ENME 808 – HW 1

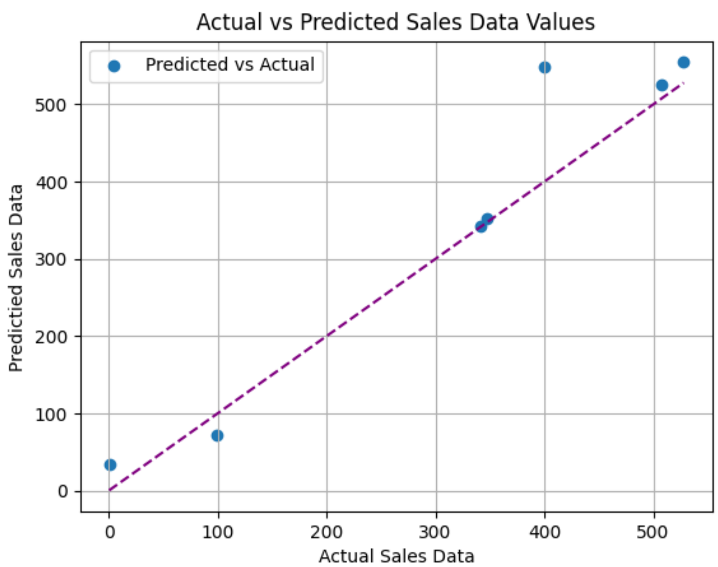
Adrienne Rudolph

02/16/2024

Question 1

```python
# 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.

Adrienne Rudolph
HW 1

Question 2

a.)

Perceptron 2 dimensions

$h(x) = \text{sign}(w^T x)$

$$w = [w_0, w_1, w_2]^T$$
$$x = [1, x_1, x_2]^T$$

$h(x) = +1$ when $w^T x > 0$

$h(x) = -1$ when $w^T x < 0$

boundary at $w^T x = 0$

so $w_0 + w_1 x_1 + w_2 x_2 = 0$

$$w_1 x_1 + w_2 x_2 = -w_0$$
$$w_2 x_2 = -w_0 - w_1 x_1$$
$$x_2 = -\frac{w_0}{w_2} - \frac{w_1}{w_2} x_1$$

$$\boxed{x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2}}$$

where $a = -\frac{w_1}{w_2}$ and $b = -\frac{w_0}{w_2}$

b.) $\omega = [1, 2, 3]^T$ $\omega_0 = 1,\ \omega_1 = 2,\ \omega_2 = 3$

$$x_2 = -\frac{2}{3}x_1 - \frac{1}{3}$$

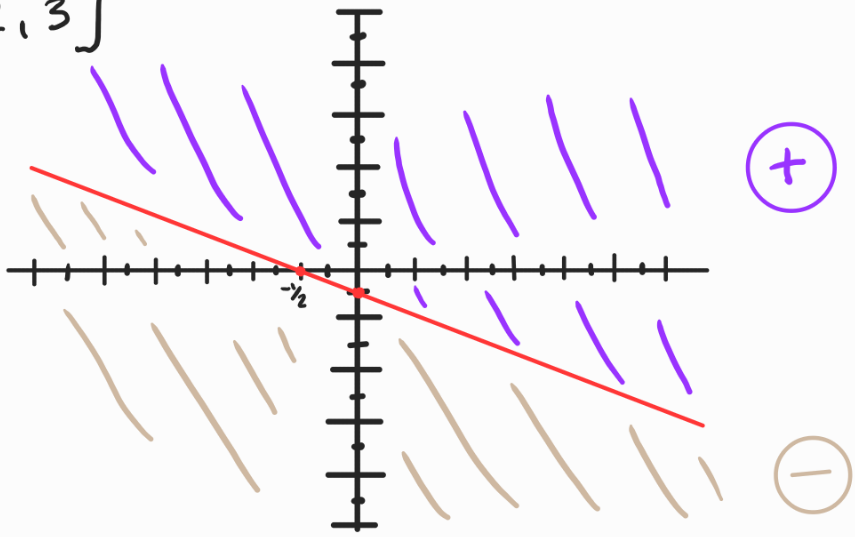
$\omega = [-1, -2, -3]^T$ $\omega_0 = -1,\ \omega_1 = -2,\ \omega_2 = -3$

$$x_2 = -\left(\frac{-2}{-3}\right)x_1 - \left(\frac{-1}{-3}\right)$$

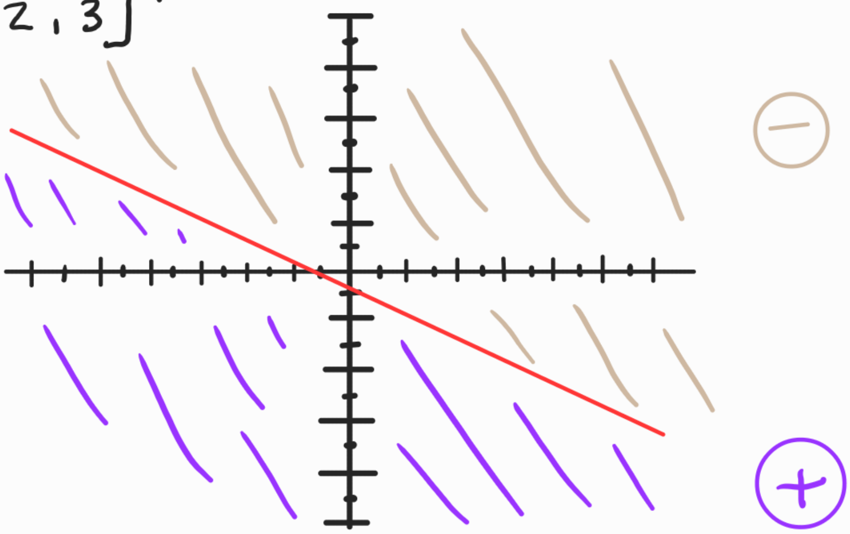
$$x_2 = -\frac{2}{3}x_1 - \frac{1}{3}$$

