



# Introduction to Machine Learning

All slides

June 22, 2023

# INTRODUCTION TO MACHINE LEARNING

## ML Basics

Supervised Regression

Supervised Classification

Performance Evaluation

k-NN

Classification and Regression Trees (CART)

Random Forests

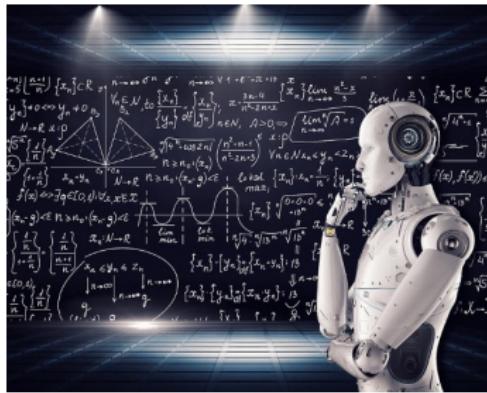
Neural Networks

Tuning

Nested Resampling

# Introduction to Machine Learning

## ML-Basics: What is Machine Learning?



### Learning goals

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

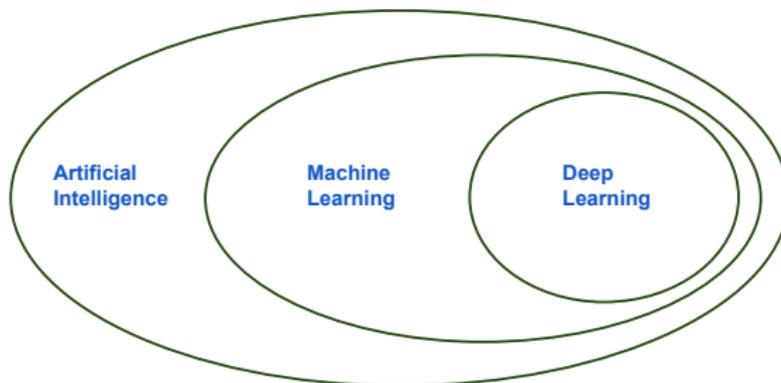
Image via [www.vpnsrus.com](http://www.vpnsrus.com)

# MACHINE LEARNING IS CHANGING OUR WORLD

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

# THE WORLD OF ARTIFICIAL INTELLIGENCE

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



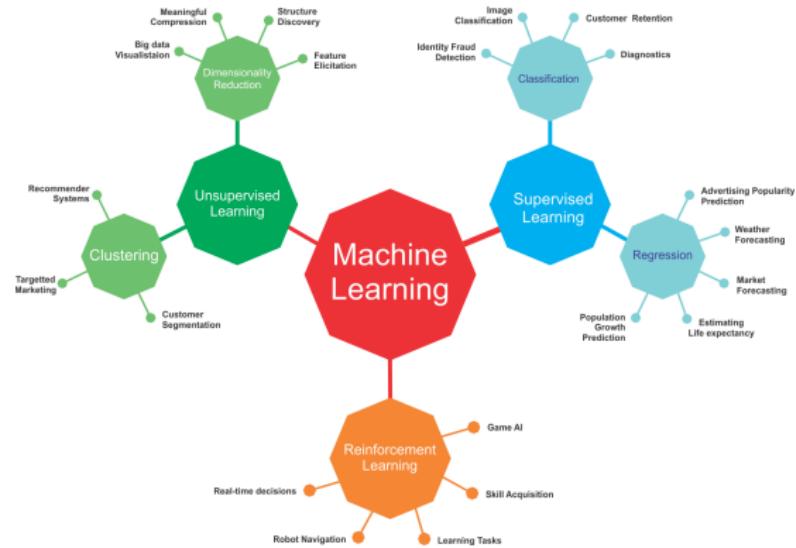
Many people are confused what these terms actually mean.

And what does all this have to do with statistics?

# ARTIFICIAL INTELLIGENCE

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

# MACHINE LEARNING



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

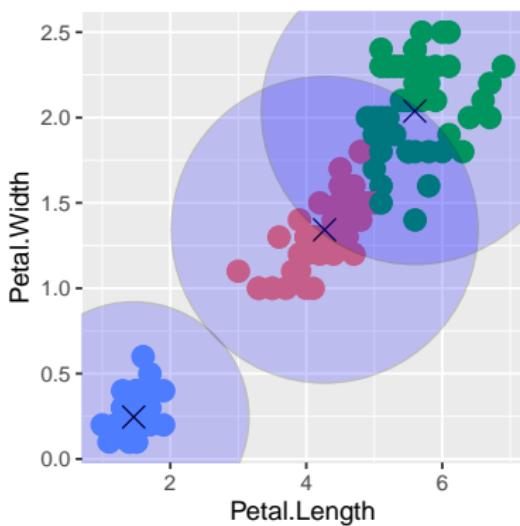
"Any sufficiently advanced technology is indistinguishable from magic."  
*Arthur C. Clarke's 3rd law*

# 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

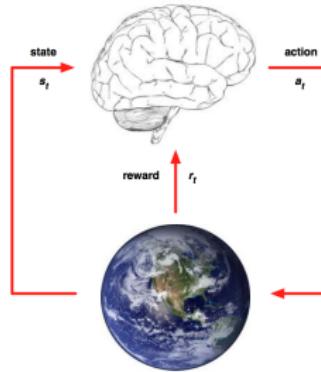
- Data without labels  $y$
- Search for patterns within the inputs  $x$
- *Unsupervised* as there is no “true” output we can optimize against



- 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.



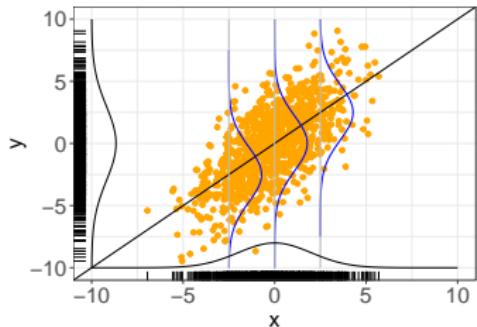
- Goal: Select actions to maximize future reward.
- Reward signals may be sparse, noisy and delayed.

# 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



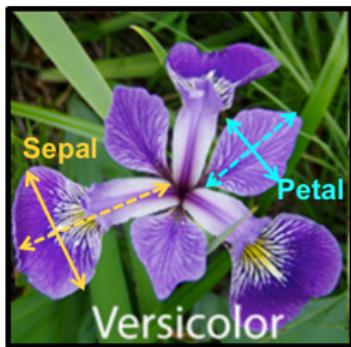
### Learning goals

- 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].

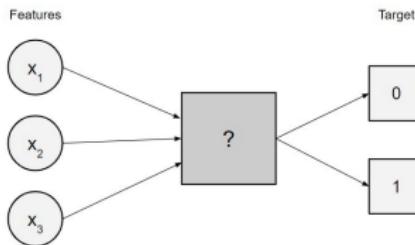


Source: <https://rpubs.com/vidhividhi/irisdataeda>

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

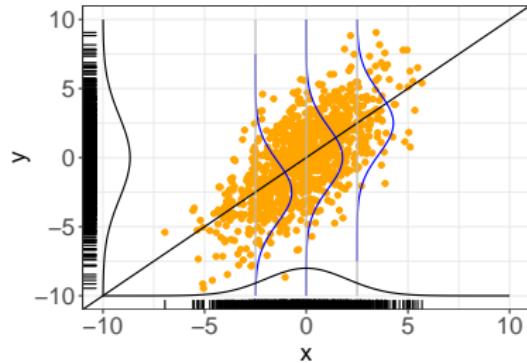
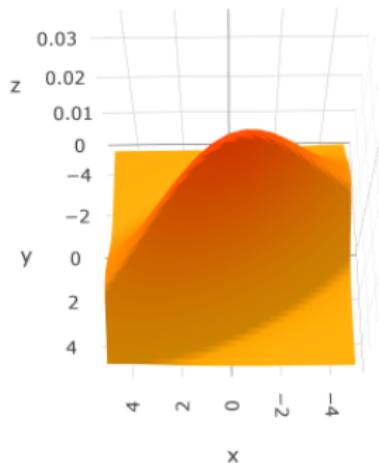
$$\mathbb{P}_{xy},$$

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.



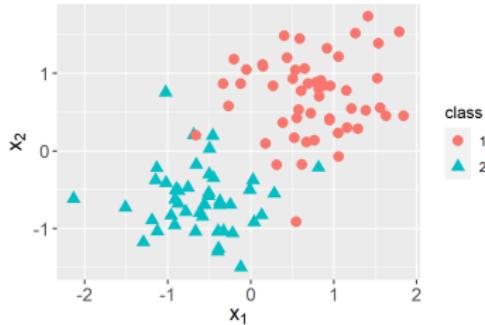
# DATA-GENERATING PROCESS

## 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  $\theta$  appear behind the  $|$  for improved legibility, not to imply that we condition on them in a probabilistic Bayesian sense. So, strictly speaking,  $p(\mathbf{x}|\theta)$  should usually be understood to mean  $p_\theta(\mathbf{x})$  or  $p(\mathbf{x}, \theta)$  or  $p(\mathbf{x}; \theta)$ . On the other hand, this notation makes it very easy to switch to a Bayesian view.

# Introduction to Machine Learning

## ML-Basics: Supervised Tasks



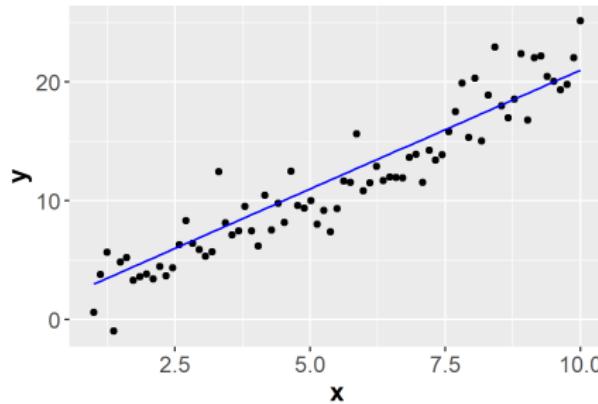
### Learning goals

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

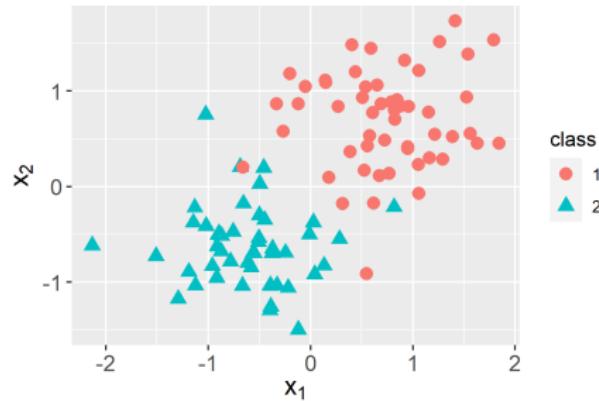
# TASKS: REGRESSION VS CLASSIFICATION

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

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



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



# PREDICT VS. EXPLAIN

We can distinguish two main reasons to learn this relationship:

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

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

# REGRESSION EXAMPLE: HOUSE PRICES

Predict the price for a house in a certain area

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



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

# REGRESSION EXAMPLE: LENGTH-OF-STAY

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

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



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

# CLASSIFICATION EXAMPLE: RISK CATEGORY

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

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

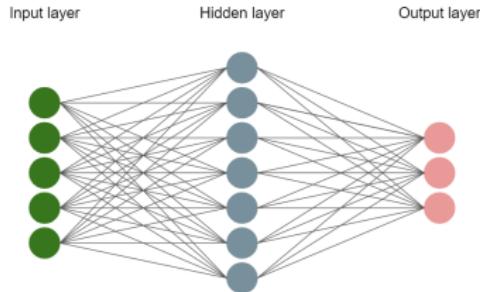


Probably *learn to predict*, but the company might be required to explain its predictions to its customers.

# Introduction to Machine Learning

## ML-Basics: Models & Parameters

### Learning goals



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

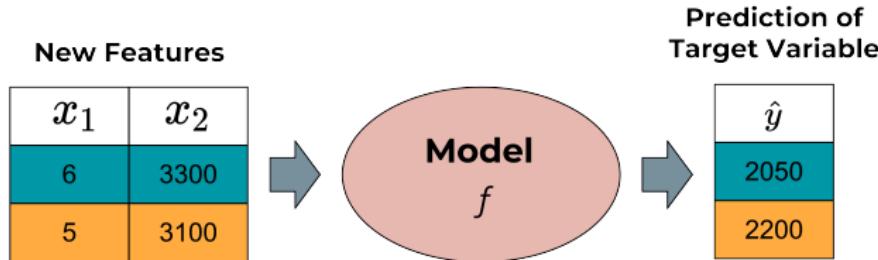
# WHAT IS A MODEL?

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

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

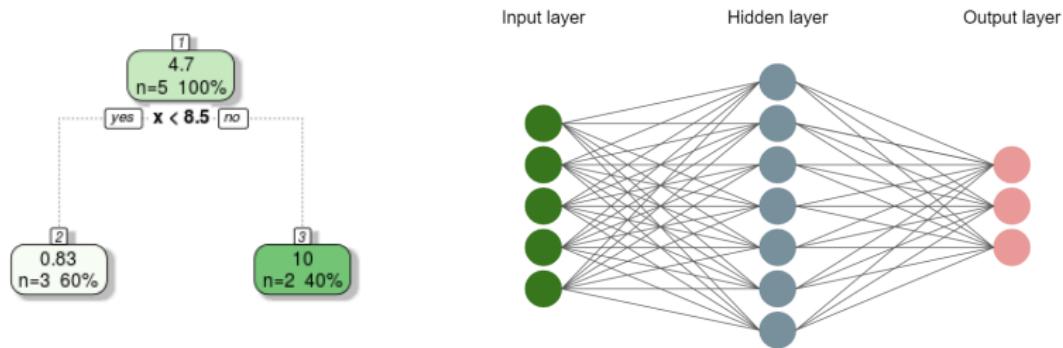
is a function that maps feature vectors to predicted target values.

- In conventional regression:  $g = 1$ ; for classification  $g$  is the number of classes, and output vectors are scores or class probabilities (details later).



# WHAT IS A MODEL?

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



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

# HYPOTHESIS SPACES

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

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

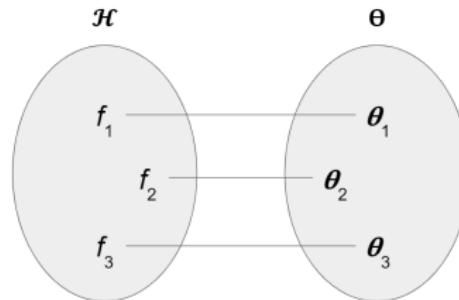
# PARAMETRIZATION

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

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

# PARAMETRIZATION

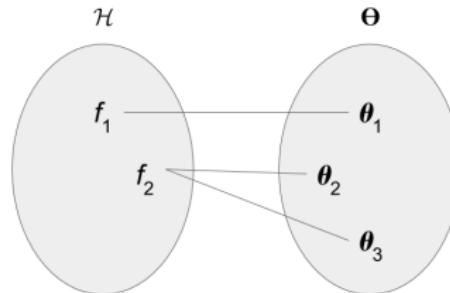
- This means: finding the optimal model is perfectly equivalent to finding the optimal set of parameter values.
- The relation between optimization over  $f \in \mathcal{H}$  and optimization over  $\theta \in \Theta$  allows us to operationalize our search for the best model via the search for the optimal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



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

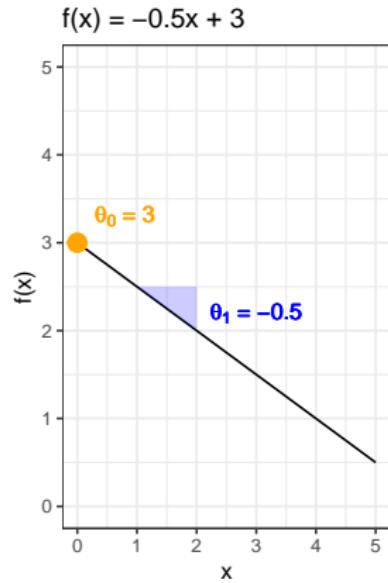
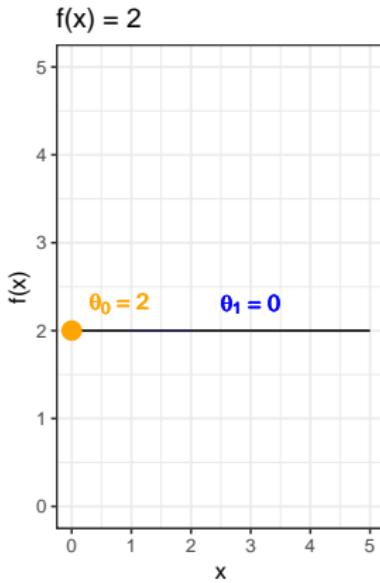
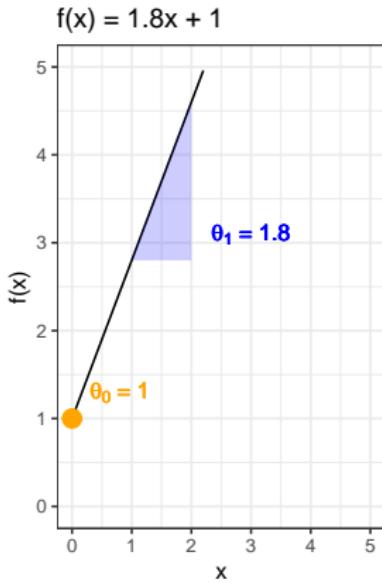
# PARAMETRIZATION

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



# EXAMPLE: UNIVARIATE LINEAR FUNCTIONS

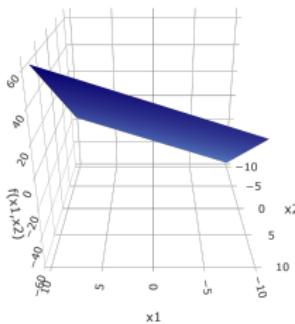
$$\mathcal{H} = \{f : f(\mathbf{x}) = \theta_0 + \theta_1 x, \theta \in \mathbb{R}^2\}$$



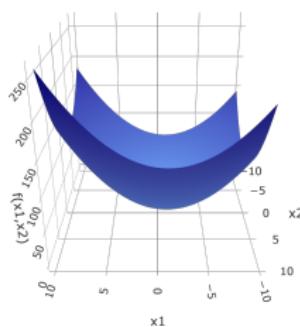
# EXAMPLE: BIVARIATE QUADRATIC FUNCTIONS

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

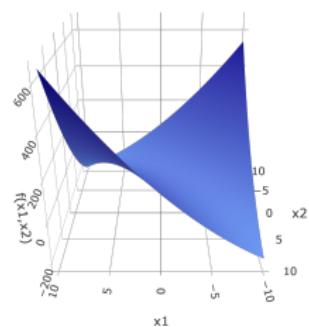
$$f(x) = 3 + 2x_1 + 4x_2$$



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



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



# EXAMPLE: RBF NETWORK

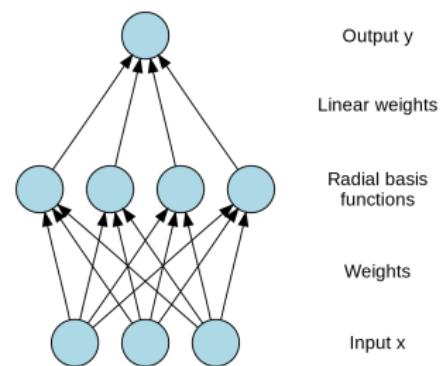
Radial basis function networks with Gaussian basis functions

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

where

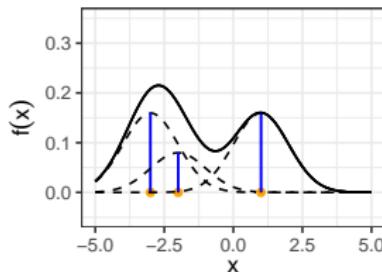
- $a_i$  is the weight of the  $i$ -th neuron,
- $\mathbf{c}_i$  its center vector, and
- $\rho(\|\mathbf{x} - \mathbf{c}_i\|) = \exp(-\beta \|\mathbf{x} - \mathbf{c}_i\|^2)$  is the  $i$ -th radial basis function with bandwidth  $\beta \in \mathbb{R}$ .

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

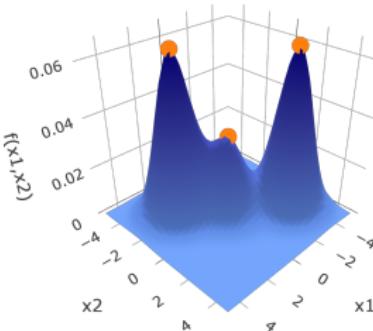


# EXAMPLE: RBF NETWORK

Exemplary setting

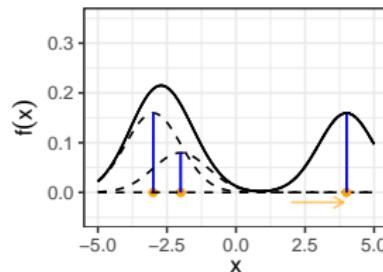


$$\begin{aligned} a_1 &= 0.4, a_2 = 0.2, a_3 = 0.4 \\ c_1 &= -3, c_2 = -2, c_3 = 1 \end{aligned}$$

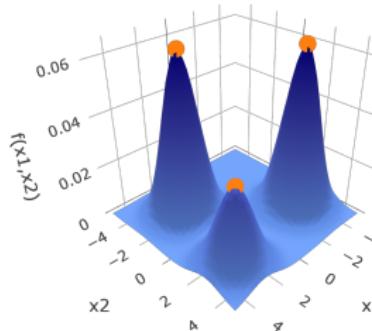


$$\begin{aligned} a_1 &= 0.4, a_2 = 0.2, a_3 = 0.4 \\ c_1 &= (2, -2), c_2 = (0, 0), \\ c_3 &= (-3, 2) \end{aligned}$$

Centers altered

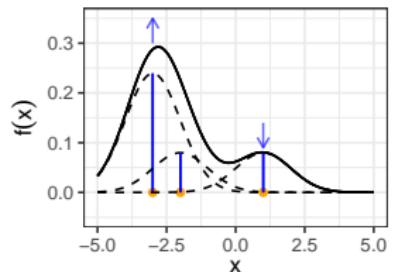


$$\begin{aligned} a_1 &= 0.4, a_2 = 0.2, a_3 = 0.4 \\ c_1 &= -3, c_2 = -2, c_3 = 4 \end{aligned}$$

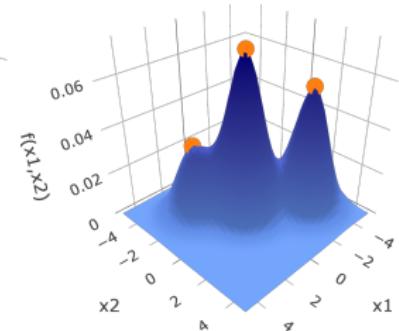


$$\begin{aligned} a_1 &= 0.4, a_2 = 0.2, a_3 = 0.4 \\ c_1 &= (2, -2), c_2 = (0, 0), \\ c_3 &= (-3, 2) \end{aligned}$$

Weights altered



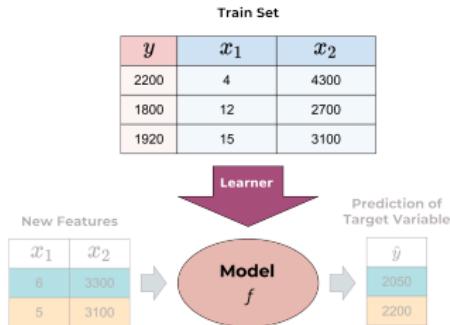
$$\begin{aligned} a_1 &= 0.6, a_2 = 0.2, a_3 = 0.2 \\ c_1 &= -3, c_2 = -2, c_3 = 1 \end{aligned}$$



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

# 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 represents the number of people in the office (Feature 1,  $x_1$ ), with values 4, 12, and 5. The second column represents salary (Feature 2,  $x_2$ ), with values 4300 €, 2700 €, and 3100 €. The third column represents the target variable, worked minutes per week ( $y$ ), with values 2220, 1800, and 1920. A brace on the left indicates there are 3 data points ( $n = 3$ ). Brackets below the first and second columns indicate there are 2 features ( $p = 2$ ). Circles labeled  $x_1^{(2)}$  and  $x_2^{(1)}$  point to the first column, and a circle labeled  $y^{(3)}$  points to the third column.

# SUPERVISED LEARNING EXAMPLE

How could we construct a model from these data?

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

- The baseline productivity of an employee with salary 3000 and 7 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.

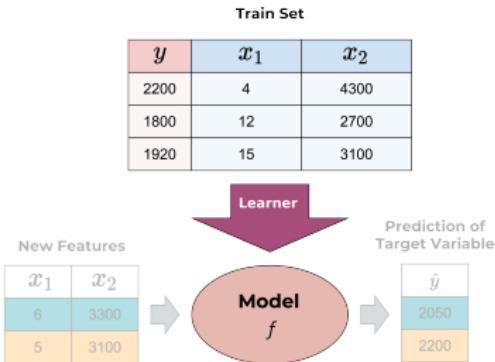
- **Supervised** learning means we make use of *labeled* data for which we observed the outcome.
- We use the labeled data to learn a model  $f$ .
- Ultimately, we use our model to compute predictions for **new** data whose target values are unknown.



# 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:

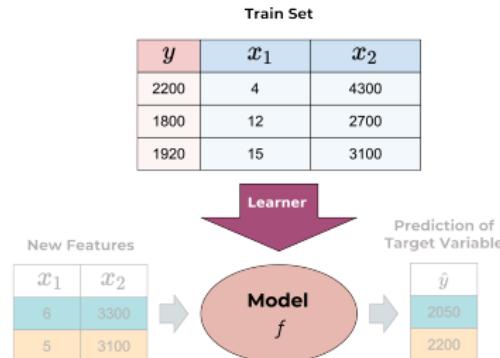
$$\mathcal{I} : \mathbb{D} \times \Lambda \rightarrow \mathcal{H}$$



# LEARNER DEFINITION

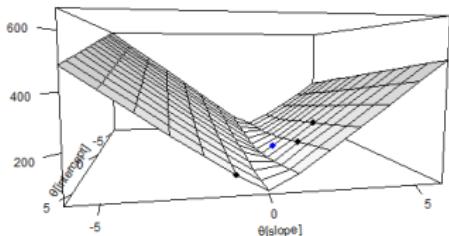
As pseudo-code template it would work like this:

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



# Introduction to Machine Learning

## ML-Basics: Losses & Risk Minimization



### Learning goals

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

# HOW TO EVALUATE MODELS

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

The diagram illustrates the components of a 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)	Worked Minutes Week (Target Variable)
4	4300 €	2220	2588
12	2700 €	1800	1644
5	3100 €	1920	1870

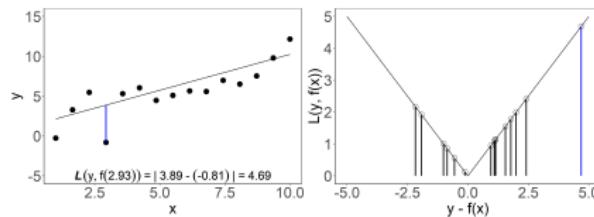
- To make this precise, we need to define now how we measure the difference between a prediction and a ground truth label pointwise.

# LOSS

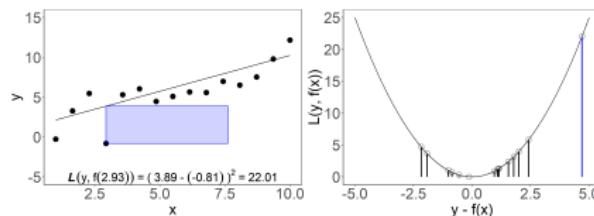
The **loss function**  $L(y, f(\mathbf{x}))$  quantifies the "quality" of the prediction  $f(\mathbf{x})$  of a single observation  $\mathbf{x}$ :

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

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  $\mathcal{D}$ .

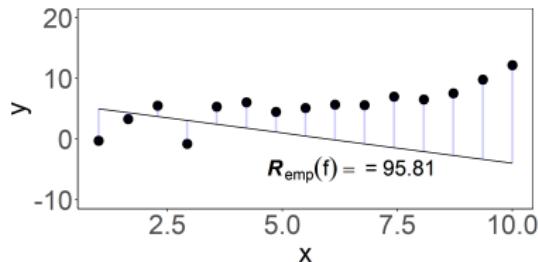
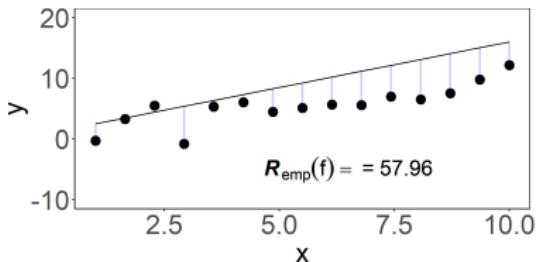
# EMPIRICAL RISK

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

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

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

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

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

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

# EMPIRICAL RISK MINIMIZATION

The best model is the model with the smallest risk.

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

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

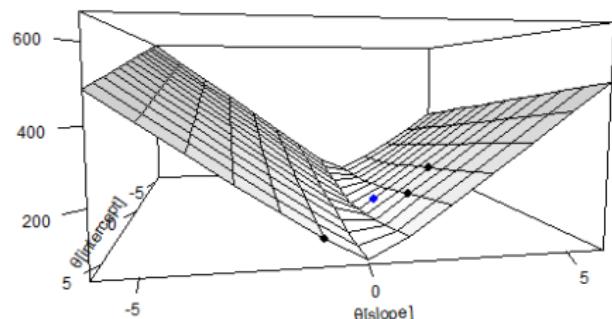
# EMPIRICAL RISK MINIMIZATION

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

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

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

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



# EMPIRICAL RISK MINIMIZATION

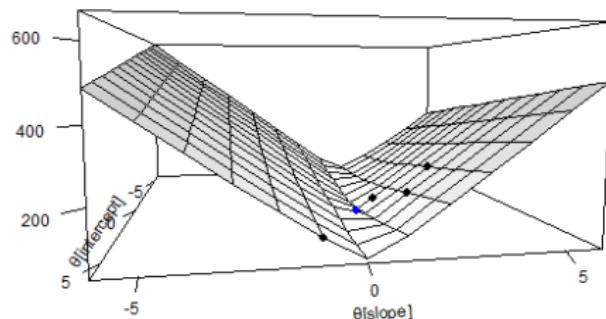
Minimizing this surface is called **empirical risk minimization** (ERM).

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

Usually we do this by numerical optimization.

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

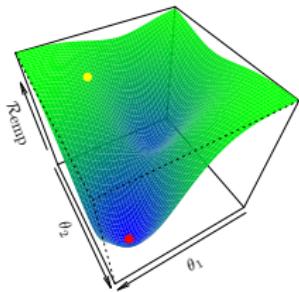
Model	$\theta_{\text{intercept}}$	$\theta_{\text{slope}}$	$\mathcal{R}_{\text{emp}}(\theta)$
$f_1$	2	3	194.62
$f_2$	3	2	127.12
$f_3$	6	-1	95.81
$f_4$	1	1.5	57.96
$f_5$	1.25	0.90	23.40



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

# Introduction to Machine Learning

## ML-Basics: Optimization

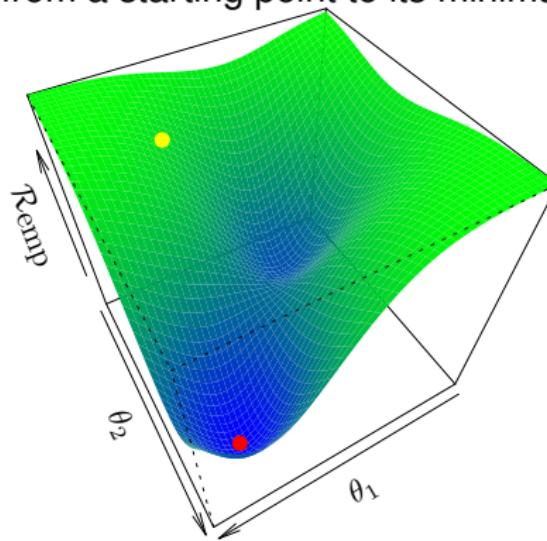


### Learning goals

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

# LEARNING AS PARAMETER OPTIMIZATION

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



# LEARNING AS PARAMETER OPTIMIZATION

The ERM optimization problem is:

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

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

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

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

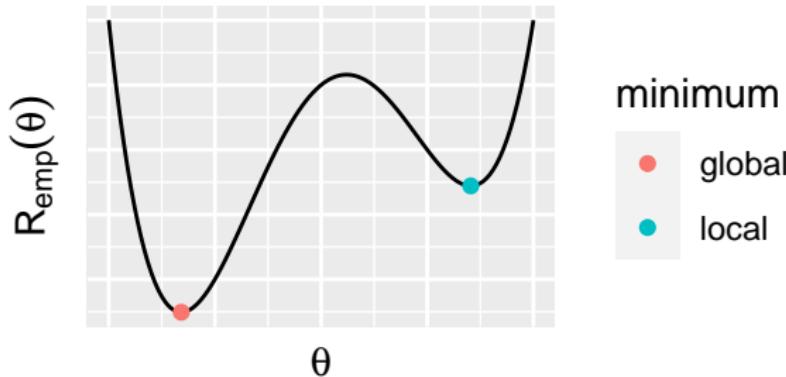
Which kind of numerical technique is reasonable for this problem strongly depends on model and parameter structure (continuous params? uni-modal  $\mathcal{R}_{\text{emp}}(\theta)$ ?). Here, we will only discuss very simple scenarios.

# LOCAL MINIMA

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

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

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

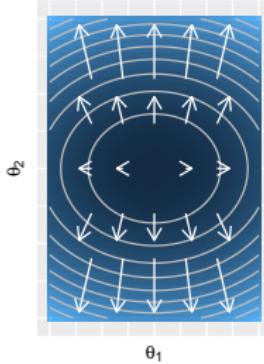


# LOCAL MINIMA AND STATIONARY POINTS

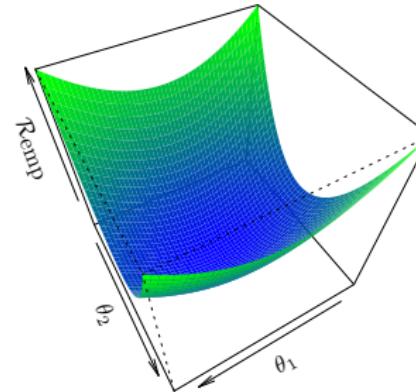
If  $\mathcal{R}_{\text{emp}}$  is continuously differentiable in  $\theta$  then a **sufficient condition** for a local minimum is that  $\hat{\theta}$  is **stationary** with 0 gradient, so no local improvement is possible:

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

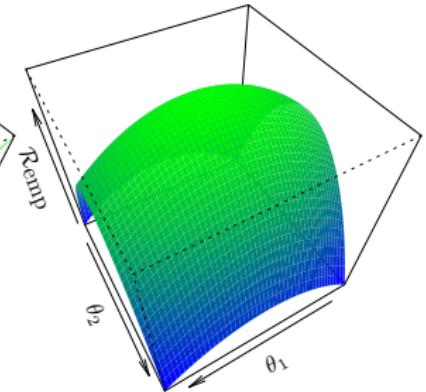
and the Hessian  $\frac{\partial^2}{\partial \theta^2} \mathcal{R}_{\text{emp}}(\hat{\theta})$  is positive definite. While the neg. gradient points into the direction of fastest local decrease, the Hessian measures local curvature of  $\mathcal{R}_{\text{emp}}$ .



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



const. pos. def. Hessian



const. neg. def. Hessian

# LEAST SQUARES ESTIMATOR

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

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

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

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

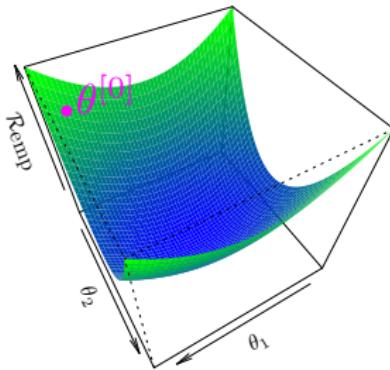
is a local minimum of  $\mathcal{R}_{\text{emp}}$ . If  $\mathbf{X}$  is full-rank,  $\mathcal{R}_{\text{emp}}$  is strictly convex and there is only one local minimum - which is also global.

