Visvesvaraya National Institute of Technology (VNIT), Nagpur

Machine Learning with Python (ECL 443)

Lab Report

Submitted by :
Pranay Dumbhare (BT19ECE089)
Semester 7

Submitted to :
Dr.Saugata Sinha
(Course Instructor)
Department of Electronics and Communication Engineering
VNIT Nagpur

Contents

1 Linear Regression 2
1.1 Abstract 2
1.2 Introduction 2
1.3 Problem Statement 2
1.4 Method/Procedure 2
1.5 Results/Discussion 4
1.6 Conclusion 6
1.7 Appendices 6

Linear Regression

1.1 Abstract: Machine Learning is a computer program which is used to train our machine to predict some outputs on the basis of previous inputs. With advancements in machine learning their are many algorithms available to find patterns in the given data so that the machine can learn from this patterns to predict future output. The accuracy of the model is how correctly it predicts the output.

1.2 Introduction: Linear Regression is a machine learning algorithm where their is a linear dependency between the dependent variable (y) and independent variables (x1,x2,...,xn). Following is the general representation of linear regression:

$$y = w_0 + w_1 * x_1 + w_2 * x_2 + ... + w_n * x_n$$

Where, y is the dependent variable,
x1,x2,...,xn are the independent variables and
w0,w1,...wn are the weights of these dependent variables which resembles how much the dependent variable(y) depends on the independent variable(xn).

1.3 Problem Statement: Predict the number of traffic fatalities caused on the basis of number of registered vehicles (in thousands), number of licensed drivers (in thousands) and the number of miles travelled (in millions) by the vehicles.

1.4 Method/Procedure:

Note: *The given matlab file (accidents.mat) contains many columns(both dependent and independent). From this columns, I have chosen the number of traffic fatalities as dependent variable and number of registered vehicles, number of licensed drivers and the number of miles travelled by the vehicles as independent variables.*

1. Load the given data file is **Matlab_accidents.mat** using the user-defined function **BT19ECE089_train_test_split**. The data is loaded as a pandas dataframe.
2. After the data is loaded into the dataframe, a new dataframe is created using the selected dependent and independent variables.
3. Normalize the data frame column wise.
4. The data is then split into train and test with split ratio 0.3. Then matrix theta train , y train , theta test , y test are created to perform training and testing.

5. **Pseudo Inverse Method**
   (a) Compute optimum weights by computing pseudo inverse of the theta matrix and multiplying it with y train.
   (b) Predict the output (y pred) by multiplying this optimum weights with theta train.
   (c) Plot y pred and y test to visualize how accurately out model predicts the output.
   (d) Calculate root mean square error , mean squared error and mean absolute error to analyze model analytically.
6. **Gradient Descent Method**
   (a) Compute optimum weights using gradient descent algorithm. Here we use number of iterations as 1000 and learning rate as 0.03
   (b) Predict the output (y pred) by multiplying this optimum weights with theta train.
   (c) Plot y pred and y test to visualize how accurately out model predicts the output.
   (d) Calculate root mean square error , mean squared error and mean absolute error to analyze model analytically.
7. For change in relationship between the input and output variables, we consider that the fatalities (y) varies as square of Registered vehicles (thousands) (x1). This term is added after squaring x1, and the accuracy is calculated again.

1.5 Results/Discussion:

Note: *The given matlab file (accidents.mat) contains many columns(both dependent and independent). From this columns, I have chosen the number of traffic fatalities as dependent variable and number of registered vehicles, number of licensed drivers and the number of miles travelled by the vehicles as independent variables.*

1. **Pseudo Inverse Method** The RMS Accuracy obtained using the Pseudo Inverse Method is 96%.

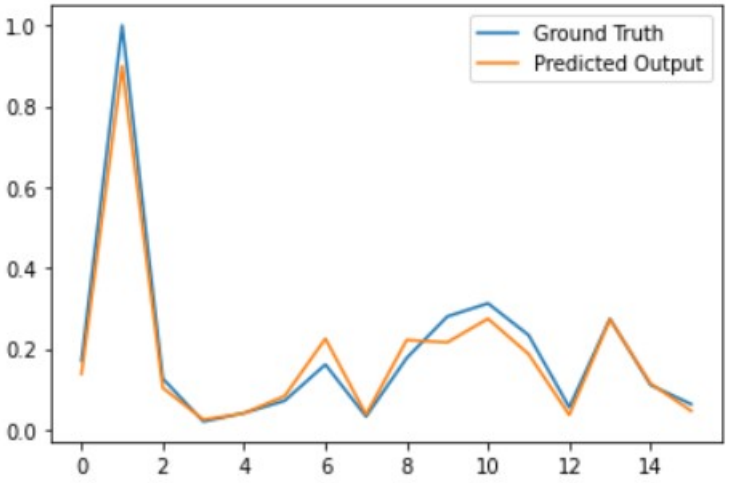

Figure 1: Predicted Output and Ground Truth using Pseudo Inverse

```
Mean Squared Error:  0.0016484915795305277
Root Mean Squared Error:  0.04060162040523171
Mean Absolute Error:  0.0297834829376438
```

2. **Gradient Descent Method** The RMS obtained is 94.6%. As we can see, this is a significant improvement over the Pseudo Inverse method.

