DonorsChoose

DonorsChoose.org receives hundreds of thousands of project proposals each year for classroom number of volunteers is needed to manually screen each submission before it's approved to be po

Next year, DonorsChoose.org expects to receive close to 500,000 project proposals. As a result, th solve:

- How to scale current manual processes and resources to screen 500,000 projects so that th as possible
- How to increase the consistency of project vetting across different volunteers to improve th
- 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 sub the text of project descriptions as well as additional metadata about the project, teacher, and sch 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	Descriptio
`project_id`	A unique identifier for the proposed project. **Example:** `p036502`
`project_title`	Title of the project. **Examples:** • `Art Will Make You Happy!` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targeted. One of th • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories for the projec • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:** • `Music & The Arts` • `Literacy & Language, Math & Science`
`school_state`	State where school is located (Two-letter U.S. postal code). **Examp**

Feature	Descriptio
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the pr • `Literacy` • `Literature & Writing, Social Scienc`
`project_resource_summary`	An explanation of the resources needed for the project. **Example:** • `My students need hands on literacy`
`project_essay_1`	First application essay*
`project_essay_2`	Second application essay*
`project_essay_3`	Third application essay*
`project_essay_4`	Fourth application essay*
`project_submitted_datetime`	Datetime when project application was submitted. **Example:** `2016`
`teacher_id`	A unique identifier for the teacher of the proposed project. **Examp**
`teacher_prefix`	Teacher's title. One of the following enumerated values: • `nan` • `Dr.` • `Mr.` • `Mrs.` • `Ms.` • `Teacher.`
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted by the same

* 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 ea
resource required by a project:

Feature	Description
`id`	A `project_id` value from the `train.csv` file. **Example:** `p0365`
`description`	Desciption of the resource. **Example:** `Tenor Saxophone Reeds`
`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 tr
resources needed for a project:

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

Label	Description
`project_is_approved`	A binary flag indicating whether DonorsChoose approved the project. A value of `0` indicates the proje

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 f following:

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

For all projects with project_submitted_datetime of 2016-05-17 and later, the values of proje

```
# importing required libraries

%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.model_selection import GridSearchCV
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

import chart_studio.plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
from sklearn.model_selection import GridSearchCV
```

1.1 Reading Data

```
from google.colab import drive

# This will prompt for authorization.
drive.mount('/content/drive',force_remount=True)
```

```
Mounted at /content/drive
```

```
!ls "/content/drive/My Drive/Colab Notebooks/Dataset/Assignments_DonorsChoose_201
```

```
'06 Implement SGD.ipynb'                confusion_matrix.png
 10_DonorsChoose_Clustering.ipynb       cooc.JPG
 11_DonorsChoose_TruncatedSVD.ipynb     glove_vectors
 2_DonorsChoose_EDA_TSNE.ipynb          haberman.csv
 2letterstabbrev.pdf                    haberman.xlsx
 3d_plot.JPG                            heat_map.JPG
 3d_scatter_plot.ipynb                  imdb.txt
 4_DonorsChoose_NB.ipynb                resources.csv
 5_DonorsChoose_LR.ipynb                response.JPG
 7_DonorsChoose_SVM.ipynb               summary.JPG
 8_DonorsChoose_DT.ipynb                test_data.csv
 9_DonorsChoose_RF_GBDT.ipynb           train_cv_auc.JPG
 Assignment_SAMPLE_SOLUTION.ipynb       train_data.csv
'Assignment_tips(1).docx'               train_test_auc.JPG
 Assignment_tips.docx
```

```
# Reading data from project and resources data file

project_data = pd.read_csv('/content/drive/My Drive/Colab Notebooks/Dataset/Assigr
resource_data = pd.read_csv('/content/drive/My Drive/Colab Notebooks/Dataset/Assig
```

```
# Getting basic information about the data

print("Number of data points in Project_train data", project_data.shape)
print('-'*100)
print("The attributes of Project_train data :", project_data.columns.values)
print('='*100)
print("Number of data points in Resource_train data", resource_data.shape)
print('-'*100)
print("The attributes of Resource_train data :", resource_data.columns.values)
```

```
Number of data points in Project_train data (109248, 17)
------------------------------------------------------------------------------
The attributes of Project_train data : ['Unnamed: 0' 'id' 'teacher_id' 'teach
 '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']
==============================================================================
Number of data points in Resource_train data (1541272, 4)
------------------------------------------------------------------------------
The attributes of Resource_train data : ['id' 'description' 'quantity' 'price
```

1.2 Data Pre-Processing

```
# Merge two column text dataframe:
# Merge 4 essays into one:
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)

# Merge Price information from resource data to project data
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).re
project_data = pd.merge(project_data, price_data, on='id', how='left')

# find how many digits are present in each project_resource_summary coloumn
summary = list(project_data['project_resource_summary'].values)
presence_of_numeric_data=[]
for i in summary:
    count = 0
    for j in i.split(' '):
        if j.isdigit():
            count+=1
    presence_of_numeric_data.append(count)

# Replace Text summary coloumn with new numerical coloumn presence_of_numeric_data
project_data['numerical_data_in_resource_summary'] = presence_of_numeric_data
project_data.drop(['project_resource_summary'], axis=1, inplace=True)

# how to replace elements in list python: https://stackoverflow.com/a/2582163/4084
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(project_dat

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

# how to reorder columns pandas python: https://stackoverflow.com/a/13148611/40840
project_data = project_data[cols]
```

```
# https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.drop
# Here we drop 3 rows where teacher_prefix is having np.nan value
project_data.dropna(axis=0,subset=['teacher_prefix'], inplace=True)

project_data.head(2)
```

	Unnamed: 0	id	teacher_id	teacher_prefix	school
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	

1.2.1 Pre-Processing Essay Text

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

```
I have been fortunate enough to use the Fairy Tale STEM kits in my classroom
==================================================
I teach high school English to students with learning and behavioral disabili
==================================================
```

```
# 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"\'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: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "y
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'h
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that'
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has'
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because',
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'thr
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off'
            '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', "dic
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma',
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't
            'won', "won't", 'wouldn', "wouldn't"]


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

# Adding preprocessed_essays coloumn to our data matrix

project_data['preprocessed_essays']=preprocessed_essays
```

```
100%|██████████| 109245/109245 [01:05<00:00, 1660.78it/s]
```

```
# after preprocesing
preprocessed_essays[100]
```

```
'a typical day campus exciting my students love learning always put smile fac
```

1.2.2 Pre-Processing Project Title Text

```
from tqdm import tqdm
preprocessed_titles = []
# tqdm is for printing the status bar
```

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

# Adding preprocessed_titles coloumn to our data matrix

project_data['preprocessed_titles']=preprocessed_titles
preprocessed_titles[1000]
```

```
100%|██████████| 109245/109245 [00:02<00:00, 37556.52it/s]
'empowering students through art learning about then now'
```

1.2.3 Pre-Processing Project Grades

```
# Remove special characters from grades
from tqdm import tqdm
preprocessed_grade_categories = []
# tqdm is for printing the status bar
for categories in tqdm(project_data['project_grade_category'].values):
    categories = decontracted(categories)
    # https://gist.github.com/sebleier/554280
    categories = '_'.join(e for e in categories.split(' ') if e not in stopwords)
    categories = '_'.join(e for e in categories.split('-') if e not in stopwords)
    preprocessed_grade_categories.append(categories.lower().strip())

# Adding preprocessed_titles coloumn to our data matrix

project_data['preprocessed_grade_category']=preprocessed_grade_categories

project_data.head(5)
```

```
100%|██████████| 109245/109245 [00:02<00:00, 49316.31it/s]
```

	Unnamed: 0	id	teacher_id	teacher_prefix	school
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	
51140	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	

1.2.4 preprocessing of `project_subject_categories`

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

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

1.2.5 preprocessing of `project_subject_subcategories`

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com

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

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", "W
        if 'The' in j.split(): # this will split each of the catogory based on spa
            j=j.replace('The','') # if we have the words "The" we are going to rep
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty)
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailir
        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)
```

```
# Drop all unnecessary featurs like project_grade_category, project_essay_1, etc.
project_data.drop(['project_grade_category'], axis=1, inplace=True)
project_data.drop(['project_essay_1'], axis=1, inplace=True)
project_data.drop(['project_essay_2'], axis=1, inplace=True)
project_data.drop(['project_essay_3'], axis=1, inplace=True)
project_data.drop(['project_essay_4'], axis=1, inplace=True)
project_data.drop(['essay'], axis=1, inplace=True)
```

```
project_data.head(5)
```

	Unnamed: 0	id	teacher_id	teacher_prefix	school
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	
51140	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	

