



Introduction to Machine Learning

All slides

December 13, 2021

INTRODUCTION TO MACHINE LEARNING

ML Basics

Supervised Regression

Supervised Classification

Performance Evaluation

k-NN

Classification and Regression Trees (CART)

Random Forests

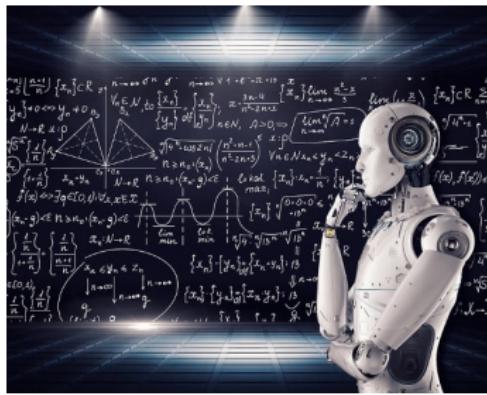
Tuning

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

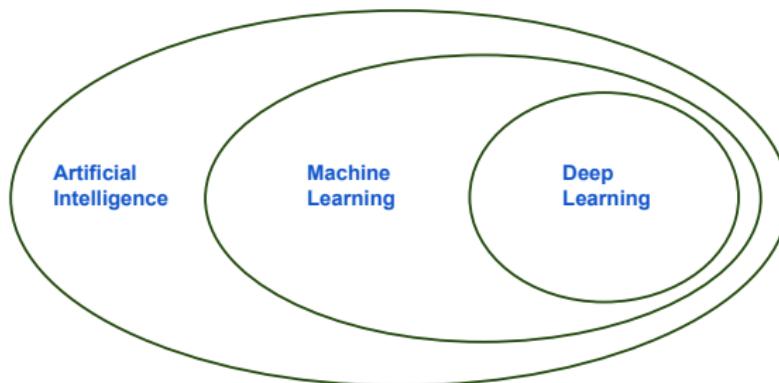
Image via www.vpnsrus.com

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



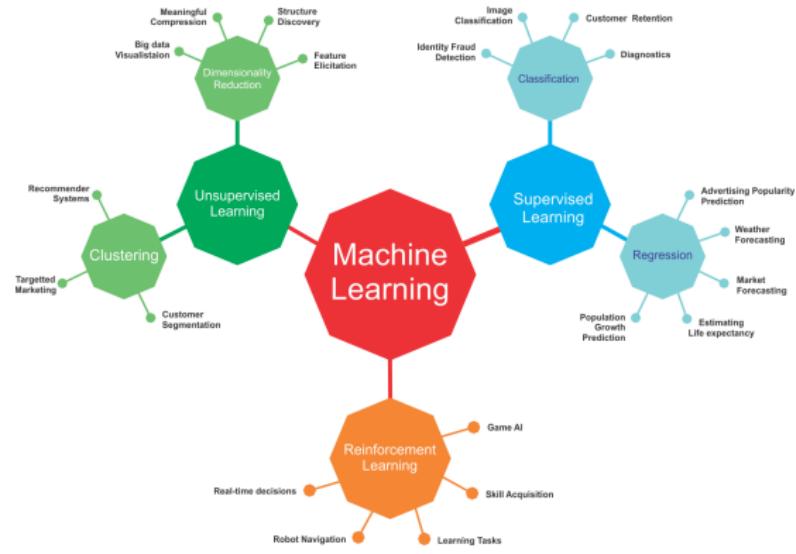
Many people are confused what these terms actually mean.

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, 1999

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.

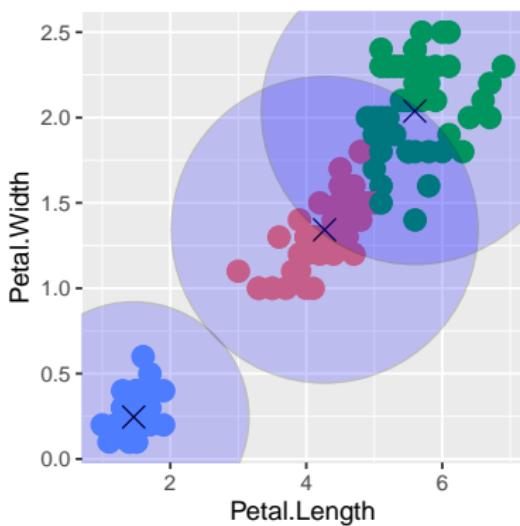
"Any sufficiently advanced technology is indistinguishable from magic."
Arthur C. Clarke's 3rd law

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

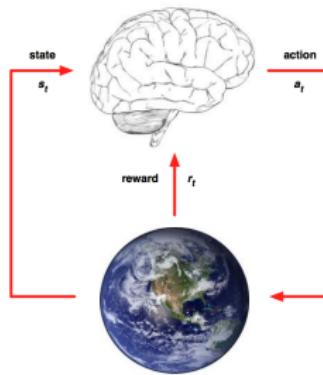
- Data without labels y
- Search for patterns within the inputs x
- *Unsupervised* as there is no “true” output we can optimize against



- 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.



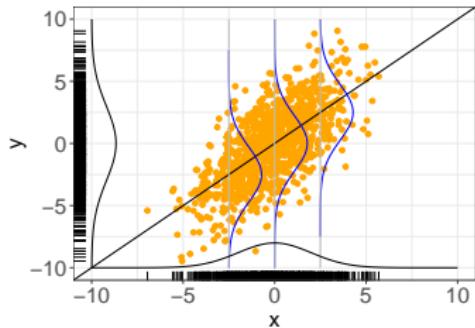
- Goal: Select actions to maximize future reward.
- Reward signals may be sparse, noisy and delayed.

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



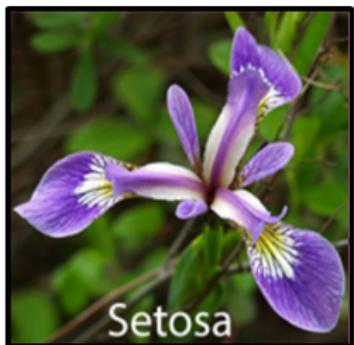
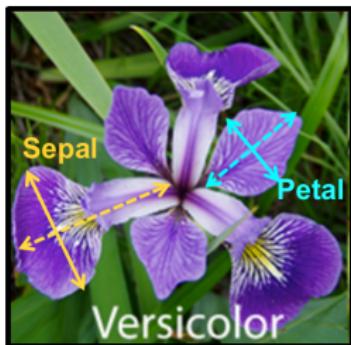
Learning goals

- 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].

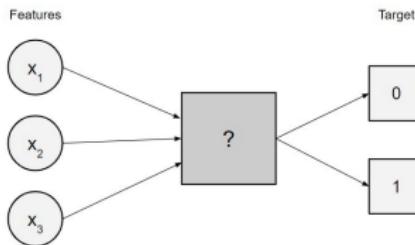


Source: <https://rpubs.com/vidhividhi/irisdataeda>

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**, i.e., cast them to numerical values.
- If not stated otherwise, we assume numerical features.

ENCODING FOR CATEGORICAL FEATURES

- We expand the representation of a feature x with k mutually exclusive categories from a scalar to a length- \tilde{k} vector with at most one element being 1, and 0 otherwise: $\mathbf{o}(x) = [\mathbb{I}(x = j)]_{j=1,2,\dots,\tilde{k}} \in \{0, 1\}^{\tilde{k}}$.
- Each entry of $\mathbf{o}(x)$ is treated as a separate feature.
- Two popular ways to do this are
 - **One-hot encoding:** $\tilde{k} = k$ dummies, so *exactly one* element is 1 ("hot").
E.g., $x \in \{a, b, c\} \mapsto \mathbf{o}(x) = (x_a, x_b, x_c)$, with $x_a = x_b = 0, x_c = 1$ and $\mathbf{o}(x) = (0, 0, 1)$ for $x = c$.
 - **Dummy encoding:** $\tilde{k} = k - 1$ dummies, so *at most one* element is 1, cutting the redundancy of one-hot encoding (necessary for learners that require non-singular input matrices, such as in linear regression).
E.g., $x \in \{a, b, c\} \mapsto \mathbf{o}(x) = (x_a, x_b)$ for reference category c , with $x_a = x_b = 0$ and $\mathbf{o}(x) = (0, 0)$ for $x = c$.
- For features with a natural **order** in their categories we resort to encodings that reflect this ordinality, e.g., a sequence of integer values.

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
unlabeled data	5.5	2.5	4.0	1.3	versicolor
	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

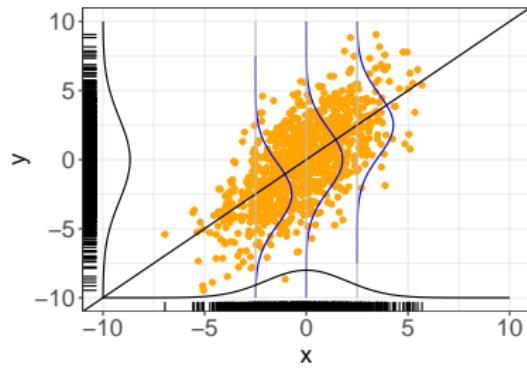
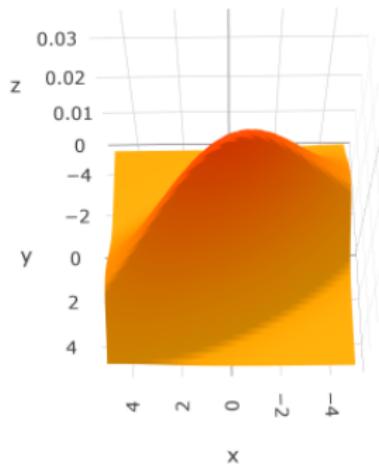
$$\mathbb{P}_{xy},$$

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.



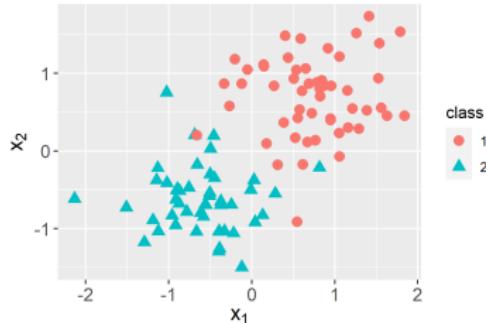
DATA-GENERATING PROCESS

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 θ 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



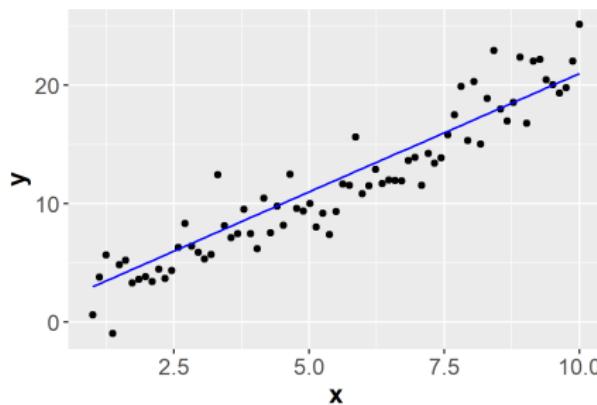
Learning goals

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

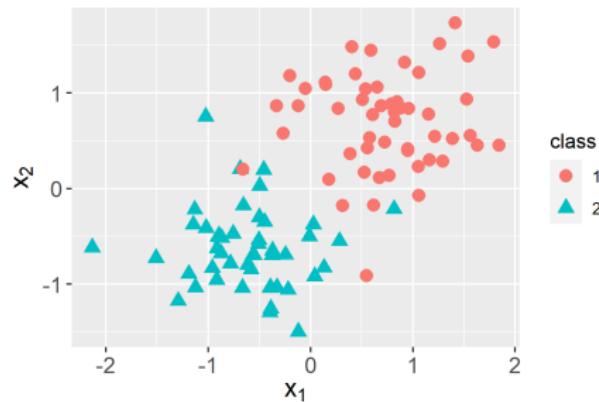
TASKS: REGRESSION VS CLASSIFICATION

- 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

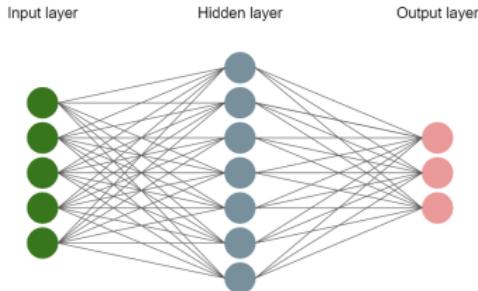


Probably *learn to predict*, but the company might be required to explain its predictions to its customers.

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

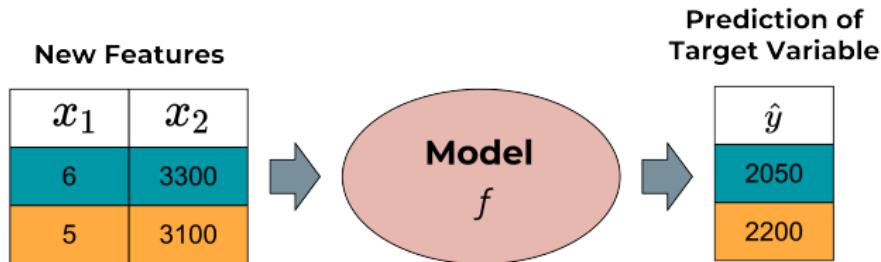
WHAT IS A MODEL?

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

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

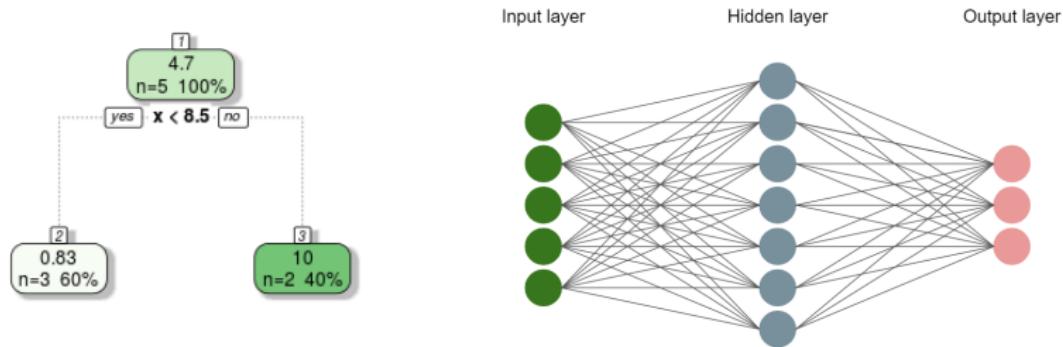
is a function that maps feature vectors to predicted target values.

- 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} :

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

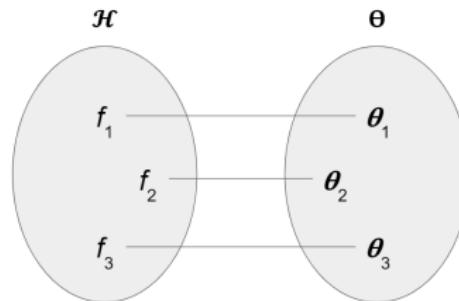
PARAMETRIZATION

- 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** Θ .
- They are our means of fixing a specific function from the family. Once set, our model is fully determined.
- Therefore, we can re-write \mathcal{H} as:

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

PARAMETRIZATION

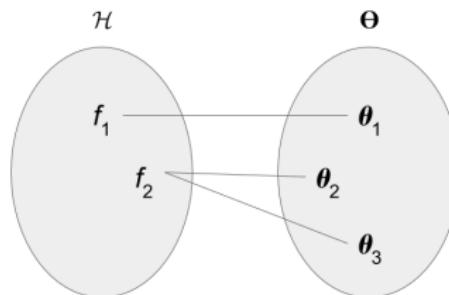
- This means: finding the optimal model is perfectly equivalent to finding the optimal set of parameter values.
- The relation between optimization over $f \in \mathcal{H}$ and optimization over $\theta \in \Theta$ allows us to operationalize our search for the best model via the search for the optimal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



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

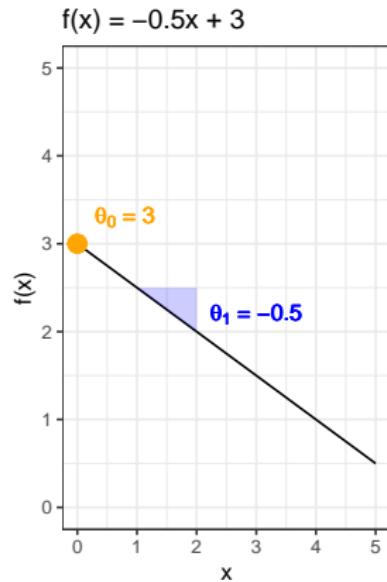
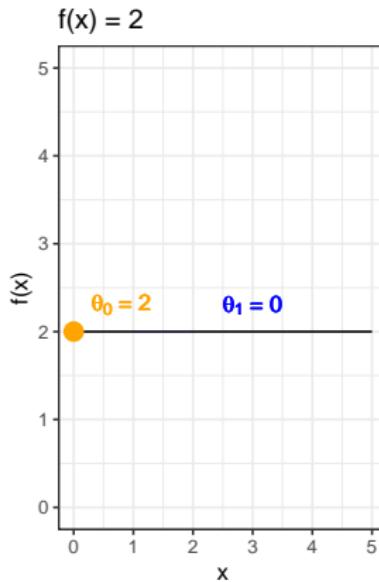
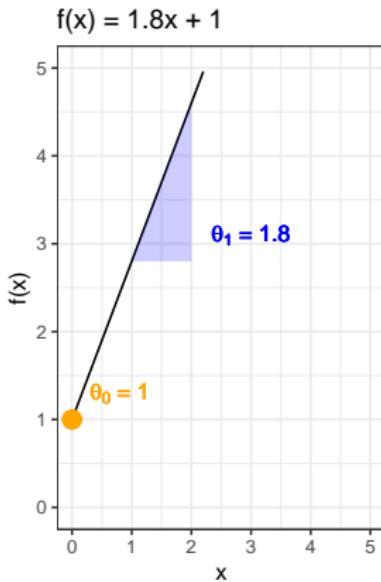
PARAMETRIZATION

- 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

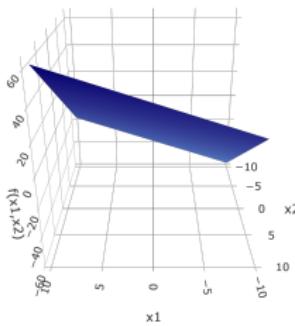
$$\mathcal{H} = \{f : f(\mathbf{x}) = \theta_0 + \theta_1 x, \theta \in \mathbb{R}^2\}$$



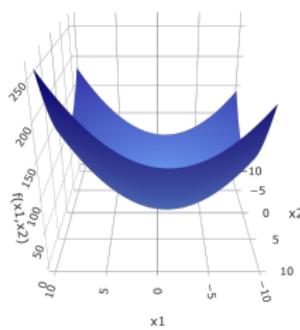
EXAMPLE: BIVARIATE QUADRATIC FUNCTIONS

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

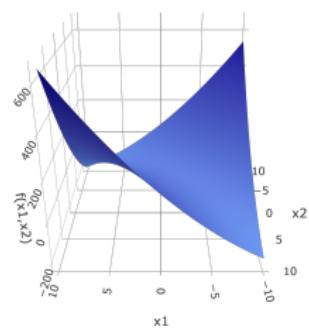
$$f(x) = 3 + 2x_1 + 4x_2$$



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



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



EXAMPLE: RBF NETWORK

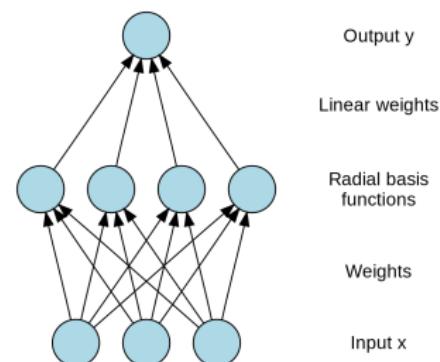
Radial basis function networks with Gaussian basis functions

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

where

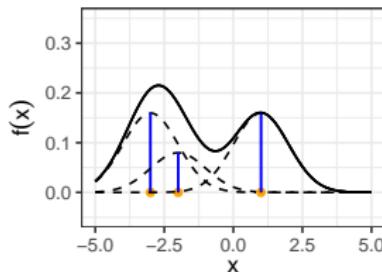
- a_i is the weight of the i -th neuron,
- \mathbf{c}_i its center vector, and
- $\rho(\|\mathbf{x} - \mathbf{c}_i\|) = \exp(-\beta \|\mathbf{x} - \mathbf{c}_i\|^2)$ is the i -th radial basis function with bandwidth $\beta \in \mathbb{R}$.

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

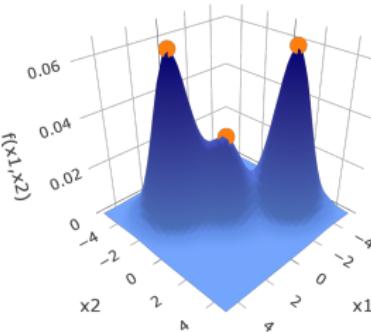


EXAMPLE: RBF NETWORK

Exemplary setting

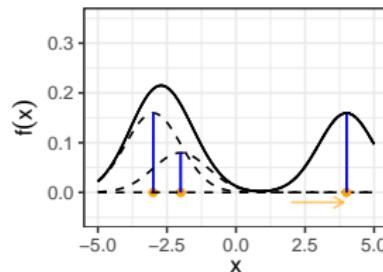


$$\begin{aligned} a_1 &= 0.4, a_2 = 0.2, a_3 = 0.4 \\ c_1 &= -3, c_2 = -2, c_3 = 1 \end{aligned}$$

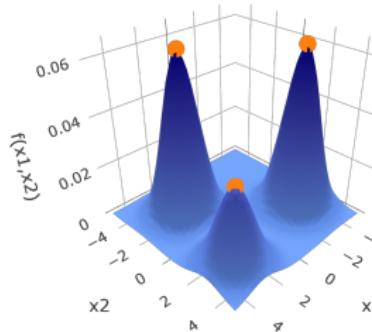


$$\begin{aligned} a_1 &= 0.4, a_2 = 0.2, a_3 = 0.4 \\ c_1 &= (2, -2), c_2 = (0, 0), \\ c_3 &= (-3, 2) \end{aligned}$$

Centers altered

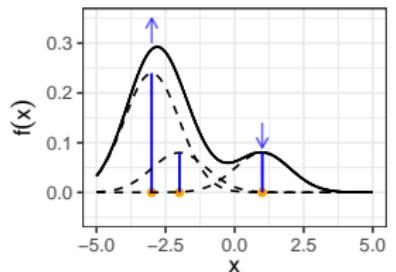


$$\begin{aligned} a_1 &= 0.4, a_2 = 0.2, a_3 = 0.4 \\ c_1 &= -3, c_2 = -2, c_3 = 1 \end{aligned}$$

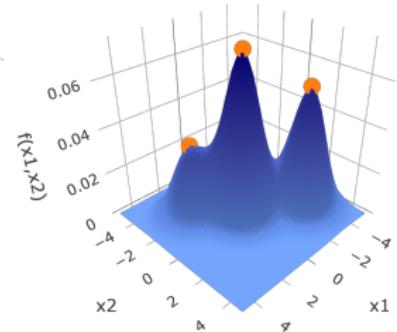


$$\begin{aligned} a_1 &= 0.4, a_2 = 0.2, a_3 = 0.4 \\ c_1 &= (2, -2), c_2 = (3, 3), \\ c_3 &= (-3, 2) \end{aligned}$$

Weights altered



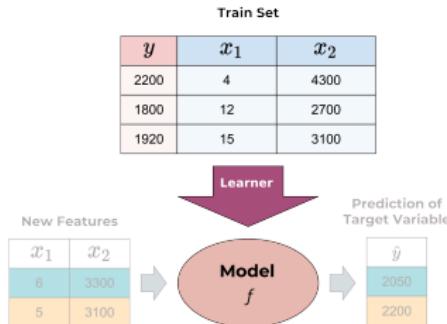
$$\begin{aligned} a_1 &= 0.6, a_2 = 0.2, a_3 = 0.2 \\ c_1 &= -3, c_2 = -2, c_3 = 1 \end{aligned}$$



$$\begin{aligned} a_1 &= 0.2, a_2 = 0.45, a_3 = 0.35 \\ c_1 &= (2, -2), c_2 = (0, 0), \\ c_3 &= (-3, 2) \end{aligned}$$

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

$n = 3$

$x_1^{(2)}$

$p = 2$

$x_2^{(1)}$

$y^{(3)}$

The diagram illustrates a supervised learning dataset with three data points. The first column is labeled 'People in Office (Feature 1) x_1 ' and the second column is labeled 'Salary (Feature 2) x_2 '. The third column is labeled 'Worked Minutes Week (Target Variable)'. A brace on the left indicates there are 3 data points, labeled $n = 3$. Brackets below the first and second columns indicate there are 2 features, labeled $p = 2$. Arrows point from labels $x_1^{(2)}$, $x_2^{(1)}$, and $y^{(3)}$ to the corresponding columns and row 3.

SUPERVISED LEARNING EXAMPLE

How could we construct a model from these data?

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

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

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

IDEA OF SUPERVISED LEARNING

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

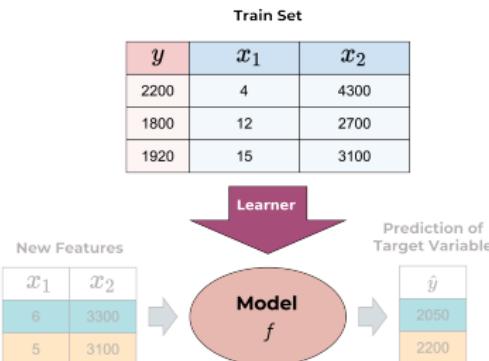
- **Supervised** learning means we make use of *labeled* data for which we observed the outcome.
- We use the labeled data to learn a model f .
- Ultimately, we use our model to compute predictions for **new** data whose target values are unknown.



LEARNER DEFINITION

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

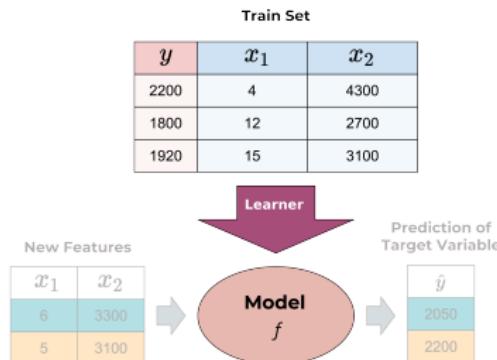
$$\mathcal{I} : \mathbb{D} \times \Lambda \rightarrow \mathcal{H}$$



LEARNER DEFINITION

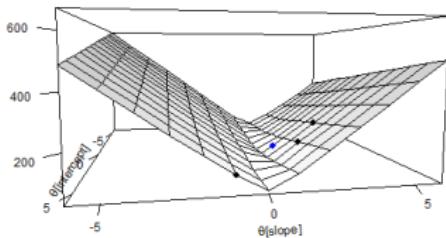
As pseudo-code template it would work like this:

- Learner has a defined model space of parametrized functions \mathcal{H} .
- User passes data set $\mathcal{D}_{\text{train}}$ and control settings λ .
- 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 training dataset. It consists of three tables arranged horizontally, separated by vertical lines. A question mark symbol (\approx) is positioned between the middle table and the rightmost table, indicating a relationship between the target values and the predicted values. Below the first two tables, a brace indicates they are part of the training set, labeled $\mathcal{D}_{\text{train}}$.

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

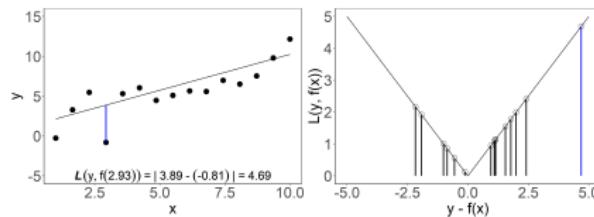
- To make this precise, we need to define now how we measure the difference between a prediction and a ground truth label pointwise.

LOSS

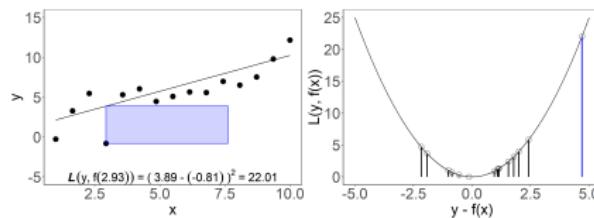
The **loss function** $L(y, f(\mathbf{x}))$ quantifies the "quality" of the prediction $f(\mathbf{x})$ of a single observation \mathbf{x} :

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

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} .

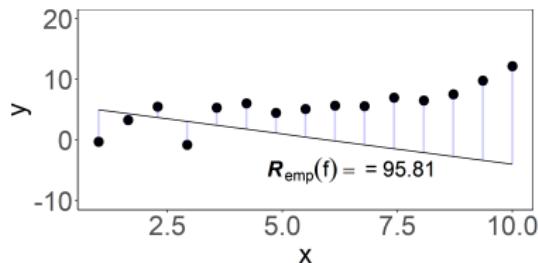
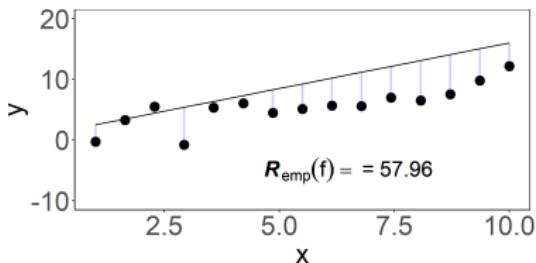
EMPIRICAL RISK

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

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

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

- Since f is usually defined by **parameters** θ , this becomes:

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

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

EMPIRICAL RISK MINIMIZATION

The best model is the model with the smallest risk.

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

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

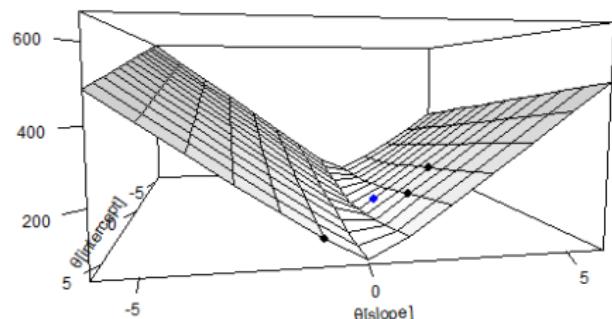
EMPIRICAL RISK MINIMIZATION

But usually \mathcal{H} is infinitely large.

Instead we can consider the risk surface w.r.t. the parameters θ .
(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	$\theta_{\text{intercept}}$	θ_{slope}	$\mathcal{R}_{\text{emp}}(\theta)$
f_1	2	3	194.62
f_2	3	2	127.12
f_3	6	-1	95.81
f_4	1	1.5	57.96



EMPIRICAL RISK MINIMIZATION

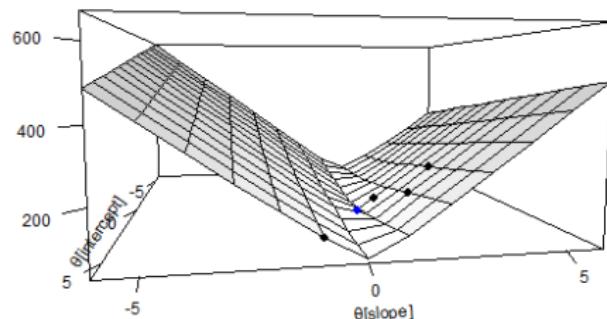
Minimizing this surface is called **empirical risk minimization** (ERM).

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

Usually we do this by numerical optimization.

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

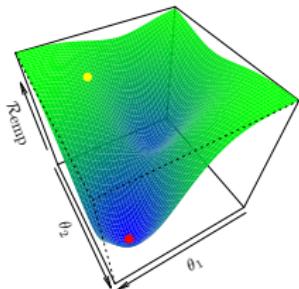
Model	$\theta_{\text{intercept}}$	θ_{slope}	$\mathcal{R}_{\text{emp}}(\theta)$
f_1	2	3	194.62
f_2	3	2	127.12
f_3	6	-1	95.81
f_4	1	1.5	57.96
f_5	1.25	0.90	23.40



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

Introduction to Machine Learning

ML-Basics: Optimization

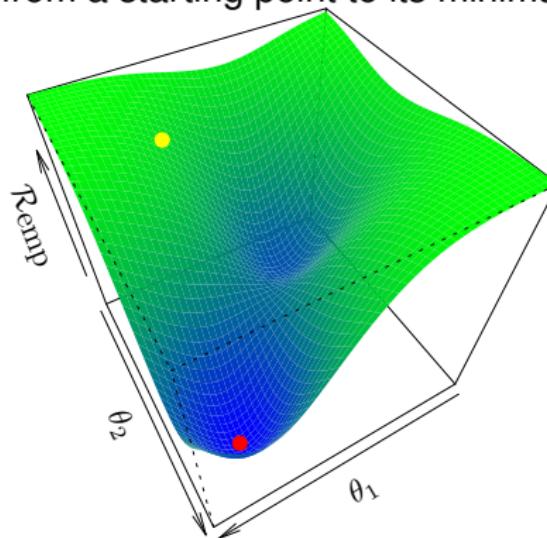


Learning goals

- 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.

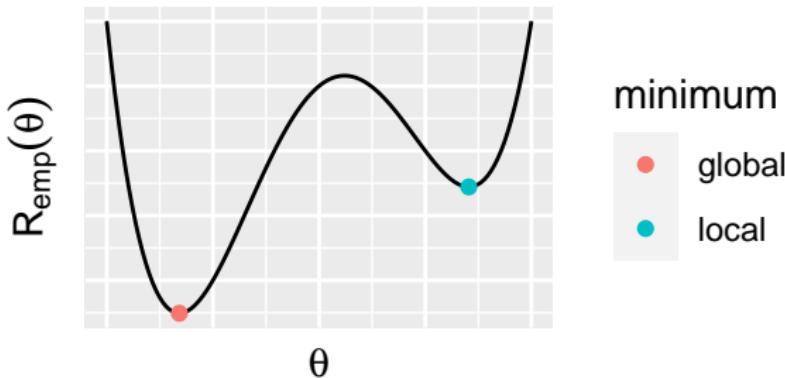
Which kind of numerical technique is reasonable for this problem strongly depends on model and parameter structure (continuous params? uni-modal $\mathcal{R}_{\text{emp}}(\theta)$?). Here, we will only discuss very simple scenarios.

LOCAL MINIMA

If \mathcal{R}_{emp} is continuous in θ 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.

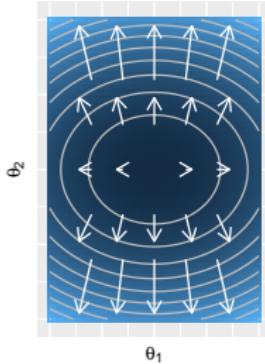


LOCAL MINIMA AND STATIONARY POINTS

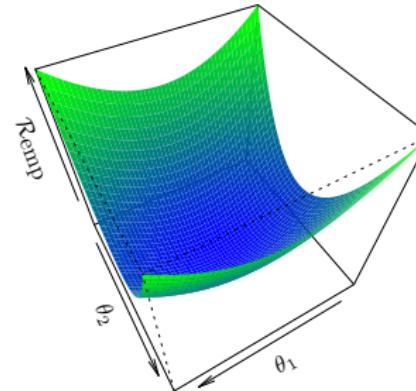
If \mathcal{R}_{emp} is continuously differentiable in θ then a **sufficient condition** for a local minimum is that $\hat{\theta}$ is **stationary** with 0 gradient, so no local improvement is possible:

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

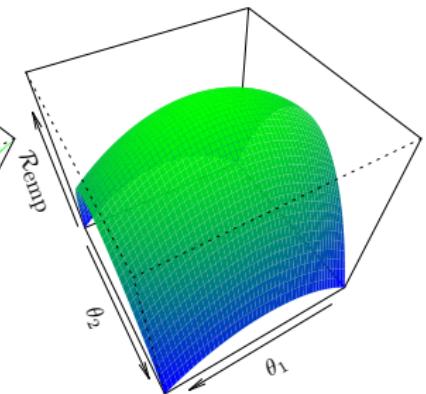
and the Hessian $\frac{\partial^2}{\partial \theta^2} \mathcal{R}_{\text{emp}}(\hat{\theta})$ is positive definite. While the neg. gradient points into the direction of fastest local decrease, the Hessian measures local curvature of \mathcal{R}_{emp} .



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



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**

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

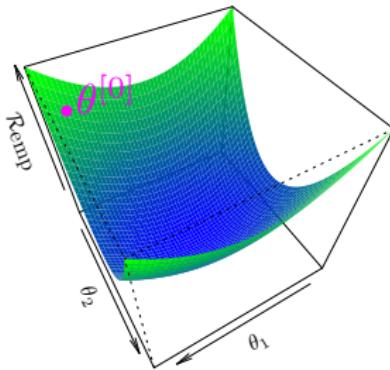
is a local minimum of \mathcal{R}_{emp} . If \mathbf{X} is full-rank, \mathcal{R}_{emp} is strictly convex and there is only one local minimum - which is also global.

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

GRADIENT DESCENT

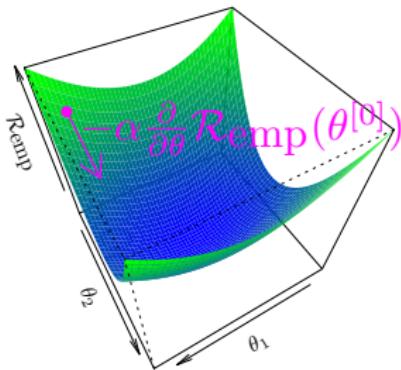
The simple idea of GD is to iteratively go from the current candidate $\theta^{[t]}$ in the direction of the negative gradient, i.e., the direction of the steepest descent, with learning rate α to the next $\theta^{[t+1]}$:

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

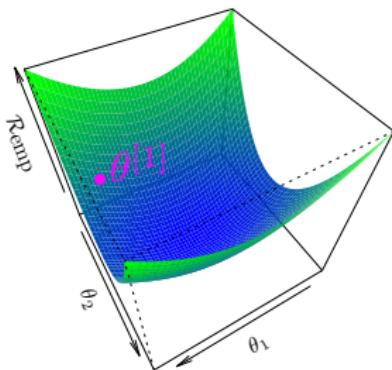


We choose a random start $\theta^{[0]}$ with risk $\mathcal{R}_{\text{emp}}(\theta^{[0]}) = 76.25$.

GRADIENT DESCENT - EXAMPLE

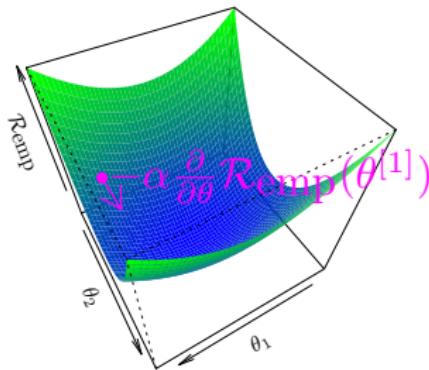


Now we follow in the direction of the negative gradient at $\theta^{[0]}$.

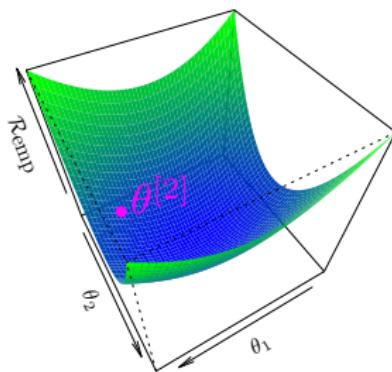


We arrive at $\theta^{[1]}$ with risk
 $\mathcal{R}_{\text{emp}}(\theta^{[1]}) \approx 42.73$.
We improved:
 $\mathcal{R}_{\text{emp}}(\theta^{[1]}) < \mathcal{R}_{\text{emp}}(\theta^{[0]}).$

GRADIENT DESCENT - EXAMPLE

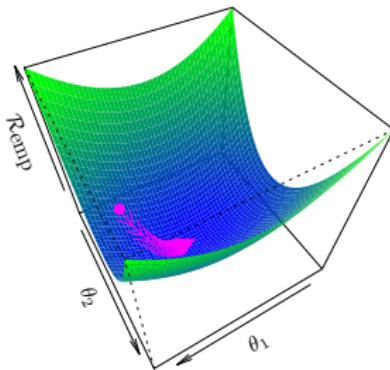


Again we follow in the direction of the negative gradient, but now at $\theta^{[1]}$.

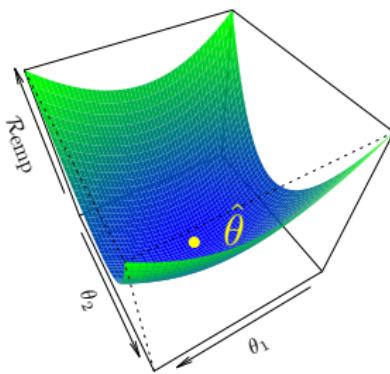


Now $\theta^{[2]}$ has risk $\mathcal{R}_{\text{emp}}(\theta^{[2]}) \approx 25.08$.

GRADIENT DESCENT - EXAMPLE



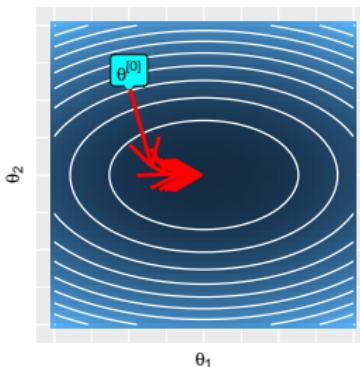
We iterate this until some form of convergence or termination.



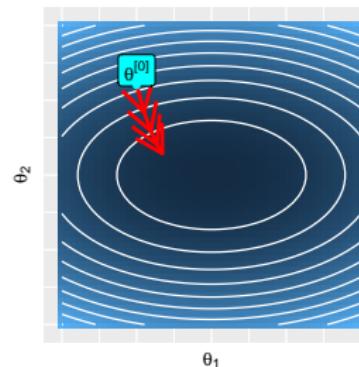
We arrive close to a stationary $\hat{\theta}$ which is hopefully at least a local minimum.

GRADIENT DESCENT - LEARNING RATE

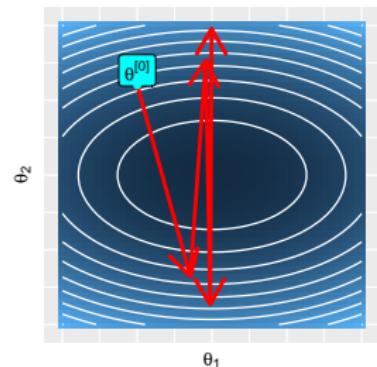
- The negative gradient is a direction that looks locally promising to reduce \mathcal{R}_{emp} .
- Hence it weights components higher in which \mathcal{R}_{emp} decreases more.
- However, the length of $-\frac{\partial}{\partial \theta} \mathcal{R}_{\text{emp}}$ measures only the local decrease rate,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



good convergence for α_1



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



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

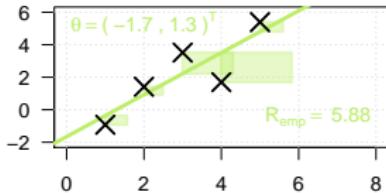
FURTHER TOPICS

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

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

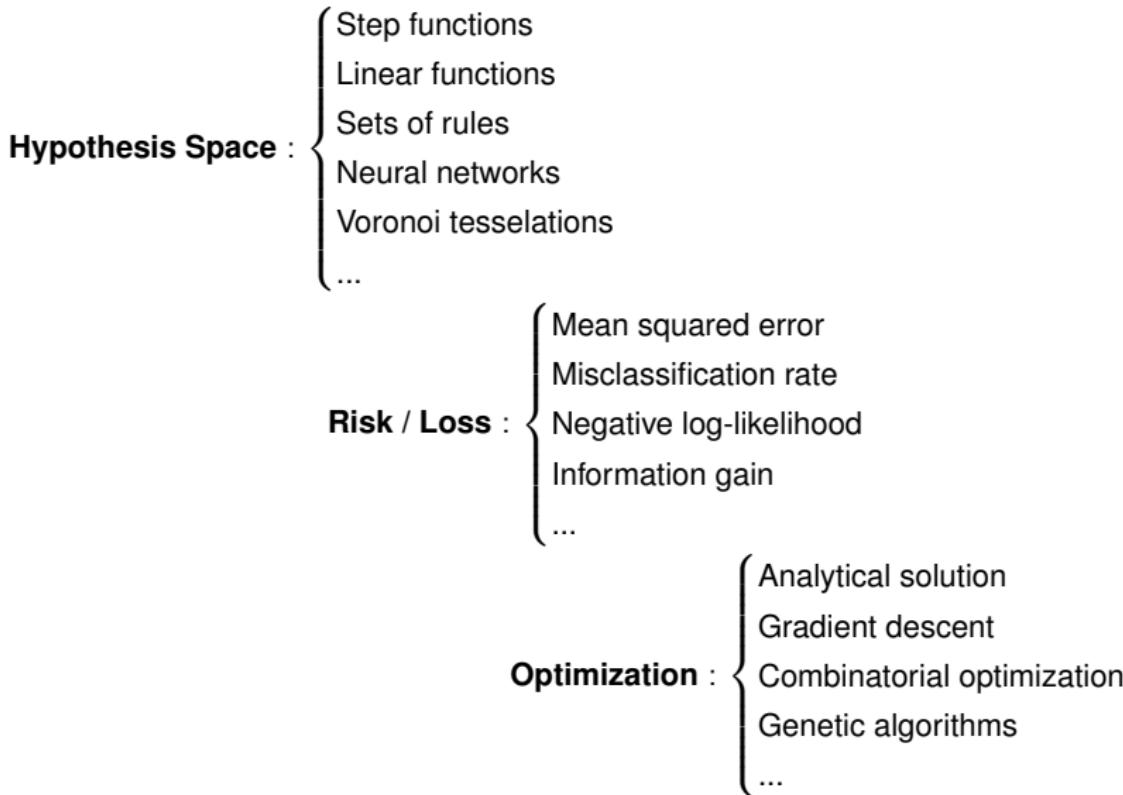
This concept can be extended by the concept of **regularization**, where the model complexity is accounted for in the risk:

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

- 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** $\mathcal{D} \in \mathbb{D}$, 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 θ 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 $\lambda \in \Lambda$) is:

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

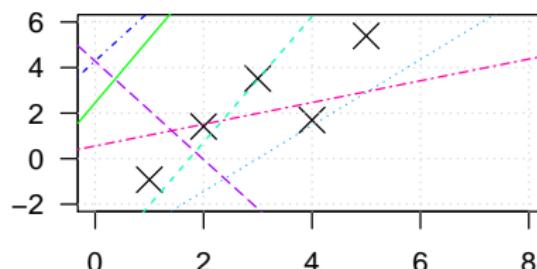
We can also adapt this concept to finding $\hat{\theta}$ for parametric models:

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

EXAMPLE: LINEAR REGRESSION ON 1D

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

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

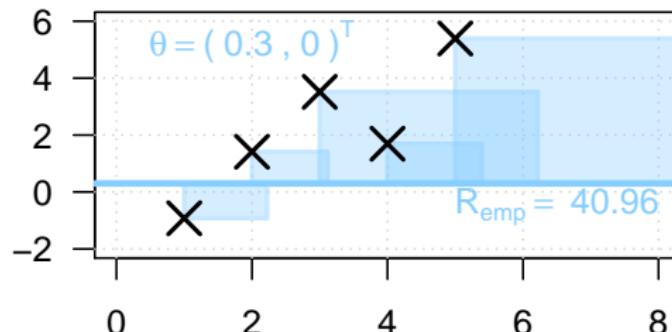


Design choice: We could add more flexibility by allowing polynomial effects or by using a spline basis.

EXAMPLE: LINEAR REGRESSION ON 1D

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

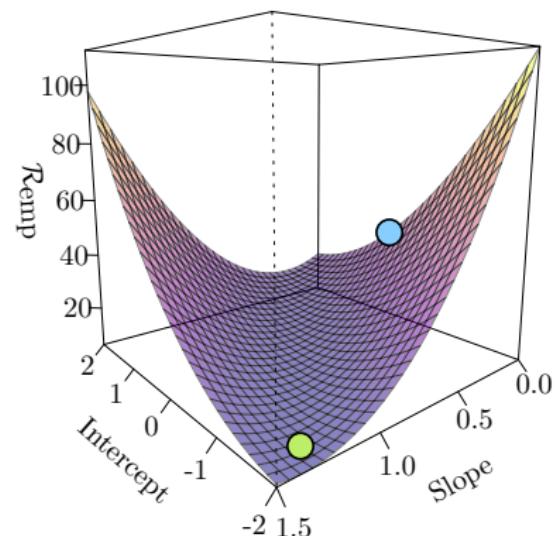
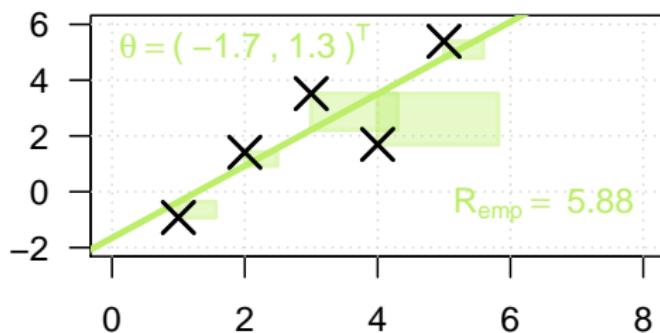
$$\mathcal{R}_{\text{emp}}(\theta) = \sum_{i=1}^n (y^{(i)} - \theta_0 - \theta_1 \mathbf{x}^{(i)})^2$$



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

EXAMPLE: LINEAR REGRESSION ON 1D

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



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

SUMMARY

By decomposing learners into these building blocks:

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

Getting this right takes a considerable amount of experience.

INTRODUCTION TO MACHINE LEARNING

ML Basics

Supervised Regression

Supervised Classification

Performance Evaluation

k-NN

Classification and Regression Trees (CART)

Random Forests

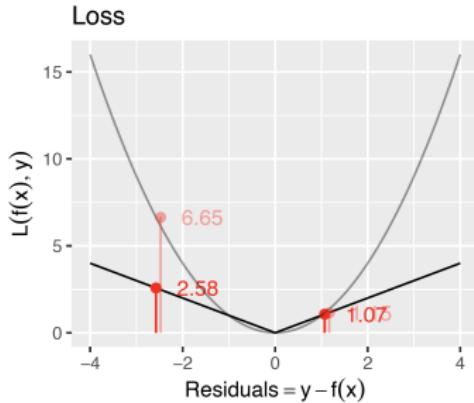
Tuning

Nested Resampling

mlr3

Introduction to Machine Learning

Loss Functions for Regression



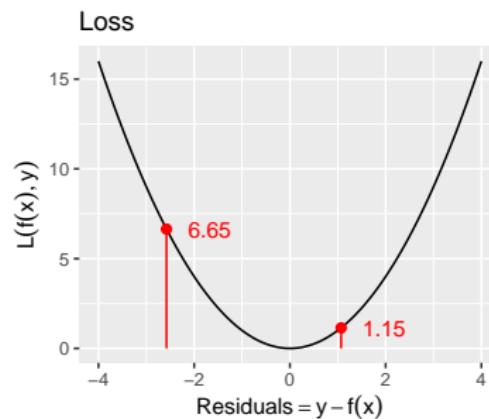
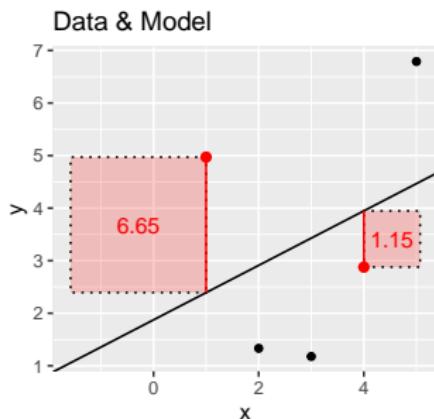
Learning goals

- 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 later: $\frac{\partial 0.5(y-f(\mathbf{x}))^2}{\partial f(\mathbf{x})} = f(\mathbf{x}) - y = -\epsilon$, derivative is negative 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.

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

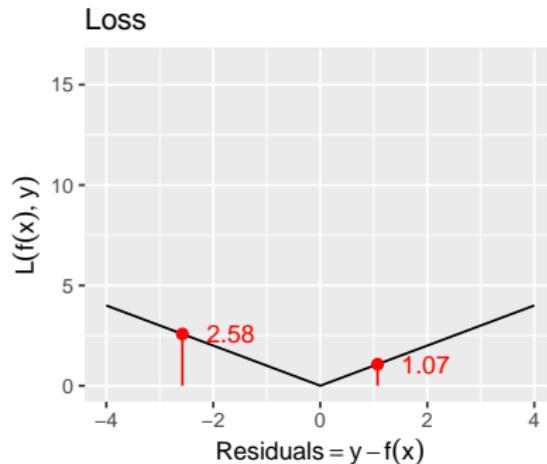
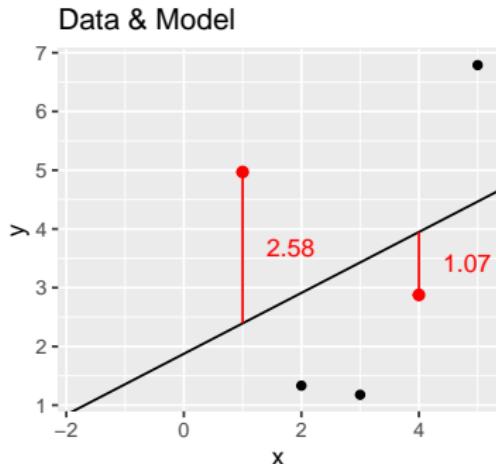
We set the derivative of the empirical risk to zero and solve for c :

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

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

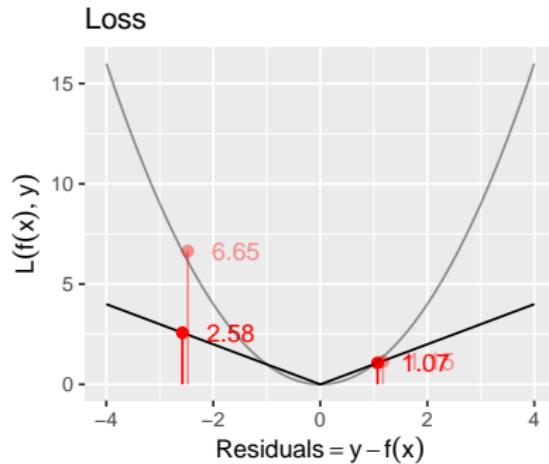
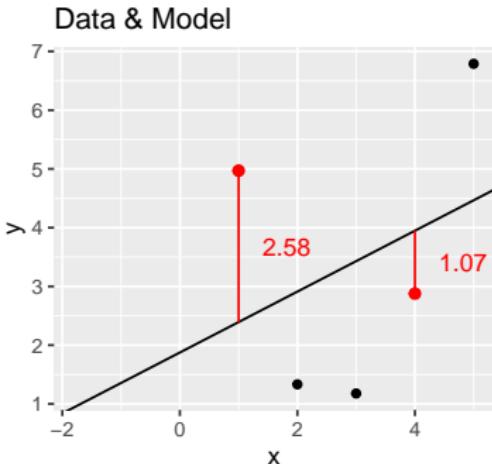
REGRESSION LOSSES - L1 / ABSOLUTE ERROR

- $L(y, f(\mathbf{x})) = |y - f(\mathbf{x})|$
- 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

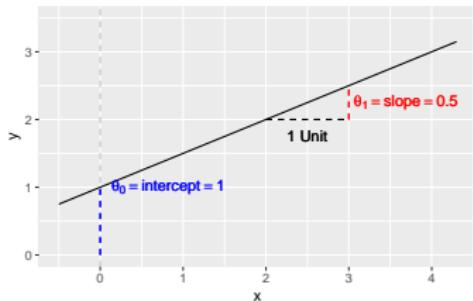
- $L(y, f(\mathbf{x})) = |y - f(\mathbf{x})|$
- Convex
- No derivatives for $\epsilon = 0$, $y = f(\mathbf{x})$, optimization becomes harder
- $\hat{f}(\mathbf{x}) = \text{median of } y|\mathbf{x}$
- More robust, outliers in y are less influential than for L2



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 $\mathbf{x} \in \mathbb{R}^p$.

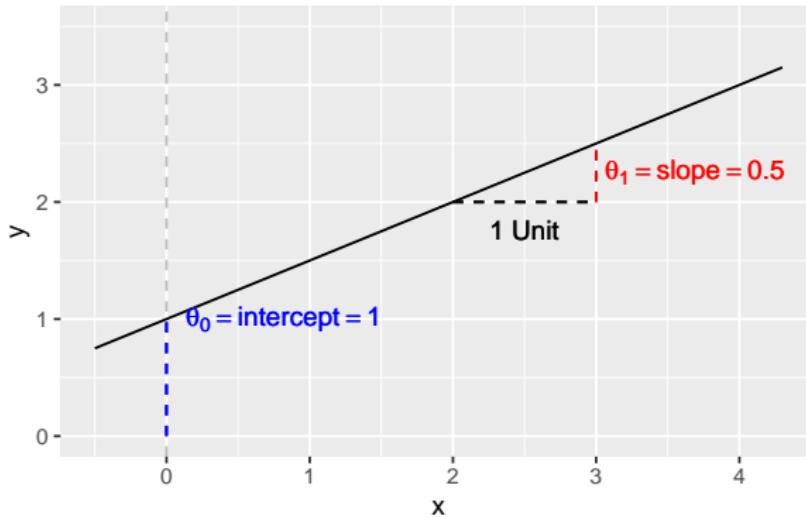
So with $\boldsymbol{\theta} \in \mathbb{R}^p$ this mapping can be written as:

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

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 (θ_0, θ) ?

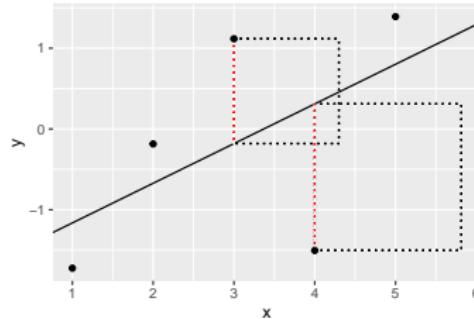
This is **learning** or parameter estimation, the learner does exactly this by **empirical risk minimization**.

NB: We assume from now on that θ_0 is included in θ .

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**:

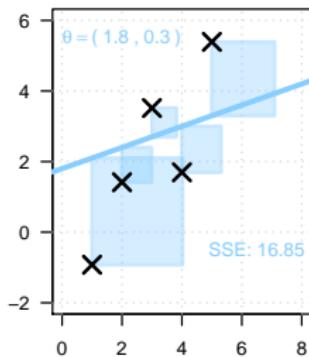
$$\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) = \text{SSE}(\boldsymbol{\theta}) = \sum_{i=1}^n L\left(y^{(i)}, f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)\right) = \sum_{i=1}^n \left(y^{(i)} - \boldsymbol{\theta}^T \mathbf{x}^{(i)}\right)^2$$



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

LINEAR MODEL: OPTIMIZATION

We want to find the parameters θ 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 θ .
A first (random) try yields a rather large SSE: (**Evaluation**).

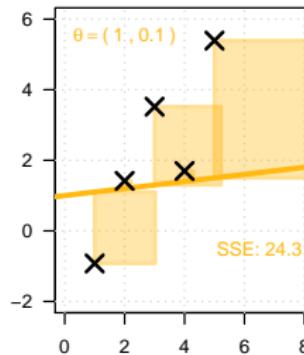
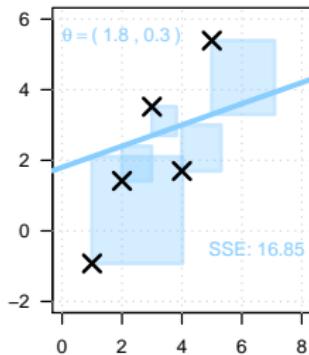


LINEAR MODEL: OPTIMIZATION

We want to find the parameters θ 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 θ .

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

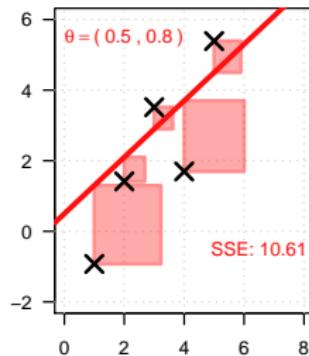
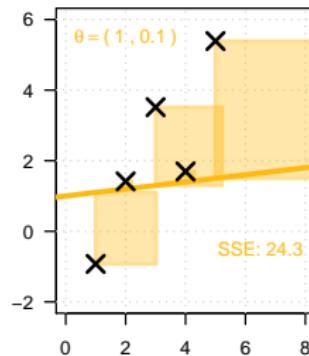
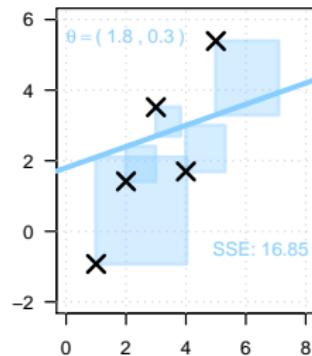


LINEAR MODEL: OPTIMIZATION

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

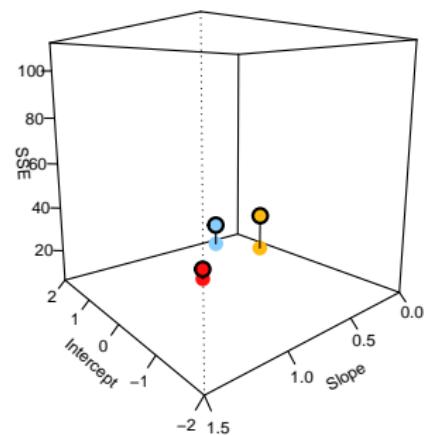
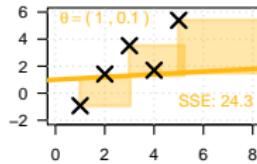
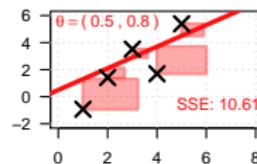
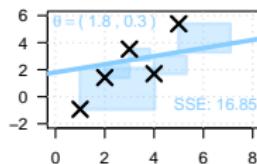
So we evaluate different candidates for θ .

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



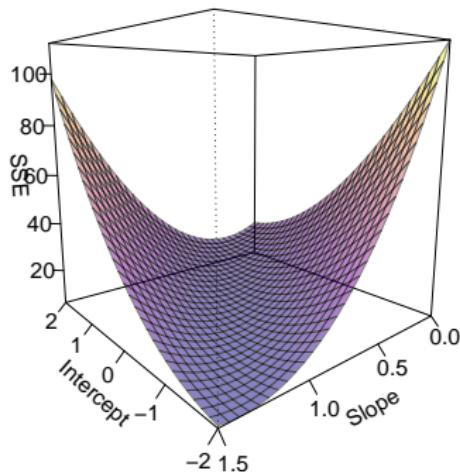
LINEAR MODEL: OPTIMIZATION

Since every θ 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:



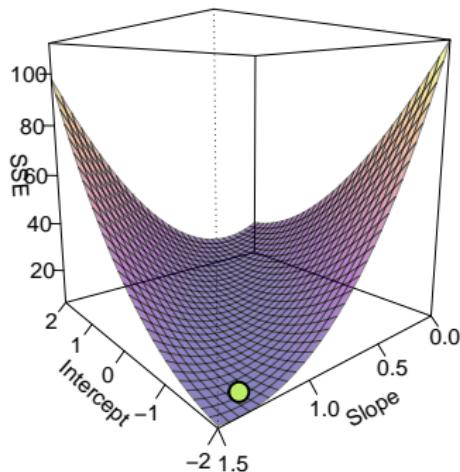
LINEAR MODEL: OPTIMIZATION

Instead of guessing, we use **optimization** to find the best θ :



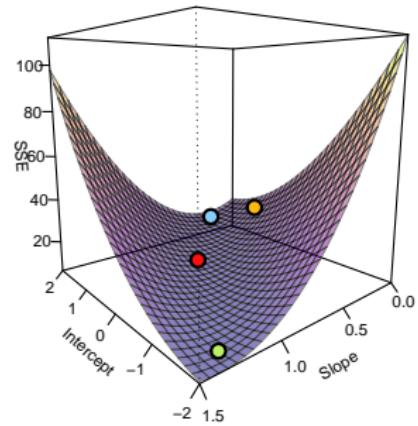
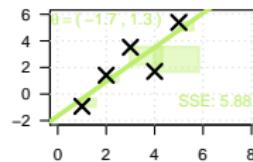
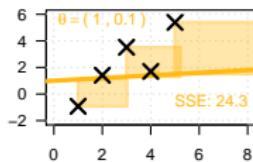
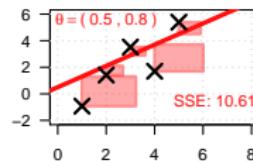
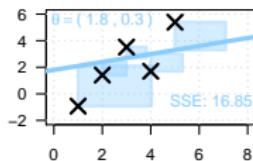
LINEAR MODEL: OPTIMIZATION

Instead of guessing, we use **optimization** to find the best θ :



LINEAR MODEL: OPTIMIZATION

Instead of guessing, we use **optimization** to find the best θ :



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**.

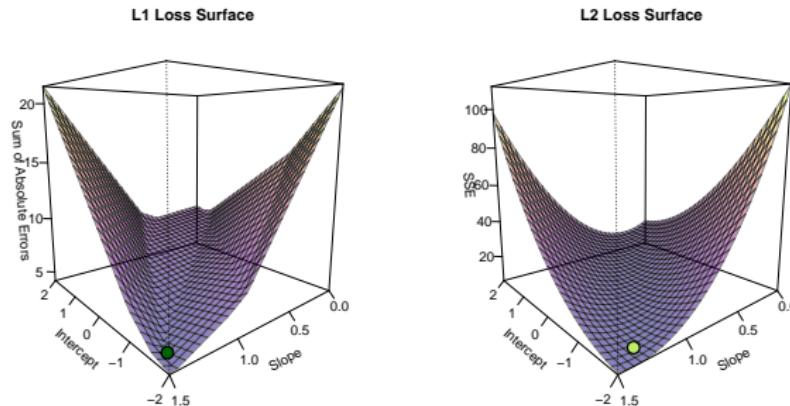
This yields the so-called normal equations for the LM:

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

EXAMPLE: REGRESSION WITH L1 VS L2 LOSS

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

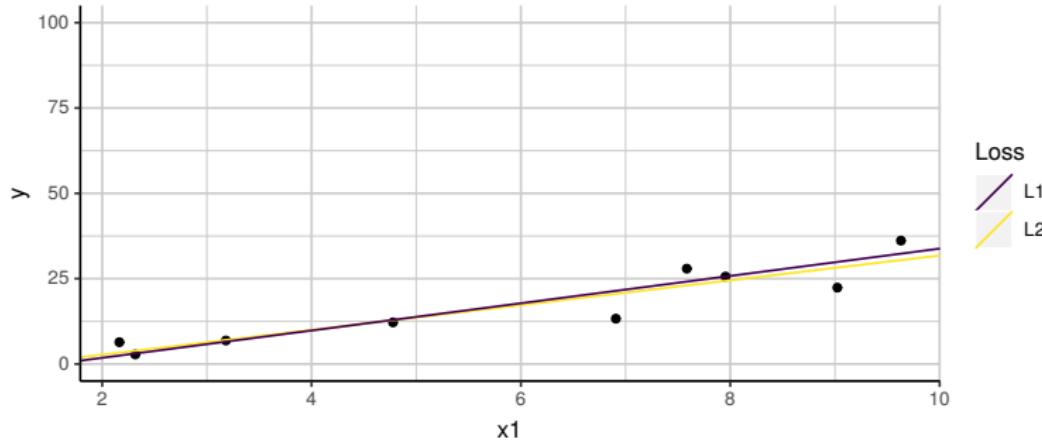
$$\mathcal{R}_{\text{emp}}(\theta) = \sum_{i=1}^n L\left(y^{(i)}, f\left(\mathbf{x}^{(i)} | \theta\right)\right) = \sum_{i=1}^n \left|y^{(i)} - \theta^T \mathbf{x}^{(i)}\right| \quad (\text{Risk})$$



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

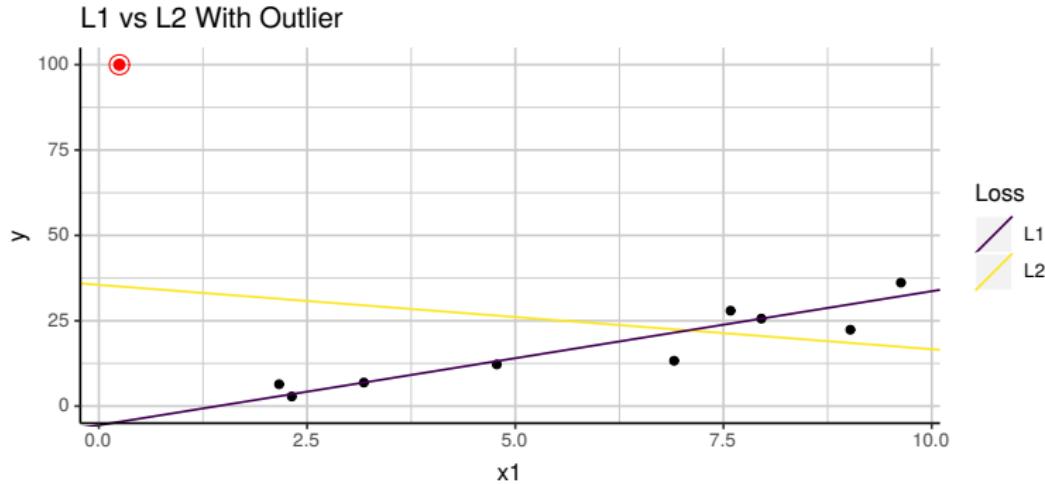
EXAMPLE: REGRESSION WITH L1 VS L2 LOSS

L1 vs L2 Without Outlier



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.

