Logistic_Regression_Full_Analysis

August 12, 2019

1 Logistic Regression

[ ]:

```

```

<p> DonorsChoose.org receives hundreds of thousands of project proposals each year for classroom projects in need of funding. Right now, a large number of volunteers is needed to manually screen each submission before it's approved to be posted on the DonorsChoose.org website. </p> <p> Next year, DonorsChoose.org expects to receive close to 500,000 project proposals. As a result, there are three main problems they need to solve: <ul> How to scale current manual processes and resources to screen 500,000 projects so that they can be posted as quickly and as efficiently as possible
F How to increase the consistency of project vetting across different volunteers to improve the experience for teachers How to focus volunteer time on the applications that need the most assistance </p> <p> The goal of the competition is to predict whether or not a DonorsChoose.org project proposal submitted by a teacher will be approved, using the text of project descriptions as well as additional metadata about the project, teacher, and school. DonorsChoose.org can then use this information to identify projects most likely to need further review before approval. </p>

1.0.1 Notes on the Essay Data

Prior to May 17, 2016, the prompts for the essays were as follows:
project_essay_1: "Introduce us to your classroom"
project_essay_2: "Tell us more about your students"
project_essay_3: "Describe how your students will use the materials you're requesting"
project_essay_3: "Close by sharing why your project will make a difference"

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

project_essay_1: "Describe your students: What makes your students special? Specific details about their background, your neighborhood, and your school are all helpful."

project_essay_2: "About your project: How will these materials make a difference in your students' learning and improve their school lives?"

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

1.1 Step 1: Import the necessary Libraries

we will need to import libraries that allow for data analysis and data visualization to get acclimated to the dataset. We will be using pandas, numpy, matplotlib and seaborn to conduct this.

Data Exploration libraries

[1]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
warnings.filterwarnings(action='ignore', category=UserWarning, module='gensim')
warnings.filterwarnings("ignore",'detected Windows; aliasing chunkize to␣
↪chunkize_serial')
warnings.filterwarnings("ignore", message="numpy.dtype size changed")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns

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

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

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

from tqdm import tqdm
import os

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

1.2 Step 2: Read in the dataset.

We will use the pandas .read_csv() method to read in the dataset. Then we will use the. head() method to observe the first few rows of the data, to understand the information better. In our case, the feature(column) headers tell us pretty little. This is fine because we are merely trying to gain insight via classifying new data points by referencing it's neighboring elements.

1.3 1.1 Reading Data

[2]:
```
project_data = pd.read_csv("C:\\VipinML\\Assignment␣
 ↪2\\Assignments_DonorsChoose_2018\\train_data.csv")
resource_data = pd.read_csv("C:\\VipinML\Assignment␣
 ↪2\\Assignments_DonorsChoose_2018\\resources.csv")
#Limit the data for testing purpose since processing takes few hours for full␣
 ↪set..

project_data = project_data.head(100000)
resource_data = resource_data.head (100000)

resource_data.head(1)
```

[2]:
```
        id                                   description  quantity  price
0  p233245  LC652 - Lakeshore Double-Space Mobile Drying Rack         1  149.0
```

[3]:
```
print("Number of data points in train data", project_data.shape)
print('-'*50)
print("The attributes of data :", project_data.columns.values)
```

```
Number of data points in train data (100000, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix'
'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

[4]:
```
# how to replace elements in list python: https://stackoverflow.com/a/2582163/
 ↪4084039
cols = ['Date' if x=='project_submitted_datetime' else x for x in␣
 ↪list(project_data.columns)]
#sort dataframe based on time pandas python: https://stackoverflow.com/a/
 ↪49702492/4084039
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/
↪4084039
project_data = project_data[cols]
project_data.head(1)
```

[4]:

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

      school_state                Date project_grade_category  \
55660           CA 2016-04-27 00:27:36          Grades PreK-2

      project_subject_categories            project_subject_subcategories  \
55660             Math & Science  Applied Sciences, Health & Life Science

                                    project_title  \
55660  Engineering STEAM into the Primary Classroom

                                          project_essay_1  \
55660  I have been fortunate enough to use the Fairy ...

                                          project_essay_2  \
55660  My students come from a variety of backgrounds...

                                          project_essay_3  \
55660  Each month I try to do several science or STEM...

                                          project_essay_4  \
55660  It is challenging to develop high quality scie...

                                 project_resource_summary  \
55660  My students need STEM kits to learn critical s...

       teacher_number_of_previously_posted_projects  project_is_approved
55660                                            53                    1
```

1.4 1.2 preprocessing of project_subject_categories

[5]:

```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.
↪com/a/47301924/4084039

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

```
cat_list = []
for i in catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science",␣
 ↪"Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on␣
 ↪space "Math & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to␣
 ↪replace it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with␣
 ↪''(empty) ex:"Math & Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the␣
 ↪trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

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

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

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

1.5 1.3 preprocessing of `project_subject_subcategories`

[6]:
```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.
 ↪com/a/47301924/4084039

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

sub_cat_list = []
for i in sub_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science",␣
 ↪"Warmth", "Care & Hunger"]
```

```
        if 'The' in j.split(): # this will split each of the catogory based on␣
 ↪space "Math & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to␣
 ↪replace it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with␣
 ↪''(empty) ex:"Math & Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the␣
 ↪trailing spaces
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

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

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

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

[7]:
```
teacher_cat = list(project_data['teacher_prefix'].values)
# remove special characters from list of strings python: https://stackoverflow.
 ↪com/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/
 ↪how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/
 ↪remove-all-whitespace-in-a-string-in-python
cat_list = []
for i in teacher_cat:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty)␣
 ↪ex:"Math & Science"=>"Math&Science"
    temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing␣
 ↪spaces
    temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())


project_data.drop(['teacher_prefix'], axis=1, inplace=True)
project_data['teacher_prefix'] = sub_cat_list
```

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

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

1.6 1.3 Text preprocessing

[8]:
```python
# merge two column text dataframe:
project_data["essay"] = project_data["project_essay_1"].map(str) +\
                        project_data["project_essay_2"].map(str) + \
                        project_data["project_essay_3"].map(str) + \
                        project_data["project_essay_4"].map(str)
```

[9]:
```python
project_data.head(1)
```

[9]:
```
       Unnamed: 0       id                        teacher_id school_state  \
55660        8393  p205479  2bf07ba08945e5d8b2a3f269b2b3cfe5           CA

                      Date project_grade_category  \
55660 2016-04-27 00:27:36          Grades PreK-2

                                    project_title  \
55660  Engineering STEAM into the Primary Classroom

                                         project_essay_1  \
55660  I have been fortunate enough to use the Fairy ...

                                         project_essay_2  \
55660  My students come from a variety of backgrounds...

                                         project_essay_3  \
55660  Each month I try to do several science or STEM...

                                         project_essay_4  \
55660  It is challenging to develop high quality scie...

                                project_resource_summary  \
55660  My students need STEM kits to learn critical s...

       teacher_number_of_previously_posted_projects  project_is_approved  \
55660                                            53                    1

      clean_categories                  clean_subcategories  \
55660     Math_Science  AppliedSciences Health_LifeScience
```

```
                         teacher_prefix  \
55660  AppliedSciences Health_LifeScience

                                                   essay
55660  I have been fortunate enough to use the Fairy ...
```

[10]:
```
#### 1.4.2.3 Using Pretrained Models: TFIDF weighted W2V
```

[11]:
```
# 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
```

[12]:
```
sent = decontracted(project_data['essay'].values[500])
print(sent[1:200])
print("="*100)
```

```
LTexas PE is different than most schools. Our kids get PE everyday and they love
it!From 7:45am-3:00pm 1,400 kids ranging from Kinder-8th grade come through our
gym EVERYDAY! As coaches we are always
================================================================================
====================
```

[13]:
```
# \r \n \t remove from string python: http://texthandler.com/info/
 ↪remove-line-breaks-python/
sent = sent.replace('\\r', ' ')
sent = sent.replace('\\"', ' ')
sent = sent.replace('\\n', ' ')
print(sent[1:200])
print(sent[1:200])
```

```
LTexas PE is different than most schools. Our kids get PE everyday and they love
it!From 7:45am-3:00pm 1,400 kids ranging from Kinder-8th grade come through our
```

```
gym EVERYDAY! As coaches we are always
LTexas PE is different than most schools. Our kids get PE everyday and they love
it!From 7:45am-3:00pm 1,400 kids ranging from Kinder-8th grade come through our
gym EVERYDAY! As coaches we are always
```

[14]:

```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
print(sent[1:200])
```

```
LTexas PE is different than most schools Our kids get PE everyday and they love
it From 7 45am 3 00pm 1 400 kids ranging from Kinder 8th grade come through our
gym EVERYDAY As coaches we are always h
```

[15]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you',␣
 ↪"you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he',␣
 ↪'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its',␣
 ↪'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this',␣
 ↪'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have',␣
 ↪'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',␣
 ↪'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into',␣
 ↪'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on',␣
 ↪'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how',␣
 ↪'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so',␣
 ↪'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should',␣
 ↪"should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn',␣
 ↪"didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't",␣
 ↪'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn',␣
 ↪"shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

1.6.1 1.4.3 Merging price with project_data

[16]:
```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).
 ↪reset_index()
project_data = pd.merge(project_data, price_data, on='id', how='left')
print (price_data[1:3])
project_data.head(1)
```

```
        id   price  quantity
1  p000031  357.98         2
2  p000052  114.98         2
```

[16]:
```
   Unnamed: 0       id                        teacher_id school_state  \
0        8393  p205479  2bf07ba08945e5d8b2a3f269b2b3cfe5           CA

                  Date project_grade_category  \
0 2016-04-27 00:27:36          Grades PreK-2

                                 project_title  \
0  Engineering STEAM into the Primary Classroom

                                     project_essay_1  \
0  I have been fortunate enough to use the Fairy ...

                                     project_essay_2  \
0  My students come from a variety of backgrounds...

                                     project_essay_3  \
0  Each month I try to do several science or STEM...

                                     project_essay_4  \
0  It is challenging to develop high quality scie...

                            project_resource_summary  \
0  My students need STEM kits to learn critical s...

   teacher_number_of_previously_posted_projects  project_is_approved  \
0                                            53                    1

  clean_categories                  clean_subcategories  \
0     Math_Science  AppliedSciences Health_LifeScience

                      teacher_prefix  \
0  AppliedSciences Health_LifeScience

                                               essay   price  quantity
0  I have been fortunate enough to use the Fairy ...  725.05       4.0
```

1.6.2 1.4.3.1 Merge Project Title Count with project_data

[17]:
```python
# Add count (total number of words) in Project Title in each row.

project_title_count = project_data['project_title'].str.split().str.len()
project_data['project_title_count'] = project_title_count
project_data.head(1)
```

[17]:
```
   Unnamed: 0       id                        teacher_id school_state  \
0        8393  p205479  2bf07ba08945e5d8b2a3f269b2b3cfe5           CA

                 Date project_grade_category  \
0 2016-04-27 00:27:36          Grades PreK-2

                                    project_title  \
0  Engineering STEAM into the Primary Classroom

                                     project_essay_1  \
0  I have been fortunate enough to use the Fairy ...

                                     project_essay_2  \
0  My students come from a variety of backgrounds...

                                     project_essay_3  ...  \
0  Each month I try to do several science or STEM...  ...

                            project_resource_summary  \
0  My students need STEM kits to learn critical s...

  teacher_number_of_previously_posted_projects  project_is_approved  \
0                                           53                    1

   clean_categories                 clean_subcategories  \
0      Math_Science  AppliedSciences Health_LifeScience

                        teacher_prefix  \
0  AppliedSciences Health_LifeScience

                                               essay   price  quantity  \
0  I have been fortunate enough to use the Fairy ...  725.05       4.0

   project_title_count
0                    6

[1 rows x 21 columns]
```

1.6.3 1.4.3.2 Essay count of words for each row and merge with project_data

[18]:
```
# Add count (total number of words) in essay in each row.

essay_count = project_data['essay'].str.split().str.len()
project_data['essay_count'] = essay_count
project_data.head(1)
```

[18]:
```
   Unnamed: 0       id                        teacher_id school_state  \
0        8393  p205479  2bf07ba08945e5d8b2a3f269b2b3cfe5           CA

                 Date project_grade_category  \
0 2016-04-27 00:27:36          Grades PreK-2

                                 project_title  \
0  Engineering STEAM into the Primary Classroom

                                     project_essay_1  \
0  I have been fortunate enough to use the Fairy ...

                                     project_essay_2  \
0  My students come from a variety of backgrounds...

                                     project_essay_3  ...  \
0  Each month I try to do several science or STEM...  ...

  teacher_number_of_previously_posted_projects project_is_approved  \
0                                           53                   1

   clean_categories                 clean_subcategories  \
0      Math_Science  AppliedSciences Health_LifeScience

                      teacher_prefix  \
0  AppliedSciences Health_LifeScience

                                               essay   price quantity  \
0  I have been fortunate enough to use the Fairy ...  725.05      4.0

   project_title_count  essay_count
0                    6          285

[1 rows x 22 columns]
```

[19]:
```
#Convert NaN value to mean of the column
project_data.fillna(project_data.mean(), inplace=True)
project_data.head(1)
```

[19]:
```
   Unnamed: 0       id                        teacher_id school_state  \
0        8393  p205479  2bf07ba08945e5d8b2a3f269b2b3cfe5           CA
```

```
                 Date project_grade_category  \
0 2016-04-27 00:27:36          Grades PreK-2

                                 project_title  \
0  Engineering STEAM into the Primary Classroom

                                    project_essay_1  \
0  I have been fortunate enough to use the Fairy ...

                                    project_essay_2  \
0  My students come from a variety of backgrounds...