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

# GRADIENT DESCENT

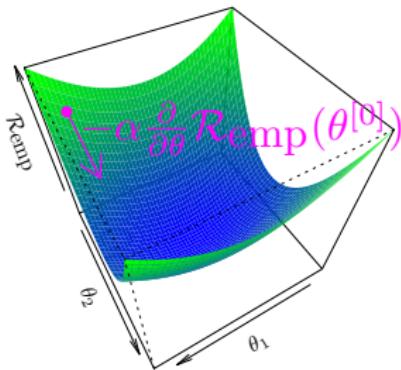
The simple idea of GD is to iteratively go from the current candidate  $\theta^{[t]}$  in the direction of the negative gradient, i.e., the direction of the steepest descent, with learning rate  $\alpha$  to the next  $\theta^{[t+1]}$ :

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

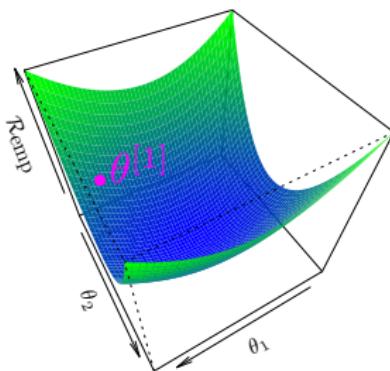


We choose a random start  $\theta^{[0]}$  with risk  $\mathcal{R}_{\text{emp}}(\theta^{[0]}) = 76.25$ .

# GRADIENT DESCENT - EXAMPLE

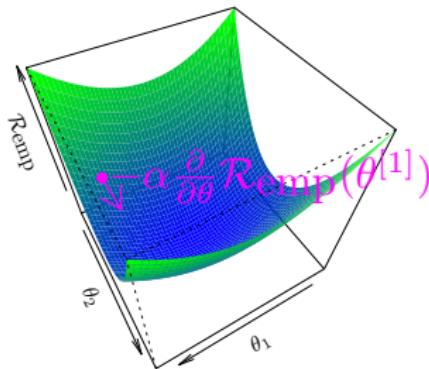


Now we follow in the direction of the negative gradient at  $\theta^{[0]}$ .

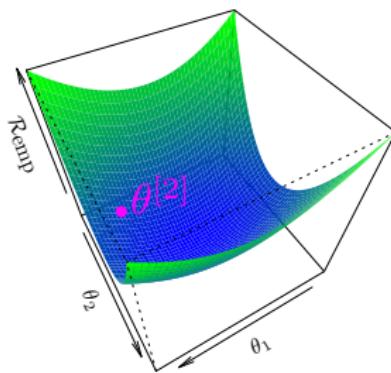


We arrive at  $\theta^{[1]}$  with risk  
 $\mathcal{R}_{\text{emp}}(\theta^{[1]}) \approx 42.73$ .  
We improved:  
 $\mathcal{R}_{\text{emp}}(\theta^{[1]}) < \mathcal{R}_{\text{emp}}(\theta^{[0]}).$

# GRADIENT DESCENT - EXAMPLE

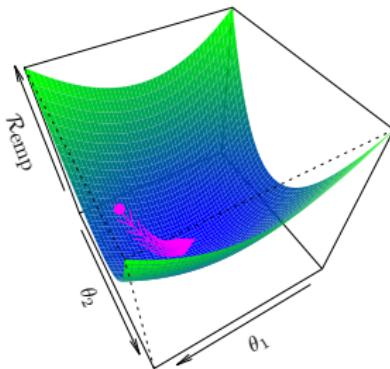


Again we follow in the direction of the negative gradient, but now at  $\theta^{[1]}$ .

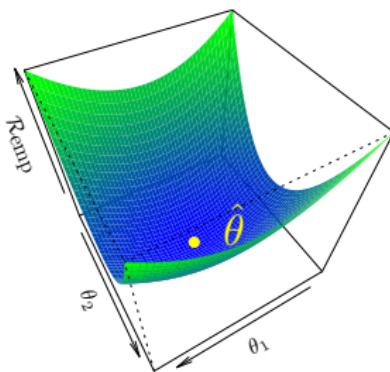


Now  $\theta^{[2]}$  has risk  $\mathcal{R}_{\text{emp}}(\theta^{[2]}) \approx 25.08$ .

# GRADIENT DESCENT - EXAMPLE



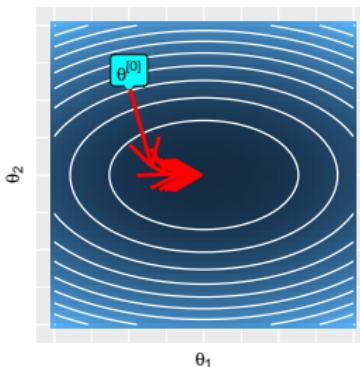
We iterate this until some form of convergence or termination.



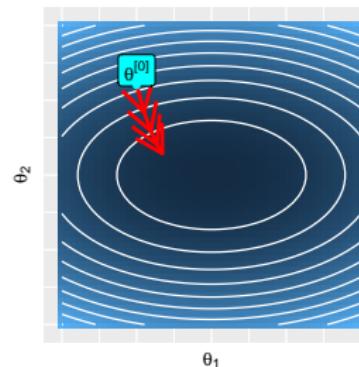
We arrive close to a stationary  $\hat{\theta}$  which is hopefully at least a local minimum.

# GRADIENT DESCENT - LEARNING RATE

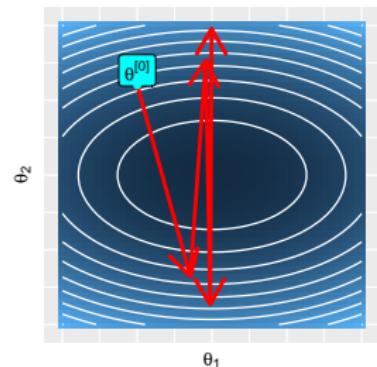
- The negative gradient is a direction that looks locally promising to reduce  $\mathcal{R}_{\text{emp}}$ .
- Hence it weights components higher in which  $\mathcal{R}_{\text{emp}}$  decreases more.
- However, the length of  $-\frac{\partial}{\partial \theta} \mathcal{R}_{\text{emp}}$  measures only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



good convergence for  $\alpha_1$



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



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

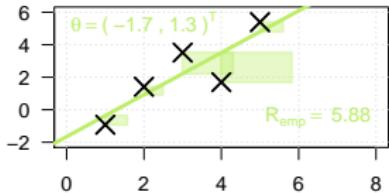
# FURTHER TOPICS

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

# Introduction to Machine Learning

## ML-Basics: Components of Supervised Learning

### Learning goals



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

# COMPONENTS OF SUPERVISED LEARNING

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

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

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

# COMPONENTS OF SUPERVISED LEARNING

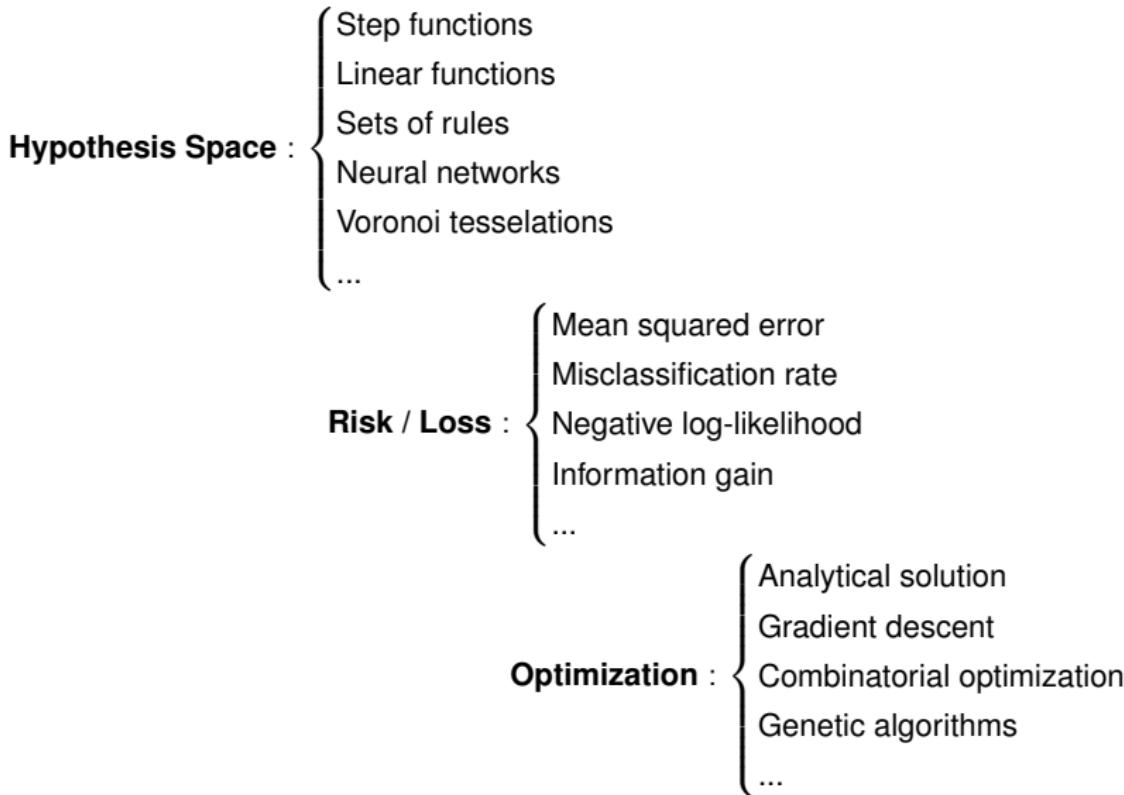
This concept can be extended by the concept of **regularization**, where the model complexity is accounted for in the risk:

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

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

# VARIETY OF LEARNING COMPONENTS

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



# SUPERVISED LEARNING, FORMALIZED

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

- receives a **training set**  $\mathcal{D} \in \mathbb{D}$ , 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  $\theta$  on  $\mathcal{D}$
- uses an **optimization** procedure to find

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

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

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

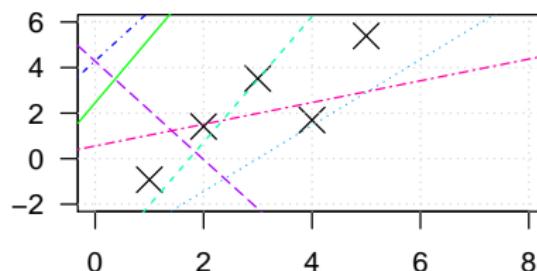
We can also adapt this concept to finding  $\hat{\theta}$  for parametric models:

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

## EXAMPLE: LINEAR REGRESSION ON 1D

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

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

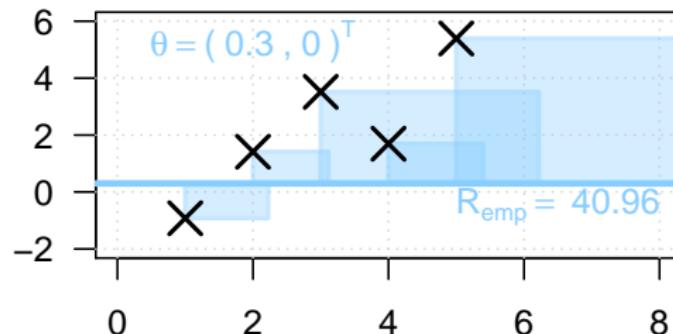


**Design choice:** We could add more flexibility by allowing polynomial effects or by using a spline basis.

# EXAMPLE: LINEAR REGRESSION ON 1D

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

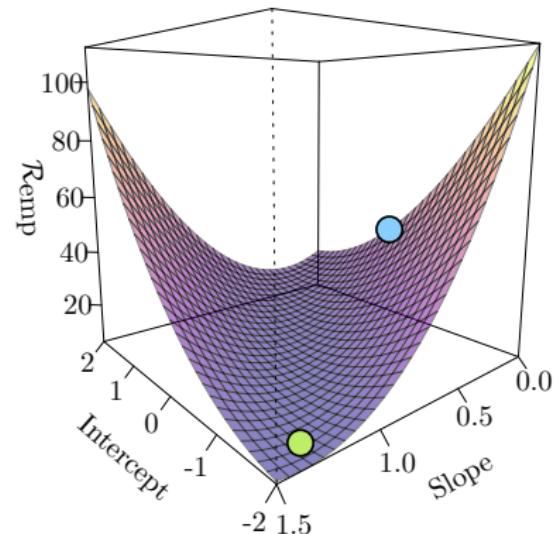
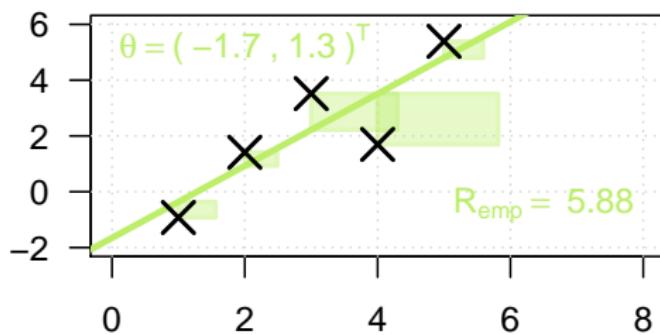
$$\mathcal{R}_{\text{emp}}(\theta) = \sum_{i=1}^n (y^{(i)} - \theta_0 - \theta_1 x^{(i)})^2$$



**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

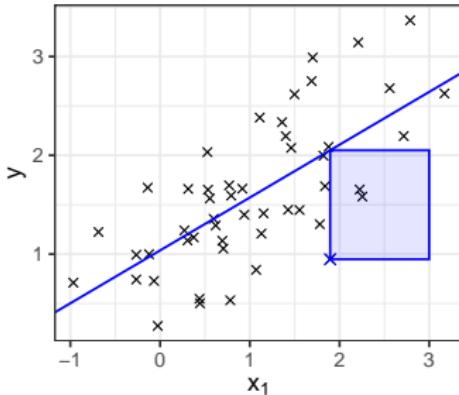
Neural Networks

Tuning

Nested Resampling

# Introduction to Machine Learning

## Supervised Regression: Linear Models with $L2$ Loss



### Learning goals

- Grasp the overall concept of linear regression
- Understand how  $L2$  loss optimization results in SSE-minimal model
- Understand this as a general template for ERM in ML

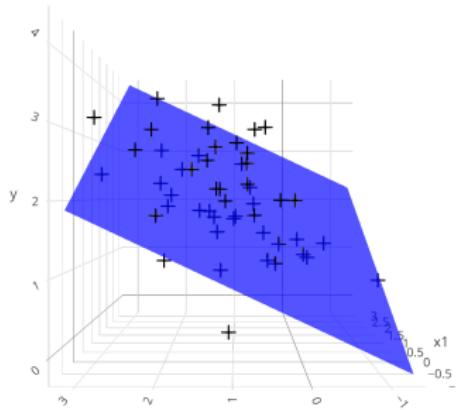
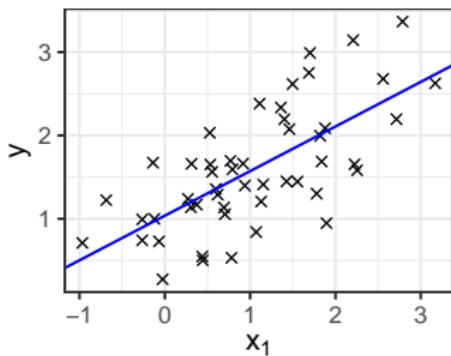
# LINEAR REGRESSION

- Idea: predict  $y \in \mathbb{R}$  as linear combination of features<sup>1</sup>:

$$\hat{y} = f(\mathbf{x}) = \boldsymbol{\theta}^\top \mathbf{x} = \theta_0 + \theta_1 x_1 + \cdots + \theta_p x_p$$

~~ find loss-optimal params to describe relation  $y|\mathbf{x}$

- Hypothesis space:  $\mathcal{H} = \{f(\mathbf{x}) = \boldsymbol{\theta}^\top \mathbf{x} \mid \boldsymbol{\theta} \in \mathbb{R}^{p+1}\}$



---

<sup>1</sup> Actually, special case of linear model, which is linear combo of basis functions of features ~~> Polynomial Regression Models

# DESIGN MATRIX

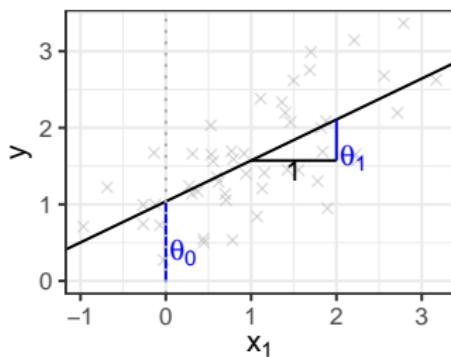
- Mismatch:  $\theta \in \mathbb{R}^{p+1}$  vs  $\mathbf{x} \in \mathbb{R}^p$  due to intercept term
- Trick: pad feature vectors with leading 1, s.t.
  - $\mathbf{x} \mapsto \mathbf{x} = (1, x_1, \dots, x_p)^\top$ , and
  - $\theta^\top \mathbf{x} = \theta_0 \cdot 1 + \theta_1 x_1 + \dots + \theta_p x_p$
- Collect all observations in **design matrix**  $\mathbf{X} \in \mathbb{R}^{n \times (p+1)}$   
~~ more compact: single param vector incl. intercept
- Resulting linear model:

$$\hat{\mathbf{y}} = \mathbf{X}\theta = \begin{pmatrix} 1 & x_1^{(1)} & \dots & x_p^{(1)} \\ 1 & x_1^{(2)} & \dots & x_p^{(2)} \\ \vdots & \vdots & & \vdots \\ 1 & x_1^{(n)} & \dots & x_p^{(n)} \end{pmatrix} \begin{pmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_p \end{pmatrix} = \begin{pmatrix} \theta_0 + \theta_1 x_1^{(1)} + \dots + \theta_p x_p^{(1)} \\ \theta_0 + \theta_1 x_1^{(2)} + \dots + \theta_p x_p^{(2)} \\ \vdots \\ \theta_0 + \theta_1 x_1^{(n)} + \dots + \theta_p x_p^{(n)} \end{pmatrix}$$

- We will make use of this notation in other contexts

# EFFECT INTERPRETATION

- Big plus of LM: immediately **interpretable** feature effects
- "Marginally increasing  $x_j$  by 1 unit increases  $y$  by  $\theta_j$  units"  
~~ *ceteris paribus* assumption:  $x_1, \dots, x_{j-1}, x_{j+1}, \dots, x_p$  fixed

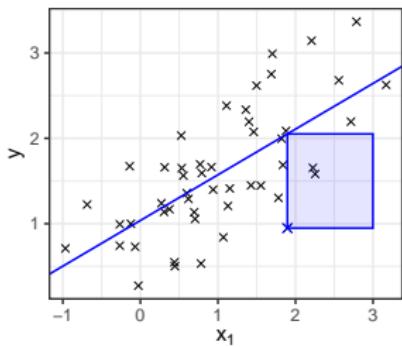


```
Call:  
lm(formula = y ~ x_1, data = dt_univ)  
  
Residuals:  
    Min      1Q  Median      3Q     Max  
-1.10346 -0.34727 -0.00766  0.31500  1.04284  
  
Coefficients:  
            Estimate Std. Error t value Pr(>|t|)  
(Intercept) 1.03727   0.11360   9.131 4.55e-12 ***  
x_1         0.53521   0.08219   6.512 4.13e-08 ***  
---  
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1  
  
Residual standard error: 0.5327 on 48 degrees of freedom  
Multiple R-squared:  0.469,    Adjusted R-squared:  0.458  
F-statistic: 42.4 on 1 and 48 DF,  p-value: 4.129e-08
```

# MODEL FIT

- How to determine LM fit?  $\rightsquigarrow$  define risk & optimize
- Popular: **L2 loss / quadratic loss / squared error**

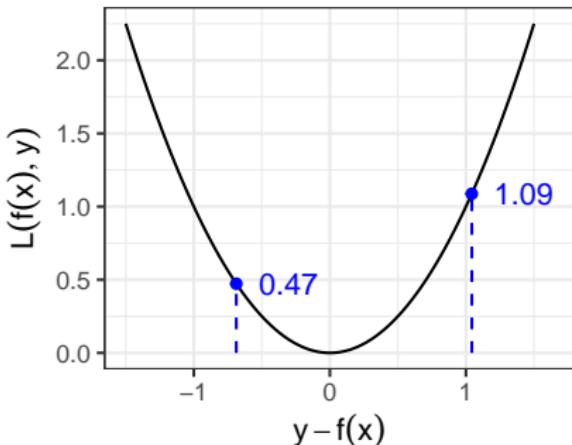
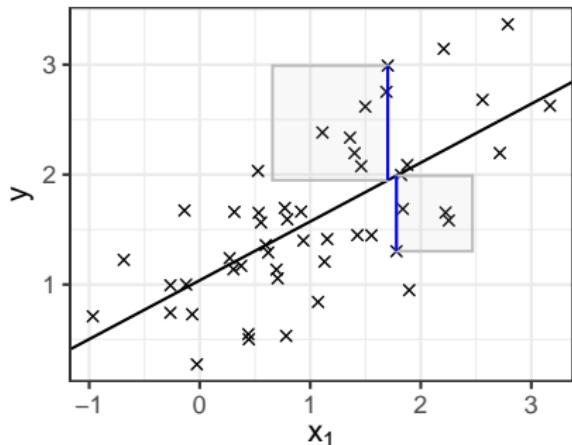
$$L(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2 \text{ or } L(y, f(\mathbf{x})) = 0.5 \cdot (y - f(\mathbf{x}))^2$$



- Why penalize **residuals**  $r = y - f(\mathbf{x})$  quadratically?
  - Easy to optimize (convex, differentiable)
  - Theoretically appealing (connection to classical stats LM)

# LOSS PLOTS

We will often visualize loss effects like this:



- Data as  $y \sim x_1$
- Prediction hypersurface  
~~ here: line
- Residuals  $r = y - f(x)$   
~~ squares to illustrate loss

- Loss as function of residuals  
~~ strength of penalty?  
~~ symmetric?
- Highlighted: loss for residuals shown on LHS

# OPTIMIZATION

- Resulting risk equivalent to **sum of squared errors (SSE)**:

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

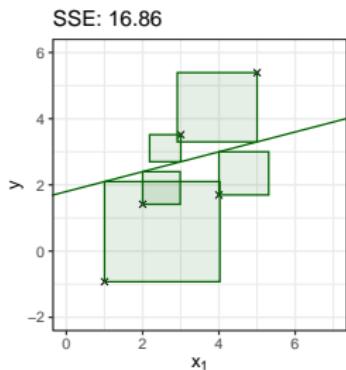
- Consider example with  $n = 5 \rightsquigarrow$  different models with varying SSE

# OPTIMIZATION

- Resulting risk equivalent to **sum of squared errors (SSE)**:

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

- Consider example with  $n = 5 \rightsquigarrow$  different models with varying SSE

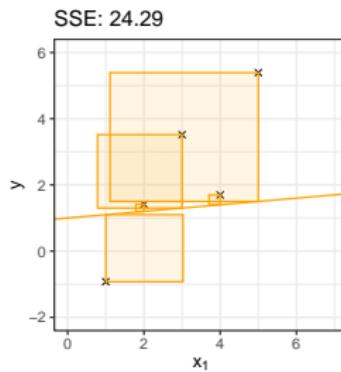
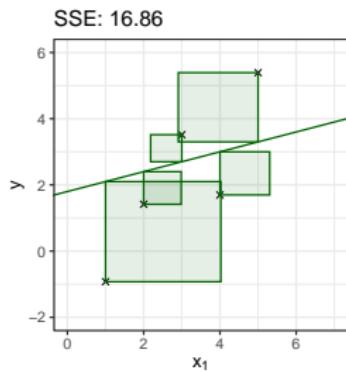


# OPTIMIZATION

- Resulting risk equivalent to **sum of squared errors (SSE)**:

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

- Consider example with  $n = 5 \rightsquigarrow$  different models with varying SSE

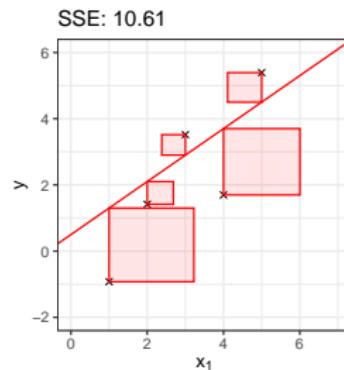
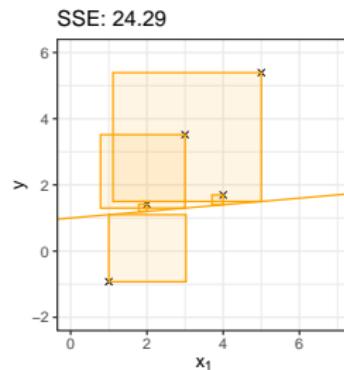
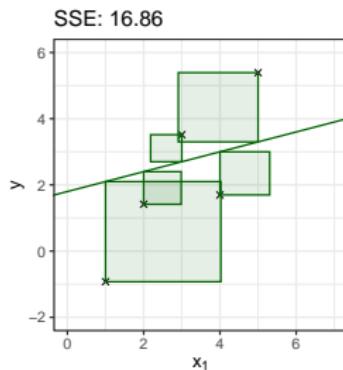


# OPTIMIZATION

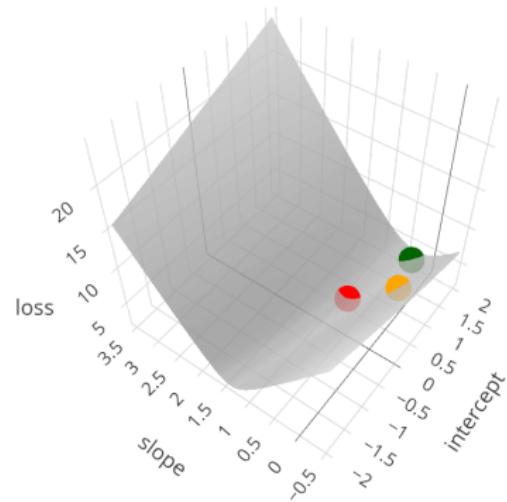
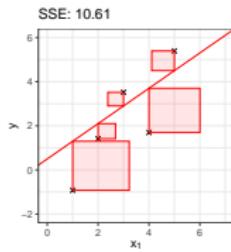
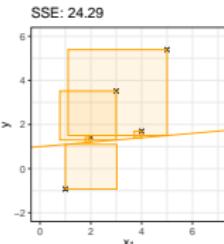
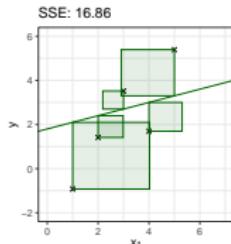
- Resulting risk equivalent to **sum of squared errors (SSE)**:

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

- Consider example with  $n = 5 \rightsquigarrow$  different models with varying SSE

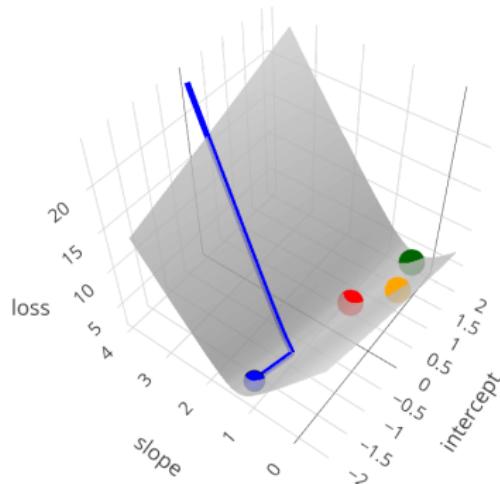
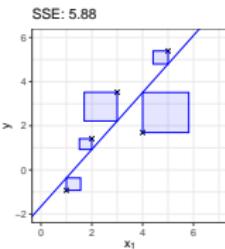
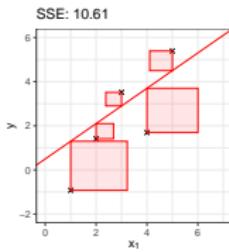
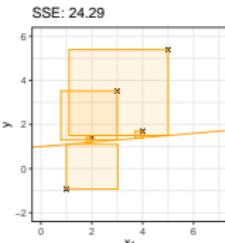
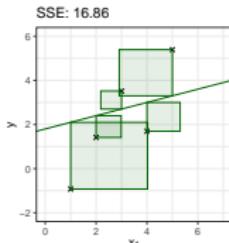


# OPTIMIZATION



Intercept $\theta_0$	Slope $\theta_1$	SSE
1.80	0.30	16.86
1.00	0.10	24.29
0.50	0.80	10.61

# OPTIMIZATION



Intercept $\theta_0$	Slope $\theta_1$	SSE
1.80	0.30	16.86
1.00	0.10	24.29
0.50	0.80	10.61
-1.65	1.29	5.88

Instead of guessing, of course, use **optimization!**

# ANALYTICAL OPTIMIZATION

- Special property of LM with  $L2$  loss: **analytical solution** available

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

- Find via **normal equations**

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

- Solution: **ordinary-least-squares (OLS)** estimator

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

# STATISTICAL PROPERTIES

- LM with  $L_2$  loss intimately related to classical stats LM
- Assumptions
  - $\mathbf{x}^{(i)}$  iid for  $i \in \{1, \dots, n\}$
  - **Homoskedastic** (equivariant) **Gaussian** errors

$$\mathbf{y} = \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2 \mathbf{I})$$

$\rightsquigarrow y_i$  conditionally independent & normal:  $\mathbf{y}|\mathbf{X} \sim \mathcal{N}(\mathbf{X}\boldsymbol{\theta}, \sigma^2 \mathbf{I})$

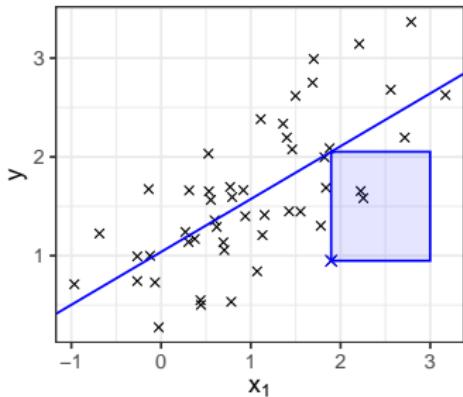
- Uncorrelated features

$\rightsquigarrow$  multicollinearity destabilizes effect estimation
- If assumptions hold: statistical **inference** applicable
  - Hypothesis tests on significance of effects, incl.  $p$ -values
  - Confidence & prediction intervals via student- $t$  distribution
  - Goodness-of-fit measure  $R^2 = 1 - \frac{\text{SSE}}{\sum_{i=1}^n (y^{(i)} - \bar{y})^2}$

$\rightsquigarrow$  SSE = part of data variance *not* explained by model

# Introduction to Machine Learning

## Supervised Regression: Deep Dive: Proof OLS Regression



### Learning goals

- Understand analytical derivation of OLS estimator for LM

# ANALYTICAL OPTIMIZATION

- Special property of LM with  $L2$  loss: **analytical solution** available

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

- Find via **normal equations**

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

- Solution: **ordinary-least-squares (OLS)** estimator

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

# ANALYTICAL OPTIMIZATION – PROOF

$$\mathcal{R}_{\text{emp}}(\theta) = \sum_{i=1}^n (\underbrace{y^{(i)} - \theta^\top \mathbf{x}^{(i)}}_{=: \epsilon_i})^2 = \|\underbrace{\mathbf{y} - \mathbf{X}\theta}_{=: \boldsymbol{\epsilon}}\|_2^2; \quad \theta \in \mathbb{R}^{\tilde{p}} \text{ with } \tilde{p} := p + 1$$

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

$$0 = \frac{\partial}{\partial \theta} \sum_{i=1}^n \epsilon_i^2 \quad | \text{ sum & chain rule}$$

$$0 = \sum_{i=1}^n \frac{\partial \epsilon_i^2}{\partial \epsilon_i} \frac{\partial \epsilon_i}{\partial \theta}$$

$$0 = \sum_{i=1}^n 2\epsilon_i (-1)(\mathbf{x}^{(i)})^\top$$

$$0 = \sum_{i=1}^n (y^{(i)} - \theta^\top \mathbf{x}^{(i)}) (\mathbf{x}^{(i)})^\top$$

$$\sum_{i=1}^n \theta^\top \mathbf{x}^{(i)} (\mathbf{x}^{(i)})^\top = \sum_{i=1}^n y^{(i)} (\mathbf{x}^{(i)})^\top \quad | \text{ transpose}$$

$$\theta \sum_{i=1}^n (\underbrace{(\mathbf{x}^{(i)})^\top \mathbf{x}^{(i)}}_{1 \times 1}) = \sum_{i=1}^n \mathbf{x}^{(i)} y^{(i)}$$

$$\theta = \underbrace{\sum_{i=1}^n \left( \underbrace{(\mathbf{x}^{(i)})^\top \mathbf{x}^{(i)}}_{1 \times 1} \right)^{-1} \underbrace{\mathbf{x}^{(i)} y^{(i)}}_{\tilde{p} \times 1}}_{\tilde{p} \times 1}$$

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

$$0 = \frac{\partial \|\boldsymbol{\epsilon}\|_2^2}{\partial \theta}$$

$$0 = \frac{\partial \boldsymbol{\epsilon}^\top \boldsymbol{\epsilon}}{\partial \theta} \quad | \text{ chain rule}$$

$$0 = \frac{\partial \boldsymbol{\epsilon}^\top \boldsymbol{\epsilon}}{\partial \boldsymbol{\epsilon}} \cdot \frac{\partial \boldsymbol{\epsilon}}{\partial \theta}$$

$$0 = 2\boldsymbol{\epsilon}^\top \cdot (-1 \cdot \mathbf{X})$$

$$0 = (\mathbf{y} - \mathbf{X}\theta)^\top \mathbf{X}$$

$$0 = \mathbf{y}^\top \mathbf{X} - \theta^\top \mathbf{X}^\top \mathbf{X}$$

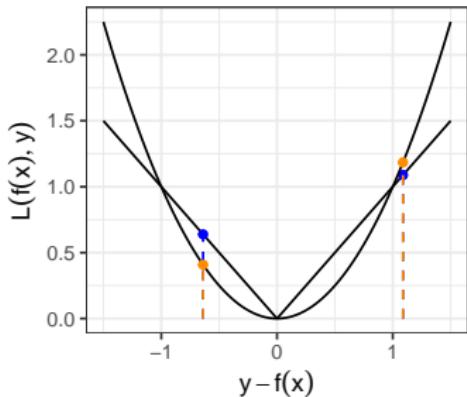
$$\theta^\top \mathbf{X}^\top \mathbf{X} = \mathbf{y}^\top \mathbf{X} \quad | \text{ transpose}$$

$$\mathbf{X}^\top \mathbf{X} \theta = \mathbf{X}^\top \mathbf{y}$$

$$\theta = \underbrace{(\mathbf{X}^\top \mathbf{X})^{-1}}_{\tilde{p} \times \tilde{p}} \underbrace{\mathbf{X}^\top}_{\tilde{p} \times n} \underbrace{\mathbf{y}}_{n \times 1}$$

# Introduction to Machine Learning

## Supervised Regression: Linear Models with $L_1$ Loss

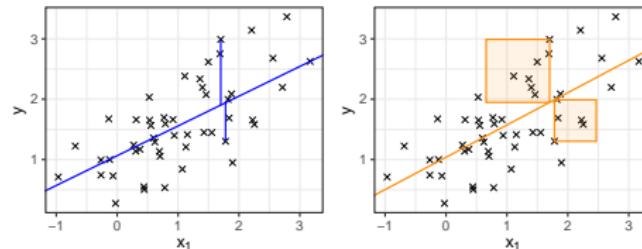


### Learning goals

- Understand difference between  $L_1$  and  $L_2$  regression
- See how choice of loss affects optimization & robustness

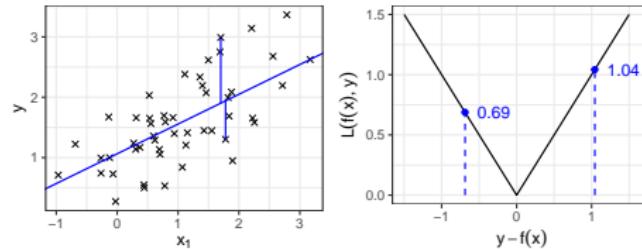
# ABSOLUTE LOSS

- $L_2$  regression minimizes quadratic residuals – wouldn't **absolute** residuals seem more natural?

