



# Introduction to Machine Learning

All slides

July 20, 2021

# INTRODUCTION TO MACHINE LEARNING

## ML Basics

Supervised Regression

Supervised Classification

k-NN

Performance Evaluation

Classification and Regression Trees (CART)

Random Forests

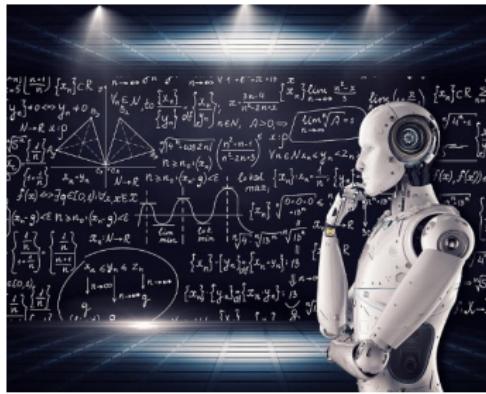
Tuning

Nested Resampling

mlr3

# Introduction to Machine Learning

## ML-Basics: What is Machine Learning?



### 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

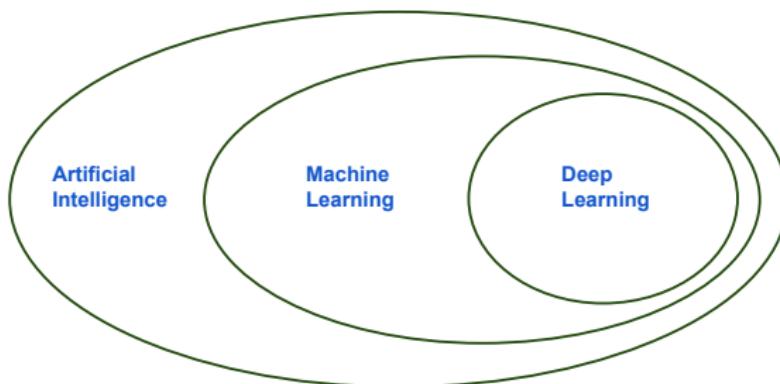
Image via [www.vpnsrus.com](http://www.vpnsrus.com)

# 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



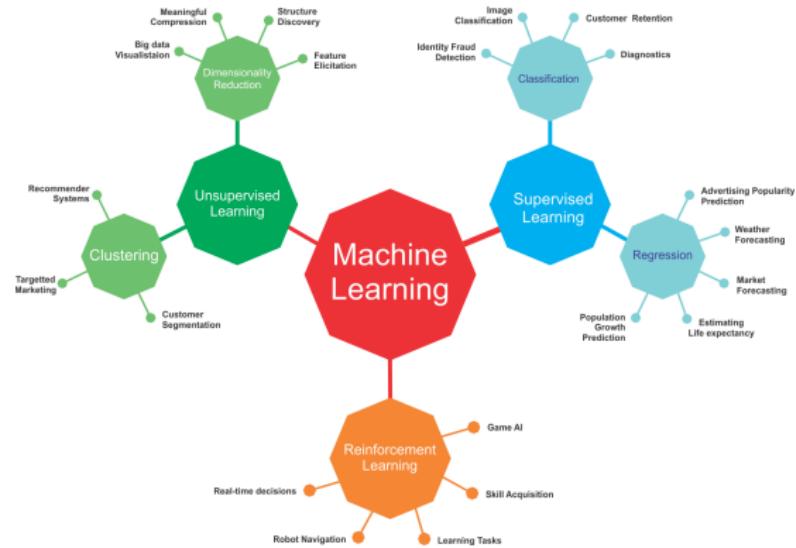
Many people are confused what these terms actually mean.

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



- 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*

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

# 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."

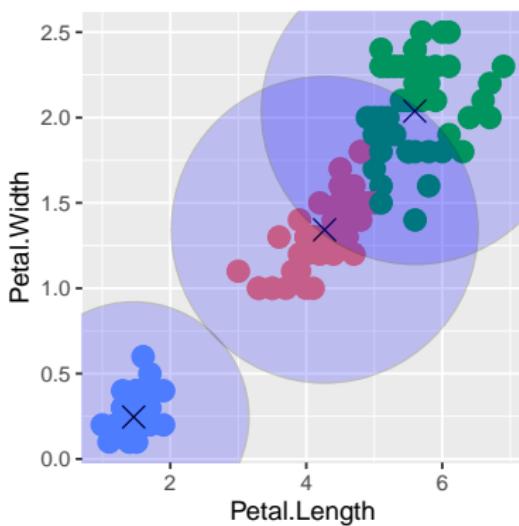
*Arthur C. Clarke's 3rd law*

# 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

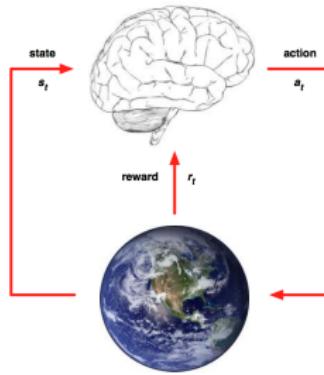
- Data without labels  $y$
- Search for patterns within the inputs  $x$
- *Unsupervised* as there is no “true” output we can optimize against



- 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.



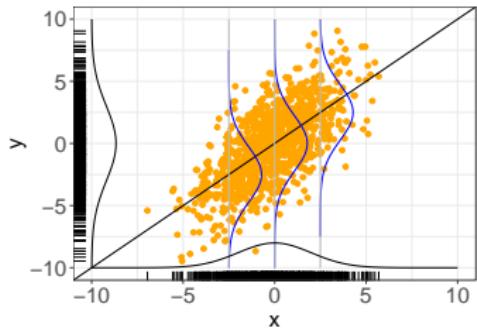
- Goal: Select actions to maximize future reward.
- Reward signals may be sparse, noisy and delayed.

# 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.

# Introduction to Machine Learning

## ML-Basics: Data



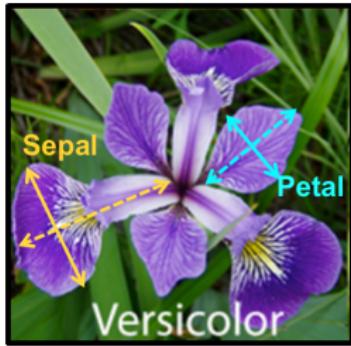
### Learning goals

- 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].

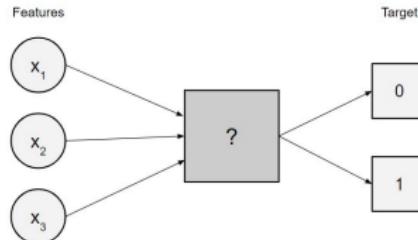


Source: <https://rpubs.com/vidhividhi/irisdataeda>

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  $\{C_1, \dots, C_g\}$
  - **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:

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

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  $(\mathbf{x}^{(i)}, y^{(i)}) \in \mathcal{X} \times \mathcal{Y}$  the  $i$ -th observation,
- $\mathbf{x}_j = \left( x_j^{(1)}, \dots, x_j^{(n)} \right)^T$  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

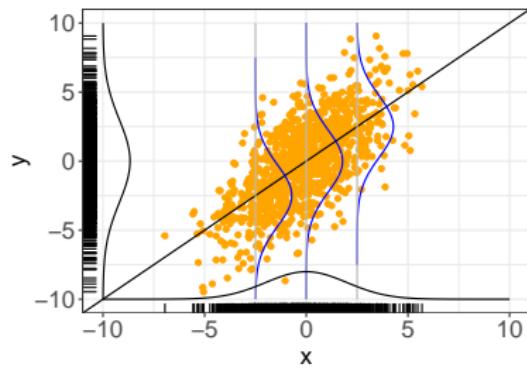
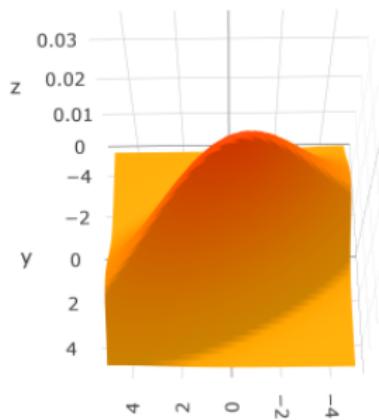
$$\mathbb{P}_{xy},$$

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 **independent** and **identically distributed**.
  - 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.



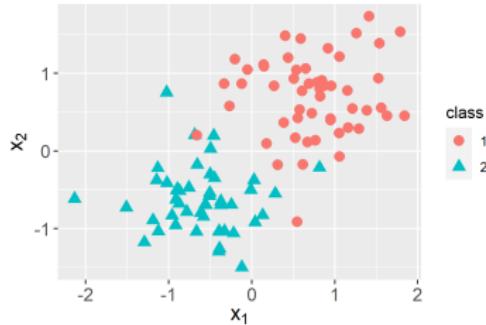
# DATA-GENERATING PROCESS

## 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  $\theta \in \Theta$ . We then write  $p(\mathbf{x}, y | \theta)$ .
- This lecture mostly takes a frequentist perspective. Distribution parameters  $\theta$  appear behind the  $|$  for improved legibility, not to imply that we condition on them in a probabilistic Bayesian sense. So, strictly speaking,  $p(\mathbf{x}|\theta)$  should usually be understood to mean  $p_\theta(\mathbf{x})$  or  $p(\mathbf{x}, \theta)$  or  $p(\mathbf{x}; \theta)$ . On the other hand, this notation makes it very easy to switch to a Bayesian view.

# Introduction to Machine Learning

## ML-Basics: Supervised Tasks



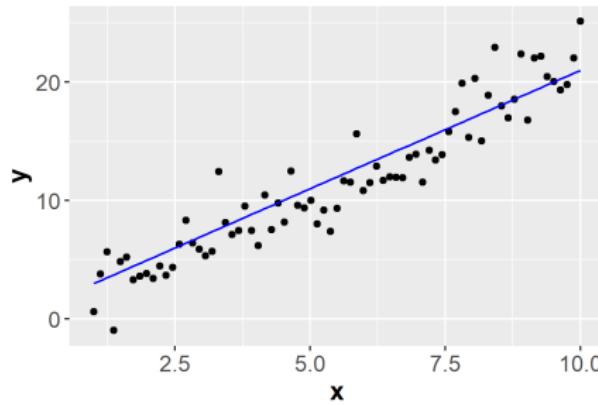
### Learning goals

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

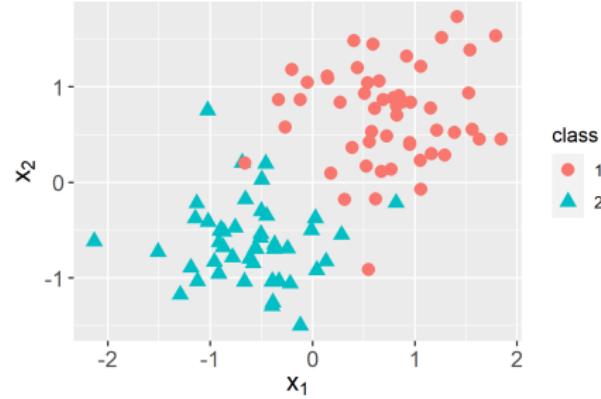
# TASKS: REGRESSION VS CLASSIFICATION

- 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:  $y \in \mathcal{Y} = \{C_1, \dots, C_g\}$ .



# 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

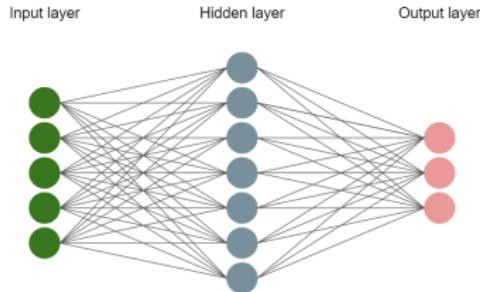


Probably *learn to predict*, but the company might be required to explain its predictions to its customers.

# Introduction to Machine Learning

## 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

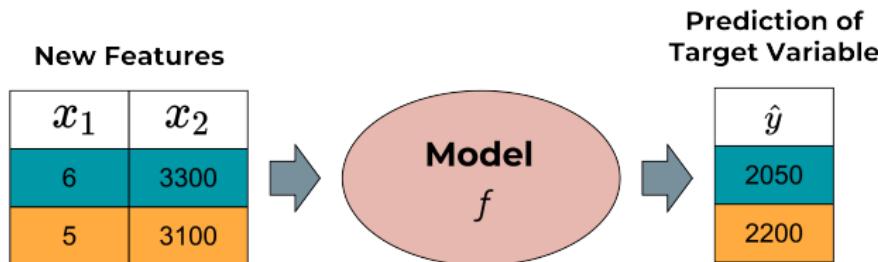
# WHAT IS A MODEL?

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

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

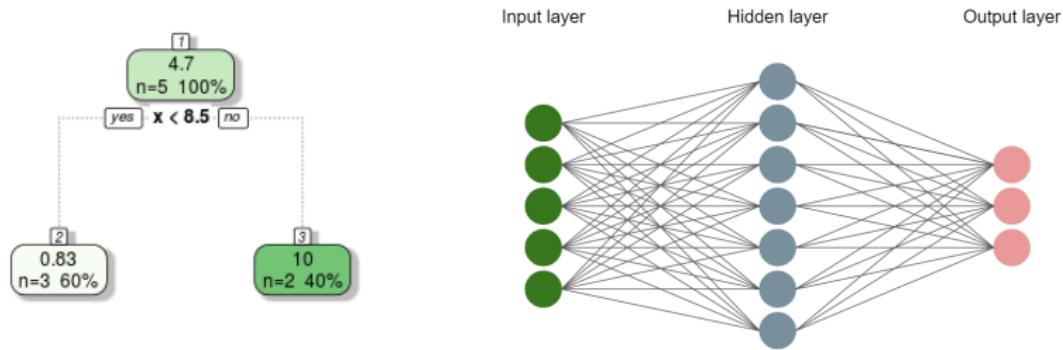
is a function that maps feature vectors to predicted target values.

- 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**  $\mathcal{H}$ :

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

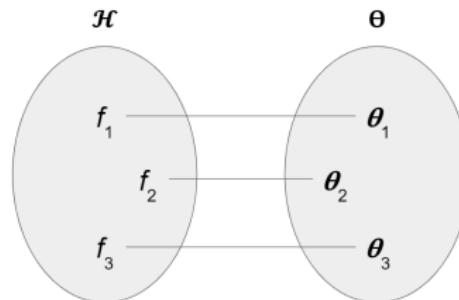
# PARAMETRIZATION

- 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**  $\theta = (\theta_1, \theta_2, \dots, \theta_d)$  from **parameter space**  $\Theta$ .
- 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:

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

# PARAMETRIZATION

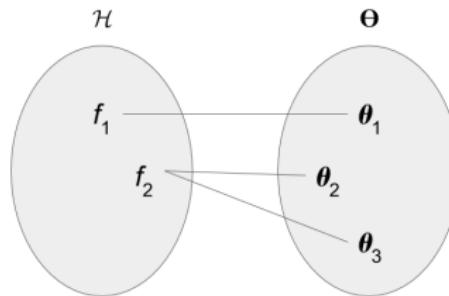
- This means: finding the optimal model is perfectly equivalent to finding the optimal set of parameter values.
- The relation between optimization over  $f \in \mathcal{H}$  and optimization over  $\theta \in \Theta$  allows us to operationalize our search for the best model via the search for the optimal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



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

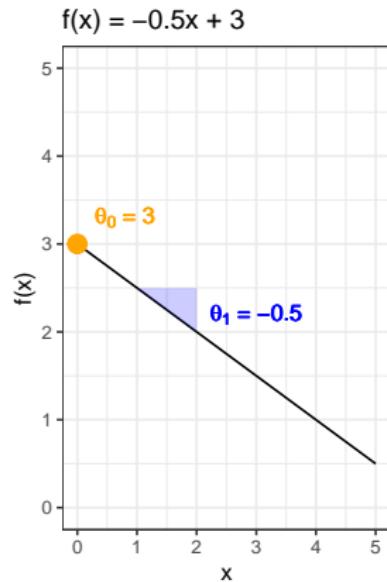
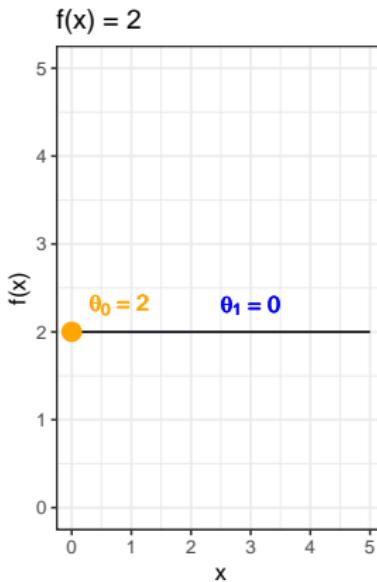
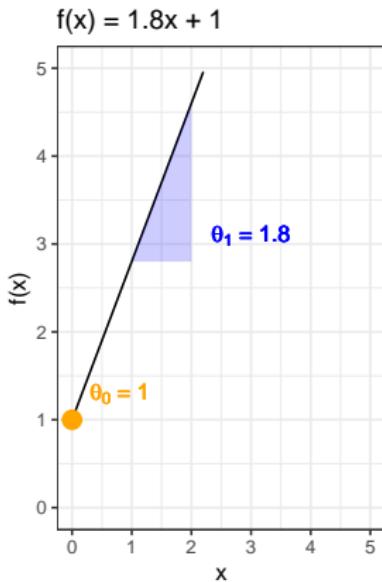
# PARAMETRIZATION

- 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

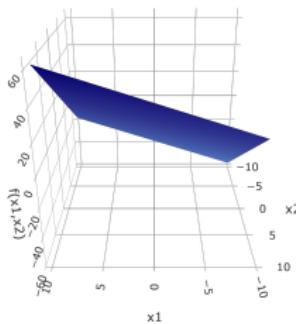
$$\mathcal{H} = \{f : f(\mathbf{x}) = \theta_0 + \theta_1 x, \theta \in \mathbb{R}^2\}$$



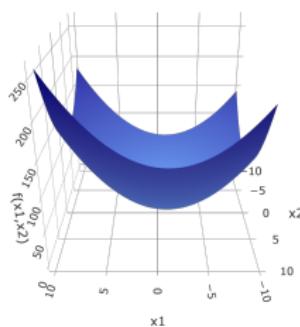
# EXAMPLE: BIVARIATE QUADRATIC FUNCTIONS

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

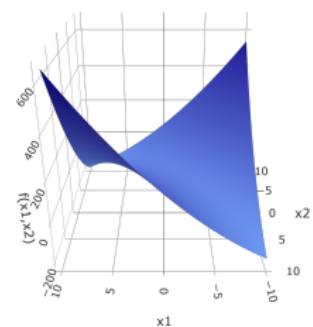
$$f(x) = 3 + 2x_1 + 4x_2$$



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



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



# EXAMPLE: RBF NETWORK

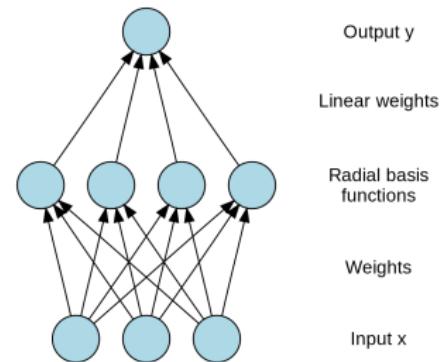
Radial basis function networks with Gaussian basis functions

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

where

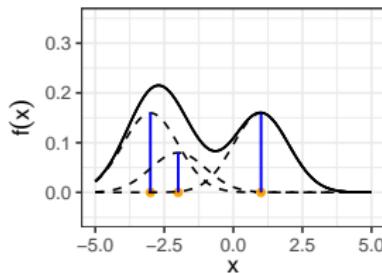
- $a_i$  is the weight of the  $i$ -th neuron,
- $\mathbf{c}_i$  its center vector, and
- $\rho(\|\mathbf{x} - \mathbf{c}_i\|) = \exp(-\beta \|\mathbf{x} - \mathbf{c}_i\|^2)$  is the  $i$ -th radial basis function with bandwidth  $\beta \in \mathbb{R}$ .

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



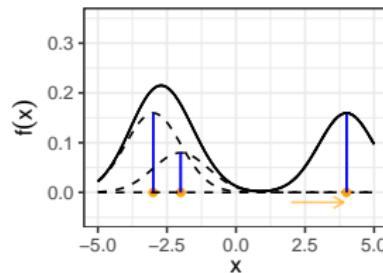
# EXAMPLE: RBF NETWORK

Exemplary setting



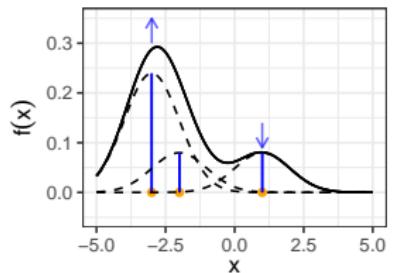
$$a_1 = 0.4, a_2 = 0.2, a_3 = 0.4 \\ c_1 = -3, c_2 = -2, c_3 = 1$$

Centers altered

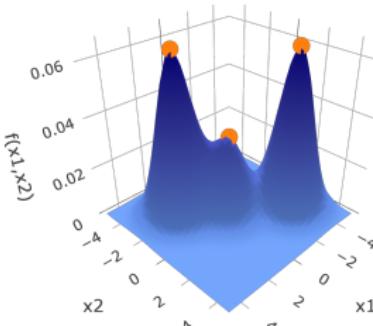


$$a_1 = 0.4, a_2 = 0.2, a_3 = 0.4 \\ c_1 = -3, c_2 = -2, \textcolor{orange}{c}_3 = 1$$

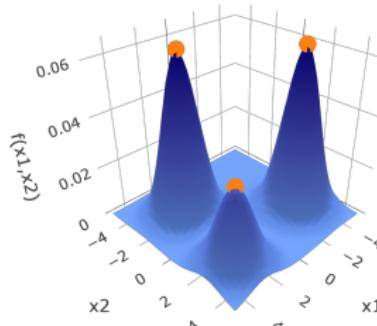
Weights altered



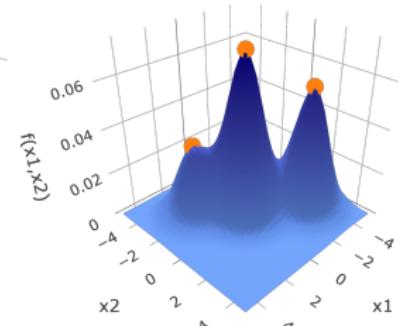
$$\textcolor{blue}{a}_1 = 0.6, a_2 = 0.2, a_3 = 0.2 \\ c_1 = -3, c_2 = -2, c_3 = 1$$



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



$$0.4, a_2 = 0.2, a_3 = 0.4 \\ c_1 = (2, -2), \textcolor{orange}{c}_2 = (3, 3), \\ c_3 = (-3, 2)$$

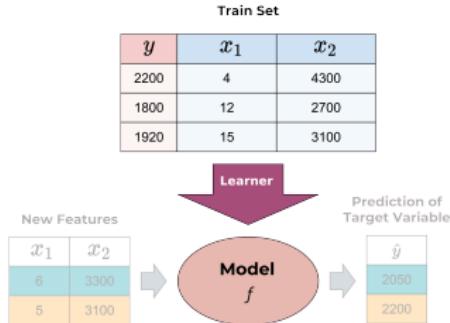


$$\textcolor{blue}{a}_1 = 0.2, a_2 = 0.45, a_3 = 0.35 \\ c_1 = (2, -2), c_2 = (0, 0), \\ c_3 = (-3, 2)$$

$$a_1 =$$

# Introduction to Machine Learning

## 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.

Features $x$		Target $y$
People in Office (Feature 1) $x_1$	Salary (Feature 2) $x_2$	Worked Minutes Week (Target Variable)
4	4300 €	2220
12	2700 €	1800
5	3100 €	1920

$p = 2$

$n = 3$

$x_1^{(2)}$

$x_2^{(1)}$

$y^{(3)}$

The diagram illustrates a supervised learning dataset. It consists of a table with three rows and two columns of features, and one column for the target variable. The first two columns are grouped by a brace labeled  $p = 2$ , indicating there are two features. The bottom of the table is grouped by a brace labeled  $n = 3$ , indicating there are three samples. To the left of the table, a circle contains  $x_1^{(2)}$ , pointing to the second row of the first feature column. To the right of the table, two circles are shown: one containing  $x_2^{(1)}$  pointing to the first row of the second feature column, and another containing  $y^{(3)}$  pointing to the third row of the target variable column.

# 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 peoples 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.

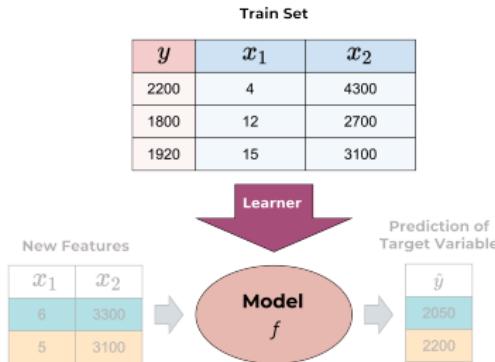
- **Supervised** learning means we make use of *labeled* data for which we observed the outcome.
- We use the labeled data to learn a model  $f$ .
- Ultimately, we use our model to compute predictions for **new** data whose target values are unknown.



# 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 (plus a vector of **hyperparameter** control settings  $\lambda$ ) to a model:

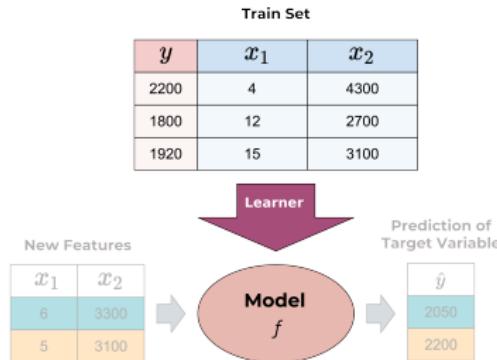
$$\mathcal{I} : \mathcal{D} \times \Lambda \rightarrow \mathcal{H}$$



# LEARNER DEFINITION

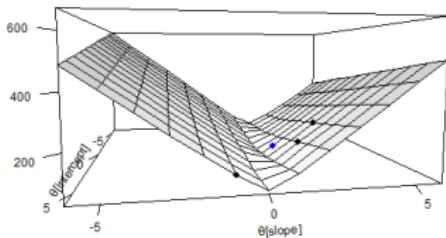
As pseudo-code template it would work like this:

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



# Introduction to Machine Learning

## 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.

The diagram illustrates the components of a machine learning training set. It consists of three tables arranged horizontally, connected by a double-headed arrow in the center.

- Features  $x$ :** This table has two columns: "People In Office (Feature 1)  $x_1$ " and "Salary (Feature 2)  $x_2$ ". The data rows are:

4	4300 €
12	2700 €
5	3100 €

- Target  $y$ :** This table has one column: "Worked Minutes Week (Target Variable)". The data rows are:

2220
1800
1920

- Prediction  $\hat{y}$ :** This table has one column: "Worked Minutes Week (Target Variable)". The data rows are:

2588
1644
1870

A bracket below the first two tables is labeled  $\mathcal{D}_{\text{train}}$ , indicating they form the training dataset.

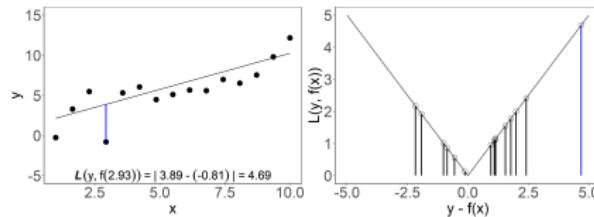
- To make this precise, we need to define now how we measure the difference between a prediction and a ground truth label pointwise.

# LOSS

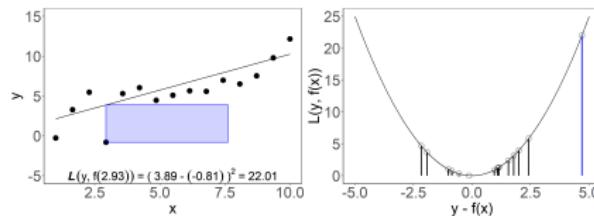
The **loss function**  $L(y, f(\mathbf{x}))$  quantifies the "quality" of the prediction  $f(\mathbf{x})$  of a single observation  $\mathbf{x}$ :

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

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**

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

- 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  $\mathcal{D}$ .

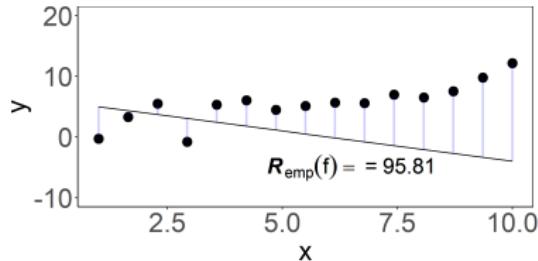
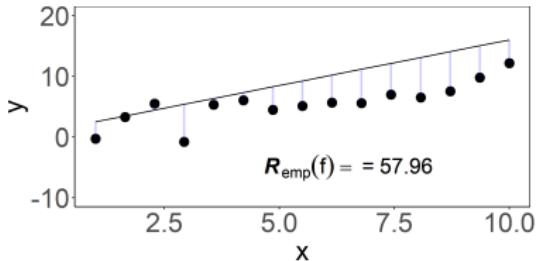
# EMPIRICAL RISK

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.

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



# EMPIRICAL RISK

- The risk can also be defined as an average loss

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

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**  $\theta$ , this becomes:

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

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

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

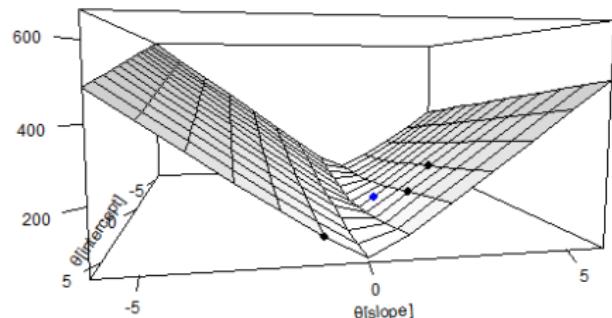
# EMPIRICAL RISK MINIMIZATION

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

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

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

Model	$\theta_{\text{intercept}}$	$\theta_{\text{slope}}$	$\mathcal{R}_{\text{emp}}(\theta)$
$f_1$	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

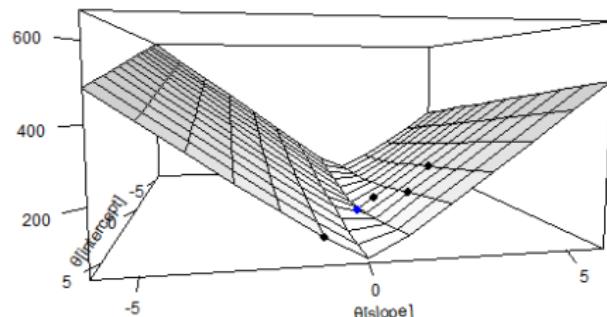
Minimizing this surface is called **empirical risk minimization** (ERM).

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

Usually we do this by numerical optimization.

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

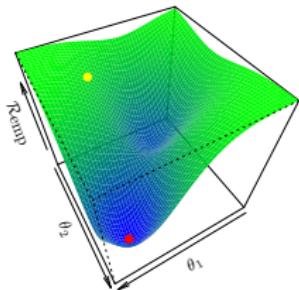
Model	$\theta_{\text{intercept}}$	$\theta_{\text{slope}}$	$\mathcal{R}_{\text{emp}}(\theta)$
$f_1$	2	3	194.62
$f_2$	3	2	127.12
$f_3$	6	-1	95.81
$f_4$	1	1.5	57.96
$f_5$	1.25	0.90	23.40



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

# Introduction to Machine Learning

## ML-Basics: Optimization

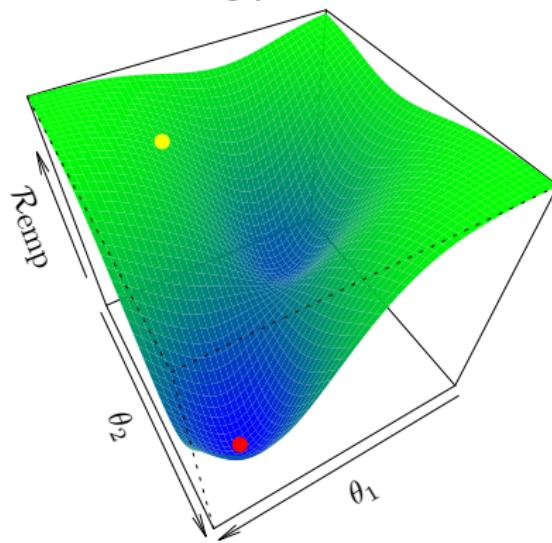


### Learning goals

- 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  $\theta \in \Theta$  with lowest empirical risk  $\mathcal{R}_{\text{emp}}(\theta)$ .
- 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:

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

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

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

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

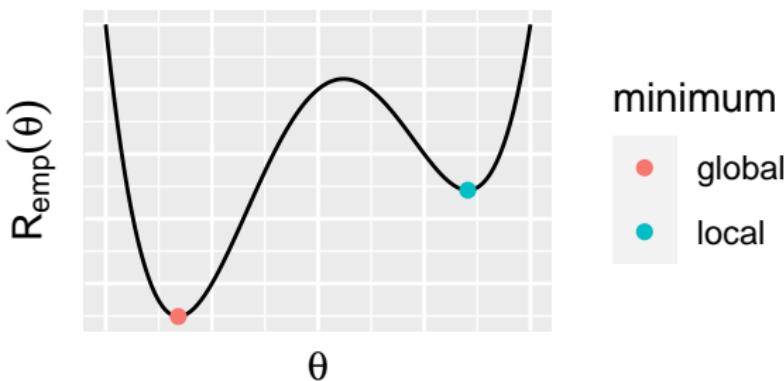
Which kind of numerical technique is reasonable for this problem strongly depends on model and parameter structure (continuous params? uni-modal  $\mathcal{R}_{\text{emp}}(\theta)$ ?). Here, we will only discuss very simple scenarios.

# LOCAL MINIMA

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

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

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

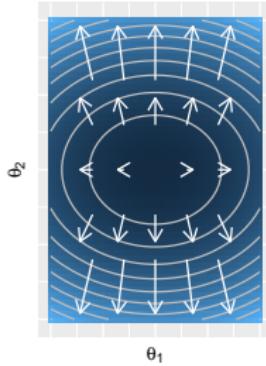


# LOCAL MINIMA AND STATIONARY POINTS

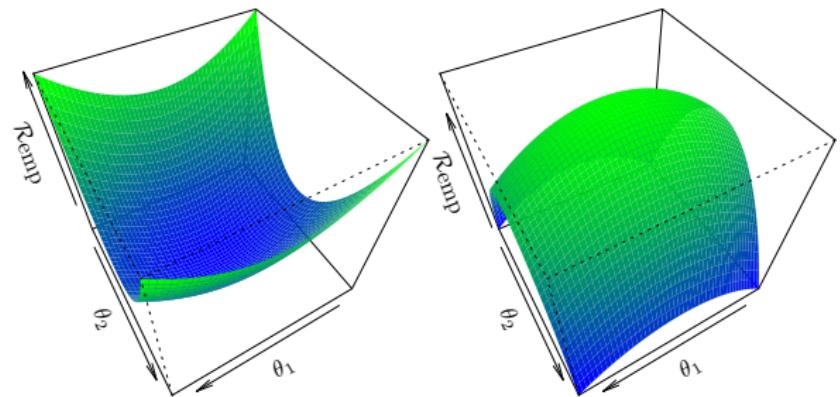
If  $\mathcal{R}_{\text{emp}}$  is continuously differentiable in  $\theta$  then a **sufficient condition** for a local minimum is that  $\hat{\theta}$  is **stationary** with 0 gradient, so no local improvement is possible:

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

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



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



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 continuously differentiable functions it can be shown that the **least squares estimator**

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

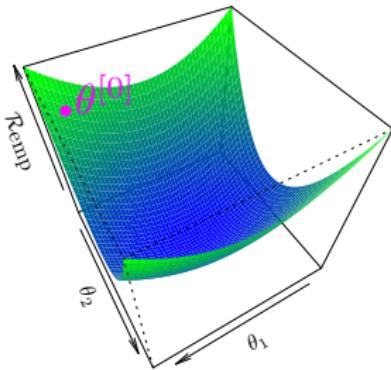
is a local minimum of  $\mathcal{R}_{\text{emp}}$ . If  $\mathbf{X}$  is full-rank,  $\mathcal{R}_{\text{emp}}$  is strictly convex and there is only one local minimum - which is also global.

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

# GRADIENT DESCENT

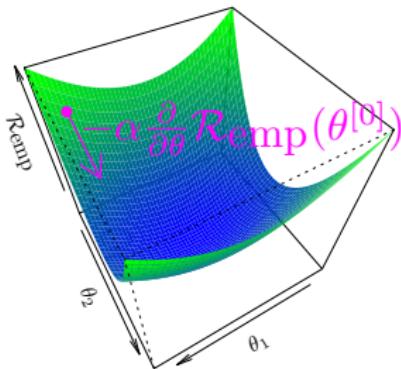
The simple idea of GD is to iteratively go from the current candidate  $\theta^{[t]}$  in the direction of the negative gradient, i.e., the direction of the steepest descent, with learning rate  $\alpha$  to the next  $\theta^{[t+1]}$ :

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

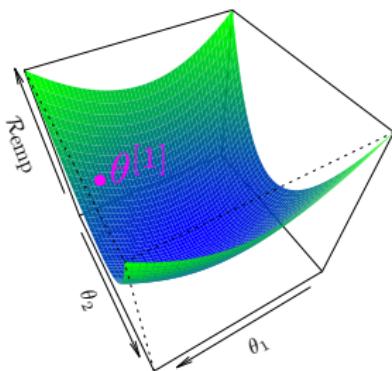


We choose a random start  $\theta^{[0]}$  with risk  $\mathcal{R}_{\text{emp}}(\theta^{[0]}) = 76.25$ .

# GRADIENT DESCENT - EXAMPLE

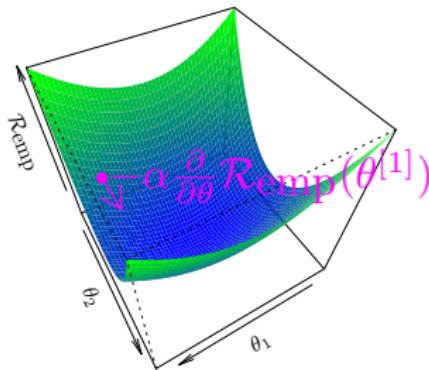


Now we follow in the direction of the negative gradient at  $\theta^{[0]}$ .

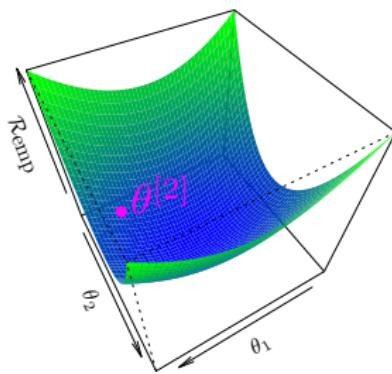


We arrive at  $\theta^{[1]}$  with risk  
 $\mathcal{R}_{\text{emp}}(\theta^{[1]}) \approx 42.73$ .  
We improved:  
 $\mathcal{R}_{\text{emp}}(\theta^{[1]}) < \mathcal{R}_{\text{emp}}(\theta^{[0]}).$

# GRADIENT DESCENT - EXAMPLE

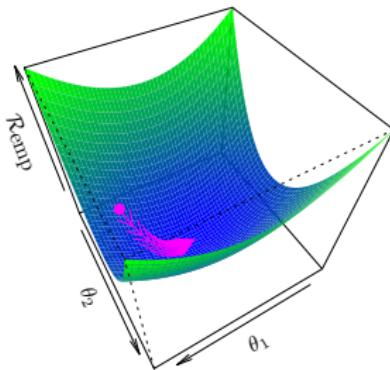


Again we follow in the direction of the negative gradient, but now at  $\theta^{[1]}$ .

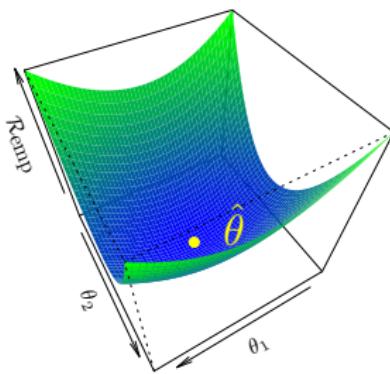


Now  $\theta^{[2]}$  has risk  $\mathcal{R}_{\text{emp}}(\theta^{[2]}) \approx 25.08$ .

# GRADIENT DESCENT - EXAMPLE



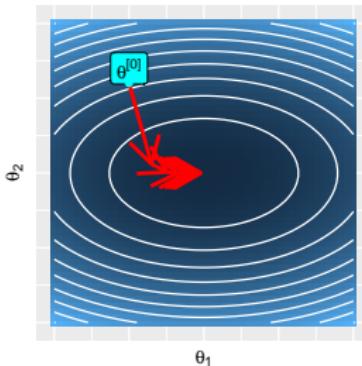
We iterate this until some form of convergence or termination.



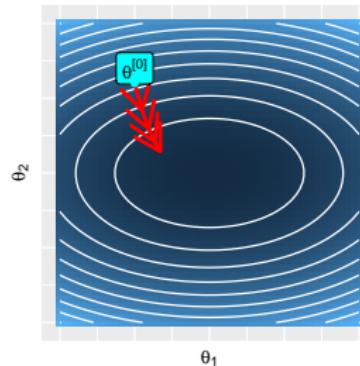
We arrive close to a stationary  $\hat{\theta}$  which is hopefully at least a local minimum.

# GRADIENT DESCENT - LEARNING RATE

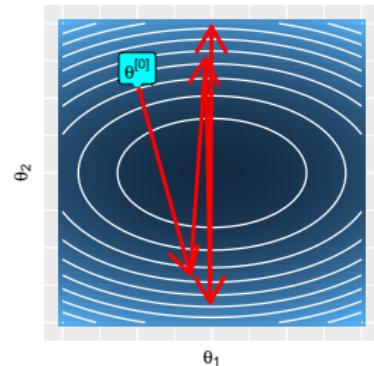
- The negative gradient is a direction that looks locally promising to reduce  $\mathcal{R}_{\text{emp}}$ .
- Hence it weights components higher in which  $\mathcal{R}_{\text{emp}}$  decreases more.
- However, the length of  $-\frac{\partial}{\partial \theta} \mathcal{R}_{\text{emp}}$  measures only the local decrease rate, i.e., there are no guarantees that we will not go "too far".
- We use a learning rate  $\alpha$  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  $\alpha_1$



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



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

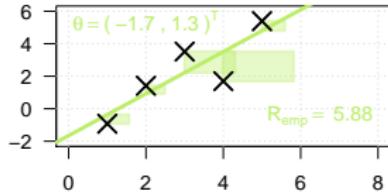
# FURTHER TOPICS

- 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.

# Introduction to Machine Learning

## 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:

$$\text{Learning} = \text{Hypothesis Space} + \text{Risk} + \text{Optimization}$$

- **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

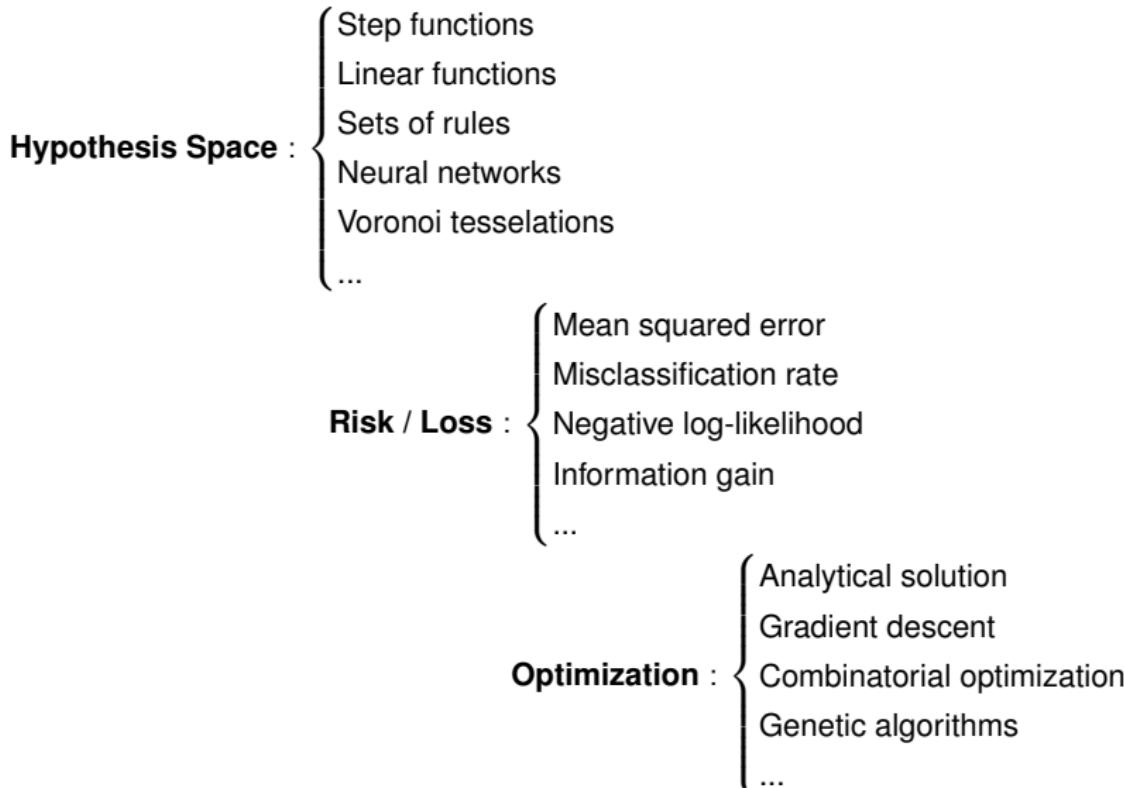
This concept can be extended by the concept of **regularization**, where the model complexity is accounted for in the risk:

$$\begin{aligned}\text{Learning} &= \text{Hypothesis Space} + \text{Risk} + \text{Optim} \\ \text{Learning} &= \text{Hypothesis Space} + \text{Loss (+ Regularization)} + \text{Optim}\end{aligned}$$

- 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:



# SUPERVISED LEARNING, FORMALIZED

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

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

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

So the inducer mapping (including hyperparameters  $\Lambda$ ) is:

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

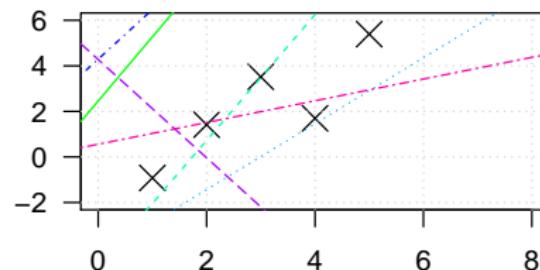
We can also adapt this concept to finding  $\hat{\theta}$  for parametric models:

$$\mathcal{I} : \mathcal{D} \times \Lambda \rightarrow \Theta$$

# EXAMPLE: LINEAR REGRESSION ON 1D

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

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

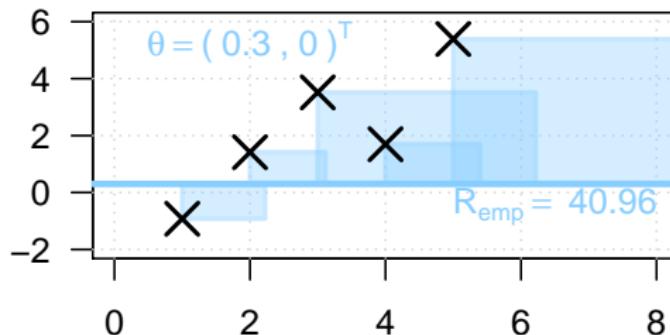


**Design choice:** We could add more flexibility by allowing polynomial effects or by using a spline basis.

# EXAMPLE: LINEAR REGRESSION ON 1D

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

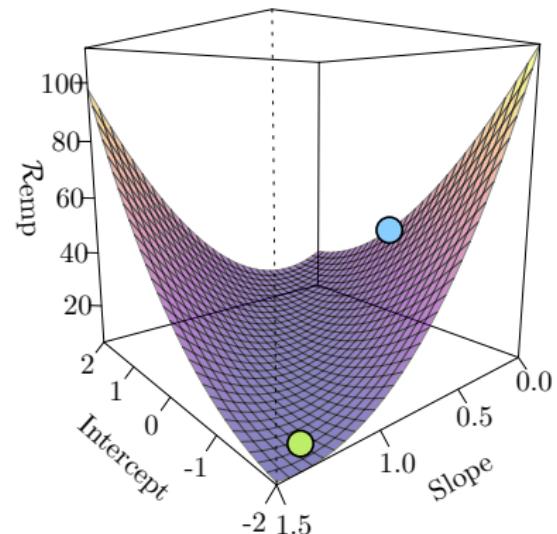
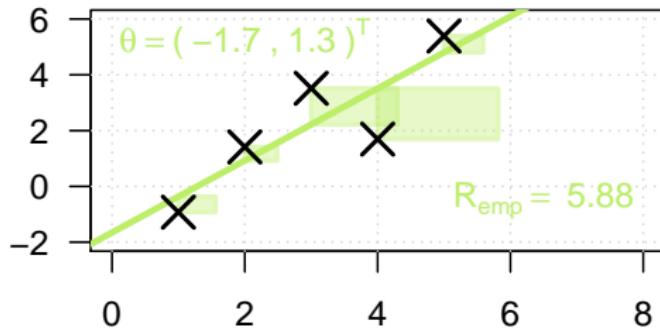
$$\mathcal{R}_{\text{emp}}(\theta) = \sum_{i=1}^n (y^{(i)} - \theta_0 - \theta \mathbf{x}^{(i)})^2$$



**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.

# INTRODUCTION TO MACHINE LEARNING

ML Basics

**Supervised Regression**

Supervised Classification

k-NN

Performance Evaluation

Classification and Regression Trees (CART)

Random Forests

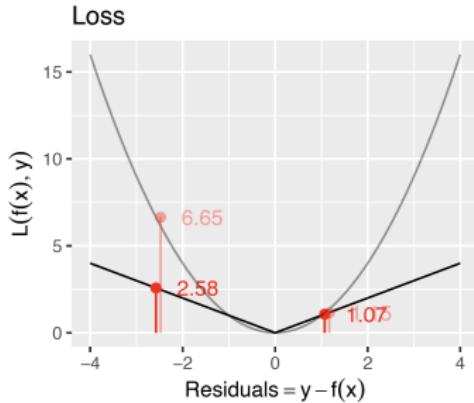
Tuning

Nested Resampling

mlr3

# Introduction to Machine Learning

## Loss Functions for Regression



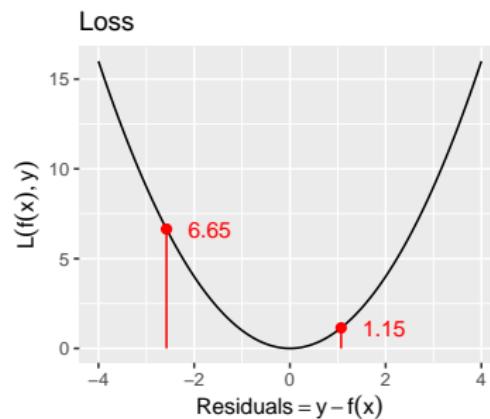
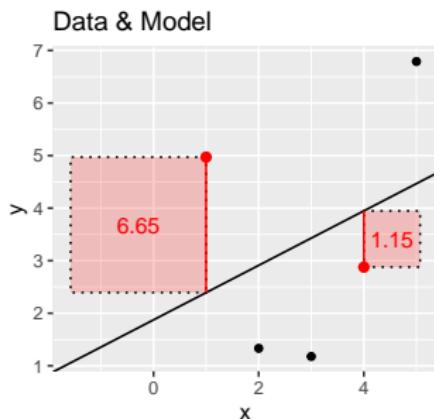
### Learning goals

- Know definitions of L1 and L2 loss
- Understand difference between L1 and L2 loss
- Understand why optimization for L1 loss is harder than for L2 loss

# REGRESSION LOSSES - L2 / SQUARED ERROR

- $L(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2$  or  $L(y, f(\mathbf{x})) = 0.5(y - f(\mathbf{x}))^2$
- Convex
- Differentiable, gradient no problem in loss minimization
- For latter:  $\frac{\partial 0.5(y-f(\mathbf{x}))^2}{\partial f(\mathbf{x})} = y - f(\mathbf{x}) = \epsilon$ , derivative is residual
- Tries to reduce large residuals (if residual is twice as large, loss is 4 times as large), hence outliers in  $y$  can become problematic
- Connection to Gaussian distribution (see later)

# REGRESSION LOSSES - L2 / SQUARED ERROR



# REGRESSION LOSSES - L2 / SQUARED ERROR

What's the optimal constant prediction  $c$  (i.e. the same  $\hat{y}$  for all  $\mathbf{x}$ )?

$$L(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2 = (y - c)^2$$

We search for the  $c$  that minimizes the empirical risk.

$$\hat{c} = \arg \min_{c \in \mathbb{R}} \mathcal{R}_{\text{emp}}(c) = \arg \min_{c \in \mathbb{R}} \frac{1}{n} \sum_{i=1}^n (y^{(i)} - c)^2$$

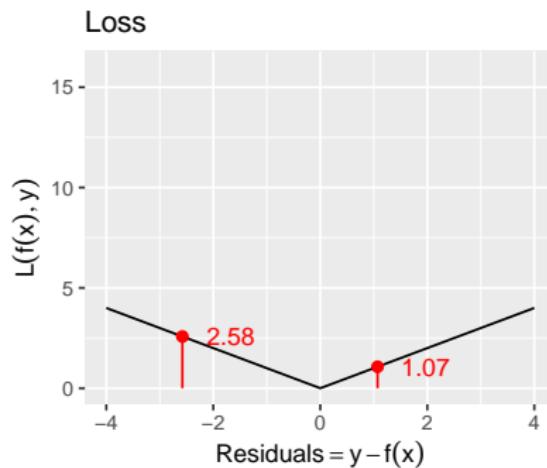
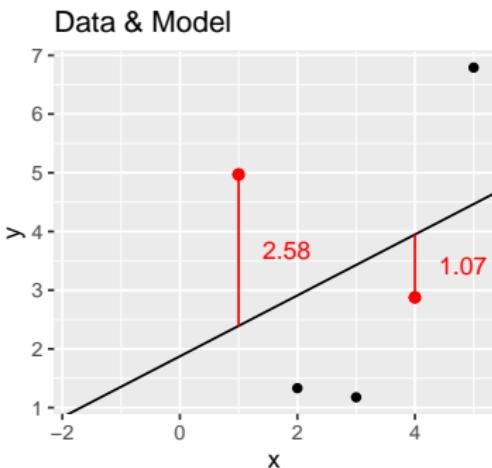
We set the derivative of the empirical risk to zero and solve for  $c$ :

$$-\frac{1}{n} \sum_{i=1}^n 2(y^{(i)} - c) = 0$$

$$\hat{c} = \frac{1}{n} \sum_{i=1}^n y^{(i)}$$

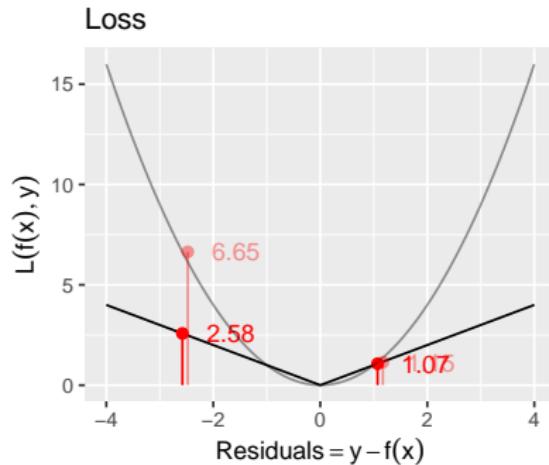
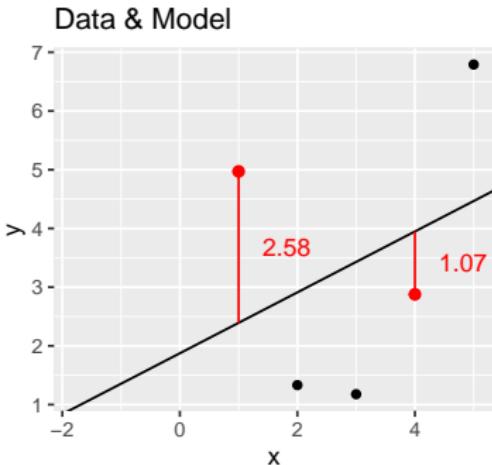
# REGRESSION LOSSES - L1 / ABSOLUTE ERROR

- $L(y, f(\mathbf{x})) = |y - f(\mathbf{x})|$
- Convex
- No derivatives for  $y = f(\mathbf{x})$ , optimization becomes harder
- $\hat{f}(\mathbf{x}) = \text{median of } y|\mathbf{x}$



# REGRESSION LOSSES - L1 / ABSOLUTE ERROR

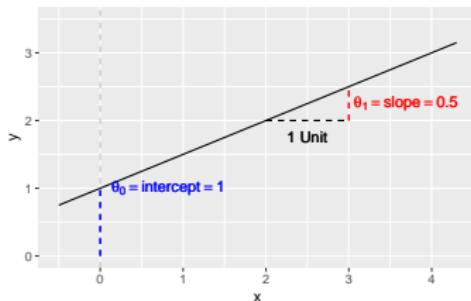
- $L(y, f(\mathbf{x})) = |y - f(\mathbf{x})|$
- Convex
- No derivatives for  $\epsilon = 0$ ,  $y = f(\mathbf{x})$ , optimization becomes harder
- $\hat{f}(\mathbf{x}) = \text{median of } y|\mathbf{x}$
- More robust, outliers in  $y$  are less influential than for L2



# Introduction to Machine Learning

## Linear Regression Models

### Learning goals



- Know the hypothesis space of the linear model
- Understand the risk function that follows with L2 loss
- Understand how optimization works for the linear model
- Understand how outliers affect the estimated model differently when using L1 or L2 loss

# LINEAR REGRESSION: HYPOTHESIS SPACE

We want to predict a numerical target variable by a *linear transformation* of the features  $\mathbf{x} \in \mathbb{R}^p$ .

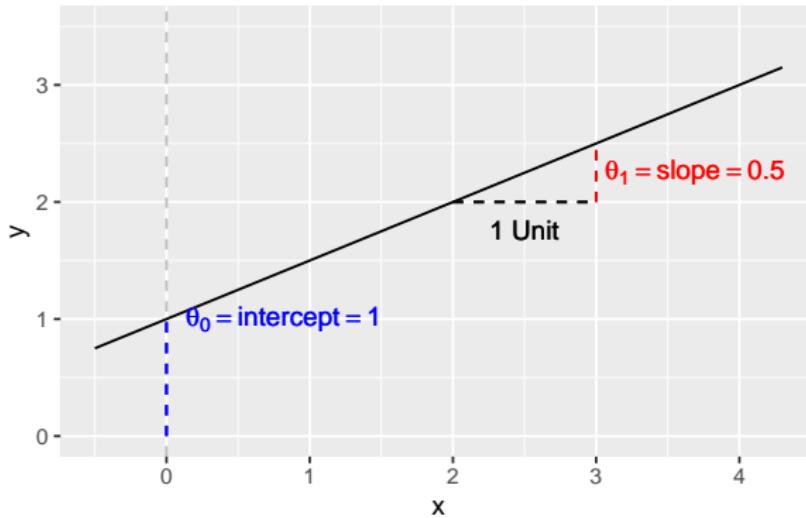
So with  $\boldsymbol{\theta} \in \mathbb{R}^p$  this mapping can be written as:

$$\begin{aligned}y &= f(\mathbf{x}) = \theta_0 + \boldsymbol{\theta}^T \mathbf{x} \\&= \theta_0 + \theta_1 x_1 + \cdots + \theta_p x_p\end{aligned}$$

This defines the hypothesis space  $\mathcal{H}$  as the set of all linear functions in  $\boldsymbol{\theta}$ :

$$\mathcal{H} = \{\theta_0 + \boldsymbol{\theta}^T \mathbf{x} \mid (\theta_0, \boldsymbol{\theta}) \in \mathbb{R}^{p+1}\}$$

# LINEAR REGRESSION: HYPOTHESIS SPACE



$$y = \theta_0 + \theta \cdot x$$

# LINEAR REGRESSION: HYPOTHESIS SPACE

Given observed labeled data  $\mathcal{D}$ , how to find  $(\theta_0, \theta)$ ?

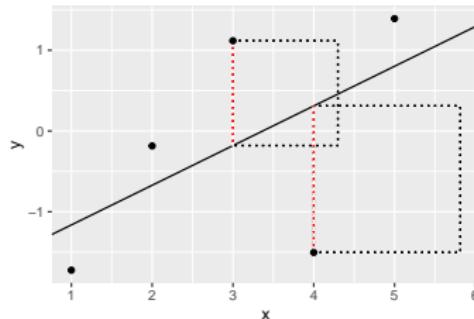
This is **learning** or parameter estimation, the learner does exactly this by **empirical risk minimization**.

NB: We assume from now on that  $\theta_0$  is included in  $\theta$ .

# LINEAR REGRESSION: RISK

We could measure training error as the sum of squared prediction errors (SSE). This is the risk that corresponds to **L2 loss**:

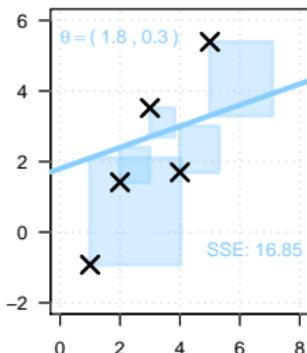
$$\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) = \text{SSE}(\boldsymbol{\theta}) = \sum_{i=1}^n L\left(y^{(i)}, f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)\right) = \sum_{i=1}^n \left(y^{(i)} - \boldsymbol{\theta}^T \mathbf{x}^{(i)}\right)^2$$



Minimizing the squared error is computationally much simpler than minimizing the absolute differences (**L1 loss**).

# LINEAR MODEL: OPTIMIZATION

We want to find the parameters  $\theta$  of the linear model, i.e., an element of the hypothesis space  $\mathcal{H}$  that fits the data optimally.  
So we evaluate different candidates for  $\theta$ .  
A first (random) try yields a rather large SSE: (**Evaluation**).

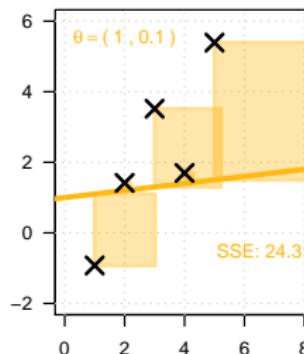
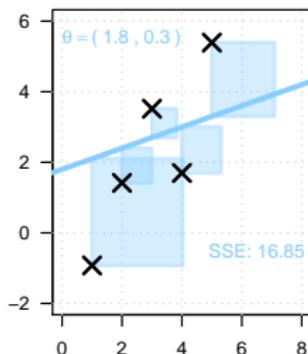


# LINEAR MODEL: OPTIMIZATION

We want to find the parameters  $\theta$  of the linear model, i.e., an element of the hypothesis space  $\mathcal{H}$  that fits the data optimally.

So we evaluate different candidates for  $\theta$ .

Another line yields an even bigger SSE (**Evaluation**). Therefore, this one is even worse in terms of empirical risk.

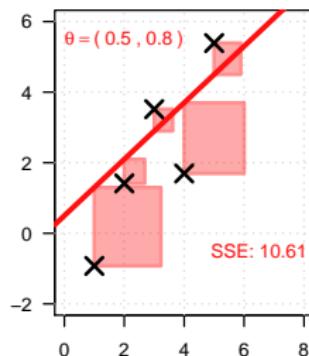
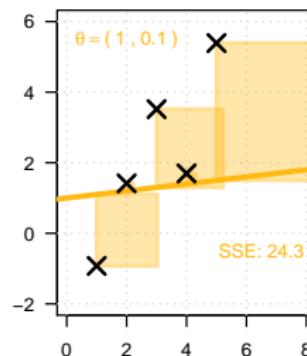
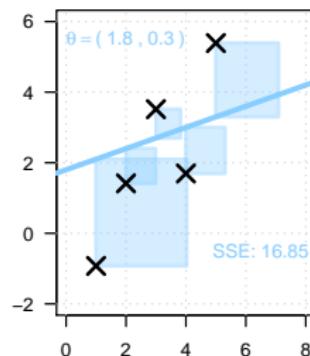


# LINEAR MODEL: OPTIMIZATION

We want to find the parameters  $\theta$  of the linear model, i.e., an element of the hypothesis space  $\mathcal{H}$  that fits the data optimally.

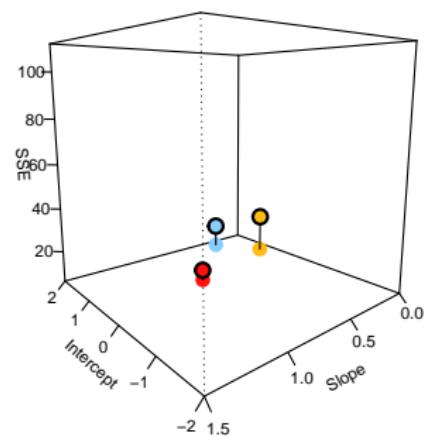
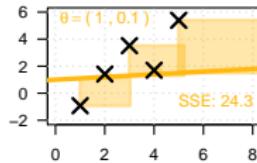
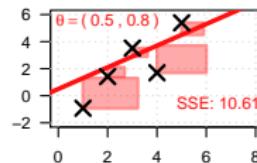
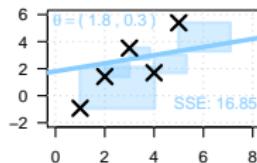
So we evaluate different candidates for  $\theta$ .

Another line yields an even bigger SSE (**Evaluation**). Therefore, this one is even worse in terms of empirical risk. Let's try again:



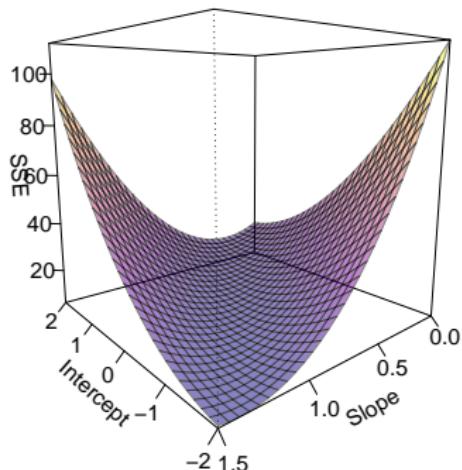
# LINEAR MODEL: OPTIMIZATION

Since every  $\theta$  results in a specific value of  $\mathcal{R}_{\text{emp}}(\theta)$ , and we try to find  $\arg \min_{\theta} \mathcal{R}_{\text{emp}}(\theta)$ , let's look at what we have so far:



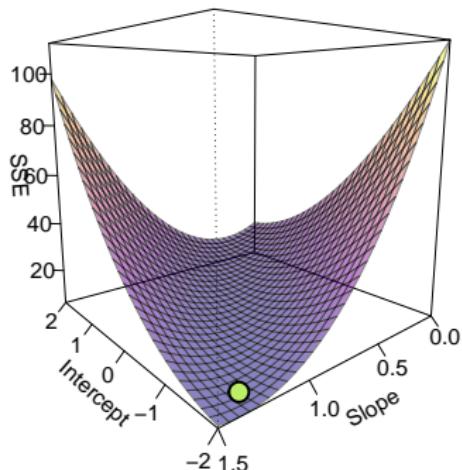
# LINEAR MODEL: OPTIMIZATION

Instead of guessing, we use **optimization** to find the best  $\theta$ :



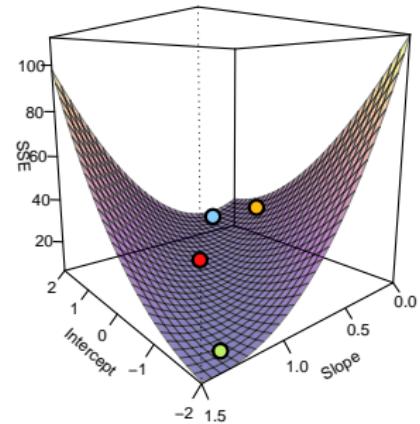
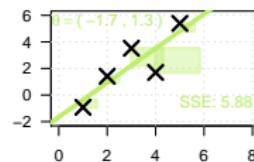
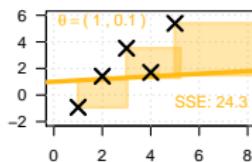
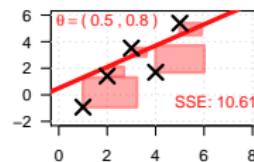
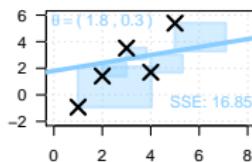
# LINEAR MODEL: OPTIMIZATION

Instead of guessing, we use **optimization** to find the best  $\theta$ :



# LINEAR MODEL: OPTIMIZATION

Instead of guessing, we use **optimization** to find the best  $\theta$ :



# LINEAR MODEL: OPTIMIZATION

For L2 regression, we can find this optimal value analytically:

$$\begin{aligned}\hat{\theta} &= \arg \min_{\theta} \mathcal{R}_{\text{emp}}(\theta) = \sum_{i=1}^n \left( y^{(i)} - \theta^T \mathbf{x}^{(i)} \right)^2 \\ &= \arg \min_{\theta} \|\mathbf{y} - \mathbf{X}\theta\|_2^2\end{aligned}$$

where  $\mathbf{X} = \begin{pmatrix} 1 & x_1^{(1)} & \dots & x_p^{(1)} \\ 1 & x_1^{(2)} & \dots & x_p^{(2)} \\ \vdots & \vdots & & \vdots \\ 1 & x_1^{(n)} & \dots & x_p^{(n)} \end{pmatrix}$  is the  $n \times (p + 1)$ -**design matrix**.

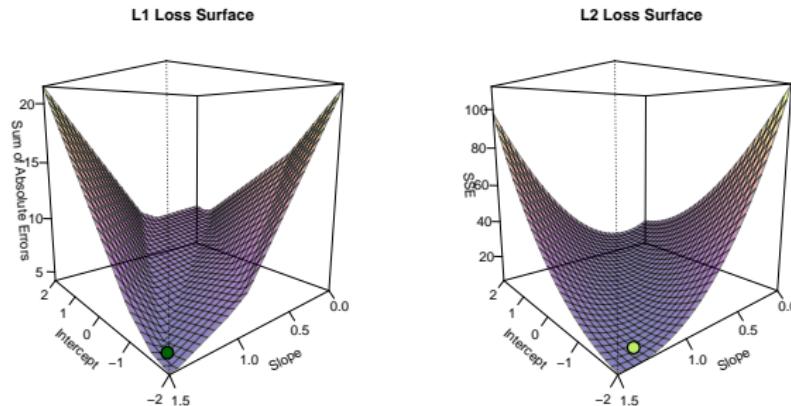
This yields the so-called normal equations for the LM:

$$\frac{\partial}{\partial \theta} \mathcal{R}_{\text{emp}}(\theta) = 0 \quad \Rightarrow \quad \hat{\theta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

# EXAMPLE: REGRESSION WITH L1 VS L2 LOSS

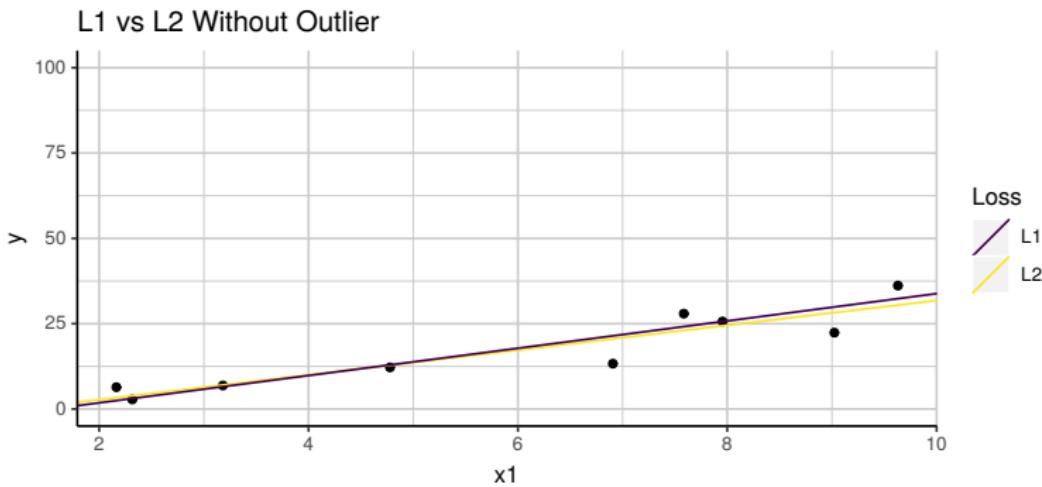
We could also minimize the L1 loss. This changes the risk and optimization steps:

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



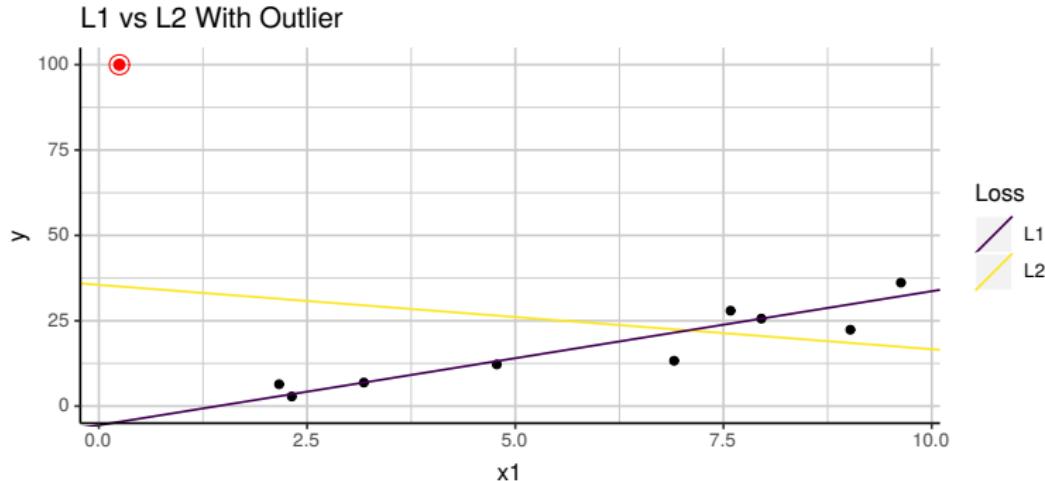
L1 loss is harder to optimize, but the model is less sensitive to outliers.

