Deep Learning (IST, 2021-22)

Practical 3: Linear and Logistic Regression

André Martins, Andreas Wichert, Taisiya Glushkova, Luis Sá Couto

Question 1

Consider the following training data:

$$\boldsymbol{x}^{(1)} = [-2.0], \boldsymbol{x}^{(2)} = [-1.0], \boldsymbol{x}^{(3)} = [0.0], \boldsymbol{x}^{(4)} = [2.0]$$

$$y^{(1)} = 2.0, y^{(2)} = 3.0, y^{(3)} = 1.0, y^{(4)} = -1.0.$$

1. Find the closed form solution for a linear regression that minimizes the sum of squared errors on the training data..

   Solution: First, we need to build the $n \times (d+1)$ design matrix to account for the bias parameter, where n is the number of examples and d is the original number of input features.

$$\mathbf{X} = \begin{bmatrix} 1 & -2.0 \\ 1 & -1.0 \\ 1 & 0.0 \\ 1 & 2.0 \end{bmatrix}$$

   Second, we construct a target vector:

$$\boldsymbol{y} = \begin{bmatrix} 2.0 \\ 3.0 \\ 1.0 \\ -1.0 \end{bmatrix}$$

   Now, the goal is to find the weight vector $\boldsymbol{w} = \begin{bmatrix} w_0 & w_1 \end{bmatrix}^\top$ that minimizes the sum of squared errors. We can do it using the pseudo-inverse:

$$\boldsymbol{w} = \underbrace{\left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top}_{\text{Pseudo-inverse } \mathbf{X}^+} \boldsymbol{y}$$

$$
\begin{aligned}
\boldsymbol{w} &= \left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \boldsymbol{y} \\
&= \left[\begin{pmatrix} 1 & -2.0 \\ 1 & -1.0 \\ 1 & 0.0 \\ 1 & 2.0 \end{pmatrix}^\top \begin{pmatrix} 1 & -2.0 \\ 1 & -1.0 \\ 1 & 0.0 \\ 1 & 2.0 \end{pmatrix}\right]^{-1} \begin{pmatrix} 1 & -2.0 \\ 1 & -1.0 \\ 1 & 0.0 \\ 1 & 2.0 \end{pmatrix}^\top \begin{pmatrix} 2.0 \\ 3.0 \\ 1.0 \\ -1.0 \end{pmatrix} \\
&= \left[\begin{pmatrix} 1 & 1 & 1 & 1 \\ -2.0 & -1.0 & 0.0 & 2.0 \end{pmatrix} \begin{pmatrix} 1 & -2.0 \\ 1 & -1.0 \\ 1 & 0.0 \\ 1 & 2.0 \end{pmatrix}\right]^{-1} \begin{pmatrix} 1 & 1 & 1 & 1 \\ -2.0 & -1.0 & 0.0 & 2.0 \end{pmatrix} \begin{pmatrix} 2.0 \\ 3.0 \\ 1.0 \\ -1.0 \end{pmatrix} \\
&= \begin{pmatrix} 4.0 & -1.0 \\ -1.0 & 9.0 \end{pmatrix}^{-1} \begin{pmatrix} 1 & 1 & 1 & 1 \\ -2.0 & -1.0 & 0.0 & 2.0 \end{pmatrix} \begin{pmatrix} 2.0 \\ 3.0 \\ 1.0 \\ -1.0 \end{pmatrix} \\
&= \begin{pmatrix} 0.2571 & 0.0286 \\ 0.0286 & 0.1143 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ -2.0 & -1.0 & 0.0 & 2.0 \end{pmatrix} \begin{pmatrix} 2.0 \\ 3.0 \\ 1.0 \\ -1.0 \end{pmatrix} \\
&= \begin{pmatrix} 0.2 & 0.2286 & 0.2571 & 0.3143 \\ -0.2 & -0.0857 & 0.0286 & 0.2571 \end{pmatrix} \begin{pmatrix} 2.0 \\ 3.0 \\ 1.0 \\ -1.0 \end{pmatrix} \\
&= \begin{pmatrix} 1.0286 \\ -0.8857 \end{pmatrix}
\end{aligned}
$$

2. Predict the target value for $\boldsymbol{x}_{\text{query}} = [1]$.

   Solution:

