**ENME 808 – HW 1**

**Adrienne Rudolph**

**02/16/2024**

**Question 1**

# Author: Adrienne Rudolph

# Class: ENME 808

# HW1 Problem 1

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import root\_mean\_squared\_error as rootMSE

#Read in the data

mlr05 = pd.read\_excel("mlr05.xlsx")

#Prepare the data

X\_varb = mlr05.iloc[:, 1:]  #X variables are the X2, X3, .. X6 predictors

Y\_varb = mlr05.iloc[:, 0]   #Y variable is the sales data, X1

#Separate Training and Testing Sets

x\_train = X\_varb[:20] #training set are X\_varb columns, first 20 rows

y\_train = Y\_varb[:20] #training set is 'sales data' column, first 20 rows

x\_test = X\_varb[20:] #test set are all columns, last 7 rows

y\_test = Y\_varb[20:] #test set is 'sales data' column, last 7 rows

#Select the linear regression model

dataModel = LinearRegression()

#Train the model

dataModel.fit(x\_train, y\_train)

#Make a prediction for the sales data

prediction = dataModel.predict(x\_test)

#Evaluate for error in actual and prediction

rmse = rootMSE(y\_test, prediction)

print(rmse)

#Reshape and print prediction and

prediction = prediction.reshape(-1,1) #Visually appealing column vector for printing purpose

print(prediction)

#Plot test and prediction, and show line of perfect prediction

plt.scatter(y\_test, prediction, label='Predicted vs Actual')

plt.xlabel('Actual Sales Data')

plt.ylabel("Predictied Sales Data")

plt.title('Actual vs Predicted Sales Data Values')

plt.plot([min(y\_test), max(y\_test)], [min(y\_test), max(y\_test)], color='purple', linestyle='--')

plt.legend()

plt.grid(True)

plt.show()



The root mean squared error between the actual and predicted data is 59.8, which doesn’t seem to be a very good outcome. This means the predictor doesn’t model the data very well, and indicates the data likely requires a nonlinear predictor. The purple dotted line shown in the plot above displays a perfect match between predicted and actual sales data.

**Question 4**

import numpy as np

import matplotlib.pyplot as plt

#pick a function x(t), y(t)

def targetFunc(x):

    return 3\*x + (1/2)

#Produce the randomly generated data

dataPts = np.random.uniform(-10, 10, size=(10100, 2))

#Separate Train and Test Data

trainPts = dataPts[:100, :]

testPts = dataPts[100:, :]

#Separate X and Y columns from Train Data

train\_feature = trainPts[:, 0]  #x data for training set

train\_label = trainPts[:, 1]    #y data for training set

#Separate X and Y columns from Test Data

test\_feature = testPts[:, 0]  #x data for test set

test\_label = testPts[:, 1]    #y data for test set

#Initialize w vector and eta (n)

w = np.zeros((3, 1))

n = 0.0001

y\_train\_label = []

#Iterate, calculate signal, and update weights

for i in range(0, 10):

    for t in range(len(trainPts)):

        y\_actual = trainPts[t,1] - targetFunc(trainPts[t,0])

        if y\_actual > 0:

            y\_actual = 1

        else:

            y\_actual = -1

        if i == 0:

            y\_train\_label.append(y\_actual)

        x\_vec = np.array([1, trainPts[t,0], trainPts[t,1]]).reshape(-1,1)

        s\_t = np.sign(np.dot(w.T, x\_vec))

        temp = y\_actual \* s\_t

        if temp <= 1:

            w += n \* (y\_actual - s\_t) \* x\_vec

w = w

w0 = w[0]

w1 = w[1]

w2 = w[2]

def output(z):

    return (-(w1/w2) \* z) - (w0/w2)

#Apply best weights to test set

predictions = []

actual = []

for a in range(len(testPts)):

    y\_actual\_test = testPts[a,1] - targetFunc(testPts[a,0])

    if y\_actual\_test > 0:

        y\_actual\_test = 1

    else:

        y\_actual\_test = -1

    actual.append(y\_actual\_test)

    x\_vec = np.array([1, test\_feature[a], test\_label[a]]).reshape(-1,1)

    s\_t = np.sign(np.dot(w.T, x\_vec))

    predictions.append(s\_t)

y\_test\_label = actual

predictions = np.array(predictions).flatten()

actual = np.array(actual).flatten()

print(np.sum(predictions==actual)/predictions.shape[0])

y\_train\_label = np.array(y\_train\_label).flatten().astype(int)

y\_test\_label = np.array(y\_test\_label).flatten().astype(int)

# Plot the training data set

z = np.linspace(-10,10)

plt.scatter(train\_feature[y\_train\_label==1], train\_label[y\_train\_label==1], color='purple',label='+1')

plt.scatter(train\_feature[y\_train\_label==-1], train\_label[y\_train\_label==-1],color='red',label='-1')

plt.plot(z, targetFunc(z), color='blue', label='Target Function')

plt.plot(z, output(z), color='green', label='Final Hypothesis')

plt.title('Target Function vs Hypothesis, n = 0.0001')

plt.xlabel('X - Train Feature Data')

