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 scho
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 pro • `Literacy` • `Literature & Writing, Social Scienc`
`project_resource_summary`	An explanation of the resources needed for the project. **Example:** • `My students need hands on literacy r`
`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. **Exampl**
`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 youı 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.ensemble import RandomForestClassifier
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
from scipy import sparse
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 xgboost import XGBClassifier


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)
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client

Enter your authorization code:
··········
Mounted at /content/drive
```

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

```
 01s_Introduction_to_Google_Colab.ipynb
 5_DonorsChoose_LR.ipynb
'Copy of SVM.ipynb'
 GBDT.ipynb
 glove_vectors
'parikshitgune@gmail (1).com_Assignment_3.ipynb'
 parikshitgune@gmail.com_Assignment_3.ipynb
 parikshitgune@gmail.com_Assignment_4.ipynb
 parikshitgunegmail.com_Assignment_4.ipynb
 resources.csv
 SVM.ipynb
 train_data.csv
 Untitled
'Untitled (1)'
```

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

project_data = pd.read_csv('/content/drive/My Drive/Colab Notebooks/train_data.csv
resource_data = pd.read_csv('/content/drive/My Drive/Colab Notebooks/resources.csv
```

```
# 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 (50000, 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
# 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_
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	

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 recently read an article about giving students a choice about how they lear
==================================================
At the beginning of every class we start out with a Math Application problem 
==================================================
```

```
# 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
    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', "did
            "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%|██████████| 49998/49998 [00:27<00:00, 1833.83it/s]
```

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

```
'linda kranz wrote only one you she felt there one great big world make bette
```

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%|██████████| 49998/49998 [00:01<00:00, 42950.74it/s]
'comfy carpet creative learning'
```

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%|██████████| 49998/49998 [00:00<00:00, 55271.31it/s]
```

	Unnamed: 0	id	teacher_id	teacher_prefix	school_
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	
29891	146723	p099708	c0a28c79fe8ad5810da49de47b3fb491	Mrs.	
23374	72317	p087808	598621c141cda5fb184ee7e8ccdd3fcc	Ms.	
49228	57854	p099430	4000cfe0c8b2df75a218347c1765e283	Ms.	

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-in
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)
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailin
        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-in

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 trailin
        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
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	
29891	146723	p099708	c0a28c79fe8ad5810da49de47b3fb491	Mrs.	
23374	72317	p087808	598621c141cda5fb184ee7e8ccdd3fcc	Ms.	
49228	57854	p099430	4000cfe0c8b2df75a218347c1765e283	Ms.	

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...
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
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	
29891	146723	p099708	c0a28c79fe8ad5810da49de47b3fb491	Mrs.	
23374	72317	p087808	598621c141cda5fb184ee7e8ccdd3fcc	Ms.	
49228	57854	p099430	4000cfe0c8b2df75a218347c1765e283	Ms.	

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_
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	
29891	146723	p099708	c0a28c79fe8ad5810da49de47b3fb491	Mrs.	
23374	72317	p087808	598621c141cda5fb184ee7e8ccdd3fcc	Ms.	
49228	57854	p099430	4000cfe0c8b2df75a218347c1765e283	Ms.	

1.3 Sampling data for random_forest Assignment

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

```
1    42284
0     7714
Name: project_is_approved, dtype: int64
```

```
X = project_data


# Split the class label from data

#X = data.drop(['project_is_approved'], axis=1)

y = X['project_is_approved'].values
X.head(1)
```

	Unnamed: 0	id	teacher_id	teacher_prefix	school_st
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	

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

```
# 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, X['project_is_approved'], t
#X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33,



#y_train = X_train.project_is_approved
X_train = X_train.drop('project_is_approved', axis=1)
#y_test = X_test.project_is_approved
X_test = X_test.drop('project_is_approved', axis=1)
```

2.2 Make Data Model Ready:

2.2.1 Function for response coding

```
def mask(df, key, value):
  return df[df[key] == value]

def get_response(feature,label):
  accepted = {};
  rejected= {};
  neg_prob = {};
  pos_prob={};

  unique_cat = np.unique(feature).tolist()
  df = pd.DataFrame({'features':feature.values.tolist(),'labels':label.values.toli
  pd.DataFrame.mask = mask

  for i in unique_cat:
    count_rej = len(df.mask('features', i).mask('labels', 0))
    count_acc = len(df.mask('features', i).mask('labels', 1))
    total = count_acc + count_rej
    prob_neg = count_rej/total
    prob_pos = count_acc/total
    accepted[i] = count_acc
    rejected[i] = count_rej
    neg_prob[i] = prob_neg
    pos_prob[i] = prob_pos

  return neg_prob,pos_prob
```

2.2.1 Encoding categorical features

2.2.1.1 Encoding School State with Response Coding

```
state_rejected_train = get_response(X_train['school_state'],y_train)[0]
state_accepted_train = get_response(X_train['school_state'],y_train)[1]

state_rejected_test = get_response(X_test['school_state'],y_test)[0]
state_accepted_test = get_response(X_test['school_state'],y_test)[1]
```

```
#X_train.drop(['state_accepted','state_rejected'],axis=1,inplace=True)
```

```
rejected = []
accepted = []
for i in X_train['school_state']:
  rejected.append(state_rejected_train[i])
  accepted.append(state_accepted_train[i])
X_train['state_accepted'] = accepted
X_train['state_rejected'] = rejected

rejected = []
accepted = []
for i in X_test['school_state']:
  rejected.append(state_rejected_test[i])
  accepted.append(state_accepted_test[i])
X_test['state_accepted'] = accepted
X_test['state_rejected'] = rejected
```

2.2.1.2 Encoding Teacher Prefix with Response Coding

```
teacher_rejected_train = get_response(X_train['teacher_prefix'],y_train)[0]
teacher_accepted_train = get_response(X_train['teacher_prefix'],y_train)[1]

teacher_rejected_test = get_response(X_test['teacher_prefix'],y_test)[0]
teacher_accepted_test = get_response(X_test['teacher_prefix'],y_test)[1]
```

```
rejected = []
accepted = []
for i in X_train['teacher_prefix']:
  rejected.append(teacher_rejected_train[i])
  accepted.append(teacher_accepted_train[i])
X_train['prefix_accepted'] = accepted
X_train['prefix_rejected'] = rejected

rejected = []
accepted = []
for i in X_test['teacher_prefix']:
  rejected.append(teacher_rejected_test[i])
  accepted.append(teacher_accepted_test[i])
X_test['prefix_accepted'] = accepted
X_test['prefix_rejected'] = rejected
```

2.2.1.3 Encoding Grade Category with Response Coding

```
preprocessed_grade_category_rejected_train = get_response(X_train['preprocessed_gr
preprocessed_grade_category_accepted_train = get_response(X_train['preprocessed_gr

preprocessed_grade_category_rejected_test = get_response(X_test['preprocessed_grad
preprocessed_grade_category_accepted_test = get_response(X_test['preprocessed_grad
```

```
rejected = []
accepted = []
for i in X_train['preprocessed_grade_category']:
  rejected.append(preprocessed_grade_category_rejected_train[i])
  accepted.append(preprocessed_grade_category_accepted_train[i])
X_train['cat_accepted'] = accepted
X_train['cat_rejected'] = rejected

rejected = []
accepted = []
for i in X_test['preprocessed_grade_category']:
  rejected.append(preprocessed_grade_category_rejected_test[i])
  accepted.append(preprocessed_grade_category_accepted_test[i])
X_test['cat_accepted'] = accepted
X_test['cat_rejected'] = rejected
```

2.2.1.4 Encoding Project Catagories with Response Coding

```
clean_categories_rejected_train = get_response(X_train['clean_categories'],y_train
clean_categories_accepted_train = get_response(X_train['clean_categories'],y_train

clean_categories_rejected_test = get_response(X_test['clean_categories'],y_test)[0
clean_categories_accepted_test = get_response(X_test['clean_categories'],y_test)[1
```

```
rejected = []
accepted = []
for i in X_train['clean_categories']:
  rejected.append(clean_categories_rejected_train[i])
  accepted.append(clean_categories_accepted_train[i])
X_train['clean_categories_accepted'] = accepted
X_train['clean_categories_rejected'] = rejected

rejected = []
accepted = []
for i in X_test['clean_categories']:
  rejected.append(clean_categories_rejected_test[i])
  accepted.append(clean_categories_accepted_test[i])
X_test['clean_categories_accepted'] = accepted
X_test['clean_categories_rejected'] = rejected
```