1.2.6 Add Sentiment Score of Preprocessed Essays

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

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
True
```

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
neg_essay=[]
neu_essay=[]
pos_essay=[]
comp_essay=[]

sid = SentimentIntensityAnalyzer()

for sent in preprocessed_titles:

    ss = sid.polarity_scores(sent)
    neg_essay.append(ss.get('neg'))
    neu_essay.append(ss.get('neu'))
    pos_essay.append(ss.get('pos'))
    comp_essay.append(ss.get('compound'))

project_data['neg_essay']=neg_essay
project_data['neu_essay']=neu_essay
project_data['pos_essay']=pos_essay
```

```
project_data['comp_essay']=comp_essay


# 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


project_data.head(5)
```

	Unnamed: 0	id	teacher_id	teacher_prefix	school
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	
51140	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	

1.2.7 Adding number of words in title and number of words in essay features

```
number_of_words_in_title=[]
for title in project_data['project_title'].values:
    list_of_words = title.split()
    number_of_words_in_title.append(len(list_of_words))

number_of_words_in_essays=[]
for title in project_data['preprocessed_essays'].values:
    list_of_words = title.split()
    number_of_words_in_essays.append(len(list_of_words))

project_data['number_of_words_in_title'] = number_of_words_in_title
project_data['number_of_words_in_essays'] = number_of_words_in_essays
```

```
project_data.head()
```

	Unnamed: 0	id	teacher_id	teacher_prefix	school
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	
51140	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	

1.3 Sampling data for SVM Assignment

```
project_data['project_is_approved'].value_counts()
```

```
1    92703
0    16542
Name: project_is_approved, dtype: int64
```

```
data = project_data
data['project_is_approved'].value_counts()
```

```
1    92703
0    16542
Name: project_is_approved, dtype: int64
```

```
data.head(5)
```

	Unnamed: 0	id	teacher_id	teacher_prefix	school
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	
51140	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	

```
# Split the class label from data
y = data['project_is_approved'].values
X = data.drop(['project_is_approved'], axis=1)
X.head(1)
```

	Unnamed: 0	id	teacher_id	teacher_prefix	school_
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	

2.1 Splitting data into Train and cross validation(or test): Str Upsampling

```
# train test split
# Not using CV data as it will be done by the GridsearchCV internally
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify
#X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33,
```

```
# Simple Upsampling for negative class data points in training dataset
# https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets
```

```
from sklearn.utils import resample
#df3 = pd.DataFrame(y_train,columns=['project_is_approved'],dtype = int)

#X = pd.concat([X_train,df3],axis = 1)
X_train['project_is_approved']=y_train
Accepted, Rejected = X_train.project_is_approved.value_counts()

# Divide by class
df_class_0 = X_train[X_train['project_is_approved'] == 0]
df_class_1 = X_train[X_train['project_is_approved'] == 1]

upsampled_data = df_class_0.sample(Accepted, replace=True,)
X_train = pd.concat([df_class_1, upsampled_data], axis=0)
print(X_train.project_is_approved.value_counts())
```

```
1    62111
0    62111
Name: project_is_approved, dtype: int64
```

```
y_train = X_train.project_is_approved
X_train = X_train.drop('project_is_approved', axis=1)
X_train.shape
```

```
(124222, 22)
```

2.2 Make Data Model Ready:

2.2.1 Encoding numerical, categorical features

2.2.1.1 Encoding School State

```
# Encoding School State

vectorizer = CountVectorizer()
vectorizer.fit(X_train['school_state'].values) # fit has to happen only on train c

# we use the fitted CountVectorizer to convert the text to vector
X_train_state_ohe = vectorizer.transform(X_train['school_state'].values)
#X_cv_state_ohe = vectorizer.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer.transform(X_test['school_state'].values)

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
#print(X_cv_state_ohe.shape, y_cv.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(124222, 51) (124222,)
(36051, 51) (36051,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia'
==============================================================================
```

2.2.1.2 Encoding Teacher Prefix

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['teacher_prefix'].values) # fit has to happen only on trair

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_ohe = vectorizer.transform(X_train['teacher_prefix'].values)
#X_cv_teacher_ohe = vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
#print(X_cv_teacher_ohe.shape, y_cv.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(124222, 5) (124222,)
(36051, 5) (36051,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
==============================================================================
```

2.2.1.3 Encoding preprocessed_grade_category

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['preprocessed_grade_category'].values) # fit has to happen

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer.transform(X_train['preprocessed_grade_category'].va
#X_cv_grade_ohe = vectorizer.transform(X_cv['preprocessed_grade_category'].values)
X_test_grade_ohe = vectorizer.transform(X_test['preprocessed_grade_category'].valu

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
#print(X_cv_grade_ohe.shape, y_cv.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(124222, 4) (124222,)
(36051, 4) (36051,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
==============================================================================
```

2.2.1.4 Encoding numerical feature Price

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['price'].values.reshape(1,-1))

X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(1,-1))
#X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(1,-1))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(1,-1))

X_train_price_norm = X_train_price_norm.reshape(-1,1)
X_test_price_norm = X_test_price_norm.reshape(-1,1)


print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
print(X_train_price_norm)
#print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_price_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(124222, 1) (124222,)
[[0.00104887]
 [0.00033864]
 [0.00155953]
 ...
 [0.00452251]
 [0.00085977]
 [0.00087059]]
(36051, 1) (36051,)
==============================================================================
```

2.2.1.5 Encoding numeric feature Quantity

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['quantity'].values.reshape(1,-1))

X_train_quantity_norm = normalizer.transform(X_train['quantity'].values.reshape(1,
X_train_quantity_norm = X_train_quantity_norm.reshape(-1,1)
#X_cv_quantity_norm = normalizer.transform(X_cv['quantity'].values.reshape(1,-1))
X_test_quantity_norm = normalizer.transform(X_test['quantity'].values.reshape(1,-1
X_test_quantity_norm = X_test_quantity_norm.reshape(-1,1)
print(X_train_quantity_norm)
print("After vectorizations")
print(X_train_quantity_norm.shape, y_train.shape)
#print(X_cv_quantity_norm.shape, y_cv.shape)
print(X_test_quantity_norm.shape, y_test.shape)
print("="*100)
```

```
[[0.00017229]
 [0.00034457]
 [0.00017229]
 ...
 [0.00077529]
 [0.00025843]
 [0.00758065]]
After vectorizations
(124222, 1) (124222,)
(36051, 1) (36051,)
==========================================================================================
```

2.2.1.6 Encoding numeric feature teacher_number_of_previously_posted_proje

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.resh
#List_of_imp_features.append('teacher_number_of_previously_posted_projects')
X_train_teacher_number_of_previously_posted_projects_norm = normalizer.transform(X
#X_cv_teacher_number_of_previously_posted_projects_norm = normalizer.transform(X_c
X_test_teacher_number_of_previously_posted_projects_norm = normalizer.transform(X_
X_train_teacher_number_of_previously_posted_projects_norm = X_train_teacher_number
X_test_teacher_number_of_previously_posted_projects_norm = X_test_teacher_number_c

print(X_test_teacher_number_of_previously_posted_projects_norm)
print("After vectorizations")
print(X_train_teacher_number_of_previously_posted_projects_norm.shape, y_train.sha
#print(X_cv_teacher_number_of_previously_posted_projects_norm.shape, y_cv.shape)
```

```
print(X_test_teacher_number_of_previously_posted_projects_norm.shape, y_test.shape
print("="*100)
```

```
[[0.00428271]
 [0.00068523]
 [0.00034262]
 ...
 [0.0229553 ]
 [0.        ]
 [0.00034262]]
After vectorizations
(124222, 1) (124222,)
(36051, 1) (36051,)
============================================================================
```

2.2.1.7 Encoding numeric feature numerical_data_in_resource_summary

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['numerical_data_in_resource_summary'].values.reshape(1,-1))
X_train_numerical_data_in_resource_summary_norm = normalizer.transform(X_train['nu
#X_cv_numerical_data_in_resource_summary_norm = normalizer.transform(X_cv['numeric
X_test_numerical_data_in_resource_summary_norm = normalizer.transform(X_test['nume

X_train_numerical_data_in_resource_summary_norm = X_train_numerical_data_in_resour
X_test_numerical_data_in_resource_summary_norm = X_test_numerical_data_in_resource

print(X_test_numerical_data_in_resource_summary_norm)
print("After vectorizations")
print(X_train_numerical_data_in_resource_summary_norm.shape, y_train.shape)
#print(X_cv_numerical_data_in_resource_summary_norm.shape, y_cv.shape)
print(X_test_numerical_data_in_resource_summary_norm.shape, y_test.shape)
print("="*100)
```

```
[[0.]
 [0.]
 [0.]
 ...
 [0.]
 [0.]
 [0.]]
After vectorizations
(124222, 1) (124222,)
(36051, 1) (36051,)
============================================================================
```

2.2.1.8 Encoding numeric feature number_of_words_in_title

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['number_of_words_in_title'].values.reshape(1,-1))

X_train_number_of_words_in_title = normalizer.transform(X_train['number_of_words_i
#X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(1,-1))
X_test_number_of_words_in_title = normalizer.transform(X_test['number_of_words_in_

