Visvesvaraya National Institute of Technology (VNIT), Nagpur

Machine Learning with Python (ECL443)

Lab Report

Submitted by :
Prajyot Jadhav (BT20ECE046)
Semester 7

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

Contents

Experiment-1

Aim: To study and implement Linear Regression using pseudo-inverse and gradient descent methods.

Abstract: Linear Regression is studied and implemented in this experiment using Pseudo-Inverse and Gradient Descent methods. The linear regression analysis predicts the value of a dependent variable based on the values of other independent variables On the MATLAB accidents dataset, the performance of linear regression using both analytical and gradient descent methods is compared. Linear regression is performed by selecting appropriate dependent and independent sets. When the results of both methods are compared, it is discovered that the gradient descent algorithm is not performing as well as expected due to some limitations. To demonstrate this, various sets of variables are chosen to test the algorithm's performance.

Introduction: In this experiment, the MATLAB dataset 'accidents.mat' is used for Linear Regression. The values from ['Fatalities per 100K licenced drivers'], ['Fatalities per 100K registered vehicles'], ['Fatalities per 100M vehicle-miles travelled'], and ['Fatalities involving high blood alcohol'] serves as independent variables to predict the value of ['Percent Alcohol-Related'] fatalities. Finally, in the final part of the experiment, the values from ['Urban Population'] and ['Rural Population'] are used as independent variables to predict ['Total Population'], as the dependent variable, in order to validate the algorithm. Since there is a linear relationship for the variables chosen in the final part, better results are obtained.

Method:

- For the first part of the lab exercise, data is processed. The dataset which is in the form of a .mat file, is loaded in the form of a dictionary with the help of the scipy library. Then this dictionary data is stored into a 'data.csv' file. This data from the csv file is then shuffled and split in a given ratio of for the training and testing datasets. The ratio used was 8:2. Thus, both these sets are completely randomized.
- Next, for implementing linear regression with the pseudo inverse method, based on the simple regression model of y = mx + c, the weight matrix is calculated using the following expression:

$$y = w.X$$

$$w = (X^T X)^{-1} X^T y$$

Using this weight value, the predicted value of y is obtained. For calculating the accuracy for the predicted values with respect to the ideal values, a tolerance of 15 percent is used.

- For the next part, for implementing linear regression, instead of the pseudo inverse method, gradient descent is used. The same dataset with the same set of independent and dependent variables is used for training the model. In order to prevent the problem of exploding gradient, the trainig set was normalized. The MSE loss was used for reducing the difference between the predicted and ideal values. Similar to the previous part, for calculating accuracy, a tolerance value of 15 percent was used. After training, with the tuned weights, the model was evaluated on the test dataset.
- Finally, a different set of dependent and independent variables are used for the raw data. Linear regression is then implemented for this set of variables with the pseudo inverse and gradient descent methods.

Results:

- The training and the testing set are obtained by splitting the original data in the ratio of 80 to 20 percent. The training set of (40,4) with the labels of (40,1) and the testing set of (11,4) with the labels of (11,1) are obtained.

```
Training data size:  (40, 4)
Training label size:  (40, 1)
Testing data size:  (11, 4)
Testing label size:  (11, 1)
```

Figure 1: Training and Testing datasets

- Linear regression is implemented with pseudo inverse method, and an accuracy of 70% is obtained for training and 81% for testing dataset.

```
Psuedo-Inverse
Accuracy on Training Set :  0.7
Accuracy on Testing Set :  0.8181818181818182
```

Figure 2: Pseudo-Inverse

- Next, when linear regression is implemented with gradient descent, with a learning rate of 0.01 for 500 iterations, accuracy of 72.5% was obtained for training and 90% for testing. Also the loss for different iterations is plotted.

```
Gradient Descent
Accuracy on Training Set :  0.725
Accuracy on Testing Set :  0.9090909090909091
```