                                    project_essay_3  ...  \
0  Each month I try to do several science or STEM...  ...

  teacher_number_of_previously_posted_projects project_is_approved  \
0                                           53                   1

   clean_categories                  clean_subcategories  \
0      Math_Science  AppliedSciences Health_LifeScience

                    teacher_prefix  \
0  AppliedSciences Health_LifeScience

                                              essay   price quantity  \
0  I have been fortunate enough to use the Fairy ...  725.05      4.0

   project_title_count  essay_count
0                    6          285

[1 rows x 22 columns]
```

1.7 Splitting data into Train and cross validation(or test): Stratified Sampling

[20]:
```python
y = project_data['project_is_approved'].values
X = project_data.drop(['project_is_approved'], axis=1)

# train test split
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=y)
```

[21]:
```python
catogories_essay = list(project_data['essay'].values)
# remove special characters from list of strings python: https://stackoverflow.
 ↪com/a/47301924/4084039
```

```
# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/
↪how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/
↪remove-all-whitespace-in-a-string-in-python
cat_essay_list = []
for i in catogories_essay:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science",␣
↪"Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on␣
↪space "Math & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to␣
↪replace it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with␣
↪''(empty) ex:"Math & Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the␣
↪trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_essay_list.append(temp.strip())

project_data['clean_essay'] = cat_essay_list

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

cat_essay_dict = dict(my_counter)
sorted_cat_essay_dict = dict(sorted(cat_essay_dict.items(), key=lambda kv:␣
↪kv[1]))
```

[22]:
```
catogories_title = list(project_data['project_title'].values)
# remove special characters from list of strings python: https://stackoverflow.
↪com/a/47301924/4084039
# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/
↪how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/
↪remove-all-whitespace-in-a-string-in-python
project_title_list = []
for i in catogories_title:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science",␣
↪"Warmth", "Care & Hunger"]
```

```
        if 'The' in j.split(): # this will split each of the catogory based on␣
↪space "Math & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to␣
↪replace it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with␣
↪''(empty) ex:"Math & Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the␣
↪trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    project_title_list.append(temp.strip())

#project_data.drop('project_title', axis=1, inplace=True)
#project_data['project_title'] = project_title_list

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

project_title_dict = dict(my_counter)
sorted_project_title_dict = dict(sorted(project_title_dict.items(), key=lambda␣
↪kv: kv[1]))
```

[23]:
```
# Combining all the above stundents
from tqdm import tqdm
X_train_preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(X_train['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    X_train_preprocessed_essays.append(sent.lower().strip())
   # print (X_train_preprocessed_essays)
```

```
100%|| 67000/67000 [00:33<00:00, 2023.24it/s]
```

[24]:
```
# Combining all the above stundents
from tqdm import tqdm
X_test_preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(X_test['essay'].values):
    sent = decontracted(sentance)
```

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

```
100%|| 33000/33000 [00:16<00:00, 2025.46it/s]
```

1.8 Step 3: Standardize (normalize) the data scale to prep for Logistic regression.

Because the distance between pairs of points plays a critical part on the classification, it is necessary to normalize the data This will generate an array of values.

1.8.1 1.4.1 Vectorizing Categorical data

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

1.8.2 Vectorization of clean_categories for X_train,X_test

[25]:
```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000,␣
 ↪vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True)
X_train_categories_one_hot = vectorizer.transform(X_train['clean_categories'].
 ↪values)
X_test_categories_one_hot = vectorizer.transform(X_test['clean_categories'].
 ↪values)
print(vectorizer.get_feature_names())
print("Shape of matrix X_train_categories_one_hot  after one hot encodig␣
 ↪",X_train_categories_one_hot.shape)
print("Shape of matrix X_test_categories_one_hot after one hot encodig␣
 ↪",X_test_categories_one_hot.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning',
'SpecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix X_train_categories_one_hot  after one hot encodig  (67000, 9)
Shape of matrix X_test_categories_one_hot after one hot encodig  (33000, 9)
```

[26]:
```
### Vectorization of project_grade_category for X_train,X_test
```

[27]:
```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
```

```
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000,␣
 ↪vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True)
X_train_project_grade_category_one_hot = vectorizer.
 ↪transform(X_train['project_grade_category'].values)
X_test_project_grade_category_one_hot = vectorizer.
 ↪transform(X_test['project_grade_category'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix X_train_project_grade_category_one_hot  after one hot␣
 ↪encodig ",X_train_project_grade_category_one_hot.shape)
print("Shape of matrix X_test_project_grade_category_one_hot after one hot␣
 ↪encodig ",X_test_project_grade_category_one_hot.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning',
'SpecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix X_train_project_grade_category_one_hot  after one hot encodig
(67000, 9)
Shape of matrix X_test_project_grade_category_one_hot after one hot encodig
(33000, 9)
```

[28]:
```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000,␣
 ↪vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True)
X_train_school_state_one_hot = vectorizer.transform(X_train['school_state'].
 ↪values)
X_test_school_state_one_hot = vectorizer.transform(X_test['school_state'].
 ↪values)
print(vectorizer.get_feature_names())
print("Shape of matrix X_train_school_state_one_hot  after one hot encodig␣
 ↪",X_train_school_state_one_hot.shape)
print("Shape of matrix X_test_school_state_one_hot after one hot encodig␣
 ↪",X_test_school_state_one_hot.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning',
'SpecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix X_train_school_state_one_hot  after one hot encodig  (67000, 9)
Shape of matrix X_test_school_state_one_hot after one hot encodig  (33000, 9)
```

1.8.3 Vectorization of clean_subcategories for X_train,X_test

[29]:
```
# we use count vectorizer to convert the values into one
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4),␣
 ↪max_features=5000,vocabulary=list(sorted_sub_cat_dict.keys()),␣
 ↪lowercase=False, binary=True)
X_train_sub_categories_one_hot = vectorizer.
 ↪transform(X_train['clean_subcategories'].values)
```

```
X_test_sub_categories_one_hot = vectorizer.
↪transform(X_test['clean_subcategories'].values)

print(vectorizer.get_feature_names())
print("Shape of matrix X_train_sub_categories_one_hot  after one hot encodig␣
↪",X_train_sub_categories_one_hot.shape)
print("Shape of matrix X_test_sub_categories_one_hot after␣
↪oneX_test_sub_categories_one_hot  hot encodig␣
↪",X_test_sub_categories_one_hot.shape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement',
'Civics_Government', 'Extracurricular', 'ForeignLanguages',
'NutritionEducation', 'Warmth', 'Care_Hunger', 'SocialSciences',
'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'History_Geography', 'Music', 'Health_LifeScience',
'EarlyDevelopment', 'ESL', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts',
'Health_Wellness', 'AppliedSciences', 'SpecialNeeds', 'Literature_Writing',
'Mathematics', 'Literacy']
Shape of matrix X_train_sub_categories_one_hot  after one hot encodig  (67000,
30)
Shape of matrix X_test_sub_categories_one_hot after
oneX_test_sub_categories_one_hot  hot encodig  (33000, 30)
```

[30]:
```
# you can do the similar thing with state, teacher_prefix and␣
↪project_grade_category also
```

1.9 TFIDF of preprocessed_essays for X_train,X_test

[31]:
```
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
X_train_dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.
↪idf_)))
X_train_tfidf_words = set(tfidf_model.get_feature_names())
print (len(X_train_tfidf_words))
```

```
46504
```

[32]:
```
## X_train TFIDF of preprocessed_essays for X_test
```

[33]:
```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_test_preprocessed_essays)
# we are converting a dictionary with word as a key, and the idf as a value
X_test_dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.
↪idf_)))
X_test_tfidf_words = set(tfidf_model.get_feature_names())
```

```
print (len(X_test_tfidf_words))
```

35500

1.9.1 1.4.2 Vectorizing Text data

[34]:
```
# stronging variables into pickle files python: http://www.jessicayung.com/
↪how-to-use-pickle-to-save-and-load-variables-in-python/
# make sure you have the glove_vectors file
with open('C:\\VipinML\\InputData\\glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

1.9.2 Vectorization of preprocessed_essays for X_train,X_test

[35]:
```
# average Word2Vec
# compute average word2vec for each review.
X_train_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored␣
↪in this list
for sentence in tqdm(X_train_preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_train_avg_w2v_vectors.append(vector)

print(len(X_train_avg_w2v_vectors))
print(len(X_train_avg_w2v_vectors[0]))
```

```
100%|| 67000/67000 [00:18<00:00, 3681.28it/s]
```

```
67000
300
```

[36]:
```
# average Word2Vec
# compute average word2vec for each review.
X_test_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored␣
↪in this list
for sentence in tqdm(X_test_preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
```

```
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_test_avg_w2v_vectors.append(vector)

print(len(X_test_avg_w2v_vectors))
print(len(X_test_avg_w2v_vectors[0]))
```

```
100%|| 33000/33000 [00:08<00:00, 3714.83it/s]

33000
300
```

[37]:
```
## TFIDF-W2W Vecorization
```

[38]:
```
# average Word2Vec
# compute average word2vec for each review.
X_train_tfidf_w2v_vectors_pessays = []; # the avg-w2v for each sentence/review␣
 ↪is stored in this list
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 X_train_tfidf_words):
           # print (word)
            if word == 'affluent':
                #print  (model[word])
                print (sentence.count(word))
                print (len(sentence.split()))
                print (X_train_dictionary[word])

            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf␣
 ↪value((sentence.count(word)/len(sentence.split())))
            tf_idf = X_train_dictionary[word]*(sentence.count(word)/
 ↪len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_train_tfidf_w2v_vectors_pessays.append(vector)

print(len(X_train_tfidf_w2v_vectors_pessays))
print(len(X_train_tfidf_w2v_vectors_pessays[0]))
```

1
133
5.920100334159983

1
103
5.920100334159983

1
149
5.920100334159983

2
172
5.920100334159983
2
172
5.920100334159983

1
306
5.920100334159983

1
173
5.920100334159983

2
171
5.920100334159983
2
171
5.920100334159983

1
217
5.920100334159983

```
1
120
5.920100334159983
1
120
5.920100334159983


1
252
5.920100334159983


1
139
5.920100334159983


1
145
5.920100334159983


1
178
5.920100334159983


1
160
5.920100334159983


1
120
5.920100334159983
1
160
5.920100334159983
```

1
117
5.920100334159983

1
138
5.920100334159983

1
171
5.920100334159983

1
155
5.920100334159983

1
241
5.920100334159983

1
232
5.920100334159983

1
133
5.920100334159983

1
151
5.920100334159983
1
122
5.920100334159983

```
1
140
5.920100334159983
1
211
5.920100334159983


1
183
5.920100334159983
1
174
5.920100334159983


1
121
5.920100334159983
1
213
5.920100334159983


1
179
5.920100334159983


1
185
5.920100334159983


2
125
5.920100334159983
2
125
5.920100334159983


1
242
5.920100334159983
```

1
146
5.920100334159983

1
150
5.920100334159983

1
235
5.920100334159983

1
109
5.920100334159983
1
241
5.920100334159983

1
108
5.920100334159983

1
115
5.920100334159983

1
147
5.920100334159983

1
116
5.920100334159983

1
204
5.920100334159983

1
136
5.920100334159983

1
165
5.920100334159983

1
166
5.920100334159983

1
98
5.920100334159983

1
115
5.920100334159983

1
329
5.920100334159983

1
187
5.920100334159983

```
1
175
5.920100334159983


1
108
5.920100334159983


1
244
5.920100334159983


1
159
5.920100334159983
1
110
5.920100334159983


1
110
5.920100334159983


1
136
5.920100334159983
1
168
5.920100334159983


1
138
5.920100334159983


1
118
5.920100334159983
```

1
130
5.920100334159983

1
132
5.920100334159983

1
107
5.920100334159983

1
143
5.920100334159983

1
186
5.920100334159983

1
117
5.920100334159983

1
249
5.920100334159983
1
104
5.920100334159983

1
127
5.920100334159983

1
194
5.920100334159983

1
125
5.920100334159983

1
201
5.920100334159983

1
268
5.920100334159983

1
147
5.920100334159983

1
111
5.920100334159983

1
181
5.920100334159983

1
209
5.920100334159983
1
112
5.920100334159983

1
114
5.920100334159983

1
254
5.920100334159983

1
244
5.920100334159983

1
111
5.920100334159983

1
160
5.920100334159983

1
109
5.920100334159983

1
192
5.920100334159983

1
122
5.920100334159983

1
290
5.920100334159983

1
169
5.920100334159983
1
267
5.920100334159983

1
134
5.920100334159983

1
137
5.920100334159983

1
206
5.920100334159983

1
118
5.920100334159983

1
109
5.920100334159983

1
137
5.920100334159983

1
196
5.920100334159983

1
259
5.920100334159983

1
149
5.920100334159983

1
124
5.920100334159983
1
218
5.920100334159983

1
210
5.920100334159983

1
122
5.920100334159983
1
143
5.920100334159983

1
160
5.920100334159983

```
3
234
5.920100334159983
3
234
5.920100334159983
3
234
5.920100334159983
1
158
5.920100334159983


1
119
5.920100334159983


1
124
5.920100334159983
1
240
5.920100334159983
1
162
5.920100334159983


2
151
5.920100334159983
2
151
5.920100334159983


1
165
5.920100334159983


1
112
5.920100334159983
```

1
223
5.920100334159983

1
156
5.920100334159983

1
235
5.920100334159983
1
210
5.920100334159983

1
122
5.920100334159983

2
139
5.920100334159983
2
139
5.920100334159983

1
222
5.920100334159983

1
182
5.920100334159983

```
1
153
5.920100334159983


1
191
5.920100334159983


1
200
5.920100334159983


1
136
5.920100334159983
1
132
5.920100334159983


1
124
5.920100334159983
1
142
5.920100334159983


2
112
5.920100334159983
2
112
5.920100334159983


1
140
5.920100334159983
1
219
5.920100334159983
```

1
118
5.920100334159983
1
177
5.920100334159983

1
133
5.920100334159983

1
188
5.920100334159983

1
127
5.920100334159983

1
216
5.920100334159983
1
263
5.920100334159983

1
148
5.920100334159983
1
118
5.920100334159983

1
140
5.920100334159983

1
173
5.920100334159983

1
119
5.920100334159983

1
157
5.920100334159983

1
198
5.920100334159983

1
116
5.920100334159983

1
111
5.920100334159983

1
156
5.920100334159983

1
107
5.920100334159983

1
132
5.920100334159983

1
187
5.920100334159983

1
129
5.920100334159983

3
129
5.920100334159983
3
129
5.920100334159983
3
129
5.920100334159983

1
104
5.920100334159983

1
165
5.920100334159983

1
108
5.920100334159983

1
206
5.920100334159983

```
1
112
5.920100334159983
1
221
5.920100334159983


1
207
5.920100334159983


1
220
5.920100334159983
1
147
5.920100334159983


1
192
5.920100334159983


1
225
5.920100334159983


2
168
5.920100334159983
2
168
5.920100334159983


1
270
5.920100334159983
```

1
119
5.920100334159983

1
125
5.920100334159983

1
118
5.920100334159983

1
128
5.920100334159983
1
112
5.920100334159983

1
264
5.920100334159983

1
180
5.920100334159983
1
176
5.920100334159983

1
190
5.920100334159983

1
111
5.920100334159983

1
135
5.920100334159983

1
171
5.920100334159983

1
110
5.920100334159983

1
119
5.920100334159983

1
118
5.920100334159983
1
215
5.920100334159983

1
270
5.920100334159983

1
185
5.920100334159983

1
162
5.920100334159983

1
115
5.920100334159983

1
204
5.920100334159983
1
149
5.920100334159983

1
195
5.920100334159983

1
115
5.920100334159983

1
118
5.920100334159983

1
146
5.920100334159983
1
171
5.920100334159983

1
280
5.920100334159983

1
182
5.920100334159983
1
157
5.920100334159983

1
229
5.920100334159983

1
196
5.920100334159983

1
159
5.920100334159983

1
104
5.920100334159983

1
159
5.920100334159983

1
117
5.920100334159983

1
136
5.920100334159983

1
110
5.920100334159983

1
149
5.920100334159983

1
217
5.920100334159983

1
129
5.920100334159983

1
148
5.920100334159983

1
136
5.920100334159983
2
148
5.920100334159983
2
148
5.920100334159983

1
155
5.920100334159983

```
1
283
5.920100334159983


1
149
5.920100334159983


1
192
5.920100334159983


1
203
5.920100334159983
1
111
5.920100334159983


1
225
5.920100334159983


1
256
5.920100334159983


1
191
5.920100334159983


1
151
5.920100334159983
```

```
1
130
5.920100334159983
1
161
5.920100334159983
1
151
5.920100334159983


1
124
5.920100334159983
1
133
5.920100334159983


1
240
5.920100334159983


1
201
5.920100334159983
1
189
5.920100334159983


1
267
5.920100334159983


1
123
5.920100334159983


1
123
5.920100334159983
```

1
206
5.920100334159983

1
167
5.920100334159983

1
175
5.920100334159983

1
260
5.920100334159983

1
157
5.920100334159983

1
139
5.920100334159983
1
131
5.920100334159983

1
171
5.920100334159983

1
214
5.920100334159983

1
116
5.920100334159983

1
154
5.920100334159983

1
110
5.920100334159983
1
134
5.920100334159983

1
178
5.920100334159983

1
180
5.920100334159983

1
134
5.920100334159983

1
108
5.920100334159983

1
117
5.920100334159983

1
119
5.920100334159983

1
113
5.920100334159983

1
123
5.920100334159983

2
110
5.920100334159983
2
110
5.920100334159983

1
150
5.920100334159983

1
229
5.920100334159983
1
177
5.920100334159983

1
204
5.920100334159983

```
1
237
5.920100334159983
1
170
5.920100334159983


1
132
5.920100334159983


1
171
5.920100334159983
1
255
5.920100334159983


1
169
5.920100334159983
1
228
5.920100334159983


1
154
5.920100334159983


1
164
5.920100334159983


1
167
5.920100334159983
```