2.2.1.5 Encoding Project Subject Subcategories with Response Coding

```
clean_subcategories_rejected_train = get_response(X_train['clean_subcategories'],y
clean_subcategories_accepted_train = get_response(X_train['clean_subcategories'],y

clean_subcategories_rejected_test = get_response(X_test['clean_subcategories'],y_t
clean_subcategories_accepted_test = get_response(X_test['clean_subcategories'],y_t
```

```
clean_subcategories_rejected_test
```

```
{'AppliedSciences': 0.16172506738544473,
 'AppliedSciences CharacterEducation': 0.25,
 'AppliedSciences Civics_Government': 0.0,
 'AppliedSciences College_CareerPrep': 0.1506849315068493,
 'AppliedSciences CommunityService': 0.0,
 'AppliedSciences ESL': 0.15384615384615385,
 'AppliedSciences EarlyDevelopment': 0.12903225806451613,
 'AppliedSciences Economics': 1.0,
 'AppliedSciences EnvironmentalScience': 0.18181818181818182,
 'AppliedSciences Extracurricular': 0.15,
 'AppliedSciences ForeignLanguages': 0.5,
 'AppliedSciences Gym_Fitness': 0.5,
 'AppliedSciences Health_LifeScience': 0.12903225806451613,
 'AppliedSciences Health_Wellness': 0.14285714285714285,
 'AppliedSciences History_Geography': 0.16666666666666666,
 'AppliedSciences Literacy': 0.13953488372093023,
 'AppliedSciences Literature_Writing': 0.11290322580645161,
 'AppliedSciences Mathematics': 0.18199608610567514,
 'AppliedSciences Music': 0.0,
 'AppliedSciences Other': 0.07692307692307693,
 'AppliedSciences ParentInvolvement': 0.16666666666666666,
 'AppliedSciences PerformingArts': 0.0,
 'AppliedSciences SocialSciences': 0.0,
 'AppliedSciences SpecialNeeds': 0.22,
 'AppliedSciences TeamSports': 0.0,
 'AppliedSciences VisualArts': 0.12745098039215685,
 'CharacterEducation': 0.21052631578947367,
 'CharacterEducation College_CareerPrep': 0.2222222222222222,
 'CharacterEducation CommunityService': 0.16666666666666666,
 'CharacterEducation ESL': 0.0,
 'CharacterEducation EarlyDevelopment': 0.12,
 'CharacterEducation EnvironmentalScience': 0.3333333333333333,
 'CharacterEducation Extracurricular': 0.14285714285714285,
 'CharacterEducation ForeignLanguages': 0.0,
 'CharacterEducation Gym_Fitness': 0.3333333333333333,
 'CharacterEducation Health_LifeScience': 1.0,
 'CharacterEducation Health_Wellness': 0.25,
 'CharacterEducation Literacy': 0.11363636363636363,
 'CharacterEducation Literature_Writing': 0.23076923076923078,
 'CharacterEducation Mathematics': 0.08333333333333333,
 'CharacterEducation Music': 0.0,
 'CharacterEducation NutritionEducation': 0.0,
 'CharacterEducation Other': 0.3181818181818182,
 'CharacterEducation ParentInvolvement': 0.125,
 'CharacterEducation SocialSciences': 0.3333333333333333,
 'CharacterEducation SpecialNeeds': 0.16666666666666666,
 'CharacterEducation TeamSports': 0.3333333333333333,
 'CharacterEducation VisualArts': 0.3,
 'Civics_Government': 0.058823529411764705,
 'Civics_Government College_CareerPrep': 0.5,
 'Civics_Government CommunityService': 0.3333333333333333,
 'Civics_Government Economics': 0.2,
 'Civics_Government EnvironmentalScience': 0.5,
 'Civics_Government Extracurricular': 1.0,
 'Civics_Government FinancialLiteracy': 0.2,
 'Civics_Government Health_LifeScience': 0.3333333333333333,
 'Civics_Government History_Geography': 0.08108108108108109,
 'Civics_Government Literacy': 0.08695652173913043,
 'Civics_Government Literature_Writing': 0.08333333333333333,
 'Civics_Government Mathematics': 0.5,
 'Civics_Government SocialSciences': 0.08333333333333333,
 'Civics_Government SpecialNeeds': 0.5,
 'Civics_Government TeamSports': 0.0,
 'Civics_Government VisualArts': 1.0,
 'College_CareerPrep': 0.1935483870967742,
 'College_CareerPrep CommunityService': 0.0,
 'College_CareerPrep ESL': 0.0,
 'College_CareerPrep EarlyDevelopment': 0.0,
 'College_CareerPrep Economics': 0.0,
 'College_CareerPrep EnvironmentalScience': 0.14285714285714285,
 'College_CareerPrep Extracurricular': 0.1111111111111111,
 'College_CareerPrep FinancialLiteracy': 0.25,
 'College_CareerPrep Health_LifeScience': 0.25,
 'College_CareerPrep Health_Wellness': 0.4,
 'College_CareerPrep Literacy': 0.06451612903225806,
 'College_CareerPrep Literature_Writing': 0.09090909090909091,
 'College_CareerPrep Mathematics': 0.125,
 'College_CareerPrep Music': 1.0,
 'College_CareerPrep NutritionEducation': 0.0,
 'College_CareerPrep Other': 0.1875,
 'College_CareerPrep ParentInvolvement': 0.25,
 'College_CareerPrep PerformingArts': 0.42857142857142855,
 'College_CareerPrep SocialSciences': 0.3333333333333333,
 'College_CareerPrep SpecialNeeds': 0.2727272727272727,
 'College_CareerPrep TeamSports': 0.0,
 'College_CareerPrep VisualArts': 0.2777777777777778,
 'College_CareerPrep Warmth Care_Hunger': 0.0,
 'CommunityService': 0.18181818181818182,
 'CommunityService ESL': 0.0,
 'CommunityService Economics': 0.0,
 'CommunityService EnvironmentalScience': 0.0,
 'CommunityService Extracurricular': 0.14285714285714285,
 'CommunityService Gym_Fitness': 0.0,
 'CommunityService Health_LifeScience': 0.0,
 'CommunityService Health_Wellness': 0.5,
 'CommunityService History_Geography': 0.0,
 'CommunityService Literacy': 0.5,
 'CommunityService Literature_Writing': 0.0,
 'CommunityService Mathematics': 0.0,
 'CommunityService Other': 0.0,
 'CommunityService ParentInvolvement': 0.0,
 'CommunityService SocialSciences': 0.0,
 'CommunityService SpecialNeeds': 1.0,
 'CommunityService VisualArts': 0.3333333333333333,
 'ESL': 0.1232876712328767,
 'ESL EarlyDevelopment': 0.3333333333333333,
 'ESL EnvironmentalScience': 0.4,
 'ESL Extracurricular': 0.0,
 'ESL FinancialLiteracy': 0.0,
 'ESL ForeignLanguages': 0.14285714285714285,
 'ESL Health_LifeScience': 0.2,
 'ESL Health_Wellness': 0.0,
 'ESL History_Geography': 0.0,
 'ESL Literacy': 0.147239263803681,
 'ESL Literature_Writing': 0.16666666666666666,
 'ESL Mathematics': 0.19148936170212766,
 'ESL Music': 0.0,
 'ESL ParentInvolvement': 0.5,
 'ESL PerformingArts': 0.0,
 'ESL SpecialNeeds': 0.14814814814814814,
 'ESL VisualArts': 0.16666666666666666,
 'EarlyDevelopment': 0.19424460431654678,
 'EarlyDevelopment EnvironmentalScience': 0.2857142857142857,
  'EarlyDevelopment EnvironmentalScience': 0.2857142857142857,
 'EarlyDevelopment Extracurricular': 0.5,
 'EarlyDevelopment FinancialLiteracy': 0.0,
 'EarlyDevelopment Gym_Fitness': 0.0,
 'EarlyDevelopment Health_LifeScience': 0.3333333333333333,
 'EarlyDevelopment Health_Wellness': 0.11904761904761904,
 'EarlyDevelopment History_Geography': 0.0,
 'EarlyDevelopment Literacy': 0.17142857142857143,
 'EarlyDevelopment Literature_Writing': 0.2,
 'EarlyDevelopment Mathematics': 0.2857142857142857,
 'EarlyDevelopment Music': 0.0,
 'EarlyDevelopment NutritionEducation': 0.0,
 'EarlyDevelopment Other': 0.25925925925925924,
 'EarlyDevelopment ParentInvolvement': 0.25,
 'EarlyDevelopment PerformingArts': 0.0,
 'EarlyDevelopment SocialSciences': 0.0,
 'EarlyDevelopment SpecialNeeds': 0.1935483870967742,
 'EarlyDevelopment TeamSports': 0.0,
 'EarlyDevelopment VisualArts': 0.14285714285714285,
 'Economics': 0.16666666666666666,
 'Economics EnvironmentalScience': 0.0,
 'Economics FinancialLiteracy': 0.375,
 'Economics History_Geography': 0.0,
 'Economics Literacy': 0.0,
 'Economics Mathematics': 0.0,
 'Economics Music': 0.0,
 'Economics SocialSciences': 0.0,
 'EnvironmentalScience': 0.17197452229299362,
 'EnvironmentalScience Extracurricular': 0.5,
 'EnvironmentalScience FinancialLiteracy': 0.0,
 'EnvironmentalScience Gym_Fitness': 1.0,
 'EnvironmentalScience Health_LifeScience': 0.21052631578947367,
 'EnvironmentalScience Health_Wellness': 0.3333333333333333,
 'EnvironmentalScience History_Geography': 0.09523809523809523,
 'EnvironmentalScience Literacy': 0.16417910447761194,
 'EnvironmentalScience Literature_Writing': 0.10810810810810811,
 'EnvironmentalScience Mathematics': 0.2158273381294964,
 'EnvironmentalScience NutritionEducation': 0.75,
 'EnvironmentalScience Other': 0.5,
 'EnvironmentalScience ParentInvolvement': 0.0,
 'EnvironmentalScience PerformingArts': 0.5,
 'EnvironmentalScience SocialSciences': 0.25,
 'EnvironmentalScience SpecialNeeds': 0.22727272727272727,
 'EnvironmentalScience VisualArts': 0.16216216216216217,
 'EnvironmentalScience Warmth Care_Hunger': 0.0,
 'Extracurricular': 0.11764705882352941,
 'Extracurricular ForeignLanguages': 0.0,
 'Extracurricular Health_LifeScience': 1.0,
 'Extracurricular Health_Wellness': 0.3333333333333333,
 'Extracurricular Literacy': 0.4,
 'Extracurricular Literature_Writing': 0.0,
 'Extracurricular Mathematics': 0.2727272727272727,
 'Extracurricular Music': 1.0,
 'Extracurricular NutritionEducation': 1.0,
 'Extracurricular Other': 0.0,
 'Extracurricular ParentInvolvement': 0.0,
 'Extracurricular PerformingArts': 0.0,
 'Extracurricular SpecialNeeds': 0.25,
 'Extracurricular TeamSports': 0.0,
 'Extracurricular VisualArts': 0.25,
 'FinancialLiteracy': 0.15789473684210525,
 'FinancialLiteracy ForeignLanguages': 0.0,
 'FinancialLiteracy Health_Wellness': 0.0,
 'FinancialLiteracy Health_Wellness': 0.0,
 'FinancialLiteracy History_Geography': 0.0,
 'FinancialLiteracy Literacy': 0.0,
 'FinancialLiteracy Mathematics': 0.2727272727272727,
 'FinancialLiteracy Other': 0.0,
 'FinancialLiteracy SocialSciences': 0.0,
 'FinancialLiteracy SpecialNeeds': 0.1111111111111111,
 'ForeignLanguages': 0.24489795918367346,
 'ForeignLanguages Gym_Fitness': 0.0,
 'ForeignLanguages Health_LifeScience': 0.0,
 'ForeignLanguages Health_Wellness': 0.0,
 'ForeignLanguages History_Geography': 0.0,
 'ForeignLanguages Literacy': 0.1111111111111111,
 'ForeignLanguages Literature_Writing': 0.2727272727272727,
 'ForeignLanguages Mathematics': 0.0,
 'ForeignLanguages Music': 0.0,
 'ForeignLanguages Other': 0.0,
 'ForeignLanguages SocialSciences': 0.0,
 'ForeignLanguages VisualArts': 0.5,
 'Gym_Fitness': 0.2155688622754491,
 'Gym_Fitness Health_LifeScience': 0.0,
 'Gym_Fitness Health_Wellness': 0.10982658959537572,
 'Gym_Fitness History_Geography': 0.0,
 'Gym_Fitness Literacy': 0.2,
 'Gym_Fitness Mathematics': 0.0,
 'Gym_Fitness Music': 0.3333333333333333,
 'Gym_Fitness NutritionEducation': 0.2222222222222222,
 'Gym_Fitness Other': 0.3333333333333333,
 'Gym_Fitness PerformingArts': 0.5,
 'Gym_Fitness SpecialNeeds': 0.10526315789473684,
 'Gym_Fitness TeamSports': 0.2413793103448276,
 'Gym_Fitness VisualArts': 0.0,
 'Health_LifeScience': 0.13513513513513514,
 'Health_LifeScience Health_Wellness': 0.037037037037037035,
 'Health_LifeScience History_Geography': 0.0,
 'Health_LifeScience Literacy': 0.075,
 'Health_LifeScience Literature_Writing': 0.07692307692307693,
 'Health_LifeScience Mathematics': 0.14864864864864866,
 'Health_LifeScience Music': 0.0,
 'Health_LifeScience NutritionEducation': 0.375,
 'Health_LifeScience Other': 0.0,
 'Health_LifeScience ParentInvolvement': 0.0,
 'Health_LifeScience SocialSciences': 0.35714285714285715,
 'Health_LifeScience SpecialNeeds': 0.2608695652173913,
 'Health_LifeScience VisualArts': 0.36363636363636365,
 'Health_LifeScience Warmth Care_Hunger': 0.0,
 'Health_Wellness': 0.1169284467713787,
 'Health_Wellness History_Geography': 0.0,
 'Health_Wellness Literacy': 0.208955223880597,
 'Health_Wellness Literature_Writing': 0.10256410256410256,
 'Health_Wellness Mathematics': 0.12903225806451613,
 'Health_Wellness Music': 0.0,
 'Health_Wellness NutritionEducation': 0.20491803278688525,
 'Health_Wellness Other': 0.22727272727272727,
 'Health_Wellness ParentInvolvement': 0.5,
 'Health_Wellness PerformingArts': 0.0,
 'Health_Wellness SocialSciences': 0.2,
 'Health_Wellness SpecialNeeds': 0.11731843575418995,
 'Health_Wellness TeamSports': 0.16326530612244897,
 'Health_Wellness VisualArts': 0.2857142857142857,
 'Health_Wellness Warmth Care_Hunger': 0.0,
 'History_Geography': 0.30864197530864196,
 'History_Geography': 0.30864197530864196,
 'History_Geography Literacy': 0.06756756756756757,
 'History_Geography Literature_Writing': 0.12371134020618557,
 'History_Geography Mathematics': 0.19047619047619047,
 'History_Geography Music': 0.0,
 'History_Geography Other': 1.0,
 'History_Geography ParentInvolvement': 0.0,
 'History_Geography PerformingArts': 0.0,
 'History_Geography SocialSciences': 0.2222222222222222,
 'History_Geography SpecialNeeds': 0.16666666666666666,
 'History_Geography VisualArts': 0.1111111111111111,
 'Literacy': 0.10812581913499344,
 'Literacy Literature_Writing': 0.13043478260869565,
 'Literacy Mathematics': 0.12392829306313329,
 'Literacy Music': 0.13636363636363635,
 'Literacy NutritionEducation': 0.0,
 'Literacy Other': 0.08,
 'Literacy ParentInvolvement': 0.25925925925925924,
 'Literacy PerformingArts': 0.125,
 'Literacy SocialSciences': 0.11764705882352941,
 'Literacy SpecialNeeds': 0.1246684350132626,
 'Literacy TeamSports': 0.0,
 'Literacy VisualArts': 0.19318181818181818,
 'Literature_Writing': 0.17363344051446947,
 'Literature_Writing Mathematics': 0.14709371293001186,
 'Literature_Writing Music': 0.0,
 'Literature_Writing Other': 0.35,
 'Literature_Writing ParentInvolvement': 0.3,
 'Literature_Writing PerformingArts': 0.19047619047619047,
 'Literature_Writing SocialSciences': 0.04081632653061224,
 'Literature_Writing SpecialNeeds': 0.20212765957446807,
 'Literature_Writing TeamSports': 0.25,
 'Literature_Writing VisualArts': 0.22641509433962265,
 'Literature_Writing Warmth Care_Hunger': 0.0,
 'Mathematics': 0.16156670746634028,
 'Mathematics Music': 0.0,
 'Mathematics NutritionEducation': 0.0,
 'Mathematics Other': 0.2631578947368421,
 'Mathematics ParentInvolvement': 0.0625,
 'Mathematics PerformingArts': 0.0,
 'Mathematics SocialSciences': 0.3,
 'Mathematics SpecialNeeds': 0.195,
 'Mathematics VisualArts': 0.2125,
 'Music': 0.14418604651162792,
 'Music Other': 1.0,
 'Music ParentInvolvement': 0.0,
 'Music PerformingArts': 0.12686567164179105,
 'Music SocialSciences': 0.5,
 'Music SpecialNeeds': 0.0625,
 'Music VisualArts': 0.2857142857142857,
 'NutritionEducation': 0.2631578947368421,
 'NutritionEducation Other': 0.3333333333333333,
 'NutritionEducation SocialSciences': 0.0,
 'NutritionEducation SpecialNeeds': 0.25,
 'NutritionEducation TeamSports': 0.0,
 'NutritionEducation VisualArts': 1.0,
 'Other': 0.232,
 'Other ParentInvolvement': 0.0,
 'Other PerformingArts': 1.0,
 'Other SocialSciences': 0.0,
 'Other SpecialNeeds': 0.20408163265306123,
 'Other TeamSports': 0.0,
 'Other VisualArts': 0.14285714285714285,
 'Other VisualArts': 0.14285714285714285,
 'ParentInvolvement': 0.16666666666666666,
 'ParentInvolvement SocialSciences': 0.5,
 'ParentInvolvement SpecialNeeds': 1.0,
 'ParentInvolvement VisualArts': 0.14285714285714285,
 'ParentInvolvement Warmth Care_Hunger': 0.0,
 'PerformingArts': 0.10112359550561797,
 'PerformingArts SocialSciences': 0.0,
 'PerformingArts SpecialNeeds': 0.3333333333333333,
 'PerformingArts TeamSports': 0.3333333333333333,
 'PerformingArts VisualArts': 0.2,
 'SocialSciences': 0.11764705882352941,
 'SocialSciences SpecialNeeds': 0.18181818181818182,
 'SocialSciences VisualArts': 0.0,
 'SpecialNeeds': 0.19148936170212766,
 'SpecialNeeds TeamSports': 0.0,
 'SpecialNeeds VisualArts': 0.18181818181818182,
 'SpecialNeeds Warmth Care_Hunger': 0.0,
 'TeamSports': 0.1751412429378531,
 'VisualArts': 0.1987179487179487,
 'Warmth Care_Hunger': 0.07075471698113207}
```

```
rejected = []
accepted = []
for i in X_train['clean_subcategories']:
  rejected.append(clean_subcategories_rejected_train[i])
  accepted.append(clean_subcategories_accepted_train[i])
X_train['clean_subcategories_accepted'] = accepted
X_train['clean_subcategories_rejected'] = rejected

rejected = []
accepted = []
for i in X_test['clean_subcategories']:
  rejected.append(clean_subcategories_rejected_test[i])
  accepted.append(clean_subcategories_accepted_test[i])
X_test['clean_subcategories_accepted'] = accepted
X_test['clean_subcategories_rejected'] = rejected
```

```
X_train.head()
```

	Unnamed: 0	id	teacher_id	teacher_prefix	school_
30950	169289	p248363	aaac93c24d1f02fb2410126e48adb3b1	Ms.	
36600	93373	p057027	728238f49b5a819fb334b8639add6f56	Ms.	
28004	57999	p199158	25bb9342cfd49cc8d84b09a36c11348f	Mrs.	
18708	133350	p236063	85d25e77ca8f2c93268b6c7e47da19f0	Mrs.	
40563	96689	p000192	9d5233d2e7c254141c84fdf1d0fe8205	Ms.	

```
X_test.head()
```

	Unnamed: 0	id	teacher_id	teacher_prefix	school
27934	21747	p054512	32b1dc593df0f7fb4df754f2e526ca3b	Mr.	
36464	28195	p083322	512d49e987036000a54506af29250f26	Mrs.	
31456	174777	p158480	159d9a9947f9eaa48dfc479b44007b1e	Ms.	
33389	44321	p256726	edad3a5f8d765039fda482426bee8686	Ms.	
39163	144002	p145319	a93d885fcae3d65556dfb62bfe00450f	Mrs.	

2.2.2 Encoding numerical features

2.2.2.1 Normalizing State (Accepted and Rejected) feature

```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).re

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_train.head()
```

	Unnamed: 0	id	teacher_id	teacher_prefix	school_sta
0	169289	p248363	aaac93c24d1f02fb2410126e48adb3b1	Ms.	
1	93373	p057027	728238f49b5a819fb334b8639add6f56	Ms.	
2	57999	p199158	25bb9342cfd49cc8d84b09a36c11348f	Mrs.	
3	133350	p236063	85d25e77ca8f2c93268b6c7e47da19f0	Mrs.	
4	96689	p000192	9d5233d2e7c254141c84fdf1d0fe8205	Ms.	

```
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['state_accepted'].values.reshape(1,-1))

X_train_state_accepted = normalizer.transform(X_train['state_accepted'].values.res
#X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(1,-1))
X_test_state_accepted = normalizer.transform(X_test['state_accepted'].values.resha

X_train_state_accepted = X_train_state_accepted.reshape(-1,1)
X_test_state_accepted = X_test_state_accepted.reshape(-1,1)


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

```
After vectorizations
(33498, 1) (33498,)
[[0.0052804 ]
 [0.00557951]
 [0.00542205]
 ...
 [0.00557951]
 [0.00537298]
 [0.00555265]]
(16500, 1) (16500,)
==============================================================================
```

```
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['state_rejected'].values.reshape(1,-1))

X_train_state_rejected = normalizer.transform(X_train['state_rejected'].values.res
#X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(1,-1))
X_test_state_rejected = normalizer.transform(X_test['state_rejected'].values.resha

X_train_state_rejected = X_train_state_rejected.reshape(-1,1)
X_test_state_rejected = X_test_state_rejected.reshape(-1,1)


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

```
After vectorizations
(33498, 1) (33498,)
[[0.00638905]
 [0.00476636]
 [0.00562059]
 ...
 [0.00476636]
 [0.0058868 ]
 [0.00491208]]
(16500, 1) (16500,)
==============================================================================
```

2.2.2.2 Normalizing Teacher Prefix (Accepted and Rejected) feature

```
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['prefix_accepted'].values.reshape(1,-1))

X_train_prefix_accepted  = normalizer.transform(X_train['prefix_accepted'].values.
#X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(1,-1))
X_test_prefix_accepted   = normalizer.transform(X_test['prefix_accepted'].values.r

X_train_prefix_accepted  = X_train_prefix_accepted.reshape(-1,1)
X_test_prefix_accepted   = X_test_prefix_accepted.reshape(-1,1)



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

```
After vectorizations
(33498, 1) (33498,)
[[0.00544208]
 [0.00544208]
 [0.00550665]
 ...
 [0.00544208]
 [0.00540479]
 [0.00544208]]
(16500, 1) (16500,)
=============================================================================
```

```
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['prefix_rejected'].values.reshape(1,-1))

X_train_prefix_rejected  = normalizer.transform(X_train['prefix_rejected'].values.
#X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(1,-1))
X_test_prefix_rejected   = normalizer.transform(X_test['prefix_rejected'].values.r

X_train_prefix_rejected  = X_train_prefix_rejected.reshape(-1,1)
X_test_prefix_rejected   = X_test_prefix_rejected.reshape(-1,1)



print("After vectorizations")
print(X_train_prefix_rejected.shape, y_train.shape)
print(X_train_prefix_rejected )
#print(X_cv_price_norm.shape, y_cv.shape)
```

