Exercise 10 – Neural Networks

Introduction to Machine Learning

Exercise 1: Logistic regression or deep learning?

Learning goals

1) Understand how stacking single neurons can solve a non-linear classification problem
2) Translate between functional & graph description of NNs

Suppose you have a set of five training points from two classes. Consider a logistic regression model $f(\mathbf{x}) = \sigma(\alpha^\top \mathbf{x}) = \sigma(\alpha_0 + \alpha_1 x_1 + \alpha_2 x_2)$, with $\sigma(\cdot)$ the logistic/sigmoid function, $\sigma(c) = \frac{1}{1+\exp(-c)}$.

Which values for $\alpha = (\alpha_0, \alpha_1, \alpha_2)^\top$ would result in correct classification for the problem in Figure 1 (assuming a threshold of 0.5 for the positive class)?

Don't use any statistical estimation to answer this question – think in geometrical terms: you need a linear hyperplane that represents a suitable decision boundary.

Solution

We need to find a linear hyperplane – i.e., a line in 2D – that separates our classes. There are infinitely many such lines, but the sketch suggests that the line with intercept 0.5 and slope 1 has maximal "safety margin": it has the broadest corridor in which we could move the line without incurring a misclassification error (indicated by dotted lines). Intuitively, this leads to better generalization than any line with a smaller such margin (the mathematical reasoning is treated in support vector machines $\rightsquigarrow$ supervised learning lecture).

Now, we need to translate this into logistic regression coefficients. Recall that the linear hyperplane for binary logistic regression is defined via

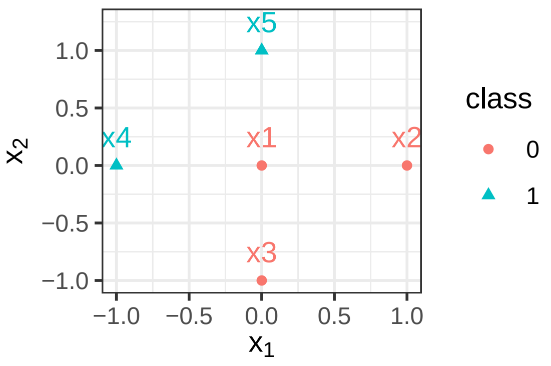
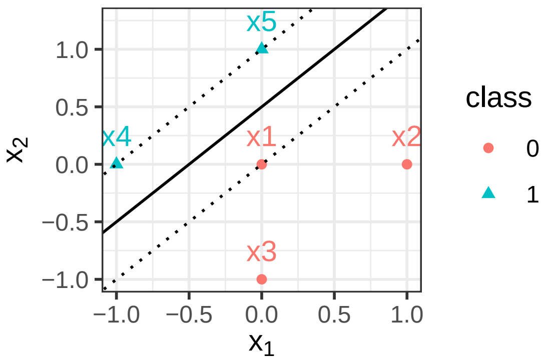


Figure 1: Classification problem I

$$\alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 = -\log(\tfrac{1}{c} - 1) \iff \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 = 0 \quad \text{for } c = 0.5, \tag{1}$$

where $c \in [0, 1]$ is the classification threshold.



Quick solution

Logistic regression is not the focus of this exercise, so TL;DR: pick values (not unique) for which points on the line fulfill Equation 1. For example: $\alpha' = (-0.5, -1, 1)^\top$.

Detailed solution

Equation 1 has 3 unknown quantities – using a system of linear equations, derived from points we know our line must cross, 3 equations should suffice if there is a unique solution. It turns out that the solution for the linear hyperplane in logistic regression is *not* necessarily unique: multiplying the coefficients in Equation 1 with a constant leaves the equality unchanged.

In ERM, we have additional constraints that usually allow us to find an optimal parameter vector: larger α driving the loss down for correctly classified observations is offset by driving it up for misclassified ones.

However, in this special case of linear separability, where a linear classifier produces no misclassification, there is no such optimum (or rather, it is loss-optimal to push the parameter vector to infinity). We will therefore, in this highly unrealistic and simplistic example, use two points the line crosses (which is enough to define any line) and just choose some value for α that fulfills Equation 1.

Let's pick the points $(0, 0.5)^\top$ and $(-0.5, 0)^\top$ (0's are always convenient):

$$\alpha_0 + \alpha_1 \cdot 0 + \alpha_2 \cdot 0.5 = 0 \tag{2}$$

$$\alpha_0 - \alpha_1 \cdot 0.5 + \alpha_2 \cdot 0 = 0 \tag{3}$$

Equation 2 - Equation 3 yields

$$\alpha_1 \cdot 0.5 + \alpha_2 \cdot 0.5 = 0 \;\Leftrightarrow\; \alpha_1 = -\alpha_2, \tag{4}$$

so set, e.g., $\alpha_1' = -1$ and $\alpha_2' = 1$. Plugging this back into either Equation 2 or Equation 3 yields $\alpha_0' = -0.5$, such that $\alpha' = (-0.5, -1, 1)^\top$, and we get the following probabilistic scores:

i	$y^{(i)}$	$\sigma(\alpha_0' + \alpha_1' x_1^{(i)} + \alpha_2' x_2^{(i)})$
1	0	0.38
2	0	0.18
3	0	0.18
4	1	0.62
5	1	0.62

(You can easily verify that a multiple $a \cdot \alpha'$ of α' leads to the same class assignments, with probabilities closer to 0/1 if $a > 1$.)