1
187
5.920100334159983

1
104
5.920100334159983

1
190
5.920100334159983

1
149
5.920100334159983

1
117
5.920100334159983

1
170
5.920100334159983
1
136
5.920100334159983
1
193
5.920100334159983
1
157
5.920100334159983

1
253
5.920100334159983

```
1
162
5.920100334159983


1
175
5.920100334159983
1
120
5.920100334159983
1
165
5.920100334159983


1
140
5.920100334159983


1
197
5.920100334159983


1
106
5.920100334159983


1
144
5.920100334159983


1
184
5.920100334159983
```

```
1
200
5.920100334159983
1
118
5.920100334159983


1
178
5.920100334159983


1
212
5.920100334159983
1
170
5.920100334159983


1
146
5.920100334159983


1
154
5.920100334159983


1
158
5.920100334159983


1
151
5.920100334159983
1
138
5.920100334159983
1
113
5.920100334159983
```

```
1
149
5.920100334159983



1
168
5.920100334159983



1
121
5.920100334159983



1
242
5.920100334159983
1
156
5.920100334159983



1
171
5.920100334159983



1
117
5.920100334159983



1
135
5.920100334159983



1
299
5.920100334159983
```

```
1
214
5.920100334159983


1
134
5.920100334159983


1
148
5.920100334159983


2
204
5.920100334159983
2
204
5.920100334159983


1
129
5.920100334159983


1
145
5.920100334159983


1
245
5.920100334159983


1
167
5.920100334159983
```

1
193
5.920100334159983
1
189
5.920100334159983

1
118
5.920100334159983

1
193
5.920100334159983

1
243
5.920100334159983

1
192
5.920100334159983

1
103
5.920100334159983
1
170
5.920100334159983

1
201
5.920100334159983

1
232
5.920100334159983

1
187
5.920100334159983

1
122
5.920100334159983

1
170
5.920100334159983
2
198
5.920100334159983
2
198
5.920100334159983

1
156
5.920100334159983

1
150
5.920100334159983

1
187
5.920100334159983
1
136
5.920100334159983
1
205
5.920100334159983

1
206
5.920100334159983

1
119
5.920100334159983

1
180
5.920100334159983

1
330
5.920100334159983

1
113
5.920100334159983

1
240
5.920100334159983

1
191
5.920100334159983
1
136
5.920100334159983

2
159
5.920100334159983
2
159
5.920100334159983

```
1
129
5.920100334159983



1
129
5.920100334159983
1
166
5.920100334159983



1
202
5.920100334159983



1
139
5.920100334159983



1
143
5.920100334159983
1
152
5.920100334159983



1
127
5.920100334159983



1
127
5.920100334159983
```

```
1
207
5.920100334159983
1
138
5.920100334159983
1
231
5.920100334159983


1
124
5.920100334159983
1
139
5.920100334159983


1
172
5.920100334159983


1
117
5.920100334159983


1
161
5.920100334159983


1
111
5.920100334159983


1
256
5.920100334159983
```

```
1
149
5.920100334159983

1
178
5.920100334159983

1
140
5.920100334159983

1
168
5.920100334159983

1
128
5.920100334159983

1
213
5.920100334159983

1
123
5.920100334159983

1
169
5.920100334159983

1
245
5.920100334159983
```

1
139
5.920100334159983

1
186
5.920100334159983
2
157
5.920100334159983
2
157
5.920100334159983

1
151
5.920100334159983

1
155
5.920100334159983

2
213
5.920100334159983
2
213
5.920100334159983

1
129
5.920100334159983

1
166
5.920100334159983

1
250
5.920100334159983

1
149
5.920100334159983

1
109
5.920100334159983

1
143
5.920100334159983

1
125
5.920100334159983

1
179
5.920100334159983
1
117
5.920100334159983

1
131
5.920100334159983
1
112
5.920100334159983

```
1
126
5.920100334159983


1
147
5.920100334159983


1
260
5.920100334159983


1
129
5.920100334159983


1
157
5.920100334159983


1
170
5.920100334159983
1
158
5.920100334159983
1
165
5.920100334159983


1
204
5.920100334159983
```

```
1
140
5.920100334159983
1
139
5.920100334159983


1
271
5.920100334159983
2
120
5.920100334159983
2
120
5.920100334159983


1
163
5.920100334159983


1
122
5.920100334159983
1
295
5.920100334159983


1
198
5.920100334159983
1
247
5.920100334159983


1
140
5.920100334159983
```

```
1
236
5.920100334159983


1
142
5.920100334159983


1
110
5.920100334159983


1
274
5.920100334159983
1
103
5.920100334159983


1
132
5.920100334159983
1
204
5.920100334159983


1
171
5.920100334159983


1
231
5.920100334159983


1
179
5.920100334159983
```

```
1
196
5.920100334159983
1
127
5.920100334159983


1
208
5.920100334159983
1
180
5.920100334159983
1
177
5.920100334159983


1
179
5.920100334159983


1
218
5.920100334159983


1
141
5.920100334159983


1
179
5.920100334159983


1
186
5.920100334159983
```

1
155
5.920100334159983
1
228
5.920100334159983

1
197
5.920100334159983

1
138
5.920100334159983

1
190
5.920100334159983

1
177
5.920100334159983
1
180
5.920100334159983

2
141
5.920100334159983
2
141
5.920100334159983

1
202
5.920100334159983

2
148
5.920100334159983
2
148
5.920100334159983

1
135
5.920100334159983

1
195
5.920100334159983

1
109
5.920100334159983

1
187
5.920100334159983

1
182
5.920100334159983

1
138
5.920100334159983

1
131
5.920100334159983

```
1
114
5.920100334159983


1
159
5.920100334159983
1
246
5.920100334159983


1
203
5.920100334159983


1
206
5.920100334159983


2
253
5.920100334159983
2
253
5.920100334159983


1
134
5.920100334159983


1
110
5.920100334159983
```

```
2
200
5.920100334159983
2
200
5.920100334159983
1
155
5.920100334159983


1
158
5.920100334159983


1
136
5.920100334159983


1
128
5.920100334159983


1
161
5.920100334159983


1
158
5.920100334159983


1
107
5.920100334159983
```

1
115
5.920100334159983
1
123
5.920100334159983

1
123
5.920100334159983

1
235
5.920100334159983

1
224
5.920100334159983

1
132
5.920100334159983
1
177
5.920100334159983

1
128
5.920100334159983

1
225
5.920100334159983

```
2
122
5.920100334159983
2
122
5.920100334159983


1
132
5.920100334159983


1
241
5.920100334159983


1
151
5.920100334159983


1
193
5.920100334159983


1
121
5.920100334159983


1
176
5.920100334159983


1
160
5.920100334159983
```

```
1
140
5.920100334159983


1
306
5.920100334159983


1
146
5.920100334159983


1
213
5.920100334159983
2
187
5.920100334159983
2
187
5.920100334159983


1
204
5.920100334159983


1
139
5.920100334159983


1
122
5.920100334159983


1
125
5.920100334159983

100%|| 67000/67000 [02:15<00:00, 496.11it/s]

67000
300
```

[39]:
```
# average Word2Vec
# compute average word2vec for each review.
X_test_tfidf_w2v_vectors_pessays = []; # the avg-w2v for each sentence/review␣
 ↪is stored in this list
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 X_test_tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf␣
 ↪value((sentence.count(word)/len(sentence.split())))
            tf_idf = X_test_dictionary[word]*(sentence.count(word)/len(sentence.
 ↪split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_test_tfidf_w2v_vectors_pessays.append(vector)

print(len(X_test_tfidf_w2v_vectors_pessays))
print(len(X_test_tfidf_w2v_vectors_pessays[0]))
```

```
100%|| 33000/33000 [01:03<00:00, 519.37it/s]
```

```
33000
300
```

[40]:
```
# average Word2Vec
# compute average word2vec for each review.
X_test_tfidf_w2v_vectors_ptitle = []; # the avg-w2v for each sentence/review is␣
 ↪stored in this list
for sentence in tqdm(X_test['project_title']): # 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 X_test_tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf␣
 ↪value((sentence.count(word)/len(sentence.split())))
            tf_idf = X_test_dictionary[word]*(sentence.count(word)/len(sentence.
 ↪split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_test_tfidf_w2v_vectors_ptitle.append(vector)
```

```
print(len(X_test_tfidf_w2v_vectors_ptitle))
print(len(X_test_tfidf_w2v_vectors_ptitle[0]))
```

```
100%|| 33000/33000 [00:00<00:00, 94954.43it/s]

33000
300
```

[41]:
```
# average Word2Vec
# compute average word2vec for each review.

X_train_tfidf_w2v_vectors_ptitle = []; # the avg-w2v for each sentence/review␣
↪is stored in this list
for sentence in tqdm(X_train['project_title']): # 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 X_train_tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf␣
↪value((sentence.count(word)/len(sentence.split())))
            tf_idf = X_train_dictionary[word]*(sentence.count(word)/
↪len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_train_tfidf_w2v_vectors_ptitle.append(vector)

print(len(X_train_tfidf_w2v_vectors_ptitle))
print(len(X_train_tfidf_w2v_vectors_ptitle[0]))
```

```
100%|| 67000/67000 [00:00<00:00, 91325.07it/s]

67000
300
```

1.9.3 Vectorization of teacher_prefix for X_train,X_test, X_cv

[42]:
```
# we use count vectorizer to convert the values into one hot encoded features
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4),␣
↪vocabulary=list(sorted_teacher_dict.keys()),max_features=5000,␣
↪lowercase=False, binary=True)
X_train_teacher_prefix_data = X_train['teacher_prefix']
```

```
X_train_teacher_prefix_data.fillna("Mrs.", inplace = True)

teacher_prefix_notnull = X_train_teacher_prefix_data[pd.
 ↪notnull(X_train_teacher_prefix_data)]

vectorizer.fit(teacher_prefix_notnull.values)

#print(vectorizer.get_feature_names())

print(teacher_prefix_notnull.values)

X_train_teacher_prefix_one_hot = vectorizer.transform(teacher_prefix_notnull.
 ↪values)
print("Shape of matrix after one hot encodig ",X_train_teacher_prefix_one_hot.
 ↪shape)
```

```
['Literature_Writing Mathematics' 'ESL' 'History_Geography PerformingArts'
 ... 'EnvironmentalScience Mathematics' 'Health_Wellness' 'SpecialNeeds']
Shape of matrix after one hot encodig  (67000, 30)
```

[43]:
```
# we use count vectorizer to convert the values into one hot encoded features
from sklearn.feature_extraction.text import CountVectorizer
vectorizer1 = CountVectorizer(min_df=10,ngram_range=(1,4),␣
 ↪max_features=5000,vocabulary=list(sorted_teacher_dict.keys()),␣
 ↪lowercase=False, binary=True)
X_test_teacher_prefix_data = X_test['teacher_prefix']
X_test_teacher_prefix_data.fillna("Mrs.", inplace = True)
teacher_prefix_notnull = X_test_teacher_prefix_data[pd.
 ↪notnull(X_test_teacher_prefix_data)]
vectorizer.fit(teacher_prefix_notnull.values)
X_test_teacher_prefix_one_hot = vectorizer1.transform(teacher_prefix_notnull.
 ↪values)
print("Shape of matrix after one hot encodig ",X_test_teacher_prefix_one_hot.
 ↪shape)
```

```
Shape of matrix after one hot encodig  (33000, 30)
```

1.9.4 Vectorization of price for X_train,X_test

[44]:
```
X_train.head(1)
X_test.head(1)
```

[44]:
```
       Unnamed: 0       id                        teacher_id school_state  \
90823       87599  p110823  9e23d6ad6df4f78f215afc6c5fe02067           AZ

                      Date project_grade_category  \
90823 2017-03-17 22:39:15             Grades 6-8
```

```
                                   project_title  \
90823  Charging Into the Past and Into the Future

                                        project_essay_1  \
90823  \"Charging Forward into the Past\", at our mid...

                                        project_essay_2 project_essay_3  ...  \
90823  My students use Chromebooks weekly in class to...             NaN  ...

                                project_resource_summary  \
90823  My students need a Charging cart to charge our...

      teacher_number_of_previously_posted_projects  clean_categories  \
90823                                            13    History_Civics

      clean_subcategories     teacher_prefix  \
90823   FinancialLiteracy  FinancialLiteracy

                                                  essay       price  \
90823  \"Charging Forward into the Past\", at our mid...  305.563174

        quantity  project_title_count  essay_count
90823  18.782149                    8          320

[1 rows x 21 columns]
```

[45]:

```python
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_test_price_norm = normalizer.transform(X_test['price'].values.reshape(-1,1))

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

```
After vectorizations
(67000, 1) (67000,)
```

```
(33000, 1) (33000,)
================================================================================
====================
```

1.9.5 Normalization of Project Title Count.

[46]:
```python
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

#print (X_train['project_title_count'])

X_train_project_title_count_norm = normalizer.
 ↪transform(X_train['project_title_count'].values.reshape(-1,1))
X_test_project_title_count_norm = normalizer.
 ↪transform(X_test['project_title_count'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_project_title_count_norm.shape, y_train.shape)
print(X_test_project_title_count_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(67000, 1) (67000,)
(33000, 1) (33000,)
================================================================================
====================
```

1.9.6 Normalization of essay count words.

[47]:
```python
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

#print (X_train['project_title_count'])

X_train_essay_count_norm = normalizer.transform(X_train['essay_count'].values.
 ↪reshape(-1,1))
X_test_essay_count_norm = normalizer.transform(X_test['essay_count'].values.
 ↪reshape(-1,1))

print("After vectorizations")
print(X_train_essay_count_norm.shape, y_train.shape)
print(X_test_essay_count_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(67000, 1) (67000,)
(33000, 1) (33000,)
```

```
================================================================================
====================
```

[48]:
```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

#print (X_train['project_title_count'])

X_train_quantity_norm = normalizer.transform(X_train['quantity'].values.
 ↪reshape(-1,1))
X_test_quantity_norm = normalizer.transform(X_test['quantity'].values.
 ↪reshape(-1,1))

print("After vectorizations")
print(X_train_quantity_norm.shape, y_train.shape)
print(X_test_quantity_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(67000, 1) (67000,)
(33000, 1) (33000,)
================================================================================
====================
```

[49]:
```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

X_train_teacher_number_of_previously_posted_projects_norm = normalizer.
 ↪transform(X_train['teacher_number_of_previously_posted_projects'].values.
 ↪reshape(-1,1))
X_test_teacher_number_of_previously_posted_projects_norm = normalizer.
 ↪transform(X_test['teacher_number_of_previously_posted_projects'].values.
 ↪reshape(-1,1))

print("After vectorizations")
print(X_train_teacher_number_of_previously_posted_projects_norm.shape, y_train.
 ↪shape)
print(X_test_teacher_number_of_previously_posted_projects_norm.shape, y_test.
 ↪shape)
print("="*100)
```

```
After vectorizations
(67000, 1) (67000,)
(33000, 1) (33000,)
================================================================================
====================
```

1.10 Bag of words of preprocessed_essays for X_train,X_test

[50]:
```
# We are considering only the words which appeared in at least 10␣
↪documents(rows or projects).
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), ␣
↪max_features=5000,vocabulary=list(sorted_cat_essay_dict.keys()))
X_train_text_bow = vectorizer.fit_transform(X_train_preprocessed_essays)
X_test_text_bow = vectorizer.transform(X_test_preprocessed_essays)

print("Shape of matrix X_train_text_bow after one hot encodig␣
↪",X_train_text_bow.shape)
print("Shape of matrix X_test_text_bow after one hot encodig ",X_test_text_bow.
↪shape)
```

