DonorsChoose

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

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

- How to scale current manual processes and resources to screen 500,000 projects so that th as possible
- How to increase the consistency of project vetting across different volunteers to improve th
- How to focus volunteer time on the applications that need the most assistance

The goal of the competition is to predict whether or not a DonorsChoose.org project proposal sub the text of project descriptions as well as additional metadata about the project, teacher, and sch information to identify projects most likely to need further review before approval.

About the DonorsChoose Data Set

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

Feature	Description
`project_id`	A unique identifier for the proposed project. **Example:** `p036502`
`project_title`	Title of the project. **Examples:** • `Art Will Make You Happy!` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targeted. One of 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). **Exam**

Feature	Description
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the pr • Literacy • Literature & Writing, Social Scienc
`project_resource_summary`	An explanation of the resources needed for the project. **Example:** • My students need hands on literacy 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. **Examp**
`teacher_prefix`	Teacher's title. One of the following enumerated values: • nan • Dr. • Mr. • Mrs. • Ms. • Teacher.
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted by the same

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

Additionally, the `resources.csv` data set provides more data about the resources required for ea resource required by a project:

Feature	Description
`id`	A `project_id` value from the `train.csv` file. **Example:** p0365
`description`	Desciption of the resource. **Example:** `Tenor Saxophone Reeds`
`quantity`	Quantity of the resource required. **Example:** `3`
`price`	Price of the resource required. **Example:** `9.95`

Note: Many projects require multiple resources. The `id` value corresponds to a `project_id` in tr resources needed for a project:

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

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

Notes on the Essay Data

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

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

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

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

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

```
# importing required libraries

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

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

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

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

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

from tqdm import tqdm
import os

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

```
from sklearn.model_selection import GridSearchCV
```

1.1 Reading Data

```
from google.colab import drive

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

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

```
!ls "/content/drive/My Drive/Colab Notebooks/Dataset/Assignments_DonorsChoose_2018
```

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

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

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



# Getting basic information about the data

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

```
Number of data points in Project_train data (109248, 17)
---------------------------------------------------------------------------
The attributes of Project_train data : ['Unnamed: 0' 'id' 'teacher_id' 'teach
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
===========================================================================
Number of data points in Resource_train data (1541272, 4)
---------------------------------------------------------------------------
The attributes of Resource_train data : ['id' 'description' 'quantity' 'price
```

1.2 Data Pre-Processing

```
# Merge two column text dataframe:
# Merge 4 essays into one:
project_data["essay"] = project_data["project_essay_1"].map(str) +\
                        project_data["project_essay_2"].map(str) + \
                        project_data["project_essay_3"].map(str) + \
                        project_data["project_essay_4"].map(str)

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

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

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

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

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

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

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

project_data.head(2)
```

4	teacher_number_of_previously_posted_projects	project_is_approved	essay
o y ..	53	1	I have been fortunate enough to use the Fairy ...
s n ..	4	1	Imagine being 8-9 years old. You're in your th...

1.2.1 Pre-Processing Essay Text

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

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

```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
```

```
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase


# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "y
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'h
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that'
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has'
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because',
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'thr
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off'
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all',
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than',
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've"
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "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%|██████████| 109245/109245 [01:04<00:00, 1691.42it/s]
```

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

```
lways put smile face they big personalities even bigger dedication learning th
```

1.2.2 Pre-Processing Project Title Text

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

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

# Adding preprocessed_titles coloumn to our data matrix

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

```
100%|██████████| 109245/109245 [00:02<00:00, 39048.52it/s]
'empowering students through art learning about then now'
```

1.2.3 Pre-Processing Project Grades

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

# Adding preprocessed_titles coloumn to our data matrix

project_data['preprocessed_grade_category']=preprocessed_grade_categories

project_data.head(5)
```

e	quantity	numerical_data_in_resource_summary	preprocessed_essays	preproces
5	4	0	i fortunate enough use fairy tale stem kits cl...	engir prima
3	8	0	imagine 8 9 years old you third grade classroo...	senso
0	1	0	having class 24 students comes diverse learner...	mobile le lis
4	9	0	i recently read article giving students choice...	flexible s
4	14	0	my students crave challenge eat obstacles brea...	going deep

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)
```

```
        j = j.replace('  ','') # we are placeing all the '  '(space) with ''(empty)
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailir
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

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

1.2.5 preprocessing of `project_subject_subcategories`

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

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-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 trailir
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

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



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



project_data.head(5)
```

	_summary	preprocessed_essays	preprocessed_titles	preprocessed_grade_category
	0	i fortunate enough use fairy tale stem kits cl...	engineering steam primary classroom	grades_prek_2
	0	imagine 8 9 years old you third grade classroo...	sensory tools focus	grades_3_5
	0	having class 24 students comes diverse learner...	mobile learning mobile listening center	grades_prek_2
	0	i recently read article giving students choice...	flexible seating flexible learning	grades_prek_2
	0	my students crave challenge eat obstacles brea...	going deep the art inner thinking	grades_3_5

1.2.6 Add Sentiment Score of Preprocessed Essays

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

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

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

sid = SentimentIntensityAnalyzer()

for sent in preprocessed_titles:

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

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

```
project_data['comp_essay']=comp_essay


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


project_data.head(5)
```

of_previously_posted_projects	project_is_approved	price	quantity	numerical_
53	1	725.05	4	
4	1	213.03	8	
10	1	329.00	1	
2	1	481.04	9	
2	1	17.74	14	

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()
```

	clean_subcategories	neg_essay	neu_essay	pos_essay	comp_essay	number_of_wo
	AppliedSciences Health_LifeScience	0.0	1.000	0.000	0.0000	
	SpecialNeeds	0.0	1.000	0.000	0.0000	
	Literacy	0.0	1.000	0.000	0.0000	
	EarlyDevelopment	0.0	0.345	0.655	0.4215	
	Literacy	0.0	1.000	0.000	0.0000	

1.3 Sampling data for decision_tree Assignment

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

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

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

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

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

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

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

	clean_subcategories	neg_essay	neu_essay	pos_essay	comp_essay	number_of_wo
	AppliedSciences Health_LifeScience	0.0	1.0	0.0	0.0	

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

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


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

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

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

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

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

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

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

```
(124222, 22)
```

2.2 Make Data Model Ready:

2.2.1 Encoding numerical, categorical features

2.2.1.1 Encoding School State

```
# Encoding School State

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

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

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
#print(X_cv_state_ohe.shape, y_cv.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer.get feature names())
```

```
print("="*100)
```

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

2.2.1.2 Encoding Teacher Prefix

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

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

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

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

2.2.1.3 Encoding preprocessed_grade_category

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

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

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

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

2.2.1.4 Encoding numerical feature Price

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

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

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


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

```
After vectorizations
(124222, 1) (124222,)
[[0.00074649]
 [0.00086372]
 [0.00567798]
 ...
 [0.00071316]
 [0.00070756]
 [0.00034576]]
(36051, 1) (36051,)
==================================================================================
```

2.2.1.5 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'].values.reshape(1,-1))

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

```
[[0.00016963]
 [0.00127225]
 [0.0010178 ]
 ...
 [0.00042408]
 [0.00025445]
 [0.00186597]]
After vectorizations
(124222, 1) (124222,)
(36051, 1) (36051,)
==========================================================================
```

2.2.1.6 Encoding numeric feature teacher_number_of_previously_posted_proje

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

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

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

```
[[0.        ]
 [0.00017525]
 [0.00315453]
 ...
 [0.00052575]
 [0.0003505 ]
 [0.00175252]]
After vectorizations
(124222, 1) (124222,)
(36051, 1) (36051,)
==========================================================================
```

2.2.1.7 Encoding numeric feature numerical_data_in_resource_summary

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

X_train_numerical_data_in_resource_summary_norm = X_train_numerical_data_in_resour
X_test_numerical_data_in_resource_summary_norm = X_test_numerical_data_in_resource

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

```
[[0.        ]
 [0.        ]
 [0.        ]
 ...
 [0.01151023]
 [0.        ]
 [0.        ]]
After vectorizations
(124222, 1) (124222,)
(36051, 1) (36051,)
==========================================================================
```

2.2.1.8 Encoding numeric feature number_of_words_in_title

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

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

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


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

```
After vectorizations
(124222, 1) (124222,)
[[0.00256739]
 [0.00154043]
 [0.00308086]
 ...
 [0.00256739]
 [0.00102695]
 [0.00154043]]
(36051, 1) (36051,)
====================================================================================
```

2.2.1.9 Encoding numeric feature number_of_words_in_essay

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

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

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

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

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

```
After vectorizations
(124222, 1) (124222,)
[[0.00220319]
 [0.00316593]
 [0.00309187]
 ...
 [0.00188845]
 [0.00194399]
 [0.00209211]]
(36051, 1) (36051,)
====================================================================================
```

2.2.1.10 Encoding numeric features of sentiment Score

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

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

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

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

2.3 Appling decision_tree on different kind of featurization a instructions

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

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

```
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_decision_tree(X_tr,y_train):
    depth = [1,5,10,50]
    min_sample = [5,10,100,500]
    train_score = []
    test_score =[]
    min_sample_list = []
    depth_list = []

    decision_tree = tree.DecisionTreeClassifier()
    #create a dictionary of all values we want to test for alpha values
    param_grid = {'max_depth': depth, 'min_samples_split':min_sample}

    decision_tree_gscv = GridSearchCV(decision_tree, param_grid, cv=2, scoring='ro
    decision_tree_gscv.fit(X_tr, y_train)
    print(decision_tree_gscv.best_params_)

