Linear Regression with Pseudo Inverse and Gradient Descent

Maddipatla Vignesh Srinivas (BT19ECE065)

August 16, 2022

Abstract

Linear Regression is the basic and one of the most important concepts in the field of Machine Learning. The data needs to be processed neatly before applying this algorithm on it. Linear regression is commonly used to predict the value of the Y (output variate) using any value of X (input variate). In this paper, we perform Linear Regression to predict 'number of Fatalities per 100k registered vehicles' using 'Number of Registered Vehicles'. We perform two methods to find the Optimal Weights. One is Pseudo Inverse method and Other is Gradient Descent Method.

1 Introduction

Predicting a Variate using another Variate is one of the most fundamental tasks in Machine Learning. This type of task is commonly known as Regression. Linear Regression is one of the most basic and fundamental Algorithms used to perform a regression Analysis. In this paper, we perform Linear Regression Analysis on MATLAB Accident data set to predict 'Number of Fatalities per 100k registered vehicles' using 'Number of Registered Vehicles'. We use two most important Algorithms to find the Optimal Weights. One is Pseudo Inverse Method and Other is Gradient Descent Method. Further, we will be predicting 'Traffic fatalities' using 'Total Population', 'Licensed drivers' and 'Registered Vehicles'.

2 Methods

Finding Optimal Weights is the most important task in Machine Learning. To find Optimal Weights in Linear Regression, we will be using two methods in this experiment. One is Pseudo Inverse Method and other is Gradient Descent Method.

Linear Regression using Pseudo Inverse Method

In this method, we will be using Pseudo Inverse Method. We will divide our dataset into Train and Test sets. For training, We will first be taking the "Number of Registered Vehicles data" as our input data. It will be a matrix with size '(length(data)) X 2'. Then, we will perform Pseudo Inverse operation on the Input data $((\mathrm{A}^T.A)^{-1}.A^T)$.

After finding this pseudo Inverse, we will multiply this with the Y-variate (i.e, with 'Number of Fatalities per 100k registered vehicles' (train labels)). This will result in our Optimal Weights. So now, we have our optimal weights. This will be an array of size 2x1. The first element of optimal weight matrix is 'm' and the second element is 'c'.

For testing and checking the error, we take TestSet. We perform $\hat{\mathrm{Y}}= \mathrm{mX} + \mathrm{c}$ operation on TestSet. X is X-variate (Number of Registered Vehicles data) of TestSet. We will get predicted values of Y-variate (which is $\hat{\mathrm{Y}}$ here) with respect to X.

Now, we take Mean Squared Error between Y (original Y-variate of TestSet) and $\hat{\mathrm{Y}}$ (predicted Y-variate).

Linear Regression using Gradient Descent Method

We will be performing same task i.e, predicting 'Number of Fatalities per 100k registered vehicles' using 'Number of Registered Vehicles'. But this time, to find optimal weights, we don't use Pseudo Inverse method. Instead, we will be using famous Gradient Descent Algorithm.

In our task, we are predicting one Y-variate using only one X-variate. So, we can directly find gradients of m and c. We find Derivative of $\hat{Y}$ (predicted Y) with respect to m and c which are named as Dm and Dc. We will subtract these derivatives from m and c and will update m and c values. This will be ran for multiple epochs with a definite learning rate

Linear Regression with multiple Independent Variables (multiple X variates)

We will perform the same experiment which we performed in Pseudo Inverse method. The change here is that, we will use multiple independent variables (X-variates) instead of only one X-variate. We predicted 'Traffic fatalities' using 'Total Population', 'Licensed drivers' and 'Registered Vehicles'.