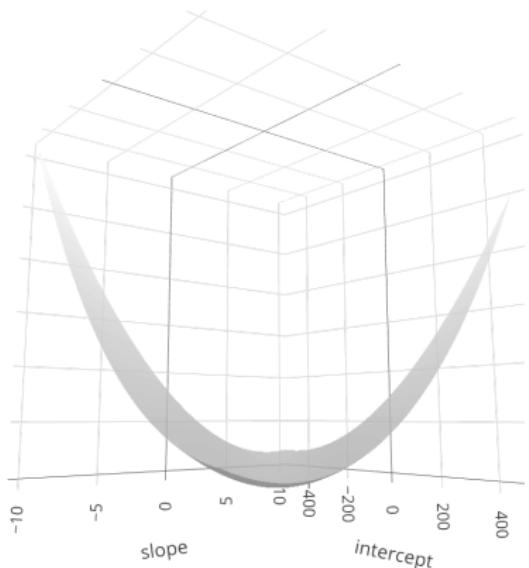
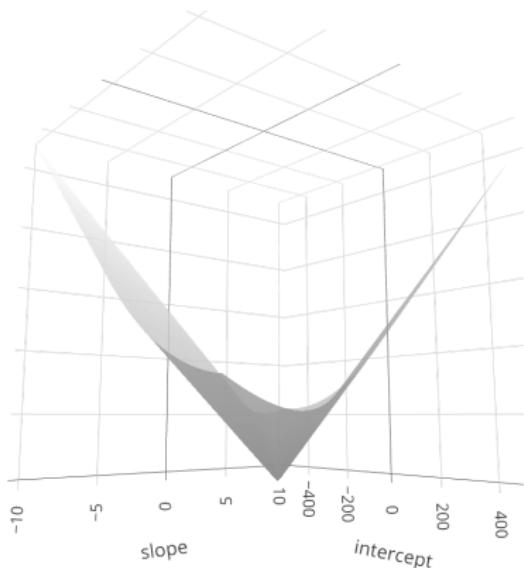


- $L_1$  loss / absolute error / least absolute deviation (LAD)

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



# $L1$ VS $L2$ – LOSS SURFACE



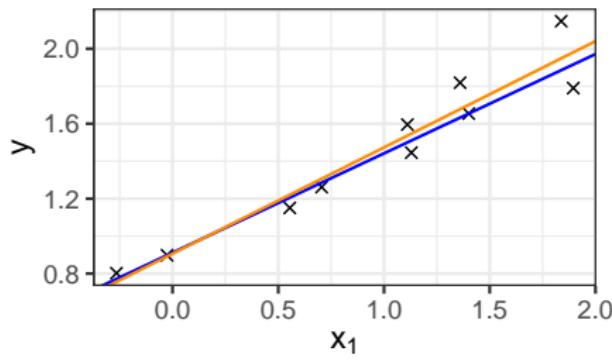
$L1$  loss (left) harder to optimize than  $L2$  loss (right)

- Convex but **not differentiable** in  $y - f(\mathbf{x}) = 0$
- No analytical solution

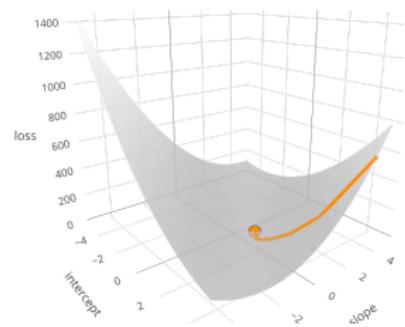
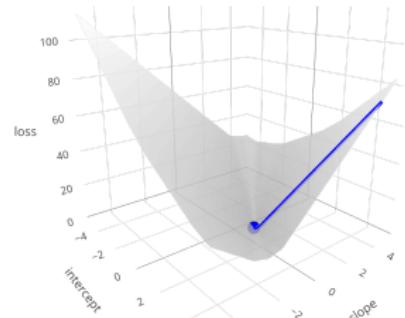
# $L1$ VS $L2$ – ESTIMATED PARAMETERS

- Results of  $L1$  and  $L2$  regression often not that different
- Simulated data:  $y^{(i)} = 1 + 0.5x_1^{(i)} + \epsilon^{(i)}$ ,  $\epsilon^{(i)} \stackrel{i.i.d}{\sim} \mathcal{N}(0, 0.01)$

	intercept	slope
$L1$	0.91	0.53
$L2$	0.91	0.57

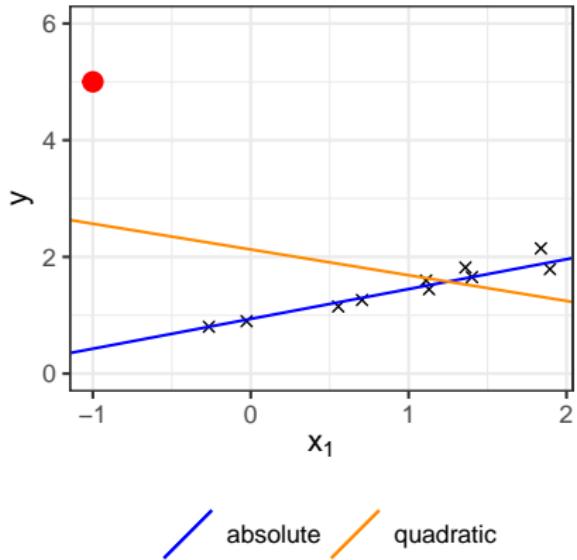
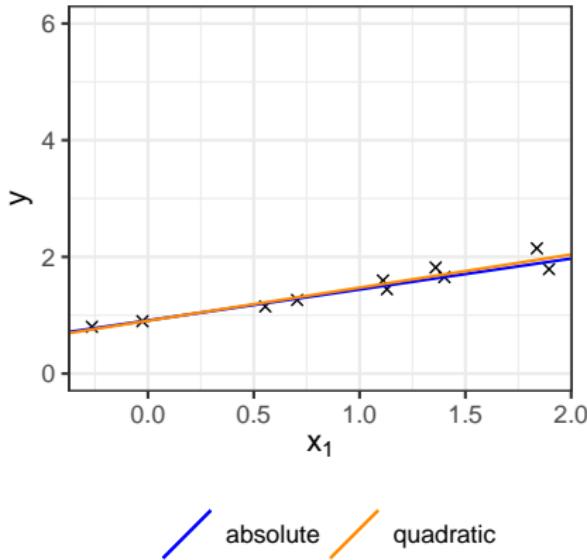


absolute   quadratic



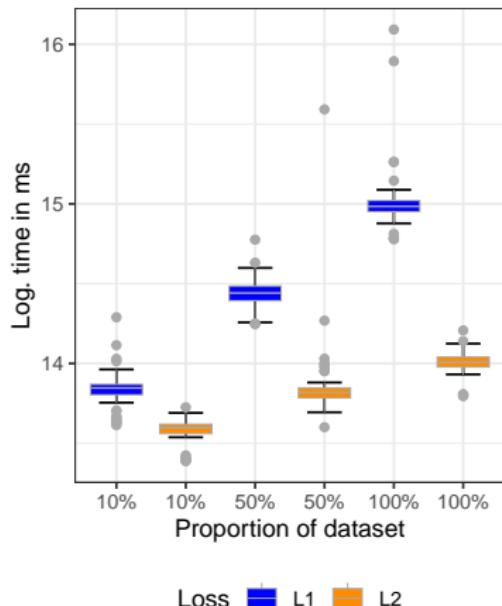
# $L1$ VS $L2$ – ROBUSTNESS

- $L2$  quadratic in residuals  $\rightsquigarrow$  outlying points carry lots of weight
- E.g.,  $3 \times$  residual  $\Rightarrow 9 \times$  loss contribution
- $L1$  more **robust** in presence of outliers (example ctd.):



# L1 VS L2 – OPTIMIZATION COST

- Real-world weather problem  $\rightsquigarrow$  predict mean temperature
- Compare **time** to fit  $L1$  (`quantreg::rq()`) vs  $L2$  (`lm::lm()`) for different dataset proportions (repeat 50 $\times$ )



Loss

	Fitted: $L1$	Fitted: $L2$
Total $L1$ loss	$8.98 \times 10^4$	$8.99 \times 10^4$
Total $L2$ loss	$5.83 \times 10^6$	$5.81 \times 10^6$

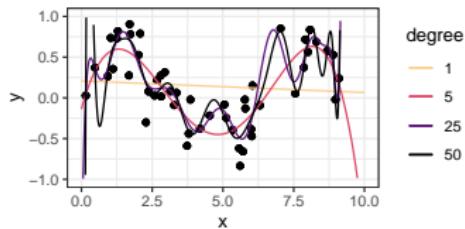
Estimated coefficients

$x_j$	$L1: \hat{\theta}_j$	$L2: \hat{\theta}_j$
Max_temperature	0.553	0.563
Min_temperature	0.441	0.427
Visibility	0.026	0.041
Wind_speed	0.002	0.010
Max_wind_speed	-0.026	-0.039
(Intercept)	-0.380	-0.102

**$L1$  slower to optimize!**

# Introduction to Machine Learning

## Supervised Regression: Polynomial Regression Models

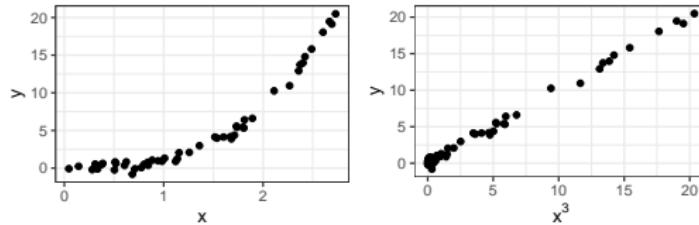


### Learning goals

- Learn about general form of linear model
- See how to add flexibility by using polynomials
- Understand that more flexibility is not necessarily better

# INCREASING FLEXIBILITY

- Recall our definition of LM: model  $y$  as linear combo of features
- But: isn't that pretty **inflexible**?
- E.g., here,  $y$  does not seem to be a linear function of  $x$ ...



... but relation to  $x^3$  looks pretty linear!

- Many other trasfos conceivable, e.g.,  $\sin(x_1)$ ,  $\max(0, x_2)$ ,  $\sqrt{x_3}$ , ...
- Turns out we can use LM much more **flexibly** (and: it's still linear)  
~~ interpretation might get less straightforward, though

# THE LINEAR MODEL

- Recall what we previously defined as LM:

$$f(x) = \theta_0 + \sum_{j=1}^p \theta_j x_j = \theta_0 + \theta_1 x_1 + \cdots + \theta_p x_p \quad (1)$$

- Actually, just special case of "true" LM
- The linear model with basis functions  $\phi_j$ :

$$f(\mathbf{x}) = \theta_0 + \sum_{j=1}^p \theta_j \phi_j(x_j) = \theta_0 + \theta_1 \phi_1(x_1) + \cdots + \theta_p \phi_p(x_p)$$

- In Eq. 1, we implicitly use identity trafo:  $\phi_j = \text{id}_x : x \mapsto x \quad \forall j$   
~~~ we often say LM and imply  $\phi_j = \text{id}_x$

# THE LINEAR MODEL

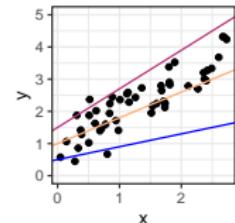
- Are models like  $f(\mathbf{x}) = \theta_0 + \theta_1 x^2$  **really linear?**

- Certainly not in covariates:

$$\begin{aligned} a \cdot f(x, \boldsymbol{\theta}) + b \cdot f(x_*, \boldsymbol{\theta}) &= \theta_0(a+b) + \theta_1(ax^2 + bx_*^2) \\ &\neq \theta_0 + \theta_1(ax + bx_*)^2 \\ &= f(ax + bx_*, \boldsymbol{\theta}) \end{aligned}$$

- Crucially, however, **linear in params**:

$$\begin{aligned} a \cdot f(x, \boldsymbol{\theta}) + b \cdot f(x, \boldsymbol{\theta}^*) &= a\theta_0 + b\theta_0^* + (a\theta_1 + b\theta_1^*)x^2 \\ &= f(x, a\boldsymbol{\theta} + b\boldsymbol{\theta}^*) \end{aligned}$$



$$\begin{aligned} \boldsymbol{\theta} &= (0.5, 0.4)^T \\ \boldsymbol{\theta} &= (1.0, 0.8)^T \\ \boldsymbol{\theta} &= (1.5, 1.2)^T \end{aligned}$$

- NB: we still call design matrix  $\mathbf{X}$ , incorporating possible trasfos:

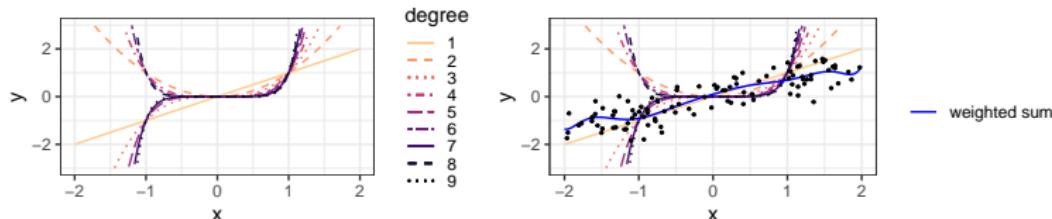
$$\mathbf{X} = \begin{pmatrix} 1 & \phi_1(x_1^{(1)}) & \dots & \phi_p(x_p^{(1)}) \\ \vdots & \vdots & & \vdots \\ 1 & \phi_1(x_1^{(n)}) & \dots & \phi_p(x_p^{(n)}) \end{pmatrix}$$

~~~ solution via normal equations as usual

# POLYNOMIAL REGRESSION

- Simple & flexible choice for basis funs:  **$d$ -polynomials**
- Idea: map  $x_j$  to (weighted) sum of its monomials up to order  $d \in \mathbb{N}$

$$\phi^{(d)} : \mathbb{R} \rightarrow \mathbb{R}, \quad x_j \mapsto \sum_{k=1}^d \beta_k x_j^k$$

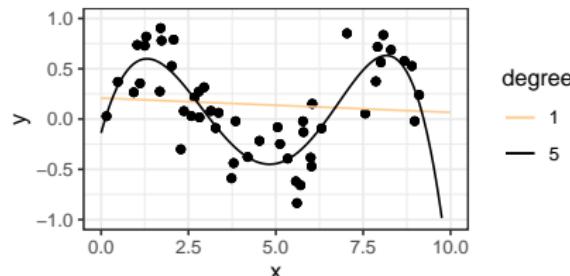


- How to estimate coefficients  $\beta_k$ ?
  - Both LM & polynomials **linear** in their params  $\rightsquigarrow$  merge
  - E.g.,  $f(\mathbf{x}) = \theta_0 + \theta_1 \phi^{(d)}(\mathbf{x}) = \theta_0 + \sum_{k=1}^d \theta_{1,k} x^k$

$$\rightsquigarrow \mathbf{X} = \begin{pmatrix} 1 & x^{(1)} & (x^{(1)})^2 & \dots & (x^{(1)})^d \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x^{(n)} & (x^{(n)})^2 & \dots & (x^{(n)})^d \end{pmatrix}, \quad \boldsymbol{\theta} \in \mathbb{R}^{d+1}$$

# POLYNOMIAL REGRESSION – EXAMPLES

Univariate regression,  $d \in \{1, 5\}$



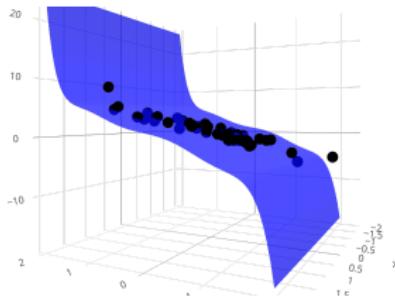
- Data-generating process:

$$y = 0.5 \sin(x) + \epsilon,$$
$$\epsilon \sim \mathcal{N}(0, 0.3^2)$$

- Model:

$$f(x) = \theta_0 + \sum_{k=1}^d \theta_{1,k} x^k$$

Bivariate regression,  $d = 7$



- Data-generating process:

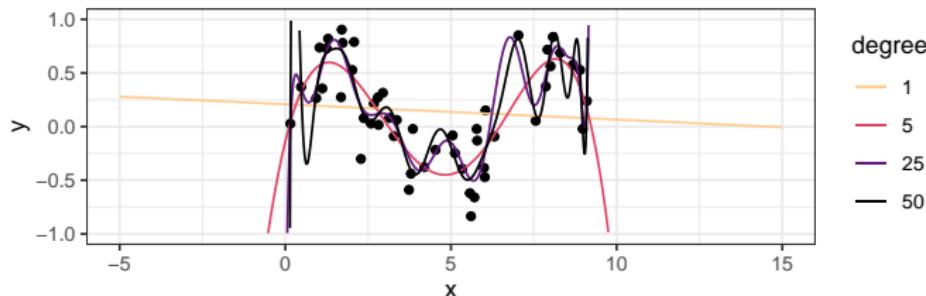
$$y = 1 + 2x_1 + x_2^3 + \epsilon,$$
$$\epsilon \sim \mathcal{N}(0, 0.5^2)$$

- Model:

$$f(x) = \theta_0 + \theta_1 x_1 + \sum_{k=1}^7 \theta_{2,k} x_2^k$$

# COMPLEXITY OF POLYNOMIALS

- Higher  $d$  allows to learn more complex functions  
~~ richer hyp space / higher **capacity**



- Should we then simply let  $d \rightarrow \infty$ ?
  - No: data contains random **noise** – not part of true DGP
  - Model with overly high capacity learns all those spurious patterns ~~ poor generalization to new data
  - Also, higher  $d$  can lead to oscillation esp. at bounds (Runge's phenomenon<sup>1</sup>)

---

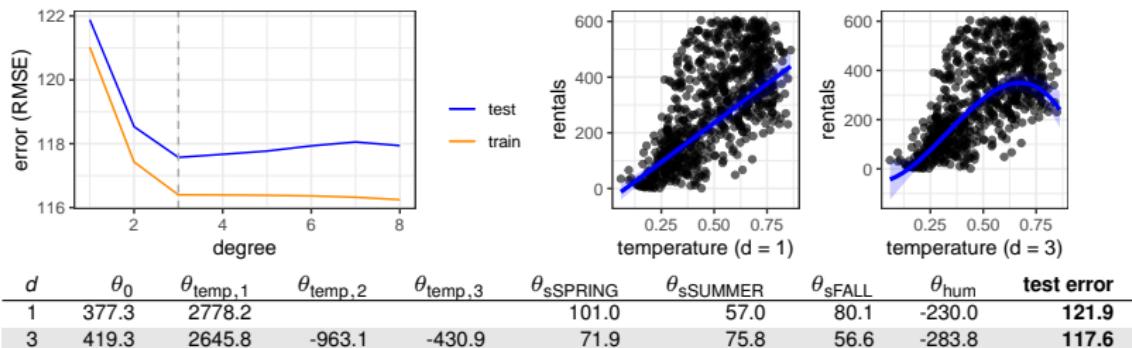
<sup>1</sup> Interpolation of  $m$  equidistant points with  $d$ -polynomial only well-conditioned for  $d < 2\sqrt{m}$ .  
Plot: 50 points, models with  $d \geq 14$  unstable (under equidistance assumption).

# BIKE RENTAL EXAMPLE

- OpenML task `dailybike`: predict `rentals` from weather conditions
- Hunch: non-linear effect of temperature  $\rightsquigarrow$  include with polynomial:

$$f(\mathbf{x}) = \sum_{k=1}^d \theta_{\text{temperature},k} x_{\text{temperature}}^k + \theta_{\text{season}} x_{\text{season}} + \theta_{\text{humidity}} x_{\text{humidity}}$$

- Test error<sup>2</sup> confirms suspicion  $\rightsquigarrow$  minimal for  $d = 3$



- Conclusion: flexible effects can improve fit/performance

<sup>2</sup> Reliable insights about model performance only via separate test dataset not used during training (here computed via 10-fold *cross validation*). Much more on this in Evaluation chapter.

# INTRODUCTION TO MACHINE LEARNING

ML Basics

Supervised Regression

## **Supervised Classification**

Performance Evaluation

k-NN

Classification and Regression Trees (CART)

Random Forests

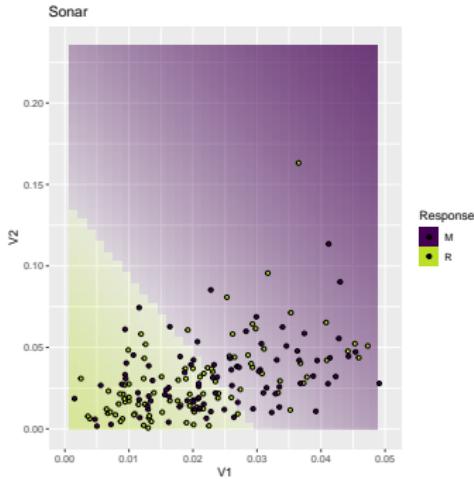
Neural Networks

Tuning

Nested Resampling

# Introduction to Machine Learning

## Classification: Tasks

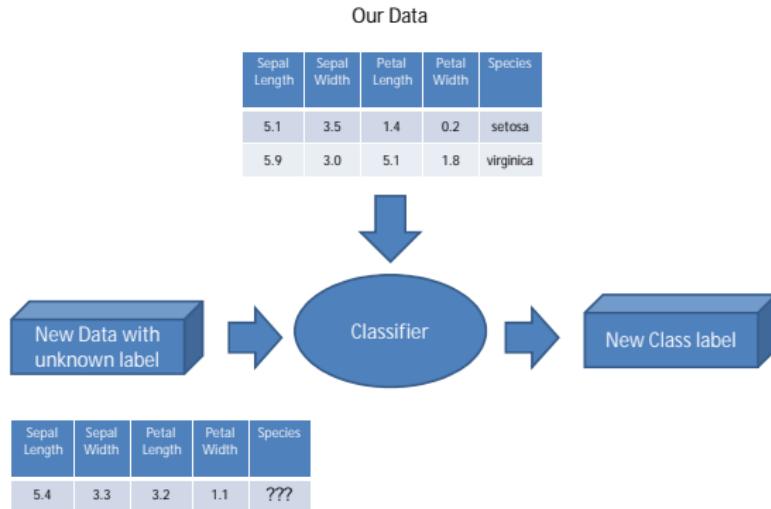


### Learning goals

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

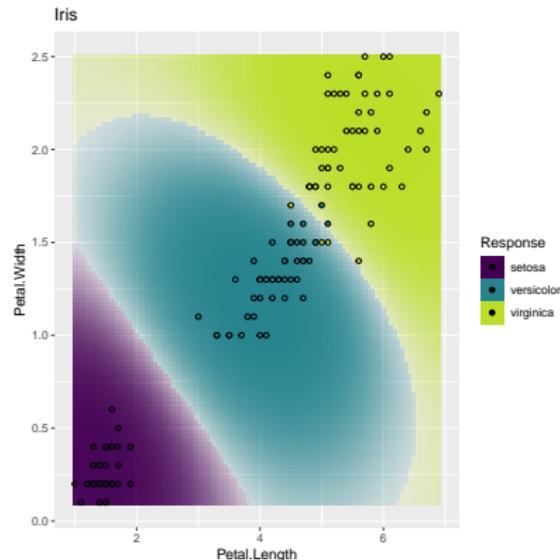
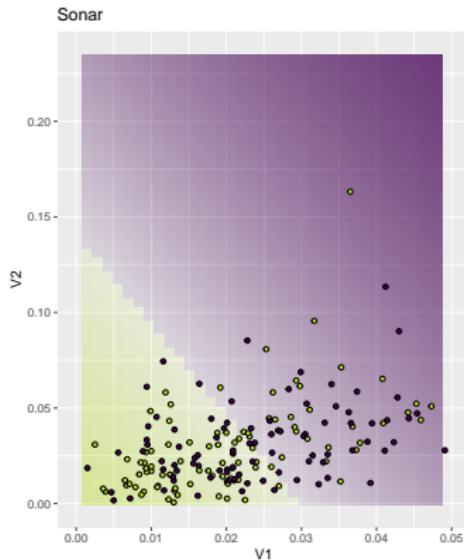
# CLASSIFICATION

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.*

### How reliable?

Supporters claim an 85-95 percent accuracy rate

Critics say there is not enough scientific evidence to say whether it detects lies or not

### What a polygraph measures

- Body movements
- Breathing (diaphragm)
- Breathing (chest)
- Perspiration
- Pulse, blood pressure

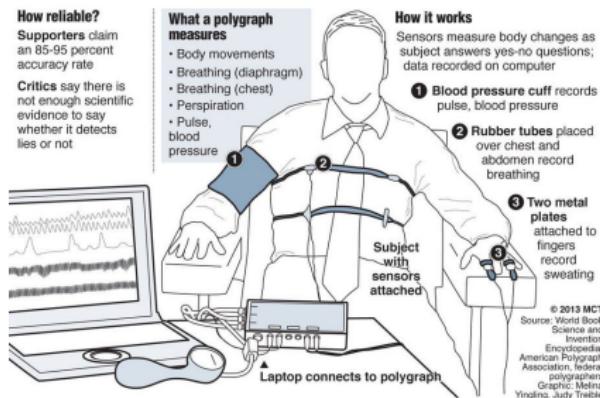
### How it works

Sensors measure body changes as subject answers yes-no questions; data recorded on computer

① Blood pressure cuff records pulse, blood pressure

② Rubber tubes placed over chest and abdomen record breathing

③ Two metal plates attached to fingers record sweating



© 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



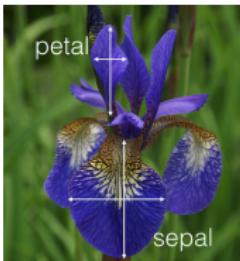
Virginica

Source:

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

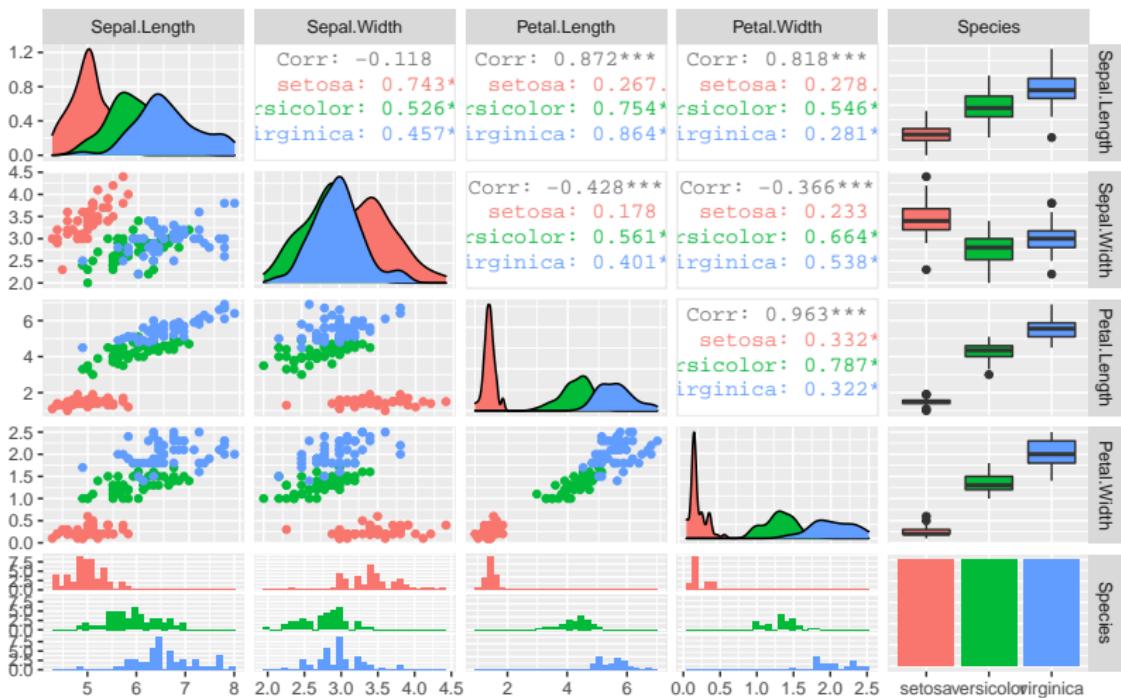
# MULTICLASS TASK - IRIS

- 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 |

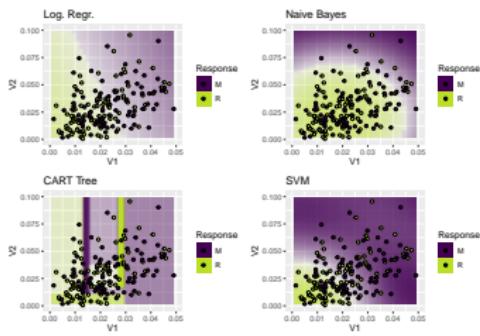
# MULTICLASS TASK - IRIS



# 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**

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

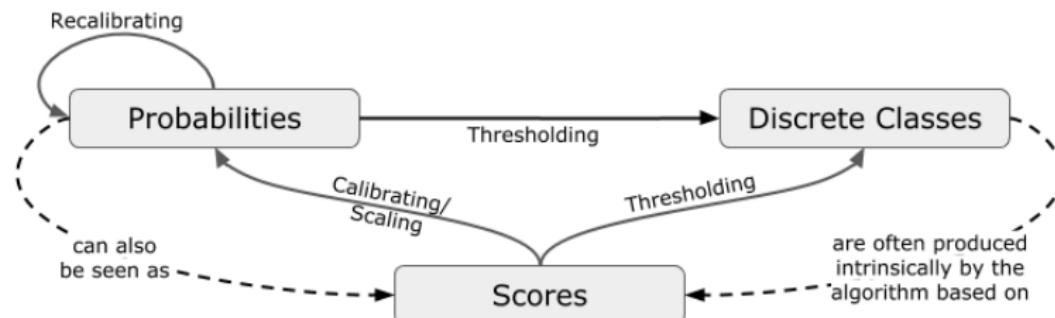
- Probabilities  $\pi_1(\mathbf{x}), \dots, \pi_g(\mathbf{x})$  are transformed into labels by predicting the class with the maximum probability

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

- 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**

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

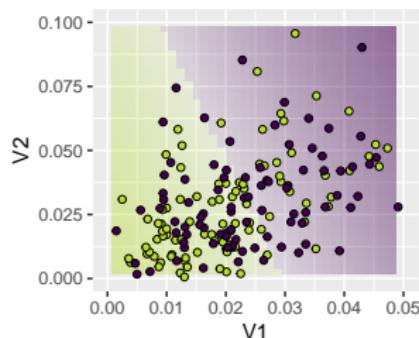
In the binary case we can simplify and generalize to the decision boundary for general threshold  $c$ :

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

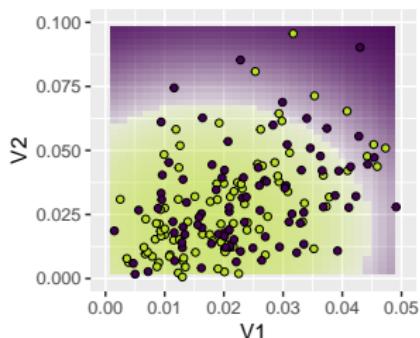
If we set  $c = 0$  for scores and  $c = 0.5$  for probabilities, this is consistent with the definition above.

# DECISION BOUNDARY EXAMPLES

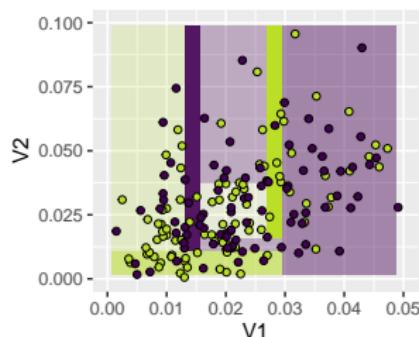
Log. Regr.



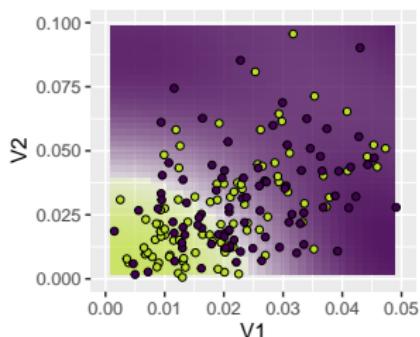
Naive Bayes



CART Tree



SVM



# CLASSIFICATION APPROACHES

Two fundamental approaches exist to construct classifiers:

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

They tackle the classification problem from different angles:

- **Generative** classification approaches assume a data-generating process in which the distribution of the features  $\mathbf{x}$  is different for the various classes of the output  $y$ , and try to learn these conditional distributions:  
“Which  $y$  tends to have  $\mathbf{x}$  like these?”
- **Discriminant** approaches use **empirical risk minimization** based on a suitable loss function:  
“What is the best prediction for  $y$  given these  $\mathbf{x}$ ? ”

# GENERATIVE APPROACH

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

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

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

Examples:

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

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

# DISCRIMINANT APPROACH

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

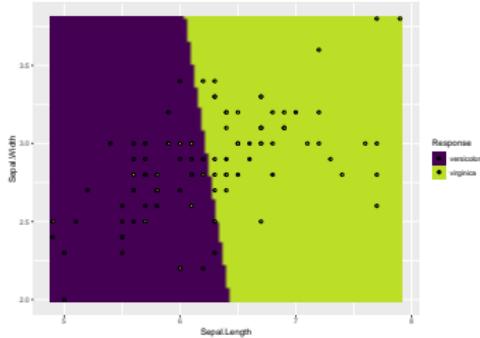
$$\hat{f} = \arg \min_{f \in \mathcal{H}} \mathcal{R}_{\text{emp}}(f) = \arg \min_{f \in \mathcal{H}} \sum_{i=1}^n L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right).$$

Examples:

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

# Introduction to Machine Learning

## Classification: Linear Classifiers



### Learning goals

- Know the definition of a linear classifier

# LINEAR CLASSIFIERS

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

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

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

NB:  $\mathbf{w}_k$  and  $b_k$  do not directly refer to the parameters  $\theta_k$  of  $k$ -th scoring function  $f_k$  but the transformed version.

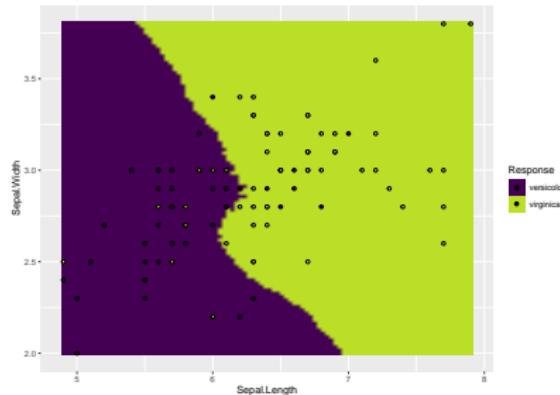
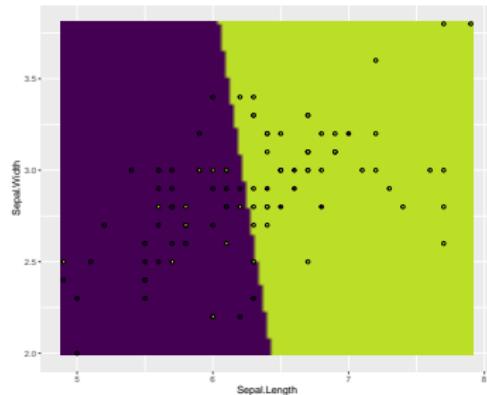
# LINEAR CLASSIFIERS

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:

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

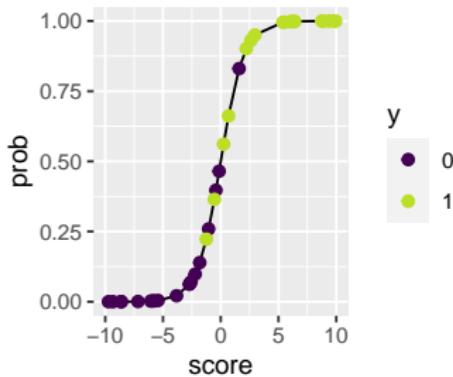
This is a **hyperplane** separating two classes.

