SS 2025
© Prof. Georgia Koppe

Course Coordinator: Andre Herz
andre.herz@iwr.uni-heidelberg.de

Machine Learning Essentials

Exercise Sheet 03

Due: 26.05.2025 11:15

This sheet covers **empirical risk minimization (ERM)**, **logistic regression** and a **first intro to neural networks**. In the first exercise, you'll dip your toes into the waters of statistical learning theory by analyzing the **ERM principle**, an important framework for estimating the parameters and evaluating the performance of ML models. It also provides a (theoretically justified) way of how to think about **generalization and overfitting**, which you'll explore in the exercise. The second exercise will show how a standard form of **neural networks** arises from the idea of multi-layer logistic regression.

Regulations

Please submit your solutions via Moodle in teams of **3** students. The coding tasks must be completed using the template `MLE25_sheet03.ipynb`. Each submission must include **exactly** one file:

- Upload a `.pdf` file containing both your Jupyter notebook and solutions to analytical exercises.

The Jupyter notebook can be exported to pdf by selecting **File** → **Download as** → **pdf** in JupyterLab. If this method does not work, you may print the notebook as a pdf instead. Your analytical solutions can be either scanned handwritten solutions or created using LaTeX.

Exercise 1: Empirical Risk Minimization

In supervised learning, we generally assume the data to be drawn from an unknown underlying distribution (the "data-generating distribution"), $(\boldsymbol{x}, \boldsymbol{y}) \sim p(\boldsymbol{x}, \boldsymbol{y})$. Probabilistic **discriminative models** then aim to learn the conditional distribution $p(\boldsymbol{y}|\boldsymbol{x})$, whereas deterministic ones directly learn a mapping $\boldsymbol{f}(\boldsymbol{x})$ from inputs to outputs.[1] In both cases, the model is restricted to a predefined **hypothesis space** $\mathcal{H}$. For deterministic models, we assume that there exists an unknown deterministic function $\tilde{\boldsymbol{f}}(\boldsymbol{x}) = \boldsymbol{y}$ that describes

[1] So far, you've seen the classification case, where the outputs $\boldsymbol{y}_i$ are discrete. If they are continuous, it's called a regression problem.

the relationship between $\boldsymbol{x}$ and $\boldsymbol{y}$. This function is called the **target function**. The goal of the model then is to learn (an approximation to) the target function within $\mathcal{H}$ (e.g. the family of all linear functions for a linear model). To do so, the model is trained on a finite **training set**

$$\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{N},$$

to obtain the best possible estimate for the unknown target function within $\mathcal{H}$. This is done by minimizing an objective function that quantifies predictive errors: the **loss function**

$$\mathcal{L}(\boldsymbol{y}, \boldsymbol{f}(\boldsymbol{x})).$$

Statistical learning theory formalizes this procedure through the **ERM principle**. It states that we should choose a model $\boldsymbol{f}$ that minimizes the **empirical risk**

$$R_{\text{emp}}(\boldsymbol{f}|\mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(\boldsymbol{y}_i, \boldsymbol{f}(\boldsymbol{x}_i)),$$

as a proxy to minimizing the **expected risk** (or **true risk**)

$$R(\boldsymbol{f}) = \mathbb{E}_{p(\boldsymbol{x},\boldsymbol{y})}\left[\mathcal{L}\left(\tilde{\boldsymbol{f}}(\boldsymbol{x}) = \boldsymbol{y}, \boldsymbol{f}(\boldsymbol{x})\right)\right].$$

The expected risk reflects the generalization capability of the model and thus is the quantity we ultimately care about. The ERM principle can hence be summarized by

$$\hat{\boldsymbol{f}} = \arg\min_{\boldsymbol{f}\in\mathcal{H}} R_{\text{emp}}(\boldsymbol{f}|\mathcal{D}).$$

ERM does not always result in good models however. The **no free lunch theorem** tells us that no single learning algorithm is optimal for every possible task, which highlights that the performance of ERM depends on how well the chosen hypothesis space and loss function match the specific data distribution. Certain choices of $\mathcal{H}$ can lead to **overfitting** or **underfitting**, where $\hat{\boldsymbol{f}}$ performs poorly in terms of generalization. In this exercise you will explore the ERM principle and the conceptual differences between fitting the training data (empirical risk) and achieving good generalization (expected risk).