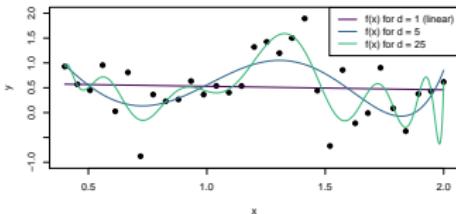
Optimization: Direct analytical solution for L2 loss, numerical optimization for L1 and others.

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

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

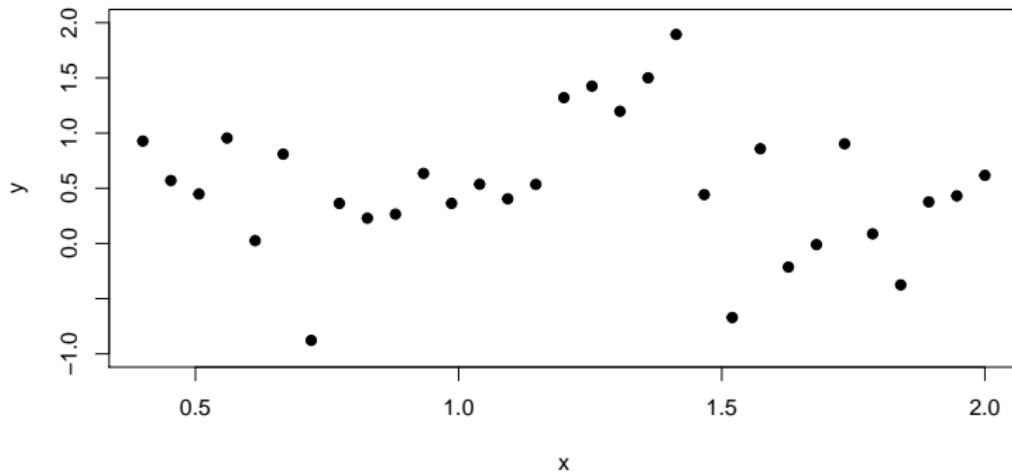
of only the original features,

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

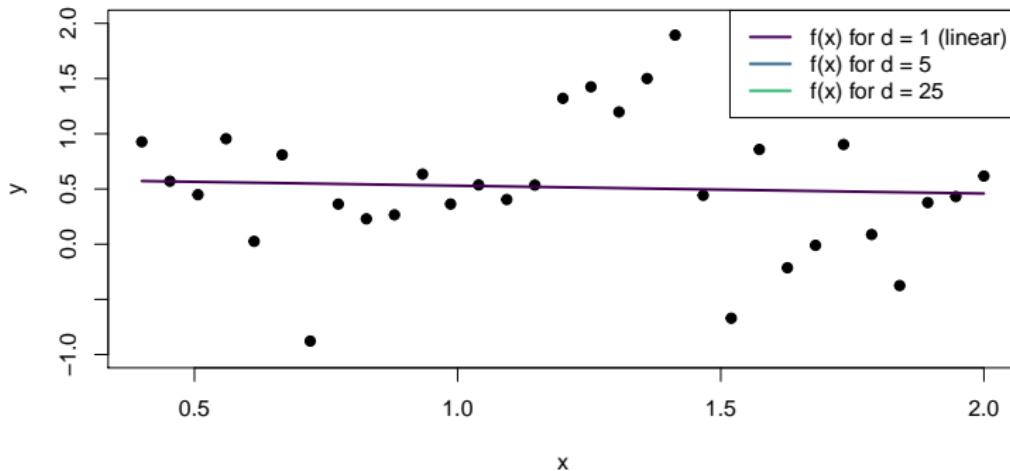
Polynomial regression example



REGRESSION: POLYNOMIALS

Polynomial regression example

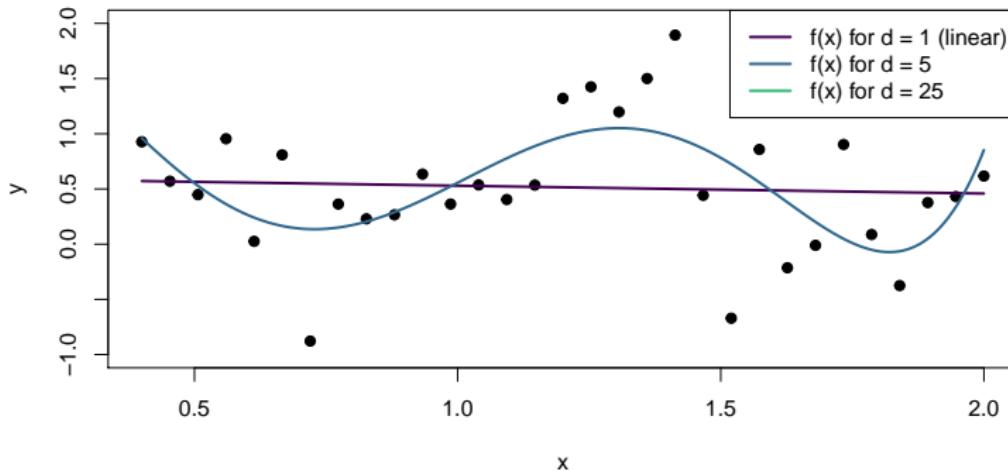
Models of different *complexity*, i.e., of different polynomial order d , are fitted to the data:



REGRESSION: POLYNOMIALS

Polynomial regression example

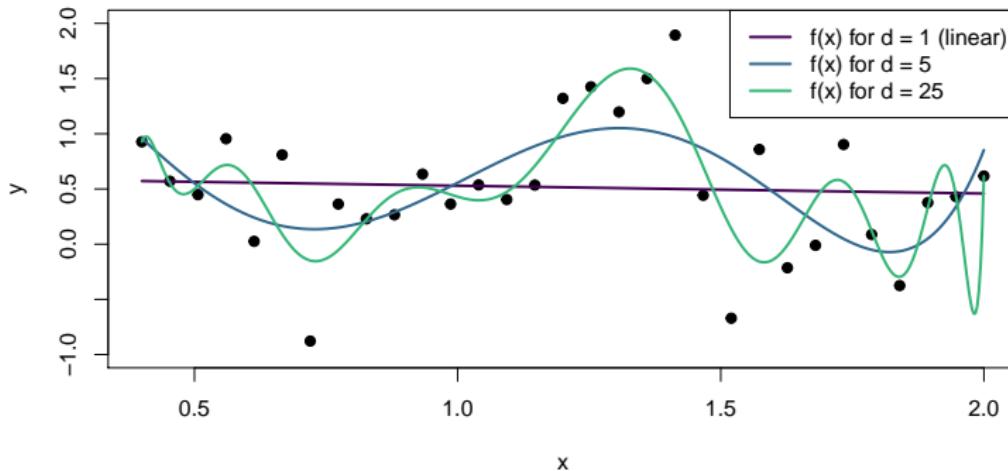
Models of different *complexity*, i.e., of different polynomial order d , are fitted to the data:



REGRESSION: POLYNOMIALS

Polynomial regression example

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



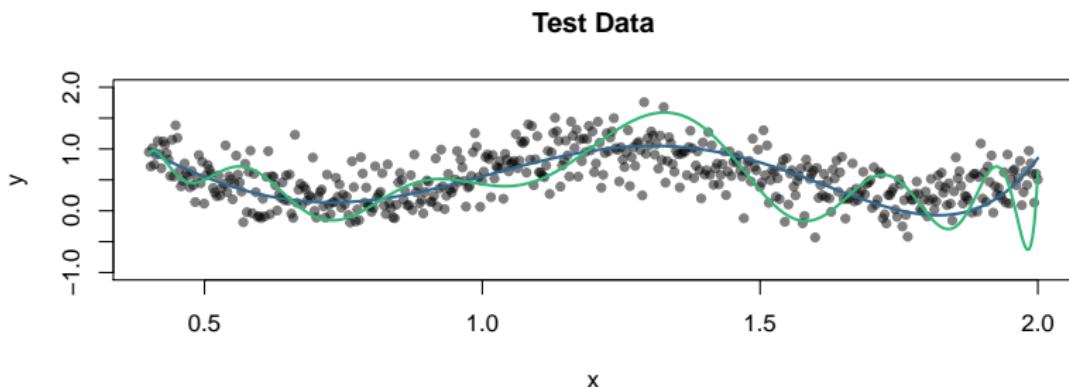
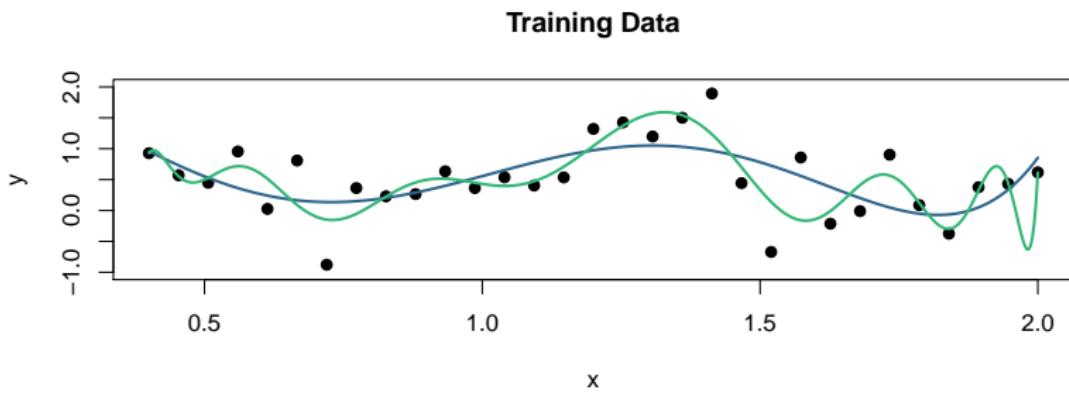
REGRESSION: POLYNOMIALS

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

Performance Evaluation

k-NN

Classification and Regression Trees (CART)

Random Forests

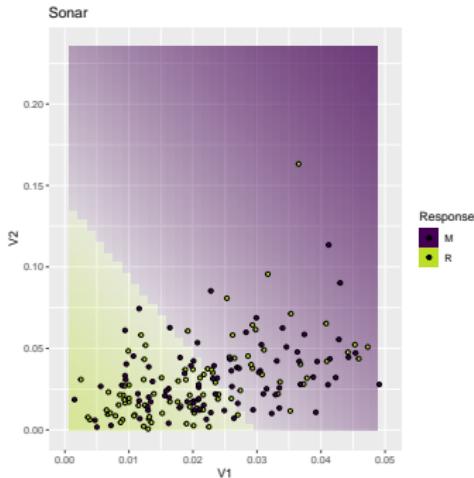
Tuning

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

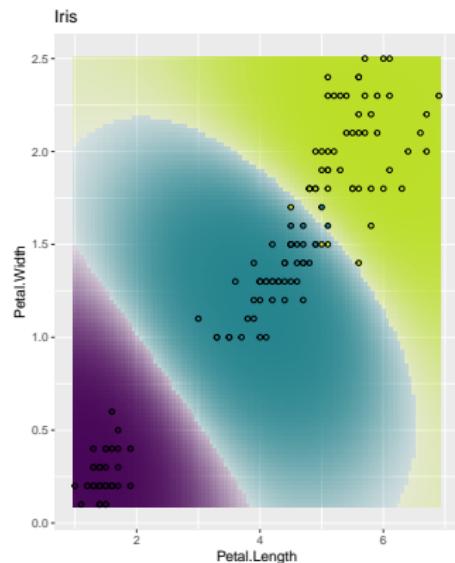
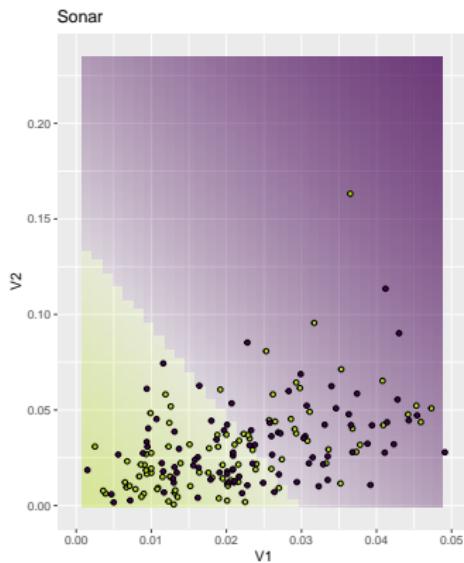
CLASSIFICATION

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).

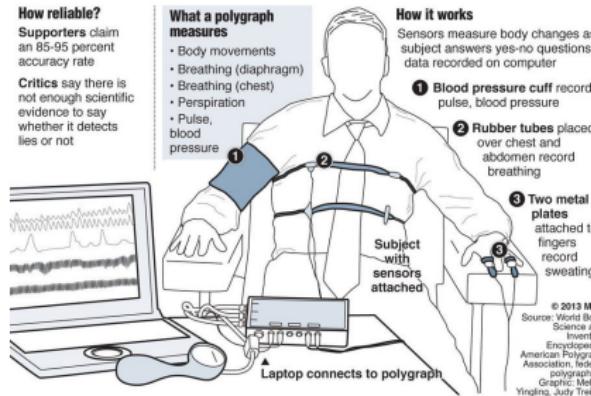


BINARY CLASSIFICATION TASK - EXAMPLES

- 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.



© 2013 MET
Source: World Book
Science and
Invention
Encyclopedia,
American Polygraph
Association, Federal
Bureau of Investigation.
Graphic: Melina
Yingling, Judy Treble

<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



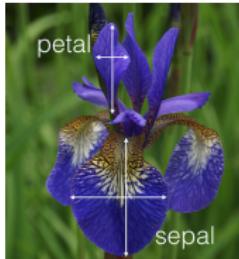
Virginica

Source:

https://en.wikipedia.org/wiki/Iris_flower_data_set

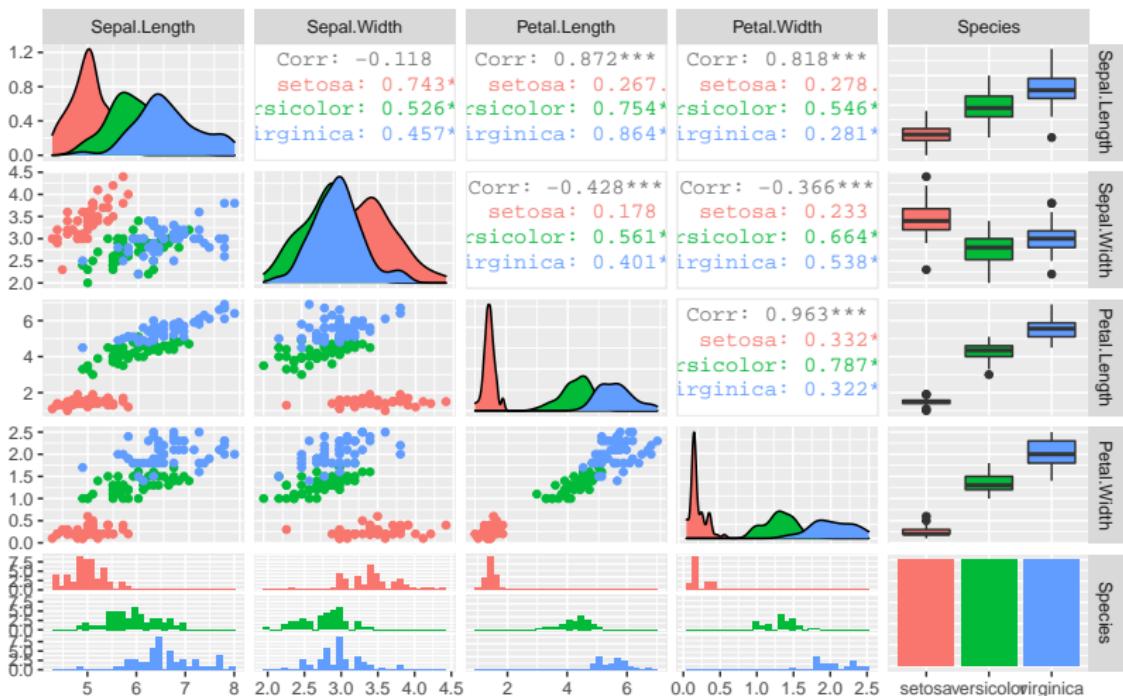
MULTICLASS TASK - IRIS

- 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

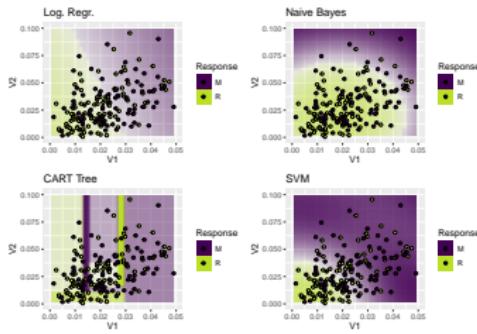
MULTICLASS TASK - IRIS



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

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

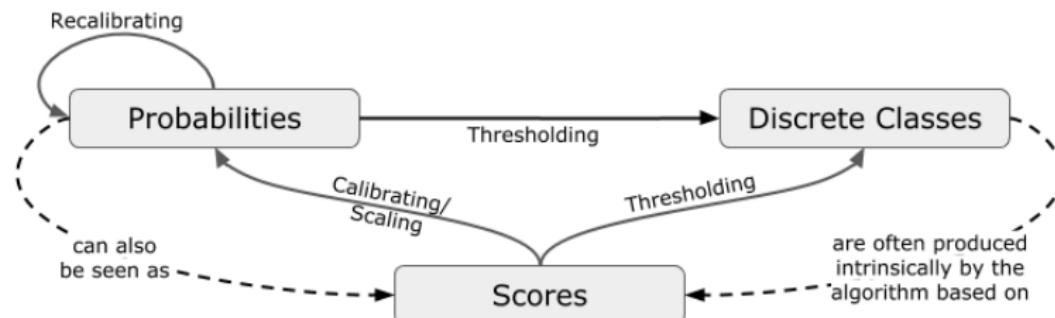
- Probabilities $\pi_1(\mathbf{x}), \dots, \pi_g(\mathbf{x})$ are transformed into labels by predicting the class with the maximum probability

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

- 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**

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

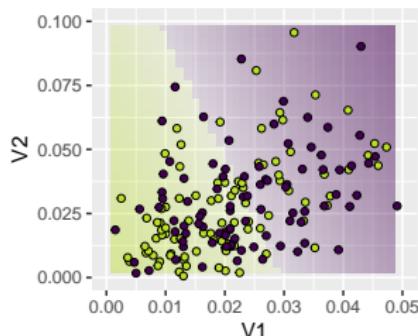
In the binary case we can simplify and generalize to the decision boundary for general threshold c :

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

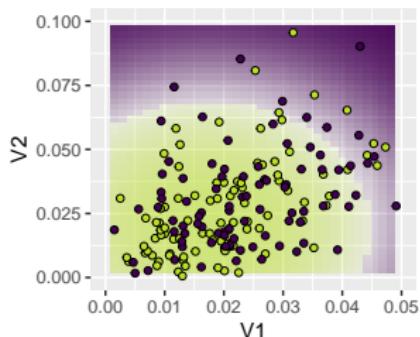
If we set $c = 0$ for scores and $c = 0.5$ for probabilities, this is consistent with the definition above.

DECISION BOUNDARY EXAMPLES

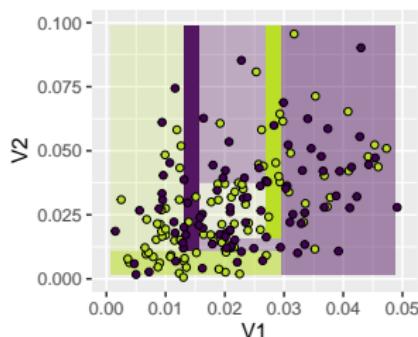
Log. Regr.



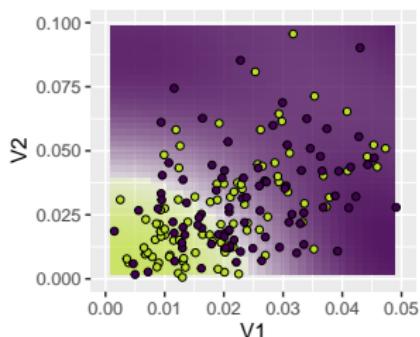
Naive Bayes



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 π_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.

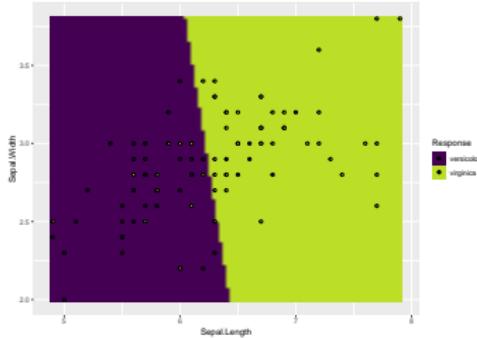
$$\hat{f} = \arg \min_{f \in \mathcal{H}} \mathcal{R}_{\text{emp}}(f) = \arg \min_{f \in \mathcal{H}} \sum_{i=1}^n L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right).$$

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**.

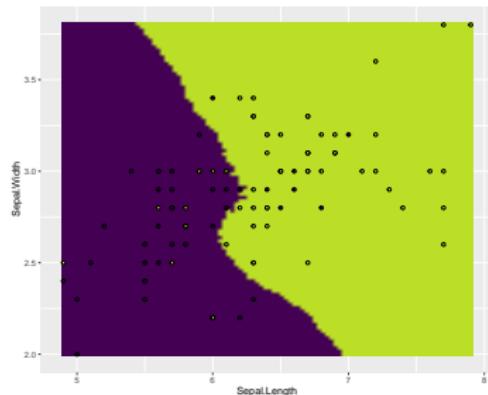
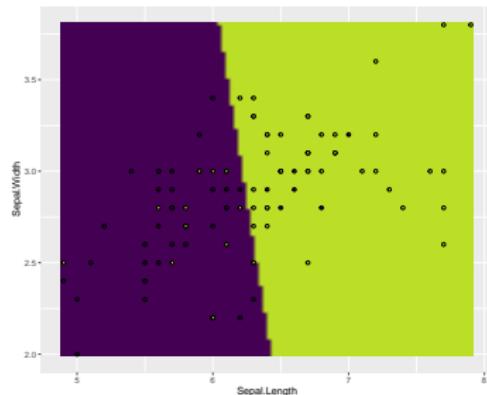
LINEAR CLASSIFIERS

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:

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

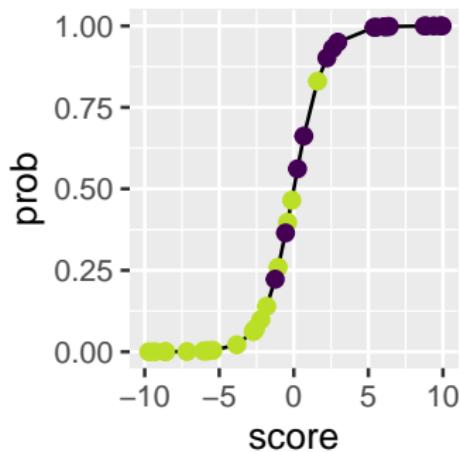
This is a **hyperplane** separating two classes.

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

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

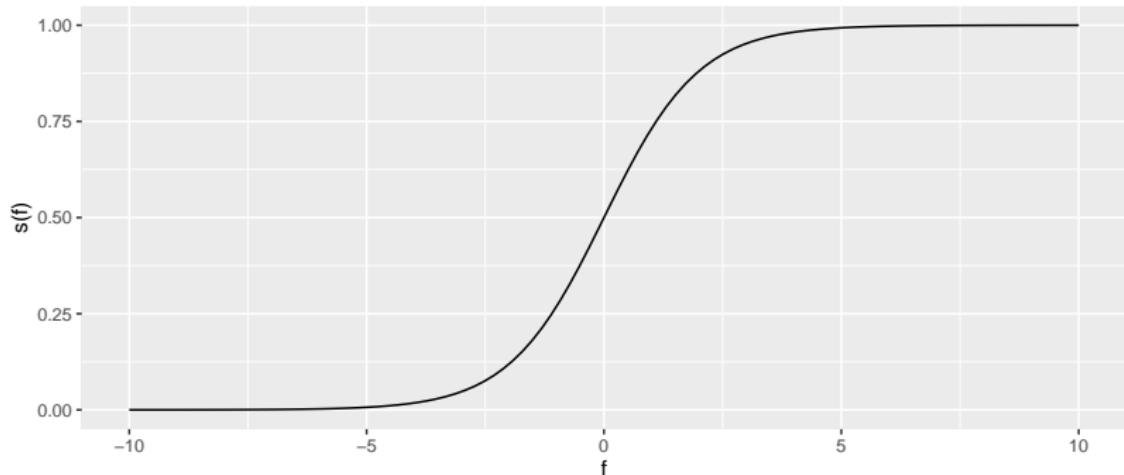
NB: We will often suppress the intercept in notation.

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

LOGISTIC FUNCTION

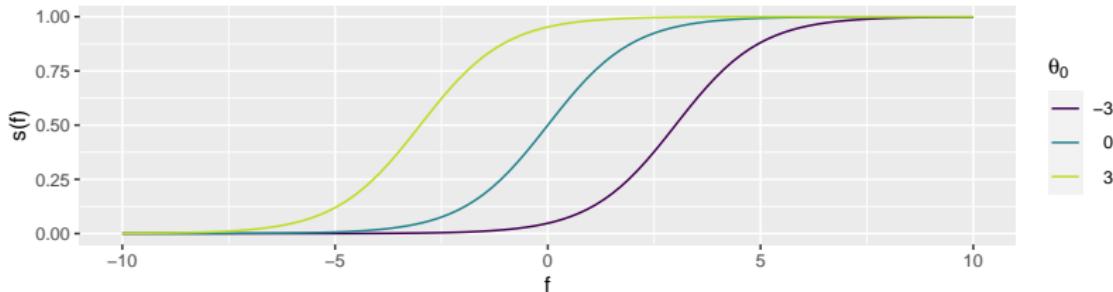
To avoid this, logistic regression “squashes” the estimated linear scores $\theta^T \mathbf{x}$ to $[0, 1]$ through the **logistic function** s :

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

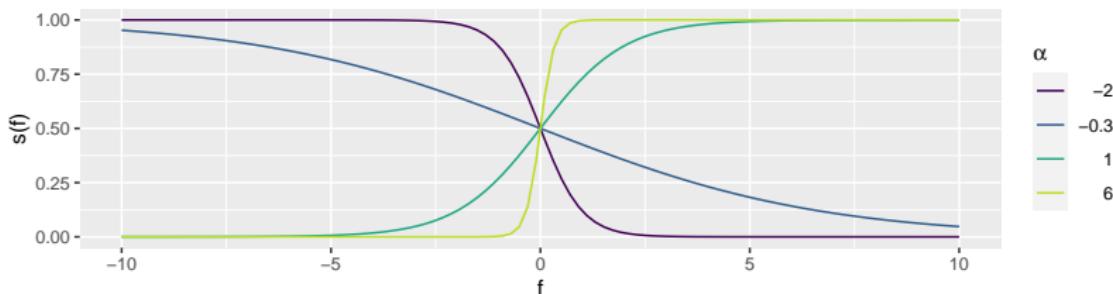


LOGISTIC FUNCTION

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



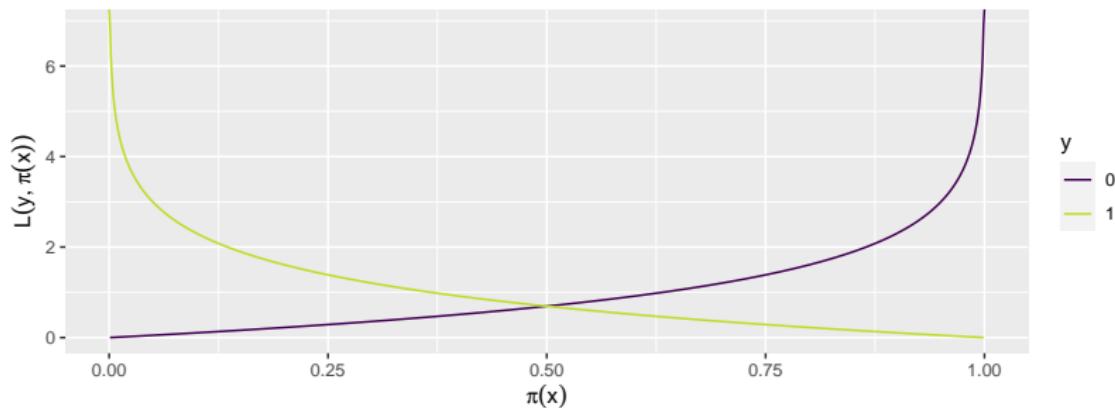
Scaling f like $s(\alpha f) = \frac{\exp(\alpha f)}{1 + \exp(\alpha f)}$ controls the slope and direction.



BERNOULLI / LOG LOSS

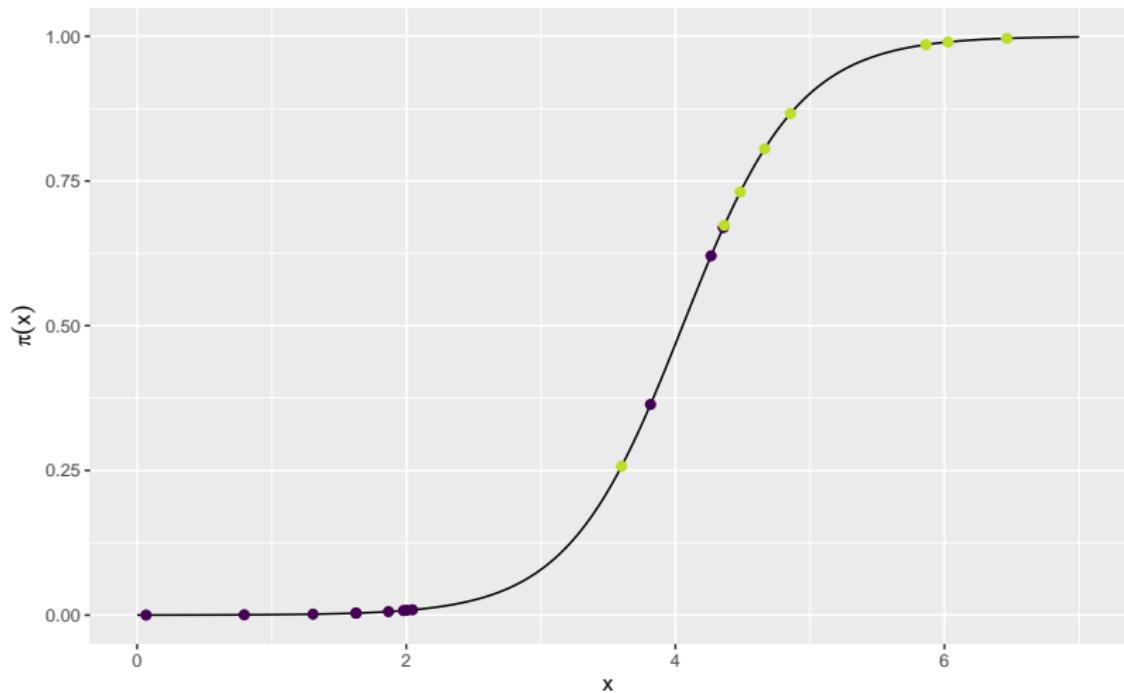
We need to define a loss function for the ERM approach:

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



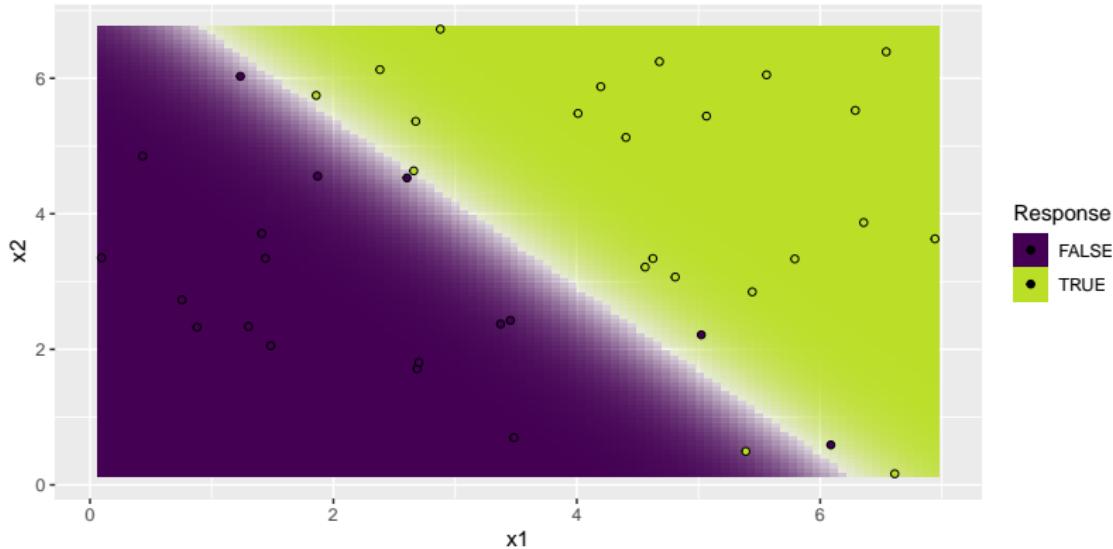
LOGISTIC REGRESSION IN 1D

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

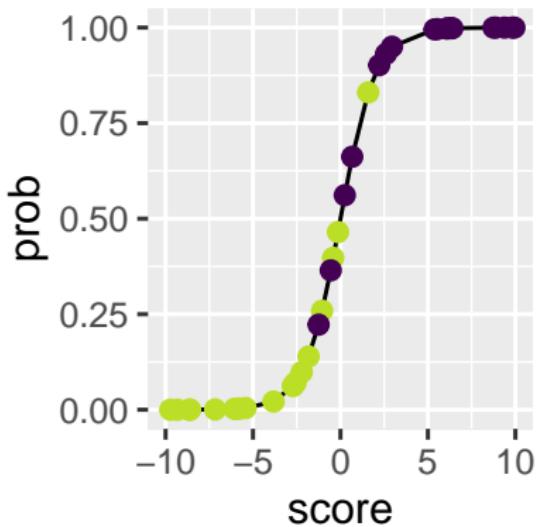
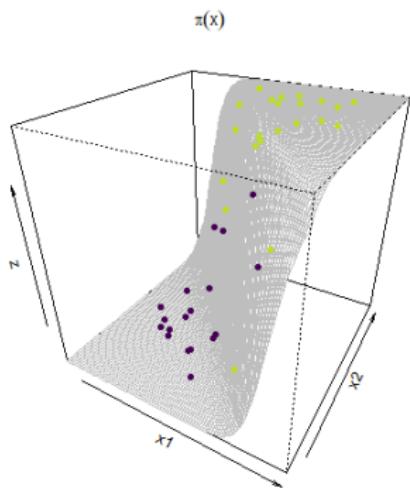


LOGISTIC REGRESSION IN 2D

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.

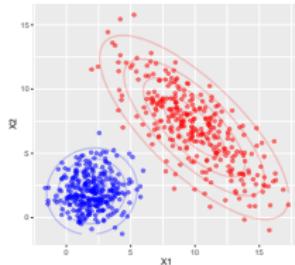
$$L(y, \pi(\mathbf{x})) = -y \ln(\pi(\mathbf{x})) - (1 - y) \ln(1 - \pi(\mathbf{x}))$$

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

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

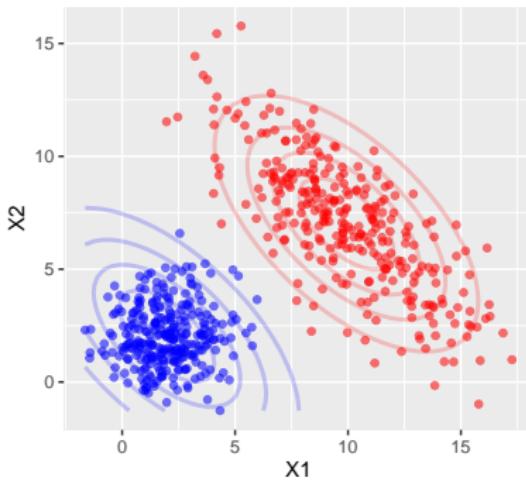
where we now have to pick a distributional form for $p(\mathbf{x}|y = k)$.

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. $\boldsymbol{\Sigma}_k = \boldsymbol{\Sigma} \quad \forall k$.



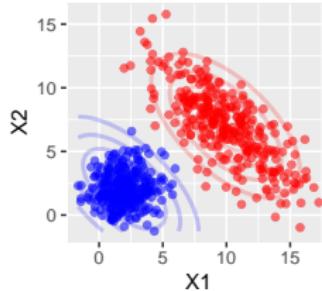
LINEAR DISCRIMINANT ANALYSIS (LDA)

Parameters θ are estimated in a straightforward manner by estimating

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

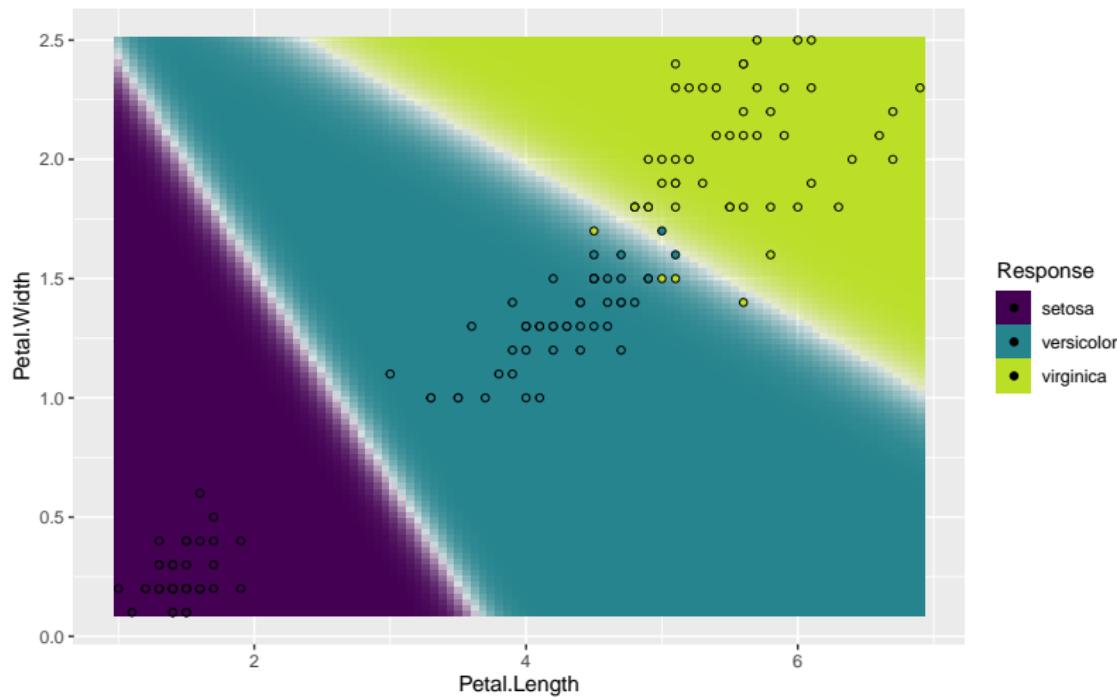
$$\hat{\mu}_k = \frac{1}{n_k} \sum_{i:y^{(i)}=k} \mathbf{x}^{(i)}$$

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



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 Σ_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:

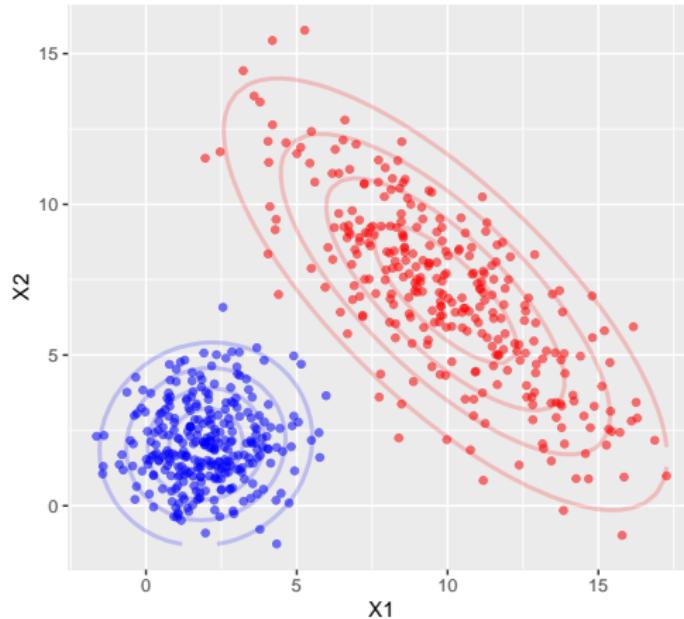
$$\hat{\pi}_k = \frac{n_k}{n}, \text{ where } n_k \text{ is the number of class-}k \text{ observations}$$

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

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

QUADRATIC DISCRIMINANT ANALYSIS (QDA)

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



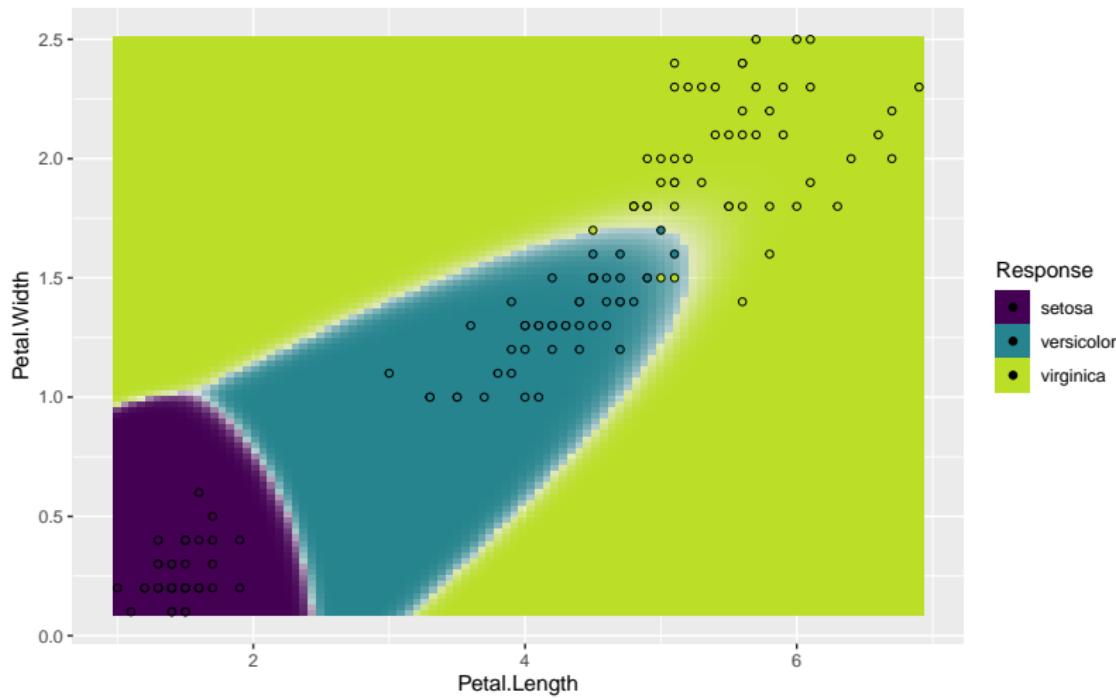
QUADRATIC DISCRIMINANT ANALYSIS (QDA)

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

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

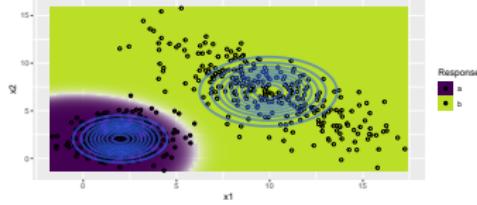
$$\log \pi_k - \frac{1}{2} \log |\Sigma_k| - \frac{1}{2} \boldsymbol{\mu}_k^T \Sigma_k^{-1} \boldsymbol{\mu}_k + \mathbf{x}^T \Sigma_k^{-1} \boldsymbol{\mu}_k - \frac{1}{2} \mathbf{x}^T \Sigma_k^{-1} \mathbf{x}$$

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:

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

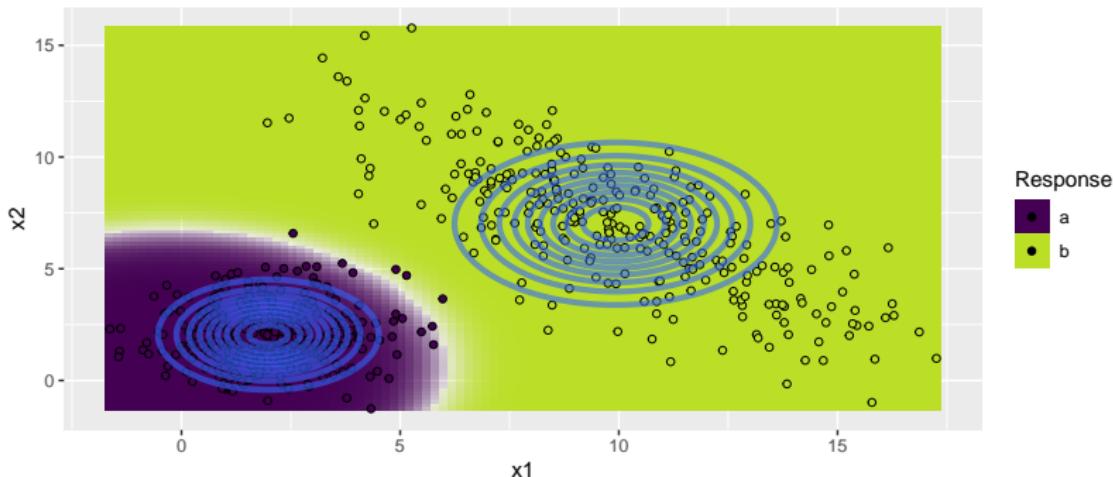
NB is based on a simple **conditional independence assumption**: the features are conditionally independent given class y .

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

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 (μ_j, σ_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

Performance Evaluation

k-NN

Classification and Regression Trees (CART)

Random Forests

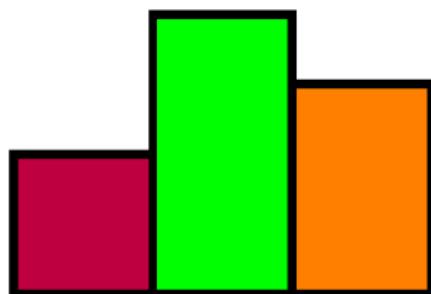
Tuning

Nested Resampling

mlr3

Introduction to Machine Learning

Evaluation: Generalization Error



Learning goals

- Understand the goal of performance estimation
- Know the formal definition of generalization error as a statistical estimator of future performance
- Understand the difference between GE for a model and GE for a learner.
- Understand the difference between outer and inner loss

PERFORMANCE ESTIMATION

- For a trained model, we want to know its future **performance**.
- Training works by ERM on $\mathcal{D}_{\text{train}}$ (inducer, loss, risk minimization):

$$\mathcal{I} : \mathbb{D} \times \Lambda \rightarrow \mathcal{H}, \quad (\mathcal{D}, \lambda) \mapsto \hat{f}_{\mathcal{D}, \lambda}.$$

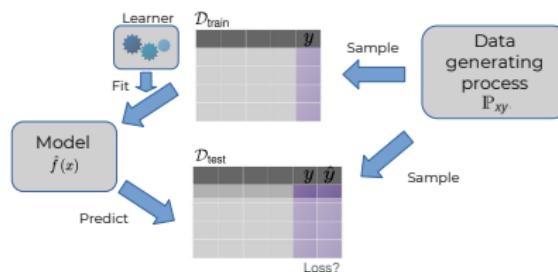
$$\min_{\theta \in \Theta} \sum_{i=1}^n L\left(y^{(i)}, f\left(\mathbf{x}^{(i)} \mid \theta\right)\right)$$

- Due to effects like overfitting, we cannot simply use this **training error** to gauge our model, this is likely optimistically biased.
(more on this later!)
- We want: the true expected loss, a.k.a. **generalization error**.
- To reliably estimate it, we need independent, unseen **test data**.
- This simply simulates the application of the model in reality.

GE FOR A FIXED MODEL

- GE for a fixed model: $\text{GE}(\hat{f}, L) := \mathbb{E} [L(y, \hat{f}(\mathbf{x}))]$
Expectation over a single, random test point $(\mathbf{x}, y) \sim \mathbb{P}_{xy}$.
- Estimator, **if a dedicated test set is available** (size m)

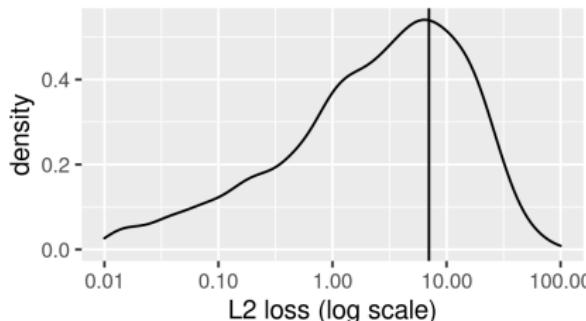
$$\widehat{\text{GE}}(\hat{f}, L) := \frac{1}{m} \sum_{(\mathbf{x}, y) \in \mathcal{D}_{\text{test}}} [L(y, \hat{f}(\mathbf{x}))]$$



NB: Very often, no dedicated test-set is available, and what we describe here is not same as hold-out splitting (see later).

EXAMPLE: TEST LOSS AS RANDOM VARIABLE

- For a fixed model and dedicated i.i.d. test set, we can easily approximate the complete test loss distribution $L(y, \hat{f}(\mathbf{x}))$.
- LM on `mlbench::friedman1` test problem
- With $n_{\text{train}} = 500$ we create a fixed model
- We feed 5000 fresh test points to model
- And plot the pointwise L_2 loss.



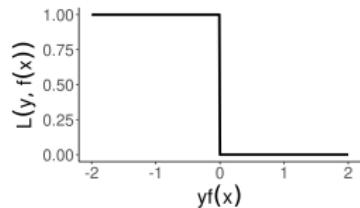
- The result is a unimodal distribution with long tails.
- Mean and one standard deviation to either side are highlighted in grey.

INNER VS OUTER LOSS

- Sometimes, we would like to evaluate our learner with a different loss L or metric ρ .
- Nomenclature: ERM and **inner loss**; evaluation and **outer loss**.
- Different losses, if computationally advantageous to deviate from outer loss of application; e.g., optimization faster with inner L2 or maybe no implementation for outer loss exists.

Example: Linear binary classifier / Logistic regression.

- Outside: We often want to eval with "nr of misclassified examples", so 0-1 loss.
- Problem: 0-1 neither differentiable nor continuous. Hence: Inner loss = binomial.
(0-1 actually NP hard).
- For evaluation, differentiability is not required.



SET-BASED PERFORMANCE METRICS

- Metric ρ measures quality of predictions as scalar on one test set.

$$\rho : \bigcup_{m \in \mathbb{N}} (\mathcal{Y}^m \times \mathbb{R}^{m \times g}) \rightarrow \mathbb{R}, \quad (\mathbf{y}, \mathbf{F}) \mapsto \rho(\mathbf{y}, \mathbf{F}).$$

- Needed as some metrics are not observation-based losses but defined on sets, e.g. AUC or metrics in survival analysis.
- For test data of size m , \mathbf{F} is prediction matrix

$$\mathbf{F} = \begin{bmatrix} \hat{f}(\mathbf{x}^{(1)}) \\ \vdots \\ \hat{f}(\mathbf{x}^{(m)}) \end{bmatrix} \in \mathbb{R}^{m \times g}$$

- Point-wise loss L can easily be extended to a ρ_L :

$$\rho_L(\mathbf{y}, \mathbf{F}) = \frac{1}{m} \sum_{i=1}^m L(y^{(i)}, \mathbf{F}^{(i)}) \quad \left(= \frac{1}{m} \sum_{i=1}^m L(y^{(i)}, \hat{f}(\mathbf{x}^{(i)})) \right).$$

MODEL GE VS. LEARNER GE

To clear up a major point of confusion (or totally confuse you):

- In ML we frequently face a weird situation.
- We are usually given a single data set, and at the end of our model fitting (and evaluation and selection) process, we will likely fit one model on exactly that complete data set.
- We only trust in unseen-test-error estimation – but have no data left for that final model.
- So in the construction of any practical estimator we cannot really use that final model!
- Hence, we will now evaluate the next best thing: The inducer, and the quality of a model produced when fitted on (nearly) the same number of points!

GENERALIZATION ERROR FOR INDUCER

$$\text{GE}(\mathcal{I}, \boldsymbol{\lambda}, n_{\text{train}}, \rho) := \lim_{n_{\text{test}} \rightarrow \infty} \mathbb{E} [\rho (\mathbf{y}, \mathbf{F}_{\mathcal{D}_{\text{test}}, \mathcal{I}(\mathcal{D}_{\text{train}}, \boldsymbol{\lambda})})]$$

- Quality of models when fitted with $\mathcal{I}_{\boldsymbol{\lambda}}$ on n_{train} points from \mathbb{P}_{xy} .
- Expectation **both** over $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$, sampled independently.
- This is estimated by all following **resampling** procedures.
- NB: All of the models produced during that phase of evaluation are only intermediate results.

GENERALIZATION ERROR FOR INDUCER

$$\text{GE}(\mathcal{I}, \boldsymbol{\lambda}, n_{\text{train}}, \rho) := \lim_{n_{\text{test}} \rightarrow \infty} \mathbb{E} [\rho (\mathbf{y}, \mathbf{F}_{\mathcal{D}_{\text{test}}, \mathcal{I}(\mathcal{D}_{\text{train}}, \boldsymbol{\lambda})})]$$

- We can already see a potential source of pessimistic bias in our estimator: While we would like to estimate a GE with $n_{\text{train}} = |\mathcal{D}|$, the size of the complete data set, in practice we can only do this for strictly smaller values, so that test data is left to work with.
- For pointwise losses ρ_L :

$$\text{GE}(\mathcal{I}, \boldsymbol{\lambda}, n_{\text{train}}, \rho_L) := \mathbb{E} [L(y, \mathcal{I}(\mathcal{D}_{\text{train}}, \boldsymbol{\lambda})(\mathbf{x})]$$

Expectation **both** over $\mathcal{D}_{\text{train}}$ and (\mathbf{x}, y) independently from \mathbb{P}_{xy} .

- Retcon for GE of model: GE of learner, conditional on $\mathcal{D}_{\text{train}}$

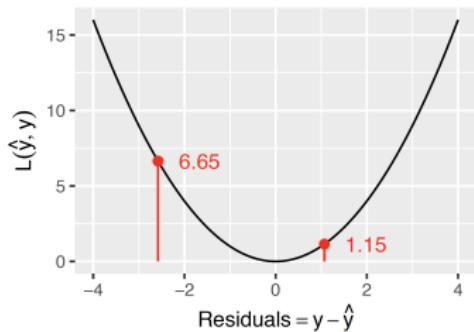
$$\text{GE}(\hat{f}, L) := \text{GE}(\mathcal{I}, \boldsymbol{\lambda}, n_{\text{train}}, \rho_L | \mathcal{D}_{\text{train}})$$

if $\hat{f} = \mathcal{I}(\mathcal{D}_{\text{train}}, \boldsymbol{\lambda})$ and $n_{\text{train}} = |\mathcal{D}_{\text{train}}|$.

Introduction to Machine Learning

Evaluation: Measures for Regression

Learning goals

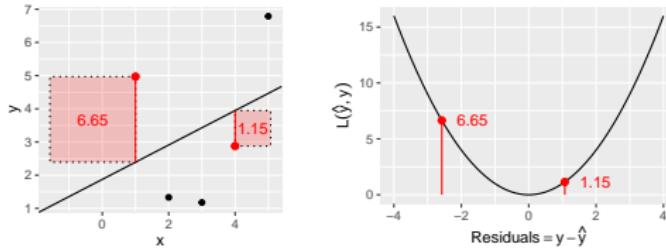


- Know the definitions of mean squared error (MSE) and mean absolute error (MAE)
- Understand the connections of MSE and MAE to L2 and L1 loss
- Know the definition of Spearman's ρ
- Know the definitions of R^2 and generalized R^2

MEAN SQUARED ERROR (MSE)

$$\rho_{MSE}(\mathbf{y}, \mathbf{F}) = \frac{1}{m} \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2 \in [0; \infty) \quad \rightarrow L2 \text{ loss.}$$

Outliers with large prediction error heavily influence the MSE, as they enter quadratically.



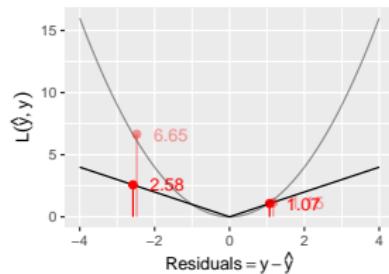
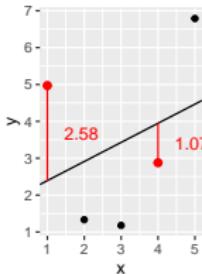
Similar measures:

- Sum of squared errors: $\rho_{SSE}(\mathbf{y}, \mathbf{F}) = \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2$
- Root MSE (orig. scale): $\rho_{RMSE}(\mathbf{y}, \mathbf{F}) = \sqrt{\frac{1}{m} \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2}$

MEAN ABSOLUTE ERROR

$$\rho_{MAE}(\mathbf{y}, \mathbf{F}) = \frac{1}{m} \sum_{i=1}^m |y^{(i)} - \hat{y}^{(i)}| \in [0; \infty) \quad \rightarrow L1 \text{ loss.}$$

More robust, less influenced by large residuals, more intuitive than MSE.



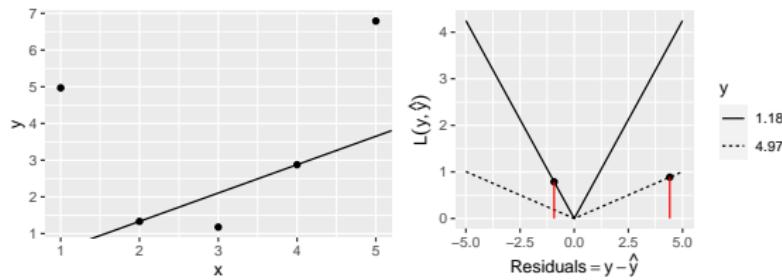
Similar measures:

- Median absolute error (for even more robustness)

MEAN ABSOLUTE PERCENTAGE ERROR

$$\rho_{MAPE}(\mathbf{y}, \mathbf{F}) = \frac{1}{m} \sum_{i=1}^m \left| \frac{y^{(i)} - \hat{y}^{(i)}}{y^{(i)}} \right| \in [0; \infty)$$

Small $|y|$ influence more strongly. Cannot handle $y = 0$.



Similar measures:

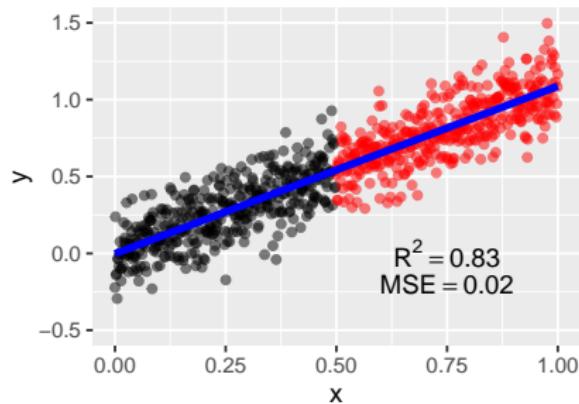
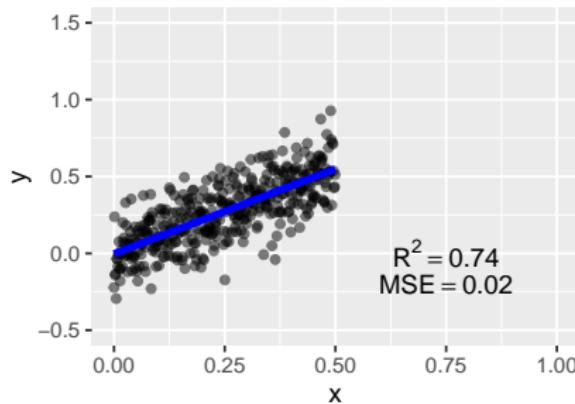
- Mean Absolute Scaled Error (MASE)
- Symmetric Mean Absolute Percentage Error (sMAPE)

$$\rho_{R^2}(\mathbf{y}, \mathbf{F}) = 1 - \frac{\sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2}{\sum_{i=1}^m (y^{(i)} - \bar{y})^2} = 1 - \frac{SSE_{LinMod}}{SSE_{Intercept}}.$$

- Well-known classical measure for LMs – on train data.
- "Fraction of variance explained" by the model.
- How much SSE of constant baseline is reduced when we use more complex model?
- $\rho_{R^2} = 1$: all residuals are 0, we predict perfectly,
- $\rho_{R^2} = 0.9$: LM reduces SSE by factor of 10.
 $\rho_{R^2} = 0$: we predict as badly as the constant model.
- Is $\in [0, 1]$ on train data; as LM is always better than intercept.

R^2 VS MSE

- Better R^2 does not necessarily imply better fit.
- Data: $y = 1.1x + \epsilon$, where $\epsilon \sim \mathcal{N}(0, 0.15)$.
- Fit half (black) and full data (black and red) with LM.



- Fit does not improve, but R^2 goes up.
- But: Invariant w.r.t. to linear scaling of y , MSE is not.

GENERALIZED R^2 FOR ML

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

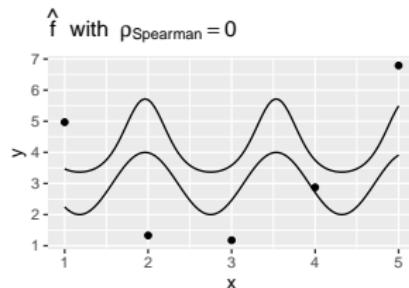
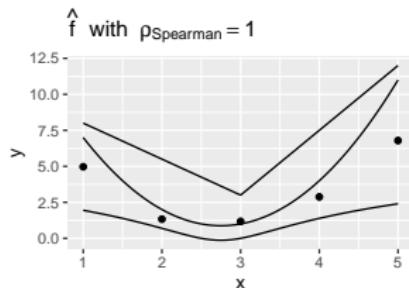
- E.g., model vs constant, LM vs non-linear model, tree vs forest, model with fewer features vs model with more, ...
- We could use arbitrary measures.
- In ML we would rather evaluate on test set.
- Can then become negative, e.g., for SSE and constant baseline, if our model fairs worse on the test set than a simple constant.

SPEARMAN'S ρ

Can be used if we care about the relative ranks of predictions:

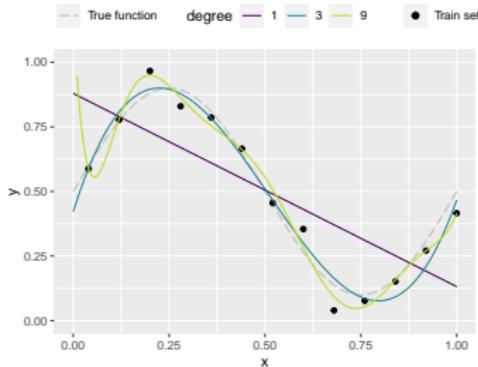
$$\rho_{\text{Spearman}}(\mathbf{y}, \mathbf{\hat{F}}) = \frac{\text{Cov}(\text{rg}(\mathbf{y}), \text{rg}(\mathbf{\hat{y}}))}{\sqrt{\text{Var}(\text{rg}(\mathbf{y}))} \cdot \sqrt{\text{Var}(\text{rg}(\mathbf{\hat{y}}))}} \in [-1, 1],$$

- Very robust against outliers
- A value of 1 or -1 means that $\hat{\mathbf{y}}$ and \mathbf{y} have a perfect monotonic relationship.
- Invariant under monotone transformations of $\hat{\mathbf{y}}$



Introduction to Machine Learning

Evaluation: Training Error



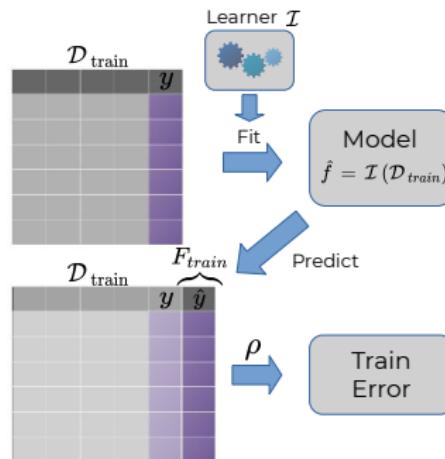
Learning goals

- Understand the definition of training error
- Understand why train error is unreliable for models of higher complexity when overfitting can occur

TRAINING ERROR

Simply plugin predictions for data that model has been trained on:

$$\rho(\mathbf{y}_{\text{train}}, F_{\text{train}}) \text{ where } F_{\text{train}} = \begin{bmatrix} \hat{f}_{\mathcal{D}_{\text{train}}}(\mathbf{x}_{\text{train}}^{(1)}) \\ \dots \\ \hat{f}_{\mathcal{D}_{\text{train}}}(\mathbf{x}_{\text{train}}^{(m)}) \end{bmatrix}$$

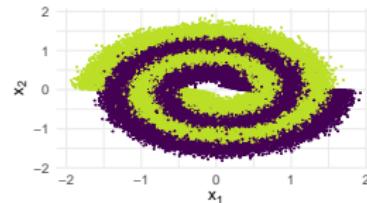


A.k.a. apparent error or resubstitution error.