    #importance = decision_tree_gscv.best_estimator_.feature_importances_
    print(decision_tree_gscv.cv_results_.keys())

    for key, value in decision_tree_gscv.cv_results_.items():
        if key == "mean_train_score":
            train_score = value
        if key == "mean_test_score":
            test_score = value
        if key == "param_min_samples_split":
           min_sample_list = value
        if key == "param_max_depth":
           depth_list= value
    '''
    print(len(min_sample))
    print(len(depth))
    print(len(min_sample_list))
    print(len(depth_list))
    print(len(train_score))
    print(len(test_score))
    print(train_score)
    print(test_score)
    print(depth_list)
```

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

    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=min_sample, yticklabels=depth,annot=True,fmt='
    plt.ylabel('Depth')
    plt.xlabel('Min_sample')
    plt.show()
    sns.heatmap(array2, xticklabels=min_sample, yticklabels=depth,annot=True,fmt='
    plt.ylabel('Depth')
    plt.xlabel('Min_sample')
    plt.show()

'''
    feature_imp = importance
    # https://plot.ly/python/3d-axes/
    trace1 = go.Scatter3d(x=min_sample,y=depth,z=train_score, name = 'train')
    trace2 = go.Scatter3d(x=min_sample,y=depth,z=test_score, name = 'Cross validat
    data = [trace1, trace2]

    layout = go.Layout(scene = dict(
            xaxis = dict(title='n_estimators'),
            yaxis = dict(title='max_depth'),
            zaxis = dict(title='AUC'),))

    fig = go.Figure(data=data, layout=layout)
    offline.iplot(fig, filename='3d-scatter-colorscale')


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

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

    plt.legend()
    plt.xlabel("alpha: hyperparameter")
    plt.ylabel("AUC")
    plt.title("ERROR PLOTS")
    plt.grid()
    plt.show()
    '''
# Test the model with optimal alpha found out using training data. Plot FPR vs TPR

def test_decision_tree(X_tr,X_te,best_depth,best_min_sample):

    y_train_pred=[]
    y_test_pred=[]

    from sklearn.metrics import roc_curve, auc
```

```
    '''
    decision_tree = tree.DecisionTreeClassifier(max_depth= best_depth, min_samples
    decision_tree.fit(X_tr, y_train)

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

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

    y_train_pred = decision_tree.predict_proba(X_tr)[:,1]

    y_test_pred = decision_tree.predict_proba(X_te)[:,1]


    for i in y_train_pred_raw:
        y_train_pred.append(i[1])
    for i in y_test_pred_raw:
        y_test_pred.append(i[1])
       '''


    decision_tree = tree.DecisionTreeClassifier(max_depth= best_depth, min_samples
    decision_tree.fit(X_tr, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estim
    # not the predicted outputs

    y_train_pred = batch_predict(decision_tree, X_tr)
    y_test_pred = batch_predict(decision_tree, X_te)


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

    plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tp
    plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
    plt.legend()
    plt.xlabel("tpr")
    plt.ylabel("fpr")
    plt.title("ERROR PLOTS")
    plt.grid()
    plt.show()
    return train_fpr,train_tpr,tr_thresholds,y_train_pred,y_test_pred

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

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

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

2.3.2 Applying decision_tree on TFIDF encoding eassay, and project

2.3.2.1 Encoding preprocessed_titles TFIDF

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

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

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

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

```
After vectorizations
(124222, 3809) (124222,)
(36051, 3809) (36051,)
====================================================================================
```

2.3.2.2 Encoding preprocessed_essays TFIDF

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

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

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

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

2.3.2.3 Merge all the features and obtain final data matrix

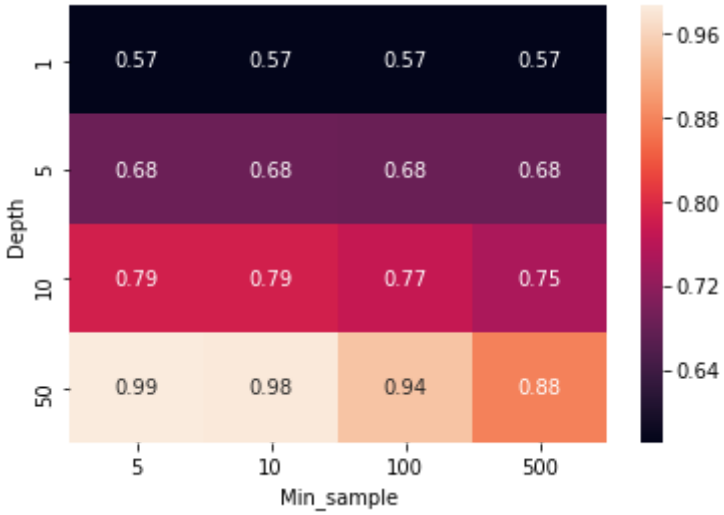
```
from scipy.sparse import hstack
X_tr = hstack((X_train_essay_tfidf,X_train_state_ohe, X_train_teacher_ohe, X_trair
#X_cr = hstack((X_cv_titles_tfidf,X_cv_essay_tfidf, X_cv_state_ohe, X_cv_teacher_c
X_te = hstack((X_test_essay_tfidf,X_test_state_ohe, X_test_teacher_ohe, X_test_gra