Both cases have the same line, but the regions change

case 1: $[1, 2, 3]^T$



case 2: $-[1, 2, 3]^T$



Question 3

- linearly separable data set
- B = minimum$\{\|w\| : \forall i \in [m],\ y_i(w^T x_i) \geq 1\}$
- $R = \max_i \|x_i\|$

$$w^{t+1} = w^t + y_i * x_i$$

optimal w^* must be equal to or greater than the min$\{\ \}$, or $>= B$.

* $w(t+1)$ will always be closer to w^* than $w(t)$

assume $\|w^*\| = 1$

$$w^{t+1} w^* = (w^t + (y_i x_i))\, w^*$$
$$= w^t w^* + \underbrace{y_i (w^* x_i)}_{\to B}$$
$$> w^t w^* + B \quad \leftarrow \text{Because minimum}$$

so after a 't' number of iterations

$$w^{t+1} \cdot w^* > tB$$

Because $w^{t+1} \cdot w^* \leq \|w^{t+1}\| \|w^*\| = \|w^{t+1}\|$

$$\|w^{t+1}\| > tB \quad \text{lower bound}$$

To get an upper bound

$\rightarrow \|w^{t+1}\|^2 = \|w^t + y_i x_i\|^2$

$$= \|w^t\|^2 + 2\|y_i x_i\|^2 + 2y_i(w^t \cdot x_i)$$
$$= \|w^t\|^2 + \|x_i\|^2 + 2y_i(w^t \cdot x_i)$$
$$\leq \|w^t\|^2 + \|x_i\|^2$$

goes away for upper bound

$\|x_i\|^2 = R^2$