3 Results

3.1 Pseudo Inverse with one X and Y variate

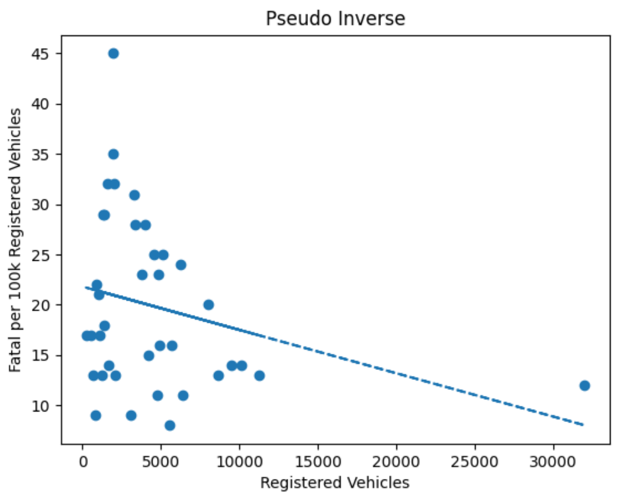


Figure 1: [Registered Vehicles] vs [Fatalities per 100k registered vehicles] using Pseudo Inverse

3.2 Gradient Descent with one X and Y variate

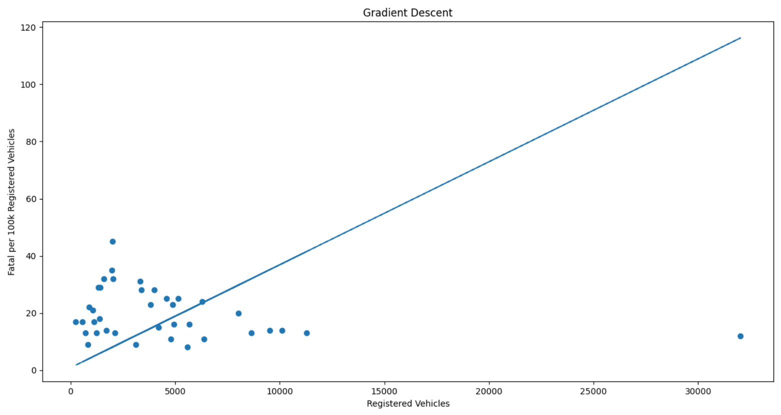
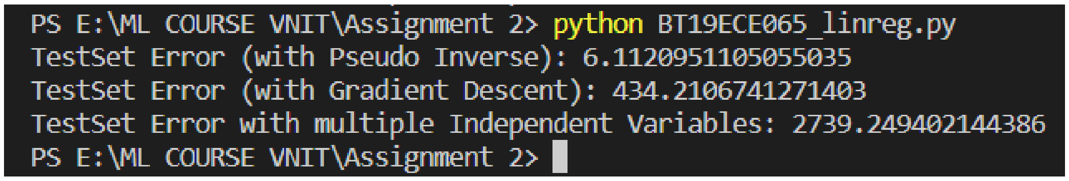


Figure 2: [Registered Vehicles] vs [Fatalities per 100k registered vehicles] using Gradient Descent

3.3 Error on TestSet using all three methods



```
PS E:\ML COURSE VNIT\Assignment 2> python BT19ECE065_linreg.py
TestSet Error (with Pseudo Inverse): 6.1120951105055035
TestSet Error (with Gradient Descent): 434.2106741271403
TestSet Error with multiple Independent Variables: 2739.249402144386
PS E:\ML COURSE VNIT\Assignment 2>
```

Figure 3: Error values

4 Discussions

The given dataset 'MATLAB Accidents' has a data whose correlation is too variant between any two columns. This made it difficult to analyse the data using regression. This is because, there's no common relation between many columns. The data appears to be dirty. This is why, the linear regression failed in solving this problem precisely. Most power algorithm, gradient descent, was unable to perform well due to this flaw in the dataset. The pseudo Inverse method worked well because the correlation between the columns 'Registered Vehicles' and 'Fatalities per registered vehicles' was moderate (because both are based on the quantity 'Registered Vehicles', so, there's some common relation between the two). Despite this, our analysis worked well.

5 Conclusion

Therefore, in this experiment, we have performed Linear regression and found optimal weights using two methods. One is Pseudo Inverse method and other is Gradient Descent method. Due to the difficulties in the dataset mentioned in Discussions, we couldn't get good values using gradient descent. But, in general, it is one of the best algorithms in finding optimal weights. We have also successfully implemented Multivariate Linear Regression.