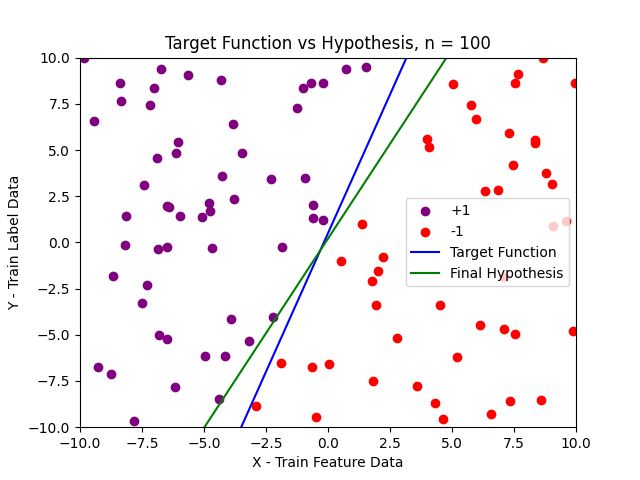
plt.ylabel('Y - Train Label Data')

plt.legend()

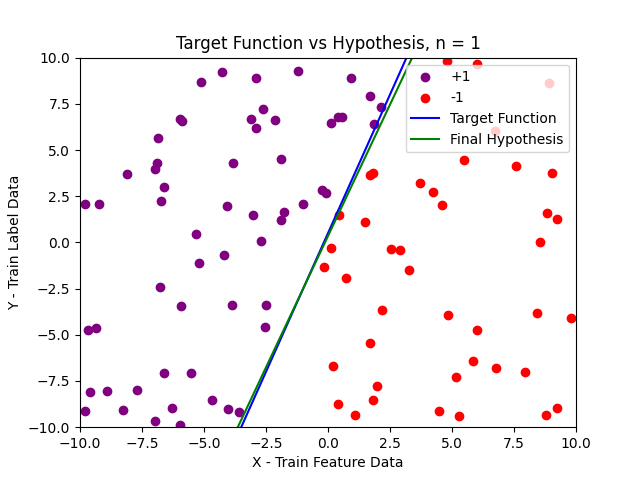
plt.xlim(-10,10)

plt.ylim(-10,10)

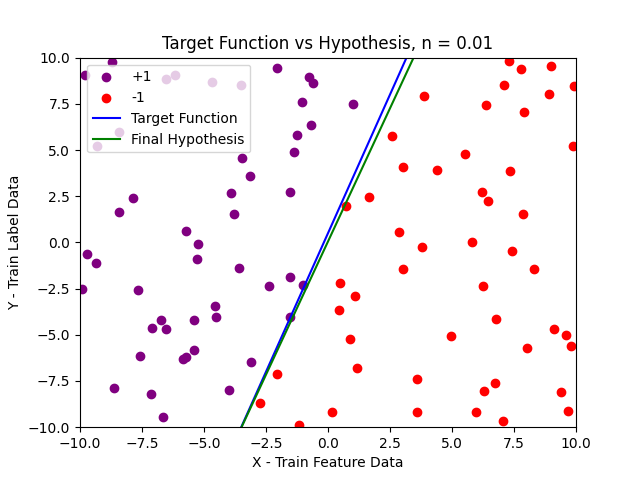
plt.show()



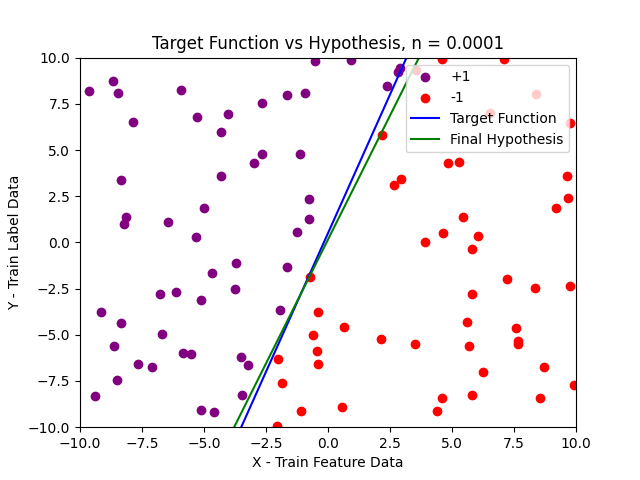
The accuracy of the hypothesis on the test set in the case of η = 100 is 95.97%.



The accuracy of the hypothesis on the test set in the case of η = 1 is 99.58%



The accuracy of the hypothesis on the test set in the case of η = 0.01 is 99.4%



The accuracy of the hypothesis on the test set in the case of η = 0.0001 is 98.96%

It is somewhat difficult to comment on just how different the results are based on the learning rate, η. The data set I generated is random, so each time I run the program with a new learning rate, the overall data set and accuracy will change a little bit. However, in general, it does appear that a smaller learning rate might help the algorithm produce a more accurate hypothesis.