Einführung in das statistische Lernen

ML-Basics: What is Machine Learning?

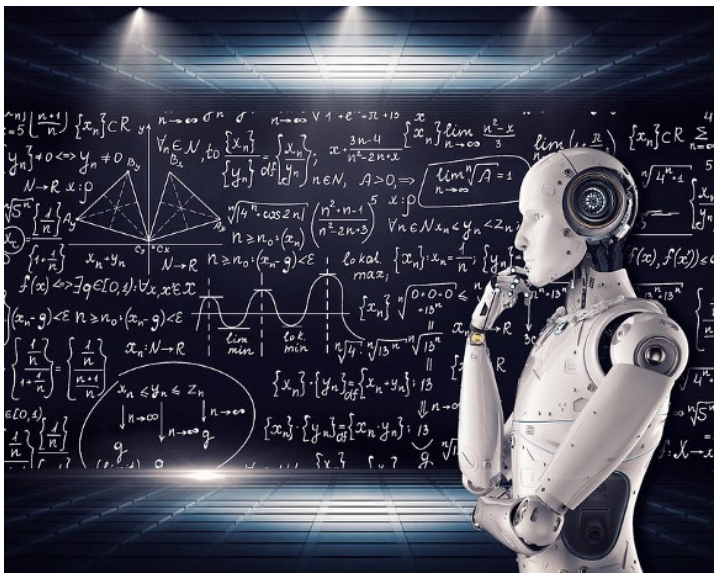


Image via `www.vpnsrus.com`

Learning goals

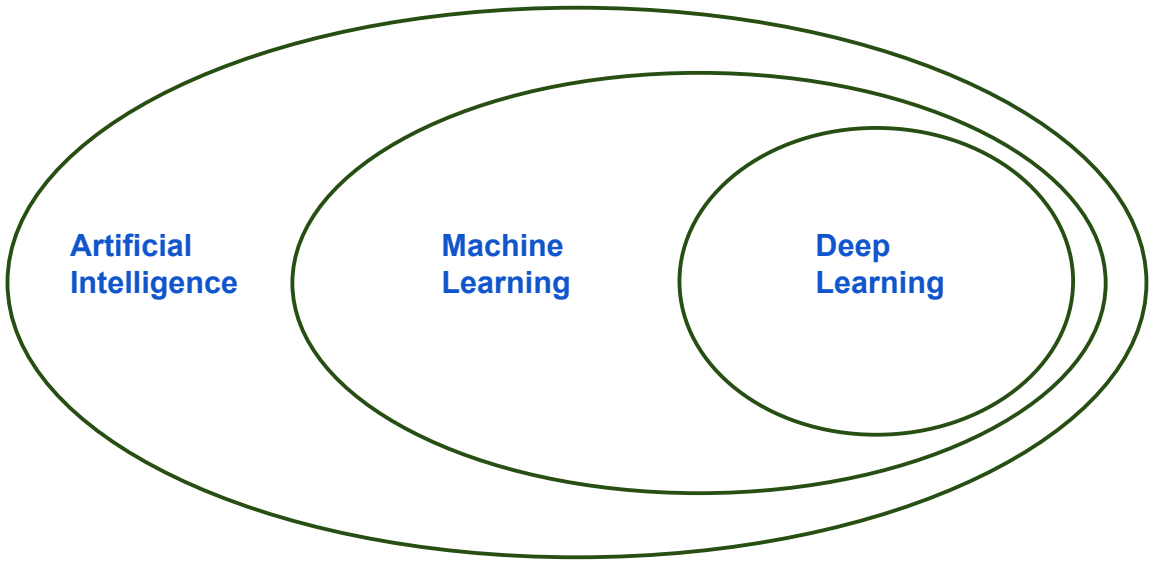
- Understand basic terminology of and connections between ML, AI, DL and statistics
- Know the main directions of ML: Supervised, Unsupervised and Reinforcement Learning

MACHINE LEARNING IS CHANGING OUR WORLD

- Search engines learn what you want
- Recommender systems learn your taste in books, music, movies,...
- Algorithms do automatic stock trading
- Google Translate learns how to translate text
- Siri learns to understand speech
- DeepMind beats humans at Go
- Cars drive themselves
- Smart-watches monitor your health
- Election campaigns use algorithmically targeted ads to influence voters
- Data-driven discoveries are made in physics, biology, genetics, astronomy, chemistry, neurology,...
- ...

THE WORLD OF ARTIFICIAL INTELLIGENCE

... and the connections to Machine Learning and Deep Learning



Many people are confused what these terms actually mean.

And what does all this have to do with statistics?

ARTIFICIAL INTELLIGENCE

- AI is a general term for a very large and rapidly developing field.
- There is no strict definition of AI, but it's often used when machines are trained to perform on tasks which until that time could only be solved by humans or are very difficult and assumed to require "intelligence".
- AI started in the 1940s - when the computer was invented. Scientists like Turing and John von Neumann immediately asked the question: If we can formalize computation, can we use computation to formalize "thinking"?
- AI includes machine learning, natural language processing, computer vision, robotics, planning, search, game playing, intelligent agents, and much more.
- Nowadays, AI is a "hype" term that many people use when they should probably say: ML or ... basic data analysis.

MACHINE LEARNING

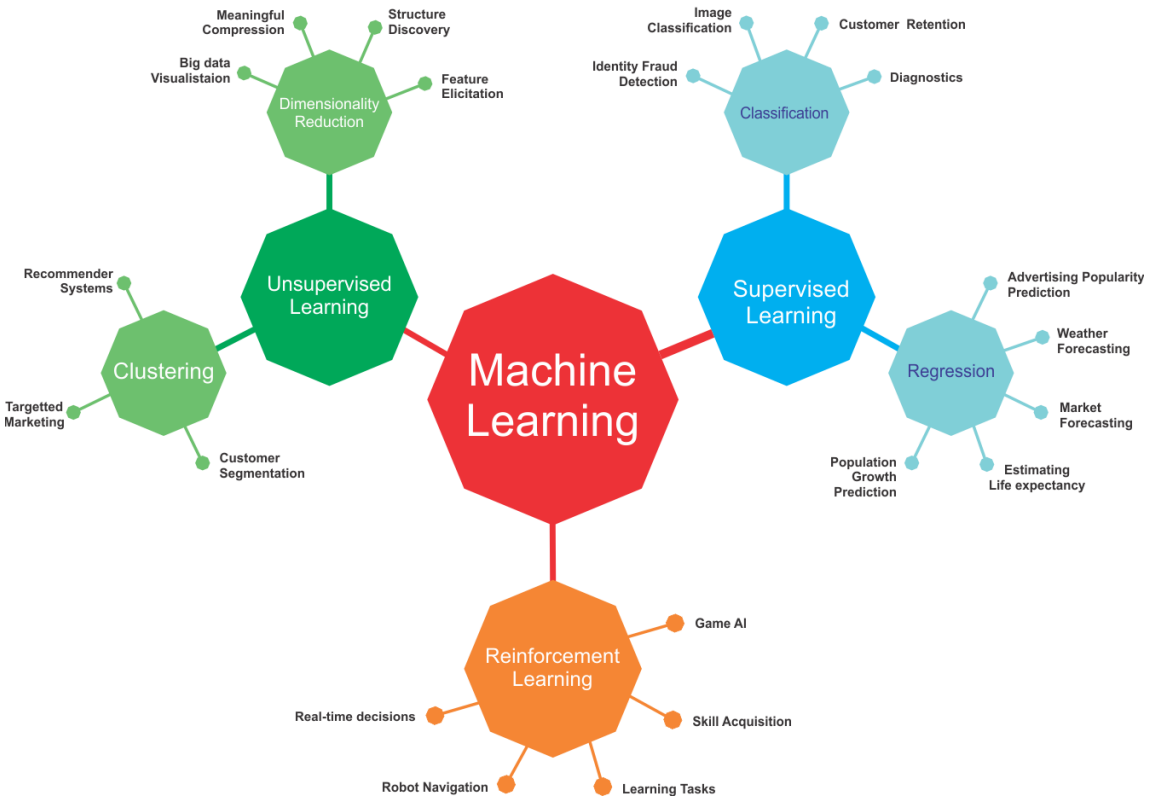


Image via `https://www.oreilly.com/library/view/java-deep-learning/9781788997454/assets/899ceaf3-c710-4675-ae99-33c76cd6ac2f.png`

- Mathematically well-defined and solves reasonably narrow tasks.
- ML algorithms usually construct predictive/decision models from data, instead of explicitly programming them.
- A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.
  Tom Mitchell, Carnegie Mellon University, 1998

DEEP LEARNING

- DL is a subfield of ML which studies neural networks.
- Artificial neural networks (ANNs) might have been (roughly) inspired by the human brain, but they are simply a certain model class of ML.
- ANNs have been studied for decades. DL uses more layers, specific neurons were invented for images and tensors and many computational improvements allow training on large data.
- DL can be used on tabular data, but typical applications are images, texts or signals.
- The last 10-15 years have produced remarkable results and imitations of human ability, where the result looked intelligent.

"Any sufficiently advanced technology is indistinguishable from magic."

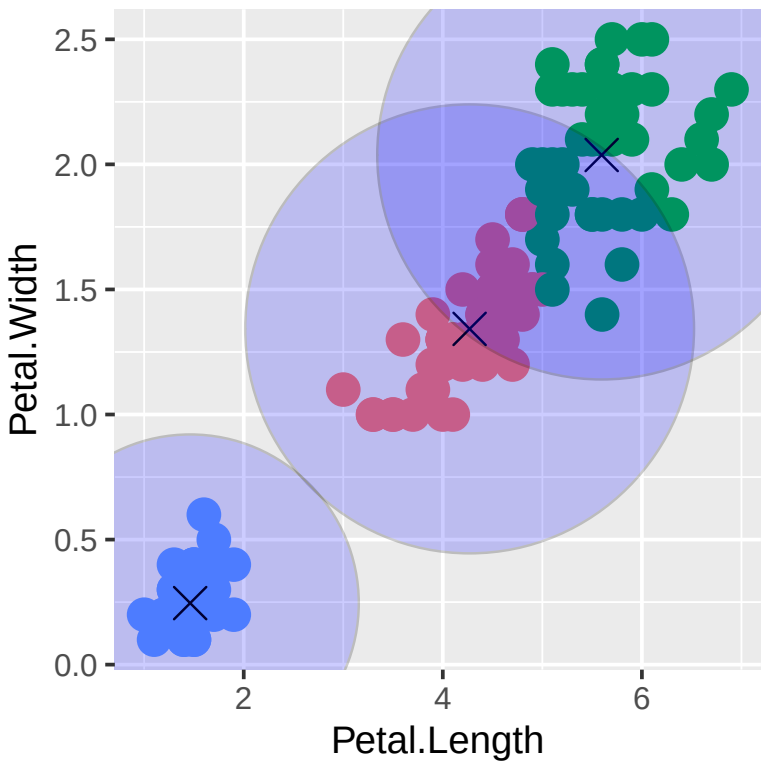
Arthur C. Clarke's 3rd law

ML VS. STATS

- ML and Statistics have historically been developed in different fields, but many methods and especially the mathematical foundations are equivalent.
- Traditionally, models from ML focused more on precise predictions whereas models from statistics focused more on the ability to interpret the patterns that generated the data and the ability to derive sound inference.
- Nowadays, ML and predictive modelling in statistics basically work on the same problems with the same tools.
- Unfortunately, the communities are still divided, don't talk to each other as much as they should and everyone is confused due to different terminology for the same concepts.
- Most parts of ML we could also call:
  Nonparametric statistics plus efficient numerical optimization.

UNSUPERVISED LEARNING

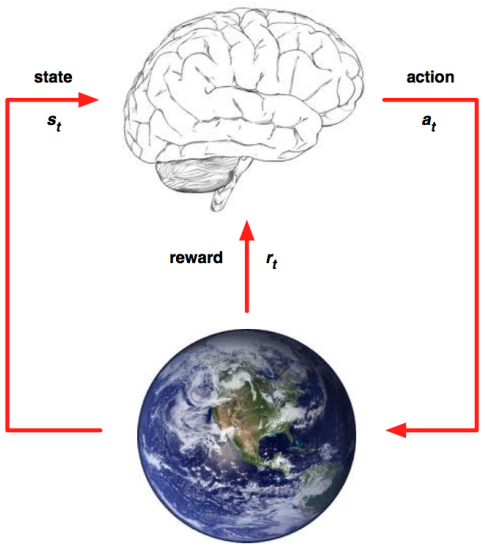
- Data without labels *y*
- Search for patterns within the inputs *x*
- *Unsupervised* as there is no “true” output we can optimize against



- Dimensionality reduction (PCA, Autoencoders ...); compress information in $\mathcal{X}$
- Clustering: group similar observations
- Outlier detection, anomaly detection
- Association rules

REINFORCEMENT LEARNING

RL is a general-purpose framework for AI. At each time step an *agent* interacts with *environment*. It: observes state; receives reward; executes action.



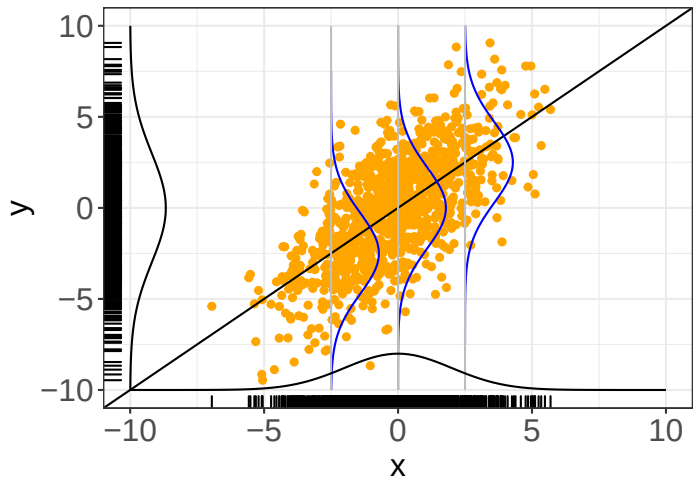
- Goal: Select actions to maximize future reward.
- Reward signals may be sparse, noisy and delayed.