# EXAMPLE: REGRESSION WITH L1 VS L2 LOSS



# EXAMPLE: REGRESSION WITH L1 VS L2 LOSS

Adding an outlier (highlighted red) pulls the line fitted with L2 into the direction of the outlier:



# LINEAR REGRESSION

**Hypothesis Space:** Linear functions  $\mathbf{x}^T \boldsymbol{\theta}$  of features  $\in \mathcal{X}$ .

**Risk:** Any regression loss function.

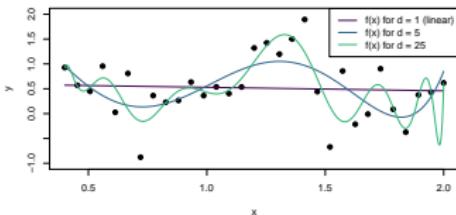
**Optimization:** Direct analytical solution for L2 loss, numerical optimization for L1 and others.

# Introduction to Machine Learning

## Polynomial Regression Models

### Learning goals

- Understand how to add flexibility to the linear model by using polynomials
- Understand that this only affects the hypothesis space, not risk or optimization
- Understand that more flexibility is not equivalent to a better model



# REGRESSION: POLYNOMIALS

We can make linear regression models much more flexible by using *polynomials*  $\mathbf{x}_j^d$  – or any other *derived features* like  $\sin(\mathbf{x}_j)$  or  $(\mathbf{x}_j \cdot \mathbf{x}_k)$  – as additional features.

The optimization and risk of the learner remain the same.

Only the hypothesis space of the learner changes:  
instead of linear functions

$$f(\mathbf{x}^{(i)} | \boldsymbol{\theta}) = \theta_0 + \theta_1 \mathbf{x}_1^{(i)} + \theta_2 \mathbf{x}_2^{(i)} + \dots$$

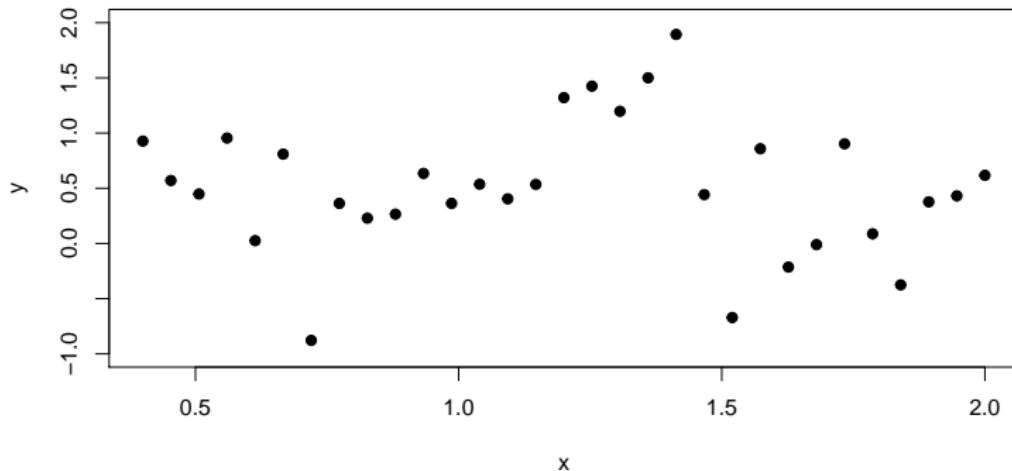
of only the original features,

it now includes linear functions of the derived features as well, e.g.

$$f(\mathbf{x}^{(i)} | \boldsymbol{\theta}) = \theta_0 + \sum_{k=1}^d \theta_{1k} \left( \mathbf{x}_1^{(i)} \right)^k + \sum_{k=1}^d \theta_{2k} \left( \mathbf{x}_2^{(i)} \right)^k + \dots$$

# REGRESSION: POLYNOMIALS

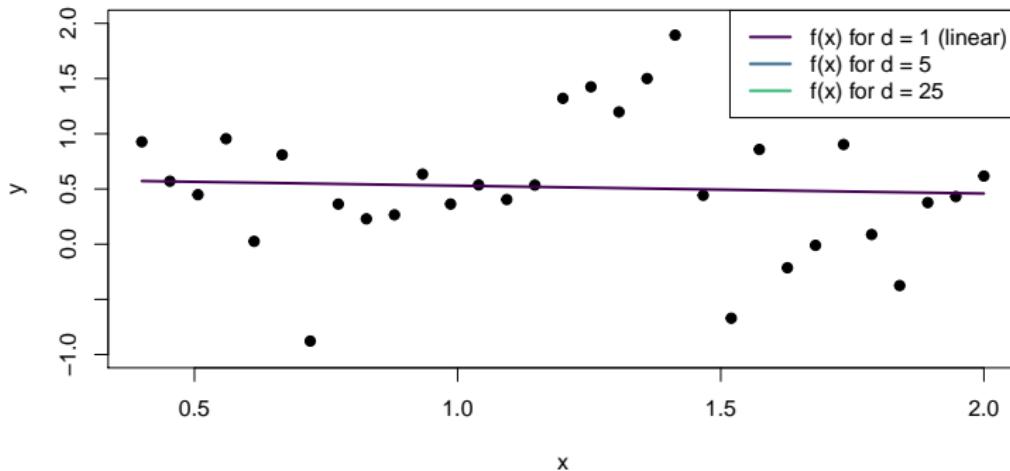
## Polynomial regression example



# REGRESSION: POLYNOMIALS

## Polynomial regression example

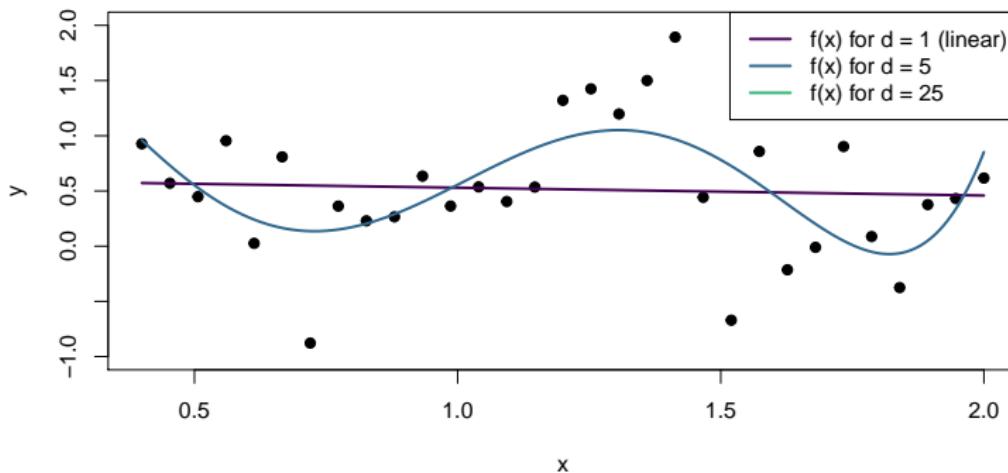
Models of different *complexity*, i.e., of different polynomial order  $d$ , are fitted to the data:



# REGRESSION: POLYNOMIALS

## Polynomial regression example

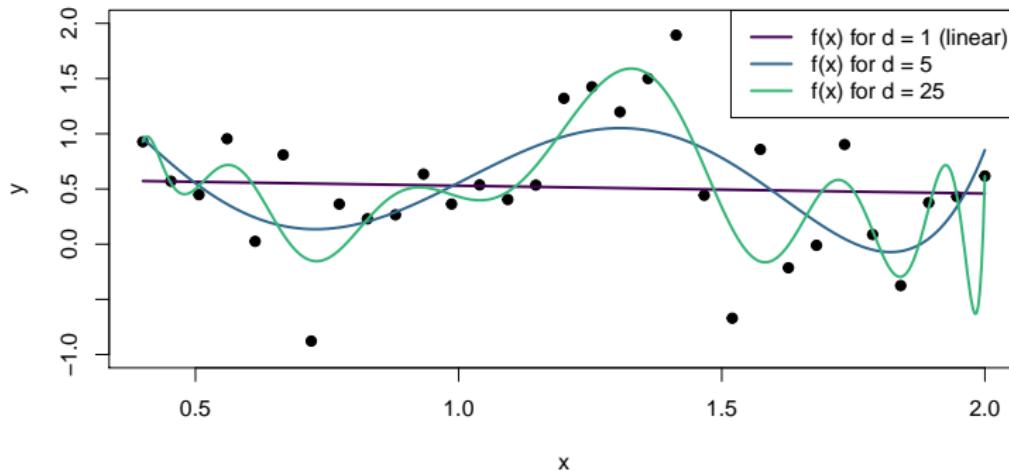
Models of different *complexity*, i.e., of different polynomial order  $d$ , are fitted to the data:



# REGRESSION: POLYNOMIALS

## Polynomial regression example

Models of different *complexity*, i.e., of different polynomial order  $d$ , are fitted to the data:



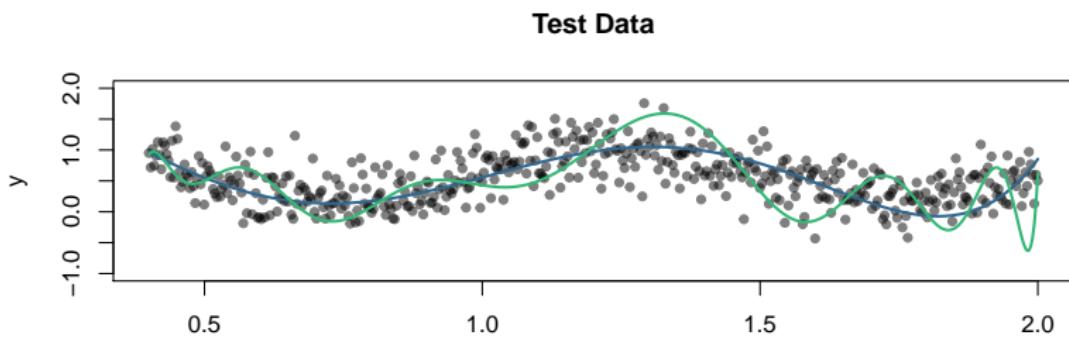
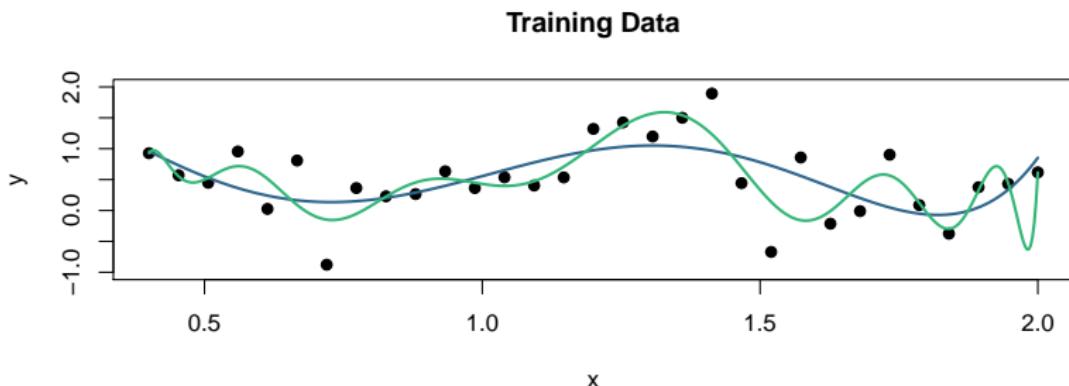
# REGRESSION: POLYNOMIALS

The higher  $d$  is, the more **capacity** the learner has to learn complicated functions of  $\mathbf{x}$ , but this also increases the danger of **overfitting**:

The model space  $\mathcal{H}$  contains so many complex functions that we are able to find one that approximates the training data arbitrarily well.

However, predictions on new data are not as successful because our model has learnt spurious “wiggles” from the random noise in the training data (much, much more on this later).

# REGRESSION: POLYNOMIALS



# INTRODUCTION TO MACHINE LEARNING

ML Basics

Supervised Regression

## Supervised Classification

k-NN

Performance Evaluation

Classification and Regression Trees (CART)

Random Forests

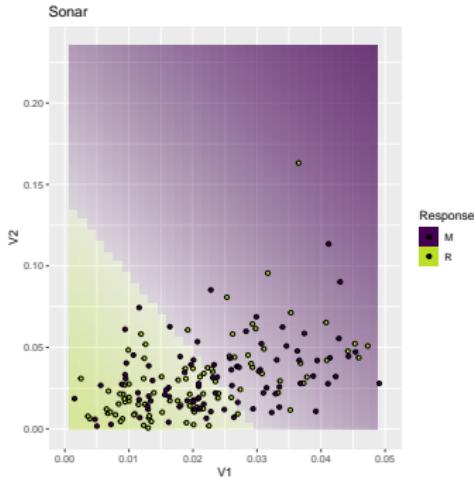
Tuning

Nested Resampling

mlr3

# Introduction to Machine Learning

## Classification: Tasks

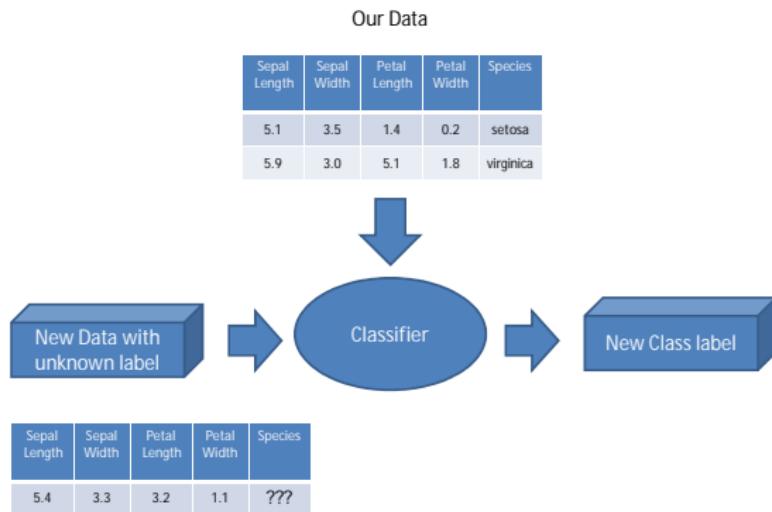


### Learning goals

- Understand the main difference between regression and classification
- Know that classification can be binary or multiclass
- Know some examples of classification tasks

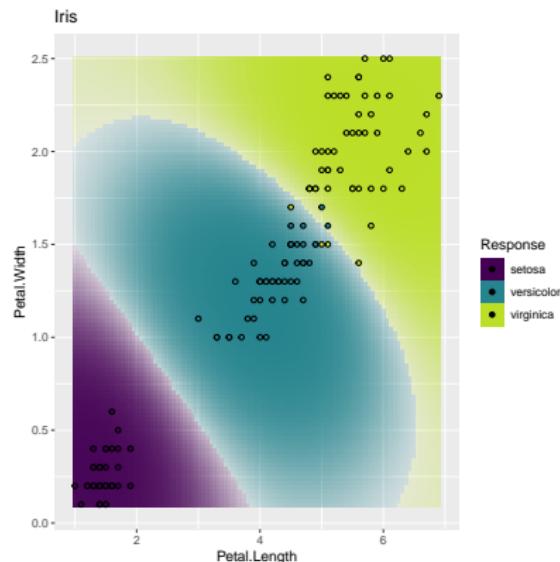
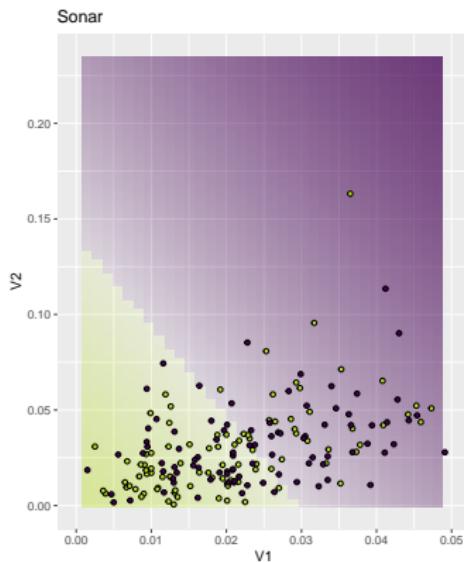
# CLASSIFICATION

Learn functions that assign class labels to observation / feature vectors.  
Each observation belongs to exactly one class. The main difference to regression is the scale of the output / label.



# BINARY AND MULTICLASS TASKS

The task can contain 2 classes (binary) or multiple (multiclass).

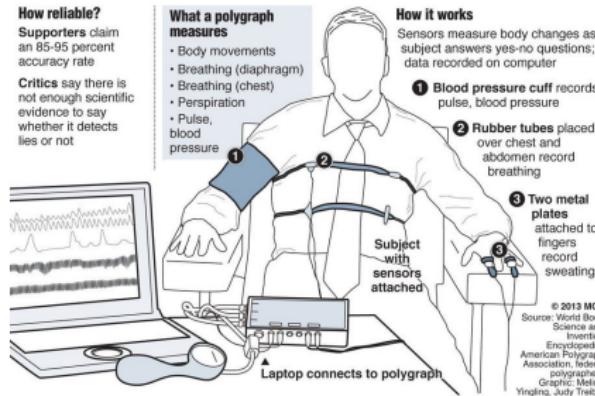


# BINARY CLASSIFICATION TASK - EXAMPLES

- Credit risk prediction, based on personal data and transactions
- Spam detection, based on textual features
- Churn prediction, based on customer behavior
- Predisposition for specific illness, based on genetic data

## Do polygraphs detect lies?

*Polygraph or "lie detector" exams continue to be used by law enforcement and government agencies for various screenings even though most criminal courts ban polygraph evidence.*



<https://www.bendbulletin.com/localstate/deschutescounty/3430324-151/fact-or-fiction-polygraphs-just-an-investigative-tool>

# MULTICLASS TASK - MEDICAL DIAGNOSIS

INFO

SYMPTOMS

QUESTIONS

CONDITIONS

DETAILS

TREATMENT

## Conditions that match your symptoms

UNDERSTANDING YOUR RESULTS 

Acute Sinusitis



Moderate match



Influenza (flu) adults



Moderate match



Common cold



Fair match



Asthma (Teen and Adult)



Fair match



Whooping cough



Fair match



LOAD MORE CONDITIONS

Gender **Female**

Age **25**

[Edit](#)

My Symptoms

[Edit](#)

cough, headache, nausea

Could you be pregnant?

[Edit](#)

No



[Start Over](#)

<https://symptoms.webmd.com>

# MULTICLASS TASK - IRIS

The iris dataset was introduced by the statistician Ronald Fisher and is one of the most frequently used data sets. Originally, it was designed for linear discriminant analysis.



Setosa



Versicolor



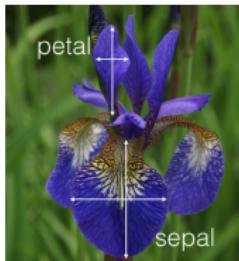
Virginica

Source:

[https://en.wikipedia.org/wiki/Iris\\_flower\\_data\\_set](https://en.wikipedia.org/wiki/Iris_flower_data_set)

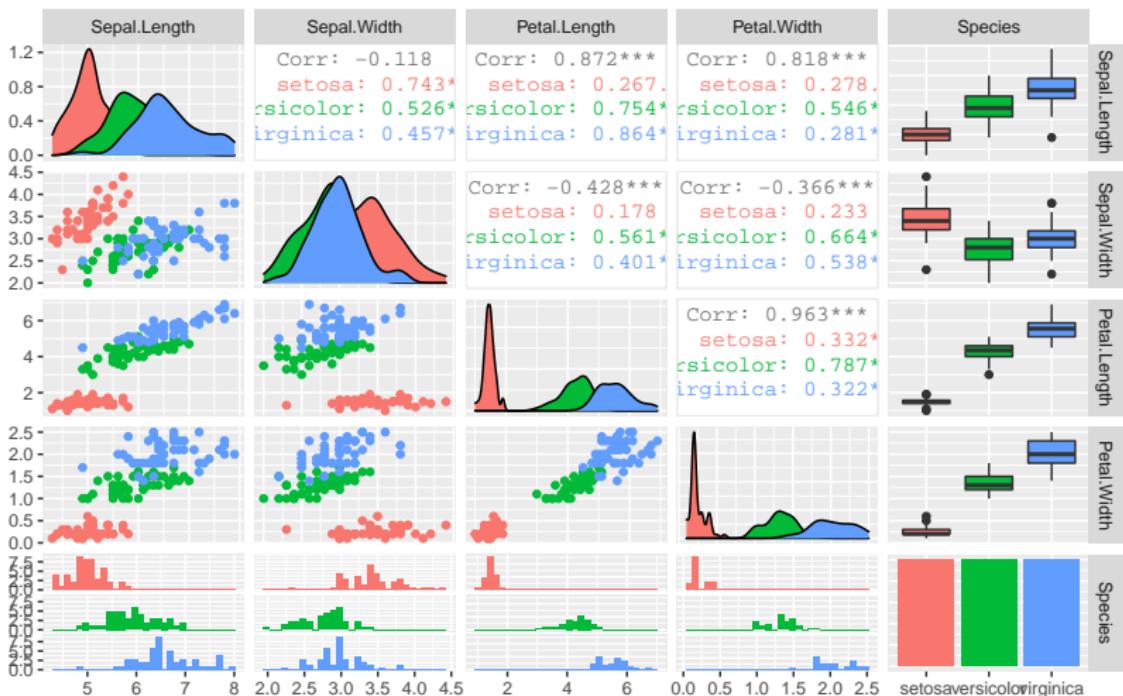
# MULTICLASS TASK - IRIS

- 150 iris flowers
- Predict subspecies
- Based on sepal and petal length / width in [cm]



##	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
## 1:	5.1	3.5	1.4	0.2	setosa
## 2:	4.9	3.0	1.4	0.2	setosa
## 3:	4.7	3.2	1.3	0.2	setosa
## 4:	4.6	3.1	1.5	0.2	setosa
## 5:	5.0	3.6	1.4	0.2	setosa
## ---					
## 146:	6.7	3.0	5.2	2.3	virginica
## 147:	6.3	2.5	5.0	1.9	virginica
## 148:	6.5	3.0	5.2	2.0	virginica
## 149:	6.2	3.4	5.4	2.3	virginica
## 150:	5.9	3.0	5.1	1.8	virginica

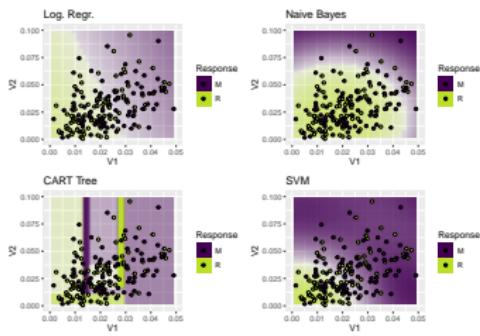
# MULTICLASS TASK - IRIS



# Introduction to Machine Learning

## Classification: Basic Definitions

### Learning goals



- Understand why classification models have a score / probability as output and not a class
- Understand the difference between scoring and probabilistic classifiers
- Know the concept of decision regions and boundaries
- Know the difference between generative and discriminant approach

# CLASSIFICATION TASKS

In classification, we aim at predicting a discrete output

$$y \in \mathcal{Y} = \{C_1, \dots, C_g\}$$

with  $2 \leq g < \infty$ , given data  $\mathcal{D}$ .

In this course, we assume the classes to be encoded as

- $\mathcal{Y} = \{0, 1\}$  or  $\mathcal{Y} = \{-1, +1\}$  (in the binary case  $g = 2$ )
- $\mathcal{Y} = \{1, \dots, g\}$  (in the multiclass case  $g \geq 3$ )

# CLASSIFICATION MODELS

We defined models  $f : \mathcal{X} \rightarrow \mathbb{R}^g$  as functions that output (continuous) **scores / probabilities** and **not** (discrete) classes. Why?

- From an optimization perspective, it is **much** (!) easier to optimize costs for continuous-valued functions
- Scores / probabilities (for classes) contain more information than the class labels alone
- As we will see later, scores can easily be transformed into class labels; but class labels cannot be transformed into scores

We distinguish **scoring** and **probabilistic** classifiers.

# SCORING CLASSIFIERS

- Construct  $g$  **discriminant / scoring functions**  $f_1, \dots, f_g : \mathcal{X} \rightarrow \mathbb{R}$
- Scores  $f_1(\mathbf{x}), \dots, f_g(\mathbf{x})$  are transformed into classes by choosing the class with the maximum score

$$h(\mathbf{x}) = \arg \max_{k \in \{1, \dots, g\}} f_k(\mathbf{x}).$$

- For  $g = 2$ , a single discriminant function  $f(\mathbf{x}) = f_1(\mathbf{x}) - f_{-1}(\mathbf{x})$  is sufficient (note that it would be natural here to label the classes with  $\{-1, +1\}$ )
- Class labels are constructed by  $h(\mathbf{x}) = \text{sgn}(f(\mathbf{x}))$
- $|f(\mathbf{x})|$  is called “confidence”

# PROBABILISTIC CLASSIFIERS

- Construct  $g$  probability functions

$$\pi_1, \dots, \pi_g : \mathcal{X} \rightarrow [0, 1], \sum_i \pi_i = 1$$

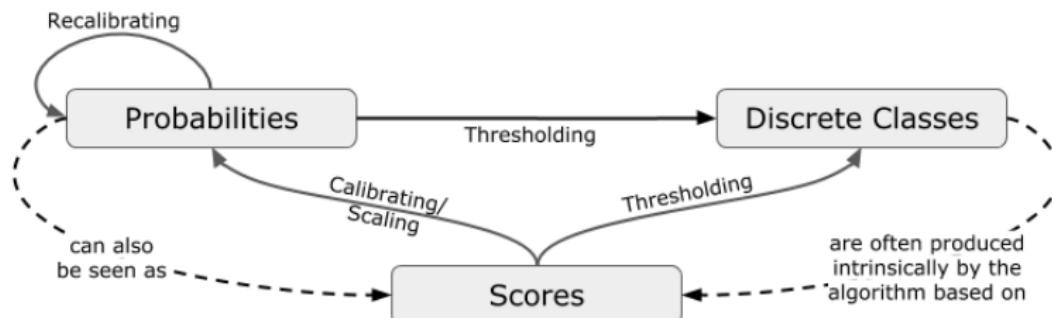
- Probabilities  $\pi_1(\mathbf{x}), \dots, \pi_g(\mathbf{x})$  are transformed into labels by predicting the class with the maximum probability

$$h(\mathbf{x}) = \arg \max_{k \in \{1, \dots, g\}} \pi_k(\mathbf{x})$$

- For  $g = 2$  one  $\pi(\mathbf{x})$  is constructed (note that it would be natural here to label the classes with  $\{0, 1\}$ )
- Probabilistic classifiers can also be seen as scoring classifiers
- If we want to emphasize that our model outputs probabilities, we denote the model as  $\pi(\mathbf{x}) : \mathcal{X} \rightarrow [0, 1]^g$ ; if we are talking about models in a general sense, we write  $f$ , comprising both probabilistic and scoring classifiers (context will make this clear!)

# PROBABILISTIC CLASSIFIERS

- Both scoring and probabilistic classifiers can output classes by thresholding (binary case) / selecting the class with the maximum score (multiclass)
- Thresholding:  $h(\mathbf{x}) := [\pi(\mathbf{x}) \geq c]$  or  $h(\mathbf{x}) = [f(\mathbf{x}) \geq c]$  for some threshold  $c$ .
- Usually  $c = 0.5$  for probabilistic,  $c = 0$  for scoring classifiers.
- There are also versions of thresholding for the multiclass case



# DECISION REGIONS AND BOUNDARIES

- A **decision region** for class  $k$  is the set of input points  $\mathbf{x}$  where class  $k$  is assigned as prediction of our model:

$$\mathcal{X}_k = \{\mathbf{x} \in \mathcal{X} : h(\mathbf{x}) = k\}$$

- Points in space where the classes with maximal score are tied and the corresponding hypersurfaces are called **decision boundaries**

$$\{\mathbf{x} \in \mathcal{X} : \begin{aligned} & \exists i \neq j \text{ s.t. } f_i(\mathbf{x}) = f_j(\mathbf{x}) \\ & \text{and } f_i(\mathbf{x}), f_j(\mathbf{x}) \geq f_k(\mathbf{x}) \forall k \neq i, j \} \end{aligned}$$

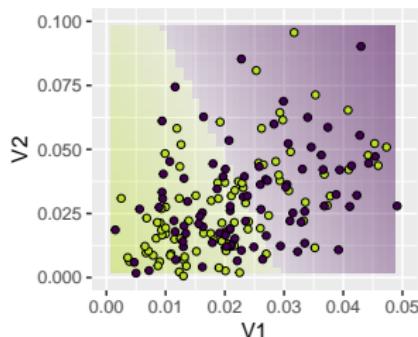
In the binary case we can simplify and generalize to the decision boundary for general threshold  $c$ :

$$\{\mathbf{x} \in \mathcal{X} : f(\mathbf{x}) = c\}$$

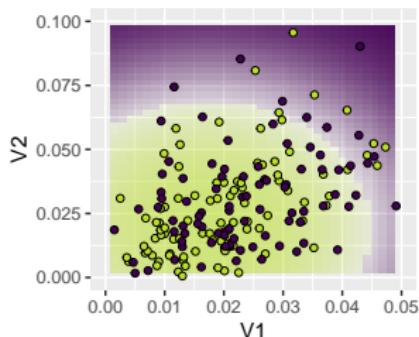
If we set  $c = 0$  for scores and  $c = 0.5$  for probabilities, this is consistent with the definition above.

# DECISION BOUNDARY EXAMPLES

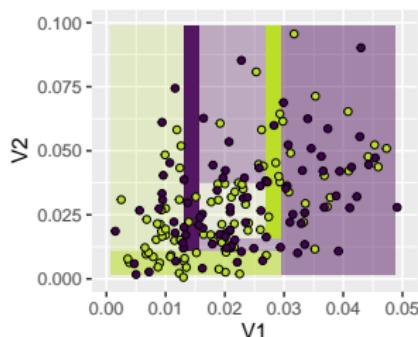
Log. Regr.



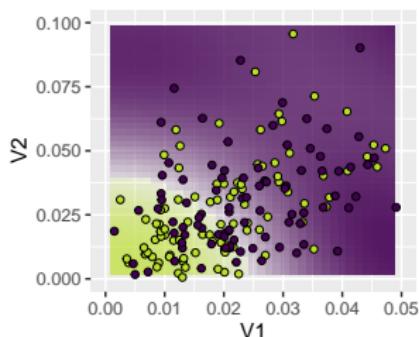
Naive Bayes



CART Tree



SVM



Response

M  
R

Response

M  
R

Response

M  
R

Response

M  
R

# CLASSIFICATION APPROACHES

Two fundamental approaches exist to construct classifiers:

The **generative approach** and the **discriminant approach**.

They tackle the classification problem from different angles:

- **Generative** classification approaches assume a data-generating process in which the distribution of the features  $\mathbf{x}$  is different for the various classes of the output  $y$ , and try to learn these conditional distributions:  
“Which  $y$  tends to have  $\mathbf{x}$  like these?”

- **Discriminant** approaches use **empirical risk minimization** based on a suitable loss function:  
“What is the best prediction for  $y$  given these  $\mathbf{x}$ ? ”

# GENERATIVE APPROACH

The **generative approach** models  $p(\mathbf{x}|y = k)$ , usually by making some assumptions about the structure of these distributions, and employs the Bayes theorem:

$$\pi_k(\mathbf{x}) = \mathbb{P}(y = k | \mathbf{x}) = \frac{\mathbb{P}(\mathbf{x}|y = k)\mathbb{P}(y = k)}{\mathbb{P}(\mathbf{x})} = \frac{p(\mathbf{x}|y = k)\pi_k}{\sum_{j=1}^g p(\mathbf{x}|y = j)\pi_j}$$

Prior class probabilities  $\pi_k$  are easy to estimate from the training data.

Examples:

- Naive Bayes classifier
- Linear discriminant analysis (generative, linear)
- Quadratic discriminant analysis (generative, not linear)

Note: LDA and QDA have 'discriminant' in their name, but are generative models! (... sorry.)

# DISCRIMINANT APPROACH

The **discriminant approach** tries to optimize the discriminant functions directly, usually via empirical risk minimization.

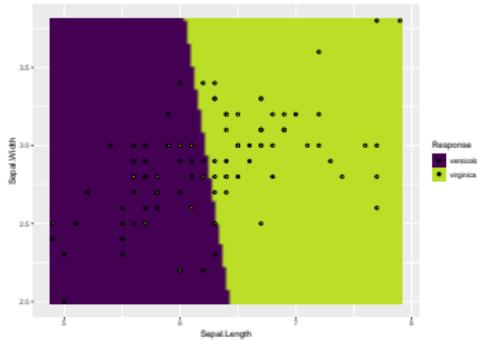
$$\hat{f} = \arg \min_{f \in \mathcal{H}} \mathcal{R}_{\text{emp}}(f) = \arg \min_{f \in \mathcal{H}} \sum_{i=1}^n L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right).$$

Examples:

- Logistic regression (discriminant, linear)
- Neural networks
- Support vector machines

# Introduction to Machine Learning

## Classification: Linear Classifiers



### Learning goals

- Know the definition of a linear classifier

# LINEAR CLASSIFIERS

Linear classifiers are an important subclass of classification models. If the discriminant function(s)  $f_k(\mathbf{x})$  can be specified as linear function(s) (possibly through a rank-preserving, monotone transformation  $g : \mathbb{R} \rightarrow \mathbb{R}$ ), i. e.

$$g(f_k(\mathbf{x})) = \mathbf{w}_k^\top \mathbf{x} + b_k,$$

we will call the classifier a **linear classifier**.

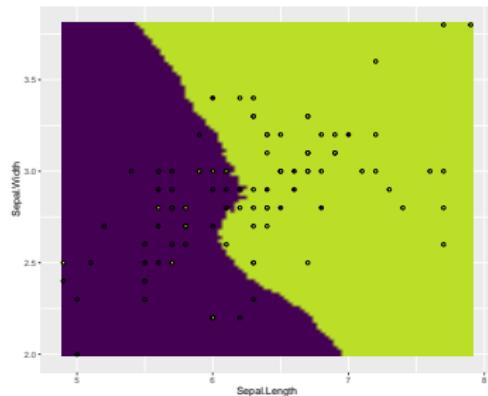
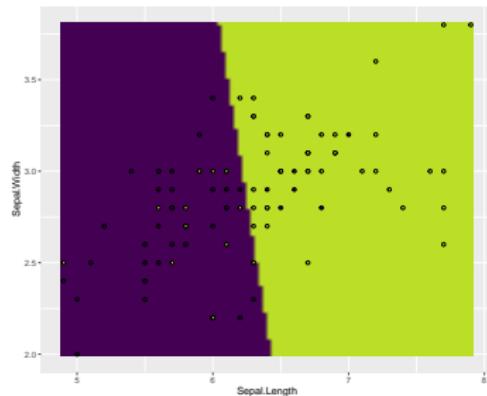
# LINEAR CLASSIFIERS

We can also easily show that the decision boundary between classes  $i$  and  $j$  is a hyperplane. For every  $\mathbf{x}$  where there is a tie in scores:

$$\begin{aligned}f_i(\mathbf{x}) &= f_j(\mathbf{x}) \\g(f_i(\mathbf{x})) &= g(f_j(\mathbf{x})) \\\mathbf{w}_i^\top \mathbf{x} + b_i &= \mathbf{w}_j^\top \mathbf{x} + b_j \\(\mathbf{w}_i - \mathbf{w}_j)^\top \mathbf{x} + (b_i - b_j) &= 0\end{aligned}$$

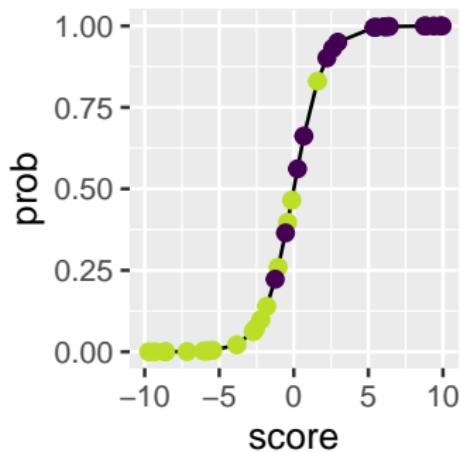
This is a **hyperplane** separating two classes.

# LINEAR VS NONLINEAR DECISION BOUNDARY



# Introduction to Machine Learning

## Classification: Logistic Regression



### Learning goals

- Understand the definition of the logit model
- Understand how a reasonable loss function for binary classification can be derived
- Know the hypothesis space that belongs to the logit model

# MOTIVATION

A **discriminant** approach for directly modeling the posterior probabilities  $\pi(\mathbf{x} | \theta)$  of the labels is **logistic regression**.

For now, let's focus on the binary case  $y \in \{0, 1\}$  and use empirical risk minimization.

$$\arg \min_{\theta \in \Theta} \mathcal{R}_{\text{emp}}(\theta) = \arg \min_{\theta \in \Theta} \sum_{i=1}^n L\left(y^{(i)}, \pi\left(\mathbf{x}^{(i)} | \theta\right)\right).$$

A naive approach would be to model

$$\pi(\mathbf{x} | \theta) = \theta^T \mathbf{x}.$$

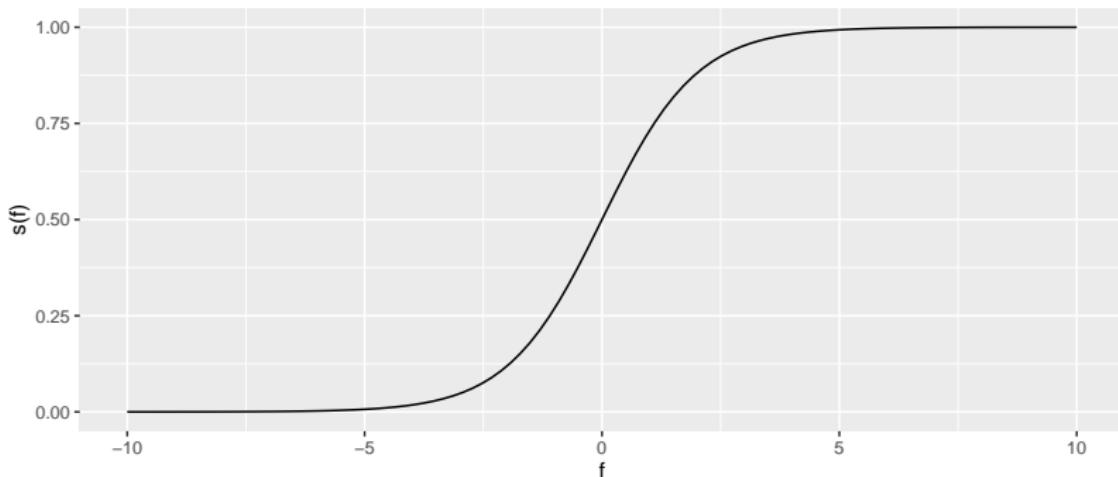
NB: We will often suppress the intercept in notation.

Obviously this could result in predicted probabilities  $\pi(\mathbf{x} | \theta) \notin [0, 1]$ .

# LOGISTIC FUNCTION

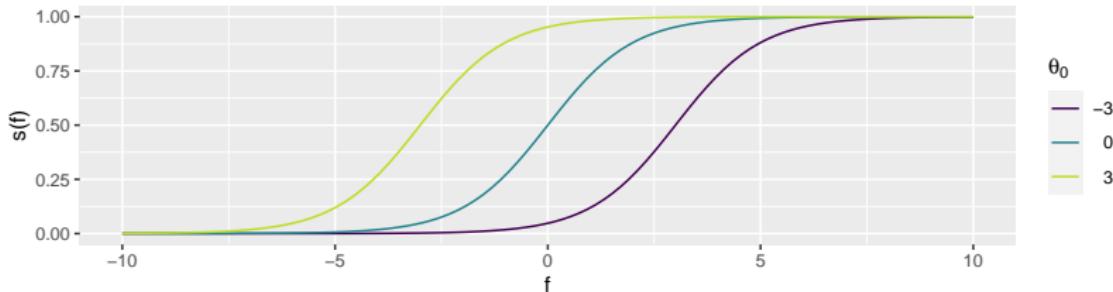
To avoid this, logistic regression “squashes” the estimated linear scores  $\theta^T \mathbf{x}$  to  $[0, 1]$  through the **logistic function**  $s$ :

$$\pi(\mathbf{x} | \theta) = \frac{\exp(\theta^T \mathbf{x})}{1 + \exp(\theta^T \mathbf{x})} = \frac{1}{1 + \exp(-\theta^T \mathbf{x})} = s(\theta^T \mathbf{x})$$

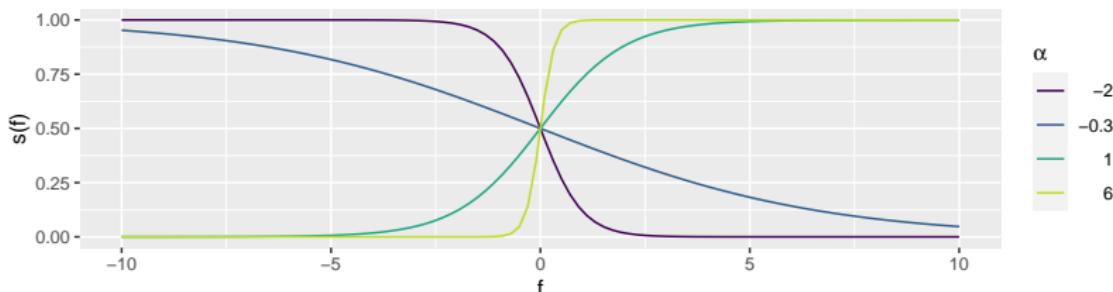


# LOGISTIC FUNCTION

The intercept shifts  $s(f)$  horizontally  $s(\theta_0 + f) = \frac{\exp(\theta_0 + f)}{1 + \exp(\theta_0 + f)}$



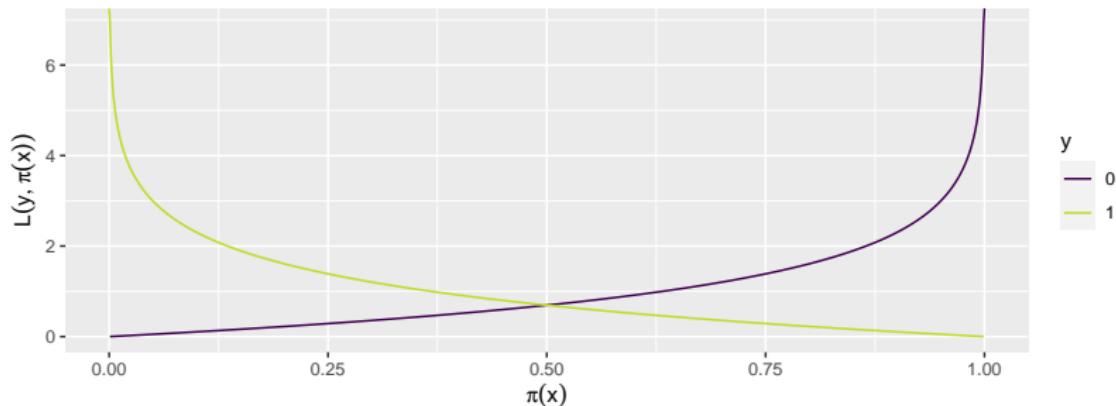
Scaling  $f$  like  $s(\alpha f) = \frac{\exp(\alpha f)}{1 + \exp(\alpha f)}$  controls the slope and direction.



# BERNOULLI / LOG LOSS

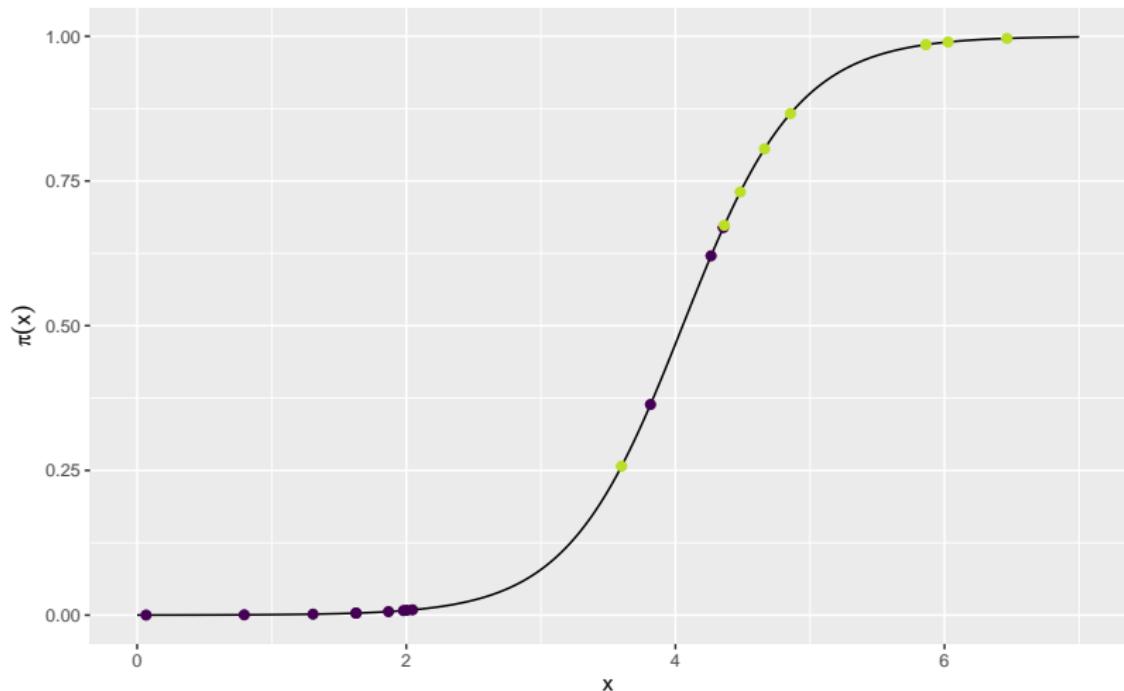
We need to define a loss function for the ERM approach:

- $L(y, \pi(\mathbf{x})) = -y \ln(\pi(\mathbf{x})) - (1 - y) \ln(1 - \pi(\mathbf{x}))$
- Penalizes confidently wrong predictions heavily
- Called Bernoulli, log or cross-entropy loss
- We can derive it from the negative log-likelihood of Bernoulli / logistic regression model in statistics
- Used for many other classifiers, e.g., in NNs or boosting



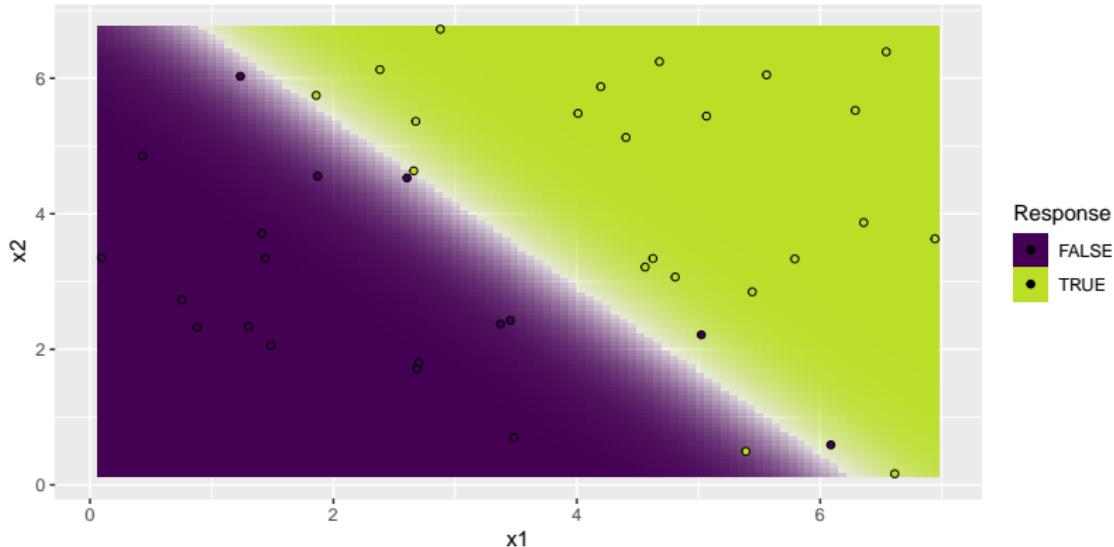
# LOGISTIC REGRESSION IN 1D

With one feature  $\mathbf{x} \in \mathbb{R}$ . The figure shows data and  $\mathbf{x} \mapsto \pi(\mathbf{x})$ .

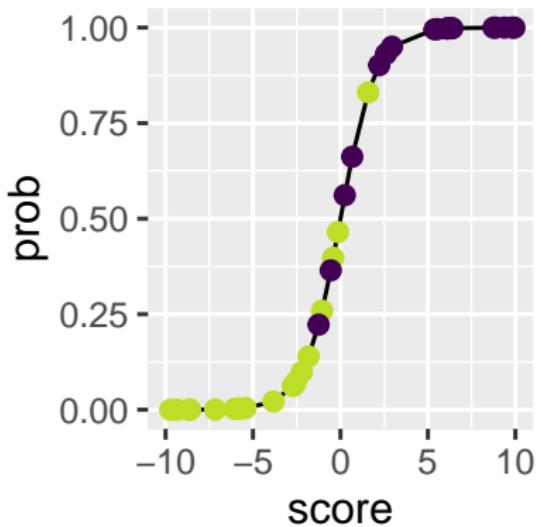
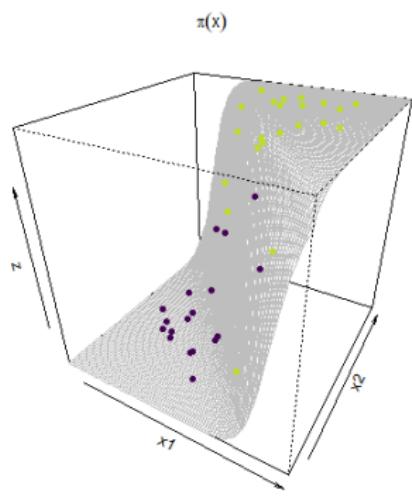


# LOGISTIC REGRESSION IN 2D

Obviously, logistic regression is a linear classifier, as  $\pi(\mathbf{x} | \theta) = s(\theta^T \mathbf{x})$  and  $s$  is isotonic.



# LOGISTIC REGRESSION IN 2D



# SUMMARY

## Hypothesis Space:

$$\mathcal{H} = \{\pi : \mathcal{X} \rightarrow [0, 1] \mid \pi(\mathbf{x}) = s(\boldsymbol{\theta}^T \mathbf{x})\}$$

**Risk:** Logistic/Bernoulli loss function.

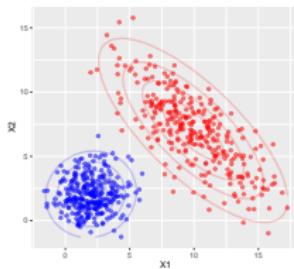
$$L(y, \pi(\mathbf{x})) = -y \ln(\pi(\mathbf{x})) - (1 - y) \ln(1 - \pi(\mathbf{x}))$$

**Optimization:** Numerical optimization, typically gradient-based methods.

# Introduction to Machine Learning

## Classification: Discriminant Analysis

### Learning goals



- Understand the ideas of linear and quadratic discriminant analysis
- Understand how parameters are estimated for LDA and QDA
- Understand how decision boundaries are computed for LDA and QDA

# LINEAR DISCRIMINANT ANALYSIS (LDA)

LDA follows a generative approach

$$\pi_k(\mathbf{x}) = \mathbb{P}(y = k \mid \mathbf{x}) = \frac{\mathbb{P}(\mathbf{x}|y = k)\mathbb{P}(y = k)}{\mathbb{P}(\mathbf{x})} = \frac{p(\mathbf{x}|y = k)\pi_k}{\sum_{j=1}^g p(\mathbf{x}|y = j)\pi_j},$$

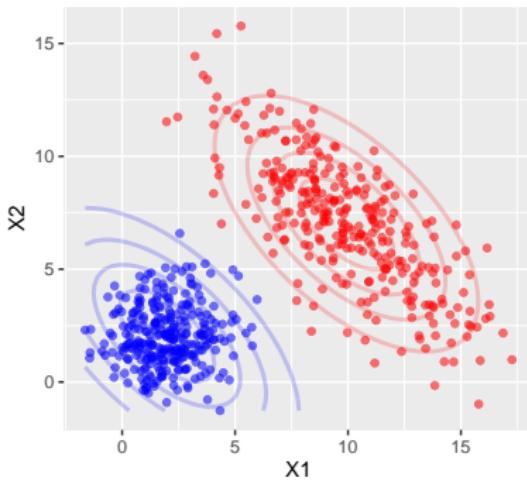
where we now have to pick a distributional form for  $p(\mathbf{x}|y = k)$ .

# LINEAR DISCRIMINANT ANALYSIS (LDA)

LDA assumes that each class density is modeled as a *multivariate Gaussian*:

$$p(\mathbf{x}|y = k) = \frac{1}{(2\pi)^{\frac{p}{2}} |\Sigma|^{\frac{1}{2}}} \exp \left( -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}_k) \right)$$

with equal covariance, i. e.  $\boldsymbol{\Sigma}_k = \boldsymbol{\Sigma} \quad \forall k$ .



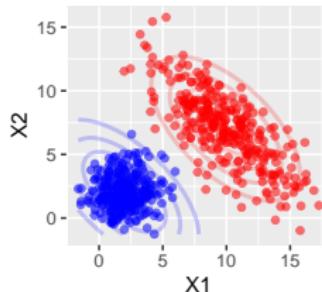
# LINEAR DISCRIMINANT ANALYSIS (LDA)

Parameters  $\theta$  are estimated in a straightforward manner by estimating

$$\hat{\pi}_k = \frac{n_k}{n}, \text{ where } n_k \text{ is the number of class-}k \text{ observations}$$

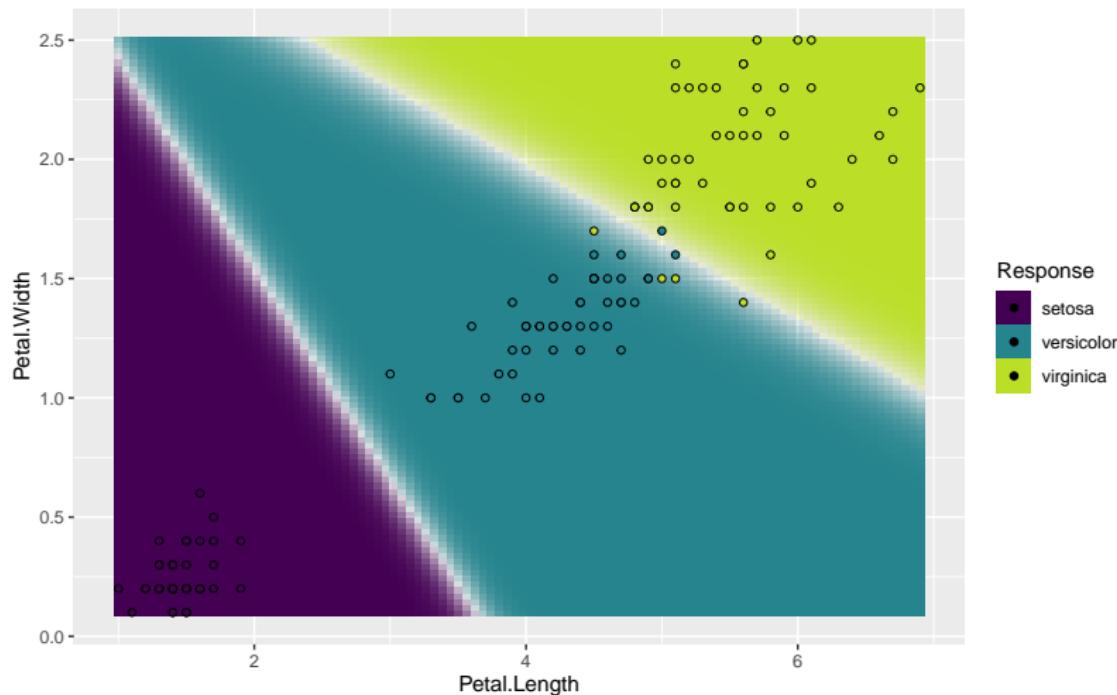
$$\hat{\mu}_k = \frac{1}{n_k} \sum_{i:y^{(i)}=k} \mathbf{x}^{(i)}$$

$$\hat{\Sigma} = \frac{1}{n-g} \sum_{k=1}^g \sum_{i:y^{(i)}=k} (\mathbf{x}^{(i)} - \hat{\mu}_k)(\mathbf{x}^{(i)} - \hat{\mu}_k)^T$$



# LDA AS LINEAR CLASSIFIER

Because of the equal covariance structure of all class-specific Gaussian, the decision boundaries of LDA are linear.



## LDA AS LINEAR CLASSIFIER

We can formally show that LDA is a linear classifier, by showing that the posterior probabilities can be written as linear scoring functions - up to any isotonic / rank-preserving transformation.

$$\pi_k(\mathbf{x}) = \frac{\pi_k \cdot p(\mathbf{x}|y=k)}{p(\mathbf{x})} = \frac{\pi_k \cdot p(\mathbf{x}|y=k)}{\sum_{j=1}^g \pi_j \cdot p(\mathbf{x}|y=j)}$$

As the denominator is the same for all classes we only need to consider

$$\pi_k \cdot p(\mathbf{x}|y=k)$$

and show that this can be written as a linear function of  $\mathbf{x}$ .

# LDA AS LINEAR CLASSIFIER

$$\begin{aligned} & \pi_k \cdot p(\mathbf{x}|y=k) \\ \propto & \pi_k \exp\left(-\frac{1}{2}\mathbf{x}^T \Sigma^{-1} \mathbf{x} - \frac{1}{2}\boldsymbol{\mu}_k^T \Sigma^{-1} \boldsymbol{\mu}_k + \mathbf{x}^T \Sigma^{-1} \boldsymbol{\mu}_k\right) \\ = & \exp\left(\log \pi_k - \frac{1}{2}\boldsymbol{\mu}_k^T \Sigma^{-1} \boldsymbol{\mu}_k + \mathbf{x}^T \Sigma^{-1} \boldsymbol{\mu}_k\right) \exp\left(-\frac{1}{2}\mathbf{x}^T \Sigma^{-1} \mathbf{x}\right) \\ = & \exp(\boldsymbol{\theta}_{0k} + \mathbf{x}^T \boldsymbol{\theta}_k) \exp\left(-\frac{1}{2}\mathbf{x}^T \Sigma^{-1} \mathbf{x}\right) \\ \propto & \exp(\boldsymbol{\theta}_{0k} + \mathbf{x}^T \boldsymbol{\theta}_k) \end{aligned}$$

by defining  $\boldsymbol{\theta}_{0k} := \log \pi_k - \frac{1}{2}\boldsymbol{\mu}_k^T \Sigma^{-1} \boldsymbol{\mu}_k$  and  $\boldsymbol{\theta}_k := \Sigma^{-1} \boldsymbol{\mu}_k$ .

We have again left out all constants which are the same for all classes  $k$ , so the normalizing constant of our Gaussians and  $\exp\left(-\frac{1}{2}\mathbf{x}^T \Sigma^{-1} \mathbf{x}\right)$ .

By finally taking the log, we can write our transformed scores as linear:

$$f_k(\mathbf{x}) = \boldsymbol{\theta}_{0k} + \mathbf{x}^T \boldsymbol{\theta}_k$$

# QUADRATIC DISCRIMINANT ANALYSIS (QDA)

QDA is a direct generalization of LDA, where the class densities are now Gaussians with unequal covariances  $\Sigma_k$ .

$$p(\mathbf{x}|y=k) = \frac{1}{(2\pi)^{\frac{p}{2}} |\Sigma_k|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x} - \boldsymbol{\mu}_k)\right)$$

Parameters are estimated in a straightforward manner by:

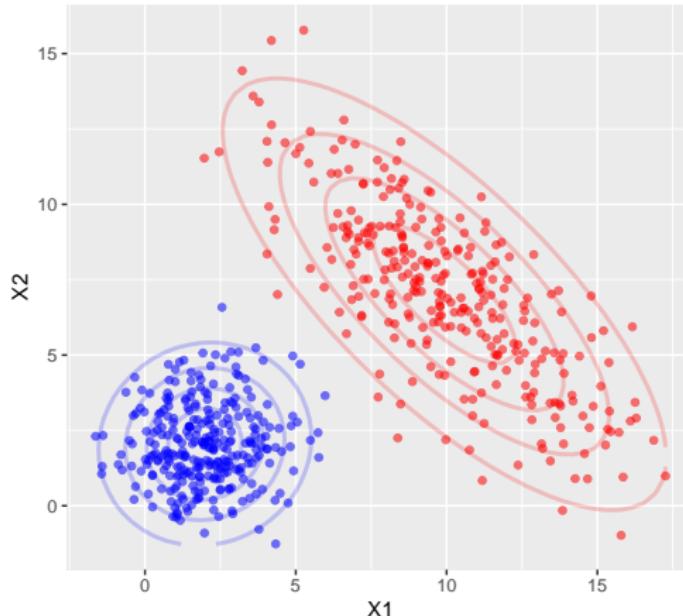
$$\hat{\pi}_k = \frac{n_k}{n}, \text{ where } n_k \text{ is the number of class-}k \text{ observations}$$

$$\hat{\boldsymbol{\mu}}_k = \frac{1}{n_k} \sum_{i:y^{(i)}=k} \mathbf{x}^{(i)}$$

$$\hat{\boldsymbol{\Sigma}}_k = \frac{1}{n_k - 1} \sum_{i:y^{(i)}=k} (\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}_k)(\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}_k)^T$$

# QUADRATIC DISCRIMINANT ANALYSIS (QDA)

- Covariance matrices can differ over classes.
- Yields better data fit but also requires estimation of more parameters.



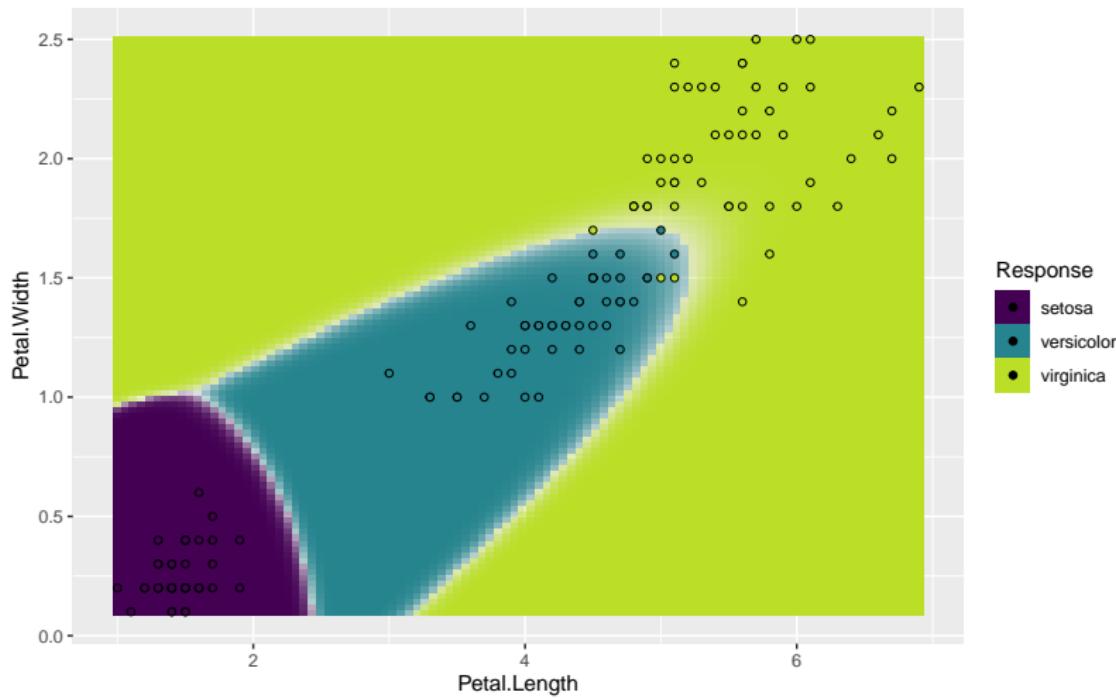
# QUADRATIC DISCRIMINANT ANALYSIS (QDA)

$$\begin{aligned}\pi_k(\mathbf{x}) &\propto \pi_k \cdot p(\mathbf{x}|y=k) \\ &\propto \pi_k |\Sigma_k|^{-\frac{1}{2}} \exp\left(-\frac{1}{2} \mathbf{x}^T \Sigma_k^{-1} \mathbf{x} - \frac{1}{2} \boldsymbol{\mu}_k^T \Sigma_k^{-1} \boldsymbol{\mu}_k + \mathbf{x}^T \Sigma_k^{-1} \boldsymbol{\mu}_k\right)\end{aligned}$$

Taking the log of the above, we can define a discriminant function that is quadratic in  $x$ .

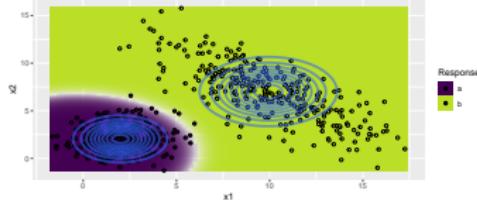
$$\log \pi_k - \frac{1}{2} \log |\Sigma_k| - \frac{1}{2} \boldsymbol{\mu}_k^T \Sigma_k^{-1} \boldsymbol{\mu}_k + \mathbf{x}^T \Sigma_k^{-1} \boldsymbol{\mu}_k - \frac{1}{2} \mathbf{x}^T \Sigma_k^{-1} \mathbf{x}$$

# QUADRATIC DISCRIMINANT ANALYSIS (QDA)



# Introduction to Machine Learning

## Classification: Naive Bayes



### Learning goals

- Understand the idea of Naive Bayes
- Understand in which sense Naive Bayes is a special QDA model

# NAIVE BAYES CLASSIFIER

NB is a generative multiclass technique. Remember: We use Bayes' theorem and only need  $p(\mathbf{x}|y = k)$  to compute the posterior as:

$$\pi_k(\mathbf{x}) = \mathbb{P}(y = k | \mathbf{x}) = \frac{\mathbb{P}(\mathbf{x}|y = k)\mathbb{P}(y = k)}{\mathbb{P}(\mathbf{x})} = \frac{p(\mathbf{x}|y = k)\pi_k}{\sum_{j=1}^g p(\mathbf{x}|y = j)\pi_j}$$

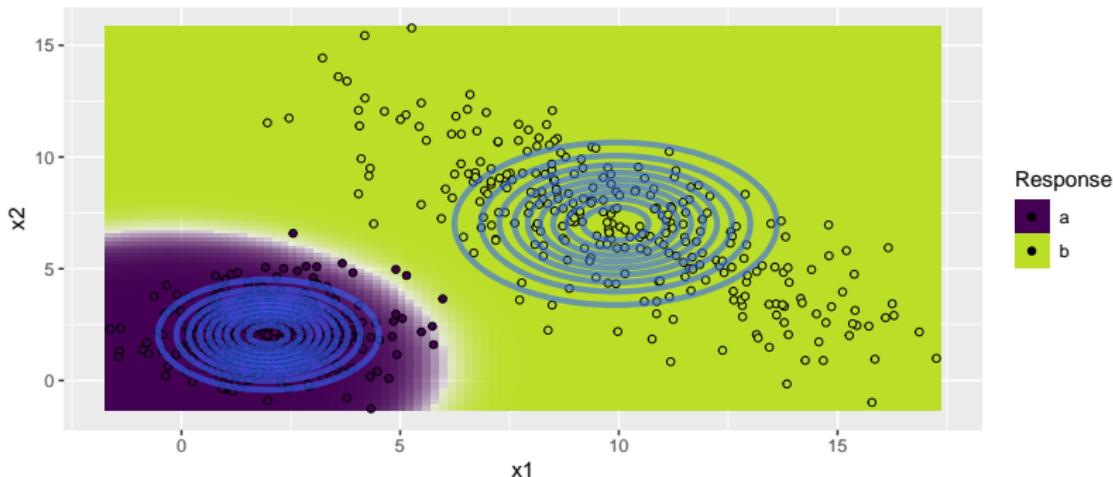
NB is based on a simple **conditional independence assumption**: the features are conditionally independent given class  $y$ .

$$p(\mathbf{x}|y = k) = p((x_1, x_2, \dots, x_p)|y = k) = \prod_{j=1}^p p(x_j|y = k).$$

So we only need to specify and estimate the distribution  $p(x_j|y = k)$ , which is considerably simpler as this is univariate.

## NB: NUMERICAL FEATURES

We use a univariate Gaussian for  $p(x_j|y = k)$ , and estimate  $(\mu_j, \sigma_j^2)$  in the standard manner. Because of  $p(\mathbf{x}|y = k) = \prod_{j=1}^p p(x_j|y = k)$ , the joint conditional density is Gaussian with diagonal but non-isotropic covariance structure, and potentially different across classes. Hence, NB is a (specific) QDA model, with quadratic decision boundary.



## NB: CATEGORICAL FEATURES

We use a categorical distribution for  $p(x_j|y = k)$  and estimate the probabilities  $p_{kjm}$  that, in class  $k$ , our  $j$ -th feature has value  $m$ ,  $x_j = m$ , simply by counting the frequencies.

$$p(x_j|y = k) = \prod_m p_{kjm}^{[x_j=m]}$$

Because of the simple conditional independence structure it is also very easy to deal with mixed numerical / categorical feature spaces.

## LAPLACE SMOOTHING

If a given class and feature value never occur together in the training data, then the frequency-based probability estimate will be zero.

This is problematic because it will wipe out all information in the other probabilities when they are multiplied.

A simple numerical correction is to set these zero probabilities to a small value to regularize against this case.

# NAIVE BAYES: APPLICATION AS SPAM FILTER

- In the late 90s, Naive Bayes became popular for e-mail spam filter programs
- Word counts were used as features to detect spam mails (e.g., "Viagra" often occurs in spam mail)
- Independence assumption implies: occurrence of two words in mail is not correlated
- Seems naive ("Viagra" more likely to occur in context with "Buy now" than "flower"), but leads to less required parameters and therefore better generalization, and often works well in practice.

# INTRODUCTION TO MACHINE LEARNING

ML Basics

Supervised Regression

Supervised Classification

**k-NN**

Performance Evaluation

Classification and Regression Trees (CART)

Random Forests

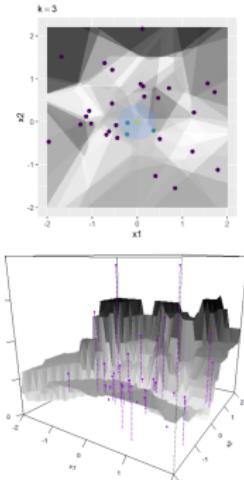
Tuning

Nested Resampling

mlr3

# Introduction to Machine Learning

## k-Nearest Neighbors

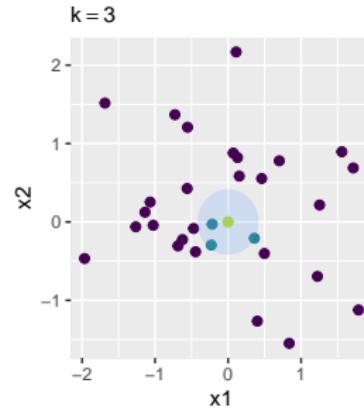
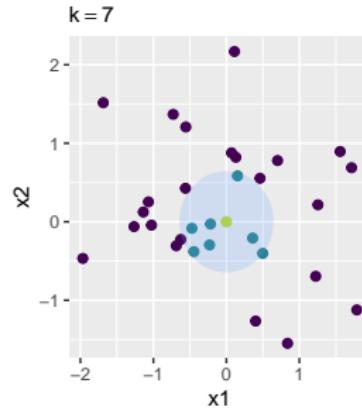
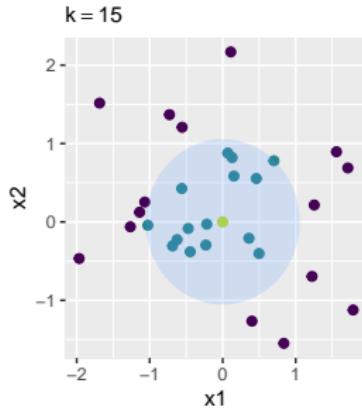


### Learning goals

- Understand the basic idea of  $k$ -NN for regression and classification
- Understand that  $k$ -NN is a non-parametric, local model
- Know different distance measures for different scales of feature variables

# K-NEAREST-NEIGHBORS

- **$k$ -NN** can be used for regression and classification.
- Generates "similar" predictions for  $\mathbf{x}$  as its  $k$  closest neighbors.
- "Closeness" requires a distance or similarity measure.
- The set containing the  $k$  closest points  $\mathbf{x}^{(i)}$  to  $\mathbf{x}$  in the training data is called the  **$k$ -neighborhood**  $N_k(\mathbf{x})$  of  $\mathbf{x}$ .