```
Shape of matrix X_train_text_bow after one hot encodig  (67000, 749042)
Shape of matrix X_test_text_bow after one hot encodig  (33000, 749042)
```

1.11 Bag of words of project_title for X_train,X_test

[51]:
```
# PROJECT_TITLE BOW
# We are considering only the words which appeared in at least 10␣
↪documents(rows or projects).
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4),␣
↪max_features=5000,vocabulary=list(sorted_project_title_dict.keys()))
X_train_project_title_bow = vectorizer.fit_transform(X_train['project_title'])
X_test_project_title_bow = vectorizer.transform(X_test['project_title'])

print("Shape of matrix X_train_project_title_bow after one hot encodig␣
↪",X_train_project_title_bow .shape)
print("Shape of matrix X_test_project_title_bow after one hot encodig␣
↪",X_test_project_title_bow .shape)
```

```
Shape of matrix X_train_project_title_bow after one hot encodig  (67000, 42224)
Shape of matrix X_test_project_title_bow after one hot encodig  (33000, 42224)
```

1.12 TFIDF of preprocessed_essays for X_train,X_test

[52]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
X_train_text_tfidf = vectorizer.fit_transform(X_train_preprocessed_essays)
X_test_text_tfidf = vectorizer.transform(X_test_preprocessed_essays)

print("Shape of matrix X_train_text_tfidf after one hot encodig␣
↪",X_train_text_tfidf.shape)
print("Shape of matrix X_test_text_tfidf after one hot encodig␣
↪",X_test_text_tfidf.shape)
```

```
Shape of matrix X_train_text_tfidf after one hot encodig  (67000, 46504)
Shape of matrix X_test_text_tfidf after one hot encodig  (33000, 46504)
```

1.13 TFIDF of Project Title for X_train,X_test

[53]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
X_train_project_title_tfidf = vectorizer.
 ↪fit_transform((X_train['project_title']))
X_test_project_title_tfidf = vectorizer.transform((X_test['project_title']))

print("Shape of matrix  X_train_project_title_tfidf after one hot encodig␣
 ↪",X_train_project_title_tfidf.shape)
print("Shape of matrix  X_test_project_title_tfidf after one hot encodig␣
 ↪",X_test_project_title_tfidf.shape)
```

```
Shape of matrix  X_train_project_title_tfidf after one hot encodig  (67000,
2523)
Shape of matrix  X_test_project_title_tfidf after one hot encodig  (33000, 2523)
```

1.13.1 TFIDF AVG W2V for Project Title for X_train,X_test

[54]:
```
# average Word2Vec
# compute average word2vec for each review.
X_train_project_title_avg_w2v_vectors = []; # the avg-w2v for each sentence/
 ↪review is stored in this list
for sentence in tqdm(X_train['project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_train_project_title_avg_w2v_vectors.append(vector)

print(len(X_train_project_title_avg_w2v_vectors))
print(len(X_train_project_title_avg_w2v_vectors[0]))
```

```
100%|| 67000/67000 [00:00<00:00, 139873.88it/s]
```

```
67000
300
```

[55]:
```
# average Word2Vec
# compute average word2vec for each review.
```

```
X_test_project_title_avg_w2v_vectors = []; # the avg-w2v for each sentence/
↪review is stored in this list
for sentence in tqdm(X_test['project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_test_project_title_avg_w2v_vectors.append(vector)

print(len(X_test_project_title_avg_w2v_vectors))
print(len(X_test_project_title_avg_w2v_vectors[0]))
```

```
100%|| 33000/33000 [00:00<00:00, 138731.67it/s]
```

```
33000
300
```

[56]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack


X_tr =␣
↪hstack((X_train_price_norm,X_train_sub_categories_one_hot,X_train_teacher_prefix_one_hot)).
↪tocsr()
X_te =␣
↪hstack((X_test_price_norm,X_test_sub_categories_one_hot,X_test_teacher_prefix_one_hot)).
↪tocsr()

#print (X_train_price_norm)
X_tr_bow =␣
↪hstack((X_train_price_norm,X_train_sub_categories_one_hot,X_train_teacher_prefix_one_hot,X_t
↪tocsr()
X_te_bow =␣
↪hstack((X_test_price_norm,X_test_sub_categories_one_hot,X_test_teacher_prefix_one_hot,X_test
↪tocsr()

X_tr_tfidf =␣
↪hstack((X_train_sub_categories_one_hot,X_train_teacher_prefix_one_hot,X_train_price_norm,X_t
↪tocsr()
X_te_tfidf =␣
↪hstack((X_test_sub_categories_one_hot,X_test_teacher_prefix_one_hot,X_test_price_norm,X_test
↪tocsr()
```

```
X_tr_tfidf_w2v =␣
↪hstack((X_train_sub_categories_one_hot,X_train_teacher_prefix_one_hot,X_train_price_norm,X_
↪tocsr()
X_te_tfidf_w2v =␣
↪hstack((X_test_sub_categories_one_hot,X_test_teacher_prefix_one_hot,X_test_price_norm,X_tes
↪tocsr()


X_tr_avg_w2v =␣
↪hstack((X_train_sub_categories_one_hot,X_train_teacher_prefix_one_hot,X_train_price_norm,X_
↪tocsr()
X_te_avg_w2v =␣
↪hstack((X_test_sub_categories_one_hot,X_test_teacher_prefix_one_hot,X_test_price_norm,X_tes
↪tocsr()

#set 5

X_te_set5␣
↪=hstack((X_test_school_state_one_hot,X_test_categories_one_hot,X_test_sub_categories_one_ho
                  ␣
↪X_test_teacher_prefix_one_hot,X_test_quantity_norm,X_test_price_norm,X_test_project_grade_c
                  ␣
↪X_test_teacher_number_of_previously_posted_projects_norm,X_test_price_norm,\
                  X_test_project_title_count_norm,X_test_essay_count_norm))

X_tr_set5 =hstack((X_train_school_state_one_hot,X_train_categories_one_hot,␣
↪X_train_sub_categories_one_hot,\
                  ␣
↪X_train_teacher_prefix_one_hot,X_train_quantity_norm,X_train_price_norm,X_train_project_gra
                  ␣
↪X_train_teacher_number_of_previously_posted_projects_norm,X_train_price_norm,X_train_projec
                  X_train_essay_count_norm))

X_te_tfidf_avg_w2v =␣
↪hstack((X_test_school_state_one_hot,X_test_categories_one_hot,X_test_sub_categories_one_hot
                  ␣
↪X_test_tfidf_w2v_vectors_pessays,X_test_tfidf_w2v_vectors_ptitle))

X_tr_tfidf_avg_w2v =␣
↪hstack((X_train_school_state_one_hot,X_train_categories_one_hot,␣
↪X_train_sub_categories_one_hot,\
                  ␣
↪X_train_tfidf_w2v_vectors_pessays,X_train_tfidf_w2v_vectors_ptitle))

#print("Final Data matrix")
print(X_tr.shape, y_train.shape)
```

```
print(X_te.shape, y_test.shape)

print("="*100)

print(X_tr_tfidf.shape, y_train.shape)
print(X_te_tfidf.shape, y_test.shape)
print("="*100)

print(X_tr_set5.shape,y_train.shape )
print(X_te_set5.shape,y_test.shape )
print("="*100)
```

```
(67000, 61) (67000,)
(33000, 61) (33000,)
================================================================================
====================
(67000, 49088) (67000,)
(33000, 49088) (33000,)
================================================================================
====================
(67000, 93) (67000,)
(33000, 93) (33000,)
================================================================================
====================
```

[ ]:

```

```

2 Assignment 5: Logistic Regression

[Task-1] Logistic Regression(either SGDClassifier with log loss, or LogisticRegression) on these feature sets

Set 1: categorical, numerical features + project_title(BOW) + preprocessed_eassay (`BOW with bi-grams` with `min_df=10` and `max_features=5000`)

Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (`TFIDF with bi-grams` with `min_df=10` and `max_features=5000`)

Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)

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

Hyper paramter tuning (find best hyper parameters corresponding the algorithm that you choose)

Find the best hyper parameter which will give the maximum AUC value

Find the best hyper paramter using k-fold cross validation or simple cross validation data

Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

```
</ul>
</li>
<br>
<li><strong>Representation of results</strong>
    <ul>
<li>You need to plot the performance of model both on train data and cross validation data for
<img src='train_cv_auc.JPG' width=300px></li>
<li>Once after you found the best hyper parameter, you need to train your model with it, and fi
<img src='train_test_auc.JPG' width=300px></li>
<li>Along with plotting ROC curve, you need to print the <a href='https://www.appliedaicourse.c
<img src='confusion_matrix.png' width=300px></li>
    </ul>
</li>
<br>
<li><strong>[Task-2] Apply Logistic Regression on the below feature set <font color='red'> Set
<li> Consider these set of features <font color='red'> Set 5 :</font>
        <ul>
            <li><strong>school_state</strong> : categorical data</li>
            <li><strong>clean_categories</strong> : categorical data</li>
            <li><strong>clean_subcategories</strong> : categorical data</li>
            <li><strong>project_grade_category</strong> :categorical data</li>
            <li><strong>teacher_prefix</strong> : categorical data</li>
            <li><strong>quantity</strong> : numerical data</li>
            <li><strong>teacher_number_of_previously_posted_projects</strong> : numerical data
            <li><strong>price</strong> : numerical data</li>
            <li><strong>sentiment score's of each of the essay</strong> : numerical data</li>
            <li><strong>number of words in the title</strong> : numerical data</li>
            <li><strong>number of words in the combine essays</strong> : numerical data</li>
        </ul>
    And apply the Logistic regression on these features by finding the best hyper paramter as s
</li>
<br>
<li><strong>Conclusion</strong>
    <ul>
<li>You need to summarize the results at the end of the notebook, summarize it in the table for
    <img src='summary.JPG' width=400px>
</li>
    </ul>
```

2. Logistic Regression

2.4 Appling Logistic Regression on different kind of featurization as mentioned in the instructions

Apply Logistic Regression on different kind of featurization as mentioned in the instructions
For Every model that you work on make sure you do the step 2 and step 3 of instrucations

[57]:
```python
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability␣
↪estimates of the positive class
```

```
    # 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%1000 = 49000
    # 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
```

[58]:
```
# 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", np.round(t,3))
    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
```

[59]:
```
def logistic_regression_validation(X_train,y_train,X_test,y_test):
    from sklearn import metrics
    from sklearn.model_selection import GridSearchCV
    from sklearn.linear_model import LogisticRegression
    from sklearn.metrics import roc_auc_score
    """
    y_true : array, shape = [n_samples] or [n_samples, n_classes]
    True binary labels or binary label indicators.

    y_score : array, shape = [n_samples] or [n_samples, n_classes]
    Target scores, can either be probability estimates of the positive class,␣
↪confidence values, or non-thresholded measure of
    decisions (as returned by decision_function on some classifiers).
    For binary y_true, y_score is supposed to be the score of the class with␣
↪greater label.
```

```
"""
# Create regularization penalty space
penalty = ['l1', 'l2']

# Create regularization hyperparameter space
C = np.logspace(0.00001, 40, 10)
print (C)

print ("*")

# Create hyperparameter options
tuned_parameters = dict(C=C, penalty=penalty)

#tuned_parameters = [{'C': [19**-6, 10**-5,10**-4, 10**-2,10**0, 10,10**2,␣
↪10**4]}]

# print ([{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}])

#Using GridSearchCV
model = GridSearchCV(LogisticRegression(), tuned_parameters, scoring =␣
↪'f1', cv=5)

model.fit(X_train, y_train)

print("model.best_estimator_ = %s" % model.best_estimator_)
print("model.score = %s" % model.score(X_test, y_test))

y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)


# calculate the fpr and tpr for all thresholds of the classification
probs = model.predict_proba(X_test)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)

# method I: plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
```

```
    plt.show()


    # calculate accuracy of class predictions
    from sklearn import metrics
    print ("Accuracy Score = %s" % metrics.accuracy_score(y_test, y_test_pred))
    print ("=====================================================")
    print("Train confusion matrix")
    print(metrics.confusion_matrix(y_train, y_train_pred))
    print("Test confusion matrix")
    print(metrics.confusion_matrix(y_test, y_test_pred))
```

[60]:
```
def logistic_regression_for_Best_Tuned_Parameter(X_train,y_train,X_test,y_test,␣
↪TunedParameter):
    from sklearn import metrics
    from sklearn.model_selection import GridSearchCV
    from sklearn.linear_model import LogisticRegression
    from sklearn.metrics import roc_auc_score

    model = LogisticRegression(C=TunedParameter)

    model.fit(X_train, y_train)

    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)


    # calculate the fpr and tpr for all thresholds of the classification
    probs = model.predict_proba(X_test)
    preds = probs[:,1]
    fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
    roc_auc = metrics.auc(fpr, tpr)

    # method I: plt
    plt.title('Receiver Operating Characteristic')
    plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
    plt.legend(loc = 'lower right')
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()

    # calculate accuracy of class predictions
    from sklearn import metrics
    print ("Accuracy Score = %s" % metrics.accuracy_score(y_test, y_test_pred))
```

```
    print ("=====================================================")
    print("Train confusion matrix")
    print(metrics.confusion_matrix(y_train, y_train_pred))
    print("Test confusion matrix")
    print(metrics.confusion_matrix(y_test, y_test_pred))
```

2.0.1 Split the normalized data into training and test sets

Logic below is simialr as covred in kanalysis_cross_validation(X,y), but here logic is for to calculate confusion matrix, acuarcy ration for best K as we already foound best K after trying best accuracy for multiple K values. We can apply K -fold CV to either the hyperparameter tuning, performance reporting, or both. The advantage of this approach is that the performance is less sensitive to unfortunate splits of data. In addition, it utilize data better since each example can be used for both training and validation/testing.

Let's use K -Fold CV to select the hyperparamter n_neighbors of the KNeighborsClassifier:

2.0.2 How to speculate the performance of the model using ROC Curve?

An excellent model has AUC near to the 1 which means it has good measure of separability. A poor model has AUC near to the 0 which means it has worst measure of separability. In fact it means it is reciprocating the result. It is predicting 0s as 1s and 1s as 0s. And when AUC is 0.5, it means model has no class separation capacity whatsoever

2.0.3 Split the normalized data into training and test sets

This step is required to prepare us for the fitting (i.e. training) the #model later. The "X" variable is a collection of all the features. The "y" variable is the target label which specifies the #classification of 1 or 0 based. Our goal will be to identify which category the new data point should fall into. Evaluate the predictions. Evaluate the Model by reviewing the classification report or confusion matrix. By reviewing these tables, we are able to evaluate how accurate our model is with new values.