WHAT COMES NEXT

- We will deal with **supervised learning** for regression and classification: predicting labels y based on features x, using patterns that we learned from labeled data.
- First, we will go through fundamental concepts in supervised ML:
  - What kind of "data" do we learn from?
  - How can we formalize the goal of learning?
  - What is a "prediction model"?
  - How can we quantify "predictive performance"?
  - What is a "learning algorithm"
  - How can we operationalize learning?
- We will also look at a couple of fairly simple ML models to obtain a basic understanding.
- More complex stuff comes later.

Einführung in das statistische Lernen

ML-Basics: Data



Learning goals

- Understand structure of tabular data in ML
- Understand difference between target and features
- Understand difference between labeled and unlabeled data
- Know concept of data-generating process

IRIS DATA SET

Introduced by the statistician Ronald Fisher and one of the most frequently used toy examples.

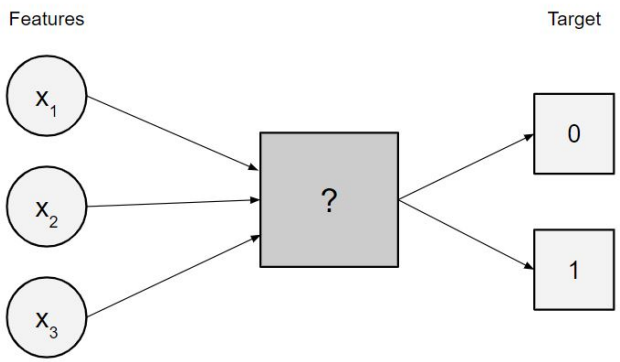
- Classify iris subspecies based on flower measurements.
- 150 iris flowers: 50 versicolor, 50 virginica, 50 setosa.
- Sepal length / width and petal length / width in [cm].

Source: `https://rpubs.com/vidhividhi/irisdataeda`

Word of warning: "iris" is a small, clean, low-dimensional data set, which is very easy to classify; this is not necessarily true in the wild.

DATA IN SUPERVISED LEARNING

- The data we deal with in supervised learning usually consists of observations on different aspects of objects:
  - **Target**: the output variable / goal of prediction
  - **Features**: measurable properties that provide a concise description of the object
- We assume some kind of relationship between the features and the target, in a sense that the value of the target variable can be explained by a combination of the features.



Features x				Target y
Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
4.3	3.0	1.1	0.1	setosa
5.0	3.3	1.4	0.2	setosa
7.7	3.8	6.7	2.2	virginica
5.5	2.5	4.0	1.3	versicolor

ATTRIBUTE TYPES

- Both features and target variables may be of different data types
  - **Numerical** variables can have values in $\mathbb{R}$
  - **Integer** variables can have values in $\mathbb{Z}$
  - **Categorical** variables can have values in $\{C_1, ..., C_g\}$
  - **Binary** variables can have values in $\{0, 1\}$
- For the **target** variable, this results in different tasks of supervised learning: *regression* and *classification*.
- Most learning algorithms can only deal with numerical features, although there are some exceptions (e.g. decision trees can use integers and categoricals without problems). For other feature types, we usually have to pick or create an appropriate encoding.
- If not stated otherwise, we assume numerical features.

OBSERVATION LABELS

- We call the entries of the target column **labels**.
- We distinguish two basic forms our data may come in:
  - For **labeled** data we have already observed the target
  - For **unlabeled** data the target labels are unknown

	Features x				Target y
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
labeled data	4.3	3.0	1.1	0.1	setosa
	5.0	3.3	1.4	0.2	setosa
	7.7	3.8	6.7	2.2	virginica
	5.5	2.5	4.0	1.3	versicolor
unlabeled data	5.9	3.0	5.1	1.8	?
	4.4	3.2	1.3	0.2	?

NOTATION FOR DATA

In formal notation, the data sets we are given are of the following form:

$$\mathcal{D} = \left(\left(\mathbf{x}^{(1)}, y^{(1)} \right), \ldots, \left(\mathbf{x}^{(n)}, y^{(n)} \right) \right) \in (\mathcal{X} \times \mathcal{Y})^n.$$

We call

- $\mathcal{X}$ the input space with $p = \dim(\mathcal{X})$ (for now: $\mathcal{X} \subset \mathbb{R}^p$),
- $\mathcal{Y}$ the output / target space,
- the tuple $\left(\mathbf{x}^{(i)}, y^{(i)} \right) \in \mathcal{X} \times \mathcal{Y}$ the i-th observation,
- $\mathbf{x}_j = \left(x_j^{(1)}, \ldots, x_j^{(n)} \right)^T$ the j-th feature vector.

We denote

- $(\mathcal{X} \times \mathcal{Y})^n$, i.e., the set of all data sets of size n, as $\mathbb{D}_n$,
- $\bigcup_{n \in \mathbb{N}} (\mathcal{X} \times \mathcal{Y})^n$, i.e., the set of all finite data sets, as $\mathbb{D}$.

So we have observed n objects, described by p features.

DATA-GENERATING PROCESS

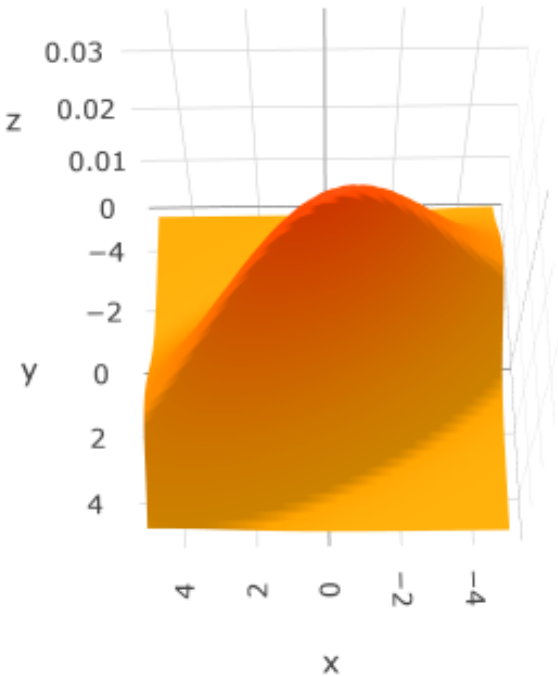
- We assume the observed data $\mathcal{D}$ to be generated by a process that can be characterized by some probability distribution

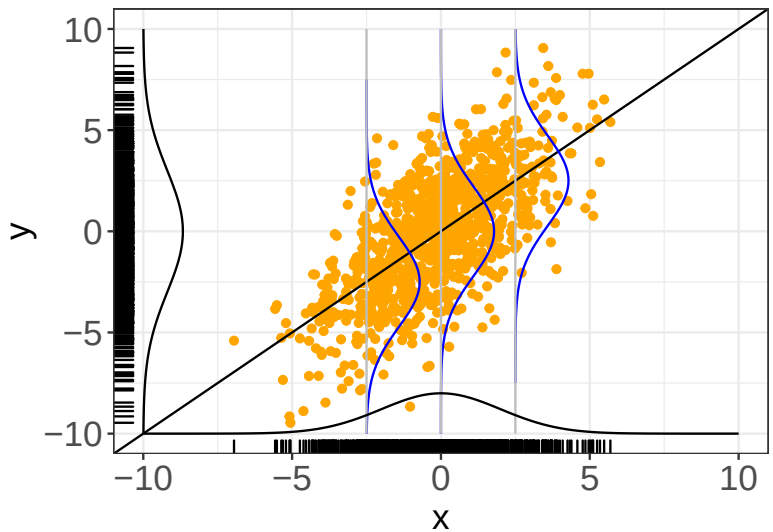
$$\mathbb{P}_{xy},$$

  defined on $\mathcal{X} \times \mathcal{Y}$.
- We denote the random variables following this distribution by lowercase **x** and y.
- It is important to understand that the true distribution is essentially **unknown** to us. In a certain sense, learning (part of) its structure is what ML is all about.

DATA-GENERATING PROCESS

- We assume data to be drawn *i.i.d.* from the joint probability density function (pdf) / probability mass function (pmf) $p(\mathbf{x}, y)$.
  - i.i.d. stands for **i**ndependent and **i**dentically **d**istributed.
  - This means: We assume that all samples are drawn from the same distribution and are mutually independent – the i-th realization does not depend on the other $n - 1$ ones.
  - This is a strong yet crucial assumption that is precondition to most theory in (basic) ML.





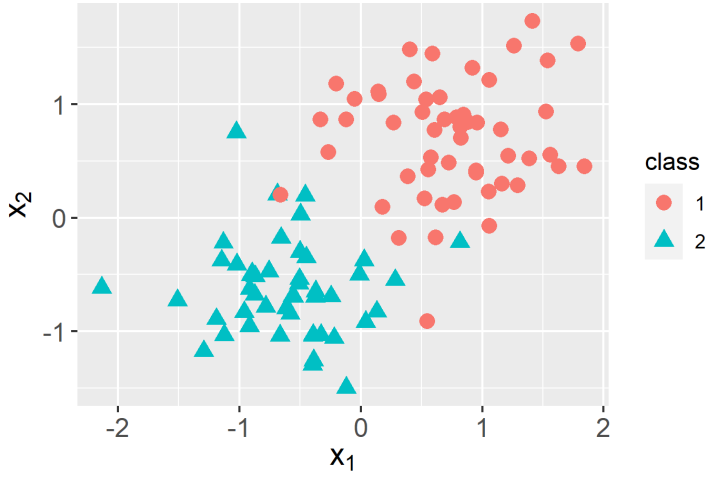
DATA-GENERATING PROCESS

Remarks:

- With a slight abuse of notation we write random variables, e.g., $\mathbf{x}$ and y, in lowercase, as normal variables or function arguments. The context will make clear what is meant.
- Often, distributions are characterized by a parameter vector $\boldsymbol{\theta} \in \Theta$. We then write $p(\mathbf{x}, y \mid \boldsymbol{\theta})$.
- This lecture mostly takes a frequentist perspective. Distribution parameters $\boldsymbol{\theta}$ appear behind the | for improved legibility, not to imply that we condition on them in a probabilistic Bayesian sense. So, strictly speaking, $p(\mathbf{x}|\boldsymbol{\theta})$ should usually be understood to mean $p_{\boldsymbol{\theta}}(\mathbf{x})$ or $p(\mathbf{x}, \boldsymbol{\theta})$ or $p(\mathbf{x}; \boldsymbol{\theta})$. On the other hand, this notation makes it very easy to switch to a Bayesian view.

Einführung in das statistische Lernen

ML-Basics: Supervised Tasks



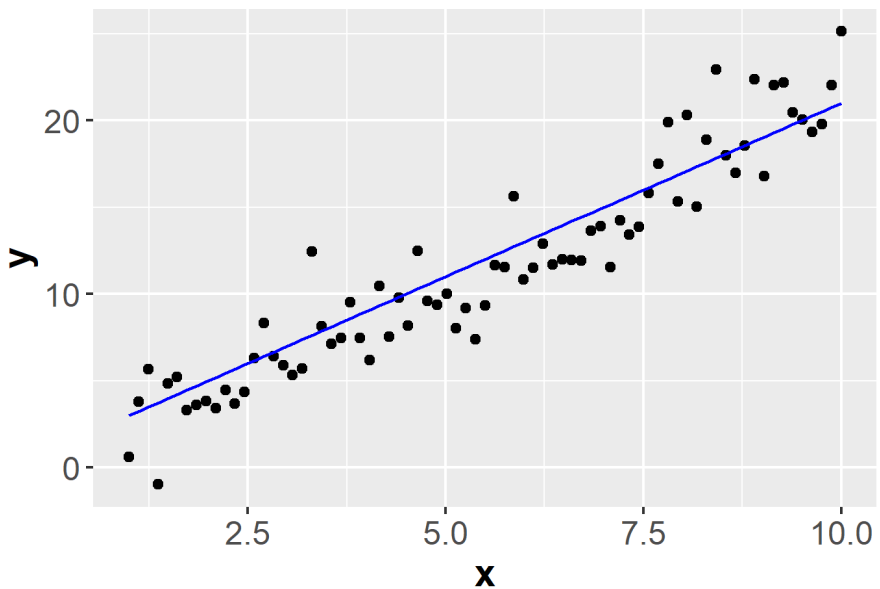
Learning goals

- Know definition and examples of supervised tasks
- Understand the difference between regression and classification

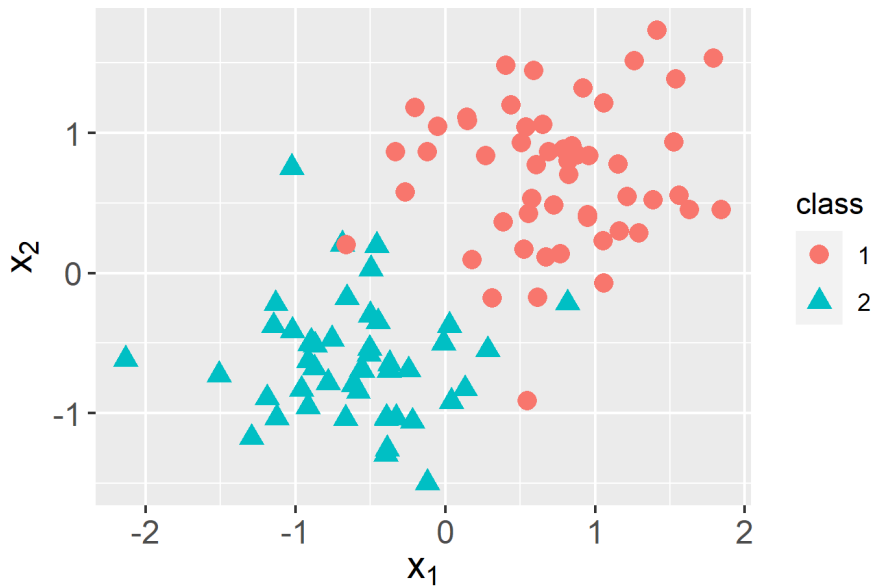
TASKS: REGRESSION VS CLASSIFICATION

- Supervised tasks are data situations where learning the functional relationship between inputs (features) and output (target) is useful.
- The two most basic tasks are regression and classification, depending on whether the target is numerical or categorical.

