from sklearn.datasets import *
from sklearn import linear_model
from sklearn.linear_model import SGDClassifier
from sklearn import svm
# hyperparameter tuning with l2 reg
""#we are using L1 Regularizer
parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}
SV = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced',)
classifier = GridSearchCV(SV, parameters, cv= 3, scoring='roc_auc',return_train_score=True)
classifier.fit(X_set5_train, y_train)

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

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

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

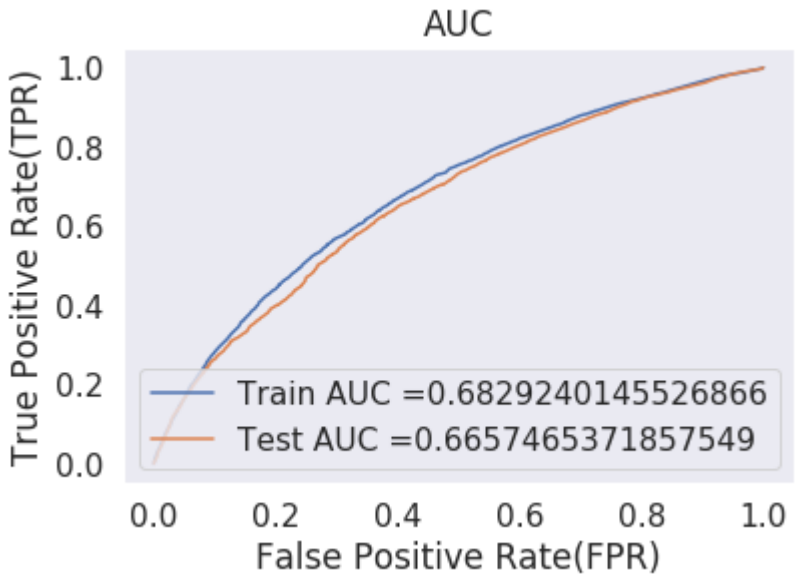
```
plt.xlabel("Alpha")
plt.ylabel("ROC_AUC score")
plt.title("ROC_AUC vs Alpha plot")
plt.grid()
plt.show()
```



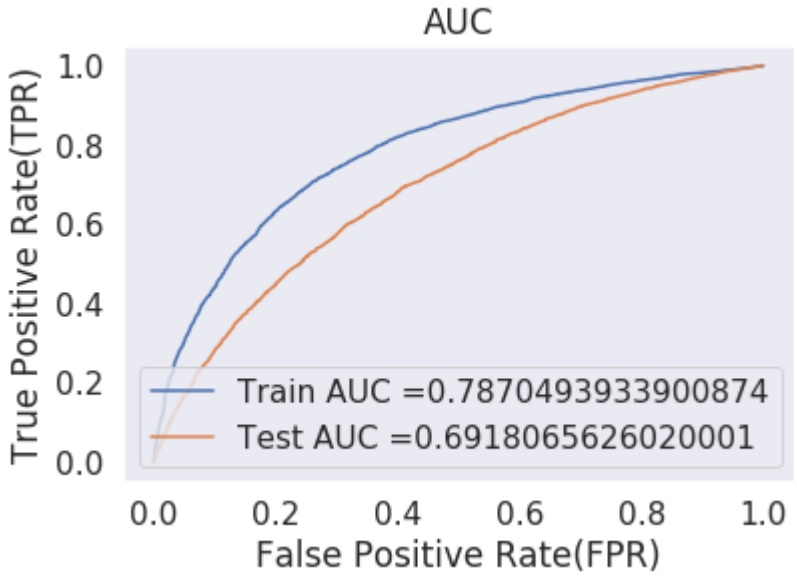

```
#BY USING L2 REGULARISER

parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}
SV = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced',)
classifier = GridSearchCV(SV, parameters, cv= 3, scoring='roc_auc',return_train_score=True)
classifier.fit(X_set5_train, y_train)

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

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
plt.legend()
plt.xscale("log")
plt.xlabel("Alpha")
plt.ylabel("ROC_AUC score")
plt.title("ROC_AUC vs Alpha plot")
plt.grid()
plt.show()
```




```
# using the L2 regularization
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import roc_curve, auc
Classifier_bow = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced' ,alpha = 10**-3)
Classifier_bow.fit(X_set2_train ,y_train)

clfcalibrated = CalibratedClassifierCV(Classifier_bow, cv=3 , method='isotonic')
clfcalibrated.fit(X_set2_train, y_train)

y_train_pred = clfcalibrated.predict_proba(X_set2_train)[:, 1]
y_test_pred = clfcalibrated.predict_proba(X_set2_test)[:, 1]
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.ylabel("True Positive Rate(TPR)")
plt.xlabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
# print(y_train_pred.shape)
# y_test_pred.shape
```