   From the previous question, we have our weights:

   $$\boldsymbol{w} = \begin{bmatrix} 1.0286 \\ -0.8857 \end{bmatrix}$$

   So, to compute the predicted value, we just need to augment the query vector with a bias dimension and apply the linear regression:

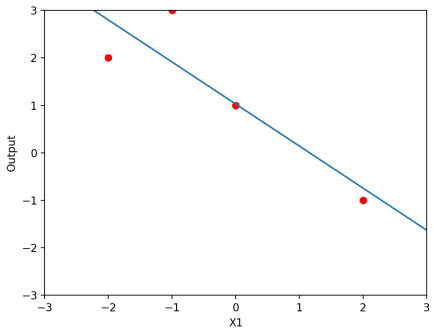
   $$\hat{y} = \boldsymbol{w} \cdot \boldsymbol{x} = \begin{bmatrix} 1.0286 \\ -0.8857 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 0.1429.$$

3. Sketch the predicted hyperplane along which the linear regression predicts points will fall.

   Solution: We can get the hyperplane's equation by taking the linear regression output for a general input $\begin{bmatrix} 1 & x_1 \end{bmatrix}^\top$ and equating it to zero:

   $$
   \begin{aligned}
   \hat{y} &= \boldsymbol{w} \cdot \boldsymbol{x} && = 0 \\
   &= \begin{bmatrix} 1.0286 \\ -0.8857 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ x_1 \end{bmatrix} && = 0 \\
   &= -0.8857 x_1 + 1.0286 && = 0
   \end{aligned}
   $$

From the equation, we get the following plot:



4. Compute the mean squared error produced by the linear regression.

Solution:

For each point in the training data, we must compute the linear regression prediction and then compute its squared error:

$$\left(y^{(1)} - \boldsymbol{w}\cdot\boldsymbol{x}^{(1)}\right)^2 = \left(2.0 - \begin{pmatrix} 1.0286 \\ -0.8857 \end{pmatrix}\cdot\begin{pmatrix} 1 \\ -2.0 \end{pmatrix}\right)^2 = (2.0 - 2.800)^2 = 0.64$$

$$\left(y^{(2)} - \boldsymbol{w}\cdot\boldsymbol{x}^{(2)}\right)^2 = \left(3.0 - \begin{pmatrix} 1.0286 \\ -0.8857 \end{pmatrix}\cdot\begin{pmatrix} 1 \\ -1.0 \end{pmatrix}\right)^2 = (3.0 - 1.9143)^2 = 1.1788$$

$$\left(y^{(3)} - \boldsymbol{w}\cdot\boldsymbol{x}^{(3)}\right)^2 = \left(1.0 - \begin{pmatrix} 1.0286 \\ -0.8857 \end{pmatrix}\cdot\begin{pmatrix} 1 \\ 0.0 \end{pmatrix}\right)^2 = (1.0 - 1.0286)^2 = 0.0008$$

$$\left(y^{(4)} - \boldsymbol{w}\cdot\boldsymbol{x}^{(4)}\right)^2 = \left(-1.0 - \begin{pmatrix} 1.0286 \\ -0.8857 \end{pmatrix}\cdot\begin{pmatrix} 1 \\ 2.0 \end{pmatrix}\right)^2 = (-1.0 - (-0.7429))^2 = 0.0661$$

So, the mean squared error is:

$$\frac{0.64 + 1.1788 + 0.0008 + 0.0661}{4} = 0.4714$$

Question 2

Consider the following training data:

$$\boldsymbol{x}^{(1)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \boldsymbol{x}^{(2)} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \boldsymbol{x}^{(3)} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}, \boldsymbol{x}^{(4)} = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

$$y^{(1)} = 1.4, y^{(2)} = 0.5, y^{(3)} = 2, y^{(4)} = 2.5$$

1. Find the closed form solution for a linear regression that minimizes the sum of squared errors on the training data..

Solution:

First, we need to build the $n \times (d+1)$ design matrix to account for the bias parameter, where n is the number of examples and d is the original number of input features.

$$\mathbf{X} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 3 \\ 1 & 3 & 3 \end{bmatrix}$$