Regression: Our observed labels come from $\mathcal{Y} \in \mathbb{R}$.



Classification: Observations are categorized: $y \in \mathcal{Y} = \{C_1, ..., C_g\}$.



PREDICT VS. EXPLAIN

We can distinguish two main reasons to learn this relationship:

- **Learning to predict**. In such a case we potentially do not care how our model is structured or whether we can understand it.
  Example: predicting how a stock price will develop.
  Simply being able to use the predictor on new data is of direct benefit to us.
- **Learning to explain**. Here, our model is only a means to a better understanding of the inherent relationship in the data.
  Example: understanding which risk factors influence the probability to get a certain disease. We might not use the learned model on new observations, but rather discuss its implications, in a scientific or social context.

While ML was traditionally more interested in the former, classical statistics addressed the latter. In many tasks nowadays both are relevant – to different degrees.

REGRESSION EXAMPLE: HOUSE PRICES

Predict the price for a house in a certain area

Features x				Target y
square footage of the house	number of bedrooms	swimming pool (yes/no)	...	house price in US$
1,180	3	0	...	221,900
2,570	3	1	...	538,000
770	2	0	...	180,000
1,960	4	1	...	604,000

Probably *learn to explain*. We might want to understand what influences a house price most. But maybe we are also looking for underpriced houses and the predictor is of direct use, too.

REGRESSION EXAMPLE: LENGTH-OF-STAY

Predict days a patient has to stay in hospital at time of admission

Features x					Target y
diagnosis category	**admission type**	**gender**	**age**	**...**	**Length-of-stay in the hospital in days**
heart disease	elective	male	75	...	4.6
injury	emergency	male	22	...	2.6
psychosis	newborn	female	0	...	8
pneumonia	urgent	female	67	...	5.5

Can be *learn to explain*, but *learn to predict* would help a hospital's planning immensely.

CLASSIFICATION EXAMPLE: RISK CATEGORY

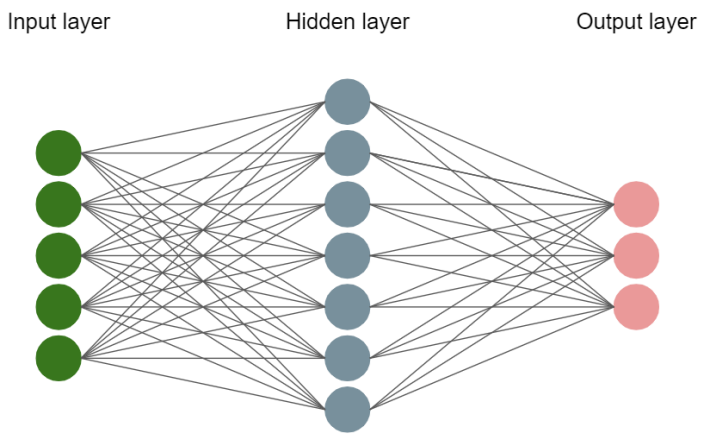
Predict one of five risk categories for a life insurance customer to determine the insurance premium

Features x				Target y
job type	**age**	**smoker**	**...**	**risk group**
carpenter	34	1	...	3
stuntman	25	0	...	5
student	23	0	...	1
white-collar worker	39	0	...	2

Probably *learn to predict*, but the company might be required to explain its predictions to its customers.

Einführung in das statistische Lernen

ML-Basics: Models & Parameters



Learning goals

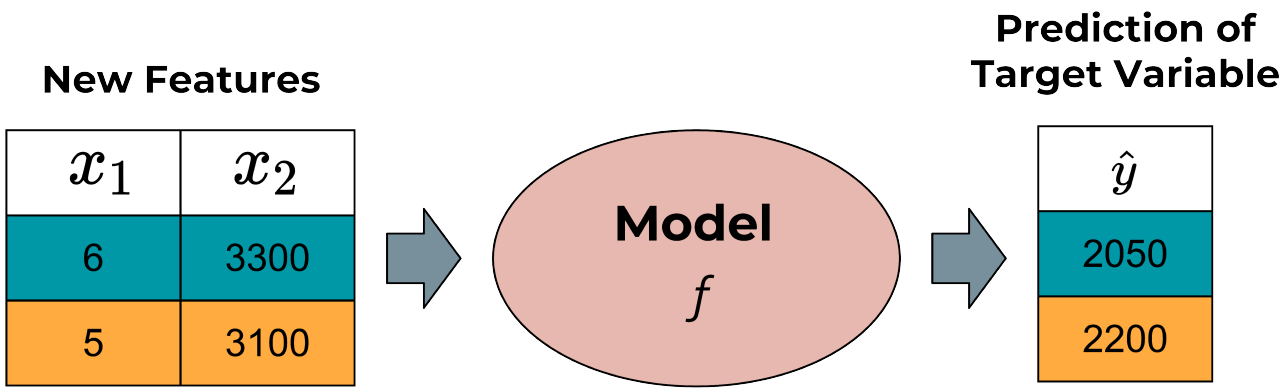
- Understand that an ML model is simply a parametrized curve
- Understand that the hypothesis space lists all admissible models for a learner
- Understand the relationship between the hypothesis space and the parameter space

WHAT IS A MODEL?

- A **model** (or **hypothesis**)

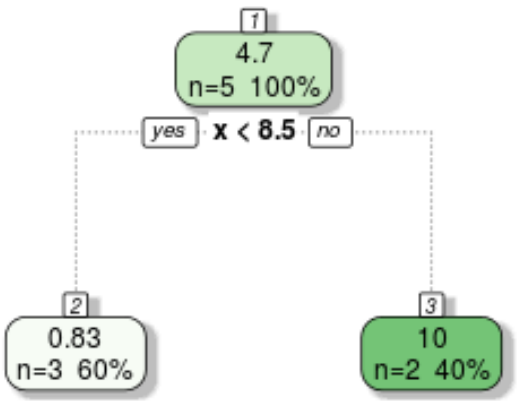
$$f : \mathcal{X} \to \mathbb{R}^g$$

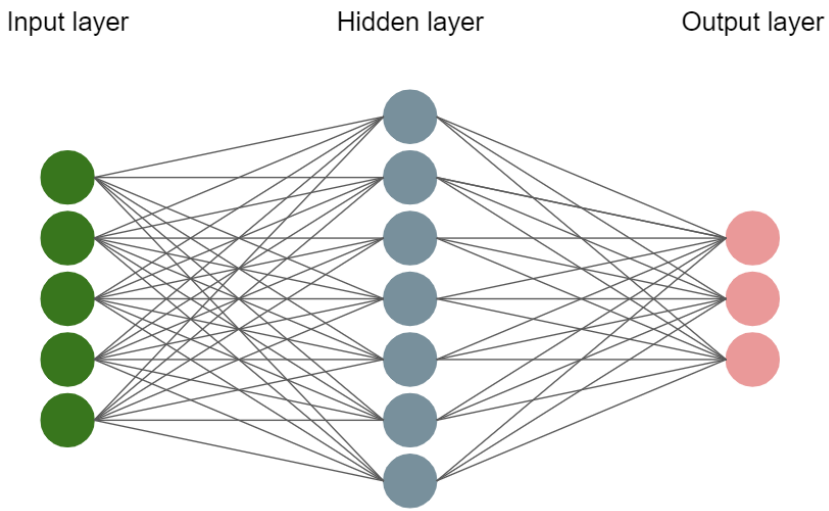
  is a function that maps feature vectors to predicted target values.
- In conventional regression: $g = 1$; for classification g is the number of classes, and output vectors are scores or class probabilities (details later).



WHAT IS A MODEL?

- *f* is meant to capture intrinsic patterns of the data, the underlying assumption being that these hold true for *all* data drawn from $\mathbb{P}_{xy}$.
- It is easily conceivable how models can range from super simple (e.g., linear, tree stumps) to very complex (e.g., deep neural networks) and there are infinitely many choices how we can construct such functions.





- In fact, ML requires **constraining** *f* to a certain type of functions.

HYPOTHESIS SPACES

- Without restrictions on the functional family, the task of finding a "good" model among all the available ones is impossible to solve.
- This means: we have to determine the class of our model *a priori*, thereby narrowing down our options considerably. We could call that a **structural prior**.
- The set of functions defining a specific model class is called a **hypothesis space** $\mathcal{H}$:

$$\mathcal{H} = \{f : f \text{ belongs to a certain functional family}\}$$

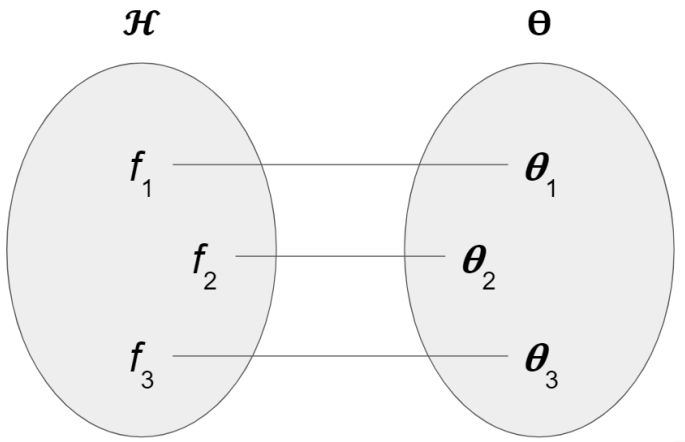
PARAMETRIZATION

- All models within one hypothesis space share a common functional structure. We usually construct the space as **parametrized family of curves**.
- We collect all parameters in a **parameter vector** $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_d)$ from **parameter space** Θ.
- They are our means of fixing a specific function from the family. Once set, our model is fully determined.
- Therefore, we can re-write $\mathcal{H}$ as:

$$\mathcal{H} = \{f_{\boldsymbol{\theta}} : \quad f_{\boldsymbol{\theta}} \text{ belongs to a certain functional family parameterized by } \boldsymbol{\theta}\}$$

PARAMETRIZATION

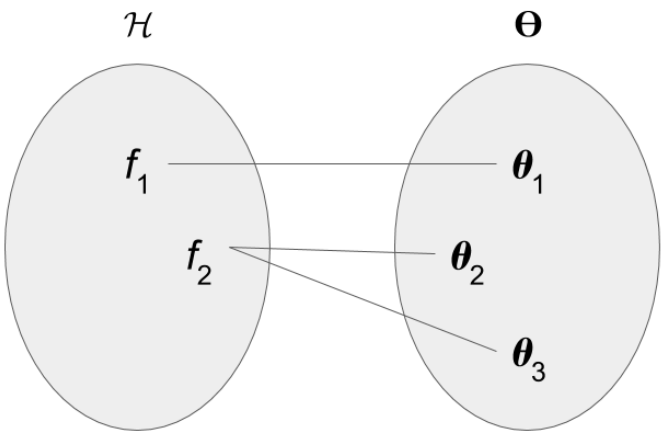
- This means: finding the optimal model is perfectly equivalent to finding the optimal set of parameter values.
- The relation between optimization over $f \in \mathcal{H}$ and optimization over $\boldsymbol{\theta} \in \Theta$ allows us to operationalize our search for the best model via the search for the optimal value on a d-dimensional parameter surface.



- $\boldsymbol{\theta}$ might be scalar or comprise thousands of parameters, depending on the complexity of our model.

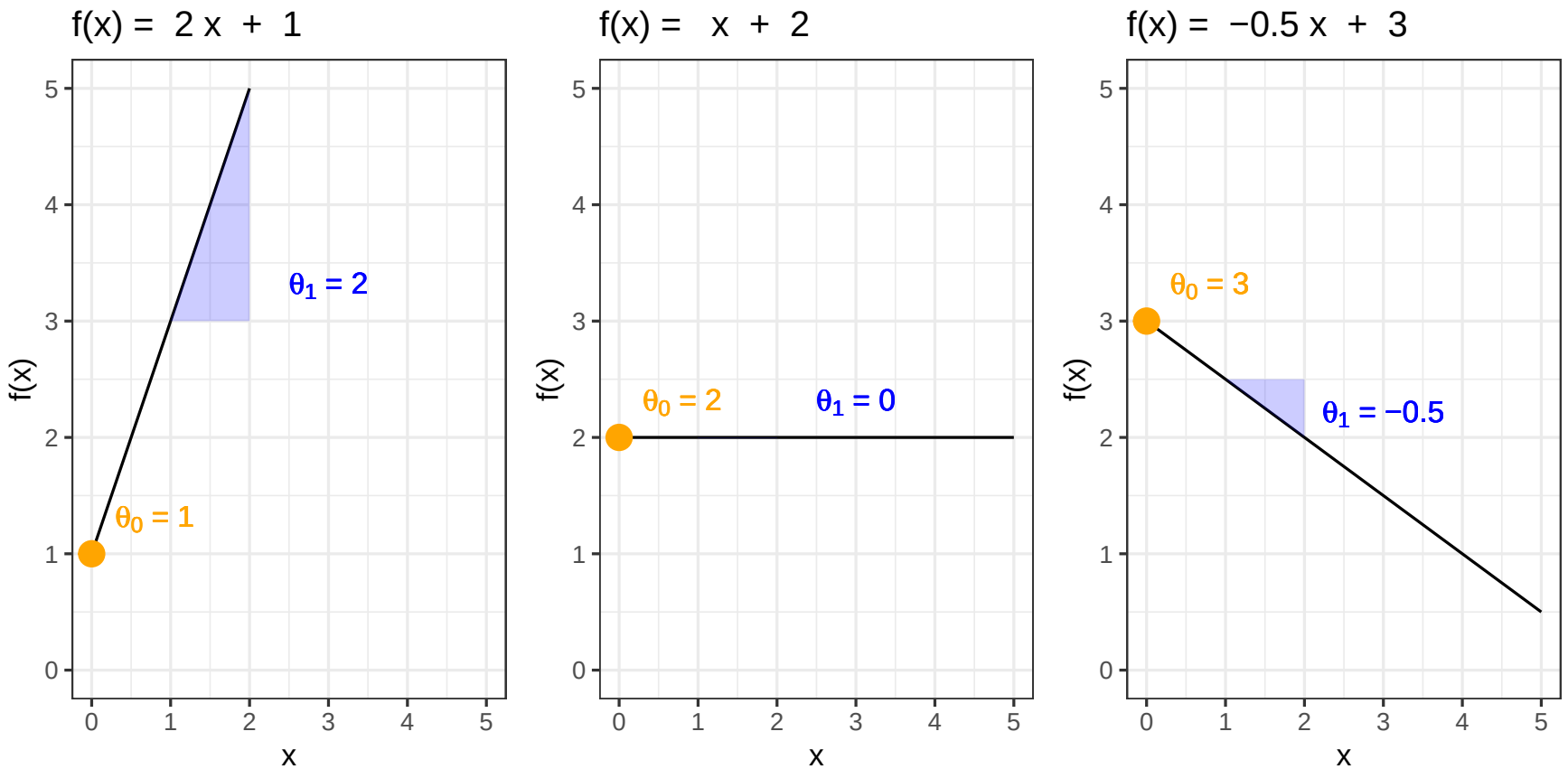
PARAMETRIZATION

- Short remark: In fact, some parameter vectors, for some model classes, might encode the same function. So the parameter-to-model mapping could be non-injective.
- We call this then a non-identifiable model.
- But this shall not concern us here.