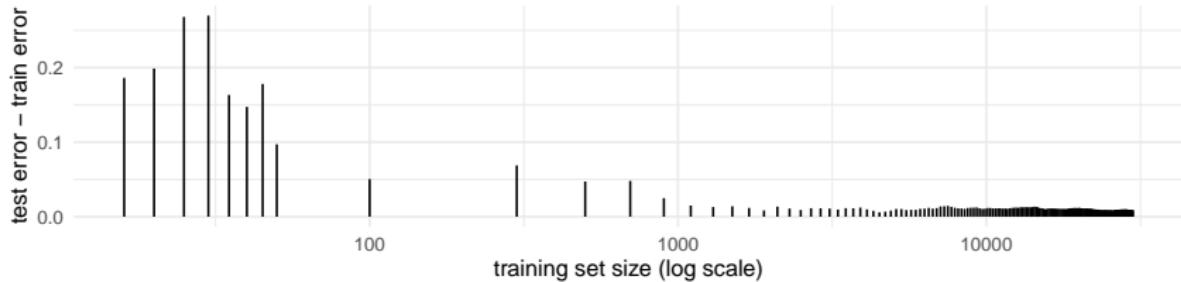
EXAMPLE 1: KNN

For large data, and some models, train error **can maybe** yield a good approximation of the GE:

- Use k -NN ($k = 15$).
- Up to 30K points from **spirals** to train.
- Use very large extra set for testing
(to measure "true GE").

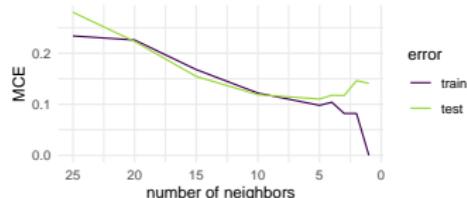


We increase train size, and see how gap between train error and GE closes.

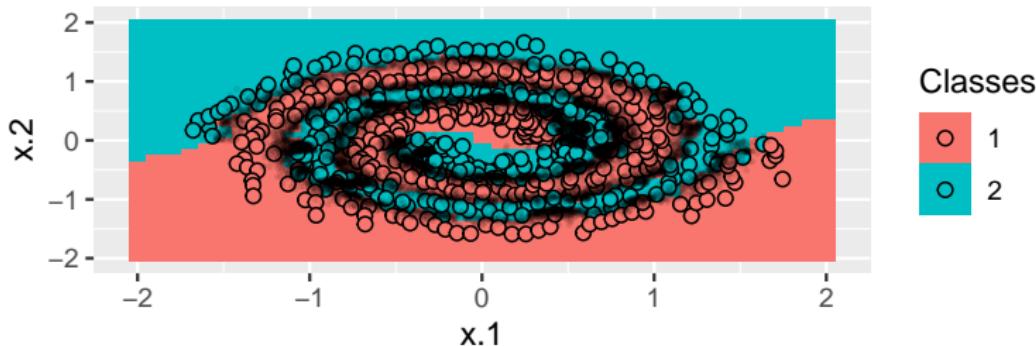


EXAMPLE 1: KNN

- Fix train size to 500 and vary k .
- Low train error for small k is deceptive.
Model is very local and overfits.



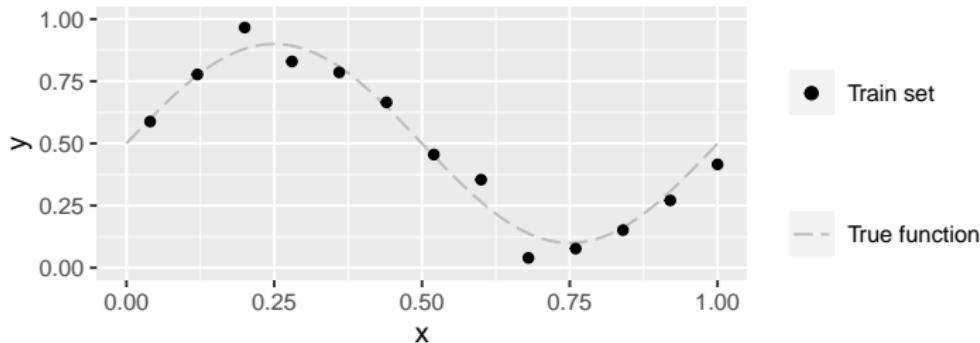
$k = 2$; trainerr = 0.08, testerr = 0.14



Black region are misclassifications from large test test.

EXAMPLE 2: POLYNOMIAL REGRESSION

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



We fit a d^{th} -degree polynomial:

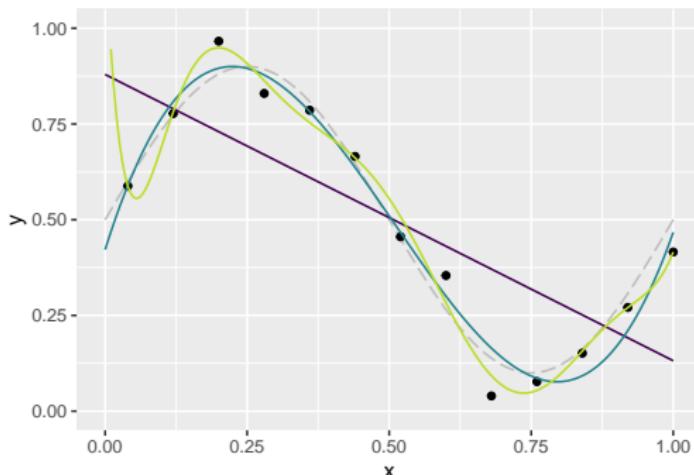
$$f(\mathbf{x} | \boldsymbol{\theta}) = \theta_0 + \theta_1 \mathbf{x} + \cdots + \theta_d \mathbf{x}^d = \sum_{j=0}^d \theta_j \mathbf{x}^j.$$

EXAMPLE 2: POLYNOMIAL REGRESSION

Simple model selection problem: Which d ?

Visual inspection vs quantitative MSE on training set:

— True function degree 1 3 9 ● Train set



- $d = 1$: MSE = 0.036: clearly underfitting
- $d = 3$: MSE = 0.003: pretty OK
- $d = 9$: MSE = 0.001: clearly overfitting

Using the train error chooses overfitting model of maximal complexity.

TRAIN ERROR CAN EASILY BECOME 0

- For 1-NN it is always 0 as each $\mathbf{x}^{(i)}$ is its own NN at test time.
- Extend any ML training in the following way: After normal fitting, we also store the training data. During prediction, we first check whether x is already stored in this set. If so, we replicate its label. The train error of such an (unreasonable) procedure will be 0.
- There are so called interpolators - interpolating splines, interpolating Gaussian processes - whose predictions can always perfectly match the regression targets, they are not necessarily good as they will interpolate noise, too.

CLASSICAL STATISTICAL GOF MEASURES

- **Goodness-of-fit** measures like R^2 , likelihood, AIC, BIC, deviance are all based on the training error.
- For models of restricted capacity, and enough data, and non-violated distributional assumptions: they might work.
- Hard to gauge when that breaks, for high-dim, more complex data.
- How do you compare to non-param ML-like models?

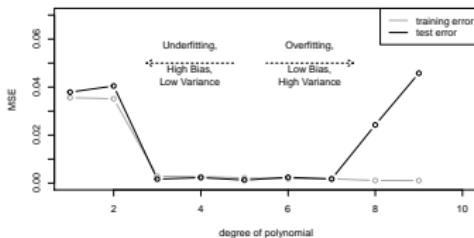
Out-of-sample testing is probably always a good idea!

Introduction to Machine Learning

Evaluation: Test Error

Learning goals

- Understand the definition of test error
- Understand that test error is more reliable than train error
- Bias-Variance analysis of holdout splitting

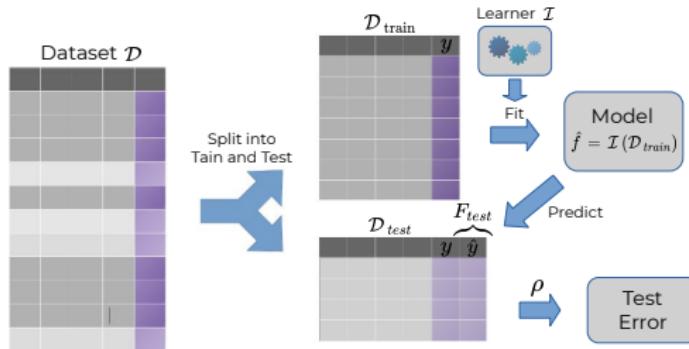


TEST ERROR AND HOLD-OUT SPLITTING

- Simulate prediction on unseen data, to avoid optimistic bias:

$$\rho(\mathbf{y}_{\text{test}}, F_{\text{test}}) \text{ where } F_{\text{test}} = \begin{bmatrix} \hat{f}_{D_{\text{train}}}(\mathbf{x}_{\text{test}}^{(1)}) \\ \dots \\ \hat{f}_{D_{\text{train}}}(\mathbf{x}_{\text{test}}^{(m)}) \end{bmatrix}$$

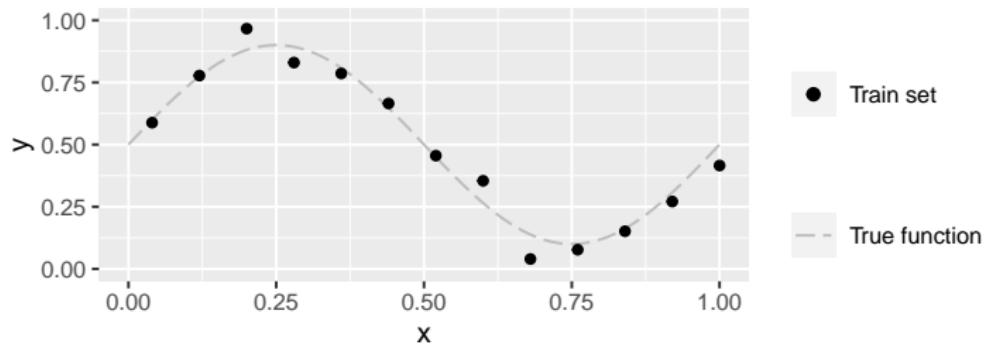
- Partition data, e.g., 2/3 for train and 1/3 for test.



A.k.a. holdout splitting.

EXAMPLE: POLYNOMIAL REGRESSION

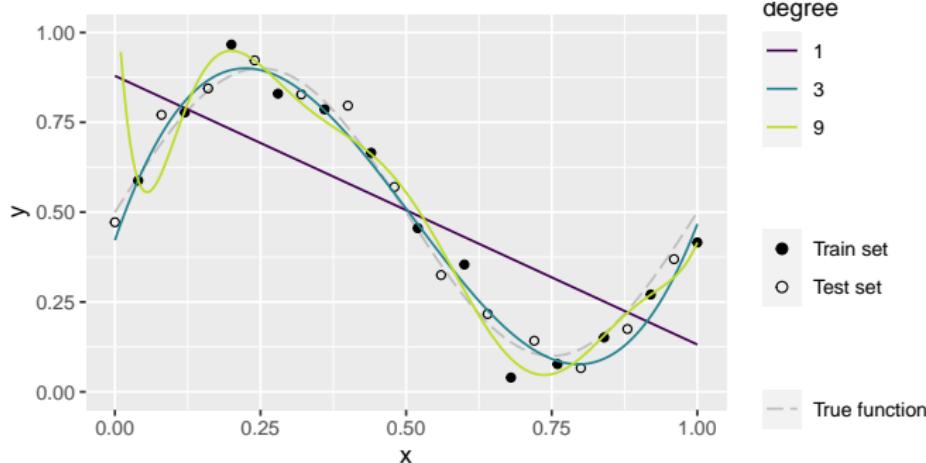
Previous example:



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

EXAMPLE: POLYNOMIAL REGRESSION

Now with fresh test data:

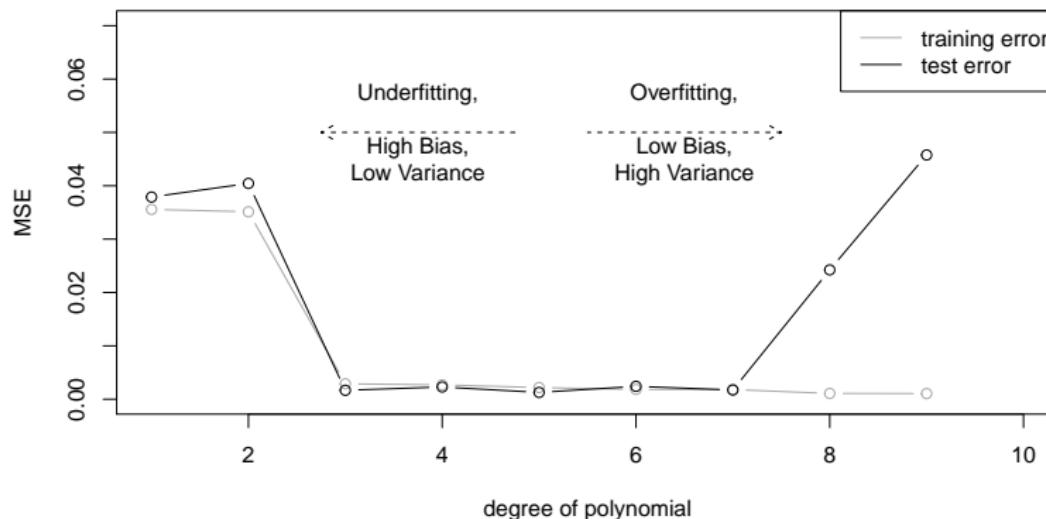


- $d = 1$: MSE = 0.038: clearly underfitting
- $d = 3$: MSE = 0.002: pretty OK
- $d = 9$: MSE = 0.046: clearly overfitting

While train error monotonically decreases in d , test error shows that high- d polynomials overfit.

TEST ERROR

Let's plot train and test MSE for all d :



Increasing model complexity tends to cause

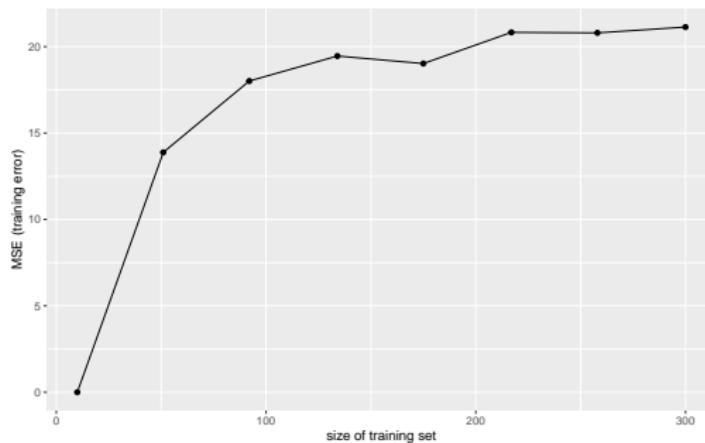
- a decrease in training error, and
- a U-shape in test error
(first underfit, then overfit, sweet-spot in the middle).

TRAINING VS. TEST ERROR

- Boston Housing data
- Polynomial regression (without interactions)

The training error...

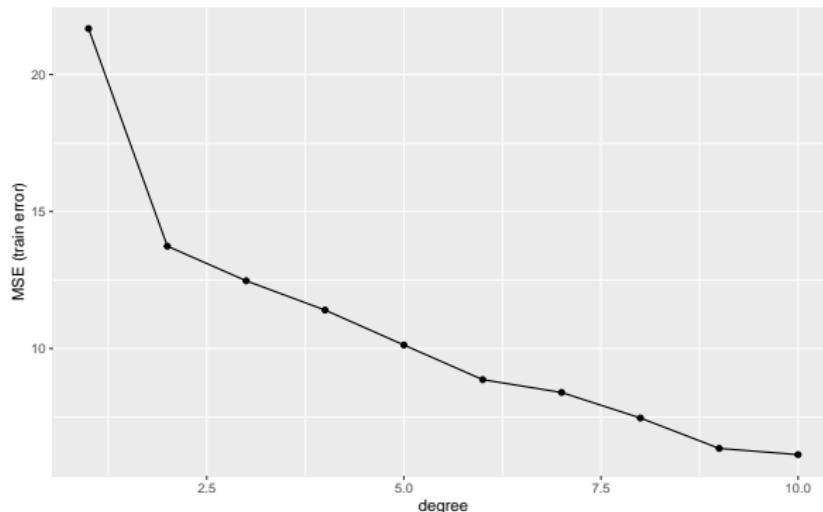
- decreases with smaller training set size as it becomes easier for the model to learn all observed patterns perfectly.



TRAINING VS. TEST ERROR

The training error...

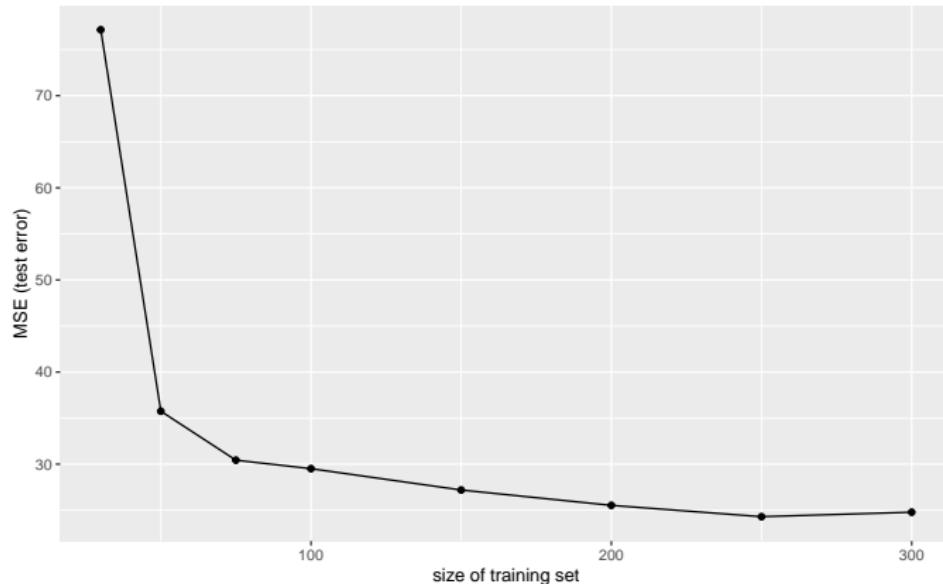
- decreases with increasing model complexity as the model gets better at learning more complex structures.



TRAINING VS. TEST ERROR

The test error...

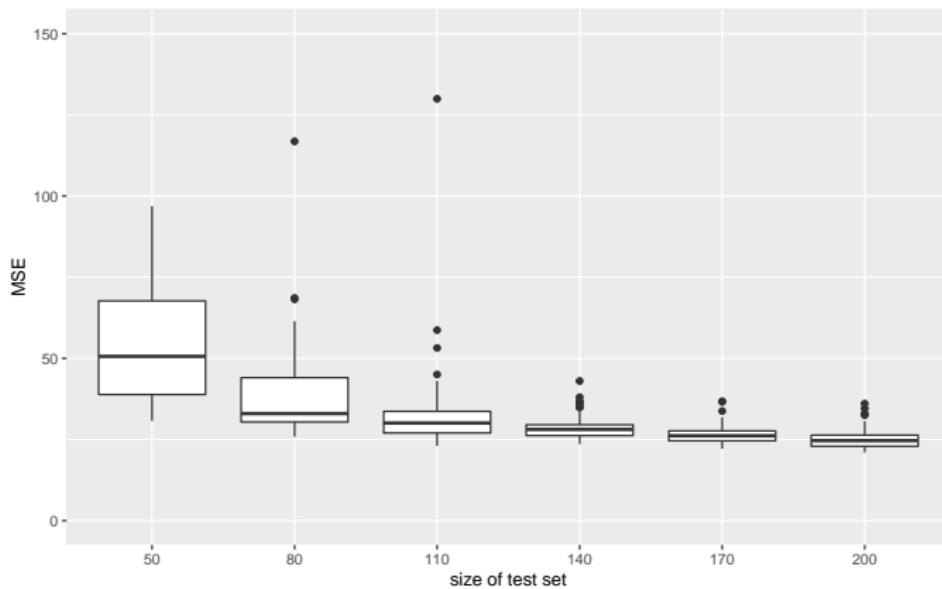
- will typically decrease with larger training set size as the model generalizes better with more data to learn from.



TRAINING VS. TEST ERROR

The test error...

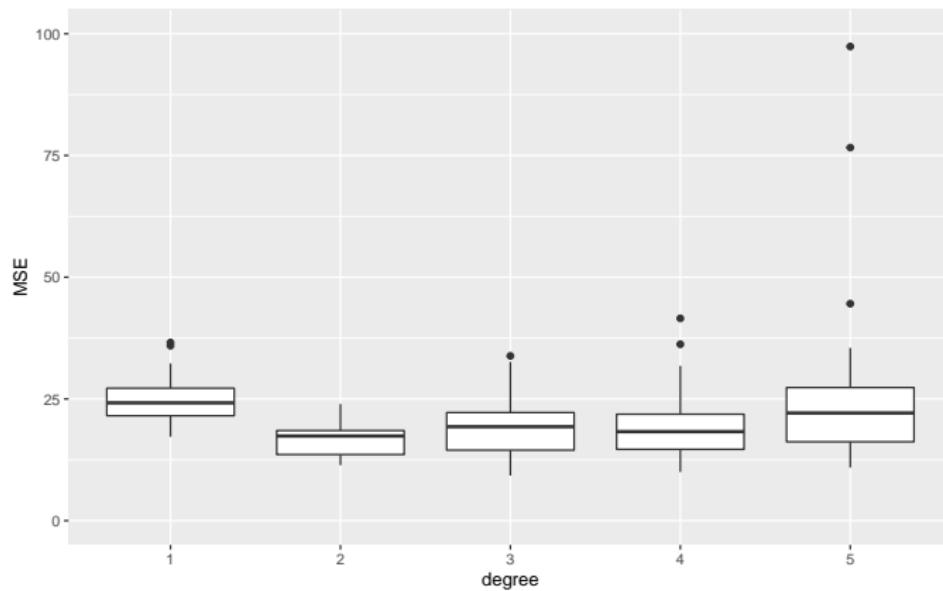
- will have higher variance with smaller test set size.



TRAINING VS. TEST ERROR

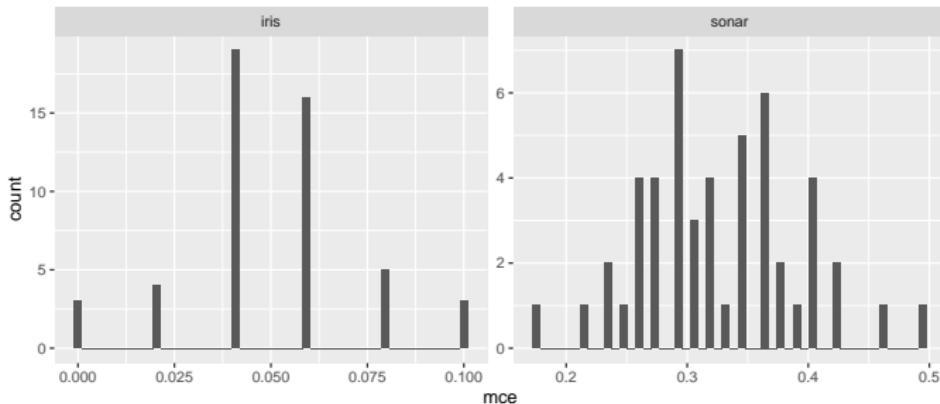
The test error...

- will have higher variance with increasing model complexity.



BIAS AND VARIANCE

- Test error is a good estimator of GE, given a) we have enough data b) test data is representative i.i.d.
- Estimates for smaller test sets can fluctuate considerably – this is why we use resampling in such situations.
Repeated $\frac{2}{3}$ / $\frac{1}{3}$ holdout splits:
`iris` ($n = 150$) and `sonar` ($n = 208$).



BIAS-VARIANCE OF HOLD-OUT – EXPERIMENT

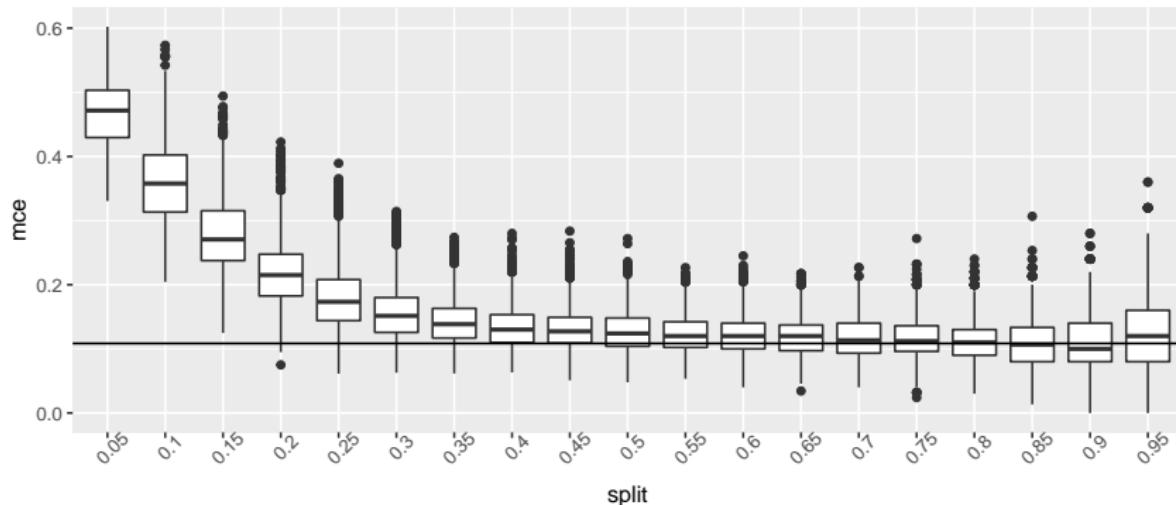
Hold-out sampling produces a trade-off between **bias** and **variance** that is controlled by split ratio.

- Smaller training set → poor fit, pessimistic bias in \widehat{GE} .
- Smaller test set → high variance.

Experiment:

- `spirals` data ($sd = 0.1$), with CART tree.
- Goal: estimate real performance of a model with $|\mathcal{D}_{\text{train}}| = 500$.
- Split rates $s \in \{0.05, 0.10, \dots, 0.95\}$ with $|\mathcal{D}_{\text{train}}| = s \cdot 500$.
- Estimate error on $\mathcal{D}_{\text{test}}$ with $|\mathcal{D}_{\text{test}}| = (1 - s) \cdot 500$.
- 50 repeats for each split rate.
- Get "true" performance by often sampling 500 points, fit learner, then eval on 10^5 fresh points.

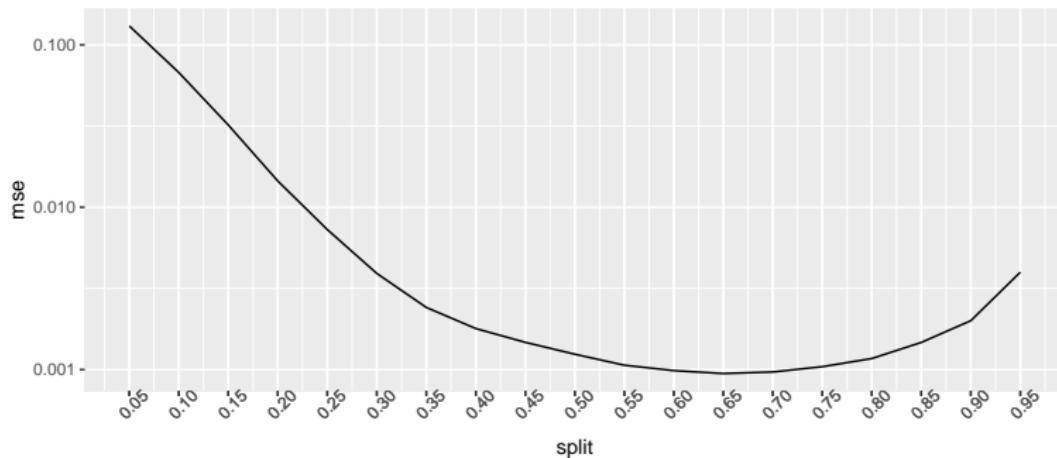
BIAS-VARIANCE OF HOLD-OUT – EXPERIMENT



- Clear pessimistic bias for small training sets – we learn a much worse model than with 500 observations.
- But increase in variance when test sets become smaller.

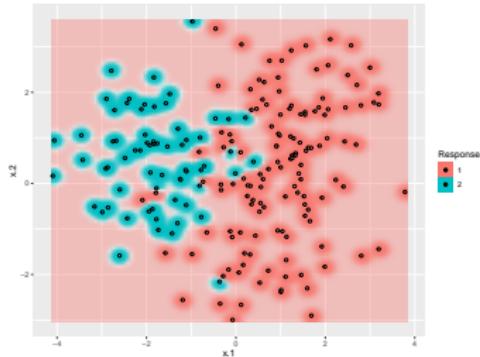
BIAS-VARIANCE OF HOLD-OUT – EXPERIMENT

- Let's now plot the MSE of the holdout estimator.
- NB: Not MSE of model, but squared difference between estimated holdout values and true performance (horiz. line in prev. plot).
- Best estimator is ca. train set ratio of 2/3.
- NB: This is a single experiment and not a scientific study, but this rule-of-thumb has also been validated in larger studies.



Introduction to Machine Learning

Evaluation: Overfitting and Underfitting



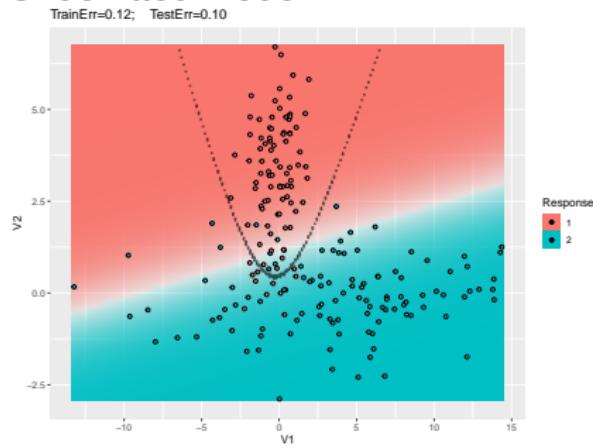
Learning goals

- Understand definitions of overfitting and underfitting

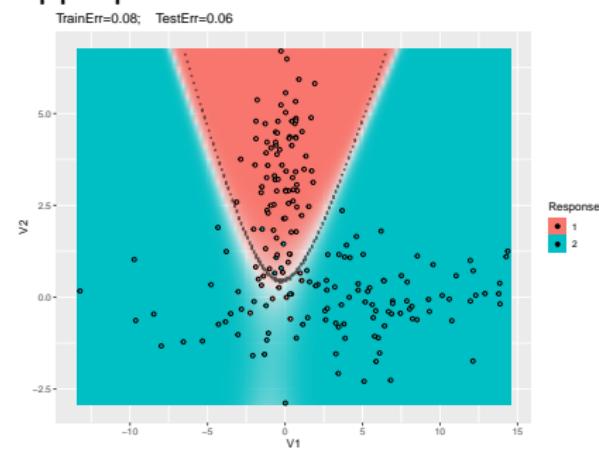
UNDERFITTING

- Occurs if model does not reflect true shape of underlying function
- Hence, predictions will be less good as they could be
- High train error and high test error
- Hard to detect, as we don't know what the Bayes error is for a task

Underfitted model



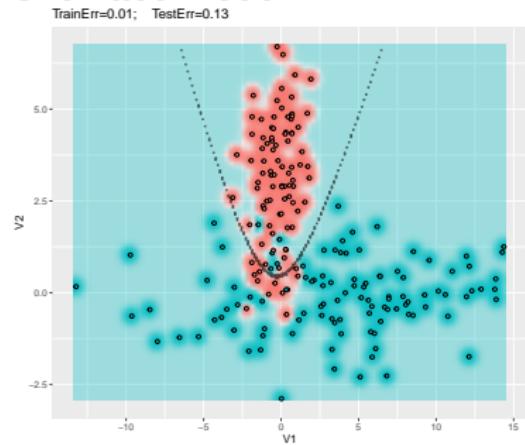
Appropriate model



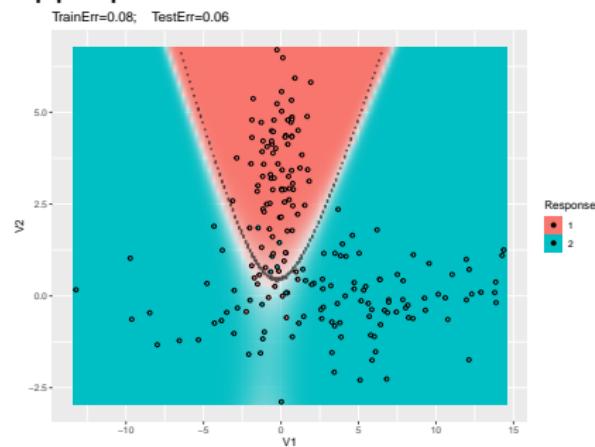
OVERFITTING

- Overfitting occurs when the model reflects noise or artifacts in training data, which do not generalize
- Small train error, at cost of test high error
- Hence, predictions of overfitting models cannot be trusted - but proper ML evaluation workflows should make it visible

Overfitted model

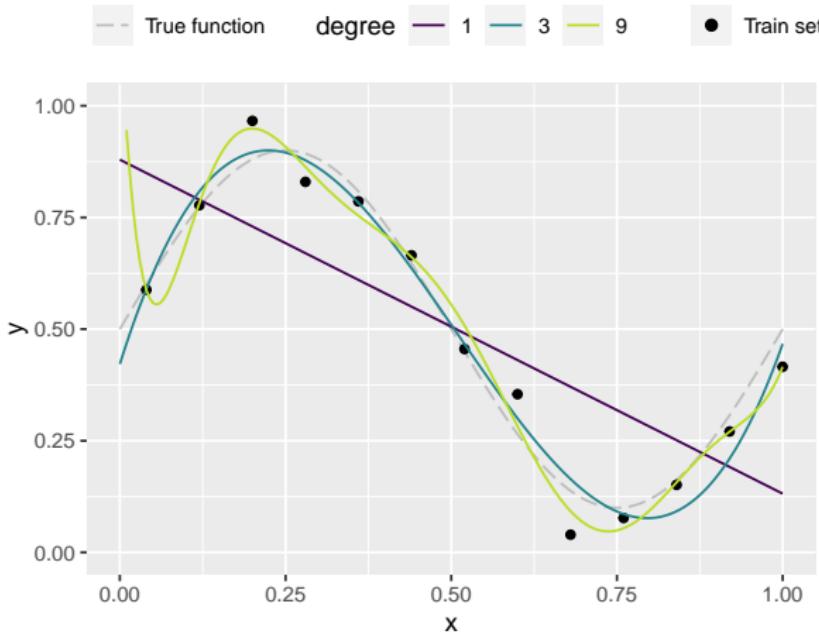


Appropriate model



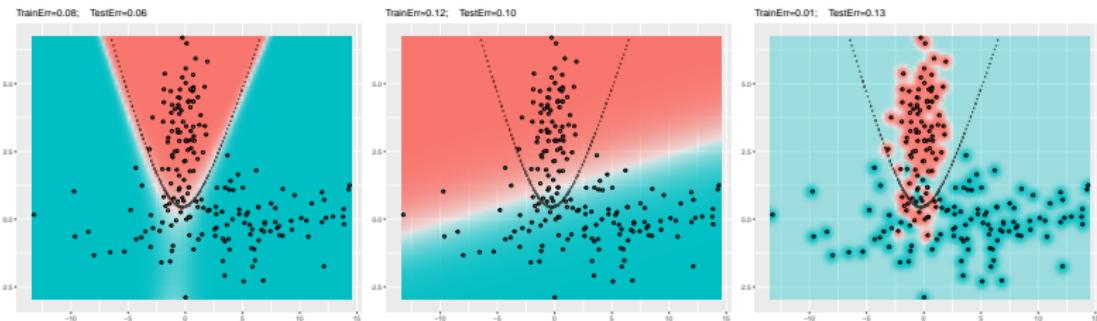
UNDER- AND OVERFITTING IN REGRESSION

- Poly-Regression, on data from sinusoidal function
- LM underfits, high-d overfits



MATHEMATICAL DEFINITIONS

- Nearly no reference does that, here is one approach
- Underfitting $UF(\hat{f}, L) := GE(\hat{f}, L) - GE(f^*, L)$
Diff in GE between \hat{f} and the Bayes optimal model
- Overfitting $OF(\hat{f}, L) := GE(\hat{f}, L) - \mathcal{R}_{\text{emp}}(\hat{f}, L)$
Diff between (theoretical) GE and training error



NB: Now, RHS is both UF and OF, let's say OF has "prio".

OVERFITTING TRADE-OFFS

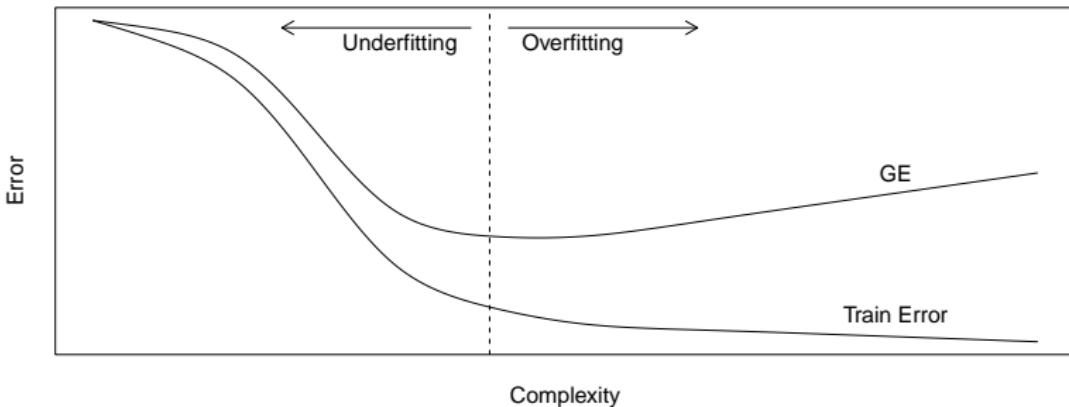
The potential for overfitting is influenced by:

- Complexity of hypothesis space
- Amount of training data
- Dimensionality of feature space
- Irreducible noise

Implications:

- The larger / more complex is \mathcal{H} , the more data we need to tell candidate models apart
- The less data we have, the more we need to stick with "constrained" \mathcal{H}
- OF can happen for LMs too: If feature space is very high-dim
- Tightly connected to the bias-var-noise decomposition of GE of a learner (\rightarrow which we study elsewhere).

COMPLEXITY VS GE



- Common U-shape of GE if complexity or train-rounds go up.
- Optimal level of complexity:
Simplest model for which GE is not significantly outperformed
- We could also call "Point of OF" the point where GE goes up.

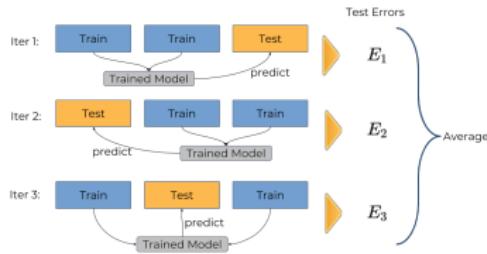
AVOIDING OVERFITTING

- Use more or better data – not always possible, but maybe can augment data, e.g., for images
- Constrain \mathcal{H} directly by using less complex model classes
- Many learners come with HPs that can constrain complexity
- Use "early-stopping"
- Occam's razor in model selection: If GE not strongly reduced for more complex class, use the simpler model.

All of the above are methods of regularization, which we study in a dedicated chapter.

Introduction to Machine Learning

Evaluation: Resampling 1

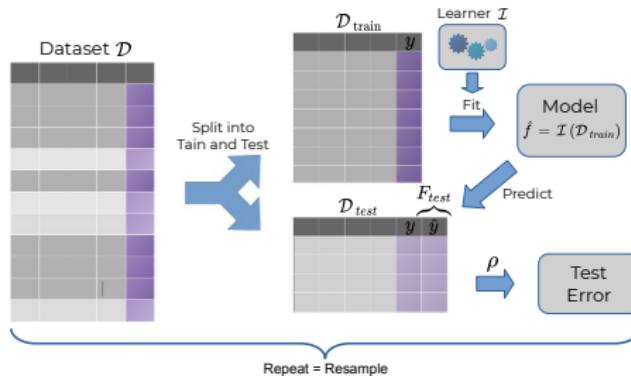


Learning goals

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

RESAMPLING

- **Goal:** estimate $\text{GE}(\mathcal{I}, \lambda, n, \rho_L) = \mathbb{E}[L(y, \mathcal{I}_\lambda(\mathcal{D}_{\text{train}})(\mathbf{x}))]$.
- Holdout: Small trainset = high pessimistic bias; small testset = high var.
- Resampling: Repeatedly split in train and test, then average results.
- Allows to have large trainsets large (low pessimistic bias) since we use $\text{GE}(\mathcal{I}, \lambda, n_{\text{train}}, \rho)$ as a proxy for $\text{GE}(\mathcal{I}, \lambda, n, \rho)$
- And reduce var from small testsets via averaging over repetitions.



RESAMPLING STRATEGIES

- Represent train and test sets by index vectors::

$$J_{\text{train}} \in \{1, \dots, n\}^{n_{\text{train}}} \text{ and } J_{\text{test}} \in \{1, \dots, n\}^{n_{\text{test}}}$$

- Resampling strategy = collection of splits:

$$\mathcal{J} = ((J_{\text{train},1}, J_{\text{test},1}), \dots, (J_{\text{train},B}, J_{\text{test},B})).$$

- Resampling estimator:

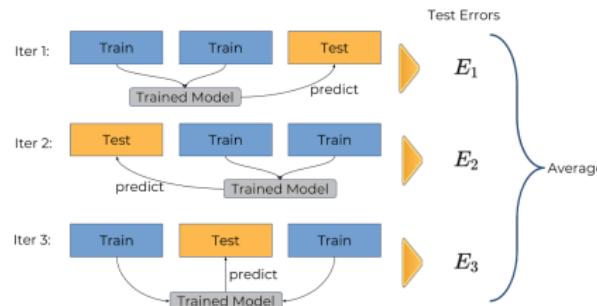
$$\widehat{\text{GE}}(\mathcal{I}, \mathcal{J}, \rho, \boldsymbol{\lambda}) = \text{agr}\left(\rho\left(\mathbf{y}_{J_{\text{test},1}}, \mathbf{F}_{J_{\text{test},1}, \mathcal{I}(\mathcal{D}_{\text{train},1}, \boldsymbol{\lambda})}\right), \right. \\ \vdots \\ \left. \rho\left(\mathbf{y}_{J_{\text{test},B}}, \mathbf{F}_{J_{\text{test},B}, \mathcal{I}(\mathcal{D}_{\text{train},B}, \boldsymbol{\lambda})}\right)\right),$$

- Aggregation agr is typically "mean" and $n_{\text{train}} \approx n_{\text{train},1} \approx \dots \approx n_{\text{train},B}$.

CROSS-VALIDATION

- Split the data into k roughly equally-sized partitions.
- Each part is test set once, join $k - 1$ parts for training.
- Obtain k test errors and average.
- Fraction $(k - 1)/k$ is used for training, so 90% for 10CV
- Each observation is tested exactly once.

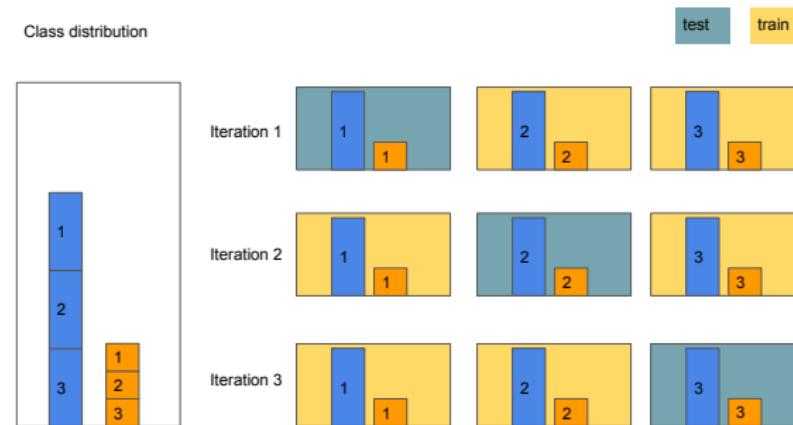
Example: 3-fold CV



CROSS-VALIDATION - STRATIFICATION

- Used when target classes are very imbalanced
- Then small classes can randomly get very small in samples
- Preserve distrib of target (or any feature) in each fold
- For classes: simply CV-split the class data, then join

Example: stratified 3-fold cross-validation

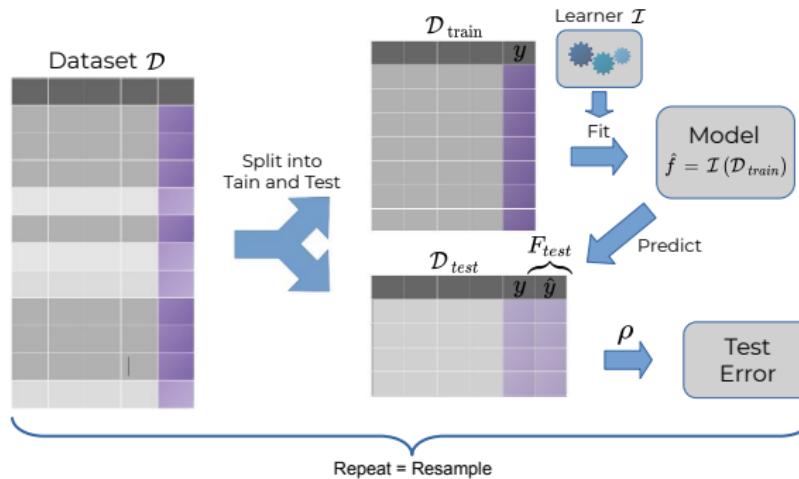


CROSS-VALIDATION

- 5 or 10 folds are common.
- $k = n$ is known as "leave-one-out" CV (LOO-CV)
- Bias of \widehat{GE} : The more folds, the smaller. LOO nearly unbiased.
- LOO has high var, better many folds for small data but not LOO
- Repeated CV (avg over high-fold CVs) good for small data.

SUBSAMPLING

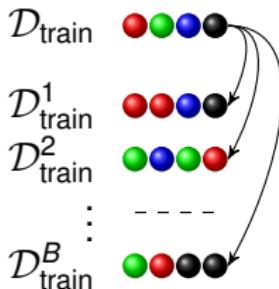
- Repeated hold-out with averaging, a.k.a. Monte Carlo CV.
- Typical choices for splitting: $\frac{4}{5}$ or $\frac{9}{10}$ for training.



- Smaller subsampling rate = larger pessimistic bias
- More reps = smaller var

BOOTSTRAP

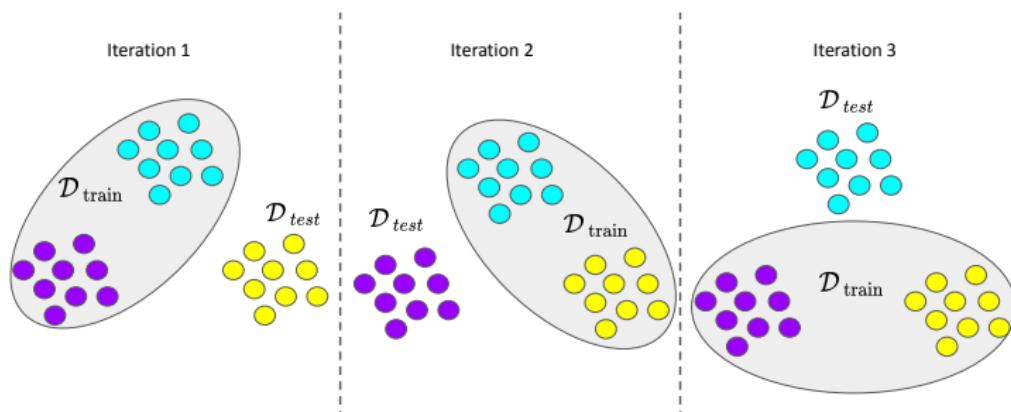
- Draw B trainsets of size n with replacement from orig \mathcal{D}
- Testsets = Out-Of-Bag points: $\mathcal{D}_{\text{test}}^b = \mathcal{D} \setminus \mathcal{D}_{\text{train}}^b$



- Similar analysis as for subsampling
- Trainsets contain about 2/3 unique points:
$$1 - \mathbb{P}((\mathbf{x}, y) \notin \mathcal{D}_{\text{train}}) = 1 - \left(1 - \frac{1}{n}\right)^n \xrightarrow{n \rightarrow \infty} 1 - \frac{1}{e} \approx 63.2\%$$
- Replicated train points can lead to problems and artifacts
- Extensions B632 and B632+ also use trainerr for better estimate when data very small

LEAVE-ONE-OBJECT-OUT

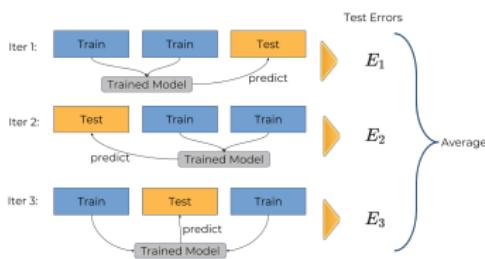
- Used when we have multiple obs from same objects, e.g., persons or hospitals or base images
- Data not i.i.d. any more
- Data from same object should **either** be in train **or** testset
- Otherwise we likely bias \widehat{GE}
- CV on objects, or leave-one-object-out



Introduction to Machine Learning

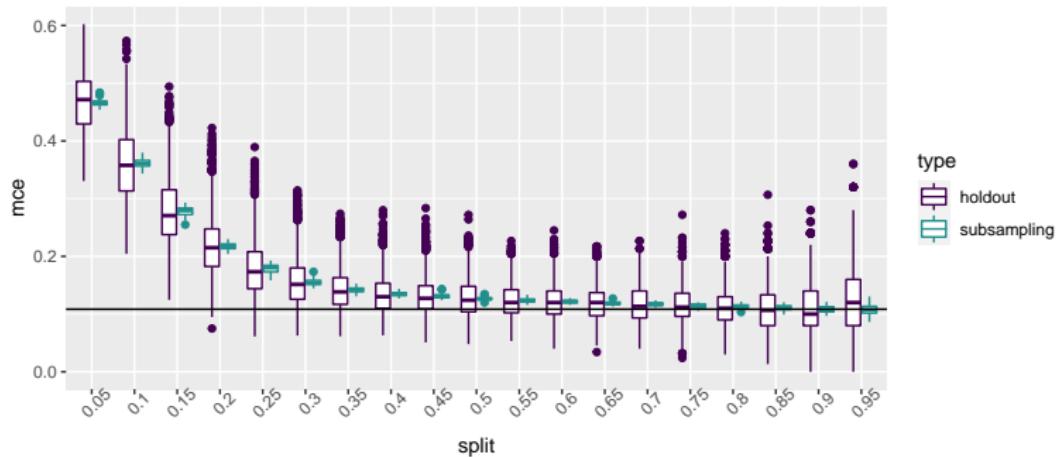
Evaluation: Resampling 2

Learning goals



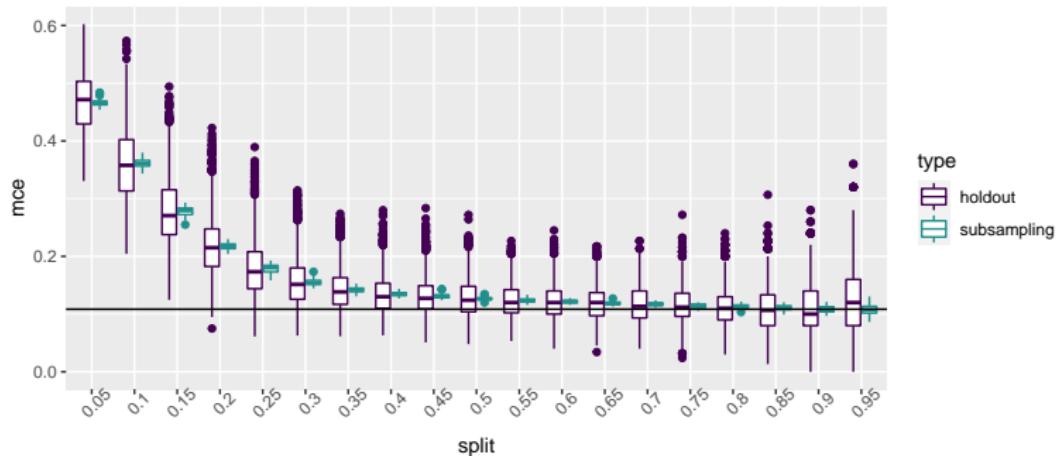
- Understand why resampling is better estimator than hold-out
- In-depth bias-var analysis of resampling estimator
- Understand that CV does not produce independent samples
- Short guideline for practical use

BIAS-VARIANCE ANALYSIS FOR SUBSAMPLING



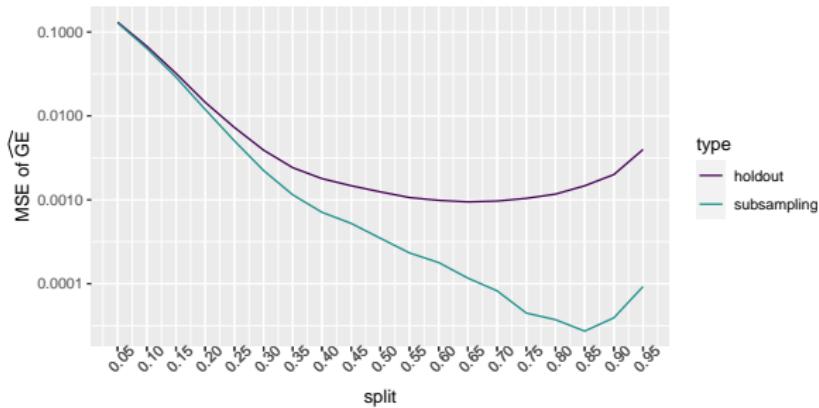
- Reconsider bias-var experiment for holdout (maybe re-read)
- Split rates $s \in \{0.05, 0.1, \dots, 0.95\}$ with $|\mathcal{D}_{\text{train}}| = s \cdot 500$.
- Holdout vs. subsampling with 50 iters
- 50 replications

BIAS-VARIANCE ANALYSIS FOR SUBSAMPLING



- Both estimators are compared to "real" MCE (black line)
- SS same pessimistic bias as holdout for given s, but much less var

BIAS-VARIANCE ANALYSIS FOR SUBSAMPLING



- $\widehat{\text{MSE}}$ of $\widehat{\text{GE}}$ strictly better for SS
- Smaller var of SS enables to use larger s for optimal choice
- The optimal split rate now is a higher $s \approx 0.8$.
- Beyond $s = 0.8$: MSE goes up because var doesn't go down as much as we want due to increasing overlap in trainsets (see later)

DEDICATED TESTSET SCENARIO - ANALYSIS

- Goal: estimate $GE(\hat{f}) = \mathbb{E}[L(y, \hat{f}(\mathbf{x}))]$ via

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

Here, only (\mathbf{x}, y) are random, they are m i.i.d. fresh test samples

- This is: average over i.i.d $L(y, \hat{f}(\mathbf{x}))$, so directly know \mathbb{E} and var.
And can use CLT to approx distrib of $\widehat{GE}(\hat{f})$ with Gaussian.
- $\mathbb{E}[\widehat{GE}(\hat{f})] = \mathbb{E}[L(y, \hat{f}(\mathbf{x}))] = GE(\hat{f})$
- $\mathbb{V}[\widehat{GE}(\hat{f})] = \frac{1}{m} \mathbb{V}[L(y, \hat{f}(\mathbf{x}))]$
- So $\widehat{GE}(\hat{f})$ is unbiased estimator of $GE(\hat{f})$, var decreases linearly in testset size, have an approx of full distrib (can do NHST, CIs, etc.)
- NB: Gaussian may work less well for e.g. 0-1 loss, with \mathbb{E} close to 0, can use binomial or other special approaches for other losses

PESSIMISTIC BIAS IN RESAMPLING

- Estim $\text{GE}(\mathcal{I}, n)$ (surrogate for $\text{GE}(\hat{f})$ when \hat{f} is fit on full \mathcal{D} , with $|\mathcal{D}| = n$) via resampling based estim $\widehat{\text{GE}}(\mathcal{I}, n_{\text{train}})$

$$\begin{aligned}\widehat{\text{GE}}(\mathcal{I}, \mathcal{J}, \rho, \boldsymbol{\lambda}) = \text{agr} & \left(\rho \left(\mathbf{y}_{J_{\text{test},1}}, \mathbf{F}_{J_{\text{test},1}, \mathcal{I}(\mathcal{D}_{\text{train},1}, \boldsymbol{\lambda})} \right), \right. \\ & \vdots \\ & \left. \rho \left(\mathbf{y}_{J_{\text{test},B}}, \mathbf{F}_{J_{\text{test},B}, \mathcal{I}(\mathcal{D}_{\text{train},B}, \boldsymbol{\lambda})} \right) \right),\end{aligned}$$

- Let's assume agr is avg and ρ is loss-based, so ρ_L
- The ρ are simple holdout estims. So:

$$\mathbb{E}[\widehat{\text{GE}}(\mathcal{I}, \mathcal{J}, \rho, \boldsymbol{\lambda})] \approx \mathbb{E}[\rho \left(\mathbf{y}_{J_{\text{test}}}, \mathbf{F}_{J_{\text{test}}, \mathcal{I}(\mathcal{D}_{\text{train}}, \boldsymbol{\lambda})} \right)]$$

- NB1: In above, as always for $\text{GE}(\mathcal{I})$, both $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$ (and so $\mathbf{x} \in \mathcal{D}_{\text{test}}$) are random vars, and we take E over them
- NB2: Need \approx as maybe not all train/test sets in resampling of exactly same size

PESSIMISTIC BIAS IN RESAMPLING

$$\mathbb{E}[\widehat{\text{GE}}(\mathcal{I}, \mathcal{J}, \rho, \lambda)] \approx \mathbb{E}[\rho\left(\mathbf{y}_{J_{\text{test}}}, \mathbf{F}_{J_{\text{test}}, \mathcal{I}(\mathcal{D}_{\text{train}}, \lambda)}\right)] = \\ \mathbb{E}\left[\frac{1}{m} \sum_{(\mathbf{x}, y) \in \mathcal{D}_{\text{test}}} L(y, \mathcal{I}(\mathcal{D}_{\text{train}})(\mathbf{x}))\right] = \text{GE}(\mathcal{I}, n_{\text{train}})$$

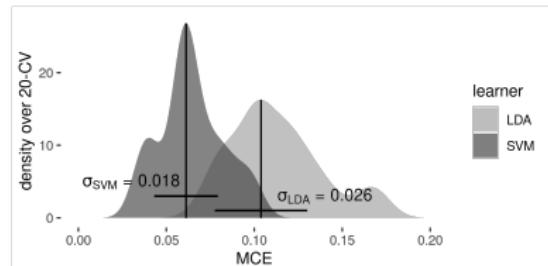
⇒

- So when we use $\widehat{\text{GE}}(\mathcal{I}, \mathcal{J}, \rho, \lambda)$ to estimate $\text{GE}(\mathcal{I}, n)$, our expected value is nearly correct, it's $\text{GE}(\mathcal{I}, n_{\text{train}})$
- But fitting \mathcal{I} on less data (n_{train} vs full n) usually results in model with worse perf, hence estimator is pessimistically biased
- Bias the stronger, the smaller our training splits in resampling.

NO INDEPENDENCE OF CV RESULTS

- Similar analysis as before holds for CV
- Might be tempted to report distribution or SD of individual CV split perf values, e.g. to test if perf of 2 learners is significantly different
- But k CV splits are not independent

A t-test on the difference of the mean GE estimators yields a highly significant p-value of $\approx 7.9 \cdot 10^{-5}$ on the 95% level.

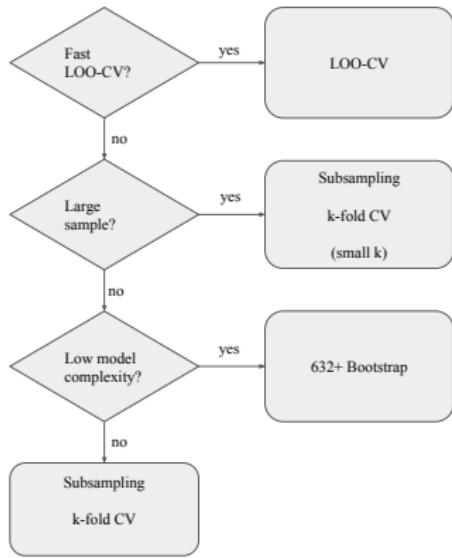


LDA vs SVM on `spam` classification problem, performance estimation via 20-CV w.r.t. MCE.

NO INDEPENDENCE OF CV RESULTS

- $\widehat{\text{V}[GE]}$ of CV is a difficult combination of
 - average variance as we estim on finite trainsets
 - covar from test errors, as models result from overlapping trainsets
 - covar due to the dependence of trainsets and test obs appear in trainsets
- Naively using the empirical var of k individual \widehat{GE} s (as on slide before) yields biased estimator of $\widehat{\text{V}[GE]}$. Usually this underestimates the true var!
- Worse: there is no unbiased estimator of $\widehat{\text{V}[GE]}$ [Bengio, 2004]
- Take into account when comparing learners by NHST
- Somewhat difficult topic, we leave it with the warning here

SHORT GUIDELINE



- 5-CV or 10-CV have become standard.
- Do not use hold-out, CV with few folds, or SS with small split rate for small n . Can bias estim and have large var.
- For small n , e.g. $n < 200$, use LOO or, probably better, repeated CV.
- For some models, fast tricks for LOO exist
- With $n = 100.000$, can have "hidden" small-sample size, e.g. one class very small
- SS usually better than bootstrapping. Repeated obs can cause problems in training, especially in nested setups where the "training" set is split up again.

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:

- 1 Class labels:

$$\mathbf{F} = \left(\hat{o}_k^{(i)} \right)_{i \in \{1, \dots, m\}, k \in \{1, \dots, g\}} \in \mathbb{R}^{m \times g},$$

where $\hat{o}_k^{(i)} = [\hat{y}^{(i)} = k], k = 1, \dots, g$ is the one-hot-encoded class label prediction.

- 2 Class probabilities:

$$\mathbf{F} = \left(\hat{\pi}_k^{(i)} \right)_{i \in \{1, \dots, m\}, k \in \{1, \dots, g\}} \in [0, 1]^{m \times g}$$

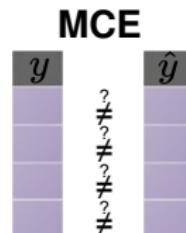
→ These form the basis for evaluation.



LABELS: MCE & ACC

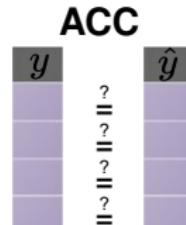
The **misclassification error rate (MCE)** counts the number of incorrect predictions and presents them as a rate:

$$\rho_{MCE} = \frac{1}{m} \sum_{i=1}^m [y^{(i)} \neq \hat{y}^{(i)}] \in [0, 1].$$



Accuracy (ACC) is defined in a similar fashion for correct classifications:

$$\rho_{ACC} = \frac{1}{m} \sum_{i=1}^m [y^{(i)} = \hat{y}^{(i)}] \in [0, 1].$$



- If the data set is small this can be brittle.
- MCE says nothing about how good/skewed predicted probabilities are.
- Errors on all classes are weighted equally, which is often inappropriate.

LABELS: CONFUSION MATRIX

Much better than reducing prediction errors to a simple number is tabulating them in a **confusion matrix**:

- true classes in columns,
- predicted classes in rows.

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

		True classes				n
Predicted classes	setosa	versicolor	virginica	error	error	
	setosa	50	0	0	0	50
	versicolor	0	46	4	4	50
	virginica	0	4	46	4	50
	error	0	4	4	8	-
n		50	50	50	-	150

LABELS: CONFUSION MATRIX

- In binary classification, we typically call one class "positive" and the other "negative".
- The positive class is the more important, often smaller one.

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

e.g.,

- **True Positive (TP)** means that an instance is classified as positive that is really positive (correct prediction).
- **False Negative (FN)** means that an instance is classified as negative that is actually positive (incorrect prediction).

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: Depending on certain features (age, income, profession, ...) a bank wants to decide whether to grant a 10,000 EUR loan.

Predict if a person is solvent (yes / no).

Should the bank lend them the money?

Exemplary costs:

Loss in event of default: 10,000 EUR

Income through interest paid: 100 EUR

		True classes	
		solvent	not solvent
Predicted classes	solvent	0	10,000
	not solvent	100	0

LABELS: COSTS

Cost matrix

		True classes	
		solvent	not solvent
Predicted classes	solvent	0	10,000
	not solvent	100	0

Confusion matrix

		True classes	
		solvent	not solvent
Predicted classes	solvent	70	3
	not solvent	7	20

- If the bank gives everyone a credit, who was predicted as *solvent*, the costs are at:

$$\begin{aligned} Costs &= \frac{1}{n} \sum_{i=1}^n C[y^{(i)}, \hat{y}^{(i)}] \\ &= \frac{1}{100} (100 \cdot 7 + 0 \cdot 70 + 10,000 \cdot 3 + 0 \cdot 20) = 307 \end{aligned}$$

- If the bank gives everyone a credit, the costs are at:

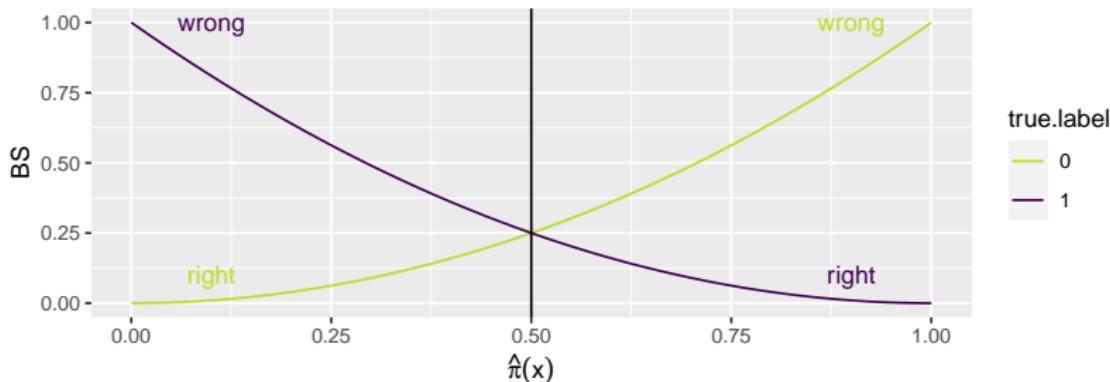
$$Costs = \frac{1}{100} (100 \cdot 0 + 0 \cdot 77 + 10,000 \cdot 23 + 0 \cdot 0) = 2.300$$

PROBABILITIES: BRIER SCORE

Measures squared distances of probabilities from the true class labels:

$$\rho_{BS} = \frac{1}{m} \sum_{i=1}^m \left(\hat{\pi}^{(i)} - y^{(i)} \right)^2$$

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



PROBABILITIES: BRIER SCORE

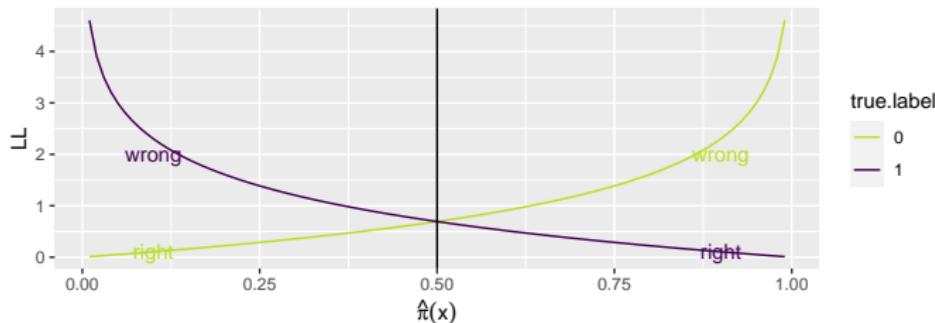
$$\rho_{BS,MC} = \frac{1}{m} \sum_{i=1}^m \sum_{k=1}^g \left(\hat{\pi}_k^{(i)} - o_k^{(i)} \right)^2$$

- Original by Brier, works also for multiple classes.
- $o_k^{(i)} = [y^{(i)} = k]$ marks the one-hot-encoded class label.
- For the binary case, $\rho_{BS,MC}$ is twice as large as ρ_{BS} : in $\rho_{BS,MC}$, we sum the squared difference for each observation regarding both class 0 **and** class 1, not only the true class.