X_train_number_of_words_in_title = X_train_number_of_words_in_title.reshape(-1,1)
X_test_number_of_words_in_title = X_test_number_of_words_in_title.reshape(-1,1)


print("After vectorizations")
print(X_train_number_of_words_in_title.shape, y_train.shape)
print(X_train_number_of_words_in_title)
#print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_number_of_words_in_title.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(124222, 1) (124222,)
[[0.00256309]
 [0.00358833]
 [0.00358833]
 ...
 [0.00307571]
 [0.00307571]
 [0.00256309]]
(36051, 1) (36051,)
=============================================================================
```

2.2.1.9 Encoding numeric feature number_of_words_in_essay

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['number_of_words_in_essays'].values.reshape(1,-1))

X_train_number_of_words_in_essay = normalizer.transform(X_train['number_of_words_i
#X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(1,-1))
```

```
X_test_number_of_words_in_essay = normalizer.transform(X_test['number_of_words_in_

X_train_number_of_words_in_essay = X_train_number_of_words_in_essay.reshape(-1,1)
X_test_number_of_words_in_essay = X_test_number_of_words_in_essay.reshape(-1,1)


print("After vectorizations")
print(X_train_number_of_words_in_essay.shape, y_train.shape)
print(X_train_number_of_words_in_essay)
#print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_number_of_words_in_essay.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(124222, 1) (124222,)
[[0.00240664]
 [0.00253623]
 [0.00386914]
 ...
 [0.00233259]
 [0.00225854]
 [0.00253623]]
(36051, 1) (36051,)
=============================================================================
```

2.2.1.10 Encoding numeric features of sentiment Score

```
train_neg_essay = X_train['neg_essay'].values.reshape(-1,1)
test_neg_essay = X_test['neg_essay'].values.reshape(-1,1)

train_neu_essay = X_train['neu_essay'].values.reshape(-1,1)
test_neu_essay = X_test['neu_essay'].values.reshape(-1,1)

train_pos_essay = X_train['pos_essay'].values.reshape(-1,1)
test_pos_essay = X_test['pos_essay'].values.reshape(-1,1)

train_comp_essay = X_train['comp_essay'].values.reshape(-1,1)
test_comp_essay = X_test['comp_essay'].values.reshape(-1,1)
```

2.3 Appling SVM on different kind of featurization as mentio

```
#Setting the values for alpha
import math
a=[]
for i in range(-4,4):
    a.append(10**i)

log_alpha = []
for i in a:
    log_alpha.append(math.log(i,10))
```

```
print(a)
```

```
[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
```

```
# Define Functions for Train LR model, Test LR Model and Plot the graphs for diffe

import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.metrics import roc_auc_score
from sklearn.calibration import CalibratedClassifierCV

def train_svm(X_tr,y_train):

    train_score=[]
    test_score=[]

    svm = linear_model.SGDClassifier(loss='hinge',class_weight="balanced")
    #create a dictionary of all values we want to test for alpha values
    param_grid = {'alpha': a}
#use gridsearch to test all values for alpha
    svm_gscv = GridSearchCV(svm, param_grid, cv=2, scoring='roc_auc', return_trair
    svm_gscv.fit(X_tr, y_train)
    print(svm_gscv.best_params_)

    print(svm_gscv.cv_results_.keys())
    for key, value in svm_gscv.cv_results_.items():
        if key == "mean_train_score":
            train_score = value
        if key == "mean_test_score":
            test_score = value


    plt.plot(log_alpha, train_score, label='Train AUC')
    plt.plot(log_alpha, test_score, label='CV AUC')

    plt.scatter(log_alpha, train_score, label='Train AUC points')
    plt.scatter(log_alpha, test_score, label='CV AUC points')

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

# Test the model with optimal alpha found out using training data. Plot FPR vs TPF

def test_svm(X_tr,X_te,best_a):

    y_train_pred=[]
    y_test_pred=[]

    from sklearn.metrics import roc_curve, auc
    svm = linear_model.SGDClassifier(loss='hinge',class_weight='balanced')
    clf=svm.fit(X_tr, y_train)

    # https://stackoverflow.com/questions/39200265/attributeerror-probability-esti

    calibrator = CalibratedClassifierCV(clf, cv='prefit')
    model=calibrator.fit(X_tr, y_train)

    y_train_pred_raw = model.predict_proba(X_tr)

    y_test_pred_raw = model.predict_proba(X_te)


    for i in y_train_pred_raw:
        y_train_pred.append(i[1])
    for i in y_test_pred_raw:
        y_test_pred.append(i[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_tp
    plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
    plt.legend()
    plt.xlabel("tpr")
    plt.ylabel("fpr")
    plt.title("ERROR PLOTS")
    plt.grid()
    plt.show()
    return train_fpr,train_tpr,tr_thresholds,y_train_pred,y_test_pred

# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", n
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

2.3.1 Applying SVM on BOW encoding eassay, and project_title

2.3.1.1 Encoding preprocessed_essays BoW

```
print(X_train.shape, y_train.shape)
#print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)


vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=10000)
vectorizer.fit(X_train['preprocessed_essays'].values) # fit has to happen only on


# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vectorizer.transform(X_train['preprocessed_essays'].values)
#X_cv_essay_bow = vectorizer.transform(X_cv['preprocessed_essays'].values)
X_test_essay_bow = vectorizer.transform(X_test['preprocessed_essays'].values)

print("After vectorizations")
print(X_train_essay_bow.shape, y_train.shape)
#print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)
print("="*100)


# print("YOU SHOULD NOT DO SOMETHING LIKE THIS")
# vectorizer = CountVectorizer()
# x_train_bow = vectorizer.fit_transform(X_train['essay'].values)
# x_cv_bow = vectorizer.fit_transform(X_cv['essay'].values)
# x_test_bow = vectorizer.fit_transform(X_test['essay'].values)

# 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("NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME")
```

```
(124222, 22) (124222,)
(36051, 22) (36051,)
============================================================================
After vectorizations
(124222, 10000) (124222,)
(36051, 10000) (36051,)
============================================================================
NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME
```

2.3.1.2 Encoding preprocessed_titles BoW

```
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=10000)
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=10000)
vectorizer.fit(X_train['preprocessed_titles'].values) # fit has to happen only on

# we use the fitted CountVectorizer to convert the text to vector
X_train_titles_bow = vectorizer.transform(X_train['preprocessed_titles'].values)
#X_cv_titles_bow = vectorizer.transform(X_cv['preprocessed_titles'].values)
X_test_titles_bow = vectorizer.transform(X_test['preprocessed_titles'].values)

print("After vectorizations")
print(X_train_titles_bow.shape, y_train.shape)
#print(X_cv_titles_bow.shape, y_cv.shape)
print(X_test_titles_bow.shape, y_test.shape)
print("="*100)


# print("YOU SHOULD NOT DO SOMETHING LIKE THIS")
# vectorizer = CountVectorizer()
# x_train_bow = vectorizer.fit_transform(X_train['essay'].values)
# x_cv_bow = vectorizer.fit_transform(X_cv['essay'].values)
# x_test_bow = vectorizer.fit_transform(X_test['essay'].values)

# 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("NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME")
```

```
After vectorizations
(124222, 10000) (124222,)
(36051, 10000) (36051,)
==============================================================================
NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME
```

2.3.1.3 Merge all the features and obtain final data matrix

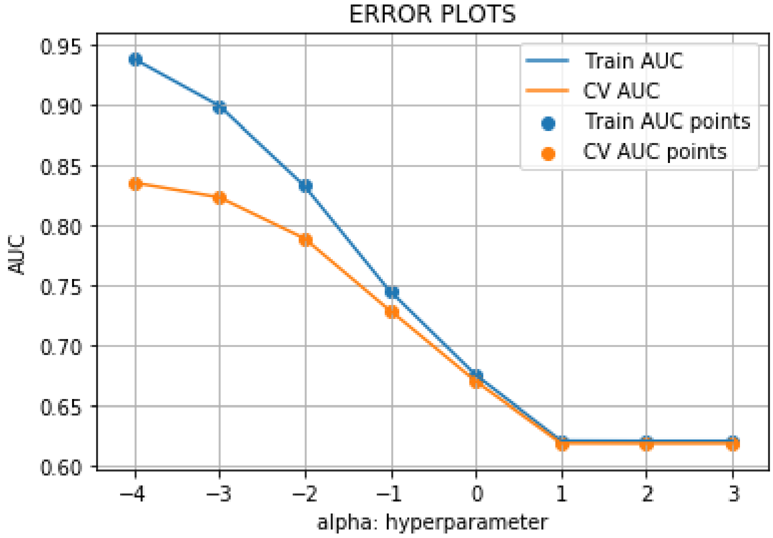
```
# Merge all the features:
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
# Matrix are merged in such a way that the order is preserved in # Matrix are merg


from scipy.sparse import hstack
X_tr = hstack((X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_train_
#X_cr = hstack((X_cv_titles_bow,X_cv_essay_bow, X_cv_state_ohe, X_cv_teacher_ohe,
X_te = hstack((X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_pric