# 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

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

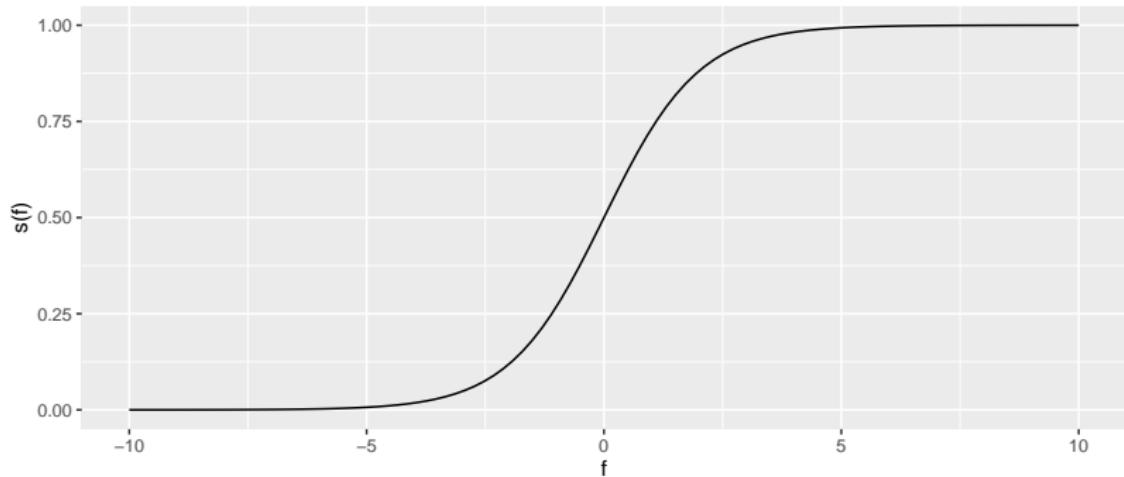
NB: We will often suppress the intercept in notation.

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

# LOGISTIC FUNCTION

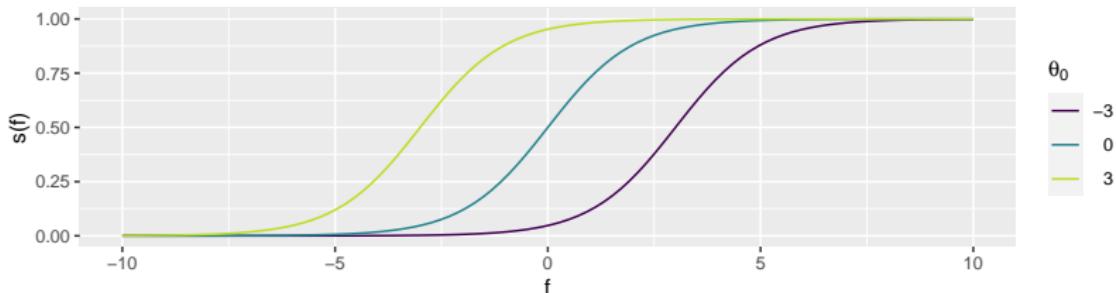
To avoid this, logistic regression “squashes” the estimated linear scores  $\theta^T \mathbf{x}$  to  $[0, 1]$  through the **logistic function**  $s$ :

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

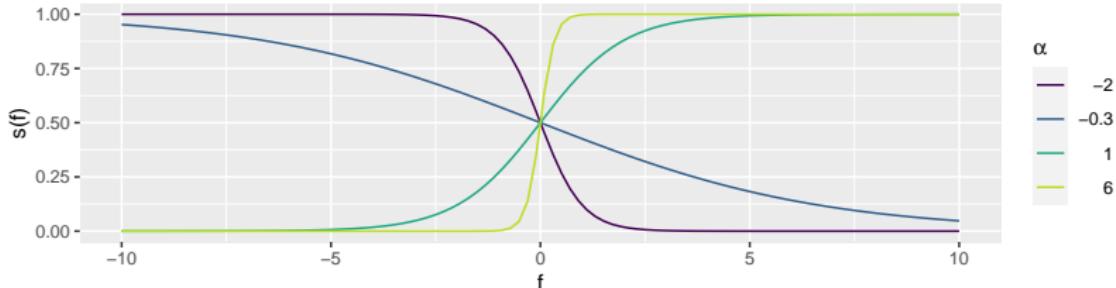


# LOGISTIC FUNCTION

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



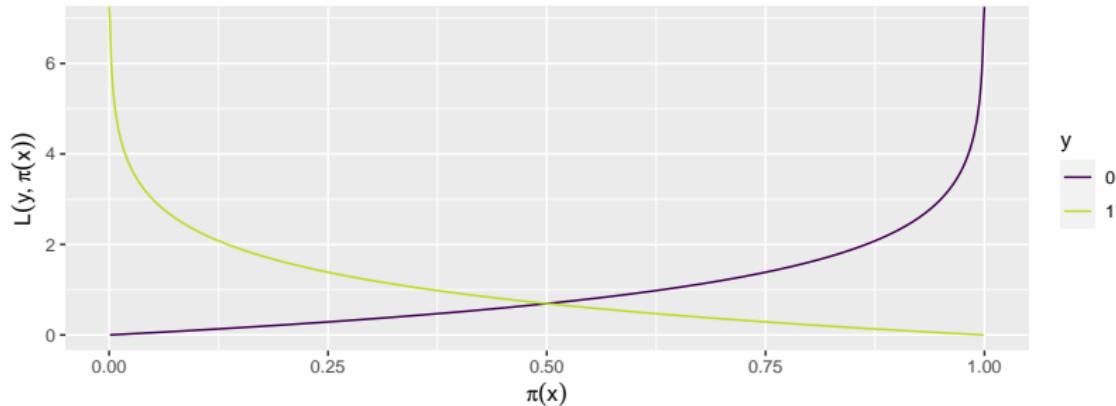
Scaling  $f$  like  $s(\alpha f) = \frac{\exp(\alpha f)}{1 + \exp(\alpha f)}$  controls the slope and direction.



# BERNOULLI / LOG LOSS

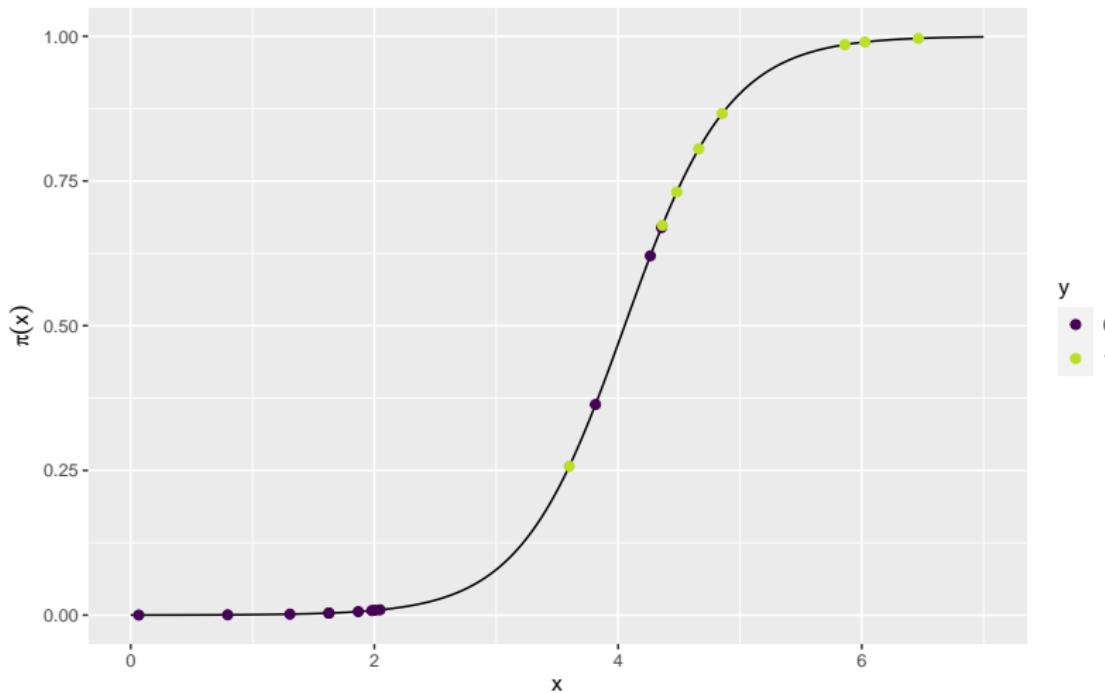
We need to define a loss function for the ERM approach:

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



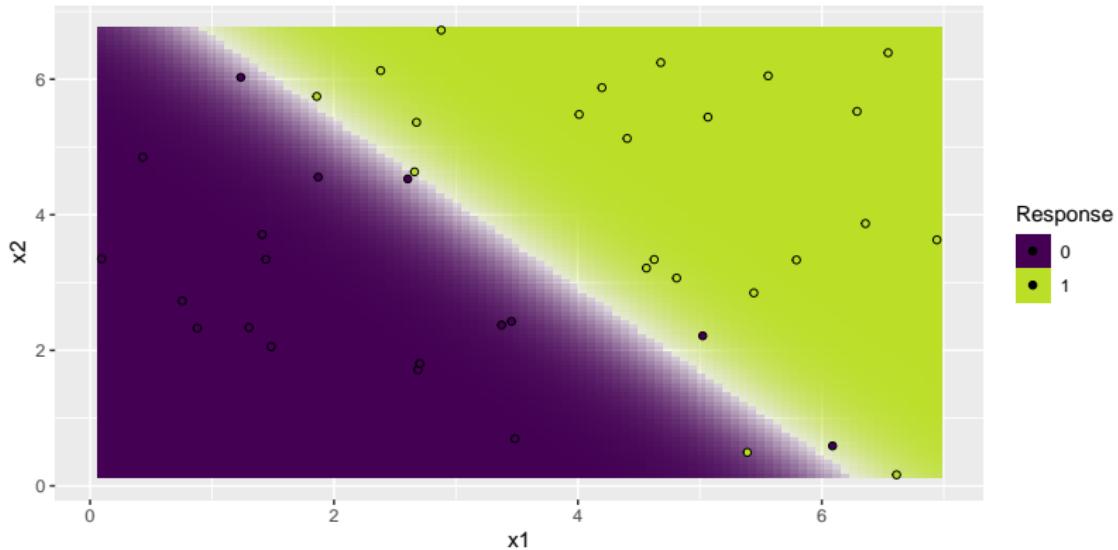
# LOGISTIC REGRESSION IN 1D

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

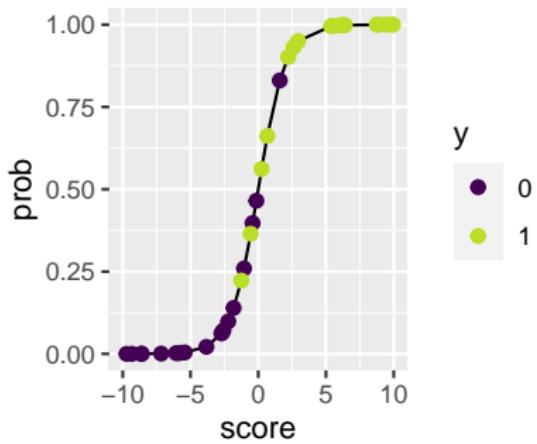
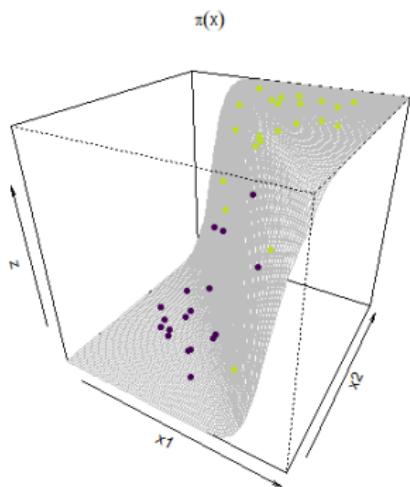


# LOGISTIC REGRESSION IN 2D

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.

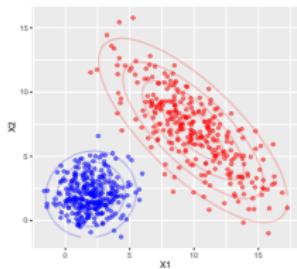
$$L(y, \pi(\mathbf{x})) = -y \log(\pi(\mathbf{x})) - (1 - y) \log(1 - \pi(\mathbf{x}))$$

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

# Introduction to Machine Learning

## Classification: Discriminant Analysis

### Learning goals



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

# LINEAR DISCRIMINANT ANALYSIS (LDA)

LDA follows a generative approach

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

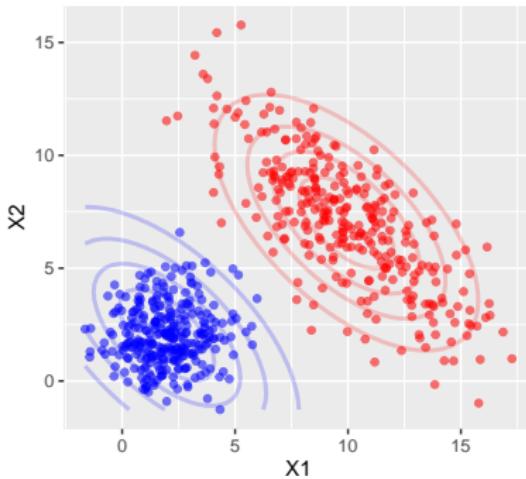
where we now have to pick a distributional form for  $p(\mathbf{x}|y = k)$ .

# LINEAR DISCRIMINANT ANALYSIS (LDA)

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

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

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



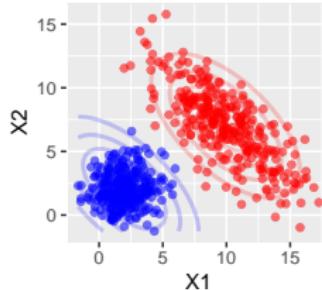
# LINEAR DISCRIMINANT ANALYSIS (LDA)

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

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

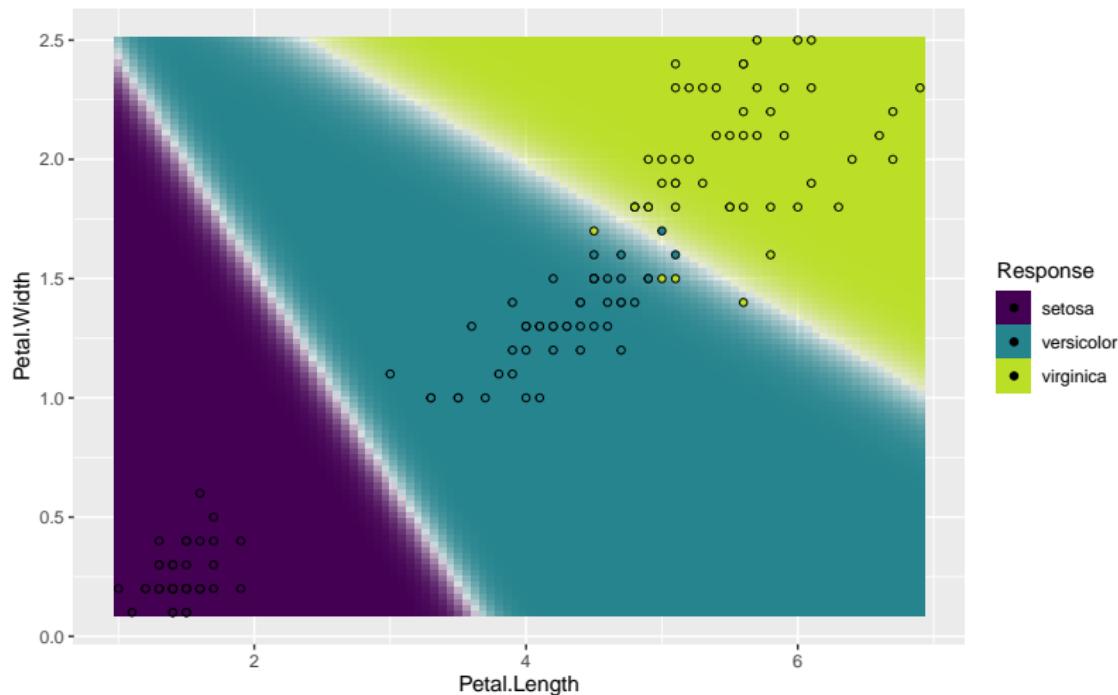
$$\hat{\mu}_k = \frac{1}{n_k} \sum_{i:y^{(i)}=k} \mathbf{x}^{(i)}$$

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



# LDA AS LINEAR CLASSIFIER

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



# LDA AS LINEAR CLASSIFIER

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

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

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

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

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

# LDA AS LINEAR CLASSIFIER

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

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

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

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

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

# QUADRATIC DISCRIMINANT ANALYSIS (QDA)

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

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

Parameters are estimated in a straightforward manner by:

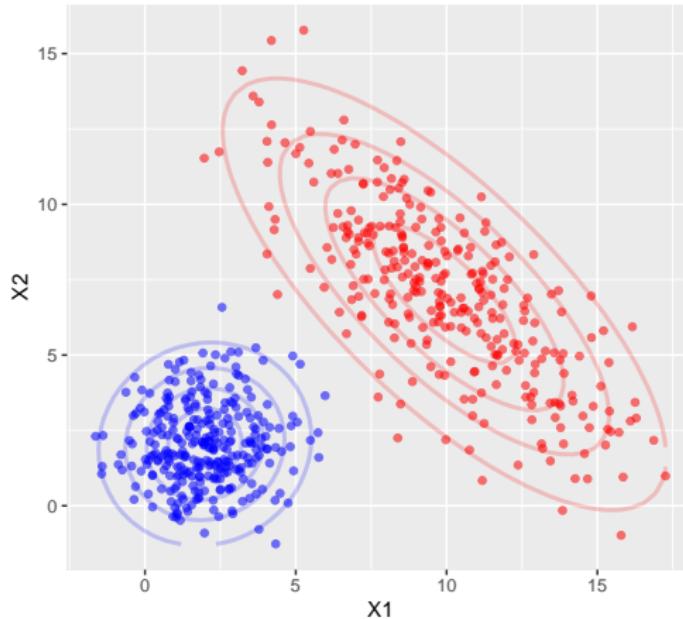
$$\hat{\pi}_k = \frac{n_k}{n}, \text{ where } n_k \text{ is the number of class-}k \text{ observations}$$

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

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

# QUADRATIC DISCRIMINANT ANALYSIS (QDA)

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



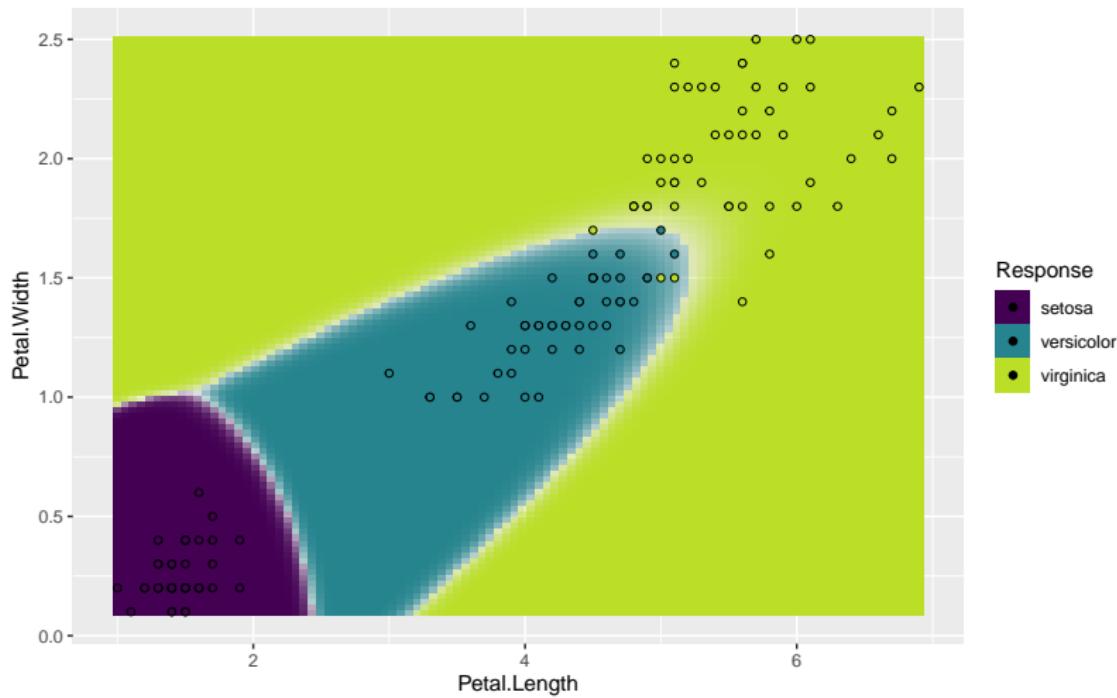
# QUADRATIC DISCRIMINANT ANALYSIS (QDA)

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

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

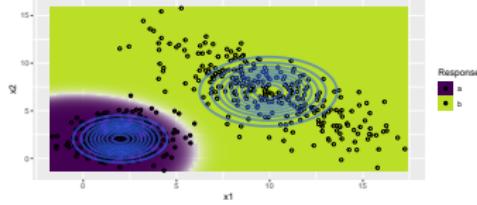
$$\log \pi_k - \frac{1}{2} \log |\Sigma_k| - \frac{1}{2} \boldsymbol{\mu}_k^T \Sigma_k^{-1} \boldsymbol{\mu}_k + \mathbf{x}^T \Sigma_k^{-1} \boldsymbol{\mu}_k - \frac{1}{2} \mathbf{x}^T \Sigma_k^{-1} \mathbf{x}$$

# QUADRATIC DISCRIMINANT ANALYSIS (QDA)



# Introduction to Machine Learning

## Classification: Naive Bayes



### Learning goals

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

# NAIVE BAYES CLASSIFIER

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

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

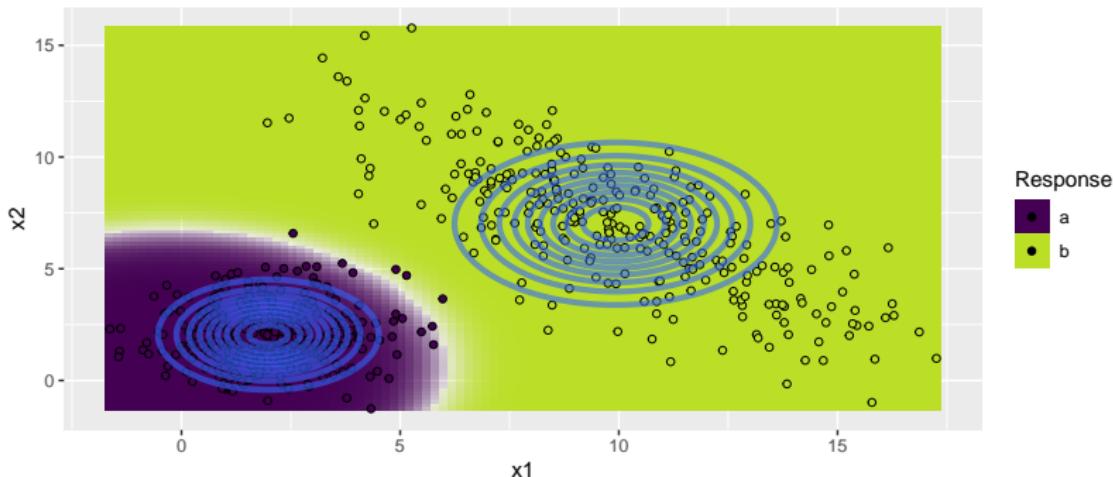
NB is based on a simple **conditional independence assumption**: the features are conditionally independent given class  $y$ .

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

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

## NB: NUMERICAL FEATURES

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



## NB: CATEGORICAL FEATURES

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

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

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

## LAPLACE SMOOTHING

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

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

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

# NAIVE BAYES: APPLICATION AS SPAM FILTER

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

# INTRODUCTION TO MACHINE LEARNING

ML Basics

Supervised Regression

Supervised Classification

## Performance Evaluation

k-NN

Classification and Regression Trees (CART)

Random Forests

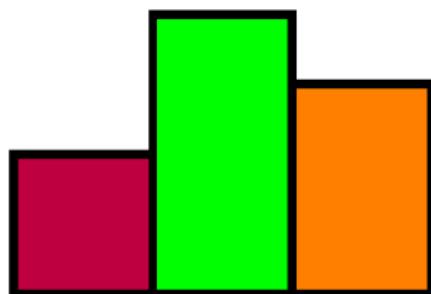
Neural Networks

Tuning

Nested Resampling

# 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):

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

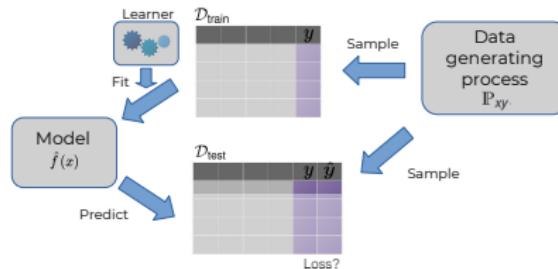
$$\min_{\theta \in \Theta} \sum_{i=1}^n L\left(y^{(i)}, f\left(\mathbf{x}^{(i)} \mid \theta\right)\right)$$

- 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  $m$ )

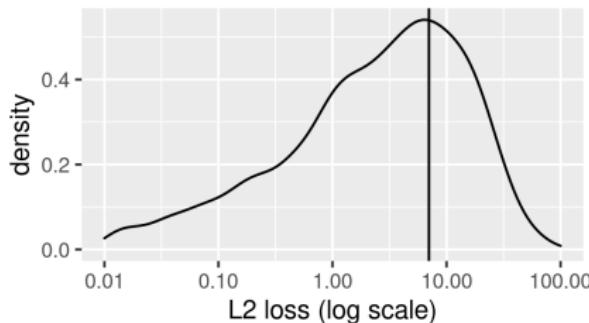
$$\widehat{\text{GE}}(\hat{f}, L) := \frac{1}{m} \sum_{(\mathbf{x}, y) \in \mathcal{D}_{\text{test}}} [L(y, \hat{f}(\mathbf{x}))]$$



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.



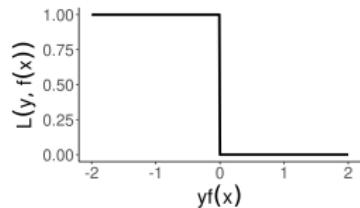
- The result is a unimodal distribution with long tails.
- Mean and one standard deviation to either side are highlighted in grey.

# INNER VS OUTER LOSS

- Sometimes, we would like to evaluate our learner with a different loss  $L$  or metric  $\rho$ .
- 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  $\rho$  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  $\rho_L$ :

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

# 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  $n_{\text{train}}$  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  $\rho_L$ :

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

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

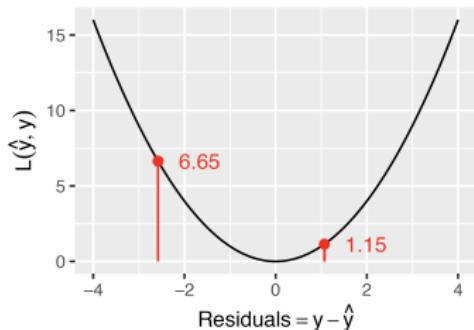
$$\text{GE}(\hat{f}, L) := \text{GE}(\mathcal{I}, \boldsymbol{\lambda}, n_{\text{train}}, \rho_L | \mathcal{D}_{\text{train}})$$

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

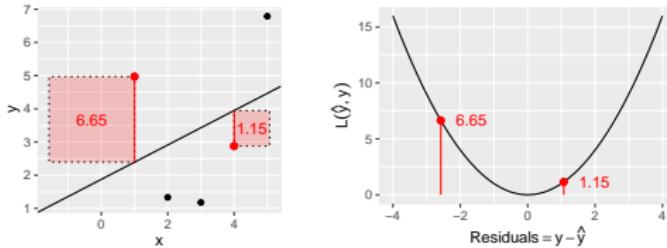


- Know the definitions of mean squared error (MSE) and mean absolute error (MAE)
- Understand the connections of MSE and MAE to L2 and L1 loss
- Know the definition of Spearman's  $\rho$
- Know the definitions of  $R^2$  and generalized  $R^2$

# 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.



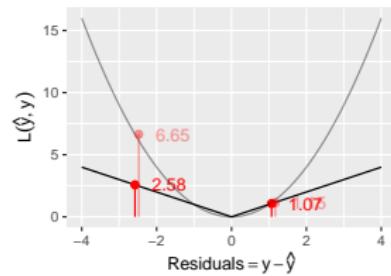
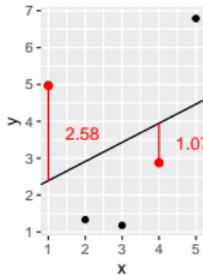
Similar measures:

- 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.



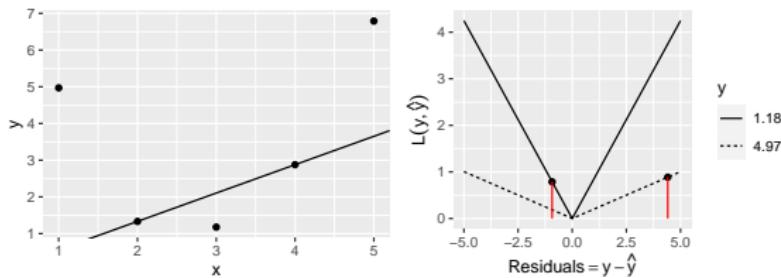
Similar measures:

- 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:

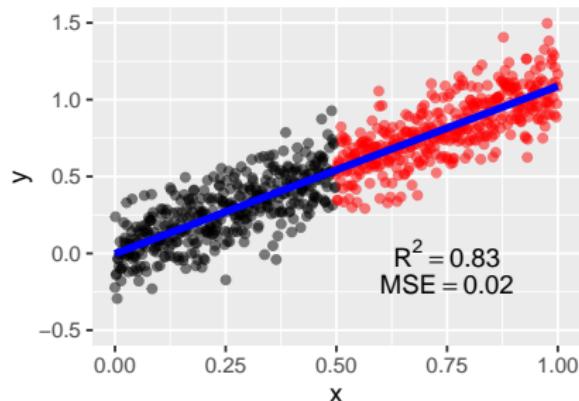
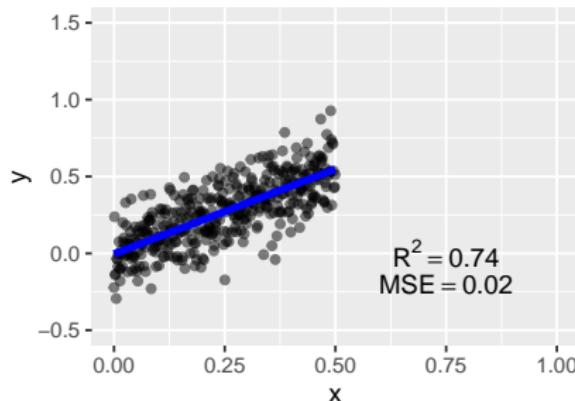
- Mean Absolute Scaled Error (MASE)
- Symmetric Mean Absolute Percentage Error (sMAPE)

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

- 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

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

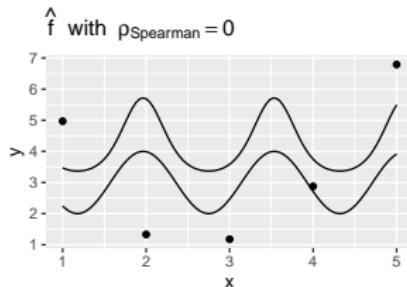
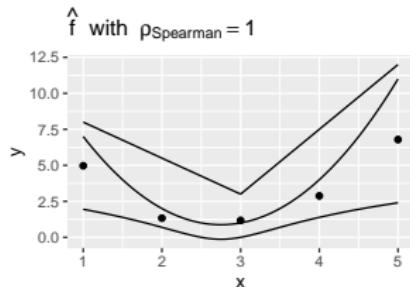
- E.g., model vs constant, LM vs non-linear model, tree vs forest, model with fewer features vs model with more, ...
- We could use arbitrary measures.
- In ML we would rather evaluate on test set.
- Can then become negative, e.g., for SSE and constant baseline, if our model fairs worse on the test set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

# SPEARMAN'S $\rho$

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

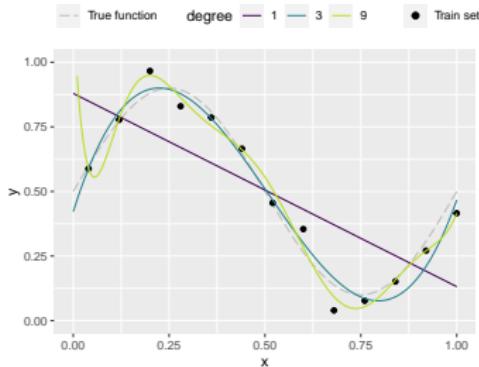
$$\rho_{\text{Spearman}}(\mathbf{y}, \hat{\mathbf{F}}) = \frac{\text{Cov}(\text{rg}(\mathbf{y}), \text{rg}(\hat{\mathbf{y}}))}{\sqrt{\text{Var}(\text{rg}(\mathbf{y}))} \cdot \sqrt{\text{Var}(\text{rg}(\hat{\mathbf{y}}))}} \in [-1, 1],$$

- 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



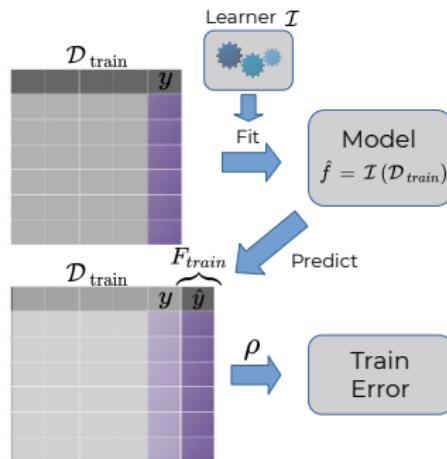
### Learning goals

- 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:

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

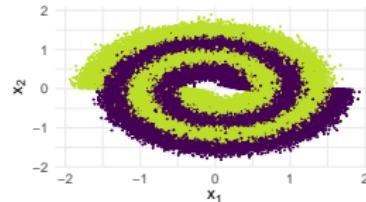


A.k.a. apparent error or resubstitution error.

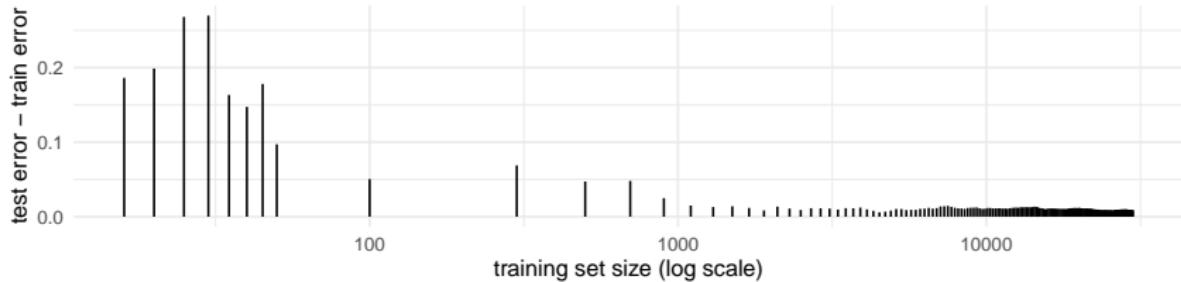
# EXAMPLE 1: KNN

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").

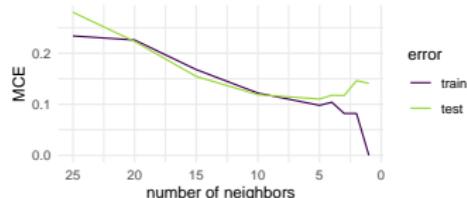


We increase train size, and see how gap between train error and GE closes.