PROBABILITIES: LOG-LOSS

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

$$\rho_{LL} = \frac{1}{m} \sum_{i=1}^m \left(-y^{(i)} \log(\hat{\pi}^{(i)}) - (1 - y^{(i)}) \log(1 - \hat{\pi}^{(i)}) \right).$$



- Optimal value is 0, “confidently wrong” is penalized heavily.
- Multi-class version: $\rho_{LL,MC} = -\frac{1}{m} \sum_{i=1}^m \sum_{k=1}^g o_k^{(i)} \log(\hat{\pi}_k^{(i)})$.

Introduction to Machine Learning

Evaluation: ROC Basics

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 = $\frac{TP+TN}{TOTAL}$

CLASS IMBALANCE

- Assume a binary classifier diagnoses a serious medical condition.
- Label distribution is often **imbalanced**, i.e, not many people have the disease.
- Evaluating on mce is often inappropriate for scenarios with imbalanced labels:
 - Assume that only 0.5 % have the disease.
 - Always predicting “no disease” has an mce of 0.5 %, corresponding to very high accuracy.
 - This sends all sick patients home → bad system
- This problem is known as the **accuracy paradox**.

CLASS IMBALANCE

Classifying all observations as “no disease” (green) yields top accuracy simply because the “disease” occurs so rarely → accuracy paradox.

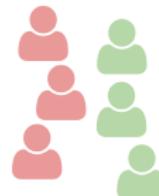


IMBALANCED COSTS

- Another point of view is **imbalanced costs**.
- In our example, classifying a sick patient as healthy should incur a much higher cost than classifying a healthy patient as sick.
- The costs depend a lot on what happens next: we can well assume that our system is some type of screening filter, and often the next step after labeling someone as sick might be a more invasive, expensive, but also more reliable test for the disease.
- Erroneously subjecting someone to this step is undesirable (psychological, economic, medical expense), but sending someone home to get worse or die seems much more so.
- Such situations not only arise under label imbalance, but also when costs differ (even though classes might be balanced).
- We could see this as imbalanced costs of misclassification, rather than imbalanced labels; both situations are tightly connected.

IMBALANCED COSTS

Imbalanced costs: classifying incorrectly as “no disease” incurs very high cost.



- Problem: if we were able to specify costs precisely, we could evaluate or even optimize on them.
- This important subfield of ML is called **cost-sensitive learning**, which we will not cover in this lecture unit.
- Unfortunately, users find it notoriously hard to come up with precise cost figures in imbalanced scenarios.
- Evaluating “from different perspectives”, with multiple metrics, often helps to get a first impression of system quality.

ROC ANALYSIS

- **ROC analysis** is a subfield of ML which studies the evaluation of binary prediction systems.
- ROC stands for “receiver operating characteristics” and was initially developed by electrical engineers and radar engineers during World War II for detecting enemy objects in battlefields – still has the funny name.



<http://media.iwm.org.uk/iwm/mediaLib//39/media-39665/large.jpg>

LABELS: ROC METRICS

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

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

- True positive rate ρ_{TPR} : how many of the true 1s did we predict as 1?
- True Negative rate ρ_{TNR} : how many of the true 0s did we predict as 0?
- Positive predictive value ρ_{PPV} : if we predict 1, how likely is it a true 1?
- Negative predictive value ρ_{NPV} : if we predict 0, how likely is it a true 0?
- Accuracy ρ_{ACC} : how many instances did we predict correctly?

LABELS: ROC METRICS

Example:

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

https://en.wikipedia.org/wiki/Receiver_operating_characteristic

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 $= \frac{\sum \text{Condition positive}}{\sum \text{Total population}}$
Predicted condition	Predicted condition positive	True positive, Power	False positive, Type I error	Positive predictive value (PPV), Precision $= \frac{\sum \text{True positive}}{\sum \text{Predicted condition positive}}$	Accuracy (ACC) = $\frac{\sum \text{True positive} + \sum \text{True negative}}{\sum \text{Total population}}$
	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) $= \frac{\sum \text{True negative}}{\sum \text{Condition negative}}$	Negative likelihood ratio (LR-) $= \frac{\text{FNR}}{\text{TNR}}$		

► Clickable version/picture source

► Interactive diagram

LABELS: F_1 MEASURE

- It is difficult to achieve high **positive predictive value** and high **true positive rate** simultaneously.
- A classifier predicting more positive will be more sensitive (higher ρ_{TPR}), but it will also tend to give more *false* positives (lower ρ_{TNR} , lower ρ_{PPV}).
- A classifier that predicts more negatives will be more precise (higher ρ_{PPV}), but it will also produce more *false* negatives (lower ρ_{TPR}).

The F_1 **score** balances two conflicting goals:

- ❶ Maximizing positive predictive value
- ❷ Maximizing true positive rate

ρ_{F_1} is the harmonic mean of ρ_{PPV} and ρ_{TPR} :

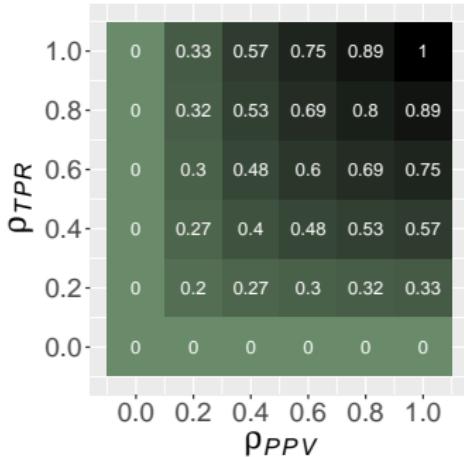
$$\rho_{F_1} = 2 \cdot \frac{\rho_{PPV} \cdot \rho_{TPR}}{\rho_{PPV} + \rho_{TPR}}$$

Note that this measure still does not account for the number of true negatives.

LABELS: F_1 MEASURE

F_1 score for different combinations of ρ_{PPV} & ρ_{TPR} .

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



- A model with $\rho_{TPR} = 0$ (no positive instance predicted as positive) or $\rho_{PPV} = 0$ (no true positives among the predicted) has $\rho_{F_1} = 0$.
- Always predicting “negative”: $\rho_{F_1} = 0$.
- Always predicting “positive”: $\rho_{F_1} = 2 \cdot \rho_{PPV}/(\rho_{PPV} + 1) = 2 \cdot n_+/(n_+ + n)$, which will be small when the size of the positive class n_+ is small.

WHICH METRIC TO USE?

- As we have seen, there is a plethora of methods.
→ This leaves practitioners with the question of which to use.
- Consider a small benchmark study.
 - We let k -NN, logistic regression, a classification tree, and a random forest compete on classifying the `credit_risk` data.
 - The data consist of 1000 observations of borrowers' financial situation and their creditworthiness (good/bad) as target.
 - Predicted probabilities are thresholded at 0.5 for the positive class.
 - Depending on the metric we use, learners are ranked differently according to performance (value of respective performance measure in parentheses):

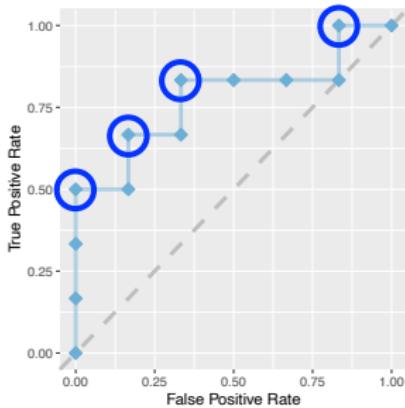
metric	k-NN	logistic regression	random forest	CART
TPR	2 (0.8777)	3 (0.8647)	1 (0.9257)	4 (0.8357)
TNR	4 (0.3764)	2 (0.4797)	3 (0.4072)	1 (0.4911)
PPV	4 (0.7665)	1 (0.7947)	3 (0.7842)	2 (0.7925)
F1	3 (0.8179)	2 (0.8279)	1 (0.8488)	4 (0.8130)
AUC	4 (0.7092)	2 (0.7731)	1 (0.7902)	3 (0.7293)
ACC	4 (0.7270)	2 (0.7490)	1 (0.7700)	3 (0.7320)

WHICH METRIC TO USE?

- We need not expect overly large discrepancies in general, but neither will we always see an unambiguous picture.
- Different metrics emphasize different aspects of performance.
→ The choice should be made in the domain context.
- For practitioners it is vital to understand what should be evaluated exactly, and which measure is appropriate.
 - Regarding credit risk, for instance, defaults are to be avoided, but not at all cost.
 - The bank must undertake a certain risk to remain profitable, so a more balanced measure such as the F_1 score might be in order.
 - On the other hand, a system detecting weapons at an airport should be able to achieve very high true positive rates, even if this comes at the expense of some false alarms.

Introduction to Machine Learning

Evaluation: Measures for Binary Classification: ROC Visualization

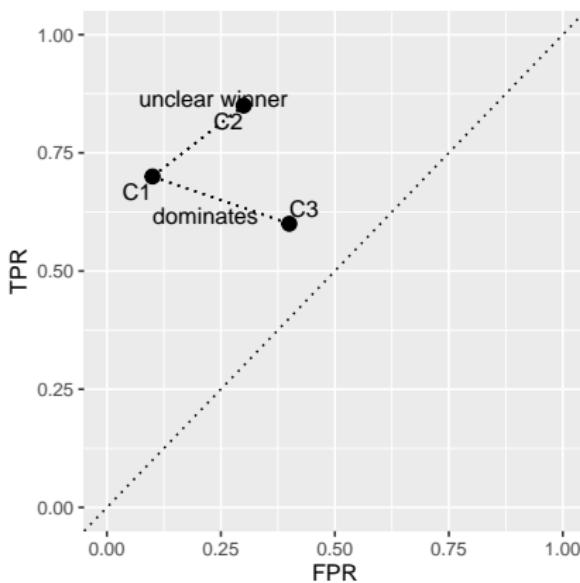


Learning goals

- Understand ROC curve
- Be able to compute a ROC curve manually
- Understand that ROC curve is invariant to class priors at test-time
- Discuss threshold selection
- Understand AUC

LABELS: ROC SPACE

- For comparing classifiers, we characterize them by their TPR and FPR values and plot them in a coordinate system.
- We could also use two different ROC metrics which define a trade-off, for instance, TPR and PPV.



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

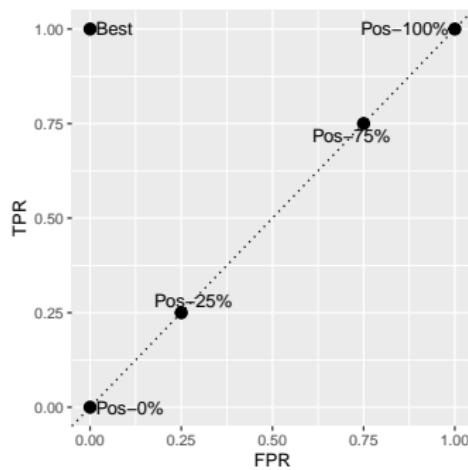
$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

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

LABELS: ROC SPACE

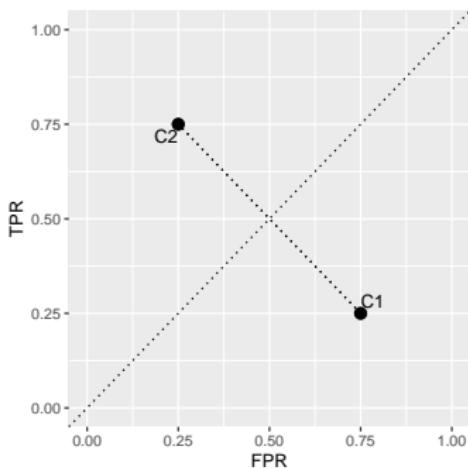
- The best classifier lies on the top-left corner, where FPR equals 0 and TPR is maximal.
- The diagonal is worst as it corresponds to a classifier producing random labels (with different proportions).

- If each positive x will be randomly classified with 25% as "pos", $\text{TPR} = 0.25$.
- If we assign each negative x randomly to "pos", $\text{FPR} = 0.25$.



LABELS: ROC SPACE

- In practice, we should never obtain a classifier below the diagonal.
- Inverting the predicted labels ($0 \mapsto 1$ and $1 \mapsto 0$) will result in a reflection at the diagonal.
⇒ $\text{TPR}_{\text{new}} = 1 - \text{TPR}$ and $\text{FPR}_{\text{new}} = 1 - \text{FPR}$.



LABEL DISTRIBUTION IN TPR AND FPR

TPR and FPR (ROC curves) are insensitive to the class distribution in the sense that they are 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

$$\text{MCE} = 35/100 = 0.35$$

$$\text{TPR} = 0.8$$

$$\text{FPR} = 0.5$$

Example 2:

Proportion $n_+/n_- = 2$

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

$$\text{MCE} = 45/150 = 0.3$$

$$\text{TPR} = 0.8$$

$$\text{FPR} = 0.5$$

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

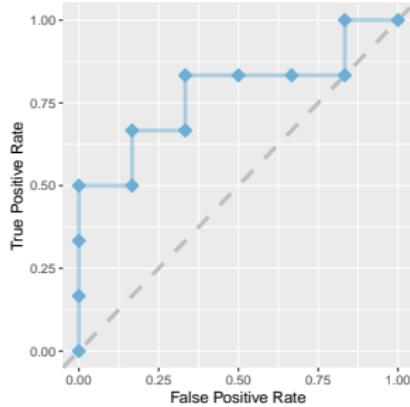
FROM PROBABILITIES TO LABELS: ROC CURVE

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

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

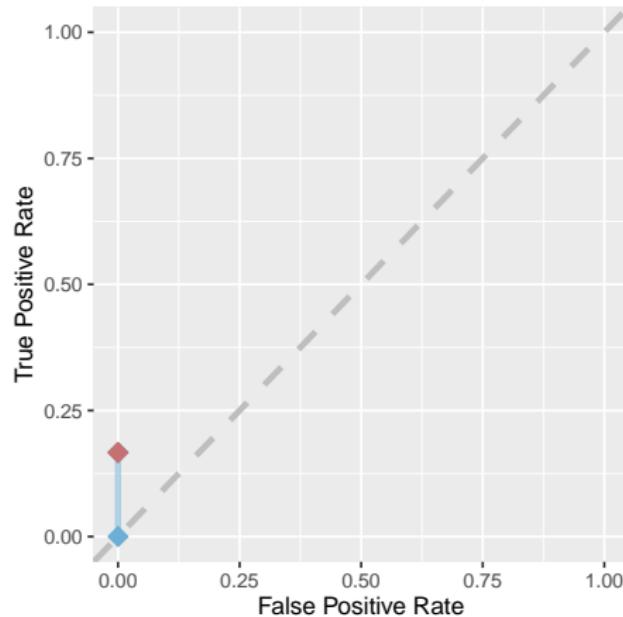
To draw a ROC curve:

- ➊ Rank test observations on decreasing score.
- ➋ Start with $c = 1$, so we start in $(0, 0)$; we predict everything as negative.
- ➌ Iterate through all possible thresholds c and proceed for each observation x as follows:
 - If x is positive, move TPR $1/n_+$ up, as we have one TP more.
 - If x is negative, move FPR $1/n_-$ right, as we have one FP more.



DRAWING ROC CURVES

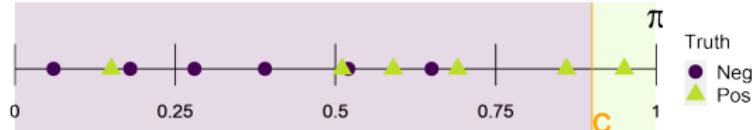
#	Truth	Score
1	Pos	0.95
2	Pos	0.86
3	Pos	0.69
4	Neg	0.65
5	Pos	0.59
6	Neg	0.52
7	Pos	0.51
8	Neg	0.39
9	Neg	0.28
10	Neg	0.18
11	Pos	0.15
12	Neg	0.06



$$c = 0.9$$

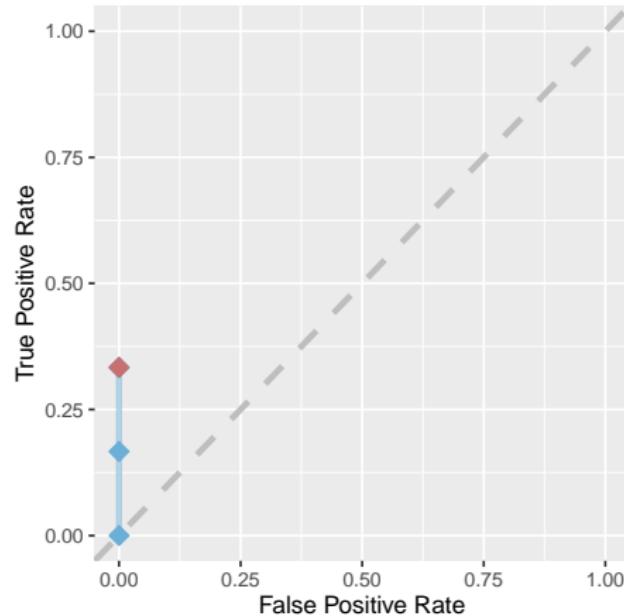
$$\rightarrow \text{TPR} = 0.167$$

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



DRAWING ROC CURVES

#	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

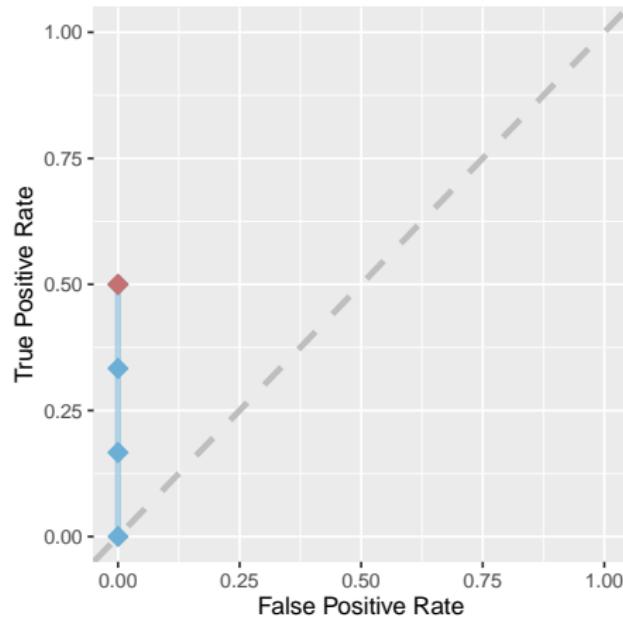


$$c = 0.85 \\ \rightarrow TPR = 0.333 \\ \rightarrow FPR = 0$$

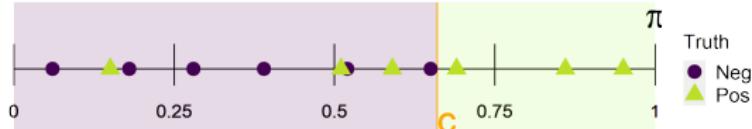


DRAWING ROC CURVES

#	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

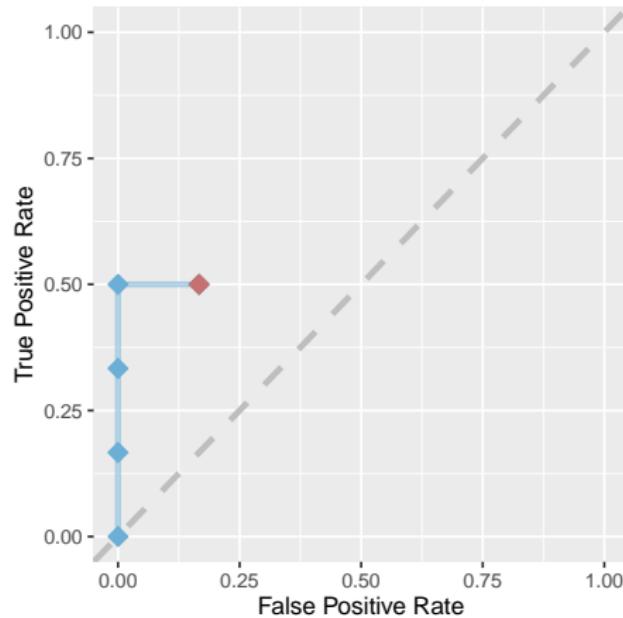


$$c = 0.66$$
$$\rightarrow \text{TPR} = 0.5$$
$$\rightarrow \text{FPR} = 0$$



DRAWING ROC CURVES

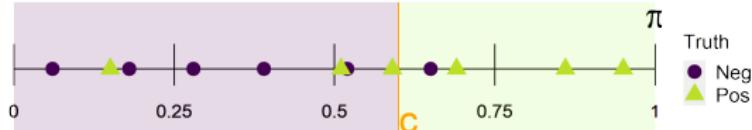
#	Truth	Score
1	Pos	0.95
2	Pos	0.86
3	Pos	0.69
4	Neg	0.65
5	Pos	0.59
6	Neg	0.52
7	Pos	0.51
8	Neg	0.39
9	Neg	0.28
10	Neg	0.18
11	Pos	0.15
12	Neg	0.06



$$c = 0.6$$

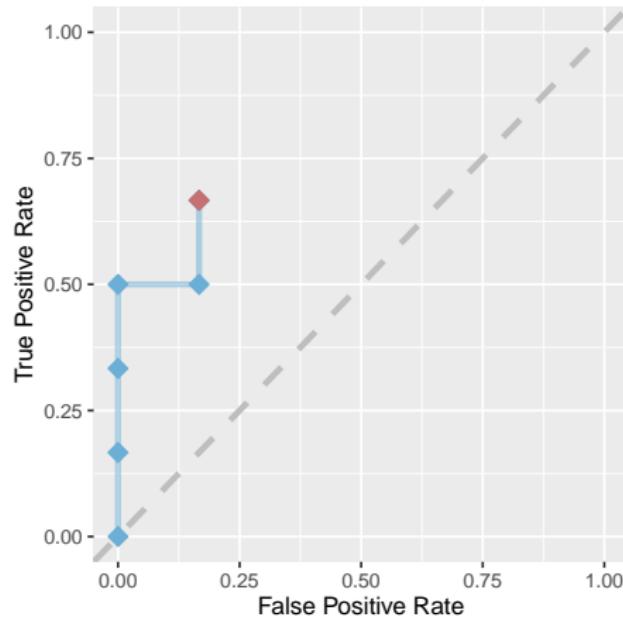
$$\rightarrow \text{TPR} = 0.5$$

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



DRAWING ROC CURVES

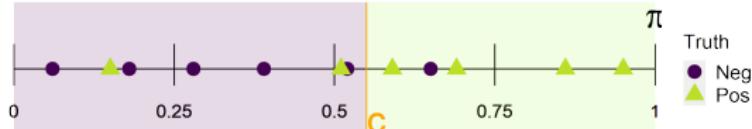
#	Truth	Score
1	Pos	0.95
2	Pos	0.86
3	Pos	0.69
4	Neg	0.65
5	Pos	0.59
6	Neg	0.52
7	Pos	0.51
8	Neg	0.39
9	Neg	0.28
10	Neg	0.18
11	Pos	0.15
12	Neg	0.06



$$c = 0.55$$

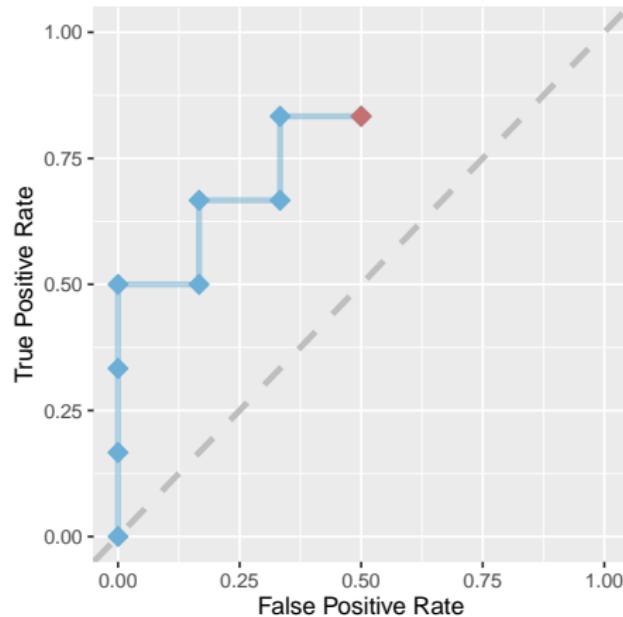
$$\rightarrow \text{TPR} = 0.667$$

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



DRAWING ROC CURVES

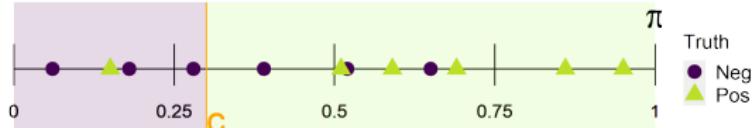
#	Truth	Score
1	Pos	0.95
2	Pos	0.86
3	Pos	0.69
4	Neg	0.65
5	Pos	0.59
6	Neg	0.52
7	Pos	0.51
8	Neg	0.39
9	Neg	0.28
10	Neg	0.18
11	Pos	0.15
12	Neg	0.06



$$c = 0.3$$

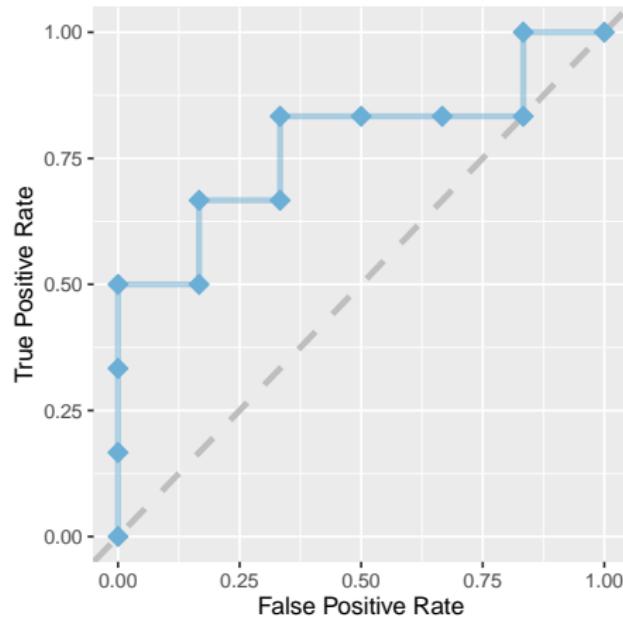
$$\rightarrow \text{TPR} = 0.833$$

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



DRAWING ROC CURVES

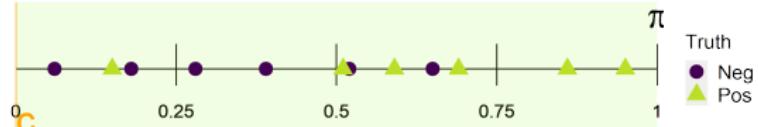
#	Truth	Score
1	Pos	0.95
2	Pos	0.86
3	Pos	0.69
4	Neg	0.65
5	Pos	0.59
6	Neg	0.52
7	Pos	0.51
8	Neg	0.39
9	Neg	0.28
10	Neg	0.18
11	Pos	0.15
12	Neg	0.06



$$c = 0$$

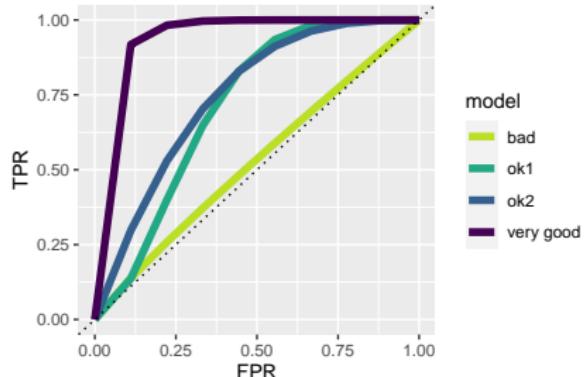
$$\rightarrow \text{TPR} = 1$$

$$\rightarrow \text{FPR} = 1$$



ROC CURVE PROPERTIES

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

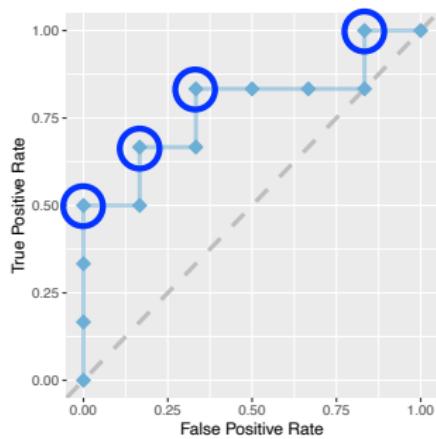


- Small thresholds will very liberally predict the positive class, and result in a potentially higher FPR, but also higher TPR.
- High thresholds will very conservatively predict the positive class, and result in a lower FPR and TPR.
- As we have not defined the trade-off between false positive and false negative costs, we cannot easily select the "best" threshold.
→ Visual inspection of all possible results seems useful.

CHOOSING THRESHOLD / OPERATING POINT

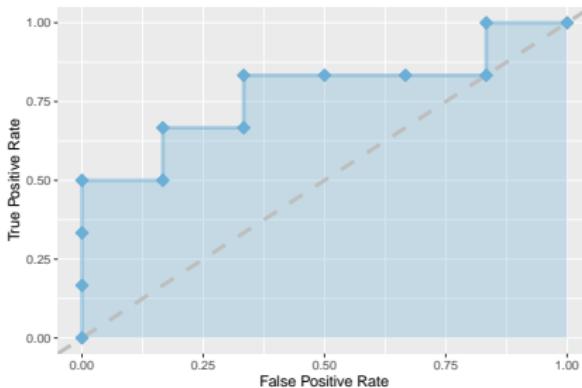
Often done visually and post-hoc, as class imbalances or costs are unknown a-priori.

- Identify non-dominated points
- Assess TPR / FPR
- Decide which combo is best for task
- Pick associated threshold



AUC: AREA UNDER ROC CURVE

- AUC $\in [0, 1]$ is a single metric to evaluate scoring classifiers – independent of the chosen threshold.
 - AUC = 1: perfect classifier
 - AUC = 0.5: random, non-discriminant classifier
 - AUC = 0: perfect, with inverted labels



AUC AS A RANK-BASED METRIC

- We can also interpret the AUC as the probability of our classifier ranking a random positive observation higher than a random negative one.
- A perfect classifier will rank all positive above all negative observations, achieving $AUC = 1$.

Truth	Score
1	0.9
1	0.76
1	0.7
0	0.5
1	0.45
0	0.3
0	0.1

Choose a random positive



1	0.76
---	------

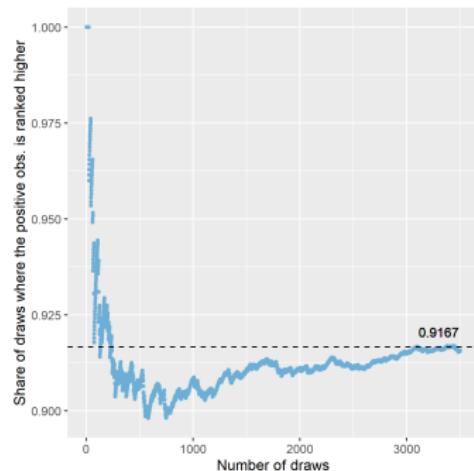
Choose a random negative



0	0.3
---	-----

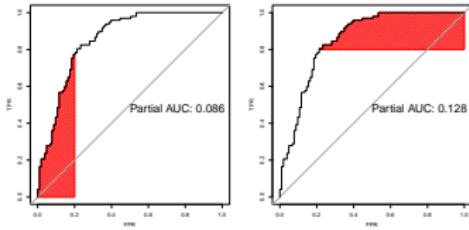
$$\text{AUC} = 0.9167$$

- Classifier ranks the positive higher than the negative
- This happens with a mean probability of 0.9167



Introduction to Machine Learning

Evaluation: Partial AUC



Learning goals

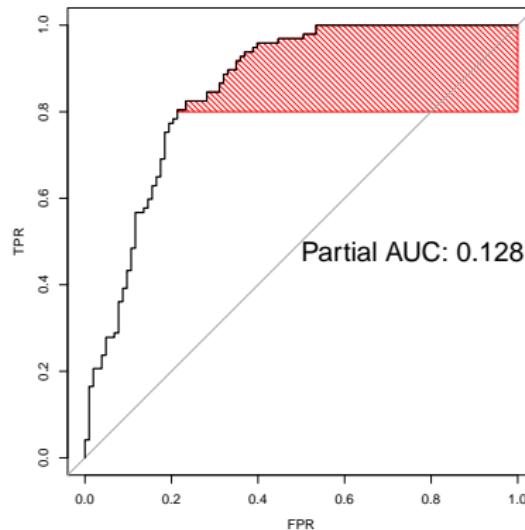
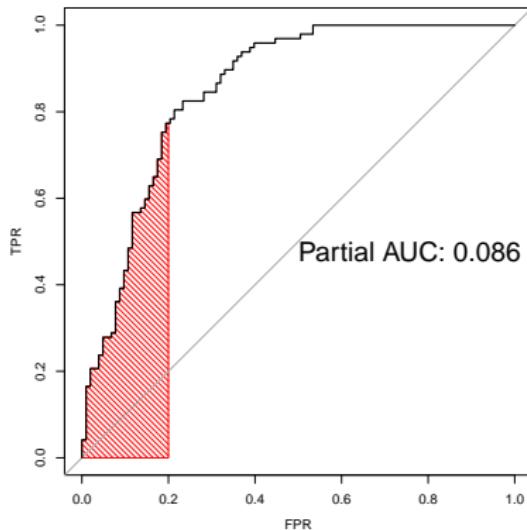
- Understand that entire AUC is not always relevant
- Learn about partial AUC

PARTIAL AUC

- TPR and FPR often treated asymmetrically in biomed contexts
- TPR = disease detection, is crucial
- But low FPR needed to avoid unnecessary treatments
- Common solution: Fix either TPR or FPR to a required value and optimize the other, but not easy to select exact point

PARTIAL AUC

- Can be useful to limit region under ROC curve
- E.g. $FPR > 0.2$ or $TPR < 0.8$ might not be acceptable for task, then we don't want to integrate over that region

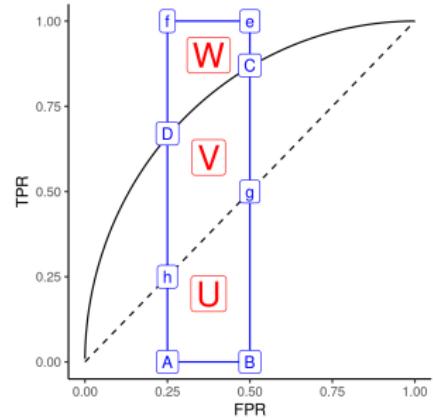


CORRECTED PARTIAL AUC

- Range of pAUC depends on cut-off values
- Normalize to [0, 1]:

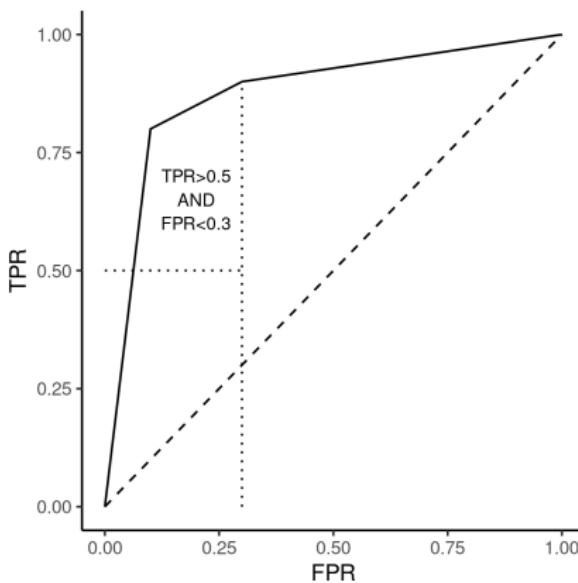
$$\text{pAUC}_{\text{corrected}} = \frac{1}{2} \left(1 + \frac{\text{pAUC} - \text{pAUC}_{\min}}{\text{pAUC}_{\max} - \text{pAUC}_{\min}} \right),$$

- pAUC is $V+U = "A-B-C-D"$
- pAUC_{\min} is pAUC of random classifier, so $U = "A-B-g-h"$
- pAUC_{\max} is $U+V+W = "A-B-e-f"$
- Compute percentage of V in $V+W$
- Rescale so random=0.5; optimal=1



2WAY PARTIAL AUC

- Can also limit both TPR and FPR
- 2way pAUC = compute area under 2way limited segment



Introduction to Machine Learning

Evaluation: Multi-Class AUC

AUC(pos neg) = AUC(1 3)				
	$\hat{\pi}_1$	$\hat{\pi}_2$	$\hat{\pi}_3$	
Y				
pos	1	0.7	0.2	0.1
pos	1	0.5	0.3	0.2
	2	0.3	0.5	0.2
	2	0.4	0.5	0.1
neg	3	0.6	0.1	0.3
neg	3	0.1	0.1	0.8

AUC(pos neg) = AUC(3 1)				
	$\hat{\pi}_1$	$\hat{\pi}_2$	$\hat{\pi}_3$	
Y				
neg	1	0.7	0.2	0.1
neg	1	0.5	0.3	0.2
	2	0.3	0.5	0.2
	2	0.4	0.5	0.1
pos	3	0.6	0.1	0.3
pos	3	0.1	0.1	0.8

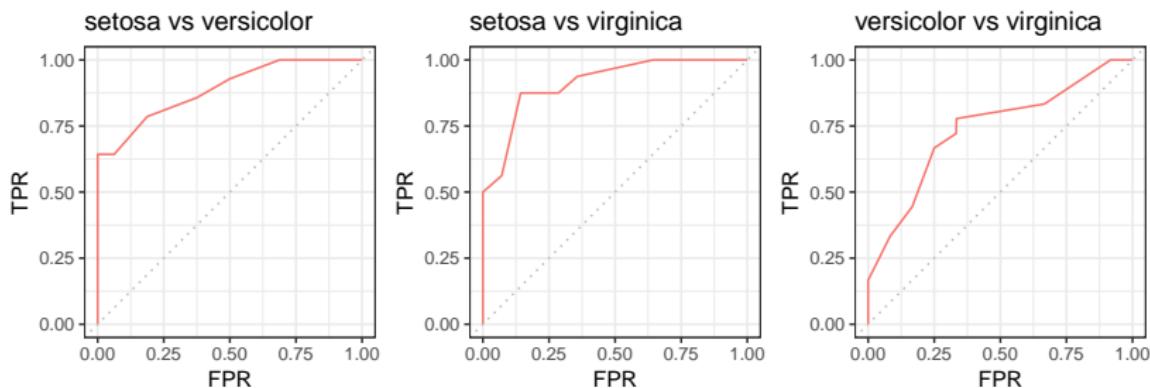
Learning goals

- Understand that generalizing AUC to multi-class is not trivial
- Learn how multi-class AUC can be derived

MULTI-CLASS AUC

- AUC and other ROC metrics for binary classification
- Different ways to estimate **multi-class AUC**
- Often based on aggregated binary AUCs:
e.g. 1-vs-1 or 1-vs-rest

Example: 1-vs-1 on iris



MULTI-CLASS AUC

- Def $AUC(k | \ell)$ for classes k (pos) and ℓ (neg)
- Compute AUC: Subset preds to rows of true k and ℓ , use $\hat{\pi}_k$
- Interpret: Prob that random member of ℓ has a lower prob to belong to class k than random member of class k .

Example: $AUC(3|1)$ with $g = 3$ classes

AUC(pos neg) = AUC(3 1)				
	Y	$\hat{\pi}_1$	$\hat{\pi}_2$	$\hat{\pi}_3$
neg	1	0.7	0.2	0.1
neg	1	0.5	0.3	0.2
	2	0.3	0.5	0.2
	2	0.4	0.5	0.1
pos	3	0.6	0.1	0.3
pos	3	0.1	0.1	0.8

- ➊ Subset pred rows to true classes 1 and 3
- ➋ Use $k = 3$ as pos and $\ell = 1$ as neg class
- ➌ Compute standard AUC with $\hat{\pi}_3$ as scores
- ➍ $AUC(3|1) = 1$:
all pos have higher $\hat{\pi}_3$ than negs

MULTI-CLASS AUC

- For binary classes: always $\text{AUC}(1|0) = \text{AUC}(0|1)$
- For multi-class usually: $\text{AUC}(k | \ell) \neq \text{AUC}(\ell | k)$
- **Example** with $g = 3$ where $\text{AUC}(1|3) \neq \text{AUC}(3|1)$:
 - $\text{AUC}(3|1) = 1$ (RHS) as before
 - $\text{AUC}(1|3) \neq 1$ (LHS)

AUC(pos neg) = AUC(1 3)				
	Y	$\hat{\pi}_1$	$\hat{\pi}_2$	$\hat{\pi}_3$
pos	1	0.7	0.2	0.1
pos	1	0.5	0.3	0.2
	2	0.3	0.5	0.2
	2	0.4	0.5	0.1
neg	3	0.6	0.1	0.3
neg	3	0.1	0.1	0.8

AUC(pos neg) = AUC(3 1)				
	Y	$\hat{\pi}_1$	$\hat{\pi}_2$	$\hat{\pi}_3$
neg	1	0.7	0.2	0.1
neg	1	0.5	0.3	0.2
	2	0.3	0.5	0.2
	2	0.4	0.5	0.1
pos	3	0.6	0.1	0.3
pos	3	0.1	0.1	0.8

MULTI-CLASS AUC

Hand and Till (2001) proposed to avg AUC via **1-vs-1**:

- For all class pairs, compute $\text{AUC}(k | \ell)$.

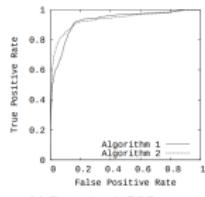
$$\text{AUC}_{MC} = \frac{1}{g(g-1)} \sum_{k \neq \ell} \text{AUC}(k|\ell) \in [0, 1].$$

Comments:

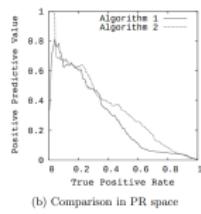
- Other defs use **1-vs-rest** and need to avg only g AUC values
- 1-vs-rest creates imbal classes even if orig classes are balanced
- Imbalanced classes can be considered by weighting individual AUC values with class priors [Ferri et al. (2003)]

Introduction to Machine Learning

Evaluation: Precision-Recall Curves



(a) Comparison in ROC space



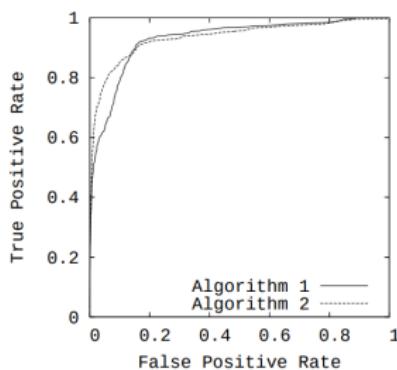
(b) Comparison in PR space

Learning goals

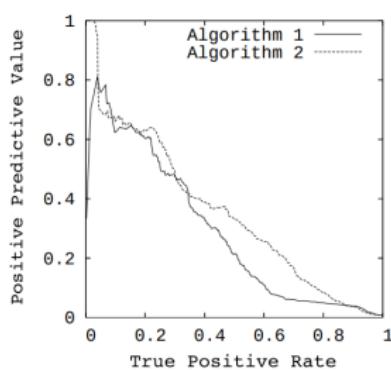
- Understand PR curves
- Same as PPV-TPR curve
- Compare to standard TPR-FPR ROC curve

PRECISION-RECALL CURVES

- Slightly changed ROC plot
- Simply plot precision and recall, instead of TPR-FPR
- Precision = $\rho_{PPV} = \frac{TP}{TP+FP}$, recall = $\rho_{TPR} = \frac{TP}{TP+FN}$
- Might call them TPR-PPV curve
- NB: Both metrics don't depend on TNs



(a) Comparison in ROC space

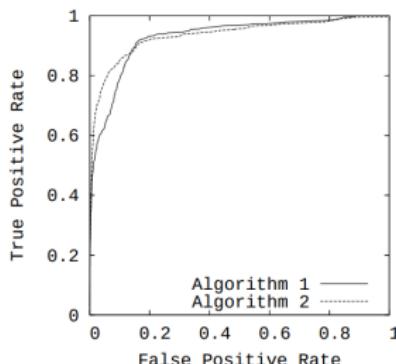


(b) Comparison in PR space

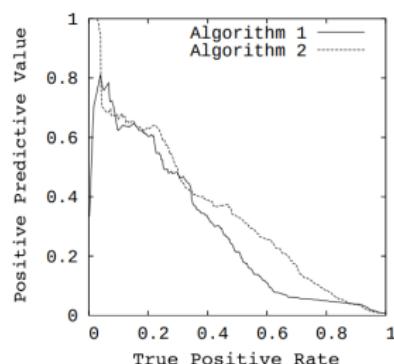
Davis and Goadrich (2006): The Relationship Between Precision-Recall and ROC Curves ([URL](#)).

PRECISION-RECALL CURVES

- Might be better for highly imbal data ($n_- \gg n_+$) than TPR-FPR
- Figure (a): ROC; both learners seem to perform well
- Figure (b): PR; visible room for improvement (top-right=best)
- PR reveals better that algo 2 has advantage over 1



(a) Comparison in ROC space



(b) Comparison in PR space

Davis and Goadrich (2006): The Relationship Between Precision-Recall and ROC Curves ([URL](#)).

IMBALANCED DATA

- Assume imbalanced classes with $n_- \gg n_+$
- If neg class large, typically less interested in high TNR = low FPR, but more in PPV
- Large (abs) change in FP yields small change in FPR
- PPV likely more informative

FP=10:

	True +1	True -1
Pred. Pos	100	10
Pred. Neg	10	9990
Total	110	10000

$$TPR = 10/11$$

$$FPR = 0.001$$

$$PPV = 10/11$$

FP=100:

	True +1	True -1
Pred. +1	100	100
Pred. -1	10	9900
Total	110	10000

$$TPR = 10/11$$

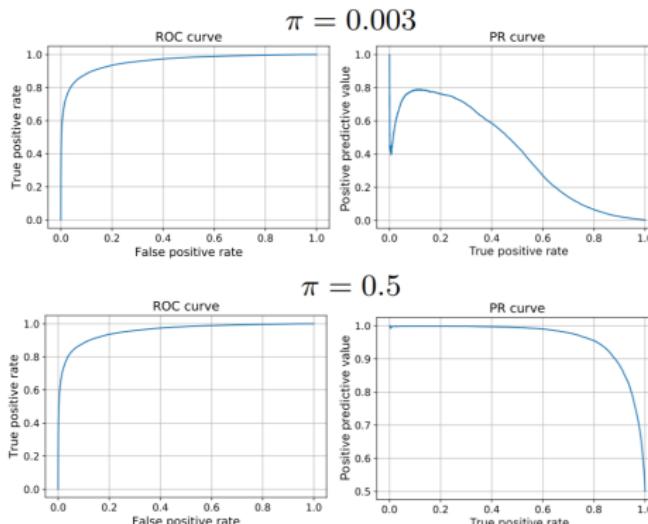
$$FPR = 0.01$$

$$PPV = 1/2$$

RHS: Given test says +1, it's now a coin flip that this is correct.

IMBALANCED DATA

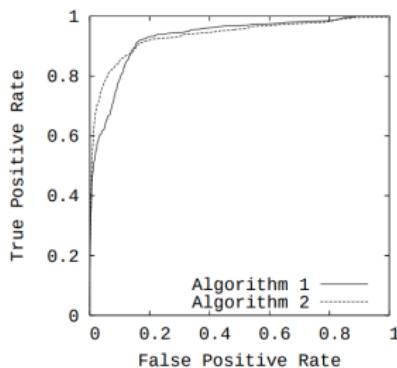
- Top row: Imbal classes with $\pi = 0.003$
- Bottom: balanced with $\pi = 0.5$
- ROC curves (LHS) are similar
- PR curve (RHS) changes strongly from imbal to bal classes



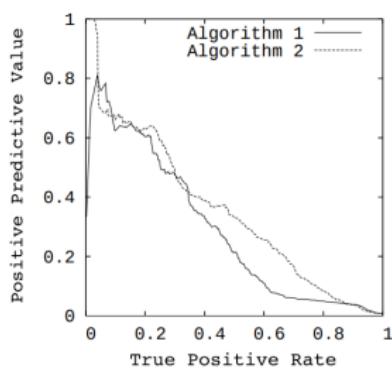
Wissam Siblini et. al. (2004): Master your Metrics with Calibration ([URL](#)).

CONCLUSIONS

- Curve fully dominates in ROC space iff dominates in PR-space
- In imbalanced situations rather use PR than standard TPR-FPR
- If comparing few models on a single task, probably plot both. Then observe and think.
- For tuning: can also use PR-AUC (or partial versions)



(a) Comparison in ROC space

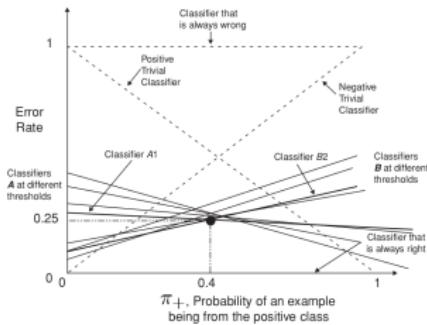


(b) Comparison in PR space

Davis and Goadrich (2006): The Relationship Between Precision-Recall and ROC Curves ([URL](#)).

Introduction to Machine Learning

Evaluation: Cost Curves



Learning goals

- Understand cost curves
- As alternative to ROC curves

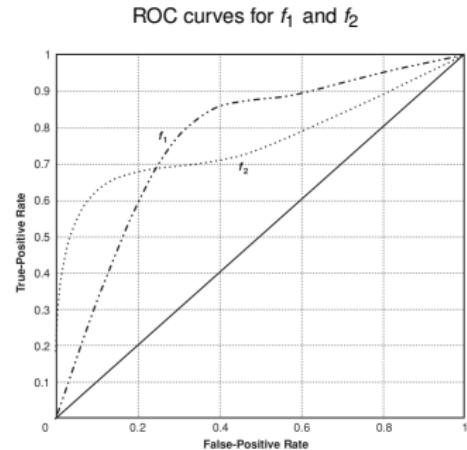
COST CURVES

- Directly plot the misclassif costs / error
- Might be easier to interpret than ROC, especially in case of different misclassif costs or priors

Example:

- f_1 and f_2 with intersecting ROC curves
- f_2 dominates first, then f_1

BUT: Unclear for which thresholds, costs or class distribs f_2 better than f_1



Nathalie Japkowicz (2004): Evaluating Learning Algorithms : A Classification Perspective. (p. 125)

COST CURVES

With law total probab, can write misclassif rate as function of π_+ :

$$\begin{aligned}\rho_{MCE}(\pi_+) &= (1 - \pi_+) \cdot \mathbb{P}(\hat{y} = 1 | y = 0) + \pi_+ \cdot \mathbb{P}(\hat{y} = 0 | y = 1) \\ &= (1 - \pi_+) \cdot FPR + \pi_+ \cdot FNR \\ &= (FNR - FPR) \cdot \pi_+ + FPR\end{aligned}$$

Can do the same for costs:

$$Costs(\pi_+) = (1 - \pi_+) \cdot FPR \cdot cost_{FP} + \pi_+ \cdot FNR \cdot cost_{FN}$$

$$Costs_{norm}(\pi_+) = \frac{(1 - \pi_+) \cdot FPR \cdot cost_{FP} + \pi_+ \cdot FNR \cdot cost_{FN}}{(1 - \pi_+) \cdot cost_{FP} + \pi_+ \cdot cost_{FN}} \in [0, 1]$$

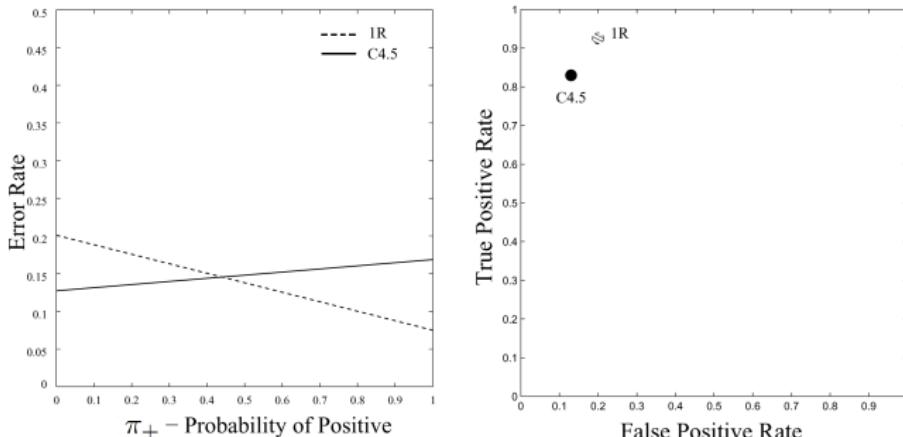
- Denominator = $\max(Costs)$ = all obs misclassified (i.e., $FPR = FNR = 1$).
- If $cost_{FN} = cost_{FP}$, then $Costs_{norm} = \rho_{MCE}$

		Confusion matrix	
		True class	
		$y = 1$	$y = 0$
Pred. class	$\hat{y} = 1$	TP	FP
	$\hat{y} = 0$	FN	TN

		Cost matrix	
		True class	
		$y = 1$	$y = 0$
Pred. class	$\hat{y} = 1$	0	$cost_{FP}$
	$\hat{y} = 0$	$cost_{FN}$	0

COST CURVES

- Simplifying assumption: equal misclassification costs, i.e., $\text{cost}_{FN} = \text{cost}_{FP}$.
- Normalized costs (or error rate in the case of $\text{cost}_{FN} = \text{cost}_{FP}$) is plotted as a function of the proportion of positive instances, $\pi_+ = \mathbb{P}(y = 1)$.
- Cost curves are point–line duals of ROC curves, i.e., a single classifier is represented by a point in the ROC space and by a line in the cost space.



Chris Drummond and Robert C. Holte (2006): Cost curves: An improved method for visualizing classifier performance.
Machine Learning, 65, 95-130 ([URL](#)).

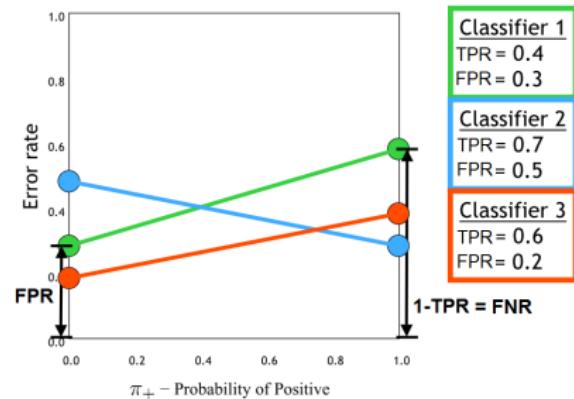
COST CURVES - EXAMPLE

Cost curve of a classifier with slope ($FNR - FPR$) and intercept FPR :

$$\rho_{MCE}(\pi_+) = (FNR - FPR) \cdot \pi_+ + FPR$$

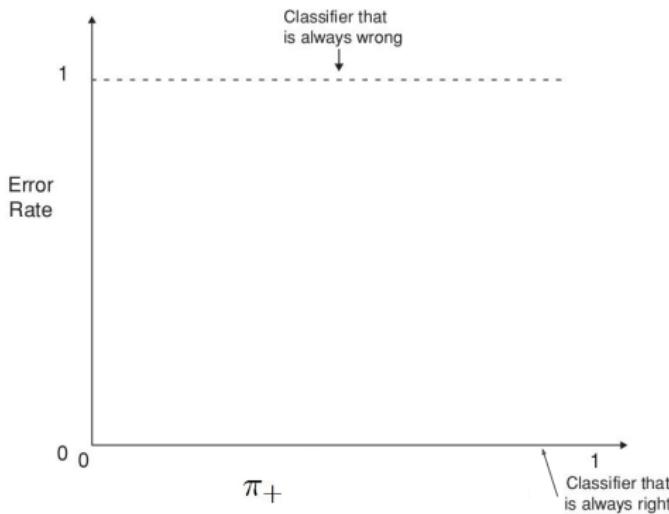
- Hard classifiers are points (TPR, FPR) in ROC space
- The cost curve of a classifier connects (π_+, ρ_{MCE}) -points at $(0, FPR)$ and $(1, 1 - TPR)$
- Classifier 3 always dominates classifier 1
- Classifier 3 is better than classifier 2 when $\pi_+ < 0.7$

Cost curves plot different values of π_+ vs. $\rho_{MCE}(\pi_+)$



COST CURVES - EXAMPLE

- Horizontal dashed line: worst classifier (100% error rate for all π_+).
 $\Rightarrow FNR = FPR = 1$
- x-axis: perfect classifier (0% error rate for all π_+). $\Rightarrow FNR = FPR = 0$



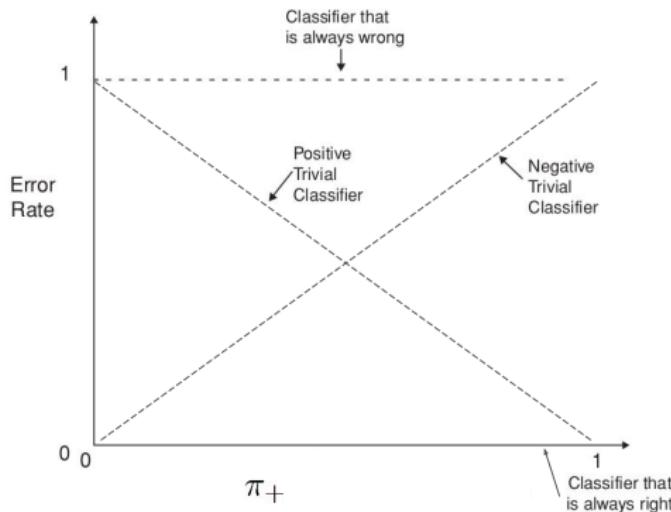
$$\rho_{MCE} = (FNR - FPR) \cdot \pi_+ + FPR$$

Confusion matrix

		True class	
		$y = 1$	$y = 0$
Pred. class	$\hat{y} = 1$	TP	FP
	$\hat{y} = 0$	FN	TN

COST CURVES - EXAMPLE

- Horizontal dashed line: worst classifier (100% error rate for all π_+).
 $\Rightarrow FNR = FPR = 1$
- x-axis: perfect classifier (0% error rate for all π_+). $\Rightarrow FNR = FPR = 0$
- Dashed diagonal lines: trivial classifiers, i.e., ascending diagonal always predicts negative instances ($\Rightarrow FNR = 1$ and $FPR = 0$) and vice versa.

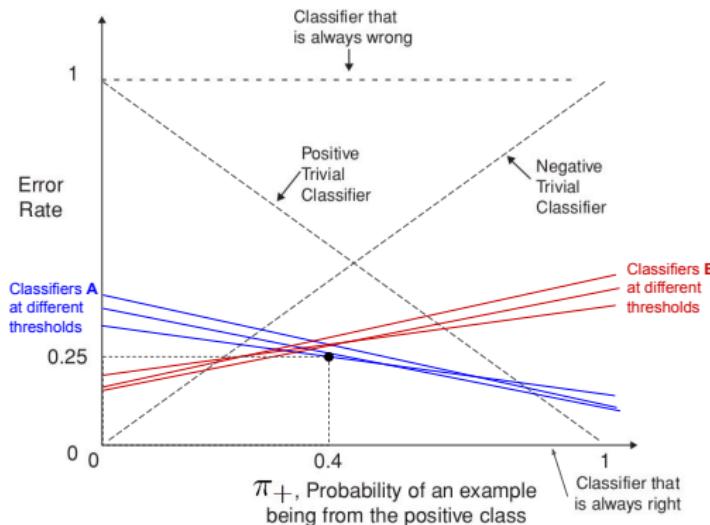


$$\rho_{MCE} = (FNR - FPR) \cdot \pi_+ + FPR$$

		Confusion matrix	
		True class	
		$y = 1$	$y = 0$
Pred.	$\hat{y} = 1$	TP	FP
class	$\hat{y} = 0$	FN	TN

COST CURVES - EXAMPLE

- Horizontal dashed line: worst classifier (100% error rate for all π_+).
 $\Rightarrow FNR = FPR = 1$
- x-axis: perfect classifier (0% error rate for all π_+).
 $\Rightarrow FNR = FPR = 0$
- Dashed diagonal lines: trivial classifiers, i.e., ascending diagonal always predicts negative instances ($\Rightarrow FNR = 1$ and $FPR = 0$) and vice versa.
- Descending/ascending bold lines: two families of classifiers A and B (represented by points in their respective ROC curves).



$$\rho_{MCE} = (FNR - FPR) \cdot \pi_+ + FPR$$

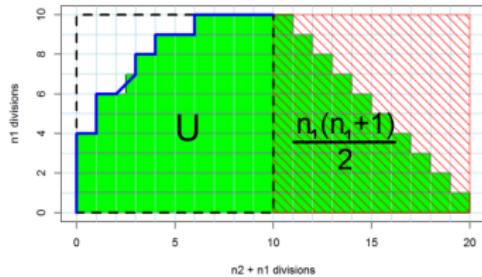
		Confusion matrix	
		True class	
		$y = 1$	$y = 0$
Pred.	class	$\hat{y} = 1$	$\hat{y} = 0$
Classifiers A	$\hat{y} = 1$	TP	FP
at different	$\hat{y} = 0$	FN	TN
thresholds			

ROC CURVES VS. COST CURVES

- ROC curves do not indicate in which situations classifier A is superior to another classifier B.
- Cost curves can do exactly that and therefore provide practically more relevant information than ROC curves.
- For simplification, we focused on cost curves based on the misclassification error by assuming $cost_{FN} = cost_{FP}$. However, cost curves can also be defined for different misclassification costs.

Introduction to Machine Learning

Evaluation: AUC & Mann-Whitney-U Test

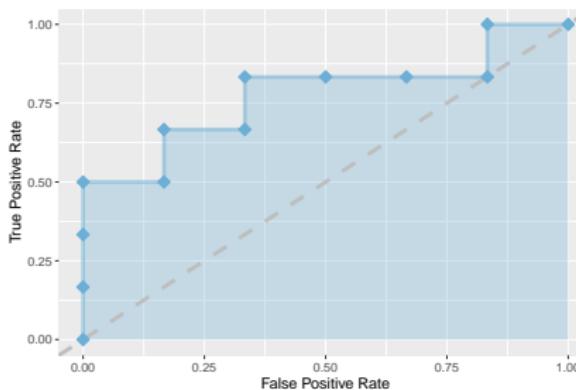


Learning goals

- Understand the rank-based nature of AUC
- See the connection between AUC and Mann-Whitney-U statistic

AUC AS A RANK-BASED METRIC

- The AUC metric is intimately related to the **Mann-Whitney-U test**, also known as **Wilcoxon rank-sum test**.
- This connection is best understood viewing the AUC from a slightly different angle: it is, in effect, a **rank-based** metric.
- Recall that, constructing the ROC curve, we count TP and FP.



- The AUC abstracts from the actual classification scores and considers only their rank.

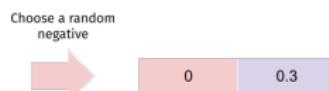
AUC AS A RANK-BASED METRIC

- We can interpret the AUC as the probability of our classifier ranking a random positive observation higher than a random negative one.
- A perfect classifier will rank all positive above all negative observations, achieving $AUC = 1$.

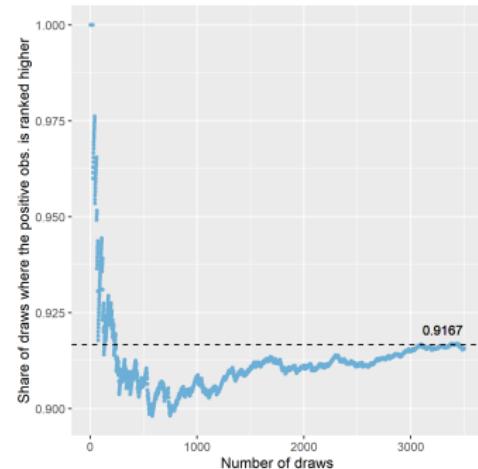
Truth	Score
1	0.9
1	0.76
1	0.7
0	0.5
1	0.45
0	0.3
0	0.1

{ }

$$AUC = 0.9167$$



- Classifier ranks the positive higher than the negative
- This happens with a mean probability of 0.9167



MANN-WHITNEY-U TEST

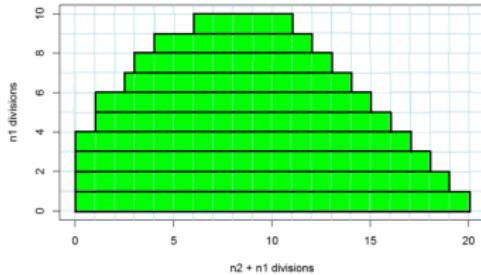
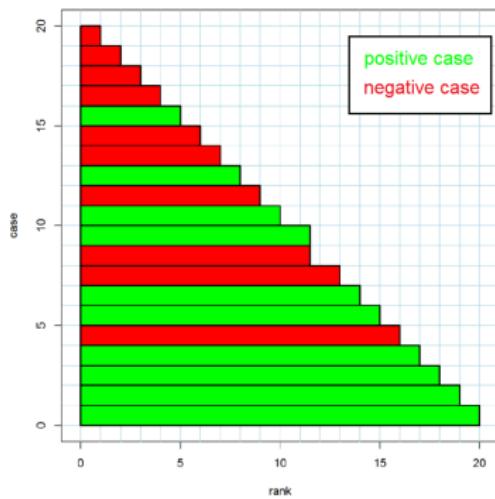
- The Mann-Whitney-U test is a **non-parametric hypothesis test** on the difference in location between two samples X_1, X_2 of sizes n_1 and n_2 , respectively.
- Under the null, X_1 and X_2 follow the same (unknown) distribution \mathbb{P} , and for any pair of observations $x_{1,1} \in X_1, x_{2,1} \in X_2$ drawn at random from \mathbb{P} , the following statement holds: $\mathbb{P}(x_{1,1} \in X_1) > \mathbb{P}(x_{2,1} \in X_2) = \mathbb{P}(x_{1,1} \in X_1) < \mathbb{P}(x_{2,1} \in X_2) = 0.5$.
- The test statistic estimates the probability of a random sample from X_1 ranking higher than one from X_2 (R_1 denoting the sum of ranks of the $x_{1,i}$):

$$U = \frac{1}{n_1 n_2} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \mathbb{I}[x_{1,i} > x_{2,j}] = R_1 - \frac{n_1(n_1 + 1)}{2}$$

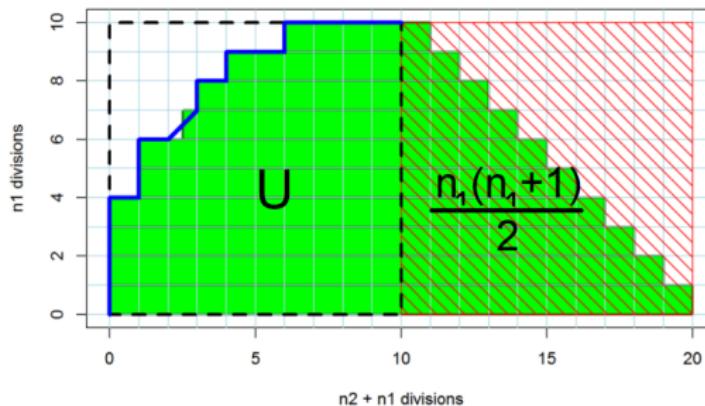
- For large samples, U is approximately normally distributed.