Figure 3: Gradient Descent

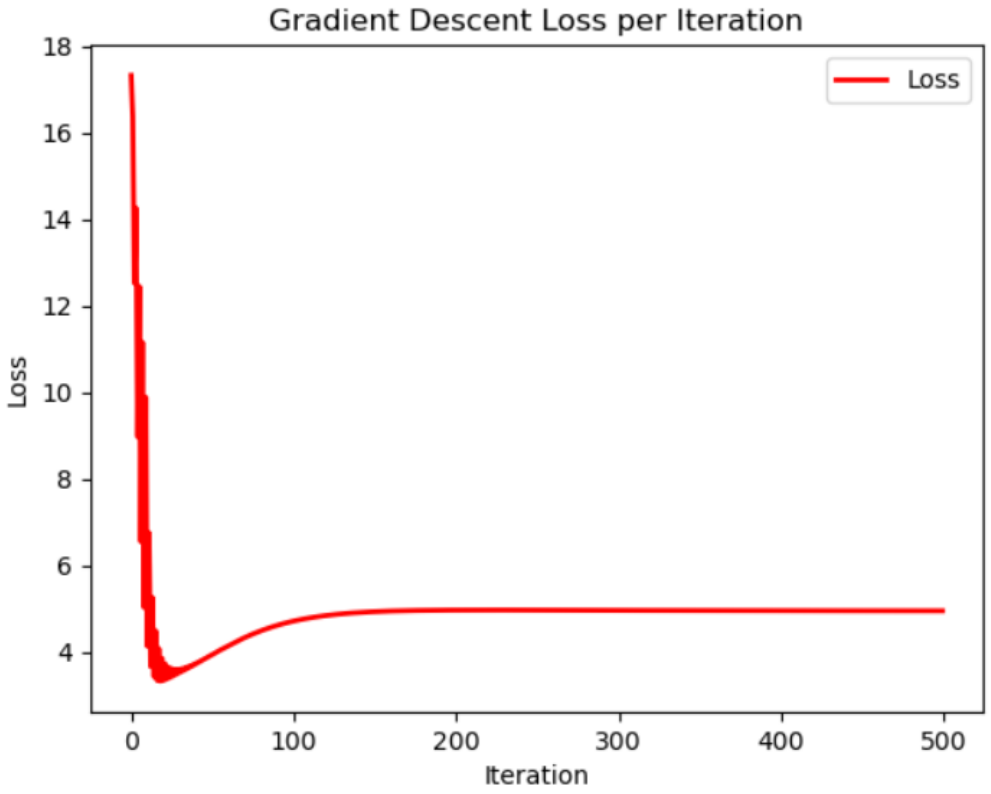

Figure 4: Gradient Descent Loss

- The above steps were repeated for a different set of independent and dependent variables, Urban Population; Rural Population and Total Population respectively. For pseudo inverse, an accuracy of 100 % is obtained for training as well as testing. Similarly, for gradient descent for training and testing 100 % accuracy is obtained and the loss is also plotted.

```
Psuedo-Inverse
Accuracy on Training Set :  1.0
Accuracy on Testing Set :  1.0
```

Figure 5: Pseudo-Inverse

```
Gradient Descent
Accuracy on Training Set :  1.0
Accuracy on Testing Set :  1.0
```