$\rightarrow \leq \|w^t\|^2 + R^2$

so after t iterations: $\|w^{t+1}\|^2 \leq tR^2$ upper bound

$$t^2 B^2 < \|w^{t+1}\|^2 \leq tR^2$$

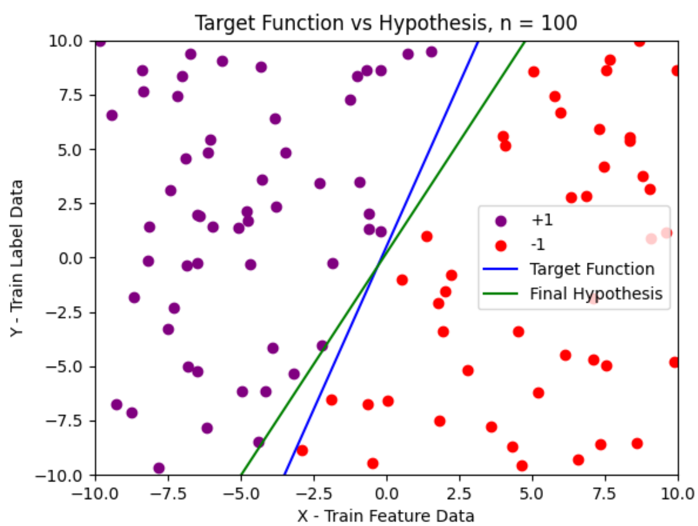
$$\boxed{t \leq \frac{R^2}{B^2} \quad \text{stops after at most}}$$

Question 4

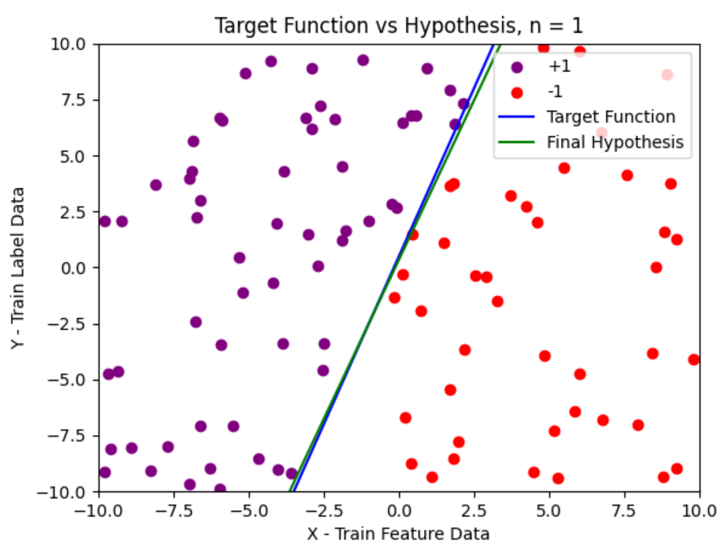
```
import numpy as np
import matplotlib.pyplot as plt


#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')
```

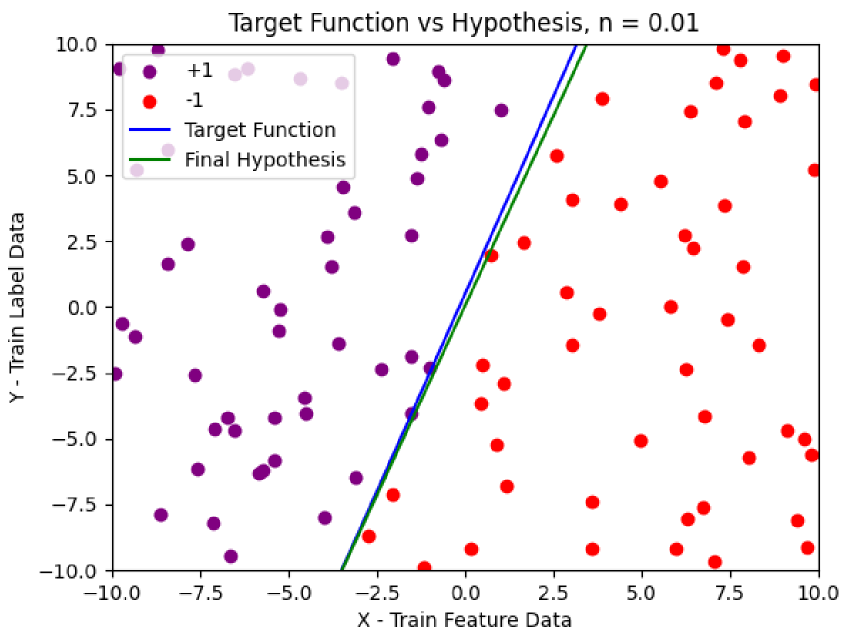
```
plt.xlabel('X - Train Feature Data')
plt.ylabel('Y - Train Label Data')
plt.legend()
plt.xlim(-10,10)
plt.ylim(-10,10)
plt.show()
```



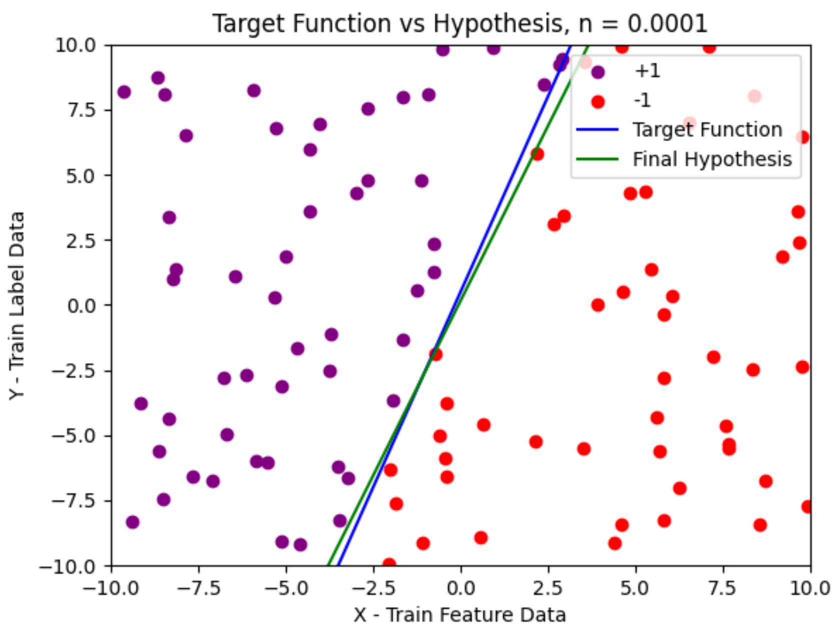
The accuracy of the hypothesis on the test set in the case of η = 100 is 95.97%.



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.

Question 5

Exercise 1.8 - If $\mu = 0.9$, what is the probability that a sample of 10 marbles will have $\nu \le 0.1$?

Hint: use binomial distribution
The answer is a very small number

$$\binom{n}{k} \mu^k (1-\mu)^{n-k}$$

$Pr(\nu \le 0.1)$ where $\nu = k/n$