```
#print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_prefix_rejected.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33498, 1) (33498,)
[[0.00556156]
 [0.00556156]
 [0.00520872]
 ...
 [0.00556156]
 [0.00576538]
 [0.00556156]]
(16500, 1) (16500,)
==============================================================================
```

2.2.2.3 Normalizing Grade Category (Accepted and Rejected) feature

```
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['cat_accepted'].values.reshape(1,-1))

X_train_cat_accepted   = normalizer.transform(X_train['cat_accepted'].values.resha
#X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(1,-1))
X_test_cat_accepted  = normalizer.transform(X_test['cat_accepted'].values.reshape(

X_train_cat_accepted   = X_train_cat_accepted.reshape(-1,1)
X_test_cat_accepted  = X_test_cat_accepted.reshape(-1,1)


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

```
After vectorizations
(33498, 1) (33498,)
[[0.00542243]
 [0.00542243]
 [0.0053398 ]
 ...
 [0.00542243]
 [0.00550776]
 [0.00547258]]
(16500, 1) (16500,)
==============================================================================
```

```
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['cat_rejected'].values.reshape(1,-1))

X_train_cat_rejected   = normalizer.transform(X_train['cat_rejected'].values.resha
#X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(1,-1))
X_test_cat_rejected  = normalizer.transform(X_test['cat_rejected'].values.reshape(

X_train_cat_rejected   = X_train_cat_rejected.reshape(-1,1)
X_test_cat_rejected  = X_test_cat_rejected.reshape(-1,1)


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

```
After vectorizations
(33498, 1) (33498,)
[[0.00568193]
 [0.00568193]
 [0.00613435]
 ...
 [0.00568193]
 [0.00521472]
 [0.00540734]]
(16500, 1) (16500,)
==============================================================================
```

2.2.2.4 Normalizing Project subject categories (Accepted and Rejected) featur

```
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['clean_categories_accepted'].values.reshape(1,-1))

X_train_clean_categories_accepted  = normalizer.transform(X_train['clean_categorie
#X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(1,-1))
X_test_clean_categories_accepted = normalizer.transform(X_test['clean_categories_a

X_train_clean_categories_accepted = X_train_clean_categories_accepted.reshape(-1,1
X_test_clean_categories_accepted = X_test_clean_categories_accepted.reshape(-1,1)
```

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

```
After vectorizations
(33498, 1) (33498,)
[[0.00542693]
 [0.00557131]
 [0.005301  ]
 ...
 [0.0055897 ]
 [0.00525989]
 [0.00559204]]
(16500, 1) (16500,)
==============================================================================
```

```
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['clean_categories_rejected'].values.reshape(1,-1))

X_train_clean_categories_rejected = normalizer.transform(X_train['clean_categories
#X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(1,-1))
X_test_clean_categories_rejected = normalizer.transform(X_test['clean_categories_r

X_train_clean_categories_rejected = X_train_clean_categories_rejected.reshape(-1,1
X_test_clean_categories_rejected = X_test_clean_categories_rejected.reshape(-1,1)


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

```
    After vectorizations
```

2.2.2.5 Normalizing Project Subject Subcategories (Accepted and Rejected) fe

```
     [0.00477821
```

```
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['clean_subcategories_accepted'].values.reshape(1,-1))

X_train_clean_subcategories_accepted   = normalizer.transform(X_train['clean_subca
#X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(1,-1))
X_test_clean_subcategories_accepted = normalizer.transform(X_test['clean_subcatego

X_train_clean_subcategories_accepted  = X_train_clean_subcategories_accepted.resha
X_test_clean_subcategories_accepted = X_test_clean_subcategories_accepted.reshape(


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

```
After vectorizations
(33498, 1) (33498,)
[[0.00645005]
 [0.00567374]
 [0.00529583]
 ...
 [0.00579447]
 [0.00507997]
 [0.0055951 ]]
(16500, 1) (16500,)
=============================================================================
```

```
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['clean_subcategories_rejected'].values.reshape(1,-1))

X_train_clean_subcategories_rejected  = normalizer.transform(X_train['clean_subcat
#X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(1,-1))
X_test_clean_subcategories_rejected = normalizer.transform(X_test['clean_subcatego

X_train_clean_subcategories_rejected  = X_train_clean_subcategories_rejected.resha
X_train_clean_subcategories_rejected  = X_train_clean_subcategories_rejected.resha
X_test_clean_subcategories_rejected = X_test_clean_subcategories_rejected.reshape(


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

```
After vectorizations
(33498, 1) (33498,)
[[0.        ]
 [0.00406962]
 [0.0060507 ]
 ...
 [0.00343671]
 [0.0071823 ]
 [0.00448185]]
(16500, 1) (16500,)
==============================================================================
```

2.2.2.6 Normalizing Price feature

```
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_x'].values.reshape(1,-1))

X_train_price_norm = normalizer.transform(X_train['price_x'].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_x'].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
(33498, 1) (33498,)
[[0.0009012 ]
 [0.00020186]
 [0.00463104]
 ...
 [0.01302799]
 [0.00682554]
 [0.00219894]]
(16500, 1) (16500,)
=============================================================================
```

2.2.2.7 Encoding numeric feature Quantity

```
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['quantity_x'].values.reshape(1,-1))

X_train_quantity_norm = normalizer.transform(X_train['quantity_x'].values.reshape(
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_x'].values.reshape(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)
```

```
[[0.00500556]
 [0.0094933 ]
 [0.00034521]
 ...
 [0.00258908]
 [0.00189866]
 [0.00086303]]
After vectorizations
(33498, 1) (33498,)
(16500, 1) (16500,)
=============================================================================
```

2.2.2.8 Encoding numeric feature teacher_number_of_previously_posted_proje

```
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['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_o

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)
```

```
[[0.00102774]
 [0.00719418]
 [0.0007708 ]
 ...
 [0.        ]
 [0.        ]
 [0.00025693]]
After vectorizations
(33498, 1) (33498,)
(16500, 1) (16500,)
=============================================================================
```

2.2.2.9 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
(33498, 1) (33498,)
(16500, 1) (16500,)
==============================================================================
```

2.2.2.10 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
(33498, 1) (33498,)
[[0.00391653]
 [0.0088122 ]
 [0.0058748 ]
 ...
 [0.0058748 ]
 [0.00489567]
 [0.00783307]]
(16500, 1) (16500,)
==============================================================================
```

2.2.2.11 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
(33498, 1) (33498,)
[[0.00458179]
 [0.00881382]
 [0.00591085]
 ...
 [0.00559607]
 [0.00444188]
 [0.00790446]]
(16500, 1) (16500,)
==============================================================================
```

2.2.2.12 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)
```

2.2.3 Vectorizing Text Data

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

```
(33498, 34) (33498,)
(16500, 34) (16500,)
=============================================================================
After vectorizations
(33498, 10000) (33498,)
(16500, 10000) (16500,)
=============================================================================
NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME
```

2.2.3.2 Encoding preprocessed titles BoW

```
vectorizer = CountVectorizer(min_df=10, 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("NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME")
```

```
After vectorizations
(33498, 1629) (33498,)
(16500, 1629) (16500,)
=============================================================================
NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME
```

2.2.3.3 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
(33498, 1629) (33498,)
(16500, 1629) (16500,)
=============================================================================
```

2.2.3.4 Encoding preprocessed Essays TFIDF

```
vectorizer = TfidfVectorizer(min_df=10)
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_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33498, 10460) (33498,)
(16500, 10460) (16500,)
=============================================================================
```

2.2.3.5 Encoding preprocessed titles TFIDF W2V

```
with open('/content/drive/My Drive/Colab Notebooks/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

```
# 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 sto
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%|██████████| 33498/33498 [00:01<00:00, 32901.43it/s]33498
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
    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%|██████████| 16500/16500 [00:00<00:00, 34242.47it/s]16500
300
```

2.2.3.6 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 sto
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%|██████████| 33498/33498 [01:01<00:00, 541.78it/s]33498
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%|██████████| 16500/16500 [00:30<00:00, 544.82it/s]16500
300
```

2.2.3.7 Encoding preprocessed titles AVG W2V

```
# 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/sen
    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%|██████████| 33498/33498 [00:09<00:00, 3658.32it/s]33498
300
```

```
avg_w2v_vectors_essays_test = []; # the avg-w2v for each sentence/review is stored
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
            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%|██████████| 16500/16500 [00:04<00:00, 3489.20it/s]16500
```

2.2.3.8 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/sen
    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%|██████████| 33498/33498 [00:00<00:00, 65455.24it/s]33498
```

```
avg_w2v_vectors_titles_test = []; # the avg-w2v for each sentence/review is stored
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%|██████████| 16500/16500 [00:00<00:00, 66126.00it/s]
```

3.1 Appling random_forest on different kind of featurization instructions

```
#X_test = sparse.load_npz("/content/drive/My Drive/Colab Notebooks/bow_test.npz")
```

```
#X_test = sparse.load_npz("/content/drive/My Drive/Colab Notebooks/avg_w2v_test.np
# 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 batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estim
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 49041%10
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred



def train_random_forest(X_tr,y_train):
    n_est =  [10, 50,100,200,500,1000]
    max_dep = [2, 4, 5, 6,7, 8, 9, 10]
    train_score=[]
    test_score=[]
    #create a dictionary of all values we want to test for alpha values
    parameters = {'n_estimators': [10, 50,100,200,500,1000], 'max_depth':[2, 4, 5,
    clf = RandomForestClassifier(class_weight = 'balanced')

    #use gridsearch to test all values for alpha
    gs = GridSearchCV(clf, parameters, cv=3, scoring='roc_auc',n_jobs=-1, return_t
    gs_results = gs.fit(X_train, y_train)

    print('Best score: ',gs_results.best_score_)
    print('k value with best score: ',gs_results.best_params_)
    print('='*50)

    print(gs.cv_results_.keys())

    for key, value in gs.cv_results_.items():
        if key == "mean_train_score":
            train_score = value
        if key == "mean_test_score":
            test_score = value
        if key == "param_n_estimators":
           param_n_estimators = value
        if key == "param_max_depth":
           depth_list= value

    # Heatmap tutorial
    # https://likegeeks.com/seaborn-heatmap-tutorial/
    # https://likegeeks.com/seaborn-heatmap-tutorial/

    import seaborn as sns; sns.set()
    max_scores1 = pd.DataFrame(gs.cv_results_).groupby(['param_n_estimators', 'par
    fig, ax = plt.subplots(1,2, figsize=(20,6))
    sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
    sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
    ax[0].set_title('Train Set')
    ax[1].set_title('CV Set')
    plt.show()
    '''
    out_arr1 = np.asarray(train_score)
    out_arr2 = np.asarray(test_score)
    array1 = out_arr1.reshape(4, 4)
    array2 = out_arr2.reshape(4, 4)

    sns.heatmap(array1, xticklabels=n_est, yticklabels=max_dep,annot=True,fmt='.2f
    plt.ylabel('Depth')
    plt.xlabel('param_n_estimators')
    plt.show()
    sns.heatmap(array2, xticklabels=n_est, yticklabels=max_dep,annot=True,fmt='.2f
    plt.ylabel('Depth')
    plt.xlabel('param_n_estimators')
    plt.show()
    '''
    return gs_results.best_params_

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

def test_random_forest(X_train,X_test,best_depth,param_n_estimators):

    from sklearn.metrics import roc_curve, auc
    model = RandomForestClassifier(max_depth = best_depth, n_estimators = param_n_

    model.fit(X_train,y_train)

    y_train_pred = batch_predict(model,X_train)
    y_test_pred = batch_predict(model,X_test)

    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.close
    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("FPR")
    plt.ylabel("TPR")
    plt.title("AUC")
    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
```

3.1.1 Apply Random Forest on BOW Vectorization

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_train = hstack((X_train_state_accepted,
X_train_state_rejected,
X_train_prefix_accepted,
X_train_prefix_rejected,
X_train_cat_accepted,
X_train_cat_rejected,
X_train_clean_categories_accepted,
X_train_clean_categories_rejected,
X_train_clean_subcategories_accepted,
X_train_clean_subcategories_rejected,
X_train_price_norm,
X_train_quantity_norm,
X_train_teacher_number_of_previously_posted_projects_norm,
X_train_numerical_data_in_resource_summary_norm,
X_train_number_of_words_in_title,
X_train_number_of_words_in_essay,
train_neg_essay,
train_neu_essay,
train_pos_essay,
train_comp_essay,
X_train_essay_bow,
X_train_titles_bow)).tocsr()

X_test = hstack((X_test_state_accepted,
X_test_state_rejected,
X_test_prefix_accepted,
X_test_prefix_rejected,
X_test_cat_accepted,
X_test_cat_rejected,
X_test_clean_categories_accepted,
X_test_clean_categories_rejected,
X_test_clean_subcategories_accepted,
X_test_clean_subcategories_rejected,
X_test_price_norm,
X_test_quantity_norm,
X_test_teacher_number_of_previously_posted_projects_norm,
X_test_numerical_data_in_resource_summary_norm,
X_test_number_of_words_in_title,
X_test_number_of_words_in_essay,
test_neg_essay,
test_neu_essay,
test_pos_essay,
test_comp_essay,
X_test_essay_bow,
X_test_titles_bow)).tocsr()
```

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

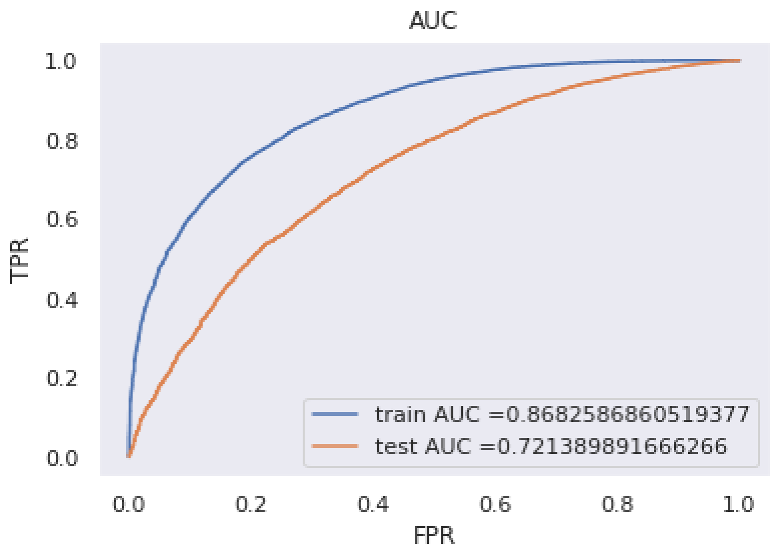
```
# Call train_random_forest function on above data

best_parameters = train_random_forest(X_train,y_train)
```

```
    Best score:  0.7152717042543592
```

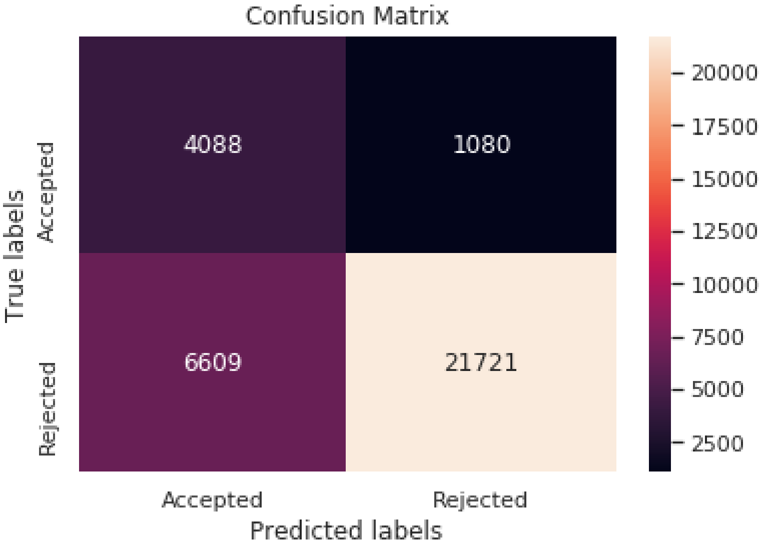
```
best_depth=best_parameters.get('max_depth')
n_estimators=best_parameters.get('n_estimators')
```

3.1.1.2 Testing the peclformance of the model on test data, plotting ROC Curv

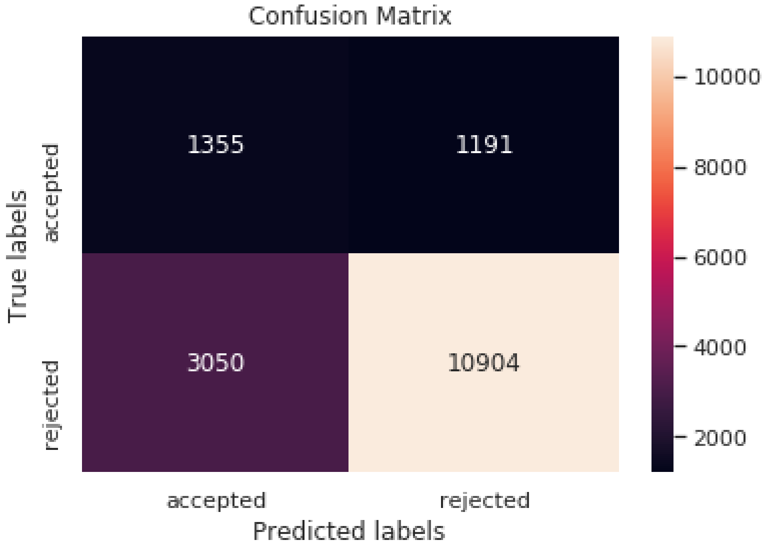
```
train_fpr,train_tpr,tr_thresholds,y_train_pred,y_test_pred=test_random_forest(X_tr
```