6 Appendices

```
import pandas as pd
import scipy.io
import numpy as np
import random
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from BT19ECE065_Assignment_1 import BT9ECE065_dataset_div_shuffle
# The above libraries need to be installed before running the code


# Linear Regression Analysis
'''
1. Predicting [Fatalities per 100k registered vehicles] using
[Registered Vehicles] with
   Linear Regression using 'Pseudo Inverse'
'''
def pseudo_inverse(TrainSet,TestSet):
    train_x = TrainSet['Registered vehicles (thousands)'] # Train data
    train_y = TrainSet['Fatalities per 100K registered vehicles'] # Train outputs

    # Y = m*theta + c
    # Input (theta)
    theta = np.ones((len(train_x),2),dtype=np.long)
    theta[:,0] = train_x

    # Outputs (Y)
    Y = np.zeros((len(train_y),1),dtype = np.uint8)
    Y[:,0] = train_y

    # Optimal Weights (initialised with zeros)
    W = np.zeros((2,1))

    # Calculation of Pseudo Inverse
    W = np.matmul(np.matmul(np.linalg.inv(np.matmul(theta.T, theta)),theta.T),Y)

    # y = mX + c
    m = W[0]
    c = W[1]
    Y_hat = m*theta[:,0] + c


    ## Testing and error
    test_x = TestSet['Registered vehicles (thousands)']
    test_y = TestSet['Fatalities per 100K registered vehicles']
    theta_test = np.ones((len(test_x),2),dtype=np.long)
    theta_test[:,0] = test_x
    Y_test = np.zeros((len(test_y),1),dtype = np.uint8)
    Y_test[:,0] = test_y # Original Labels

    '''
    Y_hat is the predicted output (whereas Y is the original output).
    Using m & c values from training
    '''
    Y_hat_test = m*theta_test[:,0] + c
    error = np.square(np.subtract(Y_test[0],Y_hat_test[0])).mean()

    print("TestSet Error (with Pseudo Inverse):",error)

    plt.scatter(theta[:,0],Y)
    plt.plot(theta[:,0],Y_hat,"--")
    plt.xlabel('Registered Vehicles')
    plt.ylabel('Fatal per 100k Registered Vehicles')
    plt.title('Pseudo Inverse')
    plt.show()


'''
2. Predicting [Fatalities per 100k registered vehicles] using [Registered Vehicles]
   Linear Regression with 'Gradient Descent'
'''
def grad_desc(TrainSet,TestSet):

    train_x = TrainSet['Registered vehicles (thousands)'] # Train data
    train_y = TrainSet['Fatalities per 100K registered vehicles'] # Train data outpu

    # Input (theta)
```

```
theta = np.ones((len(train_x),2),dtype=np.long)
theta[:,0] = train_x

# Outputs (Y)
Y = np.zeros((len(train_y),1),dtype = np.uint8)
Y[:,0] = train_y

# Hyperparameters
lr = 1e-10
epochs = 5000
n = len(theta[:,0])

# Initialising m & c with random values
m = random.random()
c = random.random()

'''
We assume loss function is MSE => (y-y_)^2/n.
So, if we differentiate wrt m & c, we get function that are used in below for lo
'''
for i in range(epochs):
    Y_hat = m*theta[:,0] + c  # The current predicted value of Y
    mat_mul = np.matmul(theta[:,0],(Y - Y_hat))

    #finding gradients
    Dm = (-2/n)*np.sum(mat_mul) # derivative (with 'm')
    Dc = (-2/n)*np.sum(Y - Y_hat)  # Derivative (with 'c')

    # Updating the values
    m = m - (lr*Dm)  # Update m
    c = c - (lr*Dc)  # Update c



## Testing and error
test_x = TestSet['Registered vehicles (thousands)']
test_y = TestSet['Fatalities per 100K registered vehicles']
theta_test = np.ones((len(test_x),2),dtype=np.long)
theta_test[:,0] = test_x
Y_test = np.zeros((len(test_y),1),dtype = np.uint8)
Y_test[:,0] = test_y # Original Labels

# Y_hat_test is predicted, Y_test is original
Y_hat_test = m*theta_test[:,0] + c
error = np.square(np.subtract(Y_test[0],Y_hat_test[0])).mean()