AUC & MANN-WHITNEY-U TEST

- We can directly interpret the AUC in the light of the U statistic.
- In order to see this, plot the ranks of all the scores as a stack of horizontal bars, and color them by label.
- Next, keep only the green ones, and slide them horizontally to get a nice even stairstep on the right edge:



AUC & MANN-WHITNEY-U TEST

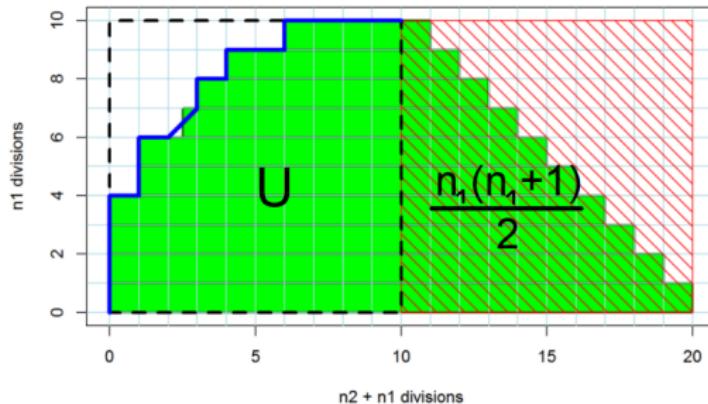


- Denoting the respective numbers of cases as n_+ and n_- , we have:

$$U = R_+ - \frac{n_+(n_+ + 1)}{2}.$$

- The area of the green bars on the right is equal to $\frac{n_+(n_+ + 1)}{2}$.

AUC & MANN-WHITNEY-U TEST



- U : area of the green bars on the left.
- $n_+ \cdot n_-$: area of the dashed rectangle.

⇒ AUC is U normalized to the unit square:

$$\text{AUC} = \frac{U}{n_+ \cdot n_-}.$$

INTRODUCTION TO MACHINE LEARNING

ML Basics

Supervised Regression

Supervised Classification

Performance Evaluation

k-NN

Classification and Regression Trees (CART)

Random Forests

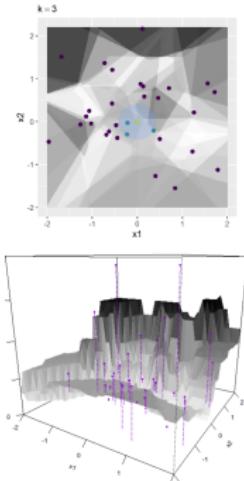
Tuning

Nested Resampling

mlr3

Introduction to Machine Learning

k-Nearest Neighbors

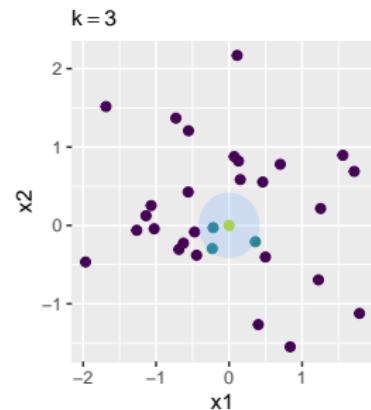
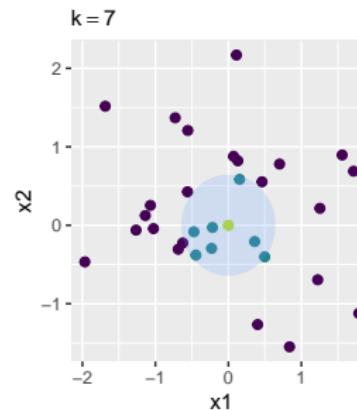
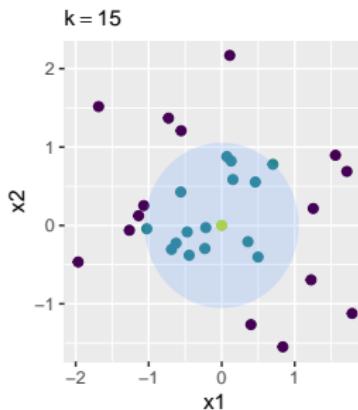


Learning goals

- 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} to 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

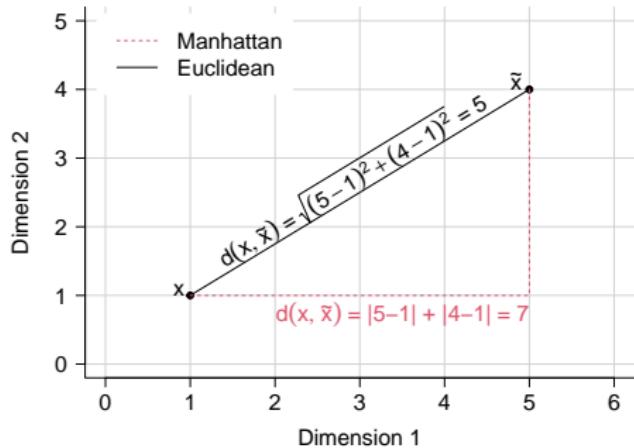
- Popular for numerical features: **Minkowski** distances of the form

$$\|\mathbf{x} - \tilde{\mathbf{x}}\|_q = \left(\sum_{j=1}^p |x_j - \tilde{x}_j|^q \right)^{\frac{1}{q}} \text{ for } \mathbf{x}, \tilde{\mathbf{x}} \in \mathcal{X} \text{ with } p \text{ numeric features}$$

- Especially, **Manhattan** ($q = 1$) and **Euclidean** ($q = 2$) distance

$$d_{\text{Manhattan}}(\mathbf{x}, \tilde{\mathbf{x}}) = \|\mathbf{x} - \tilde{\mathbf{x}}\|_1 \\ = \sum_{j=1}^p |x_j - \tilde{x}_j|$$

$$d_{\text{Euclidean}}(\mathbf{x}, \tilde{\mathbf{x}}) = \|\mathbf{x} - \tilde{\mathbf{x}}\|_2 \\ = \sqrt{\sum_{j=1}^p (x_j - \tilde{x}_j)^2}$$

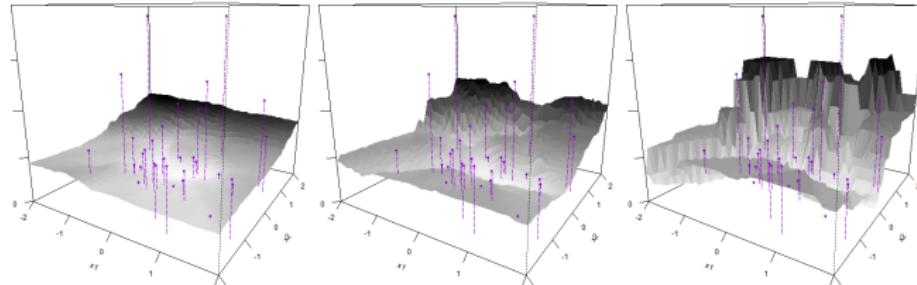
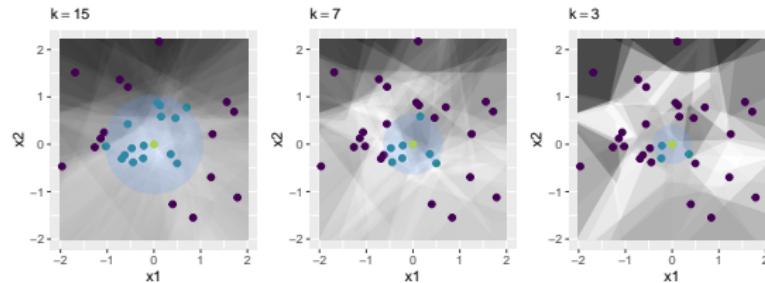


PREDICTION - REGRESSION

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} : $w^{(i)} = \frac{1}{d(\mathbf{x}^{(i)}, \mathbf{x})}$



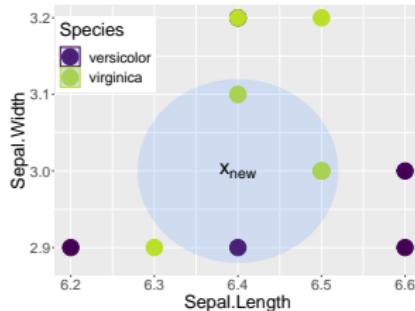
PREDICTION - CLASSIFICATION

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:

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

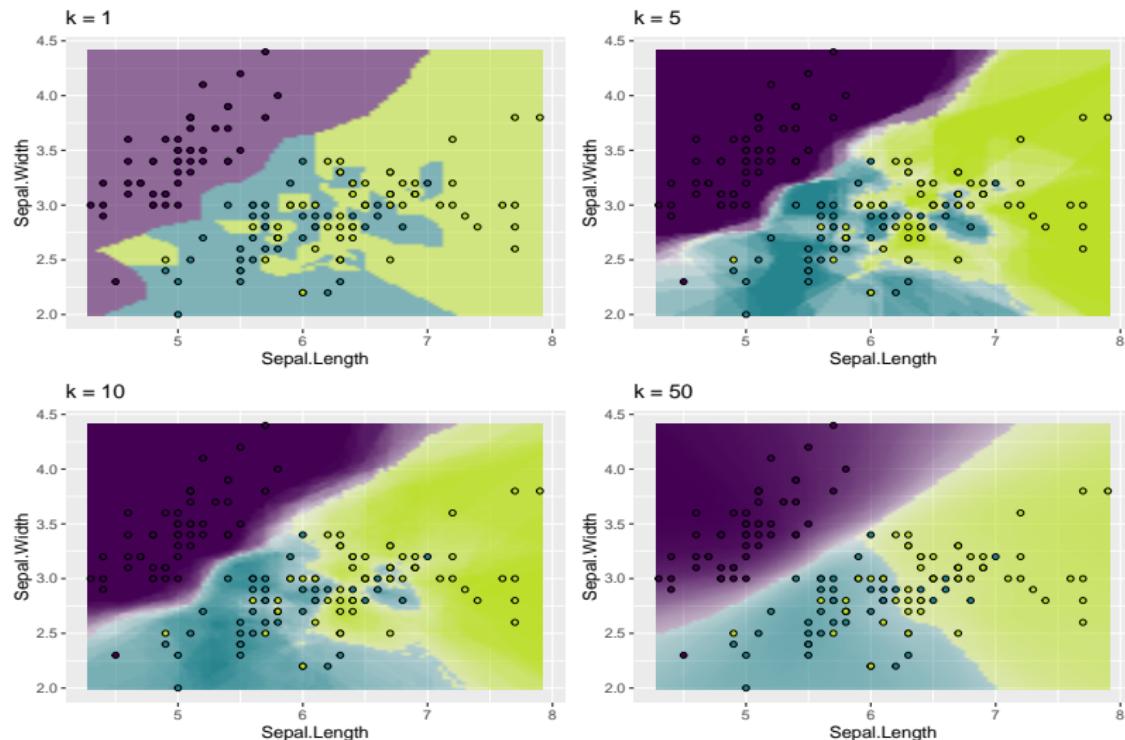


	SL	SW	Species	dist
52	6.4	3.2	versicolor	0.200
59	6.6	2.9	versicolor	0.224
75	6.4	2.9	versicolor	0.100
76	6.6	3.0	versicolor	0.200
98	6.2	2.9	versicolor	0.224
104	6.3	2.9	virginica	0.141
105	6.5	3.0	virginica	0.100
111	6.5	3.2	virginica	0.224
116	6.4	3.2	virginica	0.200
117	6.5	3.0	virginica	0.100
138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

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

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

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

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

INTRODUCTION TO MACHINE LEARNING

ML Basics

Supervised Regression

Supervised Classification

Performance Evaluation

k-NN

Classification and Regression Trees (CART)

Random Forests

Tuning

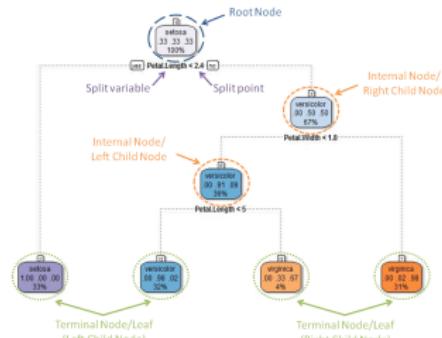
Nested Resampling

mlr3

Introduction to Machine Learning

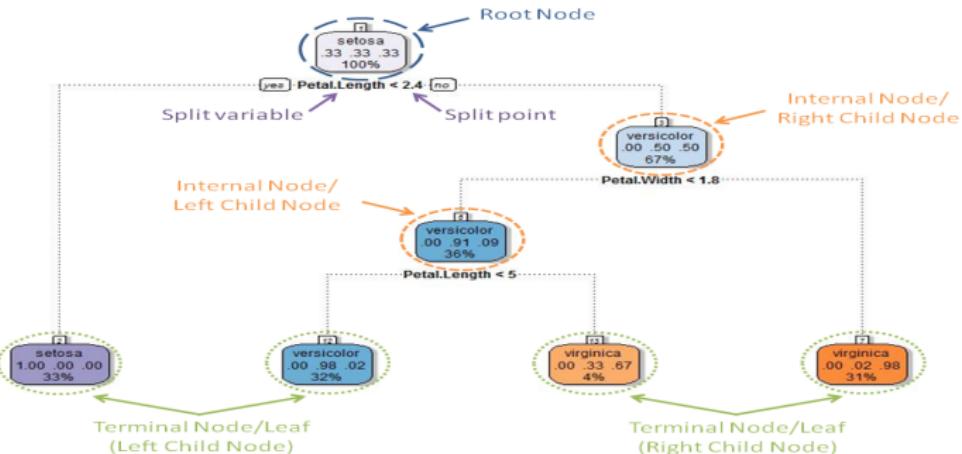
Classification and Regression Trees (CART): Basics

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

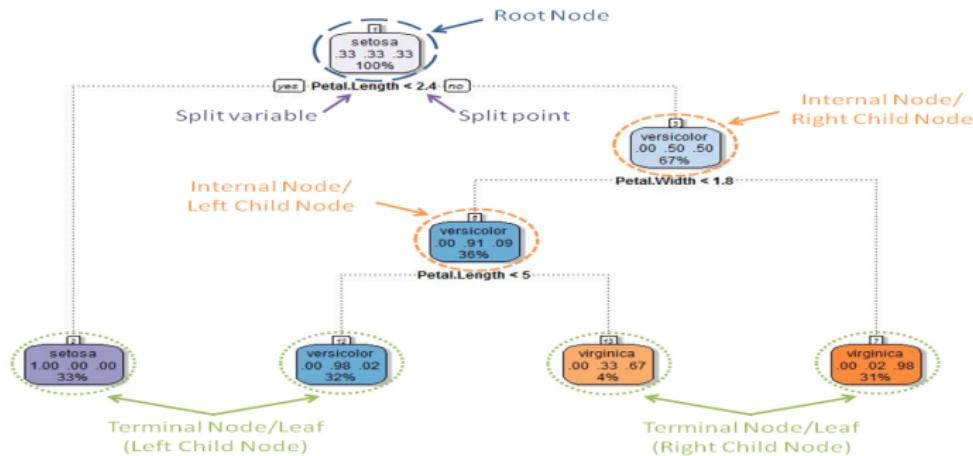
TREE MODEL AND PREDICTION



- 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.

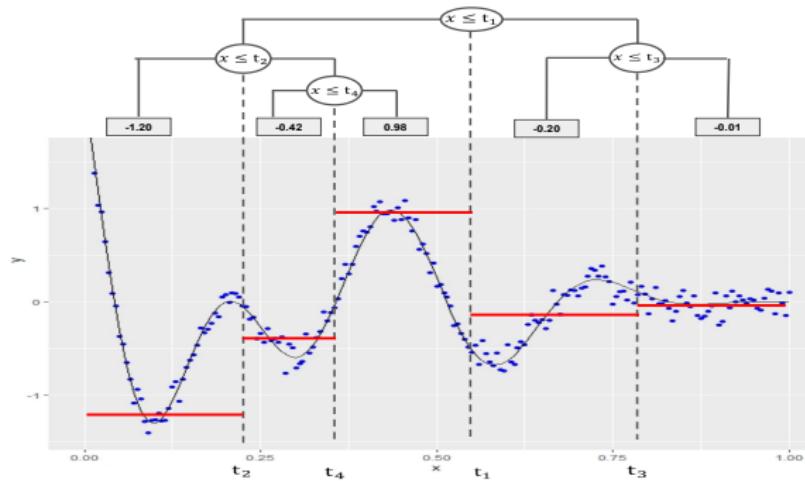
TREE MODEL AND PREDICTION

- 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



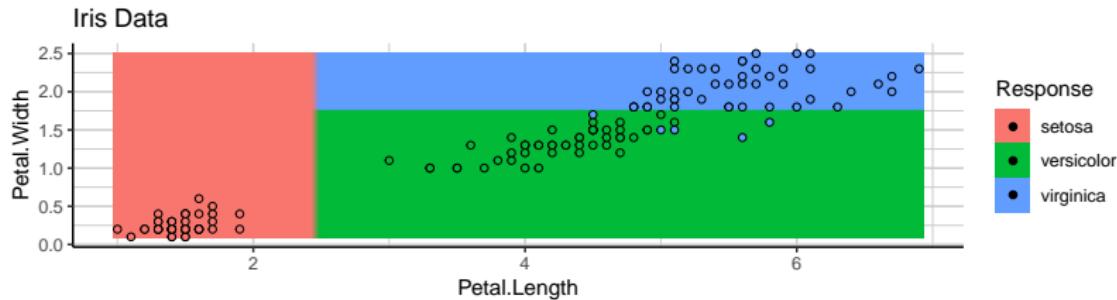
TREE AS AN ADDITIVE MODEL

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.

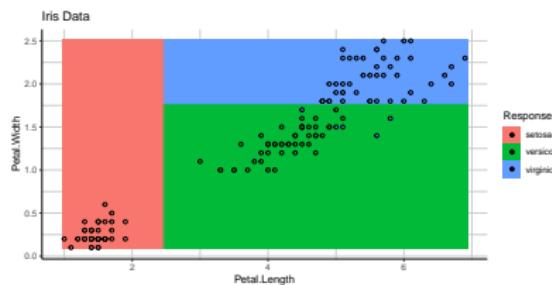


TREES

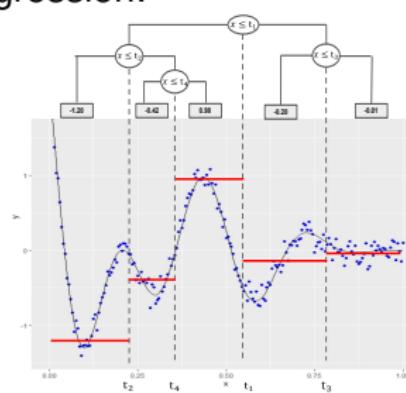
The hypothesis space of a CART is the set of all step functions over rectangular partitions of \mathcal{X} :

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

Classification:



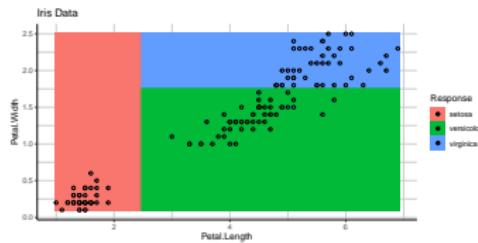
Regression:



Introduction to Machine Learning

CART: Splitting Criteria

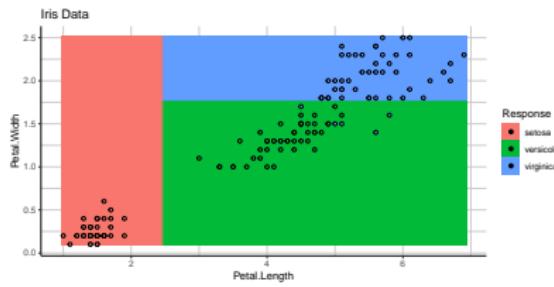
Learning goals



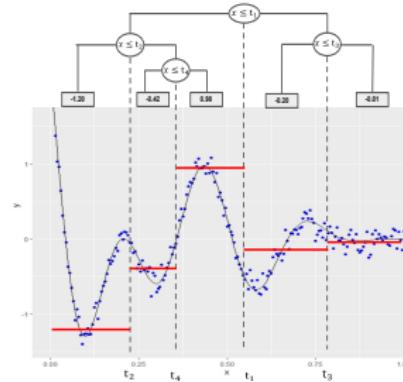
- Know definition of a tree's node
- Understand how split points are derived via empirical risk minimization
- Know which losses can be applied for regression and classification
- Know the connections between empirical risk minimization and impurity minimization

TREES

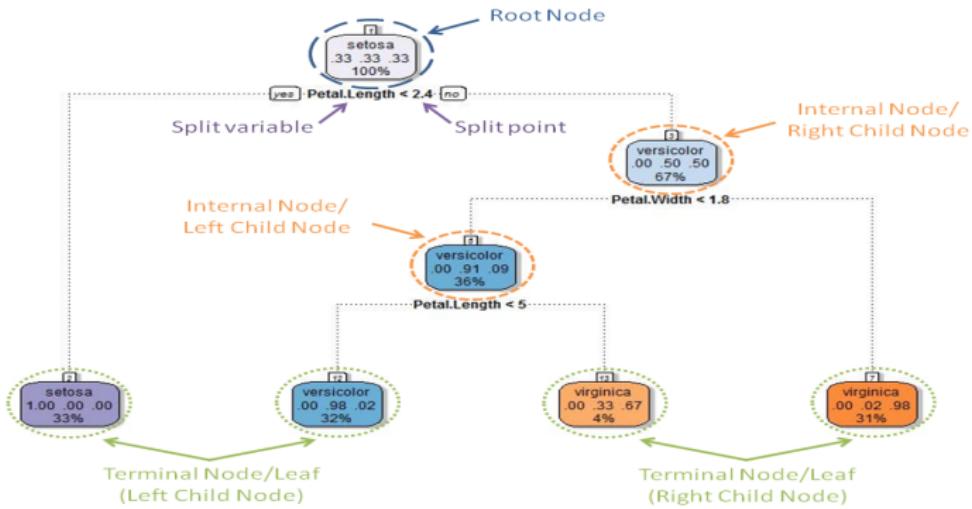
Classification Tree:



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:

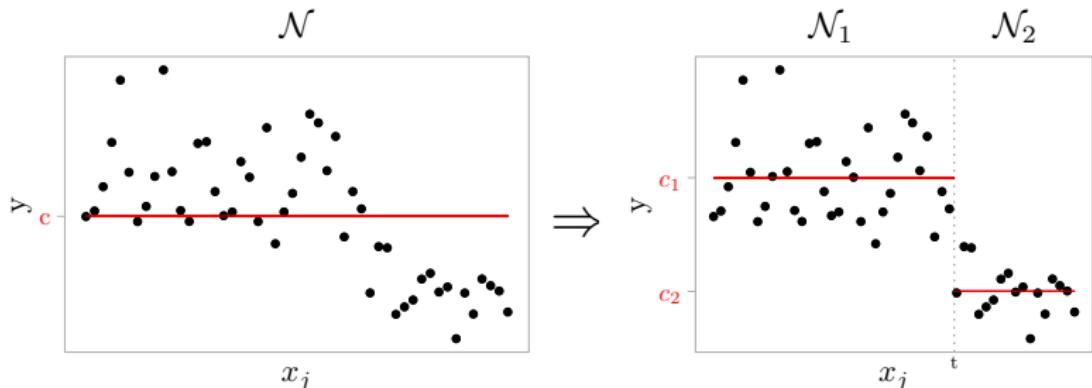
$$\mathcal{R}(\mathcal{N}) = \frac{1}{|\mathcal{N}|} \sum_{(\mathbf{x}, y) \in \mathcal{N}} (y - c)^2$$

- 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.

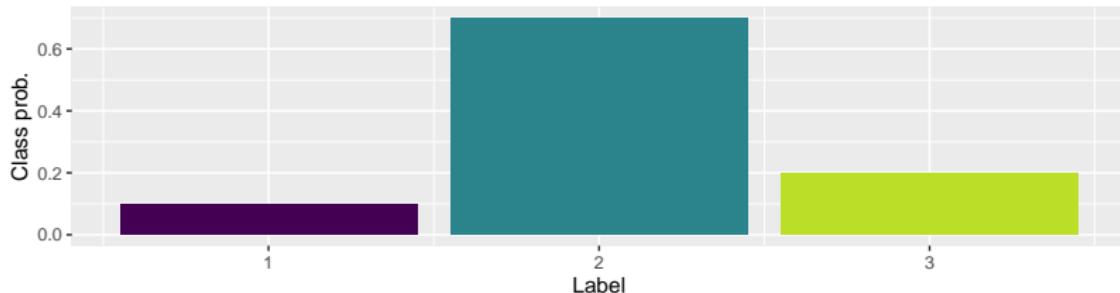
- For L_1 loss, c is the median of $y \in \mathcal{N}$.

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:

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

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

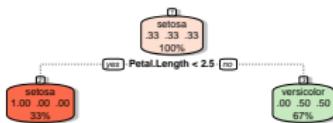
$$\text{Split 2 : } \frac{3}{4} \cdot 2 \cdot \frac{2}{3} \cdot \frac{1}{3} + \frac{1}{4} \cdot 2 \cdot 0 \cdot 1 = \frac{1}{3}$$

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.

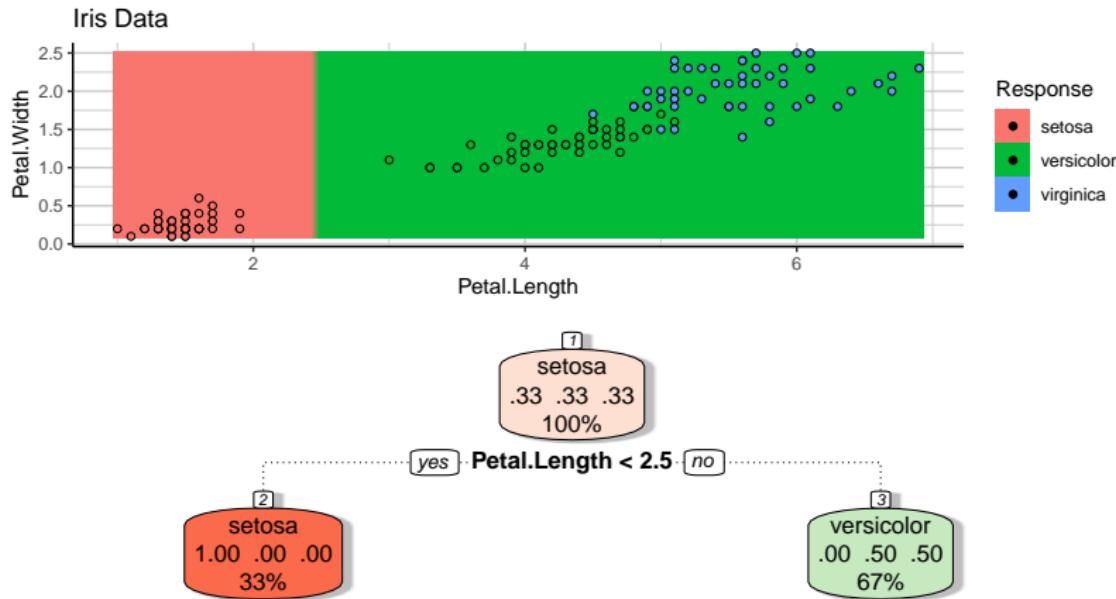
Trees are then grown by recursively applying *greedy* optimization to each node \mathcal{N} .

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

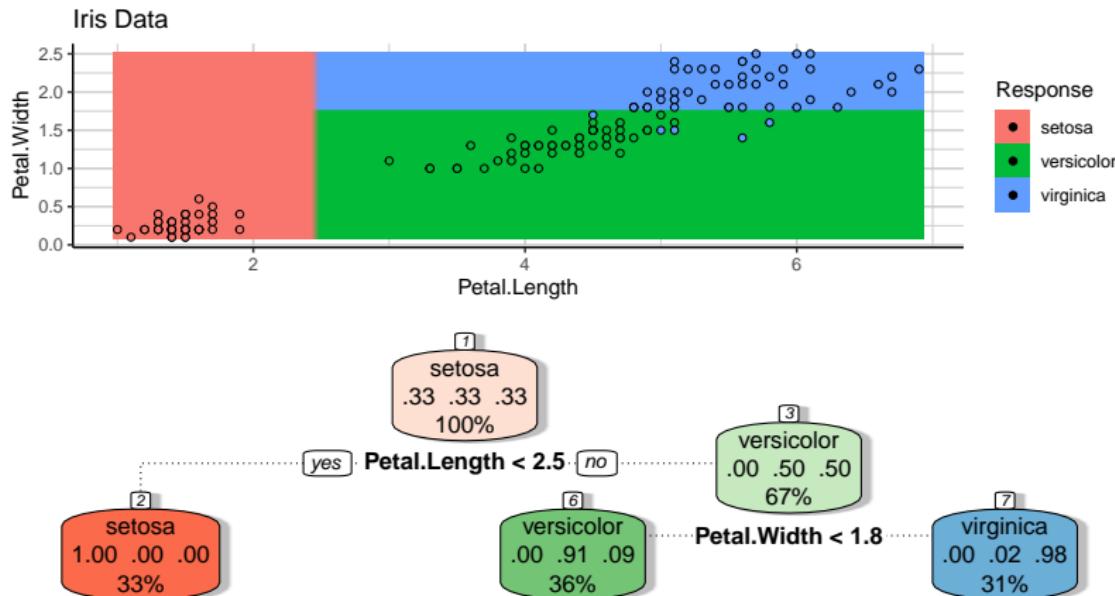
Start with a root node of all data, then search for a feature and split point that minimizes the empirical risk in the child nodes.



Nodes display their current label distribution here for illustration.

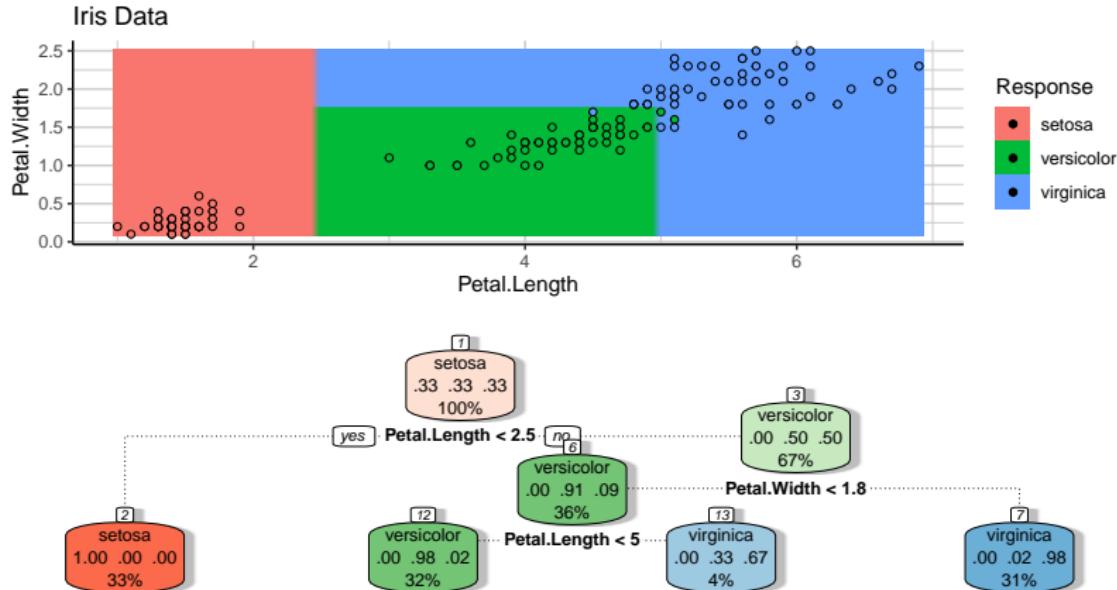
TREE GROWING

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:

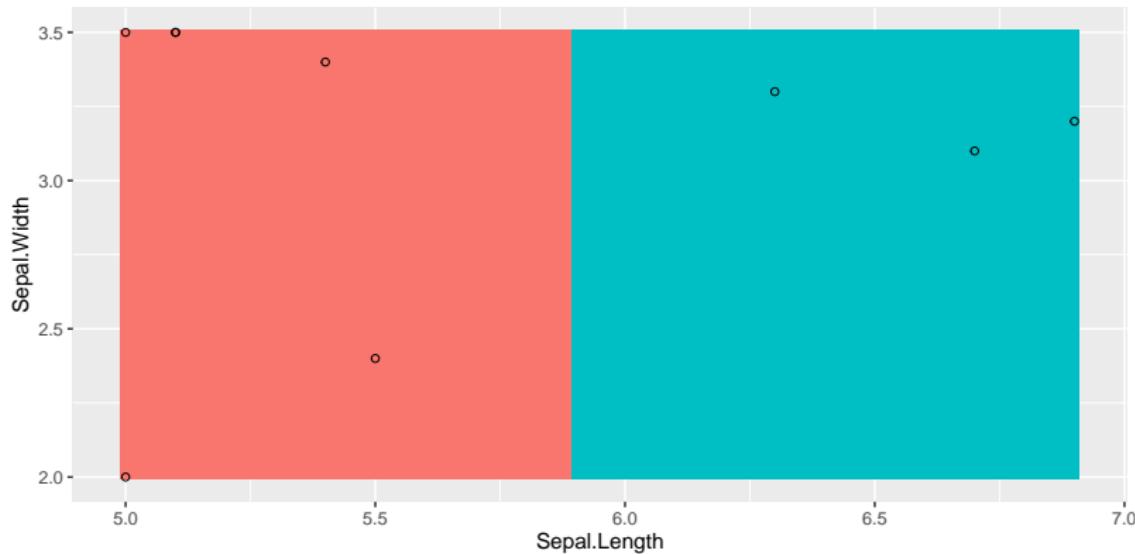


TREE GROWING

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:



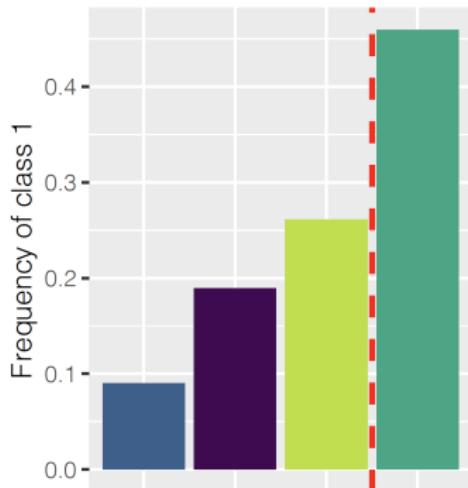
SPLIT PLACEMENT



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

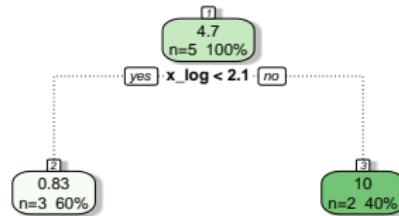
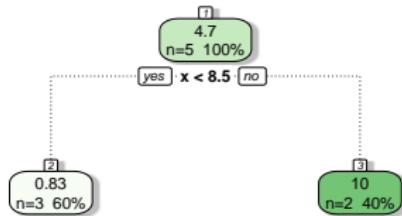
Monotone transformations of one or several features will neither change the value of the splitting criterion nor the structure of the tree, only the numerical value of the split point.

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:

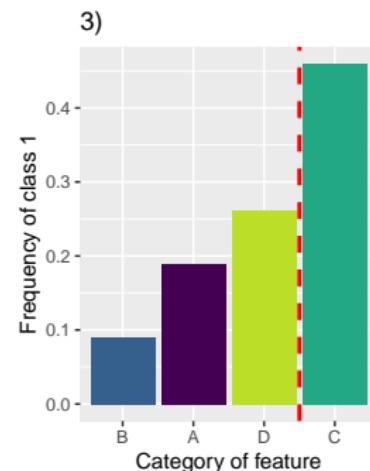
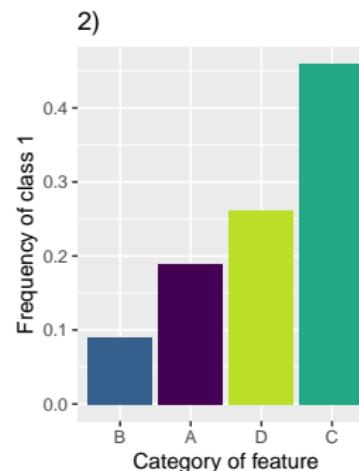
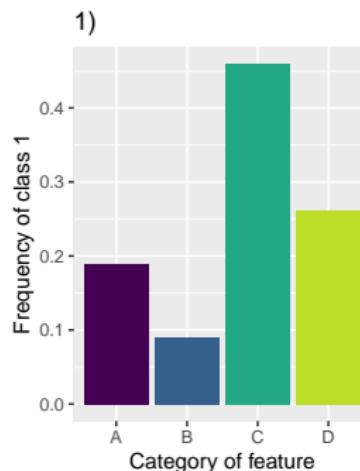
$$x_j \in \{a, c, e\} \leftarrow \mathcal{N} \rightarrow x_j \in \{b, d\}$$

- 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.



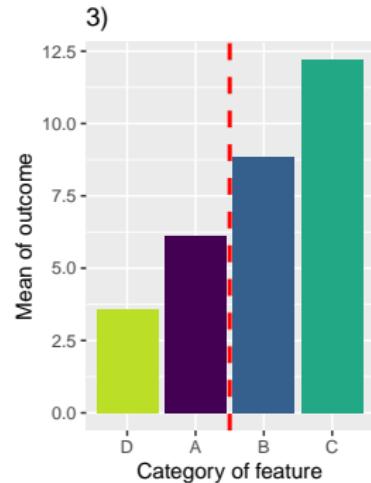
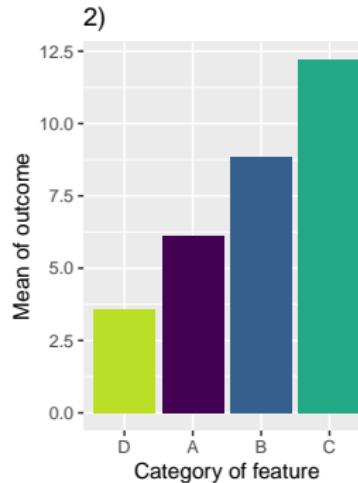
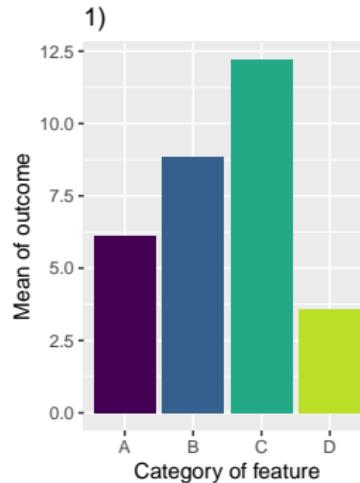
CART: NOMINAL FEATURES

- 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

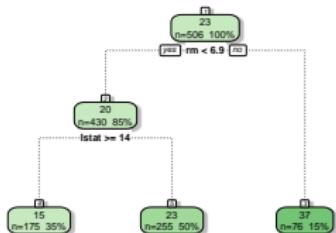
- When splits are evaluated, only observations for which the used feature is not missing are used. (This can actually bias splits towards using features with lots of missing values.)
- CART often use the so-called **surrogate split** principle to automatically deal with missing values in features used 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**:

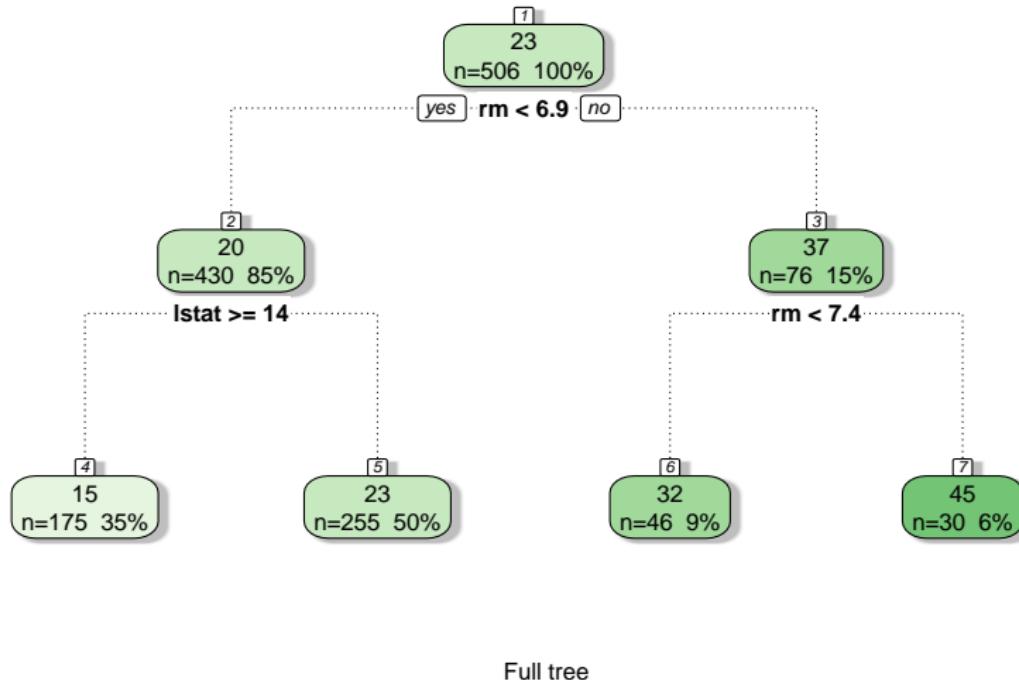
- Stop once the tree has reached a certain number of leaves.
- Don't try to split a node further if it contains too few observations.
- Don't perform a split that results in child nodes with too few observations.
- Don't perform a split unless it achieves a certain minimal improvement of the empirical 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

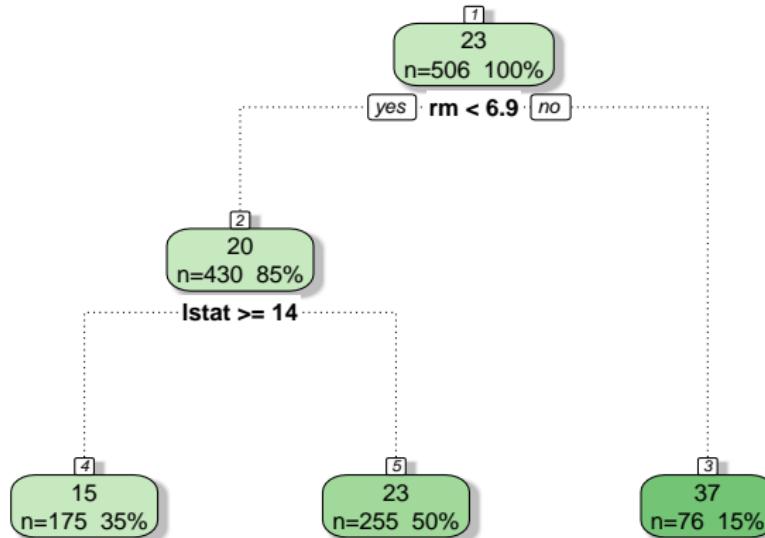
We try to solve problem 3 by **pruning**:

- 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

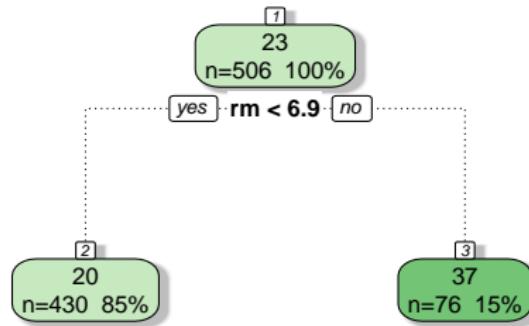


PRUNING



Pruning with complexity parameter = 0.072.

PRUNING



Pruning with complexity parameter = 0.171.

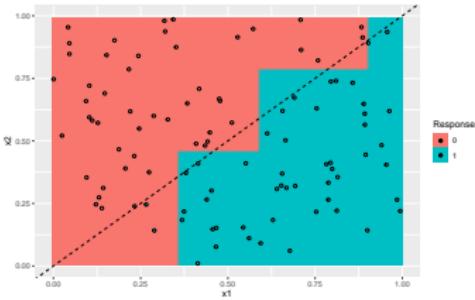
PRUNING



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

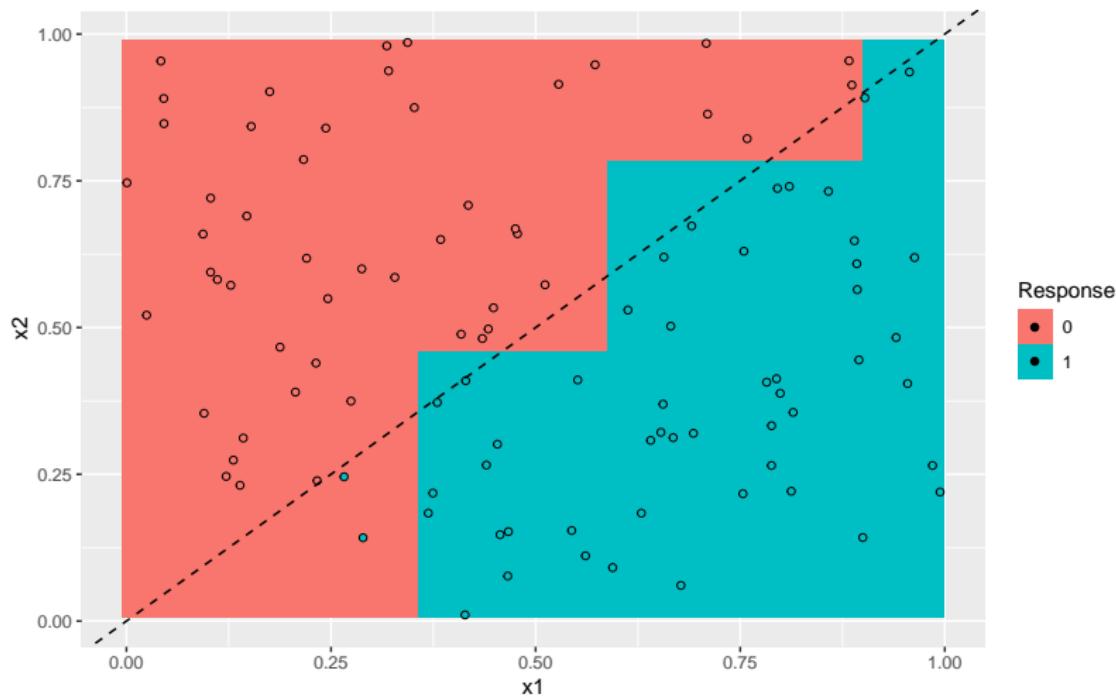
ADVANTAGES

- 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

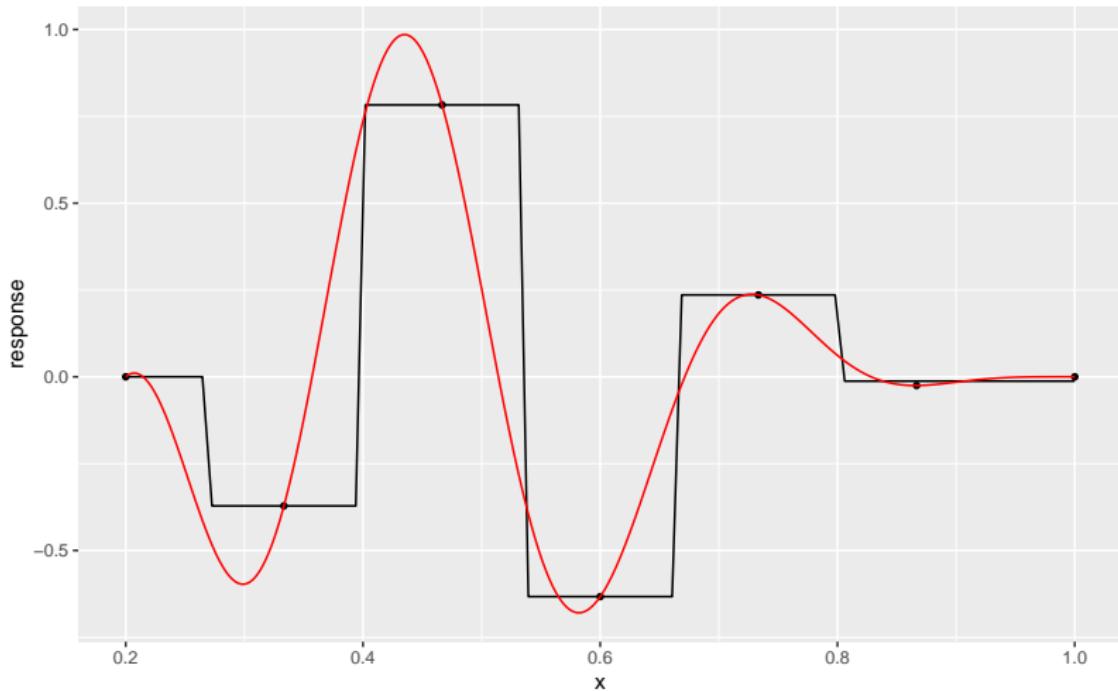
- Performs automatic feature selection
- Quite fast, scales well with larger data
- Flexibility through definition of custom split criteria or leaf-node prediction rules: clustering trees, semi-supervised trees, density estimation, etc.

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

Performance Evaluation

k-NN

Classification and Regression Trees (CART)

Random Forests

Tuning

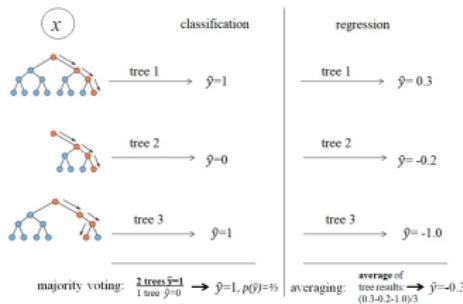
Nested Resampling

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

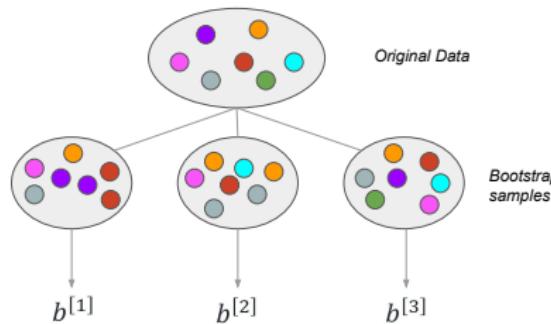
BAGGING

- 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} :

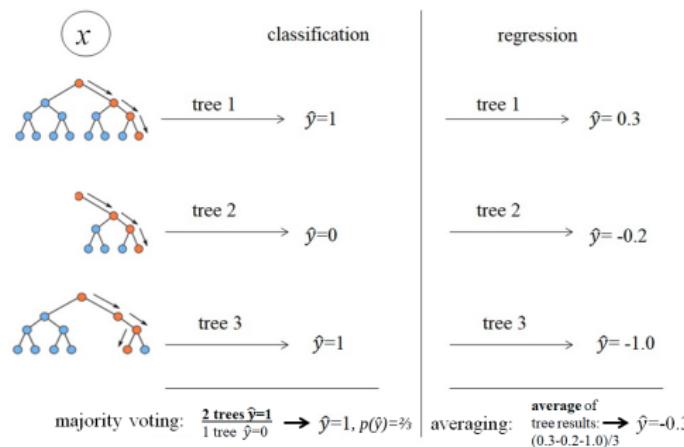
- Draw n observations from \mathcal{D} with replacement
- Fit the base learner on each of the M bootstrap samples to get models $\hat{f}(x) = \hat{b}^{[m]}(\mathbf{x})$, $m = 1, \dots, M$



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

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

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

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.

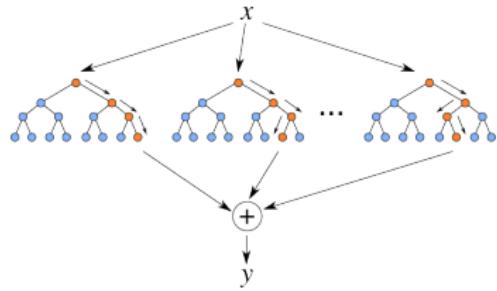
BAGGING: SYNOPSIS

- 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

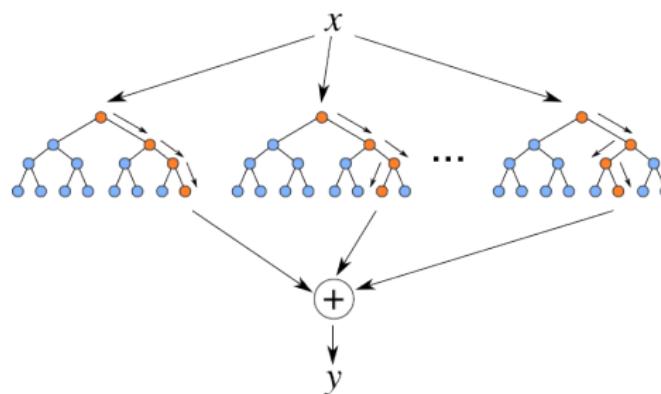


- Know how random forests are defined by extending the idea of bagging
- Understand that the goal is to decorrelate the trees
- Understand that the out-of-bag error is a way to obtain unbiased estimates of the generalization error during training

RANDOM FORESTS

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

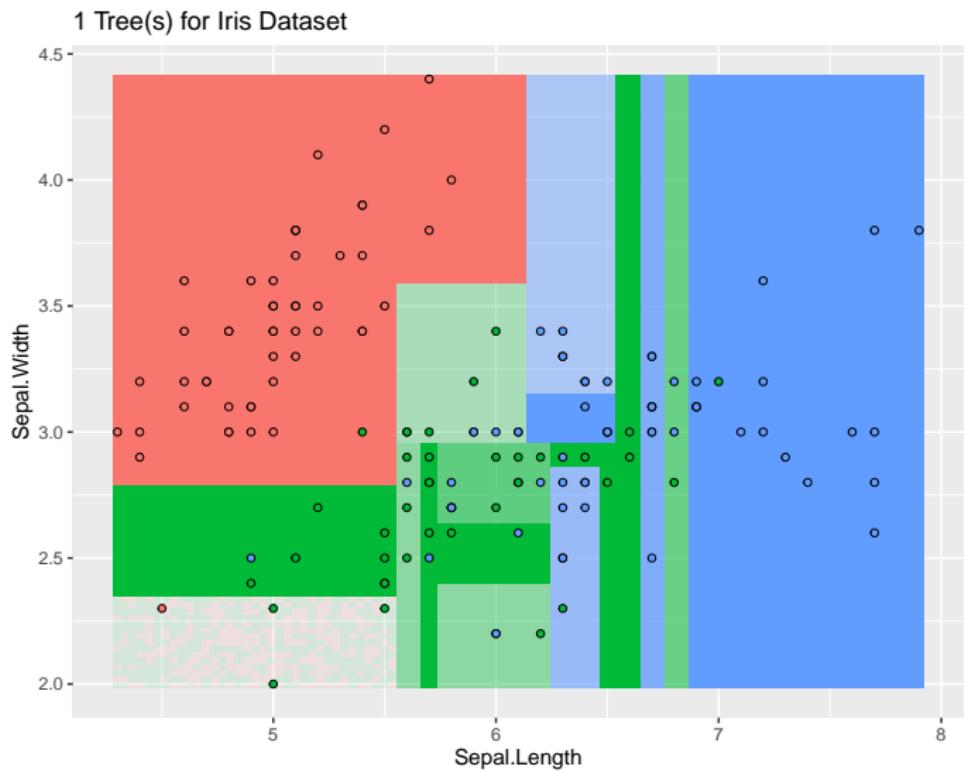
- Tree base learners on bootstrap samples of the data
- Uses **decorrelated** trees by randomizing splits (see below)
- Tree base learners are usually fully expanded, without aggressive early stopping or pruning, to **increase variance of the ensemble**



RANDOM FEATURE SAMPLING

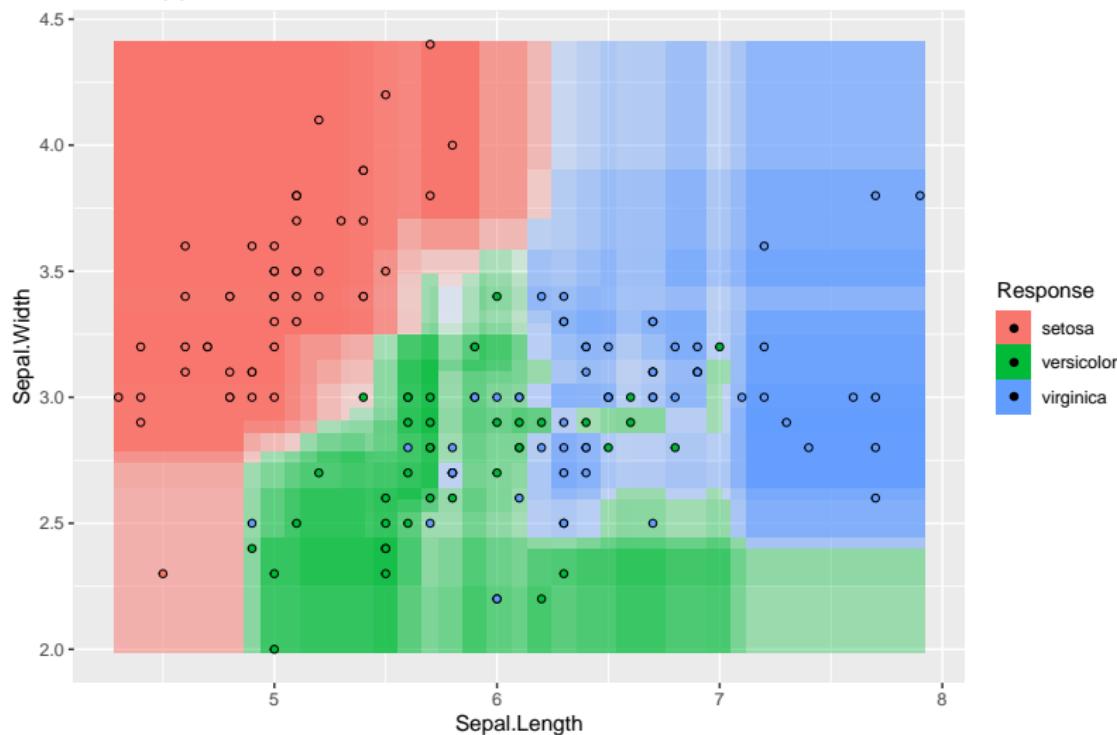
- From our analysis of bagging risk we can see that decorrelating trees improves the ensemble
- Simple randomized approach:
At each node of each tree, randomly draw $\text{mtry} \leq p$ candidate features to consider for splitting. Recommended values:
 - Classification: $\text{mtry} = \lfloor \sqrt{p}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

EFFECT OF ENSEMBLE SIZE



EFFECT OF ENSEMBLE SIZE

10 Tree(s) for Iris Dataset

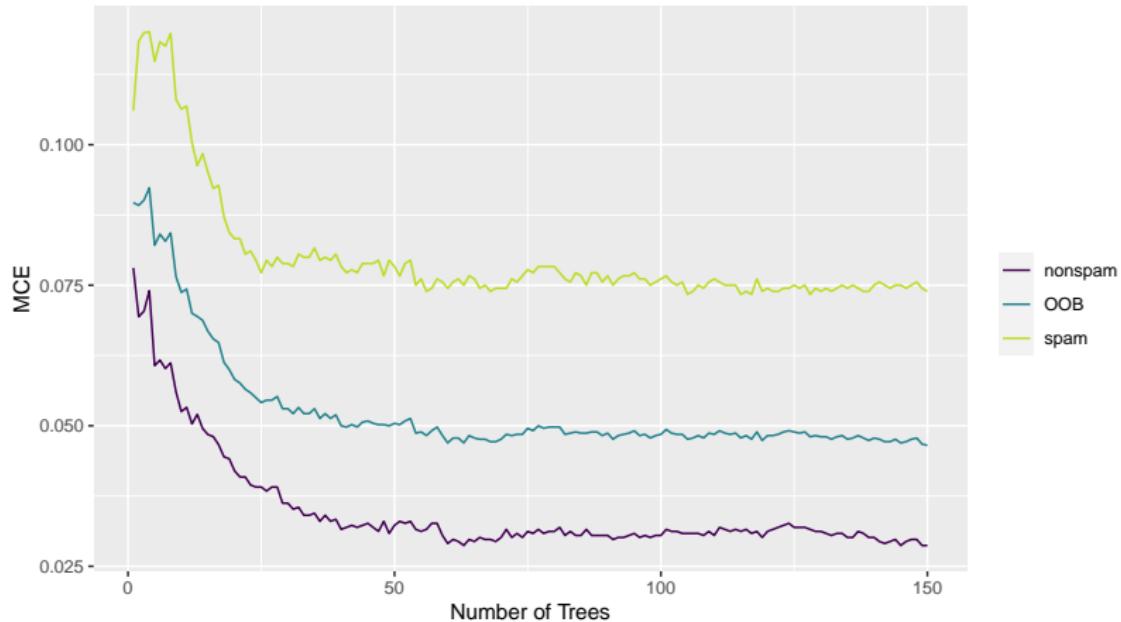


EFFECT OF ENSEMBLE SIZE



OUT-OF-BAG ERROR ESTIMATE

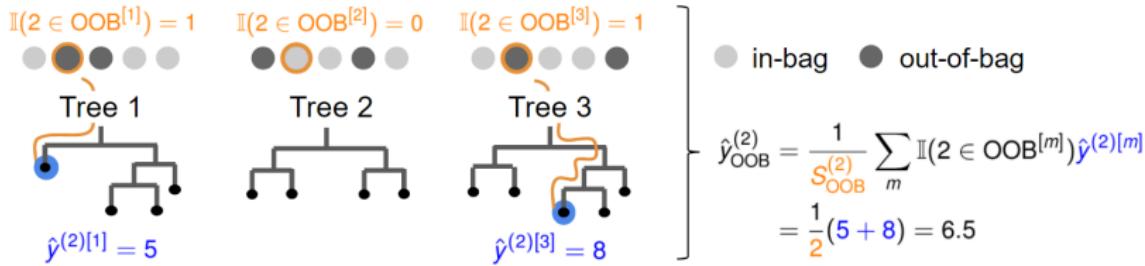
With the RF it is possible to obtain unbiased estimates of the generalization error directly during training, based on the out-of-bag observations for each tree:



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:

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

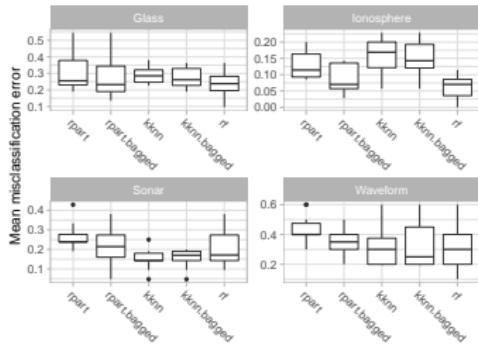
- Computing the probability of i being OOB in the m -th tree, we can see that the OOB error estimation is actually akin to 3-fold CV:

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

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

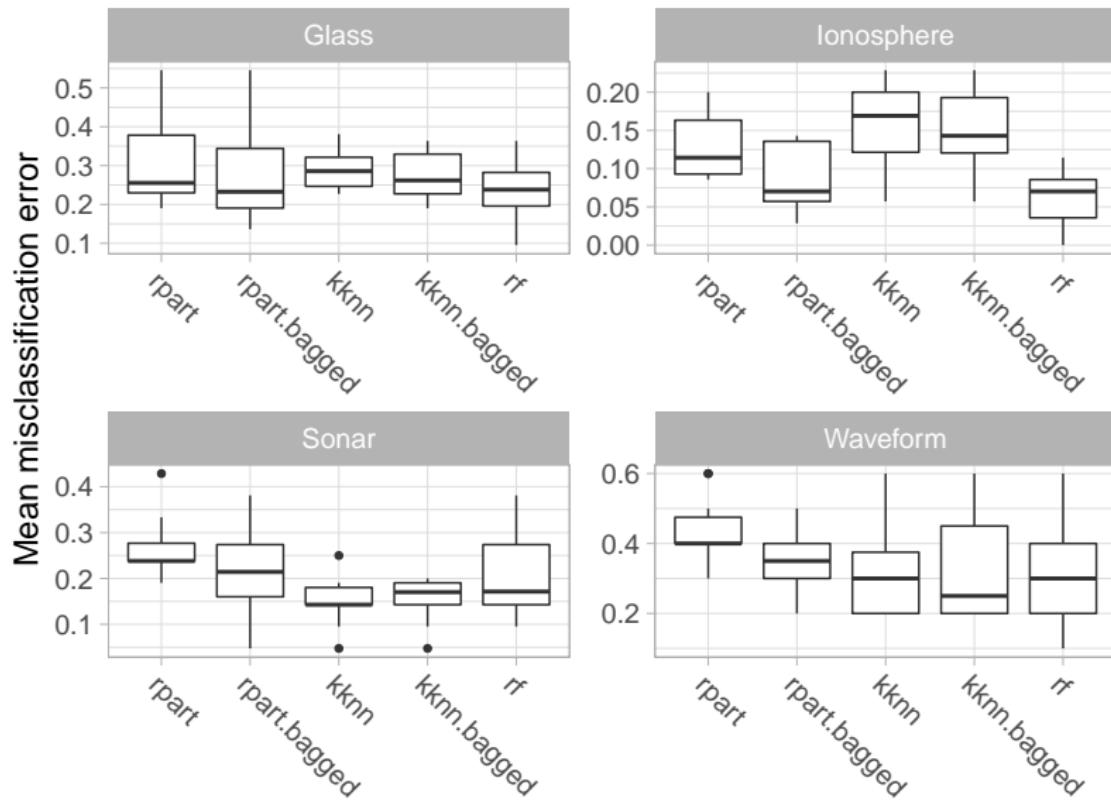
- Goal: Compare performance of random forest against (bagged) stable and (bagged) unstable methods
- Algorithms:
 - classification tree (CART, implemented in `rpart`,
`max.depth: 30, min.split: 20, cp: 0.01`)
 - bagged classification tree using 50 bagging iterations
(`bagged.rpart`)
 - k-nearest neighbors (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

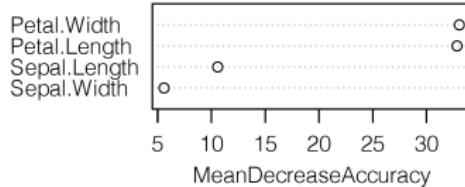
Bagging k-NN does not improve performance because:

- 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 .

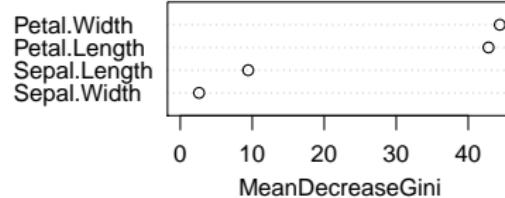
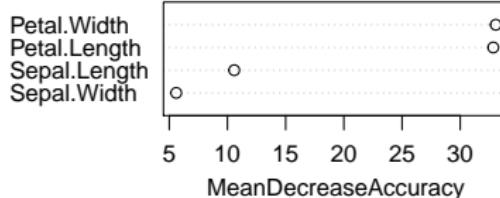
 Compute improvement in splitting criterion achieved by them.

 Add up these improvements.

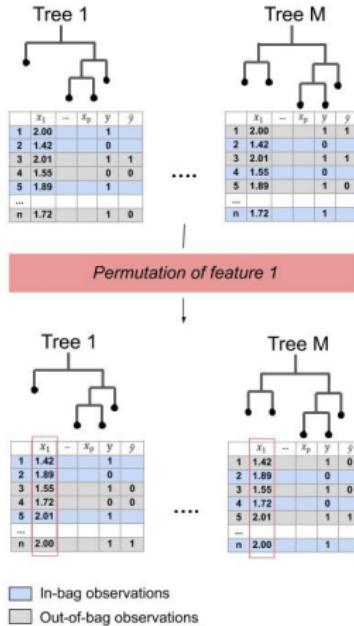
end for

 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 ψ_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

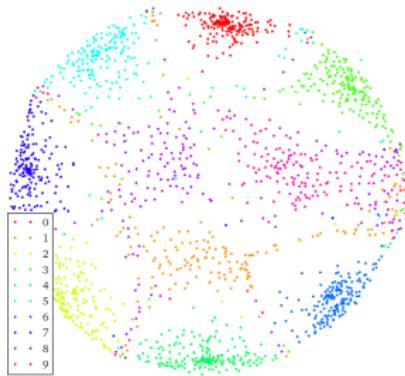
$$\widehat{VI}_j = \widehat{\text{err}}_{\text{OOB}, \psi_j} - \widehat{\text{err}}_{\text{OOB}}$$

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

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

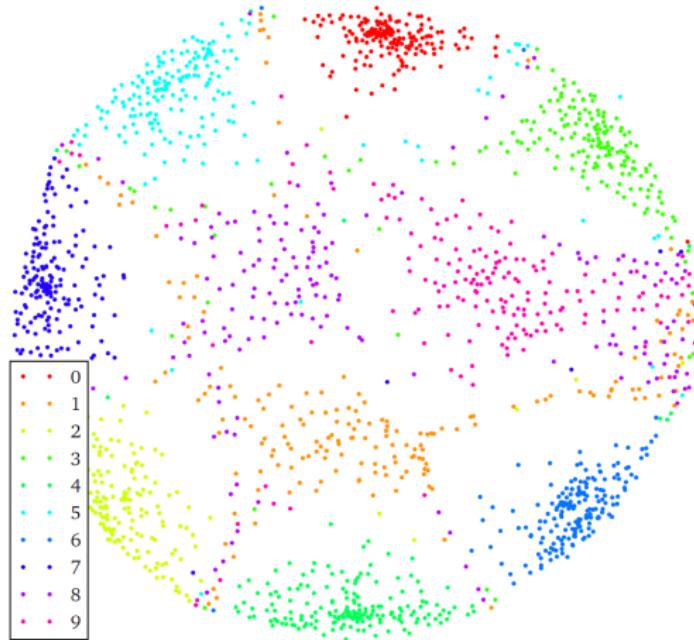
Steps 2 and 3 are then iterated a few times.