```
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) 0.6064871773295493 for threshold 0.84
Train confusion matrix
[[ 4088  1080]
 [ 6609 21721]]
```



```
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
[[ 1355  1191]
 [ 3050 10904]]
```



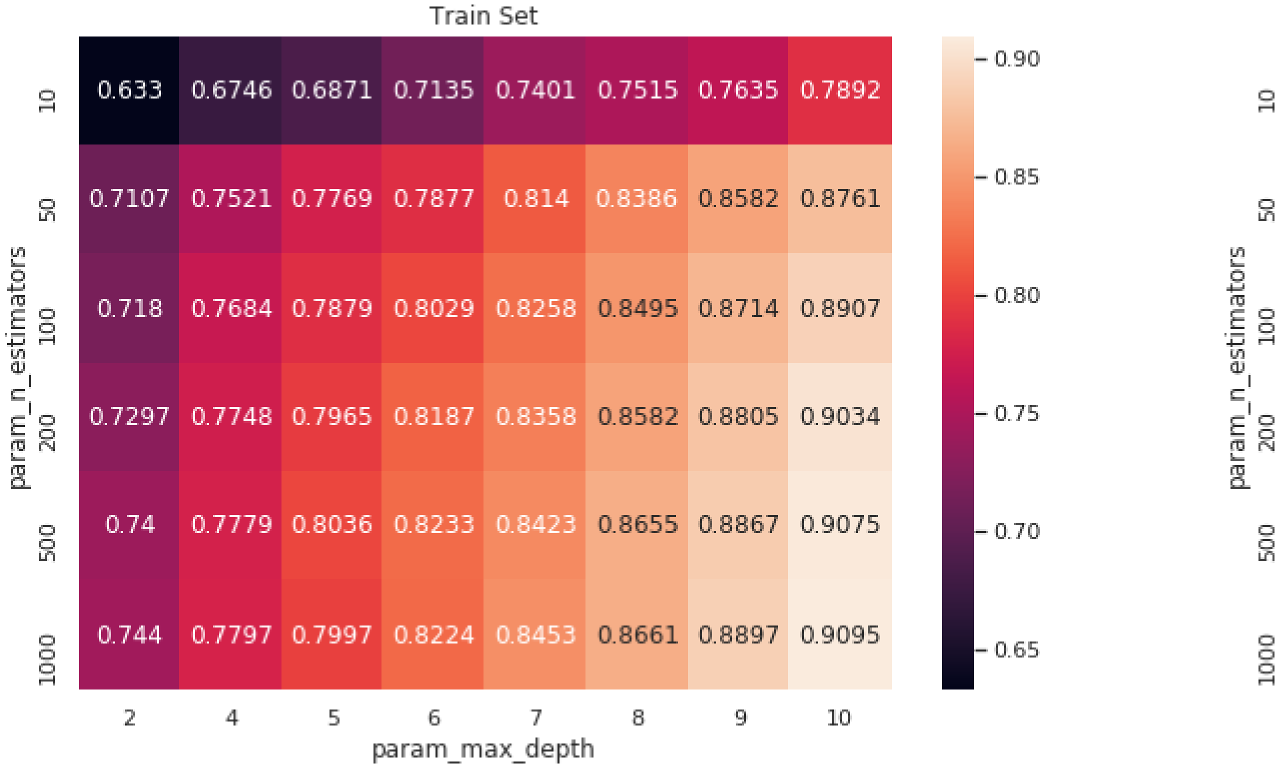
3.1.2 Applying Random Forest on tfidf vectorization

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_train = hstack((X_train_state_accepted,
X_train_state_rejected,
X_train_prefix_accepted,
X_train_prefix_rejected,
X_train_cat_accepted,
X_train_cat_rejected,
X_train_clean_categories_accepted,
X_train_clean_categories_rejected,
X_train_clean_subcategories_accepted,
X_train_clean_subcategories_rejected,
X_train_price_norm,
X_train_quantity_norm,
X_train_teacher_number_of_previously_posted_projects_norm,
X_train_numerical_data_in_resource_summary_norm,
X_train_number_of_words_in_title,
X_train_number_of_words_in_essay,
train_neg_essay,
train_neu_essay,
train_pos_essay,
train_comp_essay,
X_train_essay_tfidf,
X_train_titles_tfidf)).tocsr()

X_test = hstack((X_test_state_accepted,
X_test_state_rejected,
X_test_prefix_accepted,
X_test_prefix_rejected,
X_test_cat_accepted,
X_test_cat_rejected,
X_test_clean_categories_accepted,
X_test_clean_categories_rejected,
X_test_clean_subcategories_accepted,
X_test_clean_subcategories_rejected,
X_test_price_norm,
X_test_quantity_norm,
X_test_teacher_number_of_previously_posted_projects_norm,
X_test_numerical_data_in_resource_summary_norm,
X_test_number_of_words_in_title,
X_test_number_of_words_in_essay,
test_neg_essay,
test_neu_essay,
test_pos_essay,
test_comp_essay,
X_test_essay_tfidf,
X_test_titles_tfidf)).tocsr()
```

3.1.2.1 Training the data model to find best hyperparameter

```
# Call train_random_forest function on above data

best_parameters = train_random_forest(X_train,y_train)
```

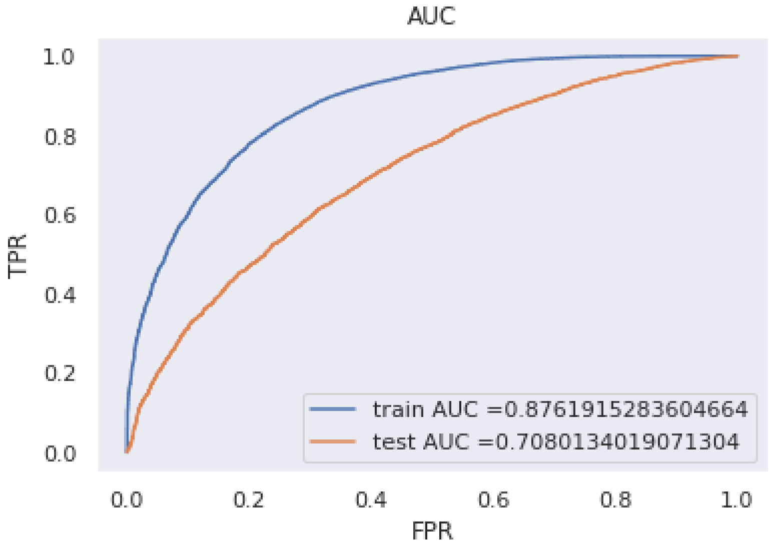
```
Best score:  0.7171150605942715
k value with best score:  {'max_depth': 10, 'n_estimators': 1000}
==================================================
dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_tim
```



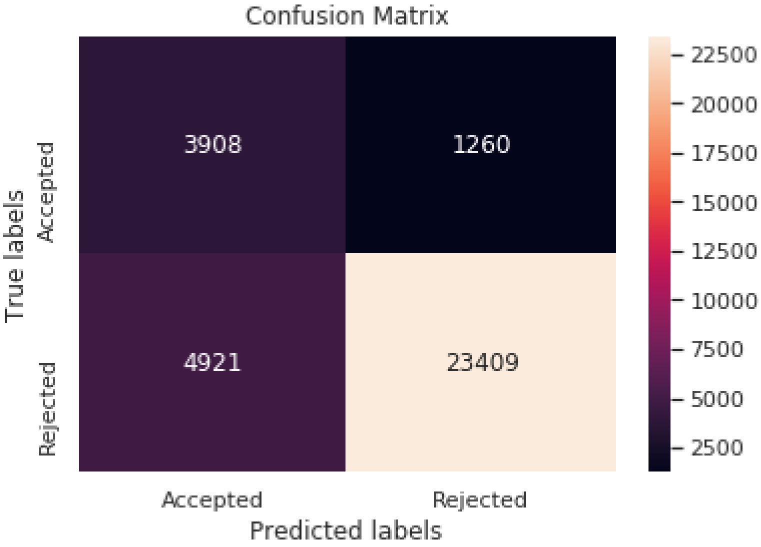
```
best_depth=best_parameters.get('max_depth')
n_estimators=best_parameters.get('n_estimators')
```

3.1.2.2 Testing the peclformance of the model on test data, plotting ROC Curv

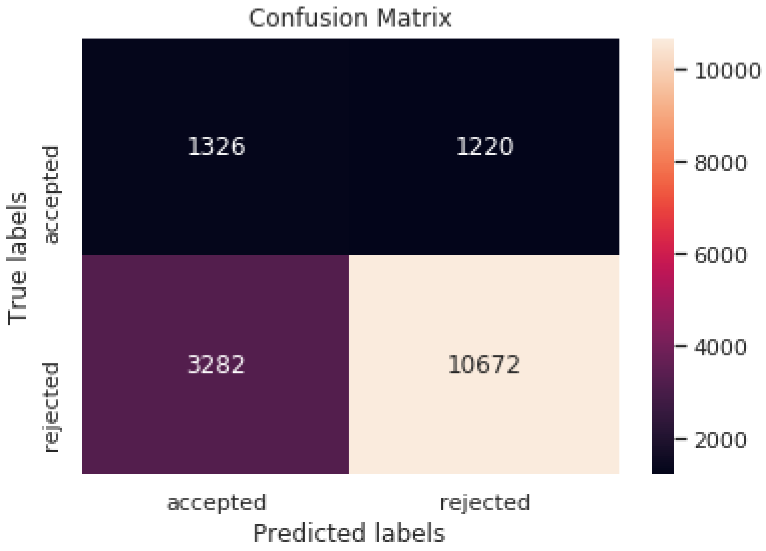
```
train_fpr,train_tpr,tr_thresholds,y_train_pred,y_test_pred=test_random_forest(X_tr
```



```
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) 0.6248392999795641 for threshold 0.839
Train confusion matrix
[[ 3908  1260]
 [ 4921 23409]]
```



```
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
[[ 1326  1220]
 [ 3282 10672]]
```



3.1.3 Applying Random Forest on TFIDF W2V

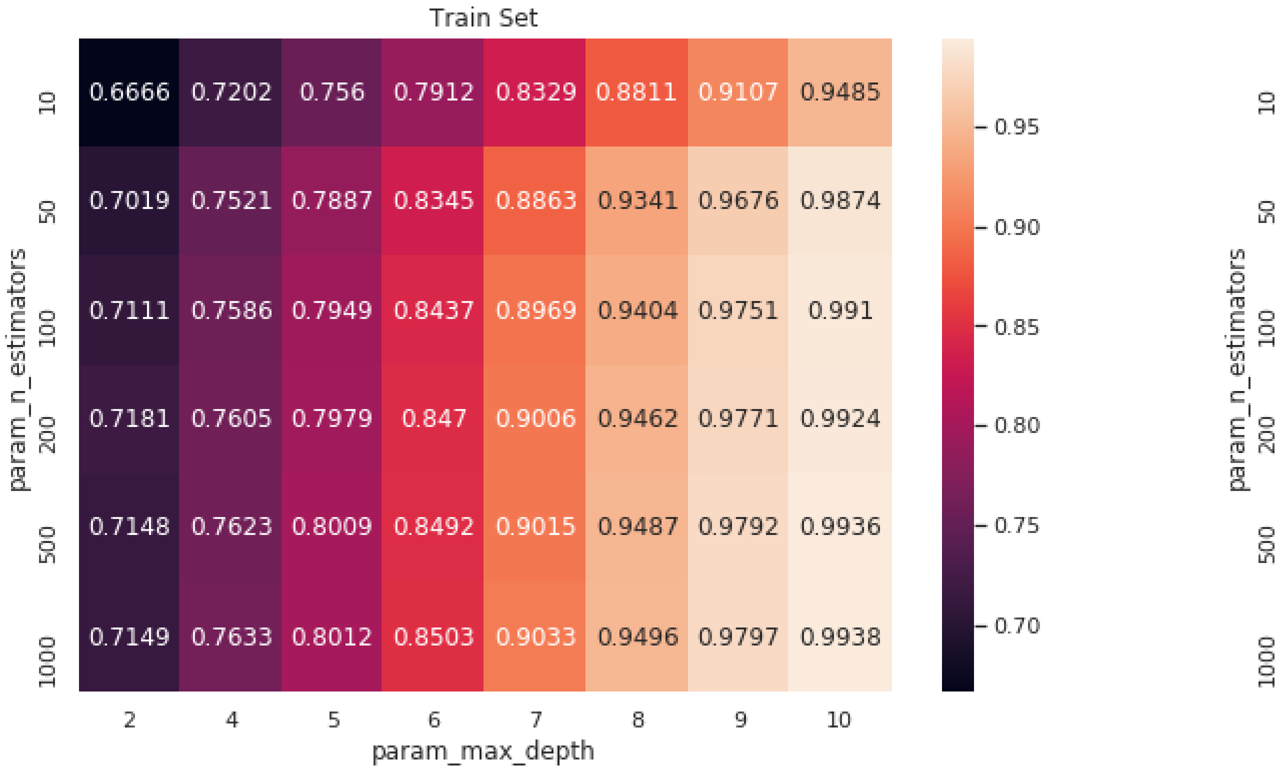
```
tfidf_w2v_vectors_essays_test_1= np.array(tfidf_w2v_vectors_essays_test)
tfidf_w2v_vectors_essays_train_1 = np.array(tfidf_w2v_vectors_essays_train)
tfidf_w2v_vectors_titles_test_1 = np.array(tfidf_w2v_vectors_titles_test)
tfidf_w2v_vectors_titles_train_1 = np.array(tfidf_w2v_vectors_titles_train)
```

```
from scipy.sparse import coo_matrix, hstack
m1 = coo_matrix(X_train_state_accepted)
m2 = coo_matrix(X_train_state_rejected)
m3 = coo_matrix(X_train_prefix_accepted)
m4 = coo_matrix(X_train_prefix_rejected)
m5 = coo_matrix(X_train_cat_accepted)
m6 = coo_matrix(X_train_cat_rejected)
m7 = coo_matrix(X_train_clean_categories_accepted)
m8 = coo_matrix(X_train_clean_categories_rejected)
m9 = coo_matrix(X_train_clean_subcategories_accepted)
m10 = coo_matrix(X_train_clean_subcategories_rejected)
m11 = coo_matrix(X_train_price_norm)
m12 = coo_matrix(X_train_quantity_norm)
m13 = coo_matrix(X_train_teacher_number_of_previously_posted_projects_norm)
m14 = coo_matrix(X_train_numerical_data_in_resource_summary_norm)
m15 = coo_matrix(X_train_number_of_words_in_title)
m16 = coo_matrix(X_train_number_of_words_in_essay)
m17 = coo_matrix(train_neg_essay)
m18 = coo_matrix(train_neu_essay)
m19 = coo_matrix(train_pos_essay)
m20 = coo_matrix(train_comp_essay)
m21 = coo_matrix(tfidf_w2v_vectors_essays_train_1)
m22 = coo_matrix(tfidf_w2v_vectors_titles_train_1)

X_train = hstack([m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12,m13,m14,m15,m16,m17,m18,m
m1 = coo_matrix(X_test_state_accepted)
m2 = coo_matrix(X_test_state_rejected)
m3 = coo_matrix(X_test_prefix_accepted)
m4 = coo_matrix(X_test_prefix_rejected)
m5 = coo_matrix(X_test_cat_accepted)
m6 = coo_matrix(X_test_cat_rejected)
m7 = coo_matrix(X_test_clean_categories_accepted)
m8 = coo_matrix(X_test_clean_categories_rejected)
m9 = coo_matrix(X_test_clean_subcategories_accepted)
m10 = coo_matrix(X_test_clean_subcategories_rejected)
m11 = coo_matrix(X_test_price_norm)
m12 = coo_matrix(X_test_quantity_norm)
m13 = coo_matrix(X_test_teacher_number_of_previously_posted_projects_norm)
m14 = coo_matrix(X_test_numerical_data_in_resource_summary_norm)
m15 = coo_matrix(X_test_number_of_words_in_title)
m16 = coo_matrix(X_test_number_of_words_in_essay)
m17 = coo_matrix(test_neg_essay)
m18 = coo_matrix(test_neu_essay)
m19 = coo_matrix(test_pos_essay)
m20 = coo_matrix(test_comp_essay)
m21 = coo_matrix(tfidf_w2v_vectors_essays_test_1)
m22 = coo_matrix(tfidf_w2v_vectors_titles_test_1)

X_test = hstack([m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12,m13,m14,m15,m16,m17,m18,m1
```