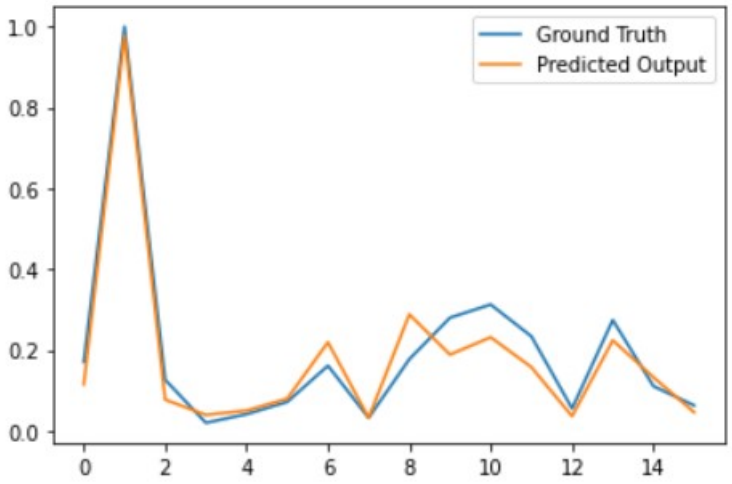

Figure 2: Predicted Output and Ground Truth using Gradient Descent

```
Mean Squared Error:  0.002906374104840904
Root Mean Squared Error:  0.05391079766466922
Mean Absolute Error:  0.043063532175202424
```

3. For change in relationship between the input and output variables, we consider that the fatalities (y) varies as square of Registered vehicles (thousands) (x1). This term is added after squaring x1. The RMS Accuracy obtained is 95.4%.

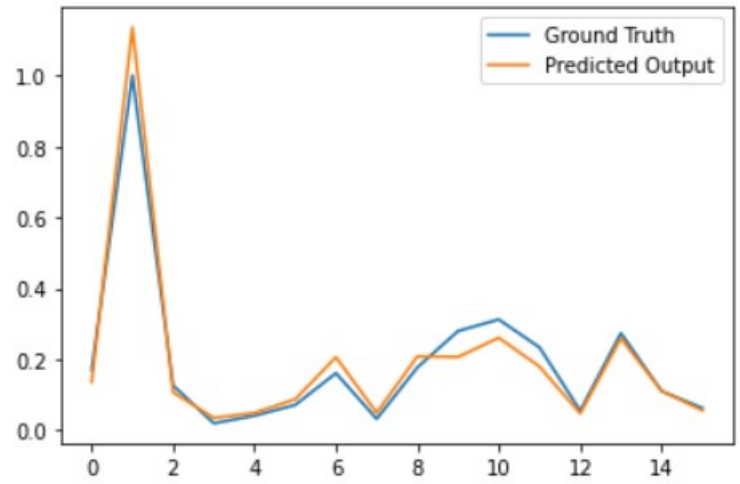

Figure 3: Predicted Output and Ground Truth After changing the relationship between input and output

```
Mean Squared Error:  0.0021891078643436262
Root Mean Squared Error:  0.04678790297014418
Mean Absolute Error:  0.033153048552636656
```

1.6 Conclusion: Linear Regression formulates a linear relationship between the independent variables as functions of dependent variables with some weights associated with each dependent variable which resembles how much the dependent variable(y) depends on the independent variable(xn). The model then tries to find the optimum value of these weights so as to minimize the error or loss.

This finding of optimum weights can be done by using two methods:

- Pseudo Inverse
- Gradient Descent

The accuracy of pseudo inverse is better than the gradient descent as gradient descent is approximation based algorithm while pseudo inverse calculates the weights precisely.