[61]:
```
def LogicRegression_Tuned_Param_Analysis(X_train,y_train,X_test,y_test):
    from sklearn import model_selection
    from mlxtend.plotting import plot_decision_regions
    from sklearn.linear_model import LogisticRegression
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import accuracy_score
    from sklearn.metrics import roc_auc_score
    # Import classification report and confusion matrix to evaluate predictions
    from sklearn.metrics import classification_report, confusion_matrix

    train_auc = []
    test_auc = []
    Tunedparams = [19**-6, 10**-5,10**-4, 10**-2,10**0, 10,10**2, 10**3, 10**4]
    for TunedParameter in Tunedparams:
        model = LogisticRegression(C=TunedParameter)
```

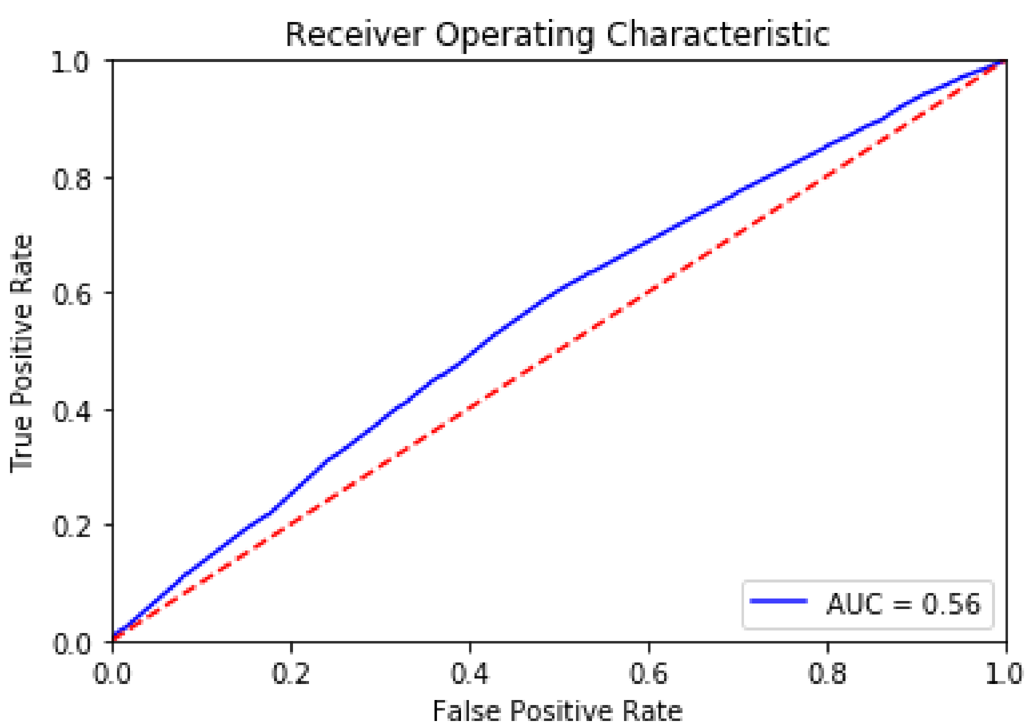
```
        # fitting the model on crossvalidation train
        model.fit(X_train, y_train)

        # predict the response on the crossvalidation train
        y_train_pred = model.predict(X_train)  # predicting the value using cross validation data.

        # predict the response on the crossvalidation test
        y_test_pred = model.predict(X_test)  # predicting the value using cross validation data.


        # evaluate CV accuracy
        acc = accuracy_score(y_test, y_test_pred, normalize=True) * float(100)  # I get the accuracy score.
        print('\n Test Accuracy for Tuned Parameter = %s is %s' % (TunedParameter, acc))
        print("=========================================")

        # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
        # not the predicted outputs
        train_auc.append(roc_auc_score(y_train,y_train_pred))
        test_auc.append(roc_auc_score(y_test, y_test_pred))


    plt.plot(Tunedparams, train_auc, label='Train AUC')
    plt.plot(Tunedparams, test_auc, label='Test AUC')

    plt.scatter(Tunedparams, train_auc, label='Train AUC points')
    plt.scatter(Tunedparams, test_auc, label='Test AUC points')

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

2.0.4 2.4.0 Applying Logistic Regression Set 1: categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW), SET 1

2.0.5 Logistic Regression feature selection

[62]:
```
logistic_regression_validation (X_tr,y_train,X_te,y_test)
```

```
[1.00002303e+00 2.78261635e+04 7.74277549e+08 2.15446776e+13
 5.99491919e+17 1.66811761e+22 4.64162446e+26 1.29155627e+31
 3.59382286e+35 1.00000000e+40]
*
model.best_estimator_ = LogisticRegression(C=1.0000230261160268,
class_weight=None, dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='warn', n_jobs=None, penalty='l1',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
model.score = 0.9180150494270398
```



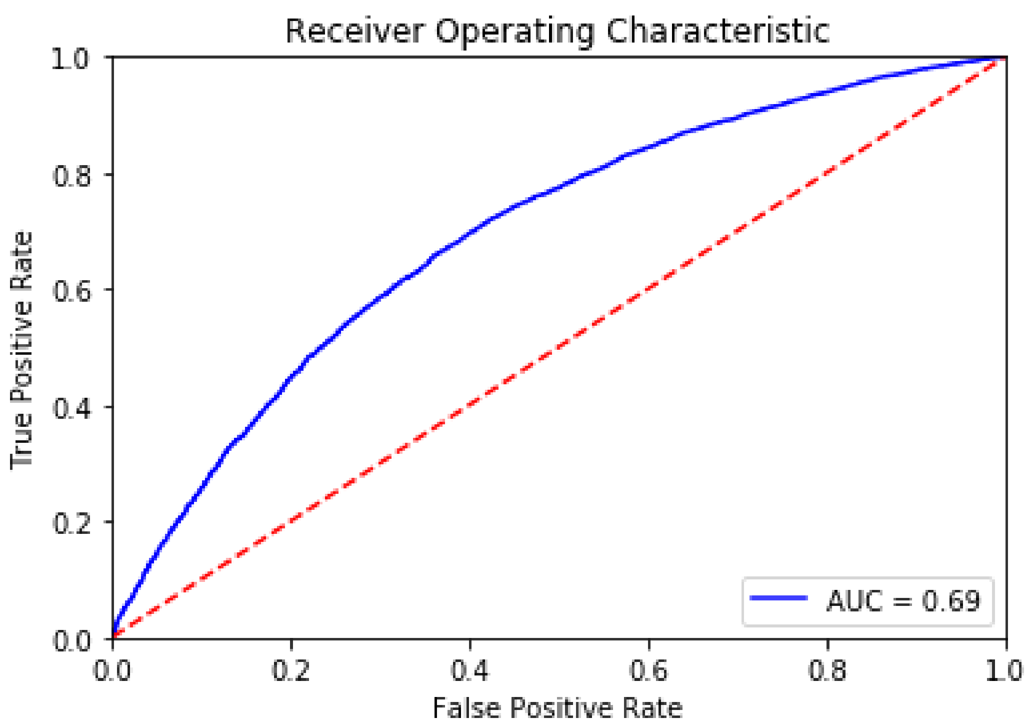

```
Accuracy Score = 0.8484545454545455
=====================================================
Train confusion matrix
[[    0 10153]
 [    0 56847]]
```

```
Test confusion matrix
[[    0  5001]
 [    0 27999]]
```

2.0.6 2.4.1 Applying Logistic Regression on BOW - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW), SET 1

[63]:
```
logistic_regression_validation (X_tr_bow,y_train,X_te_bow,y_test)
```

```
[1.00002303e+00 2.78261635e+04 7.74277549e+08 2.15446776e+13
 5.99491919e+17 1.66811761e+22 4.64162446e+26 1.29155627e+31
 3.59382286e+35 1.00000000e+40]
*
model.best_estimator_ = LogisticRegression(C=1.0000230261160268,
class_weight=None, dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='warn', n_jobs=None, penalty='l1',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
model.score = 0.9114060318581689
```



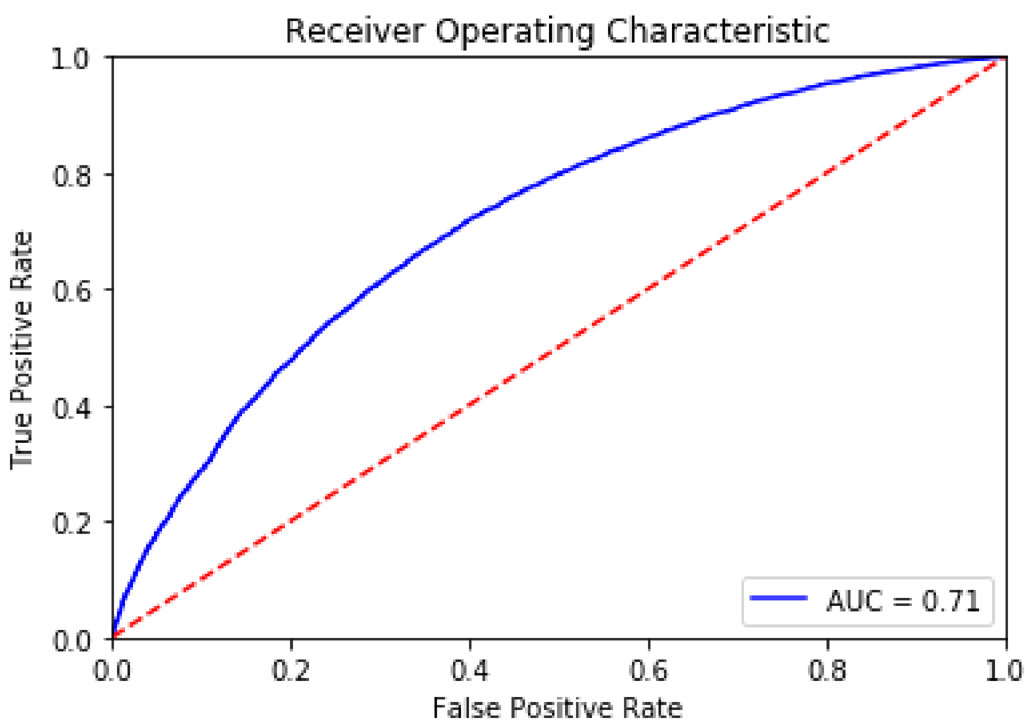

```
Accuracy Score = 0.8403939393939394
=====================================================
```

```
Train confusion matrix
[[ 2297  7856]
 [  824 56023]]
Test confusion matrix
[[  641  4360]
 [  907 27092]]
```

2.0.7 2.4.2 Applying Logistic Regression on TFIDF Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_essay (TFIDF), SET 2

[64]:
```
logistic_regression_validation(X_tr_tfidf,y_train,X_te_tfidf,y_test)
```

```
[1.00002303e+00 2.78261635e+04 7.74277549e+08 2.15446776e+13
 5.99491919e+17 1.66811761e+22 4.64162446e+26 1.29155627e+31
 3.59382286e+35 1.00000000e+40]
*
model.best_estimator_ = LogisticRegression(C=1.0000230261160268,
class_weight=None, dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
model.score = 0.9177266621337004
```

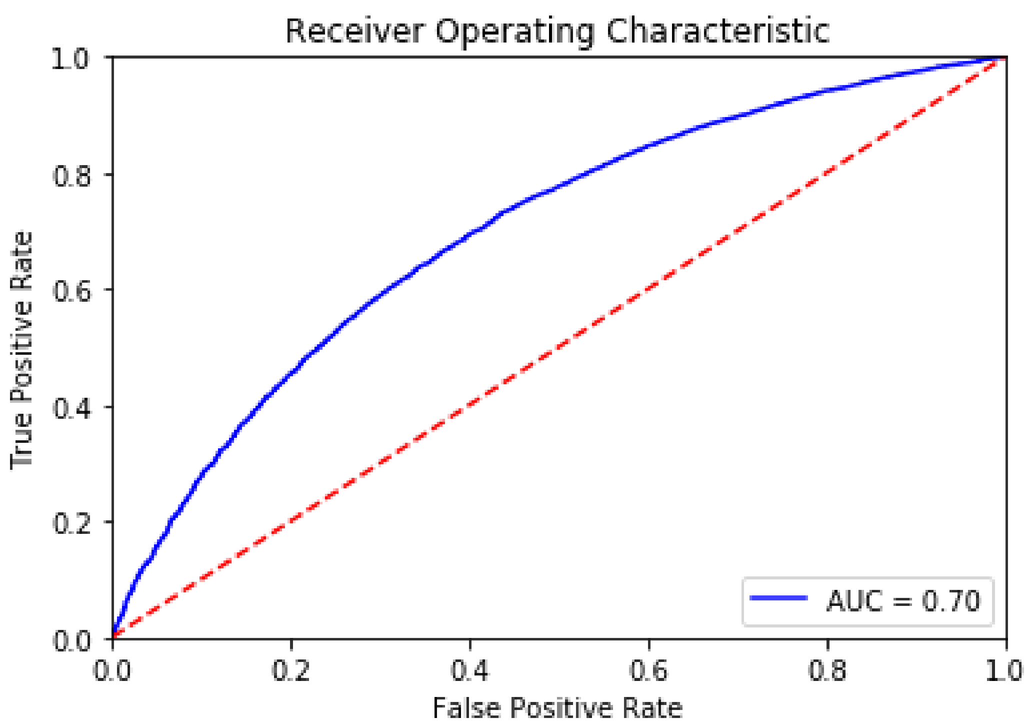


```
Accuracy Score = 0.8496666666666667
=====================================================
Train confusion matrix
[[ 1339  8814]
 [  284 56563]]
Test confusion matrix
[[  370  4631]
 [  330 27669]]
```

2.4.2 Applying Logistic Regression on TFIDF Avg W2V SET 2

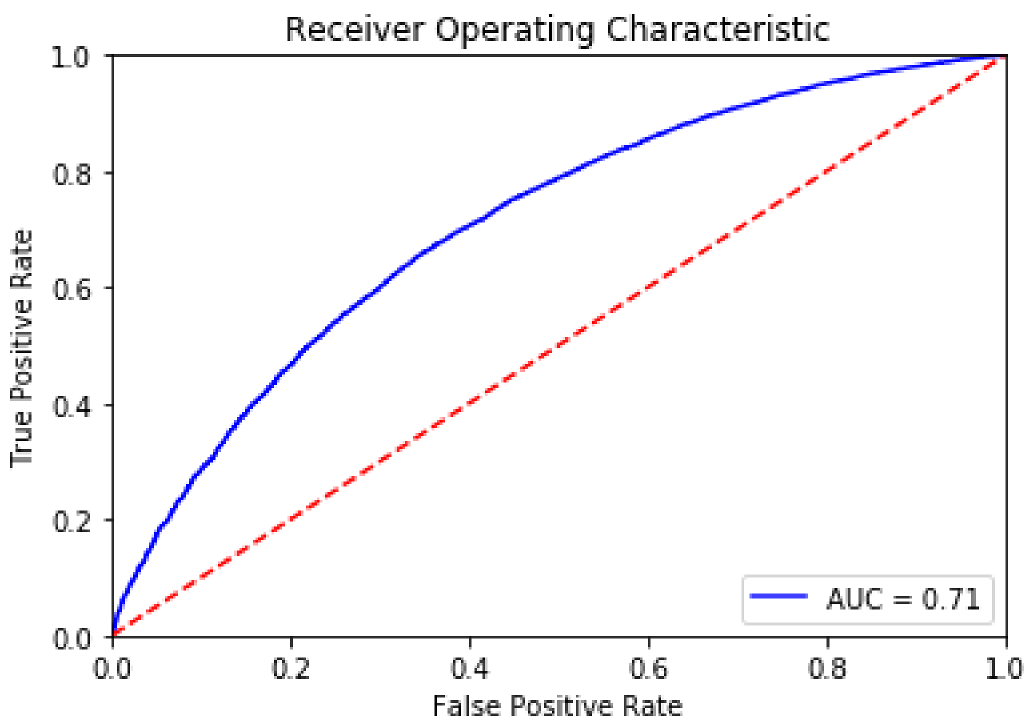
[65]:
```
logistic_regression_validation(X_tr_tfidf_avg_w2v,y_train,X_te_tfidf_avg_w2v,y_test)
```

```
[1.00002303e+00 2.78261635e+04 7.74277549e+08 2.15446776e+13
 5.99491919e+17 1.66811761e+22 4.64162446e+26 1.29155627e+31
 3.59382286e+35 1.00000000e+40]
*
model.best_estimator_ = LogisticRegression(C=1.0000230261160268,
class_weight=None, dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='warn', n_jobs=None, penalty='l1',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
model.score = 0.9176996679051722
```