3.1.3.1 Training the data model to find best hyperparameter

```
# Call train_random_forest function on above data

best_parameters = train_random_forest(X_train,y_train)
```

```
Best score:  0.704488980840742
k value with best score:  {'max_depth': 7, 'n_estimators': 1000}
==================================================
dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_tim
```

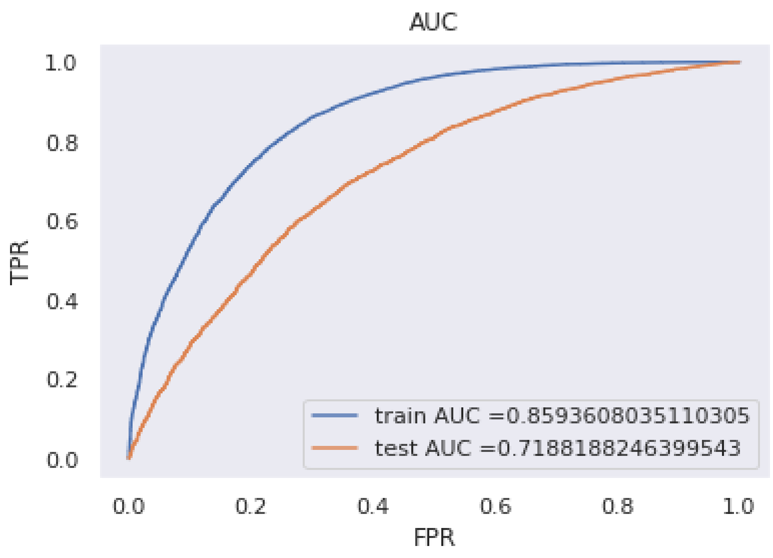


	2	4	5	6	7	8	9	10
10	0.6666	0.7202	0.756	0.7912	0.8329	0.8811	0.9107	0.9485
50	0.7019	0.7521	0.7887	0.8345	0.8863	0.9341	0.9676	0.9874
100	0.7111	0.7586	0.7949	0.8437	0.8969	0.9404	0.9751	0.991
200	0.7181	0.7605	0.7979	0.847	0.9006	0.9462	0.9771	0.9924
500	0.7148	0.7623	0.8009	0.8492	0.9015	0.9487	0.9792	0.9936
1000	0.7149	0.7633	0.8012	0.8503	0.9033	0.9496	0.9797	0.9938

```
best_depth=best_parameters.get('max_depth')
n_estimators=best_parameters.get('n_estimators')
```

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

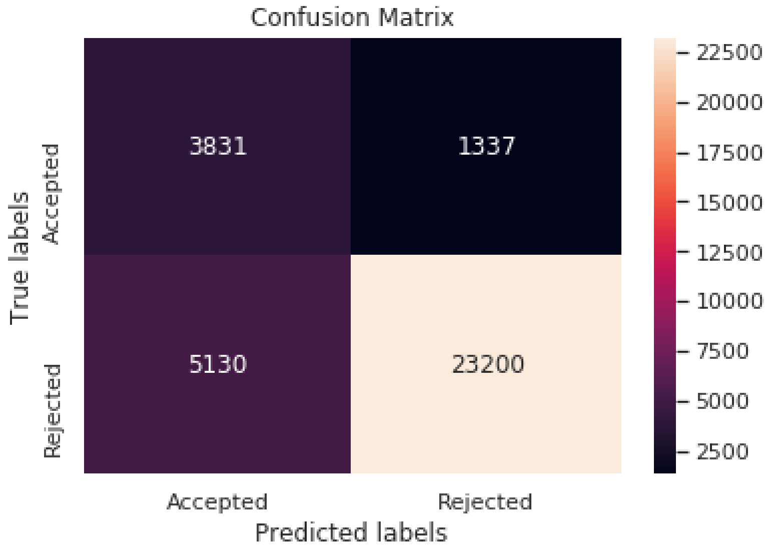
```
train_fpr,train_tpr,tr_thresholds,y_train_pred,y_test_pred=test_random_forest(X_tr
```



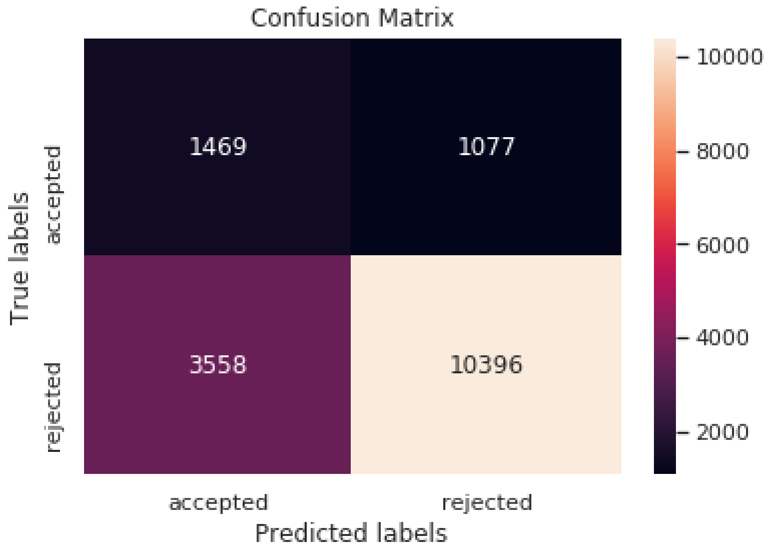
```
print("="*100)
from sklearn.metrics import confusion_matrix
```

```
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) 0.6070592169466669 for threshold 0.827
Train confusion matrix
[[ 3831  1337]
 [ 5130 23200]]
```



```
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
[[ 1469  1077]
 [ 3558 10396]]
```



3.1.4 Applying Random Forest on AVG W2V

```
avg_w2v_vectors_essays_train_1 = np.array(avg_w2v_vectors_essays_train)
avg_w2v_vectors_essays_test_1 = np.array(avg_w2v_vectors_essays_test)
avg_w2v_vectors_titles_test_1 = np.array(avg_w2v_vectors_titles_test)
avg_w2v_vectors_titles_train_1 = np.array(avg_w2v_vectors_titles_train)
```

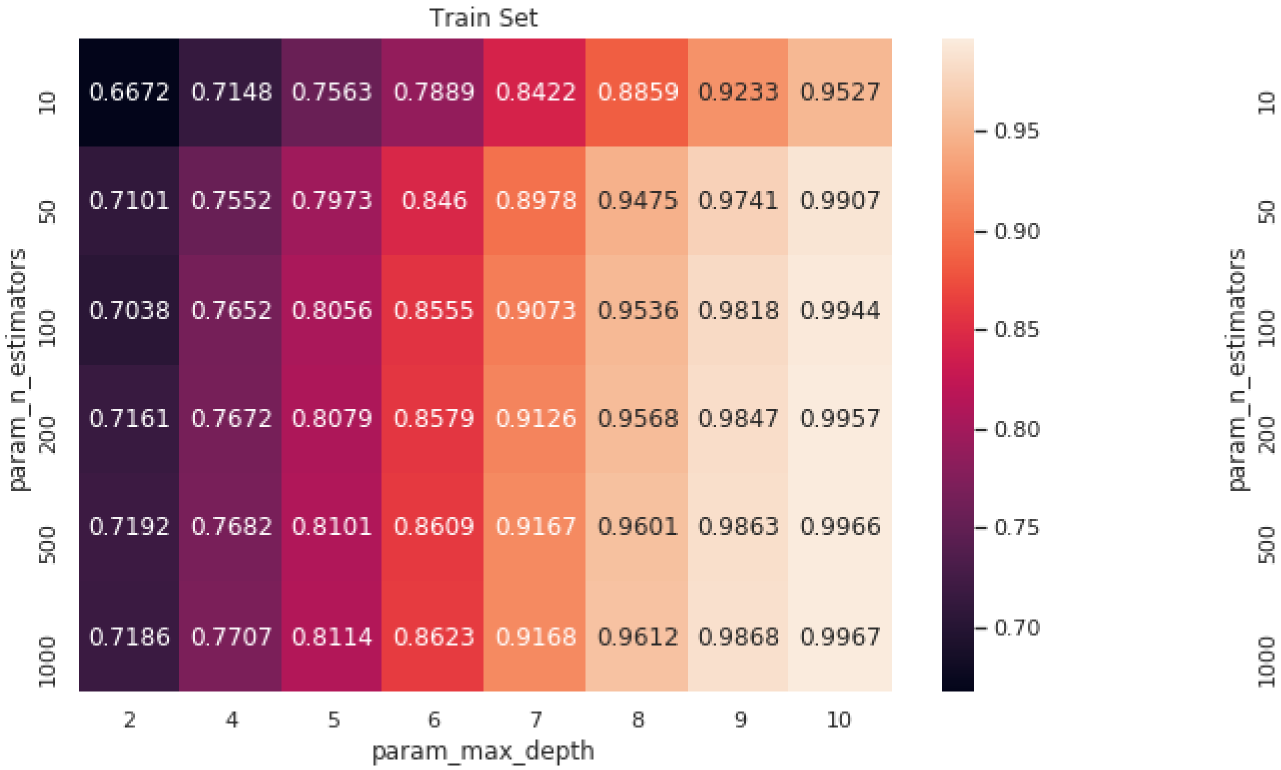
```
from scipy.sparse import coo_matrix, hstack
m1 = coo_matrix(X_train_state_accepted)
m2 = coo_matrix(X_train_state_rejected)
m3 = coo_matrix(X_train_prefix_accepted)
m4 = coo_matrix(X_train_prefix_rejected)
m5 = coo_matrix(X_train_cat_accepted)
m6 = coo_matrix(X_train_cat_rejected)
m7 = coo_matrix(X_train_clean_categories_accepted)
m8 = coo_matrix(X_train_clean_categories_rejected)
m9 = coo_matrix(X_train_clean_subcategories_accepted)
m10 = coo_matrix(X_train_clean_subcategories_rejected)
m11 = coo_matrix(X_train_price_norm)
m12 = coo_matrix(X_train_quantity_norm)
m13 = coo_matrix(X_train_teacher_number_of_previously_posted_projects_norm)
m14 = coo_matrix(X_train_numerical_data_in_resource_summary_norm)
m15 = coo_matrix(X_train_number_of_words_in_title)
m16 = coo_matrix(X_train_number_of_words_in_essay)
m17 = coo_matrix(train_neg_essay)
m18 = coo_matrix(train_neu_essay)
m19 = coo_matrix(train_pos_essay)
m20 = coo_matrix(train_comp_essay)
m21 = coo_matrix(avg_w2v_vectors_essays_train_1)
m22 = coo_matrix(avg_w2v_vectors_titles_train_1)

X_train = hstack([m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12,m13,m14,m15,m16,m17,m18,m
X_train = hstack([m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12,m13,m14,m15,m16,m17,m18,m
```

```
m1 = coo_matrix(X_test_state_accepted)
m2 = coo_matrix(X_test_state_rejected)
m3 = coo_matrix(X_test_prefix_accepted)
m4 = coo_matrix(X_test_prefix_rejected)
m5 = coo_matrix(X_test_cat_accepted)
m6 = coo_matrix(X_test_cat_rejected)
m7 = coo_matrix(X_test_clean_categories_accepted)
m8 = coo_matrix(X_test_clean_categories_rejected)
m9 = coo_matrix(X_test_clean_subcategories_accepted)
m10 = coo_matrix(X_test_clean_subcategories_rejected)
m11 = coo_matrix(X_test_price_norm)
m12 = coo_matrix(X_test_quantity_norm)
m13 = coo_matrix(X_test_teacher_number_of_previously_posted_projects_norm)
m14 = coo_matrix(X_test_numerical_data_in_resource_summary_norm)
m15 = coo_matrix(X_test_number_of_words_in_title)
m16 = coo_matrix(X_test_number_of_words_in_essay)
m17 = coo_matrix(test_neg_essay)
m18 = coo_matrix(test_neu_essay)
m19 = coo_matrix(test_pos_essay)
m20 = coo_matrix(test_comp_essay)
m21 = coo_matrix(avg_w2v_vectors_essays_test_1)
m22 = coo_matrix(avg_w2v_vectors_titles_test_1)

X_test = hstack([m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12,m13,m14,m15,m16,m17,m18,m1
```

3.1.4.1 Training the data model to find best hyperparameter

```
# Call train_random_forest function on above data

best_parameters = train_random_forest(X_train,y_train)
```

```
Best score:  0.7136192706993366
k value with best score:  {'max_depth': 8, 'n_estimators': 1000}
==================================================
dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_tim
```

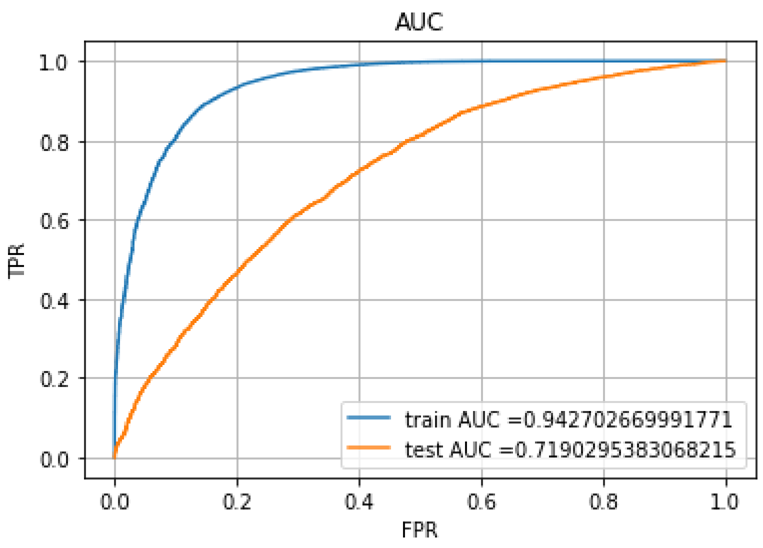


param_n_estimators	2	4	5	6	7	8	9	10
10	0.6672	0.7148	0.7563	0.7889	0.8422	0.8859	0.9233	0.9527
50	0.7101	0.7552	0.7973	0.846	0.8978	0.9475	0.9741	0.9907
100	0.7038	0.7652	0.8056	0.8555	0.9073	0.9536	0.9818	0.9944
200	0.7161	0.7672	0.8079	0.8579	0.9126	0.9568	0.9847	0.9957
500	0.7192	0.7682	0.8101	0.8609	0.9167	0.9601	0.9863	0.9966
1000	0.7186	0.7707	0.8114	0.8623	0.9168	0.9612	0.9868	0.9967

3.1.4.2 Testing the peclformance of the model on test data, plotting ROC Curv

```
best_depth=best_parameters.get('max_depth')
n_estimators=best_parameters.get('n_estimators')
```