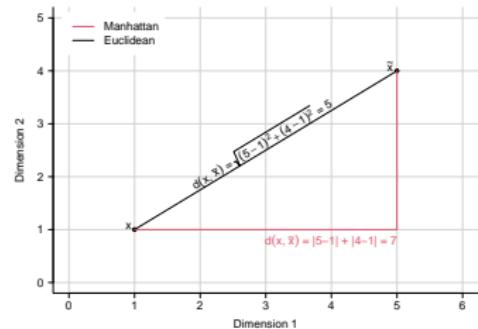


# DISTANCE MEASURES

## How to calculate distances?

- Most popular distance measure for numerical features:  
**Euclidean distance**
- For two data points  $\mathbf{x}$  and  $\tilde{\mathbf{x}}$  with  $p$  numeric features

$$\bullet d_{\text{Euclidean}}(\mathbf{x}, \tilde{\mathbf{x}}) = \sqrt{\sum_{j=1}^p (x_j - \tilde{x}_j)^2}.$$
$$\bullet d_{\text{manhattan}}(\mathbf{x}, \tilde{\mathbf{x}}) = \sum_{j=1}^p |x_j - \tilde{x}_j|.$$

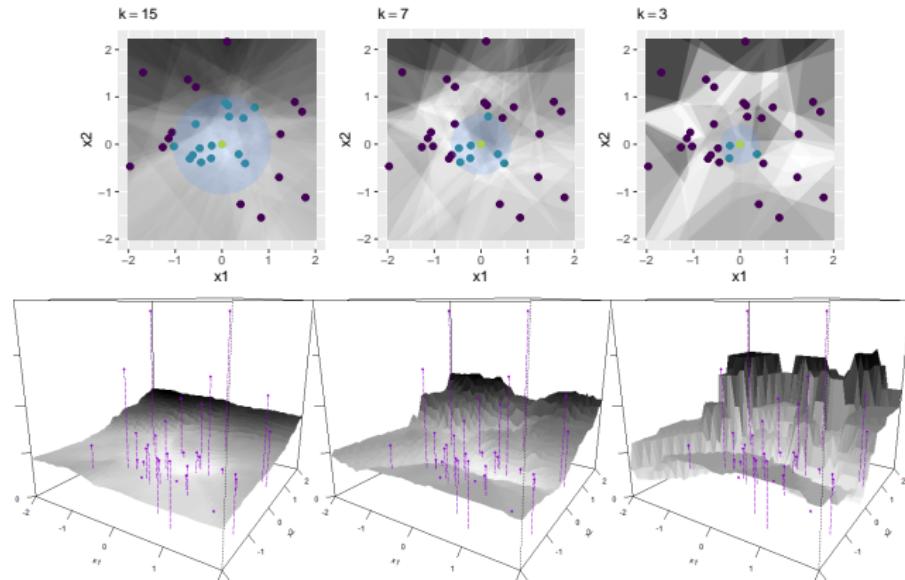


# PREDICTION - REGRESSION

Compute for each point the average output  $y$  of the  $k$ -nearest neighbours in  $N_k(\mathbf{x})$ :

$$\hat{f}(\mathbf{x}) = \frac{1}{k} \sum_{i:\mathbf{x}^{(i)} \in N_k(\mathbf{x})} y^{(i)} \text{ or } \hat{f}(\mathbf{x}) = \frac{1}{\sum_{i:\mathbf{x}^{(i)} \in N_k(\mathbf{x})} w^{(i)}} \sum_{i:\mathbf{x}^{(i)} \in N_k(\mathbf{x})} w^{(i)} y^{(i)}$$

with neighbors weighted based on their distance to  $\mathbf{x}$ :  $w^{(i)} = \frac{1}{d(\mathbf{x}^{(i)}, \mathbf{x})}$



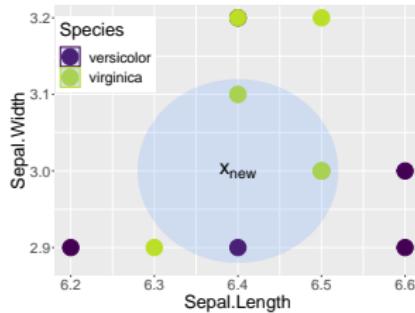
# PREDICTION - CLASSIFICATION

For classification in  $g$  groups, a majority vote is used:

$$\hat{h}(\mathbf{x}) = \arg \max_{\ell \in \{1, \dots, g\}} \sum_{i: \mathbf{x}^{(i)} \in N_k(\mathbf{x})} \mathbb{I}(y^{(i)} = \ell)$$

And posterior probabilities can be estimated with:

$$\hat{\pi}_\ell(\mathbf{x}) = \frac{1}{k} \sum_{i: \mathbf{x}^{(i)} \in N_k(\mathbf{x})} \mathbb{I}(y^{(i)} = \ell)$$

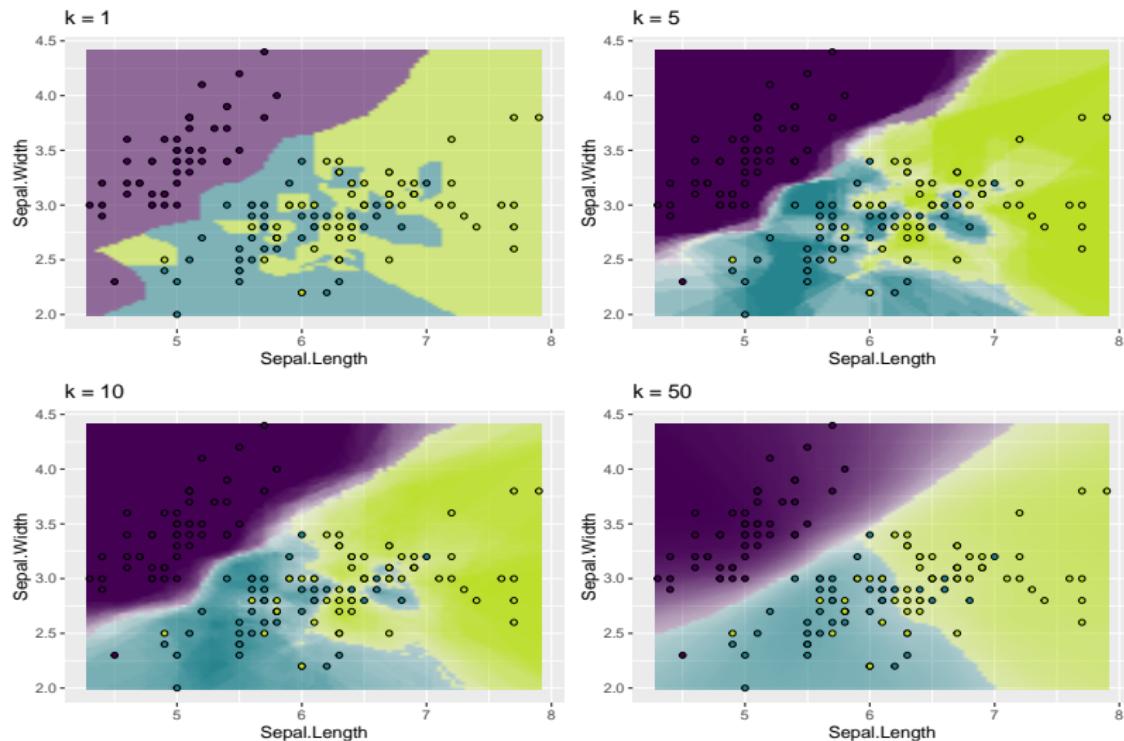


	SL	SW	Species	dist
52	6.4	3.2	versicolor	0.200
59	6.6	2.9	versicolor	0.224
75	6.4	2.9	versicolor	0.100
76	6.6	3.0	versicolor	0.200
98	6.2	2.9	versicolor	0.224
104	6.3	2.9	virginica	0.141
105	6.5	3.0	virginica	0.100
111	6.5	3.2	virginica	0.224
116	6.4	3.2	virginica	0.200
117	6.5	3.0	virginica	0.100
138	6.4	3.1	virginica	0.100
148	6.5	3.0	virginica	0.100

**Example with subset of iris data ( $k = 3$ )**

$$\hat{\pi}_{setosa}(\mathbf{x}_{new}) = \frac{0}{3} = 0\%, \hat{\pi}_{versicolor}(\mathbf{x}_{new}) = \frac{1}{3} = 33\%, \hat{\pi}_{virginica}(\mathbf{x}_{new}) = \frac{2}{3} = 67\%,$$
$$\hat{h}(\mathbf{x}_{new}) = \text{virginica}$$

# K-NN: FROM SMALL TO LARGE $K$



Complex, local model vs smoother, more global model

# K-NN SUMMARY

- $k$ -NN is a lazy classifier, it has no real training step, it simply stores the complete data - which are needed during prediction.
- Hence, its parameters are the training data, there is no real compression of information.
- As the number of parameters grows with the number of training points, we call  $k$ -NN a non-parametric model
- $k$ -NN is not based on any distributional or functional assumption, and can, in theory, model data situations of arbitrary complexity.
- The smaller  $k$ , the less stable, less smooth and more “wiggly” the decision boundary becomes.
- Accuracy of  $k$ -NN can be severely degraded by the presence of noisy or irrelevant features, or when the feature scales are not consistent with their importance.

# STANDARDIZATION AND WEIGHTS

- **Standardization:** Features in k-NN are usually standardized or normalized. If two features have values on a very different range, most distances would place a higher importance on the one with a larger range, leading to an imbalanced influence of that feature.
- **Importance:** Sometimes one feature has a higher importance (maybe we know this via domain knowledge). It can now manually be upweighted to reflect this.

$$d_{\text{Euclidean}}^{\text{weighted}}(\mathbf{x}, \tilde{\mathbf{x}}) = \sqrt{\sum_{j=1}^p w_j (x_j - \tilde{x}_j)^2}$$

- If these weights would have to be learned in a data-driven manner, we could only do this by hyperparameter tuning in k-NN. This is inconvenient, and Gaussian processes handle this much better.

# GOWER DISTANCE

- A weighted mean of univ. distances in the  $j$ -th feature.
- It can handle categoricals, missings, and different ranges.

$$d_{gower}(\mathbf{x}, \tilde{\mathbf{x}}) = \frac{\sum_{j=1}^p \delta_{x_j, \tilde{x}_j} \cdot d_{gower}(x_j, \tilde{x}_j)}{\sum_{j=1}^p \delta_{x_j, \tilde{x}_j}}.$$

- $\delta_{x_j, \tilde{x}_j}$  is 0 or 1. It's 0 if  $j$ -th feature is *missing* in at least one observation, or when the feature is asymmetric binary (where “1” is more important than “0”) and both values are zero. Otherwise 1.
- $d_{gower}(x_j, \tilde{x}_j)$ : For nominals it's 0 if both values are equal and 1 otherwise. For integers and reals, it's the absolute difference, divided by range.

# GOWER DISTANCE

Example of Gower distance with data on sex and income:

index	sex	salary
1	m	2340
2	w	2100
3	NA	2680

$$d_{gower}(\mathbf{x}, \tilde{\mathbf{x}}) = \frac{\sum_{j=1}^p \delta_{x_j, \tilde{x}_j} \cdot d_{gower}(x_j, \tilde{x}_j)}{\sum_{j=1}^p \delta_{x_j, \tilde{x}_j}}$$

$$d_{gower}(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) = \frac{1 \cdot 1 + 1 \cdot \frac{|2340 - 2100|}{|2680 - 2100|}}{1+1} = \frac{1 + \frac{240}{580}}{2} = \frac{1 + 0.414}{2} = 0.707$$

$$d_{gower}(\mathbf{x}^{(1)}, \mathbf{x}^{(3)}) = \frac{0 \cdot 1 + 1 \cdot \frac{|2340 - 2680|}{|2680 - 2100|}}{0+1} = \frac{0 + \frac{340}{580}}{1} = \frac{0 + 0.586}{1} = 0.586$$

$$d_{gower}(\mathbf{x}^{(2)}, \mathbf{x}^{(3)}) = \frac{0 \cdot 1 + 1 \cdot \frac{|2100 - 2680|}{|2680 - 2100|}}{0+1} = \frac{0 + \frac{580}{580}}{1} = \frac{0 + 1.000}{1} = 1$$

# INTRODUCTION TO MACHINE LEARNING

ML Basics

Supervised Regression

Supervised Classification

k-NN

## Performance Evaluation

Classification and Regression Trees (CART)

Random Forests

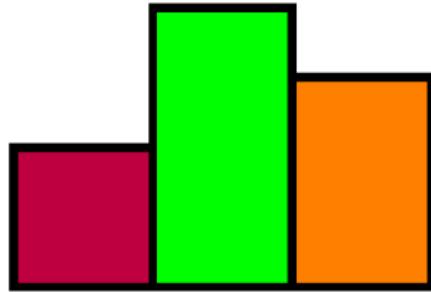
Tuning

Nested Resampling

mlr3

# Introduction to Machine Learning

## Evaluation: Introduction and Remarks



### Learning goals

- Understand the goal of performance estimation
- Understand the difference between outer and inner loss
- Know the definition of generalization error

# PERFORMANCE EVALUATION

How well does my model perform...



... on data from the same data-generating process?

In practice:

... on current data (training data)?

... on new data (test data)?

... based on a certain measure/metric?

...

# PERFORMANCE EVALUATION

ML performance evaluation provides clear and simple protocols for reliable model validation.

- Often simpler than classical statistical model diagnosis
- Relies only on few assumptions
- Still hard enough and offers **lots** of options to cheat / make mistakes

# PERFORMANCE MEASURES

We measure performance using a statistical estimator for the **generalization error** (GE).

GE = expected loss of a fixed model

$\hat{GE}$  = average loss

Example: Mean squared error (L2 loss)

$$\hat{GE} = MSE = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2$$

# MEASURES: INNER VS. OUTER LOSS

Inner loss = loss used in learning

Outer loss = loss used in evaluation  
= evaluation measure



# MEASURES: INNER VS. OUTER LOSS

Optimally: inner loss = outer loss

Not always possible:

some losses are hard to optimize / no loss is specified directly

Example:

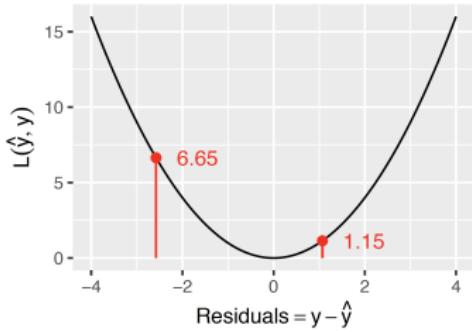
Logistic Regression → minimize binomial loss

kNN → no explicit loss minimization

- When evaluating the models we might be interested in (cost-weighted) classification error
- Or some of the more advanced measures from ROC analysis like AUC

# Introduction to Machine Learning

## Evaluation: Measures for Regression



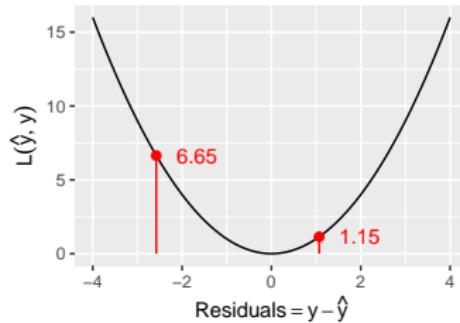
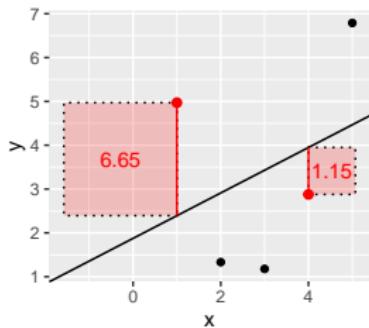
### Learning goals

- Know the definitions of mean squared error (MSE) and mean absolute error (MAE)
- Understand the connections of MSE and MAE to L2 and L1 loss
- Know the definitions of  $R^2$  and generalized  $R^2$

# MEAN SQUARED ERROR

$$MSE = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2 \in [0; \infty) \rightarrow \text{L2 loss.}$$

Single observations with a large prediction error heavily influence the **MSE**, as they enter quadratically.

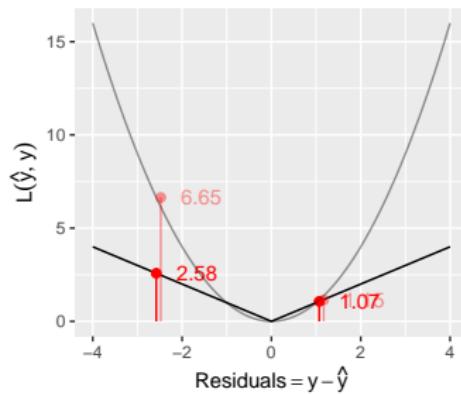
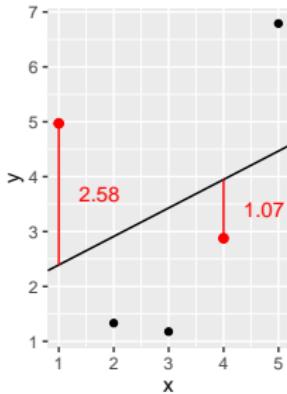


Similar measures: sum of squared errors (SSE), root mean squared error (RMSE, brings measurement back to the original scale of the outcome).

# MEAN ABSOLUTE ERROR

$$MAE = \frac{1}{n} \sum_{i=1}^n |y^{(i)} - \hat{y}^{(i)}| \in [0; \infty) \rightarrow \text{L1 loss.}$$

Less influenced by large errors and maybe more intuitive than the MSE.



Similar measures: median absolute error (for even more robustness).

$R^2$

Well-known measure from statistics.

$$R^2 = 1 - \frac{\sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2}{\sum_{i=1}^n (y^{(i)} - \bar{y})^2} = 1 - \frac{SSE_{LinMod}}{SSE_{Intercept}}$$

- Usually introduced as *fraction of variance explained* by the model
- Simpler: compares SSE of constant model (baseline) with complex model (LM)
- $R^2 = 1$ : all residuals are 0, we predict perfectly,  
 $R^2 = 0$ : we predict as badly as the constant model
- If measured on the training data,  $R^2 \in [0; 1]$  (LM must be at least as good as the constant)
- On other data  $R^2$  can even be negative as there is no guarantee that the LM generalizes better than a constant (overfitting)

# GENERALIZED $R^2$ FOR ML

A simple generalization of  $R^2$  for ML seems to be:

$$1 - \frac{\text{Loss}_{\text{ComplexModel}}}{\text{Loss}_{\text{SimplerModel}}}$$

- Works for arbitrary measures (not only SSE), for arbitrary models, on any data set of interest
- E.g. model vs constant, LM vs non-linear model, tree vs forest, model without some features vs model with them included
- Fairly unknown; our terminology (generalized  $R^2$ ) is non-standard

# Introduction to Machine Learning

## Evaluation: Simple Measures for Classification

### Learning goals

		True Class $y$	
		+	-
Pred.	+	True Positive (TP)	False Positive (FP)
	-	False Negative (FN)	True Negative (TN)
$\hat{y}$			

- Know the definitions of misclassification error rate (MCE) and accuracy (ACC)
- Understand the entries of a confusion matrix
- Understand the idea of costs
- Know definitions of Brier score and log loss

# LABELS VS PROBABILITIES

In classification we predict:

- ➊ Class labels  $\rightarrow \hat{h}(\mathbf{x}) = \hat{y}$
- ➋ Class probabilities  $\rightarrow \hat{\pi}_k(\mathbf{x})$

$\rightarrow$  We evaluate based on those

## LABELS: MCE

The misclassification error rate (MCE) counts the number of incorrect predictions and presents them as a rate:

$$MCE = \frac{1}{n} \sum_{i=1}^n [y^{(i)} \neq \hat{y}^{(i)}] \in [0; 1]$$

Accuracy is defined in a similar fashion for correct classifications:

$$ACC = \frac{1}{n} \sum_{i=1}^n [y^{(i)} = \hat{y}^{(i)}] \in [0; 1]$$

- If the data set is small this can be brittle
- The MCE says nothing about how good/skewed predicted probabilities are
- Errors on all classes are weighed equally (often inappropriate)

# LABELS: CONFUSION MATRIX

True classes in columns.

Predicted classes in rows.

	setosa	versicolor	virginica	-err.-	-n-
setosa	50	0	0	0	50
versicolor	0	46	4	4	50
virginica	0	4	46	4	50
-err.-	0	4	4	8	NA
-n-	50	50	50	NA	150

We can see class sizes (predicted and true) and where errors occur.

# LABELS: CONFUSION MATRIX

In binary classification

		True Class $y$	
		+	-
Pred.	+	True Positive (TP)	False Positive (FP)
	-	False Negative (FN)	True Negative (TN)

# LABELS: COSTS

We can also assign different costs to different errors via a cost matrix.

$$Costs = \frac{1}{n} \sum_{i=1}^n C[y^{(i)}, \hat{y}^{(i)}]$$

Example:

Predict if person has a ticket (yes / no).

Should train conductor check ticket of a person?

**Costs:**

Ticket checking: 3 EUR

Fee for fare-dodging: 40 EUR



<http://tiny.cc/meyarw>

# LABELS: COSTS

Predict if person has a ticket (yes / no).

Cost matrix C		
predicted		
true	no	yes
no	-37	0
yes	3	0

Confusion matrix		
predicted		
true	no	yes
no	7	0
yes	93	0

Confusion matrix * C		
predicted		
true	no	yes
no	-259	0
yes	279	0

## Costs:

Ticket checking: 3 EUR

Fee for fare-dodging: 40 EUR

Our model says that we should not trust anyone and check the tickets of all passengers.

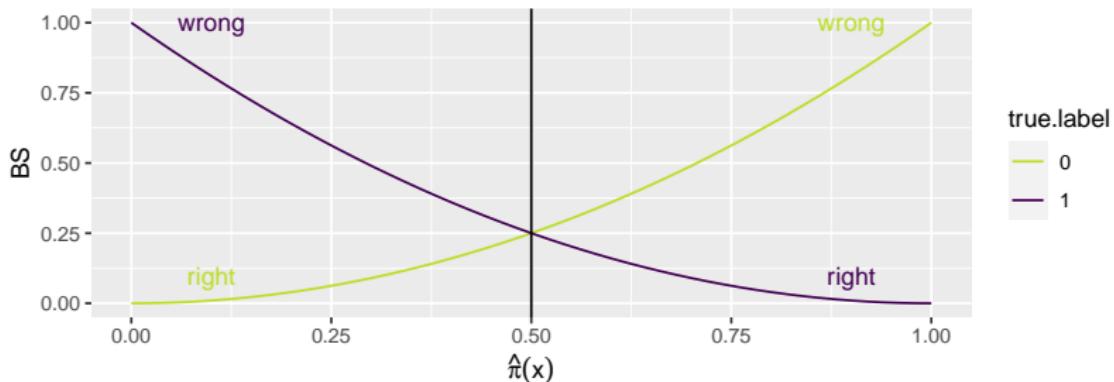
$$\begin{aligned} Costs &= \frac{1}{n} \sum_{i=1}^n C[y^{(i)}, \hat{y}^{(i)}] \\ &= \frac{1}{100} (-37 \cdot 7 + 0 \cdot 0 + 3 \cdot 93 + 0 \cdot 0) \\ &= \frac{20}{100} = 0.2 \end{aligned}$$

# PROBABILITIES: BRIER SCORE

Measures squared distances of probabilities from the true class labels:

$$BS1 = \frac{1}{n} \sum_{i=1}^n (\hat{\pi}(\mathbf{x}^{(i)}) - y^{(i)})^2$$

- Fancy name for MSE on probabilities
- Usual definition for binary case,  $y^{(i)}$  must be coded as 0 and 1.



# PROBABILITIES: BRIER SCORE

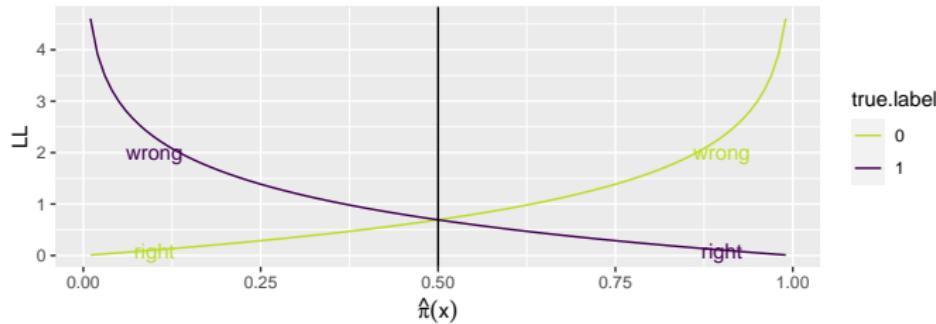
$$BS2 = \frac{1}{n} \sum_{i=1}^n \sum_{k=1}^g \left( \hat{\pi}_k(\mathbf{x}^{(i)}) - o_k^{(i)} \right)^2$$

- Original by Brier, works also for multiple classes
- $o_k^{(i)} = [y^{(i)} = k]$  is a 0-1-one-hot coding for labels
- For the binary case, BS2 is twice as large as BS1, because in BS2 we sum the squared difference for each observation regarding class 0 **and** class 1, not only the true class.

# PROBABILITIES: LOG-LOSS

Logistic regression loss function, a.k.a. Bernoulli or binomial loss,  $y^{(i)}$  coded as 0 and 1.

$$LL = \frac{1}{n} \sum_{i=1}^n \left( -y^{(i)} \log(\hat{\pi}(\mathbf{x}^{(i)})) - (1 - y^{(i)}) \log(1 - \hat{\pi}(\mathbf{x}^{(i)})) \right)$$



- Optimal value is 0, “confidently wrong” is penalized heavily
- Multiclass version:  $LL = -\frac{1}{n} \sum_{i=1}^n \sum_{k=1}^g o_k^{(i)} \log(\hat{\pi}_k(\mathbf{x}^{(i)}))$

# Introduction to Machine Learning

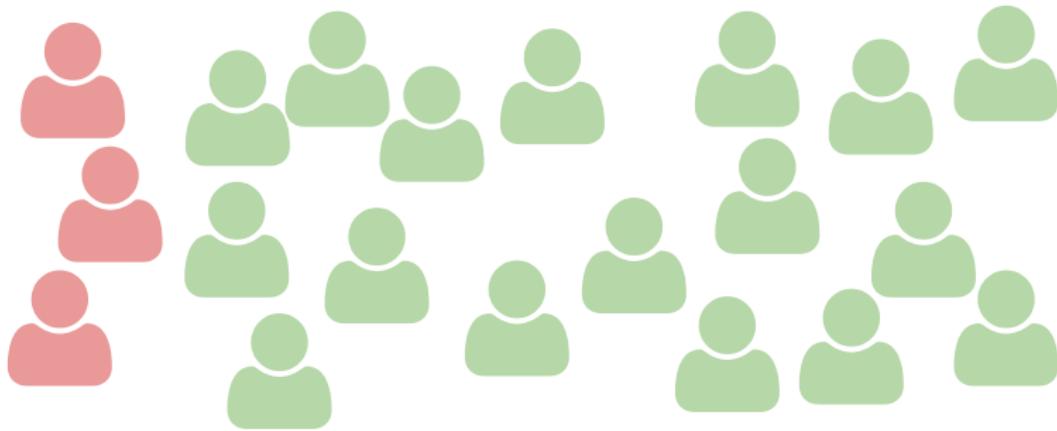
## Evaluation: Measures for Binary Classification: ROC Measures

### Learning goals

- Understand why accuracy is not an optimal performance measure for imbalanced labels
- Understand the different measures computable from a confusion matrix
- Be aware that each of these measures has a variety of names

		True Class $y$		
		+	-	
Pred.	+	TP	FP	$PPV = \frac{TP}{TP+FP}$
	-	FN	TN	$NPV = \frac{TN}{FN+TN}$
		$TPR = \frac{TP}{TP+FN}$	$TNR = \frac{TN}{FP+TN}$	Accuracy = $\frac{TP+TN}{TOTAL}$

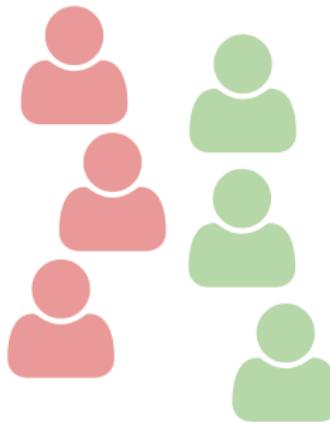
# IMBALANCED BINARY LABELS



Classify all as “no disease” (green) → high accuracy.

## Accuracy Paradox

# IMBALANCED COSTS



Classify incorrectly as “no disease” → very high cost

# CONFUSION MATRIX

		True Class $y$	
		+	-
Pred.	+	TP	FP
	-	FN	TN

- +: “positive” class
- -: “negative” class
- $n_+$ : number of observations in +
- $n_-$ : number of observations in -

# LABELS: ROC METRICS

From the confusion matrix (binary case), we can calculate "ROC" metrics.

		True Class $y$		
		+	-	
Pred.	+	TP	FP	$PPV = \frac{TP}{TP+FP}$
	-	FN	TN	$NPV = \frac{TN}{FN+TN}$
		$TPR = \frac{TP}{TP+FN}$	$TNR = \frac{TN}{FP+TN}$	$Accuracy = \frac{TP+TN}{TOTAL}$

- True Positive Rate: How many of the true 1s did we predict as 1?
- True Negative Rate: How many of the true 0s did we predict as 0?
- Positive Predictive Value: If we predict 1 how likely is it a true 1?
- Negative Predictive Value: If we predict 0 how likely is it a true 0?

# HISTORY ROC

ROC = receiver operating characteristics

Initially developed by electrical engineers and radar engineers during World War II for detecting enemy objects in battlefields.



[http:](http://media.iwm.org.uk/iwm/mediaLib//39/media-39665/large.jpg)

//media.iwm.org.uk/iwm/mediaLib//39/media-39665/large.jpg

Still has the funny name.

# LABELS: ROC

Example

		Actual Class $y$		
		Positive	Negative	
$\hat{y}$ Pred.	Positive	<b>True Positive</b> (TP) = 20	<b>False Positive</b> (FP) = 180	Positive predictive value $= TP / (TP + FP)$ $= 20 / (20 + 180)$ $= 10\%$
	Negative	<b>False Negative</b> (FN) = 10	<b>True Negative</b> (TN) = 1820	Negative predictive value $= TN / (FN + TN)$ $= 1820 / (10 + 1820)$ $\approx 99.5\%$
		True Positive Rate $= TP / (TP + FN)$ $= 20 / (20 + 10)$ $\approx 67\%$	True Negative Rate $= TN / (FP + TN)$ $= 1820 / (180 + 1820)$ $= 91\%$	

# MORE METRICS AND ALTERNATIVE TERMINOLOGY

Unfortunately, for many concepts in ROC, 2-3 different terms exist.

		True condition				
		Total population	Condition positive	Condition negative	Prevalence $= \frac{\sum \text{Condition positive}}{\sum \text{Total population}}$	Accuracy (ACC) = $\frac{\sum \text{True positive} + \sum \text{True negative}}{\sum \text{Total population}}$
Predicted condition	Predicted condition positive	True positive, Power	False positive, Type I error	Positive predictive value (PPV), Precision = $\frac{\sum \text{True positive}}{\sum \text{Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\sum \text{False positive}}{\sum \text{Predicted condition positive}}$	
	Predicted condition negative	False negative, Type II error	True negative	False omission rate (FOR) = $\frac{\sum \text{False negative}}{\sum \text{Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\sum \text{True negative}}{\sum \text{Predicted condition negative}}$	
	True positive rate (TPR), Recall, Sensitivity, probability of detection $= \frac{\sum \text{True positive}}{\sum \text{Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm $= \frac{\sum \text{False positive}}{\sum \text{Condition negative}}$	Positive likelihood ratio (LR+) $= \frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) $= \frac{\text{LR}^+}{\text{LR}^-}$	$F_1$ score = $\frac{1}{\frac{1}{\text{Recall}} + \frac{1}{\text{Precision}}} = \frac{2 \cdot \text{Recall} \cdot \text{Precision}}{\text{Recall} + \text{Precision}}$	
	False negative rate (FNR), Miss rate $= \frac{\sum \text{False negative}}{\sum \text{Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) $= \frac{\sum \text{True negative}}{\sum \text{Condition negative}}$				

► Clickable version/picture source

► Interactive diagram

## LABELS: $F_1$ -MEASURE

A measure that balances two conflicting goals

- ① Maximising Positive Predictive Value
- ② Maximising True Positive Rate

is the harmonic mean of PPV and TPR:

$$F_1 = 2 \frac{PPV \cdot TPR}{PPV + TPR}$$

Note: still doesn't account for the number of true negatives.

## LABELS: $F_1$ -MEASURE

Tabulated  $F_1$ -Score for different TPR (rows) and PPV (cols) combinations.

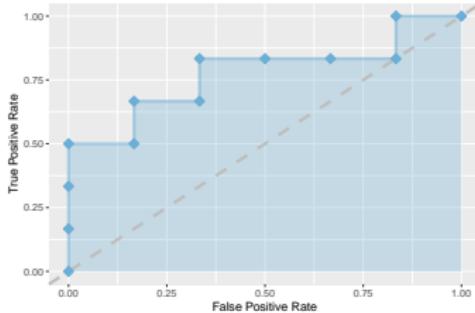
	0.0	0.2	0.4	0.6	0.8	1.0
0.0	0	0.00	0.00	0.00	0.00	0.00
0.2	0	0.20	0.27	0.30	0.32	0.33
0.4	0	0.27	0.40	0.48	0.53	0.57
0.6	0	0.30	0.48	0.60	0.69	0.75
0.8	0	0.32	0.53	0.69	0.80	0.89
1.0	0	0.33	0.57	0.75	0.89	1.00

→ Tends more towards the lower of the 2 combined values.

- $TPR = 0$  or  $PPV = 0 \Rightarrow F_1$  of 0
- Predicting always "neg":  $F_1 = 0$
- Predicting always "pos":  $F_1 = 2PPV/(PPV + 1) = 2n_+/(n_+ + n)$ , which will be rather small, if the size of the positive class  $n_+$  is small.

# Introduction to Machine Learning

## Evaluation: Measures for Binary Classification: ROC Visualization

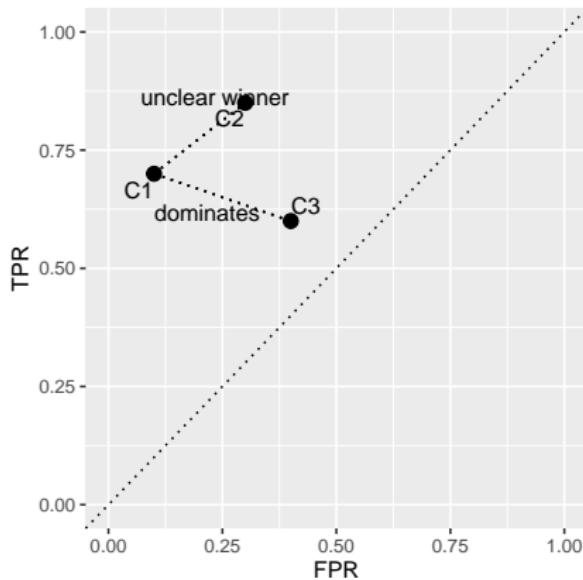


### Learning goals

- Understand the ROC curve
- Be able to compute a ROC curve manually
- Understand the definition of AUC and what a certain value of AUC means (and what not!)

# LABELS: ROC SPACE

Plot True Positive Rate and False Positive Rate:



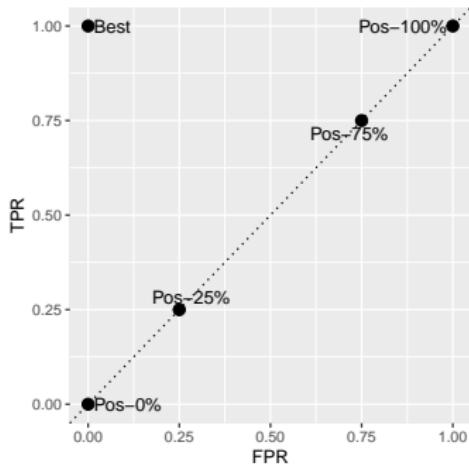
		True Class $y$	
		+	-
Pred.	+	TP	FP
	-	FN	TN

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

# LABELS: ROC SPACE

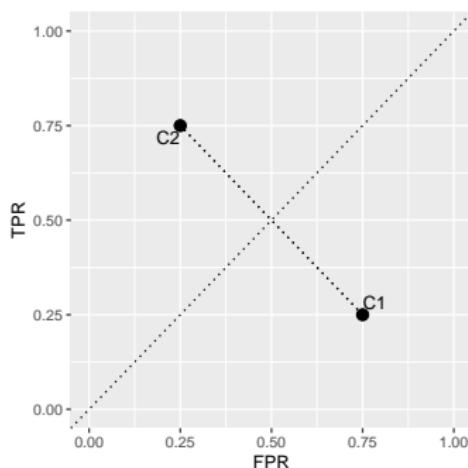
- The best classifier lies on the top-left corner
- The diagonal  $\approx$  random labels (with different proportions).  
Assign positive  $x$  as "pos" with 25% probability  $\rightarrow TPR = 0.25$ .  
Assign negative  $x$  as "pos" with 25% probability  $\rightarrow FPR = 0.25$ .



# LABELS: ROC SPACE

In practice, we should never obtain a classifier below the diagonal.

Inverting the predicted labels ( $0 \rightarrow 1$  and  $1 \rightarrow 0$ ) will result in a reflection at the diagonal.



# LABEL DISTRIBUTION IN TPR AND FPR

TPR and FPR are insensitive to the class distribution:  
Not affected by changes in the ratio  $n_+/n_-$  (at prediction).

Example 1:

Proportion  $n_+/n_- = 1$

	Actual Positive	Actual Negative
Pred. Positive	40	25
Pred. Negative	10	25

$$MCE = 35/100$$

$$TPR = 0.8$$

$$FPR = 0.5$$

Example 2:

Proportion  $n_+/n_- = 2$

	Actual Positive	Actual Negative
Pred. Positive	80	25
Pred. Negative	20	25

$$MCE = 45/150 = 30/100$$

$$TPR = 0.8$$

$$FPR = 0.5$$

Note: If class proportions differ during training, the above is not true.  
Estimated posterior probabilities can change!

# FROM PROBABILITIES TO LABELS: ROC CURVE

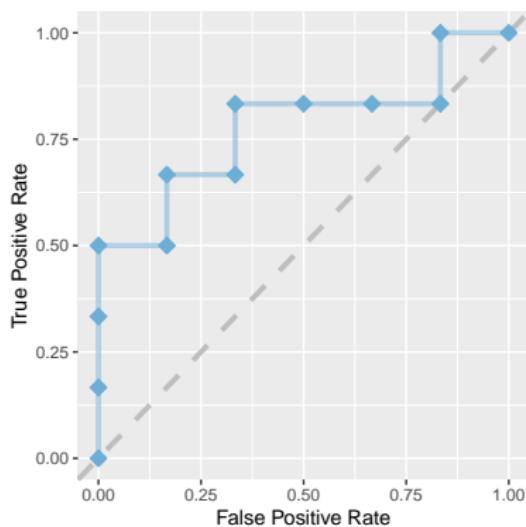
Remember: Both probabilistic and scoring classifiers can output classes by thresholding.

$$h(\mathbf{x}) := [\pi(\mathbf{x}) \geq c] \quad \text{or} \quad h(\mathbf{x}) = [f(\mathbf{x}) \geq c]$$

**To draw a ROC curve:**

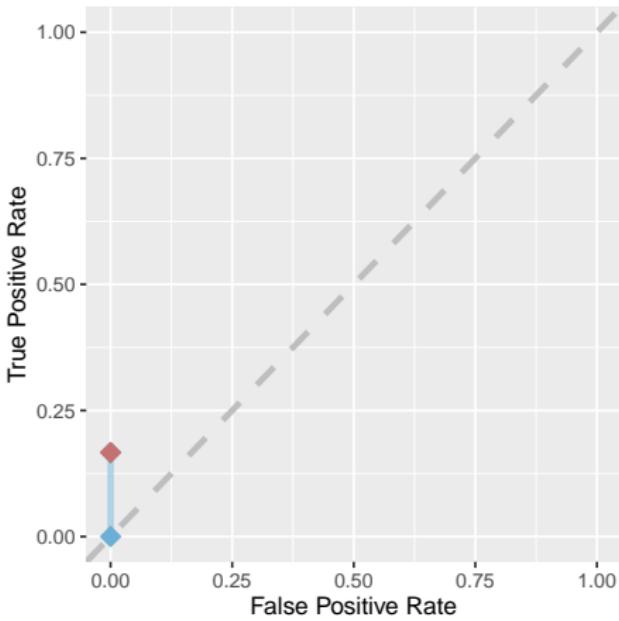
Iterate through all possible thresholds  $c$

→ Visual inspection of all possible thresholds / results



# ROC CURVE

#	Truth	Score
1	Pos	0.95
2	Pos	0.86
3	Pos	0.69
4	Neg	0.65
5	Pos	0.59
6	Neg	0.52
7	Pos	0.51
8	Neg	0.39
9	Neg	0.28
10	Neg	0.18
11	Pos	0.15
12	Neg	0.06



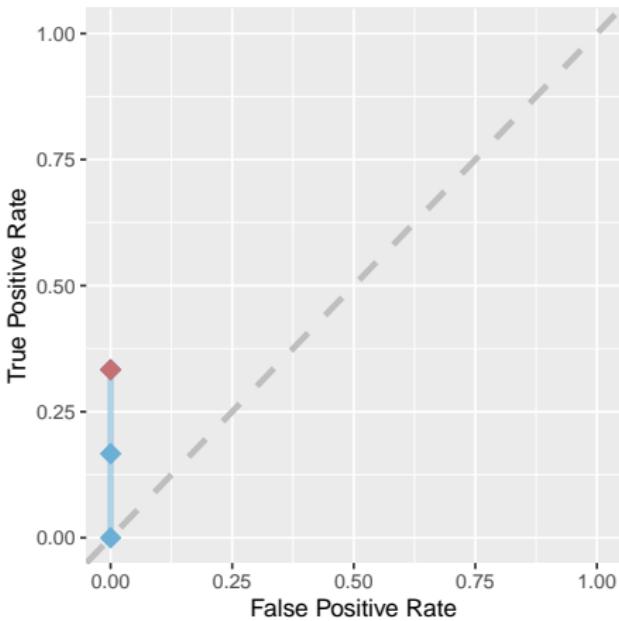
$$c = 0.9$$

$$\rightarrow \text{TPR} = 0.167$$

$$\rightarrow \text{FPR} = 0$$

# ROC CURVE

#	Truth	Score
1	Pos	0.95
2	Pos	0.86
3	Pos	0.69
4	Neg	0.65
5	Pos	0.59
6	Neg	0.52
7	Pos	0.51
8	Neg	0.39
9	Neg	0.28
10	Neg	0.18
11	Pos	0.15
12	Neg	0.06



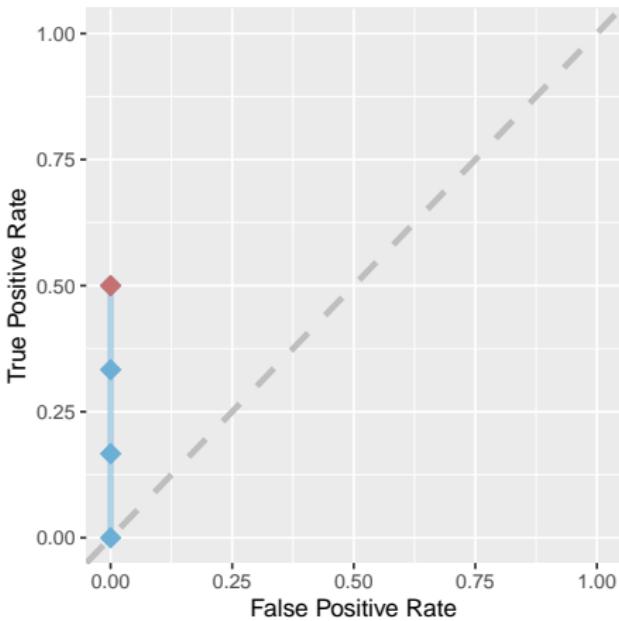
$$c = 0.85$$

$$\rightarrow \text{TPR} = 0.333$$

$$\rightarrow \text{FPR} = 0$$

# ROC CURVE

#	Truth	Score
1	Pos	0.95
2	Pos	0.86
3	Pos	0.69
4	Neg	0.65
5	Pos	0.59
6	Neg	0.52
7	Pos	0.51
8	Neg	0.39
9	Neg	0.28
10	Neg	0.18
11	Pos	0.15
12	Neg	0.06



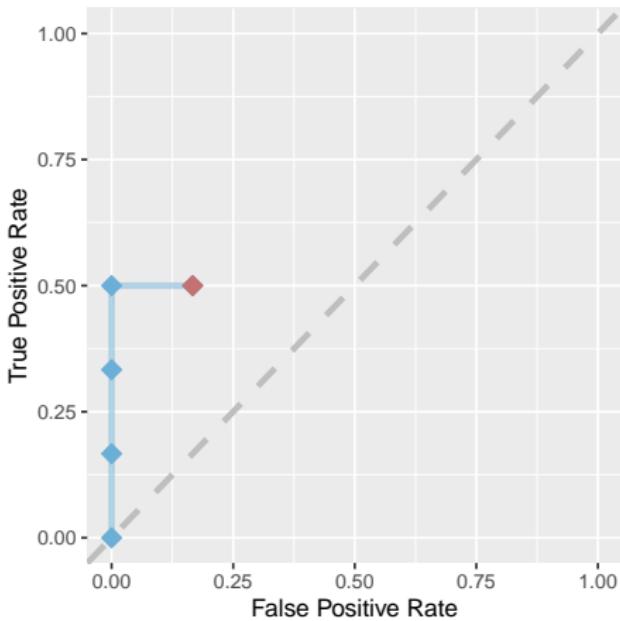
$$c = 0.66$$

$$\rightarrow \text{TPR} = 0.5$$

$$\rightarrow \text{FPR} = 0$$

# ROC CURVE

#	Truth	Score
1	Pos	0.95
2	Pos	0.86
3	Pos	0.69
4	Neg	0.65
5	Pos	0.59
6	Neg	0.52
7	Pos	0.51
8	Neg	0.39
9	Neg	0.28
10	Neg	0.18
11	Pos	0.15
12	Neg	0.06



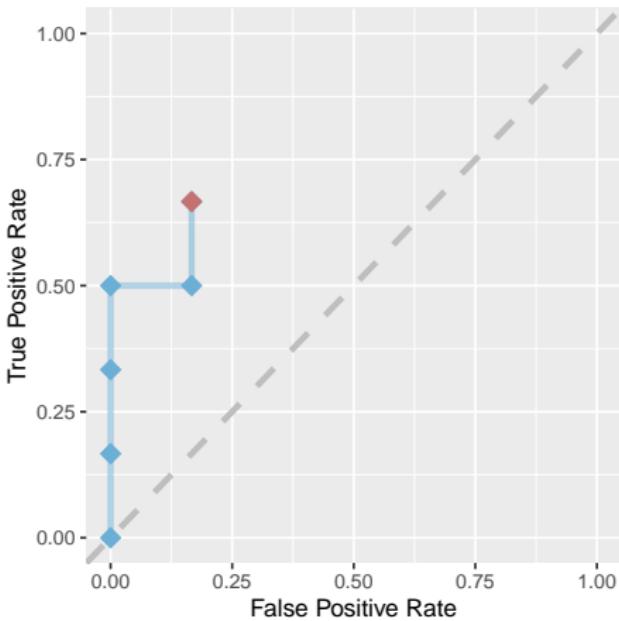
$$c = 0.6$$

$$\rightarrow \text{TPR} = 0.5$$

$$\rightarrow \text{FPR} = 0.167$$

# ROC CURVE

#	Truth	Score
1	Pos	0.95
2	Pos	0.86
3	Pos	0.69
4	Neg	0.65
5	Pos	0.59
6	Neg	0.52
7	Pos	0.51
8	Neg	0.39
9	Neg	0.28
10	Neg	0.18
11	Pos	0.15
12	Neg	0.06



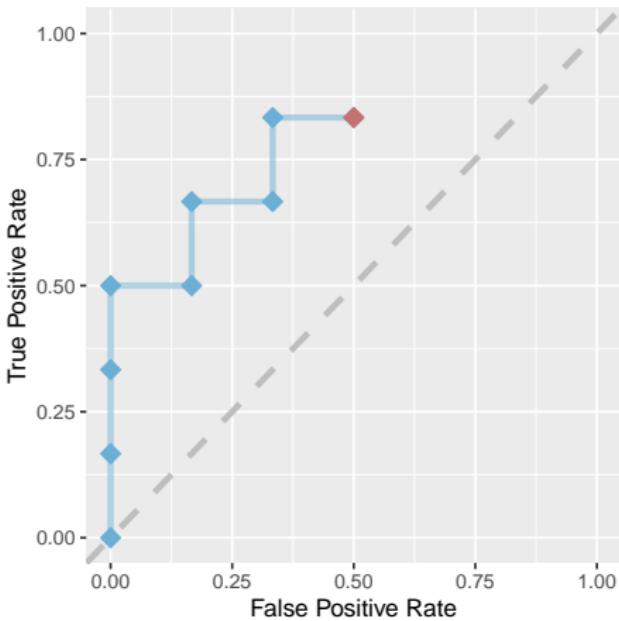
$$c = 0.55$$

$$\rightarrow \text{TPR} = 0.667$$

$$\rightarrow \text{FPR} = 0.167$$

# ROC CURVE

#	Truth	Score
1	Pos	0.95
2	Pos	0.86
3	Pos	0.69
4	Neg	0.65
5	Pos	0.59
6	Neg	0.52
7	Pos	0.51
8	Neg	0.39
9	Neg	0.28
10	Neg	0.18
11	Pos	0.15
12	Neg	0.06



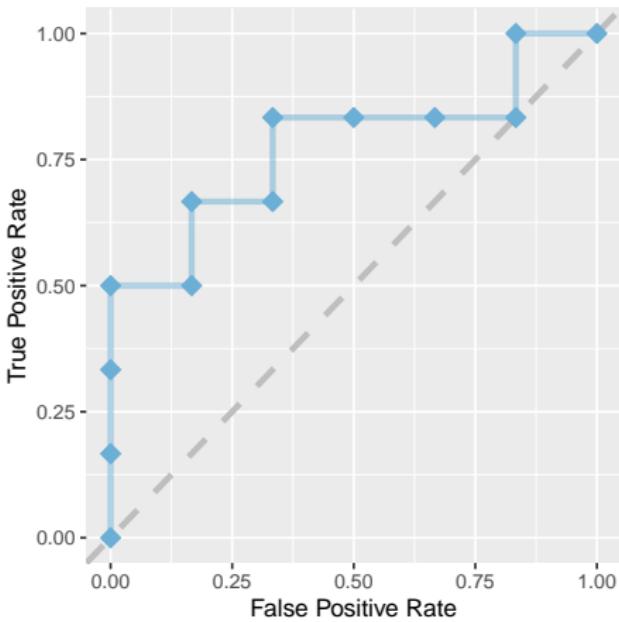
$$c = 0.3$$

$$\rightarrow \text{TPR} = 0.833$$

$$\rightarrow \text{FPR} = 0.5$$

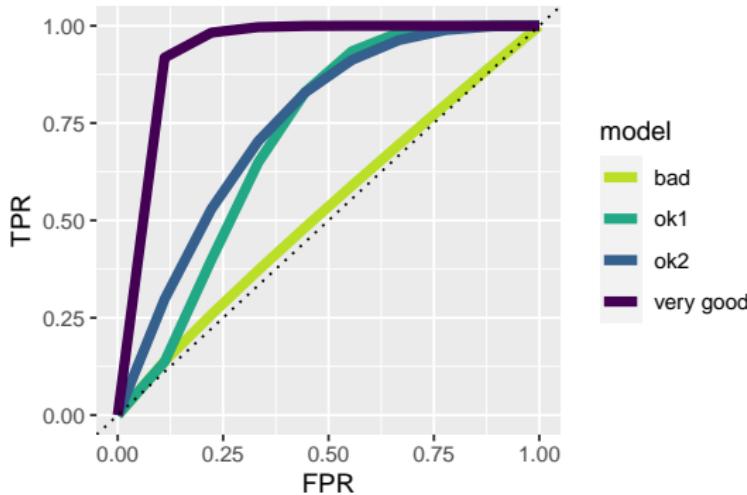
# ROC CURVE

#	Truth	Score
1	Pos	0.95
2	Pos	0.86
3	Pos	0.69
4	Neg	0.65
5	Pos	0.59
6	Neg	0.52
7	Pos	0.51
8	Neg	0.39
9	Neg	0.28
10	Neg	0.18
11	Pos	0.15
12	Neg	0.06



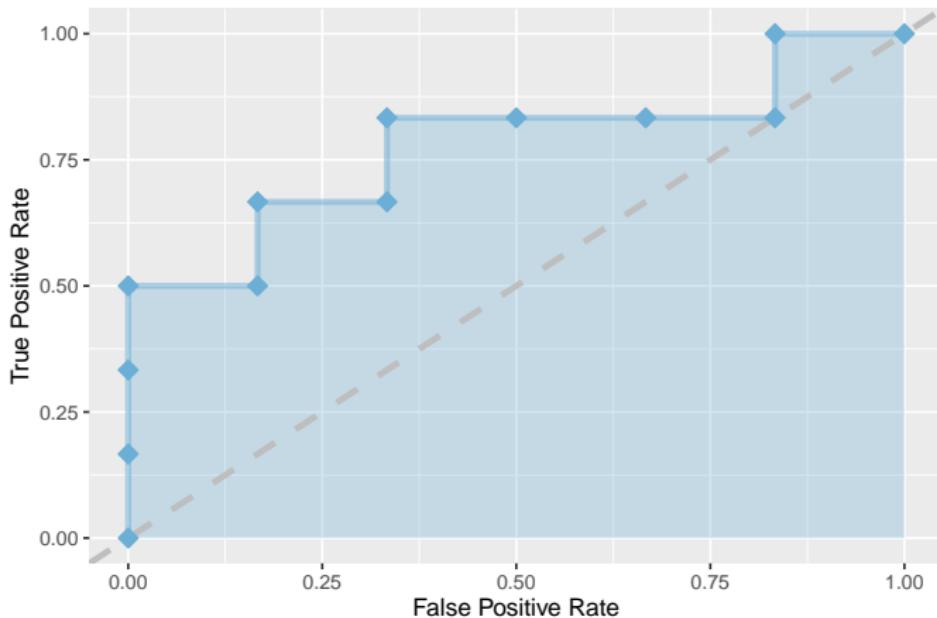
# ROC CURVE

- The closer the curve to the top-left corner, the better
- If ROC curves cross, a different model can be better in different parts of the ROC space



# AUC: AREA UNDER ROC CURVE

- The AUC (in  $[0,1]$ ) is a single metric to evaluate scoring classifiers
- AUC = 1: Perfect classifier
- AUC = 0.5: Randomly ordered



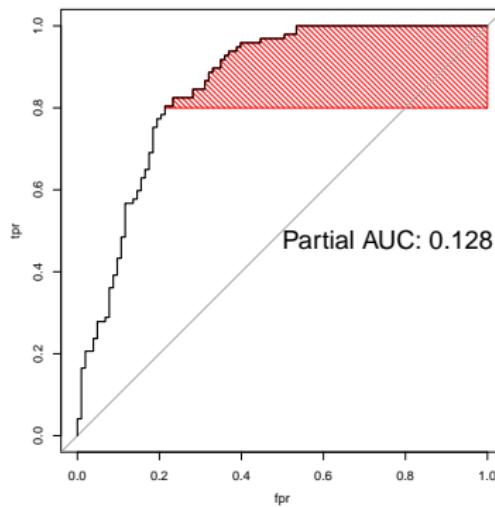
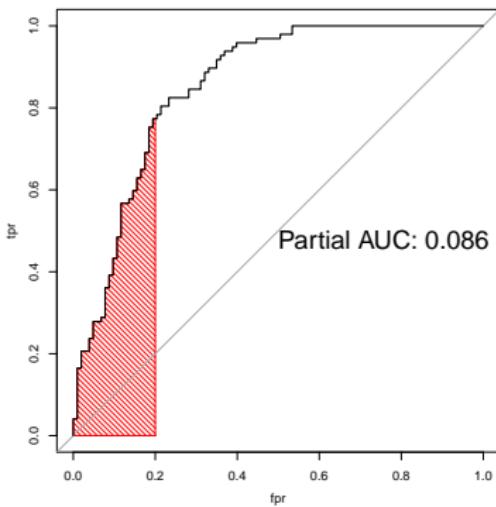
# AUC: AREA UNDER ROC CURVE

Interpretation: Probability that classifier ranks a random positive higher than a random negative observation



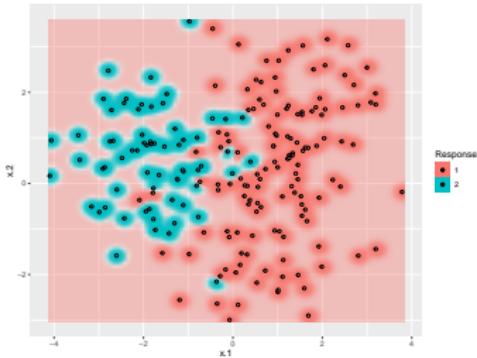
# PARTIAL AUC

- Sometimes it can be useful to look at a specific region under the ROC curve  $\Rightarrow$  partial AUC (pAUC).
- Examples: focus on a region with low FPR or a region with high TPR:



# Introduction to Machine Learning

## Evaluation: Overfitting

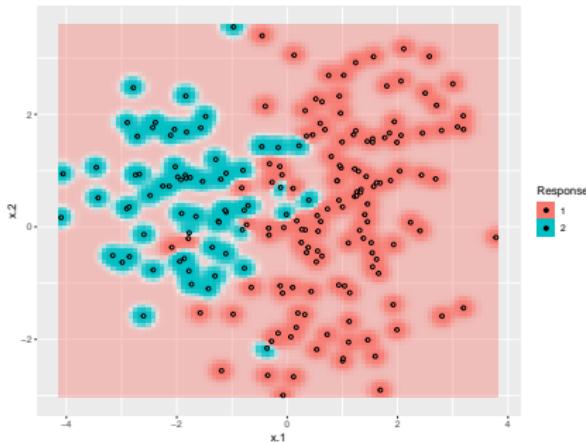


### Learning goals

- Understand what overfitting is and why it is a problem
- Understand how to avoid overfitting

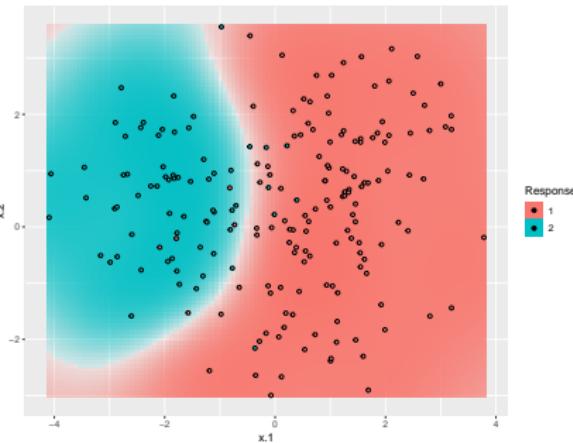
# OVERFITTING

Overfitting learner



Better training set performance  
(seen examples)

Non-overfitting learner

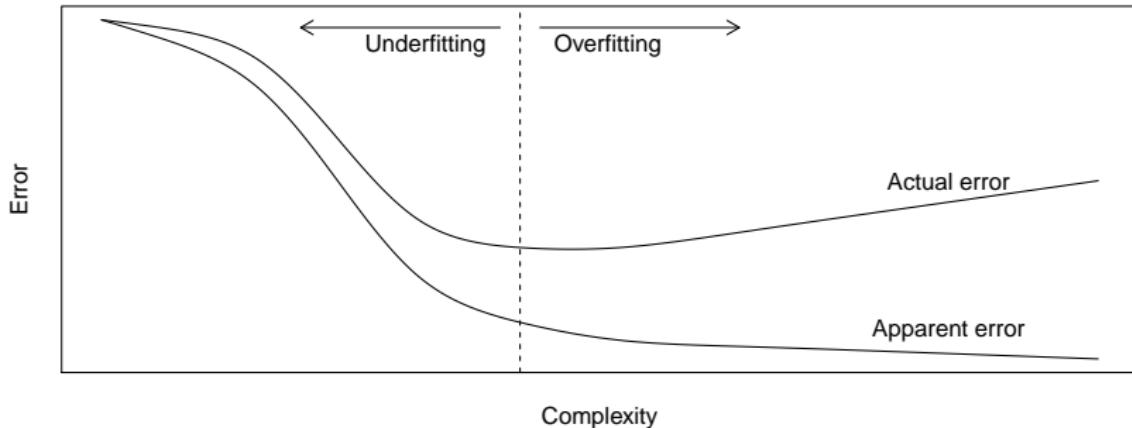


Better test set performance  
(unseen examples)

# OVERFITTING

- Happens when algorithm models patterns beyond the data-generating process, e.g., noise or artefacts in the training data
- Reason: too many hypotheses and not enough data to tell them apart
- Less in bigger data sets
- If hypothesis space is not constrained, there may never be enough data
- Many learners have a parameter that allows constraining (*regularization*)
- Check for overfitting by validating on a new unseen test data set

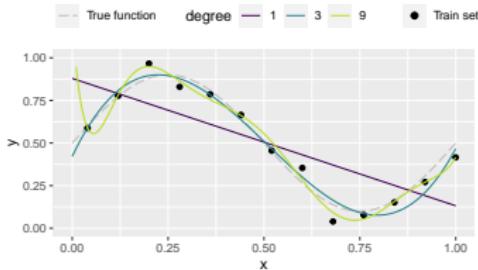
# TRADE-OFF BETWEEN GENERALIZATION ERROR AND COMPLEXITY



⇒ Optimization regarding model complexity is desirable:  
Find the right amount of complexity for the given amount of data where generalization error becomes minimal.

# Introduction to Machine Learning

## Evaluation: Training Error

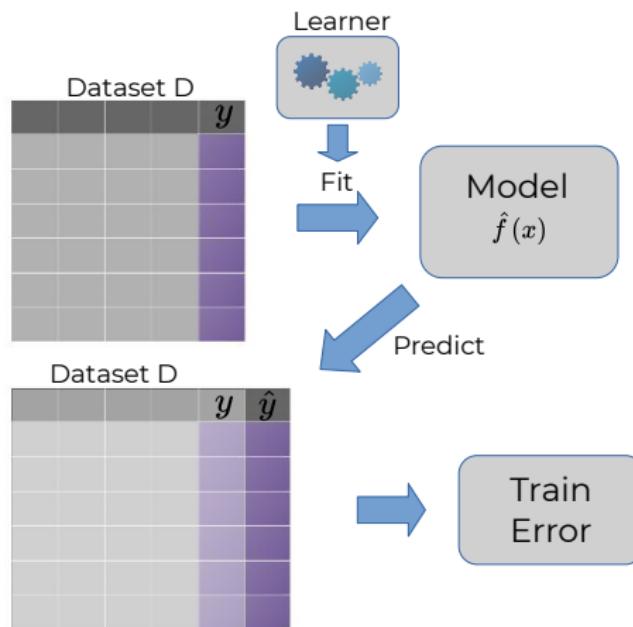


### Learning goals

- Understand the definition of training error
- Understand why training error is no reliable estimator of future performance

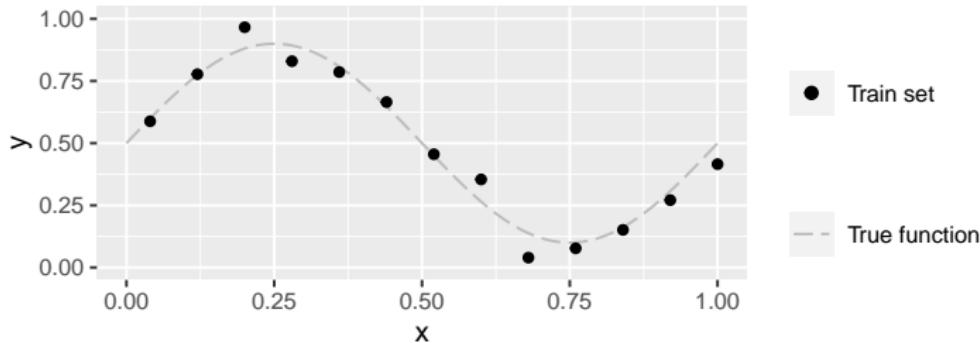
# TRAINING ERROR

(also: apparent error / resubstitution error)



# EXAMPLE: POLYNOMIAL REGRESSION

Sample data from sinusoidal function  $0.5 + 0.4 \cdot \sin(2\pi x) + \epsilon$  with measurement error  $\epsilon$ .



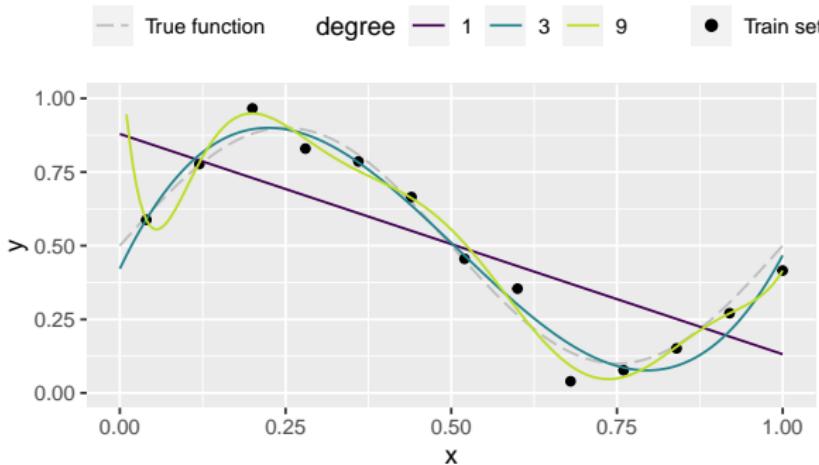
Assume data-generating process unknown.

Try to approximate with a  $d^{th}$ -degree polynomial:

$$f(\mathbf{x} | \boldsymbol{\theta}) = \theta_0 + \theta_1 x + \cdots + \theta_d x^d = \sum_{j=0}^d \theta_j x^j.$$

# EXAMPLE: POLYNOMIAL REGRESSION

Models of different *complexity*, i.e., of different orders of the polynomial are fitted. How should we choose  $d$ ?



- $d=1$ : MSE = 0.036: Clear underfitting
- $d=3$ : MSE = 0.003: Pretty OK?
- $d=9$ : MSE = 0.001: Clear overfitting

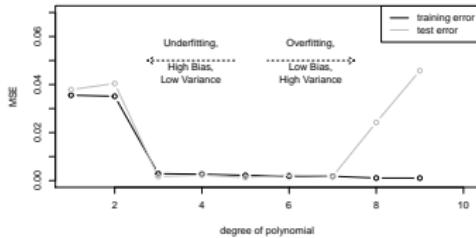
Simply using the training error seems to be a bad idea.

# TRAINING ERROR PROBLEMS

- Unreliable and overly optimistic estimator of future performance.  
E.g., training error of 1-NN is always zero as each observation is its own NN during test time.
- Goodness-of-fit measures like (classic)  $R^2$ , likelihood, AIC, BIC, deviance are all based on the training error.
- For models of restricted capacity, and given enough data, the training error may provide reliable information.  
E.g., LM with  $p = 5$  features,  $10^6$  training points.  
But: impossible to determine when training error becomes unreliable.

# Introduction to Machine Learning

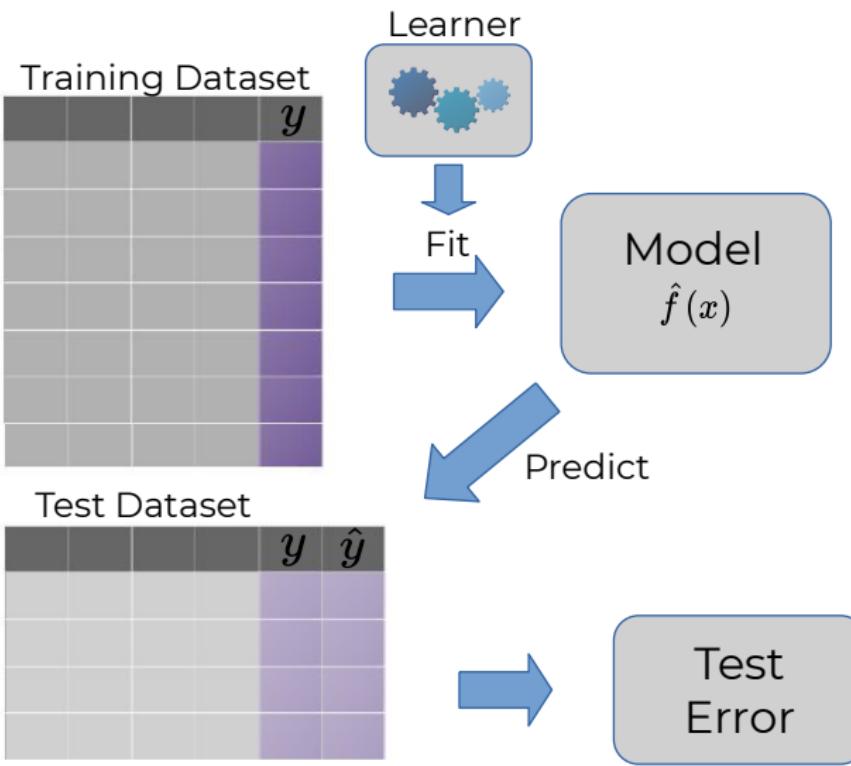
## Evaluation: Test Error



### Learning goals

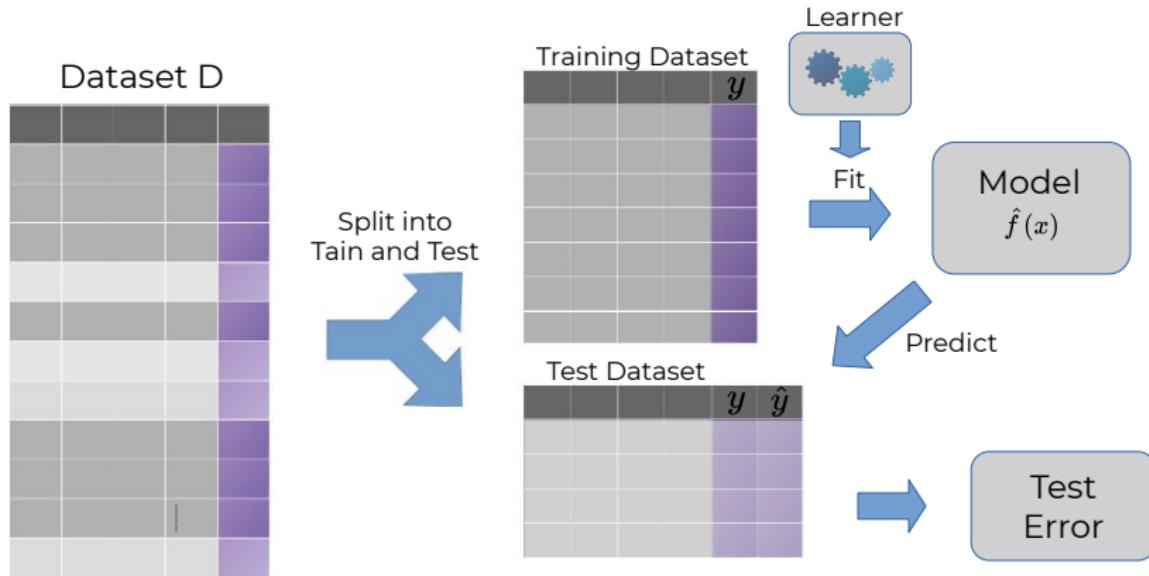
- Understand the definition of test error
- Understand how overfitting can be seen in the test error

# TEST ERROR



# TEST ERROR AND HOLD-OUT SPLITTING

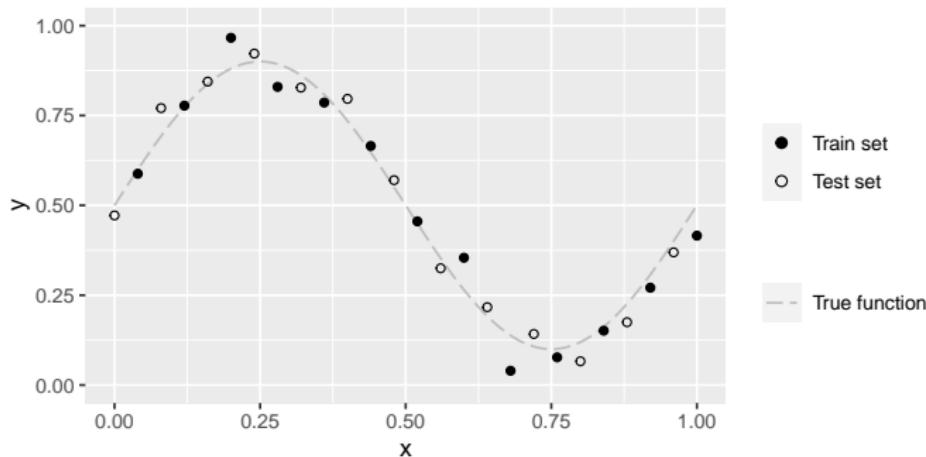
- Split data into 2 parts, e.g., 2/3 for training, 1/3 for testing
- Evaluate on data not used for model building



# TEST ERROR

Let's consider the following example:

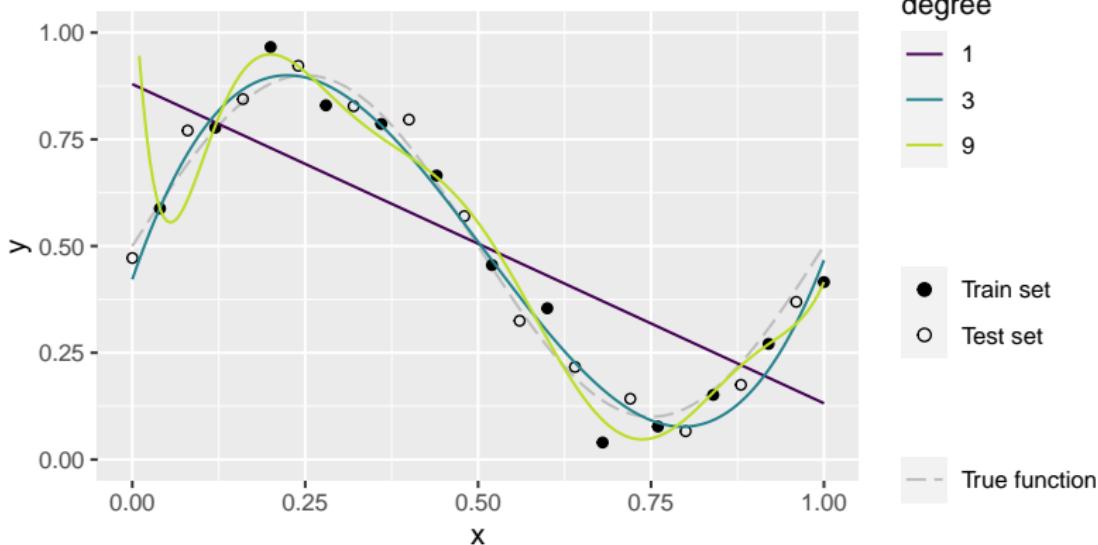
Sample data from sinusoidal function  $0.5 + 0.4 \cdot \sin(2\pi x) + \epsilon$



Try to approximate with a  $d^{th}$ -degree polynomial:

$$f(\mathbf{x} | \boldsymbol{\theta}) = \theta_0 + \theta_1 x + \cdots + \theta_d x^d = \sum_{i=0}^d \theta_i x^i.$$

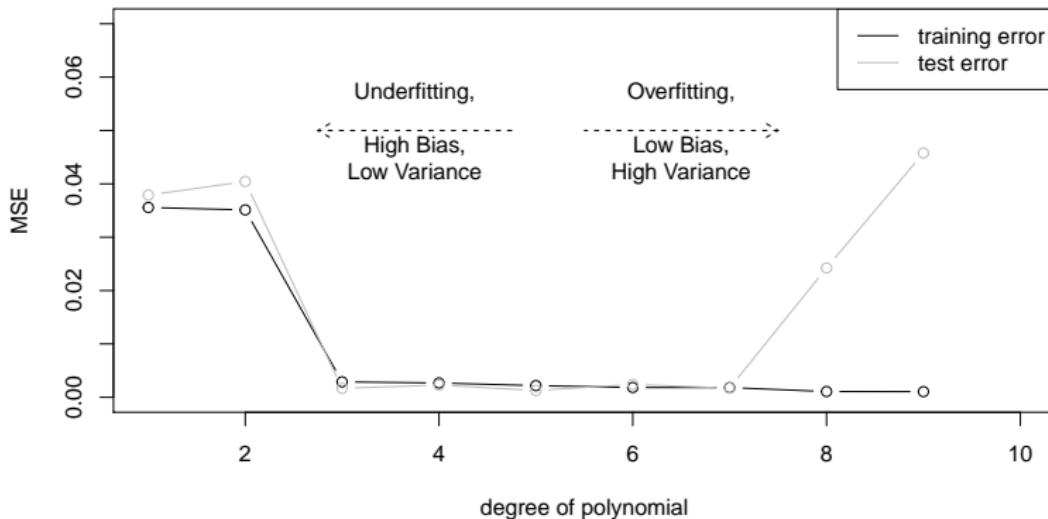
# TEST ERROR



- $d=1$ : MSE = 0.038: Clear underfitting
- $d=3$ : MSE = 0.002: Pretty OK
- $d=9$ : MSE = 0.046: Clear overfitting

# TEST ERROR

Plot evaluation measure for all polynomial degrees:



Increase model complexity (tendenitally)

- decrease in training error
- U-shape in test error  
(first underfit, then overfit, sweet-spot in the middle)

# TEST ERROR PROBLEMS

- Test data has to be i.i.d. compared to training data.
- Bias-variance of hold-out:
  - The smaller the training set, the worse the model → biased estimate.
  - The smaller the test set, the higher the variance of the estimate.
- If the size of our initial, complete data set  $\mathcal{D}$  is limited, single train-test splits can be problematic.