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
#print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(124222, 20064) (124222,)
(36051, 20064) (36051,)
==============================================================================
```

2.3.1.4 Training the data model and find best hyperparameter using ROC-AUC

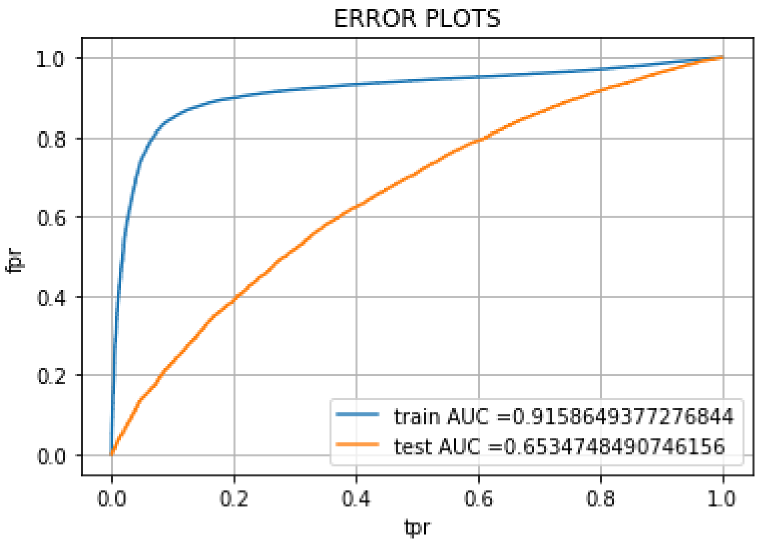
```
# Call train_svm function on above data

train_svm(X_tr,y_train)
```

```
{'alpha': 0.0001}
dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_tim
```

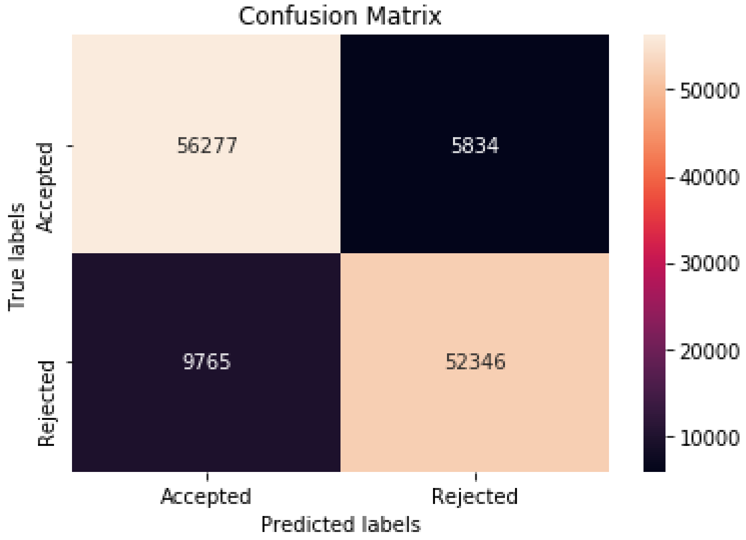


2.3.1.5 Testing the performance of the model on test data, plotting ROC Curve

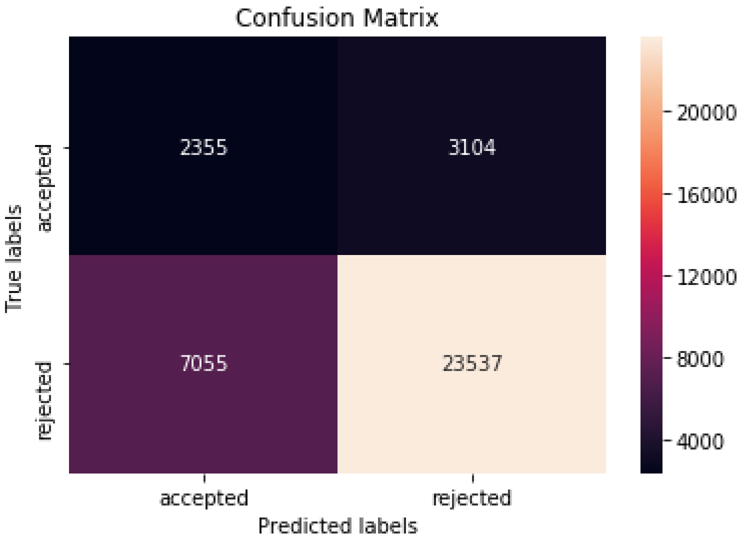
```
# Call test_svm for a obtained by training the data

best_a=0.01
train_fpr,train_tpr,tr_thresholds,y_train_pred,y_test_pred=test_svm(X_tr,X_te,best
```



```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")

ax= plt.subplot()
cm=confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
print(cm)
sns.heatmap(cm, annot=True, ax = ax,fmt='d'); #annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['Accepted', 'Rejected']); ax.yaxis.set_ticklabels(['Accep
============================================================================
the maximum value of tpr*(1-fpr) 0.7636201782783456 for threshold 0.478
Train confusion matrix
[[56277  5834]
 [ 9765 52346]]
```



```
est confusion matrix")


= confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
_test)
subplot()
map(cm_test, annot=True, ax = ax,fmt='d'); #annot=True to annotate cells

, title and ticks
label('Predicted labels');ax.set_ylabel('True labels');
itle('Confusion Matrix');
.set_ticklabels(['accepted', 'rejected']); ax.yaxis.set_ticklabels(['accepted', 'r
Test confusion matrix
[[ 2355  3104]
 [ 7055 23537]]
```



2.3.2 Applying SVM on TFIDF encoding eassay, and project_title

2.3.2.1 Encoding preprocessed_titles TFIDF

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)

#vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=10000)
vectorizer.fit(X_train['preprocessed_titles'].values) # fit has to happen only on

# we use the fitted CountVectorizer to convert the text to vector
X_train_titles_tfidf = vectorizer.transform(X_train['preprocessed_titles'].values)
#X_cv_titles_tfidf = vectorizer.transform(X_cv['preprocessed_titles'].values)
X_test_titles_tfidf = vectorizer.transform(X_test['preprocessed_titles'].values)

print("After vectorizations")
print(X_train_titles_tfidf.shape, y_train.shape)
#print(X_cv_titles_tfidf.shape, y_cv.shape)
print(X_test_titles_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(124222, 3736) (124222,)
(36051, 3736) (36051,)
=============================================================================
```

2.3.2.2 Encoding preprocessed_essays TFIDF

```
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=10000)
vectorizer.fit(X_train['preprocessed_essays'].values) # fit has to happen only on

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_tfidf = vectorizer.transform(X_train['preprocessed_essays'].values)
#X_cv_essay_tfidf = vectorizer.transform(X_cv['preprocessed_essays'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['preprocessed_essays'].values)

print("After vectorizations")
print(X_train_essay_tfidf.shape, y_train.shape)
#print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(124222, 10000) (124222,)
(36051, 10000) (36051,)
=============================================================================
```

2.3.2.3 Merge all the features and obtain final data matrix

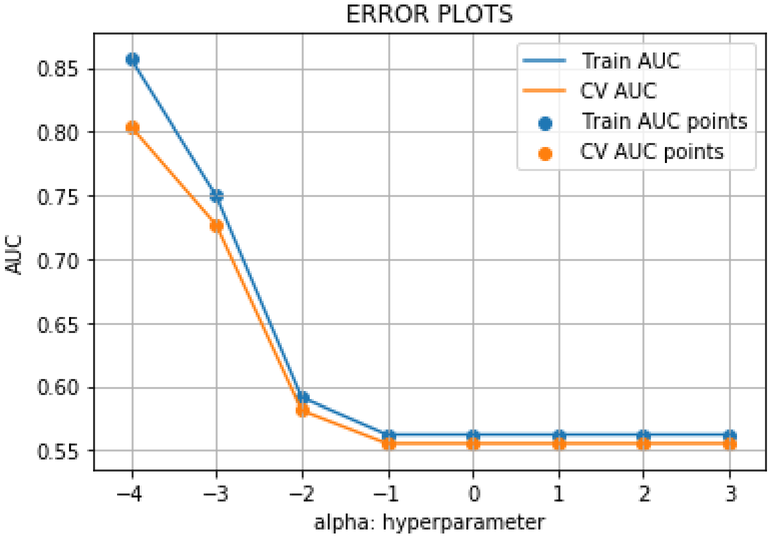
```
from scipy.sparse import hstack
X_tr = hstack((X_train_titles_tfidf,X_train_essay_tfidf, X_train_state_ohe, X_trai
from scipy.sparse import hstack
X_tr = hstack((X_train_titles_tfidf,X_train_essay_tfidf, X_train_state_ohe, X_trai
#X_cr = hstack((X_cv_titles_tfidf,X_cv_essay_tfidf, X_cv_state_ohe, X_cv_teacher_c
X_te = hstack((X_test_titles_tfidf,X_test_essay_tfidf, X_test_state_ohe, X_test_te

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
#print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(124222, 13800) (124222,)
(36051, 13800) (36051,)
=============================================================================
```