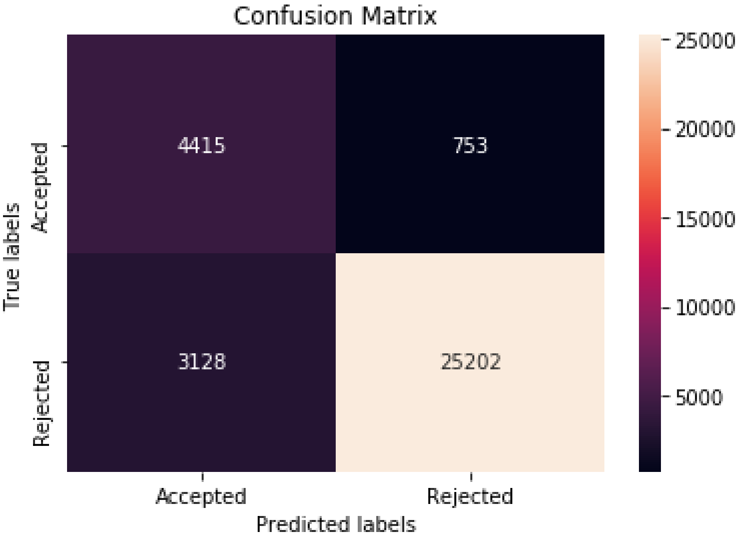
```
train_fpr,train_tpr,tr_thresholds,y_train_pred,y_test_pred=test_random_forest(X_tra
```



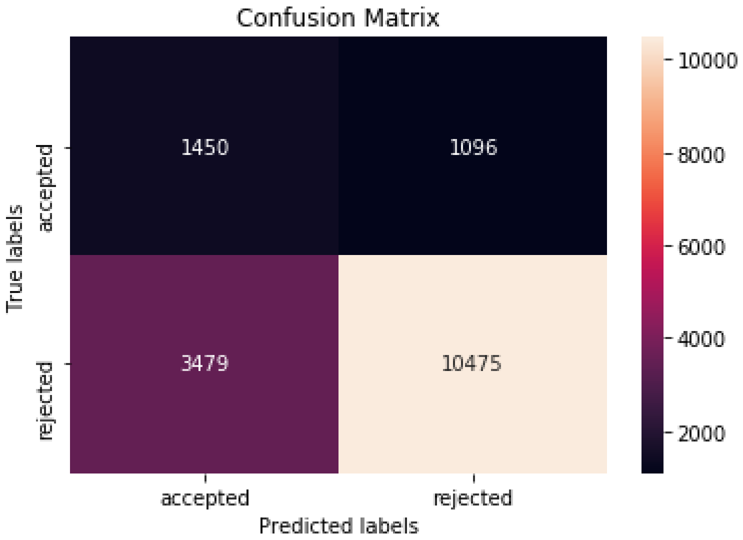
```
print("="*100)
from sklearn.metrics import confusion_matrix
```

```
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) 0.7599703270499498 for threshold 0.829
Train confusion matrix
[[ 4415   753]
 [ 3128 25202]]
```



```
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
[[ 1450  1096]
 [ 3479 10475]]
```



3.2 Appling XGBoost on different kind of featurization as me

```
# 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
from sklearn.model_selection import RandomizedSearchCV

def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estim
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 49041%10
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred


def train_XGB(X_tr,y_train):
    n_est =  [10, 50,100,200,500,1000]
    max_dep = [2, 4, 5, 6,7, 8, 9, 10]
    train_score=[]
    test_score=[]
    #create a dictionary of all values we want to test for alpha values
```

```
        parameters = {'n_estimators': [10, 50,100,200,500,1000], 'max_depth':[2, 4, 5,
        clf = XGBClassifier(class_weight = 'balanced')

        #use gridsearch to test all values for alpha
        gs = RandomizedSearchCV(clf,parameters ,cv=3, scoring='roc_auc',n_jobs=-1,retu
        #gs = GridSearchCV(clf, parameters, cv=3, scoring='roc_auc', return_train_scor
        gs_results = gs.fit(X_train, y_train)

        print('Best score: ',gs_results.best_score_)
        print('k value with best score: ',gs_results.best_params_)
        print('='*50)

        print(gs.cv_results_.keys())

        for key, value in gs.cv_results_.items():
            if key == "mean_train_score":
                train_score = value
            if key == "mean_test_score":
                test_score = value
            if key == "param_n_estimators":
               param_n_estimators = value
            if key == "param_max_depth":
               depth_list= value


        # Heatmap tutorial
        # https://likegeeks.com/seaborn-heatmap-tutorial/

        import seaborn as sns; sns.set()
        max_scores1 = pd.DataFrame(gs.cv_results_).groupby(['param_n_estimators', 'par
        fig, ax = plt.subplots(1,2, figsize=(20,6))
        sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
        sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
        ax[0].set_title('Train Set')
        ax[1].set_title('CV Set')
        plt.show()
        '''
        out_arr1 = np.asarray(train_score)
        out_arr2 = np.asarray(test_score)
        array1 = out_arr1.reshape(4, 4)
        array2 = out_arr2.reshape(4, 4)

        sns.heatmap(array1, xticklabels=n_est, yticklabels=max_dep,annot=True,fmt='.2f
        plt.ylabel('Depth')
        plt.xlabel('param_n_estimators')
        plt.show()
        sns.heatmap(array2, xticklabels=n_est, yticklabels=max_dep,annot=True,fmt='.2f
        plt.ylabel('Depth')
        plt.xlabel('param_n_estimators')
        plt.show()
        '''
        return gs_results.best_params_

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

```
def test_XGB(X_train,X_test,best_depth,param_n_estimators):

    from sklearn.metrics import roc_curve, auc
    model = RandomForestClassifier(max_depth = best_depth, n_estimators = param_n_

    model.fit(X_train,y_train)

    y_train_pred = batch_predict(model,X_train)
    y_test_pred = batch_predict(model,X_test)

    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.close
    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("FPR")
    plt.ylabel("TPR")
    plt.title("AUC")
    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
```

3.2.1 Apply XGBoost on Bag of Words vectorization

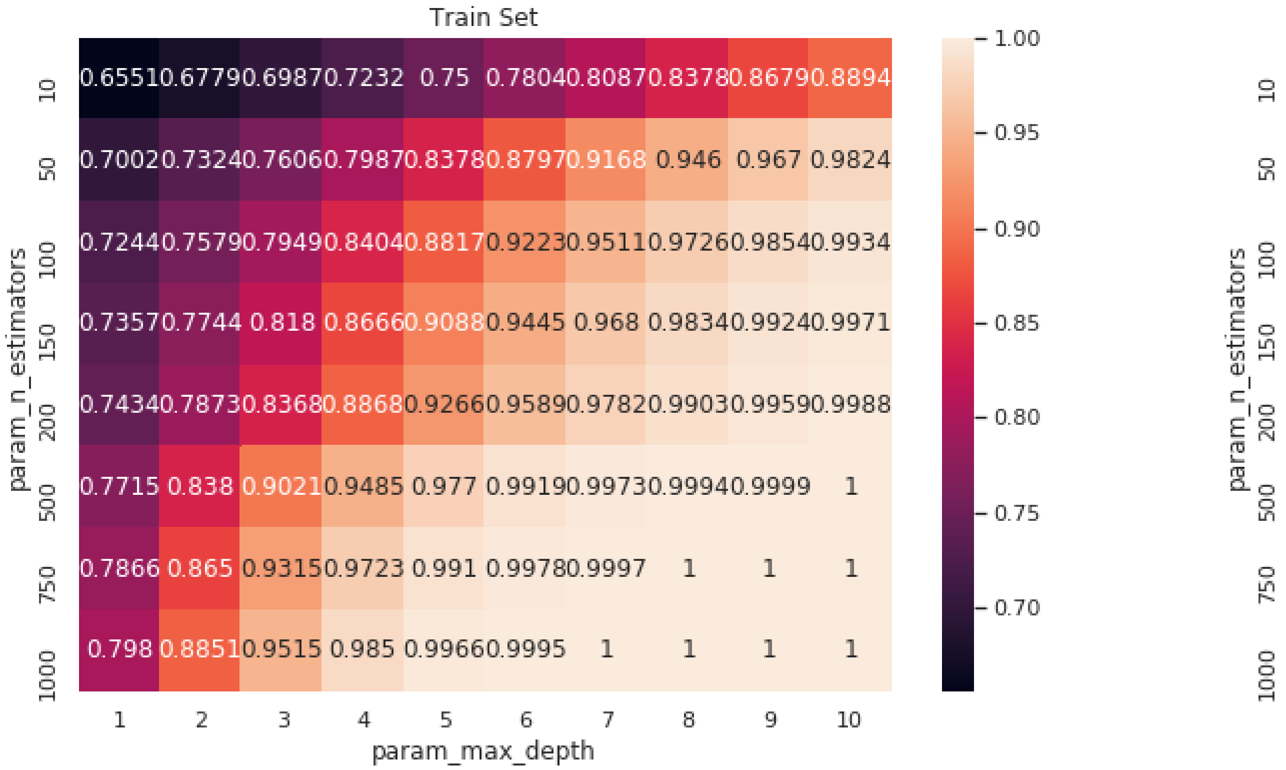
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
```

```
from scipy.sparse import hstack
X_train = hstack((X_train_state_accepted,
X_train_state_rejected,
X_train_prefix_accepted,
X_train_prefix_rejected,
X_train_cat_accepted,
X_train_cat_rejected,
X_train_clean_categories_accepted,
X_train_clean_categories_rejected,
X_train_clean_subcategories_accepted,
X_train_clean_subcategories_rejected,
X_train_price_norm,
X_train_quantity_norm,
X_train_teacher_number_of_previously_posted_projects_norm,
X_train_numerical_data_in_resource_summary_norm,
X_train_number_of_words_in_title,
X_train_number_of_words_in_essay,
train_neg_essay,
train_neu_essay,
train_pos_essay,
train_comp_essay,
X_train_essay_bow,
X_train_titles_bow)).tocsr()

X_test = hstack((X_test_state_accepted,
X_test_state_rejected,
X_test_prefix_accepted,
X_test_prefix_rejected,
X_test_cat_accepted,
X_test_cat_rejected,
X_test_clean_categories_accepted,
X_test_clean_categories_rejected,
X_test_clean_subcategories_accepted,
X_test_clean_subcategories_rejected,
X_test_price_norm,
X_test_quantity_norm,
X_test_teacher_number_of_previously_posted_projects_norm,
X_test_numerical_data_in_resource_summary_norm,
X_test_number_of_words_in_title,
X_test_number_of_words_in_essay,
test_neg_essay,
test_neu_essay,
test_pos_essay,
test_comp_essay,
X_test_essay_bow,
X_test_titles_bow)).tocsr()
```

3.2.1.1 Training the data model to find best hyperparameter

```
# Call train_random_forest function on above data

best_parameters = train_XGB(X_train,y_train)
```

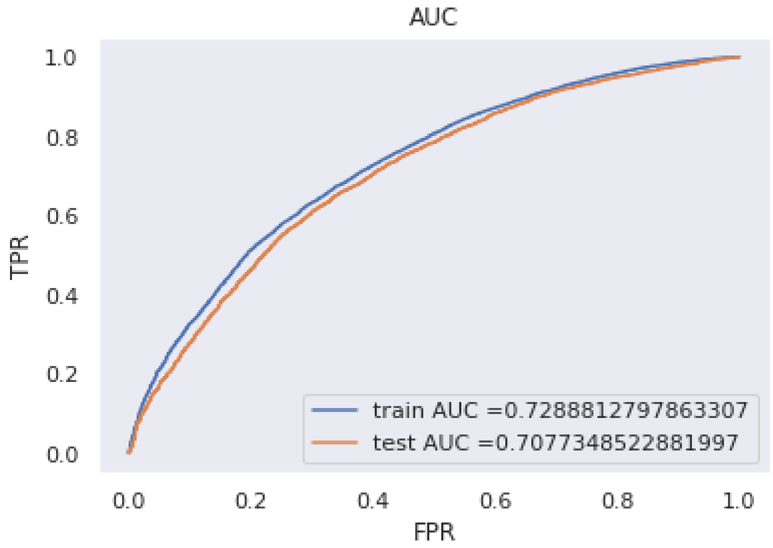
```
Best score:  0.7555041870754474
k value with best score:  {'max_depth': 2, 'n_estimators': 750}
==================================================
dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_tim
```



```
best_depth=best_parameters.get('max_depth')
n_estimators=best_parameters.get('n_estimators')
```

3.2.1.2 Testing the peclformance of the model on test data, plotting ROC Curv

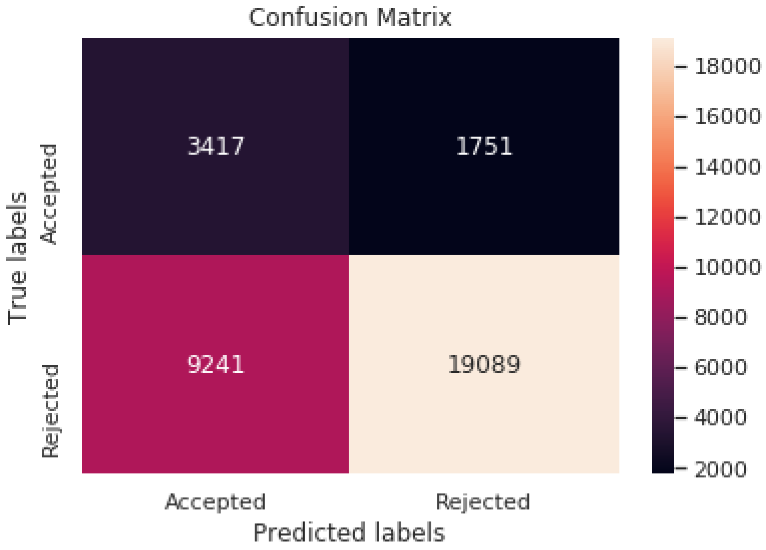
```
train_fpr,train_tpr,tr_thresholds,y_train_pred,y_test_pred=test_XGB(X_train,X_test
```



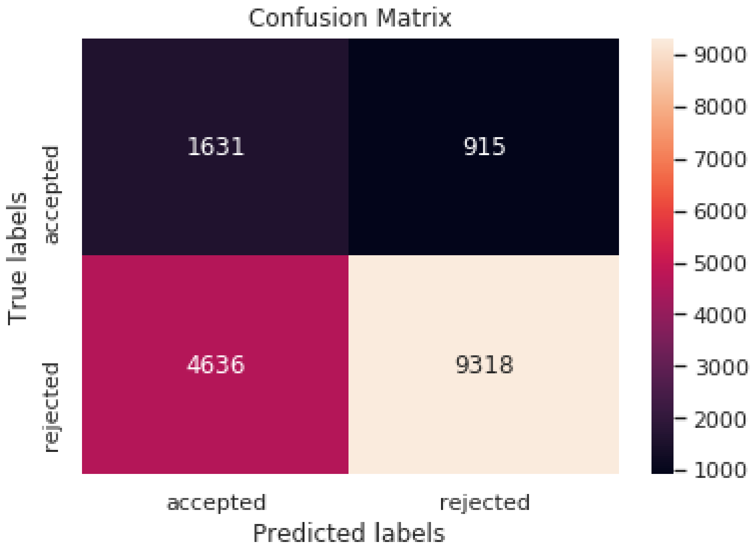
```
print("="*100)
from sklearn.metrics import confusion matrix
```

```
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) 0.4455116623627547 for threshold 0.845
Train confusion matrix
[[ 3417  1751]
 [ 9241 19089]]
```



```
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
[[1631  915]
 [4636 9318]]
```