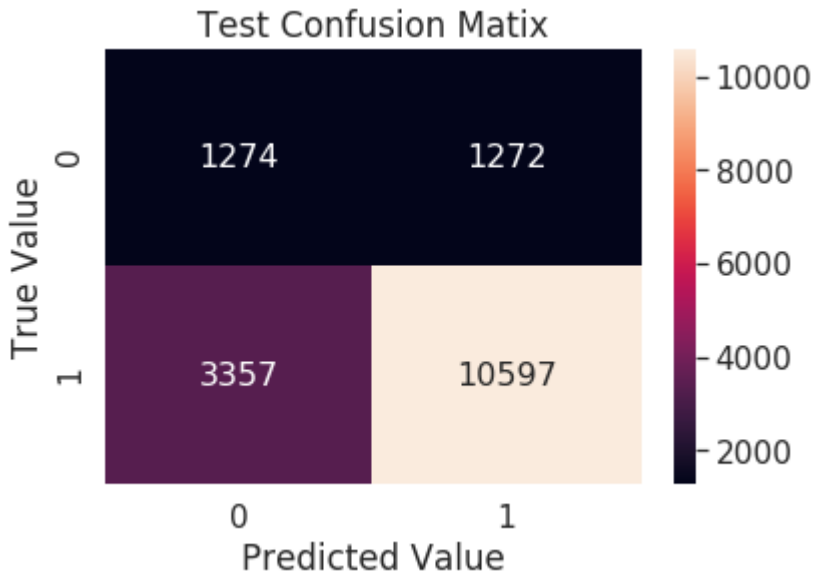



```
# using the L2 regularization
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import roc_curve, auc
Classifier_bow = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced' ,alpha = 10**-3)
Classifier_bow.fit(X_set2_train ,y_train)

clfcalibrated = CalibratedClassifierCV(Classifier_bow, cv=3 , method='isotonic')
clfcalibrated.fit(X_set2_train, y_train)

y_train_pred6 = clfcalibrated.predict_proba(X_set2_train)[:, 1]
y_test_pred6 = clfcalibrated.predict_proba(X_set2_test)[:, 1]
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred6)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test , y_test_pred6)

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.ylabel("True Positive Rate(TPR)")
plt.xlabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
# print(y_train_pred.shape)
# y_test_pred.shape
```



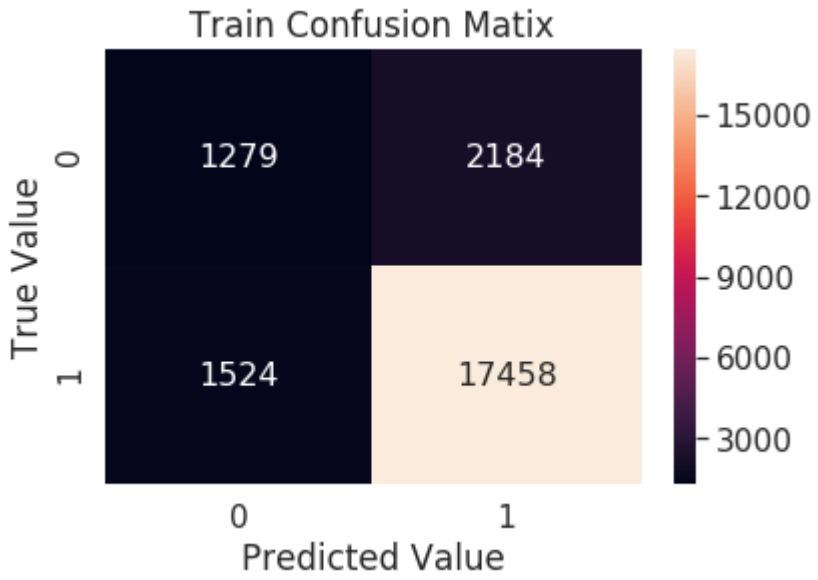

```
#CONFUSION MATRIX
import seaborn as sea
test_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred6,te_thresholds,test_fpr,test_fpr)), range(2),rang
sea.set(font_scale=1.4)
sea.heatmap(test_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = 'd')
plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Test Confusion Matix")
```

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.8
Text(0.5, 1.0, 'Test Confusion Matix')
```




```
#CONFUSION MATRIX
import seaborn as sea
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_train,predict(y_train_pred6,te_thresholds,train_fpr,train_fpr)), range(2
sea.set(font_scale=1.4)
sea.heatmap(train_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = 'd')
plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Train Confusion Matix")
```

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.73
Text(0.5, 1.0, 'Train Confusion Matix')
```




```
# Please compare all your models using Prettytable library
#how to use pretty table http://zetcode.com/python/prettytable/
from prettytable import PrettyTable
tb = PrettyTable()
tb.field_names= ("Vectorizer", " L1 Alpha ", " L2 Alpha " , " L1 AUC ", " L2 AUC ")
tb.add_row(["BOW ", 10**-3,10**-2,  67 ,70])
tb.add_row(["Tf - Idf ", 10**-3,10**-3,  66 ,70])
tb.add_row(["AVG - W2V", 10**-3,10**-3,  66 ,69])
tb.add_row(["AVG - Tf - Idf",10**-3,10**-2,  66 ,64])
tb.add_row(["SVD-Top 3000 Features", 10**-3,10**-3,  66 ,69])
print(tb.get_string(titles = "Observations for diffrent data metrix "))
```

```
+-----------------------+------------+------------+----------+----------+
|       Vectorizer      |  L1 Alpha  |  L2 Alpha  |  L1 AUC  |  L2 AUC  |
+-----------------------+------------+------------+----------+----------+
|          BOW          |   0.001    |    0.01    |    67    |    70    |
|       Tf - Idf        |   0.001    |   0.001    |    66    |    70    |
|       AVG - W2V       |   0.001    |   0.001    |    66    |    69    |
|     AVG - Tf - Idf    |   0.001    |    0.01    |    66    |    64    |
| SVD-Top 3000 Features |   0.001    |   0.001    |    66    |    69    |
+-----------------------+------------+------------+----------+----------+
```