Apply the same principle to find the parameters $\beta = (\beta_0, \beta_1, \beta_2)^\top$ for the modified problem in Figure 2.

Solution

By the same principle as above, we can derive a possible solution $\beta' = (-0.5, 1, -1)^\top$.

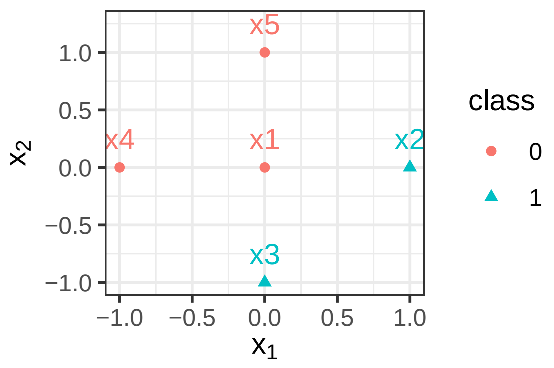
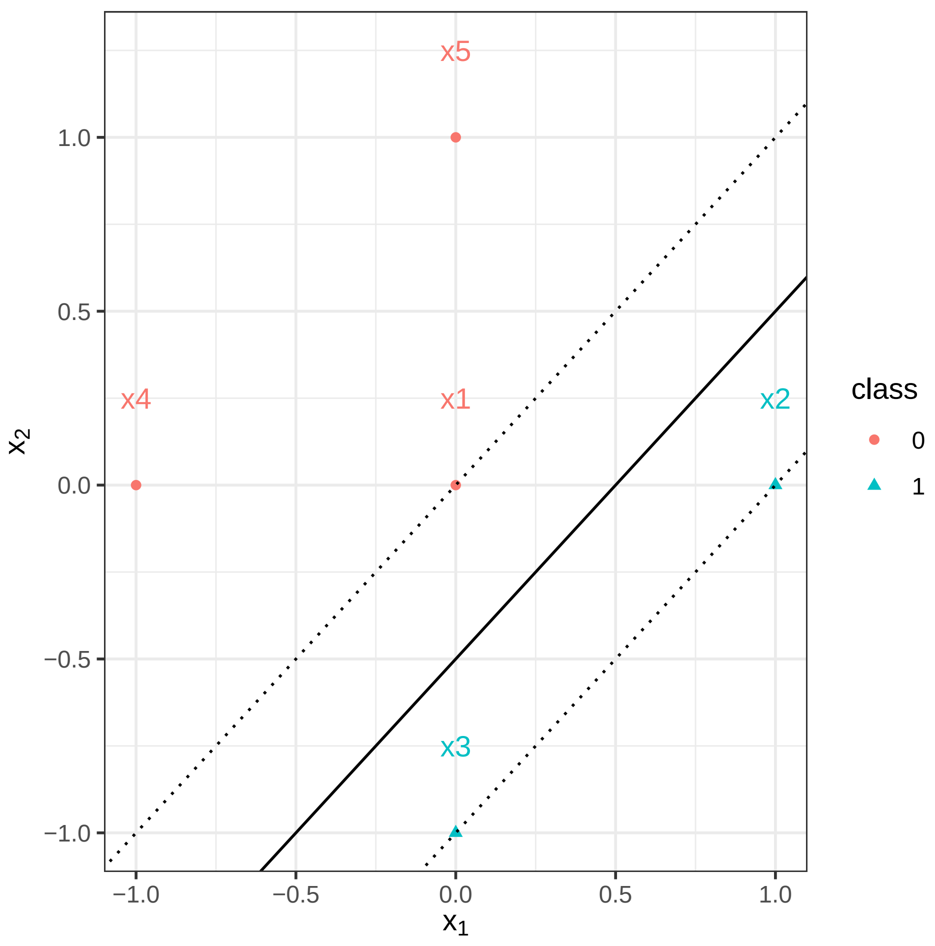


Figure 2: Classification problem II



i	$y^{(i)}$	$\sigma(\beta'_0 + \beta'_1 x_1^{(i)} + \beta'_2 x_2^{(i)})$
1	0	0.38
2	1	0.62
3	1	0.62
4	0	0.18
5	0	0.18