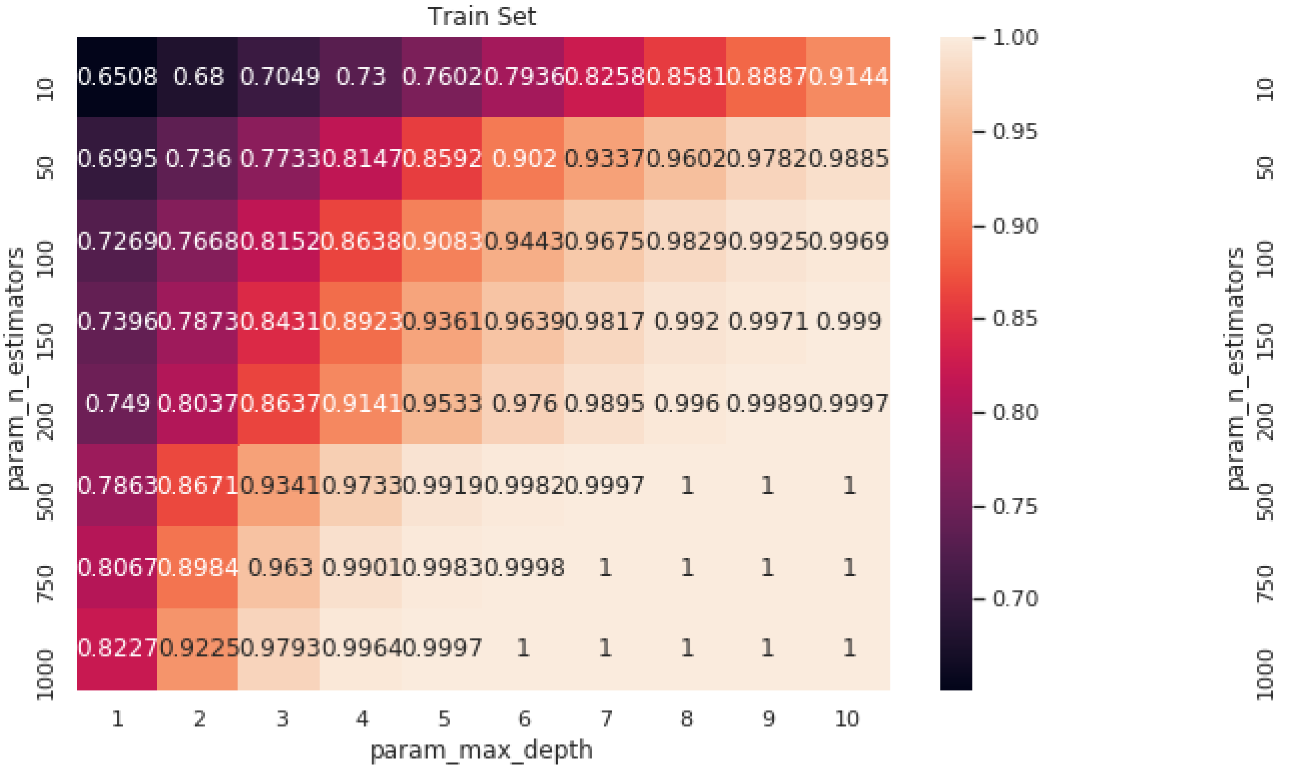
3.2.2 XGBoost on TFIDF vectorization of text data

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_train = hstack((X_train_state_accepted,
X_train_state_rejected,
X_train_prefix_accepted,
X_train_prefix_rejected,
X_train_cat_accepted,
X_train_cat_rejected,
X_train_clean_categories_accepted,
X_train_clean_categories_rejected,
X_train_clean_subcategories_accepted,
X_train_clean_subcategories_rejected,
X_train_price_norm,
X_train_quantity_norm,
X_train_teacher_number_of_previously_posted_projects_norm,
X_train_numerical_data_in_resource_summary_norm,
X_train_number_of_words_in_title,
X_train_number_of_words_in_essay,
train_neg_essay,
train_neu_essay,
train_pos_essay,
train_comp_essay,
X_train_essay_tfidf,
X_train_titles_tfidf)).tocsr()

X_test = hstack((X_test_state_accepted,
X_test_state_rejected,
X_test_prefix_accepted,
X_test_prefix_rejected,
X_test_cat_accepted,
X_test_cat_rejected,
X_test_clean_categories_accepted,
X_test_clean_categories_rejected,
X_test_clean_subcategories_accepted,
X_test_clean_subcategories_rejected,
X_test_price_norm,
X_test_quantity_norm,
X_test_teacher_number_of_previously_posted_projects_norm,
X_test_numerical_data_in_resource_summary_norm,
X_test_number_of_words_in_title,
X_test_number_of_words_in_essay,
test_neg_essay,
test_neu_essay,
test_pos_essay,
test_comp_essay,
X_test_essay_tfidf,
X_test_titles_tfidf)).tocsr()
```

3.2.2.1 Training the data model to find best hyperparameter

```
# Call train_random_forest function on above data

best_parameters = train_XGB(X_train,y_train)
```

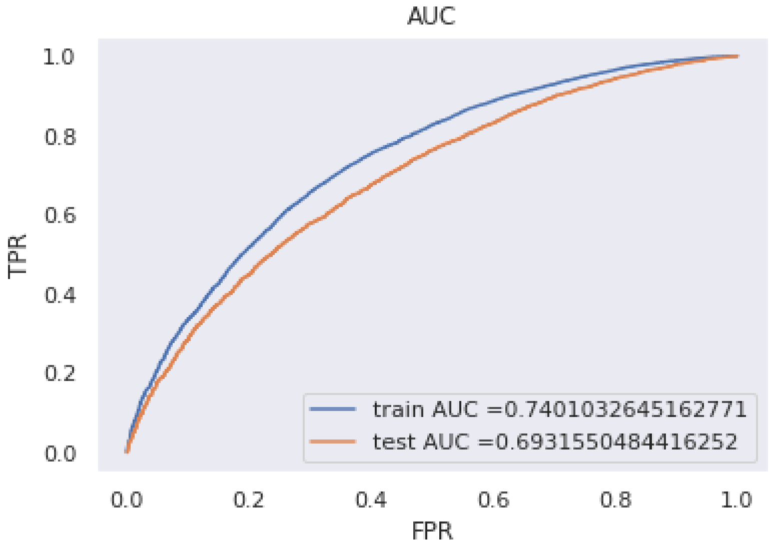
```
Best score:  0.7523980565338868
k value with best score:  {'max_depth': 2, 'n_estimators': 750}
==================================================
dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_tim
```



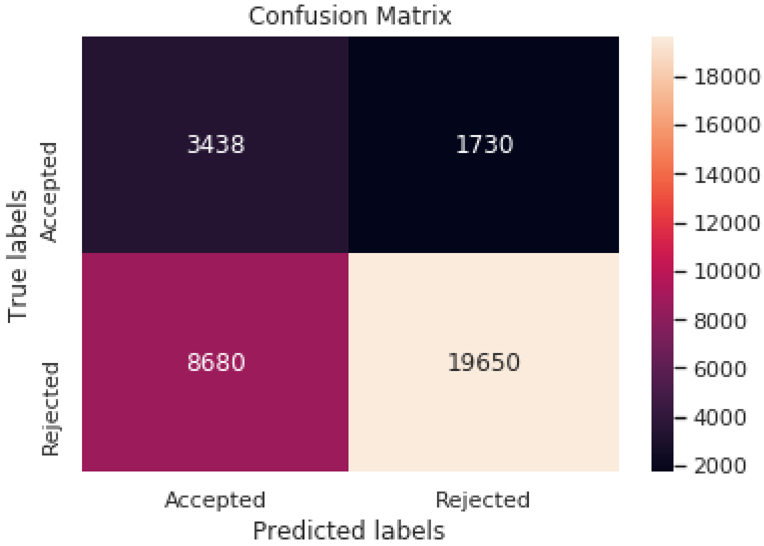
```
best_depth=best_parameters.get('max_depth')
n_estimators=best_parameters.get('n_estimators')
```

3.2.2.2 Testing the peclformance of the model on test data, plotting ROC Curv

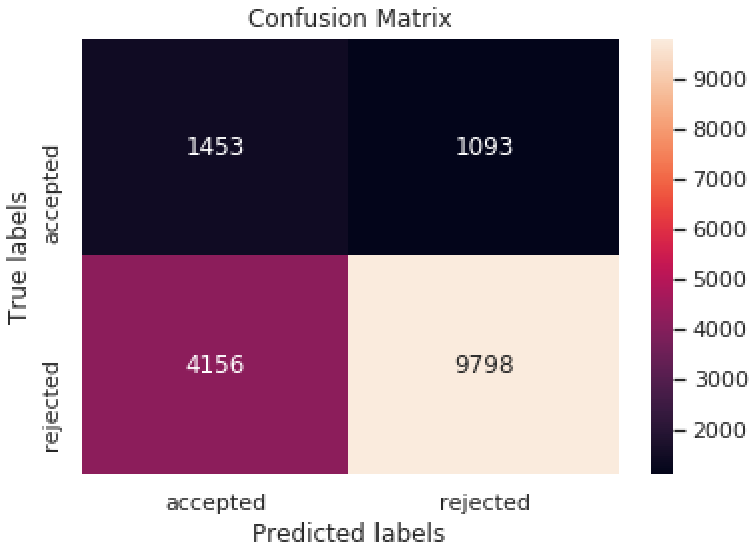
```
train_fpr,train_tpr,tr_thresholds,y_train_pred,y_test_pred=test_XGB(X_train,X_test
```



```
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) 0.4614231158865166 for threshold 0.845
Train confusion matrix
[[ 3438  1730]
 [ 8680 19650]]
```



```
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
[[1453 1093]
 [4156 9798]]
```



3.2.3 XGBoost on AVG W2V vectorization of text data

```
avg_w2v_vectors_essays_train_1 = np.array(avg_w2v_vectors_essays_train)
avg_w2v_vectors_essays_test_1 = np.array(avg_w2v_vectors_essays_test)
avg_w2v_vectors_titles_test_1 = np.array(avg_w2v_vectors_titles_test)
avg_w2v_vectors_titles_train_1 = np.array(avg_w2v_vectors_titles_train)
```

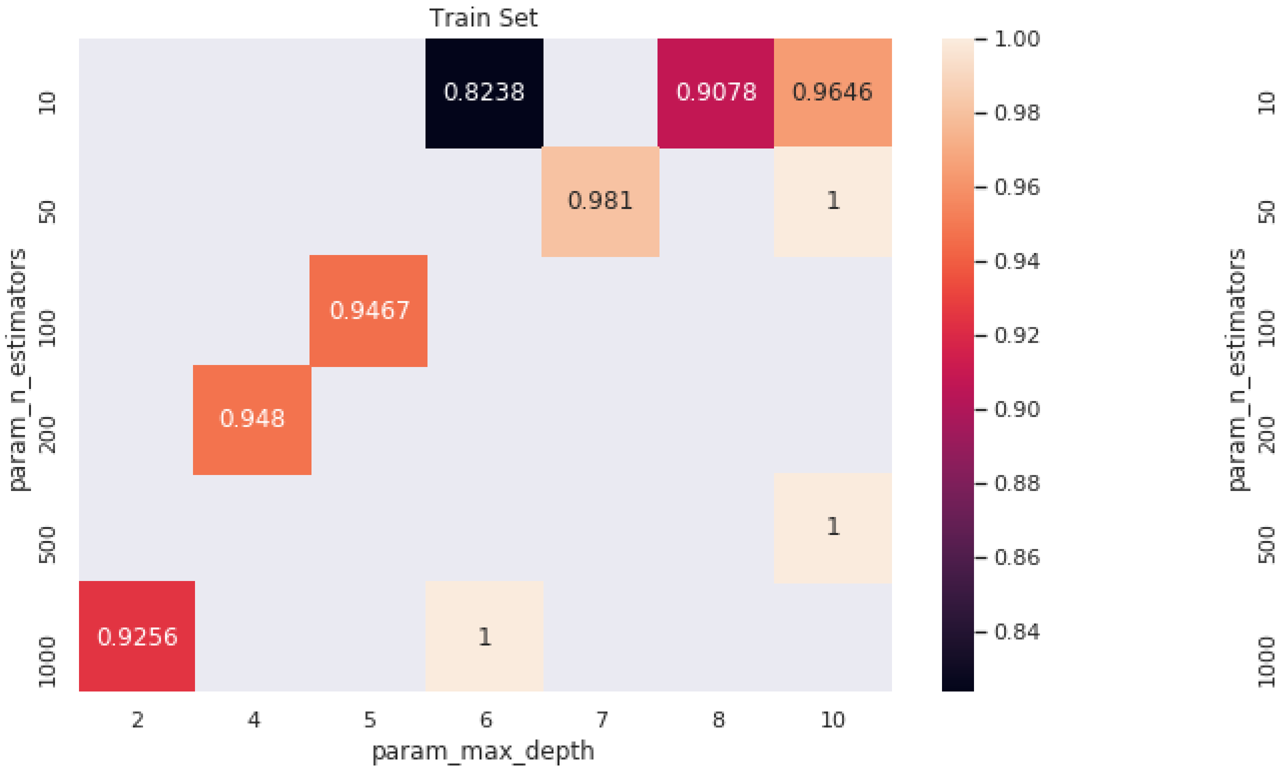
```
from scipy.sparse import coo_matrix, hstack
m1 = coo_matrix(X_train_state_accepted)
m2 = coo_matrix(X_train_state_rejected)
m3 = coo_matrix(X_train_prefix_accepted)
m4 = coo_matrix(X_train_prefix_rejected)
m5 = coo_matrix(X_train_cat_accepted)
m6 = coo_matrix(X_train_cat_rejected)
m7 = coo_matrix(X_train_clean_categories_accepted)
m8 = coo_matrix(X_train_clean_categories_rejected)
m9 = coo_matrix(X_train_clean_subcategories_accepted)
m10 = coo_matrix(X_train_clean_subcategories_rejected)
m11 = coo_matrix(X_train_price_norm)
m12 = coo_matrix(X_train_quantity_norm)
m13 = coo_matrix(X_train_teacher_number_of_previously_posted_projects_norm)
m14 = coo_matrix(X_train_numerical_data_in_resource_summary_norm)
m15 = coo_matrix(X_train_number_of_words_in_title)
m16 = coo_matrix(X_train_number_of_words_in_essay)
m17 = coo_matrix(train_neg_essay)
m18 = coo_matrix(train_neu_essay)
m19 = coo_matrix(train_pos_essay)
m20 = coo_matrix(train_comp_essay)
m21 = coo_matrix(avg_w2v_vectors_essays_train_1)
m22 = coo_matrix(avg_w2v_vectors_titles_train_1)

X_train = hstack([m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12,m13,m14,m15,m16,m17,m18,m
X_train = hstack([m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12,m13,m14,m15,m16,m17,m18,m
```

```
m1 = coo_matrix(X_test_state_accepted)
m2 = coo_matrix(X_test_state_rejected)
m3 = coo_matrix(X_test_prefix_accepted)
m4 = coo_matrix(X_test_prefix_rejected)
m5 = coo_matrix(X_test_cat_accepted)
m6 = coo_matrix(X_test_cat_rejected)
m7 = coo_matrix(X_test_clean_categories_accepted)
m8 = coo_matrix(X_test_clean_categories_rejected)
m9 = coo_matrix(X_test_clean_subcategories_accepted)
m10 = coo_matrix(X_test_clean_subcategories_rejected)
m11 = coo_matrix(X_test_price_norm)
m12 = coo_matrix(X_test_quantity_norm)
m13 = coo_matrix(X_test_teacher_number_of_previously_posted_projects_norm)
m14 = coo_matrix(X_test_numerical_data_in_resource_summary_norm)
m15 = coo_matrix(X_test_number_of_words_in_title)
m16 = coo_matrix(X_test_number_of_words_in_essay)
m17 = coo_matrix(test_neg_essay)
m18 = coo_matrix(test_neu_essay)
m19 = coo_matrix(test_pos_essay)
m20 = coo_matrix(test_comp_essay)
m21 = coo_matrix(avg_w2v_vectors_essays_test_1)
m22 = coo_matrix(avg_w2v_vectors_titles_test_1)

X_test = hstack([m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12,m13,m14,m15,m16,m17,m18,m1
```

3.2.3.1 Training the data model to find best hyperparameter

```
# Call train_random_forest function on above data

best_parameters = train_XGB(X_train,y_train)
```

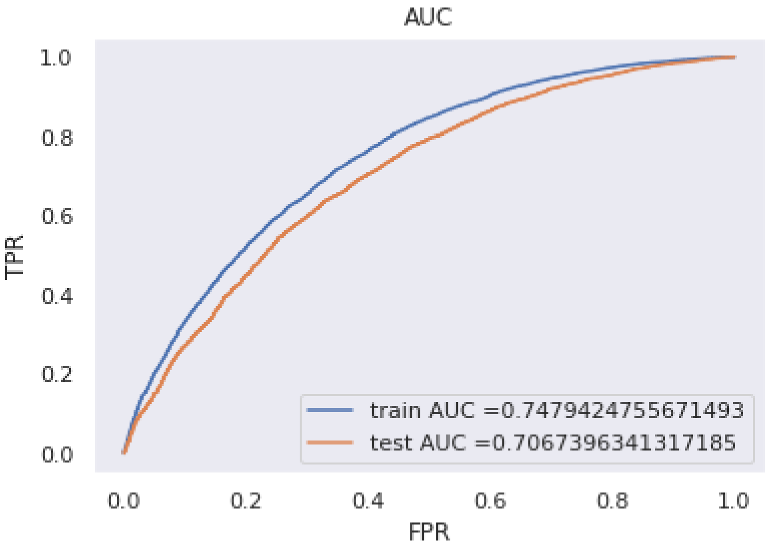
```
Best score:  0.7371774923592856
k value with best score:  {'n_estimators': 200, 'max_depth': 4}
==================================================
dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_tim
```