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

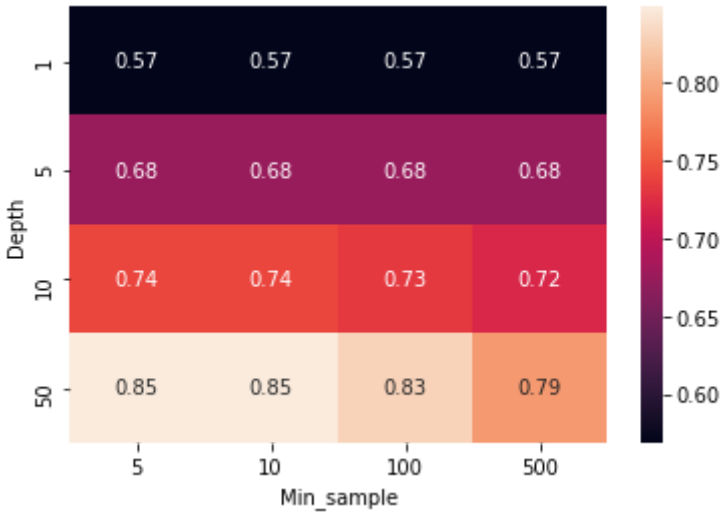
```
Final Data matrix
(124222, 10069) (124222,)
(36051, 10069) (36051,)
==========================================================================
```

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

```
# Call train_decision_tree function on above data

train_decision_tree(X_tr,y_train)
```

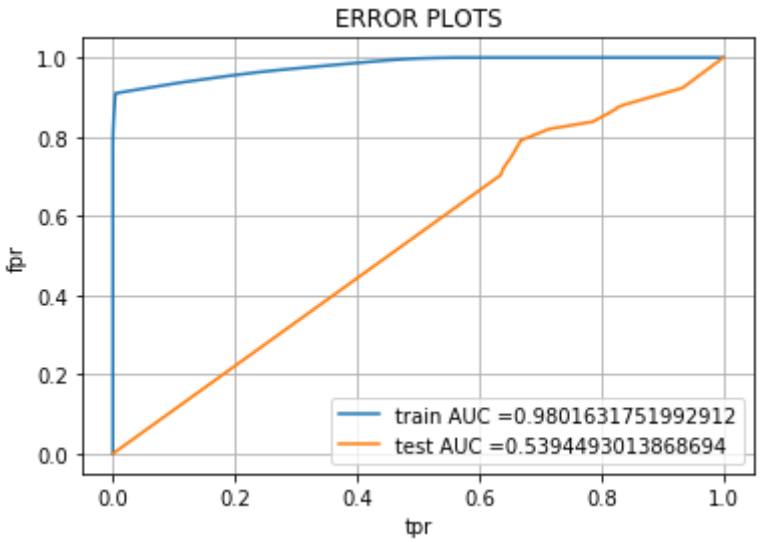
```
{'max_depth': 50, 'min_samples_split': 10}
dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_tim
4
4
16
16
16
16
[0.57044555 0.57044555 0.57044555 0.57044555 0.68441528 0.6843992
 0.68421257 0.68368722 0.78656151 0.78615658 0.77267713 0.7467282
 0.98659125 0.98480526 0.94401021 0.87711257]
[0.56864407 0.56864407 0.56864407 0.56864407 0.67578732 0.67577112
 0.67572986 0.6753808  0.74153063 0.74140304 0.73308518 0.72073372
 0.84801058 0.84899491 0.83086662 0.79062823]
[1 1 1 1 5 5 5 5 10 10 10 10 50 50 50 50]
```





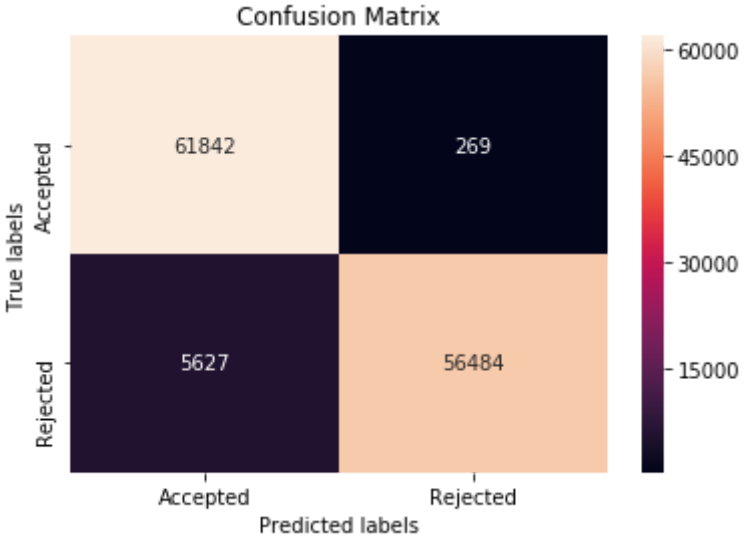
2.3.2.5 Testing the performance of the model on test data, plotting ROC Curve

```
st_depth=50
st_min_sample = 10
ain_fpr,train_tpr,tr_thresholds,y_train_pred,y_test_pred=test_decision_tree(X_tr,X
```



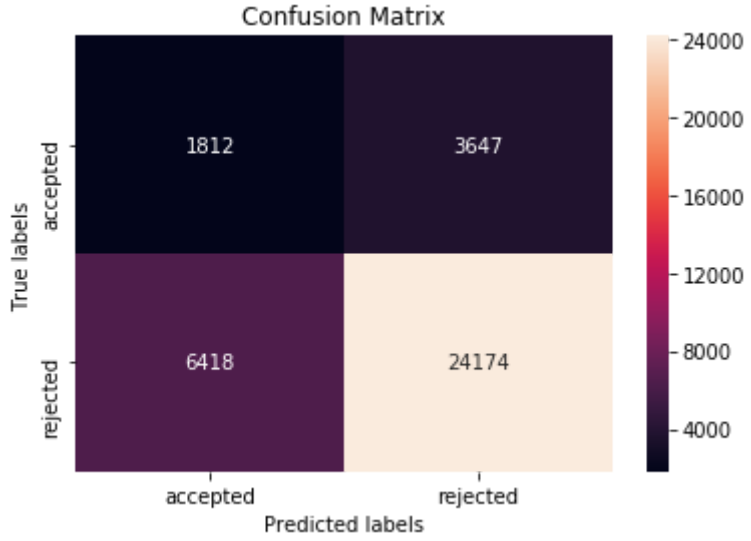

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")