Second, we construct a target vector:

$$\boldsymbol{y} = \begin{bmatrix} 1.4 \\ 0.5 \\ 2.0 \\ 2.5 \end{bmatrix}$$

Now, the goal is to find the weight vector $\boldsymbol{w} = \begin{bmatrix} w_0 & w_1 & w_2 \end{bmatrix}^\top$ that minimizes the sum of squared errors. We can do it using the pseudo-inverse:

$$\boldsymbol{w} = \left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \boldsymbol{y}$$

$$\begin{aligned}
\boldsymbol{w} &= \left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \boldsymbol{y} \\
&= \left[\begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 3 \\ 1 & 3 & 3 \end{pmatrix}^\top \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 3 \\ 1 & 3 & 3 \end{pmatrix}\right]^{-1} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 3 \\ 1 & 3 & 3 \end{pmatrix}^\top \begin{pmatrix} 1.4 \\ 0.5 \\ 2.0 \\ 2.5 \end{pmatrix} \\
&= \left[\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 1 & 3 \\ 1 & 1 & 3 & 3 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 3 \\ 1 & 3 & 3 \end{pmatrix}\right]^{-1} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 1 & 3 \\ 1 & 1 & 3 & 3 \end{pmatrix} \begin{pmatrix} 1.4 \\ 0.5 \\ 2.0 \\ 2.5 \end{pmatrix} \\
&= \begin{pmatrix} 4 & 7 & 8 \\ 7 & 15 & 15 \\ 8 & 15 & 20 \end{pmatrix}^{-1} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 1 & 3 \\ 1 & 1 & 3 & 3 \end{pmatrix} \begin{pmatrix} 1.4 \\ 0.5 \\ 2.0 \\ 2.5 \end{pmatrix} \\
&= \begin{pmatrix} 1.875 & -0.5 & -0.375 \\ -0.5 & 0.4 & -0.1 \\ -0.375 & -0.1 & 0.275 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 1 & 3 \\ 1 & 1 & 3 & 3 \end{pmatrix} \begin{pmatrix} 1.4 \\ 0.5 \\ 2.0 \\ 2.5 \end{pmatrix} \\
&= \begin{pmatrix} 1.0 & 0.5 & 0.25 & -0.75 \\ -0.2 & 0.2 & -0.4 & 0.4 \\ -0.2 & -0.3 & 0.35 & 0.15 \end{pmatrix} \begin{pmatrix} 1.4 \\ 0.5 \\ 2.0 \\ 2.5 \end{pmatrix} \\
&= \begin{pmatrix} 0.275 \\ 0.02 \\ 0.645 \end{pmatrix}
\end{aligned}$$

2. Predict the target value for $\boldsymbol{x}_{\text{query}} = \begin{bmatrix} 2 & 3 \end{bmatrix}^\top$.

Solution: From the previous question, we have our weights:

$$\boldsymbol{w} = \begin{bmatrix} 0.275 \\ 0.02 \\ 0.645 \end{bmatrix}$$

So, to compute the predicted value, we just need to augment the query vector with a bias dimension and apply the linear regression:

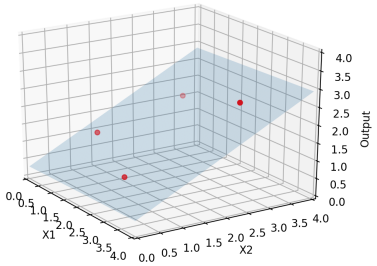
$$\hat{y} = \boldsymbol{w} \cdot \boldsymbol{x} = \begin{bmatrix} 0.275 \\ 0.02 \\ 0.645 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = 2.25$$

3. Sketch the predicted hyperplane along which the linear regression predicts points will fall.

Solution: We can get the hyperplane's equation by taking the linear regression output for a general input $\begin{bmatrix} 1 & x_1 & x_2 \end{bmatrix}^\top$ and equating it to zero:

$$\begin{aligned} \hat{y} &= \boldsymbol{w} \cdot \boldsymbol{x} && = 0 \\ &= \begin{bmatrix} 0.275 \\ 0.02 \\ 0.645 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} && = 0 \\ &= 0.02x_1 + 0.645x_2 + 0.275 && = 0 \end{aligned}$$

From the equation, we get the following plot:



4. Compute the mean squared error produced by the linear regression.

Solution:

For each point in the training data, we must compute the linear regression prediction and then compute its squared error:

$$\left(y^{(1)} - \hat{y}^{(1)}\right)^2 = \left(y^{(1)} - \boldsymbol{w} \cdot \boldsymbol{x}^{(1)}\right)^2 = \left(1.4 - \begin{pmatrix} 0.275 \\ 0.02 \\ 0.645 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}\right)^2 = (1.4 - 0.94)^2 = 0.2116$$

$$\left(y^{(2)} - \hat{y}^{(2)}\right)^2 = \left(y^{(2)} - \boldsymbol{w} \cdot \boldsymbol{x}^{(2)}\right)^2 = \left(0.5 - \begin{pmatrix} 0.275 \\ 0.02 \\ 0.645 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}\right)^2 = (0.5 - 0.96)^2 = 0.2116$$

$$\left(y^{(3)} - \hat{y}^{(3)}\right)^2 = \left(y^{(3)} - \boldsymbol{w} \cdot \boldsymbol{x}^{(3)}\right)^2 = \left(2.0 - \begin{pmatrix} 0.275 \\ 0.02 \\ 0.645 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 3 \end{pmatrix}\right)^2 = (2.0 - 2.23)^2 = 0.0529$$

$$\left(y^{(4)} - \hat{y}^{(4)}\right)^2 = \left(y^{(4)} - \boldsymbol{w} \cdot \boldsymbol{x}^{(4)}\right)^2 = \left(2.5 - \begin{pmatrix} 0.275 \\ 0.02 \\ 0.645 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 3 \\ 3 \end{pmatrix}\right)^2 = (2.5 - 2.27)^2 = 0.0529$$

So, the mean error is:

$$\frac{0.2116 + 0.2116 + 0.0529 + 0.0529}{4} = 0.13225$$

Question 3

Consider the following training data:

$$\boldsymbol{x}^{(1)} = [3], \quad \boldsymbol{x}^{(2)} = [4], \quad \boldsymbol{x}^{(3)} = [6], \quad \boldsymbol{x}^{(4)} = [10], \quad \boldsymbol{x}^{(5)} = [12]$$

$$y^{(1)} = 1.5, \quad y^{(2)} = 9.3, \quad y^{(3)} = 23.4, \quad y^{(4)} = 45.8, \quad y^{(5)} = 60.1$$

1. Adopt a logarithmic feature transformation $\phi(x_1) = \log(x_1)$ and find the closed form solution for this non-linear regression that minimizes the sum of squared errors on the training data.

   Solution:

   First, we need to build the $n \times (d+1)$ design matrix to account for the bias parameter, where n is the number of examples and d is the original number of input features. However, unlike previous exercises where we used the features themselves directly ($\phi(x) = x$), in this case, we have a non-linear transformation. So, we apply it:

$$\Phi = \begin{pmatrix} 1 & \log(3) \\ 1 & \log(4) \\ 1 & \log(6) \\ 1 & \log(10) \\ 1 & \log(12) \end{pmatrix} = \begin{pmatrix} 1 & 1.0986 \\ 1 & 1.3863 \\ 1 & 1.7918 \\ 1 & 2.3026 \\ 1 & 2.4849 \end{pmatrix}$$

   Second, we construct a target vector:

$$\boldsymbol{y} = \begin{pmatrix} 1.5 \\ 9.3 \\ 23.4 \\ 45.8 \\ 60.1 \end{pmatrix}$$

   Now, the goal is to find the weight vector $\boldsymbol{w} = \begin{bmatrix} w_0 & w_1 \end{bmatrix}^\top$ that minimizes the sum of squared errors. We can do it using the pseudo-inverse:

$$\boldsymbol{w} = \left(\Phi^\top \Phi\right)^{-1} \Phi^\top \boldsymbol{y}$$

$$\boldsymbol{w} = \left(\Phi^\top \Phi\right)^{-1} \Phi^\top \boldsymbol{y}$$

$$= \left[\begin{pmatrix} 1 & 1.0986 \\ 1 & 1.3863 \\ 1 & 1.7918 \\ 1 & 2.3026 \\ 1 & 2.4849 \end{pmatrix}^\top \begin{pmatrix} 1 & 1.0986 \\ 1 & 1.3863 \\ 1 & 1.7918 \\ 1 & 2.3026 \\ 1 & 2.4849 \end{pmatrix}\right]^{-1} \begin{pmatrix} 1 & 1.0986 \\ 1 & 1.3863 \\ 1 & 1.7918 \\ 1 & 2.3026 \\ 1 & 2.4849 \end{pmatrix}^\top \begin{pmatrix} 1.5 \\ 9.3 \\ 23.4 \\ 45.8 \\ 60.1 \end{pmatrix}$$