Tasks

1. Assume that the training set $\mathcal{D}$ consists of $N \in \mathbb{N}$ i.i.d. samples drawn from the data-generating distribution $p(\boldsymbol{x}, \boldsymbol{y})$. For a bounded loss function, show that the empirical risk is a **consistent** estimator for the expected risk:

   $$R_{\text{emp}}(\boldsymbol{f}|\mathcal{D}) \overset{a.s.}{\to} R(\boldsymbol{f}),$$

   i.e. the empirical risk converges almost surely to the expected risk, as $N \to \infty$. Convergence "almost surely" means the probability that a sequence of random variables

does not converge to the respective quantity is zero (so it converges with probability 1).
Hint: The i.i.d. nature of the samples is crucial. Consider fundamental principles from probability theory that connect empirical averages to theoretical expectations, as covered e.g. in the first 2 tutorials.

(2 pts.)

2. Consider two hypothesis spaces: a simple space $\mathcal{H}_1$ (**linear functions**) and a more complex space $\mathcal{H}_2$ (**high-degree polynomials**).

   (a) Suppose you have two candidate models $\boldsymbol{f}_1 \in \mathcal{H}_1$ and $\boldsymbol{f}_2 \in \mathcal{H}_2$ trained on $\mathcal{D}$, where $R_{\text{emp}}(\boldsymbol{f}_2|\mathcal{D}) < R_{\text{emp}}(\boldsymbol{f}_1|\mathcal{D})$ but $R(\boldsymbol{f}_2) > R(\boldsymbol{f}_1)$. Explain how this situation can arise by discussing the role of **overfitting**. Argue in terms of the **bias** and the **variance**[2] of the estimator $\hat{\boldsymbol{f}}$.

   (3 pts.)

   (b) Let the target function be a linear trend in 1D, contaminated with some Gaussian white noise, such that $\forall i = 1, \ldots, N$:

   $$y_i = wx_i + b + \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, \sigma^2).$$

   Sketch how a scenario of **overfitting** looks like for the specific case of models from $\mathcal{H}_2$ and $\mathcal{H}_1$.

   (2 pts.)

   (c) Describe how k-fold **cross validation** can help in selecting between such models. What are the advantages of using cross validation as an estimator for $R(\boldsymbol{f})$?

   (2 pts.)

3. What's the best hypothesis a model can learn, according to the ERM principle? In this task, you'll investigate this by deriving the optimal $\boldsymbol{f}$ for two fundamental losses in classification/regression.

   (a) Consider a binary classification problem with labels $y \in \{0, 1\}$ and the misclassification loss

   $$R(f) = \mathbb{E}_{p(\boldsymbol{x},y)} \left[\mathbb{1}_{\{f(\boldsymbol{x}) \neq y\}} \right],$$

   where $\mathbb{1}_A$ is the indicator function of a set A. Show that (as asserted in Sheet 2, Exercise 1) the **Bayesian decision rule**, given by the MAP estimate for $p(y \mid \boldsymbol{x})$, minimizes the expected risk $R(f)$.

   (3 pts.)

[2] Note that in this context, bias refers to errors due to overly simple assumptions about $\mathcal{H}$, whereas variance refers to errors due to sensitivity to fluctuations in the training data.

(b) Derive the optimal regression function ($\boldsymbol{y}$ is continuous here) which minimizes $R(\boldsymbol{f})$ in case of the squared error loss function:

$$\mathcal{L}(\boldsymbol{y}, \boldsymbol{f}(\boldsymbol{x})) = ||\boldsymbol{y} - \boldsymbol{f}(\boldsymbol{x})||_2^2.$$

(3 pts.)

Hint: For both cases, use

$$R(\boldsymbol{f}) = \mathbb{E}_{p(\boldsymbol{x},\boldsymbol{y})}\left[\mathcal{L}(\boldsymbol{y}, \boldsymbol{f}(\boldsymbol{x}))\right] = \mathbb{E}_{p(\boldsymbol{x})}\Big[\mathbb{E}_{p(\boldsymbol{y}|\boldsymbol{x})}\left[\mathcal{L}(\boldsymbol{y}, \boldsymbol{f}(\boldsymbol{x}))\right]\Big] = \mathbb{E}_{p(\boldsymbol{x})}\Big[R\left(\boldsymbol{f}|\boldsymbol{x}\right)\Big]$$

and minimize the inner expectation (i.e, the conditional risk) for each $\boldsymbol{x}$.