1.7 Appendices: The code for linear regression is given below:

```
#Installing Necessary Libraries

!pip install mat4py
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from mat4py import loadmat
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, mean_absolute_error

#Function to Read Data File

def BT19ECE089_dataset_div_shuffle(dir_path,split_ratio):
    #CSV XLS MAT
    if(dir_path.endswith('.csv')):
        data = pd.read_csv(dir_path)
    elif(dir_path.endswith('.xls')):
        data = pd.read_excel(dir_path)
    elif(dir_path.endswith('.mat')):
        data = loadmat(dir_path)
    else:
        print("Invalid datatype. Select .csv or .xls or .mat file")

```

```
    train_data , test_data = train_test_split(data,train_size = split_ratio,shuffle = True)

    return train_data , test_data

#Arranging the data frame to get dependent(y) and independent variables(x)

data = loadmat('/content/Matlab_accidents.mat')

data_accidents = data['accidents']
#datax = data_accidents[0][2]
'statelabel', 'hwydata', 'hwyheaders'
df = pd.DataFrame(data_accidents['hwydata'],columns = data_accidents['hwyheaders'])
states = [y[0] for y in data_accidents["statelabel"]]
df.insert(loc = 0,column = 'States',value = states)
# df.head()
df = pd.DataFrame([df['Traffic fatalities'],df['Licensed drivers (thousands)'],df['Registered vehicles (thousands)'],df['Vehicle-miles traveled (millions)']]).transpose()

#Normalizing and Splitting the data into train and test

for column in df:
  df[column] = df[column]/np.amax(df[column])
train , test = train_test_split(df,train_size = 0.7,shuffle = True)

#Creating theta matrix and y for training and testing

y_train = np.array(train.iloc[:, 0])
y_train = y_train.reshape([35,1])
x_train = np.array(train.iloc[:, 1:])
on_train = np.ones([train.shape[0],1])
theta_train = np.hstack((on_train,x_train))

y_test = np.array(test.iloc[:, 0])
x_test = np.array(test.iloc[:, 1:])
# x_test = x_test /  np.amax(x_test)
# y_test = y_test /  np.amax(y_test)
on_test = np.ones([test.shape[0],1])
y_test = y_test.reshape([y_test.shape[0],1])
theta_test = np.hstack((on_test,x_test))

#Calculating optimum weights (using pseudo inverse) and predicting output

weight_opt = np.matmul(np.linalg.pinv(theta_train),y_train)
```

```
# weight_opt.shape
y_pred = np.matmul(theta_test,weight_opt)

#Plotting ground truth and predicted outputs

plt.plot(y_test)
plt.plot(y_pred)

#Calculating mean squared error , root mean squared error and ...
    mean absolute error

mse = mean_squared_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared = False)
mae = mean_absolute_error(y_test, y_pred)
print(mse,rmse,mae)

#Finding optimum weights using Gradiend Descent

#Randomly initialising weights

grad_weights = np.random.randn(len(theta_train[0]), 1)
# grad_weights.shape

#Implementing Gradient Descent and predicting output

iterations = 1000
learning_rate = 0.03

for i in range(iterations):
  Δ_e = ...
      np.matmul(theta_train.transpose(),(np.matmul(theta_train,grad_weights)-y_train
  grad_weights = grad_weights - learning_rate * Δ_e
ygrad_pred = np.matmul(theta_test,grad_weights)

#Plotting ground truth and predicted outputs

plt.plot(y_test)
plt.plot(ygrad_pred)

#Calculating mean squared error , root mean squared error and ...
    mean absolute error

mse_g = mean_squared_error(y_test, ygrad_pred)
rmse_g = mean_squared_error(y_test, ygrad_pred, squared = False)
mae_g = mean_absolute_error(y_test, ygrad_pred)
print(mse_g,rmse_g,mae_g)

#Changing relationship between input and output variables

```

```
112 #Recalculating Theta matrix
113
114 theta_train_dash = ...
       np.hstack((on_train,np.reshape(np.square(x_train.T[0]).T,(35,1)),x_train))
115 theta_test_dash = ...
       np.hstack((on_test,np.reshape(np.square(x_test.T[0]).T,(16,1)),x_test))
116 # theta_train_dash.shape
117
118 #Calculating optimum weights (using pseudo inverse) and ...
       predicting output
119
120 weight_opt_dash = ...
       np.matmul(np.linalg.pinv(theta_train_dash),y_train)
121 # weight_opt.shape
122 y_pred_dash = np.matmul(theta_test_dash,weight_opt_dash)
123
124 #Plotting ground truth and predicted outputs
125
126 plt.plot(y_test)
127 plt.plot(y_pred_dash)
128
129 #Calculating mean squared error , root mean squared error and ...
       mean absolute error
130
131 mse_dash = mean_squared_error(y_test, y_pred_dash)
132 rmse_dash = mean_squared_error(y_test, y_pred_dash, squared = False)
133 mae_dash = mean_absolute_error(y_test, y_pred_dash)
134 print(mse_dash,rmse_dash,mae_dash)
```