# TEST ERROR PROBLEMS

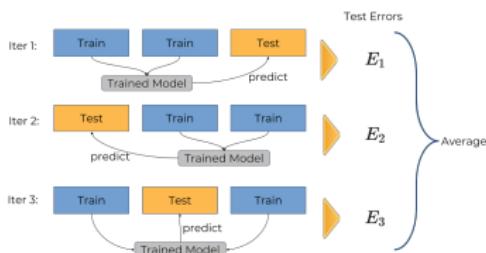
A major point of confusion:

- In ML we are in a weird situation. We are usually given one data set. At the end of our model selection and evaluation process we will likely fit one model on exactly that complete data set. As training error evaluation does not work, we have nothing left to evaluate exactly that model.
- Hold-out splitting (and resampling) are tools to estimate the future performance. All of the models produced during that phase of evaluation are intermediate results.

# Introduction to Machine Learning

## Evaluation: Resampling

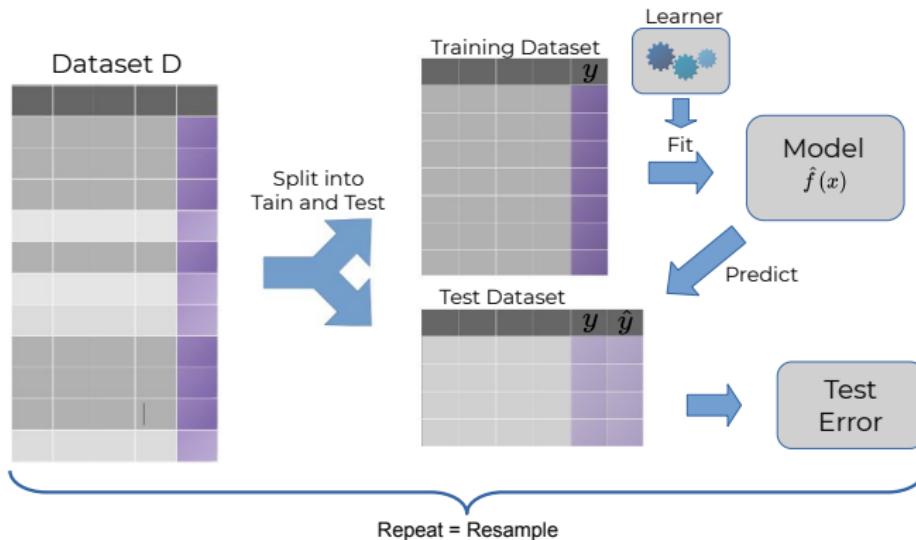
### Learning goals



- Understand how resampling techniques extend the idea of simple train-test splits
- Understand the ideas of cross-validation, bootstrap and subsampling
- Understand what pessimistic bias means

# RESAMPLING

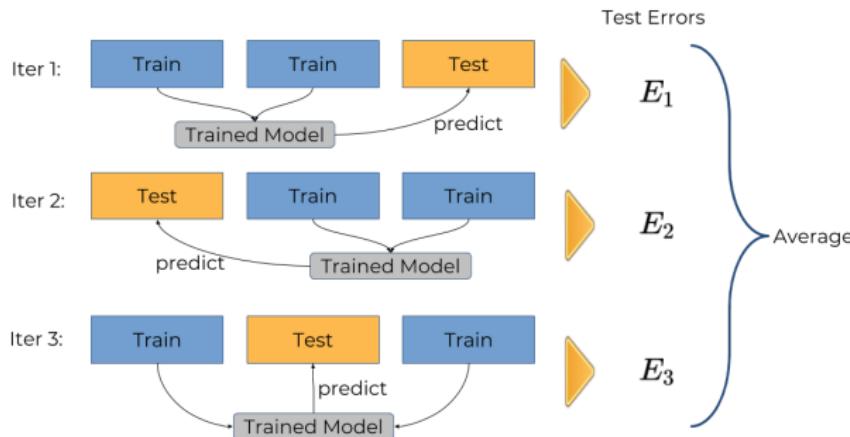
- Aim: Assess performance of learning algorithm.
- Make training sets large (to keep the pessimistic bias small), and reduce variance introduced by smaller test sets through many repetitions / averaging of results.



# CROSS-VALIDATION

- Split the data into  $k$  roughly equally-sized partitions.
- Use each part once as test set and join the  $k - 1$  others for training
- Obtain  $k$  test errors and average.

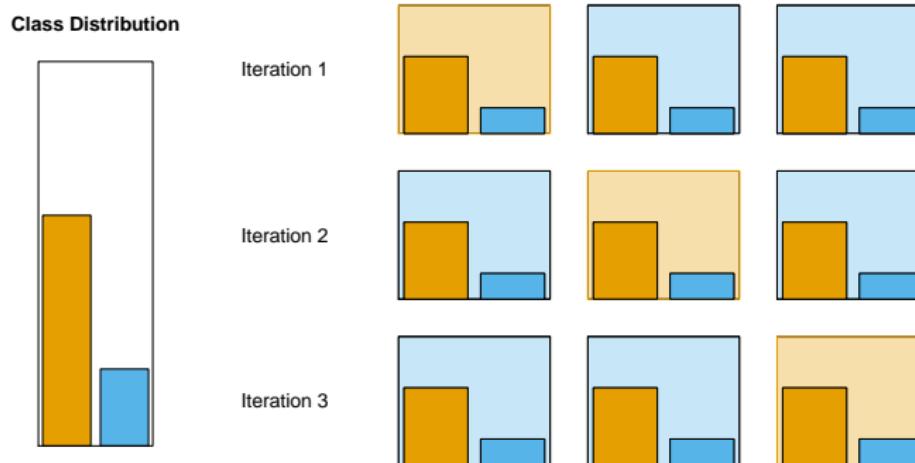
Example: 3-fold cross-validation:



# CROSS-VALIDATION - STRATIFICATION

Stratification tries to preserve the distribution of the target class (or any specific categorical feature of interest) in each fold.

Example of stratified 3-fold cross-validation:

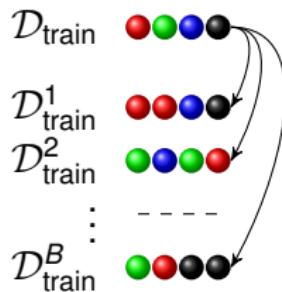


# CROSS-VALIDATION

- 5 or 10 folds are common
- $k = n$  is known as leave-one-out (LOO) cross-validation
- Estimates of the generalization error tend to be pessimistically biased  
size of the training sets is  $n - (n/k) < n$   
bias increases as  $k$  gets smaller.
- The  $k$  performance estimates are dependent because of the structured overlap of the training sets.  
⇒ Variance of the estimator increases for very large  $k$  (close to LOO), when training sets nearly completely overlap.
- Repeated  $k$ -fold CV (multiple random partitions) can improve error estimation for small sample sizes.

# BOOTSTRAP

The basic idea is to randomly draw  $B$  training sets of size  $n$  with replacement from the original training set  $\mathcal{D}_{\text{train}}$ :



We define the test set in terms of out-of-bag observations

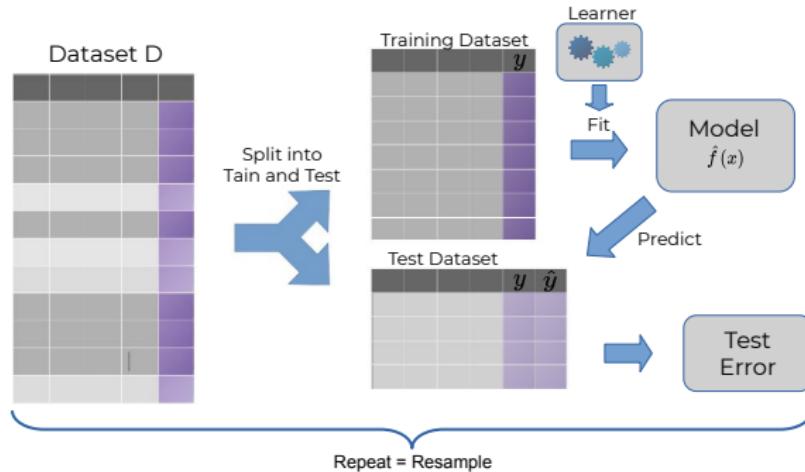
$$\mathcal{D}_{\text{test}}^b = \mathcal{D}_{\text{train}} \setminus \mathcal{D}_{\text{train}}^b.$$

# BOOTSTRAP

- Typically,  $B$  is between 30 and 200.
- The variance of the bootstrap estimator tends to be smaller than the variance of  $k$ -fold CV.
- The more iterations, the smaller the variance of the estimator.
- Tends to be pessimistically biased (because training sets contain only about 63.2% of the unique observations).
- Bootstrapping framework allows for inference (e.g. detect significant performance differences between learners).
- Extensions exist for very small data sets that also use the training error for estimation: B632 and B632+.

# SUBSAMPLING

- Repeated hold-out with averaging, a.k.a. Monte Carlo CV
- Similar to bootstrap, but draws without replacement
- Typical choices for splitting: 4/5 or 9/10 for training



# SUBSAMPLING

- The smaller the subsampling rate, the larger the pessimistic bias.
- The more subsampling repetitions, the smaller the variance.

# RESAMPLING DISCUSSION

In ML we fit, at the end, a model on all our given data.

**Problem:** We need to know how well this model performs in the future, but no data is left to reliably do this.

⇒ Approximate using hold-out / CV / bootstrap / resampling estimate

**But:** pessimistic bias because we don't use all data points

Final model is (usually) computed on all data points.

# RESAMPLING DISCUSSION

- 5CV or 10CV have become standard
- Do not use hold-out, CV with few iterations, or subsampling with a low subsampling rate for small samples, since this can cause the estimator to be extremely biased, with large variance.
- If  $n < 500$ , use repeated CV
- A  $\mathcal{D}$  with  $|\mathcal{D}| = 100.000$  can have small-sample properties if one class has few observations
- Research indicates that subsampling has better properties than bootstrapping. The repeated observations can cause problems in training.

# INTRODUCTION TO MACHINE LEARNING

ML Basics

Supervised Regression

Supervised Classification

k-NN

Performance Evaluation

## **Classification and Regression Trees (CART)**

Random Forests

Tuning

Nested Resampling

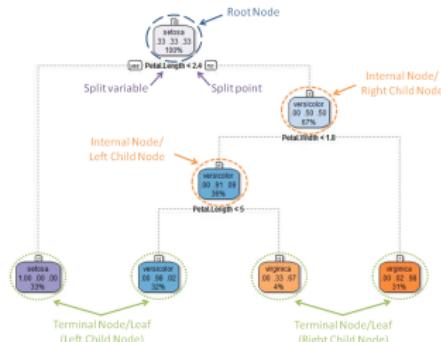
mlr3

# Introduction to Machine Learning

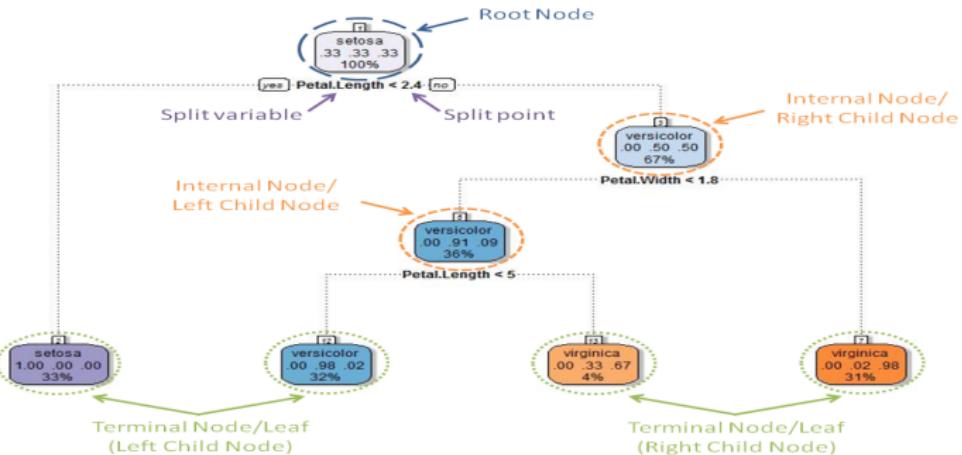
## Classification and Regression Trees (CART): Basics

### Learning goals

- Understand the basic structure of a tree model
- Understand that the basic idea of a tree model is the same for classification and regression
- Know how the label of a new observation is predicted via CART
- Know hypothesis space of CART



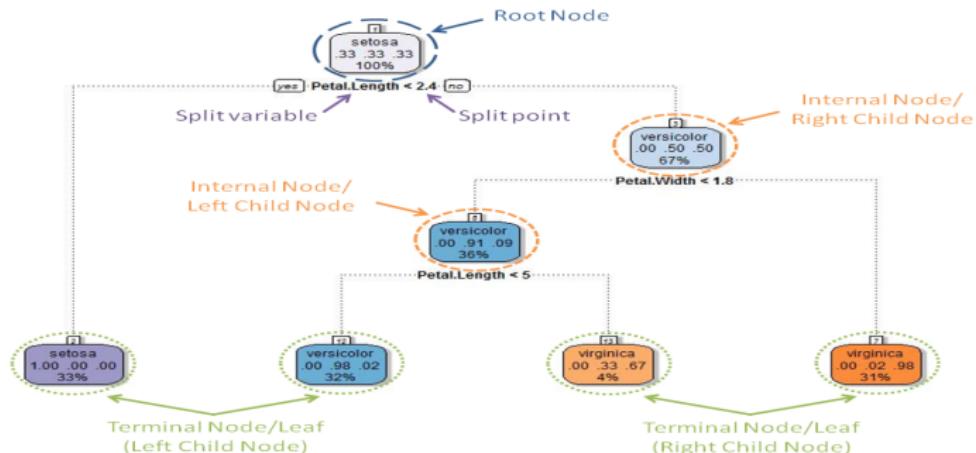
# TREE MODEL AND PREDICTION



- Classification and Regression Trees, introduced by Breiman
- Binary splits are constructed top-down
- Constant prediction in each terminal node (leaf): either a numerical value, a class label, or a probability vector over class labels.

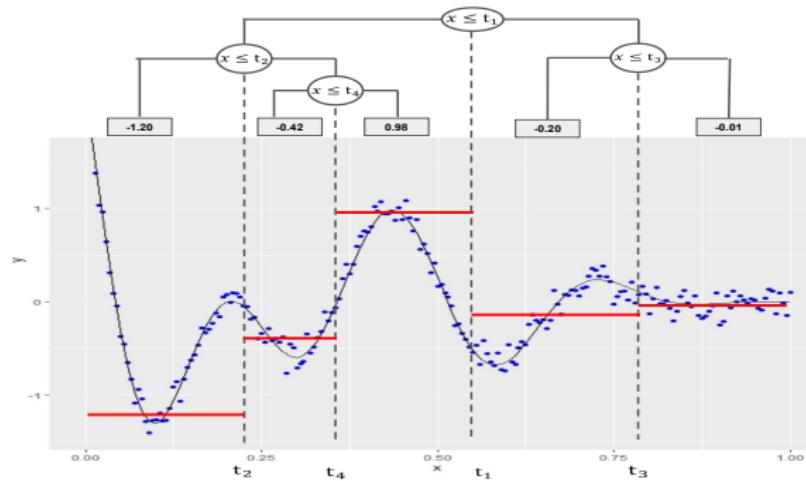
# TREE MODEL AND PREDICTION

- For predictions, observations are passed down the tree, according to the splitting rules in each node
- An observation will end up in exactly one leaf node
- All observations in a leaf node are assigned the same prediction for the target



# TREE MODEL AND PREDICTION

- For predictions, observations are passed down the tree, according to the splitting rules in each node
- An observation will end up in exactly one leaf node
- All observations in a leaf node are assigned the same prediction for the target



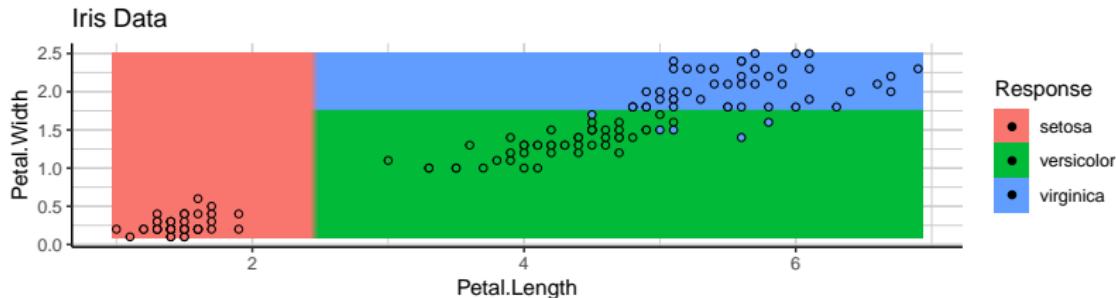
# TREE AS AN ADDITIVE MODEL

Each point in  $\mathcal{X}$  is assigned to exactly one leaf, and each leaf has a set of input points leading to it through axis-parallel splits.  
Hence, trees divide the feature space  $\mathcal{X}$  into **rectangular regions**:

$$f(\mathbf{x}) = \sum_{m=1}^M c_m \mathbb{I}(\mathbf{x} \in Q_m),$$

where a tree with  $M$  leaf nodes defines  $M$  “rectangles”  $Q_m$ .

$c_m$  is the predicted numerical response, class label or class distribution in the respective leaf node.

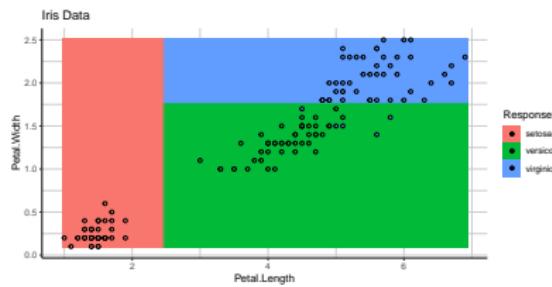


# TREES

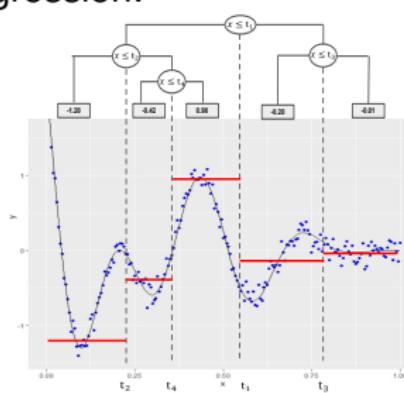
The hypothesis space of a CART is the set of all step functions over rectangular partitions of  $\mathcal{X}$ :

$$f(\mathbf{x}) = \sum_{m=1}^M c_m \mathbb{I}(\mathbf{x} \in Q_m)$$

Classification:



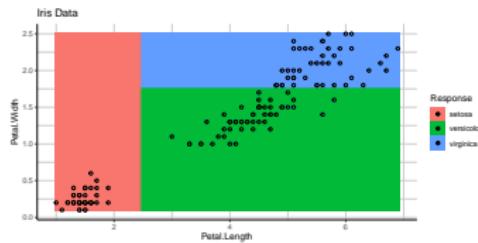
Regression:



# Introduction to Machine Learning

## CART: Splitting Criteria

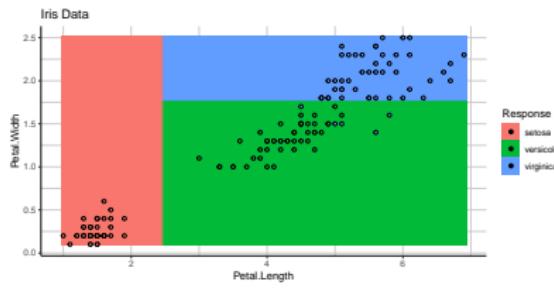
### Learning goals



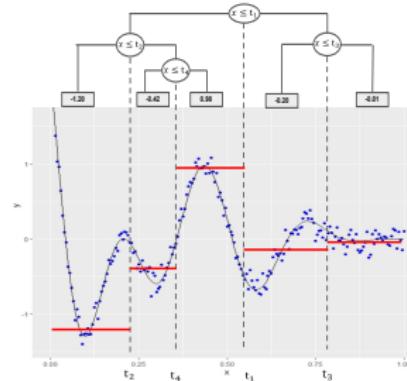
- Know definition of a tree's node
- Understand how split points are derived via empirical risk minimization
- Know which losses can be applied for regression and classification
- Know the connections between empirical risk minimization and impurity minimization

# TREES

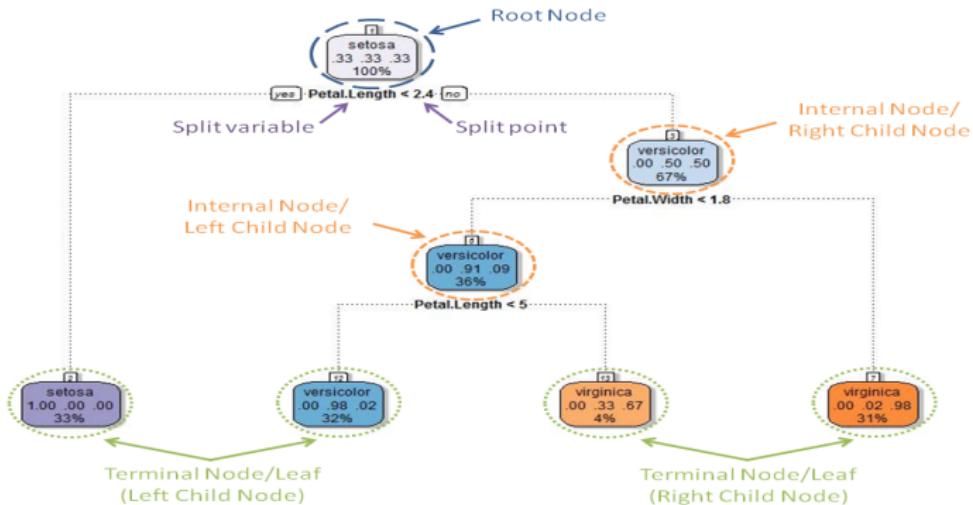
Classification Tree:



Regression Tree:



# SPLITTING CRITERIA



How to find good splitting rules to define the tree?

⇒ empirical risk minimization

# SPLITTING CRITERIA: FORMALIZATION

- Let  $\mathcal{N} \subseteq \mathcal{D}$  be the data that is assigned to a terminal node  $\mathcal{N}$  of a tree.
- Let  $c$  be the predicted constant value for the data assigned to  $\mathcal{N}$ :  
 $\hat{y} \equiv c$  for all  $(\mathbf{x}, y) \in \mathcal{N}$ .
- Then the risk  $\mathcal{R}(\mathcal{N})$  for a leaf is simply the average loss for the data assigned to that leaf under a given loss function  $L$ :

$$\mathcal{R}(\mathcal{N}) = \frac{1}{|\mathcal{N}|} \sum_{(\mathbf{x}, y) \in \mathcal{N}} L(y, c)$$

- The prediction is given by the optimal constant  $c = \arg \min_c \mathcal{R}(\mathcal{N})$

# SPLITTING CRITERIA: FORMALIZATION

- A split w.r.t. **feature**  $x_j$  at **split point**  $t$  divides a parent node  $\mathcal{N}$  into

$$\mathcal{N}_1 = \{(\mathbf{x}, y) \in \mathcal{N} : x_j \leq t\} \text{ and } \mathcal{N}_2 = \{(\mathbf{x}, y) \in \mathcal{N} : x_j > t\}.$$

- In order to evaluate how good a split is, we compute the empirical risks in both child nodes and sum them up:

$$\begin{aligned}\mathcal{R}(\mathcal{N}, j, t) &= \frac{|\mathcal{N}_1|}{|\mathcal{N}|} \mathcal{R}(\mathcal{N}_1) + \frac{|\mathcal{N}_2|}{|\mathcal{N}|} \mathcal{R}(\mathcal{N}_2) \\ &= \frac{1}{|\mathcal{N}|} \left( \sum_{(\mathbf{x}, y) \in \mathcal{N}_1} L(y, c_1) + \sum_{(\mathbf{x}, y) \in \mathcal{N}_2} L(y, c_2) \right)\end{aligned}$$

- Finding the best way to split  $\mathcal{N}$  into  $\mathcal{N}_1, \mathcal{N}_2$  means solving

$$\arg \min_{j, t} \mathcal{R}(\mathcal{N}, j, t)$$

# SPLITTING CRITERIA: REGRESSION

- For regression trees, we usually use  $L_2$  loss:

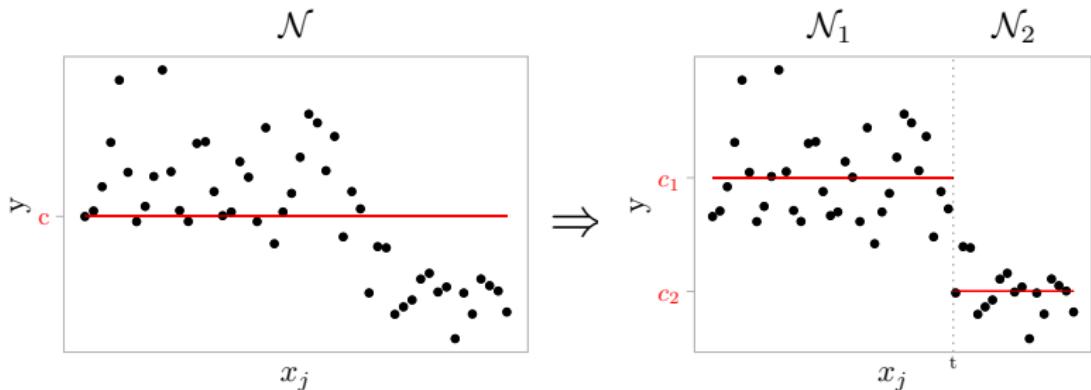
$$\mathcal{R}(\mathcal{N}) = \frac{1}{|\mathcal{N}|} \sum_{(\mathbf{x}, y) \in \mathcal{N}} (y - c)^2$$

- The best constant prediction under  $L_2$  is the mean:

$$c = \bar{y}_{\mathcal{N}} = \frac{1}{|\mathcal{N}|} \sum_{(\mathbf{x}, y) \in \mathcal{N}} y$$

# SPLITTING CRITERIA: REGRESSION

- This means the best split is the one that minimizes the (pooled) variance of the target distribution in the child nodes  $\mathcal{N}_1$  and  $\mathcal{N}_2$ :



We can also interpret this as a way of measuring the impurity of the target distribution, i.e., how much it diverges from a constant in each of the child nodes.

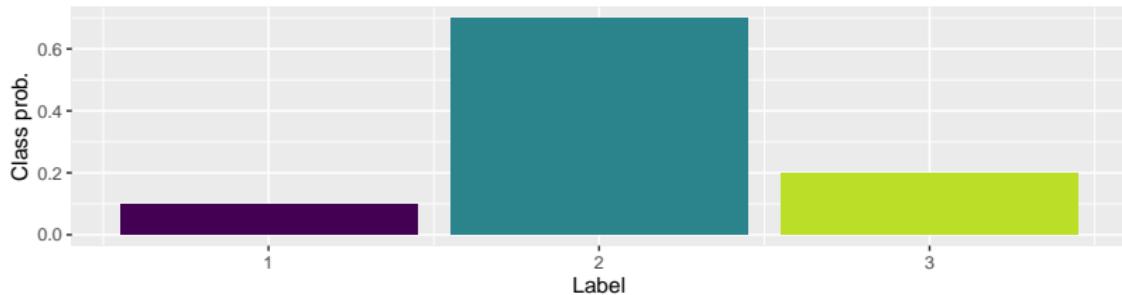
- For  $L_1$  loss,  $c$  is the median of  $y \in \mathcal{N}$ .

# SPLITTING CRITERIA: CLASSIFICATION

- Typically uses either Brier score (so:  $L_2$  loss on probabilities) or Bernoulli loss (as in logistic regression) as loss functions
- Predicted probabilities in node  $\mathcal{N}$  are simply the class proportions in the node:

$$\hat{\pi}_k^{(\mathcal{N})} = \frac{1}{|\mathcal{N}|} \sum_{(\mathbf{x}, y) \in \mathcal{N}} \mathbb{I}(y = k)$$

This is the optimal prediction under both the logistic / Bernoulli loss and the Brier loss.



## SPLITTING CRITERIA: COMMENTS

- Splitting criteria for trees are usually defined in terms of "impurity reduction". Instead of minimizing empirical risk in the child nodes over all possible splits, a measure of "impurity" of the distribution of the target  $y$  in the child nodes is minimized.
- For regression trees, the "impurity" of a node is usually defined as the variance of the  $y^{(i)}$  in the node. Minimizing this "variance impurity" is equivalent to minimizing the squared error loss for a predicted constant in the nodes.

# SPLITTING CRITERIA: COMMENTS

- Minimizing the Brier score is equivalent to minimizing the Gini impurity

$$I(\mathcal{N}) = \sum_{k=1}^g \hat{\pi}_k^{(\mathcal{N})} (1 - \hat{\pi}_k^{(\mathcal{N})})$$

- Minimizing the Bernoulli loss is equivalent to minimizing entropy impurity

$$I(\mathcal{N}) = - \sum_{k=1}^g \hat{\pi}_k^{(\mathcal{N})} \log \hat{\pi}_k^{(\mathcal{N})}$$

- The approach based on loss functions instead of impurity measures is simpler and more straightforward, mathematically equivalent and shows that growing a tree can be understood in terms of empirical risk minimization.

# SPLITTING WITH MISCLASSIFICATION LOSS

- Why don't we use the misclassification loss for classification trees?  
I.e., always predict the majority class in each child node and count how many errors we make.
- In many other cases, we are interested in minimizing this kind of error but have to approximate it by some other criterion instead, since the misclassification loss does not have derivatives that we can use for optimization.  
We don't need derivatives when we optimize the tree, so we could go for it!
- This is possible, but Brier score and Bernoulli loss are more sensitive to changes in the node probabilities, and therefore often preferred

# SPLITTING WITH MISCLASSIFICATION LOSS

Example: two-class problem with 400 obs in each class and two possible splits:

Split 1:

	class 0	class 1
$\mathcal{N}_1$	300	100
$\mathcal{N}_2$	100	300

Split 2:

	class 0	class 1
$\mathcal{N}_1$	400	200
$\mathcal{N}_2$	0	200

- Both splits are equivalent in terms of misclassification error, they each misclassify 200 observations.
- But: Split 2 produces one pure node and is probably preferable.
- Brier loss (Gini impurity) and Bernoulli loss (entropy impurity) prefer the second split

# SPLITTING WITH MISCLASSIFICATION LOSS

- Calculation for Gini:

$$\text{Formula : } \frac{|\mathcal{N}_1|}{|\mathcal{N}|} \cdot 2 \cdot \hat{\pi}_0^{(\mathcal{N}_1)} \hat{\pi}_1^{(\mathcal{N}_1)} + \frac{|\mathcal{N}_2|}{|\mathcal{N}|} \cdot 2 \cdot \hat{\pi}_0^{(\mathcal{N}_2)} \hat{\pi}_1^{(\mathcal{N}_2)} =$$

$$\text{Split 1 : } \frac{1}{2} \cdot 2 \cdot \frac{3}{4} \cdot \frac{1}{4} + \frac{1}{2} \cdot 2 \cdot \frac{1}{4} \cdot \frac{3}{4} = \frac{3}{8}$$

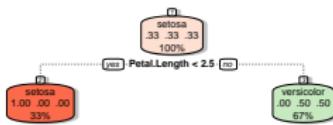
$$\text{Split 2 : } \frac{3}{4} \cdot 2 \cdot \frac{2}{3} \cdot \frac{1}{3} + \frac{1}{4} \cdot 2 \cdot 0 \cdot 1 = \frac{1}{3}$$

# Introduction to Machine Learning

## CART: Growing a Tree

### Learning goals

- Understand how a tree is grown by an exhaustive search over all possible features and split points
- Know where exactly the split point is set if several yield the same empirical risk



# TREE GROWING

We start with an empty tree, a root node that contains all the data.

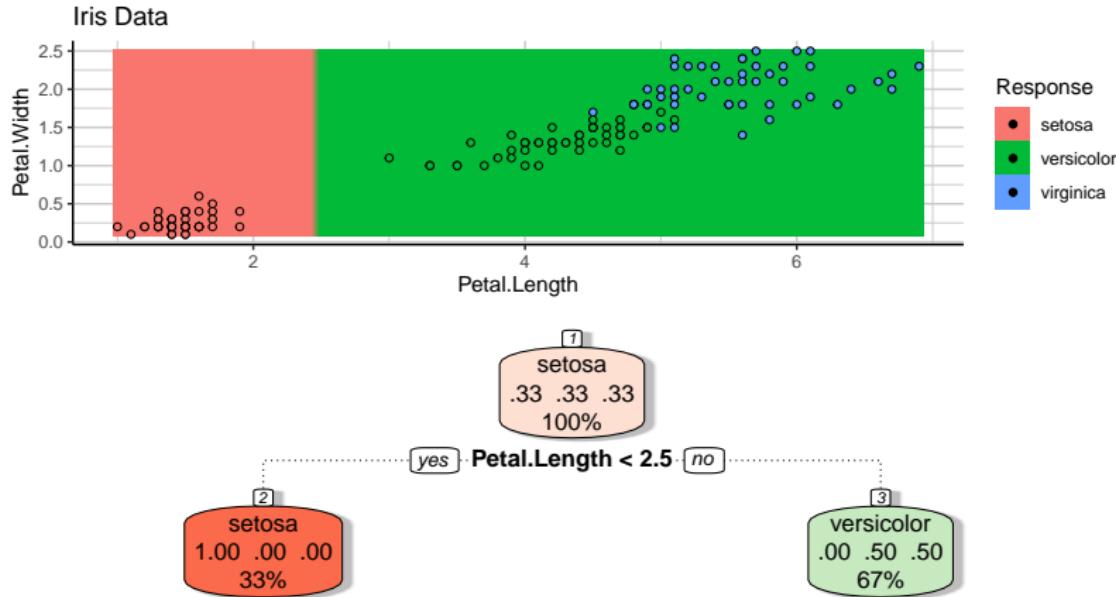
Trees are then grown by recursively applying *greedy* optimization to each node  $\mathcal{N}$ .

Greedy means we do an **exhaustive search**: All possible splits of  $\mathcal{N}$  on all possible points  $t$  for all features  $x_j$  are compared in terms of their empirical risk  $\mathcal{R}(\mathcal{N}, j, t)$ .

The training data is then distributed to child nodes according to the optimal split and the procedure is repeated in the child nodes.

# TREE GROWING

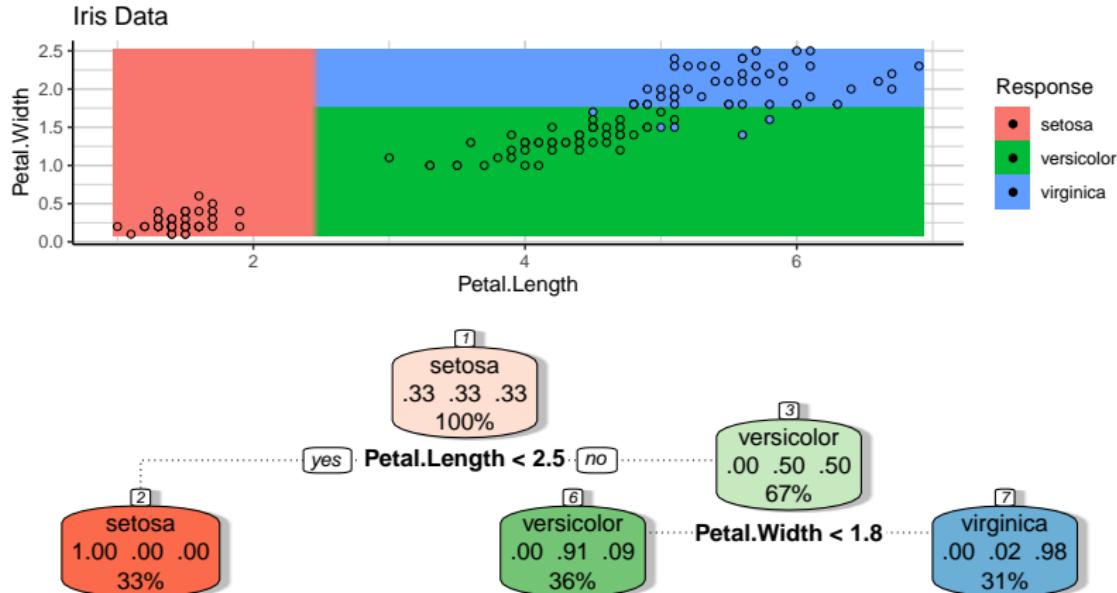
Start with a root node of all data, then search for a feature and split point that minimizes the empirical risk in the child nodes.



Nodes display their current label distribution here for illustration.

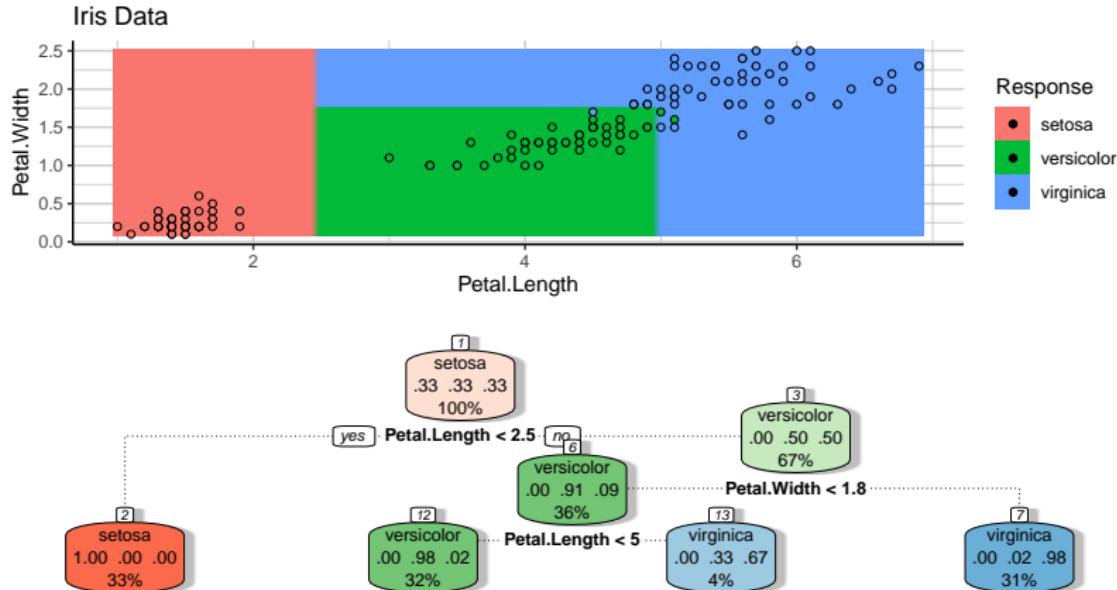
# TREE GROWING

We then proceed recursively for each child node: Iterate over all features, and for each feature over all possible split points. Select the best split and divide data in parent node into left and right child nodes:

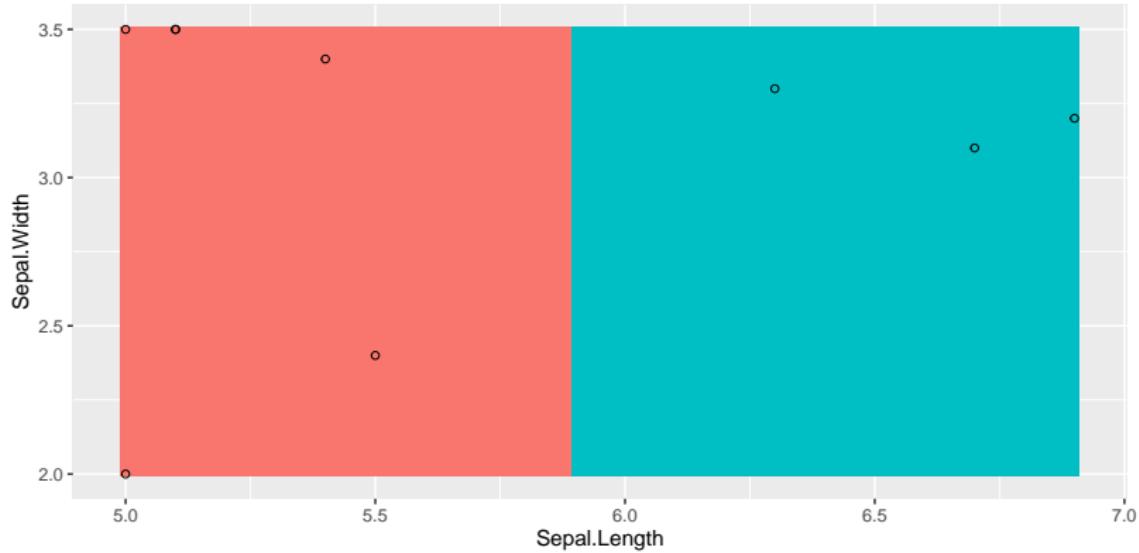


# TREE GROWING

We then proceed recursively for each child node: Iterate over all features, and for each feature over all possible split points. Select the best split and divide data in parent node into left and right child nodes:



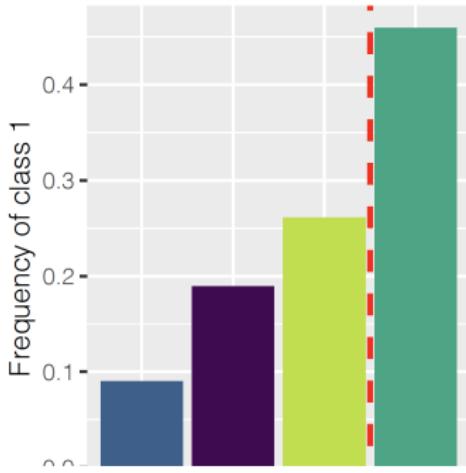
# SPLIT PLACEMENT



Splits are usually placed at the mid-point of the observations they split: the large margin to the next closest observations makes better generalization on new, unseen data more likely.

# Introduction to Machine Learning

## CART: Computational Aspects of Finding Splits



### Learning goals

- Know how monotone feature transformations affect the tree
- Understand how nominal features can be treated effectively while growing a CART
- Understand how missing values can be treated in a CART

# MONOTONE FEATURE TRANSFORMATIONS

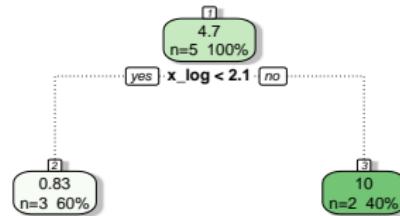
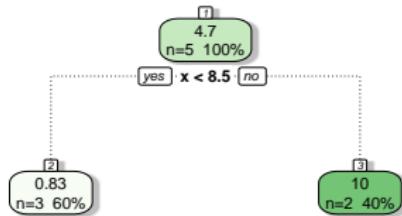
Monotone transformations of one or several features will neither change the value of the splitting criterion nor the structure of the tree, only the numerical value of the split point.

Original data

x	1	2	7.0	10	20
y	1	1	0.5	10	11

Data with log-transformed x

log(x)	0	0.7	1.9	2.3	3
y	1	1.0	0.5	10.0	11



# CART: NOMINAL FEATURES

- A split on a nominal feature partitions the feature levels:

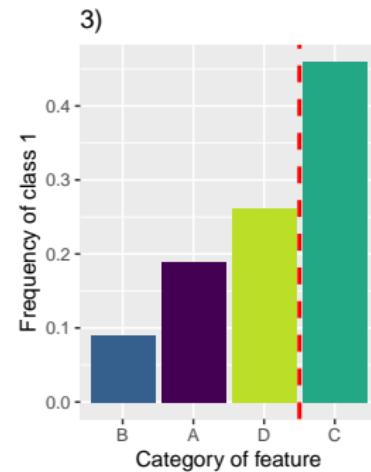
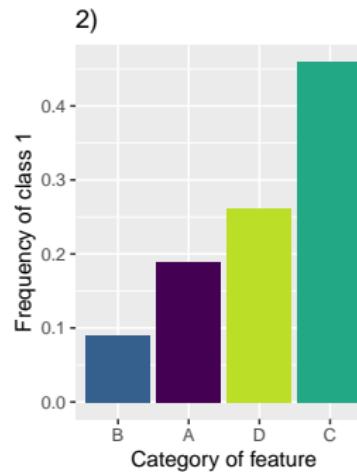
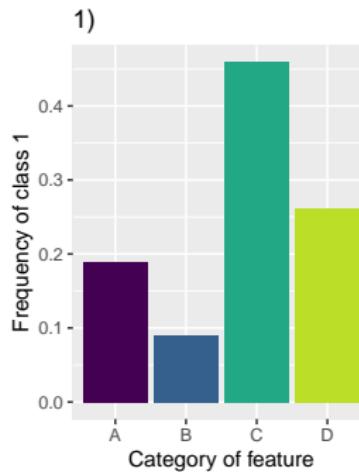
$$x_j \in \{a, c, e\} \leftarrow \mathcal{N} \rightarrow x_j \in \{b, d\}$$

- For a feature with  $m$  levels, there are about  $2^m$  different possible partitions of the  $m$  values into two groups ( $2^{m-1} - 1$  because of symmetry and empty groups).
- Searching over all these becomes prohibitive for larger values of  $m$ .
- For regression with squared loss and binary classification, we can define clever shortcuts.

# CART: NOMINAL FEATURES

For 0 – 1 responses, in each node:

- ① Calculate the proportion of 1-outcomes for each category of the feature in  $\mathcal{N}$ .
- ② Sort the categories according to these proportions.
- ③ The feature can then be treated as if it was ordinal, so we only have to investigate at most  $m - 1$  splits.



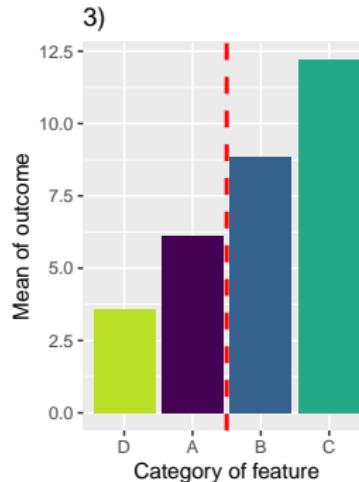
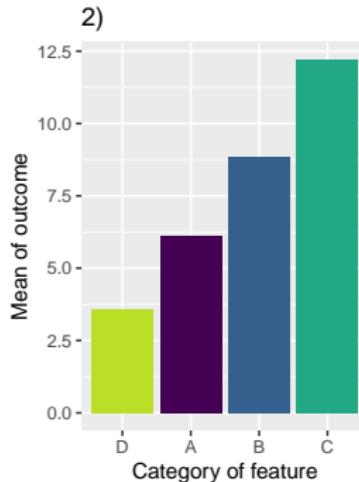
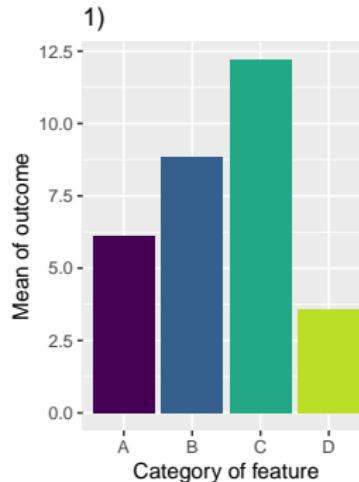
# CART: NOMINAL FEATURES

- This procedure finds the optimal split.
- This result also holds for regression trees (with squared error loss) if the levels of the feature are ordered by increasing mean of the target
- The proofs are not trivial and can be found here:
  - for 0-1 responses:
    - Breiman, 1984, Classification and Regression Trees.
    - Ripley, 1996, Pattern Recognition and Neural Networks.
  - for continuous responses:
    - Fisher, 1958, On grouping for maximum homogeneity.
- Such simplifications are not known for multiclass problems.

# CART: NOMINAL FEATURES

For continuous responses, in each node:

- ① Calculate the mean of the outcome in each category
- ② Sort the categories by increasing mean of the outcome



## CART: MISSING FEATURE VALUES

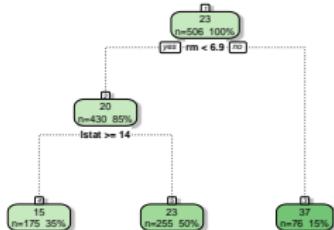
- When splits are evaluated, only observations for which the used feature is not missing are used. (This can actually bias splits towards using features with lots of missing values.)
- CART often use the so-called **surrogate split** principle to automatically deal with missing values in features used for splits during prediction.
- Surrogate splits are created during training. They define replacement splitting rules, using a different feature, that result in almost the same child nodes as the original split.
- When observations are passed down the tree (in training or prediction), and the feature value used in a split is missing, we use a "surrogate split" instead to decide to which branch of the tree the data should be assigned.

# Introduction to Machine Learning

## CART: Stopping Criteria & Pruning

### Learning goals

- Understand which problems arise when growing the tree until the end
- Know different stopping criteria
- Understand the idea of pruning



Pruning with complexity parameter = 0.072.

# OVERFITTING TREES

The **recursive partitioning** procedure used to grow a CART would run until every leaf only contains a single observation.

- Problem 1: This would take a very long time, as the amount of splits we have to try *grows exponentially* with the number of leaves in the trees.
- Problem 2: At some point before that we should stop splitting nodes into ever smaller child nodes: very complex trees with lots of branches and leaves will *overfit the training data*.
- Problem 3: However, it is very hard to tell where we should stop while we're growing the tree: Before we have actually tried all possible additional splits further down a branch, we can't know whether any one of them will be able to reduce the risk by a lot (*horizon effect*).

# STOPPING CRITERIA

Problems 1 and 2 can be “solved” by defining different **stopping criteria**:

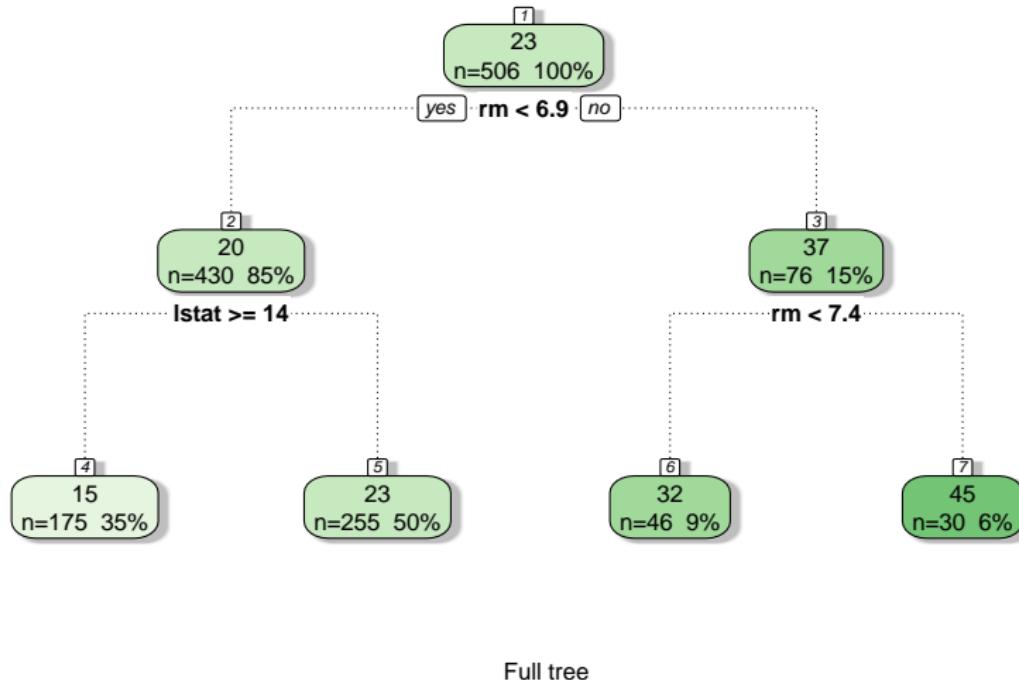
- Stop once the tree has reached a certain number of leaves.
- Don’t try to split a node further if it contains too few observations.
- Don’t perform a split that results in child nodes with too few observations.
- Don’t perform a split unless it achieves a certain minimal improvement of the empirical risk in the child nodes, compared to the empirical risk in the parent node.
- Obviously: Stop once all observations in a node have the same target value (**pure node**) or identical values for all features.

# PRUNING

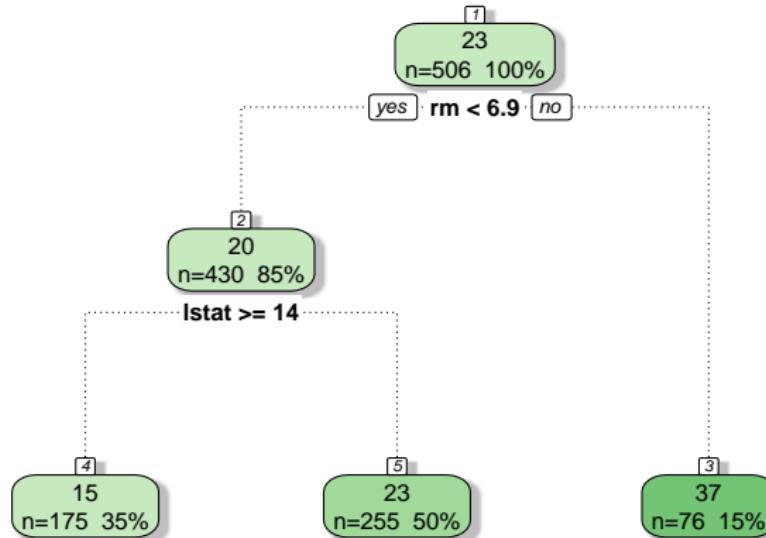
We try to solve problem 3 by **pruning**:

- A method to select the optimal size of a tree
- Finding a combination of suitable strict stopping criteria (“pre-pruning”) is a hard problem: there are many different stopping criteria and it’s hard to find the best combination (see chapter on **tuning**)
- Better: Grow a large tree, then remove branches so that the resulting smaller tree has optimal cross-validation risk
- Feasible without cross-validation: Grow a large tree, then remove branches so that the resulting smaller tree has a good balance between training set performance (risk) and complexity (i.e., number of terminal nodes). The trade-off between complexity and accuracy is governed by a **complexity parameter**.

# PRUNING

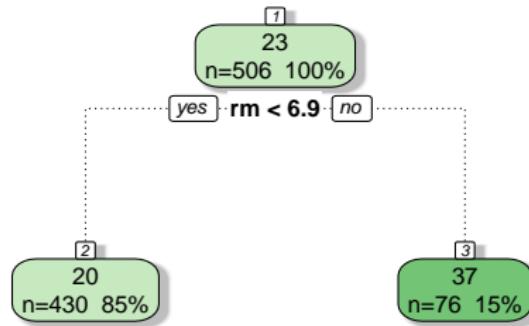


# PRUNING



Pruning with complexity parameter = 0.072.

# PRUNING



Pruning with complexity parameter = 0.171.

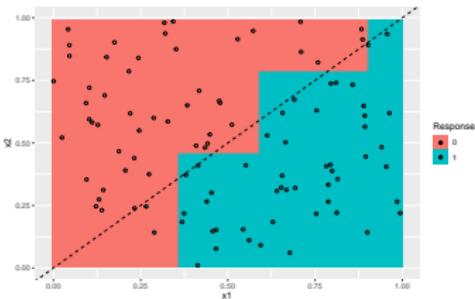
# PRUNING



Pruning with complexity parameter = 0.453.

# Introduction to Machine Learning

## CART: Advantages & Disadvantages



### Learning goals

- Understand advantages and disadvantages of CART
- Know how CART can be expressed in terms of hypothesis space, risk and optimization

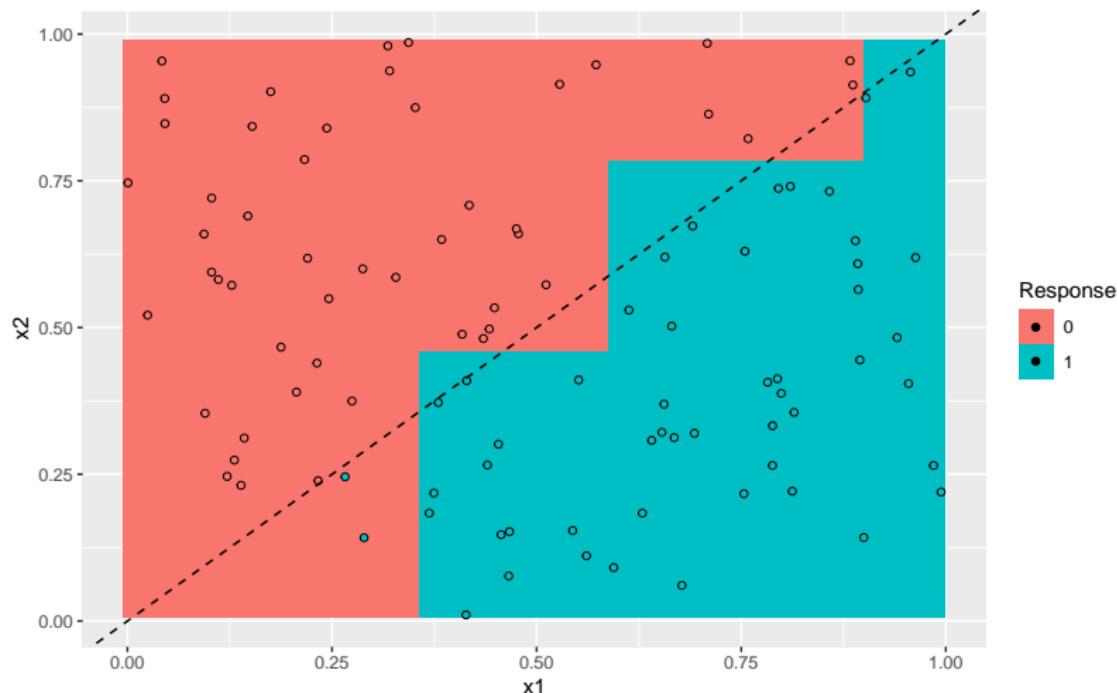
# ADVANTAGES

- Fairly easy to understand, interpret and visualize.
- Not much preprocessing required:
  - automatic handling of non-numerical features
  - automatic handling of missing values via surrogate splits
  - no problems with outliers in features
  - monotone transformations of features change nothing so scaling of features is irrelevant
- Interaction effects between features are easily possible, even of higher orders
- Can model discontinuities and non-linearities (but see "disadvantages")

# ADVANTAGES

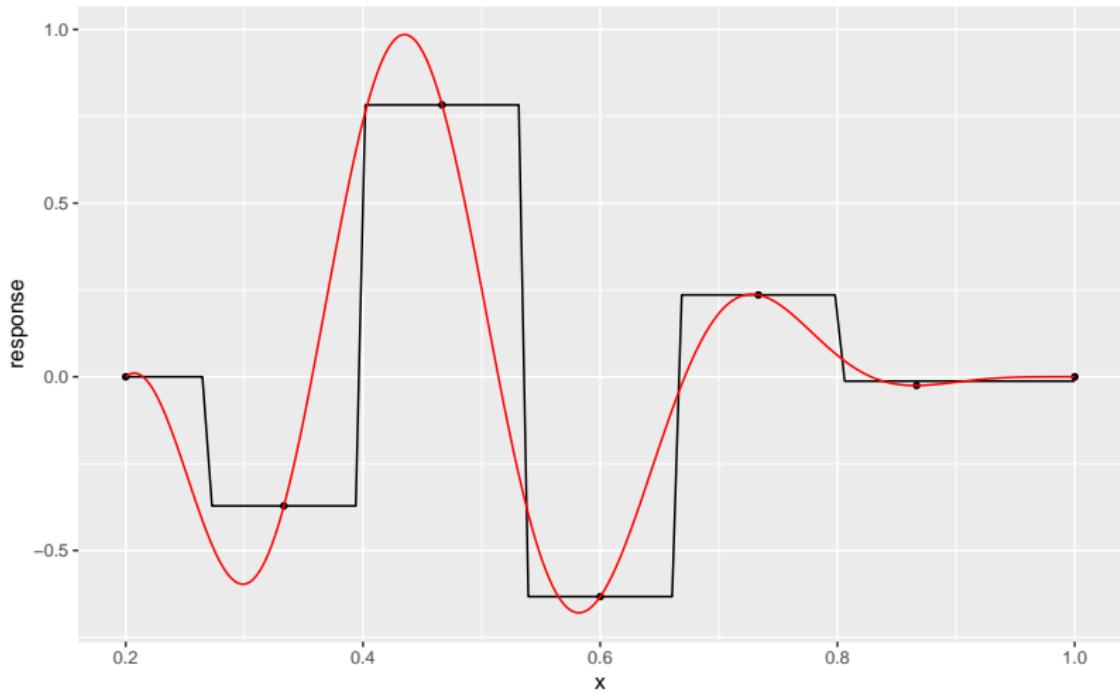
- Performs automatic feature selection
- Quite fast, scales well with larger data
- Flexibility through definition of custom split criteria or leaf-node prediction rules: clustering trees, semi-supervised trees, density estimation, etc.

# DISADVANTAGE: LINEAR DEPENDENCIES



Linear dependencies must be modeled over several splits. Logistic regression would model this easily.

# DISADVANTAGE: SMOOTH FUNCTIONS



Prediction functions of trees are never smooth as they are always step functions.

# DISADVANTAGES

- Empirically not the best predictor: Combine with bagging (forest) or boosting!
- High instability (variance) of the trees. Small changes in the training data can lead to completely different trees. This leads to reduced trust in interpretation and is a reason why prediction errors of trees are usually not the best.
- In regression: Trees define piecewise constant functions, so trees often do not extrapolate well.

# FURTHER TREE METHODOLOGIES

- AID (Sonquist and Morgan, 1964)
- CHAID (Kass, 1980)
- CART (Breiman et al., 1984)
- C4.5 (Quinlan, 1993)
- Unbiased Recursive Partitioning (Hothorn et al., 2006)

# CART: SYNOPSIS

## Hypothesis Space:

CART models are step functions over a rectangular partition of  $\mathcal{X}$ .

Their maximal complexity is controlled by the stopping criteria and the pruning method.

## Risk:

Trees can use any kind of loss function for regression or classification.

## Optimization:

Exhaustive search over all possible splits in each node to minimize the empirical risk in the child nodes.

Most literature on CARTs based on “impurity reduction” which is mathematically equivalent to empirical risk minimization:

Gini impurity  $\cong$  Brier Score loss,

entropy impurity  $\cong$  Bernoulli loss,

variance impurity  $\cong$  L2 loss.

# INTRODUCTION TO MACHINE LEARNING

ML Basics

Supervised Regression

Supervised Classification

k-NN

Performance Evaluation

Classification and Regression Trees (CART)

**Random Forests**

Tuning

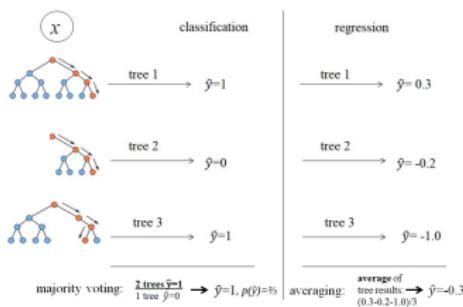
Nested Resampling

mlr3

# Introduction to Machine Learning

## Random Forests: Bagging Ensembles

### Learning goals



- Understand the basic idea of bagging
- Be able to explain the connection of bagging and bootstrap
- Understand how a prediction is computed for bagging
- Understand why bagging improves the predictive power

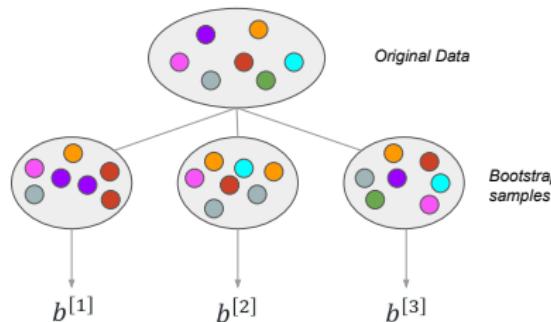
# BAGGING

- Bagging is short for **Bootstrap Aggregation**.
- It's an **ensemble method**, i.e., it combines many models into one big “meta-model”
- Such model ensembles often work much better than their members alone would.
- The constituent models of an ensemble are called **base learners**

# BAGGING

In a **bagging** ensemble, all base learners are of the same type. The only difference between the models is the data they are trained on. Specifically, we train base learners  $b^{[m]}(\mathbf{x})$ ,  $m = 1, \dots, M$  on  $M$  **bootstrap** samples of training data  $\mathcal{D}$ :

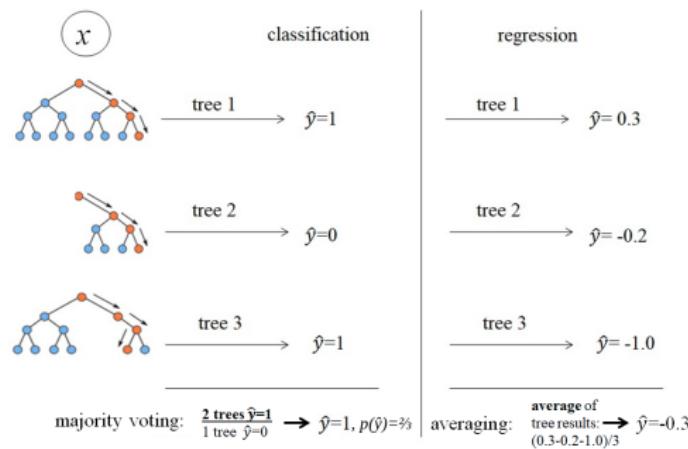
- Draw  $n$  observations from  $\mathcal{D}$  with replacement
- Fit the base learner on each of the  $M$  bootstrap samples to get models  $\hat{f}(x) = \hat{b}^{[m]}(\mathbf{x})$ ,  $m = 1, \dots, M$



# BAGGING

**Aggregate** the predictions of the  $M$  fitted base learners to get the **ensemble model**  $\hat{f}^{[M]}(\mathbf{x})$ :

- Aggregate via averaging (regression) or majority voting (classification)
- Posterior class probabilities  $\hat{\pi}_k(\mathbf{x})$  can be estimated by calculating predicted class frequencies over the ensemble



# WHY/WHEN DOES BAGGING HELP?

In one sentence:

Because the variability of the average of the predictions of many base learner models is smaller than the variability of the predictions from one such base learner model.

If the error of a base learner model is mostly due to (random) variability and not due to structural reasons, combining many such base learners by bagging helps reducing this variability.

# WHY/WHEN DOES BAGGING HELP?

Assume we use quadratic loss and measure instability of the ensemble with

$$\Delta(f^{[M]}(\mathbf{x})) = \frac{1}{M} \sum_m^M (b^{[m]}(\mathbf{x}) - f^{[M]}(\mathbf{x}))^2:$$

$$\begin{aligned}\Delta(f^{[M]}(\mathbf{x})) &= \frac{1}{M} \sum_m^M (b^{[m]}(\mathbf{x}) - f^{[M]}(\mathbf{x}))^2 \\ &= \frac{1}{M} \sum_m^M ((b^{[m]}(\mathbf{x}) - y) + (y - f^{[M]}(\mathbf{x})))^2 \\ &= \frac{1}{M} \sum_m^M L(y, b^{[m]}(\mathbf{x})) + L(y, f^{[M]}(\mathbf{x})) - 2 \underbrace{\left( y - \frac{1}{M} \sum_{m=1}^M b^{[m]}(\mathbf{x}) \right)}_{-2L(y, f^{[M]}(\mathbf{x}))} (y - f^{[M]}(\mathbf{x}))\end{aligned}$$

So, if we take the expected value over the data's distribution:

$$\mathbb{E}_{xy} [L(y, f^{[M]}(\mathbf{x}))] = \frac{1}{M} \sum_m^M \mathbb{E}_{xy} [L(y, b^{[m]}(\mathbf{x}))] - \mathbb{E}_{xy} [\Delta(f^{[M]}(\mathbf{x}))]$$

⇒ The expected loss of the ensemble is lower than the average loss of the single base learner by the amount of instability in the ensemble's base learners.

The more accurate and diverse the base learners, the better.

# IMPROVING BAGGING

How to make  $\mathbb{E}_{xy} [\Delta (f^{[M]}(\mathbf{x}))]$  as large as possible?

$$\mathbb{E}_{xy} [L(y, f^{[M]}(\mathbf{x}))] = \frac{1}{M} \sum_m^M \mathbb{E}_{xy} [L(y, b^{[m]}(\mathbf{x}))] - \mathbb{E}_{xy} [\Delta (f^{[M]}(\mathbf{x}))]$$

Assume  $\mathbb{E}_{xy} [b^{[m]}(\mathbf{x})] = 0$  for simplicity,  $\text{Var}_{xy} [b^{[m]}(\mathbf{x})] = \mathbb{E}_{xy} [(b^{[m]}(\mathbf{x}))^2] = \sigma^2$ ,

$\text{Corr}_{xy} [b^{[m]}(\mathbf{x}), b^{[m']}(\mathbf{x})] = \rho$  for all  $m, m'$ .

$$\implies \text{Var}_{xy} [f^{[M]}(\mathbf{x})] = \frac{1}{M} \sigma^2 + \frac{M-1}{M} \rho \sigma^2 \quad (\dots = \mathbb{E}_{xy} [(f^{[M]}(\mathbf{x}))^2])$$

$$\begin{aligned}\mathbb{E}_{xy} [\Delta (f^{[M]}(\mathbf{x}))] &= \frac{1}{M} \sum_m^M \mathbb{E}_{xy} \left[ (b^{[m]}(\mathbf{x}) - f^{[M]}(\mathbf{x}))^2 \right] \\ &= \frac{1}{M} \left( M \mathbb{E}_{xy} [(b^{[m]}(\mathbf{x}))^2] + M \mathbb{E}_{xy} [(f^{[M]}(\mathbf{x}))^2] - 2M \mathbb{E}_{xy} [b^{[m]}(\mathbf{x}) f^{[M]}(\mathbf{x})] \right) \\ &= \sigma^2 + \mathbb{E}_{xy} [(f^{[M]}(\mathbf{x}))^2] - 2 \frac{1}{M} \sum_{m'}^M \underbrace{\mathbb{E}_{xy} [b^{[m]}(\mathbf{x}) b^{[m']}(\mathbf{x})]}_{=\text{Cov}_{xy} [b^{[m]}(\mathbf{x}), b^{[m']}(\mathbf{x})] + \mathbb{E}_{xy} [b^{[m]}(\mathbf{x})] \mathbb{E}_{xy} [b^{[m']}(\mathbf{x})]} \\ &= \sigma^2 + \left( \frac{1}{M} \sigma^2 + \frac{M-1}{M} \rho \sigma^2 \right) - 2 \left( \frac{M-1}{M} \rho \sigma^2 + \frac{1}{M} \sigma^2 + 0 \cdot 0 \right) \\ &= \frac{M-1}{M} \sigma^2 (1 - \rho)\end{aligned}$$

# IMPROVING BAGGING

$$\mathbb{E}_{xy} \left[ L \left( y, f^{[M]}(\mathbf{x}) \right) \right] = \frac{1}{M} \sum_m^M \mathbb{E}_{xy} \left[ L \left( y, b^{[m]}(\mathbf{x}) \right) \right] - \mathbb{E}_{xy} \left[ \Delta \left( f^{[M]}(\mathbf{x}) \right) \right]$$

$$\mathbb{E}_{xy} \left[ \Delta \left( f^{[M]}(\mathbf{x}) \right) \right] \cong \frac{M-1}{M} \text{Var}_{xy} \left[ b^{[m]}(\mathbf{x}) \right] \left( 1 - \text{Corr}_{xy} \left[ b^{[m]}(\mathbf{x}), b^{[m'](\mathbf{x})} \right] \right)$$

- ⇒ **better base learners** are better (... duh)
- ⇒ **more base learners** are better (theoretically, at least...)
- ⇒ **more variable base learners** are better (as long as their risk stays the same, of course!)
- ⇒ **less correlation between base learners** is better:  
bagging helps more if base learners are wrong in different ways so that their errors “cancel” each other out.