4. **Optional** When an empirical risk is minimized by a hypothesis $\hat{f}$, and we can show that the expected risk is also minimized as a consequence, the problem is called **learnable**. The learnability therefore depends on the gap between $R_{\text{emp}}(\hat{f}|\mathcal{D})$ and $R(\hat{f})$. Assume that for a fixed hypothesis f, the loss function $\mathcal{L}(y, f(x))$ is bounded:

$$0 \leq \mathcal{L}(y, f(x)) \leq M \quad \forall (x, y).$$

(a) Hoeffding's inequality states that for i.i.d. random variables X_i, $i = 1, ..., N$, with $0 \leq X_i \leq M$, for any $\epsilon > 0$,

$$\Pr\left(\Big|\frac{1}{N}\sum_{i=1}^{N} X_i - \mathbb{E}[X_i]\Big| \geq \epsilon\right) \leq 2\exp\Big(-\frac{2N\epsilon^2}{M^2}\Big).$$

Using this inequality, derive a bound of the form

$$|R_{\text{emp}}(f|\mathcal{D}) - R(f)| < \sqrt{\frac{M^2 \ln(2/\delta)}{2N}},$$

that holds with probability at least $1 - \delta$.

(2 pts.)

(b) Explain in your own words what this bound implies about the reliability of ERM when working with a finite number of training samples.

(1 pts.)

Exercise 2: From Logistic Regression to Neural Networks

A linear classifier is insufficient to correctly solve problems where the classes are not linearly separable when using the "untransformed" feature vector $\boldsymbol{\phi}(\boldsymbol{x}) = \boldsymbol{x}$. A classic example of this is the **XOR problem**. In their book *Perceptrons* (1969), Minsky and Papert

showed that the Perceptron is incapable of learning the XOR function (or any other non-linearly separable function). In this exercise, you'll see how multiple consecutive "**layers**" of logistic regression can overcome this limitation by constructing feature representations for the output layer. In fact, these consecutive layers of logistic regression result in a standard form of what is known as a **multilayer Perceptron (MLP)**, or **fully connected neural network (FCN)**. Consider the XOR (i.e. exlusive or) problem. We have binary inputs

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad x_1, x_2 \in \{0, 1\},$$

and outputs defined by

$$y = \mathrm{XOR}(x_1, x_2) = \begin{cases} 0 & \text{if } x_1 = x_2, \\ 1 & \text{if } x_1 \neq x_2. \end{cases}$$

A plot of corresponding data shows that the positive and negative examples cannot be separated by a single line. However, by constructing intermediate feature representations

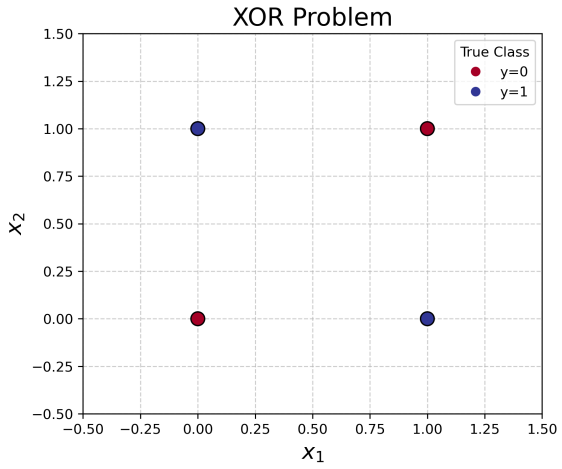

Illustration of the XOR problem with untransformed features.

using an additional layer of logistic regression, we can achieve nonlinear decision boundaries with respect to the original feature space (see also Sheet 1, Exercise 3). Consider for example the following manual design of the intermediate features:

$$\phi_1 = \sigma(8x_1 - 4x_2 - 6), \quad \phi_2 = \sigma(-4x_1 + 8x_2 - 6),$$

where σ is the logistic sigmoid function. It can be verified that ϕ_1 is high when $x_1 = 1$ and $x_2 = 0$ and low otherwise, and similarly for ϕ_2. Then, by using a logistic regression output

unit with weights $\boldsymbol{w} = \begin{bmatrix} 1 & 1 \end{bmatrix}^\top$ and bias $b = -0.5$, the overall model becomes:

$$p(y = 1|\boldsymbol{x}) = \sigma\Big(\phi_1 + \phi_2 - 0.5\Big),$$

which correctly implements the XOR function.