Now consider the problem in Figure 3, which is not linearly separable anymore, so logistic regression will not help us any longer. Suppose we had alternative coordinates $(z_1, z_2)^\top$ for our data points:

i	$z_1^{(i)}$	$z_2^{(i)}$	$y^{(i)}$
1	0	0	1
2	0	1	0
3	0	1	0
4	1	0	0
5	1	0	0

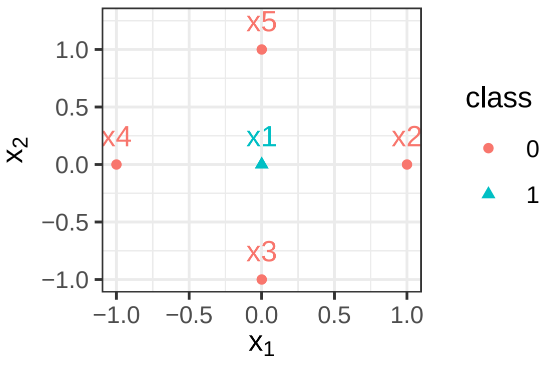


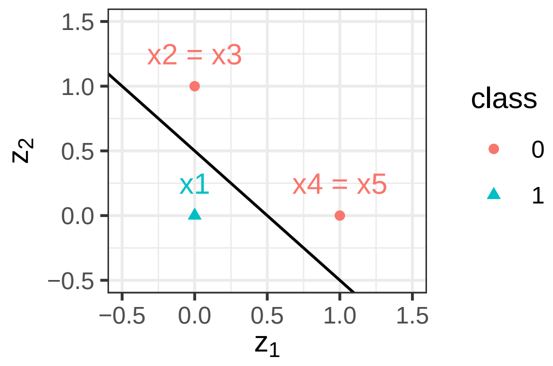
Figure 3: Classification problem III

Explain how we can use z_1 and z_2 to classify the dataset in Figure 3. The question is now, of course, how we can get these z_1 and z_2 that provide a solution to our previously unsolved problem – naturally, from the data.

Solution

Intuitively, we can see that a condition like $z_1 + z_2 > 0$ suffices to separate $\mathbf{x}^{(1)}$ from the other observations, which should remind you of the equation for a linear hyperplane.

Graphically, we can visualize our points in the new z_1, z_2 coordinates and see that this transformed dataset is indeed linearly separable:



The question is now, of course, how we can get these z1 and z2 that provide a solution to our previously unsolved problem – naturally, from the data.

Perform logistic regression to predict z_1 and z_2 (separately), treating them as target labels to the original observations with coordinates $(x_1, x_2)^\top$. Find the respective parameter vectors $\gamma, \phi \in \mathbb{R}^3$.

Solution

Looking at z_1, we find that it serves to separate observations $\{1, 2, 3\}$ from $\{4, 5\}$. Likewise, z_2 separates $\{1, 4, 5\}$ from $\{2, 3\}$. Predicting z_1 thus corresponds to classification problem I, and z_2 to classification problem II, so we can use our previous solutions $\gamma = \alpha' = (-0.5, -1, 1)^\top$ and $\phi = \beta' = (-0.5, 1, -1)^\top$.

Lastly, put together your previous results to formulate a model that predicts the original target y from the original features $(x_1, x_2)^\top$.

Solution

We can simply stack the logistic regression components as follows:

$$\begin{aligned}\hat{y} &= \sigma\left(\theta_0 + \theta_1 z_1 + \theta_2 z_2\right) \\ &= \sigma\left(\theta_0 + \theta_1 \cdot \sigma\left(\alpha_0 + \alpha_1 x_1 + \alpha_2 x_2\right) + \theta_2 \cdot \sigma\left(\beta_0 + \beta_1 x_1 + \beta_2 x_2\right)\right)\end{aligned}$$

Choosing parameters as before yields, e.g., $\theta' = (0.5, -1, -1)^\top$. In practice, we will of course use ERM to estimate, rather than hand-pick, α, β, θ.

Sketch the neural network you just created (perhaps without realizing it).

Solution

We created a feedforward neural network with two input neurons, two hidden neurons (each performing one intermediate logistic regression), and one output neuron. The biases are depicted as 1-neurons:

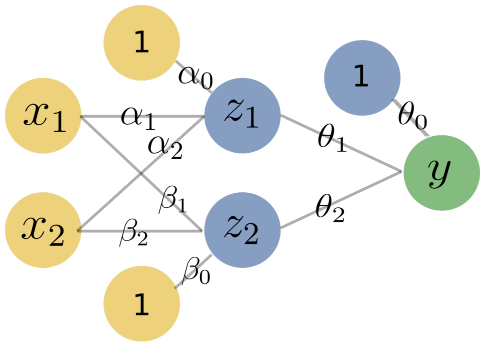


Figure 4: Feedforward neural network

Explain briefly how the chain rule is applied to the computational graph such a neural network represents. Can you think of a use we can put the resulting gradients to?

Solution

The chain rule enables us to compute (first) derivatives of the empirical risk w.r.t. arbitrary parts of the computational graph by chaining known derivatives of elementary functions, thus providing a simple and scalable solution to the problem of differentiating complicated functions.

For example, we can easily derive $\frac{\partial}{\partial \alpha_1} \mathcal{R}_{\text{emp}}$, which should remind you of the update term in gradient descent. Indeed, (variants of) gradient descent is exactly what we use to train neural networks, updating all parameters according to their contribution to $\mathcal{R}_{\text{emp}}$ during the so-called *backpropagation*.