EXAMPLE: UNIVARIATE LINEAR FUNCTIONS

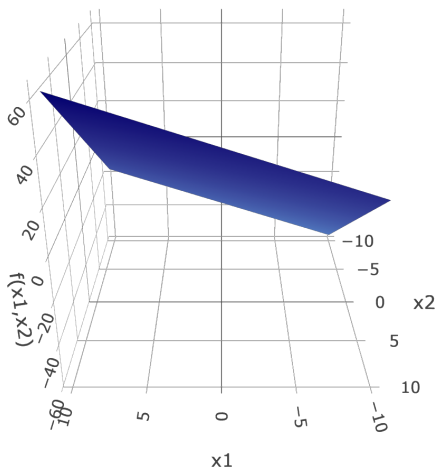
$$\mathcal{H} = \{f : f(\mathbf{x}) = \theta_0 + \theta_1 x, \boldsymbol{\theta} \in \mathbb{R}^2\}$$



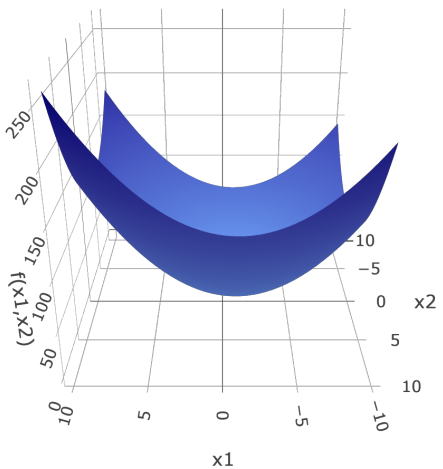
EXAMPLE: BIVARIATE QUADRATIC FUNCTIONS

$$\mathcal{H} = \{f : f(\mathbf{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2, \boldsymbol{\theta} \in \mathbb{R}^6\},$$

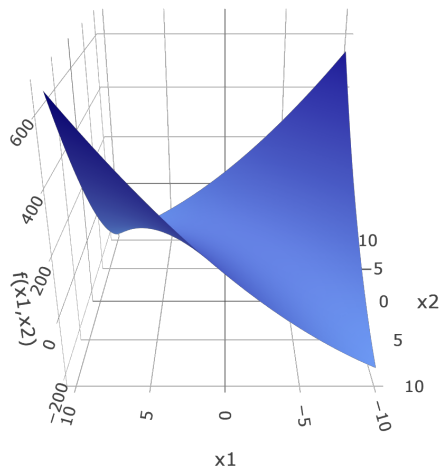
$f(x) = 3 + 2x_1 + 4x_2$



$f(x) = 3 + 2x_1 + 4x_2 +$
$+ 1x_1^2 + 1x_2^2$



$f(x) = 3 + 2x_1 + 4x_2 +$
$+ 1x_1^2 + 1x_2^2 + 4x_1 x_2$



EXAMPLE: RBF NETWORK

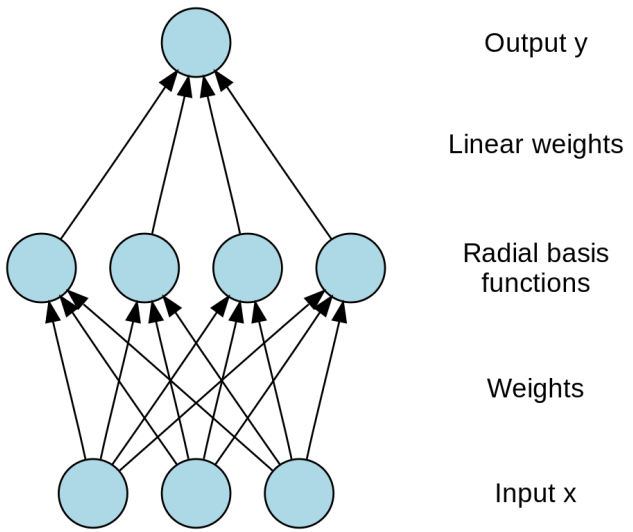
Radial basis function networks with Gaussian basis functions

$$\mathcal{H} = \left\{ f : f(\mathbf{x}) = \sum_{i=1}^{k} a_i \rho(\|\mathbf{x} - \mathbf{c}_i\|) \right\},$$

where

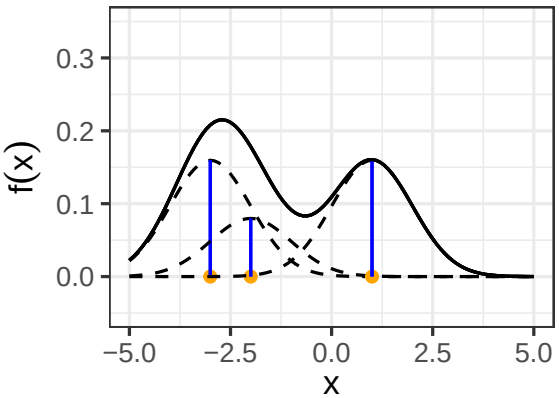
- a_i is the weight of the i-th neuron,
- $\mathbf{c}_i$ its center vector, and
- $\rho(\|\mathbf{x} - \mathbf{c}_i\|) = \exp(-\beta\|\mathbf{x} - \mathbf{c}_i\|^2)$ is the i-th radial basis function with bandwidth $\beta \in \mathbb{R}$.

Usually, the number of centers k and the bandwidth β need to be set in advance (so-called *hyperparameters*).

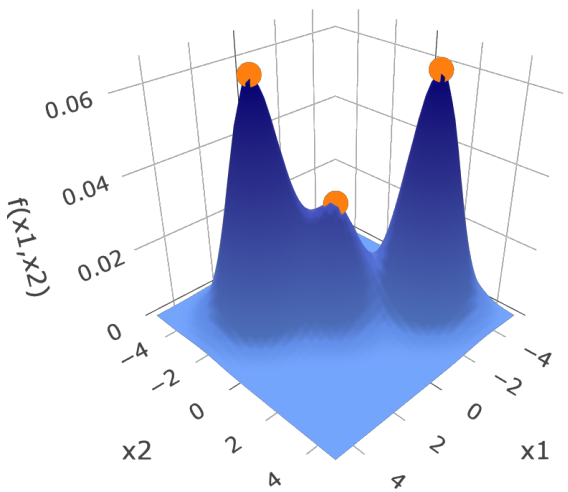


EXAMPLE: RBF NETWORK

Exemplary setting

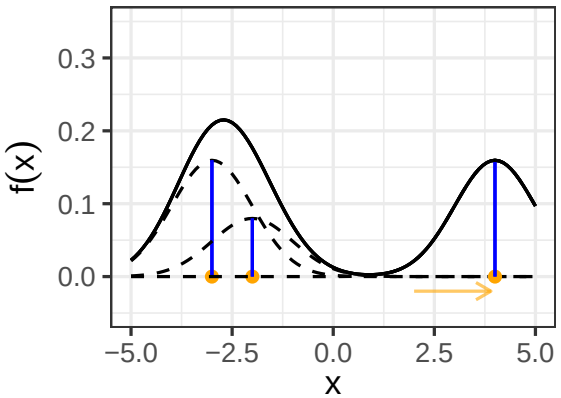


$a_1 = 0.4, a_2 = 0.2, a_3 = 0.4$
$c_1 = -3, c_2 = -2, c_3 = 1$

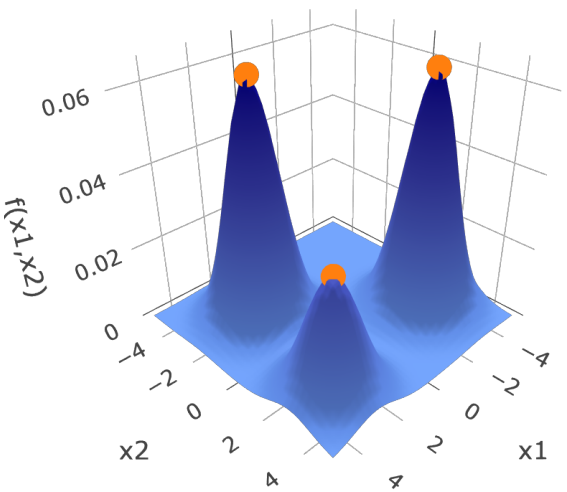


$a_1 = 0.4, a_2 = 0.2, a_3 = 0.4$
$c_1 = (2, -2), c_2 = (0, 0),$
$c_3 = (-3, 2)$

Centers altered

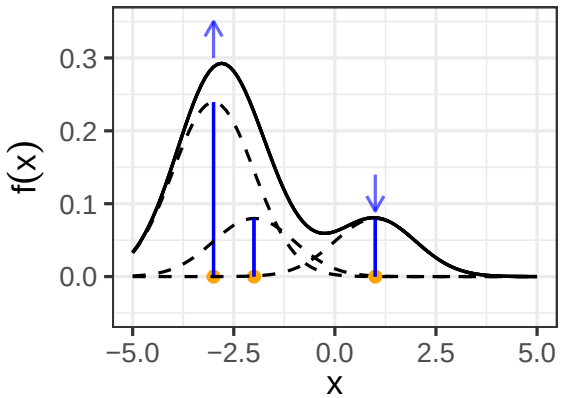


$a_1 = 0.4, a_2 = 0.2, a_3 = 0.4$
$c_1 = -3, c_2 = -2, c_3 = 1$

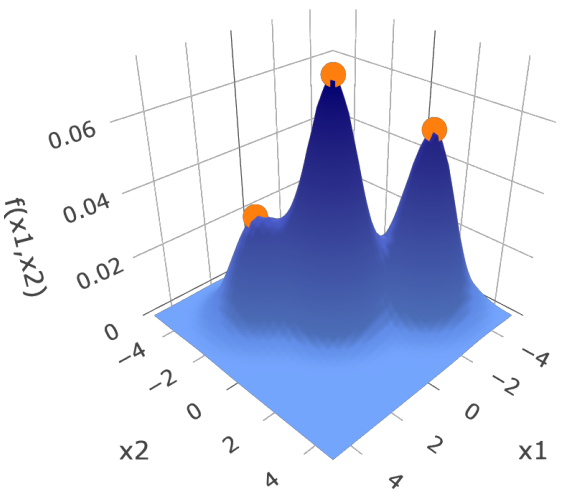


$a_1 = 0.4, a_2 = 0.2, a_3 = 0.4$
$c_1 = (2, -2), c_2 = (3, 3),$
$c_3 = (-3, 2)$

Weights altered



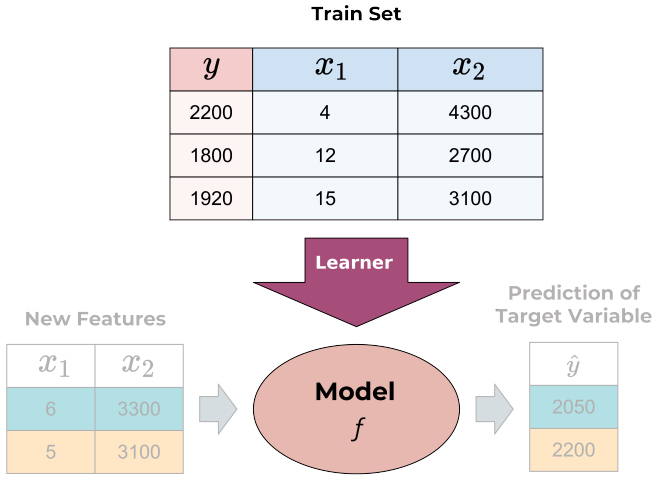
$a_1 = 0.6, a_2 = 0.2, a_3 = 0.2$
$c_1 = -3, c_2 = -2, c_3 = 1$



$a_1 = 0.2, a_2 = 0.45, a_3 = 0.35$
$c_1 = (2, -2), c_2 = (0, 0),$
$c_3 = (-3, 2)$