$$= \left[\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1.0986 & 1.3863 & 1.7918 & 2.3026 & 2.4849 \end{pmatrix} \begin{pmatrix} 1 & 1.0986 \\ 1 & 1.3863 \\ 1 & 1.7918 \\ 1 & 2.3026 \\ 1 & 2.4849 \end{pmatrix}\right]^{-1} \begin{pmatrix} 1 & 1.0986 \\ 1 & 1.3863 \\ 1 & 1.7918 \\ 1 & 2.3026 \\ 1 & 2.4849 \end{pmatrix}^\top \begin{pmatrix} 1.5 \\ 9.3 \\ 23.4 \\ 45.8 \\ 60.1 \end{pmatrix}$$

$$= \begin{pmatrix} 5.0 & 9.0642 \\ 9.0642 & 17.8158 \end{pmatrix}^{-1} \begin{pmatrix} 1 & 1.0986 \\ 1 & 1.3863 \\ 1 & 1.7918 \\ 1 & 2.3026 \\ 1 & 2.4849 \end{pmatrix}^\top \begin{pmatrix} 1.5 \\ 9.3 \\ 23.4 \\ 45.8 \\ 60.1 \end{pmatrix}$$

$$= \begin{pmatrix} 2.5745 & -1.3098 \\ -1.3098 & 0.7225 \end{pmatrix} \begin{pmatrix} 1 & 1.0986 \\ 1 & 1.3863 \\ 1 & 1.7918 \\ 1 & 2.3026 \\ 1 & 2.4849 \end{pmatrix}^\top \begin{pmatrix} 1.5 \\ 9.3 \\ 23.4 \\ 45.8 \\ 60.1 \end{pmatrix}$$

$$= \begin{pmatrix} 1.1355 & 0.7587 & 0.2276 & -0.4415 & -0.6803 \\ -0.5160 & -0.3082 & -0.0152 & 0.3539 & 0.4856 \end{pmatrix} \begin{pmatrix} 1.5 \\ 9.3 \\ 23.4 \\ 45.8 \\ 60.1 \end{pmatrix}$$

$$= \begin{pmatrix} -47.0212 \\ 41.3945 \end{pmatrix}$$

2. Repeat the exercise above for a quadratic feature transformation $\phi(x_1) = x_1^2$.

Solution: First, we need to build the $n \times (d+1)$ design matrix to account for the bias parameter, where n is the number of examples and d is the original number of input features. However, unlike previous exercises where we used the features themselves directly ($\phi(x) = x$), in this case, we have a non-linear transformation. So, we apply it:

$$\Phi = \begin{bmatrix} 1 & 3^2 \\ 1 & 4^2 \\ 1 & 6^2 \\ 1 & 10^2 \\ 1 & 12^2 \end{bmatrix} = \begin{bmatrix} 1 & 9 \\ 1 & 16 \\ 1 & 36 \\ 1 & 100 \\ 1 & 144 \end{bmatrix}$$

Second, we construct a target vector:

$$\boldsymbol{y} = \begin{bmatrix} 1.5 \\ 9.3 \\ 23.4 \\ 45.8 \\ 60.1 \end{bmatrix}$$

Now, the goal is to find the weight vector $\boldsymbol{w} = \begin{bmatrix} w_0 & w_1 \end{bmatrix}^\top$ that minimizes the sum of squared errors. We can do it using the pseudo-inverse:

$$\boldsymbol{w} = \left(\Phi^\top \Phi\right)^{-1} \Phi^\top \boldsymbol{y}$$

$$\boldsymbol{w} = \left(\Phi^\top \Phi\right)^{-1} \Phi^\top \boldsymbol{y}$$

$$= \left[\begin{pmatrix} 1 & 9 \\ 1 & 16 \\ 1 & 36 \\ 1 & 100 \\ 1 & 144 \end{pmatrix}^\top \begin{pmatrix} 1 & 9 \\ 1 & 16 \\ 1 & 36 \\ 1 & 100 \\ 1 & 144 \end{pmatrix}\right]^{-1} \begin{pmatrix} 1 & 9 \\ 1 & 16 \\ 1 & 36 \\ 1 & 100 \\ 1 & 144 \end{pmatrix}^\top \begin{pmatrix} 1.5 \\ 9.3 \\ 23.4 \\ 45.8 \\ 60.1 \end{pmatrix}$$