2.3.2.4 Training the data model and find best hyperparameter using ROC-AUC

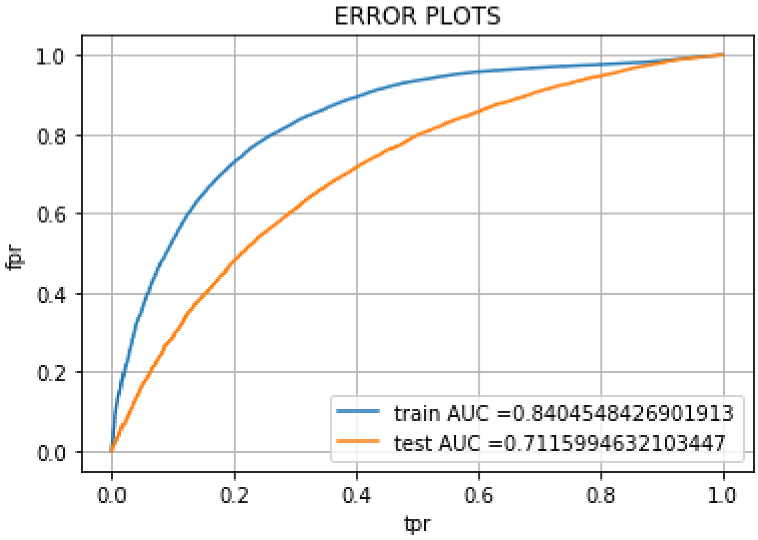
```
# Call train_svm function on above data

train_svm(X_tr,y_train)
```

```
{'alpha': 0.0001}
dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_tim
```



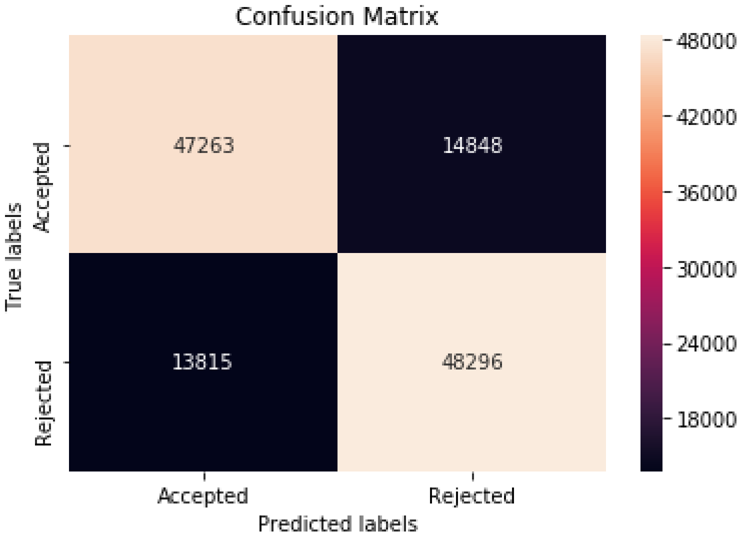
2.3.2.5 Testing the performance of the model on test data, plotting ROC Curve

```
best_a=0.001
train_fpr,train_tpr,tr_thresholds,y_train_pred,y_test_pred=test_svm(X_tr,X_te,best
```

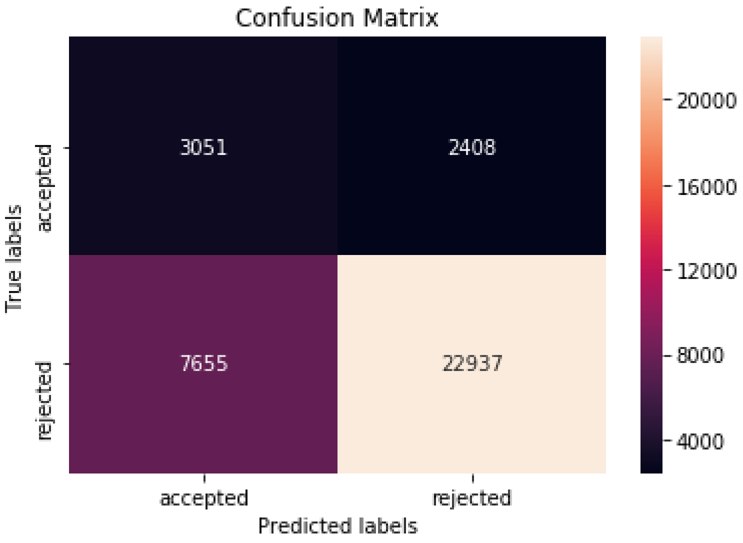


```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")

ax= plt.subplot()
cm=confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
print(cm)
sns.heatmap(cm, annot=True, ax = ax,fmt='d'); #annot=True to annotate cells

# labels, title and ticks
# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['Accepted', 'Rejected']); ax.yaxis.set_ticklabels(['Accep
```

```
===========================================================================
the maximum value of tpr*(1-fpr) 0.5916916010849247 for threshold 0.474
Train confusion matrix
[[47263 14848]
 [13815 48296]]
```



```
print("Test confusion matrix")


cm_test = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
print(cm_test)
ax= plt.subplot()
sns.heatmap(cm_test, annot=True, ax = ax,fmt='d'); #annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['accepted', 'rejected']); ax.yaxis.set_ticklabels(['accep
Test confusion matrix
[[ 3051  2408]
 [ 7655 22937]]
```



2.3.3 Applying SVM on AVG W2V

2.3.3.1 Encoding preprocessed_essays AVG W2V

```
with open('/content/drive/My Drive/Colab Notebooks/Dataset/Assignments_DonorsChoos
    model = pickle.load(f)
    glove_words =  set(model.keys())


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

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

```
100%|██████████| 124222/124222 [00:48<00:00, 2544.39it/s]124222
300
```

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

```
100%|██████████| 36051/36051 [00:14<00:00, 2432.05it/s]36051
```

2.3.3.2 Encoding preprocessed_titles AVG W2V

```
avg_w2v_vectors_titles_train = []; # the avg-w2v for each sentence/review is store
for sentence in tqdm(X_train['preprocessed_titles'].values): # for each review/ser
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_titles_train.append(vector)
print(len(avg_w2v_vectors_titles_train))
```

```
100%|██████████| 124222/124222 [00:02<00:00, 49449.54it/s]124222
```

```
avg_w2v_vectors_titles_test = []; # the avg-w2v for each sentence/review is storec
for sentence in tqdm(X_test['preprocessed_titles'].values): # for each review/sent
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_titles_test.append(vector)
```

```
100%|██████████| 36051/36051 [00:00<00:00, 45243.35it/s]
```

2.3.3.3 Merge all the features and obtain final data matrix

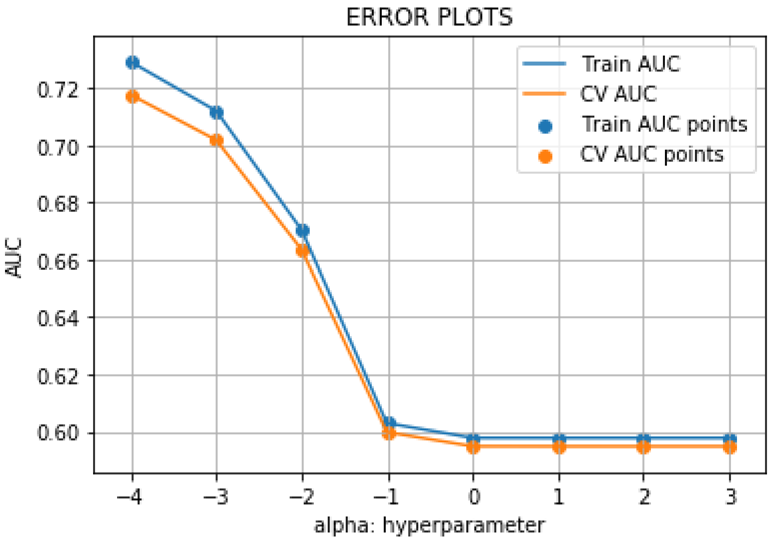
```
from scipy.sparse import hstack
X_tr = hstack((avg_w2v_vectors_titles_train,avg_w2v_vectors_essays_train, X_train_
#X_cr = hstack((avg_w2v_vectors_titles_cv,avg_w2v_vectors_cv, X_cv_state_ohe, X_cv
X_te = hstack((avg_w2v_vectors_titles_test,avg_w2v_vectors_essays_test, X_test_sta