Einführung in das statistische Lernen

ML-Basics: Learner



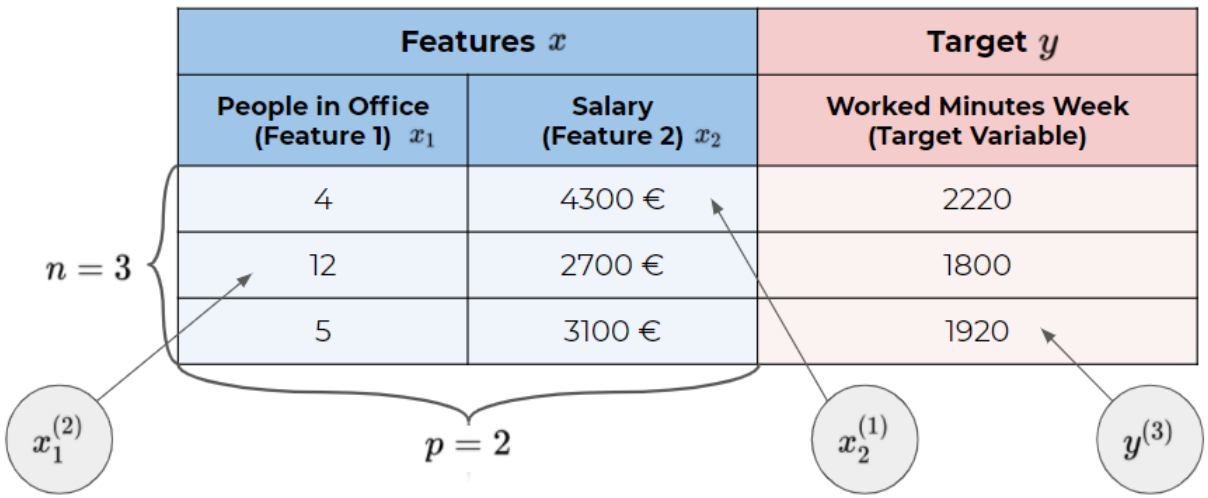
Learning goals

- Understand that a supervised learner fits models automatically from training data

SUPERVISED LEARNING EXAMPLE

Imagine we want to investigate how working conditions affect productivity of employees.

- It is a **regression** task since the target *productivity* is continuous.
- We collect data about worked minutes per week (*productivity*), how many people work in the same office as the employee in question, and the employee's salary.



SUPERVISED LEARNING EXAMPLE

How could we construct a model from these data?

We could investigate the data manually and come up with a simple, hand-crafted rule such as:

- The baseline productivity of an employee with salary 3000 and 7 people in the office is 1850 minutes
- A decrease of 1 person in the office increases productivity by 30
- An increase of the salary by 100 increases productivity by 10

=> Obviously, this is neither feasible nor leads to a good model

IDEA OF SUPERVISED LEARNING

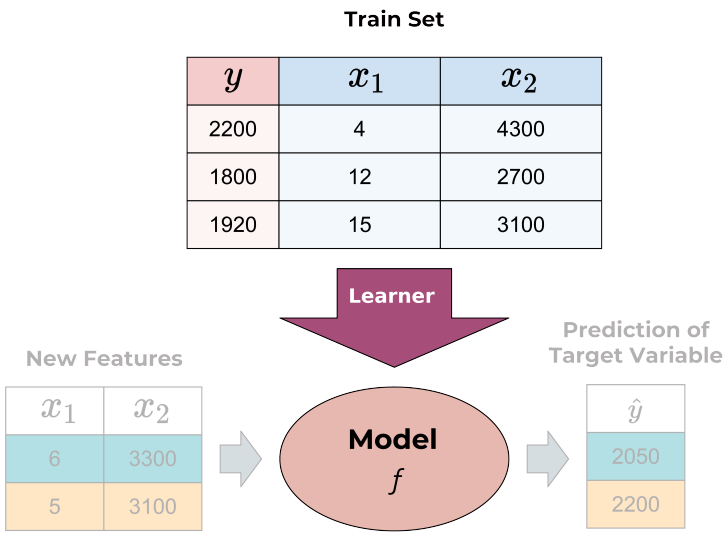
Goal: Automatically identify the fundamental functional relation in the data that maps an object's features to the target.

- **Supervised** learning means we make use of *labeled* data for which we observed the outcome.
- We use the labeled data to learn a model f.
- Ultimately, we use our model to compute predictions for **new** data whose target values are unknown.

LEARNER DEFINITION

- The algorithm for finding our f is called **learner**. It is also called **learning algorithm** or **inducer**.
- We prescribe a certain hypothesis space, the learner is our means of picking the best element from that space for our data set.
- Formally, it maps training data $\mathcal{D} \in \mathbb{D}$ (plus a vector of **hyperparameter** control settings $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$) to a model:

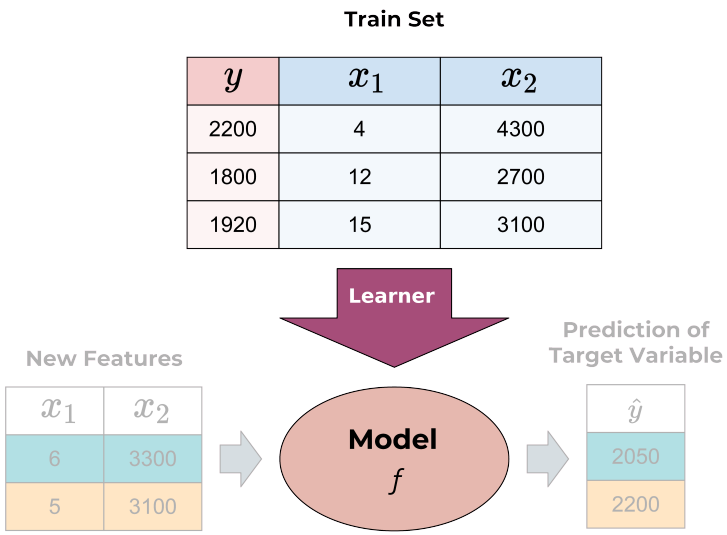
$$\mathcal{I} : \mathbb{D} \times \boldsymbol{\Lambda} \rightarrow \mathcal{H}$$



LEARNER DEFINITION

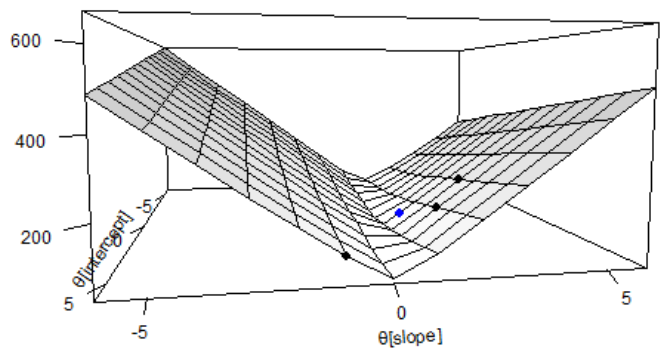
As pseudo-code template it would work like this:

- Learner has a defined model space of parametrized functions $\mathcal{H}$.
- User passes data set $\mathcal{D}_{\text{train}}$ and control settings $\boldsymbol{\lambda}$.
- Learner sets parameters so that model matches data best.
- Optimal parameters $\hat{\boldsymbol{\theta}}$ or function $\hat{f}$ is returned for later usage.



Einführung in das statistische Lernen

ML-Basics: Losses & Risk Minimization



Learning goals

- Know the concept of loss
- Understand the relationship between loss and risk
- Understand the relationship between risk minimization and finding the best model

HOW TO EVALUATE MODELS

- When training a learner, we optimize over our hypothesis space, to find the function which matches our training data best.
- This means, we are looking for a function, where the predicted output per training point is as close as possible to the observed label.

Features x		Target y		Prediction $\hat{y}$
People in Office (Feature 1) x_1	Salary (Feature 2) x_2	Worked Minutes Week (Target Variable)	$\approx^{?}$	Worked Minutes Week (Target Variable)
4	4300 €	2220		2588
12	2700 €	1800		1644
5	3100 €	1920		1870

$\mathcal{D}_{\text{train}}$

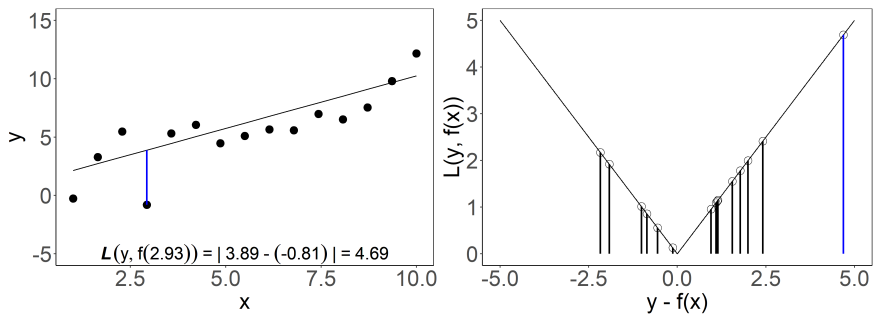
- To make this precise, we need to define now how we measure the difference between a prediction and a ground truth label pointwise.

LOSS

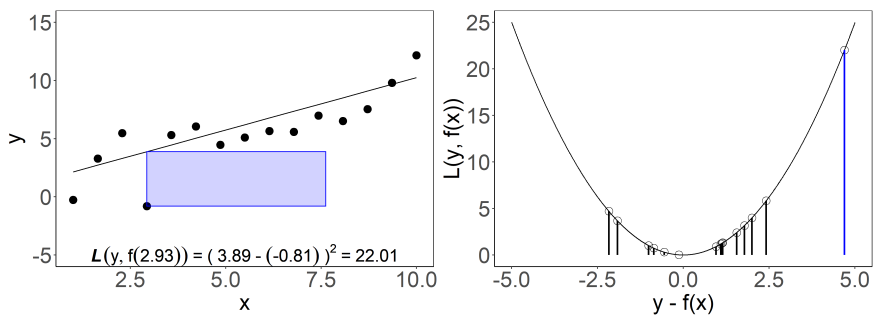
The **loss function** $L(y, f(\mathbf{x}))$ quantifies the "quality" of the prediction $f(\mathbf{x})$ of a single observation $\mathbf{x}$:

$$L : \mathcal{Y} \times \mathbb{R}^g \to \mathbb{R}.$$

In regression, we could use the absolute loss $L(y, f(\mathbf{x})) = |f(\mathbf{x}) - y|$;



or the L2-loss $L(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2$:



RISK OF A MODEL

- The (theoretical) **risk** associated with a certain hypothesis $f(\mathbf{x})$ measured by a loss function $L(y, f(\mathbf{x}))$ is the **expected loss**

$$\mathcal{R}(f) := \mathbb{E}_{xy}[L(y, f(\mathbf{x}))] = \int L(y, f(\mathbf{x})) \, \mathrm{d}\mathbb{P}_{xy}.$$

- This is the average error we incur when we use f on data from $\mathbb{P}_{xy}$.
- Goal in ML: Find a hypothesis $f(\mathbf{x}) \in \mathcal{H}$ that **minimizes** risk.

RISK OF A MODEL

Problem: Minimizing $\mathcal{R}(f)$ over f is not feasible:

- $\mathbb{P}_{xy}$ is unknown (otherwise we could use it to construct optimal predictions).
- We could estimate $\mathbb{P}_{xy}$ in non-parametric fashion from the data $\mathcal{D}$, e.g., by kernel density estimation, but this really does not scale to higher dimensions (see "curse of dimensionality").
- We can efficiently estimate $\mathbb{P}_{xy}$, if we place rigorous assumptions on its distributional form, and methods like discriminant analysis work exactly this way.

But as we have n i.i.d. data points from $\mathbb{P}_{xy}$ available we can simply approximate the expected risk by computing it on $\mathcal{D}$.

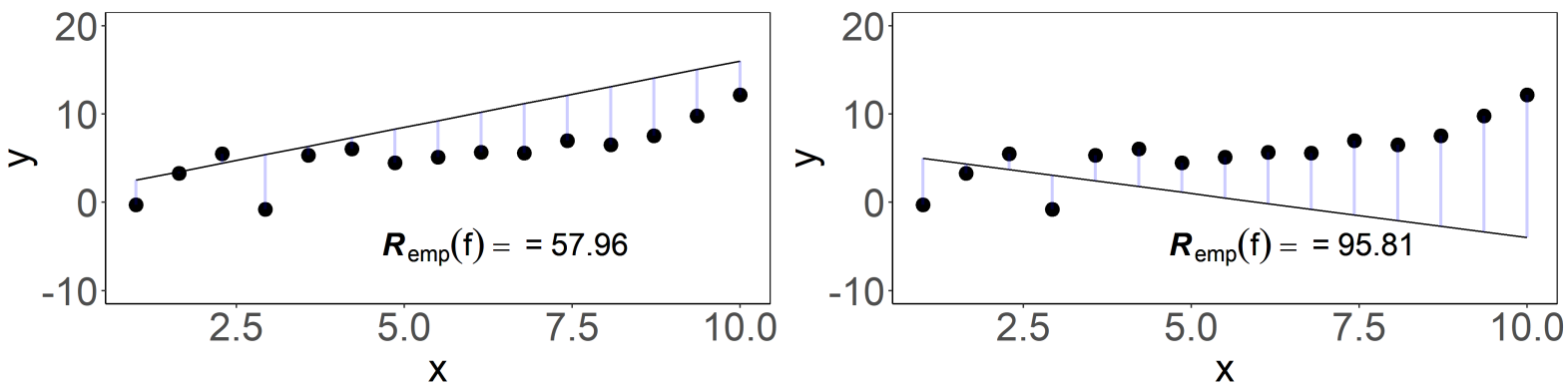
EMPIRICAL RISK

To evaluate, how well a given function f matches our training data, we now simply sum-up all f's pointwise losses.

$$\mathcal{R}_{\text{emp}}(f) = \sum_{i=1}^{n} L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right)$$

This gives rise to the **empirical risk function** which allows us to associate one quality score with each of our models, which encodes how well our model fits our training data.

$$\mathcal{R}_{\text{emp}} : \mathcal{H} \to \mathbb{R}$$



EMPIRICAL RISK

- The risk can also be defined as an average loss

$$\bar{\mathcal{R}}_{\text{emp}}(f) = \frac{1}{n} \sum_{i=1}^{n} L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right).$$

  The factor $\frac{1}{n}$ does not make a difference in optimization, so we will consider $\mathcal{R}_{\text{emp}}(f)$ most of the time.