$\rightarrow Pr(\nu = \frac{k}{n} \le 0.1) \rightarrow Pr(k \le 0.1 \cdot n)$

$\rightarrow Pr(k \le 1)$

so $Pr(0) + Pr(1) = \sum_{0}^{1} \binom{n}{x} p^x (1-p)^{n-x}$

where $\binom{n}{x} = \frac{n!}{x!(n-x)!}$

$$= \binom{10}{0} \mu^0 (1-\mu)^{10-0} + \binom{10}{1} \mu^1 (1-\mu)^{10-1}$$

$$= \frac{10!}{0!(10-0)!} (0.9)^0 (1-0.9)^{10-0} + \frac{10!}{1!(10-1)!} (0.9)^1 (1-0.9)^{10-1}$$

$$= 1 \times 10^{-10} + 9 \times 10^{-9}$$

$$\boxed{Pr(\nu \le 0.1) = 9.1 \times 10^{-9}}$$

Exercise 1.9

$\mu = 0.9$

sample of 10 marbles

$\nu \leq 0.1$

$\mathbb{P}[|\nu - \mu| > \epsilon] \leq 2e^{-2\epsilon^2 N}$

$\mathbb{P}[|0.1 - 0.9| > \epsilon] \leq 2e^{-2\epsilon^2 N}$

$\mathbb{P}[0.8 > \epsilon] \leq 2e^{-2\epsilon^2 N}$

so plug in 0.8 for ϵ

$\mathbb{P}[\nu \leq 0.1] \leq 2e^{-2(0.8)^2(10)}$

$\boxed{\mathbb{P}[\nu \leq 0.1] \leq 5.522 \times 10^{-6}}$

The probability of the "bad" event happening is higher coming from the Hoeffding Inequality than the binomial distribution. That means this calculation is more conservative.