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
#print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(124222, 664) (124222,)
(36051, 664) (36051,)
=============================================================================
```

2.3.3.4 Training the data model and find best hyperparameter using ROC-AUC

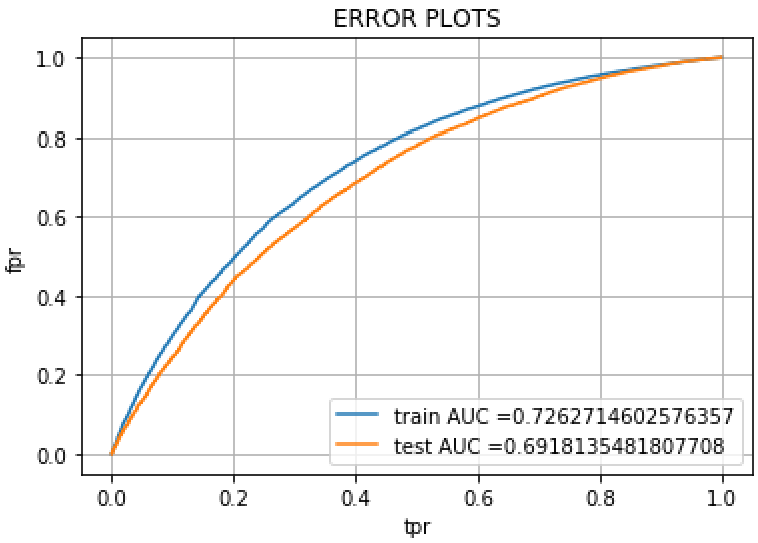
```
# Call train_svm function on above data

train_svm(X_tr,y_train)
```

```
{'alpha': 0.0001}
dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_tim
```

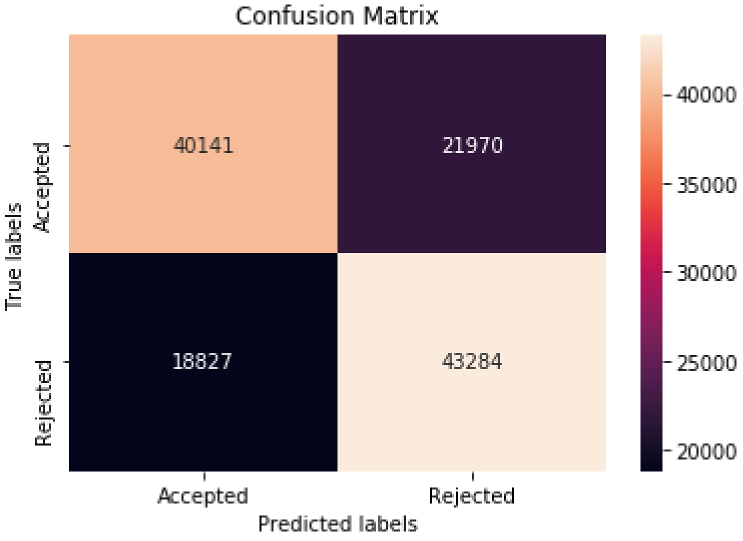


2.3.3.5 Testing the performance of the model on test data, plotting ROC Curve

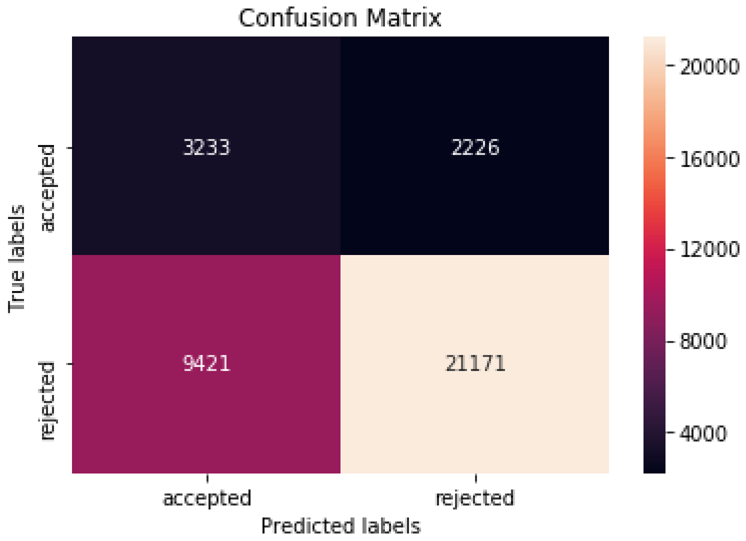
```
best_a=0.0001
train_fpr,train_tpr,tr_thresholds,y_train_pred,y_test_pred=test_svm(X_tr,X_te,best
```



```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")

ax= plt.subplot()
cm=confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
print(cm)
sns.heatmap(cm, annot=True, ax = ax,fmt='d'); #annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['Accepted', 'Rejected']); ax.yaxis.set_ticklabels(['Accep
============================================================================
the maximum value of tpr*(1-fpr) 0.4503794153492084 for threshold 0.478
Train confusion matrix
[[40141 21970]
 [18827 43284]]
```



```
print("Test confusion matrix")


cm_test = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
print(cm_test)
ax= plt.subplot()
sns.heatmap(cm_test, annot=True, ax = ax,fmt='d'); #annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['accepted', 'rejected']); ax.yaxis.set_ticklabels(['accep
Test confusion matrix
[[ 3233  2226]
 [ 9421 21171]]
```



2.3.4 Applying SVM on TFIDF W2V

2.3.4.1 Encoding preprocessed_titles tfidf W2V

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['preprocessed_titles'])
# 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())

# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_titles_train = []; # the avg-w2v for each sentence/review is stc
for sentence in tqdm(X_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
    tfidf_w2v_vectors_titles_train.append(vector)

print(len(tfidf_w2v_vectors_titles_train))
print(len(tfidf_w2v_vectors_titles_train[0]))
```

```
100%|██████████| 124222/124222 [00:05<00:00, 22253.16it/s]124222
300
```

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_titles_test = []; # the avg-w2v for each sentence/review is stor
for sentence in tqdm(X_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
    tfidf_w2v_vectors_titles_test.append(vector)

print(len(tfidf_w2v_vectors_titles_test))
print(len(tfidf_w2v_vectors_titles_test[0]))
```

```
100%|██████████| 36051/36051 [00:01<00:00, 25012.09it/s]36051
300
```

2.3.4.2 Encoding preprocessed_essays tfidf W2V

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

# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_essays_train = []; # the avg-w2v for each sentence/review is stc
for sentence in tqdm(X_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
    tfidf_w2v_vectors_essays_train.append(vector)

print(len(tfidf_w2v_vectors_essays_train))
print(len(tfidf_w2v_vectors_essays_train[0]))
```

```
100%|██████████| 124222/124222 [04:42<00:00, 440.11it/s]124222
300
```

```
tfidf_w2v_vectors_essays_test = []; # the avg-w2v for each sentence/review is stor
for sentence in tqdm(X_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
    tfidf_w2v_vectors_essays_test.append(vector)

print(len(tfidf_w2v_vectors_essays_test))
print(len(tfidf_w2v_vectors_essays_test[0]))
```

```
100%|██████████| 36051/36051 [01:24<00:00, 426.87it/s]36051
300
```

2.3.4.3 Merge all the features and obtain final data matrix

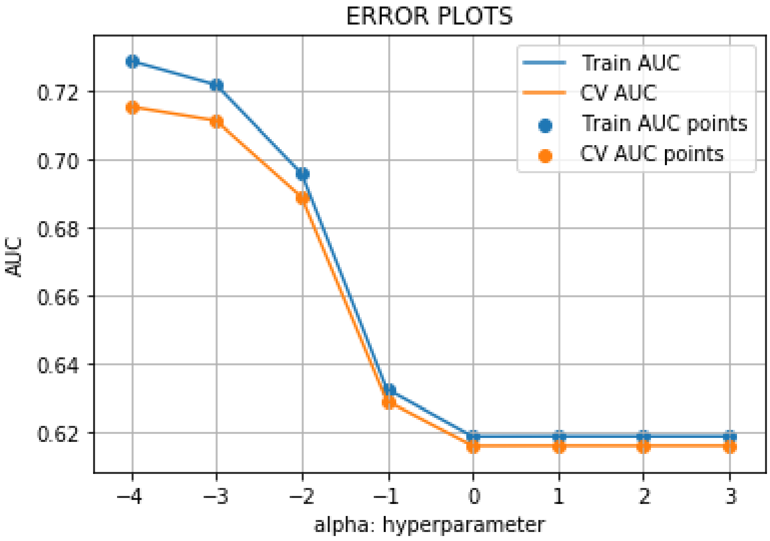
```
from scipy.sparse import hstack
X_tr = hstack((tfidf_w2v_vectors_titles_train,tfidf_w2v_vectors_essays_train, X_tr
#X_cr = hstack((tfidf_w2v_vectors_titles_cv,tfidf_w2v_vectors_essays_cv, X_cv_stat
X_te = hstack((tfidf_w2v_vectors_titles_test,tfidf_w2v_vectors_essays_test, X_test

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
#print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(124222, 664) (124222,)
(36051, 664) (36051,)
==============================================================================
```