Figure 6: Gradient Descent

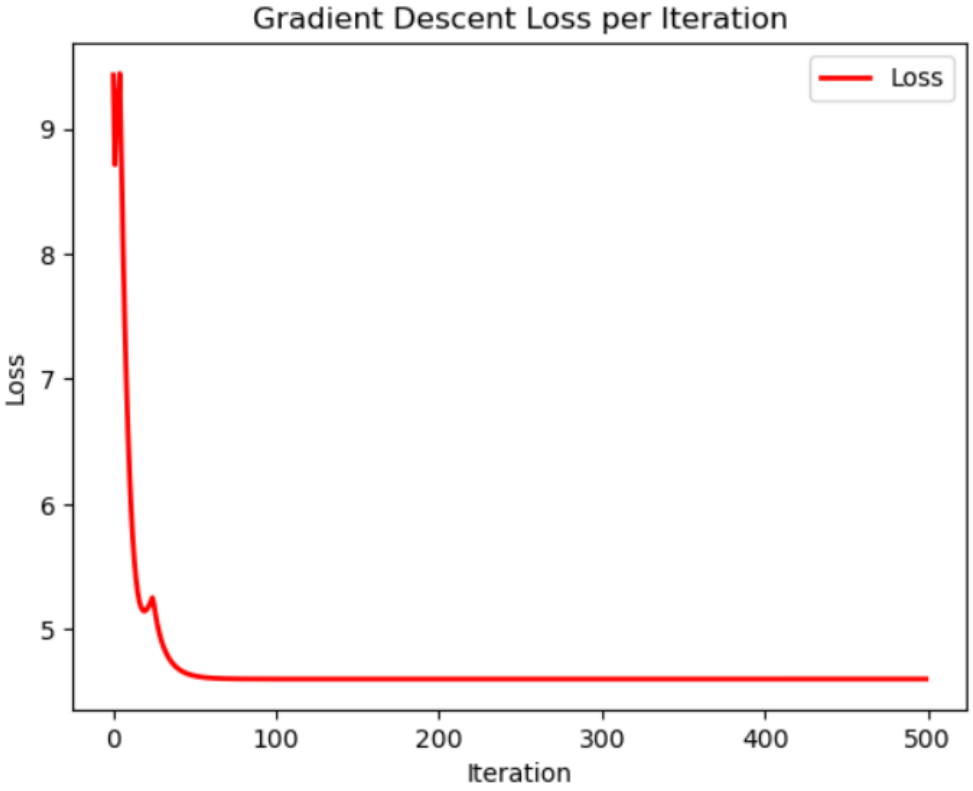

Figure 7: Gradient Descent Loss

Discussion:

- Linear Regression method is used to model a relationship between a set of dependent variables and independent variable. It is based on the assumption that the relation between the set of variables is linear. For finding this best fitting linear relation between the target variables and the independent variables, pseudo inverse and gradient descent methods are used.
- Pseudo inverse method calculates the pseudo inverse of the feature matrix to obtain the linear relationship, whereas in gradient descent by updating the

model parameters the loss(ie the error) between the predicted and ideal values is minimized iteratively.

- Using the learning rate of order 10^{-12} substantially increases the loss to infinite in just 10 iterations. This indicates that the learning rate should be very low which means the steps taken to reach the minima will be small but with large value of iterations, it is not possible to run the code as it will consume more memory. Hence, data was normalized to prevent the problem of exploding gradients in the training process.
- It was also observed that the weights should not be initialized to zero as this creates problems while training as all the weights will have the same value after each subsequent iteration. To avoid this weights are initialized to small random values.
- The reason for the low accuracy obtained can be because of non-linear relationship between the dependent and independent variables.

Conclusion: Linear Regression is implemented with the pseudo-inverse and gradient descent methods. The performance of these algorithms was analyzed for different sets of dependent and independent variables.