Task

1. Draw a diagram of the neural network. Use circles for the logistic regression units (i.e. "neurons"), and arrows to represent the weighted connections between layers and the biases for the units. Also label the arrows according to the corresponding values.

   (1 pts.)

2. To implement a **two-layer neural network** (i.e. one "hidden" and one output layer) for the XOR problem, implement the following architecture:

   - **Input Layer:** Two inputs x_1 and x_2.
   - **Hidden Layer:** Two neurons with logistic activation. Use a weight matrix

     $$\boldsymbol{W}^{(1)} = \begin{bmatrix} 8 & -4 \\ -4 & 8 \end{bmatrix} \quad \text{and bias vector } \mathbf{b}^{(1)} = \begin{bmatrix} -6 \\ -6 \end{bmatrix}.$$

     The superscript in this notation denotes the respective layer of the network.
   - **Output Layer:** One neuron with logistic activation, with weight vector

     $$\boldsymbol{w}^{(2)} = \begin{bmatrix} 1 & 1 \end{bmatrix}^\top \quad \text{and bias } b^{(2)} = -0.5.$$

   - **Forward pass:** A function that computes the final output of the network given the input:

     $$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{w}^{(2)\top}\boldsymbol{\phi}(\boldsymbol{x}) + b^{(2)}), \quad \text{where } \boldsymbol{\phi}(\boldsymbol{x}) = \sigma\left(\boldsymbol{W}^{(1)}\boldsymbol{x} + b^{(1)}\right)$$

     The subscript $\boldsymbol{\theta}$ here is an often used notation to express the parameters of the network. Here, $\boldsymbol{\theta} = \{\boldsymbol{W}^{(1)}, \boldsymbol{b}^{(1)}, \boldsymbol{w}^{(2)}, b^{(2)}\}$.

   (3 pts.)

3. For the input, hard-code the four possible input-output pairs for the XOR dataset. Use your forward pass method to compute the network's output for each input, and plot the resulting decision boundary in the (x_1, x_2) plane.

   (2 pts.)

4. In a general neural network context, one would say that the units of the network have a (elementwise applied) sigmoid **activation function**, $\boldsymbol{\phi}(\tilde{\boldsymbol{z}}) = \sigma(\tilde{\boldsymbol{z}})$ with which the **preactivations** $\tilde{\boldsymbol{z}}$ are activated. Consider a network with L layers, where each layer is linear[3] (i.e. the activation function is the identity):

$$\tilde{\boldsymbol{z}}^{(l)} = \boldsymbol{W}^{(l)}\tilde{\boldsymbol{z}}^{(l-1)} + \boldsymbol{b}^{(l)} \quad \text{for } l = 1, \ldots, L, \quad \text{with } \tilde{\boldsymbol{z}}^{(0)} = \boldsymbol{x}.$$

Show that the output of this network, for a suitable set of parameters, is equivalent to a linear 1-layer network. Then explain why this implies that activation functions need to be nonlinear in order to solve problems like the XOR problem. Or in other words, that stacking linear layers does not increase the expressiveness of the network beyond a single linear transformation.

(3 pts.)

5. We've seen that the output layer performs logistic regression on the 2D representation $\begin{bmatrix}\phi_1 & \phi_2\end{bmatrix}^\top$ given by the hidden layer. Let's explore how this feature transformation helps with solving the XOR problem.

   (a) Using your implementation, compute the hidden layer activations $\boldsymbol{\phi}$ of the four XOR input vectors.

   (1 pts.)

   (b) Create a 2D scatter plot in the (ϕ_1, ϕ_2) plane. Plot the hidden layer activations from part (a), coloring or labeling them according to their respective true XOR output ($y = 0$ or $y = 1$).

   (2 pts.)

   (c) On the same plot, draw the decision boundary defined by the output layer into the (ϕ_1, ϕ_2) plane.

   (1 pts.)

   (d) Briefly discuss: How does this visualization help explain why the 2-layer network can solve the XOR problem, while a single layer logistic regression or an L-layer linear network cannot?

   (2 pts.)

[3] You may have noticed that we often call transformations linear, even though they are technically **affine** (linear functions must satisfy $f(0) = 0$). This is often done for convenience in ML, even though the bias might provide important flexibility in terms of shifting/scaling the activation.