- 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

Performance Evaluation

k-NN

Classification and Regression Trees (CART)

Random Forests

Tuning

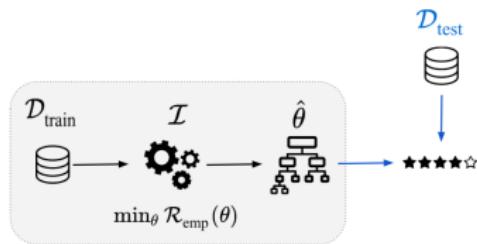
Nested Resampling

mlr3

Introduction to Machine Learning

Hyperparameter Tuning - Introduction

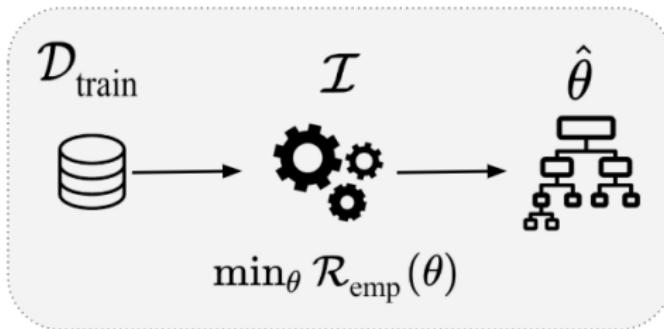
Learning goals



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

MOTIVATING EXAMPLE

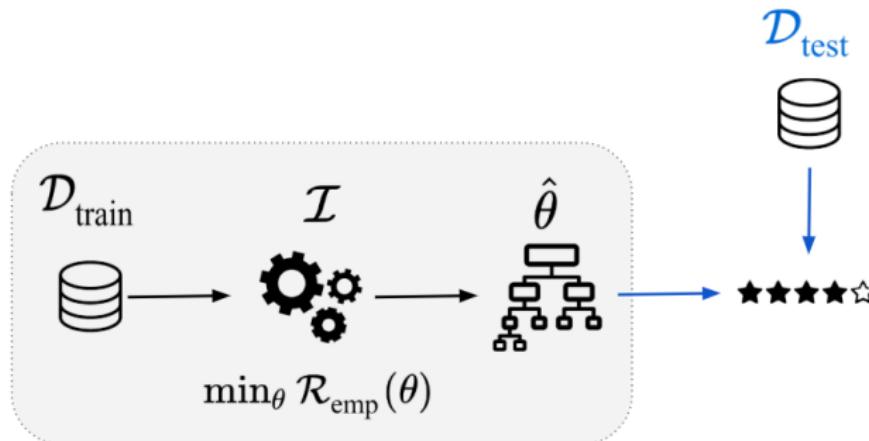
- Given a data set, we want to train a classification tree.
- We feel that a maximum tree depth of 4 has worked out well for us previously, so we decide to set this hyperparameter to 4.
- The learner ("inducer") \mathcal{I} takes the input data, internally performs **empirical risk minimization**, and 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 λ , we have to train the model **to completion** and evaluate its performance on the test set.

- We choose the tree depth λ 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 θ 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

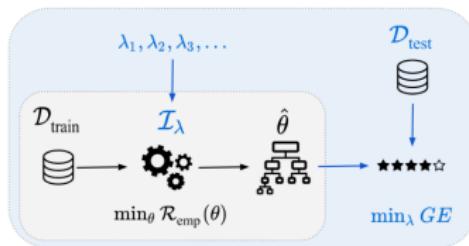
We summarize all hyperparameters we want to tune over in a vector $\lambda \in \Lambda$ of (possibly) mixed type. HPs can have different types:

- 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

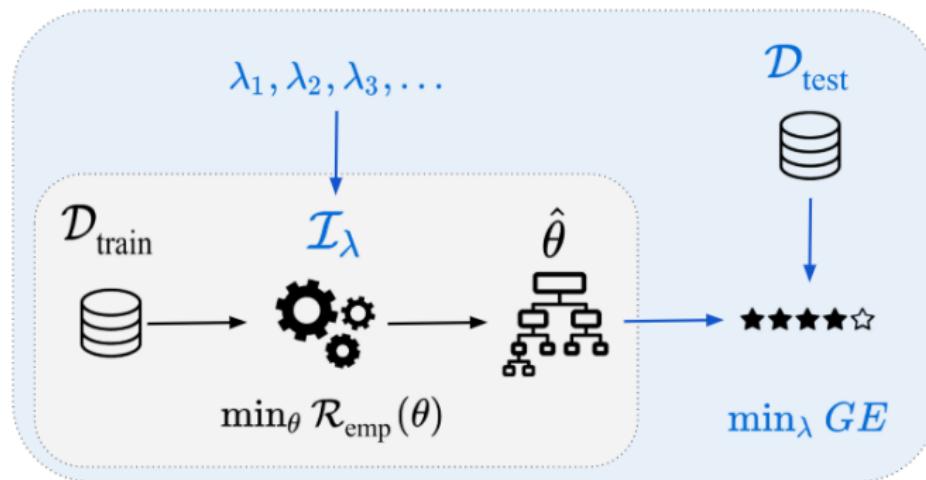
TUNING

Recall: **Hyperparameters** λ 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** θ which define the fitted model \hat{f} .

(Hyperparameter) Tuning is the process of finding good model hyperparameters λ .

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 Λ_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 λ (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:

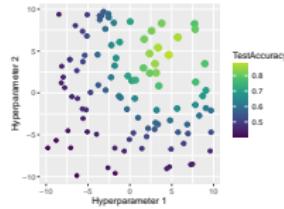
- The data set
- The learner (possibly: several competing learners?) that is tuned
- The learner's hyperparameters and their respective regions-of-interest over which we optimize
- The performance measure, as determined by the application.
Not necessarily identical to the loss function that defines the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



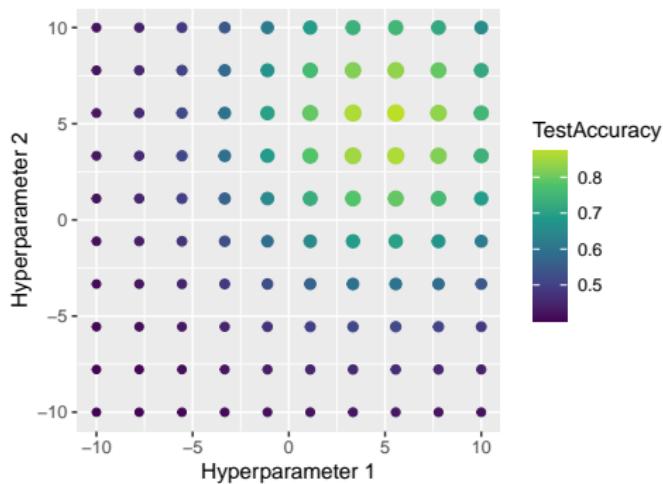
Learning goals

- 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

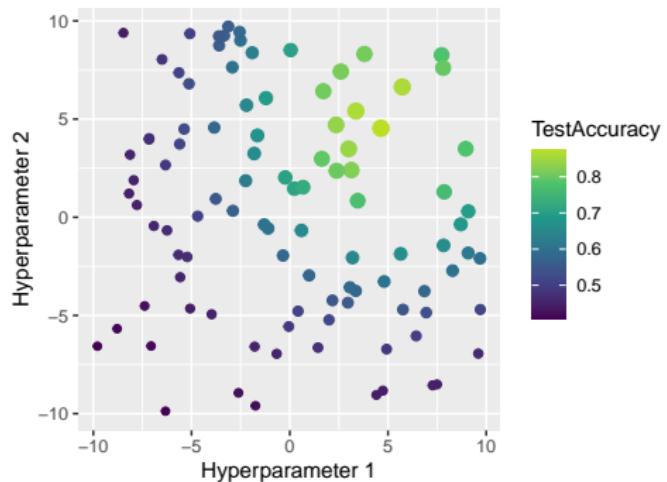
Disadvantages

- 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

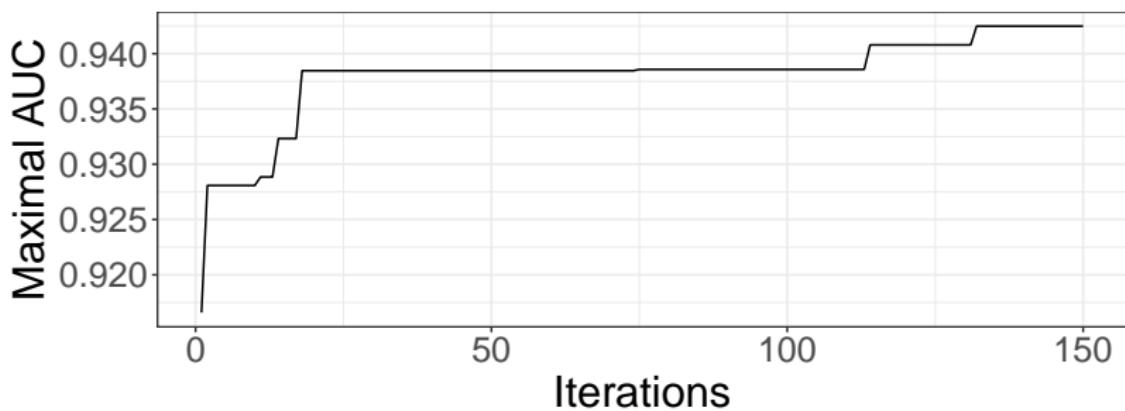
Disadvantages

- 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

Performance Evaluation

k-NN

Classification and Regression Trees (CART)

Random Forests

Tuning

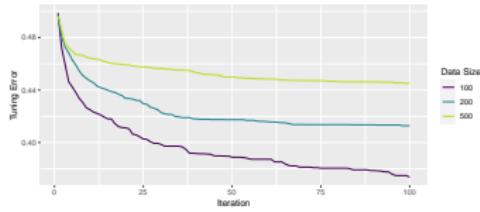
Nested Resampling

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.

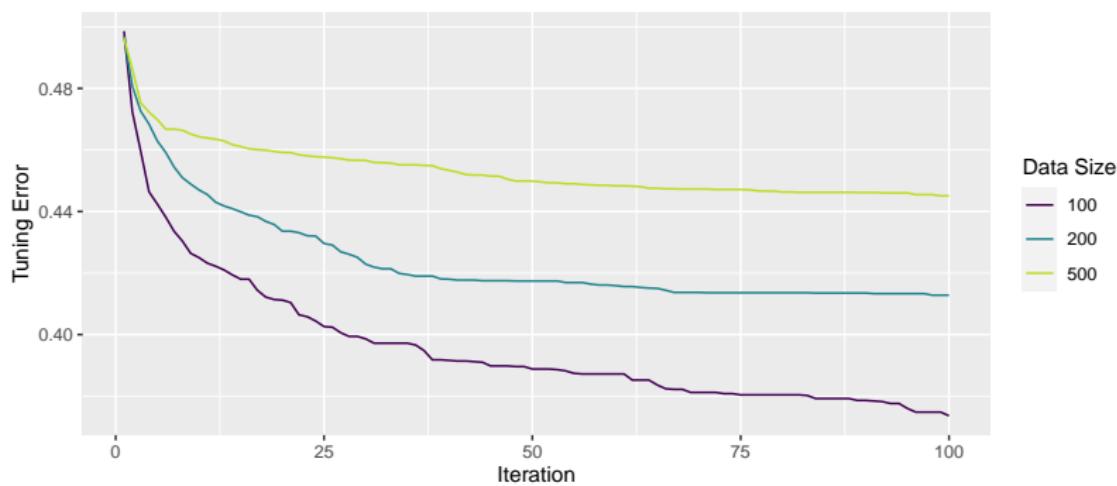
Problem

- 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 λ .
- Here, the learner is a (nonsense) feature-independent classifier, where λ 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 λ) 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 λ values.
- We repeat this experiment 50 times and average results.

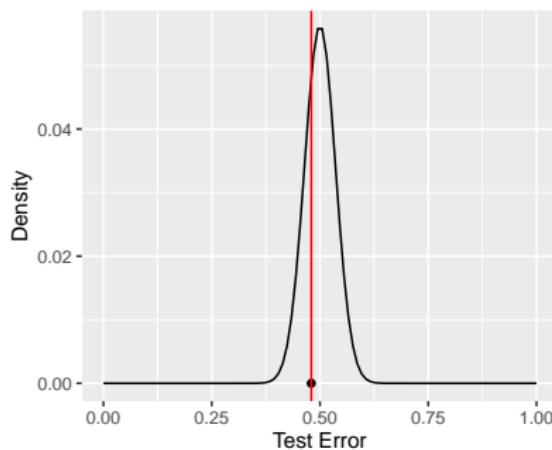
INSTRUCTIVE AND PROBLEMATIC EXAMPLE



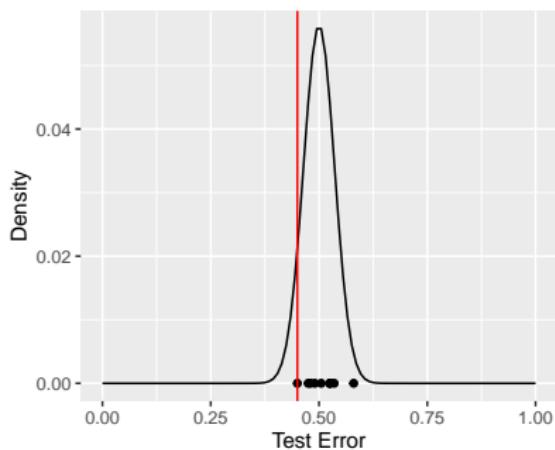
- Plotted is the best “tuning error” (i.e. the performance of the model with fixed λ as evaluated by the cross-validation) after k tuning iterations.
- We have performed the experiment for different sizes of learning data that were cross-validated.

INSTRUCTIVE AND PROBLEMATIC EXAMPLE

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



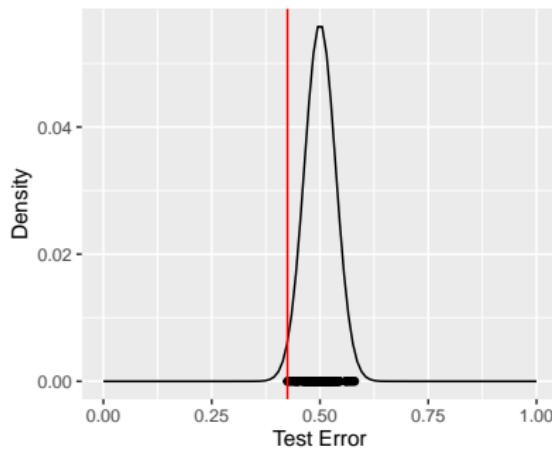
$n = 200$; #runs = 10; best err = 0.45



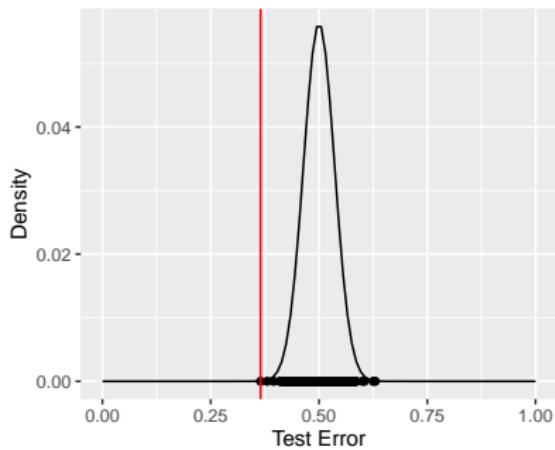
- For 1 experiment, the CV score will be nearly 0.5, as expected
- We basically sample from a (rescaled) binomial distribution when we calculate error rates
- And multiple experiment scores are also nicely arranged around the expected mean 0.5

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

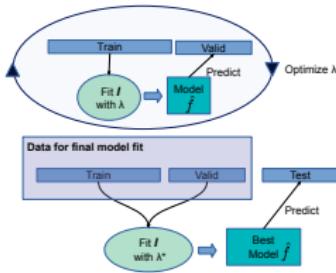
- For steps that themselves require resampling (e.g., hyperparameter tuning) this results in **nested resampling**, i.e., resampling strategies for both
 - tuning: an inner resampling loop to find what works best based on training data
 - outer evaluation on data not used for tuning to get honest estimates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

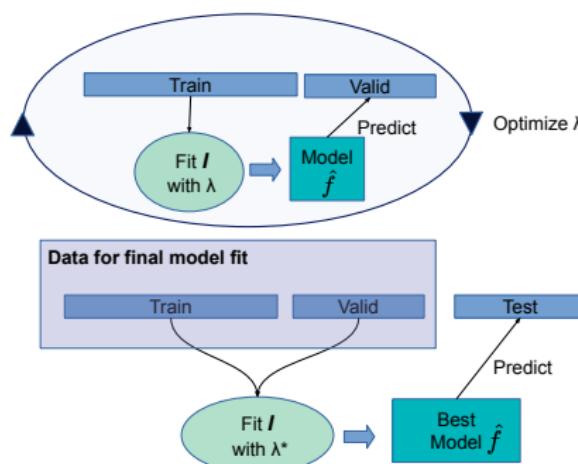
We need to

- **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

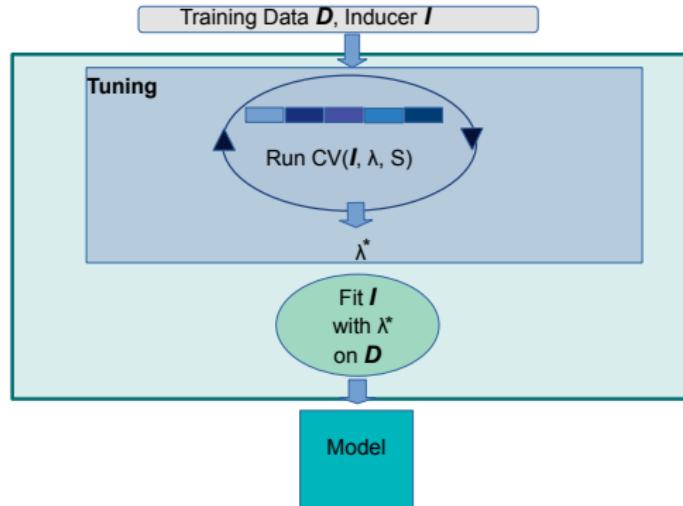
Simplest method to achieve this: a 3-way split

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



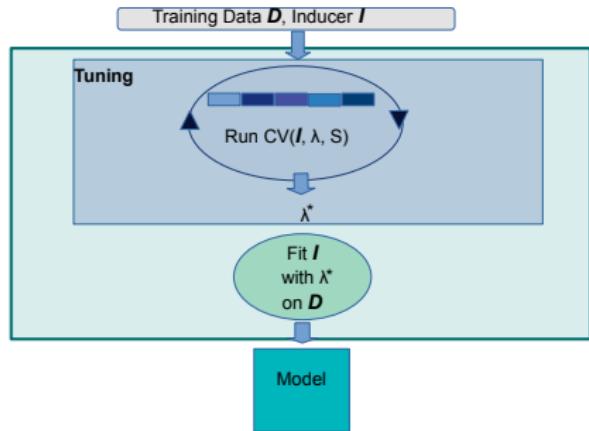
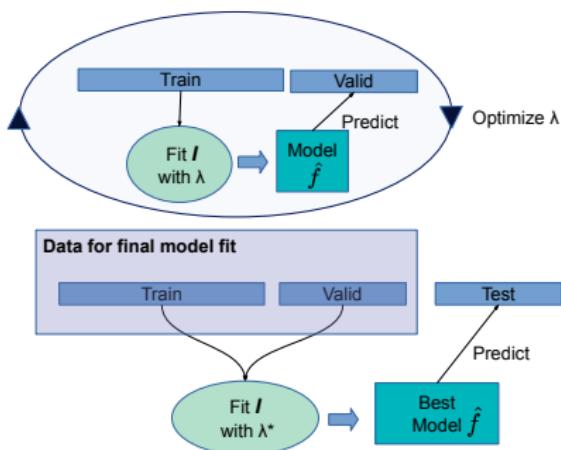
TUNING AS PART OF MODEL BUILDING

- 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

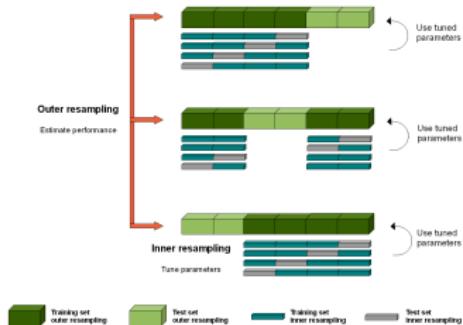
More precisely: the combined training & validation set is actually the training set for the “self-tuning” endowed algorithm.



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

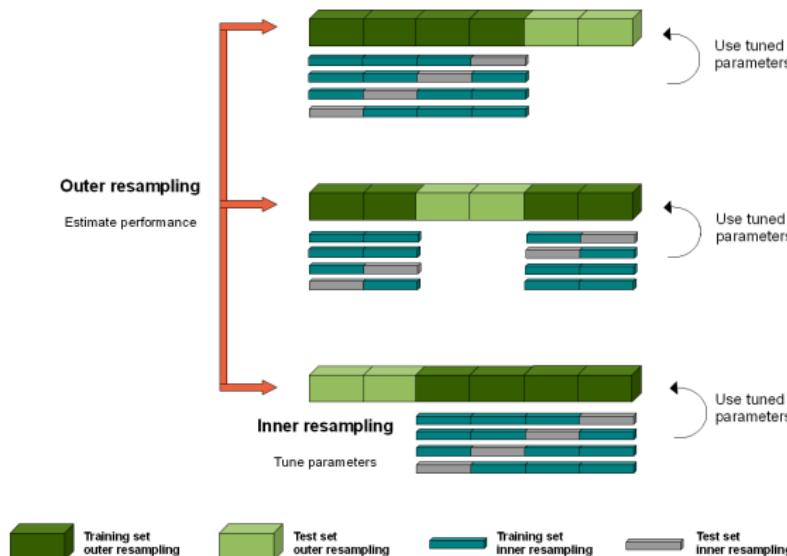
NESTED RESAMPLING

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

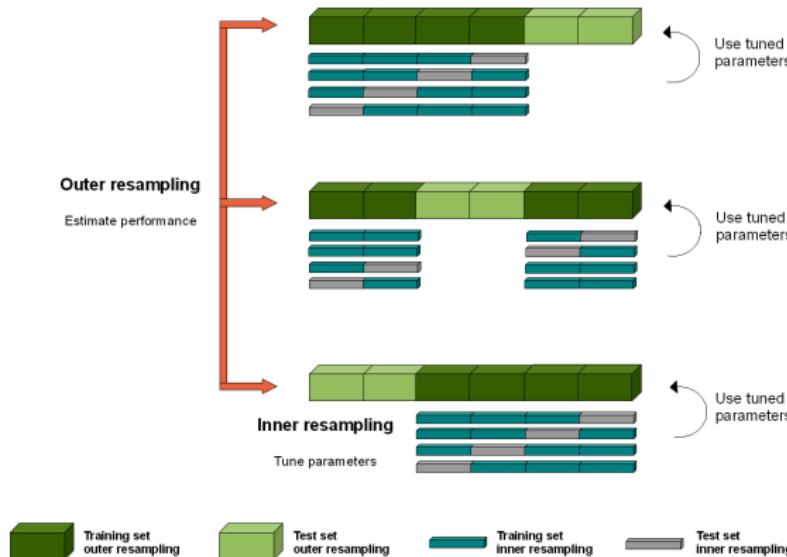
Assume we want to tune over a set of candidate HP configurations $\lambda_i; i = 1, \dots$ with 4-fold CV in the inner resampling and 3-fold CV in the outer loop. The outer loop is visualized as the light green and dark green parts.



NESTED RESAMPLING

In each iteration of the outer loop we:

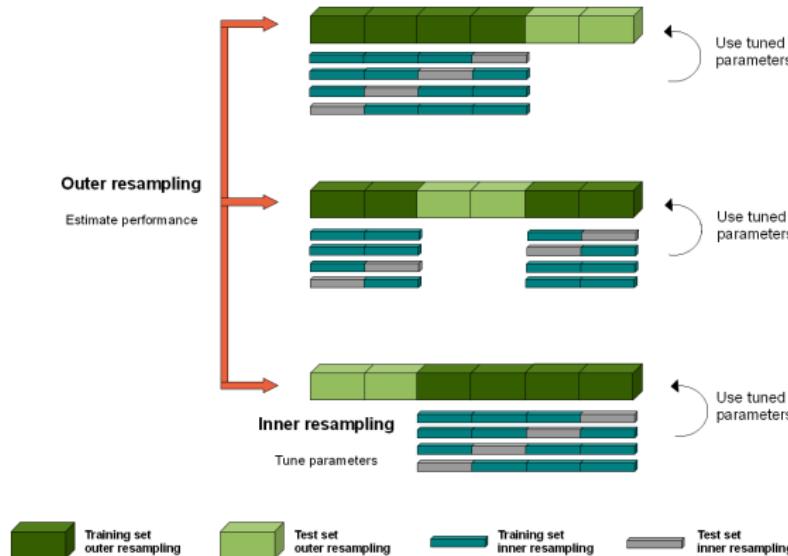
- Split off the light green testing data
- Run the tuner on the dark green part of the data, e.g., evaluate each λ_i through fourfold CV on the dark green part



NESTED RESAMPLING

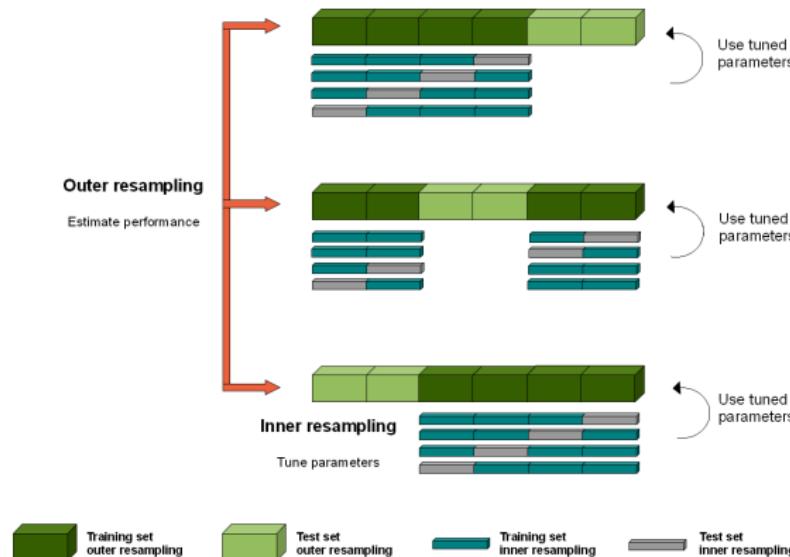
In each iteration of the outer loop we:

- Return the winning λ^* 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



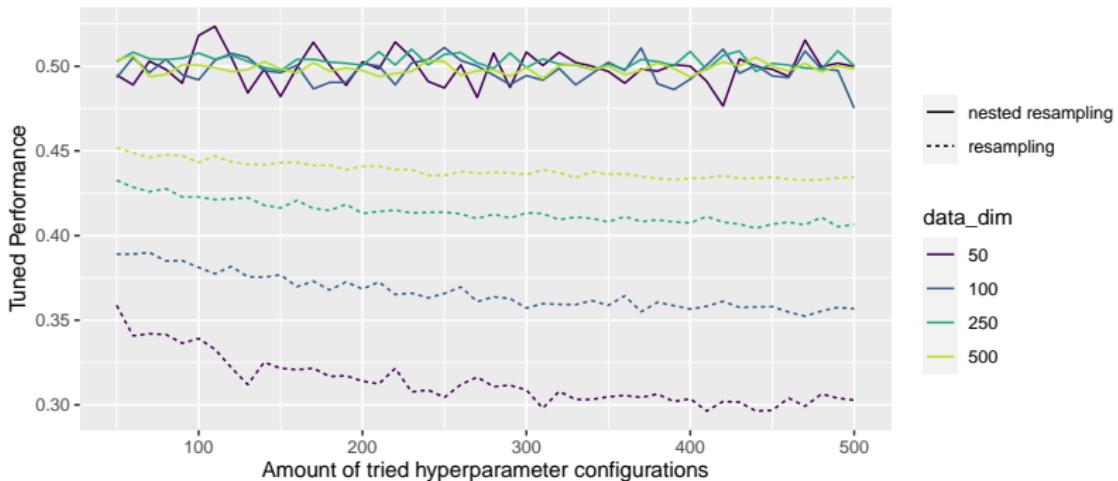
NESTED RESAMPLING

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

Performance Evaluation

k-NN

Classification and Regression Trees (CART)

Random Forests

Tuning

Nested Resampling

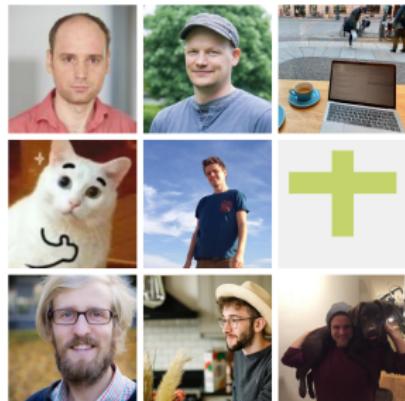
mlr3

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**

February 11, 2021

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

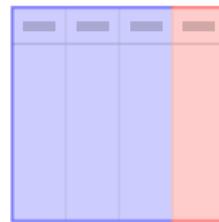
MLR3 PHILOSOPHY

- 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

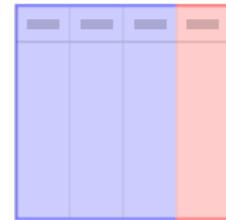
DATA

- Tabular data



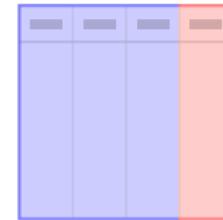
DATA

- Tabular data
- Features



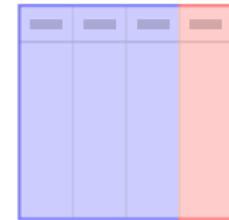
DATA

- Tabular data
- Features
- Target / outcome to predict



DATA

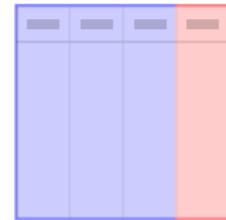
- Tabular data
- Features
- Target / outcome to predict
 - discrete for classification
 - continuous for regression



DATA

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

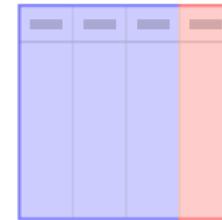
⇒ target determines the machine learning “Task”



DATA

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

⇒ target determines the machine learning “Task”



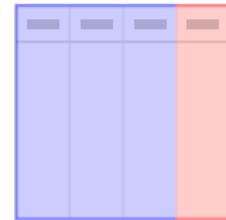
```
print(iris) # included in R

#>   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”



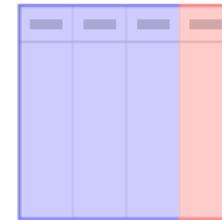
```
print(iris) # included in R
```

```
#>   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
```

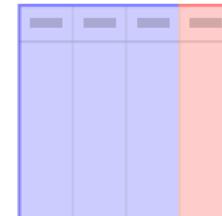
```
#> Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#> 1           5.1        3.5       1.4        0.2 setosa
#> 2           4.9        3.0       1.4        0.2 setosa
#> ...
```

```
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
#> ...
```

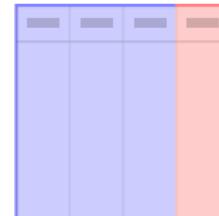
Task ID

```
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
#> ...
```

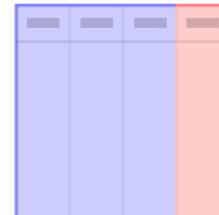
Task ID data
↓ ↓

```
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)

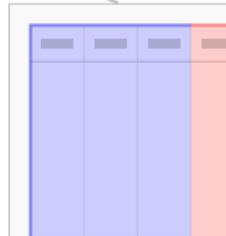
mlr_learners_table[1:10, c("key", "packages", "predict_types")]

#                               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

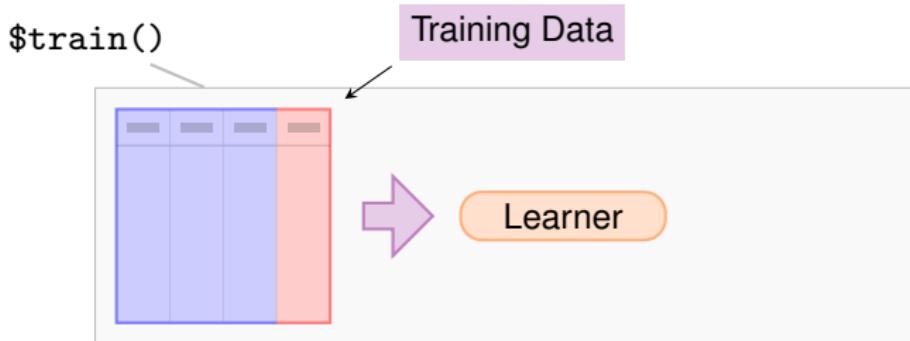
LEARNING ALGORITHMS

\$train()

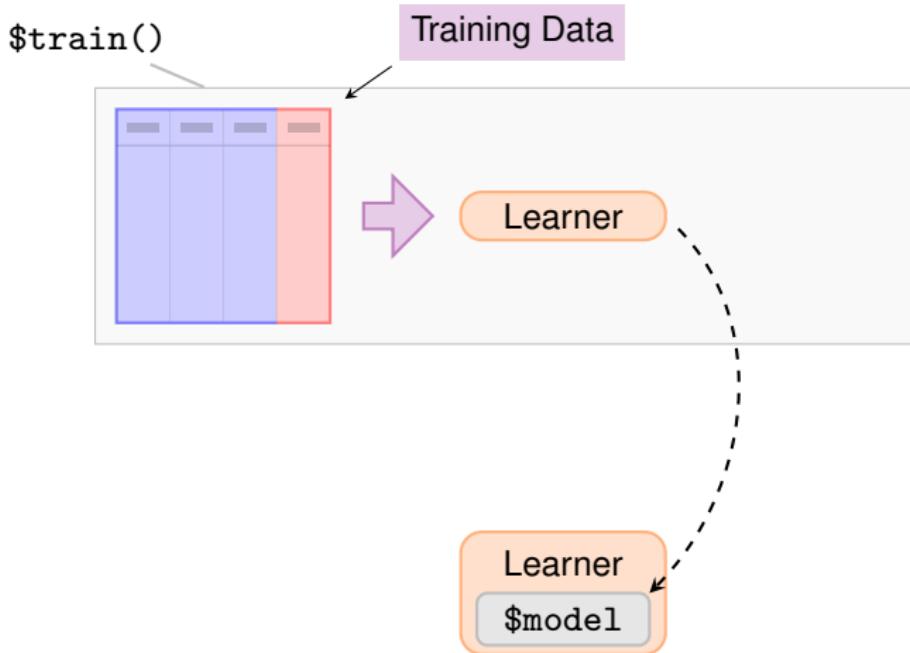


Learner

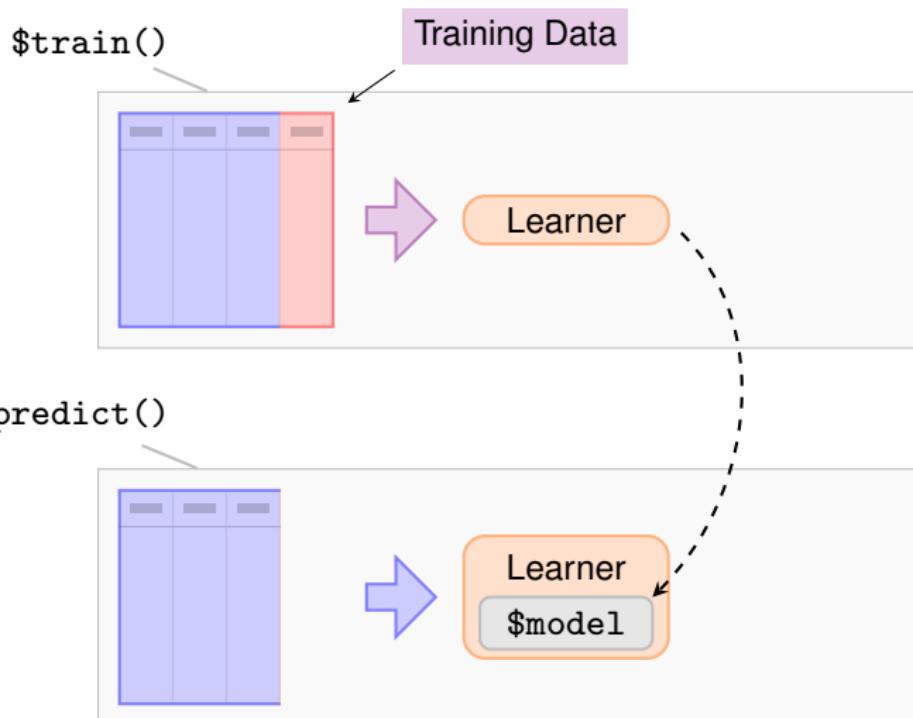
LEARNING ALGORITHMS



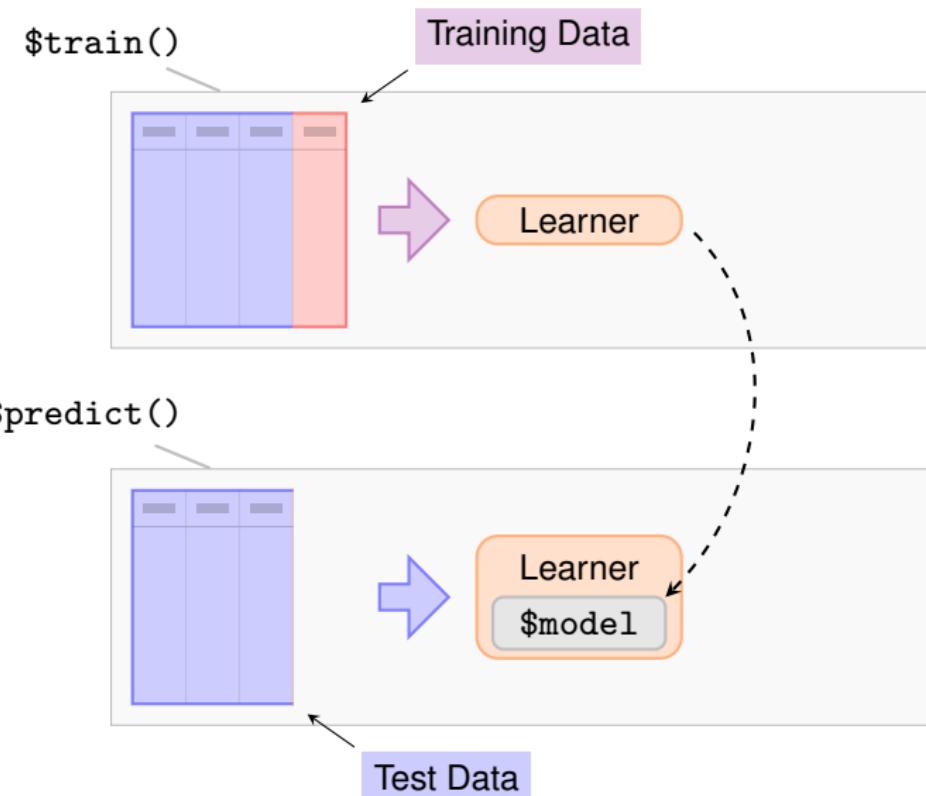
LEARNING ALGORITHMS



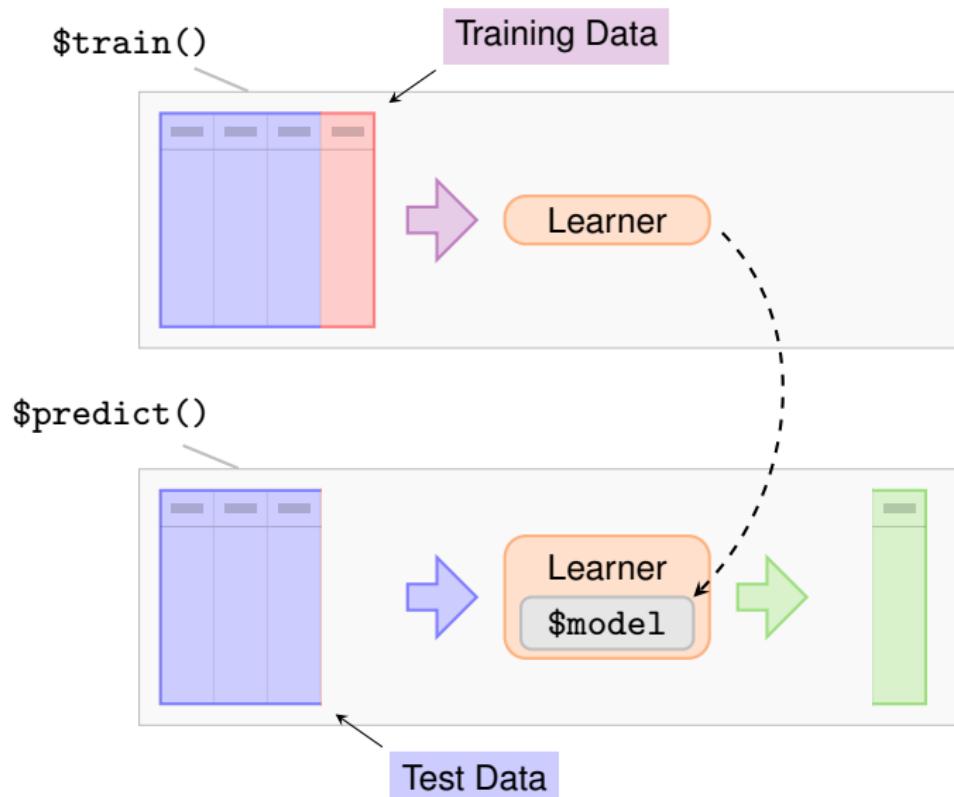
LEARNING ALGORITHMS



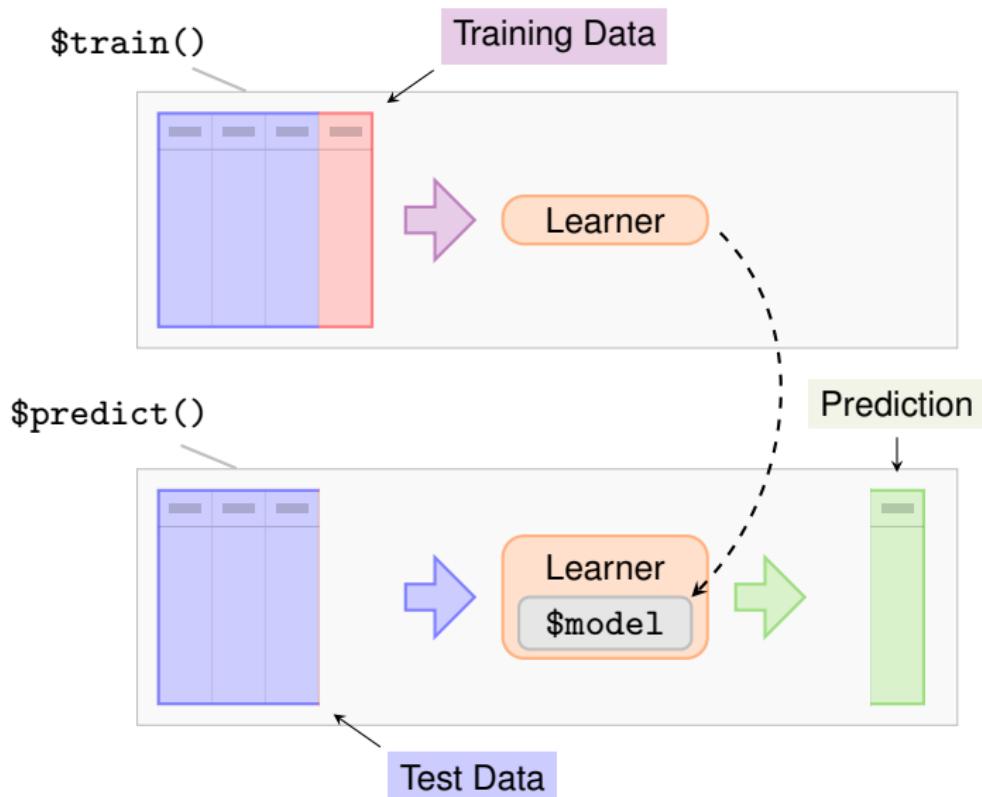
LEARNING ALGORITHMS



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

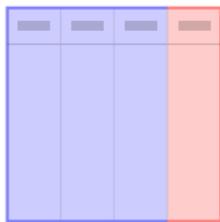
```
as.data.table(prediction)
#>   row_id truth    response
#> 1:      1 <NA>     setosa
#> 2:      2 <NA> versicolor
```

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

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

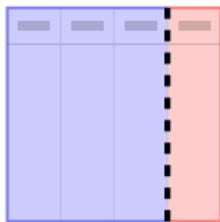
Performance

PERFORMANCE EVALUATION



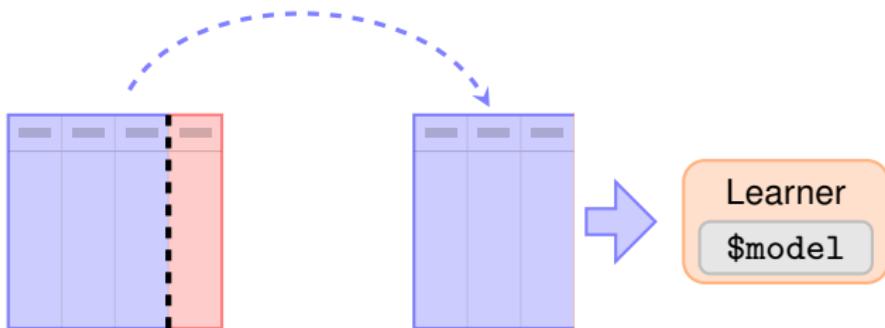
Learner
\$model

PERFORMANCE EVALUATION

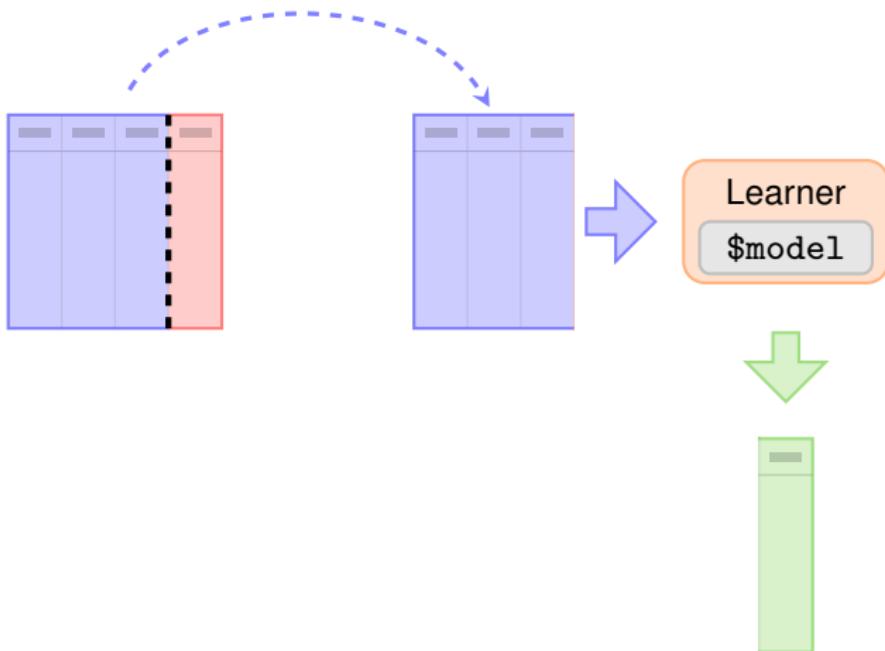


Learner
\$model

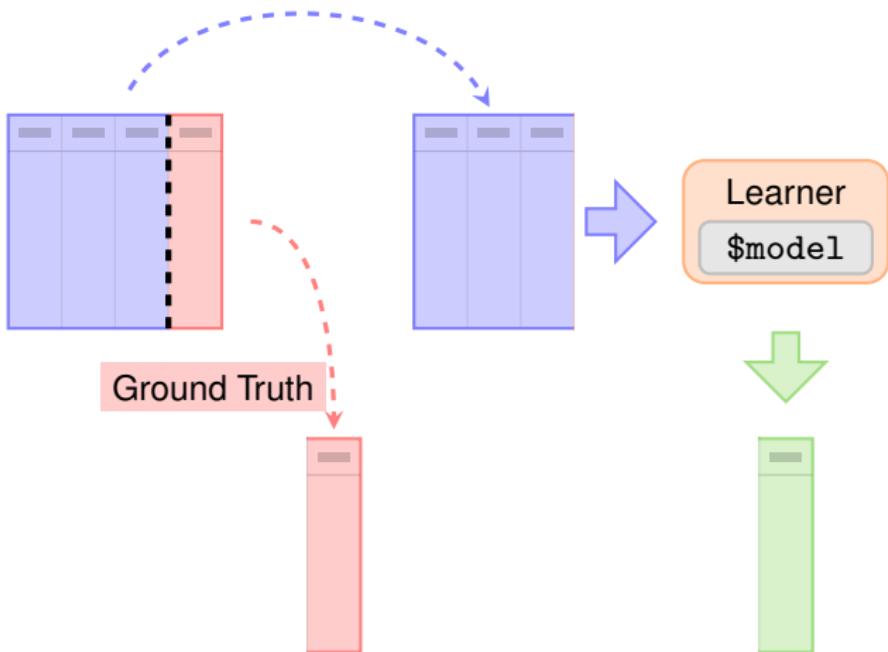
PERFORMANCE EVALUATION



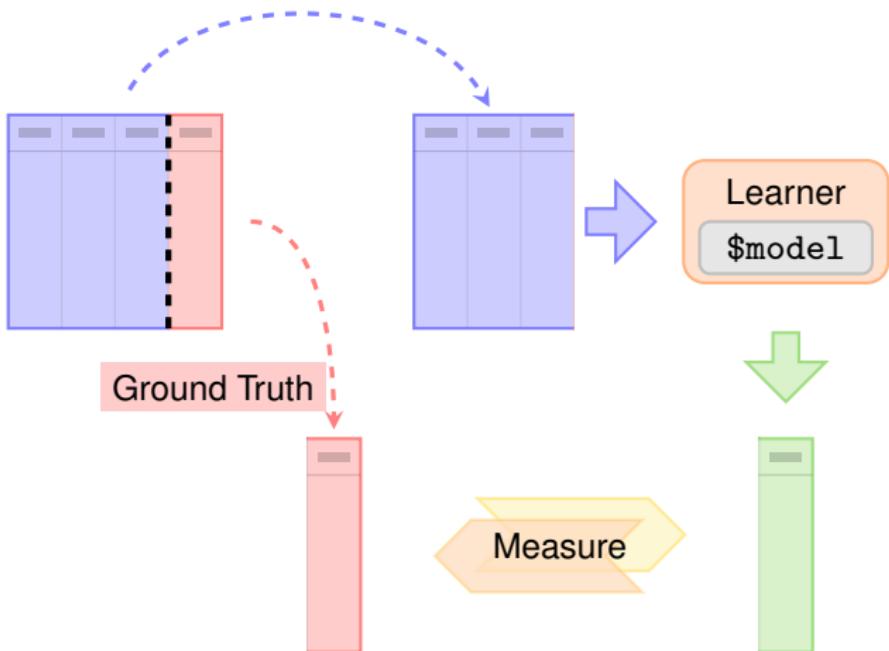
PERFORMANCE EVALUATION



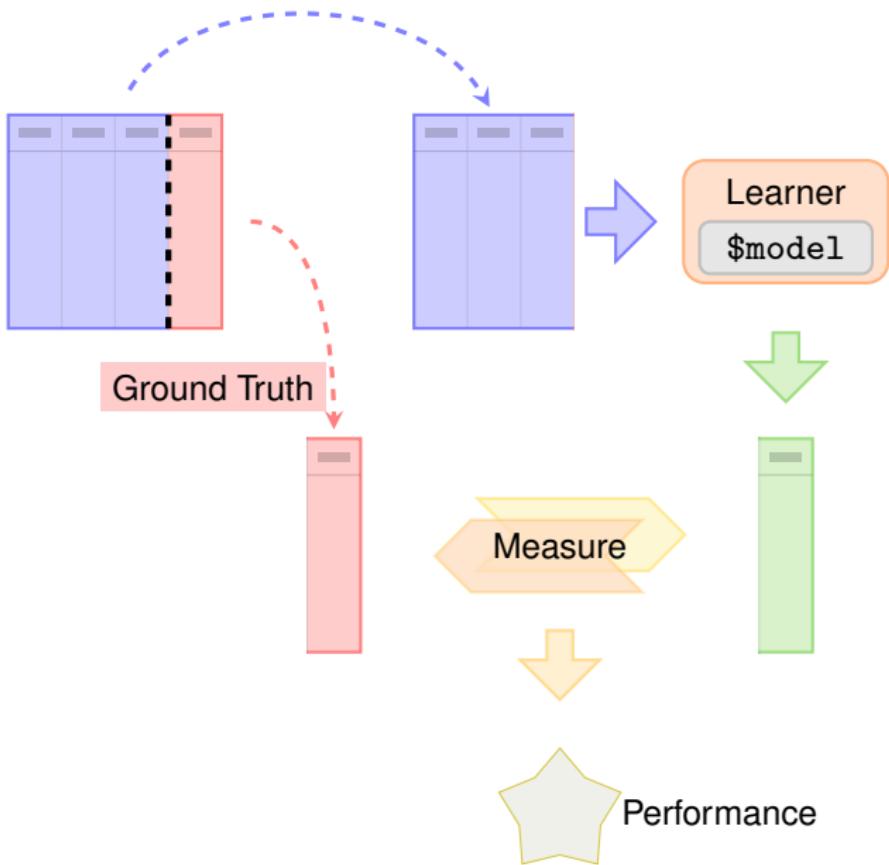
PERFORMANCE EVALUATION



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

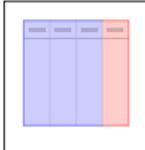
```
pred$score(msr("classif.ce"))  
#> classif.ce  
#>      0.5
```

Outro

OVERVIEW

Ingredients:

Data



TaskClassif,
TaskRegr,
tsk()

Learning Algorithms

Learner



Learner
\$model

lrn() ⇒ Learner,
↪ Learner\$train(),
↪ Learner\$predict() ⇒ Prediction

Measure Performance

Measure



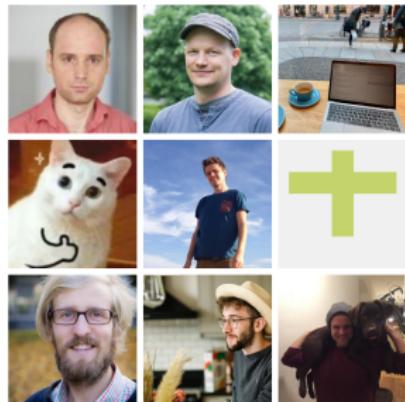
Prediction\$score(),
msr() ⇒ Measure

Modern Machine Learning in R



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

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

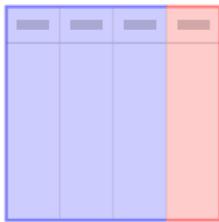


**Bernd Bischl, Michel Lang, Martin Binder, Florian Pfisterer, Jakob Richter,
Patrick Schratz, Lennart Schneider, Raphael Sonabend, Marc Becker**