```
Accuracy Score = 0.8483030303030303
=====================================================
Train confusion matrix
[[  221  9932]
 [  113 56734]]
Test confusion matrix
[[   84  4917]
 [   89 27910]]
```

[66]:
```
TunedParameter = 2
logistic_regression_for_Best_Tuned_Parameter(X_tr_tfidf,y_train,X_te_tfidf,y_test,TunedParamet
```

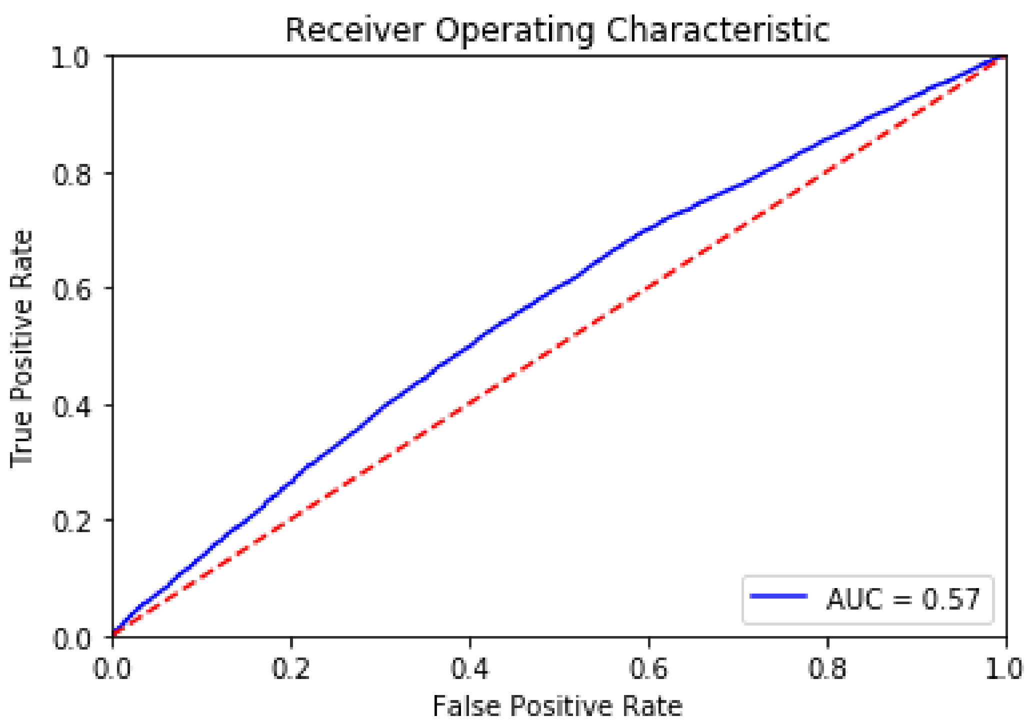



```
Accuracy Score = 0.846909090909091
=====================================================
Train confusion matrix
[[ 2124  8029]
 [  402 56445]]
Test confusion matrix
[[  523  4478]
 [  574 27425]]
```

2.0.8 2.4.3 Applying Logistic Regression on AVG W2V, SET 3

[67]:
```
logistic_regression_validation(X_tr_avg_w2v,y_train,X_te_avg_w2v,y_test)
```

```
[1.00002303e+00 2.78261635e+04 7.74277549e+08 2.15446776e+13
 5.99491919e+17 1.66811761e+22 4.64162446e+26 1.29155627e+31
 3.59382286e+35 1.00000000e+40]
*
model.best_estimator_ = LogisticRegression(C=1.0000230261160268,
class_weight=None, dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='warn', n_jobs=None, penalty='l1',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
model.score = 0.9178392207025485
```

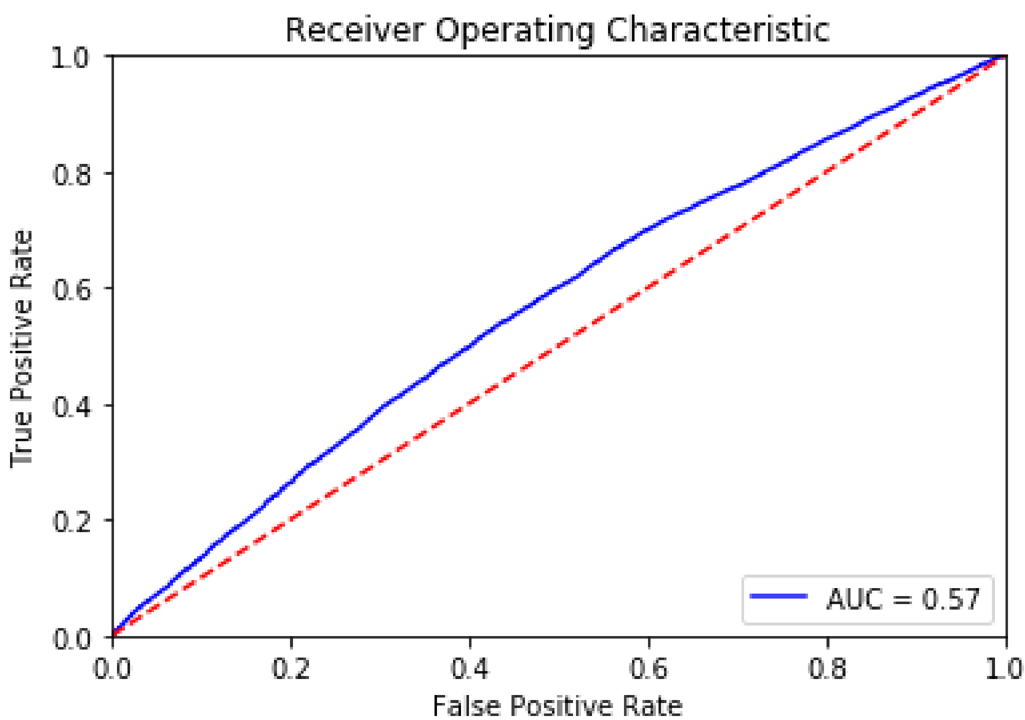



```
Accuracy Score = 0.8481818181818181
=====================================================
Train confusion matrix
[[   18 10135]
 [    4 56843]]
Test confusion matrix
[[    6  4995]
 [   15 27984]]
```

2.0.9 2.4.4 Applying Logistic Regression on TFIDF W2V, SET 4

[68]:
```
logistic_regression_validation(X_tr_tfidf_w2v,y_train,X_te_tfidf_w2v,y_test)
```

```
[1.00002303e+00 2.78261635e+04 7.74277549e+08 2.15446776e+13
 5.99491919e+17 1.66811761e+22 4.64162446e+26 1.29155627e+31
 3.59382286e+35 1.00000000e+40]
*
model.best_estimator_ = LogisticRegression(C=1.0000230261160268,
class_weight=None, dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='warn', n_jobs=None, penalty='l1',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
model.score = 0.9178214736704003
```

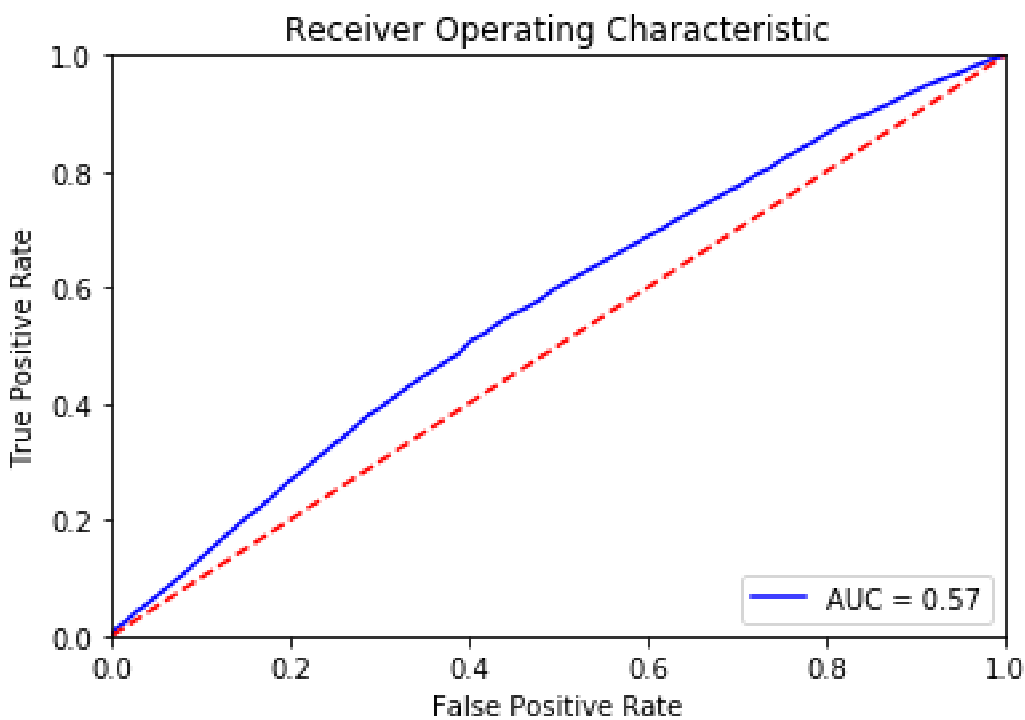



```
Accuracy Score = 0.8481515151515151
=====================================================
Train confusion matrix
[[   18 10135]
 [    4 56843]]
Test confusion matrix
[[    6  4995]
 [   16 27983]]
```

2.0.10 2.4.2 Applying Logistic Regression on TFIDF Set 5: categorical, numerical features SET 5

[69]:
```
logistic_regression_validation(X_tr_set5,y_train,X_te_set5,y_test)
```

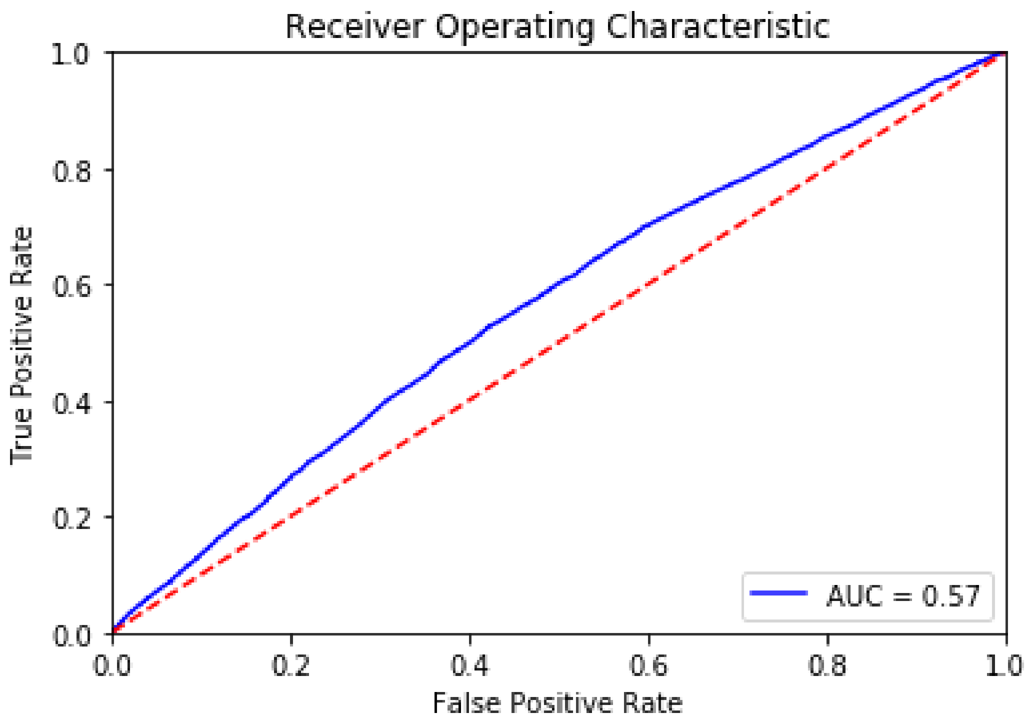
```
[1.00002303e+00 2.78261635e+04 7.74277549e+08 2.15446776e+13
 5.99491919e+17 1.66811761e+22 4.64162446e+26 1.29155627e+31
 3.59382286e+35 1.00000000e+40]
*
model.best_estimator_ = LogisticRegression(C=1.0000230261160268,
class_weight=None, dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='warn', n_jobs=None, penalty='l1',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
model.score = 0.9180150494270398
```




```
Accuracy Score = 0.8484545454545455
=====================================================
Train confusion matrix
[[    0 10153]
 [    0 56847]]
Test confusion matrix
```

```
[[    0  5001]
 [    0 27999]]
```

[70]:
```
TunedParameter =01.1
logistic_regression_for_Best_Tuned_Parameter(X_tr_avg_w2v,y_train,X_te_avg_w2v,y_test,
  ↪TunedParameter)
```



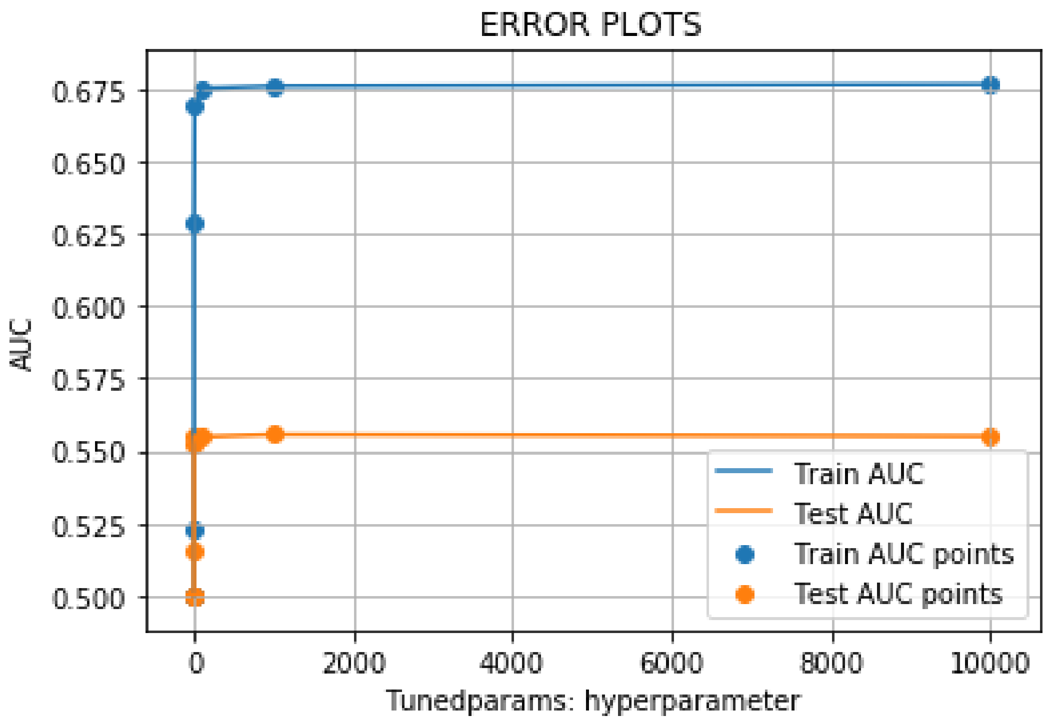

```
Accuracy Score = 0.8480606060606061
=====================================================
Train confusion matrix
[[   27 10126]
 [   17 56830]]
Test confusion matrix
[[    9  4992]
 [   22 27977]]
```

2.1 2.5 Feature selection for Best Tuned Parameter /font>

2.2 2.5.1 Tuned Param-Analysis categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW), SET 1

[71]:
```
LogicRegression_Tuned_Param_Analysis(X_tr_bow,y_train,X_te_bow,y_test)
```