Appendix:

```
import scipy.io
import csv
import numpy as np
import random
import matplotlib.pyplot as plt


# Load .mat file
mat = scipy.io.loadmat(r'C:\Users\Prajyot\Desktop\accidents.mat')

# Specify the variable name to convert to CSV
variable_name1 = 'hwyheaders'
variable_name2 = 'hwydata'
variable_name3 = 'statelabel'
# Get the data from the loaded .mat file
data1 = mat[variable_name1]
data2 = mat[variable_name2]
data3 = mat[variable_name3]
combined_data = list(zip(data3.flatten(), data2.flatten()))

# Specify the CSV file name
```

```
csv_file = 'data.csv'

# Write the data to CSV
with open(csv_file, 'w', newline='') as csvfile:
    csvwriter = csv.writer(csvfile)
    #for row in data1:
    #    csvwriter.writerow(row)
    for idx, row in enumerate(data2):
        col = data3[idx]
        #print(col)
        #print(row)
    #    row.insert(0,col)
        #csvwriter.writerow(col)
        csvwriter.writerow(row)
    #for row in combined_data:
    #    csvwriter.writerow(row)
    #print(data3)


#Reading Data from .csv file
with open('data.csv', 'r') as f:
    reader = csv.reader(f)
    data = list(reader)

data_array = np.array(data,dtype=np.float32)
#print(data_array.shape)
#print(data_array)

# Function to shuffle data and produce training and test data ... with labels
def train_n_test_data(arr, x,indep_ls, dep_ls):
    dep_arr = arr[:,indep_ls]
    labels_arr = arr[:,dep_ls]
    arr_shape = arr.shape
    train = int(x*arr_shape[0])
    idx = np.random.randint(low=0, high=51, size=51, dtype=int)
    new_arr = dep_arr[idx]
    new_lbs = labels_arr[idx]
    train_arr = new_arr[0:train]
    test_arr = new_arr[train:]
    train_lb = new_lbs[0:train]
    test_lb = new_lbs[train:]
    return train_arr, train_lb, test_arr, test_lb

train_arr, train_lb, test_arr, test_lb = train_n_test_data(data_array,0.8,[8,9,10,11],[12])
print("Training data size: ",train_arr.shape)
print("Training label size: ",train_lb.shape)
print("Testing data size: ",test_arr.shape)
```

```
69 print("Testing label size: ",test_lb.shape)
70
71
72 #Function to perform Linear Regression by Pseudo-Inverse Method
73 def pseudo_inverse(train_arr, train_lb, test_arr, test_lb, tol):
74
75   y = train_lb
76   (a,b) = train_arr.shape
77   # Add a column of ones to X for the intercept term
78   X_b = np.c_[np.ones((a, 1)), train_arr]
79
80   # Calculate the coefficients using the pseudo-inverse method
81   theta = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)
82   #print(theta)
83   # Print the coefficients
84   for idx in range(len(theta)):
85     if idx==0:
86       print("Intercept:", theta[0])
87     else:
88       print(f"Slope{idx}:", theta[idx])
89
90   # Make predictions
91   #tol = 15
92   (c,d) = test_arr.shape
93   X_b_test = np.c_[np.ones((c,1)),test_arr]
94   y_pred = X_b_test.dot(theta)
95
96   #(e,f) = test_lb.shape
97   ls = []
98   for idx in range(c):
99       if abs(test_lb[idx]-y_pred[idx])≤test_lb[idx]*tol/100:
100        ls.append(1)
101      else:
102        ls.append(0)
103
104
105  test_acc = sum(ls)/len(ls)
106  #print(test_acc)
107
108
109
110  y_pred_train = X_b.dot(theta)
111  ls_train = []
112
113  for idx in range(a):
114      if abs(train_lb[idx]-y_pred_train[idx])≤train_lb[idx]*tol/100:
115        ls_train.append(1)
116      else:
117        ls_train.append(0)
```

```
118
119   train_acc = sum(ls_train)/len(ls_train)
120   #print(accc)
121   return train_acc, test_acc
122
123
124 tol = 15
125 train_acc, test_acc = pseudo_inverse(train_arr, train_lb, ...
        test_arr, test_lb, tol)
126
127 print("Psuedo-Inverse")
128 print("Accuracy on Training Set : ", train_acc)
129 print("Accuracy on Testing Set : ", test_acc)
130
131
132 # Plot the results
133 #plt.scatter(X, y, color='blue', label='Actual')
134 #plt.plot(X, y_pred, color='red', linewidth=2, label='Predicted')
135 #plt.xlabel('X')
136 #plt.ylabel('y')