- Since f is usually defined by **parameters $\boldsymbol{\theta}$**, this becomes:

$$\mathcal{R} : \mathbb{R}^d \to \mathbb{R}$$

$$\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) = \sum_{i=1}^{n} L\left(y^{(i)}, f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)\right)$$

EMPIRICAL RISK MINIMIZATION

The best model is the model with the smallest risk.

If we have a finite number of models f, we could simply tabulate them and select the best.

Model	$\theta_{intercept}$	θ_{slope}	$\mathcal{R}_{\text{emp}}(\theta)$
f_1	2	3	194.62
f_2	3	2	127.12
f_3	6	-1	95.81
f_4	1	1.5	57.96

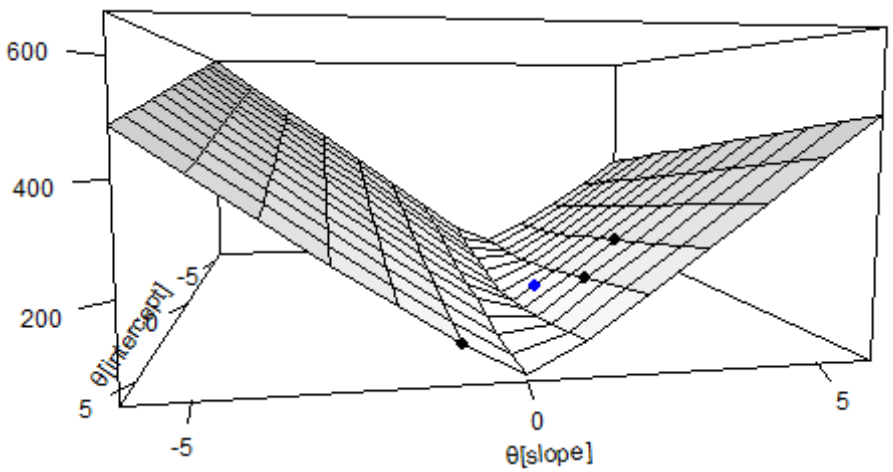
EMPIRICAL RISK MINIMIZATION

But usually $\mathcal{H}$ is infinitely large.

Instead we can consider the risk surface w.r.t. the parameters $\boldsymbol{\theta}$.
(By this I simply mean the visualization of $\mathcal{R}_{\text{emp}}(\boldsymbol{\theta})$)

$$\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) : \mathbb{R}^d \to \mathbb{R}.$$

Model	$\boldsymbol{\theta}_{intercept}$	$\boldsymbol{\theta}_{slope}$	$\mathcal{R}_{\text{emp}}(\boldsymbol{\theta})$
f_1	2	3	194.62
f_2	3	2	127.12
f_3	6	-1	95.81
f_4	1	1.5	57.96



EMPIRICAL RISK MINIMIZATION

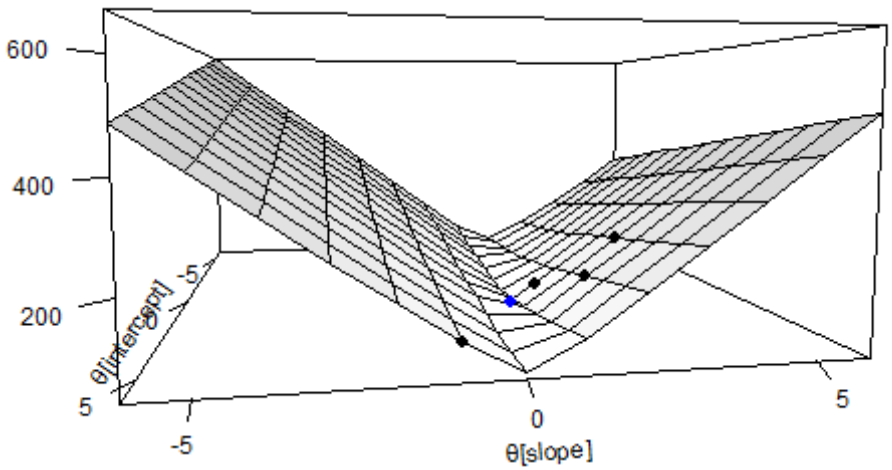
Minimizing this surface is called **empirical risk minimization** (ERM).

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta} \in \Theta} \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}).$$

Usually we do this by numerical optimization.

$$\mathcal{R} : \mathbb{R}^d \to \mathbb{R}.$$

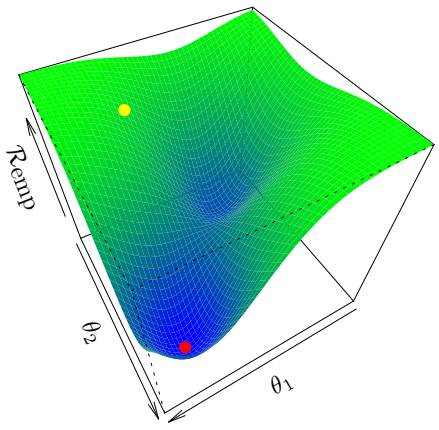
Model	$\theta_{intercept}$	θ_{slope}	$\mathcal{R}_{\text{emp}}(\boldsymbol{\theta})$
f_1	2	3	194.62
f_2	3	2	127.12
f_3	6	-1	95.81
f_4	1	1.5	57.96
f_5	1.25	0.90	23.40



In a certain sense, we have now reduced the problem of learning to **numerical parameter optimization**.

Einführung in das statistische Lernen

ML-Basics: Optimization

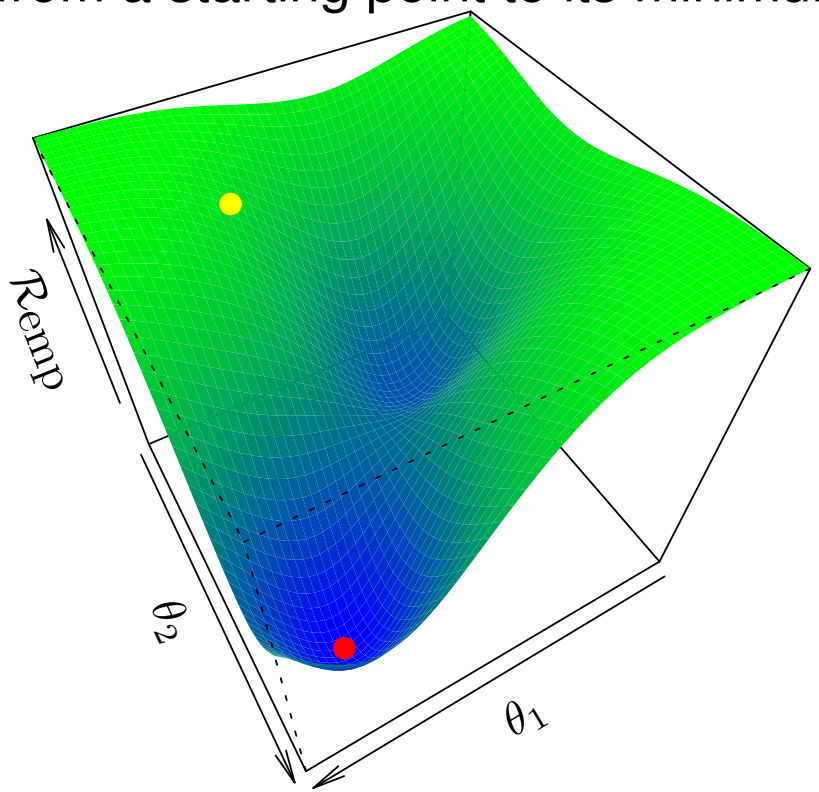


Learning goals

- Understand how the risk function is optimized to learn the optimal parameters of a model
- Understand the idea of gradient descent as a basic risk optimizer

LEARNING AS PARAMETER OPTIMIZATION

- We have seen, we can operationalize the search for a model f that matches training data best, by looking for its parametrization $\boldsymbol{\theta} \in \Theta$ with lowest empirical risk $\mathcal{R}_{\text{emp}}(\boldsymbol{\theta})$.
- Therefore, we usually traverse the error surface downwards; often by local search from a starting point to its minimum.



LEARNING AS PARAMETER OPTIMIZATION

The ERM optimization problem is:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta} \in \Theta} \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}).$$

For a **(global) minimum** $\hat{\boldsymbol{\theta}}$ it obviously holds that

$$\forall \boldsymbol{\theta} \in \Theta : \quad \mathcal{R}_{\text{emp}}(\hat{\boldsymbol{\theta}}) \leq \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}).$$

This does not imply that $\hat{\boldsymbol{\theta}}$ is unique.

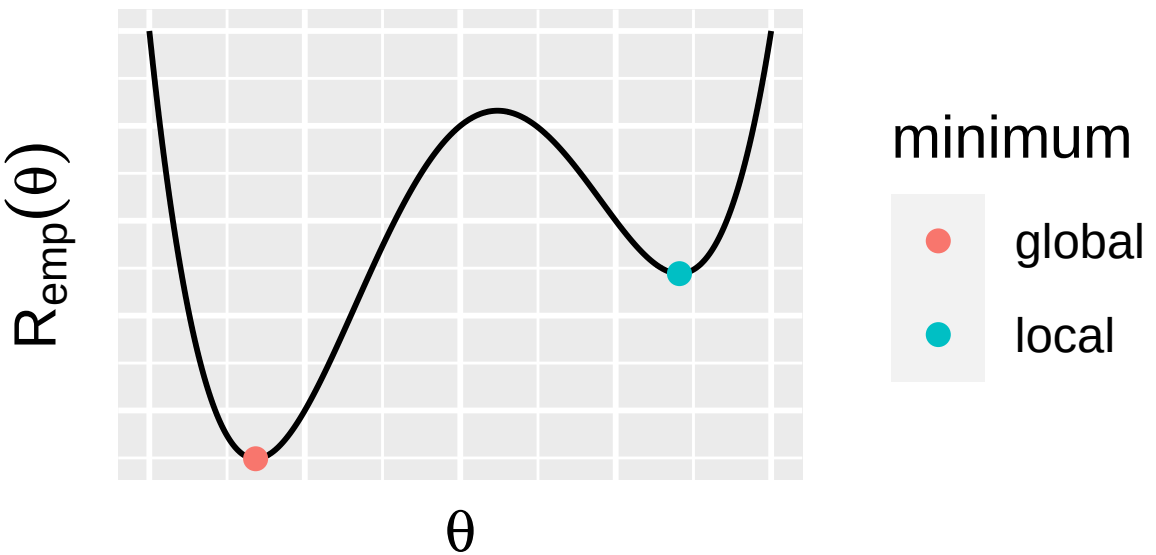
Which kind of numerical technique is reasonable for this problem strongly depends on model and parameter structure (continuous params? uni-modal $\mathcal{R}_{\text{emp}}(\boldsymbol{\theta})$?). Here, we will only discuss very simple scenarios.

LOCAL MINIMA

If $\mathcal{R}_{\text{emp}}$ is continuous in $\boldsymbol{\theta}$ we can define a **local minimum** $\hat{\boldsymbol{\theta}}$:

$$\exists \epsilon > 0 \; \forall \boldsymbol{\theta} \text{ with } \left\| \hat{\boldsymbol{\theta}} - \boldsymbol{\theta} \right\| < \epsilon : \quad \mathcal{R}_{\text{emp}}(\hat{\boldsymbol{\theta}}) \leq \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}).$$

Clearly every global minimum is also a local minimum. Finding a local minimum is easier than finding a global minimum.