```
 Test Accuracy for Tuned Parameter = 2.1255845968746978e-08 is 84.84545454545454
==========================================

 Test Accuracy for Tuned Parameter = 1e-05 is 84.84545454545454
==========================================

 Test Accuracy for Tuned Parameter = 0.0001 is 84.84242424242424
==========================================

 Test Accuracy for Tuned Parameter = 0.01 is 84.92121212121212
==========================================

 Test Accuracy for Tuned Parameter = 1 is 83.4939393939394
==========================================

 Test Accuracy for Tuned Parameter = 10 is 82.08484848484848
==========================================

 Test Accuracy for Tuned Parameter = 100 is 81.71212121212122
==========================================

 Test Accuracy for Tuned Parameter = 1000 is 81.74242424242424
==========================================

 Test Accuracy for Tuned Parameter = 10000 is 81.67878787878789
==========================================
```

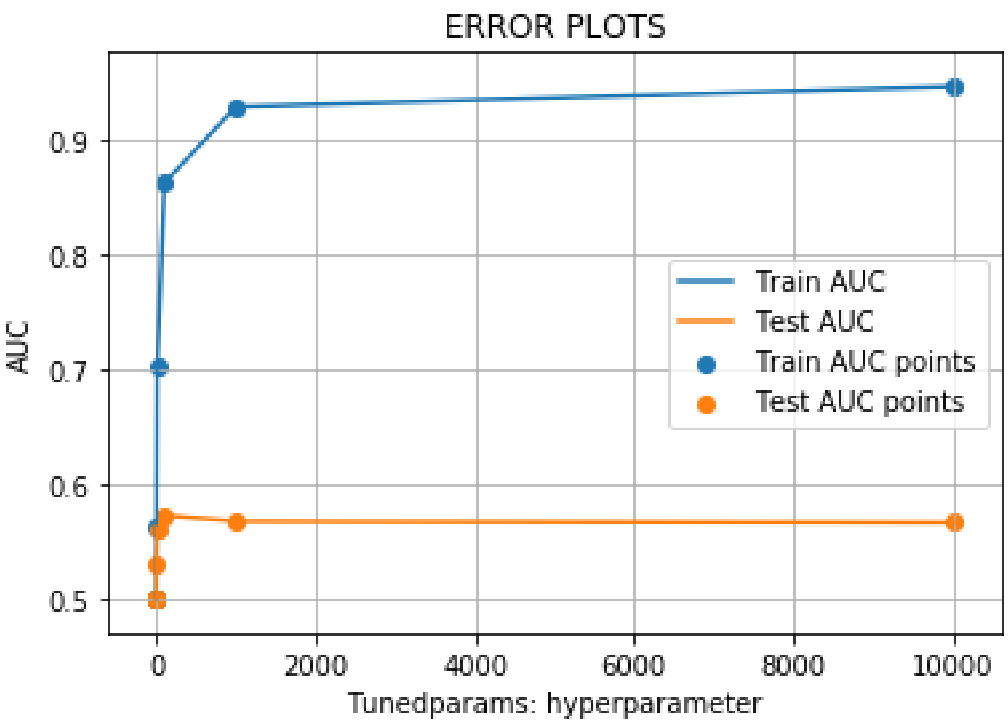


2.2.1 2.5.2 Tuned Param -Analysis on TFIDF Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_essay (TFIDF), SET 2

[72]:
```
LogicRegression_Tuned_Param_Analysis(X_tr_tfidf,y_train,X_te_tfidf,y_test)
```

```
 Test Accuracy for Tuned Parameter = 2.1255845968746978e-08 is 84.84545454545454
==========================================

 Test Accuracy for Tuned Parameter = 1e-05 is 84.84545454545454
==========================================

 Test Accuracy for Tuned Parameter = 0.0001 is 84.84545454545454
==========================================

 Test Accuracy for Tuned Parameter = 0.01 is 84.84545454545454
==========================================

 Test Accuracy for Tuned Parameter = 1 is 84.96666666666667
==========================================

 Test Accuracy for Tuned Parameter = 10 is 83.30606060606061
==========================================
```

```
Test Accuracy for Tuned Parameter = 100 is 80.46666666666667
==========================================

Test Accuracy for Tuned Parameter = 1000 is 78.18787878787879
==========================================

Test Accuracy for Tuned Parameter = 10000 is 77.43939393939394
==========================================
```

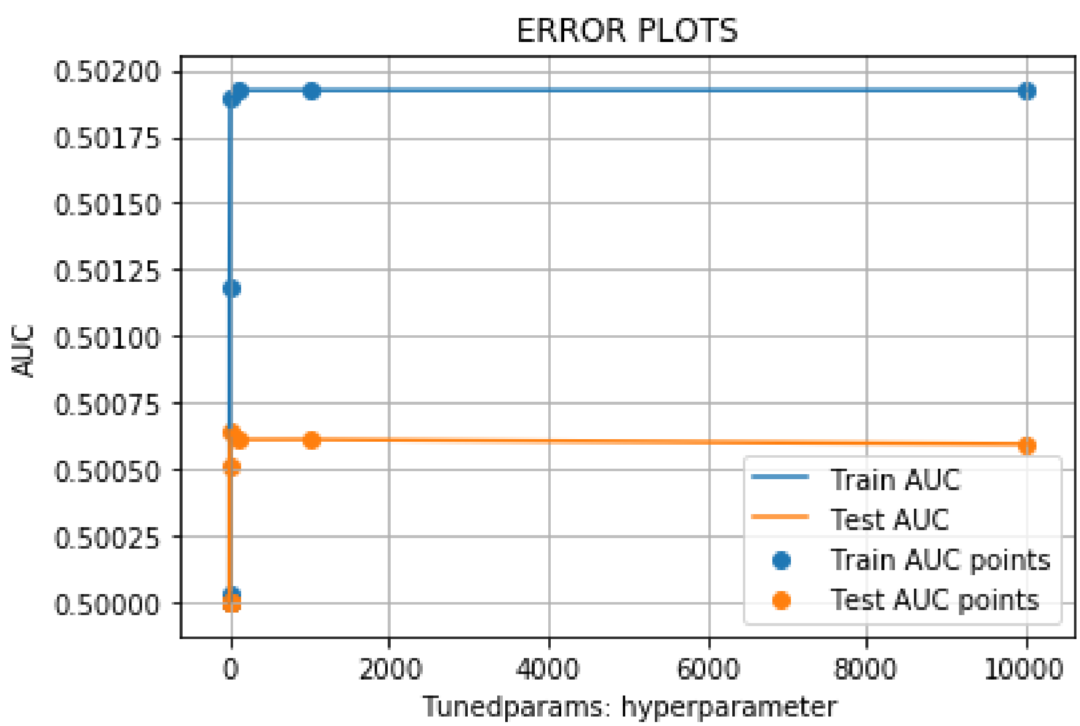


2.2.2 2.5.3 Tuned Parameter-Analysis on AVG W2V - categorical, numerical features + project_title(AVG W2V)+ preprocessed_essay (AVG W2V), SET 3

[73]:
```
LogicRegression_Tuned_Param_Analysis(X_tr_avg_w2v,y_train,X_te_avg_w2v,y_test)
```

```
Test Accuracy for Tuned Parameter = 2.1255845968746978e-08 is 84.84545454545454
==========================================

Test Accuracy for Tuned Parameter = 1e-05 is 84.84545454545454
==========================================

Test Accuracy for Tuned Parameter = 0.0001 is 84.84545454545454
```

```
=========================================

 Test Accuracy for Tuned Parameter = 0.01 is 84.84545454545454
=========================================

 Test Accuracy for Tuned Parameter = 1 is 84.80606060606061
=========================================

 Test Accuracy for Tuned Parameter = 10 is 84.7878787878788
=========================================

 Test Accuracy for Tuned Parameter = 100 is 84.78181818181818
=========================================

 Test Accuracy for Tuned Parameter = 1000 is 84.78181818181818
=========================================

 Test Accuracy for Tuned Parameter = 10000 is 84.77878787878788
=========================================
```

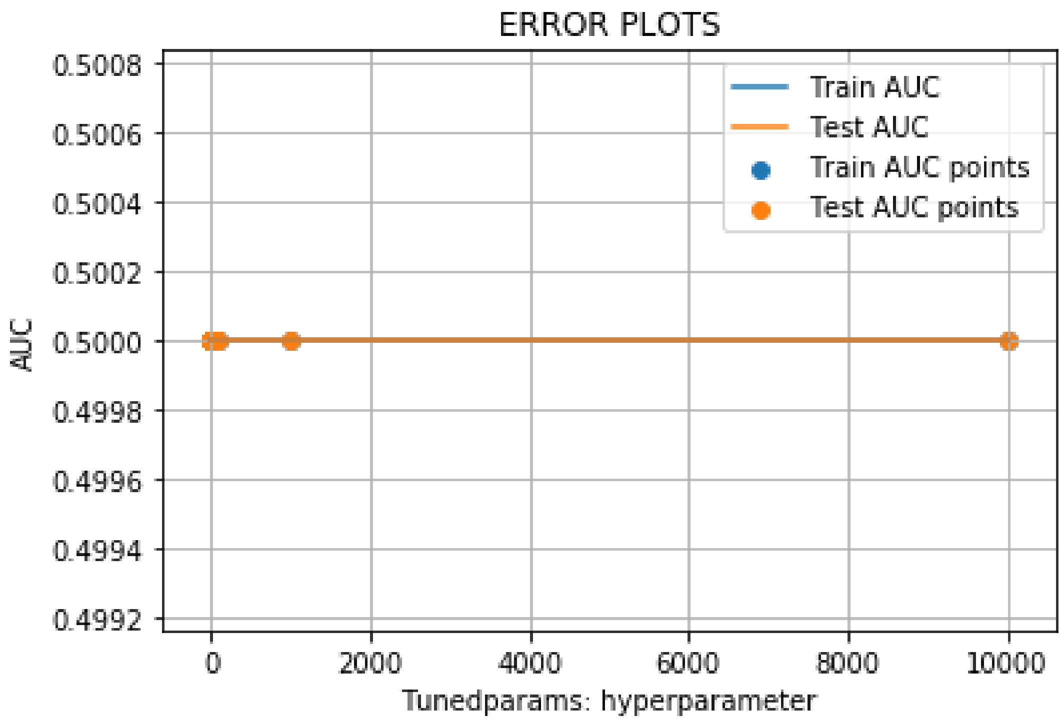


2.2.3 Tuned Param Analysis on TFIDF Set 5: categorical, numerical features + SET 5

2.2.4 Logic Regression Analysis on Best Tuned Parameter

[74]:
```
LogicRegression_Tuned_Param_Analysis(X_tr_set5,y_train,X_te_set5,y_test)
```

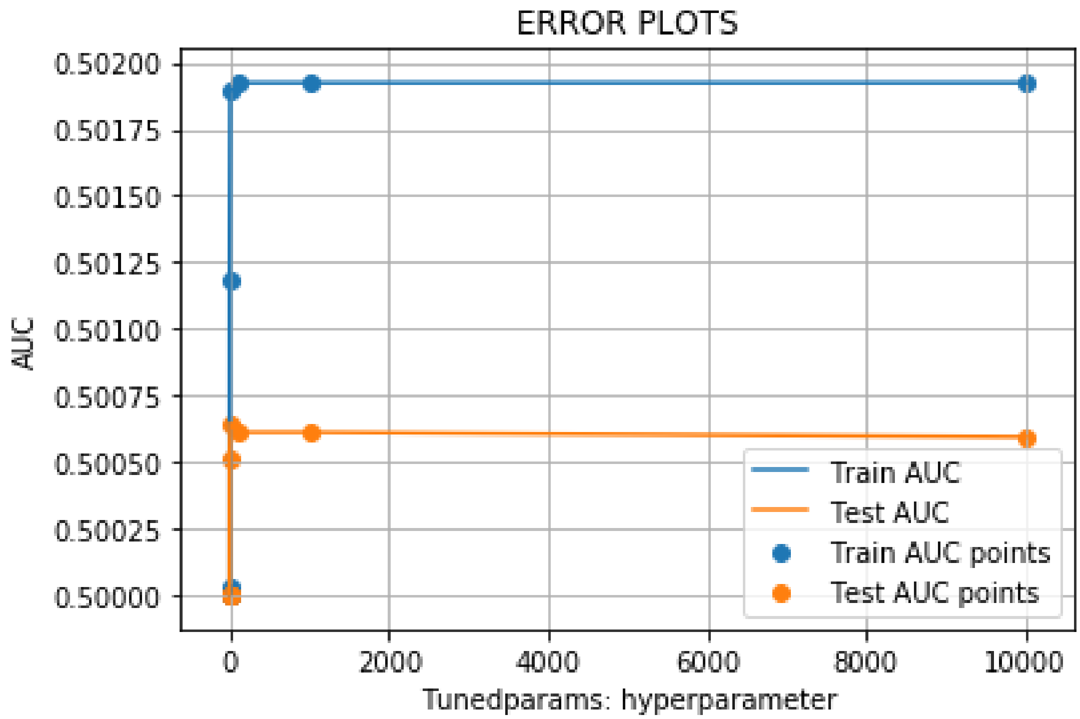
```
 Test Accuracy for Tuned Parameter = 2.1255845968746978e-08 is 84.84545454545454
=========================================

 Test Accuracy for Tuned Parameter = 1e-05 is 84.84545454545454
=========================================

 Test Accuracy for Tuned Parameter = 0.0001 is 84.84545454545454
=========================================

 Test Accuracy for Tuned Parameter = 0.01 is 84.84545454545454
=========================================

 Test Accuracy for Tuned Parameter = 1 is 84.84545454545454
=========================================

 Test Accuracy for Tuned Parameter = 10 is 84.84545454545454
=========================================

 Test Accuracy for Tuned Parameter = 100 is 84.84545454545454
=========================================

 Test Accuracy for Tuned Parameter = 1000 is 84.84545454545454
=========================================

 Test Accuracy for Tuned Parameter = 10000 is 84.84545454545454
=========================================
```



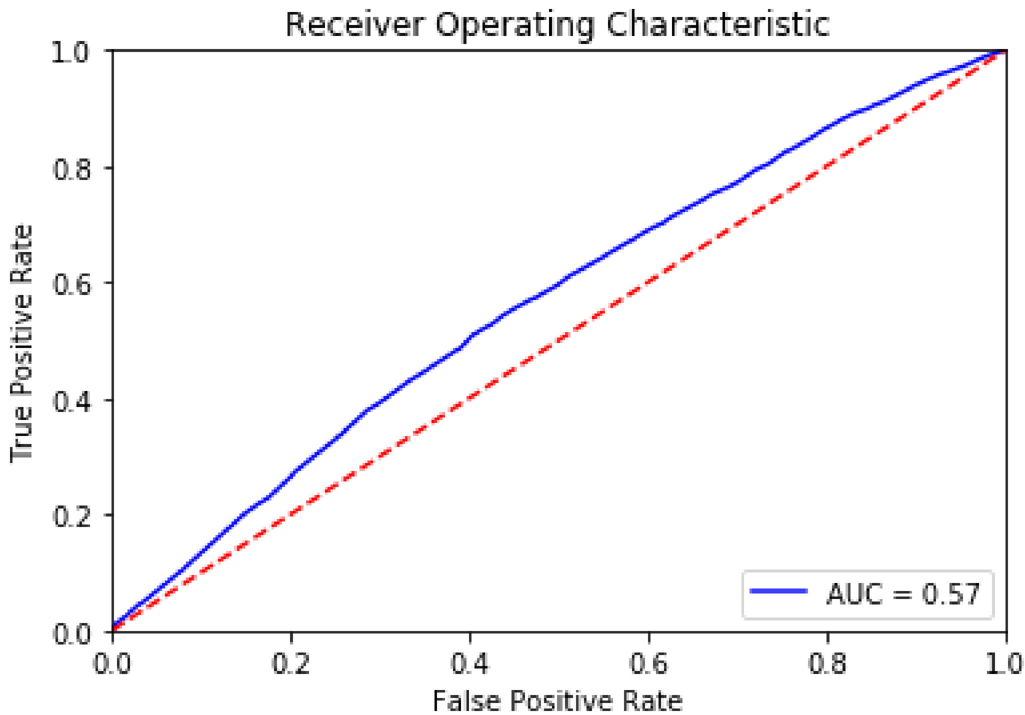
[75]:
```
LogicRegression_Tuned_Param_Analysis(X_tr_tfidf_w2v,y_train,X_te_tfidf_w2v,y_test)
```