137 #plt.title('Linear Regression using Pseudo Inverse Method')
138 #plt.legend()
139 #plt.show()
140
141
142 def grad_des(train_arr, train_lb, test_arr, test_lb, tol):
143
144   # Add a column of ones to X for the intercept term
145   y = train_lb
146   (a,b) = train_arr.shape
147   # Add a column of ones to X for the intercept term
148   X_b = np.c_[np.ones((a, 1)), train_arr]
149   #Normalizing every independent variable columns to prevent ...
        exploding gradient
150   for idx in range(1,b+1):
151     X_b[:,idx] = X_b[:,idx]/np.max(X_b[:,idx])
152
153
154   # Hyperparameters for gradient descent
155   learning_rate = 0.01
156   n_iterations = 500
157   loss_ls = []
158   # Initialize coefficients randomly
159   theta = np.random.randn(b+1, 1)
160
161   # Gradient Descent
162   for iteration in range(n_iterations):
163       gradients = -2 * X_b.T.dot(y - X_b.dot(theta))
164       loss = np.square(y - X_b.dot(theta))
```

```
165         norm_loss = np.sum(loss)/np.max(loss)
166         loss_ls.append(norm_loss)
167         theta -= learning_rate * gradients
168 
169     # Print the coefficients
170     for idx in range(len(theta)):
171       if idx==0:
172         print("Intercept:", theta[0][0])
173       else:
174         print(f"Slope{idx}:", theta[idx][0])
175 
176     # Make predictions
177     y_pred = X_b.dot(theta)
178 
179     (c,d) = test_arr.shape
180     X_b_test = np.c_[np.ones((c,1)),test_arr]
181     #Normalizing every independent variable columns to prevent ...
          exploding gradient
182     for idx in range(1,d+1):
183       X_b_test[:,idx] = X_b_test[:,idx]/np.max(X_b_test[:,idx])
184 
185     y_pred = X_b_test.dot(theta)
186 
187     ls = []
188     for idx in range(c):
189         if abs(test_lb[idx]-y_pred[idx])≤test_lb[idx]*tol/100:
190           ls.append(1)
191         else:
192           ls.append(0)
193 
194     test_acc = sum(ls)/len(ls)
195 
196 
197 
198     y_pred_train = X_b.dot(theta)
199     ls_train = []
200 
201     for idx in range(a):
202         if abs(train_lb[idx]-y_pred_train[idx])≤train_lb[idx]*tol/100:
203             ls_train.append(1)
204         else:
205             ls_train.append(0)
206 
207     train_acc = sum(ls_train)/len(ls_train)
208 
209     return train_acc, test_acc, loss_ls
210 
211 tol = 15
212 train_acc, test_acc, loss_ls = grad_des(train_arr, train_lb, ...
```

```
    test_arr, test_lb, tol)

print("Gradient Descent")
print("Accuracy on Training Set : ", train_acc)
print("Accuracy on Testing Set : ", test_acc)

#Plot the loss
#plt.scatter(X, y, color='blue', label='Actual')
X = np.arange(len(loss_ls))
plt.plot(X, loss_ls, color='red', linewidth=2, label='Loss')
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.title('Gradient Descent Loss per Iteration')
plt.legend()
plt.show()

train_arr, train_lb, test_arr, test_lb = ...
    train_n_test_data(data_array,0.8,[14,15],[13])
print("Training data size: ",train_arr.shape)
print("Training label size: ",train_lb.shape)
print("Testing data size: ",test_arr.shape)
print("Testing label size: ",test_lb.shape)
print()
print()


tol = 15
train_acc, test_acc = pseudo_inverse(train_arr, train_lb, ...
    test_arr, test_lb, tol)

print("Psuedo-Inverse")
print("Accuracy on Training Set : ", train_acc)
print("Accuracy on Testing Set : ", test_acc)
print()
print()


tol = 15
train_lb = train_lb/np.max(train_lb)
test_lb = test_lb/np.max(test_lb)
train_acc, test_acc, loss_ls = grad_des(train_arr, train_lb, ...
    test_arr, test_lb, tol)

print("Gradient Descent")
print("Accuracy on Training Set : ", train_acc)
print("Accuracy on Testing Set : ", test_acc)

#Plot the loss
#plt.scatter(X, y, color='blue', label='Actual')
```

```
X = np.arange(len(loss_ls))
plt.plot(X, loss_ls, color='red', linewidth=2, label='Loss')
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.title('Gradient Descent Loss per Iteration')
plt.legend()
plt.show()
```