print("TestSet Error (with Gradient Descent):",error)

plt.scatter(theta[:,0],Y)
plt.plot(theta[:,0],Y_hat,"--")
plt.xlabel('Registered Vehicles')
plt.ylabel('Fatal per 100k Registered Vehicles')
plt.title('Gradient Descent')
plt.show()
```

```
'''
3. Solving a Problem with multiple Independent Variables with Linear Regression
'''
def pseu_inv_multiple(TrainSet,TestSet):
    '''
    In this, we will use [Total Population], [Licensed drivers], [Registered Vehicle
    will predict [Traffic fatalities]
    '''

    train_x1 = TrainSet['Total Population']
    train_x2 = TrainSet['Licensed drivers (thousands)']
    train_x3 = TrainSet['Registered vehicles (thousands)']
    train_y1 = TrainSet['Traffic fatalities']

    # Y = theta*W
    theta = np.ones((len(train_x1),4),dtype=np.long)
    theta[:,0] = train_x1
    theta[:,1] = train_x2
    theta[:,2] = train_x3

    Y = np.zeros((len(train_y1),1),dtype = np.uint8)
    Y[:,0] = train_y1
    W_opt= np.zeros((4,1))


    W_opt = np.matmul(np.matmul(np.linalg.inv(np.matmul(theta.T, theta)),theta.T),Y)

    Y_hat = W_opt[0]*train_x1 + W_opt[1]*train_x2 + W_opt[2]*train_x3 + W_opt[3]


    # Testing and Error
    test_x1 = TestSet['Total Population']
    test_x2 = TestSet['Licensed drivers (thousands)']
    test_x3 = TestSet['Registered vehicles (thousands)']
    test_y1 = TestSet['Traffic fatalities']


    # Y = theta*W
    theta_test = np.ones((len(test_x1),4),dtype=np.long)
    theta_test[:,0] = test_x1
    theta_test[:,1] = test_x2
    theta_test[:,2] = test_x3

    Y_test = np.zeros((len(test_y1),1),dtype = np.uint8)
    Y_test[:,0] = test_y1

    Y_hat_test = W_opt[0]*test_x1 + W_opt[1]*test_x2 + W_opt[2]*test_x3 + W_opt[3]

    error = np.square(np.subtract(Y_test[0],Y_hat_test)).mean()

    print("TestSet Error with multiple Independent Variables:",error)

def BT19ECE065_linreg(path = './Matlab_accidents.mat'):
    mat = scipy.io.loadmat(path)
    data = mat['accidents'] # Loading the data
    states = data[0][0][1] # US State names
    num_data = data[0][0][2] # Accident - Numerical data
    columns = data[0][0][3][0] # Column names

    cols = [] # Column Names. Storing them in list
    for i in range(0,len(columns)):
        cols.append(columns[i][0])


    '''
    Building a dictionary which stores the data that should come under a respective
    The keys of the dictionary are the column names.
    This helps in organising the dataframe clearly
    '''
    data_val = data[0][0][2]
    df_dict = {}
    for i in range(0,len(cols)):
        val = []
        for j in range(0,len(num_data)):
            val.append(num_data[j][i])
        df_dict[cols[i]] = val

    # Inserting the state names as a column to make a clear database
    df = pd.DataFrame(df_dict)
    df.insert(0, 'State', states)

    '''
    This data is converted into .csv and will be shuffled, divided into sets using t
    which was written and submitted as previous lab exercise
    '''
    path_store = './us_acc.csv'
    df.to_csv(path_store,index=False)
    ratio = 0.75
    TrainSet,TestSet = BT9ECE065_dataset_div_shuffle(path_store,ratio)

    TrainSet = shuffle(TrainSet) # TrainSet is further shuffled to improve accuracy

    pseudo_inverse(TrainSet,TestSet)
    grad_desc(TrainSet,TestSet)
    pseu_inv_multiple(TrainSet,TestSet)


'''
Give path as an arguement to the function
(or, place the dataset and .py file in same folder and uncomment the below line)
'''
#BT19ECE065_linreg('./Matlab_accidents.mat')
```