$$= \left[\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 9 & 16 & 36 & 100 & 144 \end{pmatrix} \begin{pmatrix} 1 & 9 \\ 1 & 16 \\ 1 & 36 \\ 1 & 100 \\ 1 & 144 \end{pmatrix}\right]^{-1} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 9 & 16 & 36 & 100 & 144 \end{pmatrix} \begin{pmatrix} 1.5 \\ 9.3 \\ 23.4 \\ 45.8 \\ 60.1 \end{pmatrix}$$

$$= \begin{pmatrix} 5.0 & 305.0 \\ 305.0 & 32369.0 \end{pmatrix}^{-1} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 9 & 16 & 36 & 100 & 144 \end{pmatrix} \begin{pmatrix} 1.5 \\ 9.3 \\ 23.4 \\ 45.8 \\ 60.1 \end{pmatrix}$$

$$= \begin{pmatrix} 0.4703 & -0.0044 \\ -0.0044 & 0.00007 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 9 & 16 & 36 & 100 & 144 \end{pmatrix} \begin{pmatrix} 1.5 \\ 9.3 \\ 23.4 \\ 45.8 \\ 60.1 \end{pmatrix}$$

$$= \begin{pmatrix} 0.4305 & 0.3994 & 0.3108 & 0.0272 & -0.1678 \\ -0.0038 & -0.0033 & -0.0018 & 0.0028 & 0.0060 \end{pmatrix} \begin{pmatrix} 1.5 \\ 9.3 \\ 23.4 \\ 45.8 \\ 60.1 \end{pmatrix}$$

$$= \begin{pmatrix} 2.7895 \\ 0.4136 \end{pmatrix}$$

3. Plot both regressions.

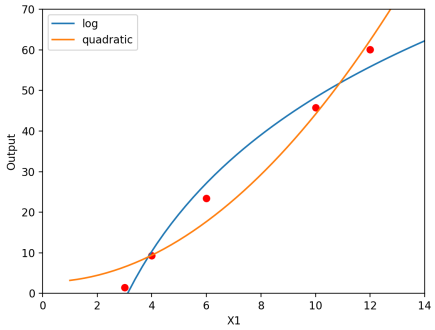
Solution: For the logarithmic, we get:

$$\hat{y} = \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}) = -47.0212 + 41.395 \log(x_1)$$

For the quadratic, we get:

$$\hat{y} = \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}) = 2.7895 + 0.4136x_1^2$$

With the equations, we can build the plots:



4. Which is a better fit, a) or b)?

Solution: Recall the equations for both regressions:

$$\hat{y}_{\log} = \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}) = -47.0212 + 41.395 \log(x_1)$$

$$\hat{y}_{\text{quadratic}} = \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}) = 2.7895 + 0.4136x_1^2$$

To measure which fit is better we will measure the mean squared errors.

For the logarithmic transform, we have:

$$\left(\Phi \boldsymbol{w}_{\log} - \boldsymbol{y}\right)^2 = \left(\begin{pmatrix} 1 & 1.0986 \\ 1 & 1.3863 \\ 1 & 1.7918 \\ 1 & 2.3026 \\ 1 & 2.4849 \end{pmatrix} \begin{pmatrix} -47.0212 \\ 41.3945 \end{pmatrix} - \begin{pmatrix} 1.5 \\ 9.3 \\ 23.4 \\ 45.8 \\ 60.1 \end{pmatrix}\right)^2 = \begin{pmatrix} 9.2704 \\ 1.1315 \\ 14.0455 \\ 6.2155 \\ 18.1460 \end{pmatrix}$$

Which yields a mean squared error of:

$$\frac{9.2704 + 1.1315 + 14.0455 + 6.2155 + 18.1460}{5} = 9.7618$$

For the quadratic transform, we have:

$$\left(\Phi \boldsymbol{w}_{\text{quadratic}} - \boldsymbol{y}\right)^2 = \left(\begin{pmatrix} 1 & 9 \\ 1 & 16 \\ 1 & 36 \\ 1 & 100 \\ 1 & 144 \end{pmatrix} \begin{pmatrix} 2.7895 \\ 0.4136 \end{pmatrix} - \begin{pmatrix} 1.5 \\ 9.3 \\ 23.4 \\ 45.8 \\ 60.1 \end{pmatrix}\right)^2 = \begin{pmatrix} 25.1202 \\ 0.0152 \\ 32.7228 \\ 2.7192 \\ 5.0628 \end{pmatrix}$$

Which yields a mean squared error of:

$$\frac{25.1202 + 0.0152 + 32.7228 + 2.7192 + 5.0628}{5} = 13.1273$$

So, given the availale data, the logarithmic transform appears to fit the data better.