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) 0.905465542153189 for threshold 0.667
Train confusion matrix
[[61842   269]
 [ 5627 56484]]
```



```
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
[[ 1812  3647]
 [ 6418 24174]]
```



2.3.3 Applying decision_tree on TFIDF W2V

2.3.3.1 Encoding preprocessed_titles tfidf W2V

```
with open('/content/drive/My Drive/Colab Notebooks/Dataset/Assignments_DonorsChoos
    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 stc
for sentence in tqdm(X_train['preprocessed_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_titles_train.append(vector)

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

```
100%|██████████| 124222/124222 [00:05<00:00, 21280.36it/s]124222
300
```

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_titles_test = []; # the avg-w2v for each sentence/review is stor
for sentence in tqdm(X_test['preprocessed_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_titles_test.append(vector)

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

```
100%|██████████| 36051/36051 [00:01<00:00, 22446.77it/s]36051
300
```

2.3.3.2 Encoding preprocessed_essays tfidf W2V

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['preprocessed_essays'])
# we are converting a dictionary with word as a key, and the idf as a value
```

```
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())

# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_essays_train = []; # the avg-w2v for each sentence/review is stc
for sentence in tqdm(X_train['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_essays_train.append(vector)

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

```
100%|██████████| 124222/124222 [04:17<00:00, 482.59it/s]124222
300
```

```
tfidf_w2v_vectors_essays_test = []; # the avg-w2v for each sentence/review is stor
for sentence in tqdm(X_test['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_essays_test.append(vector)

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

```
100%|██████████| 36051/36051 [01:17<00:00, 465.93it/s]36051
300
```

2.3.3.3 Merge all the features and obtain final data matrix

```
from scipy.sparse import hstack
X_tr = hstack((X_train_essay_tfidf,X_train_state_ohe, X_train_teacher_ohe, X_trair
#X_cr = hstack((X_cv_titles_tfidf,X_cv_essay_tfidf, X_cv_state_ohe, X_cv_teacher_c
```

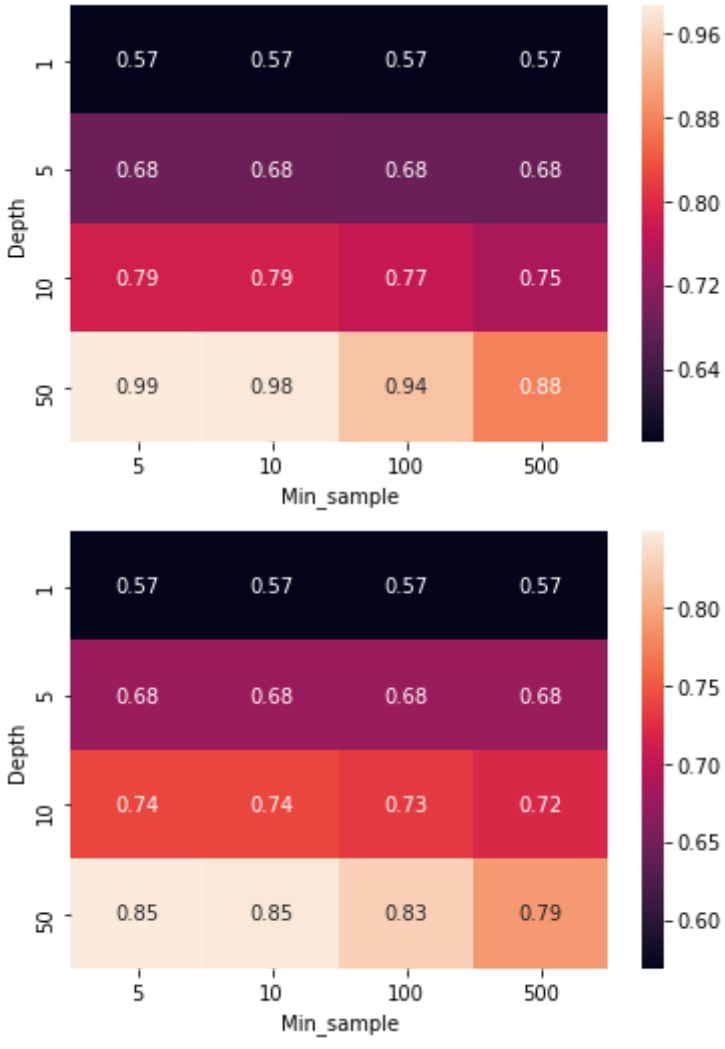
```
X_te = hstack((X_test_essay_tfidf,X_test_state_ohe, X_test_teacher_ohe, X_test_gra