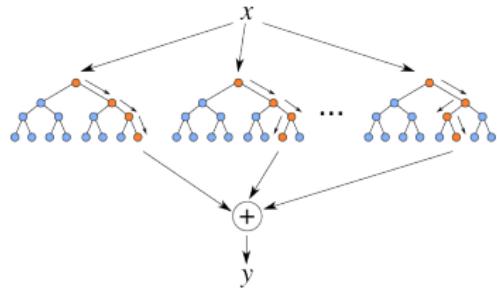
# BAGGING: SYNOPSIS

- Basic idea: fit the same model repeatedly on many **bootstrap** replications of the training data set and **aggregate** the results
- Gains performance by reducing the variance of predictions, but (slightly) increases the bias: it reuses training data many times, so small mistakes can get amplified.
- Works best for unstable/high-variance base learners, where small changes in the training set can cause large changes in predictions: e.g., CART, neural networks, step-wise/forward/backward variable selection for regression
- Works best if base learners' predictions are only weakly correlated: they don't all make the same mistakes.
- Can degrade performance for stable methods like  $k$ -NN, LDA, Naive Bayes, linear regression

# Introduction to Machine Learning

## Random Forest: Introduction

### Learning goals

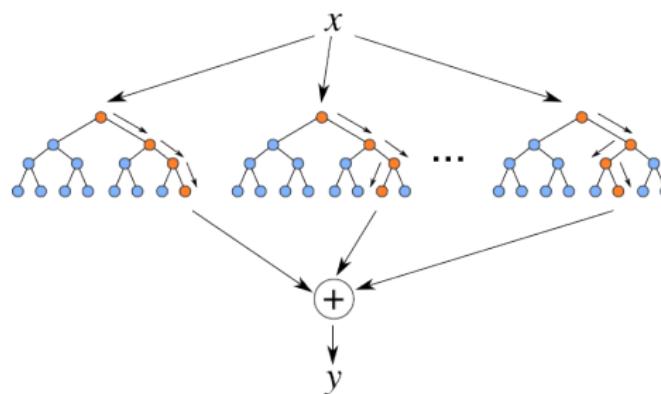


- Know how random forests are defined by extending the idea of bagging
- Understand that the goal is to decorrelate the trees
- Understand that the out-of-bag error is a way to obtain unbiased estimates of the generalization error during training

# RANDOM FORESTS

Modification of bagging for trees proposed by Breiman (2001):

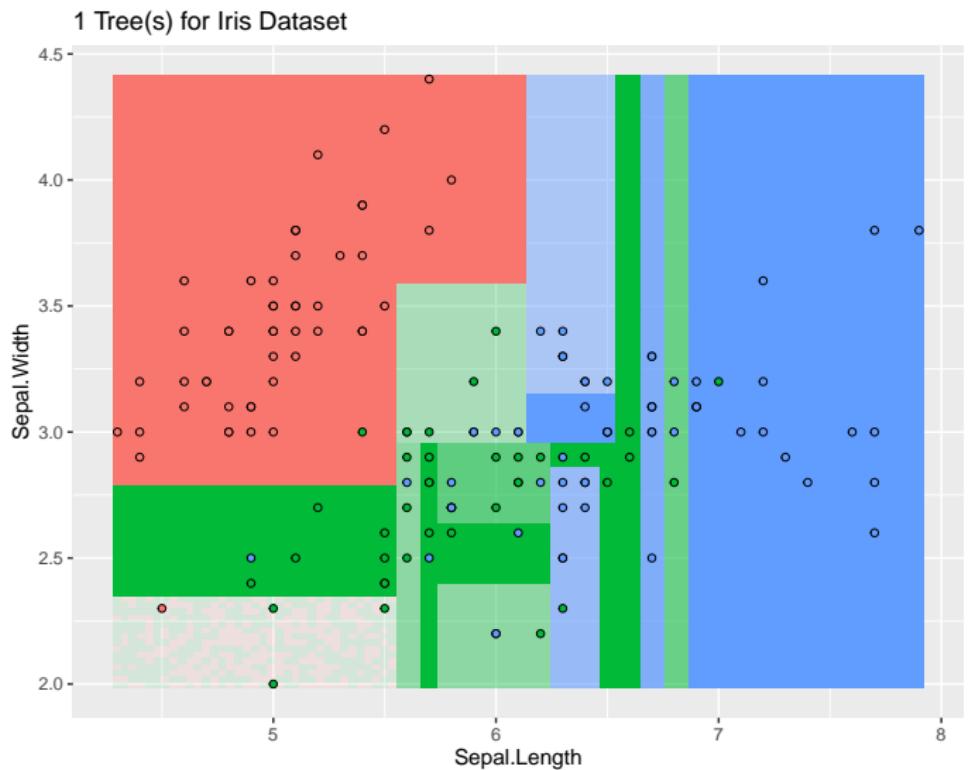
- Tree base learners on bootstrap samples of the data
- Uses **decorrelated** trees by randomizing splits (see below)
- Tree base learners are usually fully expanded, without aggressive early stopping or pruning, to **increase variance of the ensemble**



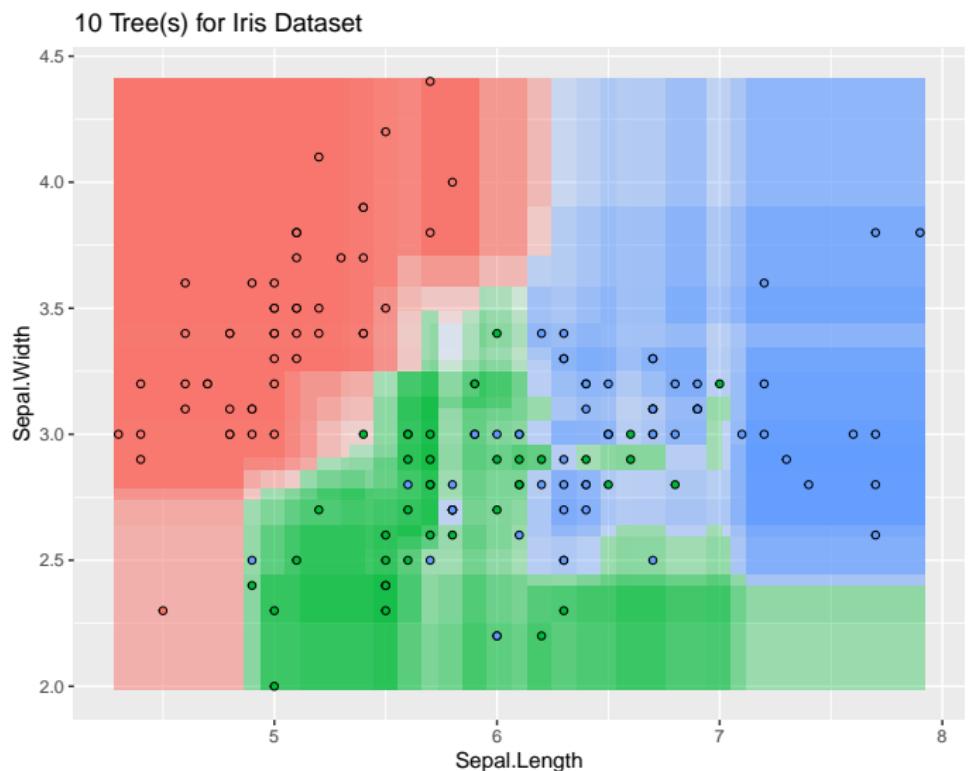
# RANDOM FEATURE SAMPLING

- From our analysis of bagging risk we can see that decorrelating trees improves the ensemble
- Simple randomized approach:  
At each node of each tree, randomly draw  $\text{mtry} \leq p$  candidate features to consider for splitting. Recommended values:
  - Classification:  $\text{mtry} = \lfloor \sqrt{p} \rfloor$
  - Regression:  $\text{mtry} = \lfloor p/3 \rfloor$

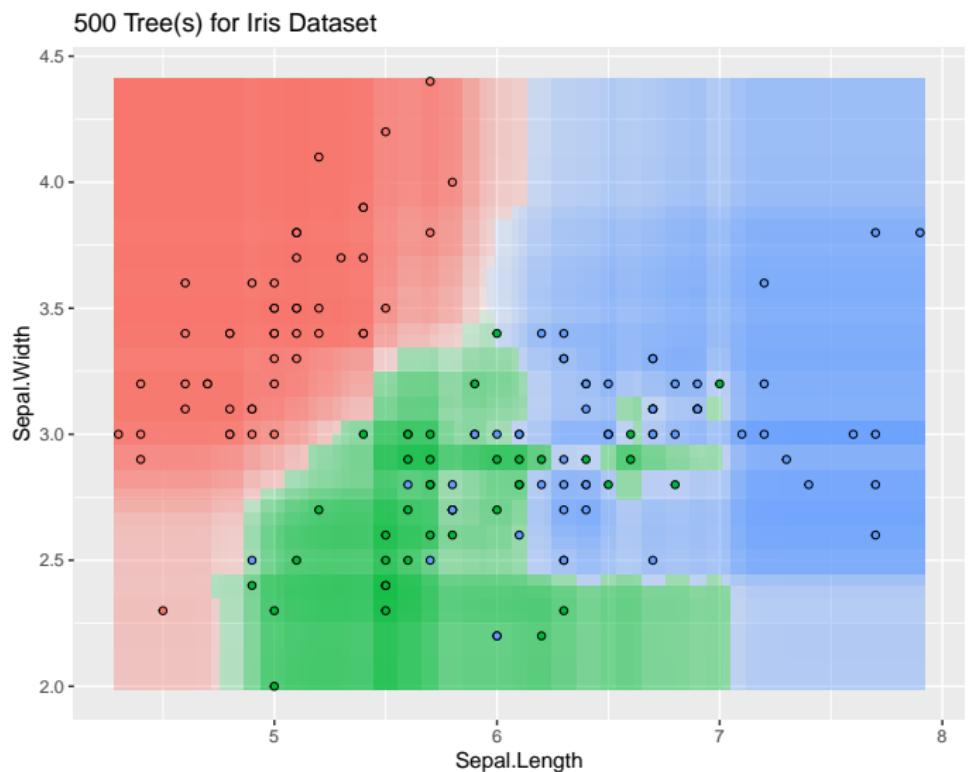
# EFFECT OF ENSEMBLE SIZE



# EFFECT OF ENSEMBLE SIZE

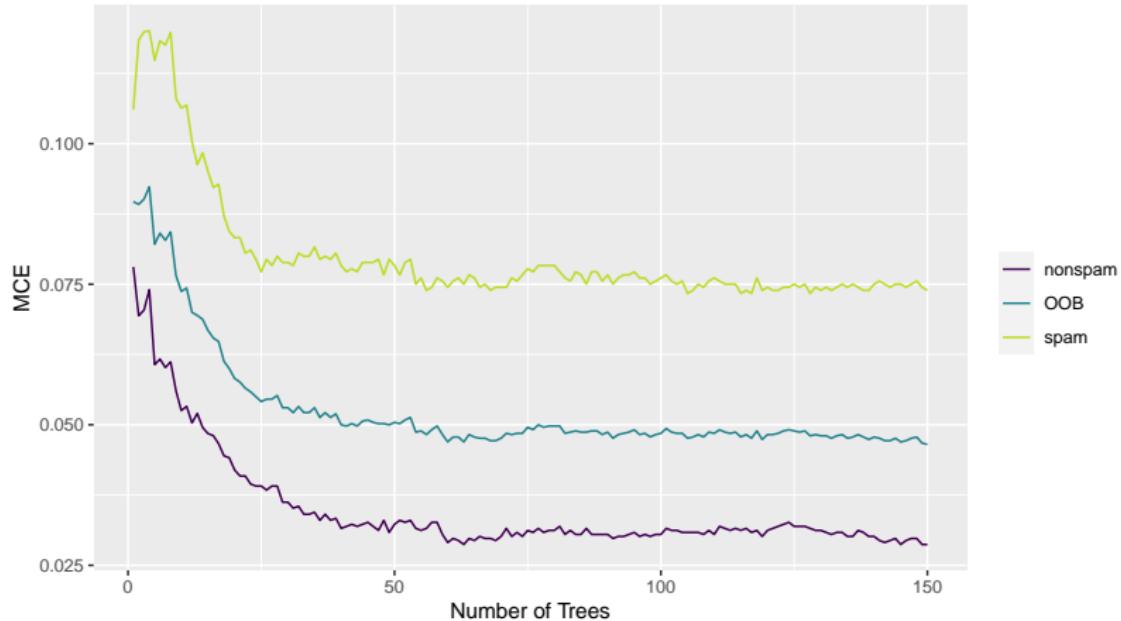


# EFFECT OF ENSEMBLE SIZE



# OUT-OF-BAG ERROR ESTIMATE

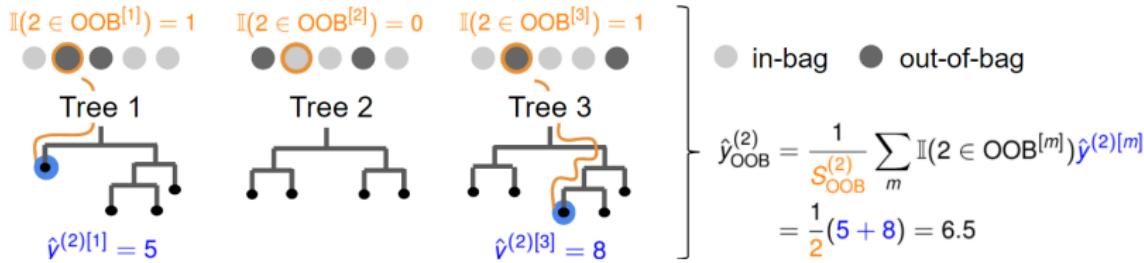
With the RF it is possible to obtain unbiased estimates of the generalization error directly during training, based on the out-of-bag observations for each tree:



# OUT-OF-BAG PREDICTIONS

- For an estimation of the generalization error, we exploit the fact that the  $i$ -th observation acts as unseen test point for all trees in which it is OOB.
- Let  $\text{OOB}^{[m]}$  denote the index set  $\{i \in \{1, \dots, n\} | (\mathbf{x}^{(i)}, y^{(i)}) \text{ is OOB for } b^{[m]}(\mathbf{x})\}$ .
- The number of trees for which the  $i$ -th observation is OOB is then given by  $S_{\text{OOB}}^{(i)} = \sum_{m=1}^M \mathbb{I}(i \in \text{OOB}^{[m]})$ .
- We can compute the ensemble OOB prediction for each observation as:

$$\hat{y}_{\text{OOB}}^{(i)} = \begin{cases} \frac{1}{S_{\text{OOB}}^{(i)}} \sum_{m=1}^M \mathbb{I}(i \in \text{OOB}^{[m]}) \cdot \hat{y}^{(i)[m]} & \text{in regression,} \\ \left[ \frac{1}{S_{\text{OOB}}^{(i)}} \sum_{m=1}^M \mathbb{I}(i \in \text{OOB}^{[m]}) \cdot \mathbb{I}(\hat{h}^{(i)[m]} = k) \right]_{k \in \{1, \dots, g\}} & \text{in classification.} \end{cases}$$



# OUT-OF-BAG ERROR

- Note that the ensemble OOB predictions  $\hat{y}_{\text{OOB}}^{(i)}$  are scalars in regression and  $g$ -valued probability vectors in classification.
- Now we take the average of the resulting point-wise losses to estimate the OOB error of the forest:

$$\widehat{\text{err}}_{\text{OOB}} = \frac{1}{n} \sum_{i=1}^n L(y^{(i)}, \hat{y}_{\text{OOB}}^{(i)})$$

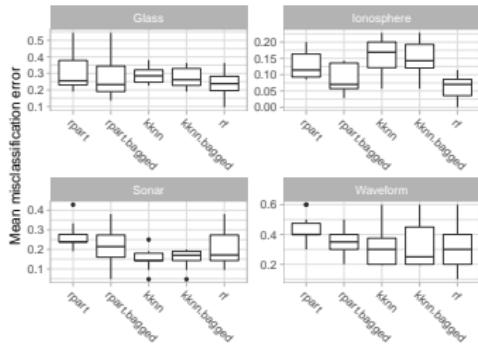
- Computing the probability of  $i$  being OOB in the  $m$ -th tree, we can see that the OOB error estimation is actually akin to 3-fold CV:

$$\mathbb{P}(i \in \text{OOB}^{[m]}) = \left(1 - \frac{1}{n}\right)^n \xrightarrow{n \rightarrow \infty} \frac{1}{e} \approx 0.37$$

for  $i \in \{1, \dots, n\}$ .

# Introduction to Machine Learning

## Random Forest: Benchmarking Trees, Forests, and Bagging K-NN



### Learning goals

- Understand for which kind of learners bagging can improve predictive power

# BENCHMARK: RANDOM FOREST VS. (BAGGED) CART VS. (BAGGED) K-NN

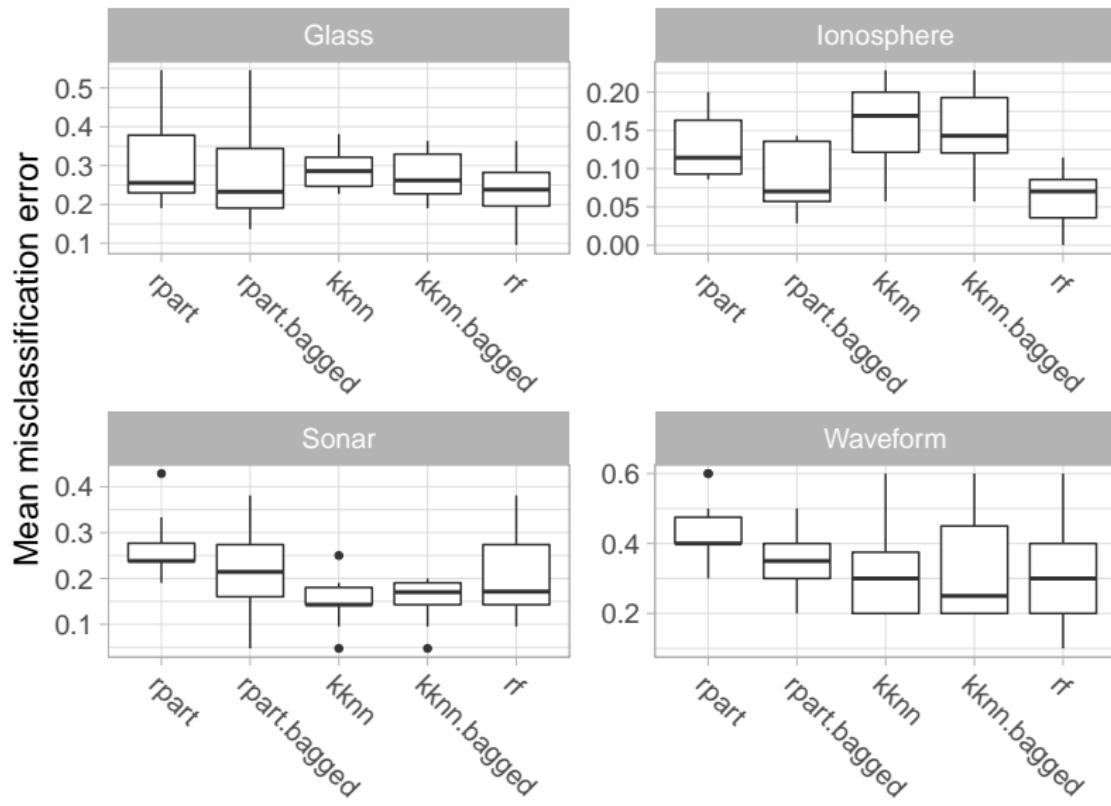
- Goal: Compare performance of random forest against (bagged) stable and (bagged) unstable methods
- Algorithms:
  - classification tree (CART, implemented in `rpart`,  
`max.depth: 30, min.split: 20, cp: 0.01`)
  - bagged classification tree using 50 bagging iterations  
(`bagged.rpart`)
  - k-nearest neighbors (k-NN, implemented in `kknn`,  $k = 7$ )
  - bagged k-nearest neighbors using 50 bagging iterations  
(`bagged.knn`)
  - random forest with 50 trees (implemented in `randomForest`)
- Method to evaluate performance: 10-fold cross-validation
- Performance measure: mean misclassification error on test sets

# BENCHMARK: RANDOM FOREST VS. (BAGGED) CART VS. (BAGGED) K-NN

- Datasets from **mlbench**:

Name	Kind of data	n	p	Task
Glass	Glass identification data	214	10	Predict the type of glass (6 levels) on the basis of the chemical analysis of the glasses represented by the 10 features
Ionosphere	Radar data	351	35	Predict whether the radar returns show evidence of some type of structure in the ionosphere ("good") or not ("bad")
Sonar	Sonar data	208	61	Discriminate between sonar signals bounced off a metal cylinder ("M") and those bounced off a cylindrical rock ("R")
Waveform	Artificial data	100	21	Simulated 3-class problem which is considered to be a difficult pattern recognition problem. Each class is generated by the waveform generator.

# BENCHMARK: RANDOM FOREST VS. (BAGGED) CART VS. (BAGGED) K-NN



## BENCHMARK: RANDOM FOREST VS. (BAGGED) CART VS. (BAGGED) K-NN

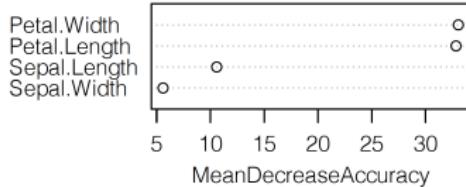
Bagging k-NN does not improve performance because:

- k-NN is stable w.r.t. perturbations
- In a 2-class problem, nearest-neighbor-based classification only changes under bagging if both
  - the nearest neighbor in the learning set is **not** in at least half of the bootstrap samples, but the probability that any given observation is in the bootstrap sample is 63%, which is greater than 50%,
  - and, simultaneously, the *new* nearest neighbor(s) all have a different label than the missing nearest neighbor in those bootstrap samples, which is unlikely for most regions of  $\mathcal{X} \times \mathcal{Y}$ .

# Introduction to Machine Learning

## Random Forests: Feature Importance

### Learning goals



- Understand that the goal of defining variable importance is to enhance interpretability of the random forest
- Know definition of variable importance based on improvement in split criterion
- Know definition of variable importance based on permutations of OOB observations

# VARIABLE IMPORTANCE

- Single trees are highly interpretable
- Random forests as ensembles of trees lose this feature
- Contributions of the different features to the model are difficult to evaluate
- Way out: variable importance measures
- Basic idea: by how much would the performance of the random forest decrease if a specific feature were removed or rendered useless?

# VARIABLE IMPORTANCE

Measure based on improvement in split criterion

**for** features  $x_j, j = 1$  to  $p$  **do**

**for** tree base learners  $\hat{b}^{[m]}(\mathbf{x}), m = 1$  to  $M$  **do**

        Find all nodes  $\mathcal{N}$  in  $\hat{b}^{[m]}(\mathbf{x})$  that use  $x_j$ .

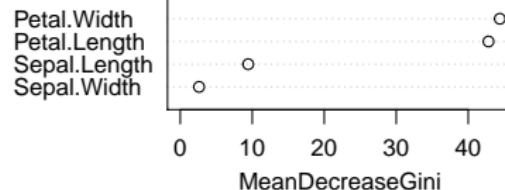
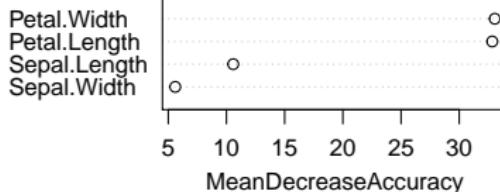
        Compute improvement in splitting criterion achieved by them.

        Add up these improvements.

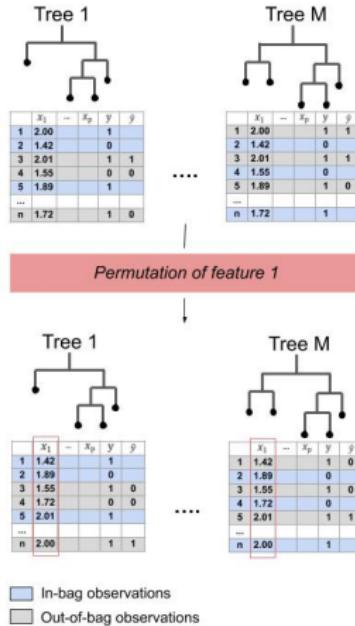
**end for**

    Add up improvements over all trees to get feature importance of  $x_j$ .

**end for**



# VARIABLE IMPORTANCE



Measure based on permutations of OOB observations

Estimate OOB error  $\widehat{\text{err}}_{\text{OOB}}$ .

**for** features  $x_j, j = 1$  to  $p$  **do**

    Perform permutation  $\psi_j$  on  $x_j$  to distort

    feature-target relation for  $x_j$ .

**for** distorted observations  $(\mathbf{x}_{\psi_j}^{(i)}, y^{(i)})$ ,  $i = 1$  to  $n$  **do**

        Compute OOB prediction  $\hat{y}_{\text{OOB}, \psi_j}^{(i)}$ .

        Compute corresponding loss  $L(y^{(i)}, \hat{y}_{\text{OOB}, \psi_j}^{(i)})$ .

**end for**

    Estimate importance of  $j$ -th variable

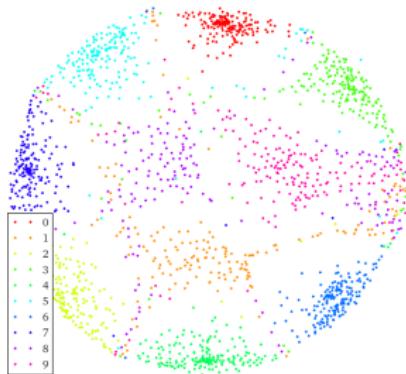
$$\widehat{VI}_j = \widehat{\text{err}}_{\text{OOB}, \psi_j} - \widehat{\text{err}}_{\text{OOB}}$$

$$= \frac{1}{n} \sum_{i=1}^n L(y^{(i)}, \hat{y}_{\text{OOB}, \psi_j}^{(i)}) - \widehat{\text{err}}_{\text{OOB}}.$$

**end for**

# Introduction to Machine Learning

## Random Forests: Proximities



### Learning goals

- Understand how a random forest can be used to define proximities of observations
- Know how proximities can be used for missing data, outliers, mislabeled data and a visualization of the forest

# RANDOM FOREST PROXIMITIES

- One of the most useful tools in random forests
- A measure of similarity ("closeness" or "nearness") of observations derived from random forests
- Can be calculated for each pair of observations
- Definition:
  - The proximity between two observations  $\mathbf{x}^{(i)}$  and  $\mathbf{x}^{(j)}$  is calculated by measuring the number of times that these two observations are placed in the same terminal node of the same tree of the random forest, divided by the number of trees in the forest
  - The proximity of observations  $\mathbf{x}^{(i)}$  and  $\mathbf{x}^{(j)}$  can be written as  $\text{prox}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$
  - The proximities form an intrinsic similarity measure between pairs of observations
- The proximities of all observations form a symmetric  $n \times n$  matrix.

# RANDOM FOREST PROXIMITIES

- Algorithm:
  - Once a random forest has been trained, all of the training data is put through each tree (both in- and out-of-bag).
  - Every time two observations  $\mathbf{x}^{(i)}$  and  $\mathbf{x}^{(j)}$  end up in the same terminal node of a tree, their proximity is increased by one.
  - Once all data has been put through all trees and the proximities have been counted, the proximities are normalized by dividing them by the number of trees.

# USING RANDOM FOREST PROXIMITIES

- Imputing missing data:
  - ➊ Replace missing values for a given variable using the median of the non-missing values
  - ➋ Get proximities
  - ➌ Replace missing values in observation  $\mathbf{x}^{(i)}$  by a weighted average of non-missing values, with weights proportional to the proximity between observation  $\mathbf{x}^{(i)}$  and the observations with the non-missing values

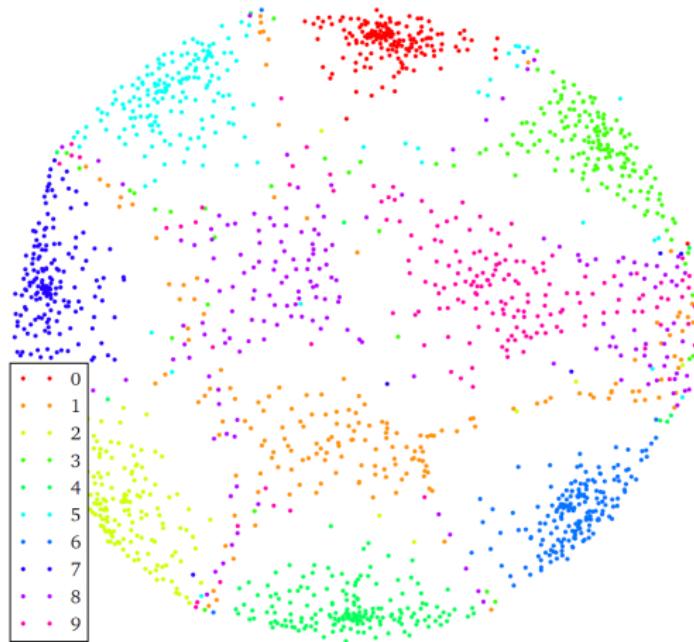
Steps 2 and 3 are then iterated a few times.

- Locating outliers:
  - An outlier is an observation whose proximities to all other observations are small
  - Measure of outlyingness can be computed for each observation in the training sample

# USING RANDOM FOREST PROXIMITIES

- If the measure is unusually large, the observation should be carefully inspected
- Identifying mislabeled data:
  - Instances in the training data set are sometimes labeled ambiguously or incorrectly, especially in “manually” created data sets.
  - Proximities can help in finding them: they often show up as outliers in terms of their proximity values.
- Visualizing the forest:
  - The values  $1 - \text{prox}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$  can be thought of as distances in a high-dimensional space
  - They can be projected onto a low-dimensional space using metric multidimensional scaling (MDS)
  - Metric multidimensional scaling uses eigenvectors of a modified version of the proximity matrix to get scaling coordinates

# USING RANDOM FOREST PROXIMITIES



Proximity plot for a 10-class handwritten digit classification task.

image from G. Louppe (2014) *Understanding Random Forests* arXiv:1407.7502.

# USING RANDOM FOREST PROXIMITIES

- The figure depicts the proximity matrix learnt for a 10-class handwritten digit classification task
  - proximity matrix distances projected onto the plane using multidimensional scaling
  - samples from the same class form identifiable clusters, which suggests that they share a common structure
  - also shows the fact for which classes errors occur, e.g., digits 1 and 8 have high within-class variance and have overlaps with other classes

# Introduction to Machine Learning

## Random Forests: Advantages and Disadvantages

### Learning goals

- Know advantages and disadvantages of random forests
- Be able to explain random forests in terms of hypothesis space, risk and optimization

# RANDOM FOREST: ADVANTAGES

- All advantages of trees also apply to RF: not much preprocessing required, missing value handling, etc.
- Easy to parallelize
- Often works well (enough)
- Integrated variable importance
- Integrated estimation of generalization performance via OOB error
- Works well on high-dimensional data
- Works well on data with irrelevant “noise” variables
- Often not much tuning necessary

# RANDOM FOREST: DISADVANTAGES

- Often sub-optimal for regression
- Same extrapolation problem as for trees
- Harder to interpret than trees (but many extra tools are nowadays available for interpreting RFs)
- Implementation can be memory-hungry
- Prediction is computationally demanding for large ensembles

# RANDOM FOREST: SYNOPSIS

## Hypothesis Space:

Random forest models are (sums of) step functions over rectangular partitions of (subspaces of)  $\mathcal{X}$ .

Their maximal complexity is controlled by the number of trees in the random forest ensemble and the stopping criteria for the constituent trees.

## Risk:

Like trees, random forests can use any kind of loss function for regression or classification.

## Optimization:

Exhaustive search over all (randomly selected!) candidate splits in each node of each tree to minimize the empirical risk in the child nodes.

Like all bagging methods, optimization can be done in parallel over the ensemble members.

# INTRODUCTION TO MACHINE LEARNING

ML Basics

Supervised Regression

Supervised Classification

k-NN

Performance Evaluation

Classification and Regression Trees (CART)

Random Forests

**Tuning**

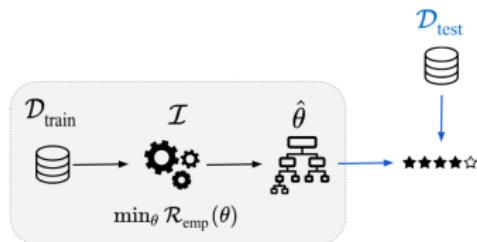
Nested Resampling

mlr3

# Introduction to Machine Learning

## Hyperparameter Tuning - Introduction

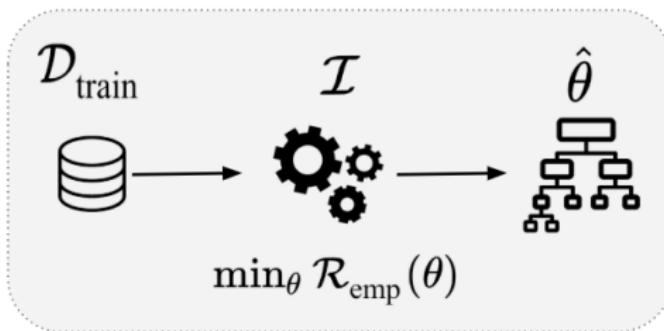
### Learning goals



- Understand the difference between model parameters and hyperparameters
- Know different types of hyperparameters
- Be able to explain the goal of hyperparameter tuning

# MOTIVATING EXAMPLE

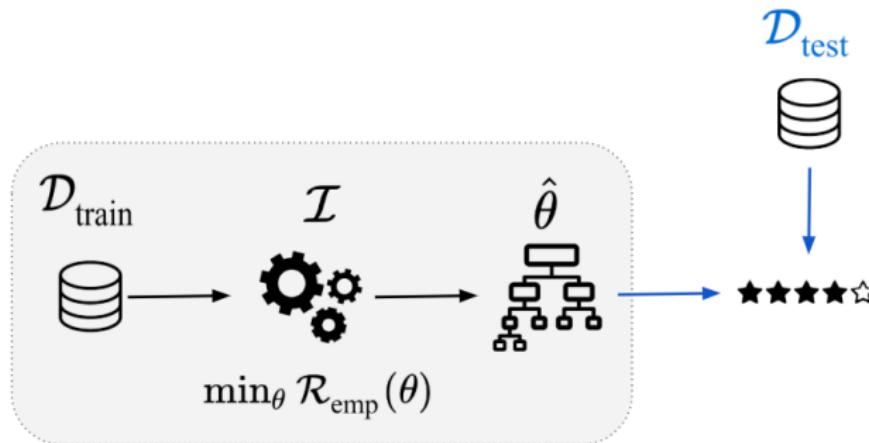
- Given a data set, we want to train a classification tree.
- We feel that a maximum tree depth of 4 has worked out well for us previously, so we decide to set this hyperparameter to 4.
- The learner ("inducer")  $\mathcal{I}$  takes the input data, internally performs **empirical risk minimization**, and returns a fitted tree model  $\hat{f}(\mathbf{x}) = f(\mathbf{x}, \hat{\theta})$  of at most depth  $\lambda = 4$  that minimizes the empirical risk.



# MOTIVATING EXAMPLE

- We are **actually** interested in the **generalization performance**  $GE(\hat{f})$  of the estimated model on new, previously unseen data.
- We estimate the generalization performance by evaluating the model  $\hat{f}$  on a test set  $\mathcal{D}_{\text{test}}$ :

$$\widehat{GE}_{\mathcal{D}_{\text{test}}}(\hat{f}) = \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}_{\text{test}}} L(y, \hat{f}(\mathbf{x}))$$



# MOTIVATING EXAMPLE

- But many ML algorithms are sensitive w.r.t. a good setting of their hyperparameters, and generalization performance might be bad if we have chosen a suboptimal configuration:
  - The data may be too complex to be modeled by a tree of depth 4
  - The data may be much simpler than we thought, and a tree of depth 4 overfits

⇒ Algorithmically try out different values for the tree depth. For each maximum depth  $\lambda$ , we have to train the model **to completion** and evaluate its performance on the test set.

- We choose the tree depth  $\lambda$  that is **optimal** w.r.t. the generalization error of the model.

# MODEL PARAMETERS VS. HYPERPARAMETERS

It is critical to understand the difference between model parameters and hyperparameters.

**Model parameters** are optimized during training, typically via loss minimization. They are an **output** of the training. Examples:

- The splits and terminal node constants of a tree learner
- Coefficients  $\theta$  of a linear model  $f(\mathbf{x}) = \theta^T \mathbf{x}$

# MODEL PARAMETERS VS. HYPERPARAMETERS

In contrast, **hyperparameters** (HPs) are not decided during training. They must be specified before the training, they are an **input** of the training. Hyperparameters often control the complexity of a model, i.e., how flexible the model is. But they can in principle influence any structural property of a model or computational part of the training process.

Examples:

- The maximum depth of a tree
- $k$  and which distance measure to use for  $k$ -NN
- The number and maximal order of interactions to be included in a linear regression model

# TYPES OF HYPERPARAMETERS

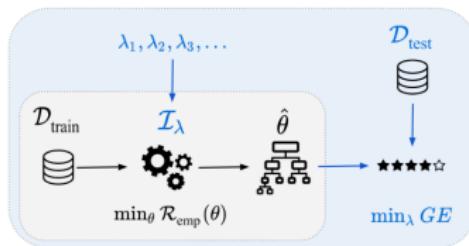
We summarize all hyperparameters we want to tune over in a vector  $\lambda \in \Lambda$  of (possibly) mixed type. HPs can have different types:

- Real-valued parameters, e.g.:
  - Minimal error improvement in a tree to accept a split
  - Bandwidths of the kernel density estimates for Naive Bayes
- Integer parameters, e.g.:
  - Neighborhood size  $k$  for  $k$ -NN
  - $mtry$  in a random forest
- Categorical parameters, e.g.:
  - Which split criterion for classification trees?
  - Which distance measure for  $k$ -NN?

Hyperparameters are often **hierarchically dependent** on each other, e.g., if we use a kernel-density estimate for Naive Bayes, what is its width?

# Introduction to Machine Learning

## Hyperparameter Tuning - Problem Definition



### Learning goals

- Understand tuning as a bi-level optimization problem
- Know the components of a tuning problem
- Be able to explain what makes tuning a complex problem

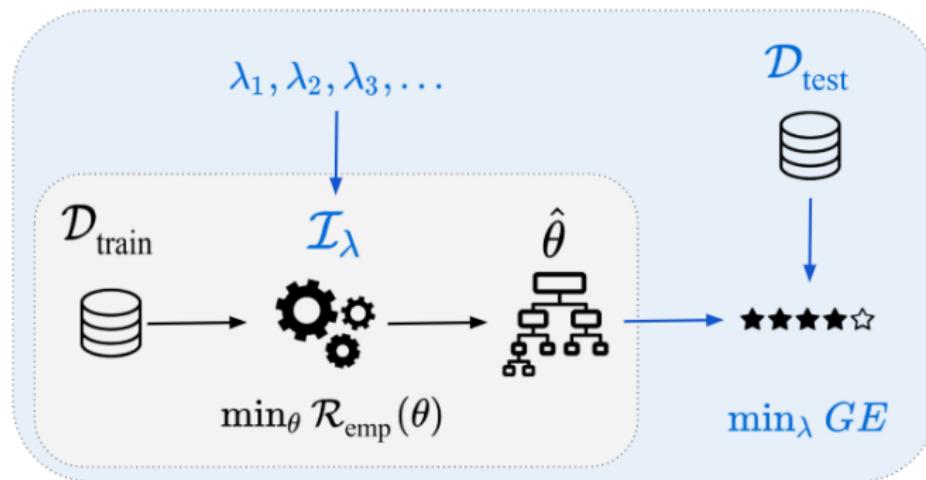
# TUNING

Recall: **Hyperparameters**  $\lambda$  are parameters that are *inputs* to the training problem in which a learner  $\mathcal{I}$  minimizes the empirical risk on a training data set in order to find optimal **model parameters**  $\theta$  which define the fitted model  $\hat{f}$ .

**(Hyperparameter) Tuning** is the process of finding good model hyperparameters  $\lambda$ .

# TUNING: A BI-LEVEL OPTIMIZATION PROBLEM

We face a **bi-level** optimization problem: The well-known risk minimization problem to find  $\hat{f}$  is **nested** within the outer hyperparameter optimization (also called second-level problem):



# TUNING: A BI-LEVEL OPTIMIZATION PROBLEM

- For a learning algorithm  $\mathcal{I}$  (also inducer) with  $d$  hyperparameters, the hyperparameter **configuration space** is:

$$\Lambda = \Lambda_1 \times \Lambda_2 \times \dots \times \Lambda_d,$$

where  $\Lambda_i$  is the domain of the  $i$ -th hyperparameter.

- The domains can be continuous, discrete or categorical.
- For practical reasons, the domain of a continuous or integer-valued hyperparameter is typically bounded.
- A vector in this configuration space is denoted as  $\lambda \in \Lambda$ .
- A learning algorithm  $\mathcal{I}$  takes a (training) dataset  $\mathcal{D} \in \mathbb{D}$  and a hyperparameter configuration  $\lambda \in \Lambda$  and returns a trained model (through risk minimization)

$$\begin{aligned}\mathcal{I} : \left( \bigcup_{n \in \mathbb{N}} (\mathcal{X} \times \mathcal{Y})^n \right) \times \Lambda &\rightarrow \mathcal{H} \\ (\mathcal{D}, \lambda) &\mapsto \mathcal{I}(\mathcal{D}, \lambda) = \hat{t}_{\mathcal{D}, \lambda}\end{aligned}$$

# TUNING: A BI-LEVEL OPTIMIZATION PROBLEM

We formally state the nested hyperparameter tuning problem as:

$$\min_{\lambda \in \Lambda} \widehat{GE}_{\mathcal{D}_{\text{test}}}(\mathcal{I}(\mathcal{D}_{\text{train}}, \lambda))$$

- The learner  $\mathcal{I}(\mathcal{D}_{\text{train}}, \lambda)$  takes a training data set as well as hyperparameter settings  $\lambda$  (e.g., the maximal depth of a classification tree) as an input.
- $\mathcal{I}(\mathcal{D}_{\text{train}}, \lambda)$  performs empirical risk minimization on the training data and returns the optimal model  $\hat{f}$  for the given hyperparameters.
- Note that for the estimation of the generalization error, more sophisticated resampling strategies like cross-validation can be used.

# TUNING: A BI-LEVEL OPTIMIZATION PROBLEM

The components of a tuning problem are:

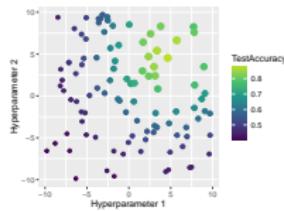
- The data set
- The learner (possibly: several competing learners?) that is tuned
- The learner's hyperparameters and their respective regions-of-interest over which we optimize
- The performance measure, as determined by the application.  
Not necessarily identical to the loss function that defines the risk minimization problem for the learner!
- A (resampling) procedure for estimating the predictive performance

# WHY IS TUNING SO HARD?

- Tuning is derivative-free (“black box problem”): It is usually impossible to compute derivatives of the objective (i.e., the resampled performance measure) that we optimize with regard to the HPs. All we can do is evaluate the performance for a given hyperparameter configuration.
- Every evaluation requires one or multiple train and predict steps of the learner. I.e., every evaluation is very **expensive**.
- Even worse: the answer we get from that evaluation is **not exact, but stochastic** in most settings, as we use resampling.
- Categorical and dependent hyperparameters aggravate our difficulties: the space of hyperparameters we optimize over has a non-metric, complicated structure.

# Introduction to Machine Learning

## Hyperparameter Tuning - Basic Techniques



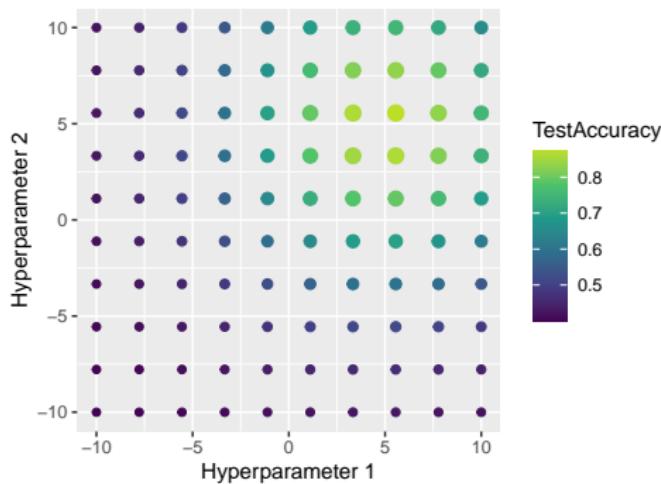
### Learning goals

- Understand the idea of grid search
- Understand the idea of random search
- Be able to discuss advantages and disadvantages of the two methods

# GRID SEARCH

- Simple technique which is still quite popular, tries all HP combinations on a multi-dimensional discretized grid
- For each hyperparameter a finite set of candidates is predefined
- Then, we simply search all possible combinations in arbitrary order

Grid search over 10x10 points



# GRID SEARCH

## Advantages

- Very easy to implement
- All parameter types possible
- Parallelizing computation is trivial

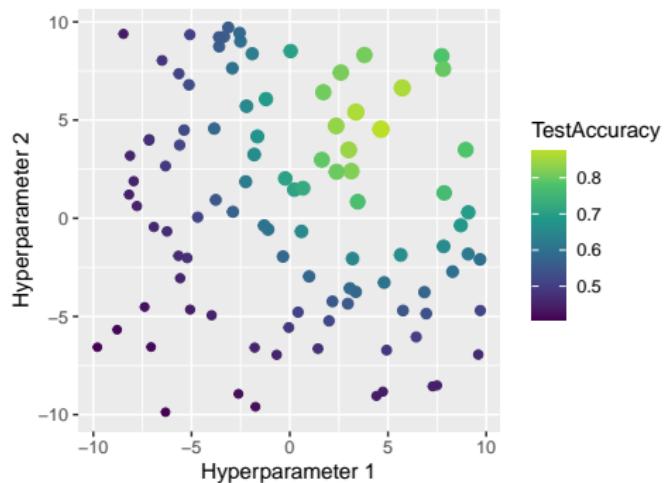
## Disadvantages

- Scales badly: combinatorial explosion
- Inefficient: searches large irrelevant areas
- Arbitrary: which values / discretization?

# RANDOM SEARCH

- Small variation of grid search
- Uniformly sample from the region-of-interest

Random search over 100 points



# RANDOM SEARCH

## Advantages

- Like grid search: very easy to implement, all parameter types possible, trivial parallelization
- Anytime algorithm: can stop the search whenever our budget for computation is exhausted, or continue until we reach our performance goal.
- No discretization: each individual parameter is tried with a different value every time

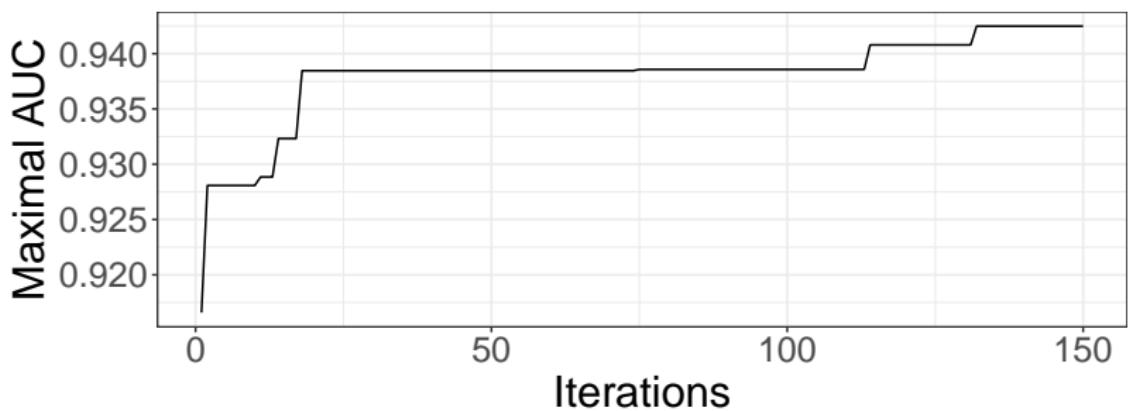
## Disadvantages

- Inefficient: many evaluations in areas with low likelihood for improvement
- Scales badly: high-dimensional hyperparameter spaces need *lots* of samples to cover.

# TUNING EXAMPLE

Tuning random forest with random search and 5CV on the sonar data set for AUC:

Hyperparameter	Type	Min	Max
num.trees	integer	3	500
mtry	integer	5	50
min.node.size	integer	10	100



# INTRODUCTION TO MACHINE LEARNING

ML Basics

Supervised Regression

Supervised Classification

k-NN

Performance Evaluation

Classification and Regression Trees (CART)

Random Forests

Tuning

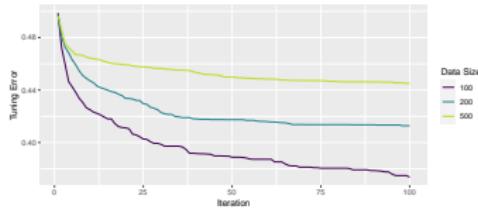
**Nested Resampling**

mlr3

# Introduction to Machine Learning

## Nested Resampling Motivation

### Learning goals



- Understand the problem of overfitting
- Be able to explain the untouched test set principle and how it motivates the idea of nested resampling

# MOTIVATION

Selecting the best model from a set of potential candidates (e.g., different classes of learners, different hyperparameter settings, different feature sets, different preprocessing, ....) is an important part of most machine learning problems.

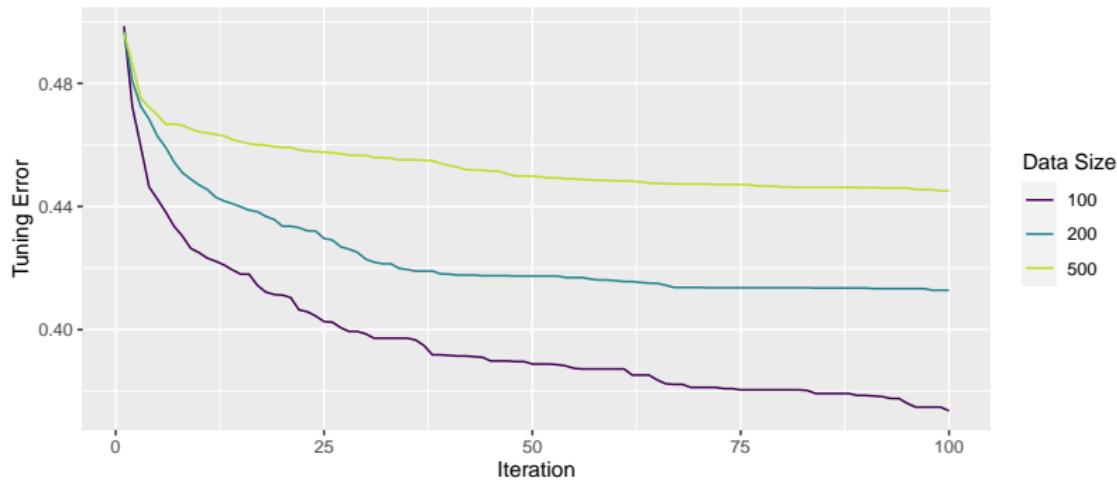
## Problem

- We cannot evaluate our finally selected learner on the same resampling splits that we have used to perform model selection for it, e.g., to tune its hyperparameters.
- By repeatedly evaluating the learner on the same test set, or the same CV splits, information about the test set “leaks” into our evaluation.
- Danger of overfitting to the resampling splits / overtuning!
- The final performance estimate will be optimistically biased.
- One could also see this as a problem similar to multiple testing.

# INSTRUCTIVE AND PROBLEMATIC EXAMPLE

- Assume a binary classification problem with equal class sizes.
- Assume a learner with hyperparameter  $\lambda$ .
- Here, the learner is a (nonsense) feature-independent classifier, where  $\lambda$  has no effect. The learner simply predicts random labels with equal probability.
- Of course, its true generalization error is 50%.
- A cross-validation of the learner (with any fixed  $\lambda$ ) will easily show this (given that the partitioned data set for CV is not too small).
- Now let's "tune" it, by trying out 100 different  $\lambda$  values.
- We repeat this experiment 50 times and average results.

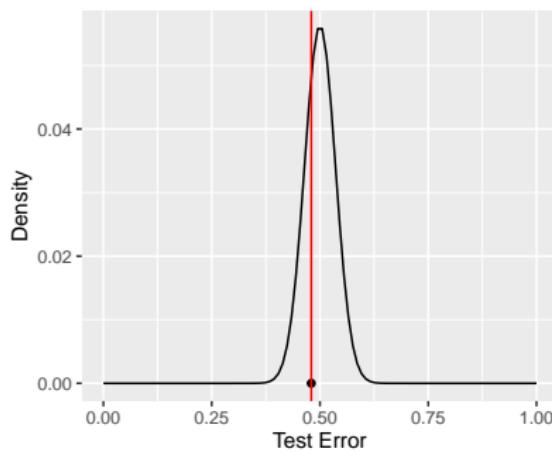
# INSTRUCTIVE AND PROBLEMATIC EXAMPLE



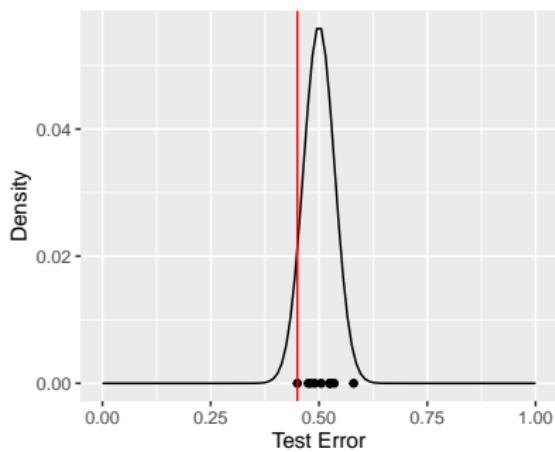
- Plotted is the best “tuning error” (i.e. the performance of the model with fixed  $\lambda$  as evaluated by the cross-validation) after  $k$  tuning iterations.
- We have performed the experiment for different sizes of learning data that were cross-validated.

# INSTRUCTIVE AND PROBLEMATIC EXAMPLE

$n = 200$ ; #runs = 1; best err = 0.48



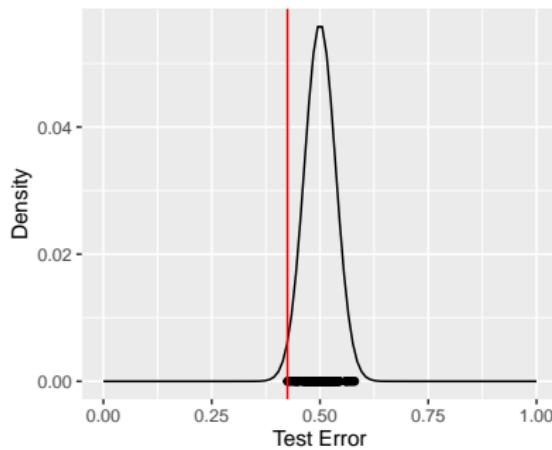
$n = 200$ ; #runs = 10; best err = 0.45



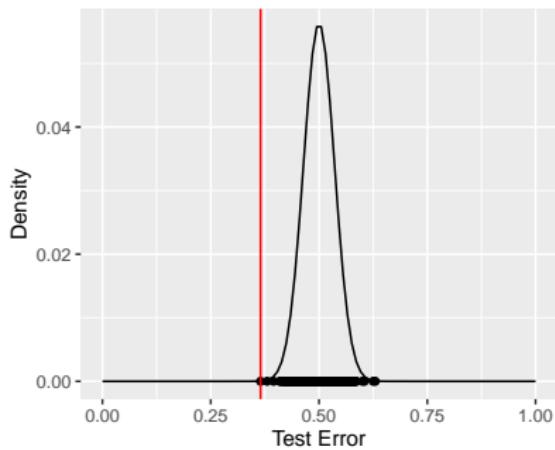
- For 1 experiment, the CV score will be nearly 0.5, as expected
- We basically sample from a (rescaled) binomial distribution when we calculate error rates
- And multiple experiment scores are also nicely arranged around the expected mean 0.5

# INSTRUCTIVE AND PROBLEMATIC EXAMPLE

$n = 200$ ; #runs = 100; best err = 0.42



$n = 200$ ; #runs = 1000; best err = 0.36



- But in tuning we take the minimum of those! So we don't really estimate the "average performance" anymore, we get an estimate of "best case" performance instead.
- The more we sample, the more "biased" this value becomes.

# UNTOUCHED TEST SET PRINCIPLE

Countermeasure: simulate what actually happens in model application.

- All parts of the model building (including model selection, preprocessing) should be embedded in the model-finding process **on the training data**.
- The test set should only be touched once, so we have no way of “cheating”. The test data set is only used once *after* a model is completely trained, after deciding, for example, on specific hyperparameters.  
Only if we do this are the performance estimates we obtained from the test set **unbiased estimates** of the true performance.

# UNTOUCHED TEST SET PRINCIPLE

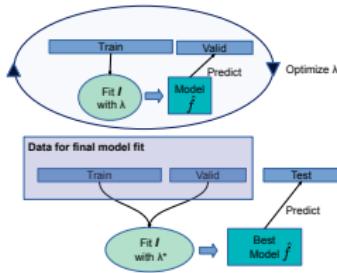
- For steps that themselves require resampling (e.g., hyperparameter tuning) this results in **nested resampling**, i.e., resampling strategies for both
  - tuning: an inner resampling loop to find what works best based on training data
  - outer evaluation on data not used for tuning to get honest estimates of the expected performance on new data

# Introduction to Machine Learning

## Training - Validation - Test

### Learning goals

- Understand how to fulfill the untouched test set principle by a 3-way split of the data
- Understand how thereby the tuning step can be seen as part of a more complex training procedure



# TUNING PROBLEM

Remember:

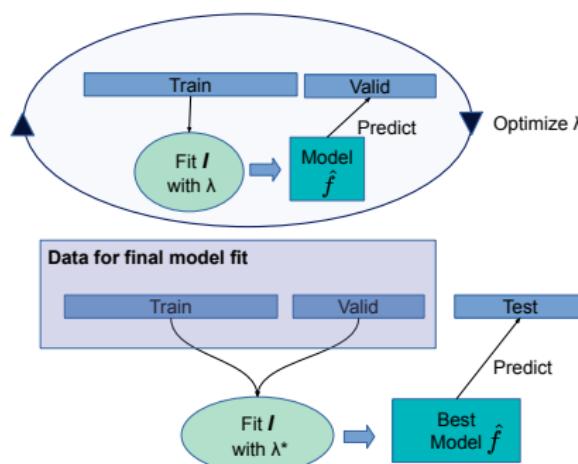
We need to

- **select an optimal learner**
  - without compromising the **accuracy of the performance estimate** for that learner
- for that we need an **untouched test set!**

# TRAIN - VALIDATION - TEST

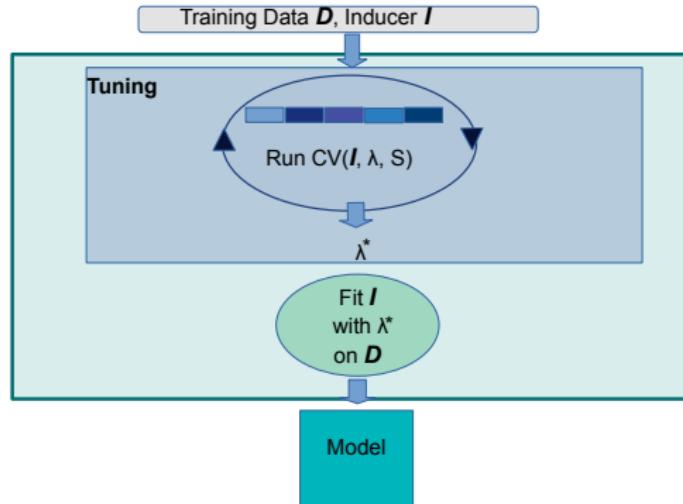
Simplest method to achieve this: a 3-way split

- During tuning, a learner is trained on the **training set**, evaluated on the **validation set**
- After the best model configuration  $\lambda^*$  has been selected, we re-train on the joint (training+validation) set and evaluate the model's performance on the **test set**.



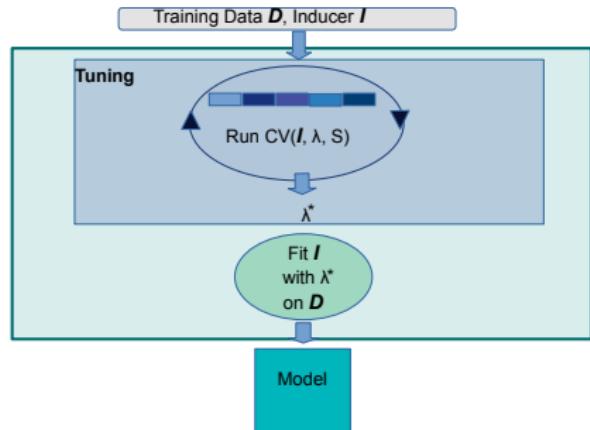
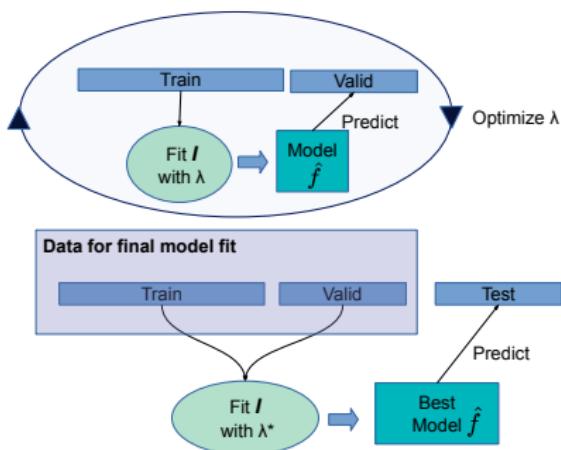
# TUNING AS PART OF MODEL BUILDING

- Effectively, the tuning step is now simply part of a more complex training procedure.
- We could see this as removing the hyperparameters from the inputs of the algorithm and making it “self-tuning”.



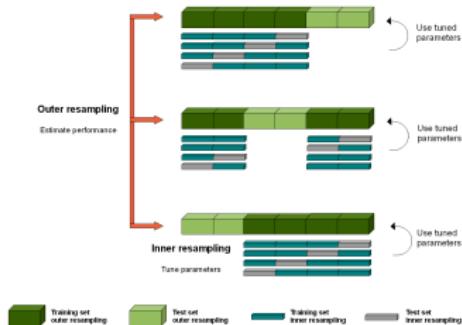
# TUNING AS PART OF MODEL BUILDING

More precisely: the combined training & validation set is actually the training set for the “self-tuning” endowed algorithm.



# Introduction to Machine Learning

## Nested Resampling



### Learning goals

- Understand how the 3-way split of the data can be generalized to nested resampling
- Understand the goal of nested resampling
- Be able to explain how resampling allows to estimate the generalization error

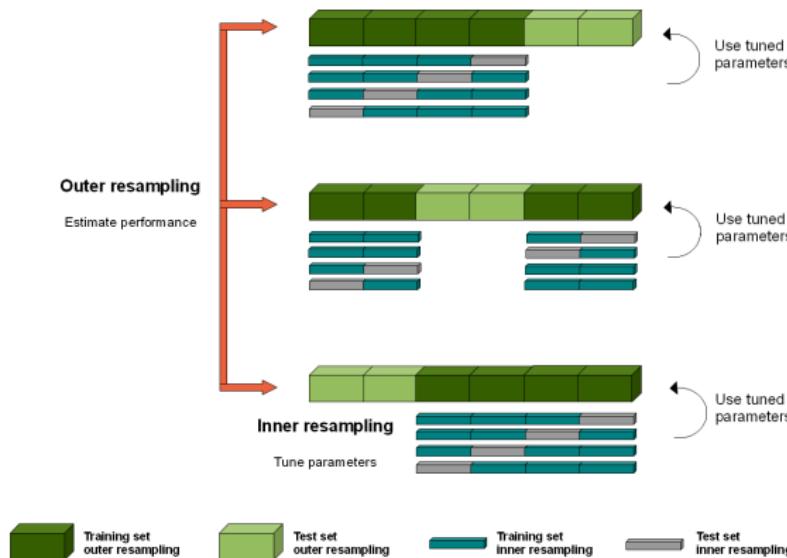
# NESTED RESAMPLING

Just like we can generalize hold-out splitting to resampling to get more reliable estimates of the predictive performance, we can generalize the training/validation/test approach to **nested resampling**.

This results in two nested resampling loops, i.e., resampling strategies for both tuning and outer evaluation.

# NESTED RESAMPLING

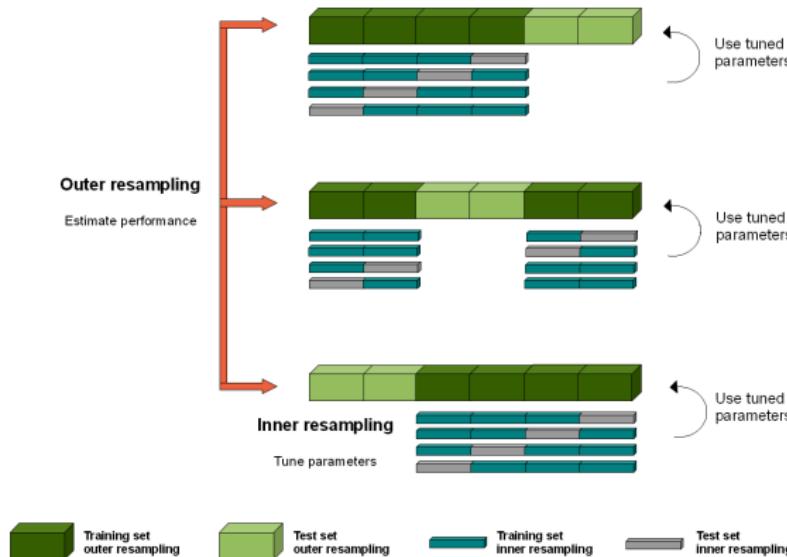
Assume we want to tune over a set of candidate HP configurations  $\lambda_i; i = 1, \dots$  with 4-fold CV in the inner resampling and 3-fold CV in the outer loop. The outer loop is visualized as the light green and dark green parts.



# NESTED RESAMPLING

In each iteration of the outer loop we:

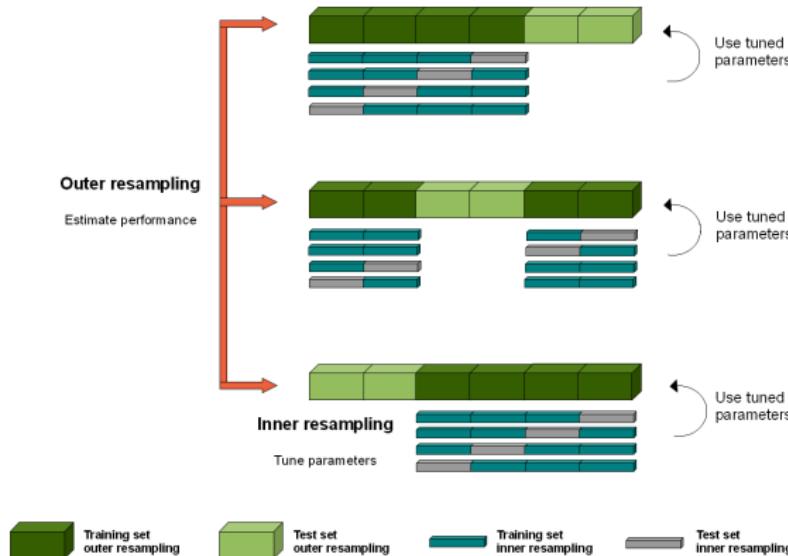
- Split off the light green testing data
- Run the tuner on the dark green part of the data, e.g., evaluate each  $\lambda_i$  through fourfold CV on the dark green part



# NESTED RESAMPLING

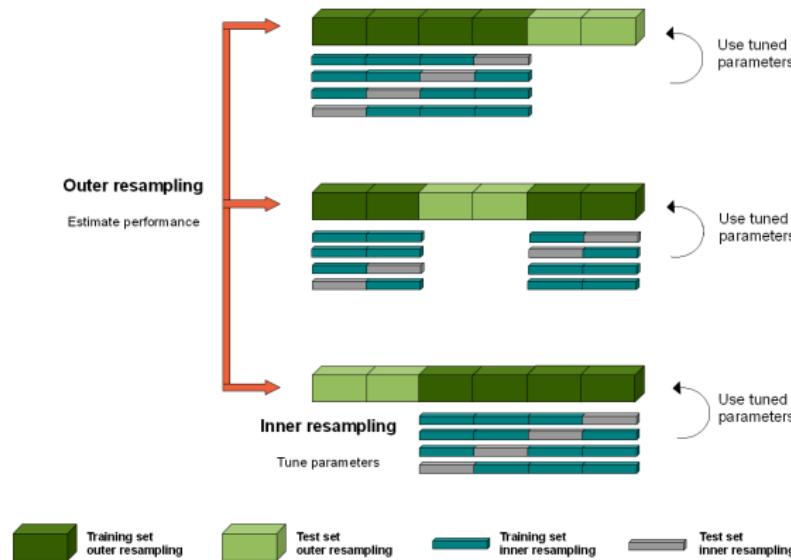
In each iteration of the outer loop we:

- Return the winning  $\lambda^*$  that performed best on the grey inner test sets
- Re-train the model on the full outer dark green train set
- Evaluate it on the outer light green test set



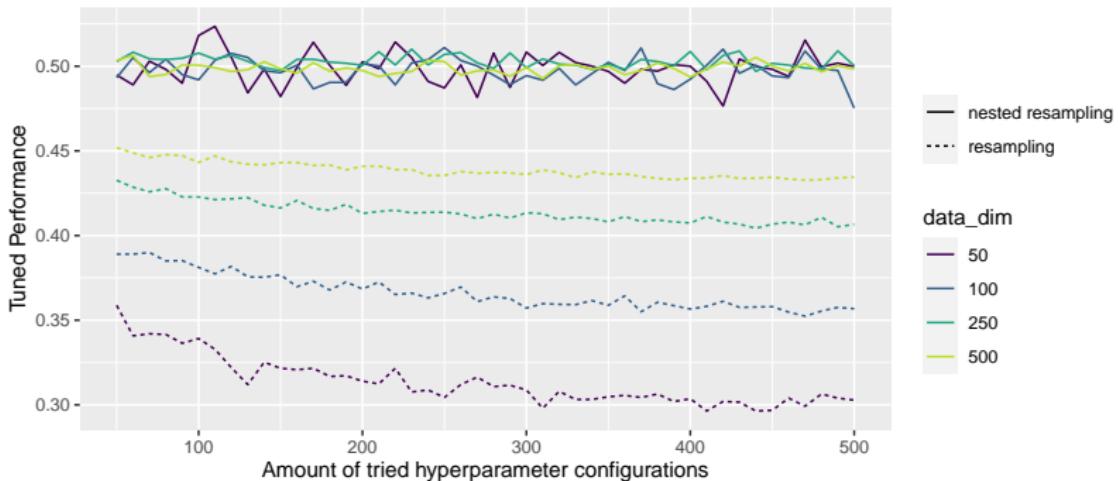
# NESTED RESAMPLING

The error estimates on the outer samples (light green) are unbiased because this data was strictly excluded from the model-building process of the model that was tested on.



# NESTED RESAMPLING - INSTRUCTIVE EXAMPLE

Taking again a look at the motivating example and adding a nested resampling outer loop, we get the expected behavior:



# INTRODUCTION TO MACHINE LEARNING

ML Basics

Supervised Regression

Supervised Classification

k-NN

Performance Evaluation

Classification and Regression Trees (CART)

Random Forests

Tuning

Nested Resampling

**mlr3**

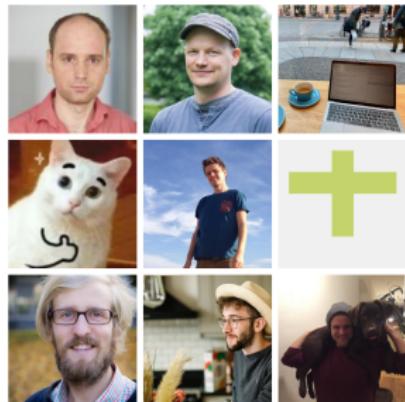
# Modern Machine Learning in R

---



<https://mlr-org.com/>

<https://github.com/mlr-org>



---

**Bernd Bischl, Michel Lang, Martin Binder, Florian Pfisterer, Jakob Richter,  
Patrick Schratz, Lennart Schneider, Raphael Sonabend, Marc Becker**

# **Intro**

# SO YOU WANT TO DO ML IN R

- R gives you access to many machine learning methods

# SO YOU WANT TO DO ML IN R

- R gives you access to many machine learning methods
- ...but without a unified interface

# SO YOU WANT TO DO ML IN R

- R gives you access to many machine learning methods
- ...but without a unified interface
- things like performance evaluation are cumbersome

# SO YOU WANT TO DO ML IN R

- R gives you access to many machine learning methods
- ...but without a unified interface
- things like performance evaluation are cumbersome

Example:

```
# Specify what we want to model in a formula: target ~ features
svm_model = e1071::svm(Species ~ ., data = iris)
```

# SO YOU WANT TO DO ML IN R

- R gives you access to many machine learning methods
- ...but without a unified interface
- things like performance evaluation are cumbersome

Example:

```
# Specify what we want to model in a formula: target ~ features
svm_model = e1071::svm(Species ~ ., data = iris)
```

vs.

```
# Pass the features as a matrix and the target as a vector
xgb_model = xgboost::xgboost(data = as.matrix(iris[1:4]),
                             label = iris$Species, nrounds = 10)
```

# SO YOU WANT TO DO ML IN R

```
library("mlr3")
```

Ingredients:

- Data / Task
- Learning Algorithms
- Performance Evaluation
- Performance Comparison

**R6**

## R6 – ALL YOU NEED TO KNOW

mlr3 uses the *R6* class system. Some things may seem unusual if you see them for the first time.

- *Objects* are created using `<Class>$new()`.

```
task = TaskClassif$new("iris", iris, "Species")
```

## R6 – ALL YOU NEED TO KNOW

mlr3 uses the *R6* class system. Some things may seem unusual if you see them for the first time.

- *Objects* are created using `<Class>$new()`.

```
task = TaskClassif$new("iris", iris, "Species")
```

- Objects have *fields* that contain information about the object.

```
task$nrow  
#> [1] 150
```

# R6 – ALL YOU NEED TO KNOW

mlr3 uses the *R6* class system. Some things may seem unusual if you see them for the first time.

- *Objects* are created using `<Class>$new()`.

```
task = TaskClassif$new("iris", iris, "Species")
```

- Objects have *fields* that contain information about the object.

```
task$nrow  
#> [1] 150
```

- Objects have *methods* that are called like functions:

```
task$filter(rows = 1:10)
```

# R6 – ALL YOU NEED TO KNOW

mlr3 uses the *R6* class system. Some things may seem unusual if you see them for the first time.

- *Objects* are created using `<Class>$new()`.

```
task = TaskClassif$new("iris", iris, "Species")
```

- Objects have *fields* that contain information about the object.

```
task$nrow  
#> [1] 150
```

- Objects have *methods* that are called like functions:

```
task$filter(rows = 1:10)
```

- Methods may change (“mutate”) the object (reference semantics)!

```
task$nrow  
#> [1] 10
```

# R6 AND ACTIVE BINDINGS

Some fields of R6-objects may be “*Active Bindings*”. Internally they are realized as functions that are called whenever the value is set or retrieved.

- Active bindings for read-only fields

```
task$nrow = 11  
#> Error: Field/Binding is read-only
```

# R6 AND ACTIVE BINDINGS

Some fields of R6-objects may be “*Active Bindings*”. Internally they are realized as functions that are called whenever the value is set or retrieved.

- Active bindings for read-only fields

```
task$nrow = 11  
#> Error: Field/Binding is read-only
```

- Active bindings for argument checking

```
task$properties = NULL  
#> Error in assert_set(rhs, .var.name = "properties"):  
Assertion on 'properties' failed: Must be of type  
'character', not 'NULL'.  
task$properties = c("property1", "property2") # works
```

# MLR3 PHILOSOPHY

- Overcome limitations of S3 with the help of **R6**
  - Truly object-oriented: data and methods live in the same object
  - Make use of inheritance
  - Reference semantics

# MLR3 PHILOSOPHY

- Overcome limitations of S3 with the help of **R6**
  - Truly object-oriented: data and methods live in the same object
  - Make use of inheritance
  - Reference semantics
- Embrace **data.table**, both for arguments and internally
  - Fast operations for tabular data
  - List columns to arrange complex objects in tabular structure

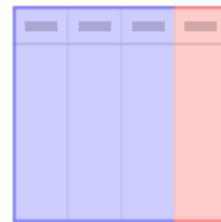
# MLR3 PHILOSOPHY

- Overcome limitations of S3 with the help of **R6**
  - Truly object-oriented: data and methods live in the same object
  - Make use of inheritance
  - Reference semantics
- Embrace **data.table**, both for arguments and internally
  - Fast operations for tabular data
  - List columns to arrange complex objects in tabular structure
- Be **light on dependencies**:
  - R6, data.table, lgr, uuid, mlbench, digest
  - Plus some of our own packages (backports, checkmate, ...)

