Deep Learning (IST, 2022-23)

Practical 3: Linear and Logistic Regression

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

Pen-and-Paper Exercises

The following questions should be solved by hand. You can use, of course, tools for auxiliary numerical computations.

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:

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

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

$$= \left[\begin{pmatrix} 1 & -2.0 \\ 1 & -1.0 \\ 1 & 0.0 \\ 1 & 2.0 \end{pmatrix}^\top \begin{pmatrix} 1 & -2.0 \\ 1 & -1.0 \\ 1 & 0.0 \\ 1 & 2.0 \end{pmatrix}\right]^{-1} \begin{pmatrix} 1 & -2.0 \\ 1 & -1.0 \\ 1 & 0.0 \\ 1 & 2.0 \end{pmatrix}^\top \begin{pmatrix} 2.0 \\ 3.0 \\ 1.0 \\ -1.0 \end{pmatrix}$$

$$= \left[\begin{pmatrix} 1 & 1 & 1 & 1 \\ -2.0 & -1.0 & 0.0 & 2.0 \end{pmatrix} \begin{pmatrix} 1 & -2.0 \\ 1 & -1.0 \\ 1 & 0.0 \\ 1 & 2.0 \end{pmatrix}\right]^{-1} \begin{pmatrix} 1 & 1 & 1 & 1 \\ -2.0 & -1.0 & 0.0 & 2.0 \end{pmatrix} \begin{pmatrix} 2.0 \\ 3.0 \\ 1.0 \\ -1.0 \end{pmatrix}$$

$$= \begin{pmatrix} 4.0 & -1.0 \\ -1.0 & 9.0 \end{pmatrix}^{-1} \begin{pmatrix} 1 & 1 & 1 & 1 \\ -2.0 & -1.0 & 0.0 & 2.0 \end{pmatrix} \begin{pmatrix} 2.0 \\ 3.0 \\ 1.0 \\ -1.0 \end{pmatrix}$$

$$= \begin{pmatrix} 0.2571 & 0.0286 \\ 0.0286 & 0.1143 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ -2.0 & -1.0 & 0.0 & 2.0 \end{pmatrix} \begin{pmatrix} 2.0 \\ 3.0 \\ 1.0 \\ -1.0 \end{pmatrix}$$

$$= \begin{pmatrix} 0.2 & 0.2286 & 0.2571 & 0.3143 \\ -0.2 & -0.0857 & 0.0286 & 0.2571 \end{pmatrix} \begin{pmatrix} 2.0 \\ 3.0 \\ 1.0 \\ -1.0 \end{pmatrix}$$

$$= \begin{pmatrix} 1.0286 \\ -0.8857 \end{pmatrix}$$

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:

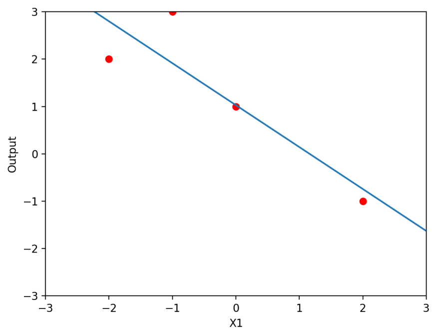
$$\hat{y} = \boldsymbol{w} \cdot \boldsymbol{x} = \begin{bmatrix} 1.0286 \\ -0.8857 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 0.1429.$$

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:

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

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:

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

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

In this exercise, we will consider binary logistic regression:

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

And we will use the cross-entropy loss function:

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

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:

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

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

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

   So, we can write our update rule as follows.

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

2. Compute the first 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:

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

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

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

Programming Exercises

The following exercises should be solved using Python, you can use the corresponding practical's notebook for guidance.

1. Consider the following training data:

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

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

   (a) Find the closed form solution for a linear regression that minimizes the sum of squared errors on the training data.
   (b) Predict the target value for $\boldsymbol{x}_{\text{query}} = \begin{bmatrix} 2 & 3 \end{bmatrix}^\top$.
   (c) Sketch the predicted hyperplane along which the linear regression predicts points will fall.
   (d) Compute the mean squared error produced by the linear regression.

2. Consider the following training data:

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

$$
y^{(1)} = 1.5, \quad y^{(2)} = 11.3, \quad y^{(3)} = 20.4, \quad y^{(4)} = 35.8, \quad y^{(5)} = 70.1
$$

   (a) 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.
   (b) Repeat the exercise above for a quadratic feature transformation $\phi(x_1) = x_1^2$.
   (c) Plot both regressions.
   (d) Which is a better fit, a) or b)?

3. Consider training set and problem setting of **Question 2** from the Pen-and-Paper exercises.

(a) Compute three epochs of gradient descent update assuming an initialization of all zeros. Assume a learning rate of 1.0.

(b) Compute three epochs of stochastic gradient descent update assuming an initialization of all zeros. Assume a learning rate of 1.0.

(c) Plot final predicted separation hyperplanes.

4. Now it's time to try multi-class logistic regression on real data and see what happens. Load the UCI handwritten digits dataset using `scikit-learn`.
   This is a dataset containing 1797 8x8 input images of digits, each corresponding to one out of 10 output classes.

   (a) Randomly split this data into training (80%) and test (20%) partitions.

   (b) Implement a function that performs one epoch of SGD for multi-class logistic regression

   (c) Run 100 epochs of your algorithm on the training data, initializing all weights to zero and a learning rate of 0.001

   (d) Compute the accuracies on both train and test sets

   (e) Use `scikit-learn`'s implementation of multi-class logistic regression and compare the resulting accuracies.