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

```
Final Data matrix
(124222, 10069) (124222,)
(36051, 10069) (36051,)
==========================================================================================
```

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

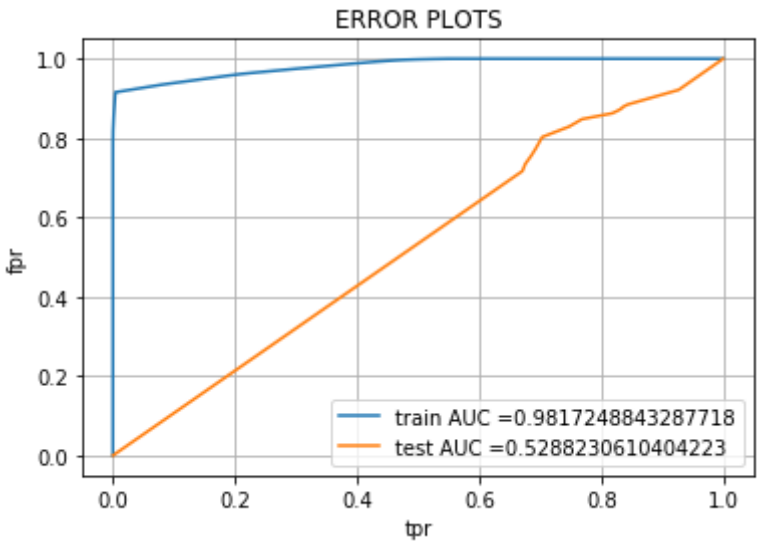
```
# Call train_decision_tree function on above data
train_decision_tree(X_tr,y_train)
```

```
{'max_depth': 50, 'min_samples_split': 5}
dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_tim
```



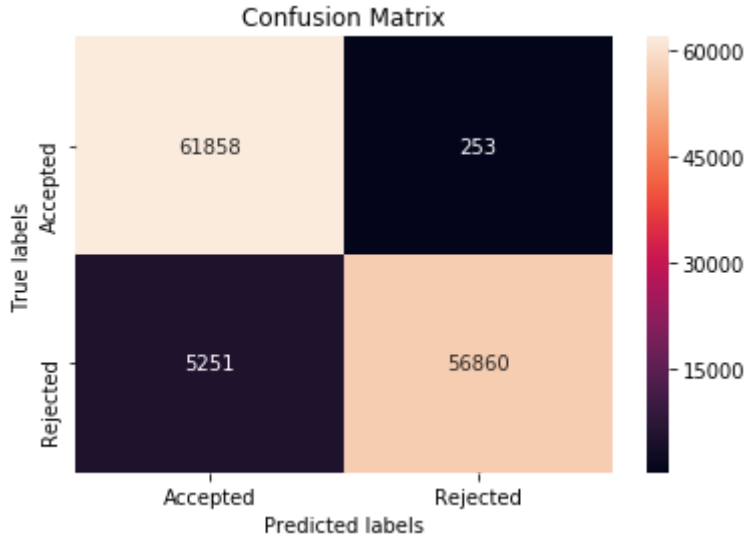
2.3.3.5 Testing the performance of the model on test data, plotting ROC Curve

```
best_depth=50
best_min_sample =5
train_fpr,train_tpr,tr_thresholds,y_train_pred,y_test_pred=test_decision_tree(X_tr
```



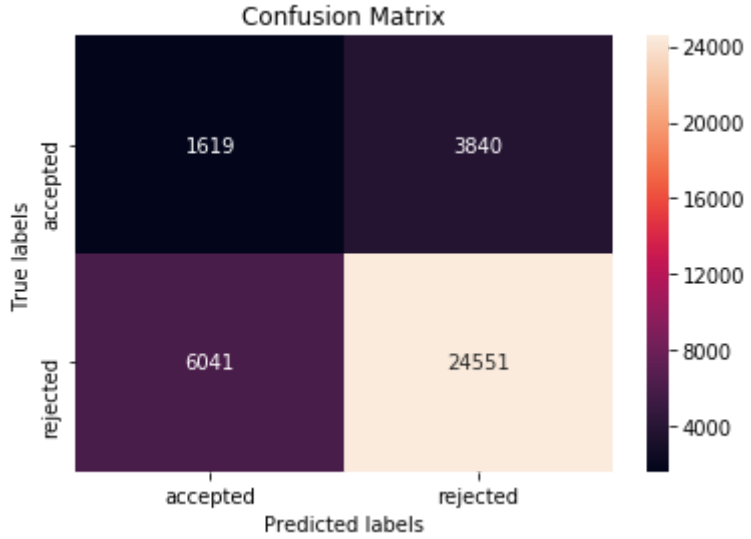

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")

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) 0.9117288270068159 for threshold 0.75
Train confusion matrix
[[61858   253]
 [ 5251 56860]]
```




```
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
[[ 1619  3840]
 [ 6041 24551]]
```



2.3.4 Applying decision_tree on features having feature importance

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

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

2.3.4.1 Merging the features