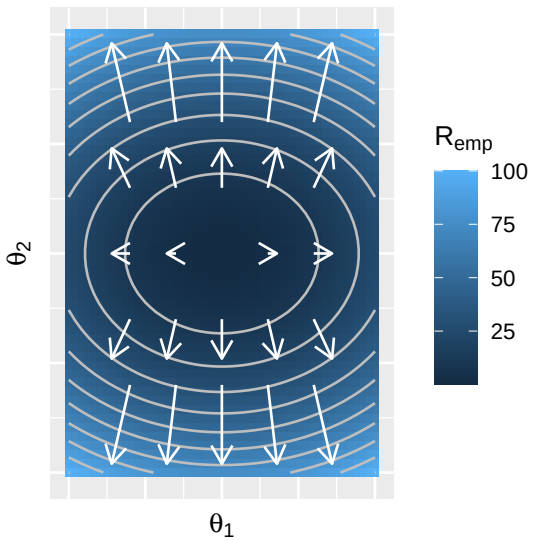


LOCAL MINIMA AND STATIONARY POINTS

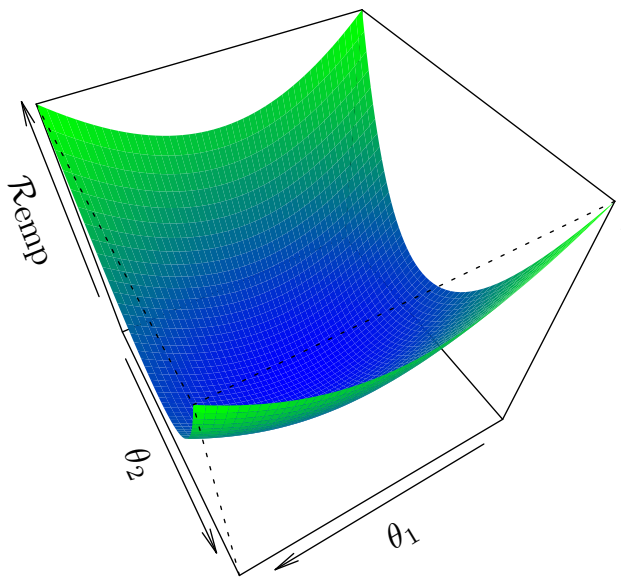
If $\mathcal{R}_{\text{emp}}$ is continuously differentiable in $\boldsymbol{\theta}$ then a **sufficient condition** for a local minimum is that $\hat{\boldsymbol{\theta}}$ is **stationary** with 0 gradient, so no local improvement is possible:

$$\frac{\partial}{\partial \boldsymbol{\theta}} \mathcal{R}_{\text{emp}}(\hat{\boldsymbol{\theta}}) = 0$$

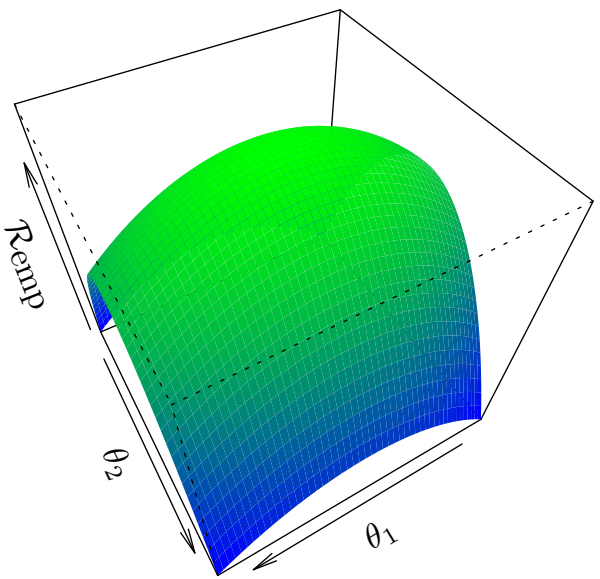
and the Hessian $\frac{\partial^2}{\partial \boldsymbol{\theta}^2} \mathcal{R}_{\text{emp}}(\hat{\boldsymbol{\theta}})$ is positive definite. While the neg. gradient points into the direction of fastest local decrease, the Hessian measures local curvature of $\mathcal{R}_{\text{emp}}$.



$\frac{\partial}{\partial \boldsymbol{\theta}} \mathcal{R}_{\text{emp}}(\boldsymbol{\theta})$



const. pos. def. Hessian



const. neg. def. Hessian

LEAST SQUARES ESTIMATOR

Now, for given features $\mathbf{X} \in \mathbb{R}^{n \times p}$ and target $\mathbf{y} \in \mathbb{R}^n$, we want to find the best linear model regarding the squared error loss, i.e.,

$$\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) = \|\mathbf{X}\boldsymbol{\theta} - \mathbf{y}\|_2^2 = \sum_{i=1}^{n} (\boldsymbol{\theta}^\top \mathbf{x}^{(i)} - y^{(i)})^2 \ .$$

With the sufficient condition for continously differentiable functions it can be shown that the **least squares estimator**

$$\hat{\boldsymbol{\theta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}.$$

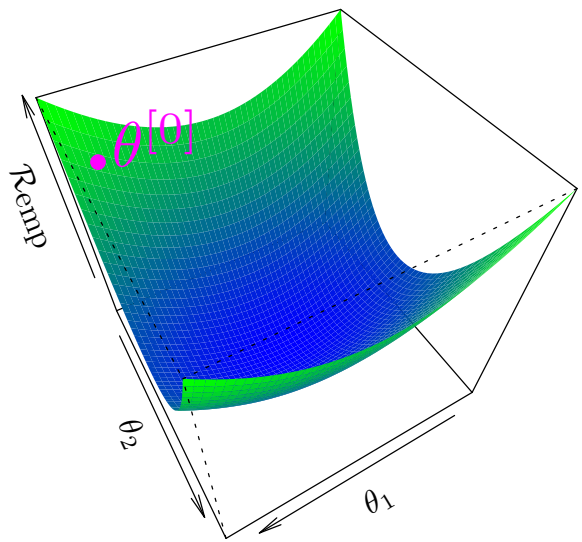
is a local minimum of $\mathcal{R}_{\text{emp}}$. If $\mathbf{X}$ is full-rank, $\mathcal{R}_{\text{emp}}$ is strictly convex and there is only one local minimum - which is also global.

Note: Often such analytical solutions in ML are not possible, and we rather have to use iterative numerical optimization.

GRADIENT DESCENT

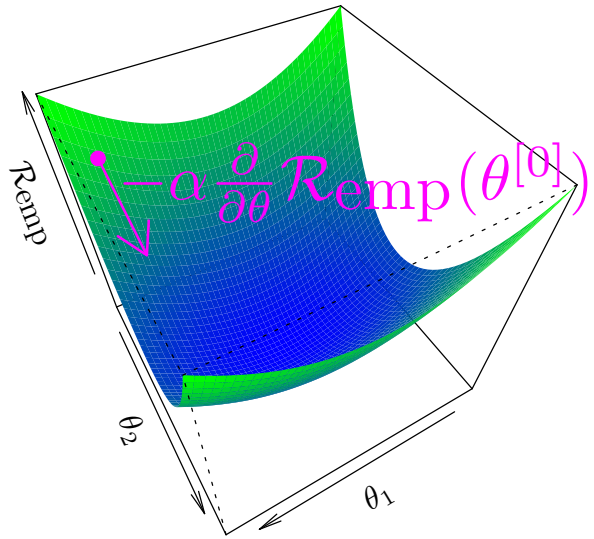
The simple idea of GD is to iteratively go from the current candidate $\boldsymbol{\theta}^{[t]}$ in the direction of the negative gradient, i.e., the direction of the steepest descent, with learning rate α to the next $\boldsymbol{\theta}^{[t+1]}$:

$$\boldsymbol{\theta}^{[t+1]} = \boldsymbol{\theta}^{[t]} - \alpha \frac{\partial}{\partial \boldsymbol{\theta}} \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}^{[t]}).$$

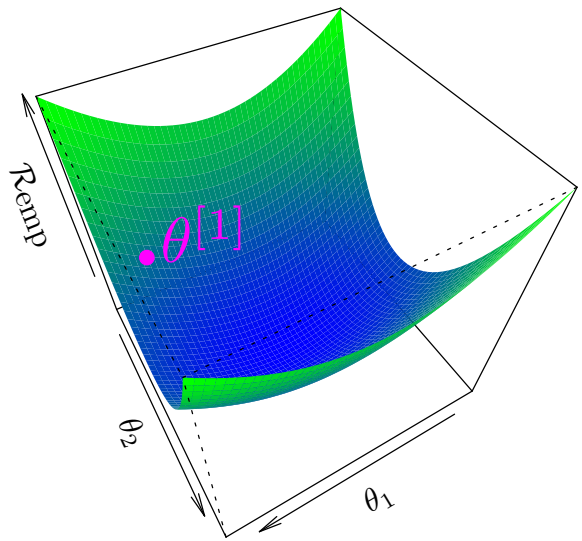


We choose a random start $\boldsymbol{\theta}^{[0]}$ with risk $\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}^{[0]}) = 76.25$.

GRADIENT DESCENT - EXAMPLE

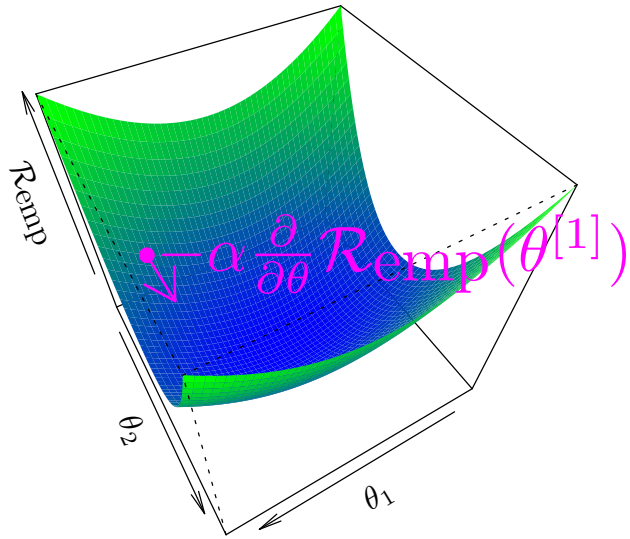


Now we follow in the direction of the negative gradient at $\theta^{[0]}$.

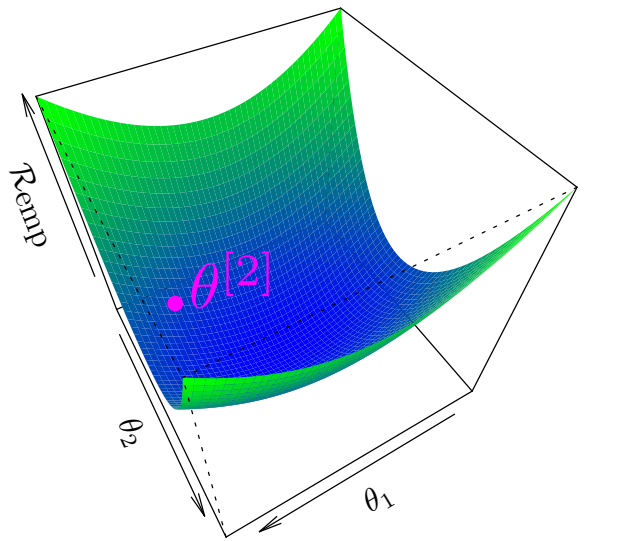


We arrive at $\theta^{[1]}$ with risk
$\mathcal{R}_{\text{emp}}(\theta^{[1]}) \approx 42.73$.
We improved:
$\mathcal{R}_{\text{emp}}(\theta^{[1]}) < \mathcal{R}_{\text{emp}}(\theta^{[0]})$.

GRADIENT DESCENT - EXAMPLE

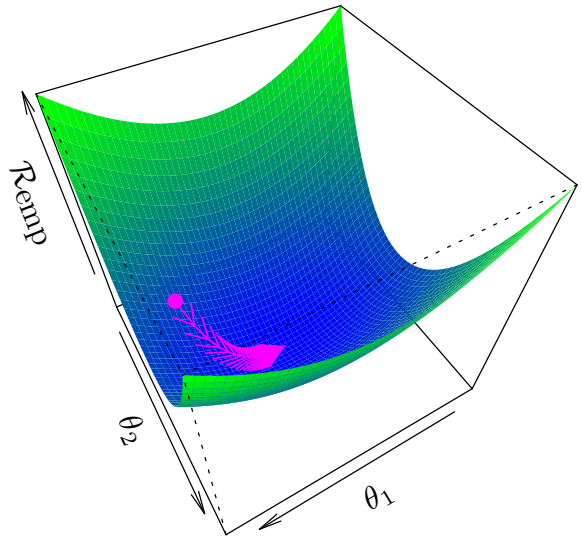


Again we follow in the direction of the negative gradient, but now at $\theta^{[1]}$.

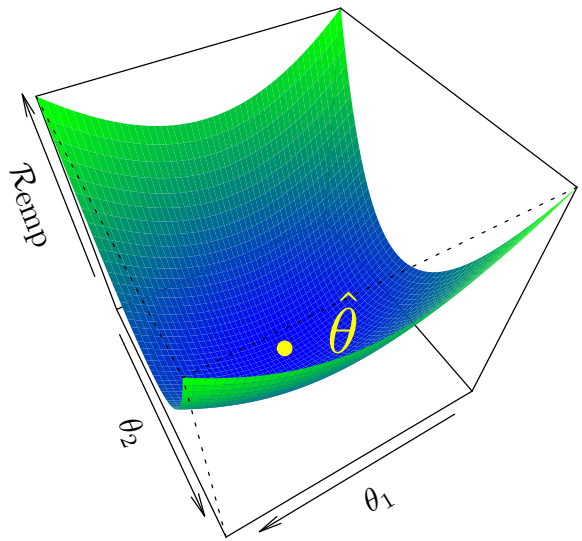


Now $\theta^{[2]}$ has risk $\mathcal{R}_{\text{emp}}(\theta^{[2]}) \approx 25.08$.

GRADIENT DESCENT - EXAMPLE



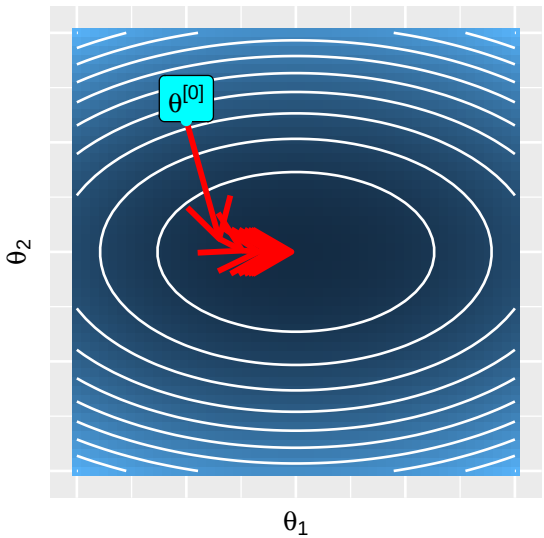
We iterate this until some form of convergence or termination.



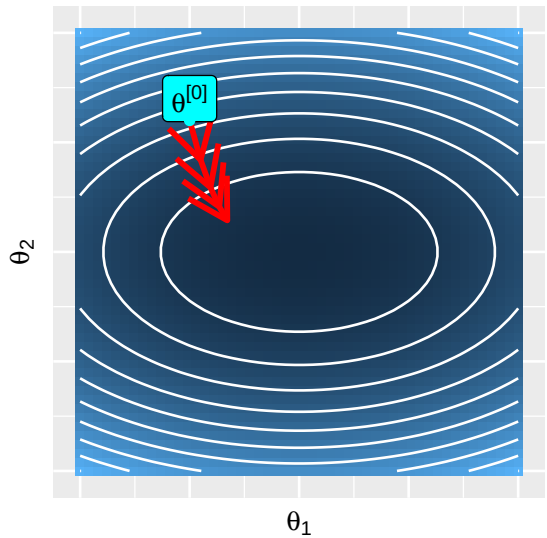
We arrive close to a stationary $\hat{\theta}$ which is hopefully at least a local minimum.