2.3.4.4 Training the data model and find best hyperparameter using ROC-AUC

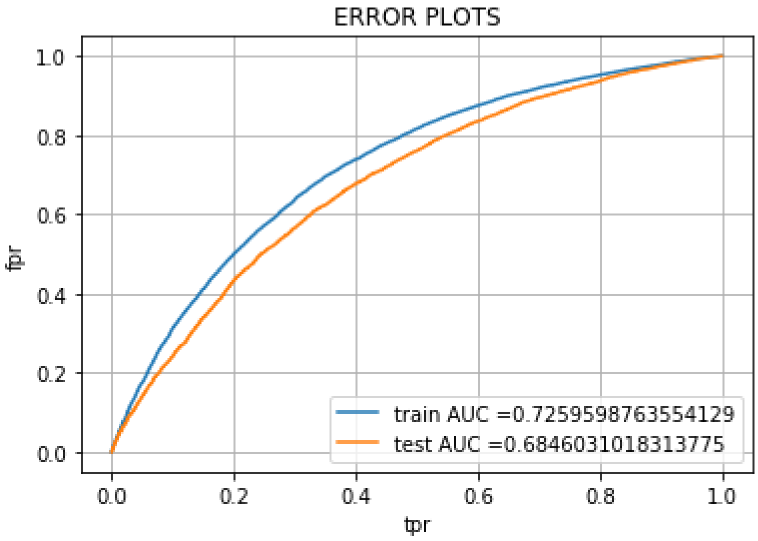
```
# Call train_svm function on above data

train_svm(X_tr,y_train)
```

```
{'alpha': 0.0001}
dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_tim
```



2.3.4.5 Testing the performance of the model on test data, plotting ROC Curve

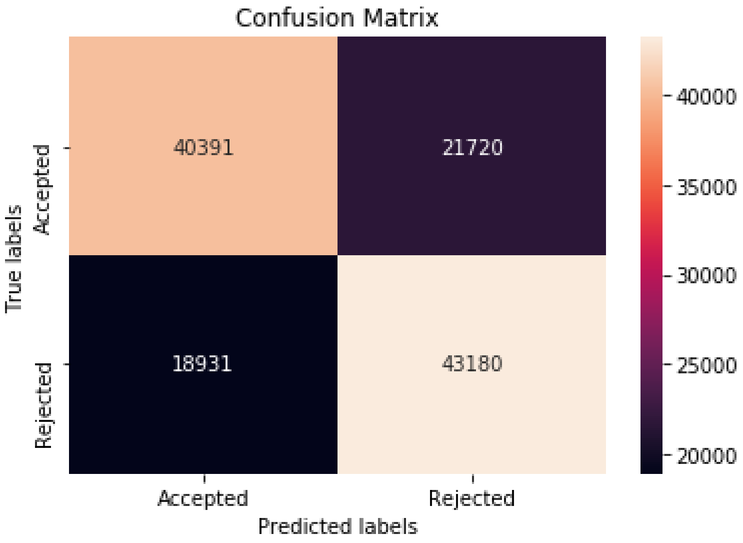
```
# Call test_LR for a obtained by training the data

best_a=0.001
train_fpr,train_tpr,tr_thresholds,y_train_pred,y_test_pred=test_svm(X_tr,X_te,best
```



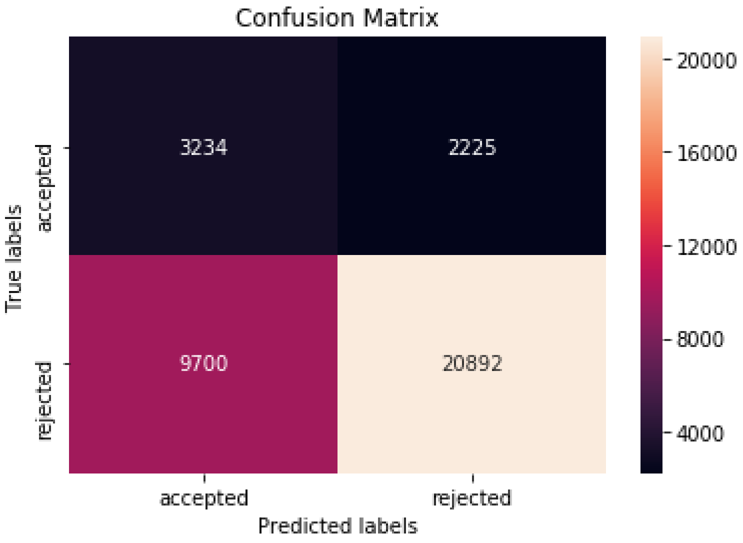
```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")

ax= plt.subplot()
cm=confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
print(cm)
```

```
sns.heatmap(cm, annot=True, ax = ax,fmt='d'); #annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['Accepted', 'Rejected']); ax.yaxis.set_ticklabels(['Accep
```

```
================================================================================
the maximum value of tpr*(1-fpr) 0.45209551691890326 for threshold 0.471
Train confusion matrix
[[40391 21720]
 [18931 43180]]
```



```
print("Test confusion matrix")


cm_test = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
print(cm_test)
ax= plt.subplot()
sns.heatmap(cm_test, annot=True, ax = ax,fmt='d'); #annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['accepted', 'rejected']); ax.yaxis.set_ticklabels(['accep
Test confusion matrix
[[ 3234  2225]
 [ 9700 20892]]
```



2.3.5 Applying SVM on only numerical data and essay text with feat

```
import nltk
nltk.downloader.download('vader_lexicon')
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
True
```

```
X_train_essay_tfidf_svd = X_train_essay_tfidf[:50000,:]
X_train_essay_tfidf_svd.shape
```

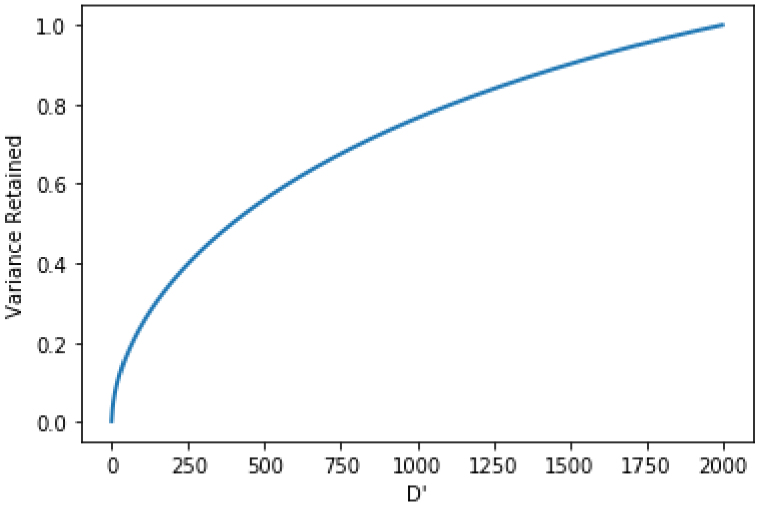
```
(50000, 10000)
```

2.3.5.1 Apply truncated SVD on essay text

```
from sklearn.decomposition import TruncatedSVD
```

```
svd = TruncatedSVD(n_components=2000)
svd_data= svd.fit_transform(X_train_essay_tfidf_svd)
percentage_var_explained = svd.explained_variance_ / np.sum(svd.explained_variance
cum_var_explained = np.cumsum(percentage_var_explained)

   # df = svd.fit_transform(X_train_essay_tfidf)
    #var.append(svd.explained_variance_ratio_.sum())

#plt.figure()
#plt.plot(var,n_comp)
#plt.xlabel("Variance retained")
#plt.ylabel("D'")
#plt.show()
```

```
plt.figure(1, figsize=(6, 4))
plt.plot(cum_var_explained, linewidth=2)
plt.xlabel("D'")
plt.ylabel("Variance Retained")
plt.show()
```



2.3.5.2 Merging all the non_text Features

```
X_train_essay_tfidf_new = X_train_essay_tfidf[:,:1750]
X_test_essay_tfidf_new = X_test_essay_tfidf[:,:1750]
```

```
from scipy.sparse import hstack
X_tr = hstack((X_train_essay_tfidf_new,X_train_state_ohe, X_train_teacher_ohe, X_t
#X_cr = hstack((X_cv_titles_tfidf,X_cv_essay_tfidf, X_cv_state_ohe, X_cv_teacher_c
X_te = hstack((X_test_essay_tfidf_new,X_test_state_ohe, X_test_teacher_ohe, X_test

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
#print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
'''
neg_essay= np.asarray(neg_essay)
neg_essay = neg_essay.reshape(-1,1)

neu_essay=np.asarray(neu_essay)
neu_essay = neu_essay.reshape(-1,1)

pos_essay=np.asarray(pos_essay)
pos_essay = pos_essay.reshape(-1,1)

comp_essay=np.asarray(comp_essay)
comp_essay = comp_essay.reshape(-1,1)
```