Question 4

Consider the following training data:

$$\boldsymbol{x}^{(1)} = \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \quad \boldsymbol{x}^{(2)} = \begin{bmatrix} 0 \\ 0.25 \end{bmatrix}, \quad \boldsymbol{x}^{(3)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \boldsymbol{x}^{(4)} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

$$y^{(1)} = 0, \quad y^{(2)} = 1, \quad y^{(3)} = 1, \quad y^{(4)} = 0$$

In this exercise, we will consider binary logistic regression:

$$p_{\boldsymbol{w}}\left(y = 1 \mid \boldsymbol{x}\right) = \sigma(\boldsymbol{w} \cdot \boldsymbol{x}) = \frac{1}{1 + \exp\left(-\boldsymbol{w} \cdot \boldsymbol{x}\right)}$$

And we will use the cross-entropy loss function:

$$L\left(\boldsymbol{w}\right) = -\sum_{i=1}^{N} \log\left(p_{\boldsymbol{w}}\left(y^{(i)} \mid \boldsymbol{x}^{(i)}\right)\right) = -\sum_{i=1}^{N} \left(y^{(i)} \log \sigma\left(\boldsymbol{w} \cdot \boldsymbol{x}^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - \sigma\left(\boldsymbol{w} \cdot \boldsymbol{x}^{(i)}\right)\right)\right)$$

1. Determine the gradient descent learning rule for this unit.

   Solution: To apply gradient descent, we want an update rule that moves a step of size η towards the opposite direction from the gradient of the error function with respect to the weights:

   $$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \frac{\partial L\left(\boldsymbol{w}\right)}{\partial \boldsymbol{w}}$$

   To find the learning rule, we must compute the gradient:

   $$\frac{\partial L\left(\boldsymbol{w}\right)}{\partial \boldsymbol{w}} = -\sum_{i=1}^{N} \boldsymbol{x}^{(i)} (y^{(i)} - \sigma(\boldsymbol{w} \cdot \boldsymbol{x}^{(i)}))$$

   So, we can write our update rule as follows.

   $$\begin{aligned} \boldsymbol{w} &= \boldsymbol{w} - \eta \frac{\partial L\left(\boldsymbol{w}\right)}{\partial \boldsymbol{w}} \\ &= \boldsymbol{w} - \eta \left(-\sum_{i=1}^{N} \boldsymbol{x}^{(i)} \left(y^{(i)} - \sigma\left(\boldsymbol{w} \cdot \boldsymbol{x}^{(i)}\right)\right)\right) \\ &= \boldsymbol{w} + \eta \sum_{i=1}^{N} \boldsymbol{x}^{(i)} \left(y^{(i)} - \sigma\left(\boldsymbol{w} \cdot \boldsymbol{x}^{(i)}\right)\right) \end{aligned}$$

2. Compute the first gradient descent update assuming an initialization of all zeros. Assume a learning rate of 1.0.

   Solution:

   The original gradient descent does one update per epoch because its learning rule requires contributions from all data points to do one step. Let us compute it.

   According to the problem statement, we start with weights $\mathbf{w} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$ and learning rate $\eta = 1.0$.