GRADIENT DESCENT - LEARNING RATE

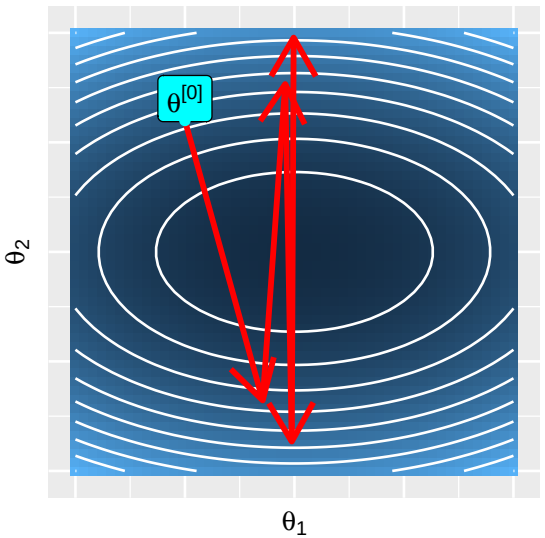
- The negative gradient is a direction that looks locally promising to reduce $\mathcal{R}_{\text{emp}}$.
- Hence it weights components higher in which $\mathcal{R}_{\text{emp}}$ decreases more.
- However, the length of $-\frac{\partial}{\partial \boldsymbol{\theta}} \mathcal{R}_{\text{emp}}$ measures only the local decrease rate, i.e., there are no guarantees that we will not go "too far".
- We use a learning rate α to scale the step length in each iteration. Too much can lead to overstepping and no converge, too low leads to slow convergence.
- Usually, a simple constant rate or rate-decrease mechanisms to enforce local convergence are used



good convergence for α_1



poor convergence for $\alpha_2 (< \alpha_1)$



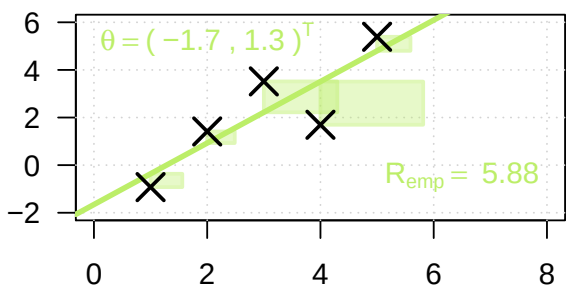
no convergence for $\alpha_3 (> \alpha_1)$

FURTHER TOPICS

- GD is a so-called first-order method. Second-order methods use the Hessian to refine the search direction for faster convergence.
- There exist many improvements of GD, e.g., to smartly control the learn rate, to escape saddle points, to mimic second order behavior without computing the expensive Hessian.
- If the gradient of GD is not derived from the empirical risk of the whole data set, but instead from a randomly selected subset, we call this **stochastic gradient descent** (SGD). For large-scale problems this can lead to higher computational efficiency.

Einführung in das statistische Lernen

ML-Basics: Components of Supervised Learning



Learning goals

- Know the three components of a learner: Hypothesis space, risk, optimization
- Understand that defining these separately is the basic design of a learner
- Know a variety of choices for all three components

COMPONENTS OF SUPERVISED LEARNING

Summarizing what we have seen before, many supervised learning algorithms can be described in terms of three components:

Learning = Hypothesis Space + Risk + Optimization

- **Hypothesis Space:** Defines (and restricts!) what kind of model f can be learned from the data.
- **Risk:** Quantifies how well a specific model performs on a given data set. This allows us to rank candidate models in order to choose the best one.
- **Optimization:** Defines how to search for the best model in the **hypothesis space**, i.e., the model with the smallest **risk**.

COMPONENTS OF SUPERVISED LEARNING

This concept can be extended by the concept of **regularization**, where the model complexity is accounted for in the risk:

Learning = Hypothesis Space + Risk + Optim
Learning = Hypothesis Space + Loss (+ Regularization) + Optim

- For now you can just think of the risk as sum of the losses.
- While this is a useful framework for most supervised ML problems, it does not cover all special cases, because some ML methods are not defined via risk minimization and for some models, it is not possible (or very hard) to explicitly define the hypothesis space.

VARIETY OF LEARNING COMPONENTS

The framework is a good orientation to not get lost here:

Hypothesis Space :
- Step functions
- Linear functions
- Sets of rules
- Neural networks
- Voronoi tesselations
- ...

Risk / **Loss** :
- Mean squared error
- Misclassification rate
- Negative log-likelihood
- Information gain
- ...

Optimization :
- Analytical solution
- Gradient descent
- Combinatorial optimization
- Genetic algorithms
- ...

SUPERVISED LEARNING, FORMALIZED

A **learner** (or **inducer**) $\mathcal{I}$ is a *program* or *algorithm* which

- receives a **training set** $\mathcal{D} \in \mathbb{D}$, and,
- for a given **hypothesis space** $\mathcal{H}$ of **models** $f : \mathcal{X} \to \mathbb{R}^g$,
- uses a **risk** function $\mathcal{R}_{\text{emp}}(f)$ to evaluate $f \in \mathcal{H}$ on $\mathcal{D}$;
  or we use $\mathcal{R}_{\text{emp}}(\boldsymbol{\theta})$ to evaluate f's parametrization $\boldsymbol{\theta}$ on $\mathcal{D}$
- uses an **optimization** procedure to find

$$\hat{f} = \arg\min_{f \in \mathcal{H}} \mathcal{R}_{\text{emp}}(f) \qquad \text{or} \qquad \hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta} \in \Theta} \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}).$$

So the inducer mapping (including hyperparameters $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$) is:

$$\mathcal{I} : \mathbb{D} \times \boldsymbol{\Lambda} \to \mathcal{H}$$

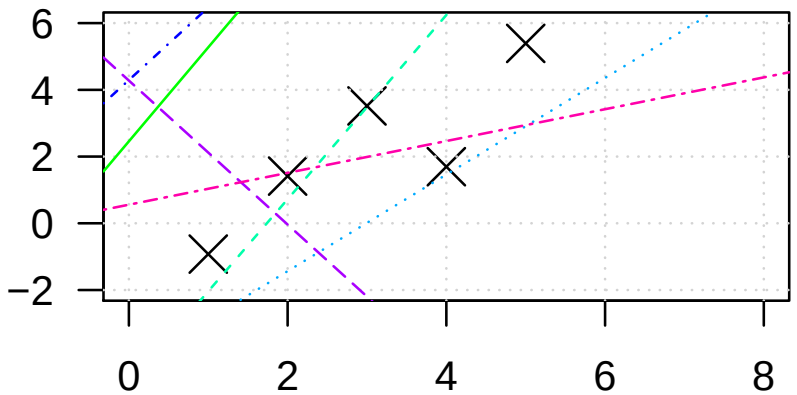
We can also adapt this concept to finding $\hat{\boldsymbol{\theta}}$ for parametric models:

$$\mathcal{I} : \mathbb{D} \times \boldsymbol{\Lambda} \to \Theta$$

EXAMPLE: LINEAR REGRESSION ON 1D

- The **hypothesis space** in univariate linear regression is the set of all linear functions, with $\boldsymbol{\theta} = (\theta_0, \theta_1)^\top$:

$$\mathcal{H} = \{f(\mathbf{x}) = \theta_0 + \theta \mathbf{x} : \theta_0, \theta_1 \in \mathbb{R}\}$$

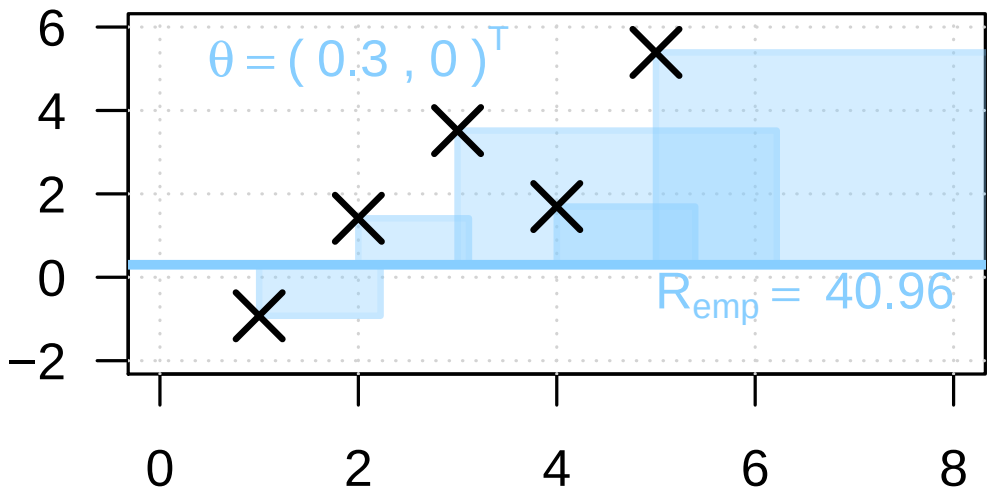


Design choice: We could add more flexibility by allowing polynomial effects or by using a spline basis.

EXAMPLE: LINEAR REGRESSION ON 1D

- We might use the squared error as loss function to our **risk**, punishing larger distances more severely:

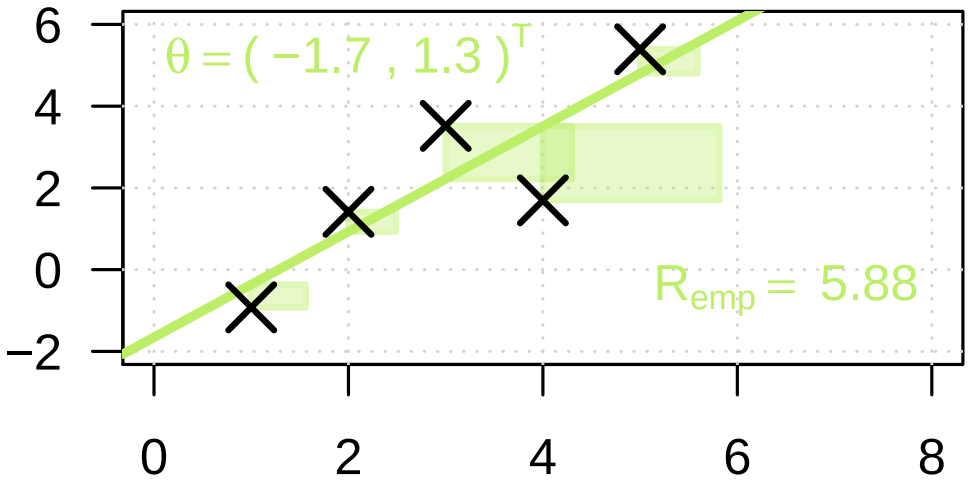
$$\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) = \sum_{i=1}^{n} (y^{(i)} - \theta_0 - \theta_1 \mathbf{x}^{(i)})^2$$

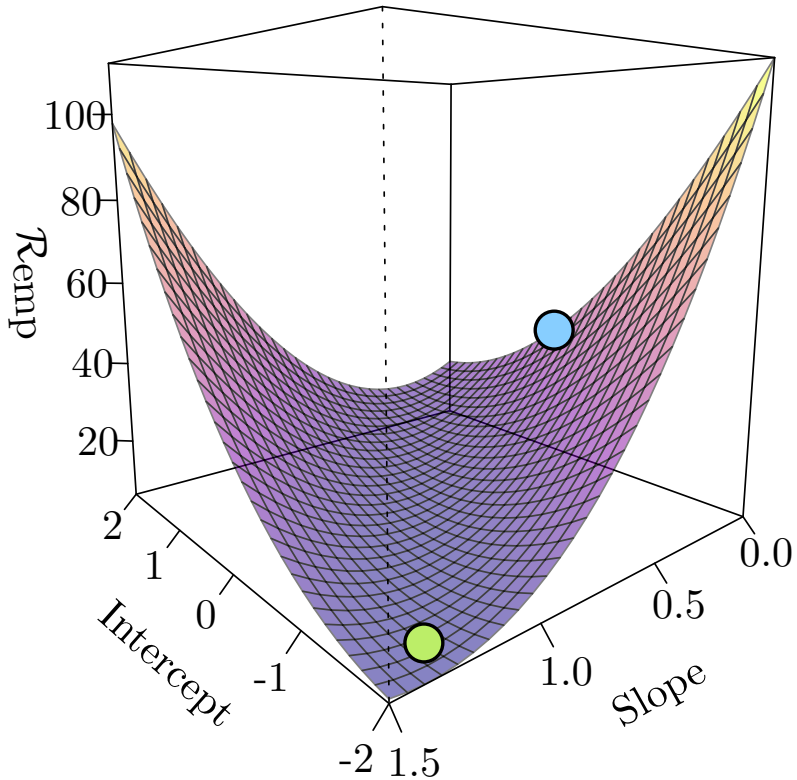


Design choice: Use absolute error / the $L1$ loss to create a more robust model which is less sensitive regarding outliers.

EXAMPLE: LINEAR REGRESSION ON 1D

- **Optimization** will usually mean deriving the ordinary-least-squares (OLS) estimator $\hat{\theta}$ analytically.





Design choice: We could use stochastic gradient descent to scale better to very large or out-of-memory data.

SUMMARY

By decomposing learners into these building blocks:

- we have a framework to better understand how they work,
- we can more easily evaluate in which settings they may be more or less suitable, and
- we can tailor learners to specific problems by clever choice of each of the three components.

Getting this right takes a considerable amount of experience.