Travis Munyer

Linear Regression Discussion

September 3, 2021

1. Train utils file

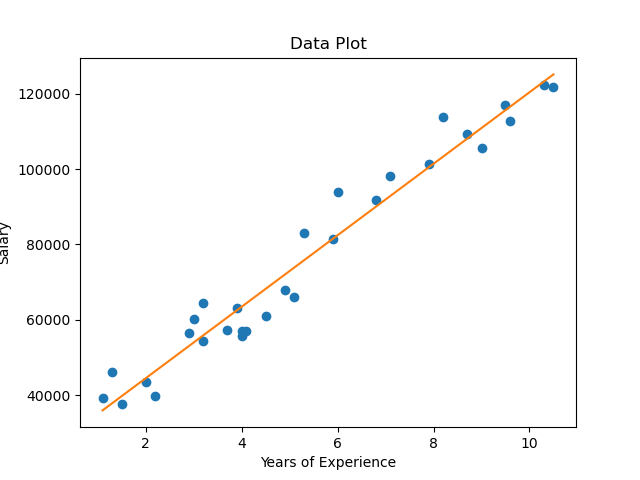
import random  
import numpy as np  
  
*# This function returns a scalar representing the quality of input weight and bias, lower is better.*def loss\_funct(weight, bias, true, x):  
 return pow(true - predict(x, weight, bias), 2)  
  
*# Loss function that allows a pre-generated prediction as input.*def loss\_funct\_pred\_provided(true, prediction):  
 return pow((true - prediction), 2)  
  
*# Performs a prediction using supplied x, weight, and bias.*def predict(x, weight, bias):  
 return weight\*x + bias  
  
*# Initialized training and returns the minimized weight and bias.*def linear\_train\_init(training\_data, learning\_rate):  
  
 *# Initialize (somewhat randomly) the weight and bias.* random.seed(123)  
 weight\_0 = random.uniform(0, np.mean(training\_data[1], 0))  
 bias\_0 = random.uniform(0, np.amax(training\_data[1], 0))  
 i = 1  
 loss\_hist = []  
  
 *# Perform initial training using the random weight and bias.* loss, bias\_1, weight\_1 = linear\_train(training\_data, weight\_0, bias\_0, learning\_rate)  
 loss\_hist.append(loss)  
 print("Loss - epoch " + "0" + ": " + str(loss\_hist[0]))  
  
 *# This loop will continue to train until the loss is the same 4 times in a row.* while True:  
 loss, bias\_1, weight\_1 = linear\_train(training\_data, weight\_1, bias\_1, learning\_rate)  
 loss\_hist.append(loss)  
 print("Loss - epoch " + str(i) + ": " + str(loss\_hist[i]))  
 if i > 3:  
 if loss\_hist[i] == loss\_hist[i-1] and loss\_hist[i-1] == loss\_hist[i-2]:  
 break  
  
 i += 1  
  
 print()  
  
 return loss\_hist, bias\_1, weight\_1  
  
*# Utility function that performs the bulk of the training.*def linear\_train(training\_data, weight, bias, learning\_rate):  
 loss = 0  
 bias\_gradient = 0  
 weight\_gradient = 0  
  
 *# Train for each value in the training data (an epoch).* for entry in training\_data:  
 x\_value = entry[0]  
 true = entry[1]  
  
 *# Compute the loss and gradient for each data point in the training set.* loss += loss\_funct(weight, bias, true, x\_value)  
 bias\_grad\_new, weight\_grad\_new = gradient\_descent(weight, bias, true, x\_value)  
 bias\_gradient += bias\_grad\_new  
 weight\_gradient += weight\_grad\_new  
  
 *# Adjust the weight and bias according to the gradient computed during training.* new\_bias = bias - learning\_rate \* bias\_gradient  
 new\_weight = weight - learning\_rate \* weight\_gradient  
 return loss, new\_bias, new\_weight  
  
*# Returns the bias gradient and the weight gradient in this respective order.*def gradient\_descent(weight, bias, true, x):  
 return (2\*(true - predict(x, weight, bias))\*(-1)), (2\*(true - predict(x, weight, bias))\*(-x))

**evaluation utils file**

import train\_utils  
import matplotlib.pyplot as plt  
  
*# Computer validation loss.*def validate(validation\_data, weight, bias):  
 total\_loss = 0  
  
 for entry in validation\_data:  
 x\_value = entry[0]  
 true = entry[1]  
  
 *# Generate predictions and compute loss.* prediction = train\_utils.predict(x\_value, weight, bias)  
 loss = train\_utils.loss\_funct\_pred\_provided(true, prediction)  
 total\_loss += loss  
  
 *# Show the prediction and true values.* print("Prediction: " + str(prediction))  
 print("True: " + str(true))  
 print()  
  
 print("Used weight: " + str(weight))  
 print("Used bias: " + str(bias))  
 print("Validation loss: " + str(loss))  
  
*# Plot the loss.*def plot\_loss(loss):  
 plt.figure().clear()  
 plt.plot(loss)  
 plt.title("Training Loss")  
 plt.ylabel('loss')  
 plt.xlabel('epoch')  
 plt.savefig("loss.png")  
  
*# Plot regression line based on input weight and bias.*def plot\_reg\_line(x, y, label\_x, label\_y, plot\_title, filename, weight, bias):  
 plt.figure().clear()  
 plt.plot(x, y, 'o')  
 plt.plot(x, weight\*x+bias)  
 plt.xlabel(label\_x)  
 plt.ylabel(label\_y)  
 plt.title(plot\_title)  
 plt.savefig(filename)

**main file**

*# Assignment One for UNO CSCI 3470 Machine Learning by Travis Munyer.  
# This code is for the linear regression portion of assignment one.  
# For a description of this assignment, see Assignment\_1.pdf.*import numpy as np  
import pandas as pd  
import train\_utils  
import evaluation\_utils  
from sklearn.model\_selection import train\_test\_split  
  
data = pd.read\_csv('AssignmentOneData.csv').to\_numpy(dtype='float32')  
test\_size = 0.2  
learning\_rate = 0.0001  
np.set\_printoptions(suppress=True)  
print(data)  
*# Split the dataset into train and validate, then train the model.*train, validate = train\_test\_split(data, test\_size=test\_size, random\_state=30, shuffle=True)  
loss, final\_bias, final\_weight = train\_utils.linear\_train\_init(train, learning\_rate)  
  
*# Evaluate the performance of the model on the validation dataset.*evaluation\_utils.validate(validate, final\_weight, final\_bias)  
  
*# Plot loss and the regression line.*evaluation\_utils.plot\_loss(loss)  
evaluation\_utils.plot\_reg\_line(data.transpose()[0], data.transpose()[1], "Years of Experience", "Salary", "Data Plot", "data.png", final\_weight, final\_bias)

1. The code can be ran to get a new data plot with regression line (saved as a file), output weight values, and the results of predictions on the testing set. However, here is the requested information for redundancy.
   1. Optimized Weight: 9481.03
   2. Optimized Bias: 25566.43
   3. Data plot with regression line:
   4. Additionally, here is a loss plot: 
2. It was a great learning experience to implement a linear regression model from scratch, and I am glad I did the extra work to do so. Before, I had a decent understanding of how the weights and biases are optimized but implementing it myself expanded my understanding of the optimization process.