February 11, 2021

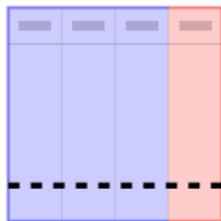
Resampling

RESAMPLING



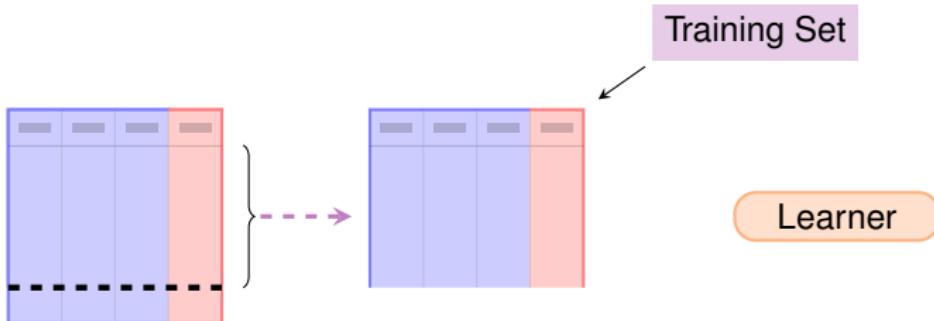
Learner

RESAMPLING

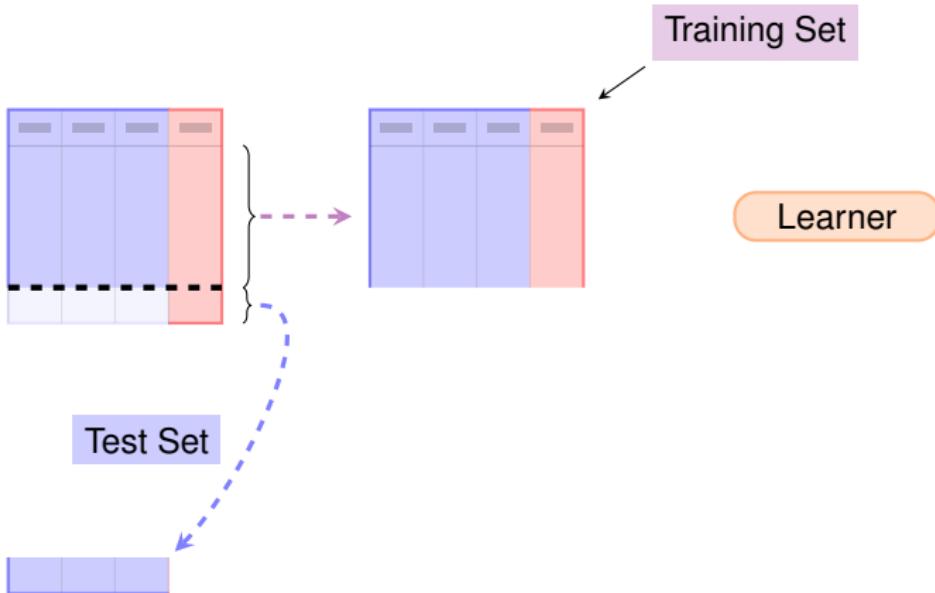


Learner

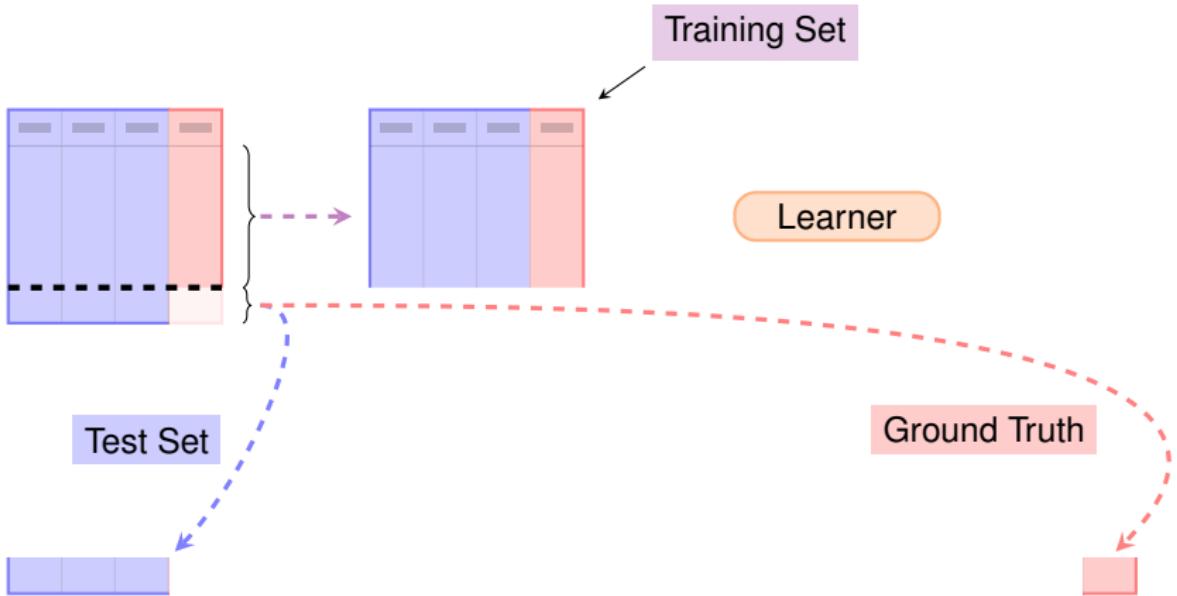
RESAMPLING



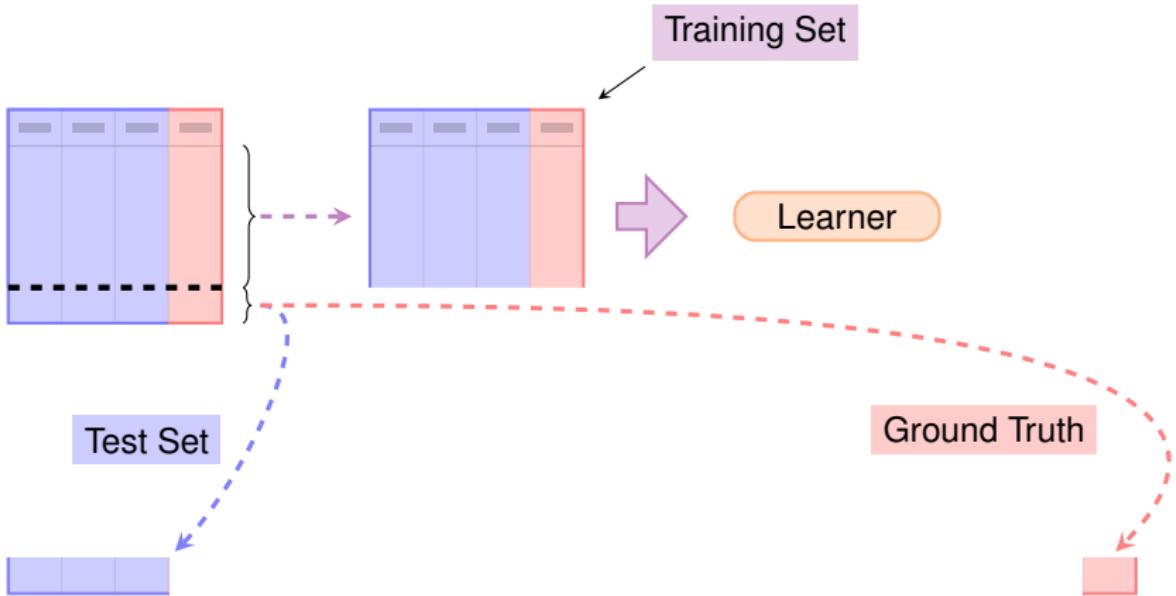
RESAMPLING



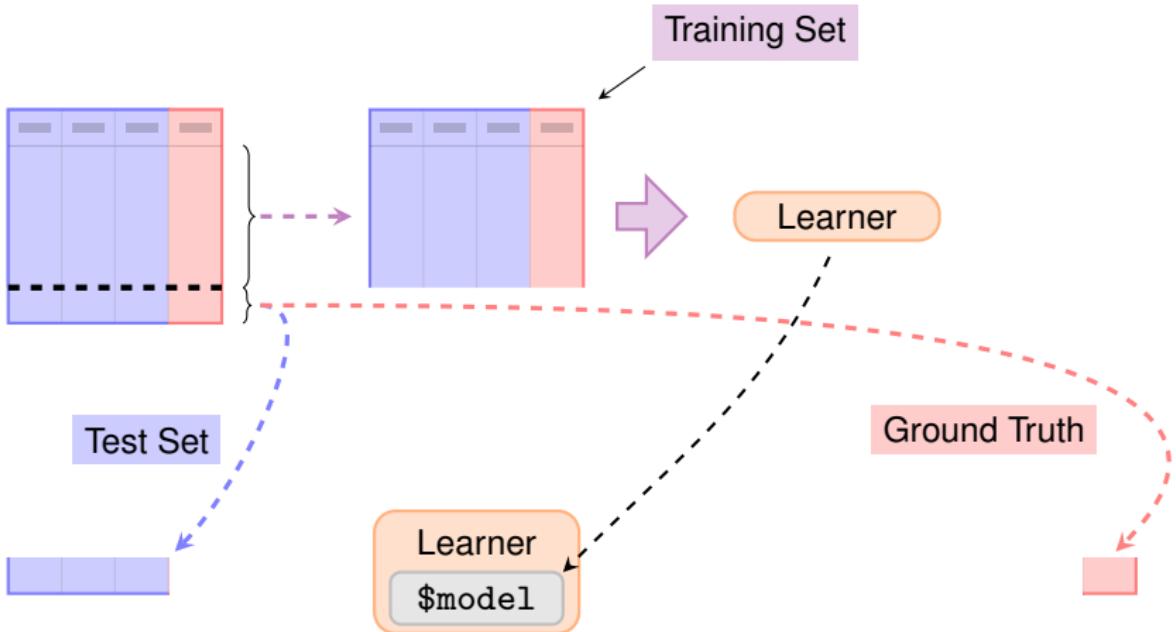
RESAMPLING



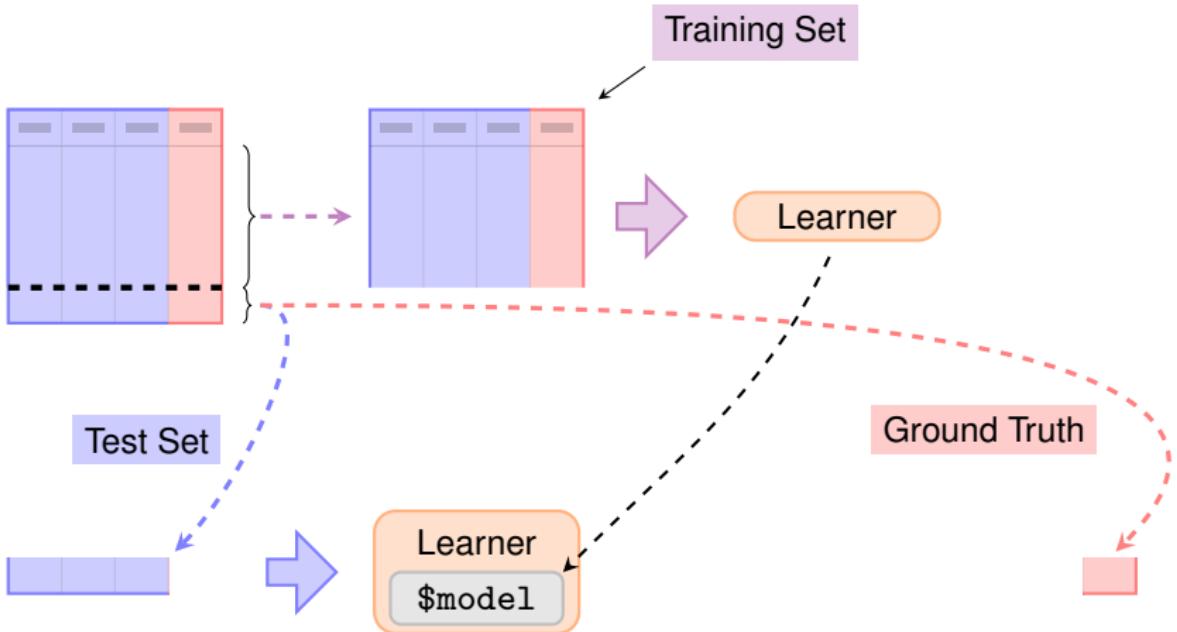
RESAMPLING



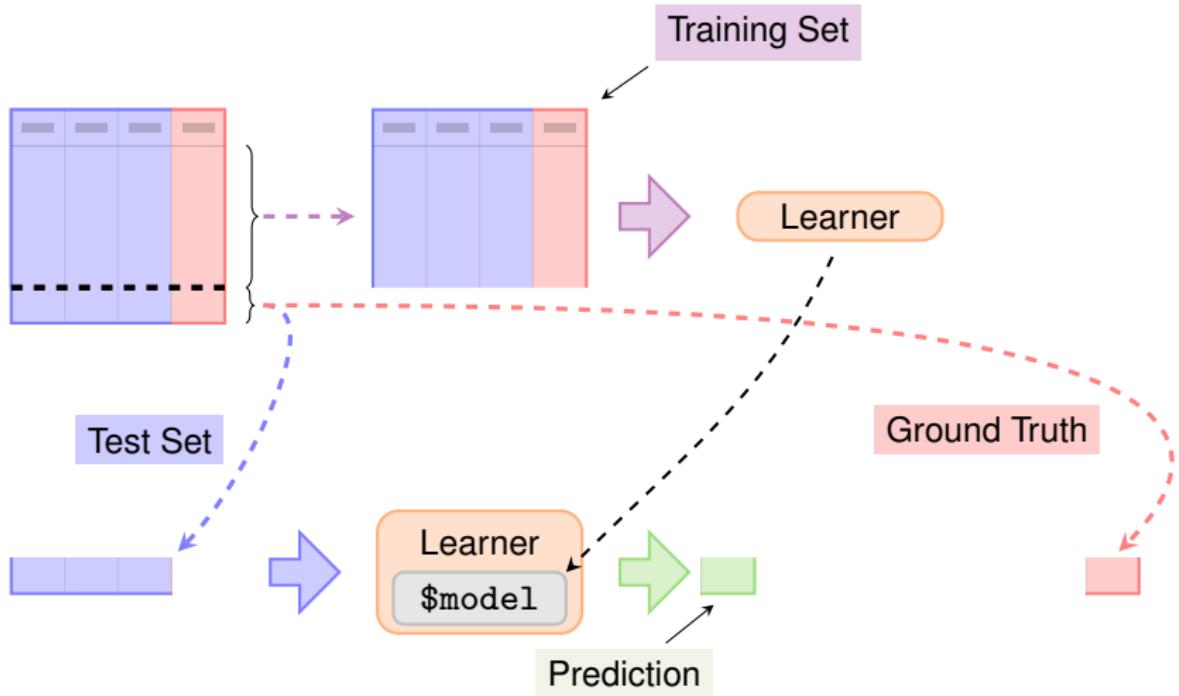
RESAMPLING



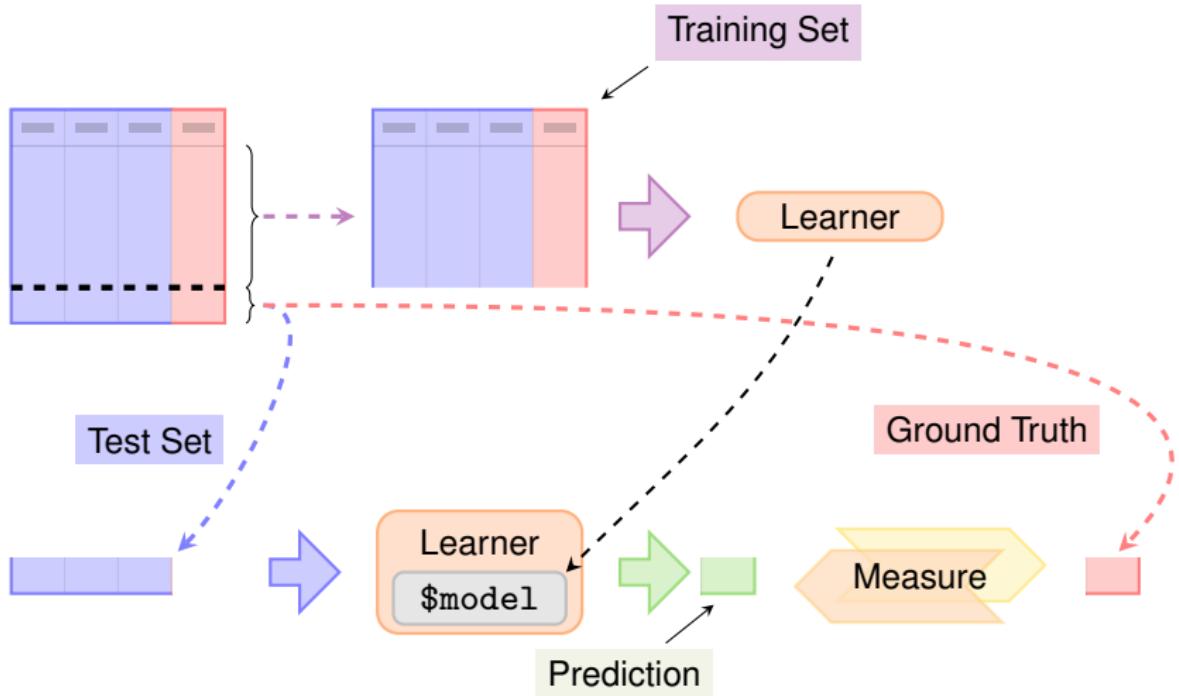
RESAMPLING



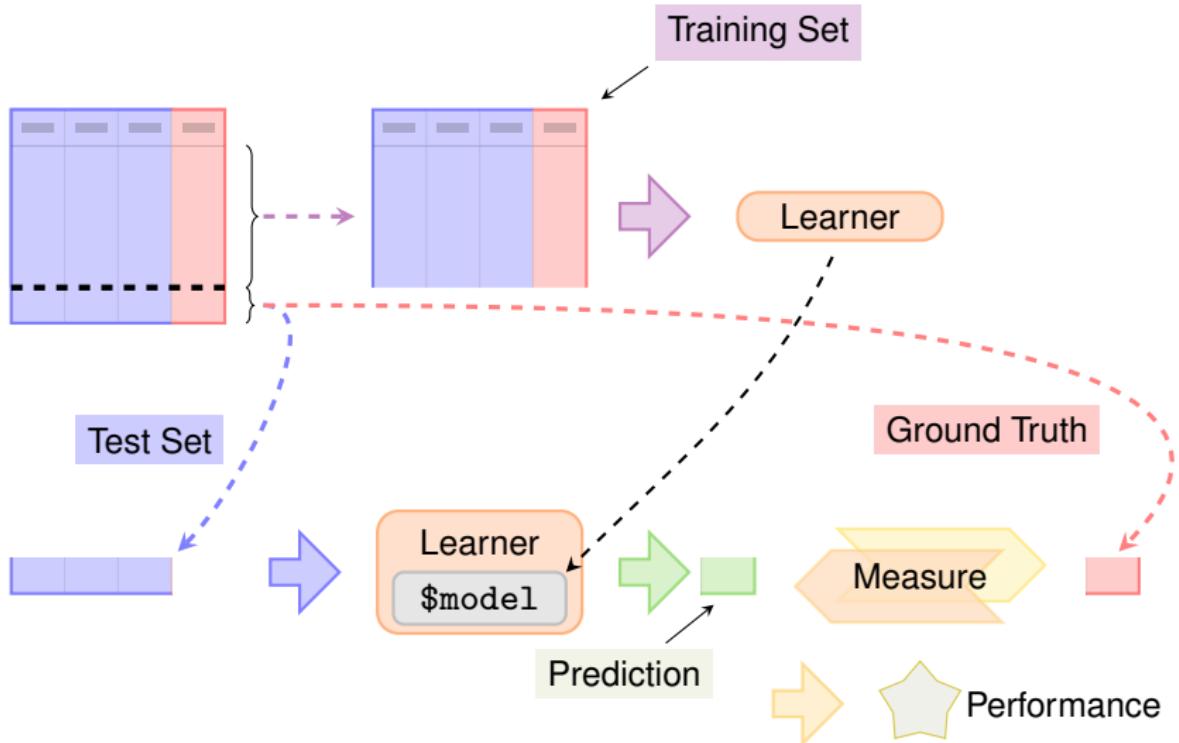
RESAMPLING



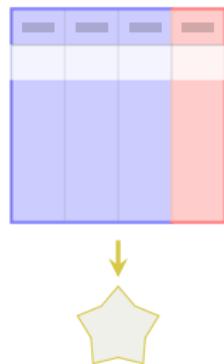
RESAMPLING



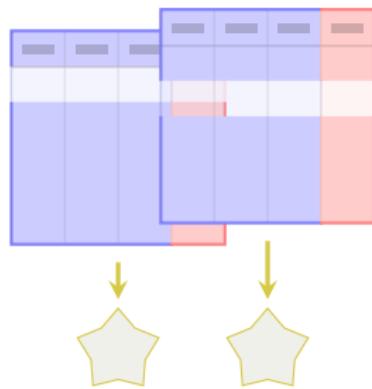
RESAMPLING



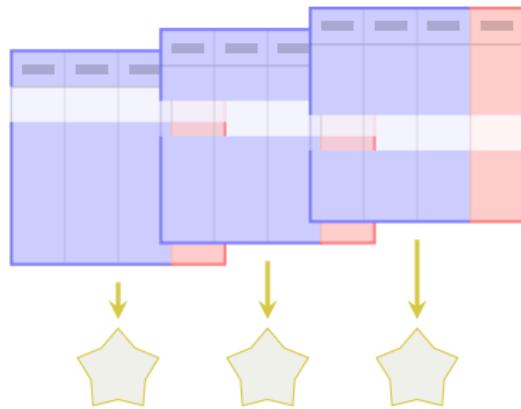
RESAMPLING



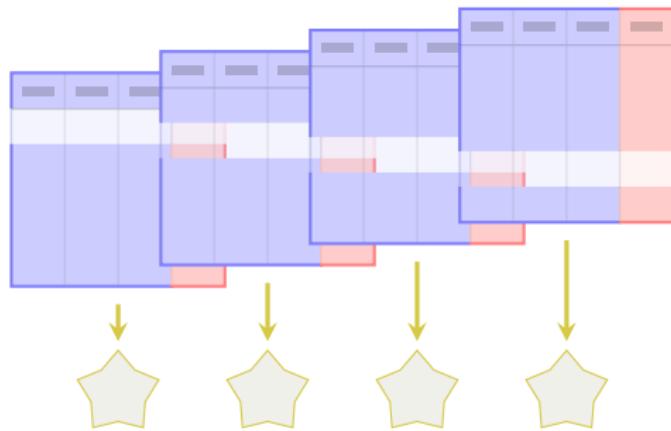
RESAMPLING



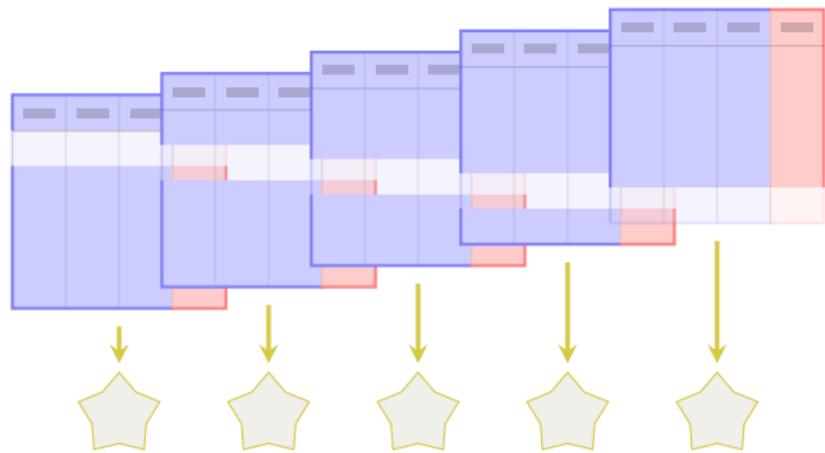
RESAMPLING



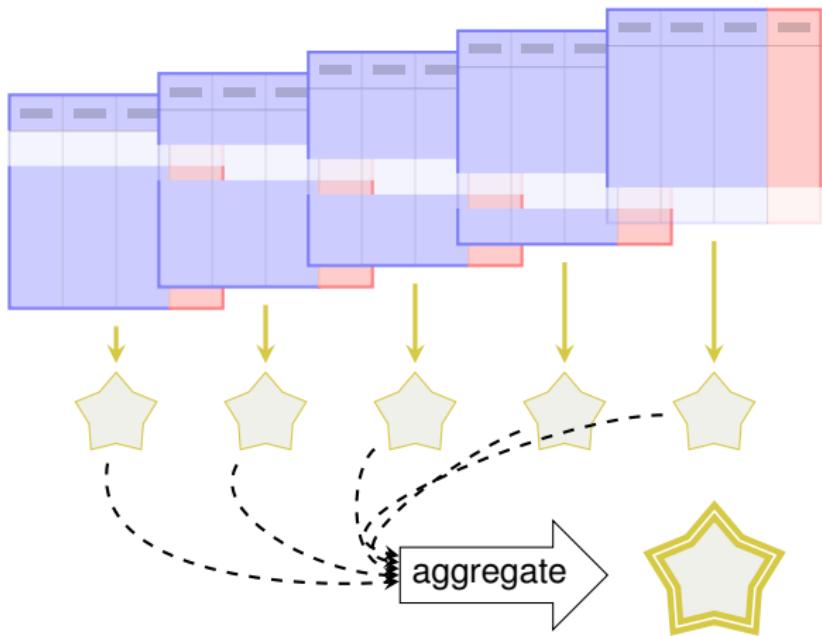
RESAMPLING



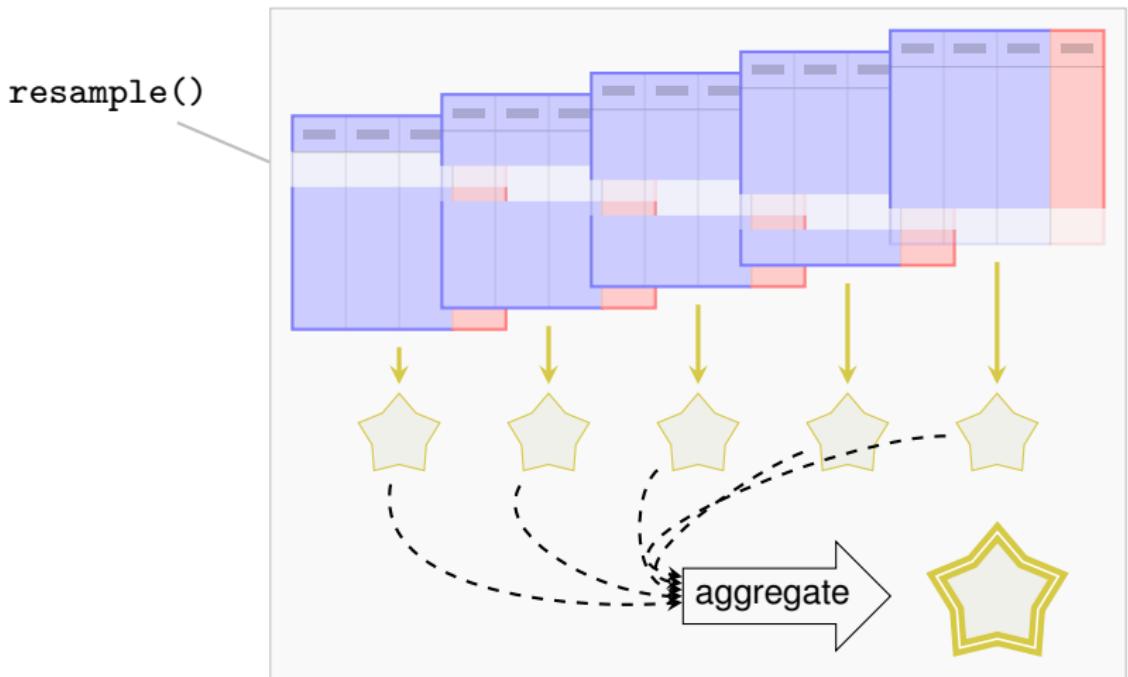
RESAMPLING



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

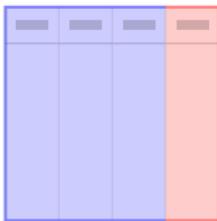
```
predictions = rr$predictions()  
predictions[[1]]  
  
#> <PredictionClassif> for 30 observations:  
#>   row_id   truth  response  
#>       3    setosa    setosa  
#>       8    setosa    setosa  
#>      10    setosa    setosa  
#>     ---  
#>      136 virginica virginica  
#>      140 virginica 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

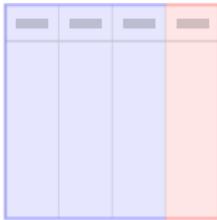
PERFORMANCE COMPARISON



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?

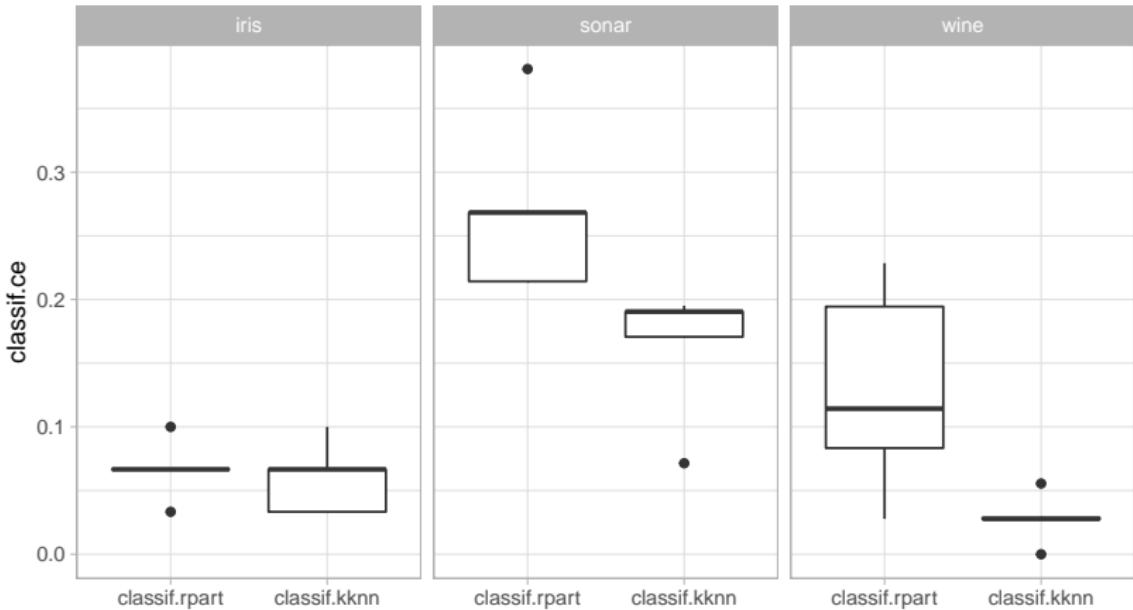
Just like `Prediction` and `ResamplingResult`!

- 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

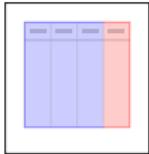
```
learner$help()
```

Outro

OVERVIEW

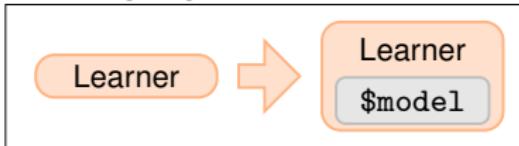
Ingredients:

Data



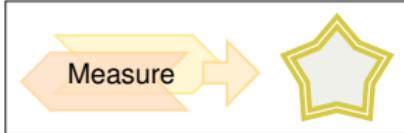
TaskClassif,
TaskRegr,
tsk()

Learning Algorithms



lrn() ⇒ Learner,
→ Learner\$train(),
→ Learner\$predict() ⇒ Prediction

Performance Evaluation



rsmp() ⇒ Resampling,
msr() ⇒ Measure,
resample() ⇒ ResamplingResult,
→ ResamplingResult\$score(),
→ ResamplingResult\$aggregate()

Performance Comparison



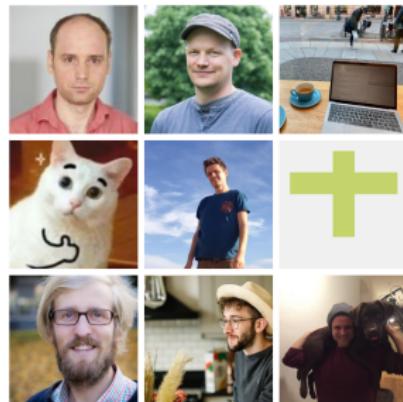
benchmark_grid(),
benchmark() ⇒ BenchmarkResult

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**

February 18, 2021

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

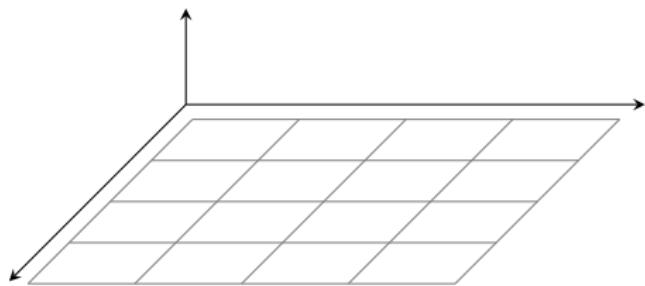
- Behavior of most methods depends on *hyperparameters*
 - We want to choose them so our algorithm performs well
 - Good hyperparameters are data-dependent
- ⇒ We do *black box optimization* (“Try stuff and see what works”)

Tuning toolbox for `mlr3`:

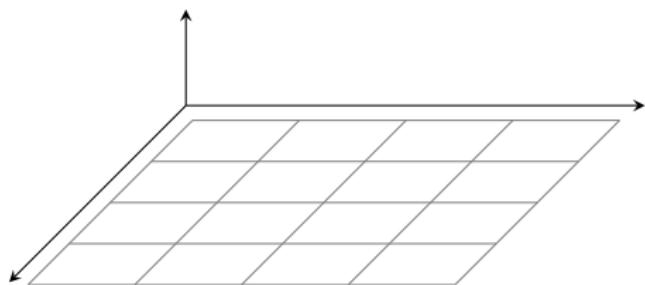
```
library("bbotk")
library("mlr3tuning")
```

Tuning

TUNING

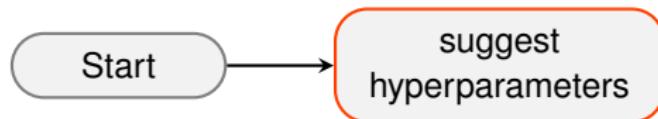
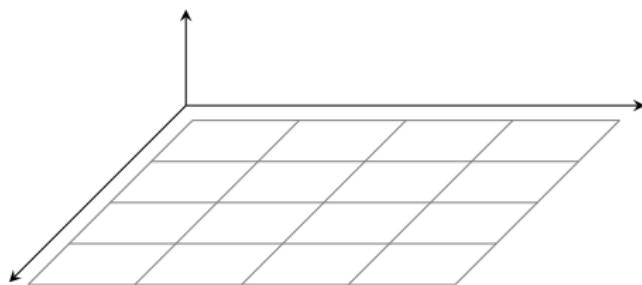


TUNING

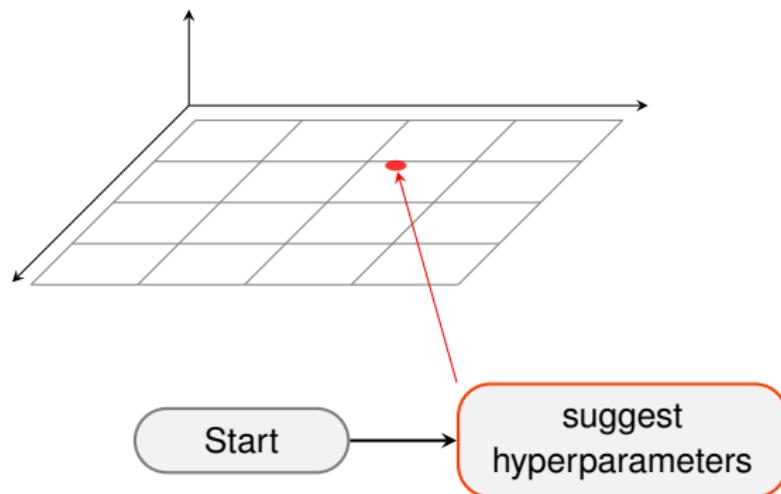


Start

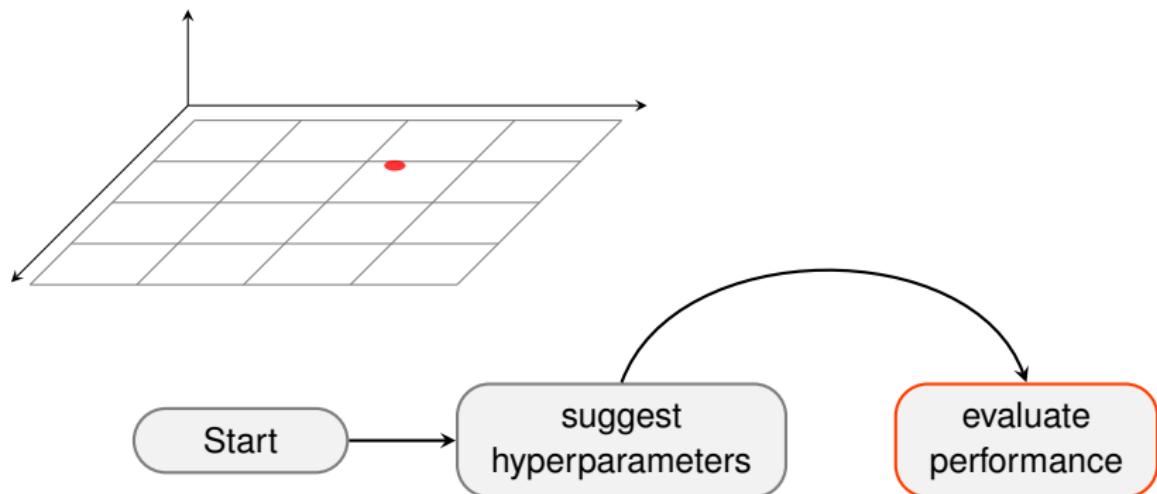
TUNING



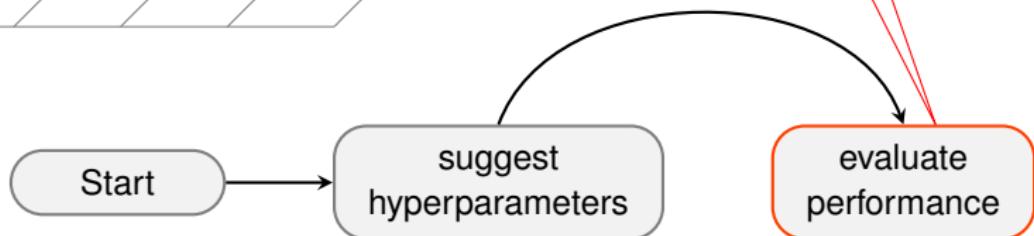
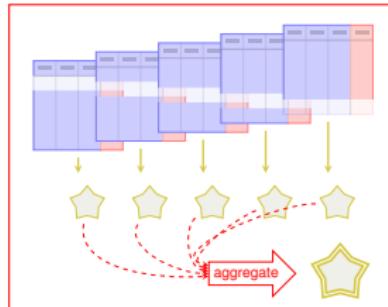
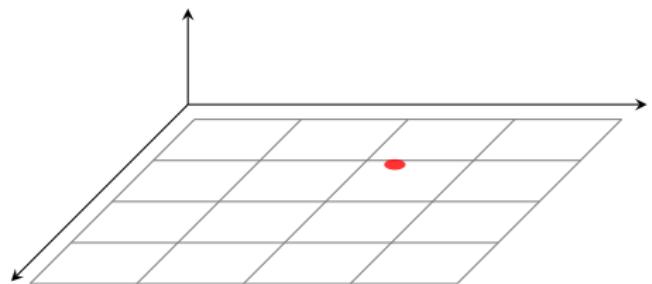
TUNING



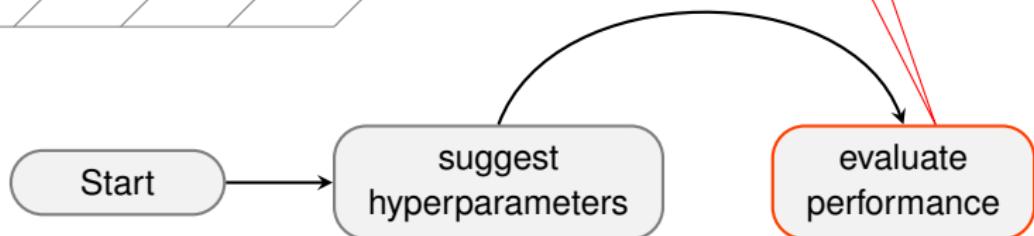
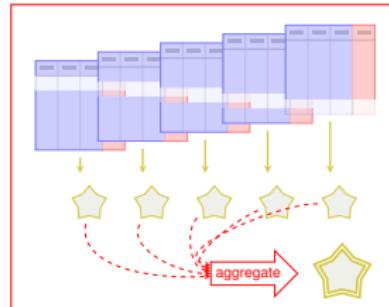
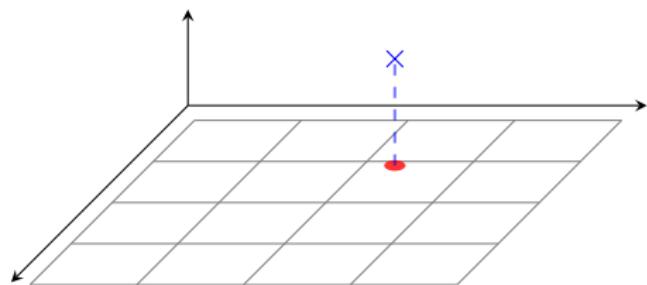
TUNING



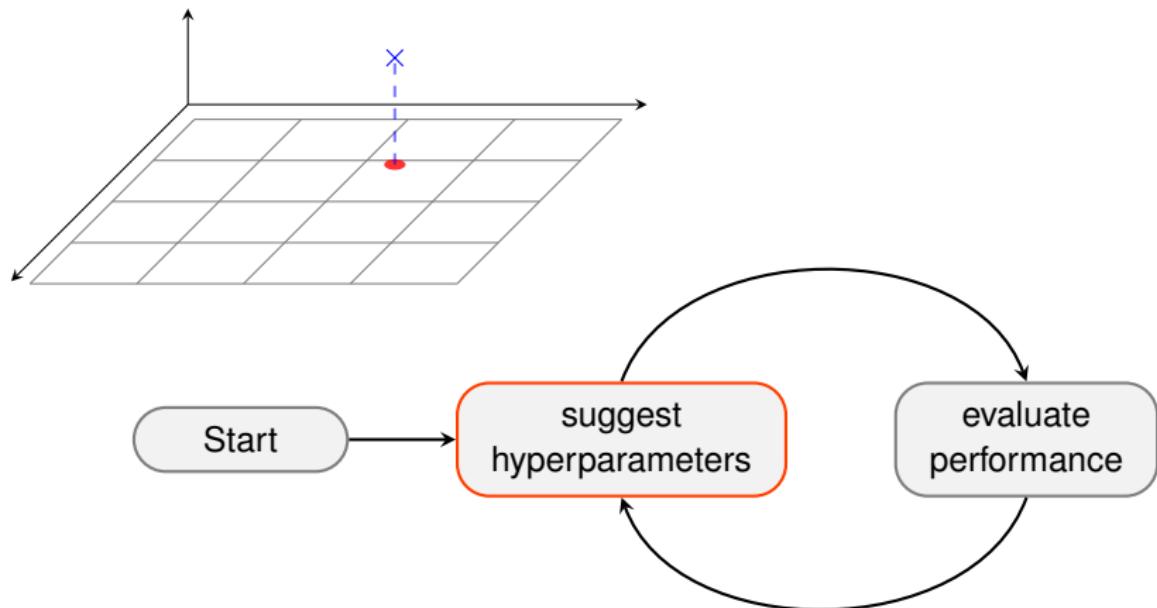
TUNING



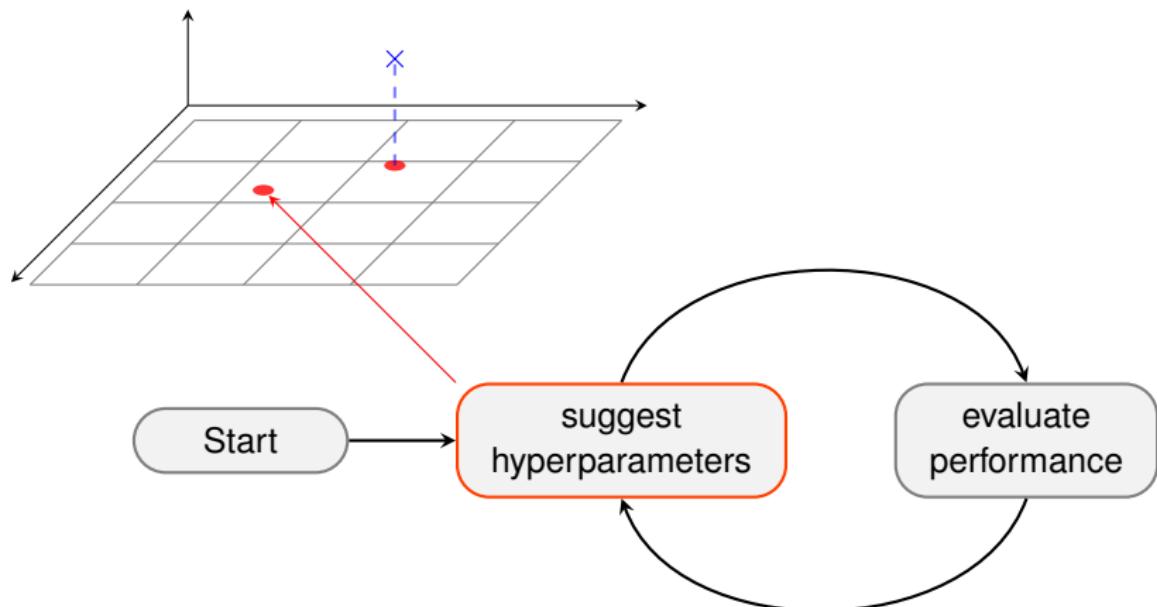
TUNING



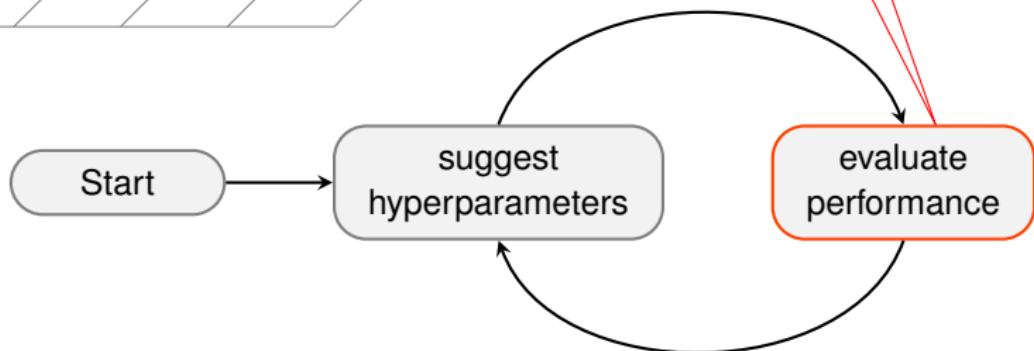
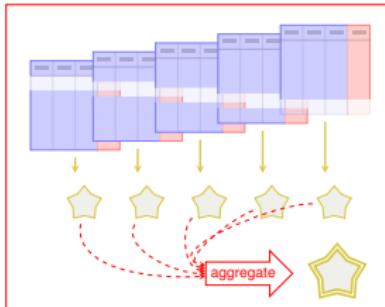
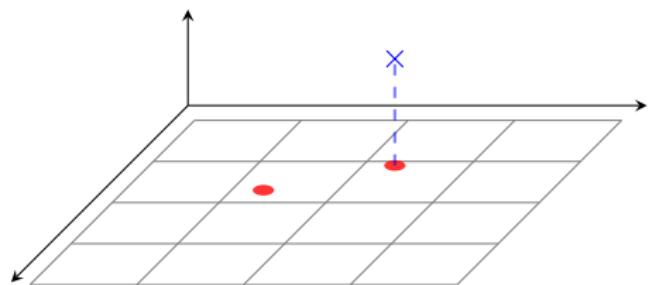
TUNING



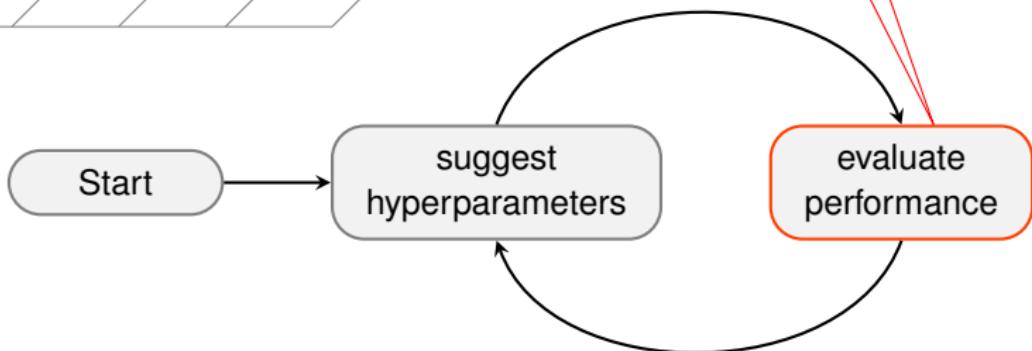
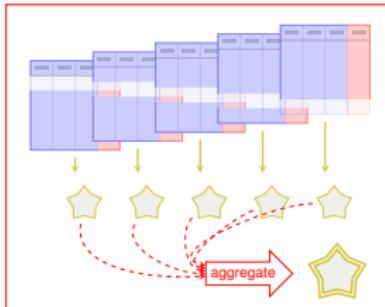
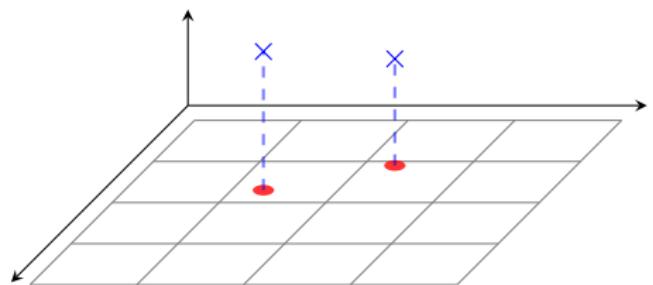
TUNING



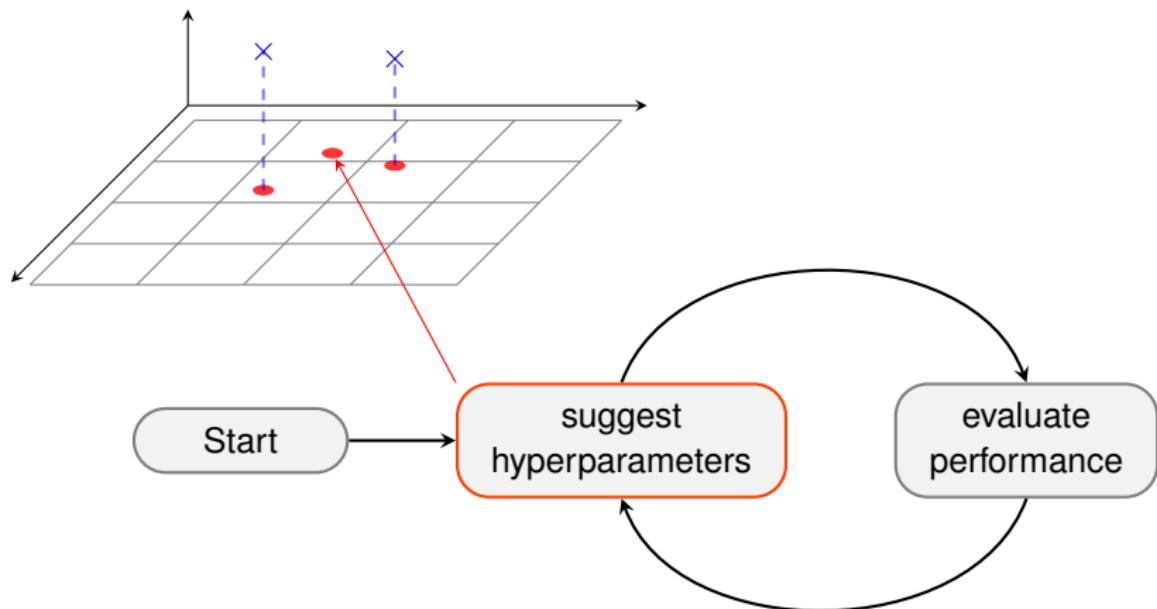
TUNING



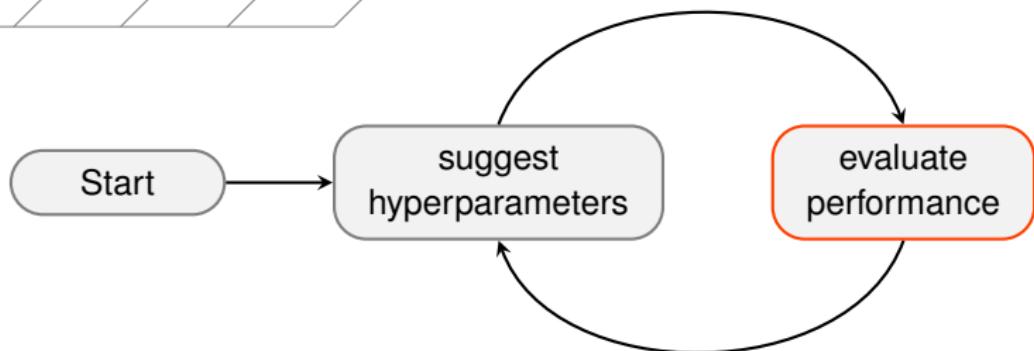
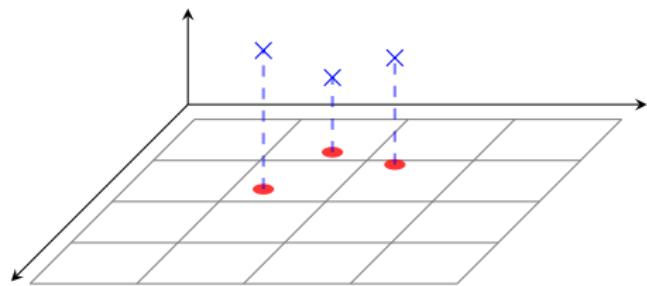
TUNING



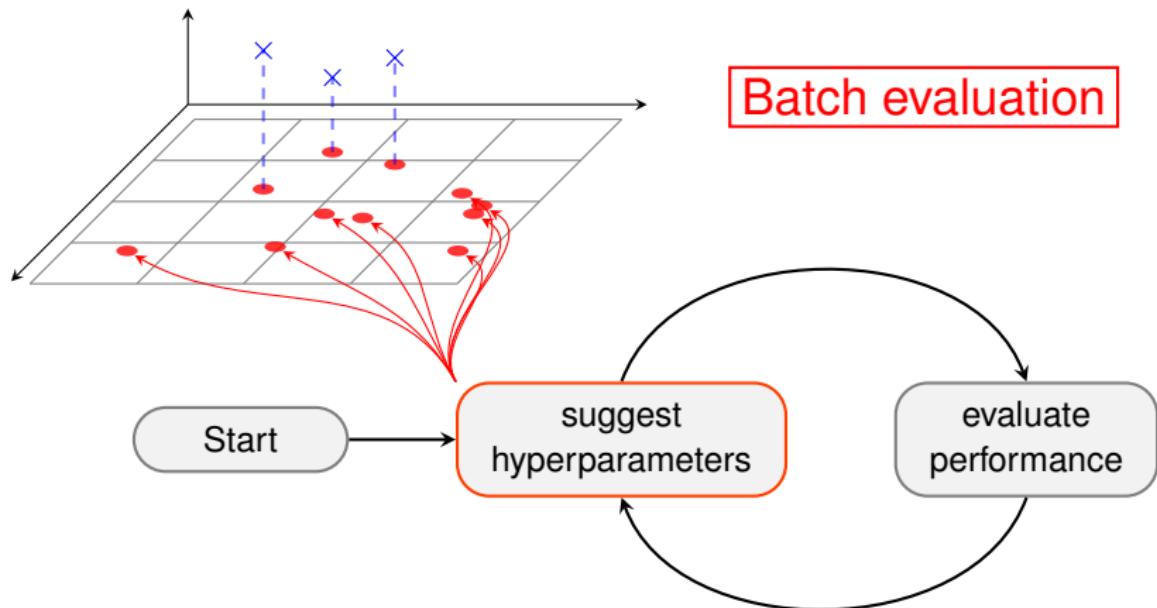
TUNING



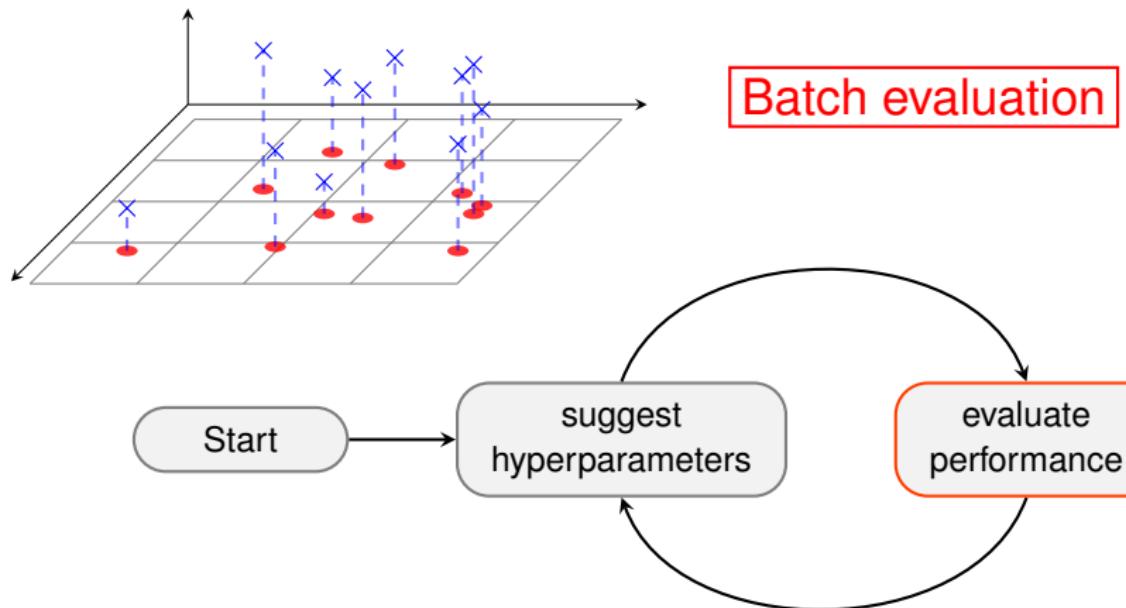
TUNING



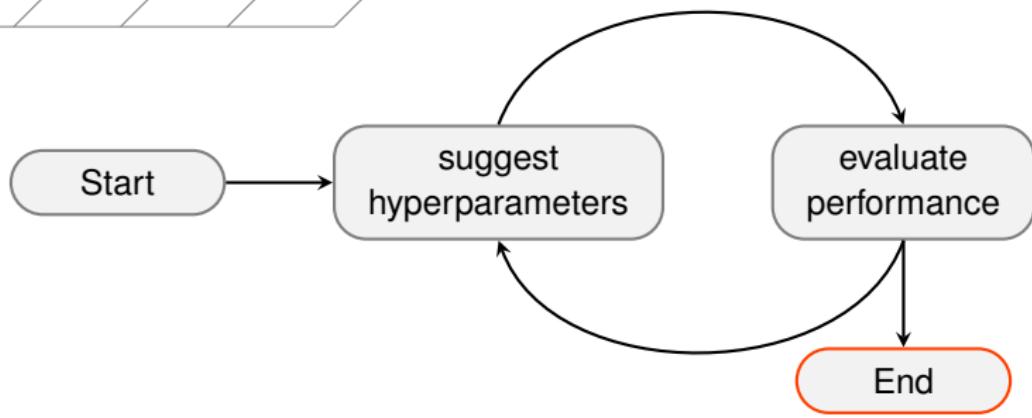
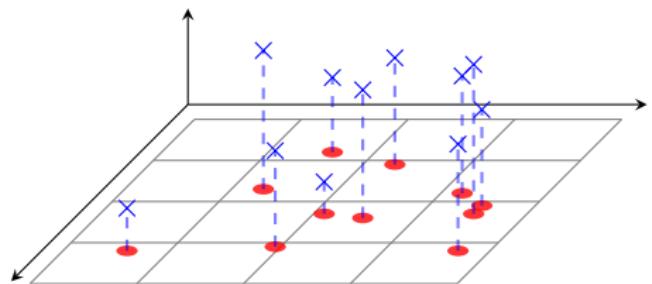
TUNING



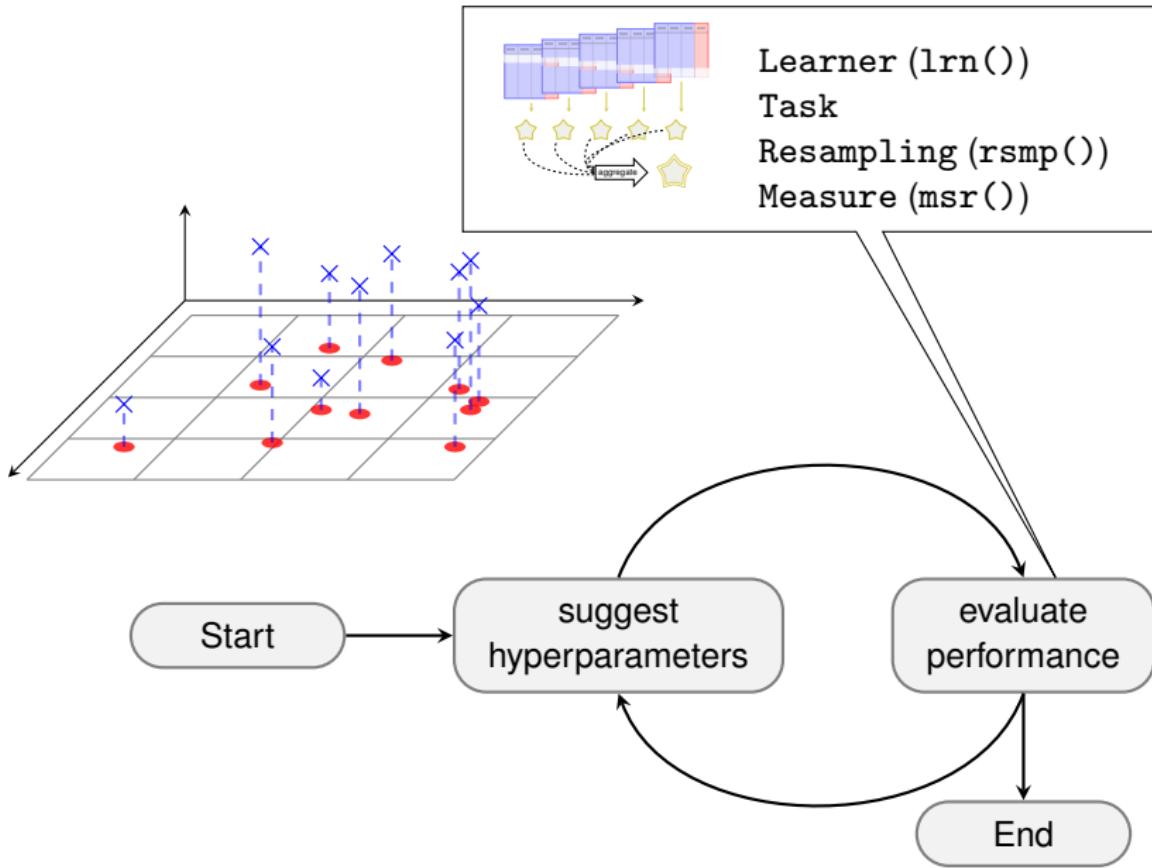
TUNING



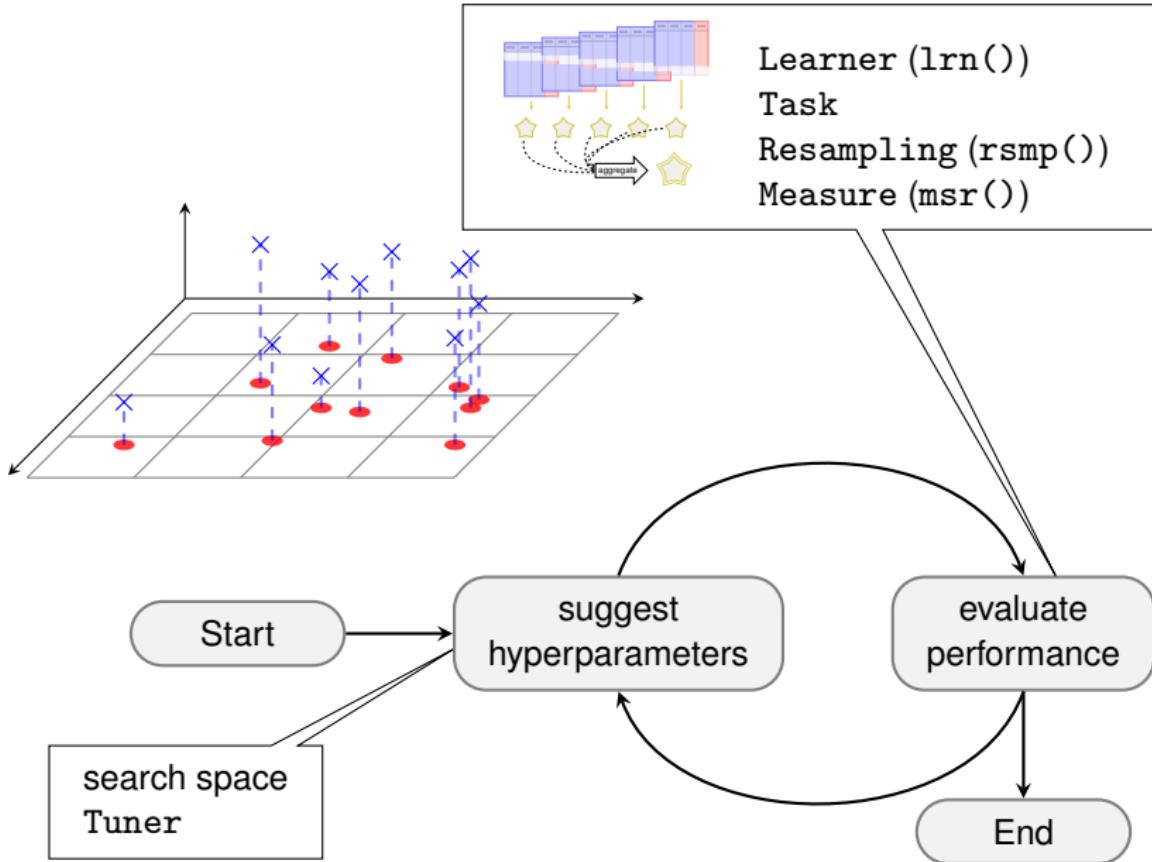
TUNING



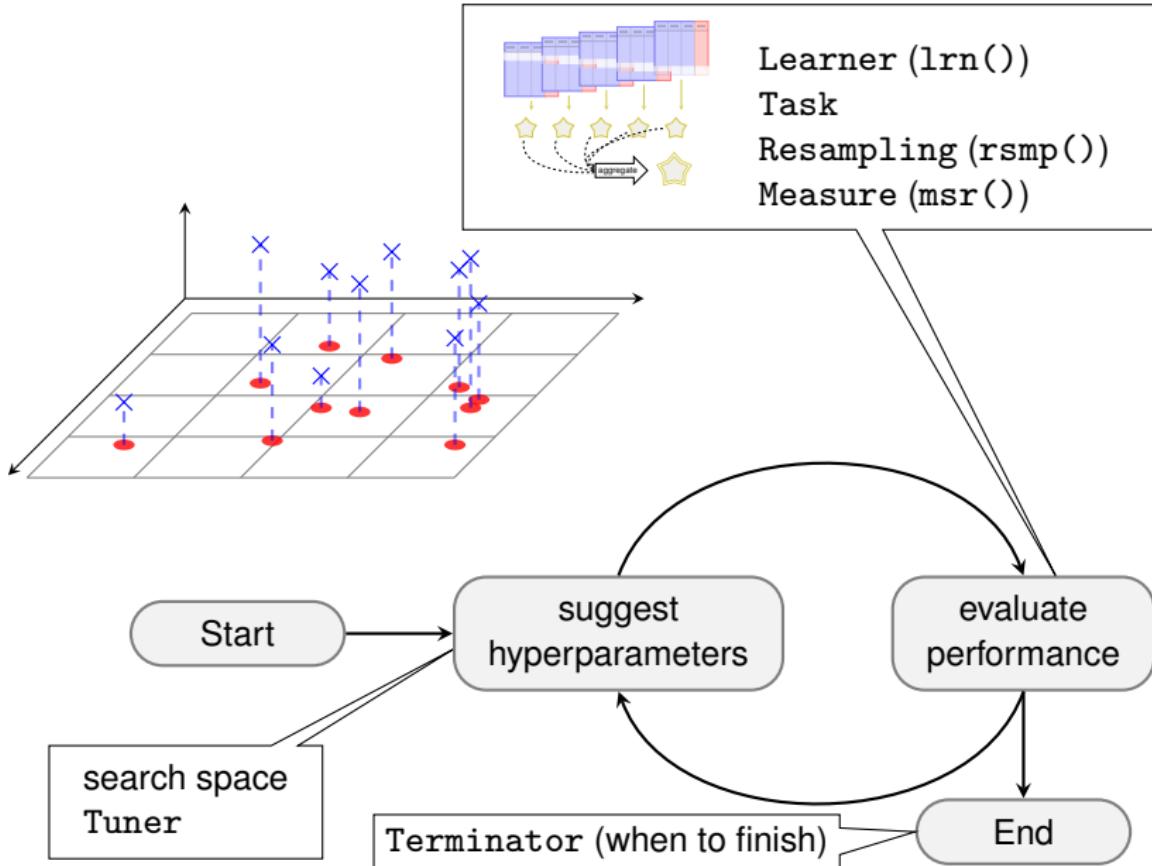
TUNING



TUNING

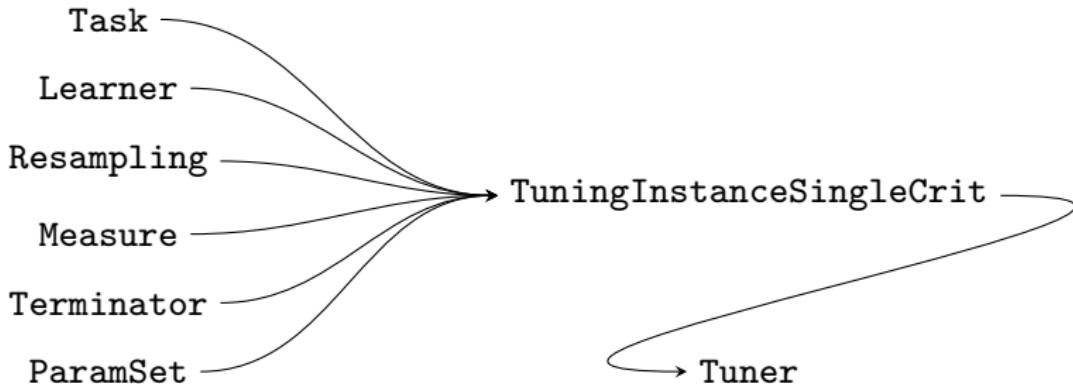


TUNING

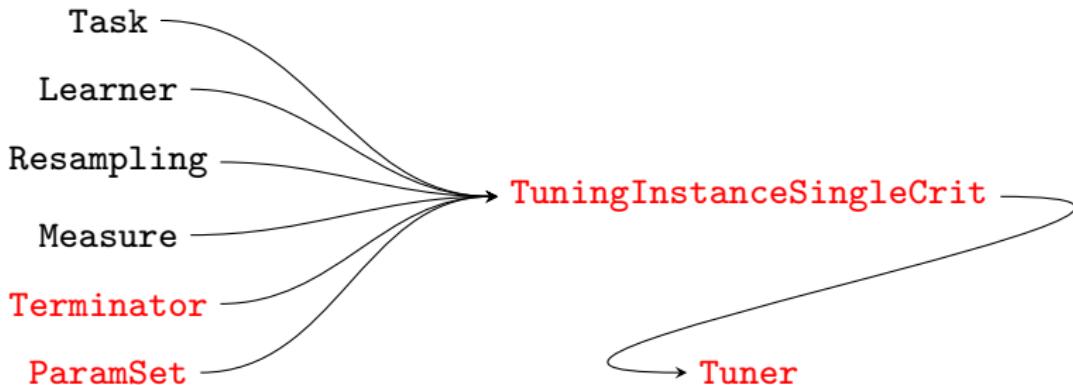


Tuning in mlr3

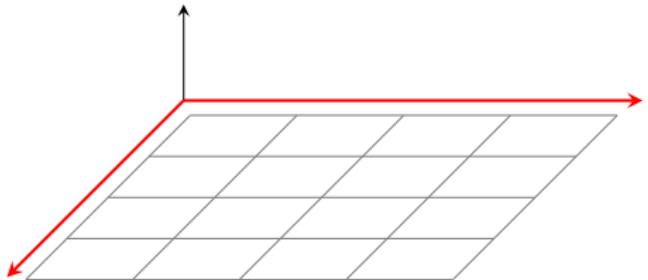
OBJECTS IN TUNING



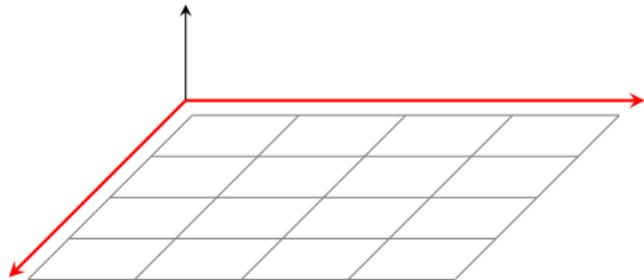
OBJECTS IN TUNING



SEARCH SPACE

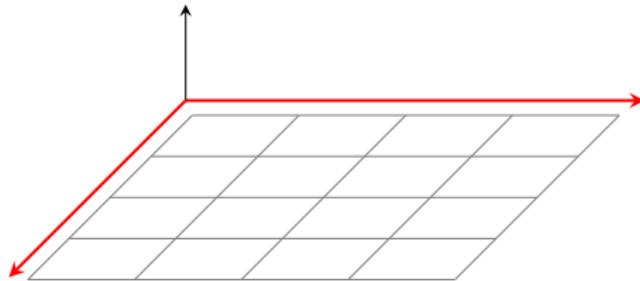


SEARCH SPACE



```
ParamSet$new(list(param1, param2, ...))
```

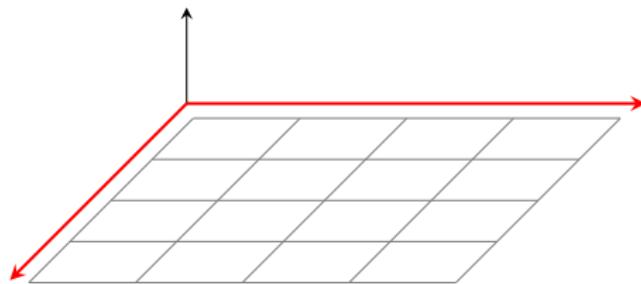
SEARCH SPACE



```
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

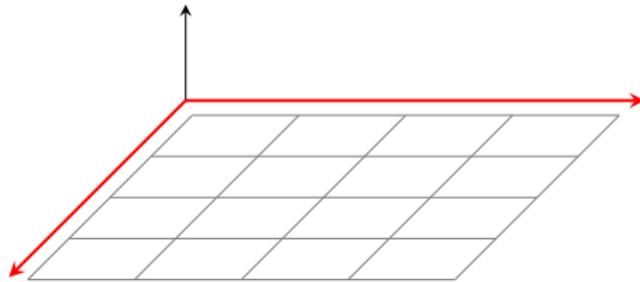


```
ParamSet$new(list(param1, param2, ...))
```

<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 `tunr()`, 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

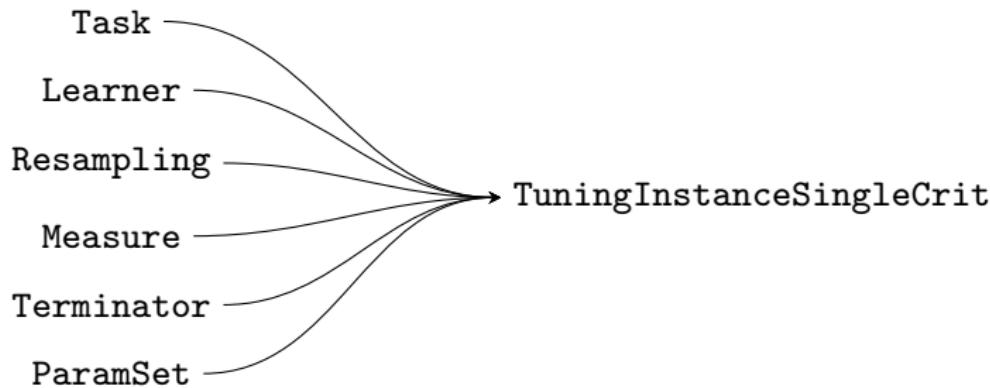
- load Tuner with `tnr()`, set parameters

- `gsearch = tnr("grid_search", resolution = 3)`

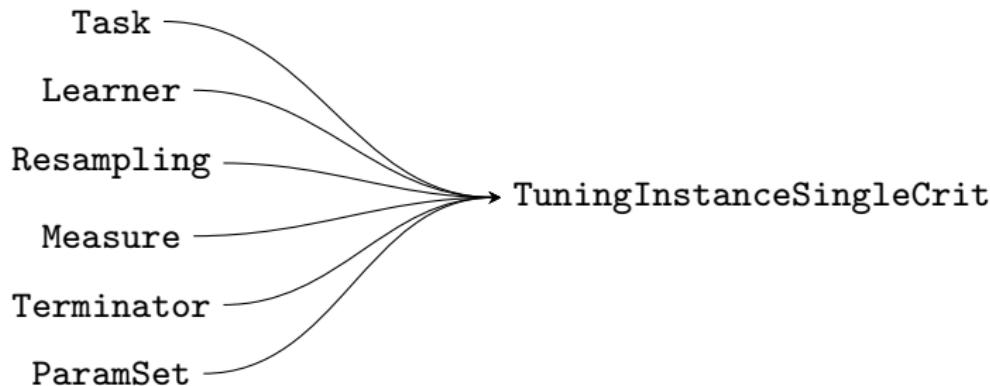
```
print(gsearch)
#> <TunerGridSearch>
#> * Parameters: resolution=3, batch_size=1
#> * Parameter classes: ParamLgl, ParamInt, ParamDbl, ParamFct
#> * Properties: dependencies, single-crit, multi-crit
#> * Packages: -
```

- 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

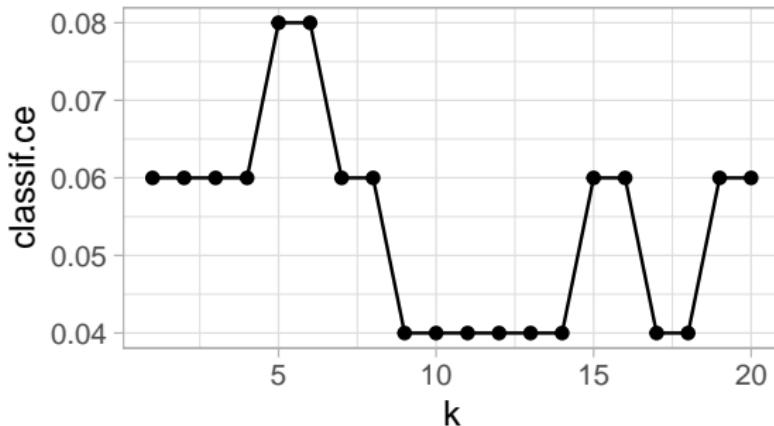
```
gsearch$optimize(inst)
#> INFO [14:35:12.185] [bbotk] Starting to optimize 1 parameter(s) with '<OptimizerGridSea
#> INFO [14:35:12.321] [bbotk] Evaluating 1 configuration(s)
#> INFO [14:35:13.781] [bbotk] Result of batch 1:
#> INFO [14:35:13.783] [bbotk]   k_classif.ce                               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)  
}
```

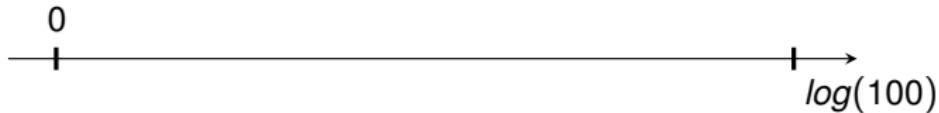
PARAMETER TRANSFORMATION

What is our transformation doing?