# **Data**

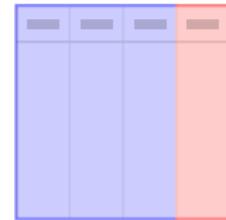
# DATA

- Tabular data



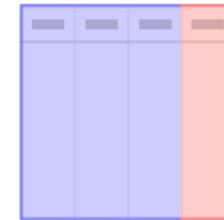
# DATA

- Tabular data
- Features



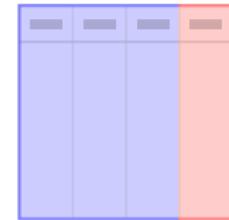
# DATA

- Tabular data
- Features
- Target / outcome to predict



# DATA

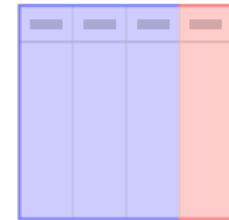
- Tabular data
- Features
- Target / outcome to predict
  - discrete for classification
  - continuous for regression



# DATA

- Tabular data
- Features
- Target / outcome to predict
  - discrete for classification
  - continuous for regression

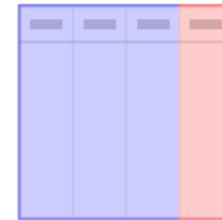
⇒ target determines the machine learning “Task”



# DATA

- Tabular data
- Features
- Target / outcome to predict
  - discrete for classification
  - continuous for regression

⇒ target determines the machine learning “Task”



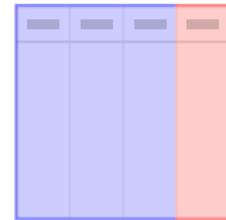
```
print(iris) # included in R

#>   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#> 1       5.1        3.5       1.4        0.2  setosa
#> 2       4.9        3.0       1.4        0.2  setosa
#> ...
```

# DATA

- Tabular data
- Features
- Target / outcome to predict
  - discrete for classification
  - continuous for regression

⇒ target determines the machine learning “Task”



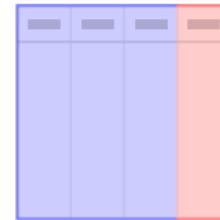
```
print(iris) # included in R
```

```
#> Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#> 1           5.1        3.5       1.4        0.2 setosa
#> 2           4.9        3.0       1.4        0.2 setosa
#> ...
```

# DATA

- Tabular data
- Features
- Target / outcome to predict
  - discrete for classification
  - continuous for regression

⇒ target determines the machine learning “Task”



```
print(iris) # included in R
```

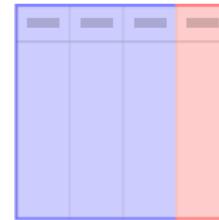
```
#> Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#> 1           5.1        3.5       1.4        0.2 setosa
#> 2           4.9        3.0       1.4        0.2 setosa
#> ...
```

```
task = TaskClassif$new("iris", iris, "Species")
```

# DATA

- Tabular data
- Features
- Target / outcome to predict
  - discrete for classification
  - continuous for regression

⇒ target determines the machine learning “Task”



```
print(iris) # included in R
```

```
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#> 1         5.1        3.5       1.4        0.2 setosa
#> 2         4.9        3.0       1.4        0.2 setosa
#> ...
```

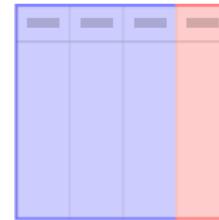
Task ID

```
task = TaskClassif$new("iris", iris, "Species")
```

# DATA

- Tabular data
- Features
- Target / outcome to predict
  - discrete for classification
  - continuous for regression

⇒ target determines the machine learning “Task”



```
print(iris) # included in R
```

```
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#> 1         5.1        3.5       1.4        0.2 setosa
#> 2         4.9        3.0       1.4        0.2 setosa
#> ...
```

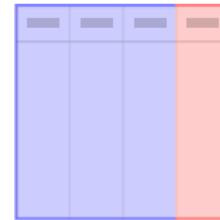
Task ID      data  
↓            ↓

```
task = TaskClassif$new("iris", iris, "Species")
```

# DATA

- Tabular data
- Features
- Target / outcome to predict
  - discrete for classification
  - continuous for regression

⇒ target determines the machine learning “Task”



```
print(iris) # included in R
```

```
#> Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#> 1           5.1        3.5       1.4        0.2 setosa
#> 2           4.9        3.0       1.4        0.2 setosa
#> ...
```

Task ID      data      target name  
↓            ↓          ↓

```
task = TaskClassif$new("iris", iris, "Species")
```

# DATA

```
task = TaskClassif$new("iris", iris, "Species")
```

```
print(task)

# <TaskClassif:iris> (150 x 5)
# * Target: Species
# * Properties: multiclass
# * Features (4):
#   - dbl (4): Petal.Length, Petal.Width, Sepal.Length,
#     Sepal.Width
```

```
task$ncol
task$nrow
task$feature_names
task$target_names
```

```
task$head(n = )
task$truth(row_ids = )
task$data(rows = ,
          cols = )
```

```
task$select(cols = )
task$filter(rows = )
task$cbind(data = )
task$rbind(data = )
```

# **Dictionaries**

# DICTIONARIES

- Ordinary constructors: `TaskClassif$new()` /  
`LearnerClassifRpart$new()`

# DICTIONARIES

- Ordinary constructors: `TaskClassif$new()` /  
`LearnerClassifRpart$new()`
- ⇒ `mlr3` offers *Short Form Constructors* that are less verbose

# DICTIONARIES

- Ordinary constructors: `TaskClassif$new()` /  
`LearnerClassifRpart$new()`  
⇒ `mlr3` offers *Short Form Constructors* that are less verbose
- They access Dictionary of objects:

# DICTIONARIES

- Ordinary constructors: `TaskClassif$new()` /  
`LearnerClassifRpart$new()`

⇒ `mlr3` offers *Short Form Constructors* that are less verbose

- They access Dictionary of objects:

Object	Dictionary	Short Form
Task	<code>mlr_tasks</code>	<code>tsk()</code>
Learner	<code>mlr_learners</code>	<code>lrn()</code>
Measure	<code>mlr_measures</code>	<code>msr()</code>
Resampling	<code>mlr_resamplings</code>	<code>rsmp()</code>

Dictionaries can get populated by add-on packages (e.g. `mlr3learners`)

# DICTIONARIES

```
# list items
tsk()

#> <DictionaryTask> with 10 stored values
#> Keys: boston_housing, breast_cancer, german_credit, iris,
#> mtcars, pima, sonar, spam, wine, zoo

# retrieve object
tsk("iris")

#> <TaskClassif:iris> (150 x 5)
#> * Target: Species
#> * Properties: multiclass
#> * Features (4):
#>   - dbl (4): Petal.Length, Petal.Width, Sepal.Length,
#>     Sepal.Width
```

# SHORT FORMS AND DICTIONARIES

`as.data.table(<DICTIONARY>)` creates a `data.table` with metadata about objects in dictionaries:

```
mlr_learners_table = as.data.table(mlr_learners)

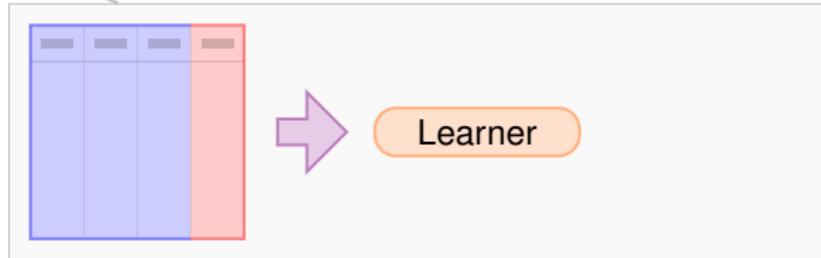
mlr_learners_table[1:10, c("key", "packages", "predict_types")]

#                               key packages predict_types
# 1:    classif.cv_glmnet    glmnet response,prob
# 2:    classif.debug        response,prob
# 3: classif.featureless     response,prob
# 4:    classif.glmnet       glmnet response,prob
# 5:    classif.kknn          kknn  response,prob
# 6:    classif.lda           MASS  response,prob
# 7:    classif.log_reg       stats  response,prob
# 8:    classif.multinom      nnet  response,prob
# 9: classif.naive_bayes     e1071 response,prob
# 10:   classif.nnet          nnet  prob,response
```

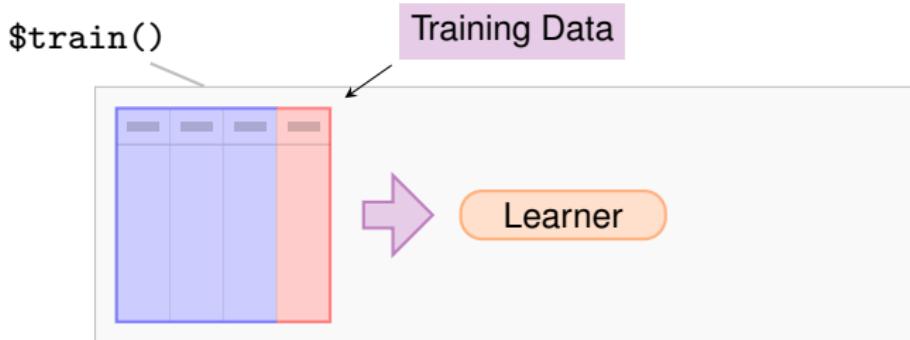
# **Learning Algorithms**

# LEARNING ALGORITHMS

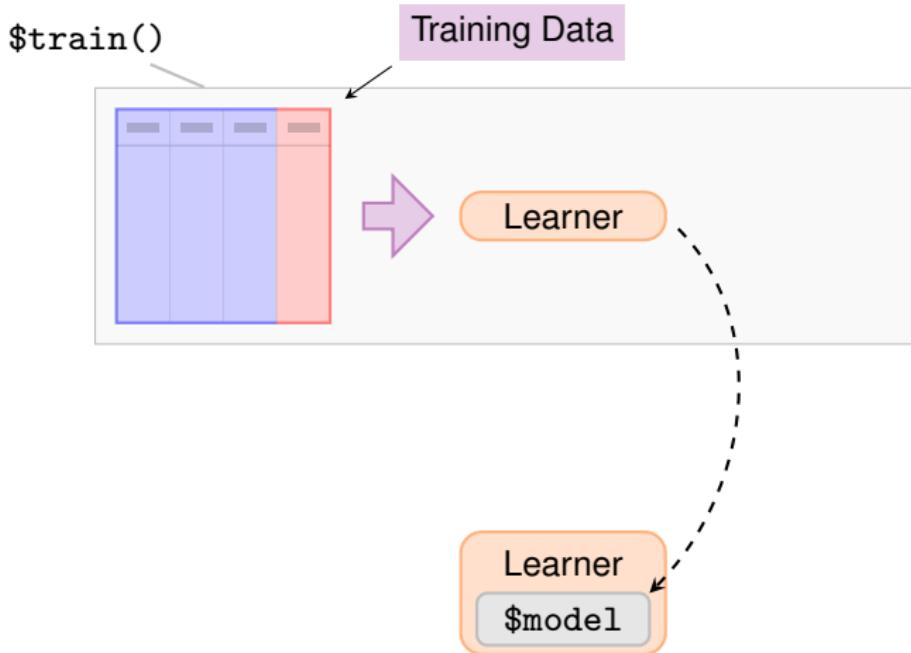
\$train()



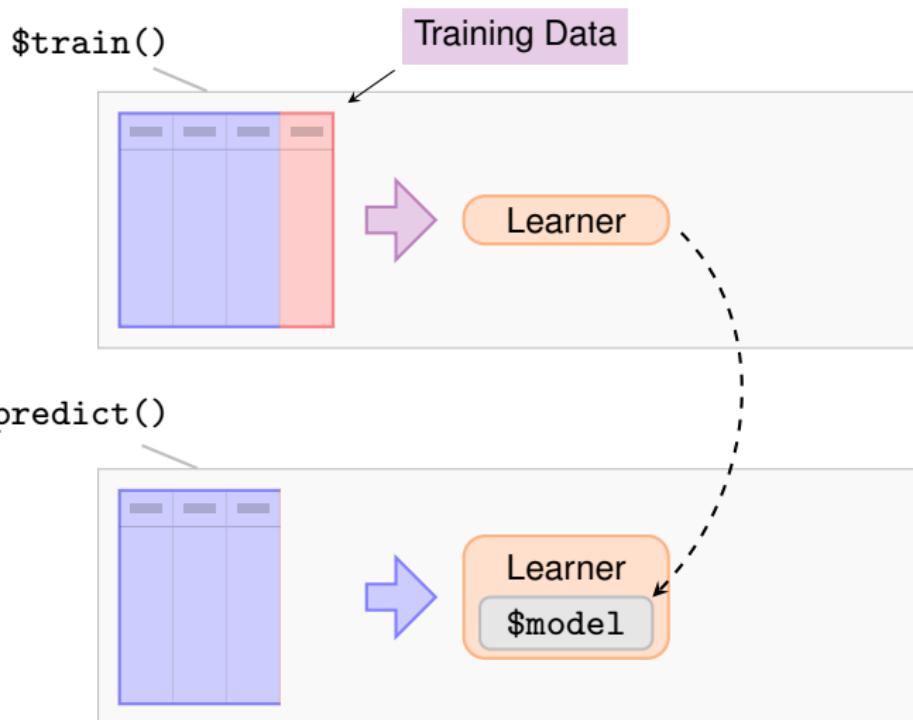
# LEARNING ALGORITHMS



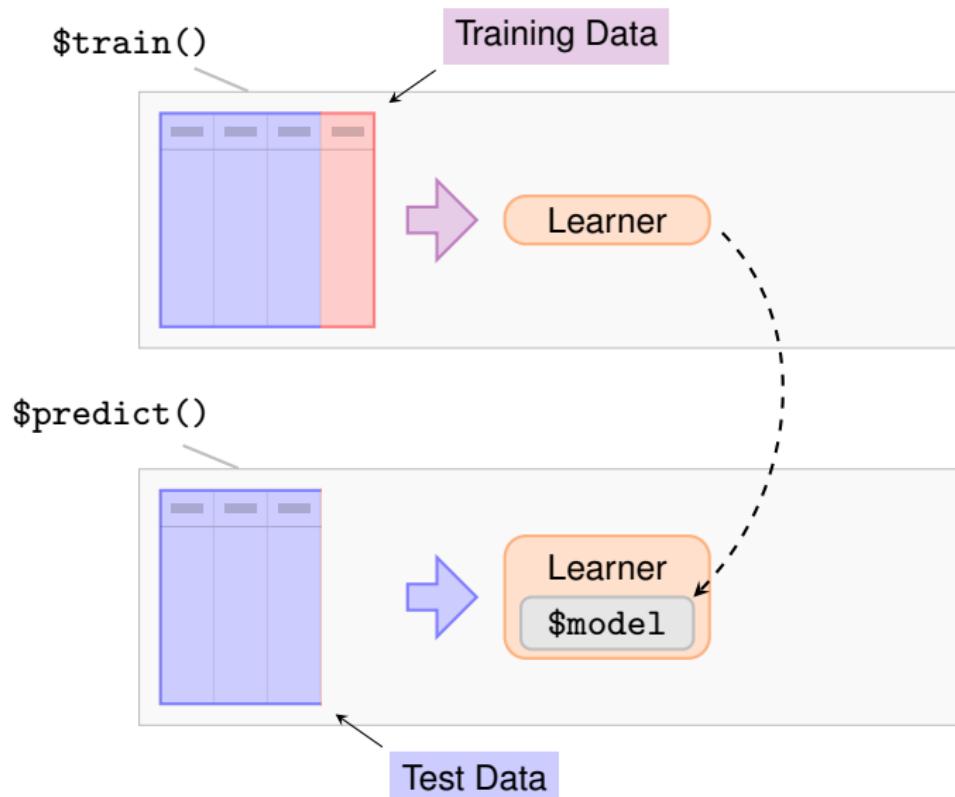
# LEARNING ALGORITHMS



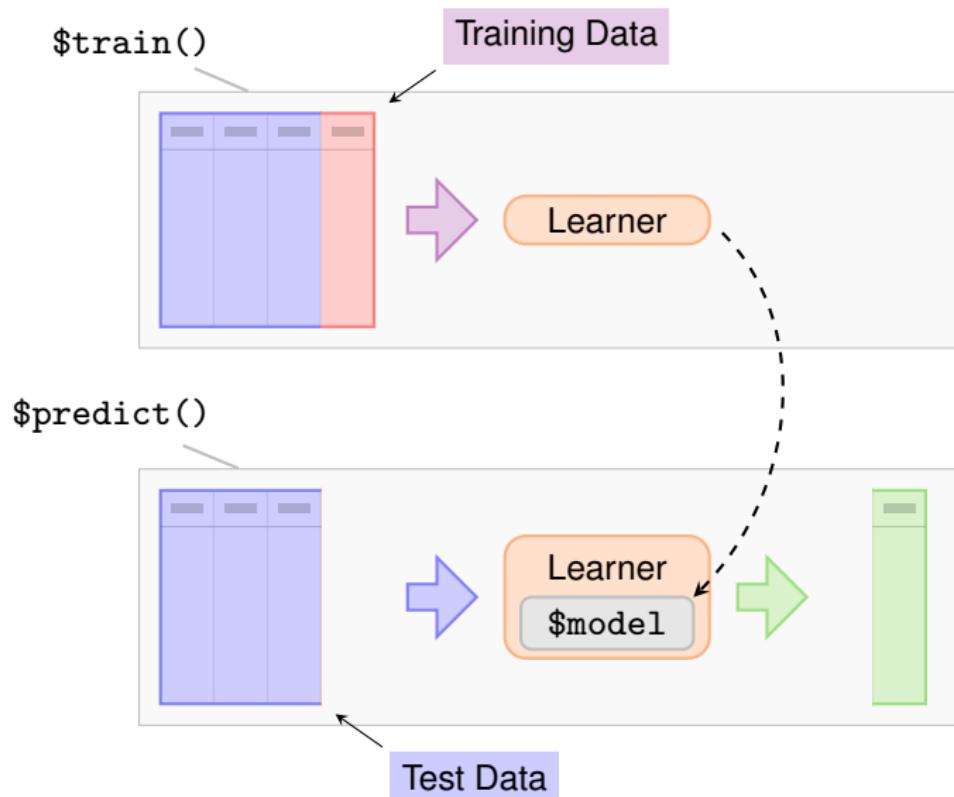
# LEARNING ALGORITHMS



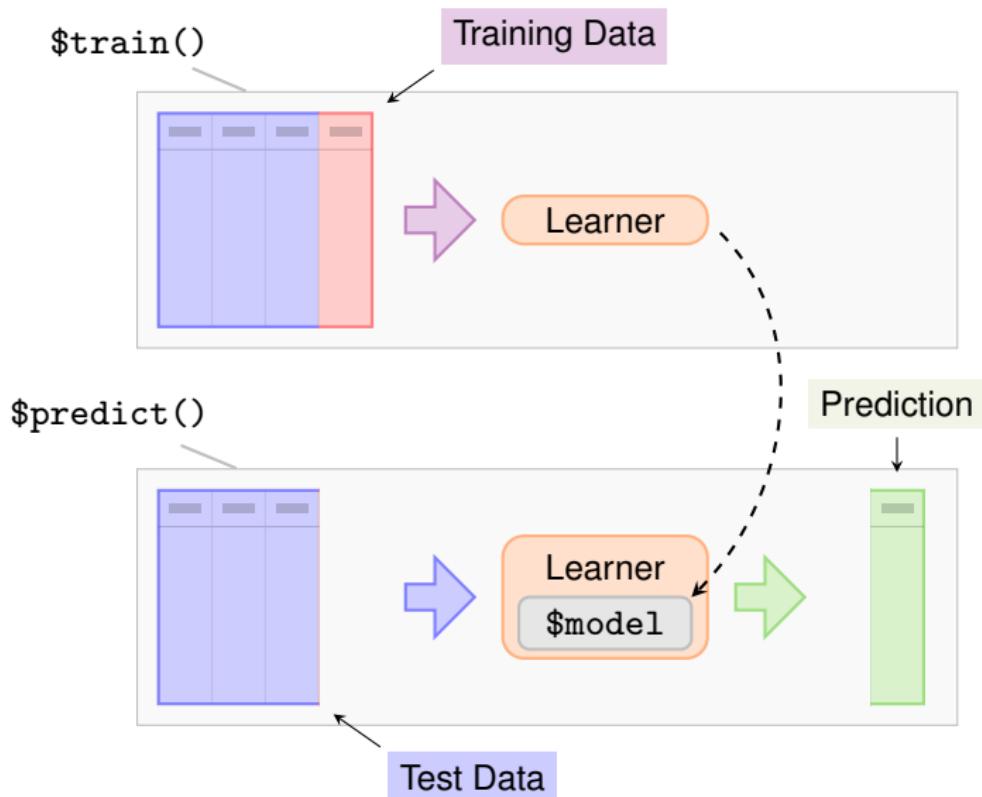
# LEARNING ALGORITHMS



# LEARNING ALGORITHMS



# LEARNING ALGORITHMS



# LEARNING ALGORITHMS

- Get a Learner provided by `mlr`

```
learner = lrn("classif.rpart")
```

# LEARNING ALGORITHMS

- Get a Learner provided by `mlr`

```
learner = lrn("classif.rpart")
```

- Train the Learner

```
learner$train(task)
```

# LEARNING ALGORITHMS

- Get a Learner provided by mlr

```
learner = lrn("classif.rpart")
```

- Train the Learner

```
learner$train(task)
```

- The \$model is the rpart model: a decision tree

```
print(learner$model)

#> n= 150
#>
#> node), split, n, loss, yval, (yprob)
#>      * denotes terminal node
#>
#> 1) root 150 100 setosa (0.333 0.333 0.333)
#>    2) Petal.Length< 2.4 50  0 setosa (1.000 0.000 0.000) *
#>    3) Petal.Length>=2.4 100  50 versicolor (0.000 0.500 0.500)
#>      6) Petal.Width< 1.8 54   5 versicolor (0.000 0.907 0.093) *
#>      7) Petal.Width>=1.8 46   1 virginica (0.000 0.022 0.978) *
```

# HYPERPARAMETERS

- Learners have *hyperparameters*

```
as.data.table(learner$param_set)[, 1:6]
```

#>		id	class	lower	upper	levels	nlevels
#> 1:		minsplit	ParamInt	1	Inf		Inf
#> 2:		minbucket	ParamInt	1	Inf		Inf
#> 3:		cp	ParamDbl	0	1		Inf
#> 4:		maxcompete	ParamInt	0	Inf		Inf
#> 5:		maxsurrogate	ParamInt	0	Inf		Inf
#> 6:		maxdepth	ParamInt	1	30		30
#> 7:		usesurrogate	ParamInt	0	2		3
#> 8:		surrogatestyle	ParamInt	0	1		2
#> 9:		xval	ParamInt	0	Inf		Inf
#> 10:		keep_model	ParamLgl	NA	NA	TRUE, FALSE	2

# HYPERPARAMETERS

- Learners have *hyperparameters*

```
as.data.table(learner$param_set)[, 1:6]
```

#>		id	class	lower	upper	levels	nlevels
#> 1:		minsplit	ParamInt	1	Inf		Inf
#> 2:		minbucket	ParamInt	1	Inf		Inf
#> 3:		cp	ParamDbl	0	1		Inf
#> 4:		maxcompete	ParamInt	0	Inf		Inf
#> 5:		maxsurrogate	ParamInt	0	Inf		Inf
#> 6:		maxdepth	ParamInt	1	30		30
#> 7:		usesurrogate	ParamInt	0	2		3
#> 8:		surrogatestyle	ParamInt	0	1		2
#> 9:		xval	ParamInt	0	Inf		Inf
#> 10:		keep_model	ParamLgl	NA	NA	TRUE, FALSE	2

- Changing them changes the Learner behavior

```
learner$param_set$values = list(maxdepth = 1, xval = 0)
```

```
learner$train(task)
```

# HYPERPARAMETERS

- This gives a smaller decision tree

```
print(learner$model)

#> n= 150
#>
#> node), split, n, loss, yval, (yprob)
#>       * denotes terminal node
#>
#> 1) root 150 100 setosa (0.33 0.33 0.33)
#>    2) Petal.Length< 2.4 50    0 setosa (1.00 0.00 0.00) *
#>    3) Petal.Length>=2.4 100   50 versicolor (0.00 0.50 0.50) *
```

# PREDICTION

- Let's make a prediction for some new data, e.g.:

```
new_data
```

```
#   Sepal.Length Sepal.Width Petal.Length Petal.Width
# 1          4         3          2          1
# 2          2         2          3          2
```

# PREDICTION

- Let's make a prediction for some new data, e.g.:

```
new_data  
#   Sepal.Length Sepal.Width Petal.Length Petal.Width  
# 1          4         3          2          1  
# 2          2         2          3          2
```

- To do so, we call the `$predict_newdata()` method using the new data:

```
prediction = learner$predict_newdata(new_data)
```

# PREDICTION

- Let's make a prediction for some new data, e.g.:

```
new_data
#   Sepal.Length Sepal.Width Petal.Length Petal.Width
# 1           4         3          2          1
# 2           2         2          3          2
```

- To do so, we call the `$predict_newdata()` method using the new data:

```
prediction = learner$predict_newdata(new_data)
```

- We get a `Prediction` object:

```
prediction
#> <PredictionClassif> for 2 observations:
#>   row_id truth    response
#>     1  <NA>    setosa
#>     2  <NA> versicolor
```

# PREDICTION

- Let's make a prediction for some new data, e.g.:

```
new_data  
# Sepal.Length Sepal.Width Petal.Length Petal.Width  
# 1 4 3 2 1  
# 2 2 2 3 2
```

- To do so, we call the `$predict_newdata()` method using the new data:

```
prediction = learner$predict_newdata(new_data)
```

- We get a `Prediction` object:

```
prediction  
#> <PredictionClassif> for 2 observations:  
#>   row_id truth    response  
#>   1 <NA>      setosa  
#>   2 <NA> versicolor
```

# PREDICTION

- Let's make a prediction for some new data, e.g.:

```
new_data  
# Sepal.Length Sepal.Width Petal.Length Petal.Width  
# 1 4 3 2 1  
# 2 2 2 3 2
```

- To do so, we call the `$predict_newdata()` method using the new data:

```
prediction = learner$predict_newdata(new_data)
```

- We get a `Prediction` object:

```
prediction  
#> <PredictionClassif> for 2 observations:  
#>   row_id truth  response  
#>   1 <NA>    setosa  
#>   2 <NA> versicolor
```

# PREDICTION

- We can make the Learner predict *probabilities* when we set `predict_type`:

```
learner$predict_type = "prob"  
learner$predict_newdata(new_data)  
  
# <PredictionClassif> for 2 observations:  
#   row_id truth    response prob.setosa prob.versicolor  
#     1 <NA>      setosa          1          0.0  
#     2 <NA> versicolor          0          0.5  
#   prob.virginica  
#     0.0  
#     0.5
```

# PREDICTION

What exactly is a Prediction object?

- Contains predictions and offers useful access fields / methods

# PREDICTION

What exactly is a Prediction object?

- Contains predictions and offers useful access fields / methods
- ⇒ Use `as.data.table()` to extract data

```
as.data.table(prediction)
#>   row_id truth    response
#> 1:      1 <NA>     setosa
#> 2:      2 <NA> versicolor
```

# PREDICTION

What exactly is a Prediction object?

- Contains predictions and offers useful access fields / methods
- ⇒ Use `as.data.table()` to extract data

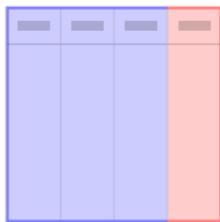
```
as.data.table(prediction)
#>   row_id truth    response
#> 1:      1 <NA>    setosa
#> 2:      2 <NA> versicolor
```

- ⇒ Active bindings and functions that give further information:  
`$response`, `$truth`, ...

```
prediction$response
#> [1] setosa    versicolor
#> Levels: setosa versicolor virginica
```

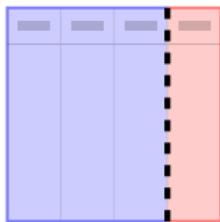
# **Performance**

# PERFORMANCE EVALUATION



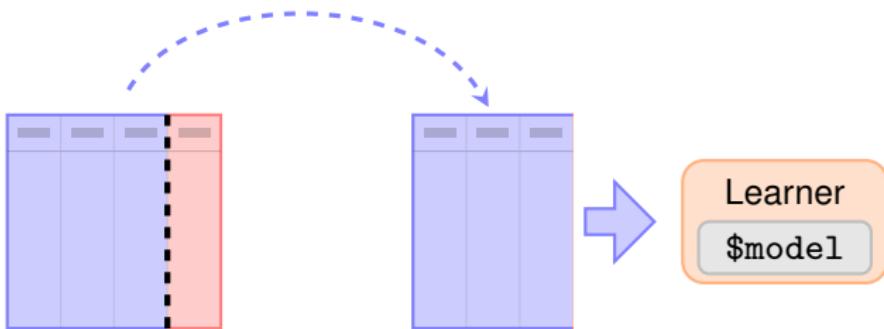
Learner  
\$model

# PERFORMANCE EVALUATION

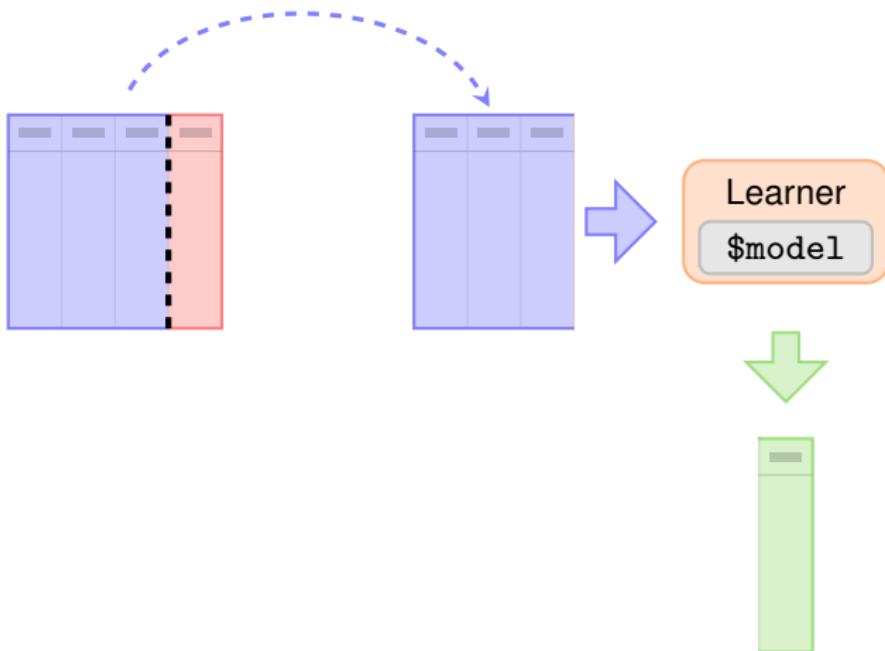


Learner  
\$model

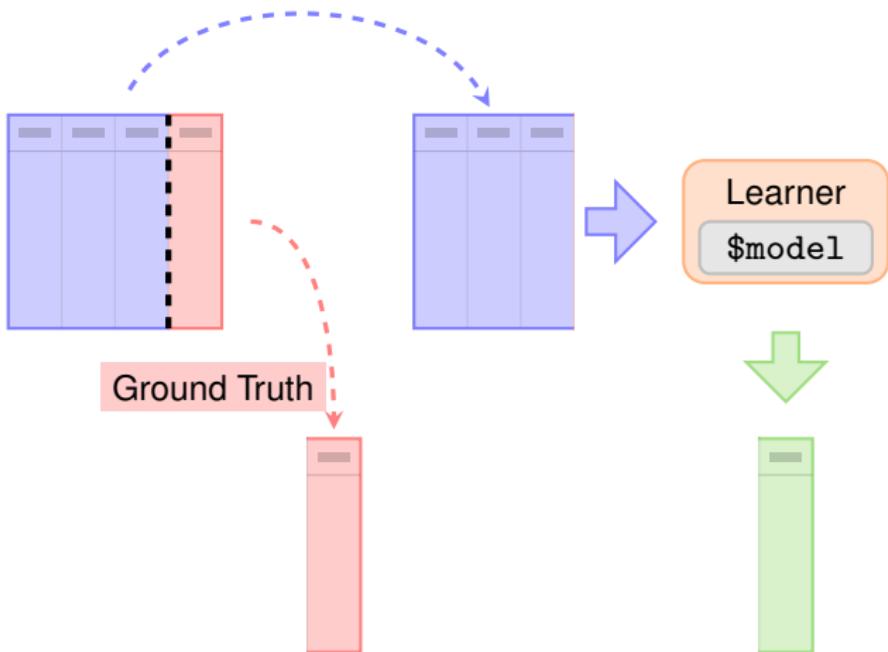
# PERFORMANCE EVALUATION



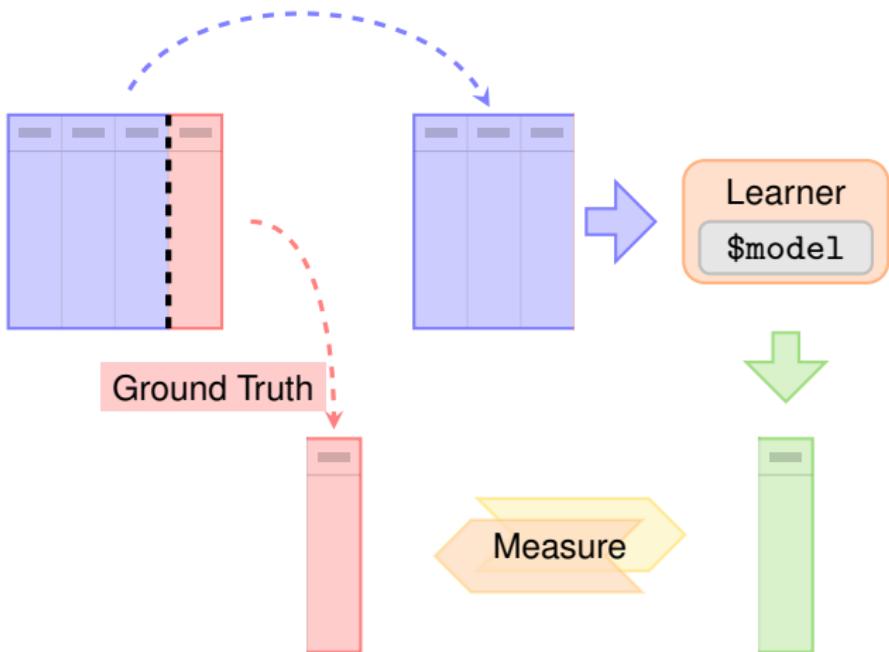
# PERFORMANCE EVALUATION



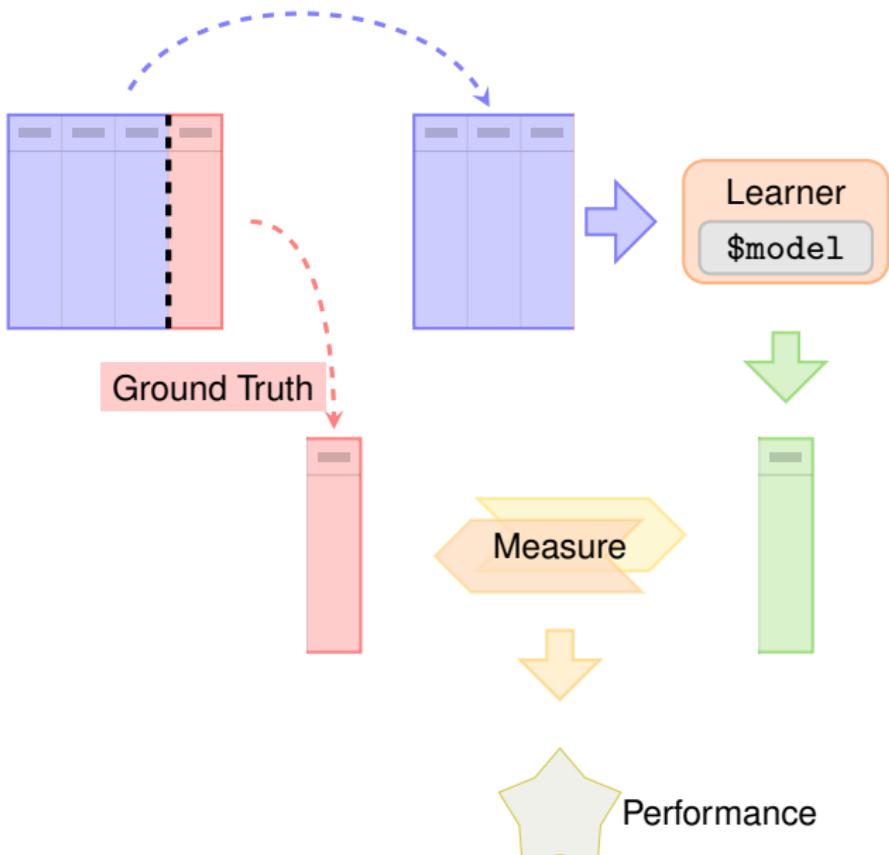
# PERFORMANCE EVALUATION



# PERFORMANCE EVALUATION



# PERFORMANCE EVALUATION



# PERFORMANCE EVALUATION

- Prediction ‘Task’ with known data

```
known_truth_task$data()  
#   Species Petal.Length Petal.Width Sepal.Length Sepal.Width  
# 1: setosa      2         1         4         3  
# 2: setosa      3         2         2         2
```

# PERFORMANCE EVALUATION

- Prediction ‘Task’ with known data

```
known_truth_task$data()  
#   Species Petal.Length Petal.Width Sepal.Length Sepal.Width  
# 1: setosa          2           1          4          3  
# 2: setosa          3           2          2          2
```

- Predict again

```
pred = learner$predict(known_truth_task)  
pred  
#> <PredictionClassif> for 2 observations:  
#>   row_id  truth  response  
#>     1 setosa    setosa  
#>     2 setosa virginica
```

# PERFORMANCE EVALUATION

- Prediction ‘Task’ with known data

```
known_truth_task$data()  
#   Species Petal.Length Petal.Width Sepal.Length Sepal.Width  
# 1: setosa          2           1           4           3  
# 2: setosa          3           2           2           2
```

- Predict again

```
pred = learner$predict(known_truth_task)  
pred  
#> <PredictionClassif> for 2 observations:  
#>   row_id  truth  response  
#>     1 setosa    setosa  
#>     2 setosa virginica
```

- Score the prediction

```
pred$score(msr("classif.ce"))  
#> classif.ce  
#>     0.5
```

# PERFORMANCE EVALUATION

- Prediction ‘Task’ with known data

```
known_truth_task$data()  
#   Species Petal.Length Petal.Width Sepal.Length Sepal.Width  
# 1: setosa          2           1           4           3  
# 2: setosa          3           2           2           2
```

- Predict again

```
pred = learner$predict(known_truth_task)  
pred  
#> <PredictionClassif> for 2 observations:  
#>   row_id  truth  response  
#>     1 setosa    setosa  
#>     2 setosa virginica
```

- Score the prediction

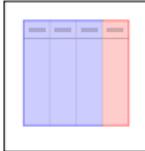
```
pred$score(msr("classif.ce"))  
#> classif.ce  
#>      0.5
```

# **Outro**

# OVERVIEW

Ingredients:

Data



TaskClassif,  
TaskRegr,  
tsk()

Learning Algorithms

Learner



Learner  
\$model

lrn() ⇒ Learner,  
↪ Learner\$train(),  
↪ Learner\$predict() ⇒ Prediction

Measure Performance

Measure



Prediction\$score(),  
msr() ⇒ Measure

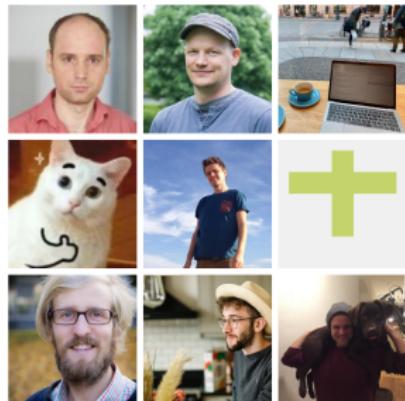
# Modern Machine Learning in R

---



<https://mlr-org.com/>

<https://github.com/mlr-org>

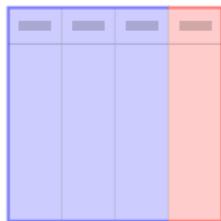


---

**Bernd Bischl, Michel Lang, Martin Binder, Florian Pfisterer, Jakob Richter,  
Patrick Schratz, Lennart Schneider, Raphael Sonabend, Marc Becker**

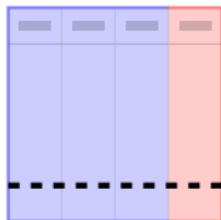
# Resampling

# RESAMPLING



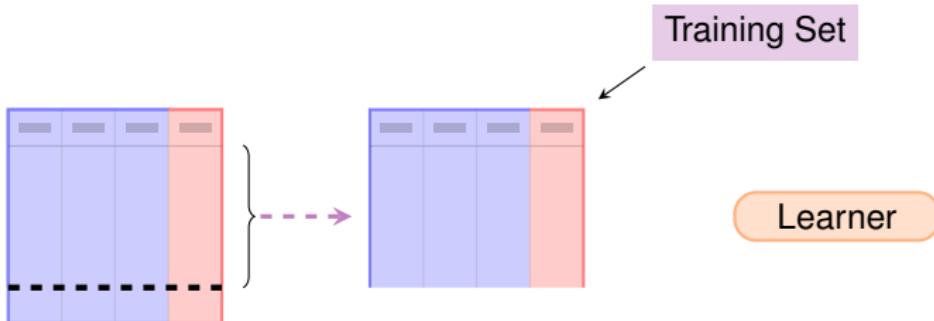
Learner

# RESAMPLING

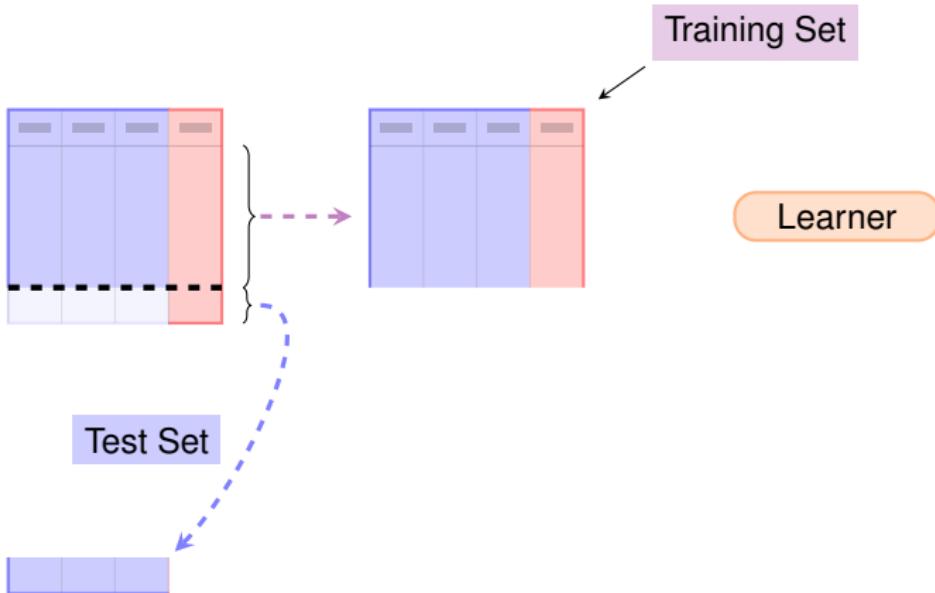


Learner

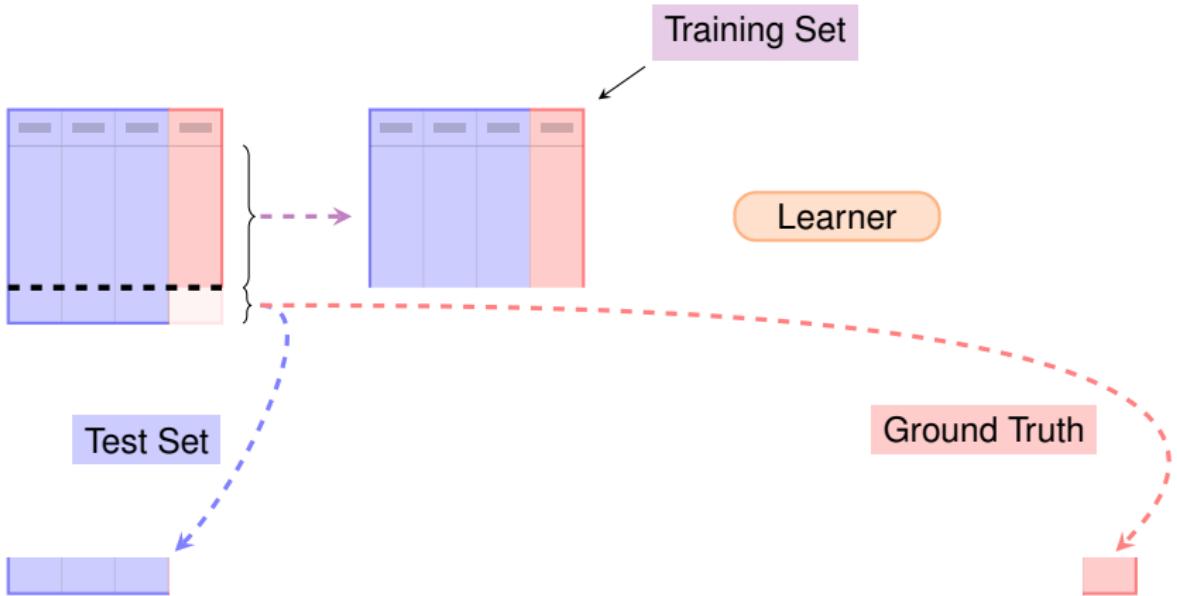
# RESAMPLING



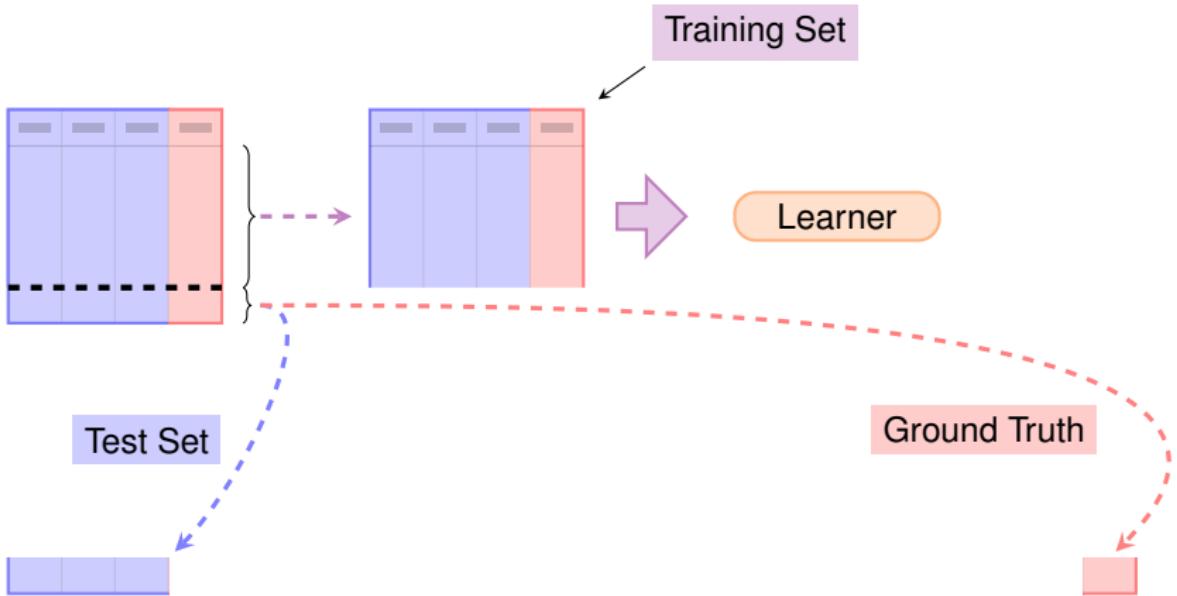
# RESAMPLING



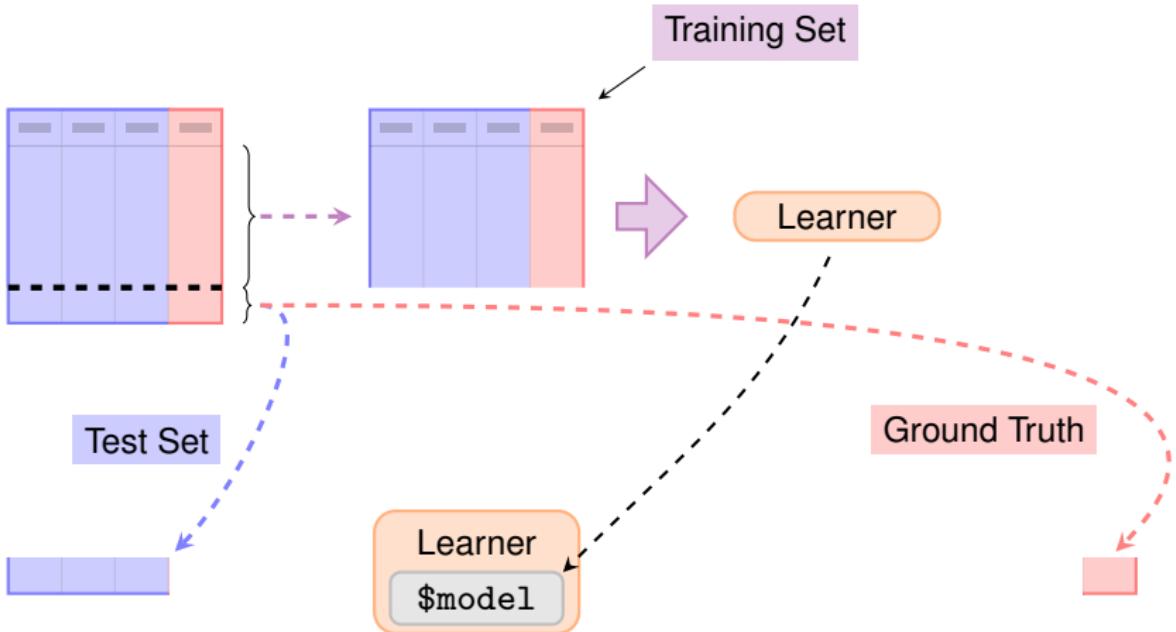
# RESAMPLING



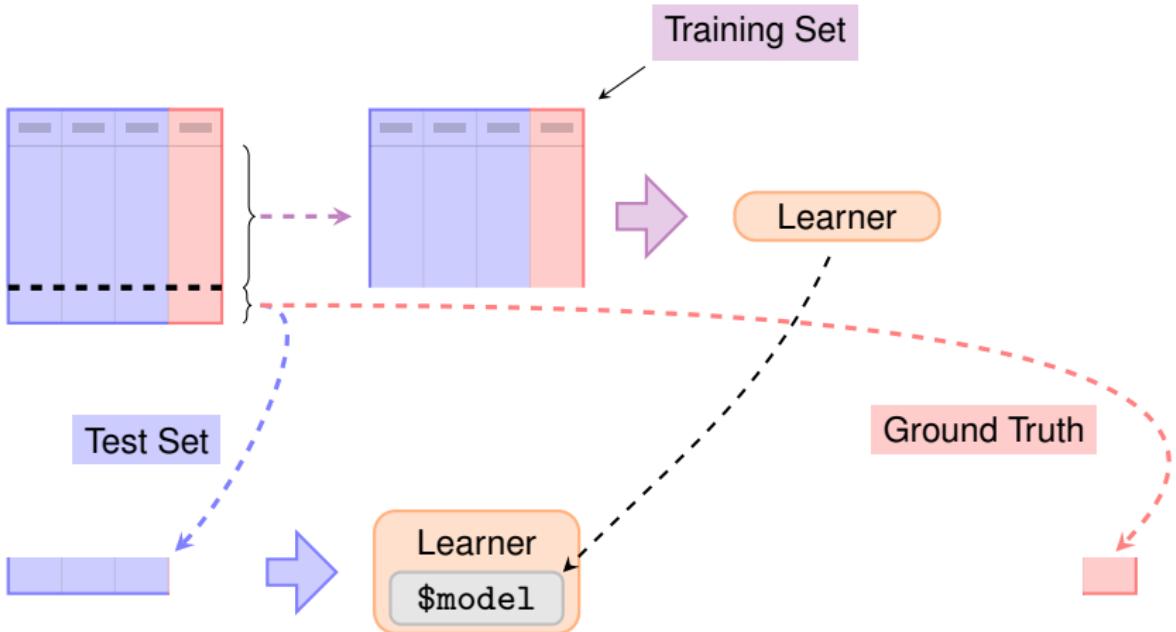
# RESAMPLING



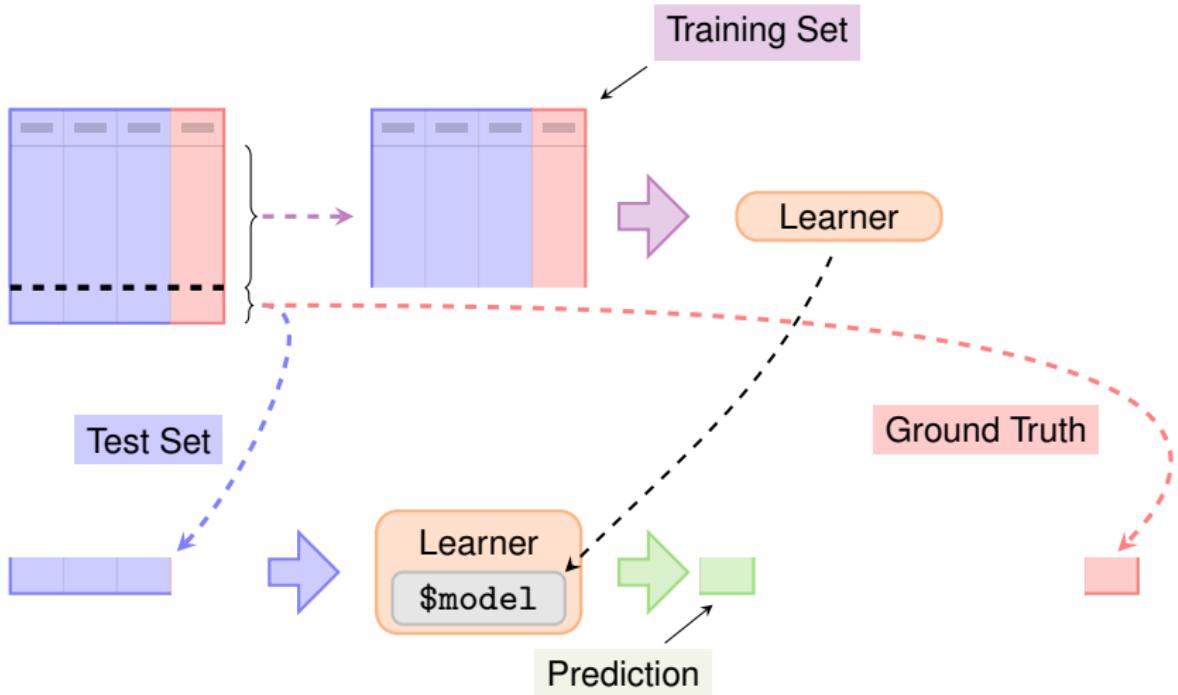
# RESAMPLING



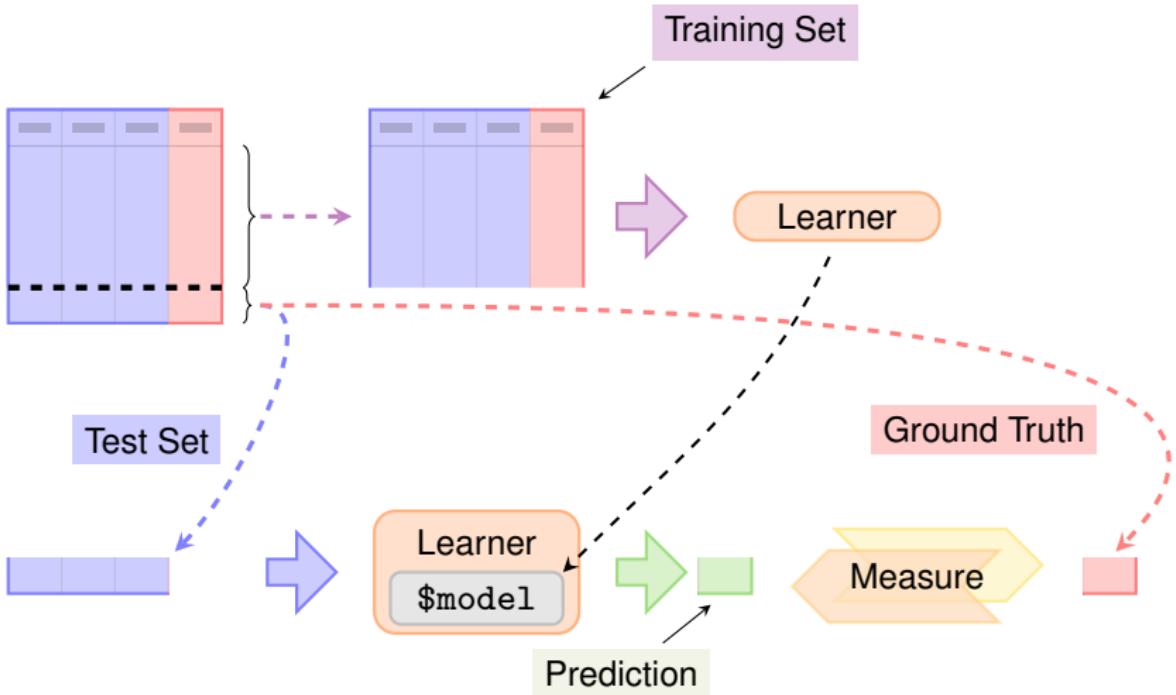
# RESAMPLING



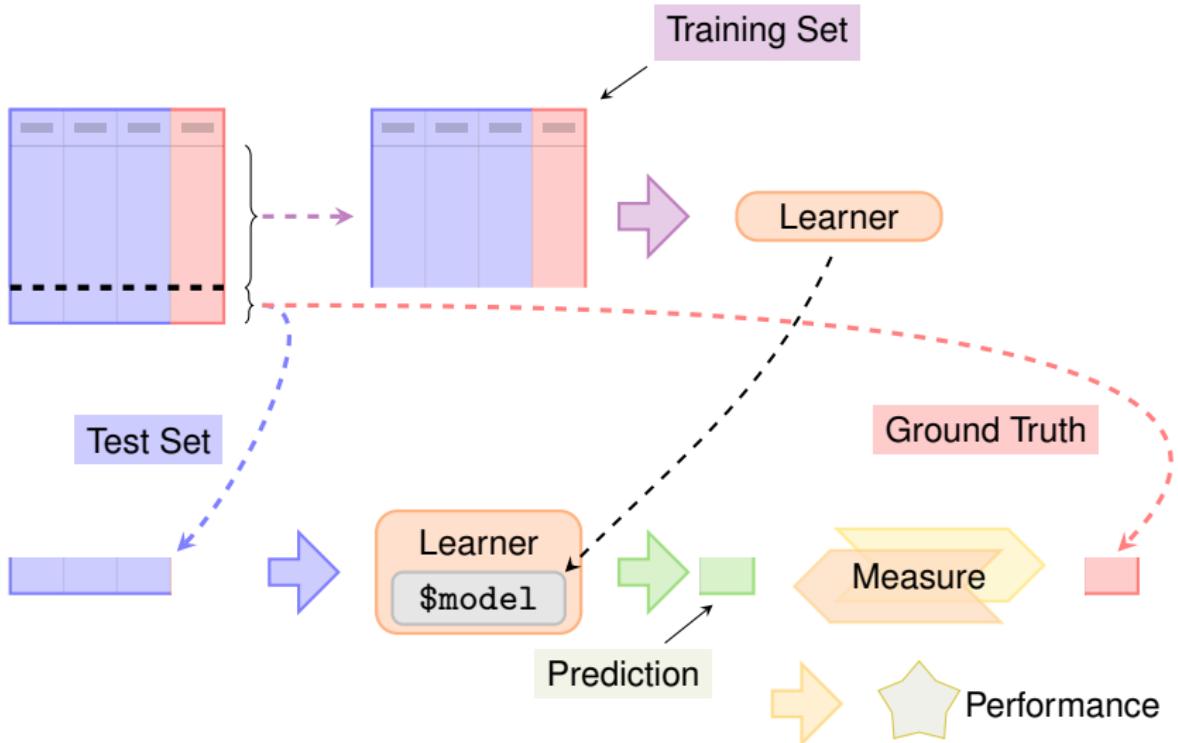
# RESAMPLING



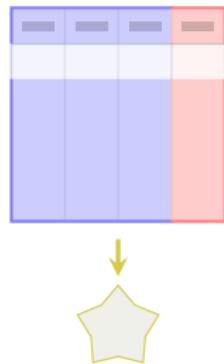
# RESAMPLING



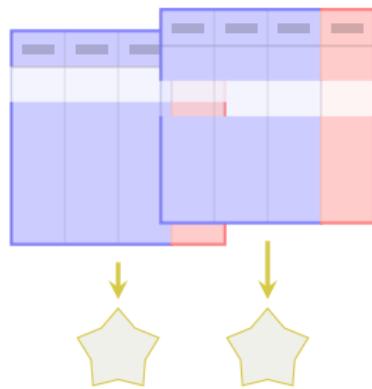
# RESAMPLING



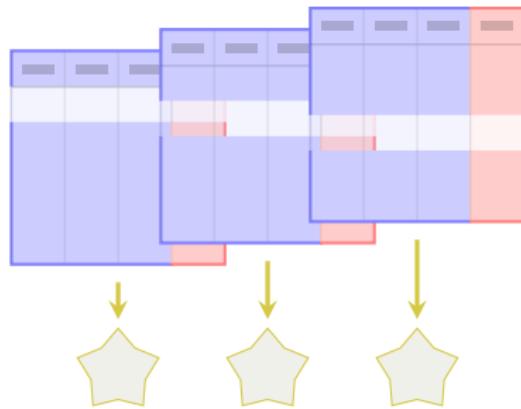
# RESAMPLING



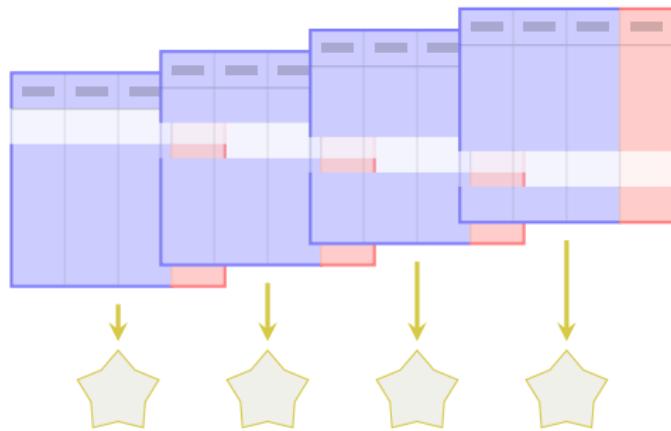
# RESAMPLING



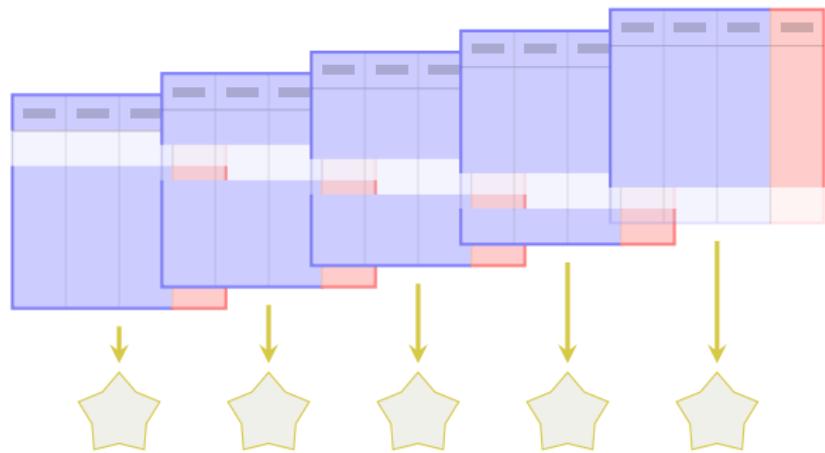
# RESAMPLING



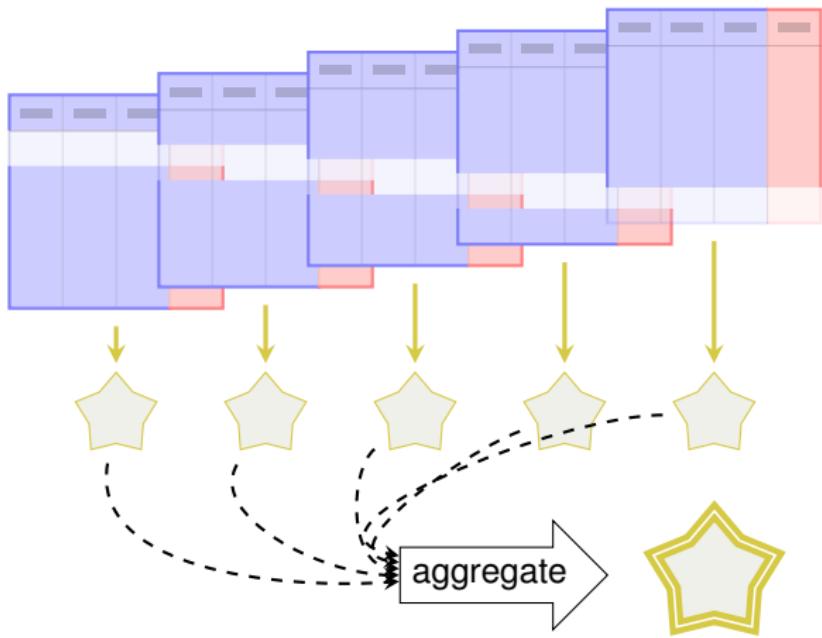
# RESAMPLING



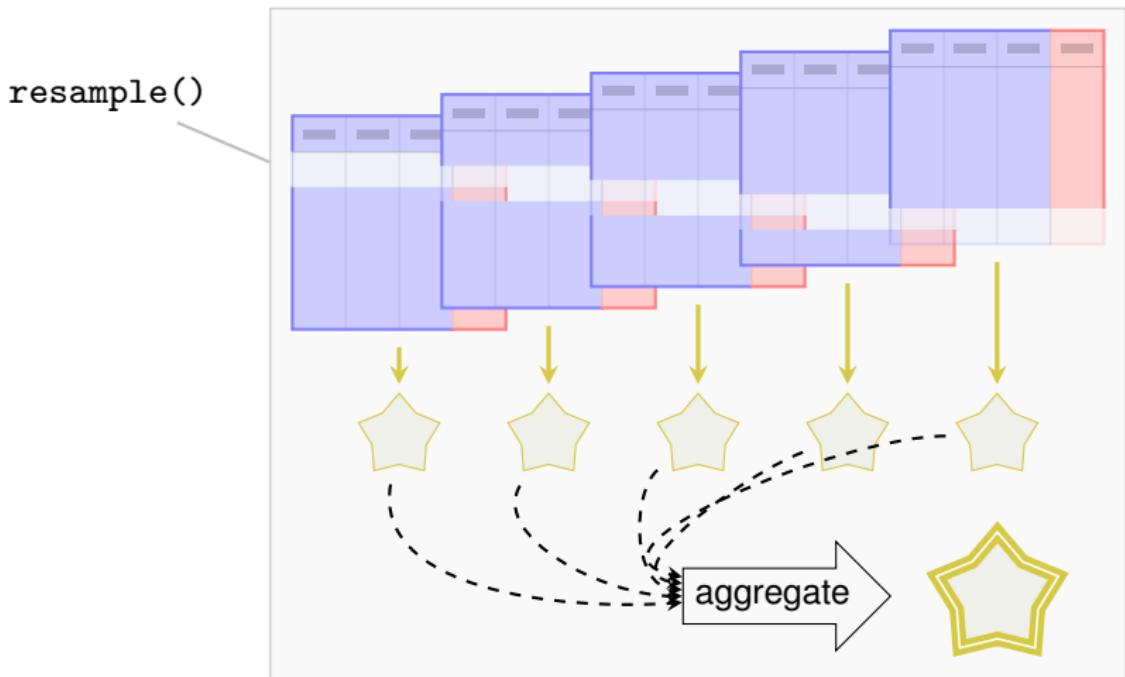
# RESAMPLING



# RESAMPLING



# RESAMPLING



# RESAMPLING

- Resample description: How to split the data

```
cv5 = rsmp("cv", folds = 5)
```

# RESAMPLING

- Resample description: How to split the data

```
cv5 = rsmp("cv", folds = 5)
```

- Use the `resample()` function for resampling:

```
task = TaskClassif$new("iris", iris, "Species")
learner = lrn("classif.rpart")
rr = resample(task, learner, cv5)
```

# RESAMPLING

- Resample description: How to split the data

```
cv5 = rsmp("cv", folds = 5)
```

- Use the `resample()` function for resampling:

```
task = TaskClassif$new("iris", iris, "Species")
learner = lrn("classif.rpart")
rr = resample(task, learner, cv5)
```

- We get a `ResamplingResult` object:

```
print(rr)
#> <ResamplingResult> of 5 iterations
#> * Task: iris
#> * Learner: classif.rpart
#> * Warnings: 0 in 0 iterations
#> * Errors: 0 in 0 iterations
```

# RESAMPLING RESULTS

What exactly is a ResamplingResult object?

# RESAMPLING RESULTS

What exactly is a ResamplingResult object?

Remember Prediction:

# RESAMPLING RESULTS

What exactly is a ResamplingResult object?

Remember Prediction:

- Get a table representation using `as.data.table()`

```
rr_table = as.data.table(rr)

print(rr_table)
#           task          learner      resampling
# 1: <TaskClassif[45]> <LearnerClassifRpart[34]> <ResamplingCV[19]>
# 2: <TaskClassif[45]> <LearnerClassifRpart[34]> <ResamplingCV[19]>
# 3: <TaskClassif[45]> <LearnerClassifRpart[34]> <ResamplingCV[19]>
# 4: <TaskClassif[45]> <LearnerClassifRpart[34]> <ResamplingCV[19]>
# 5: <TaskClassif[45]> <LearnerClassifRpart[34]> <ResamplingCV[19]>
#   iteration      prediction
# 1:           1 <PredictionClassif[19]>
# 2:           2 <PredictionClassif[19]>
# 3:           3 <PredictionClassif[19]>
# 4:           4 <PredictionClassif[19]>
# 5:           5 <PredictionClassif[19]>
```

# RESAMPLING RESULTS

What exactly is a ResamplingResult object?

Remember Prediction:

- Get a table representation using `as.data.table()`

```
rr_table = as.data.table(rr)

print(rr_table)

#           task          learner      resampling
# 1: <TaskClassif[45]> <LearnerClassifRpart[34]> <ResamplingCV[19]>
# 2: <TaskClassif[45]> <LearnerClassifRpart[34]> <ResamplingCV[19]>
# 3: <TaskClassif[45]> <LearnerClassifRpart[34]> <ResamplingCV[19]>
# 4: <TaskClassif[45]> <LearnerClassifRpart[34]> <ResamplingCV[19]>
# 5: <TaskClassif[45]> <LearnerClassifRpart[34]> <ResamplingCV[19]>
#   iteration      prediction
# 1:           1 <PredictionClassif[19]>
# 2:           2 <PredictionClassif[19]>
# 3:           3 <PredictionClassif[19]>
# 4:           4 <PredictionClassif[19]>
# 5:           5 <PredictionClassif[19]>
```

- Active bindings and functions that make information easily accessible

# RESAMPLING RESULTS

- Calculate performance:

```
rr$aggregate(msr("classif.ce"))
#> classif.ce
#>      0.06
```

# RESAMPLING RESULTS

- Calculate performance:

```
rr$aggregate(msr("classif.ce"))

#> classif.ce
#>      0.06
```

- Get predictions

```
rr$prediction()

#> <PredictionClassif> for 150 observations:
#>   row_id    truth  response
#>     3    setosa    setosa
#>     8    setosa    setosa
#>    10    setosa    setosa
#>   ---
#>   143 virginica virginica
#>   144 virginica virginica
#>   145 virginica virginica
```

# RESAMPLING

- Predictions of individual folds

```
predictions = rr$predictions()  
predictions[[1]]  
  
#> <PredictionClassif> for 30 observations:  
#>   row_id     truth  response  
#>       3     setosa    setosa  
#>       8     setosa    setosa  
#>      10     setosa    setosa  
#>     ---  
#>     136 virginica virginica  
#>     140 virginica virginica  
#>     142 virginica virginica
```

# RESAMPLING

- Predictions of individual folds

```
predictions = rr$predictions()
predictions[[1]]

#> <PredictionClassif> for 30 observations:
#>   row_id    truth  response
#>   3      setosa    setosa
#>   8      setosa    setosa
#>   10     setosa    setosa
#>   ---
#>   136    virginica virginica
#>   140    virginica virginica
#>   142    virginica virginica
```

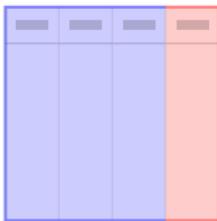
- Score of individual folds

```
scores = rr$score()
scores[1:3, c("iteration", "classif.ce")]

#>   iteration classif.ce
#> 1:          1      0.100
#> 2:          2      0.067
#> 3:          3      0.033
```

# **Benchmark**

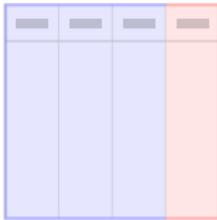
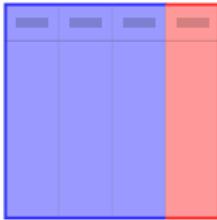
# PERFORMANCE COMPARISON



Learner 1

Learner 2

Learner 3



# PERFORMANCE COMPARISON

- Multiple Learners, multiple Tasks:

```
library("mlr3learners")
learners = list(lrn("classif.rpart"), lrn("classif.kknn"))
tasks = list(tsk("iris"), tsk("sonar"), tsk("wine"))
```

# PERFORMANCE COMPARISON

- Multiple Learners, multiple Tasks:

```
library("mlr3learners")
learners = list(lrn("classif.rpart"), lrn("classif.kknn"))
tasks = list(tsk("iris"), tsk("sonar"), tsk("wine"))
```

- Set up the *design* and execute benchmark:

```
design = benchmark_grid(tasks, learners, cv5)
bmr = benchmark(design)
```

# PERFORMANCE COMPARISON

- Multiple Learners, multiple Tasks:

```
library("mlr3learners")
learners = list(lrn("classif.rpart"), lrn("classif.kknn"))
tasks = list(tsk("iris"), tsk("sonar"), tsk("wine"))
```

- Set up the *design* and execute benchmark:

```
design = benchmark_grid(tasks, learners, cv5)
bmr = benchmark(design)
```

- We get a `BenchmarkResult` object which shows that `kknn` outperforms `rpart`:

```
bmr_ag = bmr$aggregate()
bmr_ag[, c("task_id", "learner_id", "classif.ce")]

#>   task_id   learner_id classif.ce
#> 1:   iris classif.rpart    0.067
#> 2:   iris classif.kknn    0.060
#> 3:  sonar classif.rpart    0.269
#> 4:  sonar classif.kknn    0.164
#> 5:   wine classif.rpart    0.130
#> 6:   wine classif.kknn    0.028
```

# BENCHMARK RESULT

What exactly is a BenchmarkResult object?