```
from scipy.sparse import hstack
X_tr = hstack((X_train_essay_tfidf,X_train_state_ohe, X_train_teacher_ohe, X_trair
#X_cr = hstack((X_cv_titles_tfidf,X_cv_essay_tfidf, X_cv_state_ohe, X_cv_teacher_c
X_te = hstack((X_test_essay_tfidf,X_test_state_ohe, X_test_teacher_ohe, X_test_gra

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

```
Final Data matrix
(124222, 10069) (124222,)
(36051, 10069) (36051,)
====================================================================================
```

2.3.4.2 Finding features having feature importance > 0

```
# How to get feature importance in GridsearchCV Decision tree
# https://stackoverflow.com/questions/48377296/get-feature-importance-from-gridsea
# https://datascience.stackexchange.com/questions/31406/tree-decisiontree-feature-

min_sample = [5,10,100,500]
feature_importance=[]
decision_tree = tree.DecisionTreeClassifier()
#create a dictionary of all values we want to test for alpha values
param_grid = {'min_samples_split':min_sample}

decision_tree_gscv = GridSearchCV(decision_tree, param_grid, cv=2, scoring='roc_au
decision_tree_gscv.fit(X_tr, y_train)
print(decision_tree_gscv.best_params_)

feature_importance = decision_tree_gscv.best_estimator_.feature_importances_
print(feature_importance)
```

```
{'min_samples_split': 10}
[0.00000000e+00 8.68785801e-05 0.00000000e+00 ... 8.45866265e-04
 1.28326355e-03 1.18885878e-03]
```

```
print(len(feature_importance))
```

```
10069
```

```
# How to extract cols in numpy array
# https://stackoverflow.com/questions/8386675/extracting-specific-columns-in-numpy

idx_OUT_columns = []
idx_IN_columns = []

size = len(feature_importance)
for i in range(size):
  if feature_importance[i]<=0:
    idx_OUT_columns.append(i)

idx_IN_columns = [i for i in range(np.shape(X_tr)[1]) if i not in idx_OUT_columns]
extractedData_train = X_tr[:,idx_IN_columns]
idx_IN_columns = [i for i in range(np.shape(X_te)[1]) if i not in idx_OUT_columns]
extractedData_test = X_te[:,idx_IN_columns]


print(extractedData_train.shape)
print(extractedData_test.shape)
```