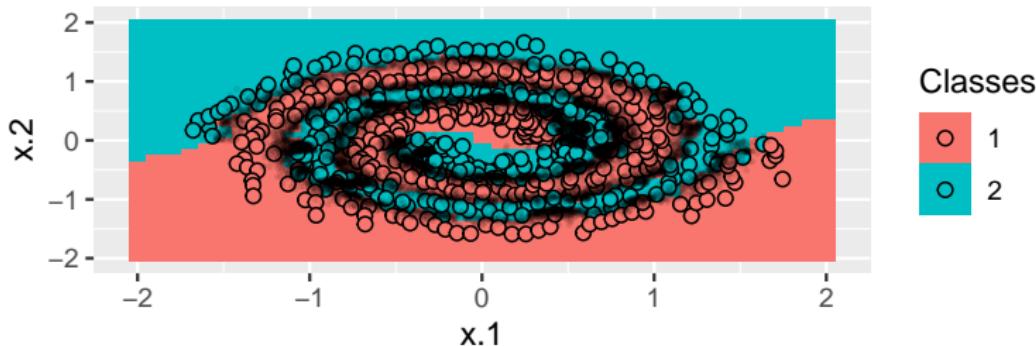


# EXAMPLE 1: KNN

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



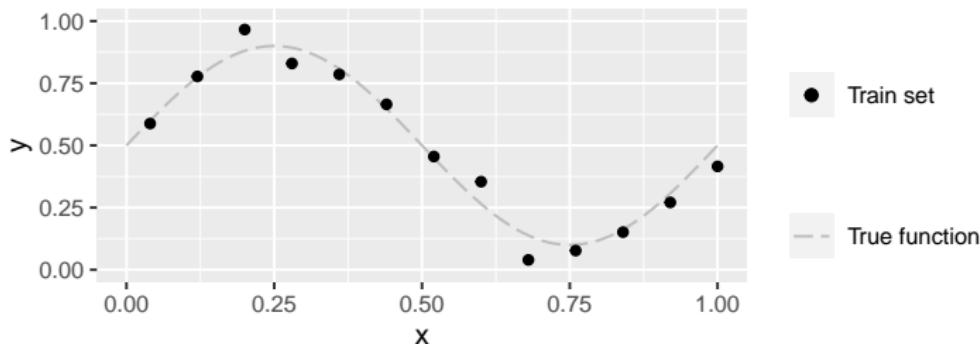
$k = 2$ ; trainerr = 0.08, testerr = 0.14



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:

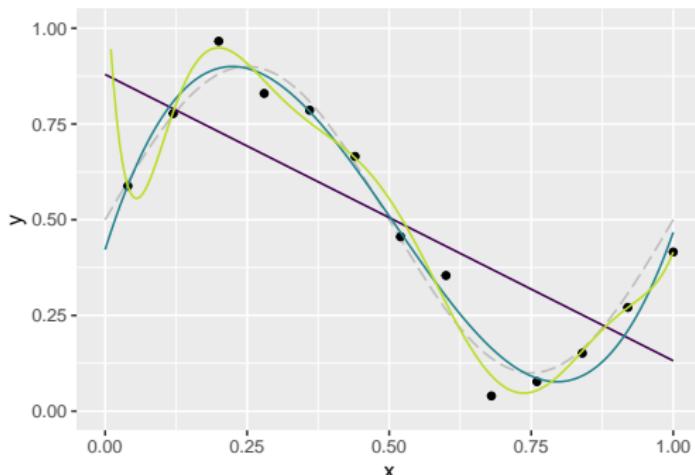
$$f(\mathbf{x} | \boldsymbol{\theta}) = \theta_0 + \theta_1 \mathbf{x} + \cdots + \theta_d \mathbf{x}^d = \sum_{j=0}^d \theta_j \mathbf{x}^j.$$

## 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?

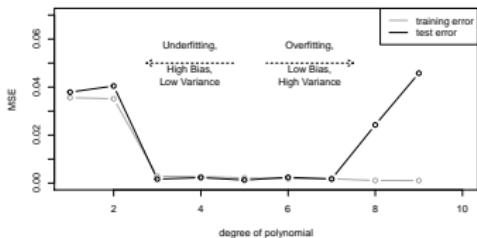
**Out-of-sample testing is probably always a good idea!**

# 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

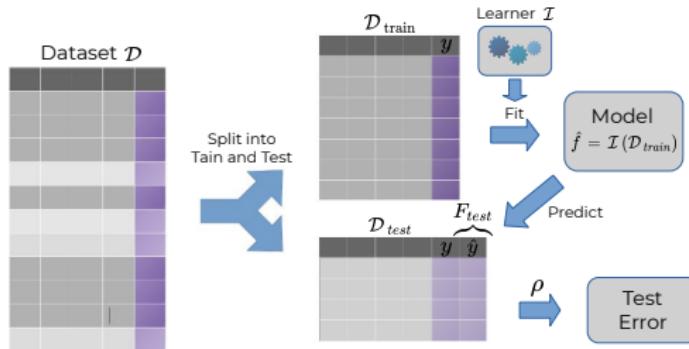


# TEST ERROR AND HOLD-OUT SPLITTING

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

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

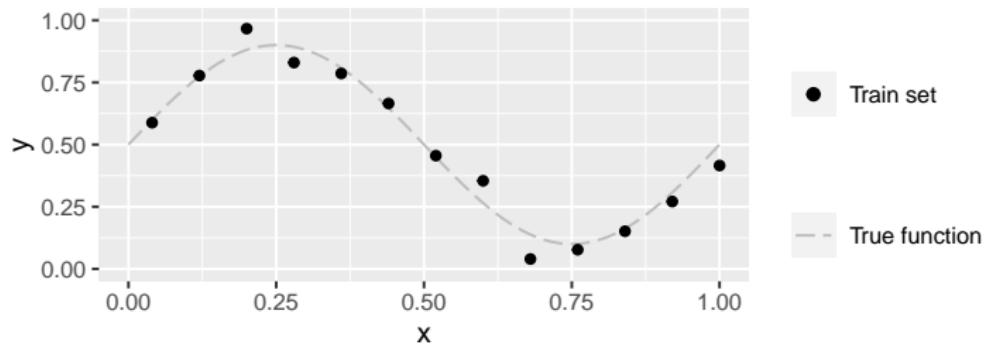
- Partition data, e.g., 2/3 for train and 1/3 for test.



A.k.a. holdout splitting.

# EXAMPLE: POLYNOMIAL REGRESSION

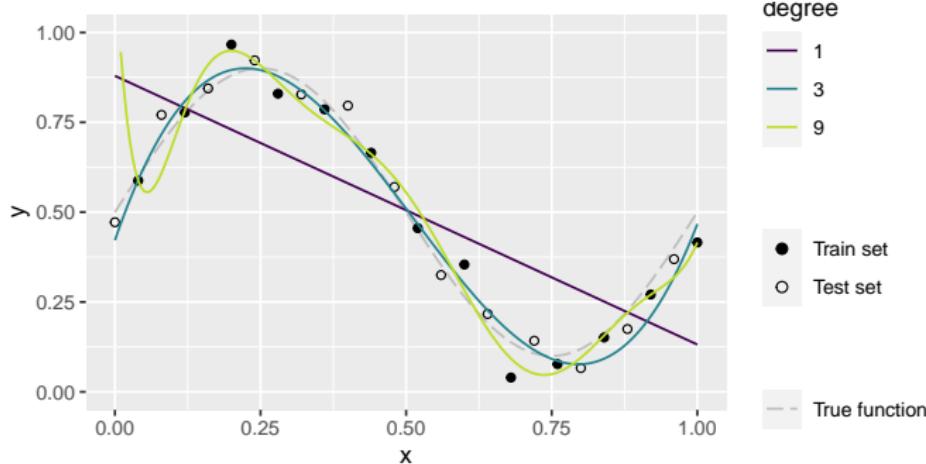
Previous example:



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

# EXAMPLE: POLYNOMIAL REGRESSION

Now with fresh test data:

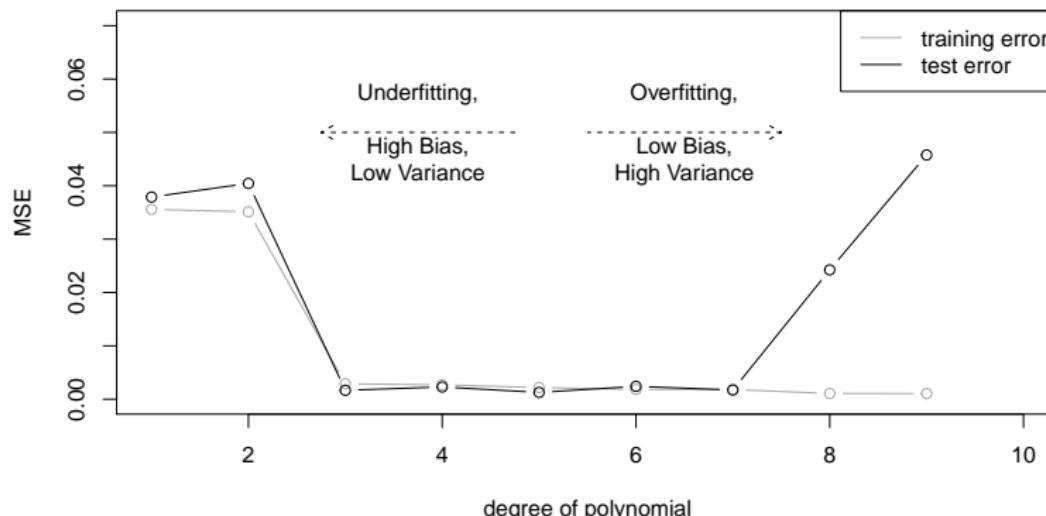


- $d = 1$ : MSE = 0.038: clearly underfitting
- $d = 3$ : MSE = 0.002: pretty OK
- $d = 9$ : MSE = 0.046: clearly overfitting

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

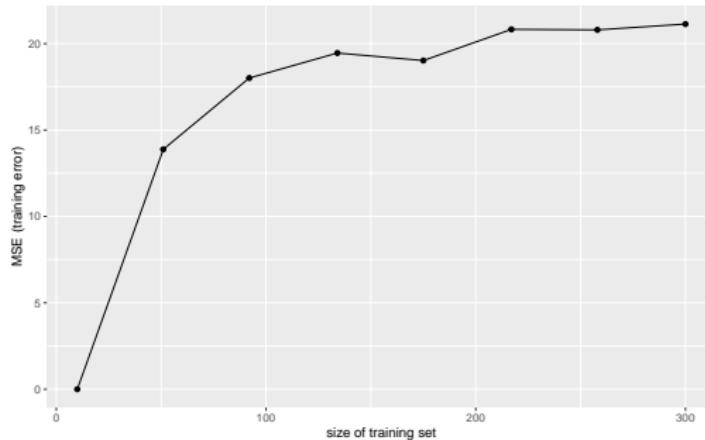
- a decrease in training error, and
- a U-shape in test error  
(first underfit, then overfit, sweet-spot in the middle).

# TRAINING VS. TEST ERROR

- Boston Housing data
- Polynomial regression (without interactions)

The training error...

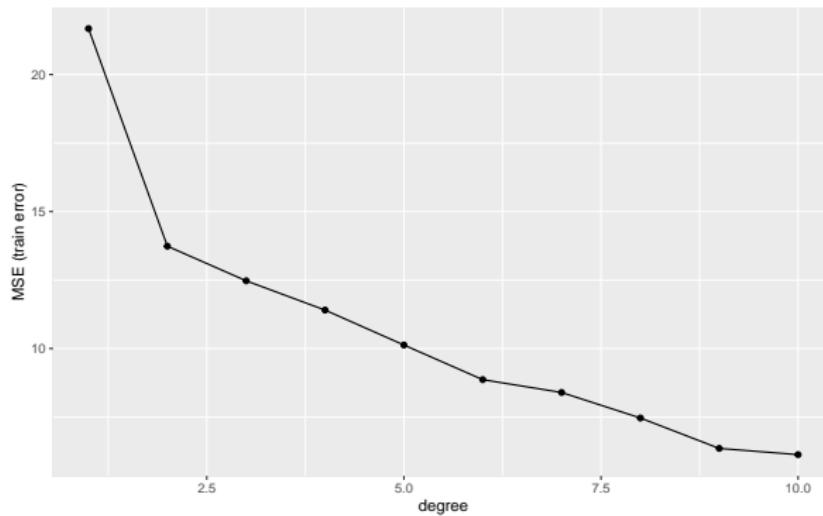
- decreases with smaller training set size as it becomes easier for the model to learn all observed patterns perfectly.



# TRAINING VS. TEST ERROR

The training error...

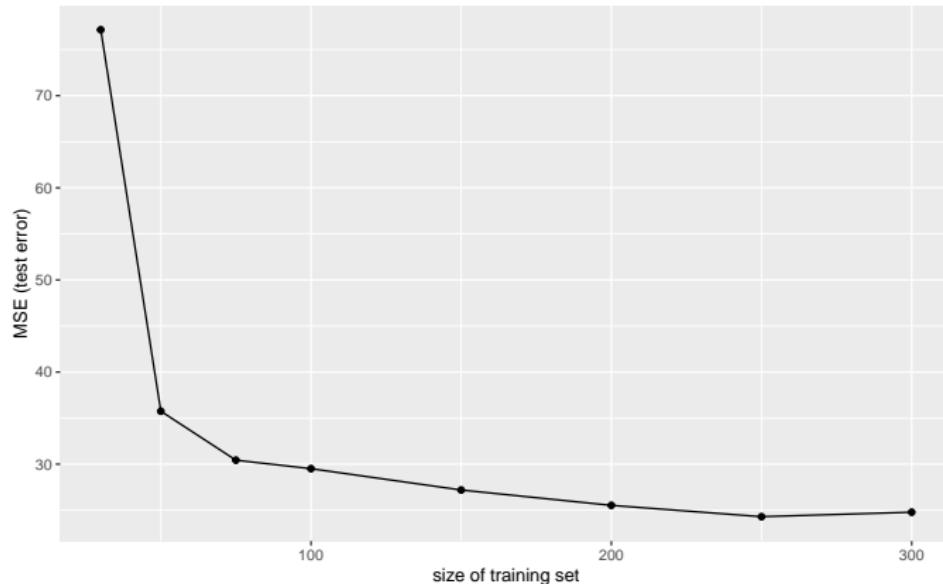
- decreases with increasing model complexity as the model gets better at learning more complex structures.



# TRAINING VS. TEST ERROR

The test error...

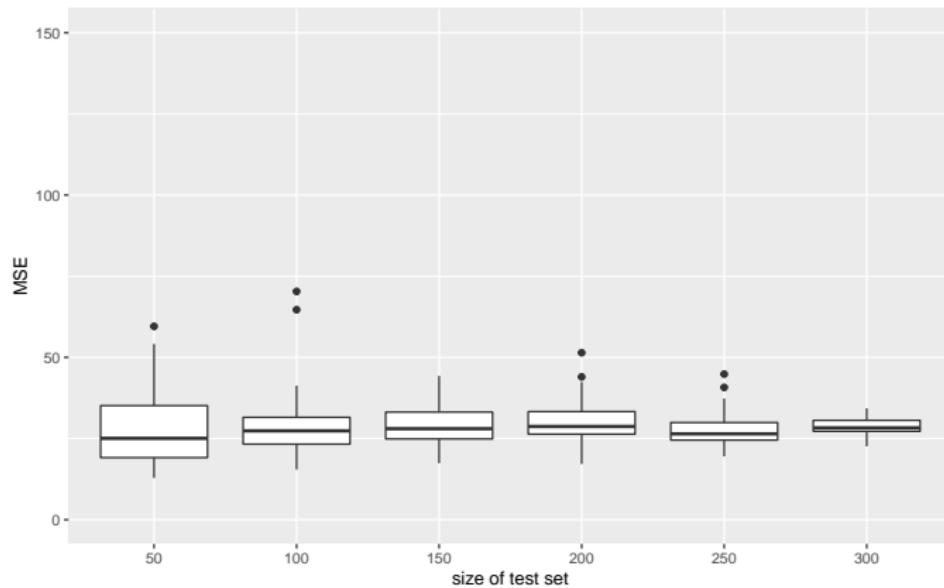
- will typically decrease with larger training set size as the model generalizes better with more data to learn from.



# TRAINING VS. TEST ERROR

The test error...

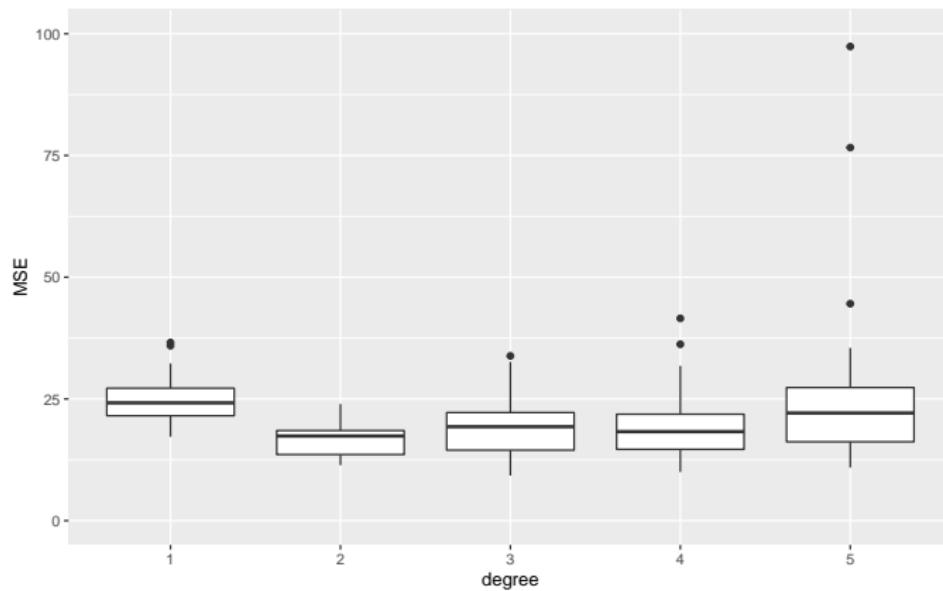
- will have higher variance with smaller test set size.



# TRAINING VS. TEST ERROR

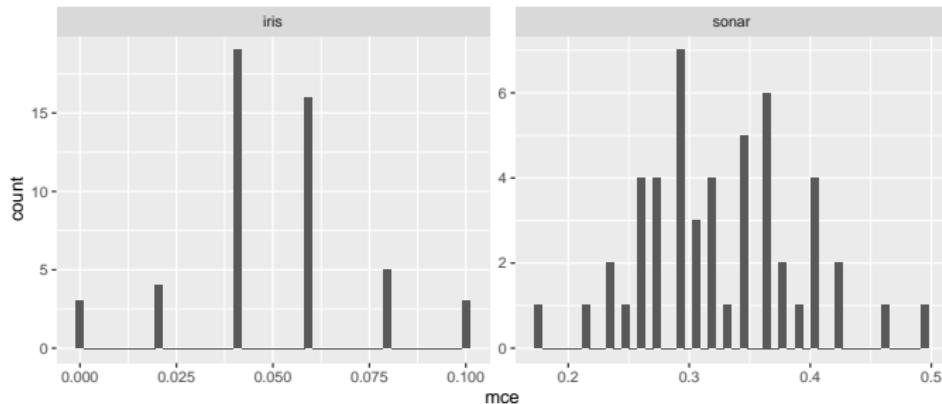
The test error...

- 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

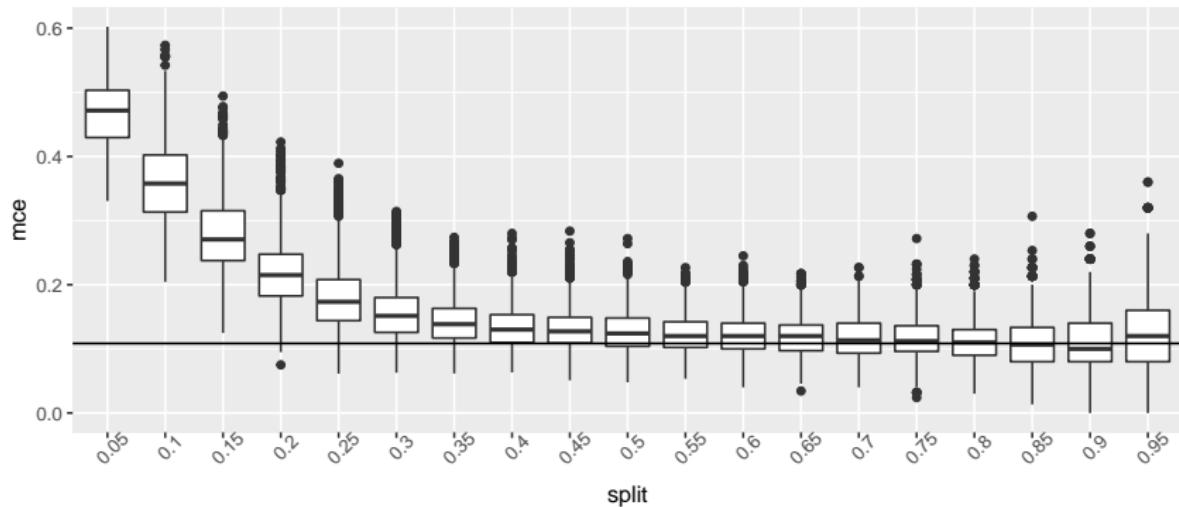
Hold-out sampling produces a trade-off between **bias** and **variance** that is controlled by split ratio.

- 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.

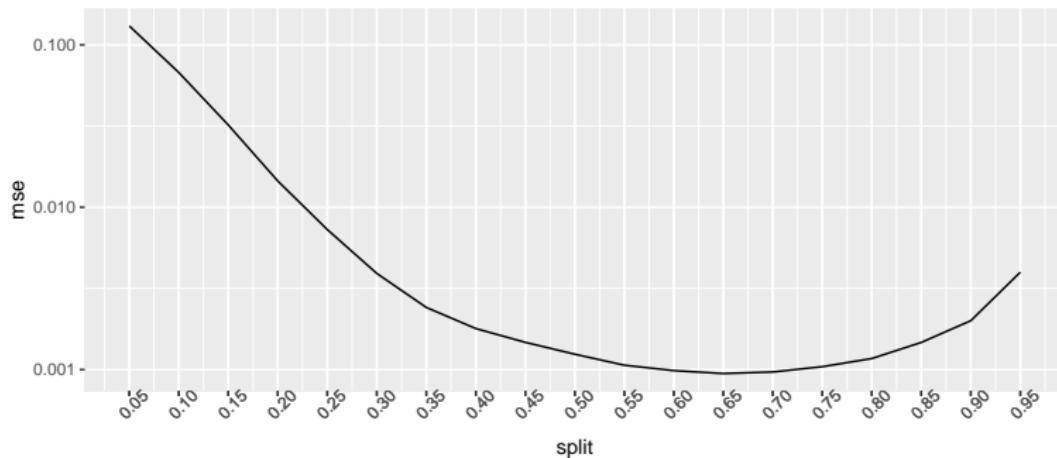
# BIAS-VARIANCE OF HOLD-OUT – EXPERIMENT



- 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.

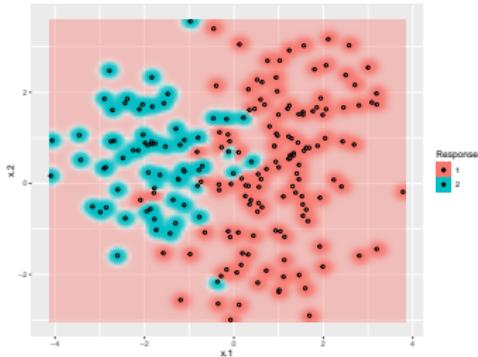
# BIAS-VARIANCE OF HOLD-OUT – EXPERIMENT

- 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



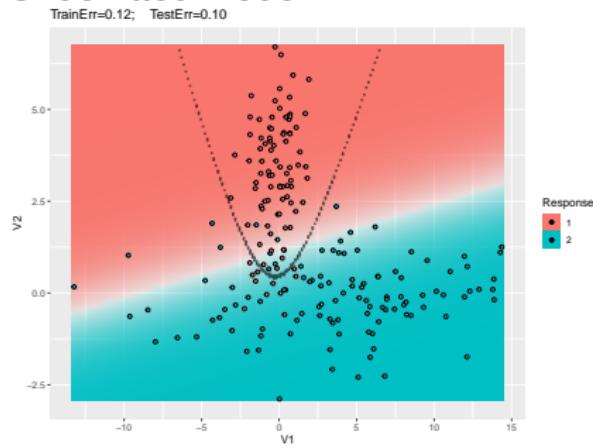
### Learning goals

- Understand definitions of overfitting and underfitting

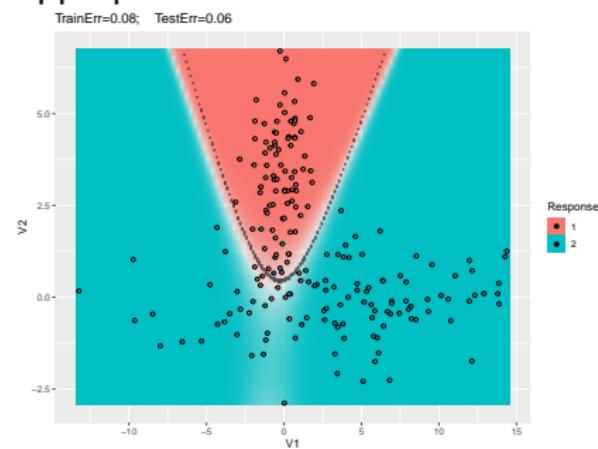
# UNDERFITTING

- 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



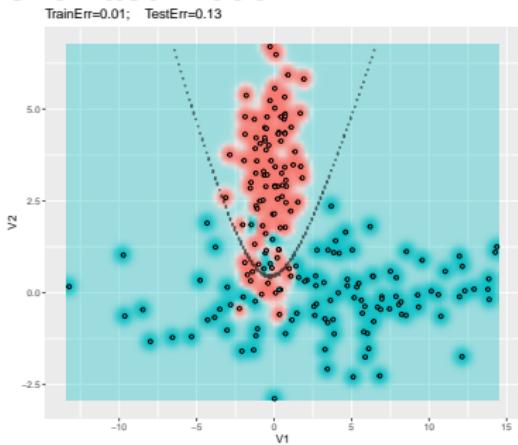
Appropriate model



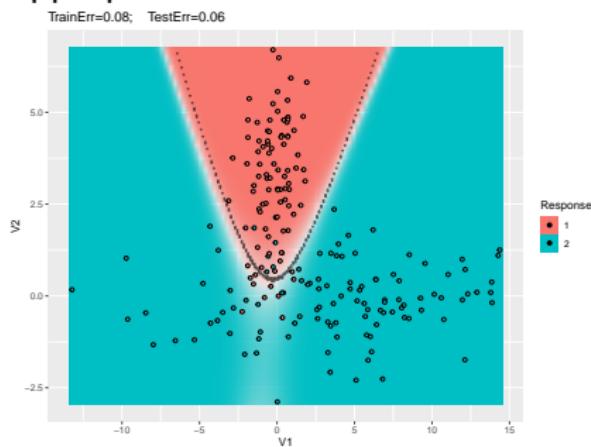
# OVERFITTING

- 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

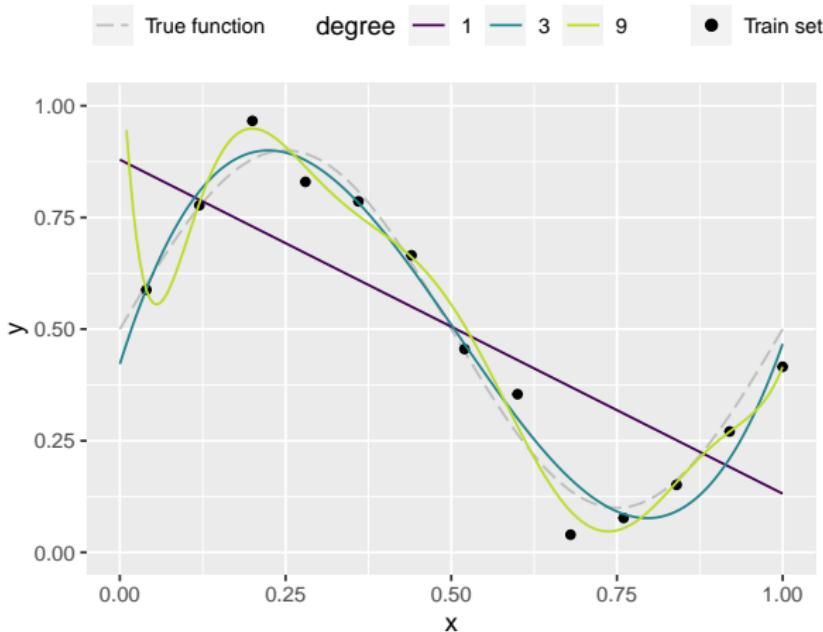


Appropriate model



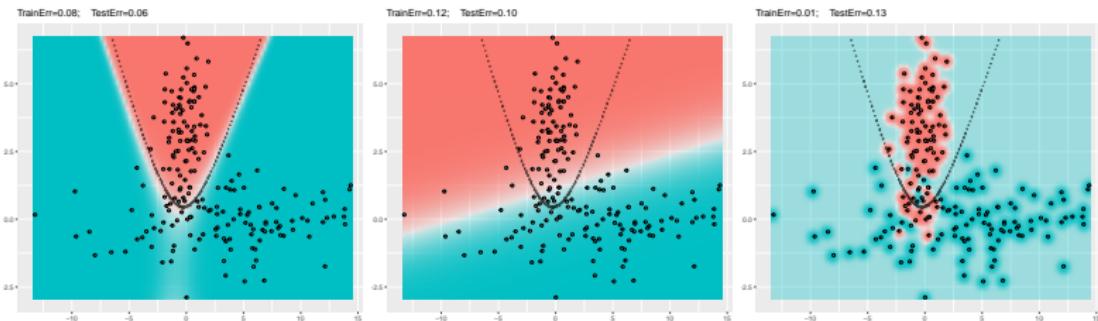
# UNDER- AND OVERFITTING IN REGRESSION

- 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

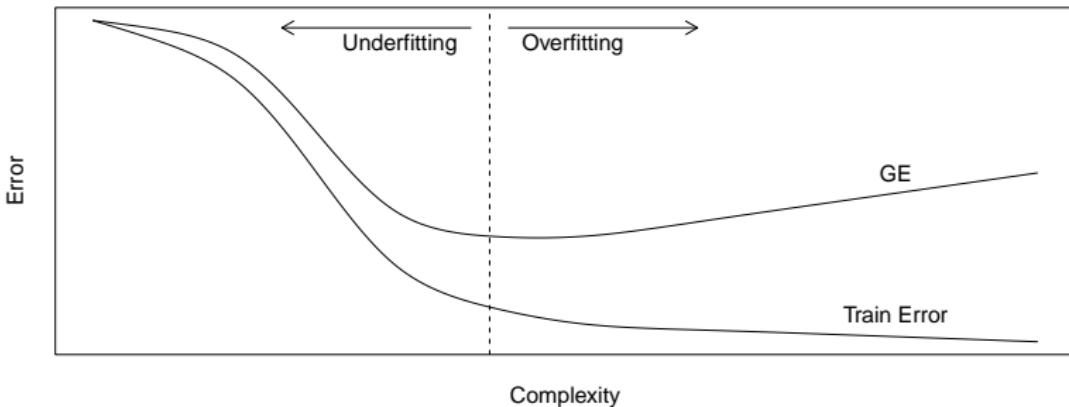
The potential for overfitting is influenced by:

- 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.

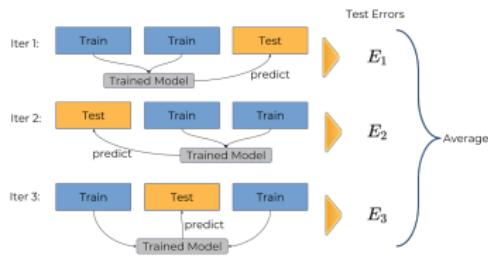
# AVOIDING OVERFITTING

- 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

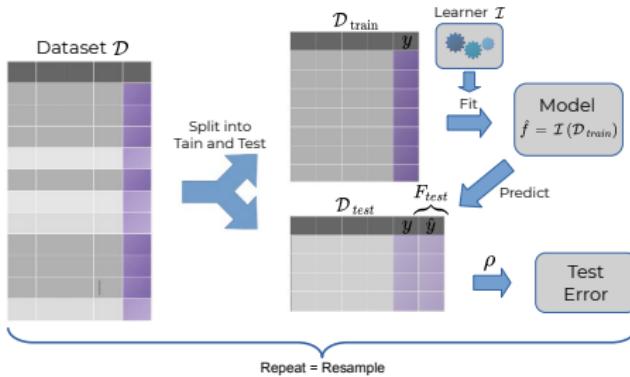


### Learning goals

- 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:

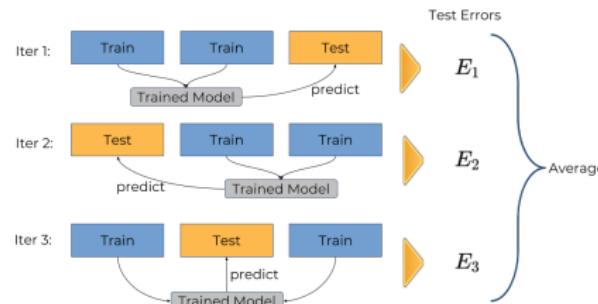
$$\widehat{\text{GE}}(\mathcal{I}, \mathcal{J}, \rho, \boldsymbol{\lambda}) = \text{agr}\left(\rho\left(\mathbf{y}_{J_{\text{test},1}}, \mathbf{F}_{J_{\text{test},1}, \mathcal{I}(\mathcal{D}_{\text{train},1}, \boldsymbol{\lambda})}\right), \right. \\ \vdots \\ \left. \rho\left(\mathbf{y}_{J_{\text{test},B}},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

- 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.

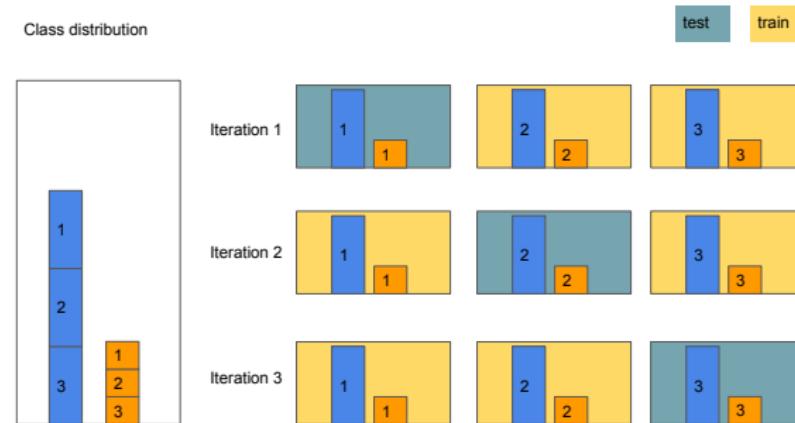
## Example: 3-fold CV



# 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

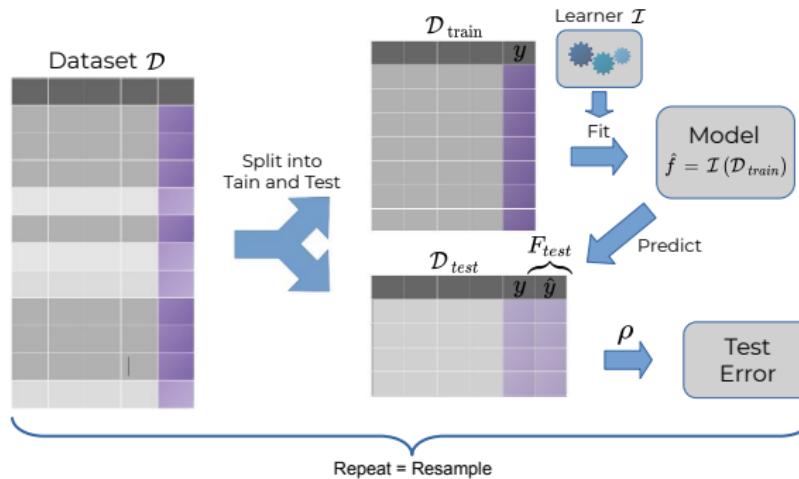


# CROSS-VALIDATION

- 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

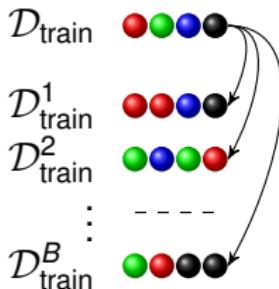
- Repeated hold-out with averaging, a.k.a. Monte Carlo CV.
- Typical choices for splitting:  $\frac{4}{5}$  or  $\frac{9}{10}$  for training.