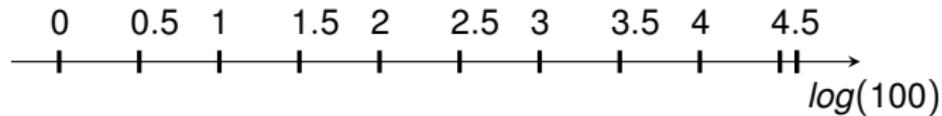
PARAMETER TRANSFORMATION

What is our transformation doing?



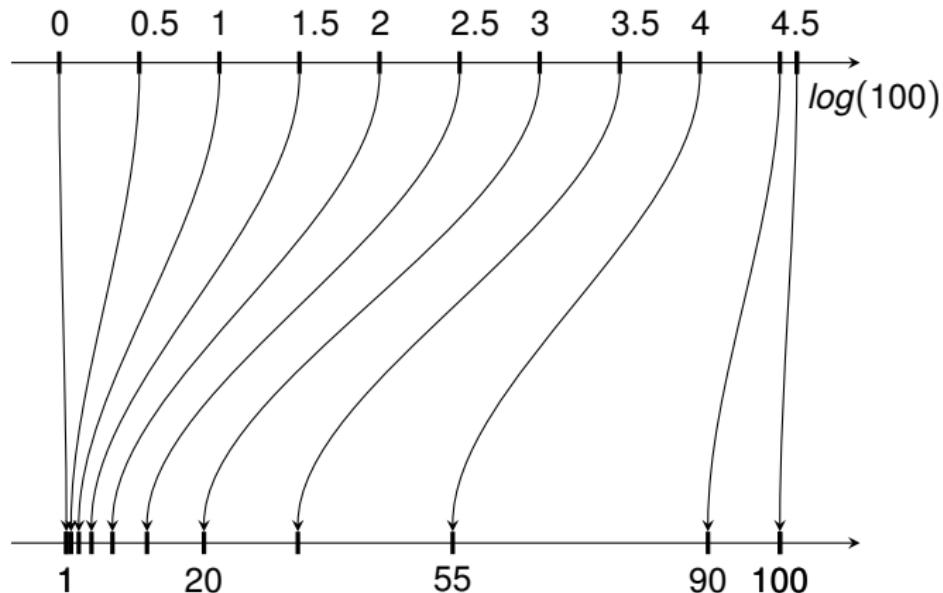
PARAMETER TRANSFORMATION

What is our transformation doing?



PARAMETER TRANSFORMATION

What is our transformation doing?



PARAMETER TRANSFORMATION

Tuning again...

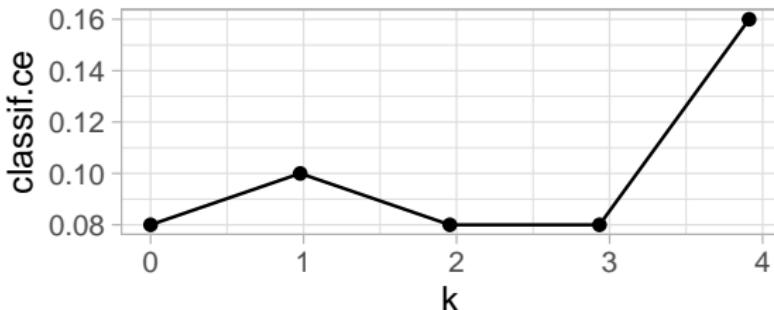
PARAMETER TRANSFORMATION

Tuning again...

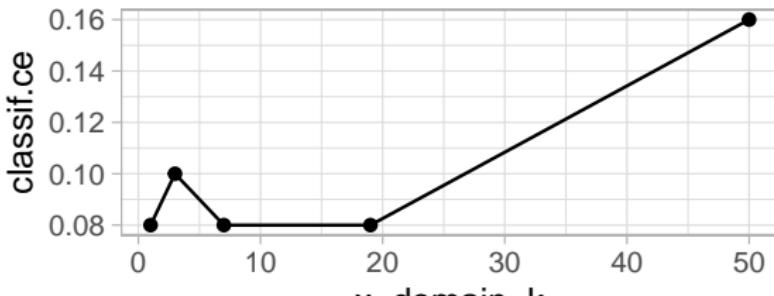
```
inst$result  
  
#>      k learner_param_vals  x_domain classif.ce  
#> 1: 2.9          <list[2]> <list[1]>      0.08  
  
inst$result$x_domain  
  
#> [[1]]  
#> [[1]]$k  
#> [1] 19
```

PARAMETER TRANSFORMATION

```
ggplot(as.data.table(inst$archive), aes(x = k, y = classif.ce)) +  
  geom_line() + geom_point()
```



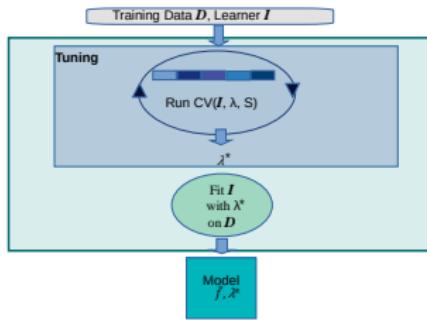
```
ggplot(as.data.table(inst$archive), aes(x = x_domain_k, y = classif.ce)) +  
  geom_line() + geom_point()
```



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

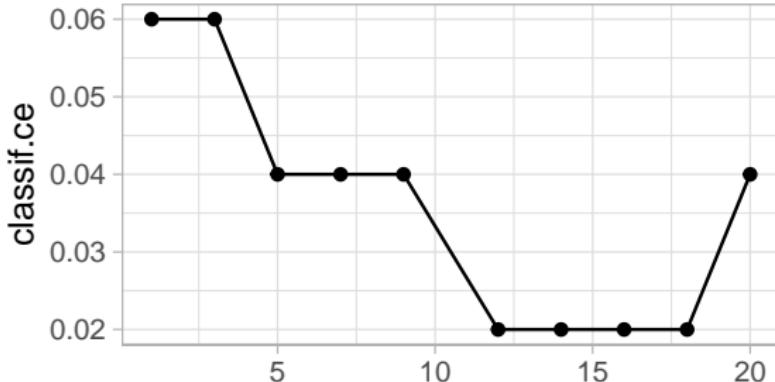
```
optlrn = AutoTuner$new(  
  learner = lrn("classif.kknn", kernel = "rectangular"),  
  resampling = rsmp("holdout"), measure = msr("classif.ce"),  
  terminator = trm("none"),  
  tuner = tnr("grid_search", resolution = 10),  
  search_space = searchspace_knn)
```

```
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
```

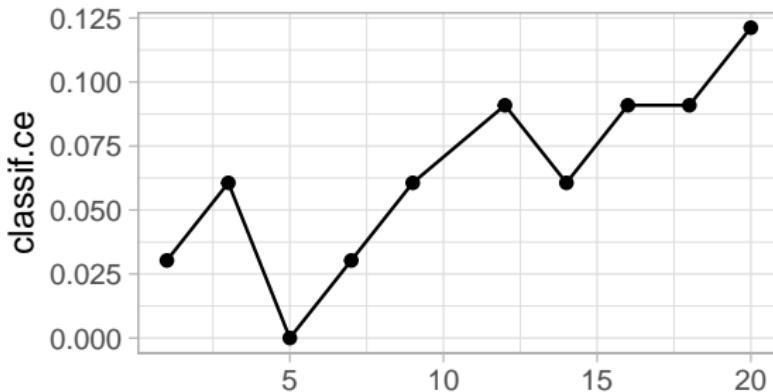
NESTED RESAMPLING

```
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("")
```

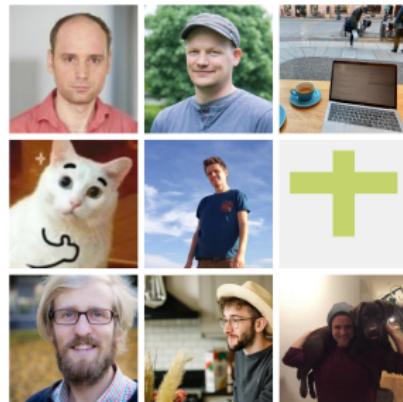


Machine Learning Pipelines in R



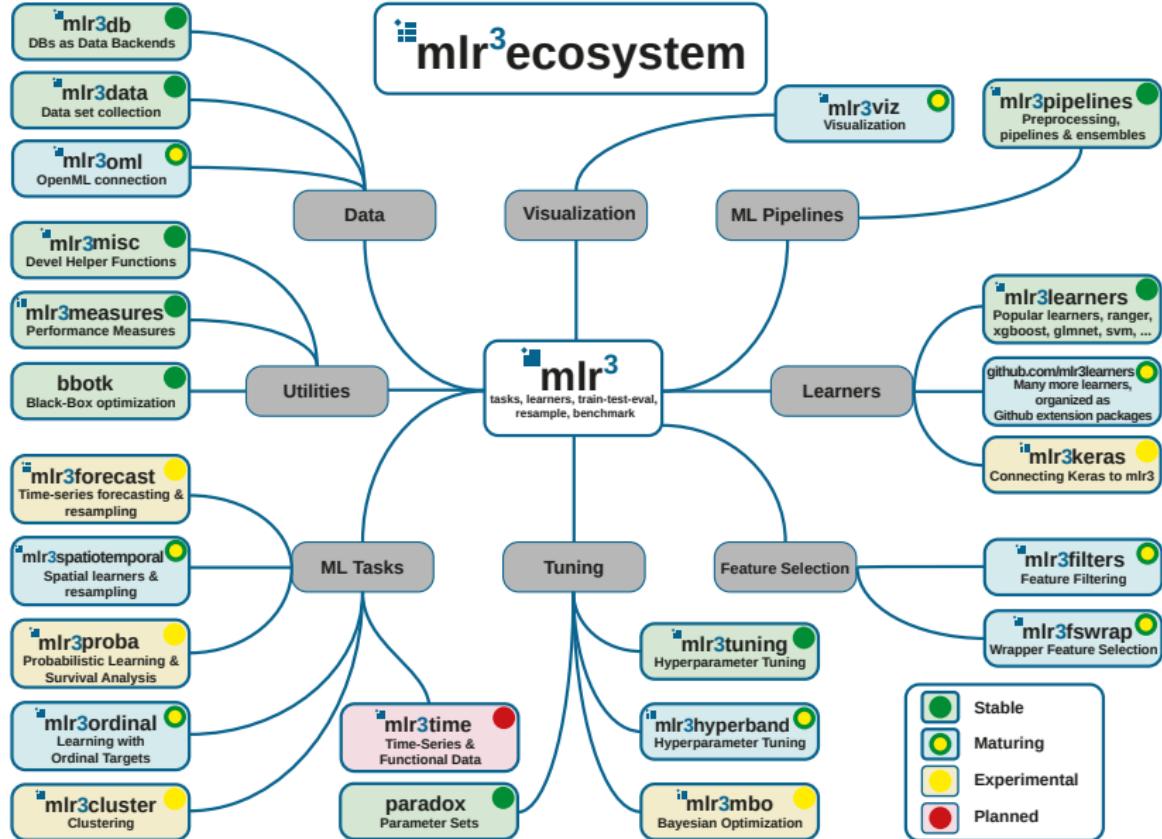
<https://mlr-org.com/>

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



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

February 11, 2021



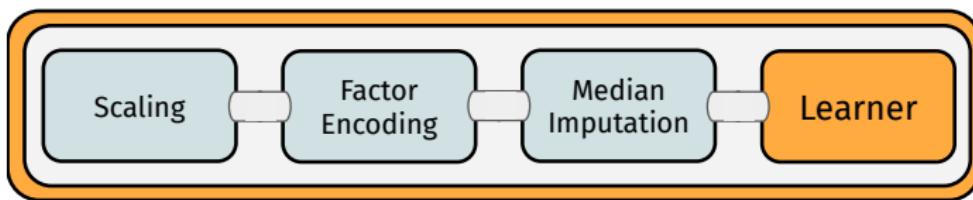
Intro

MLR3PIPELINES

Machine Learning Workflows:

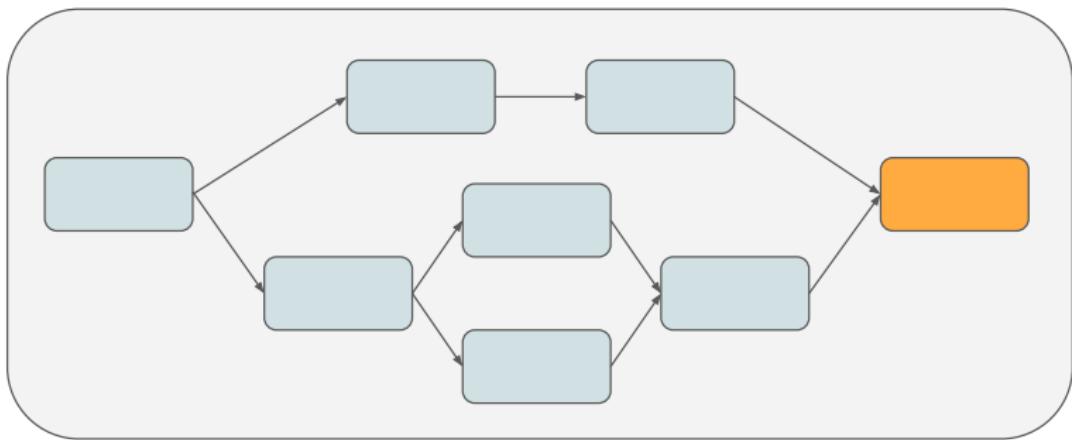
- **Preprocessing:** Feature extraction, feature selection, missing data imputation,...
- **Ensemble methods:** Model averaging, model stacking
- **mlr3:** modular model fitting

⇒ **mlr3pipelines:** modular ML workflows



MACHINE LEARNING WORKFLOWS

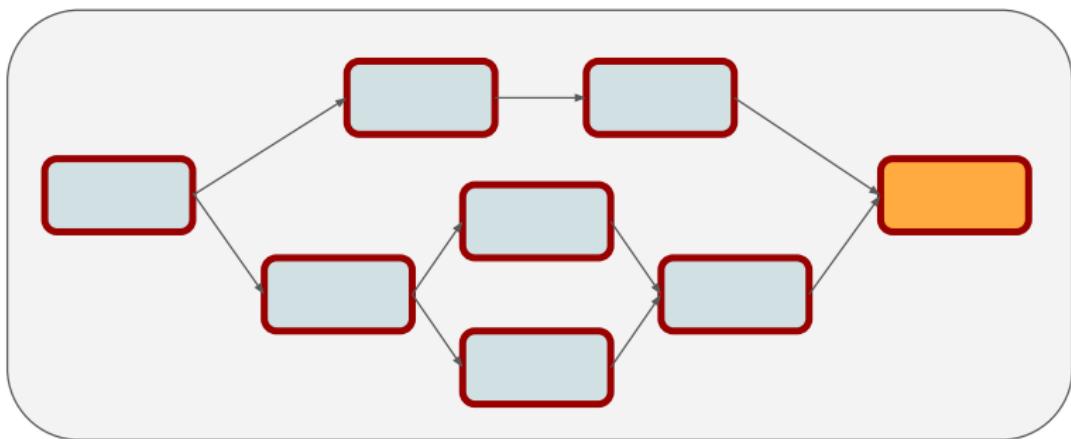
- what do they look like?



MACHINE LEARNING WORKFLOWS

– what do they look like?

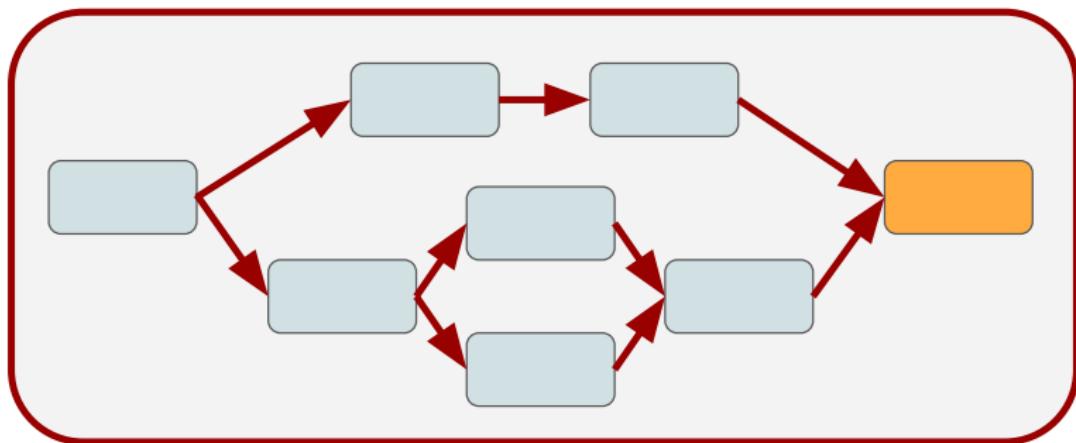
- **Building blocks:** *what is happening? → PipeOp*



MACHINE LEARNING WORKFLOWS

– what do they look like?

- **Building blocks:** *what is happening?* → PipeOp
- **Structure:** *in what sequence is it happening?* → Graph



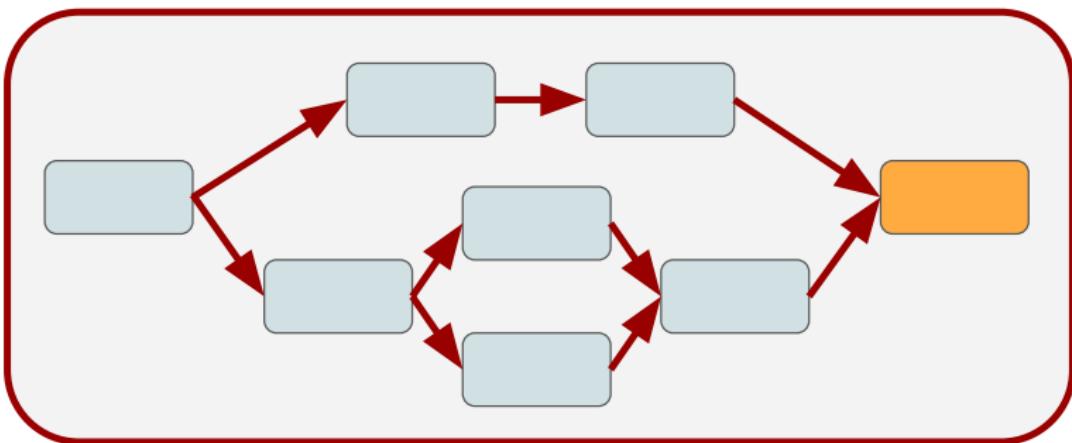
MACHINE LEARNING WORKFLOWS

– 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

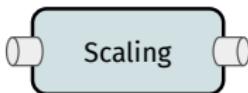


PipeOps

THE BUILDING BLOCKS

PipeOp: Single Unit of Data Operation

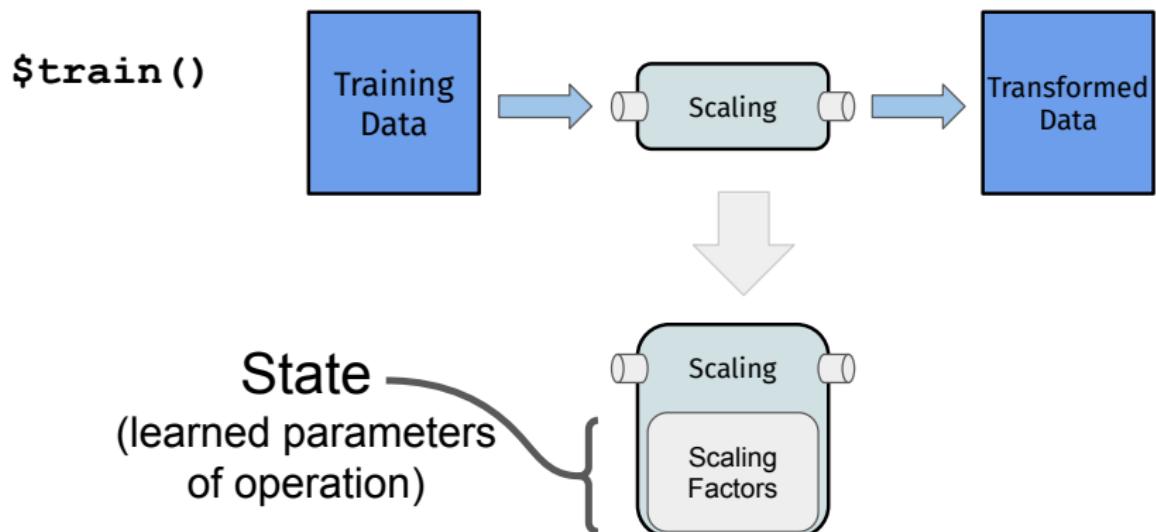
- `pip = po("scale")` to construct



THE BUILDING BLOCKS

PipeOp: Single Unit of Data Operation

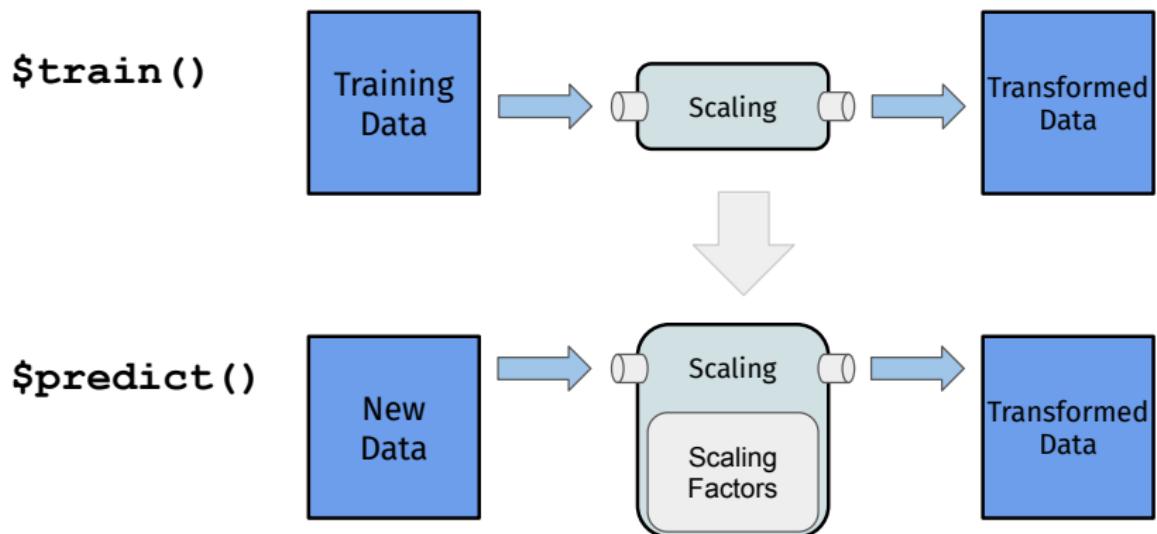
- `pip = po("scale")` to construct
- `pip$train()`: process data and create `pip$state`



THE BUILDING BLOCKS

PipeOp: Single Unit of Data Operation

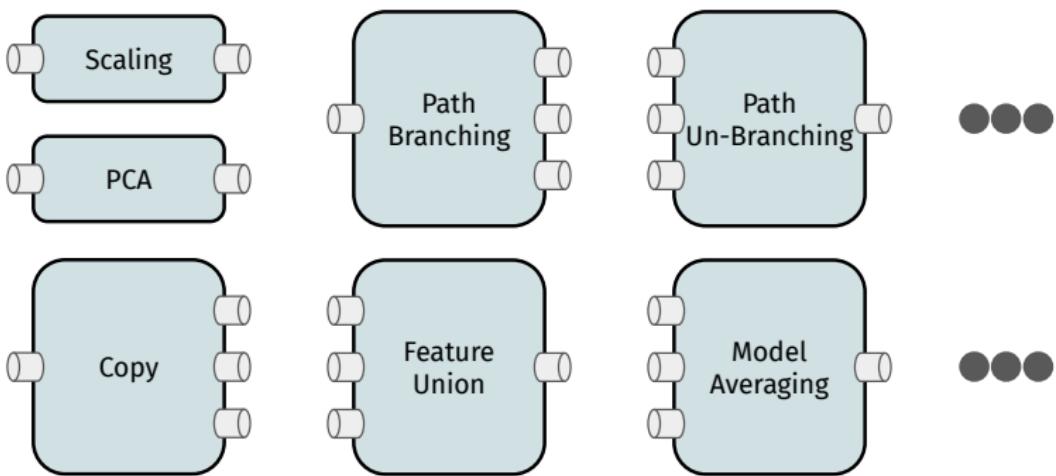
- `pip = po("scale")` to construct
- `pip$train()`: process data and create `pip$state`
- `pip$predict()`: process data depending on the `pip$state`



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"
#> [...]
```

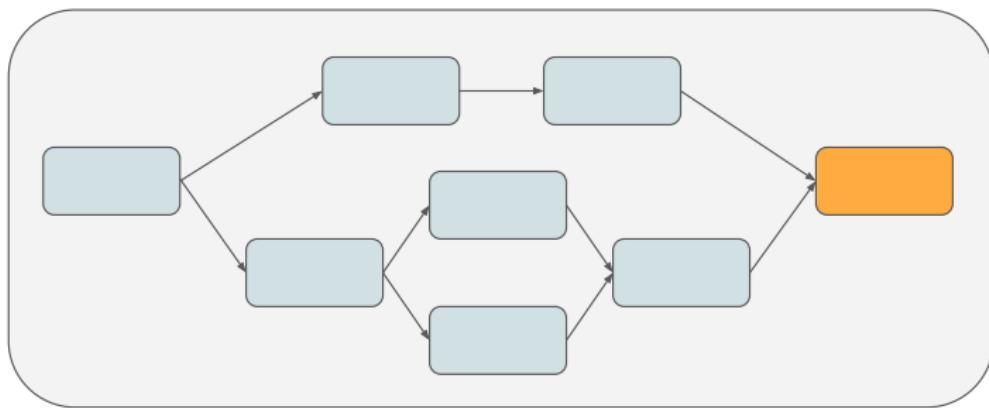
PIPEOPS SO FAR AND PLANNED

- 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

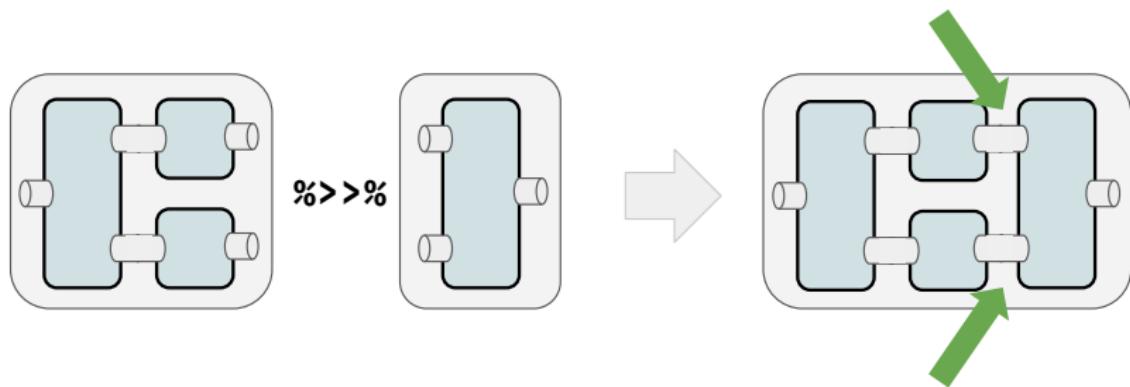
Graph Operations



THE STRUCTURE

Graph Operations

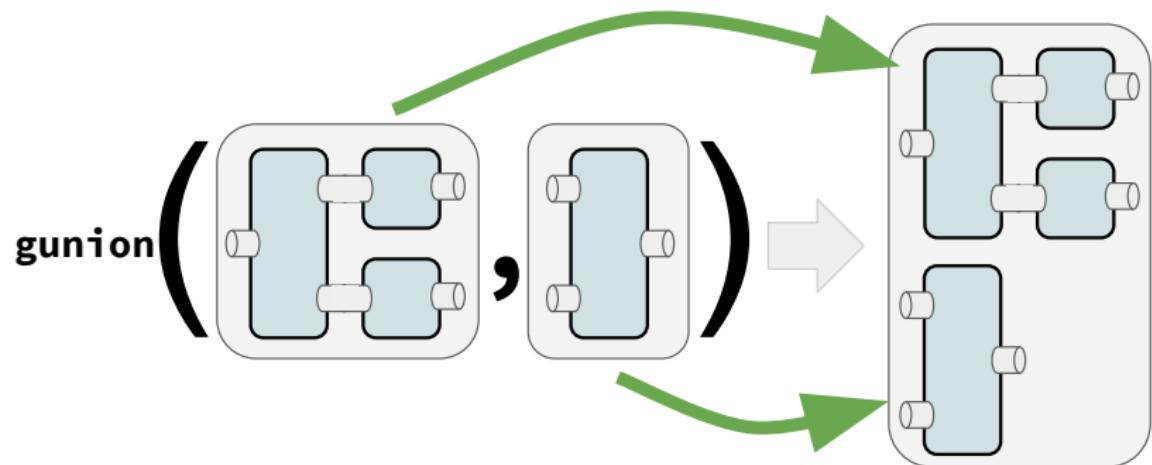
- The `%>>%`-operator concatenates Graphs and PipeOps



THE STRUCTURE

Graph Operations

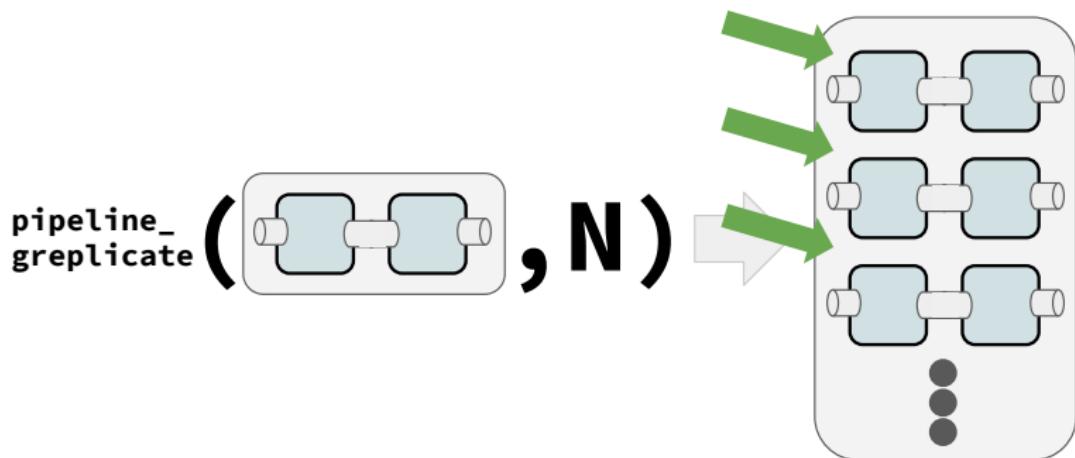
- The `%>>%`-operator concatenates Graphs and PipeOps
- The `gunion()`-function unites Graphs and PipeOps



THE STRUCTURE

Graph Operations

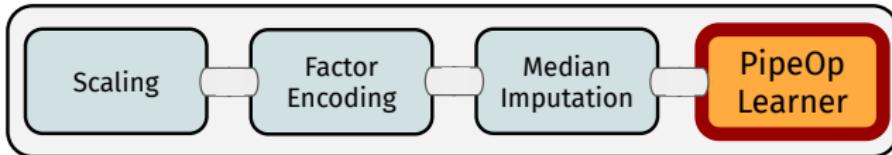
- The `%>>%`-operator concatenates Graphs and PipeOps
- The `gunion()`-function unites Graphs and PipeOps
- The `pipeline_greplicate()`-function unites copies of Graphs and PipeOps



LEARNERS AND GRAPHS

PipeOpLearner

- Learner as a PipeOp
- Fits a model, output is Prediction



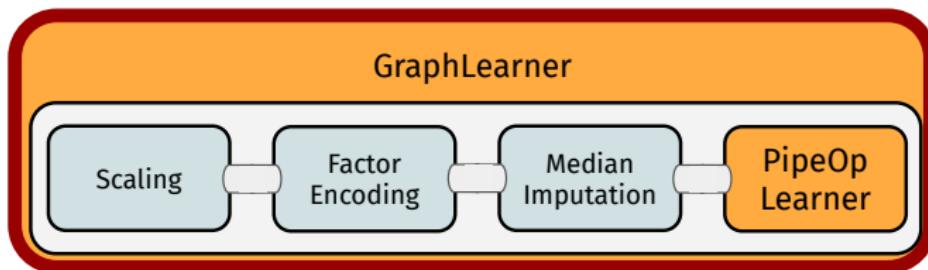
LEARNERS AND GRAPHS

PipeOpLearner

- Learner as a PipeOp
- Fits a model, output is Prediction

GraphLearner

- Graph as a Learner
- All benefits of mlr3: **resampling, tuning, nested resampling, ...**

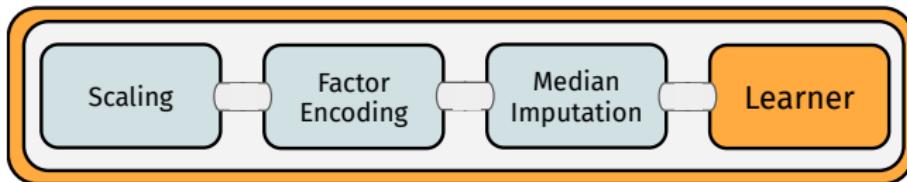


Linear Pipelines

MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline

```
graph_pp = po("scale") %>>%  
  po("encode") %>>%  
  po("imputemedian") %>>%  
  lrn("classif.rpart")
```

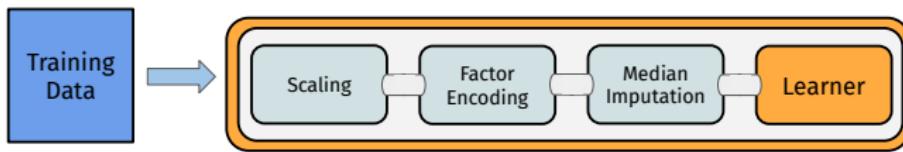


MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline

- `train()`ing: Data propagates and creates \$states

```
glrn = GraphLearner$new(graph_pp)  
glrn$train(task)
```

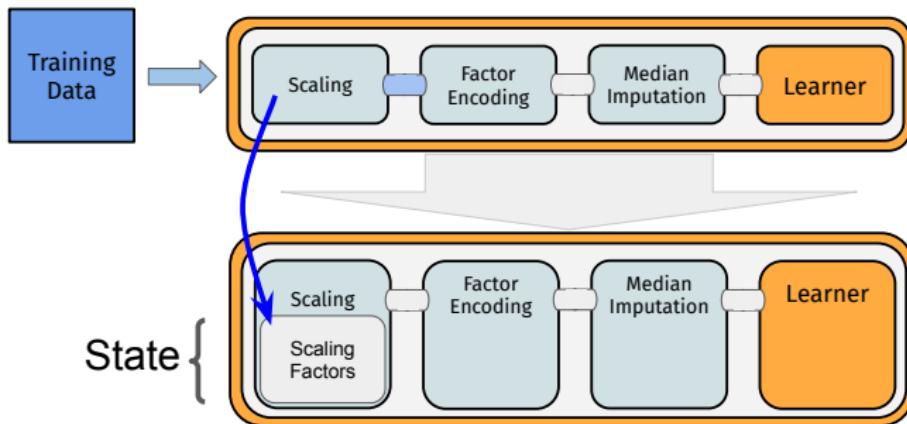


MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline

- `train()`ing: Data propagates and creates \$states

```
glrn = GraphLearner$new(graph_pp)  
glrn$train(task)
```

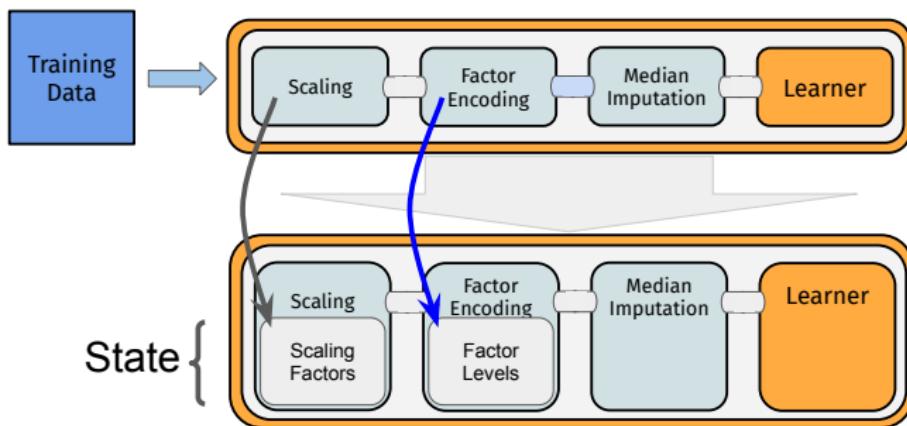


MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline

- `train()`ing: Data propagates and creates \$states

```
glrn = GraphLearner$new(graph_pp)  
glrn$train(task)
```

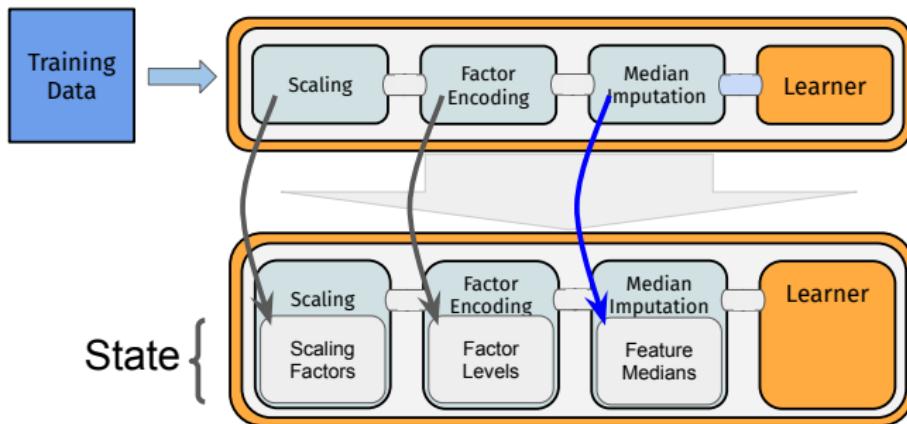


MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline

- `train()`ing: Data propagates and creates \$states

```
glrn = GraphLearner$new(graph_pp)  
glrn$train(task)
```

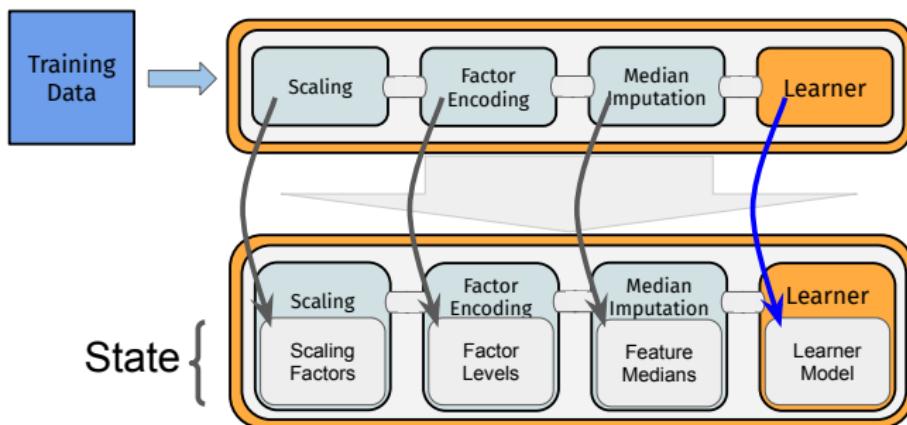


MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline

- `train()`ing: Data propagates and creates \$states

```
glrn = GraphLearner$new(graph_pp)  
glnr$train(task)
```

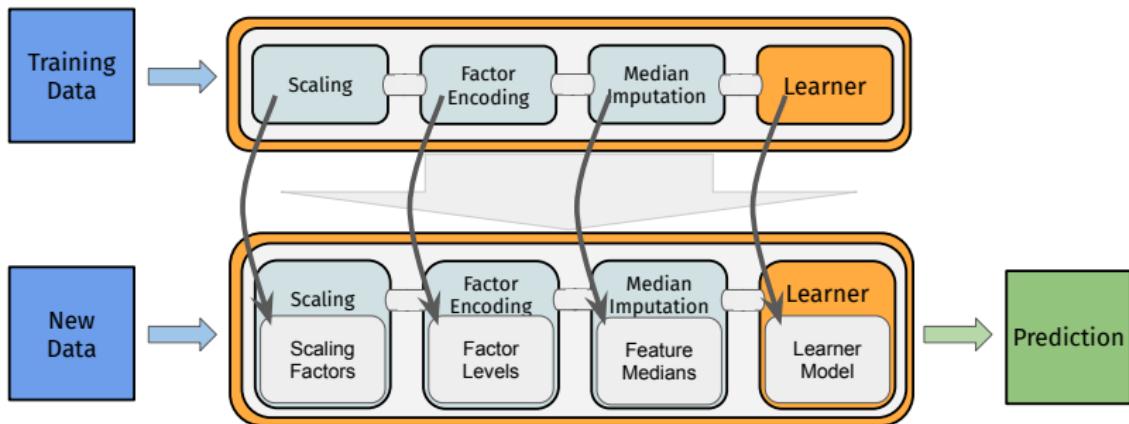


MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline

- `train()`ing: Data propagates and creates `$states`
- `predict()`ition: Data propagates, uses `$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()*)

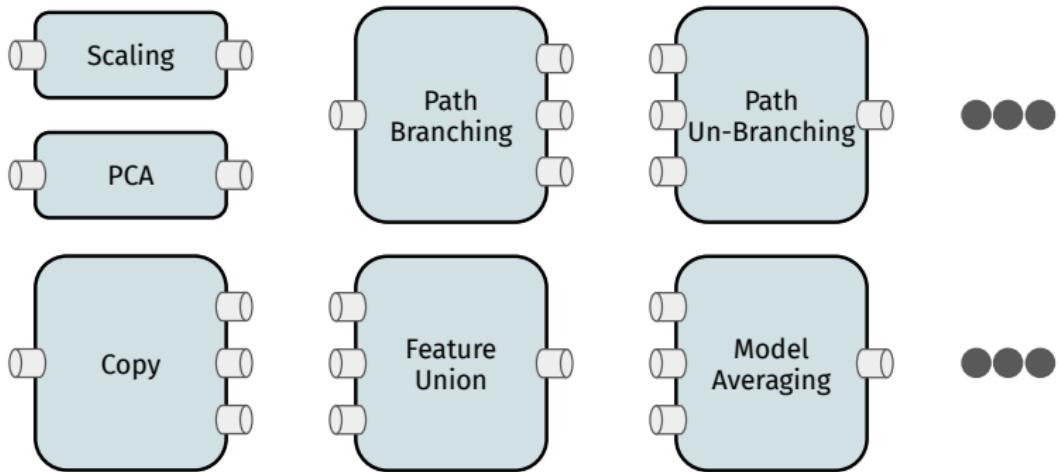
```
graph_pp$pipeops$scale$state$scale  
#> Petal.Length  Petal.Width Sepal.Length  Sepal.Width  
#>          4.2           1.4          5.9           3.1
```

- 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

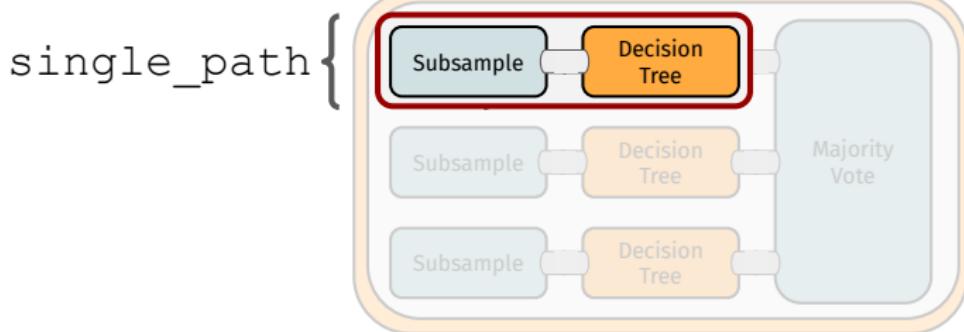
PIPEOPS WITH MULTIPLE INPUTS / OUTPUTS



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

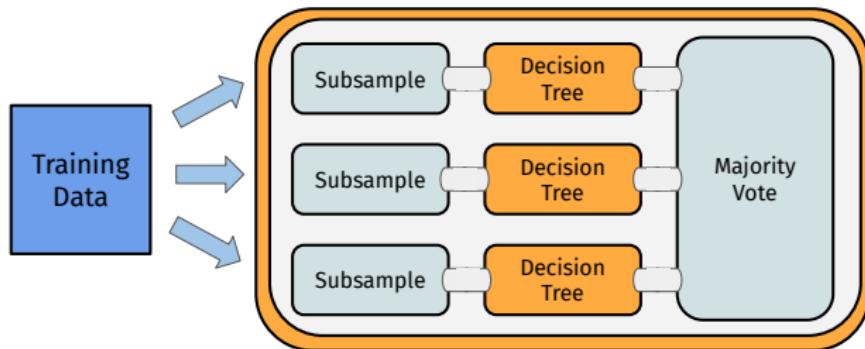
```
single_path = po("subsample") %>>% lrn("classif.rpart")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

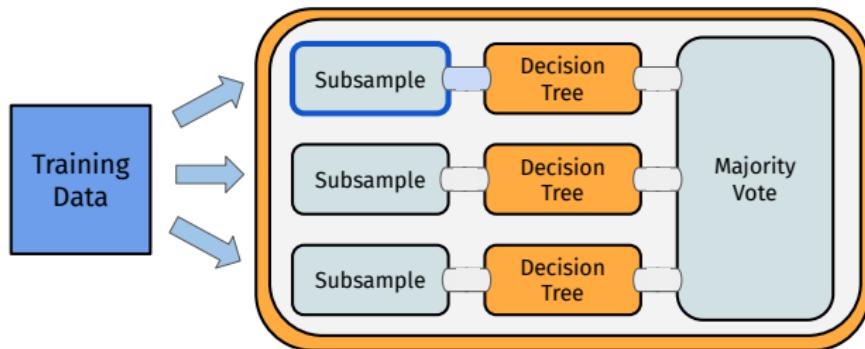
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

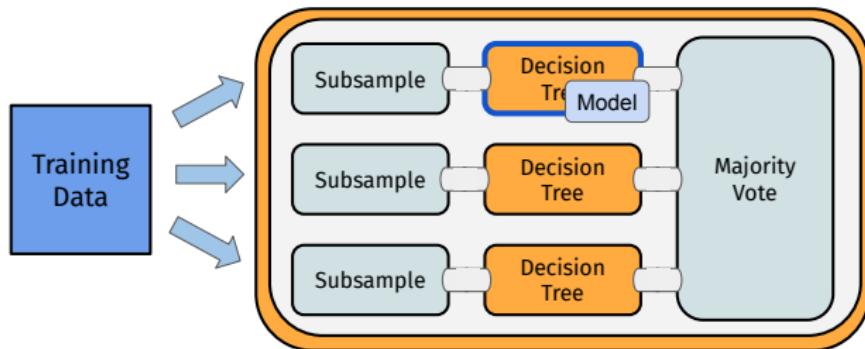
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

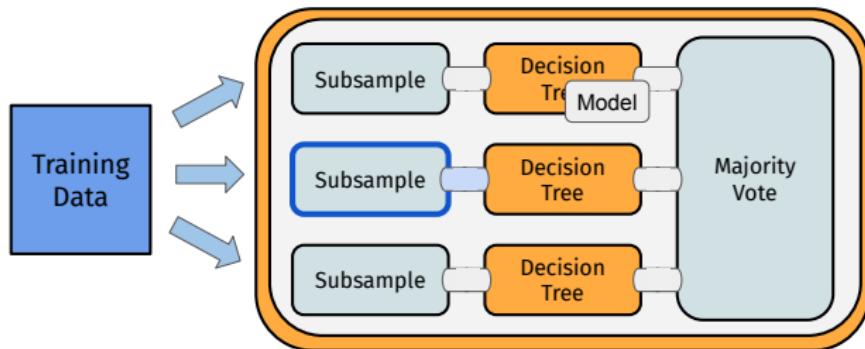
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

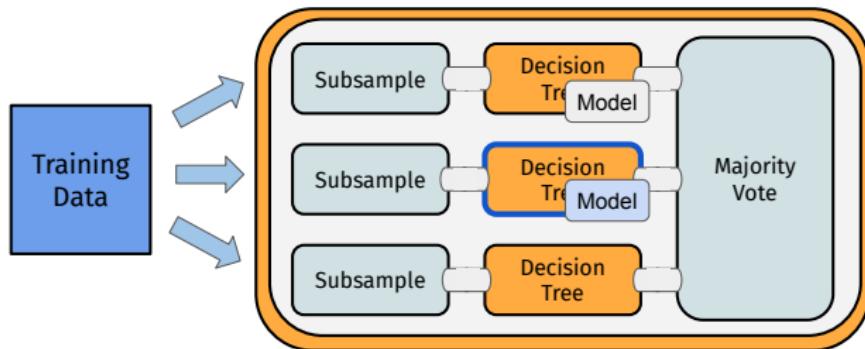
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

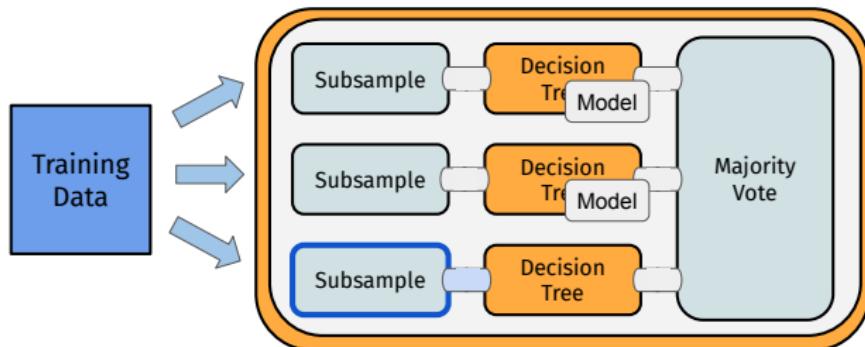
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

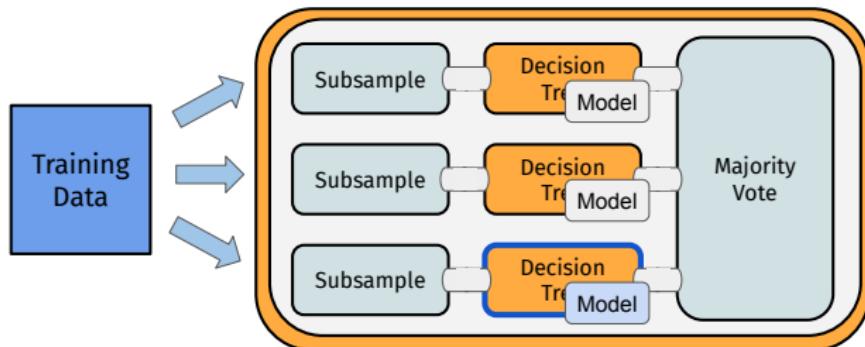
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

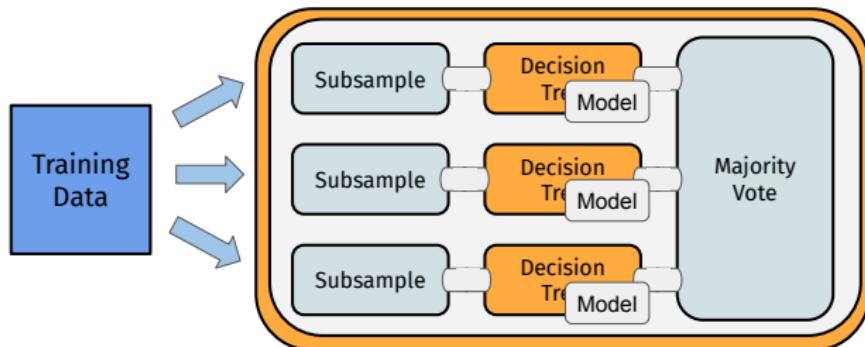
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

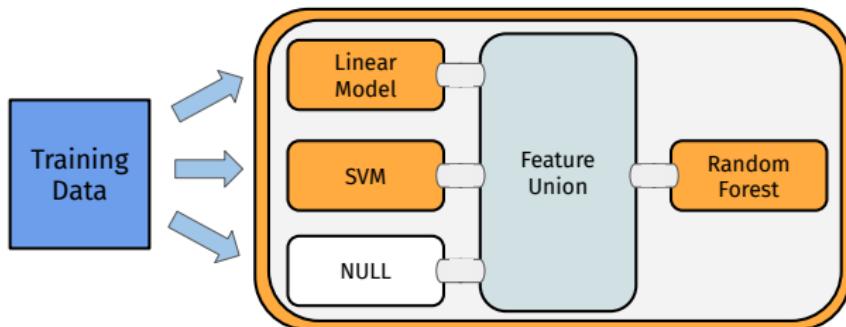
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



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")
```

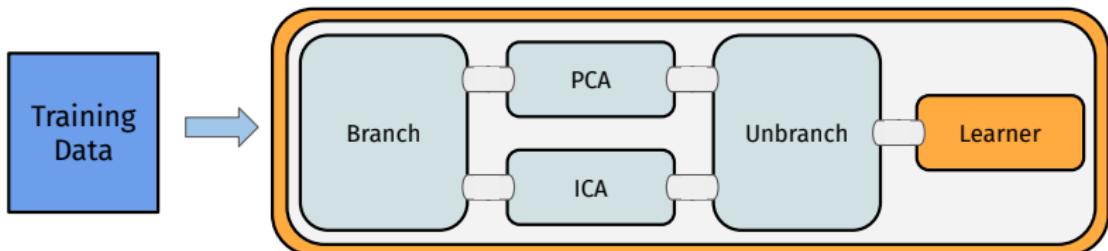


MLR3PIPELINES IN ACTION

Branching

```
graph_branch = ppl("branch", list(  
  pca = po("pca"),  
  ica = po("ica"))) %>>%  
  lrn("classif.kknn")
```

Execute only one of several alternative paths

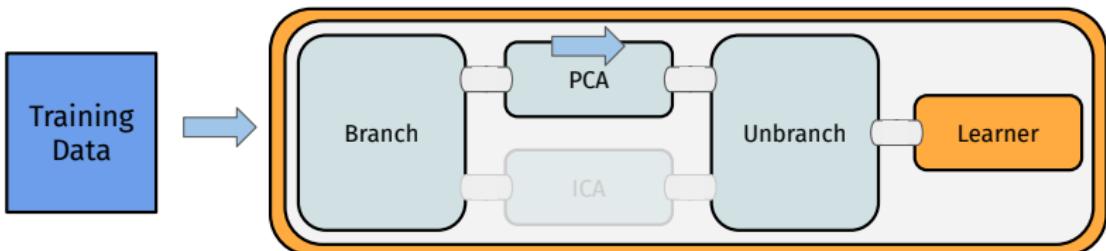


MLR3PIPELINES IN ACTION

Branching

```
graph_branch = ppl("branch", list(  
  pca = po("pca"),  
  ica = po("ica"))) %>>%  
  lrn("classif.kknn")
```

```
> graph_branch$pipeops$branch$  
  param_set$values$selection = "pca"
```

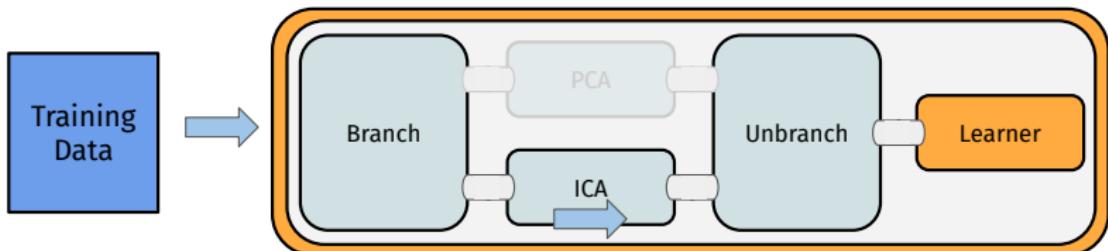


MLR3PIPELINES IN ACTION

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)

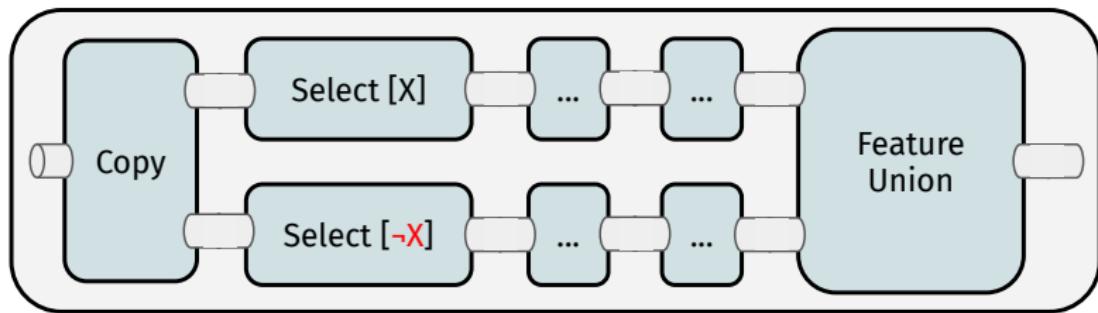
#>   Species    PC1     PC2 Petal.Length Petal.Width
#> 1:  setosa -0.78  0.378         1.4        0.2
#> 2:  setosa -0.94 -0.137         1.4        0.2
#> 3:  setosa -1.15  0.045         1.3        0.2
```

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`

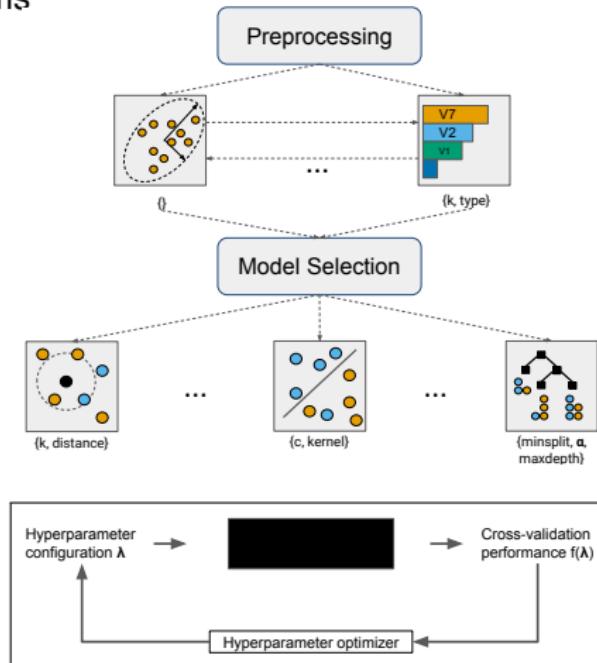
AUTOML <3 PIPELINES

- 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

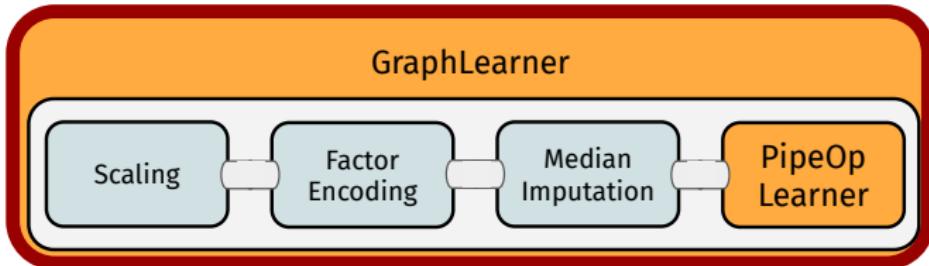
AutoML in a Nutshell

- Preprocessing steps
- ML Algorithms
- Tuner



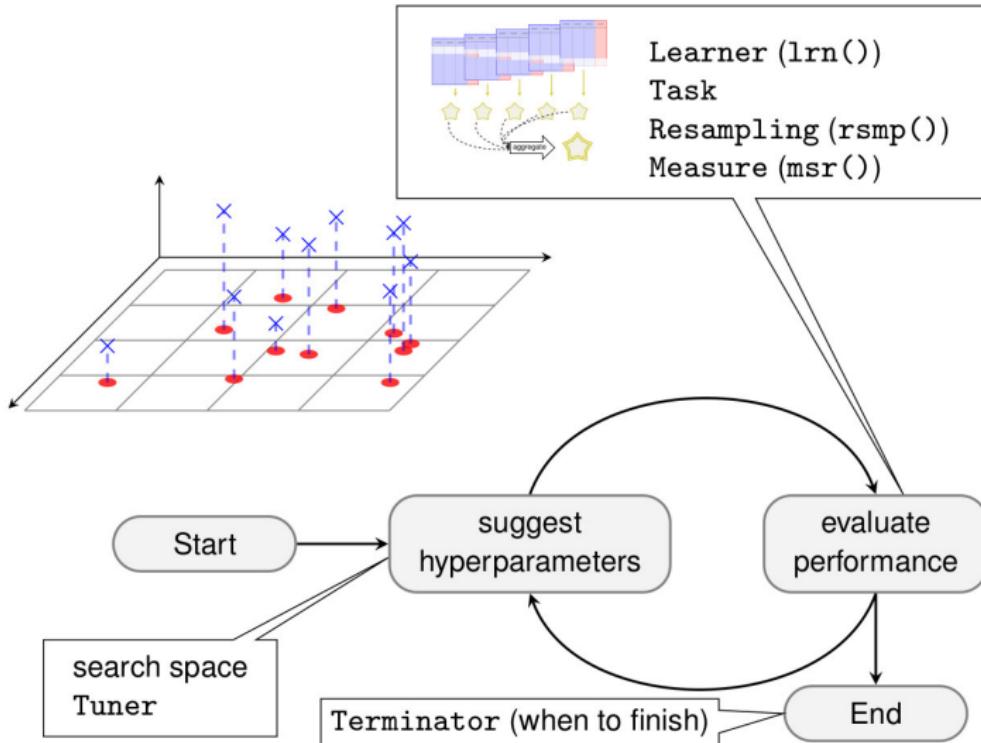
GRAPHLEARNER

- 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

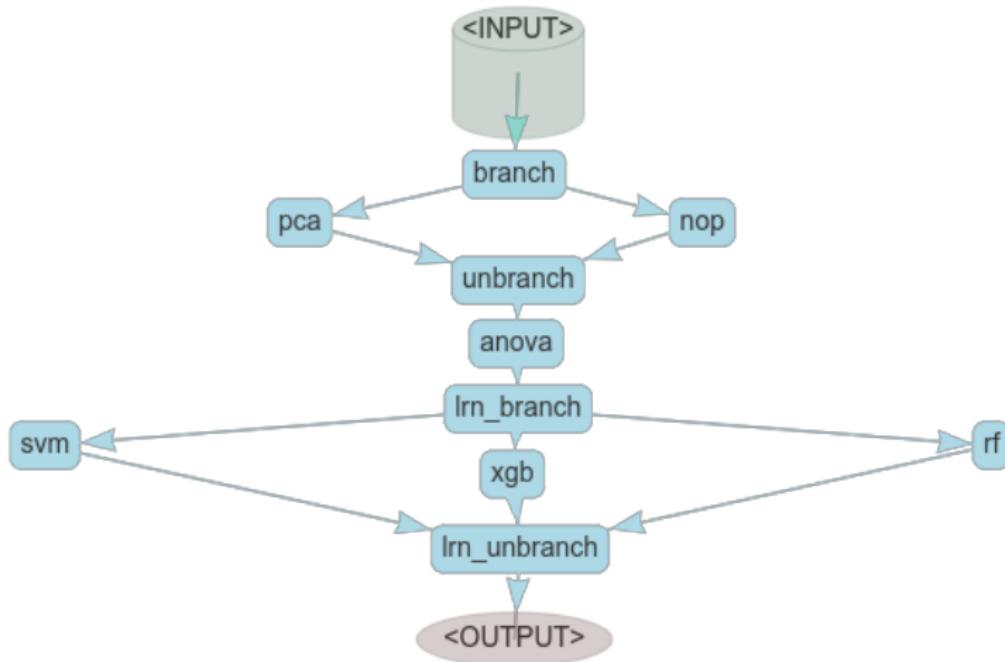


PIPELINES TUNING

- 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's Pipelines chapter. The left sidebar contains a navigation tree with sections like Quickstart, Introduction and Overview, Basics, Model Operations, Pipelines, and Technical. The main content area is titled "4 Pipelines". It includes a paragraph about mlr3pipelines being a dataflow programming toolkit, followed by a section on Machine learning workflows. Below this is a diagram illustrating a dataflow pipeline with four steps: Scaling, Factor Encoding, Median Imputation, and Learner. A note below the diagram says "Single computational steps can be represented as so-called Piplets, 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. The top navigation bar has links for mlr3, mlr3gallery, and mlr3book. The main content area is titled "mlr3 and OpenML - Moneyball use case". It includes a paragraph about how to make use of OpenML data and handle missing values in a ML problem. Below this is a section titled "A pipeline for the titanic data set - Advanced". It provides instructions on how to build a graph using the mlr3pipelines package on the "titanic" dataset. The section also covers feature engineering, data imputation, and benchmarking. To the right is a bar chart comparing survival rates for different gender groups. The bottom section is titled "Tuning a stacked learner" and explains how to create and tune a multilevel machine learning model using the mlr3pipelines package.

<https://mlr3gallery.mlr-org.com/>

“cheat sheets”

The screenshot shows the mlr3 cheat sheets website. It features several large, overlapping rectangular boxes, each containing a different type of cheat sheet. The visible titles include "Machine learning with mlr3 :: CHEAT SHEET", "Hyperparameter Tuning with mlr3tuning :: CHEAT SHEET", "Dataflow programming with mlr3pipelines :: CHEAT SHEET", "Graph", "Graph Construction", "GraphLearner", "Tuning", "Feature Engineering", "Linear Graphs", "Nonlinear Graphs", "Example - Feature Union", "Example - Bagging Ensemble", and "Crossing". Each box contains a brief description and some sample code or snippets.

<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