$$
\begin{aligned}
\boldsymbol{w} &= \boldsymbol{w} + \eta \sum_{i=1}^{N} \boldsymbol{x}^{(i)} \left(y^{(i)} - \sigma\left(\boldsymbol{w} \cdot \boldsymbol{x}^{(i)} \right) \right) \\
&= \boldsymbol{w} + \eta \sum_{i=1}^{4} \boldsymbol{x}^{(i)} \left(y^{(i)} - \sigma\left(\boldsymbol{w} \cdot \boldsymbol{x}^{(i)} \right) \right) \\
&= \boldsymbol{w} + \eta \boldsymbol{x}^{(1)} \left(y^{(1)} - \sigma\left(\boldsymbol{w} \cdot \boldsymbol{x}^{(1)} \right) \right) + \eta \boldsymbol{x}^{(2)} \left(y^{(2)} - \sigma\left(\boldsymbol{w} \cdot \boldsymbol{x}^{(2)} \right) \right) + \\
&+ \eta \boldsymbol{x}^{(3)} \left(t^{(3)} - \sigma\left(\boldsymbol{w} \cdot \boldsymbol{x}^{(3)} \right) \right) + \eta \boldsymbol{x}^{(4)} \left(t^{(4)} - \sigma\left(\boldsymbol{w} \cdot \boldsymbol{x}^{(4)} \right) \right) \\
&= \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix} \left(0 - \sigma\left(\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix} \right) \right) + \begin{pmatrix} 1 \\ 0 \\ 0.25 \end{pmatrix} \left(1 - \sigma\left(\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0.25 \end{pmatrix} \right) \right) + \\
&+ \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \left(1 - \sigma\left(\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right) \right) + \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix} \left(0 - \sigma\left(\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix} \right) \right) \\
&= \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix} (0 - \sigma(0)) + \begin{pmatrix} 1 \\ 0 \\ 0.25 \end{pmatrix} (1 - \sigma(0)) + \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} (1 - \sigma(0)) + \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix} (0 - \sigma(0)) \\
&= \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} -0.5 \\ 0.5 \\ 0 \end{pmatrix} + \begin{pmatrix} 0.5 \\ 0 \\ 0.125 \end{pmatrix} + \begin{pmatrix} 0.5 \\ 0.5 \\ 0.5 \end{pmatrix} + \begin{pmatrix} -0.5 \\ -0.5 \\ 0.5 \end{pmatrix} \\
&= \begin{pmatrix} 0 \\ 0.5 \\ 1.125 \end{pmatrix}
\end{aligned}
$$

3. Compute the first stochastic gradient descent update assuming an initialization of all zeros. Assume a learning rate of 1.0.

Solution:

In stochastic gradient descent we make one update for each training example. So, instead of summing across all data points we adapt the learning rule for one example only:

$$\boldsymbol{w} = \boldsymbol{w} + \eta \boldsymbol{x} \left(y - \sigma\left(\boldsymbol{w} \cdot \boldsymbol{x} \right) \right)$$

We can now do the updates. Let us start with the first example:

$$
\begin{aligned}
\boldsymbol{w} &= \boldsymbol{w} + \eta \boldsymbol{x} \left(y - \sigma\left(\boldsymbol{w} \cdot \boldsymbol{x} \right) \right) \\
&= \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + 1 \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix} \left(0 - \sigma\left(\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix} \right) \right) \\
&= \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix} (0 - \sigma(0)) \\
&= \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} -0.5 \\ 0.5 \\ 0 \end{pmatrix} \\
&= \begin{pmatrix} -0.5 \\ 0.5 \\ 0 \end{pmatrix}
\end{aligned}
$$

Question 5

Now it's time to try multi-class logistic regression on real data and see what happens.

1. Load the UCI handwritten digits dataset using `scikit-learn`:

   ```
   from sklearn.datasets import load_digits
   data = load_digits()
   ```

   This is a dataset containing 1797 8x8 input images of digits, each corresponding to one out of 10 output classes. You can print the dataset description and visualize some input examples with:

   ```
   print(data.DESCR)

   import matplotlib.pyplot as plt
   plt.gray()
   for i in range(10):
       plt.matshow(data.images[i])
   plt.show()
   ```

   Randomly split this data into training (80%) and test (20%) partitions. This can be done with:

   ```
   from sklearn.model_selection import train_test_split
   X_train, X_test, y_train, y_test = train_test_split(
       X, y, test_size=0.2, random_state=42)
   ```

2. Run your implementation of the multi-class logistic regression algorithm on this dataset, using stochastic gradient descent with $\eta = 0.001$. Plot the loss and the training and test accuracy over the epochs.

3. Use `scikit-learn`'s implementation of multi-class logistic regression. This can be done with

   ```
   from sklearn.linear_model import LogisticRegression
   clf = LogisticRegression(fit_intercept=False, penalty='none')
   clf.fit(X_train, y_train)
   print(clf.score(X_train, y_train))
   print(clf.score(X_test, y_test))
   ```

   Compare the resulting accuracies.