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

# BOOTSTRAP

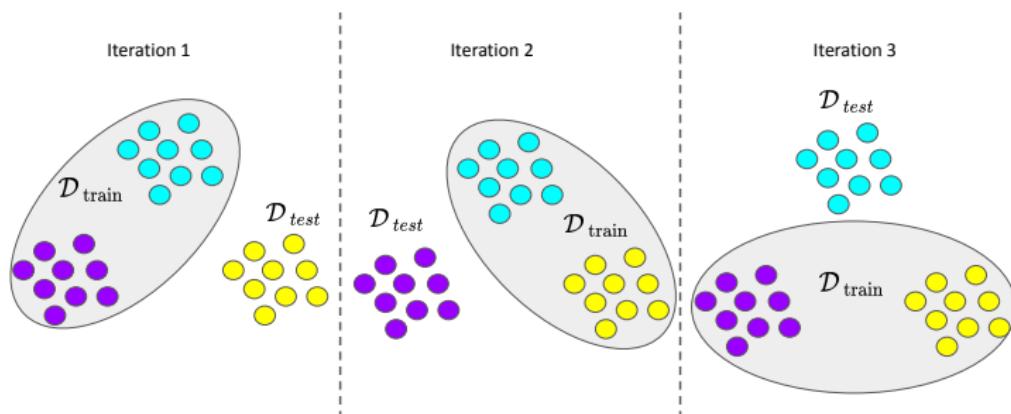
- Draw  $B$  trainsets of size  $n$  with replacement from orig  $\mathcal{D}$
- Testsets = Out-Of-Bag points:  $\mathcal{D}_{\text{test}}^b = \mathcal{D} \setminus \mathcal{D}_{\text{train}}^b$



- 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

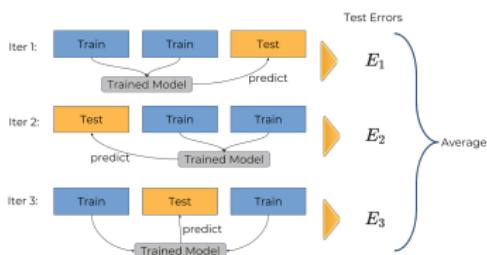
- Used when we have multiple obs from same objects, e.g., persons or hospitals or base images
- Data not i.i.d. any more
- Data from same object should **either** be in train **or** testset
- Otherwise we likely bias  $\widehat{GE}$
- CV on objects, or leave-one-object-out



# Introduction to Machine Learning

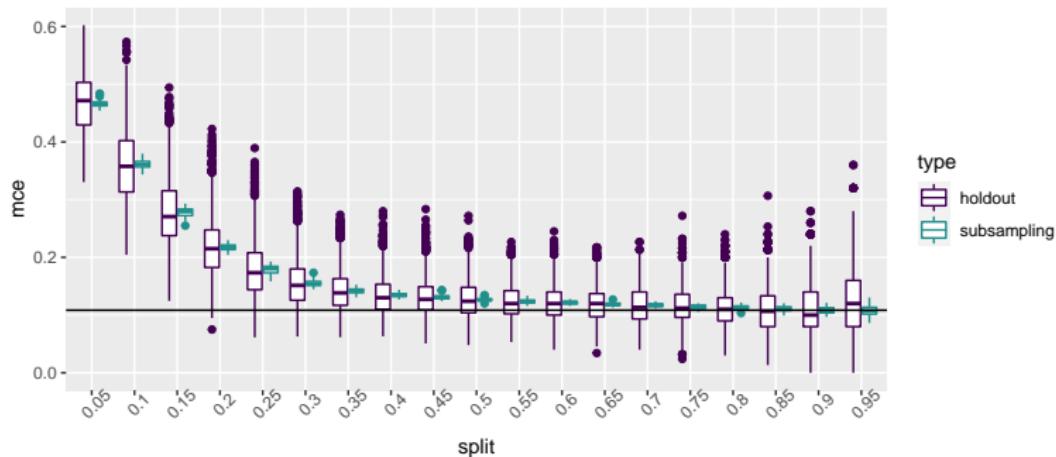
## Evaluation: Resampling 2

### Learning goals



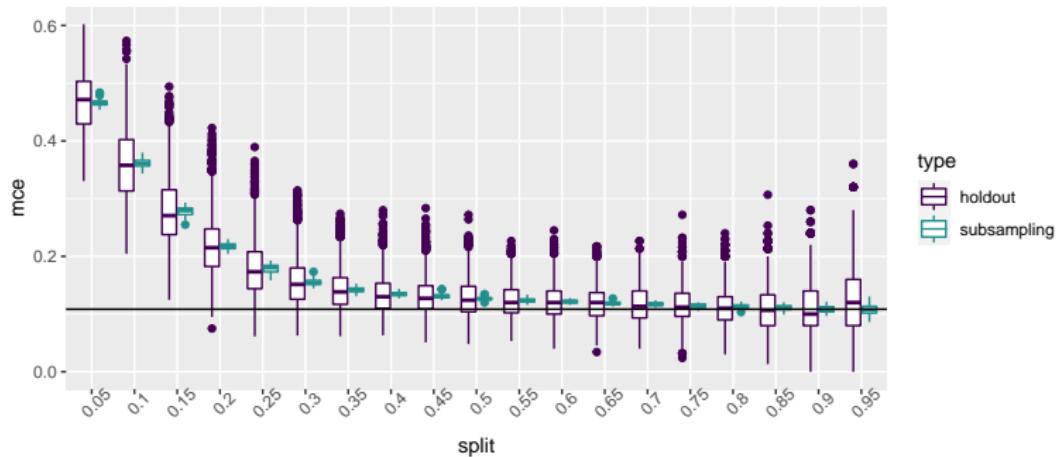
- Understand why resampling is better estimator than hold-out
- In-depth bias-var analysis of resampling estimator
- Understand that CV does not produce independent samples
- Short guideline for practical use

# BIAS-VARIANCE ANALYSIS FOR SUBSAMPLING



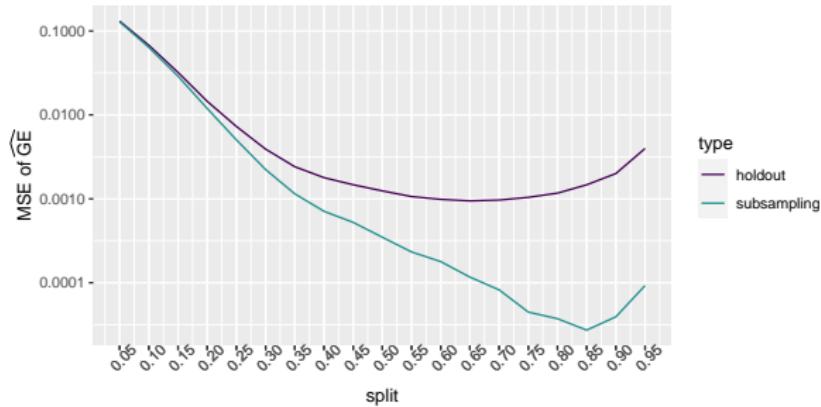
- Reconsider bias-var experiment for holdout (maybe re-read)
- Split rates  $s \in \{0.05, 0.1, \dots, 0.95\}$  with  $|\mathcal{D}_{\text{train}}| = s \cdot 500$ .
- Holdout vs. subsampling with 50 iters
- 50 replications

# 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  $\widehat{\text{GE}}(\mathcal{I}, n_{\text{train}})$

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

- Let's assume  $\text{agr}$  is avg and  $\rho$  is loss-based, so  $\rho_L$
- The  $\rho$  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

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

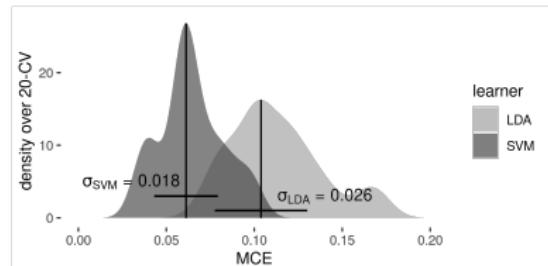
⇒

- 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 ( $n_{\text{train}}$  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.

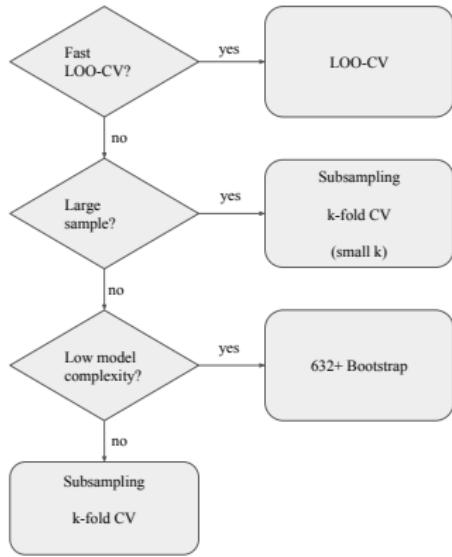


LDA vs SVM on `spam` classification problem, performance estimation via 20-CV w.r.t. MCE.

# 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<br>(TP)  | False Positive<br>(FP) |
|           | - | False Negative<br>(FN) | True Negative<br>(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:

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

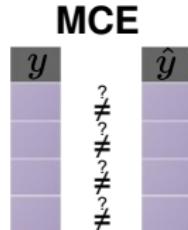
→ These form the basis for evaluation.



# LABELS: MCE & ACC

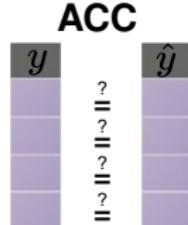
The **misclassification error rate (MCE)** counts the number of incorrect predictions and presents them as a rate:

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



**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<br>classes | setosa     | versicolor   | virginica | 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<br>(TP)  | False Positive<br>(FP) |
|       | - | False Negative<br>(FN) | True Negative<br>(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<br>classes | solvent     | 0            | 10,000      |
|                      | not solvent | 100          | 0           |

# LABELS: COSTS

Cost matrix

|                      |             | True classes |             |
|----------------------|-------------|--------------|-------------|
|                      |             | solvent      | not solvent |
| Predicted<br>classes | solvent     | 0            | 10,000      |
|                      | not solvent | 100          | 0           |

Confusion matrix

|                      |             | True classes |             |
|----------------------|-------------|--------------|-------------|
|                      |             | solvent      | not solvent |
| Predicted<br>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:

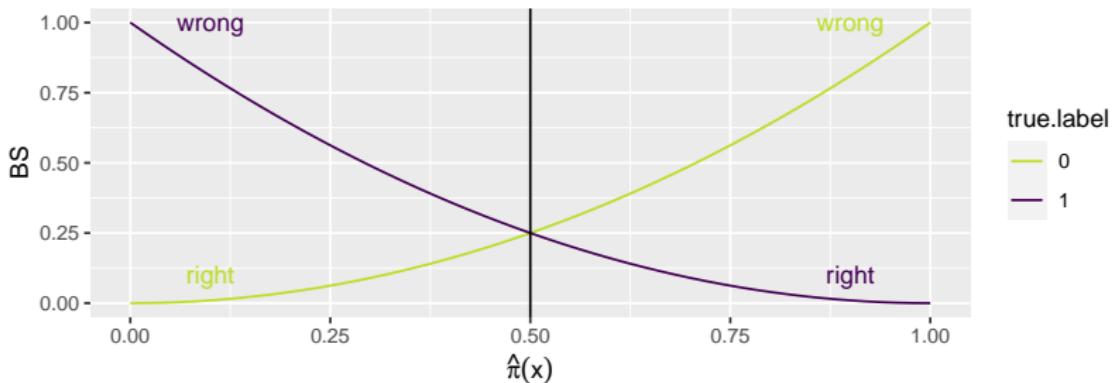
$$Costs = \frac{1}{100} (100 \cdot 0 + 0 \cdot 77 + 10,000 \cdot 23 + 0 \cdot 0) = 2.300$$

# 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

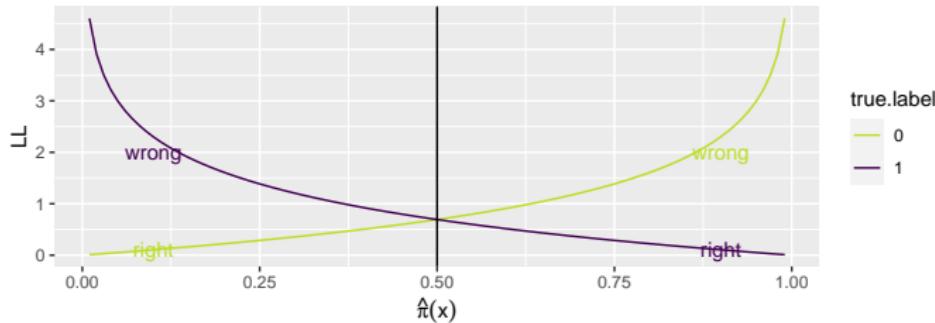
$$\rho_{BS,MC} = \frac{1}{m} \sum_{i=1}^m \sum_{k=1}^g \left( \hat{\pi}_k^{(i)} - o_k^{(i)} \right)^2$$

- 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  $\rho_{BS}$ : 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.

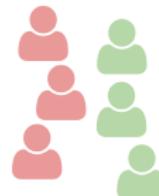


# IMBALANCED COSTS

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

# LABELS: ROC METRICS

Example:

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

[https://en.wikipedia.org/wiki/Receiver\\_operating\\_characteristic](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<br>$= \frac{\sum \text{Condition positive}}{\sum \text{Total population}}$   |
| Predicted condition | Predicted condition positive  | True positive, Power   | False positive, Type I error   | Positive predictive value (PPV), Precision<br>$= \frac{\sum \text{True positive}}{\sum \text{Predicted condition positive}}$ | Accuracy (ACC) =<br>$\frac{\sum \text{True positive} + \sum \text{True negative}}{\sum \text{Total population}}$  |
|                     | Predicted condition negative  | False negative, Type II error  | True negative  | False omission rate (FOR) =<br>$= \frac{\sum \text{False negative}}{\sum \text{Predicted condition negative}}$               | Negative predictive value (NPV)<br>$= \frac{\sum \text{True negative}}{\sum \text{Predicted condition negative}}$   |
|                     | True positive rate (TPR), Recall, Sensitivity, probability of detection<br>$= \frac{\sum \text{True positive}}{\sum \text{Condition positive}}$ | False positive rate (FPR), Fall-out, probability of false alarm<br>$= \frac{\sum \text{False positive}}{\sum \text{Condition negative}}$ | Positive likelihood ratio (LR+)<br>$= \frac{\text{TPR}}{\text{FPR}}$ | Diagnostic odds ratio (DOR)<br>$= \frac{\text{LR+}}{\text{LR-}}$   | F <sub>1</sub> score =<br>$\frac{1}{\frac{1}{\text{Recall}} + \frac{1}{\text{Precision}}}$<br>$= \frac{2 \cdot \text{Recall} \cdot \text{Precision}}{\text{Recall} + \text{Precision}}$ |
|                     | False negative rate (FNR), Miss rate<br>$= \frac{\sum \text{False negative}}{\sum \text{Condition positive}}$                                   | Specificity (SPC), Selectivity, True negative rate (TNR)<br>$= \frac{\sum \text{True negative}}{\sum \text{Condition negative}}$         | Negative likelihood ratio (LR-)<br>$= \frac{\text{FNR}}{\text{TNR}}$ |  |   |

► 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  $\rho_{TPR}$ ), but it will also tend to give more *false* positives (lower  $\rho_{TNR}$ , lower  $\rho_{PPV}$ ).
- A classifier that predicts more negatives will be more precise (higher  $\rho_{PPV}$ ), but it will also produce more *false* negatives (lower  $\rho_{TPR}$ ).

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

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

$\rho_{F_1}$  is the harmonic mean of  $\rho_{PPV}$  and  $\rho_{TPR}$ :

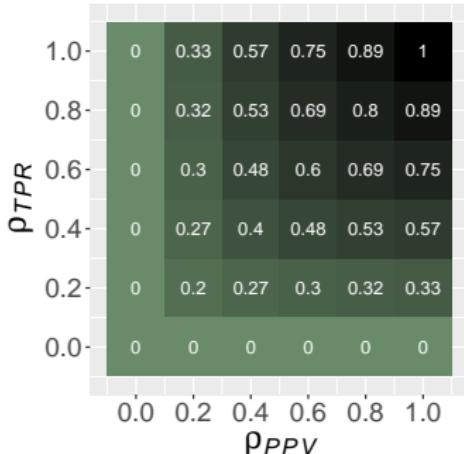
$$\rho_{F_1} = 2 \cdot \frac{\rho_{PPV} \cdot \rho_{TPR}}{\rho_{PPV} + \rho_{TPR}}$$

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  $\rho_{PPV}$  &  $\rho_{TPR}$ .

→ 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):

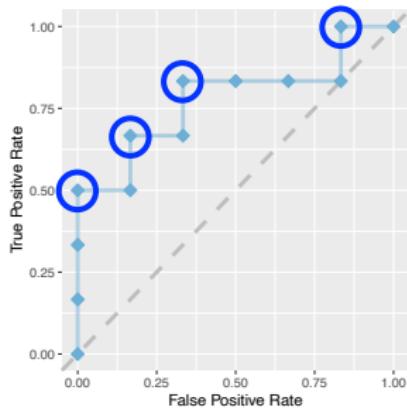
| metric | k-NN       | logistic regression | random forest | CART       |
|--------|------------|---------------------|---------------|------------|
| TPR    | 2 (0.8777) | 3 (0.8647)          | 1 (0.9257)    | 4 (0.8357) |
| TNR    | 4 (0.3764) | 2 (0.4797)          | 3 (0.4072)    | 1 (0.4911) |
| 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

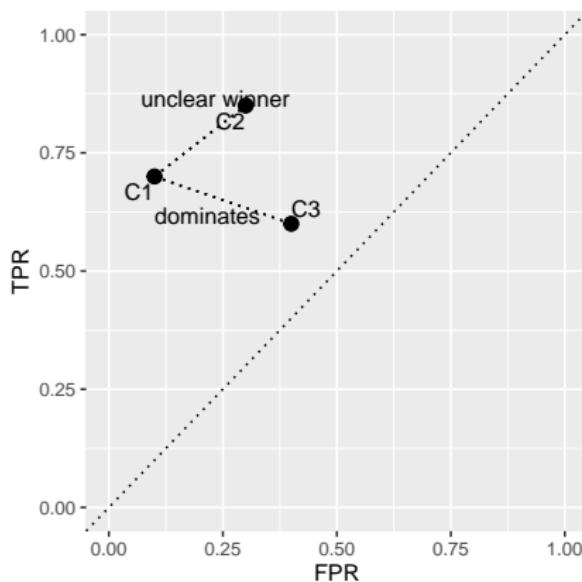


### Learning goals

- 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.     | +              | -  |
| $\hat{y}$ | TP             | FP |
|           | FN             | TN |

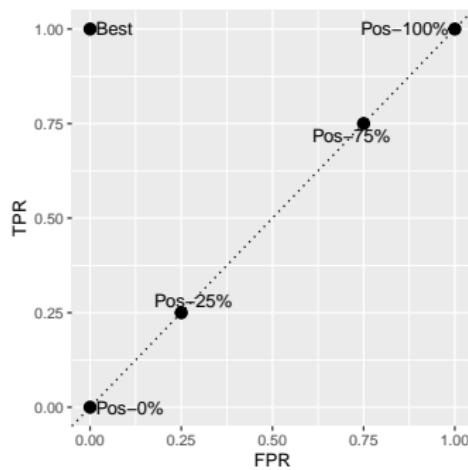
$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

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

# LABELS: ROC SPACE

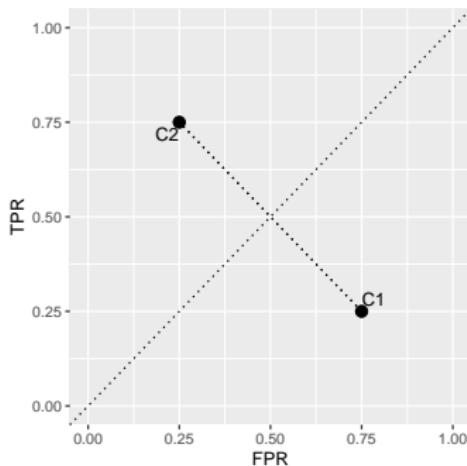
- The best classifier lies on the top-left corner, where FPR equals 0 and TPR is maximal.
- The diagonal is worst as it corresponds to a classifier producing random labels (with different proportions).

- 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!

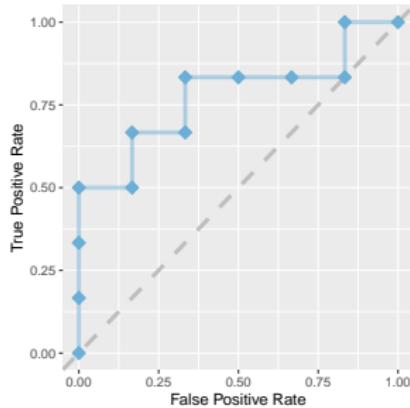
# FROM PROBABILITIES TO LABELS: ROC CURVE

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

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

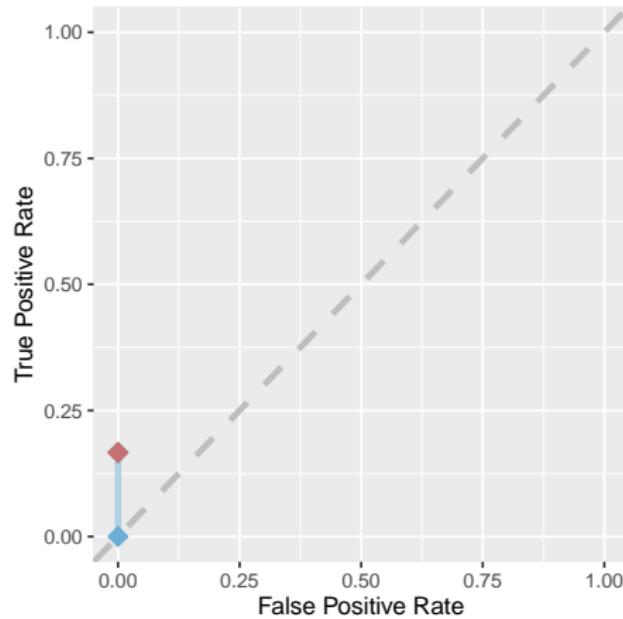
To draw a ROC curve:

- 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

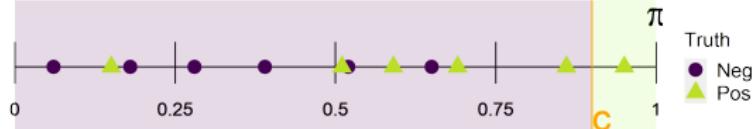
| #  | Truth | Score |
|----|-------|-------|
| 1  | Pos   | 0.95  |
| 2  | Pos   | 0.86  |
| 3  | Pos   | 0.69  |
| 4  | Neg   | 0.65  |
| 5  | Pos   | 0.59  |
| 6  | Neg   | 0.52  |
| 7  | Pos   | 0.51  |
| 8  | Neg   | 0.39  |
| 9  | Neg   | 0.28  |
| 10 | Neg   | 0.18  |
| 11 | Pos   | 0.15  |
|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



$$c = 0.9$$

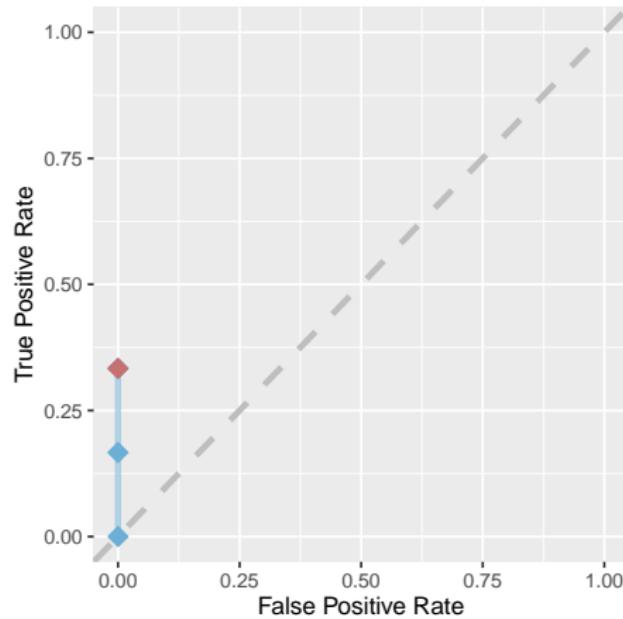
$$\rightarrow \text{TPR} = 0.167$$

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

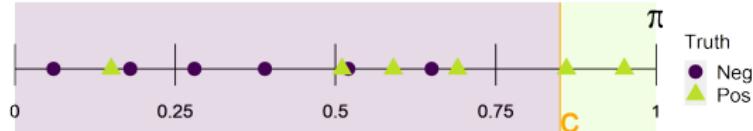


# DRAWING ROC CURVES

| #  | 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  |

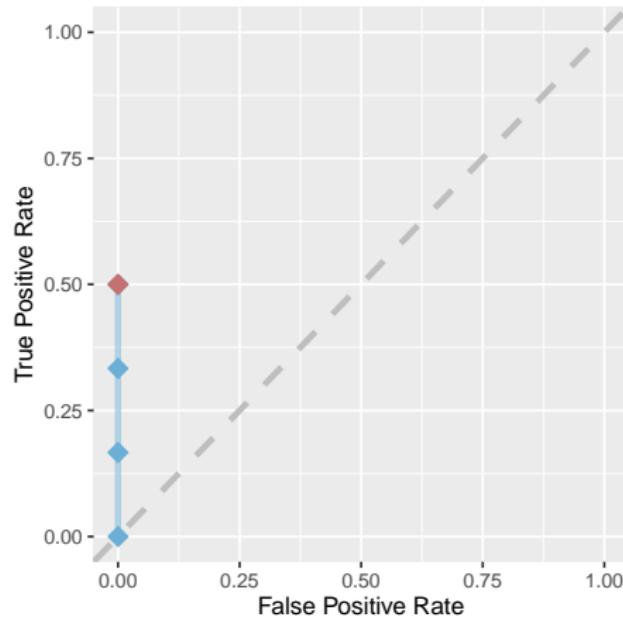


$$c = 0.85 \\ \rightarrow TPR = 0.333 \\ \rightarrow FPR = 0$$

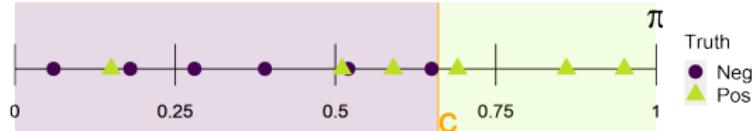


# DRAWING ROC CURVES

| #  | 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  |

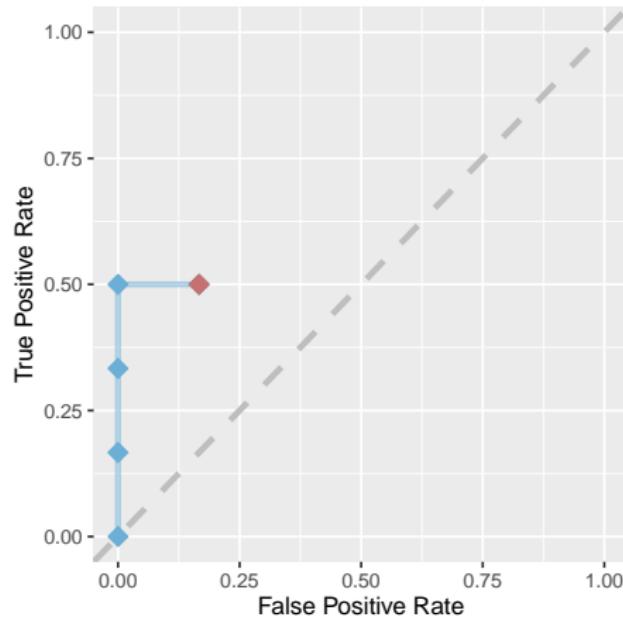


$c = 0.66$   
→ TPR = 0.5  
→ FPR = 0



# DRAWING ROC CURVES

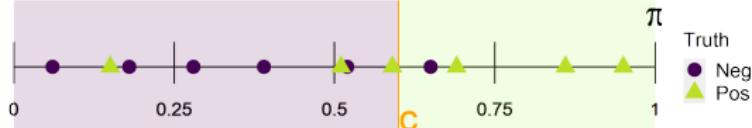
| #  | Truth | Score |
|----|-------|-------|
| 1  | Pos   | 0.95  |
| 2  | Pos   | 0.86  |
| 3  | Pos   | 0.69  |
| 4  | Neg   | 0.65  |
| 5  | Pos   | 0.59  |
| 6  | Neg   | 0.52  |
| 7  | Pos   | 0.51  |
| 8  | Neg   | 0.39  |
| 9  | Neg   | 0.28  |
| 10 | Neg   | 0.18  |
| 11 | Pos   | 0.15  |
|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



$$c = 0.6$$

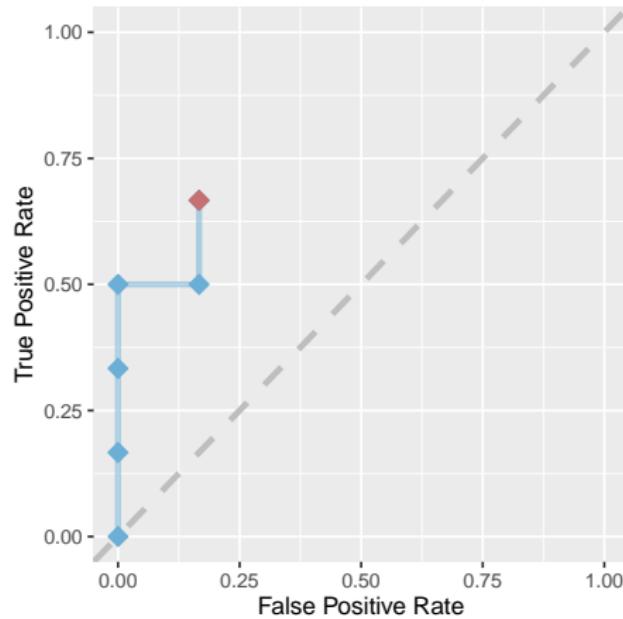
$$\rightarrow \text{TPR} = 0.5$$

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

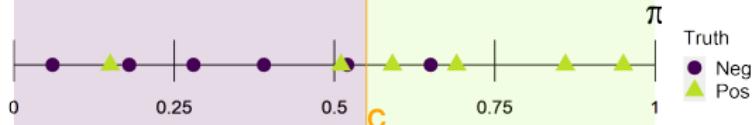


# DRAWING ROC CURVES

| #  | 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  |

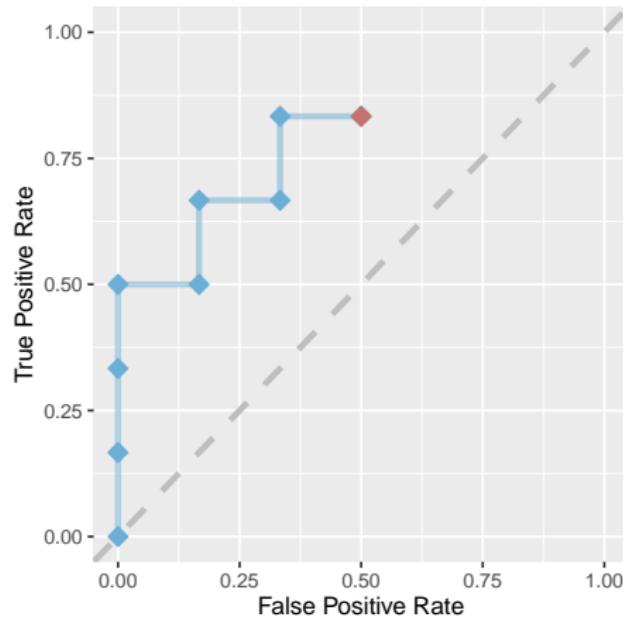


$$\begin{aligned}C &= 0.55 \\ \rightarrow \text{TPR} &= 0.667 \\ \rightarrow \text{FPR} &= 0.167\end{aligned}$$



# DRAWING ROC CURVES

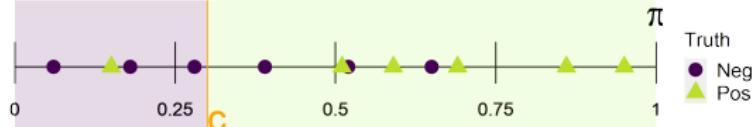
| #  | Truth | Score |
|----|-------|-------|
| 1  | Pos   | 0.95  |
| 2  | Pos   | 0.86  |
| 3  | Pos   | 0.69  |
| 4  | Neg   | 0.65  |
| 5  | Pos   | 0.59  |
| 6  | Neg   | 0.52  |
| 7  | Pos   | 0.51  |
| 8  | Neg   | 0.39  |
| 9  | Neg   | 0.28  |
| 10 | Neg   | 0.18  |
| 11 | Pos   | 0.15  |
|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



$$c = 0.3$$

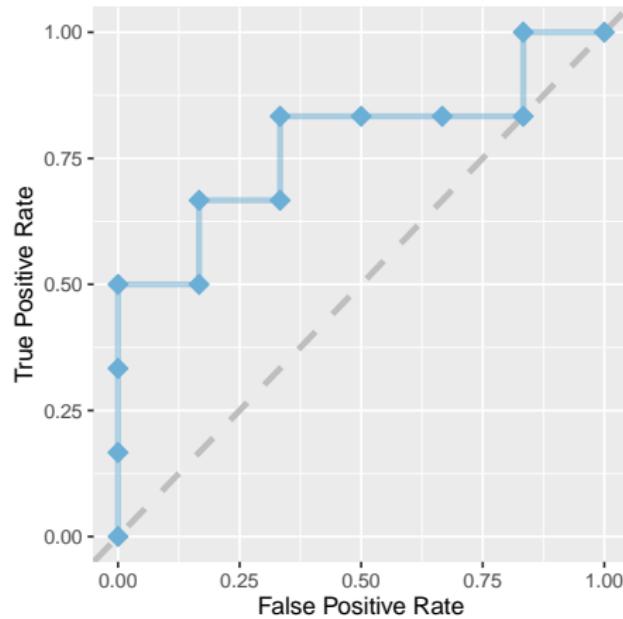
$$\rightarrow \text{TPR} = 0.833$$

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



# DRAWING ROC CURVES

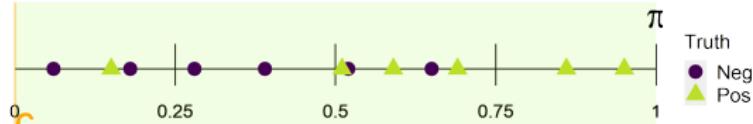
| #  | Truth | Score |
|----|-------|-------|
| 1  | Pos   | 0.95  |
| 2  | Pos   | 0.86  |
| 3  | Pos   | 0.69  |
| 4  | Neg   | 0.65  |
| 5  | Pos   | 0.59  |
| 6  | Neg   | 0.52  |
| 7  | Pos   | 0.51  |
| 8  | Neg   | 0.39  |
| 9  | Neg   | 0.28  |
| 10 | Neg   | 0.18  |
| 11 | Pos   | 0.15  |
|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



$$c = 0$$

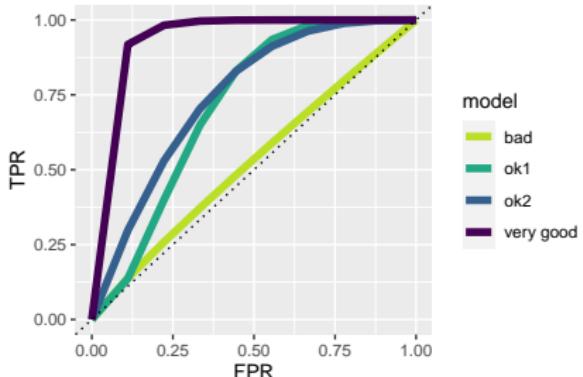
$$\rightarrow \text{TPR} = 1$$

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



# 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.

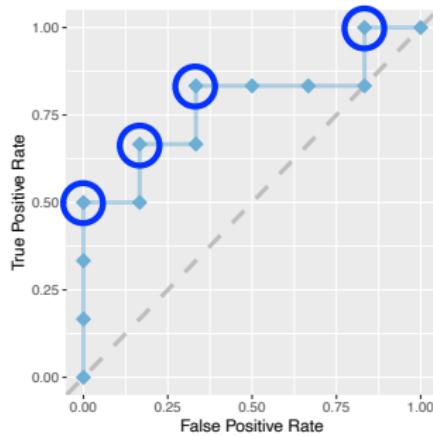


- Small thresholds will very liberally predict the positive class, and result in a potentially higher FPR, but also higher TPR.
- High thresholds will very conservatively predict the positive class, and result in a lower FPR and TPR.
- As we have not defined the trade-off between false positive and false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

# CHOOSING THRESHOLD / OPERATING POINT

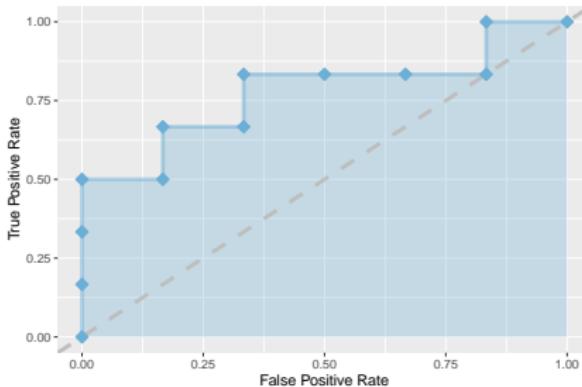
Often done visually and post-hoc, as class imbalances or costs are unknown a-priori.