```
best_depth=best_parameters.get('max_depth')
n_estimators=best_parameters.get('n_estimators')
```

3.2.3.2 Testing the peclformance of the model on test data, plotting ROC Curv

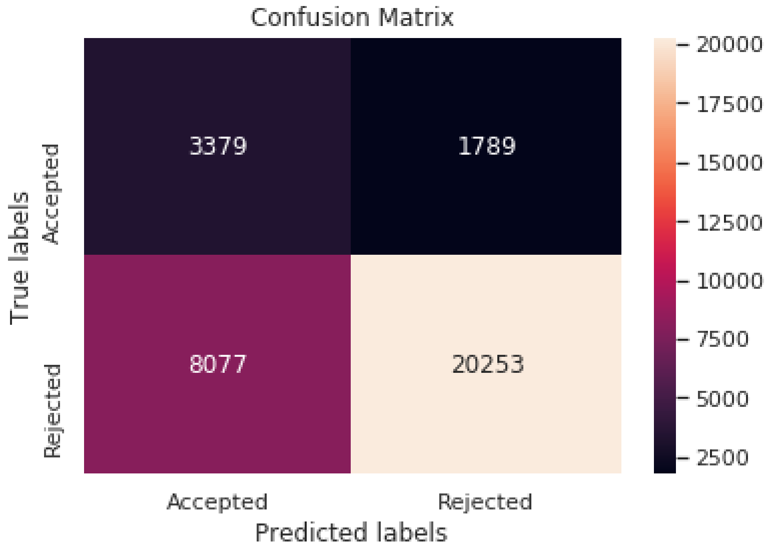
```
train_fpr,train_tpr,tr_thresholds,y_train_pred,y_test_pred=test_XGB(X_train,X_test
```



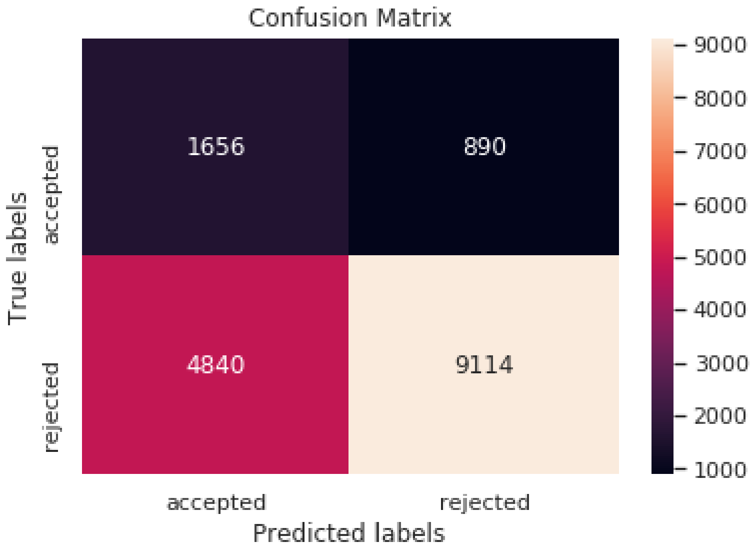
```
print("="*100)
from sklearn.metrics import confusion matrix
```

```
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) 0.4674212742019913 for threshold 0.838
Train confusion matrix
[[ 3379  1789]
 [ 8077 20253]]
```



```
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
[[1656  890]
 [4840 9114]]
```



3.2.4 XGBoost on TFIDF W2V vectorization of text data

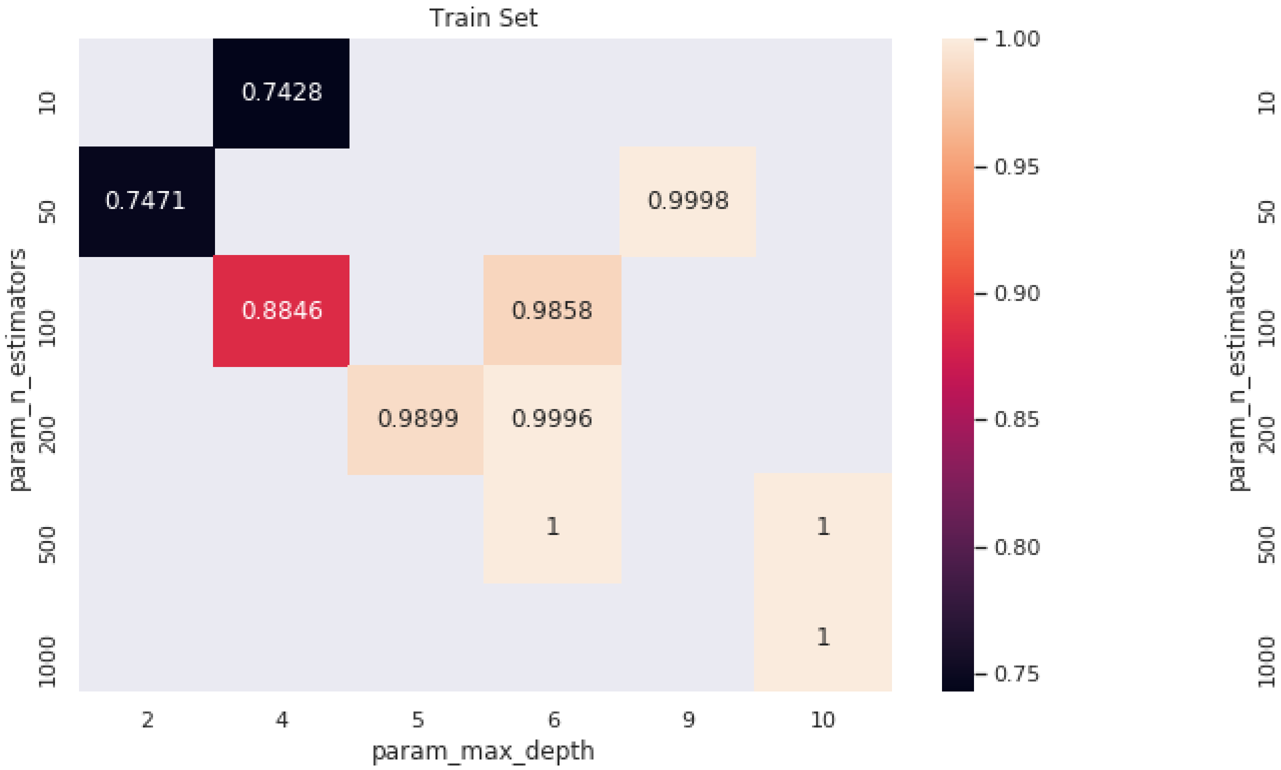
```
tfidf_w2v_vectors_essays_test_1= np.array(tfidf_w2v_vectors_essays_test)
tfidf_w2v_vectors_essays_train_1 = np.array(tfidf_w2v_vectors_essays_train)
tfidf_w2v_vectors_titles_test_1 = np.array(tfidf_w2v_vectors_titles_test)
tfidf_w2v_vectors_titles_train_1 = np.array(tfidf_w2v_vectors_titles_train)
```

```
from scipy.sparse import coo_matrix, hstack
m1 = coo_matrix(X_train_state_accepted)
m2 = coo_matrix(X_train_state_rejected)
m3 = coo_matrix(X_train_prefix_accepted)
m4 = coo_matrix(X_train_prefix_rejected)
m5 = coo_matrix(X_train_cat_accepted)
m6 = coo_matrix(X_train_cat_rejected)
m7 = coo_matrix(X_train_clean_categories_accepted)
m8 = coo_matrix(X_train_clean_categories_rejected)
m9 = coo_matrix(X_train_clean_subcategories_accepted)
m10 = coo_matrix(X_train_clean_subcategories_rejected)
m11 = coo_matrix(X_train_price_norm)
m12 = coo_matrix(X_train_quantity_norm)
m13 = coo_matrix(X_train_teacher_number_of_previously_posted_projects_norm)
m14 = coo_matrix(X_train_numerical_data_in_resource_summary_norm)
m15 = coo_matrix(X_train_number_of_words_in_title)
m16 = coo_matrix(X_train_number_of_words_in_essay)
m17 = coo_matrix(train_neg_essay)
m18 = coo_matrix(train_neu_essay)
m19 = coo_matrix(train_pos_essay)
m20 = coo_matrix(train_comp_essay)
m21 = coo_matrix(tfidf_w2v_vectors_essays_train_1)
m22 = coo_matrix(tfidf_w2v_vectors_titles_train_1)

X_train = hstack([m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12,m13,m14,m15,m16,m17,m18,m
m1 = coo_matrix(X_test_state_accepted)
m2 = coo_matrix(X_test_state_rejected)
m3 = coo_matrix(X_test_prefix_accepted)
m4 = coo_matrix(X_test_prefix_rejected)
m5 = coo_matrix(X_test_cat_accepted)
m6 = coo_matrix(X_test_cat_rejected)
m7 = coo_matrix(X_test_clean_categories_accepted)
m8 = coo_matrix(X_test_clean_categories_rejected)
m9 = coo_matrix(X_test_clean_subcategories_accepted)
m10 = coo_matrix(X_test_clean_subcategories_rejected)
m11 = coo_matrix(X_test_price_norm)
m12 = coo_matrix(X_test_quantity_norm)
m13 = coo_matrix(X_test_teacher_number_of_previously_posted_projects_norm)
m14 = coo_matrix(X_test_numerical_data_in_resource_summary_norm)
m15 = coo_matrix(X_test_number_of_words_in_title)
m16 = coo_matrix(X_test_number_of_words_in_essay)
m17 = coo_matrix(test_neg_essay)
m18 = coo_matrix(test_neu_essay)
m19 = coo_matrix(test_pos_essay)
m20 = coo_matrix(test_comp_essay)
m21 = coo_matrix(tfidf_w2v_vectors_essays_test_1)
m22 = coo_matrix(tfidf_w2v_vectors_titles_test_1)

X_test = hstack([m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12,m13,m14,m15,m16,m17,m18,m1
```

3.2.4.1 Training the data model to find best hyperparameter

```
# Call train_random_forest function on above data

best_parameters = train_XGB(X_train,y_train)
```

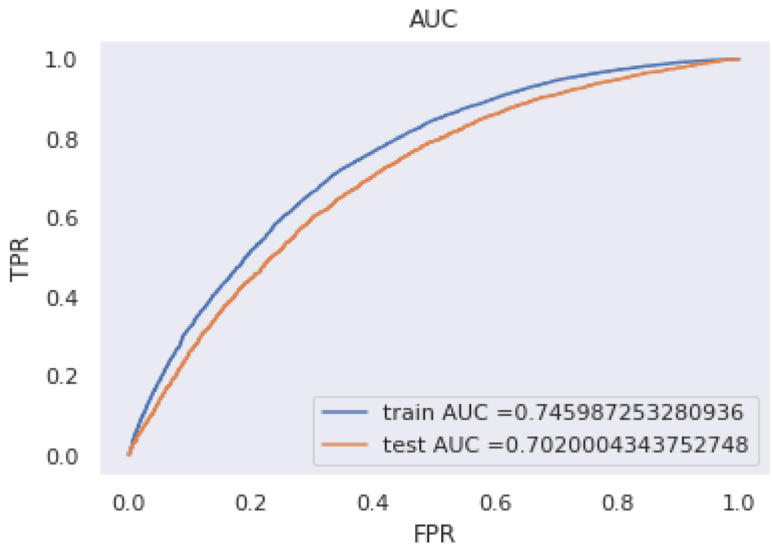
```
Best score:  0.7337654749246649
k value with best score:  {'n_estimators': 100, 'max_depth': 4}
==================================================
dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_tim
```



```
best_depth=best_parameters.get('max_depth')
n_estimators=best_parameters.get('n_estimators')
```

3.2.4.2 Testing the peclformance of the model on test data, plotting ROC Curv

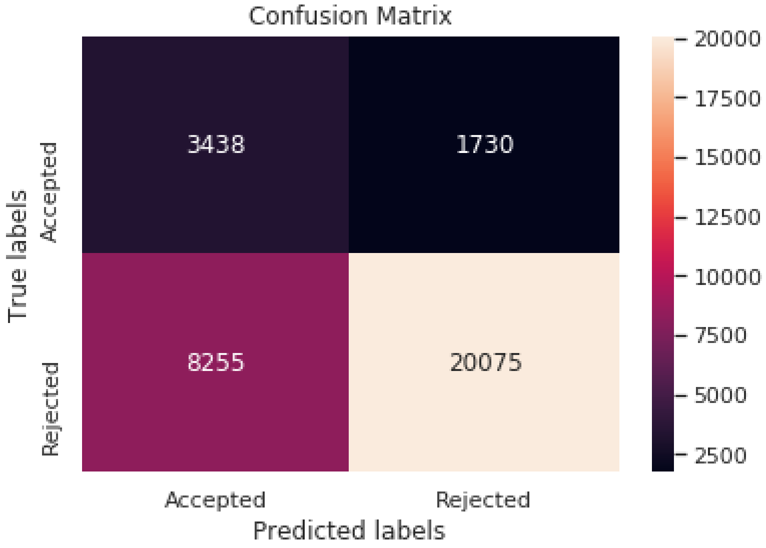
```
train_fpr,train_tpr,tr_thresholds,y_train_pred,y_test_pred=test_XGB(X_train,X_test
```



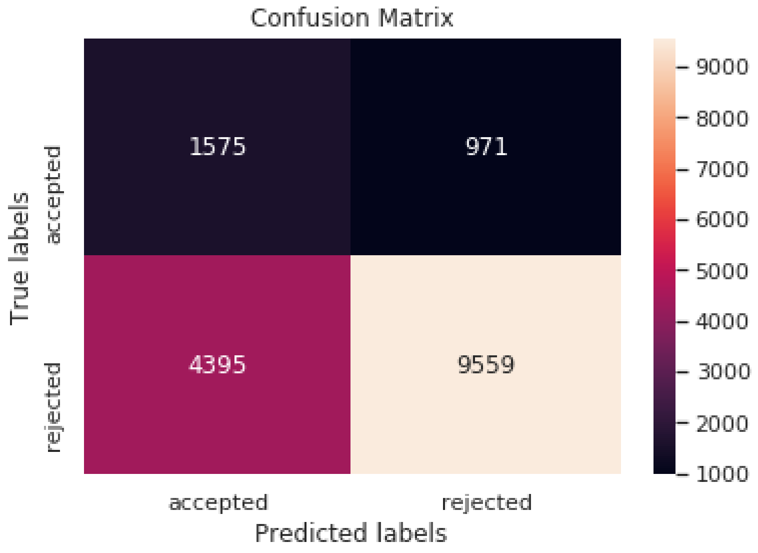
```
print("="*100)
from sklearn.metrics import confusion matrix
```

```
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) 0.4714030051614158 for threshold 0.837
Train confusion matrix
[[ 3438  1730]
 [ 8255 20075]]
```



```
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
[[1575  971]
 [4395 9559]]
```



4.1 Summary

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer", "Model","Max_depth" ,"n_estimators","Test AUC"]
x.add_row(["BOW", "RF", 10,1000, 0.7213])
x.add_row(["TFIDF", "RF", 10, 1000, 0.7080])
x.add_row(["TFIDF W2V", "RF", 7,1000, 0.7067])
x.add_row(["AVG W2V", "RF", 8, 1000, 0.7190])
x.add_row(["BOW", "GBDT",2,750, 0.7077])
x.add_row(["TFIDF", "GBDT",2,750, 0.6931])
x.add_row(["AVG W2V", "GBDT", 4,200, 0.7067])
x.add_row(["TFIDF W2V", "GBDT", 4,100 ,0.7020])
print(x)
```

```
+------------+-------+-----------+--------------+----------+
| Vectorizer | Model | Max_depth | n_estimators | Test AUC |
+------------+-------+-----------+--------------+----------+
|    BOW     |   RF  |     10    |     1000     |  0.7213  |
|   TFIDF    |   RF  |     10    |     1000     |  0.708   |
| TFIDF W2V  |   RF  |     7     |     1000     |  0.7067  |
|  AVG W2V   |   RF  |     8     |     1000     |  0.719   |
|    BOW     |  GBDT |     2     |     750      |  0.7077  |
|   TFIDF    |  GBDT |     2     |     750      |  0.6931  |
|  AVG W2V   |  GBDT |     4     |     200      |  0.7067  |
| TFIDF W2V  |  GBDT |     4     |     100      |  0.702   |
+------------+-------+-----------+--------------+----------+
```