```
 Test Accuracy for Tuned Parameter = 2.1255845968746978e-08 is 84.84545454545454
=========================================

 Test Accuracy for Tuned Parameter = 1e-05 is 84.84545454545454
=========================================

 Test Accuracy for Tuned Parameter = 0.0001 is 84.84545454545454
=========================================

 Test Accuracy for Tuned Parameter = 0.01 is 84.84545454545454
=========================================

 Test Accuracy for Tuned Parameter = 1 is 84.80606060606061
=========================================

 Test Accuracy for Tuned Parameter = 10 is 84.78787878787878
=========================================

 Test Accuracy for Tuned Parameter = 100 is 84.78181818181818
=========================================
```

```
Test Accuracy for Tuned Parameter = 1000 is 84.78181818181818
==========================================

Test Accuracy for Tuned Parameter = 10000 is 84.77878787878788
==========================================
```



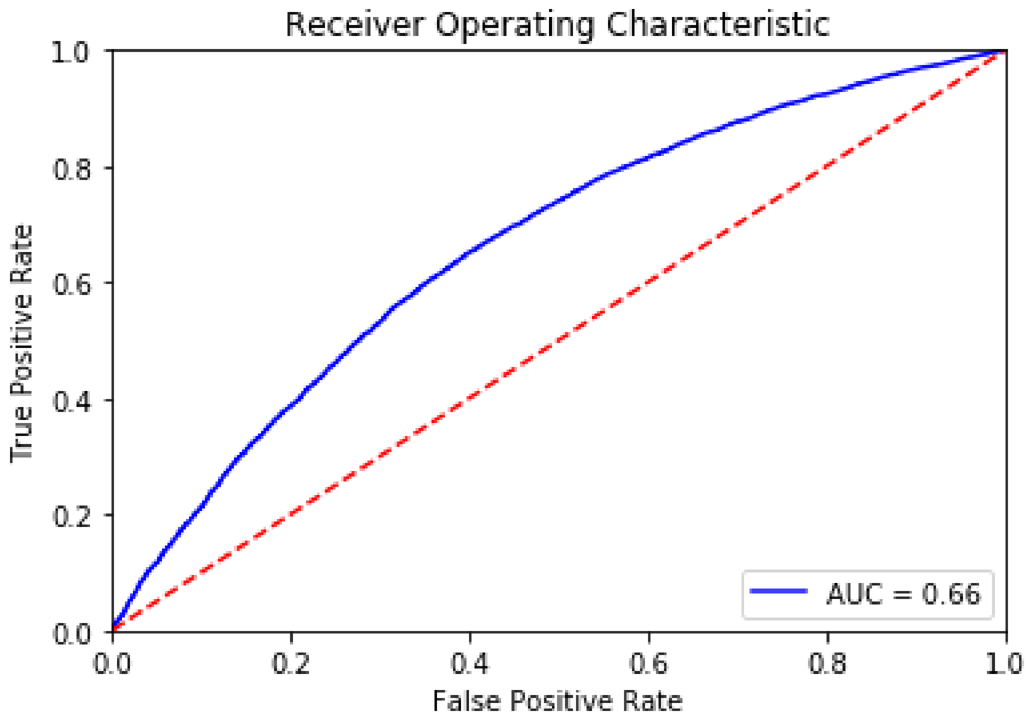
[76]:
```
TunedParameter = 13.09
logistic_regression_for_Best_Tuned_Parameter(X_tr_set5,y_train,X_te_set5,y_test,TunedParameter
```




```
Accuracy Score = 0.8484545454545455
=====================================================
Train confusion matrix
[[    0 10153]
 [    0 56847]]
Test confusion matrix
[[    0  5001]
 [    0 27999]]
```

[77]:

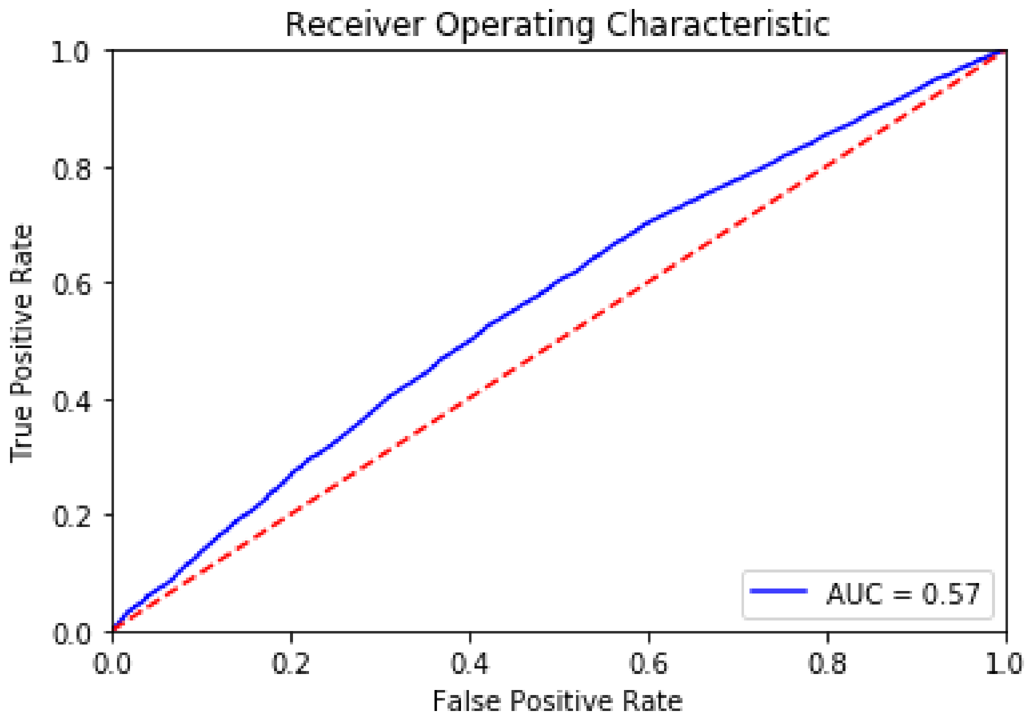
```
TunedParameter =10.01
logistic_regression_for_Best_Tuned_Parameter(X_tr_bow,y_train,X_te_bow,y_test,
 ↪TunedParameter)
```




```
Accuracy Score = 0.8208484848484848
=====================================================
Train confusion matrix
[[ 3663  6490]
 [ 1268 55579]]
Test confusion matrix
[[  871  4130]
 [ 1782 26217]]
```

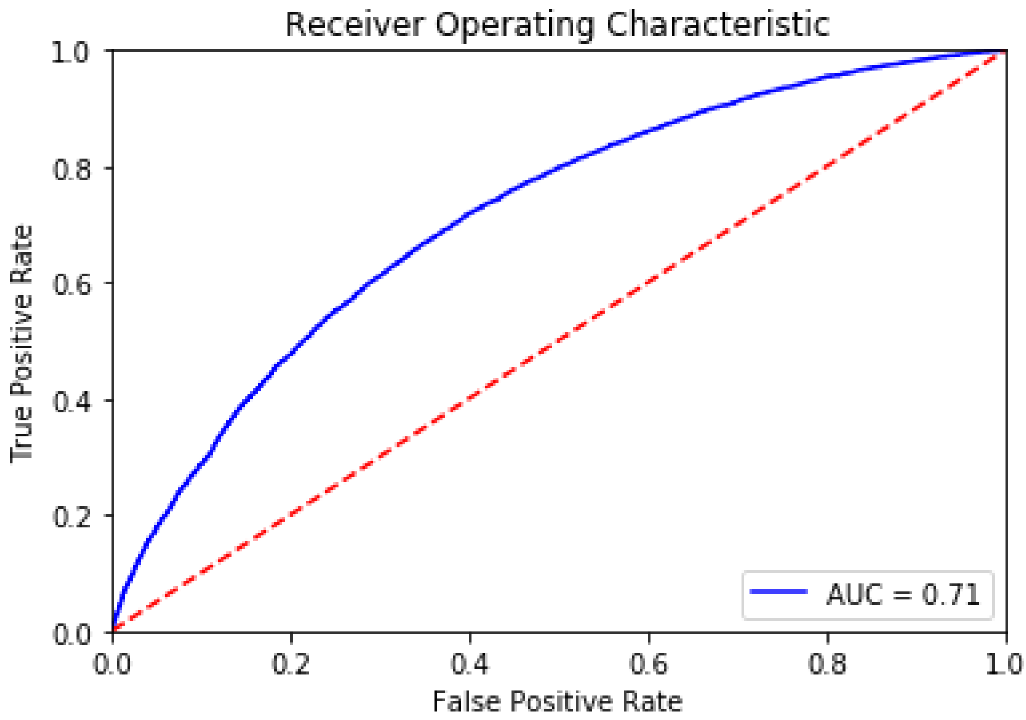
[78]:

```
TunedParameter = 2.0
logistic_regression_for_Best_Tuned_Parameter(X_tr_avg_w2v,y_train,X_te_avg_w2v,y_test,␣
↪TunedParameter)
```




```
Accuracy Score = 0.8478484848484849
=====================================================
Train confusion matrix
[[   34 10119]
 [   19 56828]]
Test confusion matrix
[[    9  4992]
 [   29 27970]]
```

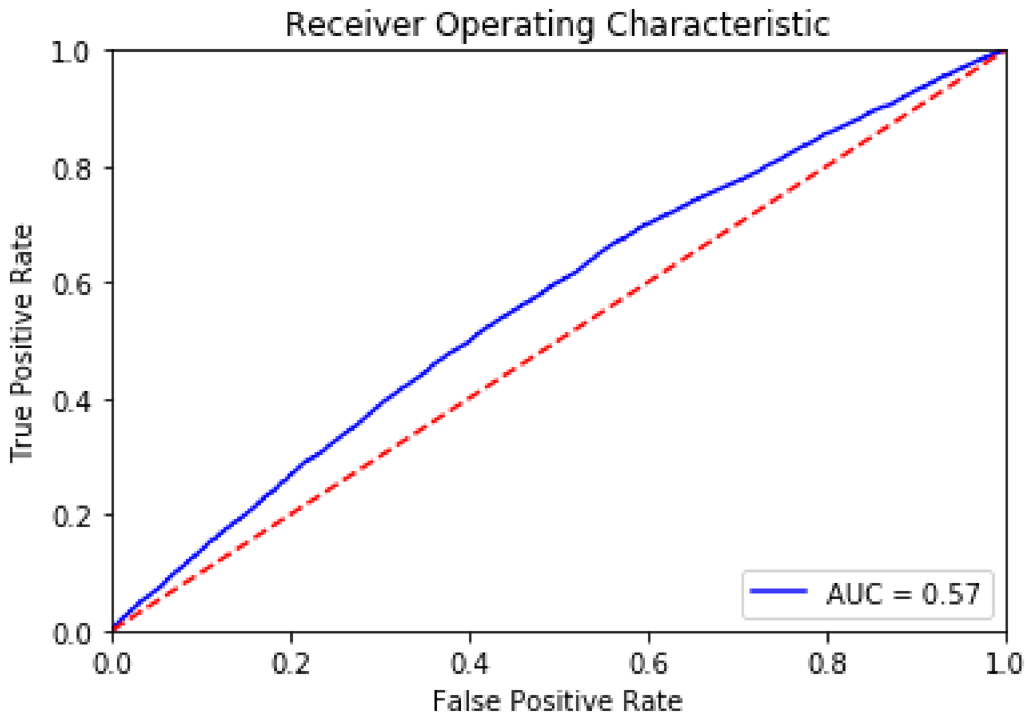
[79]:
```
TunedParameter = 1.02
logistic_regression_for_Best_Tuned_Parameter(X_tr_tfidf,y_train,X_te_tfidf,y_test,TunedParamet
```




```
Accuracy Score = 0.8496363636363636
=====================================================
Train confusion matrix
[[ 1360  8793]
 [  289 56558]]
Test confusion matrix
[[  375  4626]
 [  336 27663]]
```

[84]:

```
TunedParameter = 0.1
logistic_regression_for_Best_Tuned_Parameter(X_tr_tfidf_w2v,y_train,X_te_tfidf_w2v,y_test,Tune
```




```
Accuracy Score = 0.8483030303030303
=====================================================
Train confusion matrix
[[    10 10143]
 [     2 56845]]
Test confusion matrix
[[     4  4997]
 [     9 27990]]
```

3. Conclusions

Logistic Regression is a stastical method for analyzing a dataset in which there are one or more independent variables that determine the outcome.

2.3 Summary of above program as below:

2.3.1 Step 1: Import the necessary Libraries

we will need to import libraries that allow for data analysis and data visualization to get acclimated to the dataset. We will be using pandas, numpy, matplotlib and seaborn to conduct this. Data Exploration libraries

2.3.2 Step 2: Read in the dataset.

We will use the pandas .read_csv() method to read in the dataset. Then we will use the. head() method to observe the first few rows of the data, to understand the information better. In our case, the feature(column) headers tell us pretty little. This is fine because we are merely trying to gain insight via classifying new data points by referencing it's neighboring elements.

2.3.3 Step 3: Standardize (normalize) the data scale to prep for Logistic regression.

Because the distance between pairs of points plays a critical part on the classification, it is necessary to normalize the data This will generate an array of values.

2.3.4 Step 4: Split the normalized data into training and test sets.

This step is required to prepare us for the fitting (i.e. training) the model later. The "X" variable is a collection of all the features. The "y" variable is the target label which specifies the classification of 1 or 0 based. Our goal will be to identify which category the new data point should fall into.

2.3.5 Step 5: Create and Train the Model.

Here we create a Logistic Regression Object and use the .fit() method to train the model. Upon completion of the model we should receive confirmation that the training has been complete

Please see functions as covered below, used in above program: def logistic_regression_validation(X,y): def

2.3.6 Step 6: Make Predictions.

Here we review where our model was accurate and where it misclassified elements.

Please see functions as covered below, used in above program: def logistic_regression_validation(X,y):

2.3.7 Step 7: Evaluate the predictions.

Evaluate the Model by reviewing the classification report or confusion matrix. By reviewing these tables, we are able to evaluate how accurate our model is with new values.

def logistic_regression_validation(X,y):

2.3.8 Setp 8:Classification Report :

This tells us our model was around 84% accurate... Print out classification report and confusion matrix

I have covered various set to show confusion matrix.

Please see section 2. covered various data sets and created confusion matrix.

2.3.9 Step 9: Evaluate alternative Tuned Parameter for better predictions.

To simplify the process of evaluating multiple cases of Alpha values, we create a function to derive the error using the average where our predictions were not equal to the test values.

Please see section 2. covered various data sets and created error accuracy reports.

2.3.10 Step 10: Adjust Tuned Parameter value per error rate evaluations

This is just fine tuning our model to increase accuracy. We will need to retrain our model with the new Alpha. Please see section 3 in above program. we have created confusion matrix for optimal Alpha value for various data sets. As we can see for optimal Alpha, Accuracy is much higher - so prediction is much better.