- 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 |
|---|------|

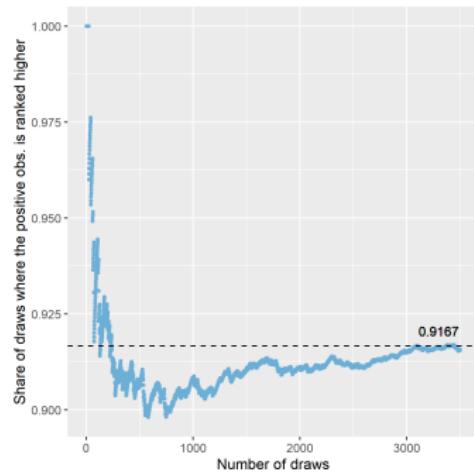
Choose a random negative



|   |     |
|---|-----|
| 0 | 0.3 |
|---|-----|

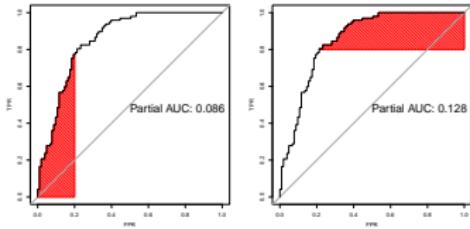
$$\text{AUC} = 0.9167$$

- 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

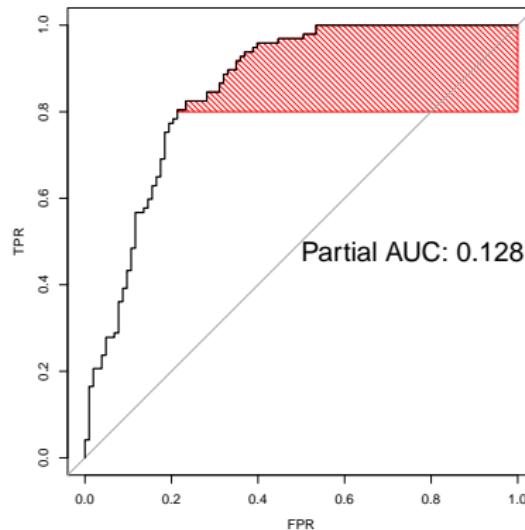
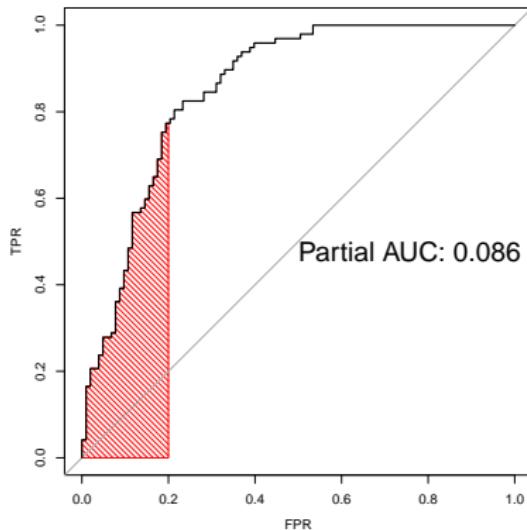
- Understand that entire AUC is not always relevant
- Learn about partial AUC

## 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

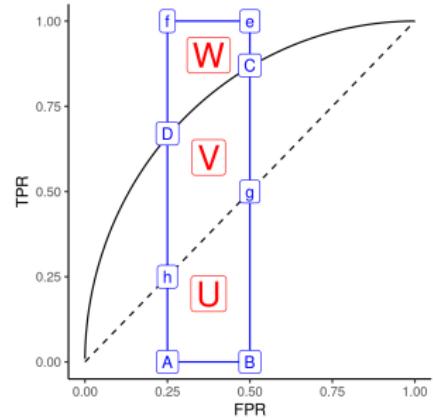


# CORRECTED PARTIAL AUC

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

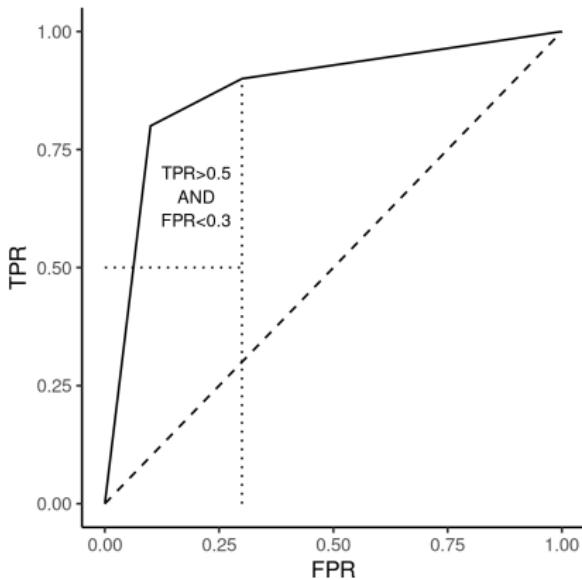
$$\text{pAUC}_{\text{corrected}} = \frac{1}{2} \left( 1 + \frac{\text{pAUC} - \text{pAUC}_{\min}}{\text{pAUC}_{\max} - \text{pAUC}_{\min}} \right),$$

- pAUC is  $V+U = \text{"A-B-C-D"}$
- $\text{pAUC}_{\min}$  is pAUC of random classifier, so  $U = \text{"A-B-g-h"}$
- $\text{pAUC}_{\max}$  is  $U+V+W = \text{"A-B-e-f"}$
- 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 |

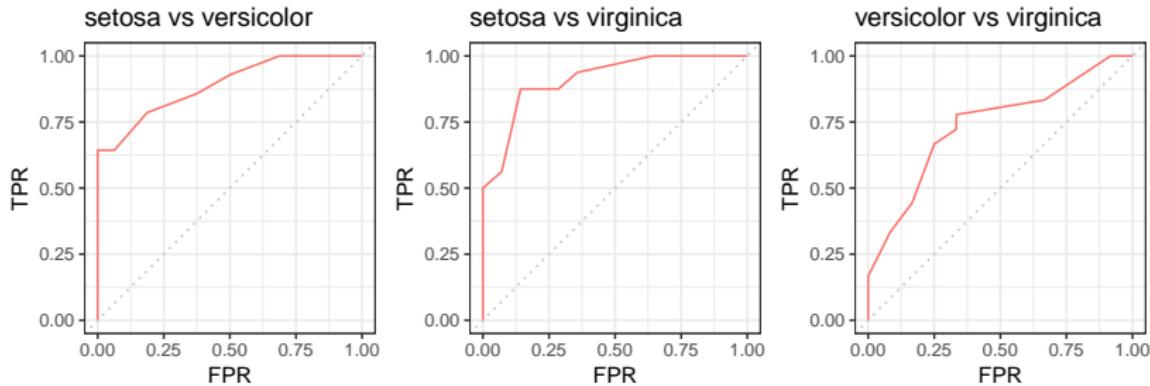
### Learning goals

- 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  $AUC(k \mid \ell)$  for classes  $k$  (pos) and  $\ell$  (neg)
- Compute AUC: Subset preds to rows of true  $k$  and  $\ell$ , use  $\hat{\pi}_k$
- Interpret: Prob that random member of  $\ell$  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  $\text{AUC}(k | \ell)$ .

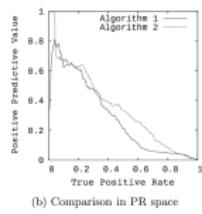
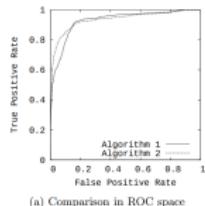
$$\text{AUC}_{MC} = \frac{1}{g(g-1)} \sum_{k \neq \ell} \text{AUC}(k|\ell) \in [0, 1].$$

## 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

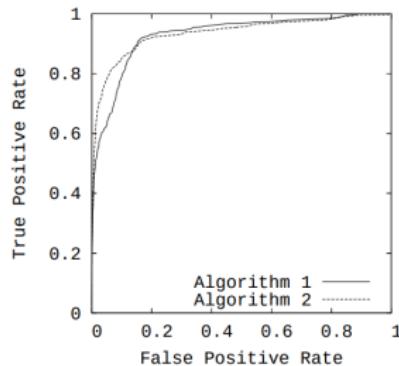


### Learning goals

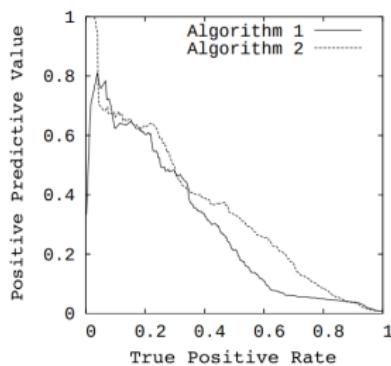
- Understand PR curves
- Same as PPV-TPR curve
- Compare to standard TPR-FPR ROC curve

# 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

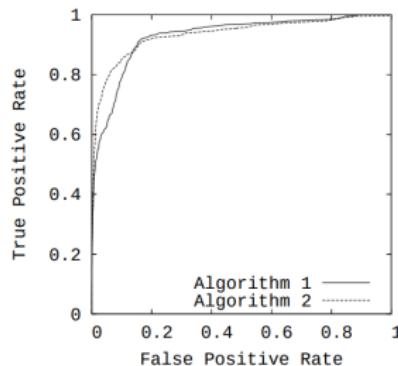


(b) Comparison in PR space

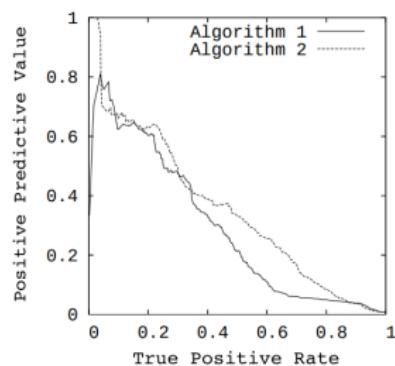
Davis and Goadrich (2006): The Relationship Between Precision-Recall and ROC Curves ([URL](#)).

# 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   |

$$TPR = 10/11$$

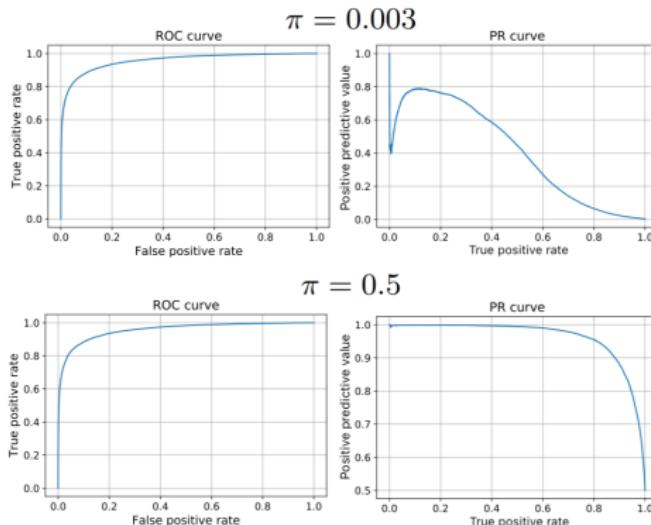
$$FPR = 0.01$$

$$PPV = 1/2$$

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

# IMBALANCED DATA

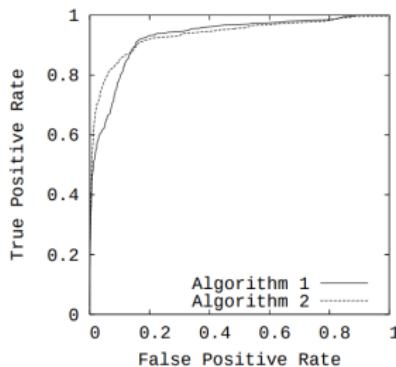
- Top row: Imbal classes with  $\pi = 0.003$
- Bottom: balanced with  $\pi = 0.5$
- ROC curves (LHS) are similar
- PR curve (RHS) changes strongly from imbal to bal classes



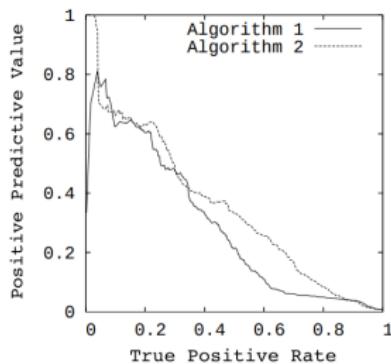
Wissam Siblini et. al. (2004): Master your Metrics with Calibration ([URL](#)).

# 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

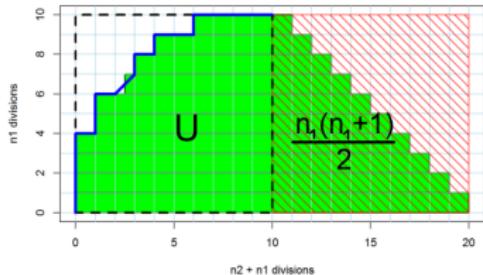


(b) Comparison in PR space

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

# Introduction to Machine Learning

## Evaluation: AUC & Mann-Whitney-U Test

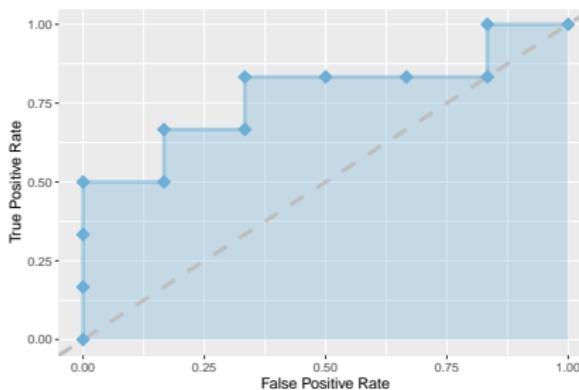


### Learning goals

- 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.

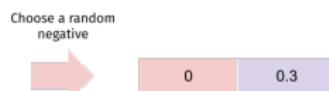
# AUC AS A RANK-BASED METRIC

- 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  $\text{AUC} = 1$ .

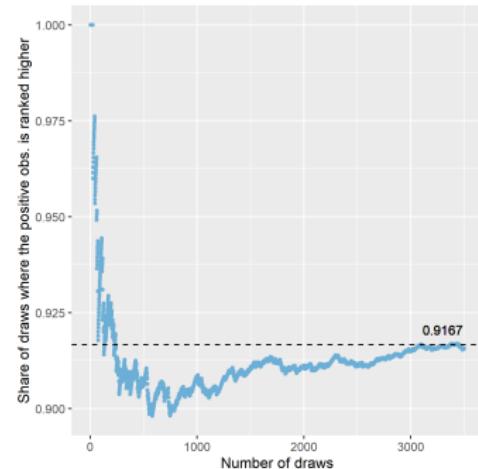
| Truth | Score |
|-------|-------|
| 1     | 0.9   |
| 1     | 0.76  |
| 1     | 0.7   |
| 0     | 0.5   |
| 1     | 0.45  |
| 0     | 0.3   |
| 0     | 0.1   |

{ }

$$\text{AUC} = 0.9167$$



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



# MANN-WHITNEY-U TEST

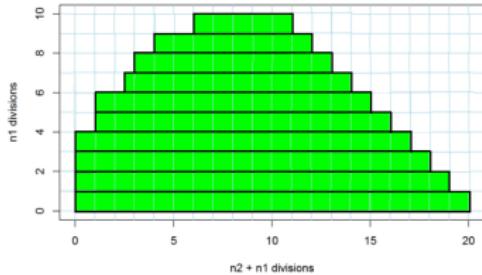
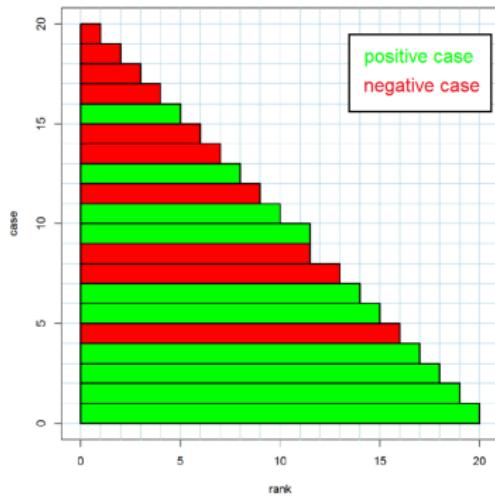
- The Mann-Whitney-U test is a **non-parametric hypothesis test** on the difference in location between two samples  $X_1, X_2$  of sizes  $n_1$  and  $n_2$ , respectively.
- Under the null,  $X_1$  and  $X_2$  follow the same (unknown) distribution  $\mathbb{P}$ , and for any pair of observations  $x_{1,1} \in X_1, x_{2,1} \in X_2$  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  $x_{1,i}$ ):

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

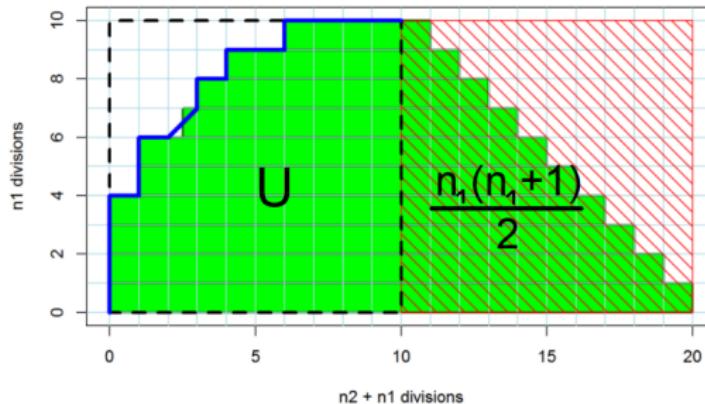
- For large samples,  $U$  is approximately normally distributed.

# 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

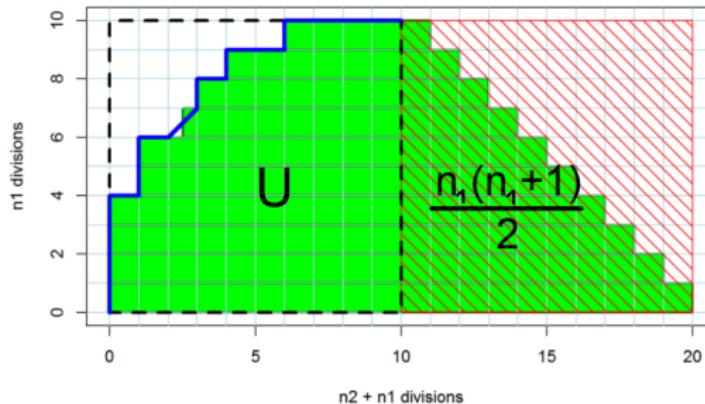


- Denoting the respective numbers of cases as  $n_+$  and  $n_-$ , we have:

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

- 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

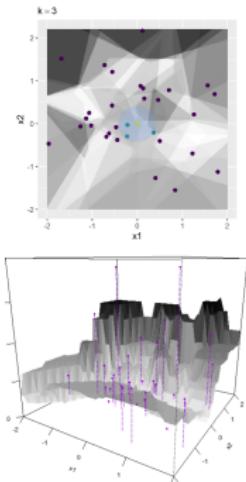
Neural Networks

Tuning

Nested Resampling

# Introduction to Machine Learning

## k-Nearest Neighbors



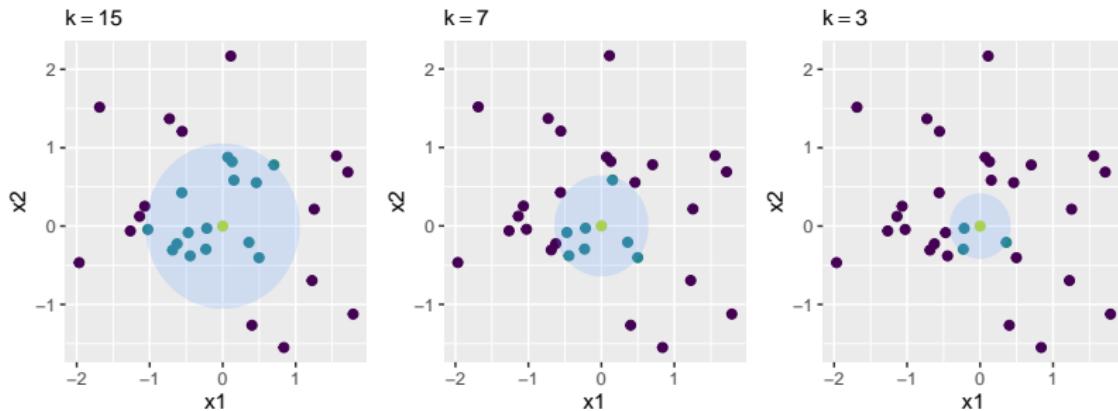
### Learning goals

- 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 subset of the feature space  $\mathcal{X}$  that is as least as close to  $\mathbf{x}$  as its  $k$ -th closest neighbor  $\mathbf{x}^{(k)}$  in the training data set is called the  **$k$ -neighborhood**  $N_k(\mathbf{x})$  of  $\mathbf{x}$ :

$$N_k(\mathbf{x}) = \{\mathbf{x}^{(i)} \in \mathcal{X} \mid d(\mathbf{x}^{(i)}, \mathbf{x}) \leq d(\mathbf{x}^{(k)}, \mathbf{x})\}$$



# DISTANCE MEASURES

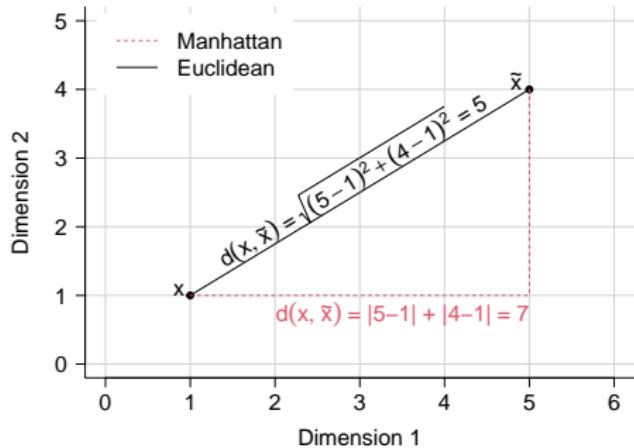
- Popular for numerical features: **Minkowski** distances of the form

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

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

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

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

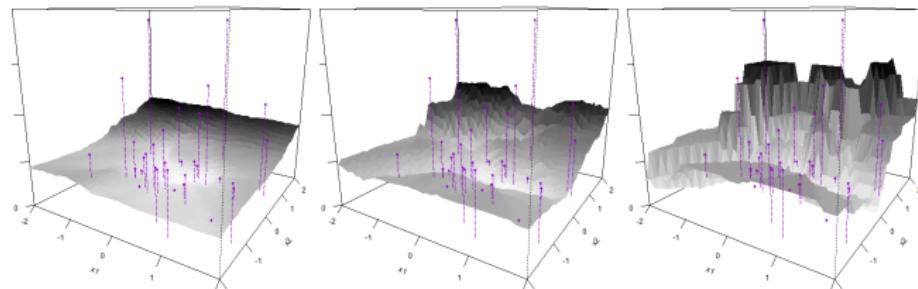
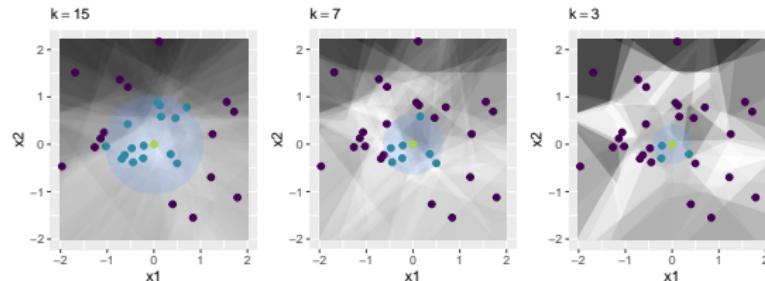


# PREDICTION - REGRESSION

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



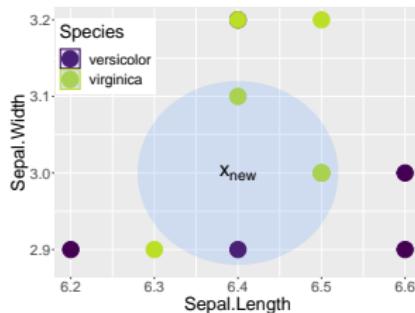
# PREDICTION - CLASSIFICATION

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:

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

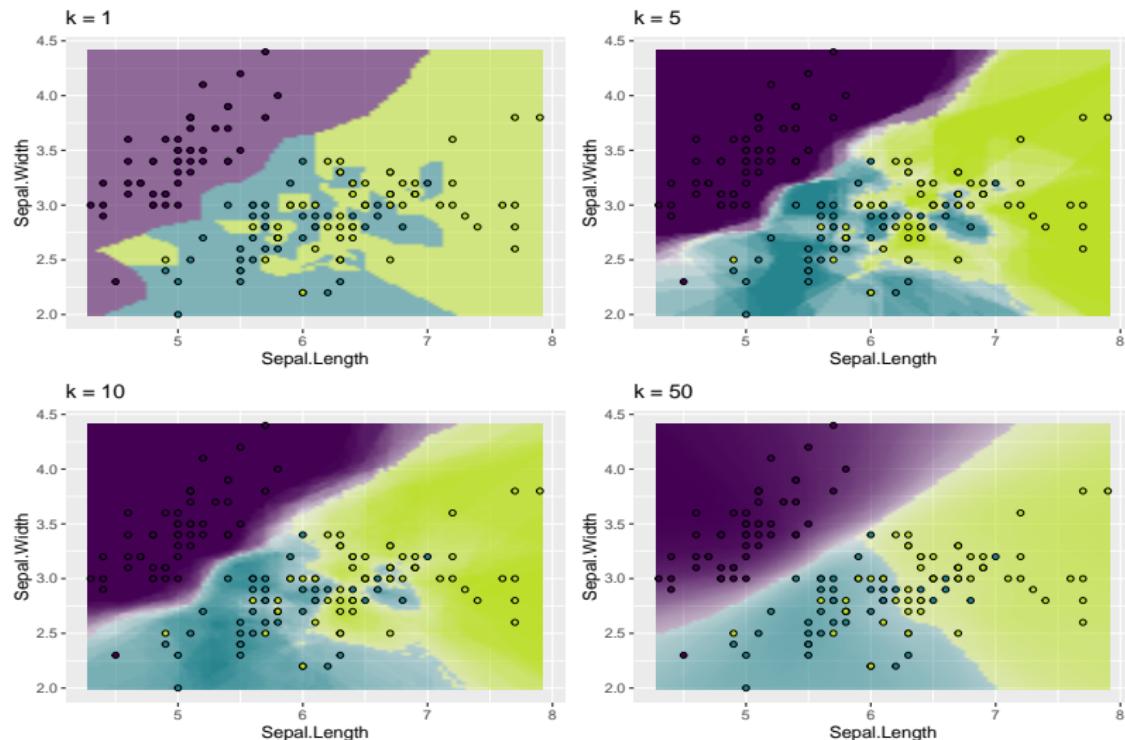


|     | SL  | SW  | Species    | dist  |
|-----|-----|-----|------------|-------|
| 52  | 6.4 | 3.2 | versicolor | 0.200 |
| 59  | 6.6 | 2.9 | versicolor | 0.224 |
| 75  | 6.4 | 2.9 | versicolor | 0.100 |
| 76  | 6.6 | 3.0 | versicolor | 0.200 |
| 98  | 6.2 | 2.9 | versicolor | 0.224 |
| 104 | 6.3 | 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   |

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

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

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

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

# INTRODUCTION TO MACHINE LEARNING

ML Basics

Supervised Regression

Supervised Classification

Performance Evaluation

k-NN

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

Random Forests

Neural Networks

Tuning

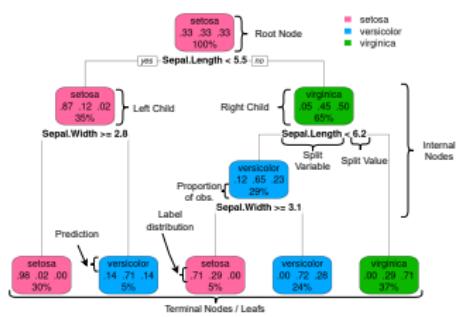
Nested Resampling

# Introduction to Machine Learning

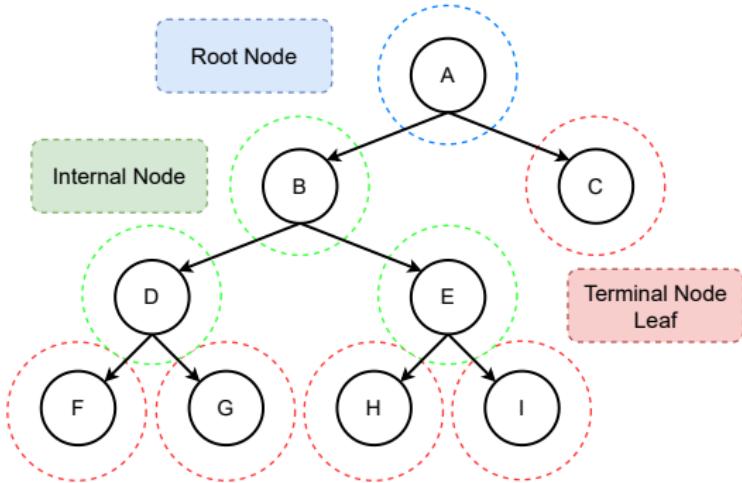
## CART: Predictions with CART

### 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
- Understand how the label of a new observation is predicted via CART
- Know hypothesis space of CART

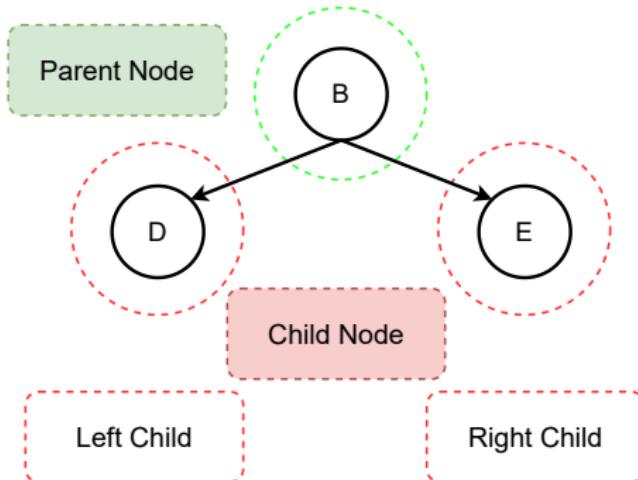


# BINARY TREES



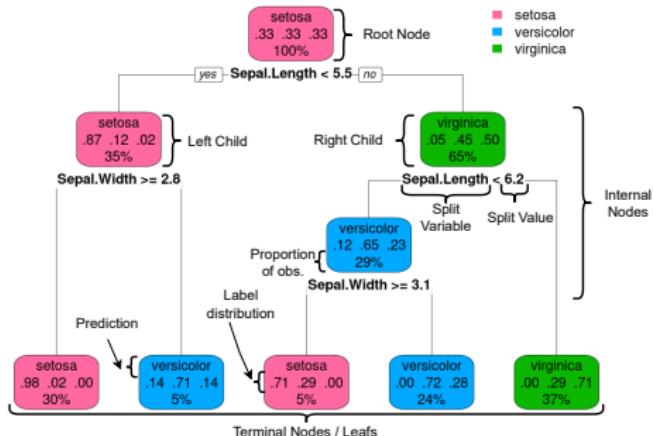
- Binary trees represent a top-down hierarchy with binary splits
- A tree is composed of different node types: a) the root node, b) internal nodes, and c) terminal nodes (also leaves).

# BINARY TREES



- Nodes have relative relationships, they can be:
  - Parent nodes
  - Child nodes
- Root nodes don't have parents – leaves don't have children

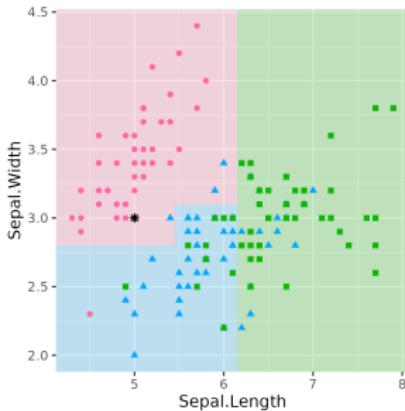
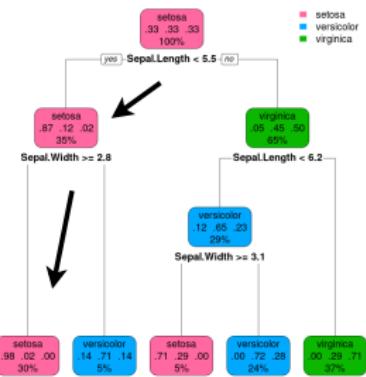
# CLASSIFICATION TREES



- Classification trees use the structure of a binary tree
- Binary splits are constructed top-down in a *data optimal* way
- Each split is a threshold decision for a single feature
- Each node contains the training points which follow its path
- Each leaf contains a constant prediction

# CLASSIFICATION TREE MODEL AND PREDICTION

- When predicting new data (here\*: Sepal.Length = 5, Sepal.Width = 3) we use the learned split points and pass an observation through the tree
- Each observation is assigned to exactly one leaf
- Classification trees can make hard-label predictions (here: setosa) or predict probabilities (here: 0.98, 0.02, 0.00)



# CART AS A RULE BASED MODEL

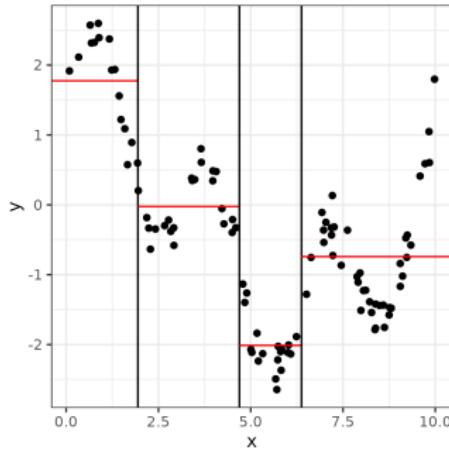
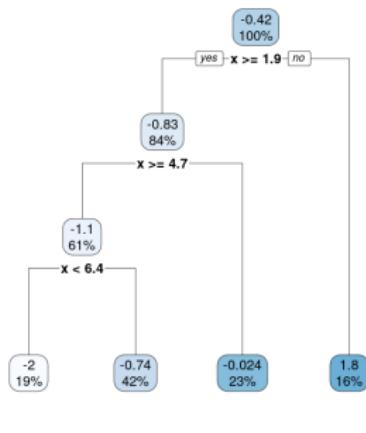
Leaf nodes can be expressed by a set of rules (left to right):

| Hard label prediction | Label distribution | Sepal.Width | Sepal.Length        |
|-----------------------|--------------------|-------------|---------------------|
| setosa                | 0.98, 0.02, 0.00   | $\geq 2.8$  | < 5.5               |
| versicolor            | 0.14, 0.71, 0.14   | < 2.8       | < 5.5               |
| setosa                | 0.71, 0.29, 0.00   | $\geq 3.1$  | $\geq 5.5 \& < 6.2$ |
| versicolor            | 0.00, 0.72, 0.28   | < 3.1       | $\geq 5.5 \& < 6.2$ |
| virginica             | 0.00, 0.29, 0.71   | -           | $\geq 6.2$          |



# REGRESSION TREE MODEL AND PREDICTION

- Works the same way as for classification
- But predictions in leaf nodes are a numerical scalar



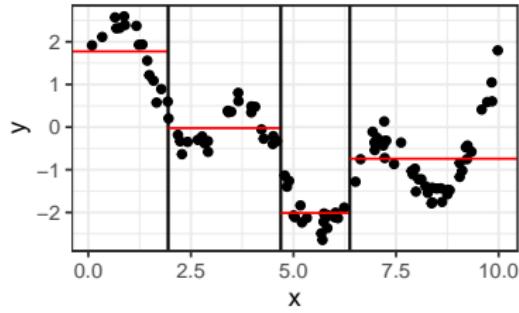
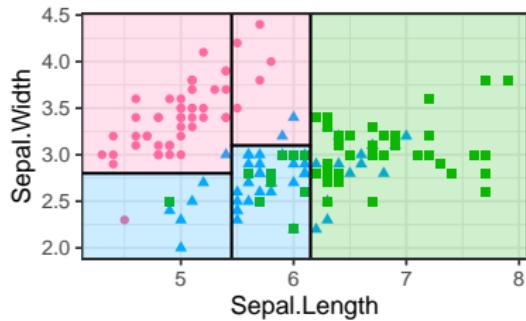
# TREE AS AN ADDITIVE MODEL

Trees divide the feature space  $\mathcal{X}$  into **rectangular regions**:

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

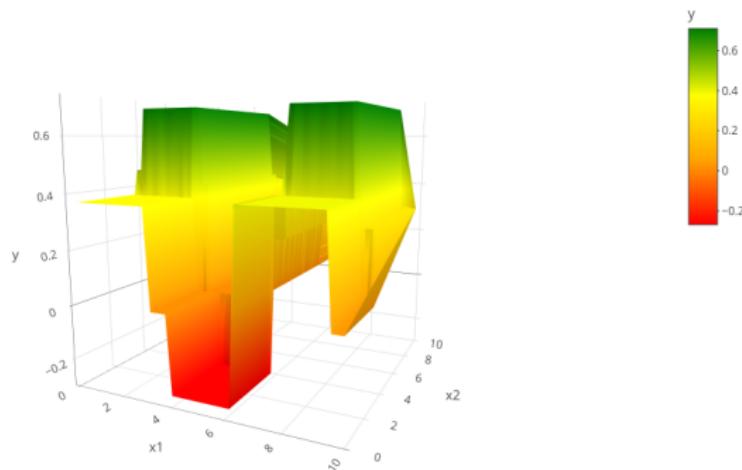
where a tree with  $M$  leaf nodes defines  $M$  “rectangles”  $Q_m$ .