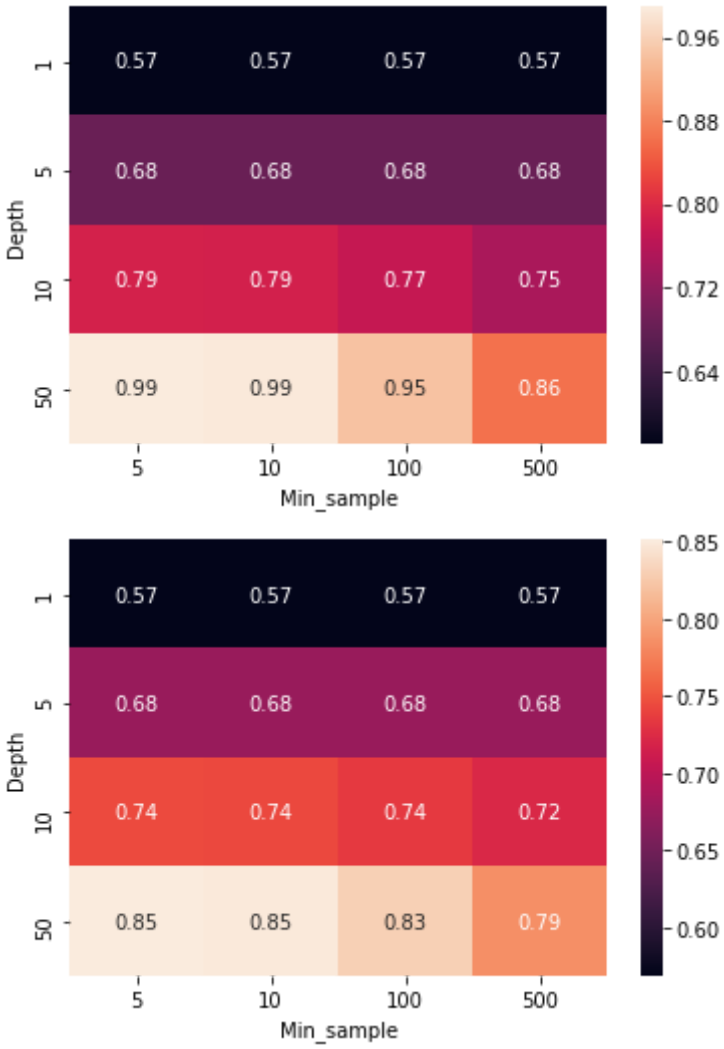
```
(124222, 3167)
(36051, 3167)
```

Double-click (or enter) to edit

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

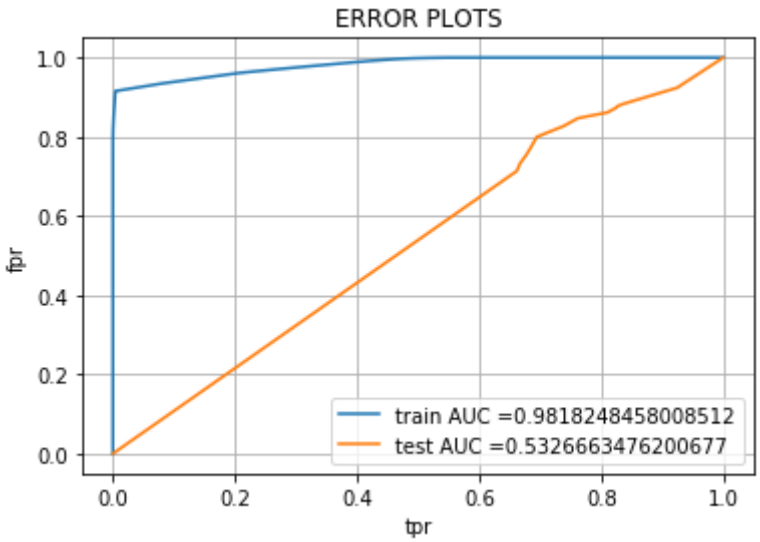
```
# Call train_decision_tree function on above data

train_decision_tree(extractedData_train,y_train)
```

```
{'max_depth': 50, 'min_samples_split': 5}
dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_tim
```



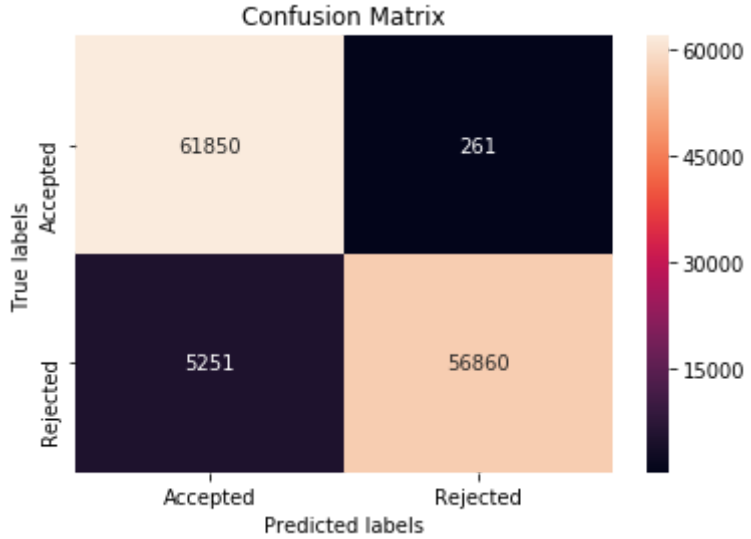
2.3.4.4 Testing the performance of the model on test data, plotting ROC Curve

```
best_depth=50
best_min_sample = 5
train_fpr,train_tpr,tr_thresholds,y_train_pred,y_test_pred=test_decision_tree(extr
```



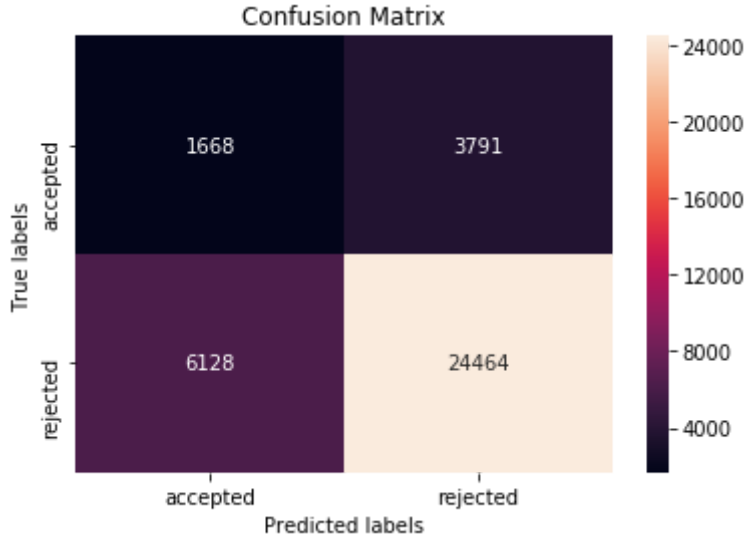

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")

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) 0.9116109145198934 for threshold 0.636
Train confusion matrix
[[61850   261]
 [ 5251 56860]]
```



```
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
[[ 1668  3791]
 [ 6128 24464]]
```



2.4 Summary

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

from prettytable import PrettyTable

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

x.field_names = ["Vectorizer", "Model", "Depth","Min_sample", "Train_AUC","Test_AU

x.add_row(["TF-IDF", "decision_tree",50,10 ,0.98,0.53])
x.add_row(["TF-IDF W2V", "decision_tree" ,50,5,0.98,0.52])
x.add_row(["TF-IDF with feature Importance", "decision_tree", 50,5,0.98,0.53])


print(x)
```

```
+--------------------------------+---------------+-------+------------+------
|           Vectorizer           |     Model     | Depth | Min_sample | Train
+--------------------------------+---------------+-------+------------+------
|             TF-IDF             | decision_tree |   50  |     10     |    0.
|           TF-IDF W2V           | decision_tree |   50  |     5      |    0.
| TF-IDF with feature Importance | decision_tree |   50  |     5      |    0.
+--------------------------------+---------------+-------+------------+------
```