# BENCHMARK RESULT

What exactly is a `BenchmarkResult` object?  
Just like `Prediction` and `ResamplingResult`!

# BENCHMARK RESULT

What exactly is a `BenchmarkResult` object?

Just like `Prediction` and `ResamplingResult`!

- Table representation using `as.data.table()`

# BENCHMARK RESULT

What exactly is a `BenchmarkResult` object?

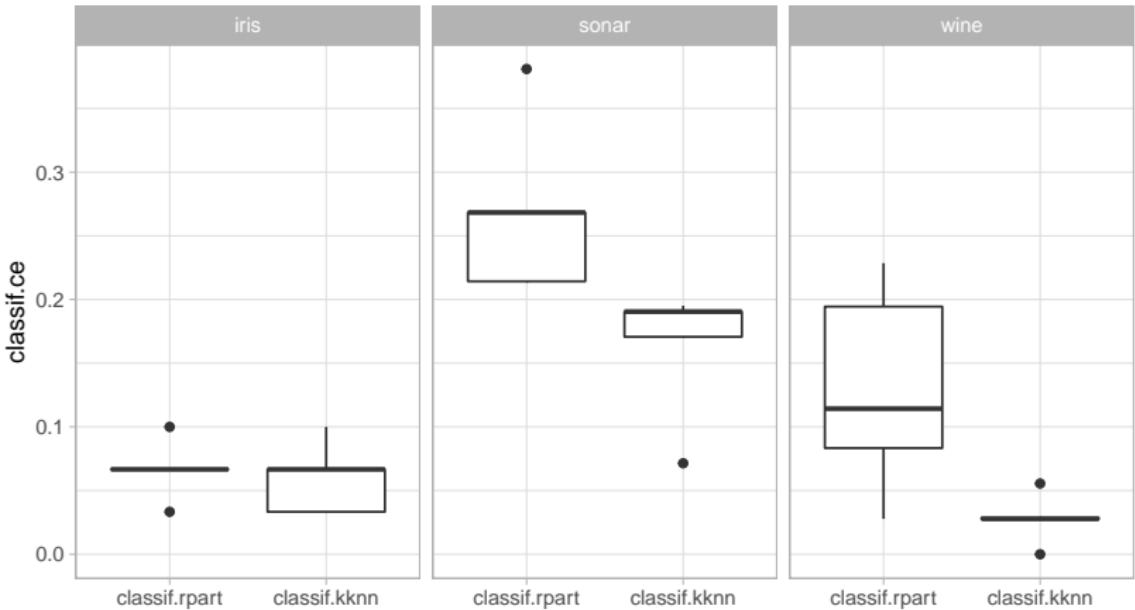
Just like `Prediction` and `ResamplingResult`!

- Table representation using `as.data.table()`
- Active bindings and functions that make information easily accessible

# BENCHMARK RESULT

The `mlr3viz` package contains `autoplot()` functions for many `mlr3` objects

```
library(mlr3viz)
autoplot(bmr)
```



# **Control of Execution**

# CONTROL OF EXECUTION

## Parallelization

```
future::plan("multicore")
```

- runs each resampling iteration as a job
- also allows nested resampling (although not needed here)

## Encapsulation

```
learner$encapsulate = c(train = "callr", predict = "callr")
```

- Spawns a separate R process to train the learner
- Learner may segfault without tearing down the session
- Logs are captured
- Possibility to have a fallback to create predictions

# **How to get Help**

# HOW TO GET HELP

- Where to start?
  - Check these slides
  - **Check the mlr3book <https://mlr3book.mlr-org.com>**

# HOW TO GET HELP

- Where to start?
  - Check these slides
  - **Check the mlr3book <https://mlr3book.mlr-org.com>**
- Get help for R6 objects?

- ① Find out what kind of R6 object you have:

```
class(bmr)
#> [1] "BenchmarkResult" "R6"
```

- ② Go to the corresponding help page:

```
?BenchmarkResult
```

New: open the corresponding man page with

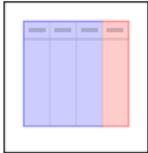
```
learner$help()
```

# **Outro**

# OVERVIEW

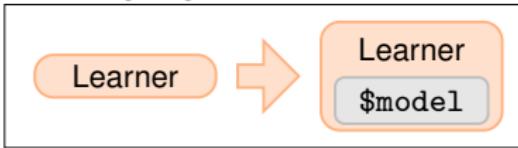
Ingredients:

Data



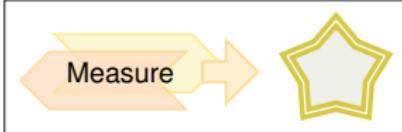
TaskClassif,  
TaskRegr,  
tsk()

Learning Algorithms



lrn() ⇒ Learner,  
→ Learner\$train(),  
→ Learner\$predict() ⇒ Prediction

Performance Evaluation



rsmp() ⇒ Resampling,  
msr() ⇒ Measure,  
resample() ⇒ ResamplingResult,  
→ ResamplingResult\$score(),  
→ ResamplingResult\$aggregate()

Performance Comparison



benchmark\_grid(),  
benchmark() ⇒ BenchmarkResult

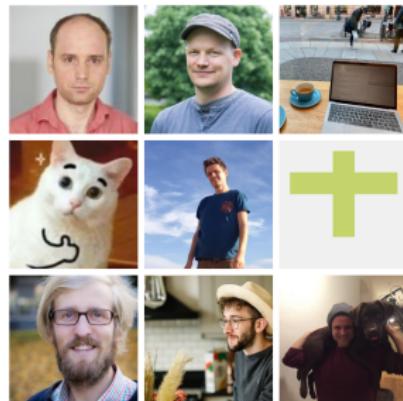
# Tuning Machine Learning Algorithms with mlr3

---



<https://mlr-org.com/>

<https://github.com/mlr-org>



---

**Bernd Bischl, Michel Lang, Martin Binder, Florian Pfisterer, Jakob Richter,  
Patrick Schratz, Lennart Schneider, Raphael Sonabend, Marc Becker,  
Giuseppe Casalicchio**

# **Intro**

# TUNING

- Behavior of most methods depends on *hyperparameters*

# TUNING

- Behavior of most methods depends on *hyperparameters*
- We want to choose them so our algorithm performs well

# TUNING

- Behavior of most methods depends on *hyperparameters*
- We want to choose them so our algorithm performs well
- Good hyperparameters are data-dependent

# TUNING

- Behavior of most methods depends on *hyperparameters*
  - We want to choose them so our algorithm performs well
  - Good hyperparameters are data-dependent
- ⇒ We do *black box optimization* (“Try stuff and see what works”)

# TUNING

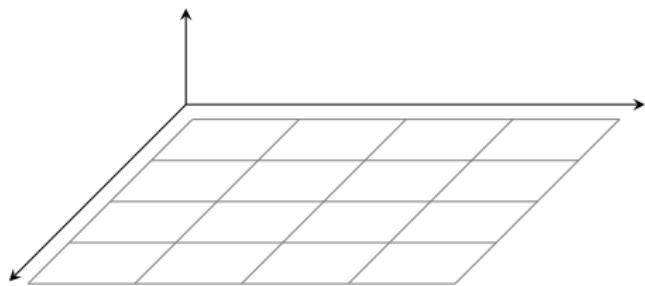
- Behavior of most methods depends on *hyperparameters*
  - We want to choose them so our algorithm performs well
  - Good hyperparameters are data-dependent
- ⇒ We do *black box optimization* (“Try stuff and see what works”)

Tuning toolbox for `mlr3`:

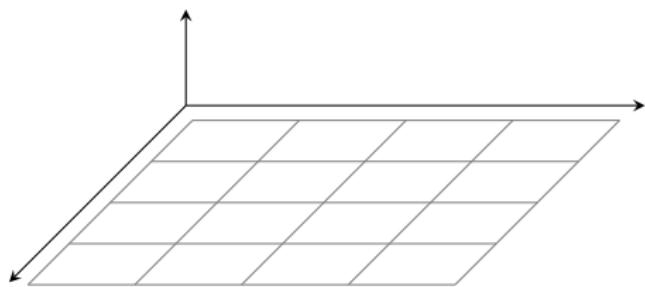
```
library("bbotk")
library("mlr3tuning")
```

# Tuning

# TUNING

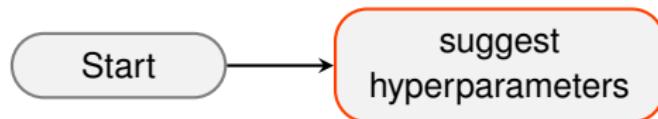
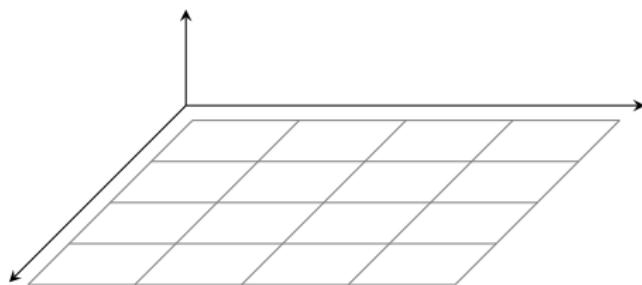


# TUNING

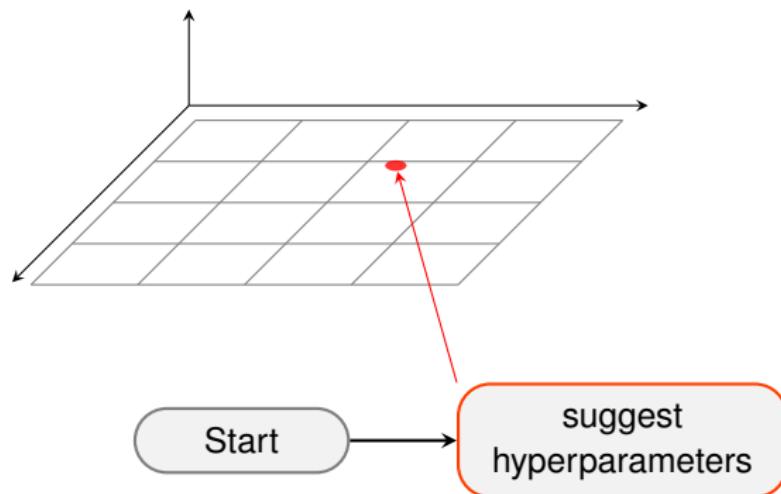


Start

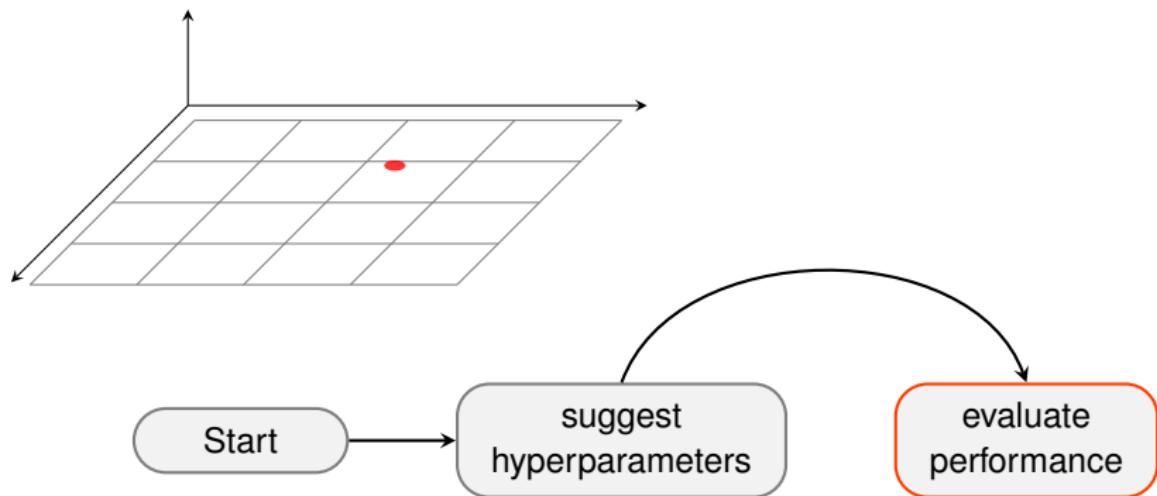
# TUNING



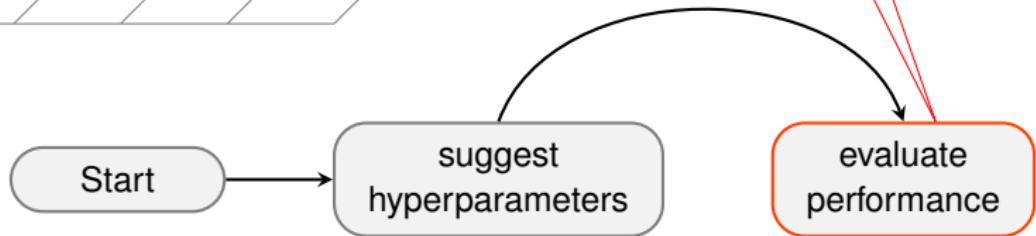
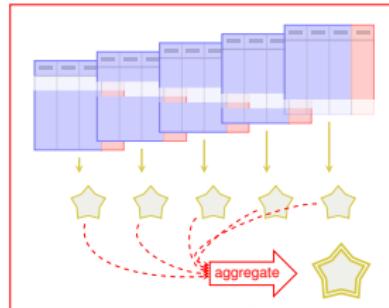
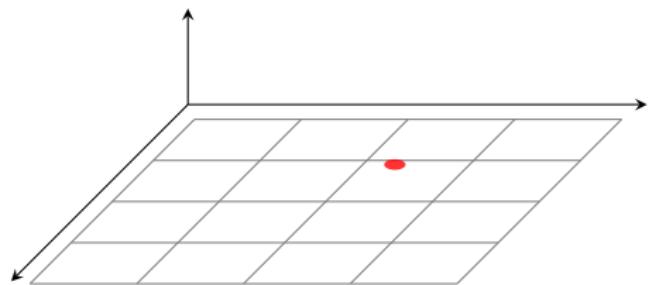
# TUNING



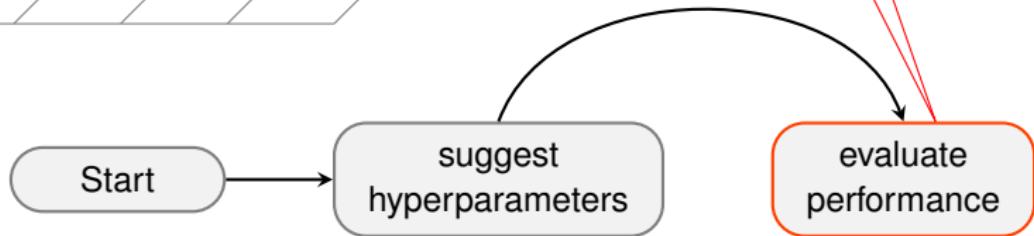
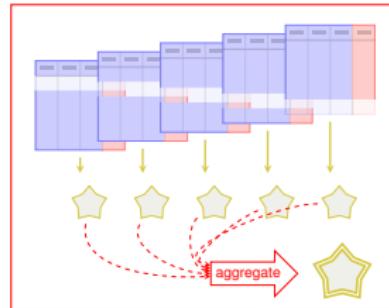
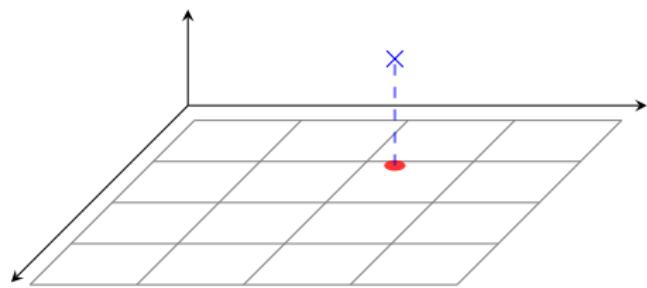
# TUNING



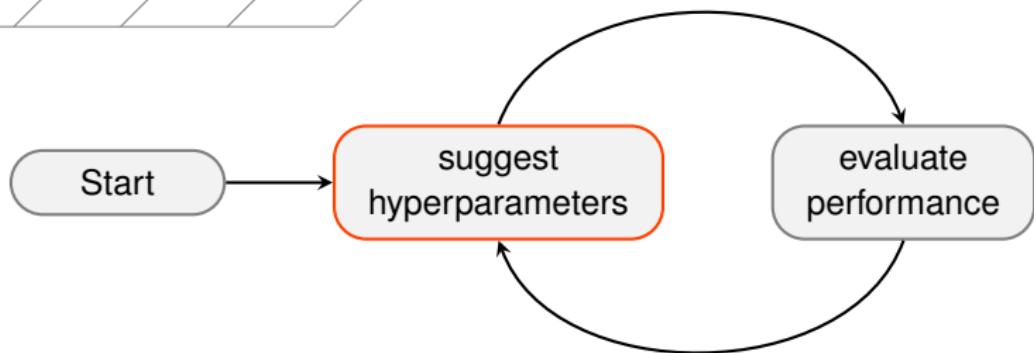
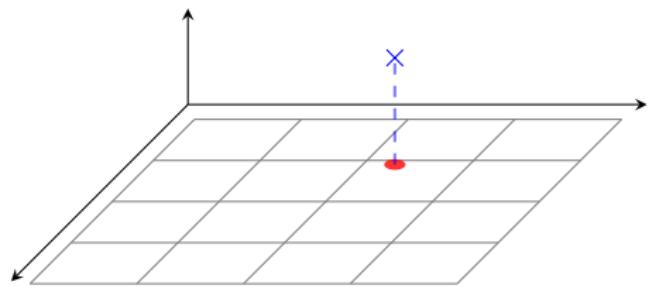
# TUNING



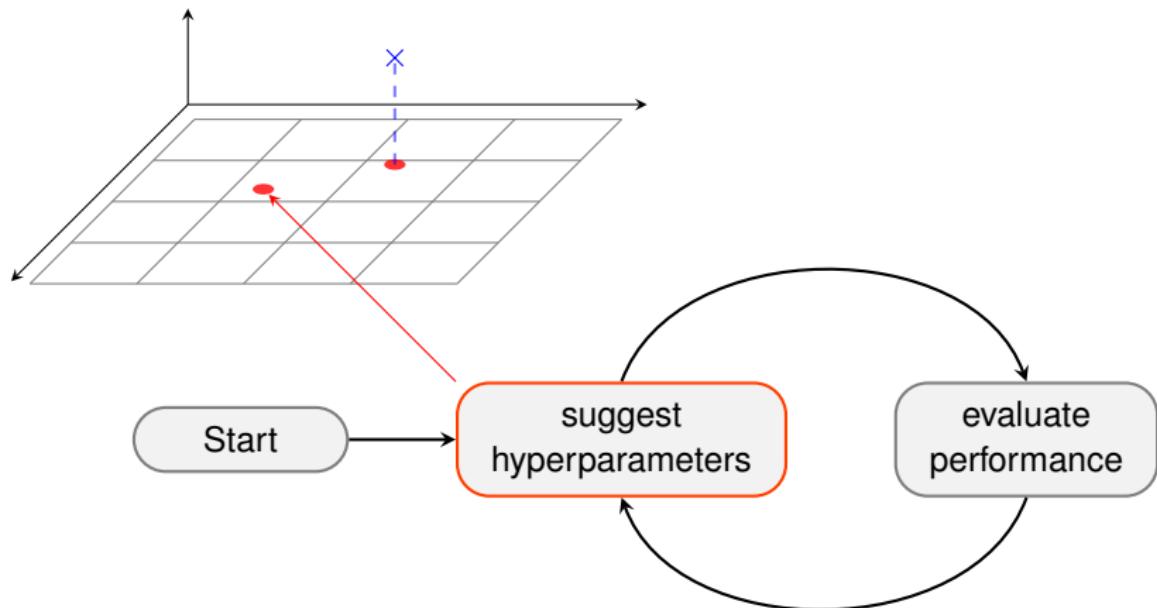
# TUNING



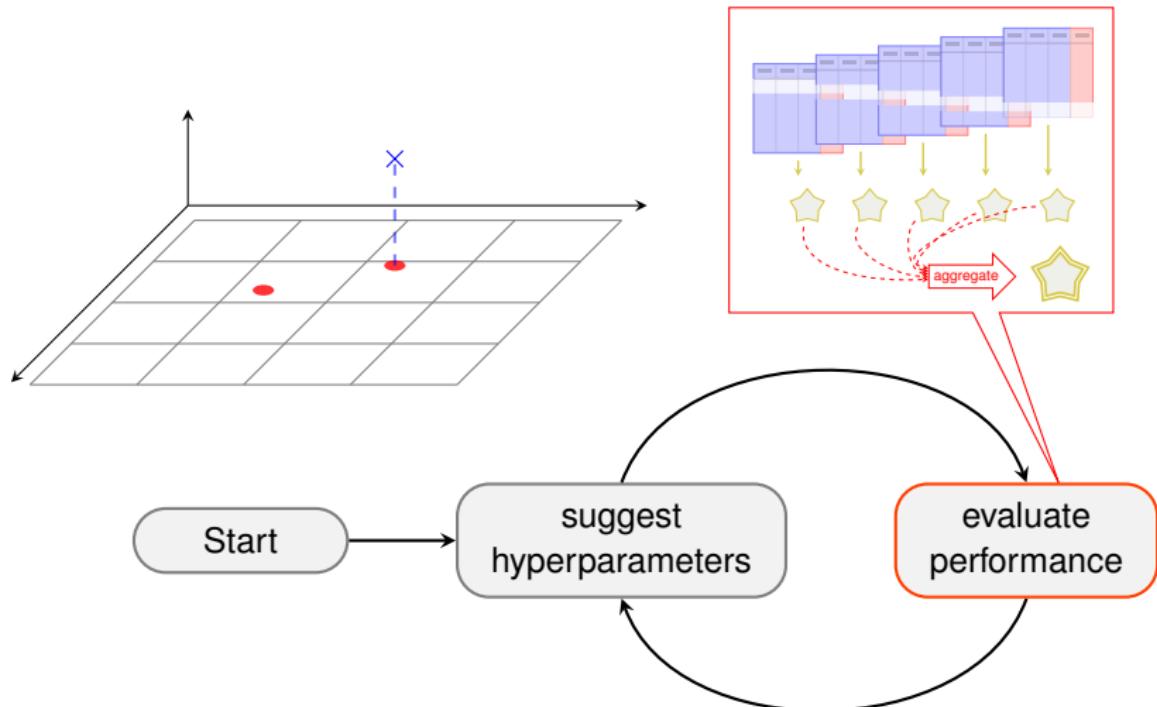
# TUNING



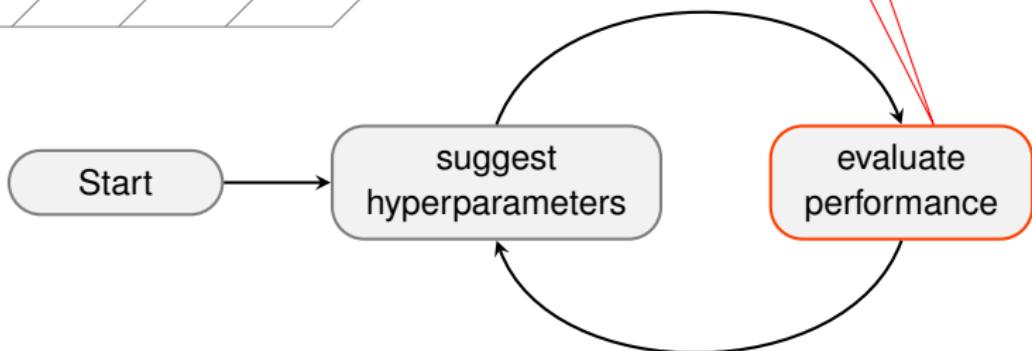
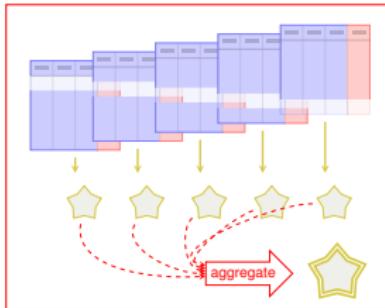
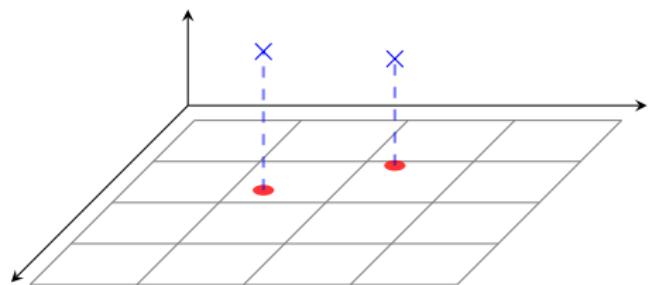
# TUNING



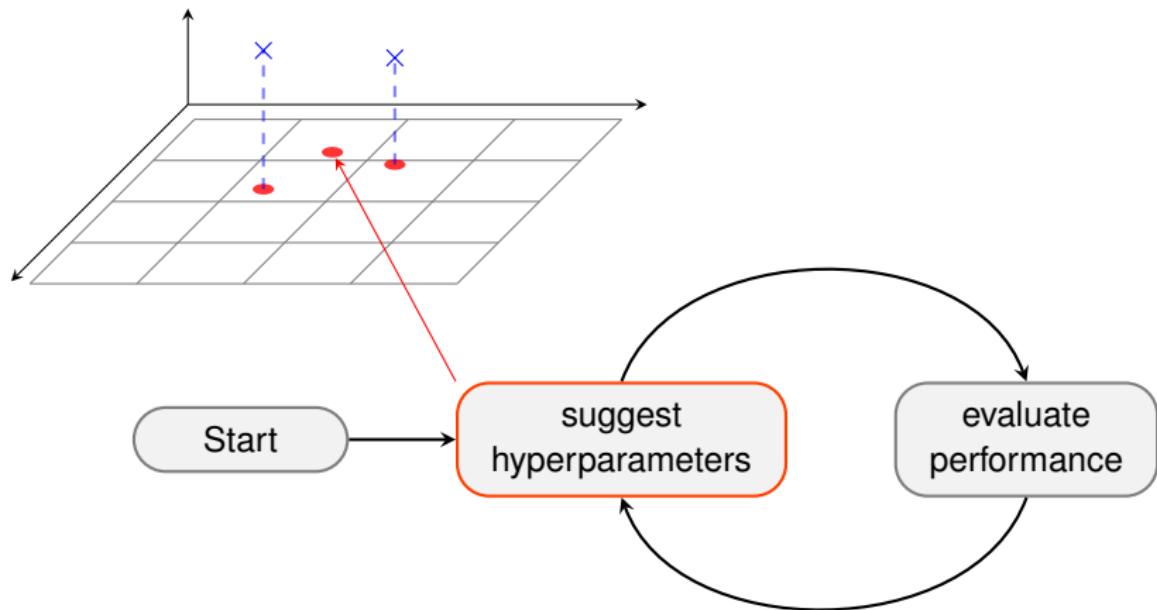
# TUNING



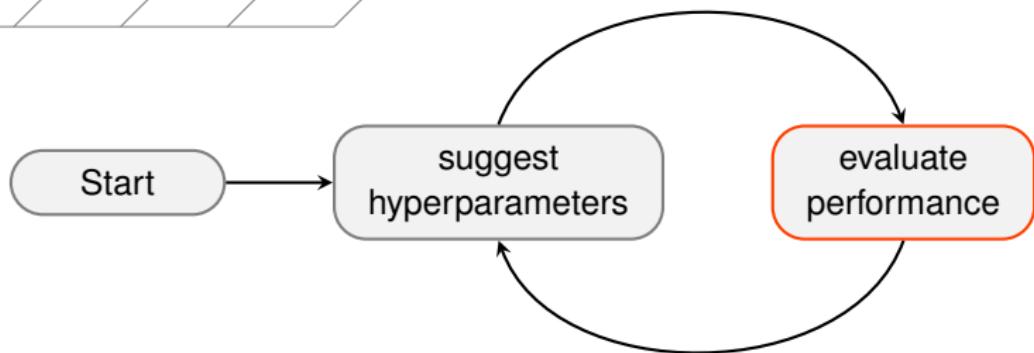
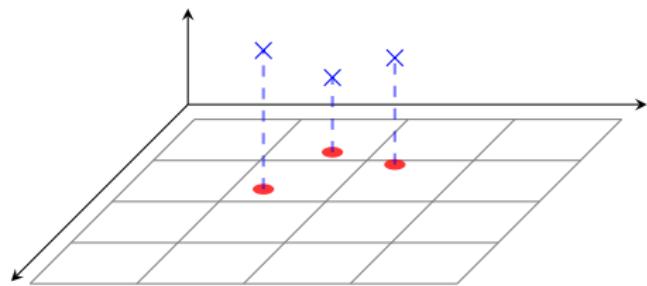
# TUNING



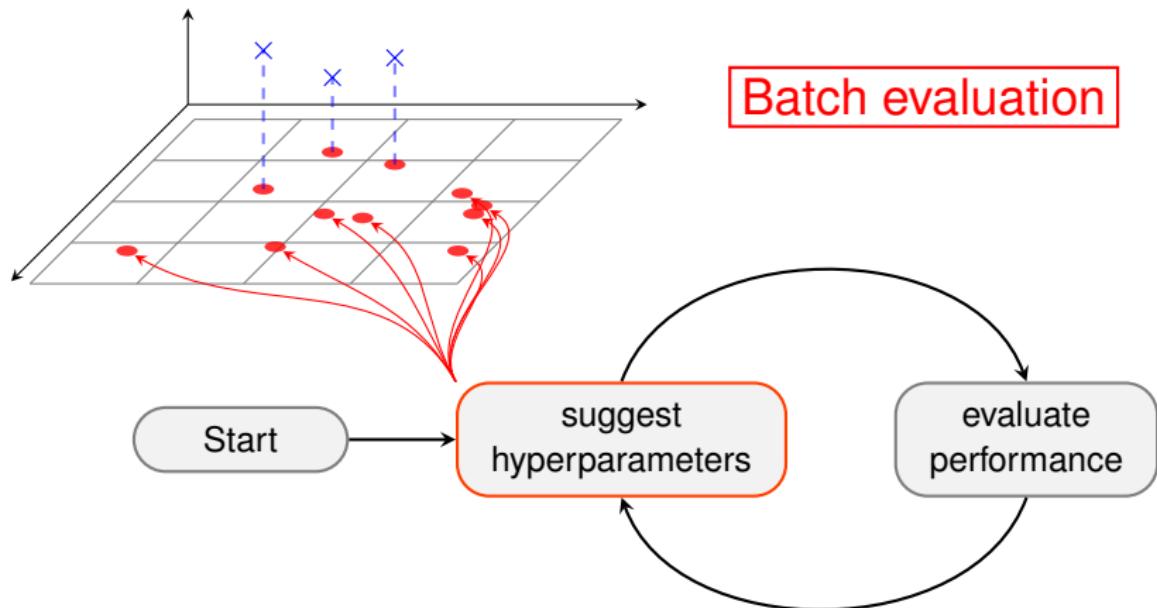
# TUNING



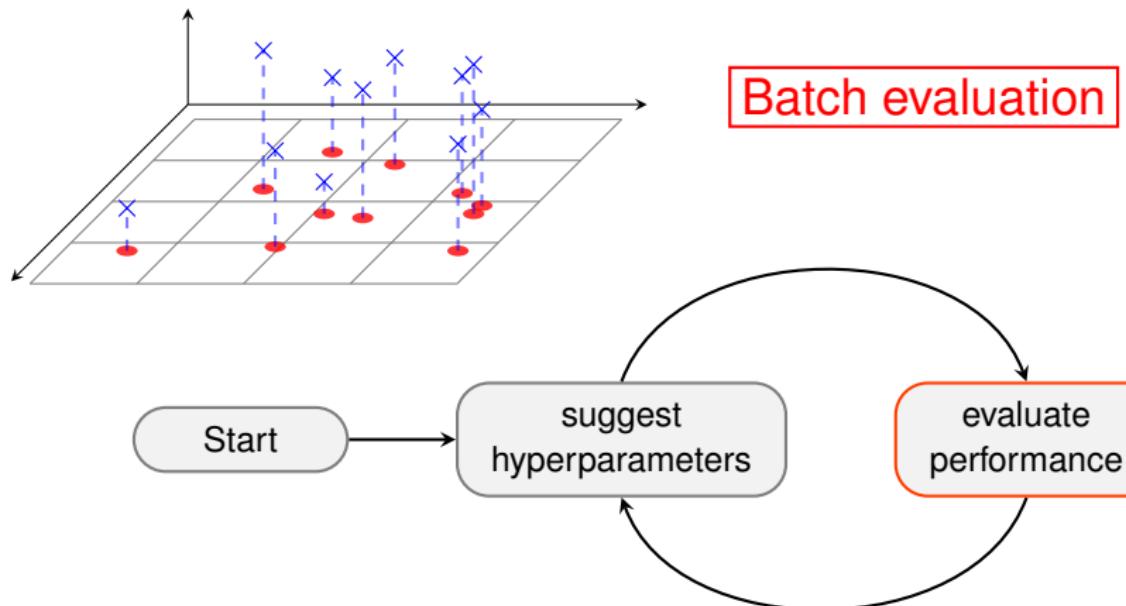
# TUNING



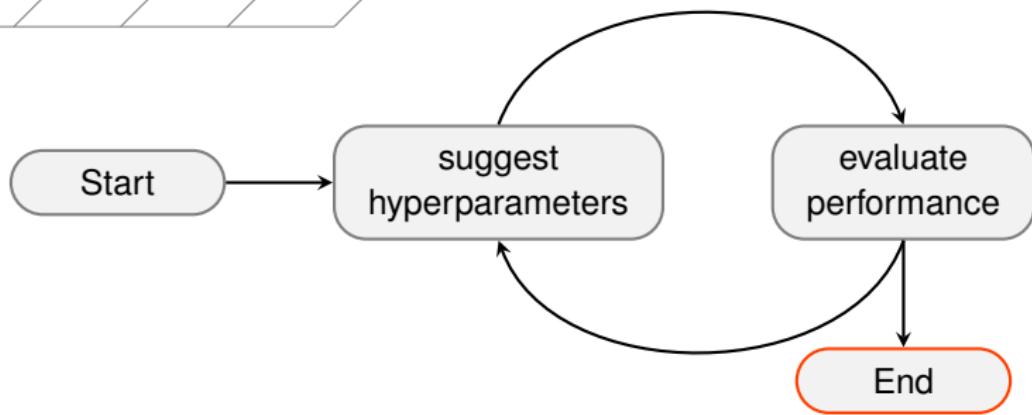
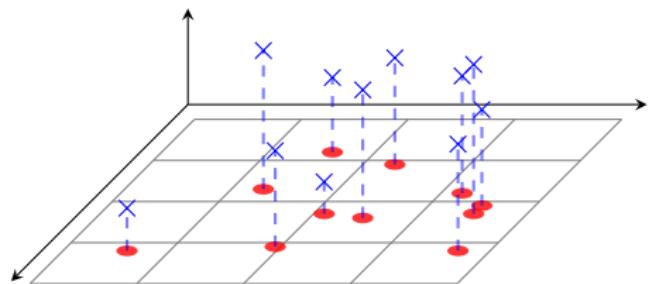
# TUNING



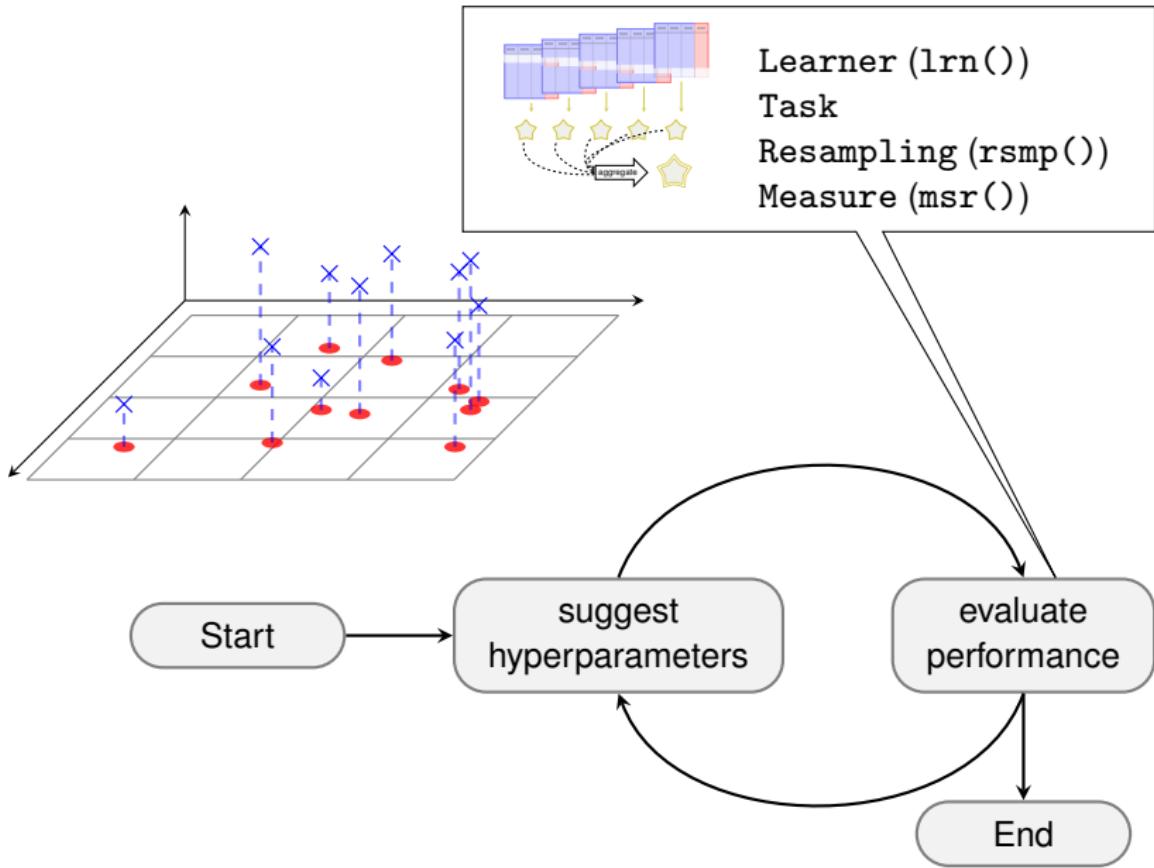
# TUNING



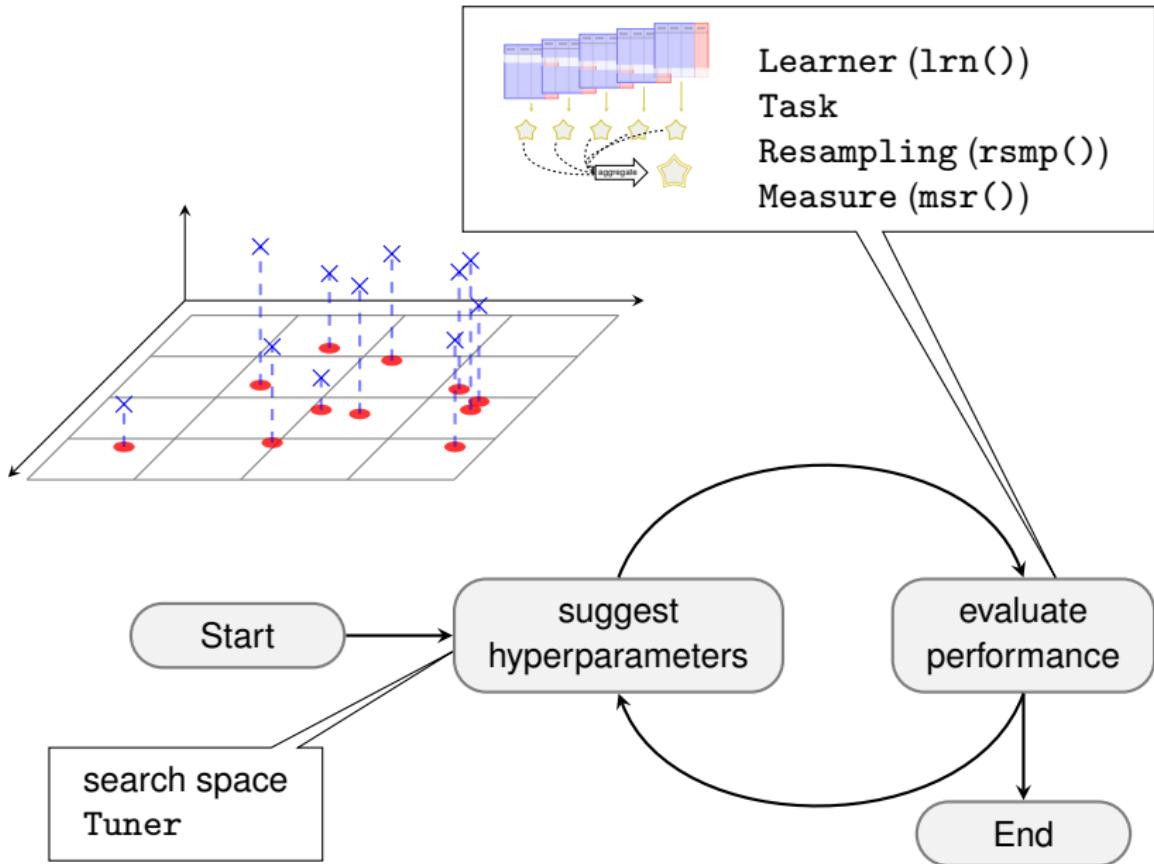
# TUNING



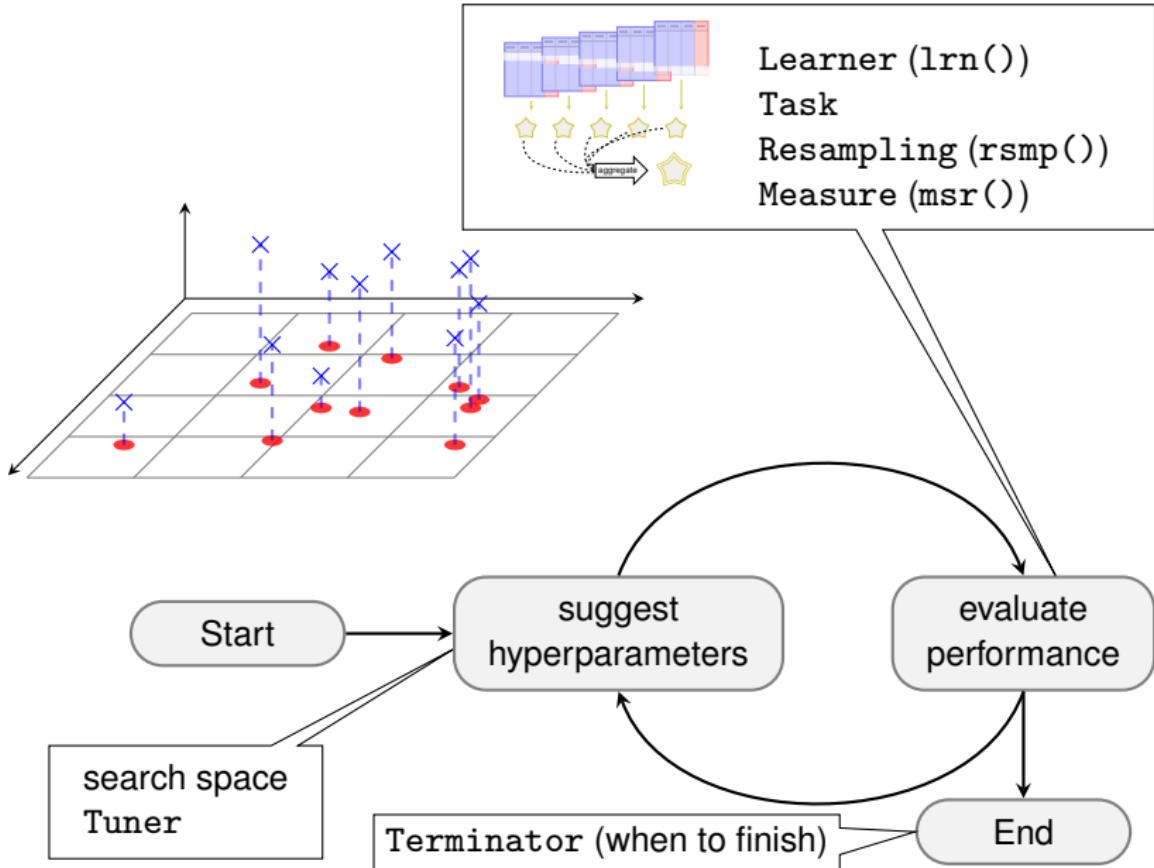
# TUNING



# TUNING

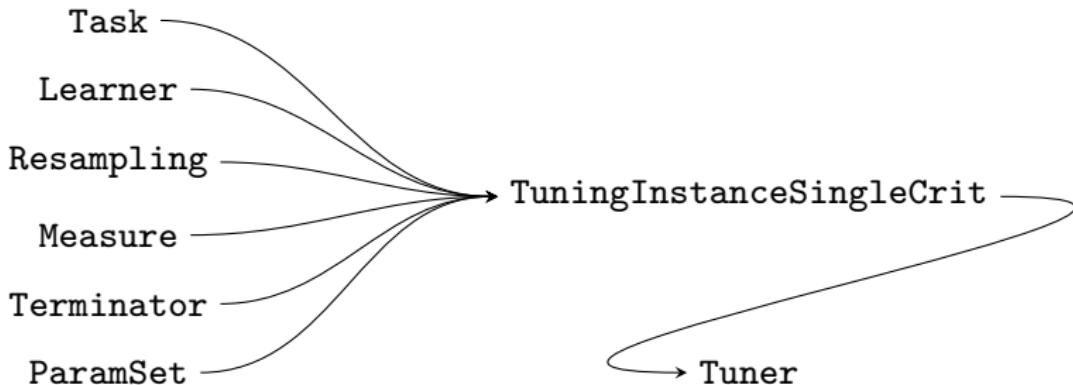


# TUNING

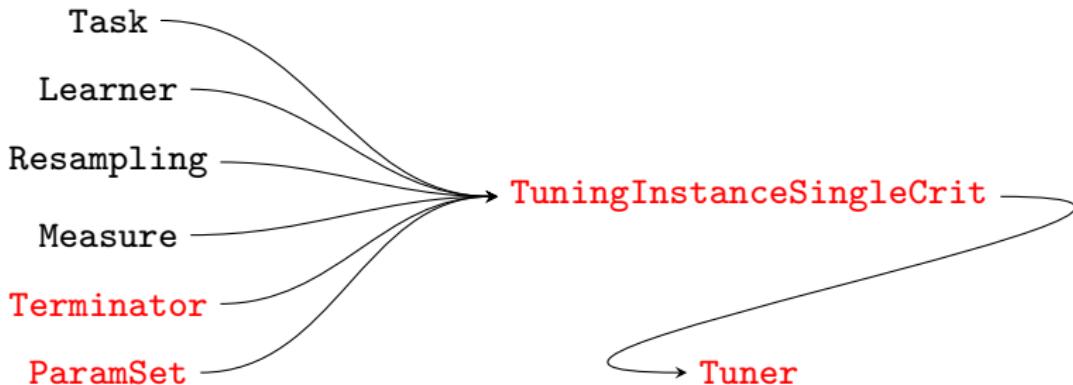


## Tuning in mlr3

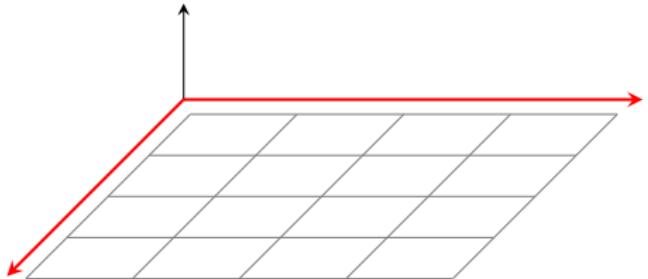
# OBJECTS IN TUNING



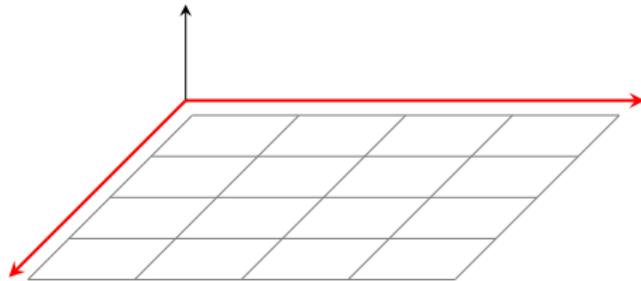
# OBJECTS IN TUNING



# SEARCH SPACE

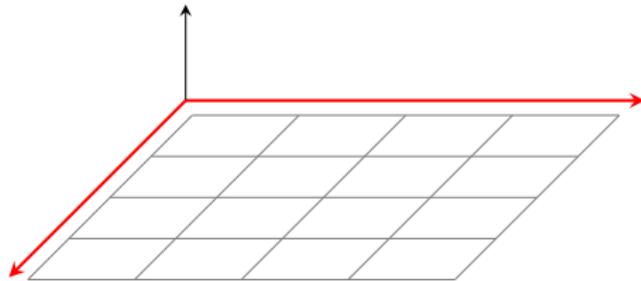


# SEARCH SPACE



```
ParamSet$new(list(param1, param2, ...))
```

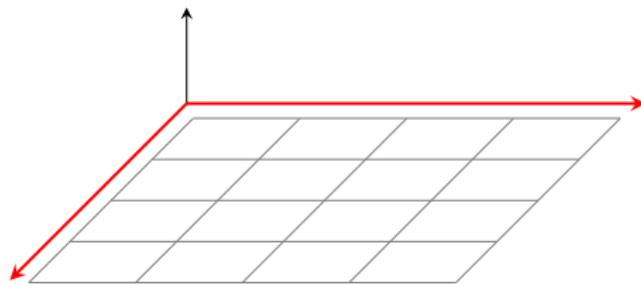
# SEARCH SPACE



```
ParamSet$new(list(param1, param2, ...))
```

<i>Numerical</i> parameter	ParamDbl\$new(id, lower, upper)
<i>Integer</i> parameter	ParamInt\$new(id, lower, upper)
<i>Discrete</i> parameter	ParamFct\$new(id, levels)
<i>Logical</i> parameter	ParamLgl\$new(id)
<i>Untyped</i> parameter	ParamUty\$new(id)

# SEARCH SPACE

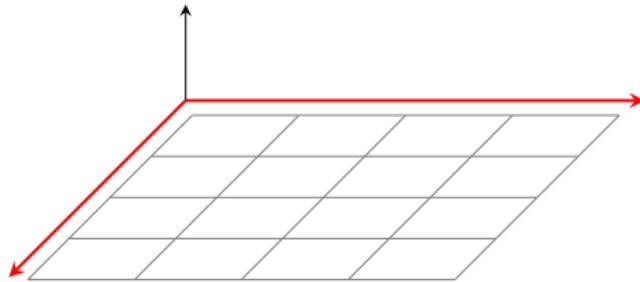


```
ParamSet$new(list(param1, param2, ...))
```

<i>Numerical</i> parameter	ParamDbl\$new(id, lower, upper)
<i>Integer</i> parameter	ParamInt\$new(id, lower, upper)
<i>Discrete</i> parameter	ParamFct\$new(id, levels)
<i>Logical</i> parameter	ParamLgl\$new(id)
<i>Untyped</i> parameter	ParamUty\$new(id)

```
library("paradox")
searchspace_knn = ParamSet$new(list(
  ParamInt$new("k", lower = 1, upper = 20)
))
```

# SEARCH SPACE SHORT FORM



```
ps(id1 = domain1, id2 = domain2, ...)
```

*Numerical* parameter `p_dbl(lower, upper)`

*Integer* parameter `p_int(lower, upper)`

*Discrete* parameter `p_fct(levels)`

*Logical* parameter `p_lgl()`

*Untyped* parameter `p_uty()`

```
library("paradox")
searchspace_knn = ps(
  "k" = p_int(lower = 1, upper = 20)
)
```

# TERMINATION

- Tuning needs a *termination condition*: when to finish

# TERMINATION

- Tuning needs a *termination condition*: when to finish
- Terminator class

# TERMINATION

- Tuning needs a *termination condition*: when to finish
- Terminator class
- `mlr_terminators` dictionary, `trm()` short form

# TERMINATION

- Tuning needs a *termination condition*: when to finish
- Terminator class
- `mlr_terminators` dictionary, `trm()` short form
- `as.data.table(mlr_terminators)`

```
#>           key
#> 1:      clock_time
#> 2:      combo
#> 3:      evals
#> 4:      none
#> 5:      perf_reached
#> 6:      run_time
#> 7:      stagnation
#> 8:      stagnation_batch
```

# TERMINATION

- Tuning needs a *termination condition*: when to finish
- Terminator class
- `mlr_terminators` dictionary, `trm()` short form

- `as.data.table(mlr_terminators)`

```
#>                 key
#> 1:      clock_time
#> 2:      combo
#> 3:      evals
#> 4:      none
#> 5:      perf_reached
#> 6:      run_time
#> 7:      stagnation
#> 8: stagnation_batch
```

- `trm("evals", n_evals = 20)`

```
#> <TerminatorEvals>
#> * Parameters: n_evals=20
```

# TUNING METHOD

- need to choose a *tuning method*

# TUNING METHOD

- need to choose a *tuning method*
- Tuner class

# TUNING METHOD

- need to choose a *tuning method*
- Tuner class
- `mlr_tuners` dictionary, `tnr()` short form

# TUNING METHOD

- need to choose a *tuning method*
- Tuner class
- `mlr_tuners` dictionary, `tnr()` short form

- `as.data.table(mlr_tuners)`

```
#>           key
#> 1:      cmaes
#> 2: design_points
#> 3:      gensa
#> 4:    grid_search
#> 5:      nloptr
#> 6: random_search
```

# TUNING METHOD

- load Tuner with `tnr()`, set parameters

# TUNING METHOD

- load Tuner with `tnr()`, set parameters

- `gsearch = tnr("grid_search", resolution = 3)`

```
print(gsearch)
#> <TunerGridSearch>
#> * Parameters: resolution=3, batch_size=1
#> * Parameter classes: ParamLgl, ParamInt, ParamDbl, ParamFct
#> * Properties: dependencies, single-crit, multi-crit
#> * Packages: -
```

# TUNING METHOD

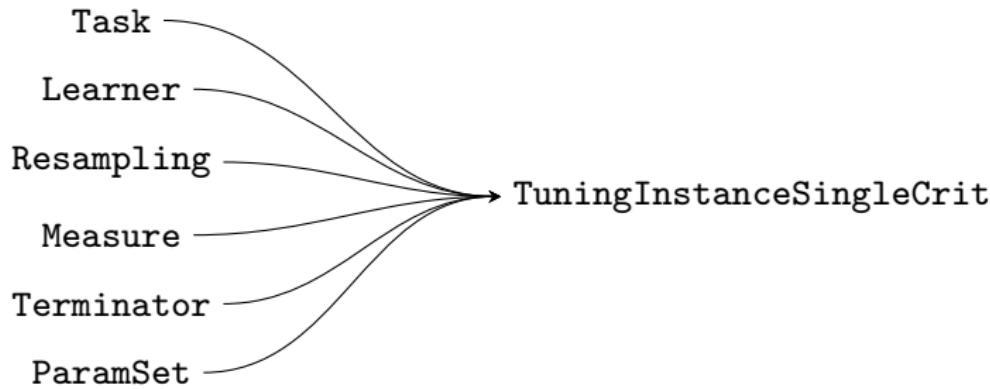
- load Tuner with `tnr()`, set parameters

- `gsearch = tnr("grid_search", resolution = 3)`

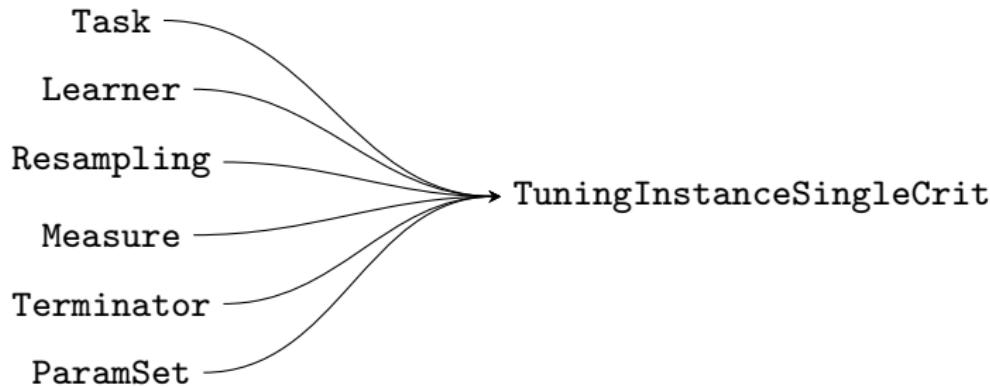
```
print(gsearch)
#> <TunerGridSearch>
#> * Parameters: resolution=3, batch_size=1
#> * Parameter classes: ParamLgl, ParamInt, ParamDbl, ParamFct
#> * Properties: dependencies, single-crit, multi-crit
#> * Packages: -
```

- common parameter `batch_size` for parallelization

# CALLING THE TUNER



# CALLING THE TUNER



```
inst = TuningInstanceSingleCrit$new(task = tsk("iris"),
  learner = lrn("classif.kknn", kernel = "rectangular"),
  resampling = rsmp("holdout"), measure = msr("classif.ce"),
  terminator = trm("none"), search_space = searchspace_knn
)
```

# CALLING THE TUNER

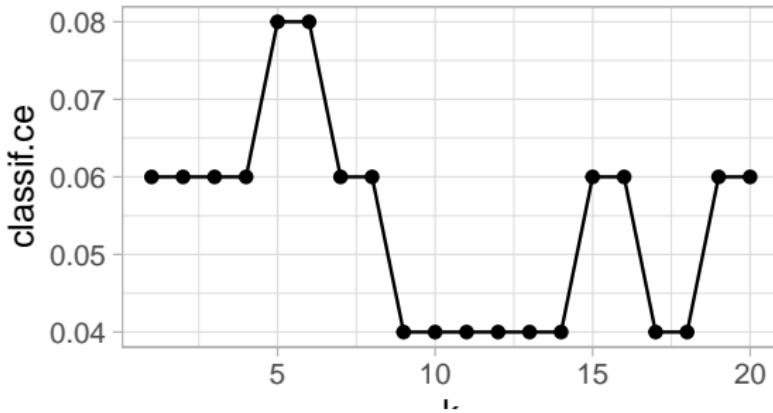
```
gsearch$optimize(inst)
#> INFO [14:35:12.185] [bbotk] Starting to optimize 1 parameter(s) with '<OptimizerGridSea
#> INFO [14:35:12.321] [bbotk] Evaluating 1 configuration(s)
#> INFO [14:35:13.781] [bbotk] Result of batch 1:
#> INFO [14:35:13.783] [bbotk]   k_classif.ce                               uhash
#> INFO [14:35:13.783] [bbotk]   10      0.04 f70af94d-3f75-4e31-8a3c-c23d36711d49
#> INFO [14:35:13.785] [bbotk] Evaluating 1 configuration(s)
#> INFO [14:35:13.948] [bbotk] Result of batch 2:
#> INFO [14:35:13.951] [bbotk]   k_classif.ce                               uhash
#> INFO [14:35:13.951] [bbotk]   1       0.06 9e54838b-dea7-4d75-b1ce-57dba5d1e603
#> INFO [14:35:13.953] [bbotk] Evaluating 1 configuration(s)
#> INFO [14:35:14.108] [bbotk] Result of batch 3:
#> INFO [14:35:14.111] [bbotk]   k_classif.ce                               uhash
#> INFO [14:35:14.111] [bbotk]   20      0.08 f53981d7-6028-4639-9cb1-763e98eb2f73
#> INFO [14:35:14.120] [bbotk] Finished optimizing after 3 evaluation(s)
#> INFO [14:35:14.122] [bbotk] Result:
#> INFO [14:35:14.124] [bbotk]   k_learner_param_vals  x_domain classif.ce
#> INFO [14:35:14.124] [bbotk]   10      <list[2]> <list[1]>      0.04
#>     k_learner_param_vals  x_domain classif.ce
#> 1: 10      <list[2]> <list[1]>      0.04
```

# TUNING RESULTS

```
inst = TuningInstanceSingleCrit$new(task = tsk("iris"),
  learner = lrn("classif.kknn", kernel = "rectangular"),
  resampling = rsmp("holdout"), measure = msr("classif.ce"),
  terminator = trm("none"), search_space = searchspace_knn)
gsearch = tnr("grid_search", resolution = 20)
gsearch$optimize(inst)

#>      k learner_param_vals  x_domain classif.ce
#> 1: 11              <list[2]> <list[1]>      0.04

ggplot(as.data.table(inst$archive), aes(x = k, y = classif.ce)) +
  geom_line() + geom_point()
```



# RECAP

- ❶ Create a Task, Learner, Resampling, Measure, Terminator (defines when to stop), and a ParamSet (defines the search space):

```
task = tsk("iris")
learner = lrn("classif.kknn", kernel = "rectangular")
resampling = rsmp("holdout")
measure = msr("classif.ce")
terminator = trm("evals", n_evals = 2)
searchspace_knn = ParamSet$new(list(
  ParamInt$new("k", lower = 1, upper = 20)
))
```

- ❷ Create the TuningInstanceSingleCrit object:

```
inst = TuningInstanceSingleCrit$new(task, learner,
  resampling, measure, terminator, searchspace_knn)
```

- ❸ Create the Tuner (tuning method) and optimize the learner by passing over the previously created instance to the \$optimize method:

```
gsearch = tnr("grid_search", resolution = 3)
gsearch$optimize(inst)
#>   k learner_param_vals  x_domain classif.ce
#> 1: 1                  <list[2]> <list[1]>      0.04
```

# **Parameter Transformation**

# PARAMETER TRANSFORMATION

- Sometimes we do not want to optimize over an evenly spaced range

# PARAMETER TRANSFORMATION

- Sometimes we do not want to optimize over an evenly spaced range
- $k = 1$  vs.  $k = 2$  probably more interesting than  $k = 101$  vs.  $k = 102$

# PARAMETER TRANSFORMATION

- Sometimes we do not want to optimize over an evenly spaced range
  - $k = 1$  vs.  $k = 2$  probably more interesting than  $k = 101$  vs.  $k = 102$
- ⇒ Transformations

# PARAMETER TRANSFORMATION

- Sometimes we do not want to optimize over an evenly spaced range
  - $k = 1$  vs.  $k = 2$  probably more interesting than  $k = 101$  vs.  $k = 102$
- ⇒ Transformations
- Part of ParamSet

# PARAMETER TRANSFORMATION

- Sometimes we do not want to optimize over an evenly spaced range
  - $k = 1$  vs.  $k = 2$  probably more interesting than  $k = 101$  vs.  $k = 102$
- ⇒ Transformations
- Part of ParamSet

Example:

# PARAMETER TRANSFORMATION

- Sometimes we do not want to optimize over an evenly spaced range
  - $k = 1$  vs.  $k = 2$  probably more interesting than  $k = 101$  vs.  $k = 102$
- ⇒ Transformations
- Part of ParamSet

Example:

- ➊ optimize from  $\log(1) \dots \log(100)$

# PARAMETER TRANSFORMATION

- Sometimes we do not want to optimize over an evenly spaced range
  - $k = 1$  vs.  $k = 2$  probably more interesting than  $k = 101$  vs.  $k = 102$
- ⇒ Transformations
- Part of ParamSet

Example:

- ➊ optimize from  $\log(1) \dots \log(100)$
- ➋ transform by `exp()` in `trafo` function

# PARAMETER TRANSFORMATION

- Sometimes we do not want to optimize over an evenly spaced range
  - $k = 1$  vs.  $k = 2$  probably more interesting than  $k = 101$  vs.  $k = 102$
- ⇒ Transformations
- Part of ParamSet

Example:

- ➊ optimize from  $\log(1) \dots \log(100)$
- ➋ transform by `exp()` in `trafo` function
- ➌ don't forget to `round` ( $k$  must be integer)

# PARAMETER TRANSFORMATION

- Sometimes we do not want to optimize over an evenly spaced range
  - $k = 1$  vs.  $k = 2$  probably more interesting than  $k = 101$  vs.  $k = 102$
- ⇒ Transformations
- Part of ParamSet

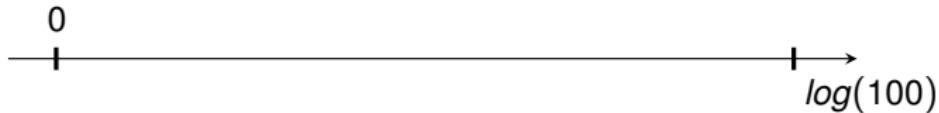
Example:

- ➊ optimize from  $\log(1) \dots \log(100)$
- ➋ transform by `exp()` in `trafo` function
- ➌ don't forget to `round` ( $k$  must be integer)

```
searchspace_knn_trafo = ParamSet$new(list(  
  ParamDbl$new("k", log(1), log(50))  
))  
searchspace_knn_trafo$trafo = function(x, param_set) {  
  x$k = round(exp(x$k))  
  return(x)  
}
```

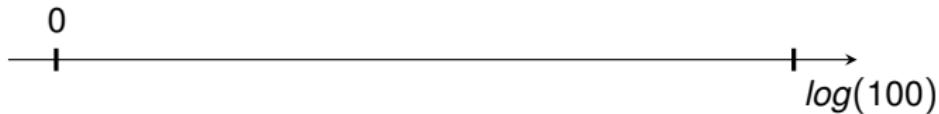
# PARAMETER TRANSFORMATION

What is our transformation doing?



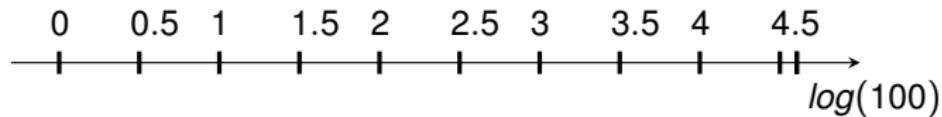
# PARAMETER TRANSFORMATION

What is our transformation doing?



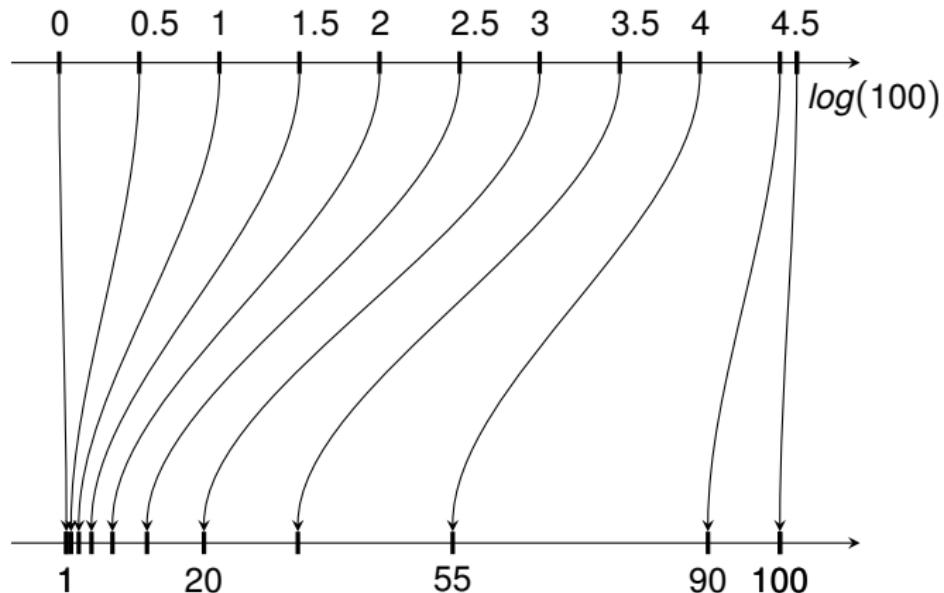
# PARAMETER TRANSFORMATION

What is our transformation doing?



# PARAMETER TRANSFORMATION

What is our transformation doing?



# PARAMETER TRANSFORMATION

Tuning again...

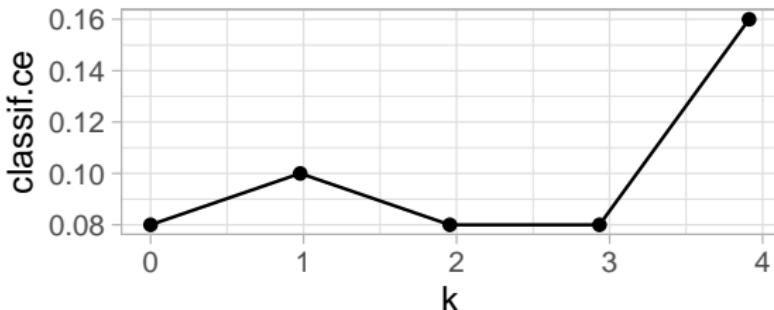
# PARAMETER TRANSFORMATION

Tuning again...