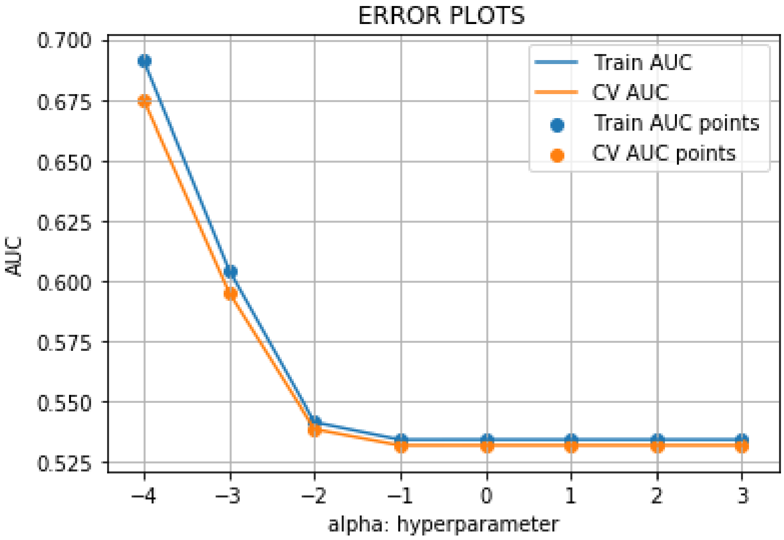
```
print(comp_essay)
'''
```

```
Final Data matrix
(124222, 1819) (124222,)
(36051, 1819) (36051,)
==============================================================================
'\nneg_essay= np.asarray(neg_essay)\nneg_essay = neg_essay.reshape(-1,1)\n\nn
```

2.3.5.3 Training the data model and find best hyperparameter using ROC-AUC

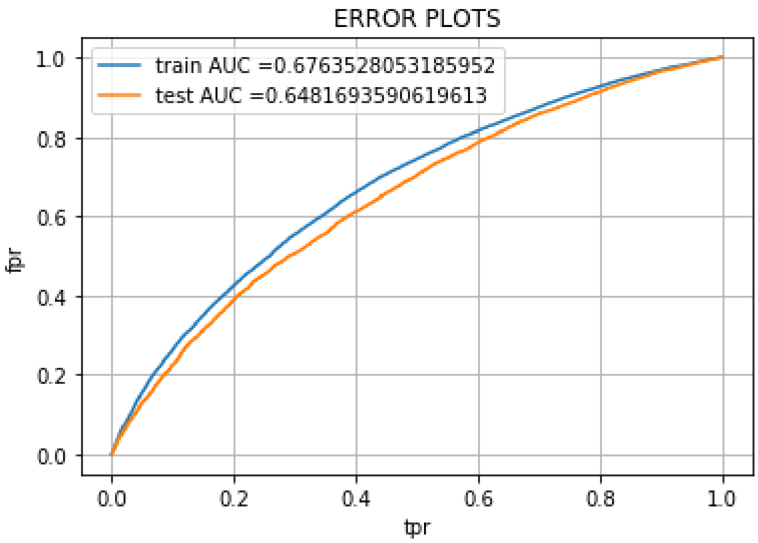
```
# Call train_svm function on above data

train_svm(X_tr,y_train)
```

```
{'alpha': 0.0001}
dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_tim
```

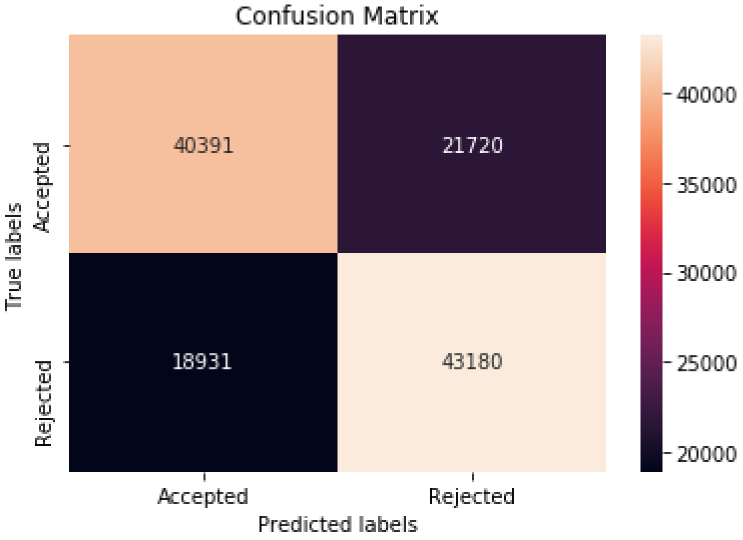


2.3.5.4 Testing the performance of the model on test data, plotting ROC Curve

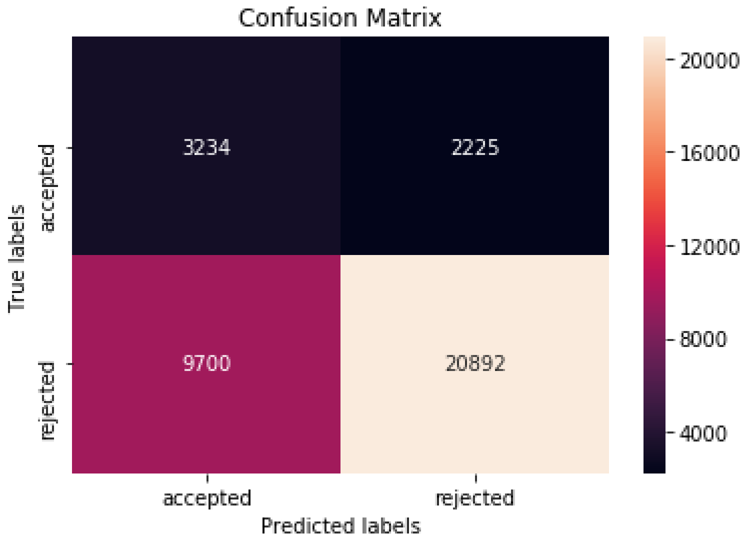
```
best_a=0.0001
train_fpr,train_tpr,tr_thresholds,y_train_pred,y_test_pred=test_svm(X_tr,X_te,best
```



```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")

ax= plt.subplot()
cm=confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
print(cm)
sns.heatmap(cm, annot=True, ax = ax,fmt='d'); #annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['Accepted', 'Rejected']); ax.yaxis.set_ticklabels(['Accep
============================================================================
the maximum value of tpr*(1-fpr) 0.45209551691890326 for threshold 0.471
Train confusion matrix
[[40391 21720]
 [18931 43180]]
```



```
print("Test confusion matrix")


cm_test = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
print(cm_test)
ax= plt.subplot()
sns.heatmap(cm_test, annot=True, ax = ax,fmt='d'); #annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['accepted', 'rejected']); ax.yaxis.set_ticklabels(['accep
Test confusion matrix
[[ 3234  2225]
 [ 9700 20892]]
```



2.4 Summary

```
# To summarize the results:
# summary table in jupyter notebook
# http://zetcode.com/python/prettytable/
# https://stackoverflow.com/questions/35160256/how-do-i-output-lists-as-a-table-ir

from prettytable import PrettyTable

x = PrettyTable(header_color='\033[40m')

x.field_names = ["Vectorizer", "Model", "Hyperparameter", "Train_AUC","Test_AUC"]

x.add_row(["Bag of Words", "SVM",0.01 ,0.91, 0.65])
x.add_row(["TF-IDF", "SVM", 0.001,0.84,0.71])
x.add_row(["Avg_W2V", "SVM", 0.0001,0.72,0.69])
x.add_row(["TF-IDF W2V", "SVM" , 0.0001,0.72,0.68])
x.add_row(["Numerical Features", "SVM", 0.0001,0.67,0.64])


print(x)
```

```
+--------------------+-------+----------------+-----------+----------+
|     Vectorizer     | Model | Hyperparameter | Train_AUC | Test_AUC |
+--------------------+-------+----------------+-----------+----------+
|    Bag of Words    |  SVM  |      0.01      |    0.91   |   0.65   |
|       TF-IDF       |  SVM  |     0.001      |    0.84   |   0.71   |
|      Avg_W2V       |  SVM  |     0.0001     |    0.72   |   0.69   |
|     TF-IDF W2V     |  SVM  |     0.0001     |    0.72   |   0.68   |
| Numerical Features |  SVM  |     0.0001     |    0.67   |   0.64   |
+--------------------+-------+----------------+-----------+----------+
```