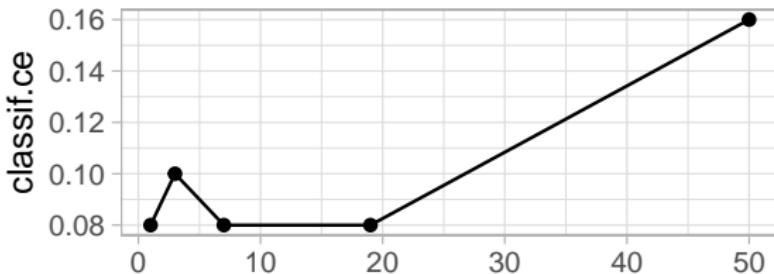
```
inst$result  
  
#>      k learner_param_vals  x_domain classif.ce  
#> 1: 2.9          <list[2]> <list[1]>      0.08  
  
inst$result$x_domain  
  
#> [[1]]  
#> [[1]]$k  
#> [1] 19
```

# PARAMETER TRANSFORMATION

```
ggplot(as.data.table(inst$archive), aes(x = k, y = classif.ce)) +  
  geom_line() + geom_point()
```



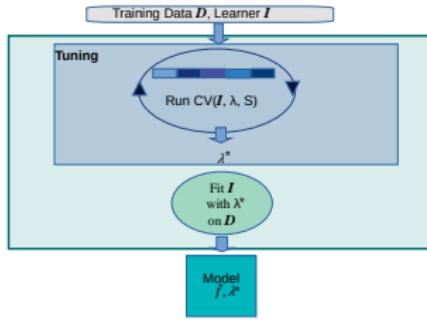
```
ggplot(as.data.table(inst$archive), aes(x = x_domain_k, y = classif.ce)) +  
  geom_line() + geom_point()
```



# **Nested Resampling**

# NESTED RESAMPLING

- Need to perform nested resampling to estimate tuned learner performance
- ⇒ Treat tuning as if it were a Learner!
  - Training:
    - ➊ Tune model using (inner) resampling
    - ➋ Train final model with best parameters on all (i.e. outer resampling) data
  - Predicting: Just use final model



# NESTED RESAMPLING

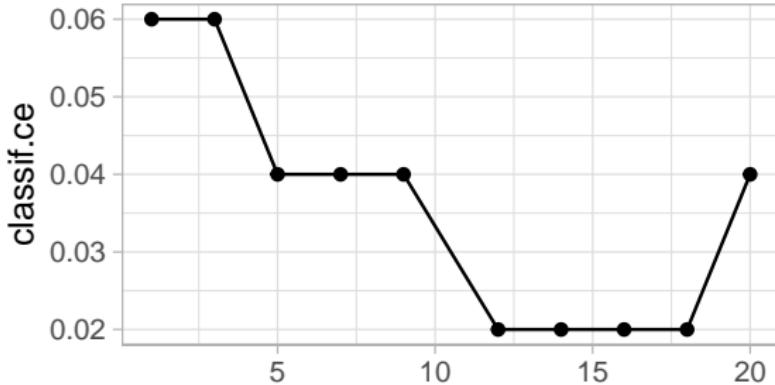
```
optlrn = AutoTuner$new(  
  learner = lrn("classif.kknn", kernel = "rectangular"),  
  resampling = rsmp("holdout"), measure = msr("classif.ce"),  
  terminator = trm("none"),  
  tuner = tnr("grid_search", resolution = 10),  
  search_space = searchspace_knn)
```

```
optlrn$train(tsk("iris"))
```

```
optlrn$model$learner  
  
#> <LearnerClassifKKNN:classif.kknn>  
#> * Model: list  
#> * Parameters: kernel=rectangular, k=18  
#> * Packages: kknn  
#> * Predict Type: response  
#> * Feature types: logical, integer, numeric, factor, ordered  
#> * Properties: multiclass, two-class
```

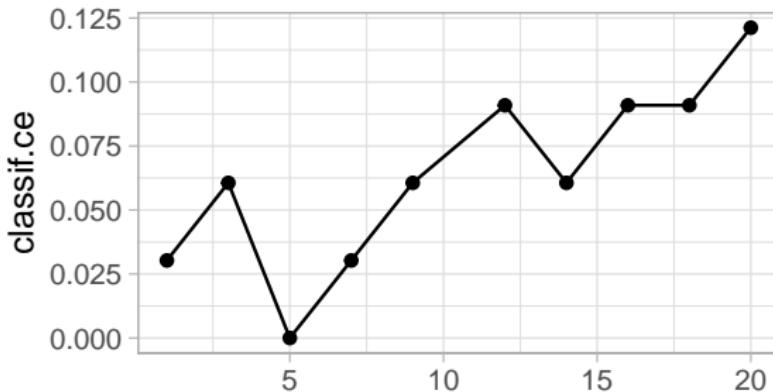
# NESTED RESAMPLING

```
archive = as.data.table(optlrn$tuning_instance$archive)
ggplot(archive, aes(x = k, y = classif.ce)) +
  geom_line() + geom_point() + xlab("")
```



# NESTED RESAMPLING

```
rr = resample(task = tsk("iris"), learner = optlrn,
  resampling = rsmp("holdout"), store_models = TRUE)
archive = as.data.table(rr$learners[[1]]$tuning_instance$archive)
ggplot(archive, aes(x = k, y = classif.ce)) +
  geom_line() + geom_point() + xlab("")
```



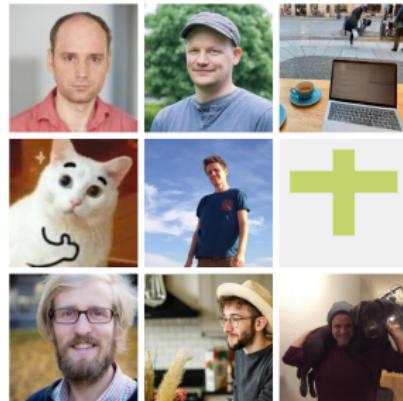
# Machine Learning Pipelines in R

---



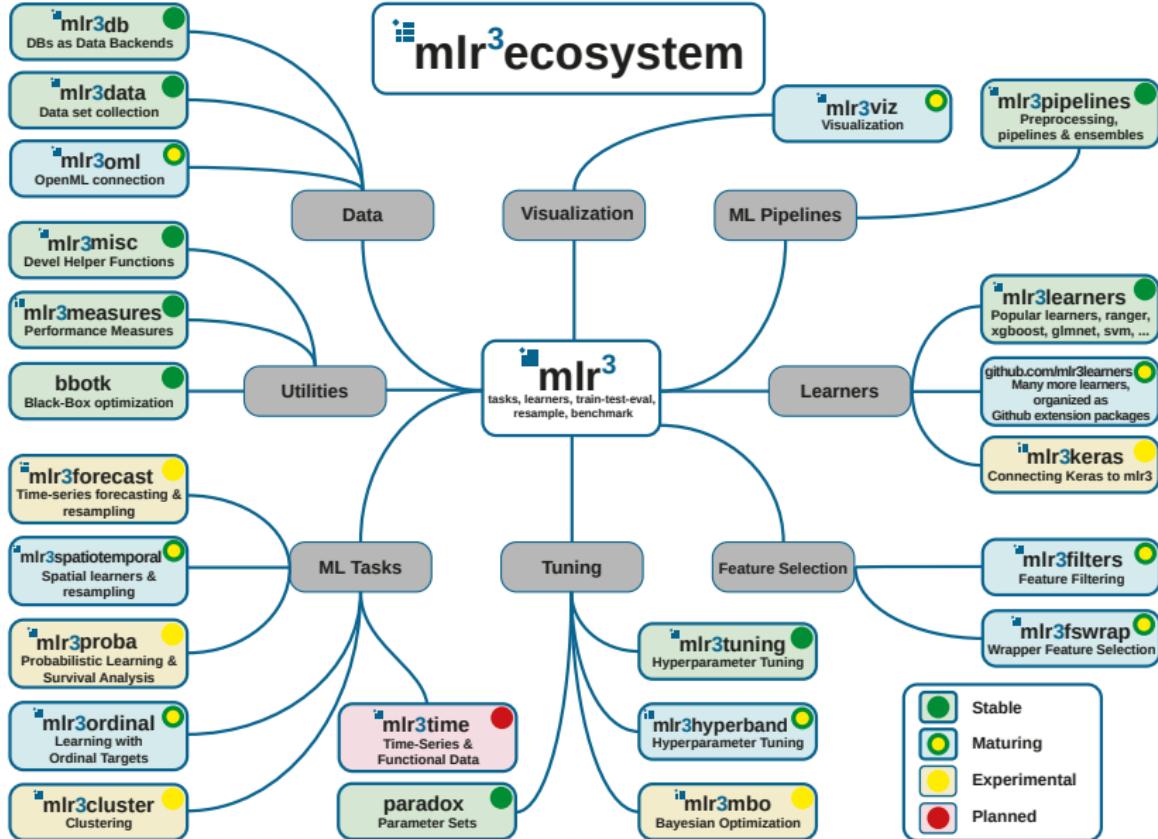
<https://mlr-org.com/>

<https://github.com/mlr-org>



---

**Bernd Bischl, Michel Lang, Martin Binder, Florian Pfisterer, Jakob Richter,  
Patrick Schratz, Lennart Schneider, Raphael Sonabend, Marc Becker**



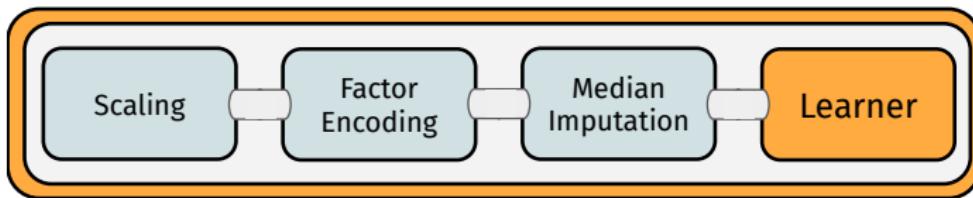
# **Intro**

# MLR3PIPELINES

## Machine Learning Workflows:

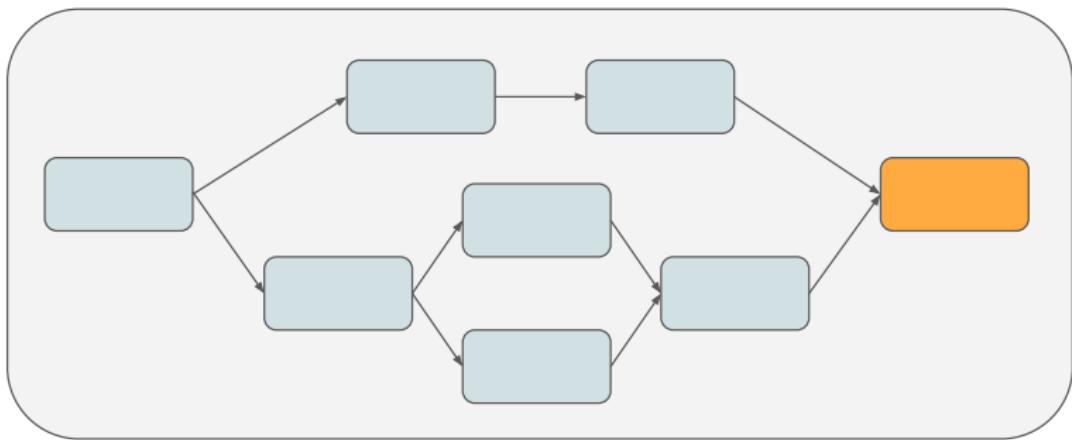
- **Preprocessing:** Feature extraction, feature selection, missing data imputation,...
- **Ensemble methods:** Model averaging, model stacking
- **mlr3:** modular model fitting

⇒ **mlr3pipelines:** modular ML workflows



# MACHINE LEARNING WORKFLOWS

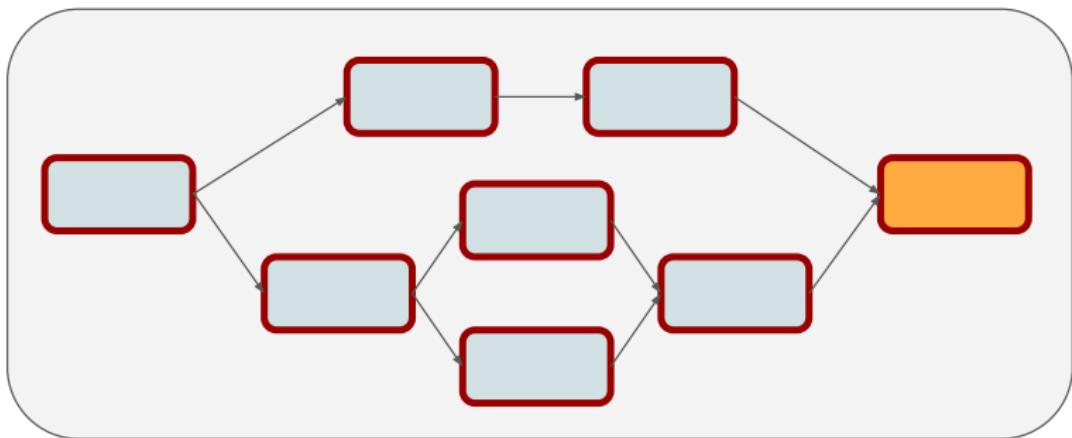
- what do they look like?



# MACHINE LEARNING WORKFLOWS

– what do they look like?

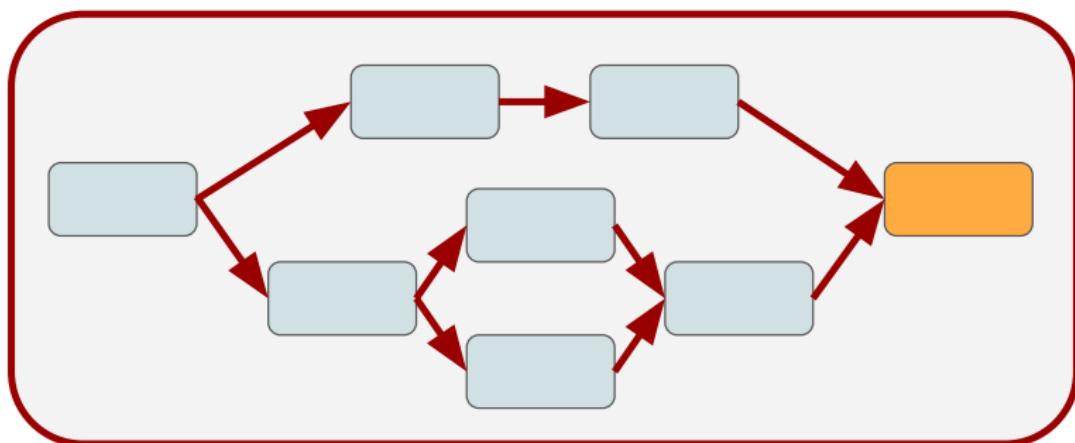
- **Building blocks:** *what is happening? → PipeOp*



# MACHINE LEARNING WORKFLOWS

– what do they look like?

- **Building blocks:** *what is happening?* → PipeOp
- **Structure:** *in what sequence is it happening?* → Graph



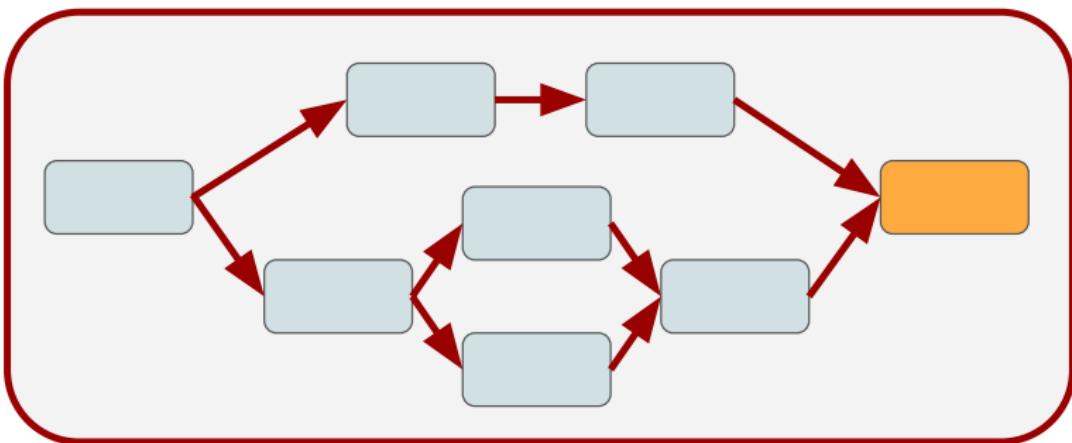
# MACHINE LEARNING WORKFLOWS

– what do they look like?

- **Building blocks:** what is happening? → PipeOp

- **Structure:** in what *sequence* is it happening? → Graph

⇒ Graph: PipeOps as **nodes** with **edges** (data flow) between them

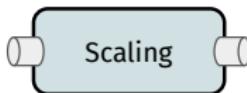


# PipeOps

# THE BUILDING BLOCKS

## PipeOp: Single Unit of Data Operation

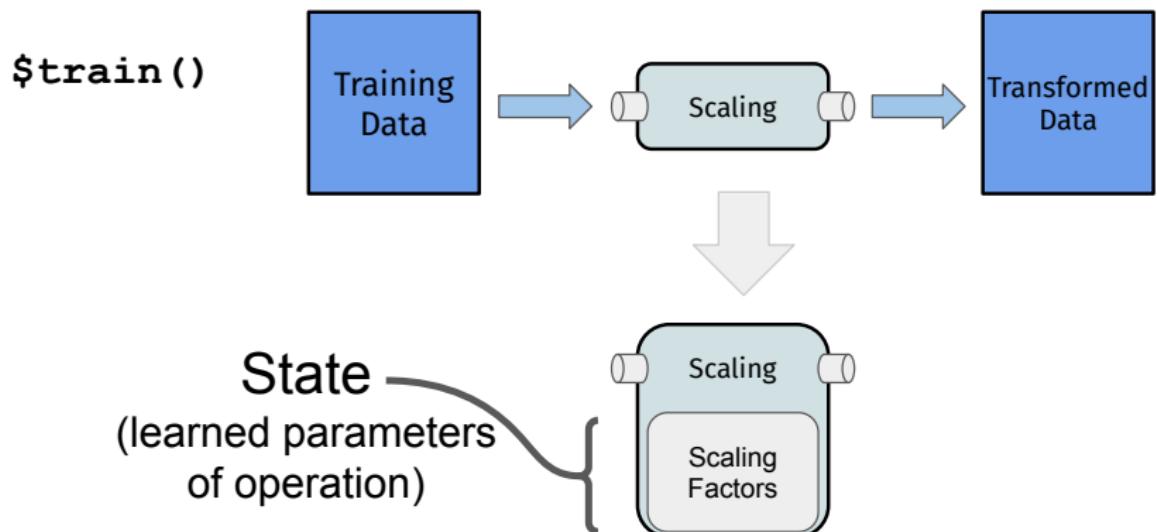
- `pip = po("scale")` to construct



# THE BUILDING BLOCKS

## PipeOp: Single Unit of Data Operation

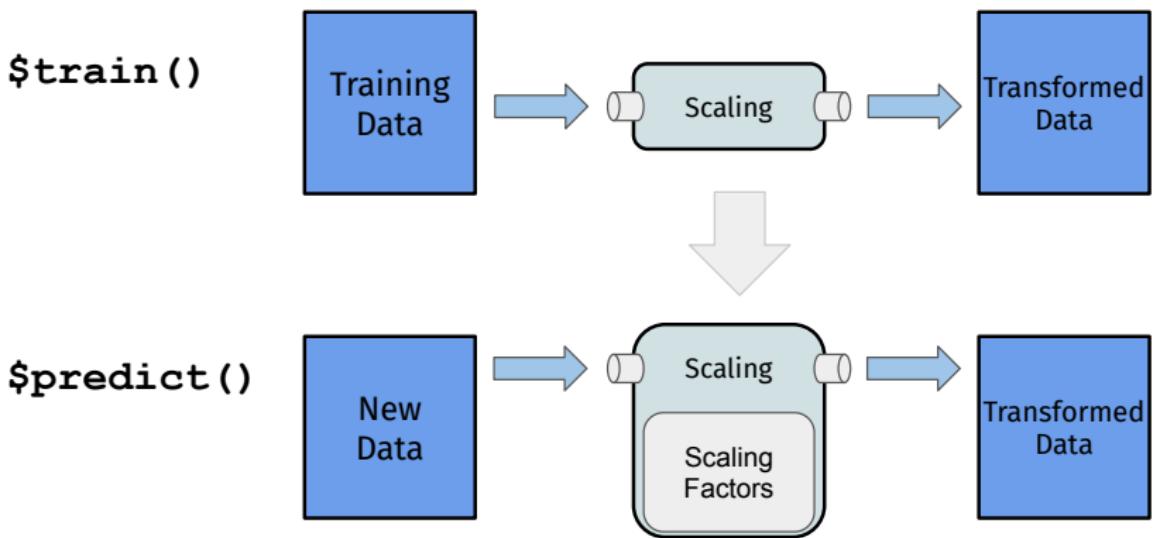
- `pip = po("scale")` to construct
- `pip$train()`: process data and create `pip$state`



# THE BUILDING BLOCKS

## PipeOp: Single Unit of Data Operation

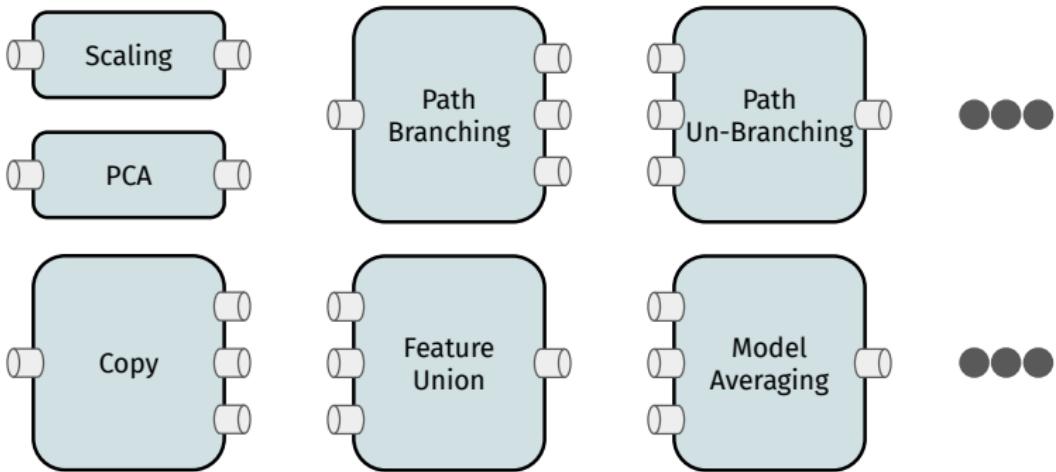
- `pip = po("scale")` to construct
- `pip$train()`: process data and create `pip$state`
- `pip$predict()`: process data depending on the `pip$state`



# THE BUILDING BLOCKS

## PipeOp: Single Unit of Data Operation

- `pip = po("scale")` to construct
- `pip$train()`: process data and create `pip$state`
- `pip$predict()`: process data depending on the `pip$state`
- Multiple inputs or multiple outputs



# THE BUILDING BLOCKS

```
po = po("scale")
trained = po$train(list(task))
trained$output$head(3)

#>   Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#> 1: setosa     -1.3      -1.3      -0.9      1.02
#> 2: setosa     -1.3      -1.3      -1.1     -0.13
#> 3: setosa     -1.4      -1.3      -1.4      0.33
```

# THE BUILDING BLOCKS

```
po = po("scale")
trained = po$train(list(task))
trained$output$head(3)

#>   Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#> 1: setosa     -1.3      -1.3     -0.9      1.02
#> 2: setosa     -1.3      -1.3     -1.1     -0.13
#> 3: setosa     -1.4      -1.3     -1.4      0.33

head(po$state, 2)

#> $center
#> Petal.Length  Petal.Width Sepal.Length  Sepal.Width
#>          3.8        1.2        5.8        3.1
#>
#> $scale
#> Petal.Length  Petal.Width Sepal.Length  Sepal.Width
#>          1.77       0.76       0.83       0.44
```

# THE BUILDING BLOCKS

```
po = po("scale")
trained = po$train(list(task))
trained$output$head(3)

#>   Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#> 1: setosa     -1.3      -1.3      -0.9       1.02
#> 2: setosa     -1.3      -1.3      -1.1      -0.13
#> 3: setosa     -1.4      -1.3      -1.4       0.33
```

```
smalltask = task$clone()
smalltask = smalltask$filter(1:3)
pred = po$predict(list(smalltask))
pred$output$data()

#>   Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#> 1: setosa     -1.3      -1.3      -0.9       1.02
#> 2: setosa     -1.3      -1.3      -1.1      -0.13
#> 3: setosa     -1.4      -1.3      -1.4       0.33
```

# PIPEOPS SO FAR

```
mlr_pipeops$keys()

#> [1] "boxcox"                      "branch"                  "chunk"
#> [4] "classbalancing"                "classifavg"               "classweights"
#> [7] "colapply"                     "collapsefactors"        "colroles"
#> [10] "copy"                        "datefeatures"            "encode"
#> [13] "encodeimpact"                 "encodelmer"                "featureunion"
#> [16] "filter"                      "fixfactors"                "histbin"
#> [19] "ica"                         "imputeconstant"          "imputehist"
#> [22] "imputearner"                 "imputemean"                "imputemedian"
#> [25] "imputemode"                  "imputeoor"                  "imputesample"
#> [28] "kernelpca"                   "learner"                  "learner_cv"
#> [31] "missind"                     "modelmatrix"                "multiplicityexply"
#> [34] "multiplicityimply"           "mutate"                    "nmf"
#> [37] "nop"                         "ovrsplit"                  "ovrunite"
#> [40] "pca"                          "proxy"                     "quantilebin"
#> [43] "randomprojection"           "randomresponse"           "regravg"
#> [46] "removeconstants"             "renamecolumns"              "replicate"
#> [49] "scale"                        "scalemaxabs"                "scalerange"
#> [52] "select"                      "smote"                     "spatialsign"
#> [55] "subsample"                   "targetinvert"                "targetmutate"
#> [58] "targettrafoscalerange"       "textvectorizer"              "threshold"
#> [...]
```

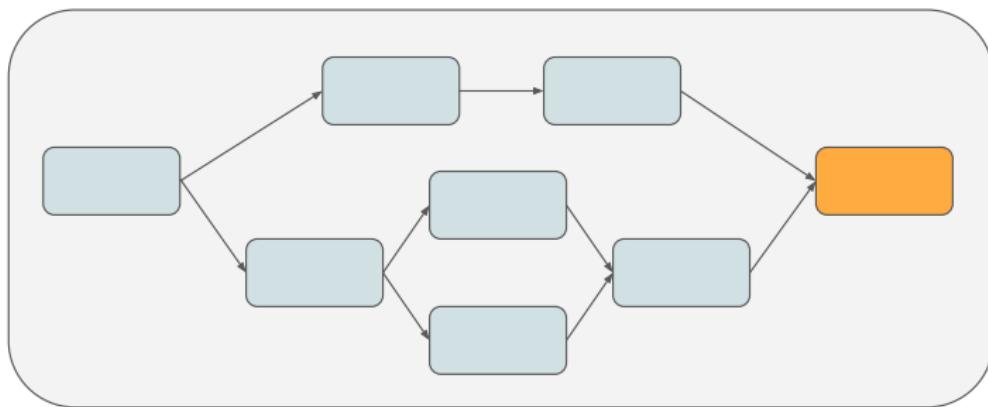
# PIPEOPS SO FAR AND PLANNED

- Simple data preprocessing operations (scaling, Box Cox, Yeo Johnson, PCA, ICA)
- Missing value imputation (sampling, mean, median, mode, new level, ...)
- Feature selection (by name, by type, using filter methods)
- Categorical data encoding (one-hot, treatment, impact)
- Sampling (subsampling for speed, sampling for class balance)
- Ensemble methods on Predictions (weighted average, possibly learned weights)
- Branching (simultaneous branching, alternative branching)
- Combination of data: `featureunion`
- Text processing
- Date processing
- Time series and spatio-temporal data (*planned*)
- Multi-output and ordinal targets (*planned*)
- Outlier detection (*planned*)

# **Graph Operations**

# THE STRUCTURE

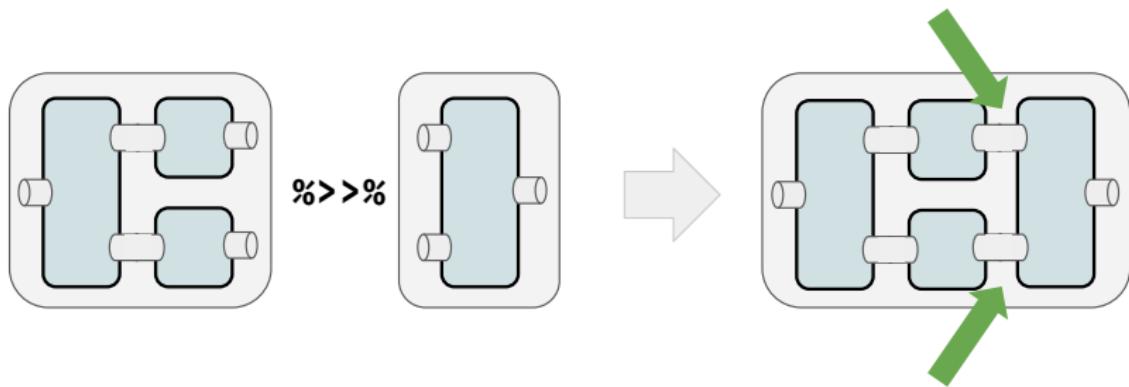
## Graph Operations



# THE STRUCTURE

## Graph Operations

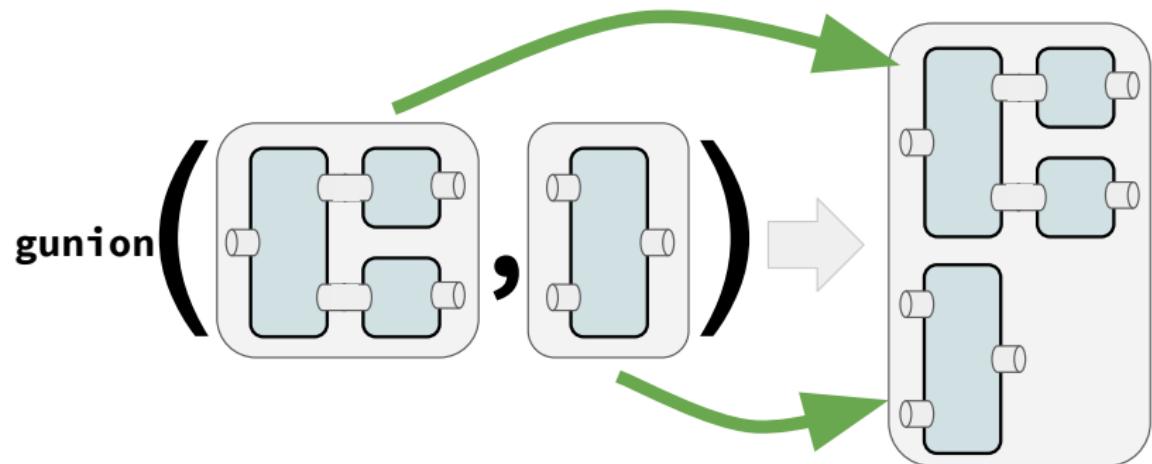
- The `%>>%`-operator concatenates Graphs and PipeOps



# THE STRUCTURE

## Graph Operations

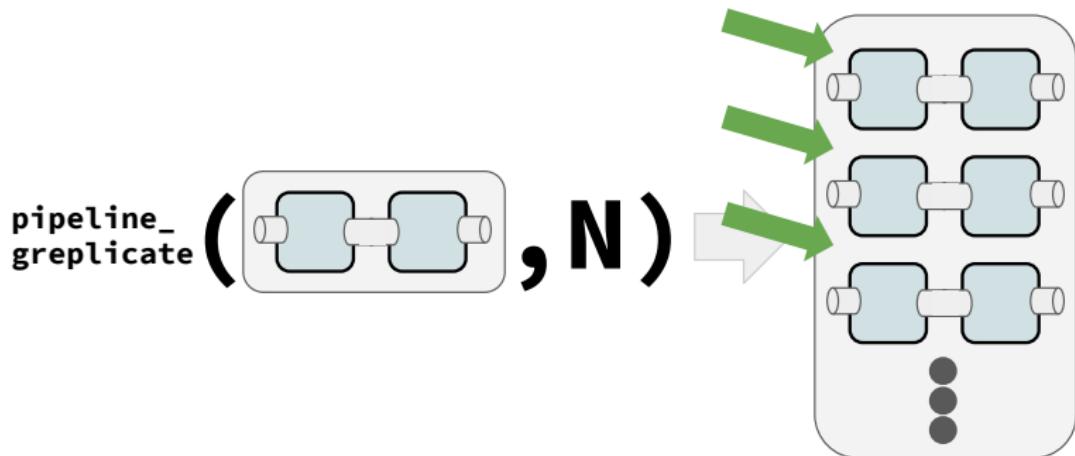
- The `%>>%`-operator concatenates Graphs and PipeOps
- The `gunion()`-function unites Graphs and PipeOps



# THE STRUCTURE

## Graph Operations

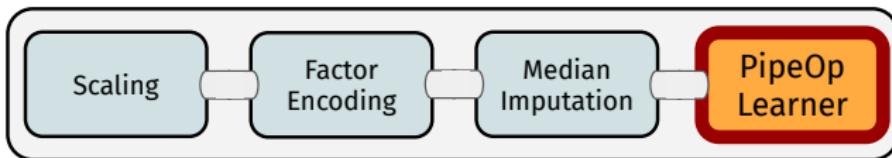
- The `%>>%`-operator concatenates Graphs and PipeOps
- The `gunion()`-function unites Graphs and PipeOps
- The `pipeline_greplicate()`-function unites copies of Graphs and PipeOps



# LEARNERS AND GRAPHS

## PipeOpLearner

- Learner as a PipeOp
- Fits a model, output is Prediction



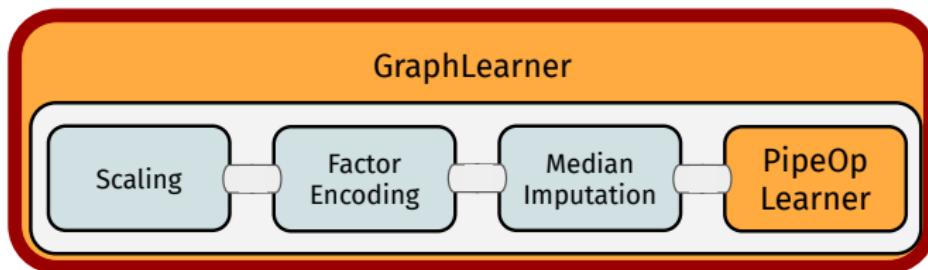
# LEARNERS AND GRAPHS

## PipeOpLearner

- Learner as a PipeOp
- Fits a model, output is Prediction

## GraphLearner

- Graph as a Learner
- All benefits of mlr3: **resampling, tuning, nested resampling, ...**

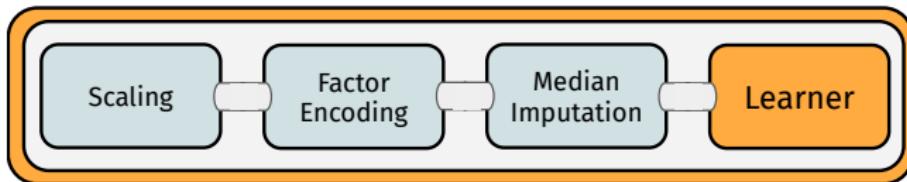


# **Linear Pipelines**

# MLR3PIPELINES IN ACTION

## Linear Preprocessing Pipeline

```
graph_pp = po("scale") %>>%  
  po("encode") %>>%  
  po("imputemedian") %>>%  
  lrn("classif.rpart")
```

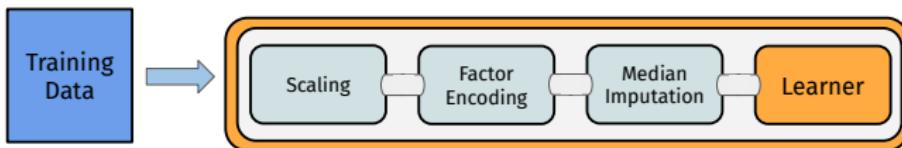


# MLR3PIPELINES IN ACTION

## Linear Preprocessing Pipeline

- `train()`ing: Data propagates and creates \$states

```
glrn = GraphLearner$new(graph_pp)  
glrn$train(task)
```

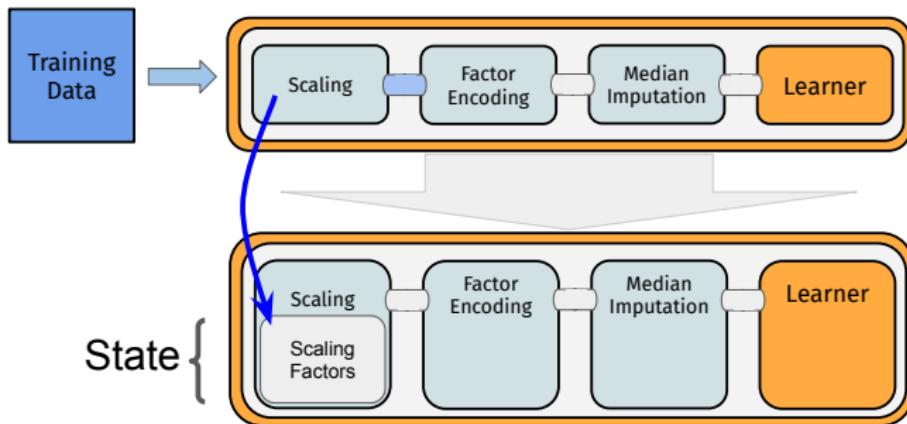


# MLR3PIPELINES IN ACTION

## Linear Preprocessing Pipeline

- `train()`ing: Data propagates and creates \$states

```
glrn = GraphLearner$new(graph_pp)  
glrn$train(task)
```

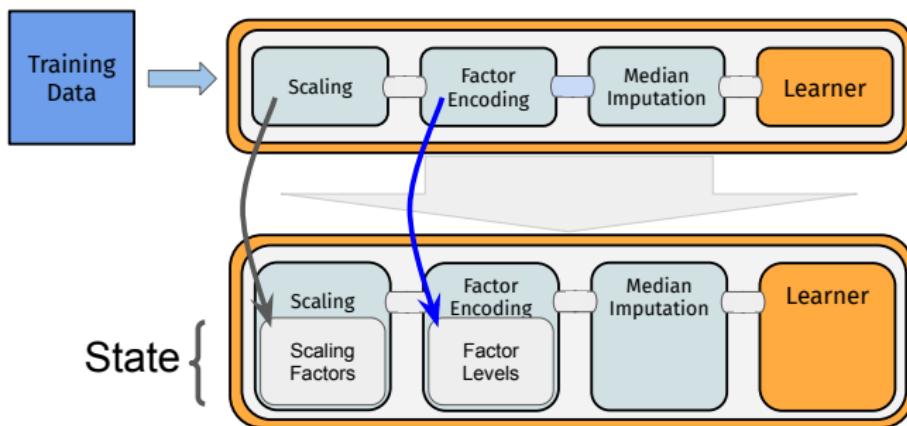


# MLR3PIPELINES IN ACTION

## Linear Preprocessing Pipeline

- `train()`ing: Data propagates and creates \$states

```
glrn = GraphLearner$new(graph_pp)  
glrn$train(task)
```

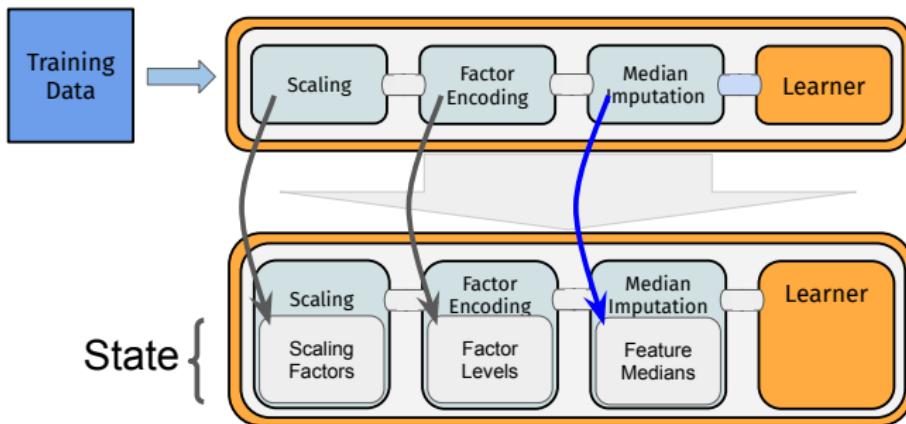


# MLR3PIPELINES IN ACTION

## Linear Preprocessing Pipeline

- `train()`ing: Data propagates and creates \$states

```
glrn = GraphLearner$new(graph_pp)  
glnr$train(task)
```

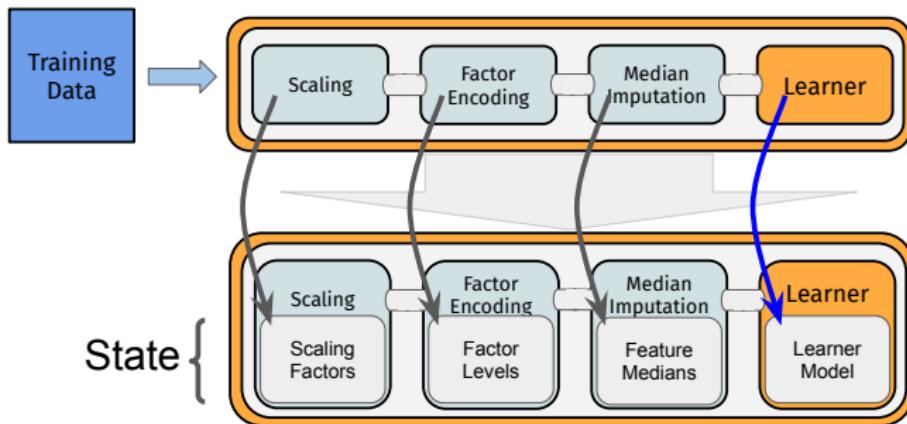


# MLR3PIPELINES IN ACTION

## Linear Preprocessing Pipeline

- `train()`ing: Data propagates and creates \$states

```
glrn = GraphLearner$new(graph_pp)  
glnr$train(task)
```

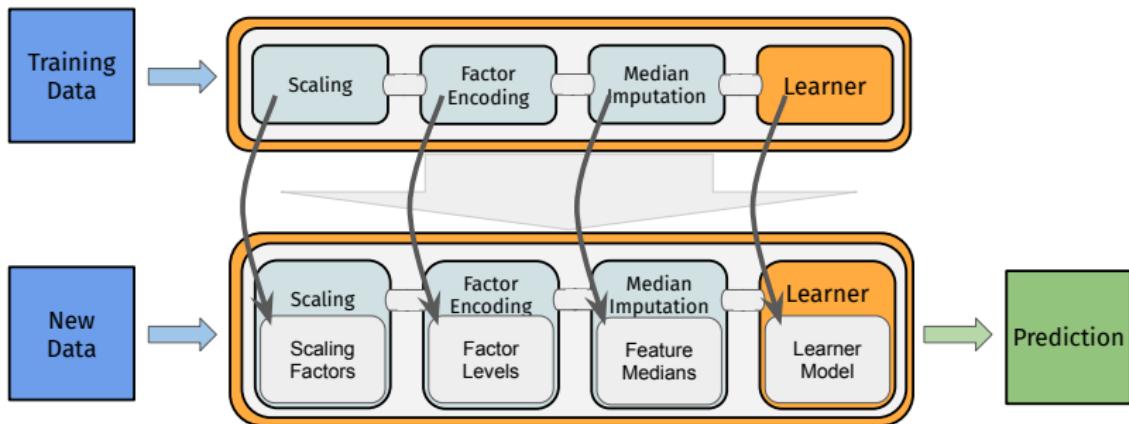


# MLR3PIPELINES IN ACTION

## Linear Preprocessing Pipeline

- `train()`ing: Data propagates and creates `$states`
- `predict()`ition: Data propagates, uses `$states`

```
glrn$predict(task)
```



# MLR3PIPELINES IN ACTION

## Linear Preprocessing Pipeline `scale %>>% encode %>>% impute %>>% rpart`

- Setting / retrieving parameters: `$param_set`

```
graph_pp$pipeops$scale$param_set$values$center = FALSE
```

# MLR3PIPELINES IN ACTION

## Linear Preprocessing Pipeline `scale %>>% encode %>>% impute %>>% rpart`

- Setting / retrieving parameters: `$param_set`

```
graph_pp$pipeops$scale$param_set$values$center = FALSE
```

- Retrieving state: `$state` of individual PipeOps (*after \$train()*)

```
graph_pp$pipeops$scale$state$scale  
#> Petal.Length  Petal.Width Sepal.Length  Sepal.Width  
#>          4.2          1.4          5.9          3.1
```

# MLR3PIPELINES IN ACTION

## Linear Preprocessing Pipeline `scale %>>% encode %>>% impute %>>% rpart`

- Setting / retrieving parameters: `$param_set`

```
graph_pp$pipeops$scale$param_set$values$center = FALSE
```

- Retrieving state: `$state` of individual PipeOps (*after \$train()*)

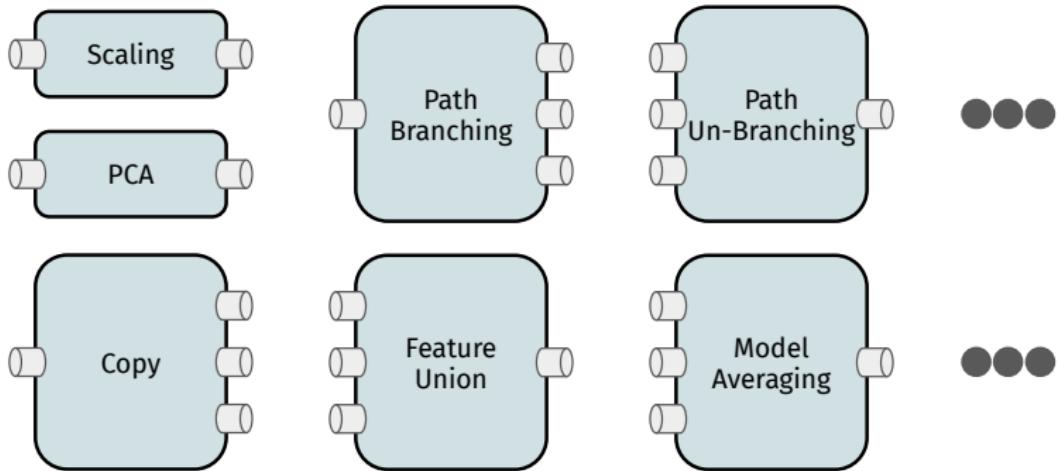
```
graph_pp$pipeops$scale$state$scale  
#> Petal.Length Petal.Width Sepal.Length Sepal.Width  
#>          4.2        1.4        5.9        3.1
```

- Retrieving intermediate results: `$.result` (set debug option before)

```
graph_pp$pipeops$scale$.result[[1]]$head(3)  
#>   Species Petal.Length Petal.Width Sepal.Length Sepal.Width  
#> 1:  setosa     0.34      0.14      0.86      1.13  
#> 2:  setosa     0.34      0.14      0.83      0.97  
#> 3:  setosa     0.31      0.14      0.79      1.03
```

# **Nonlinear Pipelines**

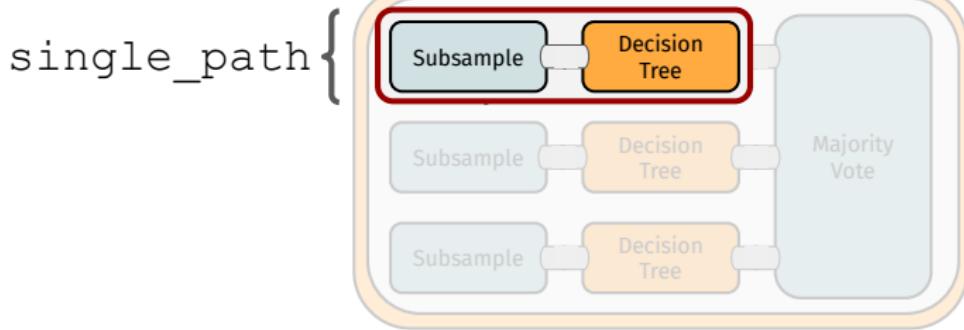
# PIPEOPS WITH MULTIPLE INPUTS / OUTPUTS



# MLR3PIPELINES IN ACTION

## Ensemble Method: Bagging

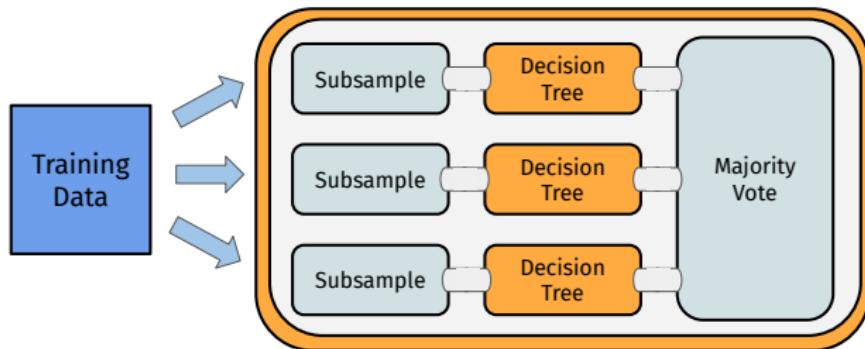
```
single_path = po("subsample") %>>% lrn("classif.rpart")
```



# MLR3PIPELINES IN ACTION

## Ensemble Method: Bagging

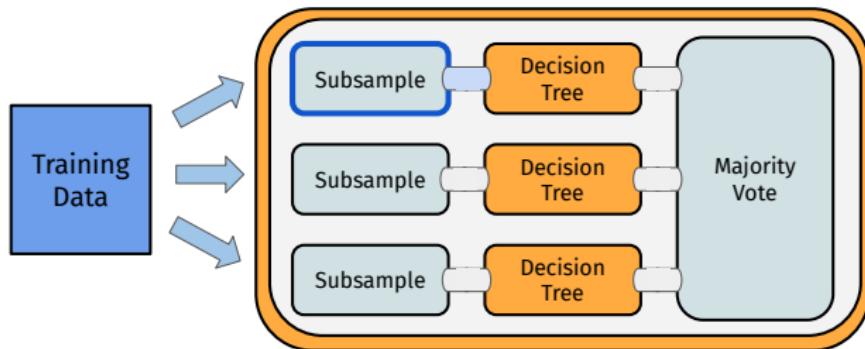
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



# MLR3PIPELINES IN ACTION

## Ensemble Method: Bagging

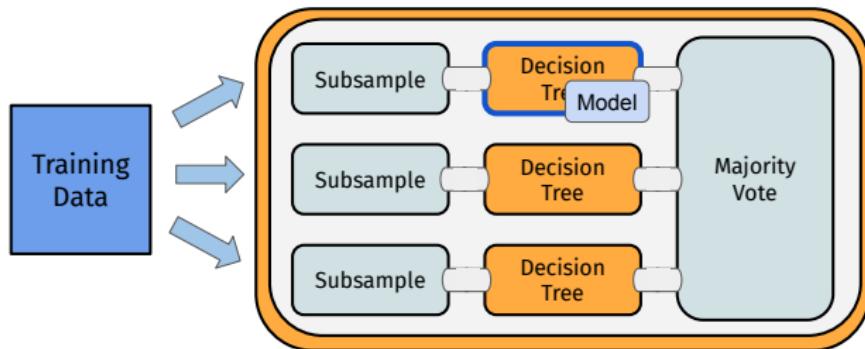
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



# MLR3PIPELINES IN ACTION

## Ensemble Method: Bagging

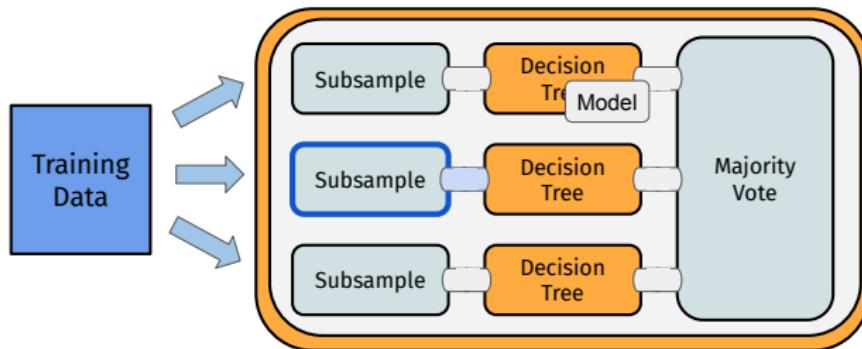
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



# MLR3PIPELINES IN ACTION

## Ensemble Method: Bagging

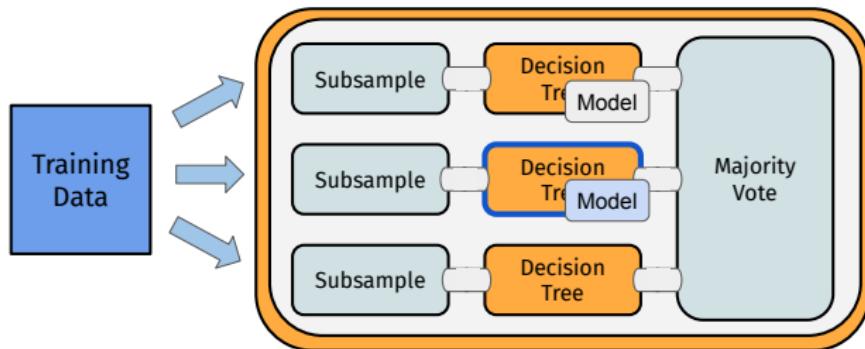
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



# MLR3PIPELINES IN ACTION

## Ensemble Method: Bagging

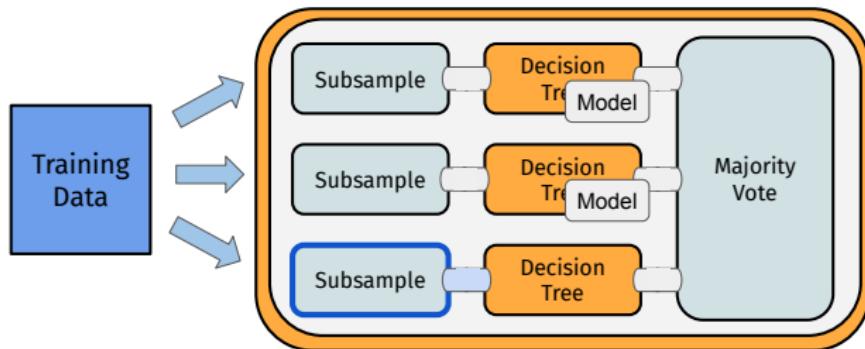
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



# MLR3PIPELINES IN ACTION

## Ensemble Method: Bagging

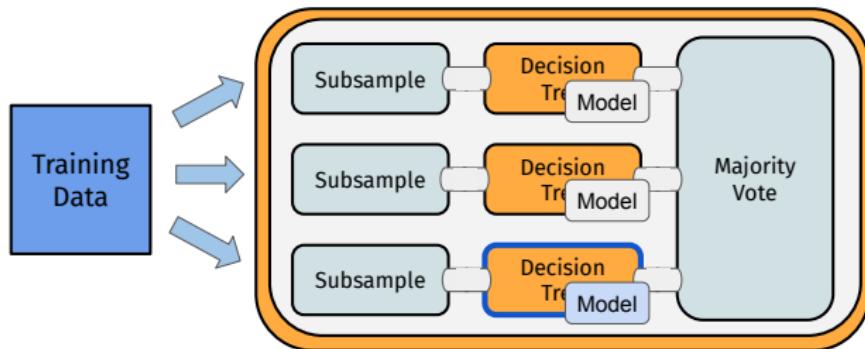
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



# MLR3PIPELINES IN ACTION

## Ensemble Method: Bagging

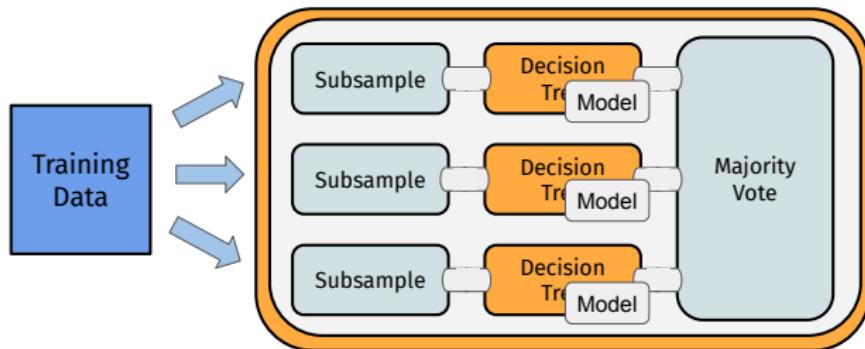
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



# MLR3PIPELINES IN ACTION

## Ensemble Method: Bagging

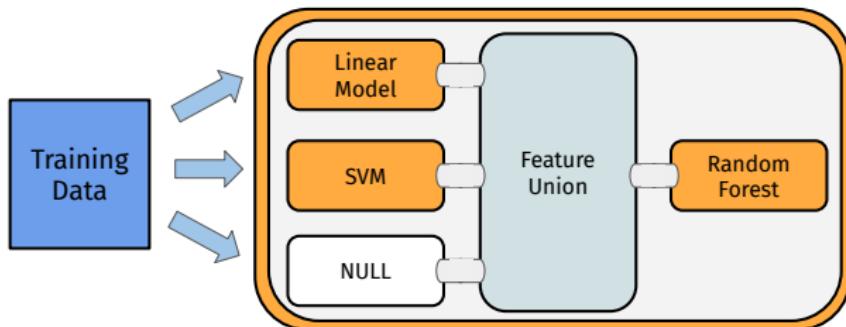
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



# MLR3PIPELINES IN ACTION

## Ensemble Method: Stacking

```
graph_stack = gunion(list(
  po("learner_cv", learner = lrn("regr.lm")),
  po("learner_cv", learner = lrn("regr.svm")),
  po("nop")))) %>>%
po("featureunion") %>>%
lrn("regr.ranger")
```

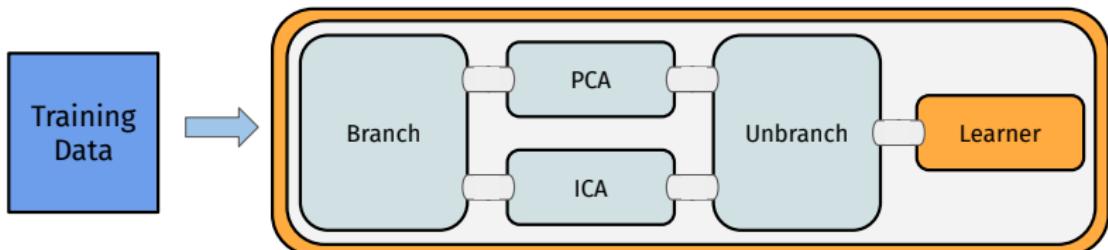


# MLR3PIPELINES IN ACTION

## Branching

```
graph_branch = ppl("branch", list(  
    pca = po("pca"),  
    ica = po("ica")) %>>%  
    lrn("classif.kknn")
```

Execute only one of several alternative paths

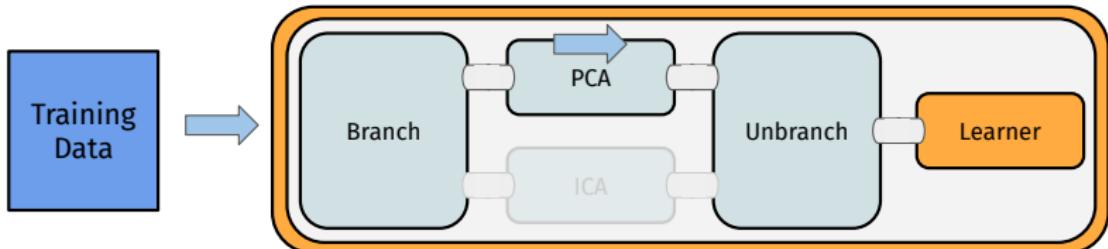


# MLR3PIPELINES IN ACTION

## Branching

```
graph_branch = ppl("branch", list(  
  pca = po("pca"),  
  ica = po("ica"))) %>>%  
  lrn("classif.kknn")
```

```
> graph_branch$pipeops$branch$  
  param_set$values$selection = "pca"
```

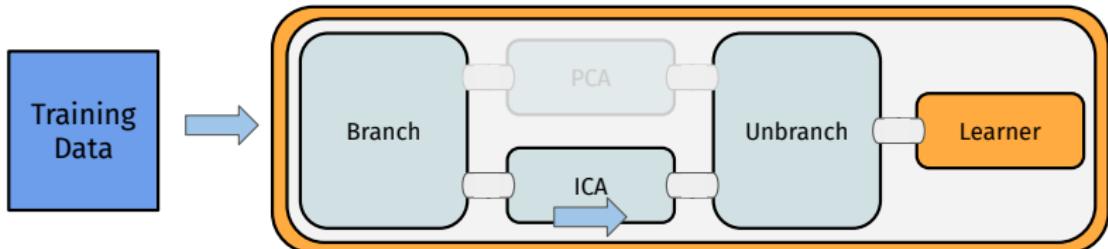


# MLR3PIPELINES IN ACTION

## Branching

```
graph_branch = ppl("branch", list(  
  pca = po("pca"),  
  ica = po("ica"))) %>>%  
  lrn("classif.kknn")
```

```
> graph_branch$pipeops$branch$  
  param_set$values$selection = "ica"
```



# **Targeting Columns**

# TARGETING COLUMNS

Two ways of restricting actions to individual columns:

- Individual PipeOps: `affect_columns` parameter
  - Subgraphs, `po("select")`, and `po("featureunion")`
- ⇒ Both make use of column Selectors

Suppose we only want PCA on some columns of our data:

```
task$data(1:9)

#>   Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#> 1: setosa      1.4       0.2      5.1      3.5
#> 2: setosa      1.4       0.2      4.9      3.0
#> 3: setosa      1.3       0.2      4.7      3.2
#> 4: setosa      1.5       0.2      4.6      3.1
#> 5: setosa      1.4       0.2      5.0      3.6
#> 6: setosa      1.7       0.4      5.4      3.9
#> 7: setosa      1.4       0.3      4.6      3.4
#> 8: setosa      1.5       0.2      5.0      3.4
#> 9: setosa      1.4       0.2      4.4      2.9
```

# TARGETING COLUMNS

Two ways of restricting actions to individual columns:

- Individual PipeOps: `affect_columns` parameter
- Subgraphs, `po("select")`, and `po("featureunion")`  
⇒ Both make use of column Selectors

Using `affect_columns`:

```
sel = selector_grep("^Sepal")

partial_pca = po("pca", affect_columns = sel)

result = partial_pca$train(list(task))

result[[1]]$data(1:3)

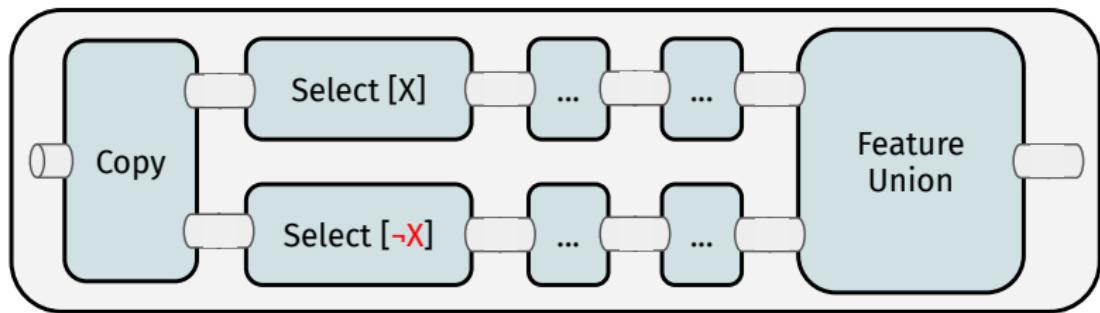
#>   Species    PC1     PC2 Petal.Length Petal.Width
#> 1:  setosa -0.78  0.378         1.4        0.2
#> 2:  setosa -0.94 -0.137         1.4        0.2
#> 3:  setosa -1.15  0.045         1.3        0.2
```

# TARGETING COLUMNS

Two ways of restricting actions to individual columns:

- Individual PipeOps: `affect_columns` parameter
- Subgraphs, `po("select")`, and `po("featureunion")`  
⇒ Both make use of column Selectors

Using `po("select")`:



# TARGETING COLUMNS

Two ways of restricting actions to individual columns:

- Individual PipeOps: `affect_columns` parameter
- Subgraphs, `po("select")`, and `po("featureunion")`  
⇒ Both make use of column Selectors

Using `po("select")`:

```
sel = selector_grep("^Sepal")
selcomp = selector_invert(sel)

partial_pca = gunion(list(
  po("select", selector = sel) %>>% po("pca"),
  po("select", selector = selcomp, id = "select2")))) %>>%
  po("featureunion")

partial_pca$train(task)[[1]]$data(1:3)

#>   Species    PC1     PC2 Petal.Length Petal.Width
#> 1:  setosa -0.78  0.378         1.4        0.2
#> 2:  setosa -0.94 -0.137         1.4        0.2
#> 3:  setosa -1.15  0.045         1.3        0.2
```

# **“Pipelines” Dictionary & Short Form**

# “PIPELINES” DICTIONARY & SHORT FORM

Many frequently used *patterns* for pipelines

- Making Learners robust to bad data (imputation + feature encoding + ...)
- Bagging
- Branching

# “PIPELINES” DICTIONARY & SHORT FORM

## Many frequently used *patterns* for pipelines

- Making Learners robust to bad data (imputation + feature encoding + ...)
- Bagging
- Branching

## Collection of these is in mlr3pipelines

```
head(as.data.table(mlr_pipeops), 5)[, list(key, input.num, output.num)]  
  
#>          key input.num output.num  
#> 1:      boxcox      1         1  
#> 2:     branch      1        NA  
#> 3:      chunk      1        NA  
#> 4: classbalancing      1         1  
#> 5: classifavg      NA         1
```

# “PIPELINES” DICTIONARY & SHORT FORM

## Many frequently used *patterns* for pipelines

- Making Learners robust to bad data (imputation + feature encoding + ...)
- Bagging
- Branching

## Collection of these is in `mlr3pipelines`

`po()` accesses the `mlr_pipeops` “Dictionary”.

```
pca = po("pca")
pca

#> PipeOp: <pca> (not trained)
#> values: <list()>
#> Input channels <name [train type, predict type]>:
#>   input [Task,Task]
#> Output channels <name [train type, predict type]>:
#>   output [Task,Task]
```

# **AutoML with mlr3pipelines**

# AUTOML <3 PIPELINES

- AutoML: Automatic Machine Learning

# AUTOML <3 PIPELINES

- AutoML: Automatic Machine Learning
- Let the algorithm make decisions about
  - ➊ *what learner* to use,
  - ➋ *what preprocessing* to use, and
  - ➌ *what hyperparameters* to use.

# AUTOML <3 PIPELINES

- AutoML: Automatic Machine Learning
- Let the algorithm make decisions about
  - ➊ *what learner* to use,
  - ➋ *what preprocessing* to use, and
  - ➌ *what hyperparameters* to use.
- (1) and (2) are decisions about *graph structure* in `mlr3pipelines`

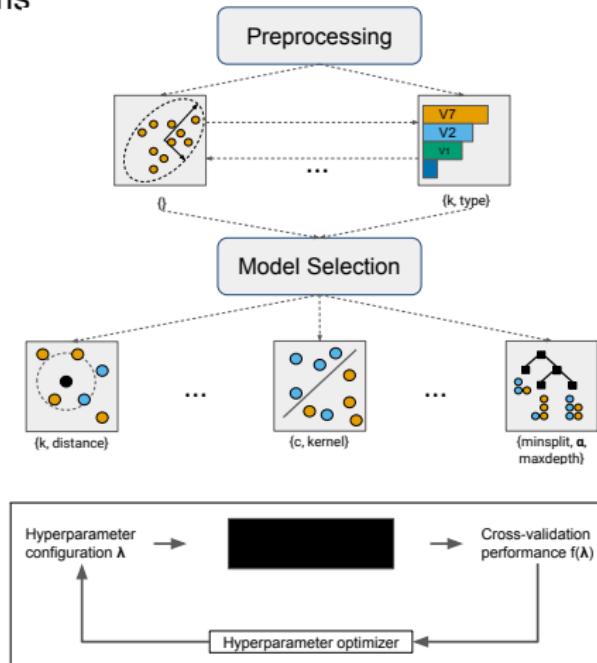
# AUTOML <3 PIPELINES

- AutoML: Automatic Machine Learning
  - Let the algorithm make decisions about
    - ➊ *what learner* to use,
    - ➋ *what preprocessing* to use, and
    - ➌ *what hyperparameters* to use.
  - (1) and (2) are decisions about *graph structure* in `mlr3pipelines`
- ⇒ The problem reduces to **pipelines + parameter tuning**

# AUTOML WITH MLR3PIPELINES

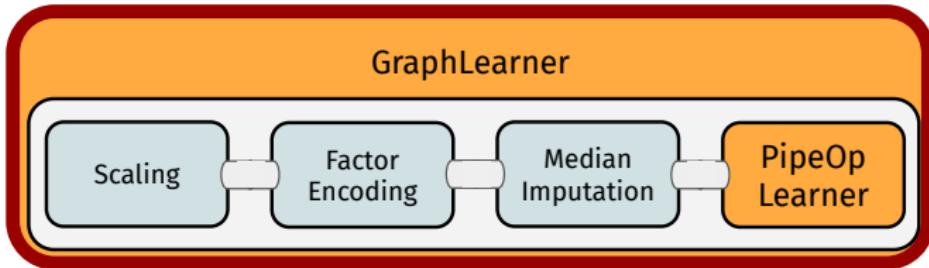
## AutoML in a Nutshell

- Preprocessing steps
- ML Algorithms
- Tuner



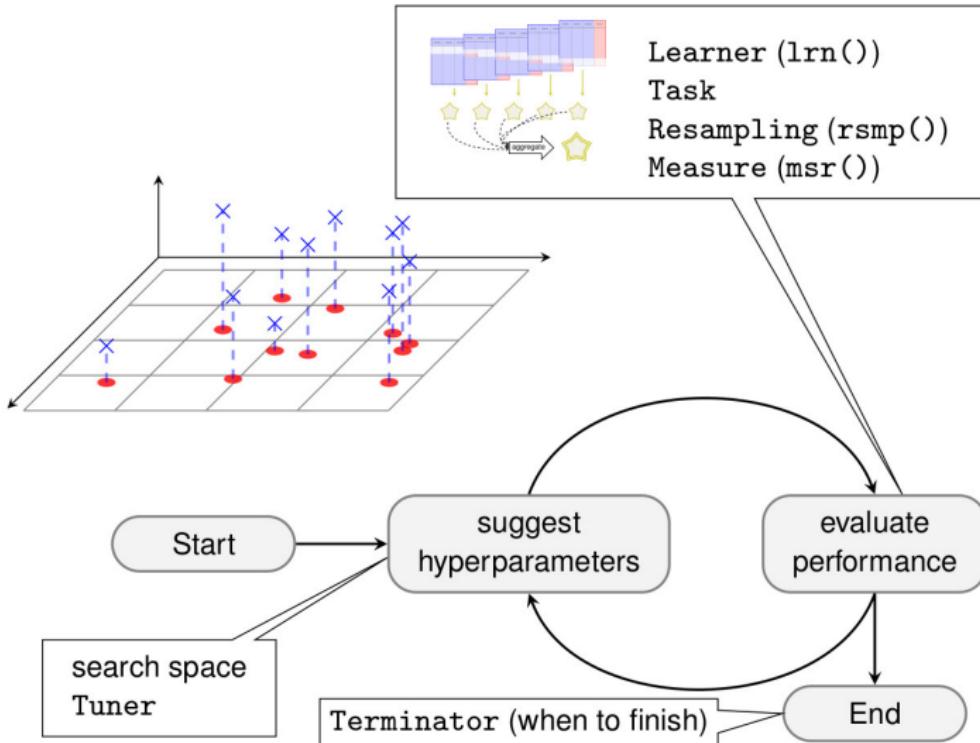
# GRAPHLEARNER

- Graph as a Learner
- All benefits of `mlr3`: **resampling**, **tuning**, **nested resampling**, ...



```
graph_pp = po("scale") %>>% po("encode") %>>%
  po("imputemedian") %>>% lrn("classif.rpart")
glrn = GraphLearner$new(graph_pp)
glrn$train(task)
glrn$predict(task)
resample(task, glrn, rsmp("cv", folds = 3))
```

# TUNING

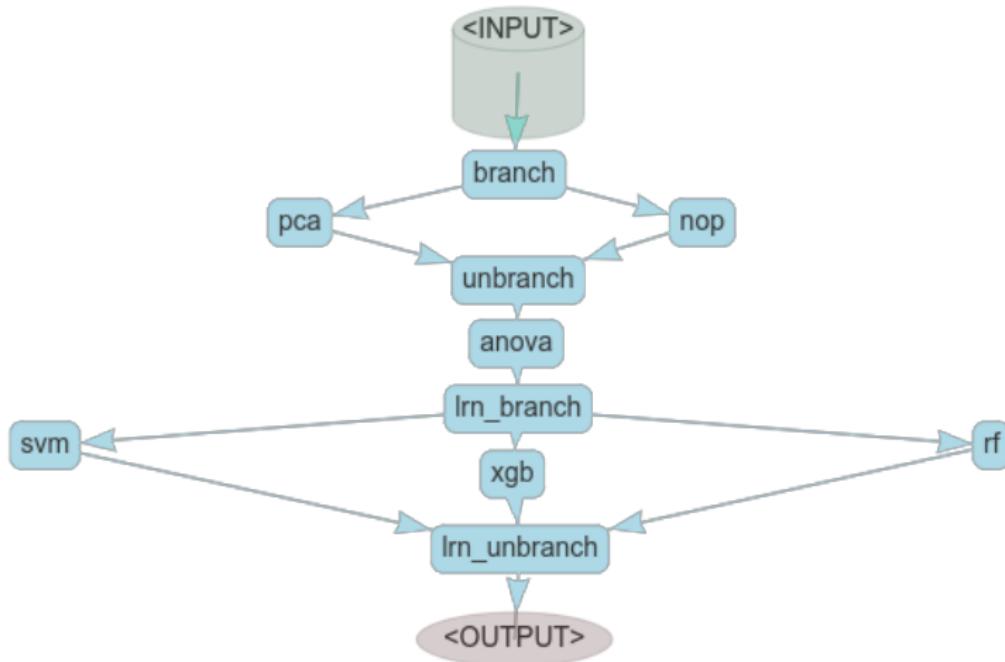


# PIPELINES TUNING

- Works **exactly** as in basic `mlr3` / `mlr3tuning`
- PipeOps have *hyperparameters* (using `paradox` pkg)
- Graphs have hyperparameters of all components *combined*
- ⇒ Joint **tuning** and nested CV of complete graph

```
p1 = ppl("branch", list(  
  "pca" = po("pca"),  
  "nothing" = po("nop")))  
p2 = flt("anova")  
p3 = ppl("branch", list(  
  "svm" = lrn("classif.svm", id = "svm", kernel = "radial",  
    type = "C-classification"),  
  "xgb" = lrn("classif.xgboost", id = "xgb"),  
  "rf" = lrn("classif.ranger", id = "rf"))  
, prefix_branchops = "lrn_")  
gr = p1 %>>% p2 %>>% p3  
glrn = GraphLearner$new(gr)
```

# PIPELINES TUNING



# PIPELINES TUNING

```
ps = ParamSet$new(list(
  ParamFct$new("branch.selection", levels = c("pca", "nothing")),
  ParamDbl$new("anova.filter.frac", lower = 0.1, upper = 1),
  ParamFct$new("lrn_branch.selection", levels = c("svm", "xgb", "rf")),
  ParamInt$new("rf.mtry", lower = 1L, upper = 20L),
  ParamInt$new("xgb.nrounds", lower = 1, upper = 500),
  ParamDbl$new("svm.cost", lower = -12, upper = 4),
  ParamDbl$new("svm.gamma", lower = -12, upper = -1)))
ps$add_dep("rf.mtry", "lrn_branch.selection", CondEqual$new("rf"))
ps$add_dep("xgb.nrounds", "lrn_branch.selection", CondEqual$new("xgb"))
ps$add_dep("svm.cost", "lrn_branch.selection", CondEqual$new("svm"))
ps$add_dep("svm.gamma", "lrn_branch.selection", CondEqual$new("svm"))
ps$trafo = function(x, param_set) {
  if (x$lrn_branch.selection == "svm") {
    x$svm.cost = 2^x$svm.cost; x$svm.gamma = 2^x$svm.gamma
  }
  return(x)
}
inst = TuningInstanceSingleCrit$new(tsk("sonar"), glrn, rsmp("cv", folds=3),
  msr("classif.ce"), ps, trm("evals", n_evals = 10))
tnr("random_search")$optimize(inst)
```

## **mlr3(pipelines) Resources**

# MLR3(PIPELINES) RESOURCES

## mlr3 book

The screenshot shows the mlr3 book website at <https://mlr3book.mlr-org.com/pipelines.html>. The page title is "4 Pipelines". The content discusses mlr3pipelines as a dataflow programming toolkit, mentioning its use in "Dolphin" vignettes. It explains that machine learning workflows can be written as directed "Dolphin" Pipelines, which represent data flows between preprocessing, model fitting, and ensemble learning units in an expressive and intuitive language. A diagram illustrates a pipeline flow from "Scaling" to "Factor Encoding" to "Median Imputation" leading to a "Learner". Below the diagram, it says "Single computational steps can be represented as so-called Pipelops, which can then be connected with directed edges in a graph. The scope of mlr3pipelines is still growing. Currently supported features are:

<https://mlr3book.mlr-org.com/>

## mlr3 Use Case “Gallery”

The screenshot shows the mlr3 gallery website at <https://cheatsheets.mlr-org.com/>. The main page displays a section titled "A pipeline for the titanic data set - Advanced". It includes a bar chart comparing survival rates by gender. The sidebar lists categories such as basics, classification, regression, feature engineering, feature importance, filter, filtering, imbalanced data, imputation, mlr3, mlr3pipelines, mlr3tuning, regression, resampling, stacking, stratification, tuning, and visualization.

<https://mlr3gallery.mlr-org.com/>

## “cheat sheets”

The screenshot shows the mlr3 cheat sheets website at <https://cheatsheets.mlr-org.com/>. It features several "CHEAT SHEET" cards:

- Machine learning with mlr3 :: CHEAT SHEET**: Basic Task, Class Learner, Train & Predict.
- Hyperparameter Tuning with mlr3tuning :: CHEAT SHEET**: Terminology - When to stop, Tuner - Search Strategy, Tuner/Tuner - Tune before Train.
- Dataflow programming with mlr3pipelines :: CHEAT SHEET**: Pipelines, Graph, Graph Construction, GraphTuning, GraphLearner, Tuning, Feature Engineering.
- Machine learning with mlr3pipelines :: CHEAT SHEET**: Pipeline, Pipelines, Pipelines - Ingestion, Pipelines - Transformation, Pipelines - Processing, Pipelines - Evaluation, Pipelines - Persistence.
- Nonlinear Graphics**: Nonlinear Graphics - Overview, Nonlinear Graphics - Examples.

<https://cheatsheets.mlr-org.com/>

# OUTLOOK

## What is to come?

- `mlr3pipelines`: caching, parallelization
- Better **tuners**: Bayesian Optimization, Hyperband
- Survival and Forecasting (via `mlr3proba`, `mlr3forecast`)
- Deep Learning (via `mlr3keras`)

Thanks! Please ask questions!

# **Outro**

# MLR3PIPELINES

mlr3pipelines overview:

- Construct a PipeOp using `po()`
- Use Graph operators to connect them
  - `%>>%`—chain operations
  - `gunion()`—put operations in parallel
  - `pipeline_greplicate()`—put many copies of an operation in parallel
- Train/predict with the PipeOp or Graph using `$train()/$predict()`
- Inspect the trained state through `$state`
- Encapsulate the Graph in a GraphLearner for resampling, benchmarking, and tuning