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



# TREE AS AN ADDITIVE MODEL

A 2D regression example:



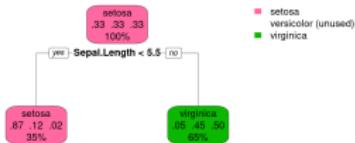
(For binary classification with probabilities, 2D surface looks similar.)

# Introduction to Machine Learning

## CART: Growing a Tree

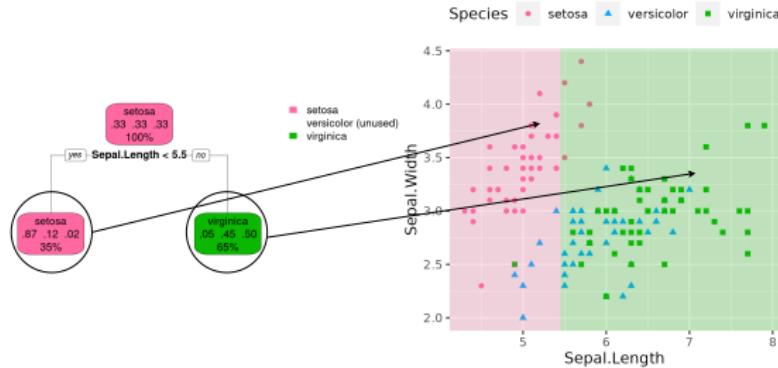
### Learning goals

- Understand how a tree is grown by an exhaustive search
- Know where and how the split point is set



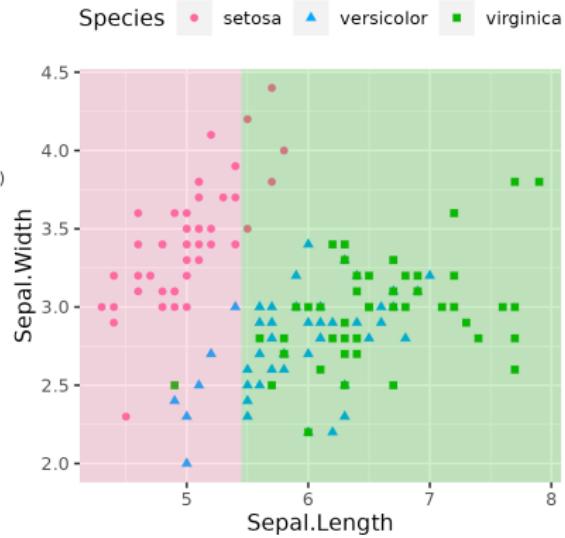
# TREE GROWING

- 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**: Ideally, 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.



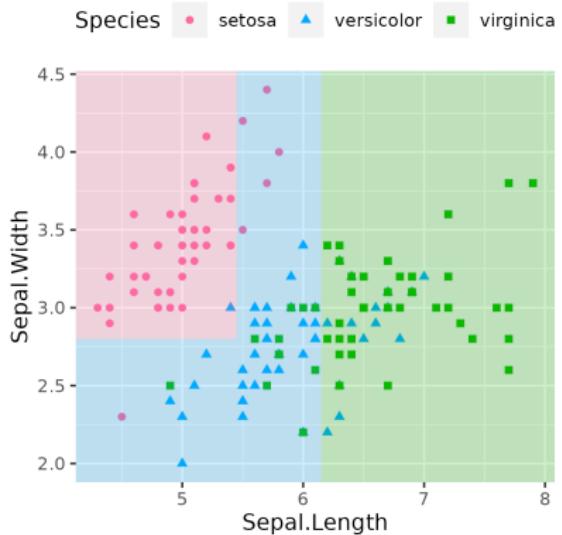
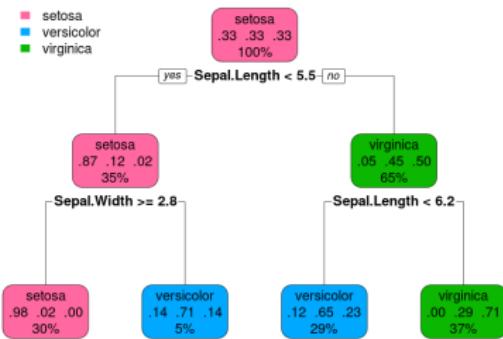
# TREE GROWING

- ① Start with a root node of all data.
- ② Search for feature and split point that minimizes the empirical risk in child nodes – makes label distribution more homogenous.



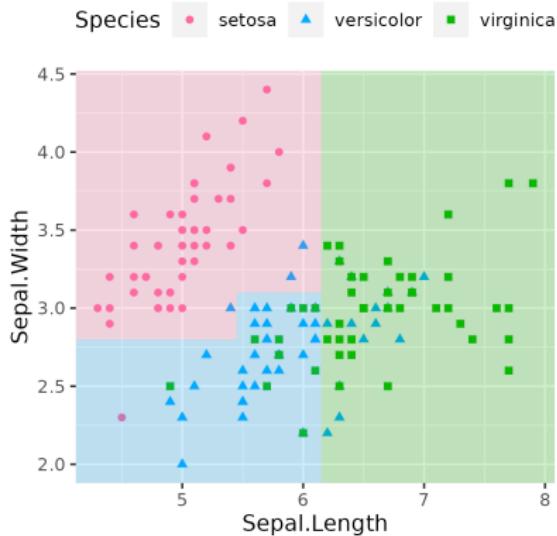
# TREE GROWING

- ③ Proceed recursively for each child node: Select best split and divide data from parent node into left and right child nodes.

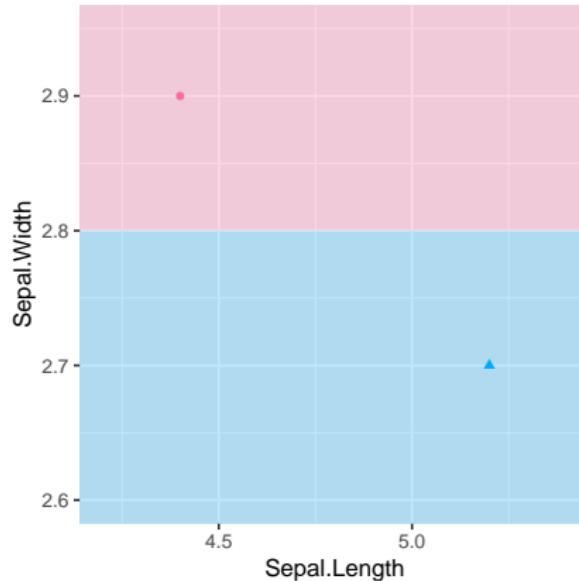


# TREE GROWING

- 4 Repeat until we reach a stop criterion, e.g., until each leaf cannot be split further.



# SPLIT PLACEMENT

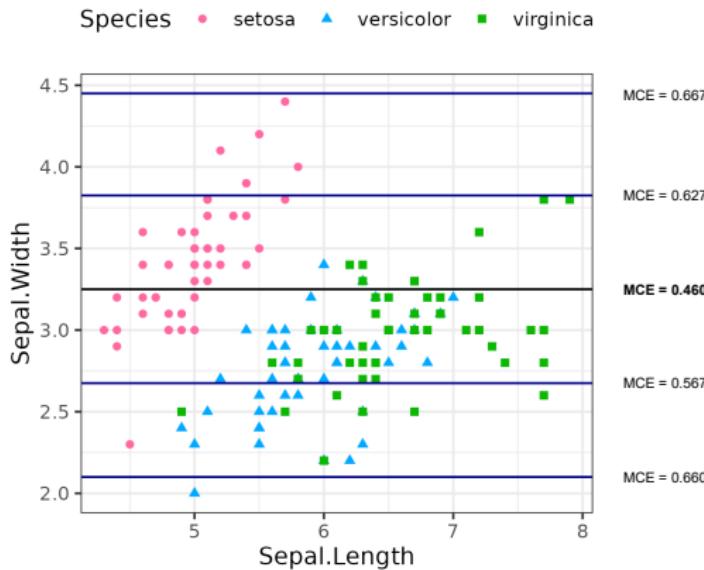


Splits are usually placed at the mid-point of the observations they split: the large margin to the next closest observations makes better generalization on new, unseen data more likely.

# FINDING THE SPLIT

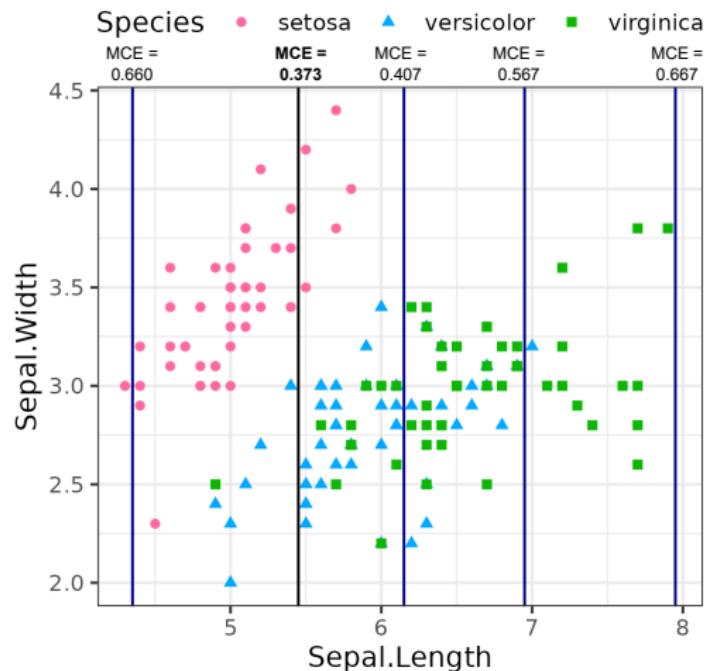
Assume we split the data so that the misclassification error (MCE) is minimal through the splitting.

First, we check a set of potential splits for `Sepal.Width`



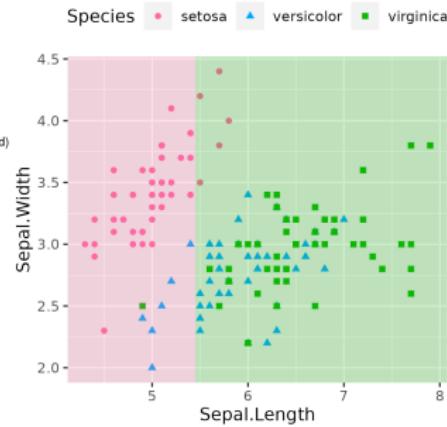
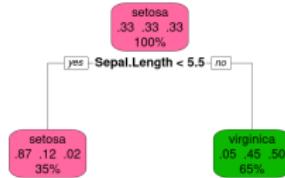
# FINDING THE SPLIT

Then we check a set of potential splits for Sepal.Length



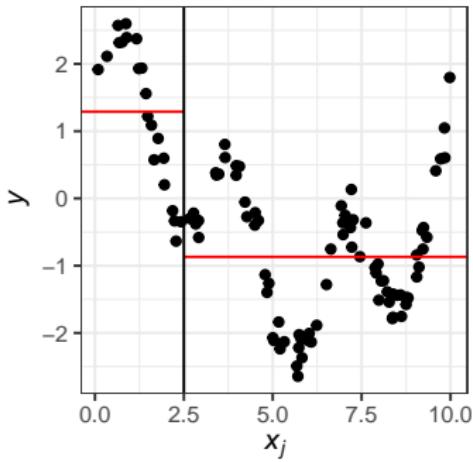
# FINDING THE SPLIT

- We take the split with lowest MCE:  $\text{Sepal.Length} = 5.5$
- In real life, we actually search over many more splitting points. Common strategies involve: a) Searching over all possible split points (exhaustive search), b) searching quantile-wise
- MCE is rarely used, we will cover split criteria in detail later.



# Introduction to Machine Learning

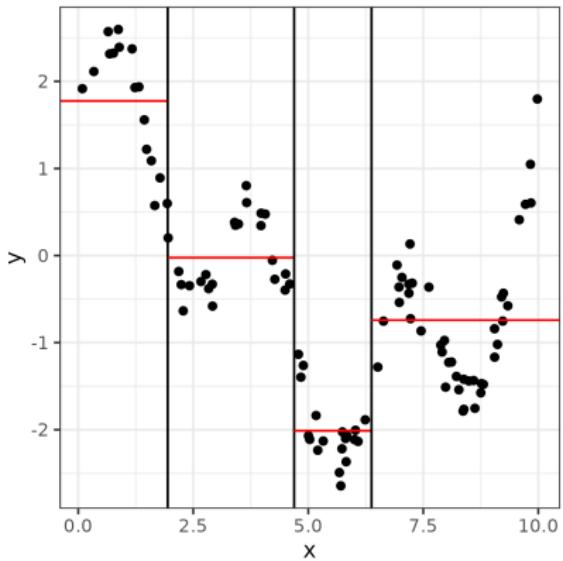
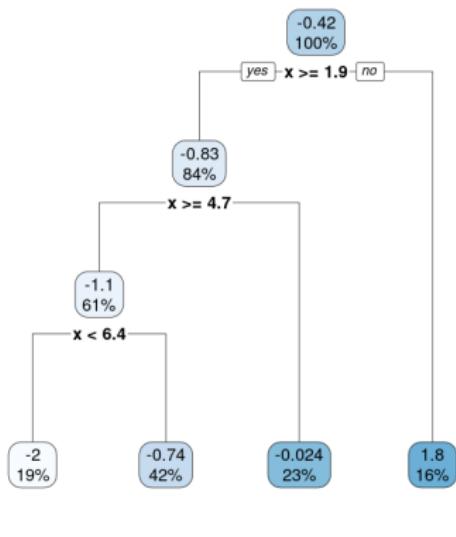
## CART: Splitting Criteria for Regression



### Learning goals

- Understand how to define split criteria via ERM
- Understand how to find splits in regression with  $L_2$  loss

# SPLITTING CRITERIA



How to find good splitting rules?  $\implies$  **Empirical Risk Minimization**

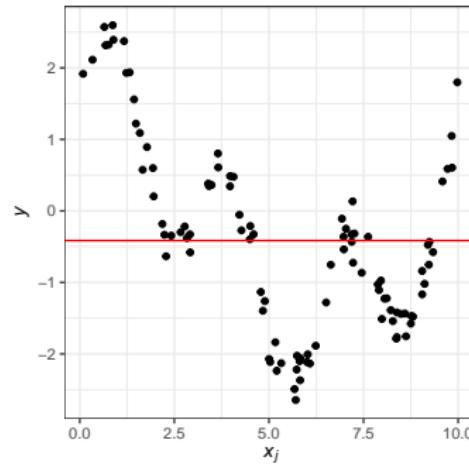
# OPTIMAL CONSTANTS IN LEAVES

Idea: A split is good if each child's point predictor reflects its data well.

For each child  $\mathcal{N}$ , predict with optimal constant, e.g., the mean

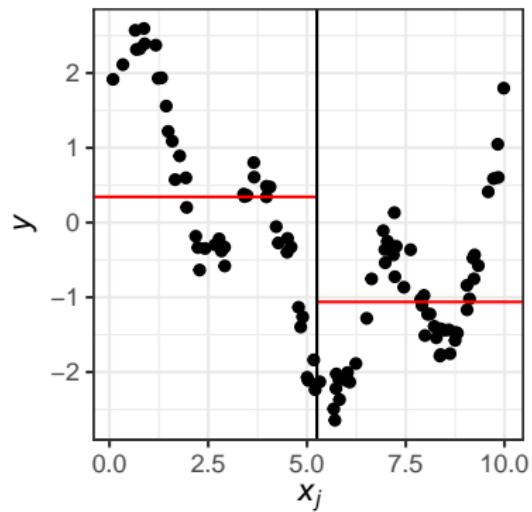
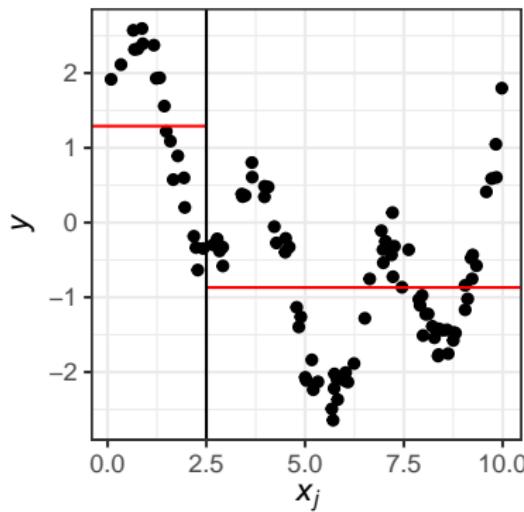
$$c_{\mathcal{N}} = \frac{1}{|\mathcal{N}|} \sum_{(\mathbf{x}, y) \in \mathcal{N}} y \text{ for the } L_2 \text{ loss, i.e., } \mathcal{R}(\mathcal{N}) = \sum_{(\mathbf{x}, y) \in \mathcal{N}} (y - c_{\mathcal{N}})^2.$$

Root node:

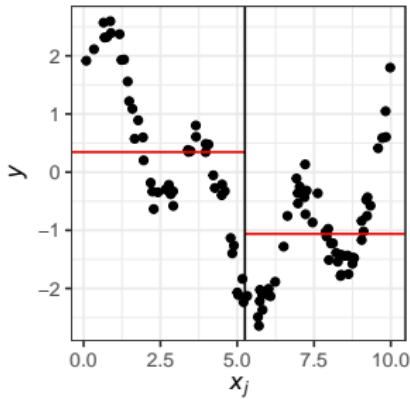
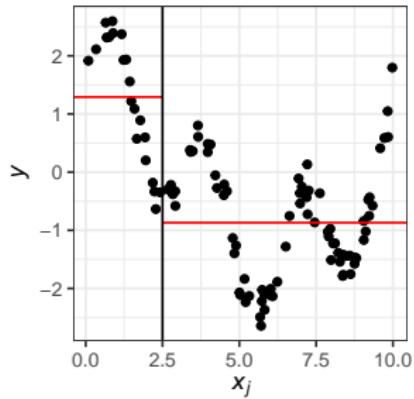


# OPTIMAL CONSTANTS IN LEAVES

Which of these two splits is better?



# RISK OF A SPLIT



$$\mathcal{R}(\mathcal{N}_1) = 23.4, \mathcal{R}(\mathcal{N}_2) = 72.4$$

$$\mathcal{R}(\mathcal{N}_1) = 78.1, \mathcal{R}(\mathcal{N}_2) = 46.1$$

The total risk is the sum of the individual losses:

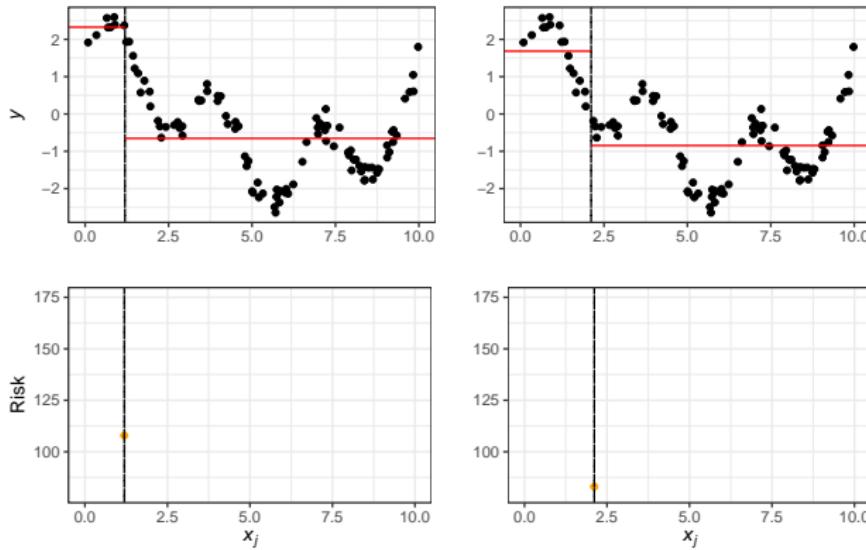
$$23.4 + 72.4 = 95.8$$

$$78.0 + 46.1 = 124.1$$

Based on the SSE, we prefer the first split.

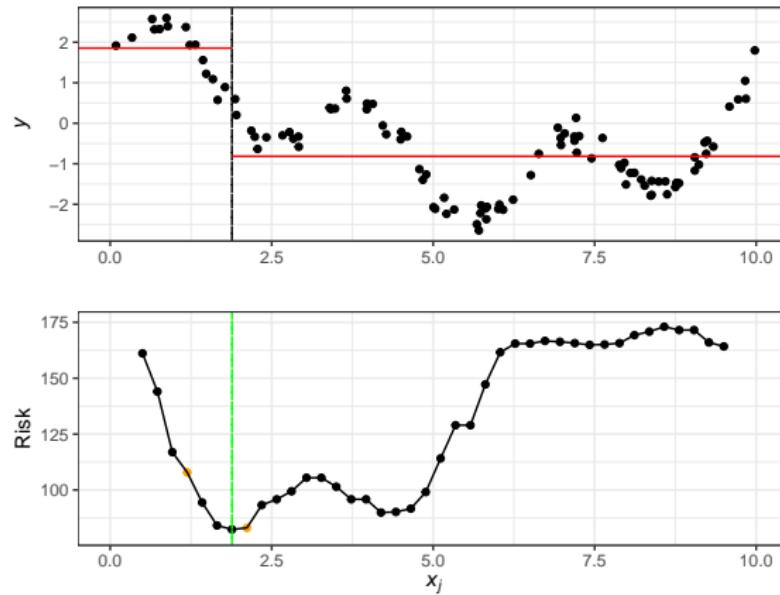
# SEARCHING THE BEST SPLIT

Let's find the best split for this feature by tabulating results.



# SEARCHING THE BEST SPLIT

Let's iterate – quantile-wise or over all points.



We have reduced the problem to a simple loop.

# FORMALIZATION

- $\mathcal{N} \subseteq \mathcal{D}$  is the data contained in this node
- Let  $c_{\mathcal{N}}$  be the predicted constant for  $\mathcal{N}$
- The risk  $\mathcal{R}(\mathcal{N})$  for a node is:

$$\mathcal{R}(\mathcal{N}) = \sum_{(\mathbf{x}, y) \in \mathcal{N}} L(y, c_{\mathcal{N}})$$

- The optimal constant is  $c_{\mathcal{N}} = \arg \min_c \sum_{(\mathbf{x}, y) \in \mathcal{N}} L(y, c)$
- We often know what that is from theoretical considerations – or we can perform a simple univariate optimization

# FORMALIZATION

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

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

- To evaluate its quality, we compute the risk of our new, finer model

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

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

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

# FORMALIZATION

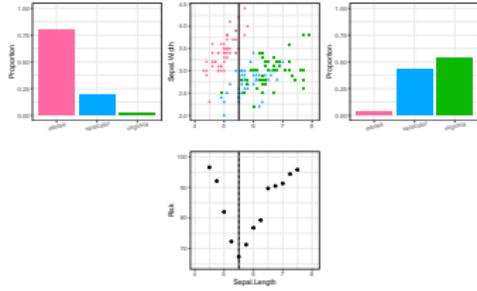
- $\mathcal{R}(\mathcal{N}, j, t) = \mathcal{R}(\mathcal{N}_1) + \mathcal{R}(\mathcal{N}_2)$ , makes sense if  $\mathcal{R}$  is a simple sum
- If we use averages, we have to reweight the terms to obtain a global average w.r.t.  $\mathcal{N}$  as the children have different sizes

$$\bar{\mathcal{R}}(\mathcal{N}, j, t) = \frac{|\mathcal{N}_1|}{|\mathcal{N}|} \bar{\mathcal{R}}(\mathcal{N}_1) + \frac{|\mathcal{N}_2|}{|\mathcal{N}|} \bar{\mathcal{R}}(\mathcal{N}_2)$$

- We mention this for clarity, as quite a few texts contain only the (more complicated) weighted formula without clear explanation

# Introduction to Machine Learning

## CART: Splitting Criteria for Classification



### Learning goals

- Understand different splitting criteria for classification
- Know the connections between empirical risk minimization and impurity minimization

# OPTIMAL CONSTANT MODELS

As losses in classification, we typically use:

- (Multi-class) Brier score  $L(y, \pi) = \sum_{k=1}^g (\pi_k - o_k(y))^2$ ,  
a.k.a.  $L_2$  loss on probabilities
- (Multi-class) Log loss  $L(y, \pi) = - \sum_{k=1}^g o_k(y) \log(\pi_k)$ ,  
as in logistic regression

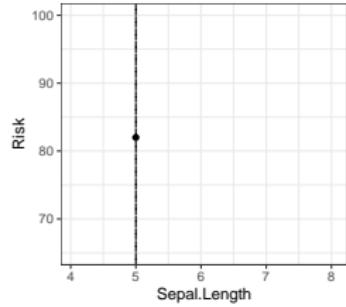
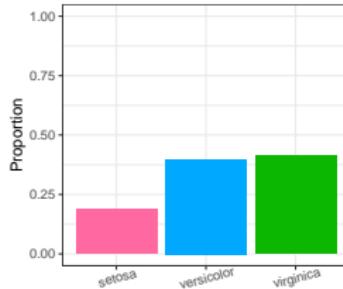
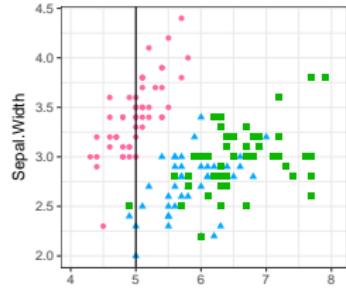
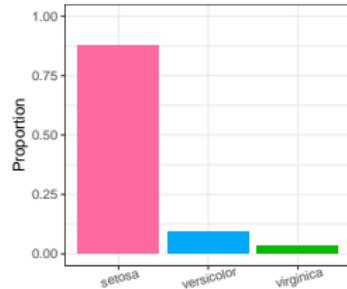
Optimal constant predictions (in a node) for both losses are simply the proportions of the contained classes:

$$c_{\mathcal{N}} = (\hat{\pi}_1^{(\mathcal{N})}, \dots, \hat{\pi}_g^{(\mathcal{N})}) \quad \text{with}$$

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

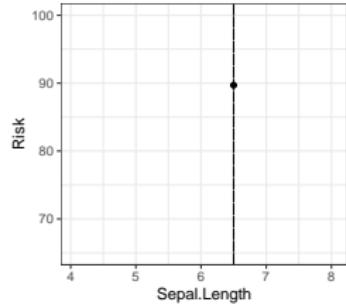
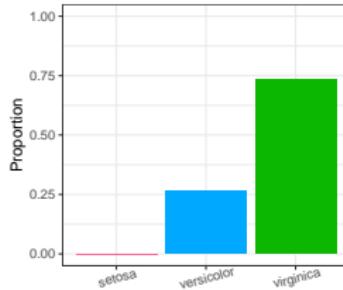
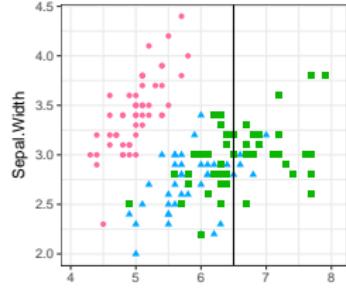
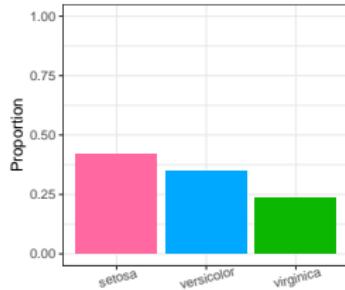
# FINDING THE BEST SPLIT

Let's compute the Brier score for all splits, with optimal constant probability vectors in both children



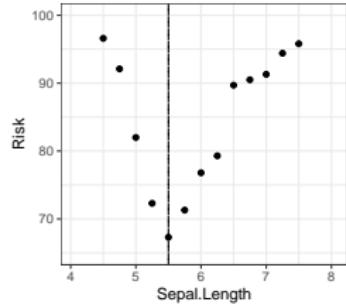
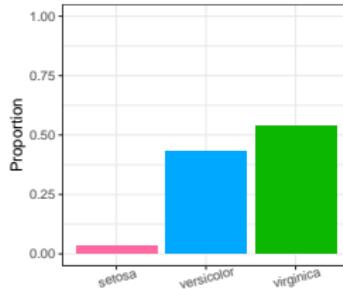
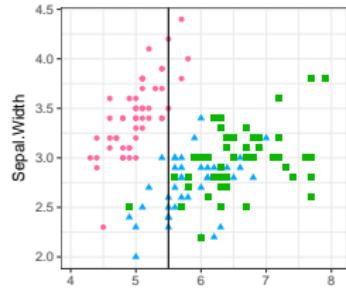
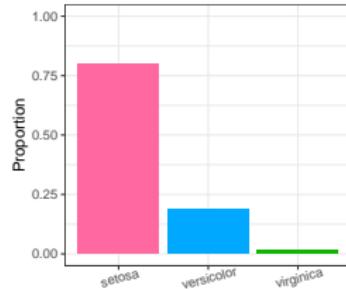
# FINDING THE BEST SPLIT

Let's compute the Brier score for all splits, with optimal constant probability vectors in both children



# FINDING THE BEST SPLIT

The optimal split point typically creates greatest imbalance or purity of label distribution



# RISK MINIMIZATION VS. IMPURITY

- Split crits are sometimes defined in terms of impurity reduction instead of ERM, where a measure of “impurity” is defined per node
- For regression trees, “impurity” is simply defined as variance of  $y$ , which is quite obviously  $L_2$  loss
- Brier score is equivalent to Gini impurity

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

- Log loss is equivalent to entropy

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

- Trees can be understood completely through the lens of ERM, so this new terminology is unnecessary and perhaps confusing

# SPLITTING WITH MISCLASSIFICATION LOSS

- Often, we want to minimize the MCE in classification
- Zero-One-Loss is not differentiable, but that is a non-issue in the tree-optimization based on loops
- Brier score and Log loss more sensitive to changes in the node probs, often produce purer nodes, and are still preferred

Split 1:

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

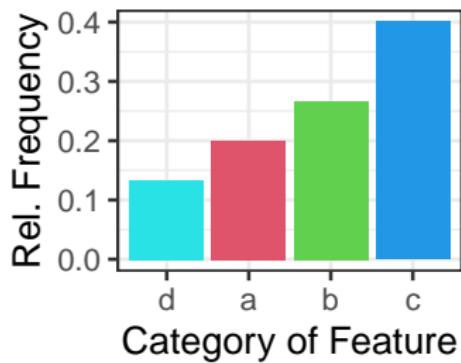
Split 2:

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

- Both splits are equivalent in MCE
- But: Split 2 results in purer nodes, both Brier score (Gini) and Log loss (Entropy) prefer 2nd split

# Introduction to Machine Learning

## CART: Computational Aspects of Finding Splits



### Learning goals

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

# MONOTONE FEATURE TRANSFORMATIONS

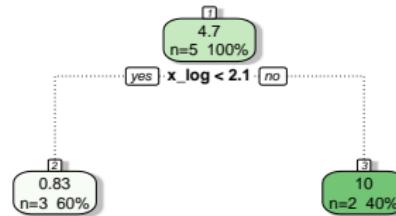
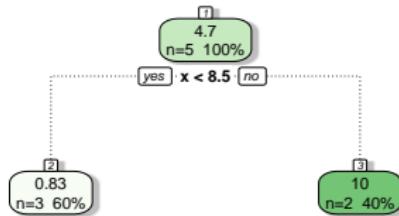
Monotone transformations of one or several features will neither change the value of the splitting criterion nor the structure of the tree, only the numerical value of the split point.

Original data

|   |     |     |     |      |      |
|---|-----|-----|-----|------|------|
| x | 1.0 | 2.0 | 7.0 | 10.0 | 20.0 |
| y | 1.0 | 1.0 | 0.5 | 9.0  | 11.0 |

Data with log-transformed x

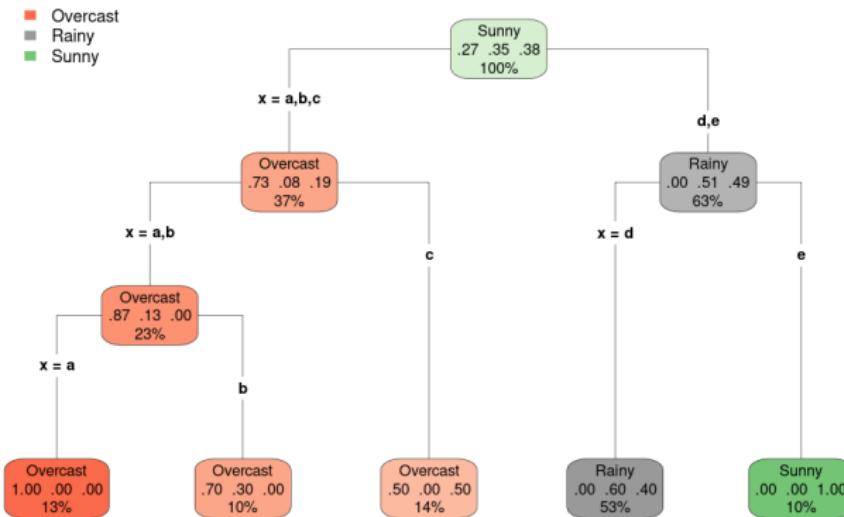
|        |     |     |     |     |      |
|--------|-----|-----|-----|-----|------|
| log(x) | 0.0 | 0.7 | 1.9 | 2.3 | 3.0  |
| y      | 1.0 | 1.0 | 0.5 | 9.0 | 11.0 |



# CATEGORICAL FEATURES

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

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



# CATEGORICAL FEATURES

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

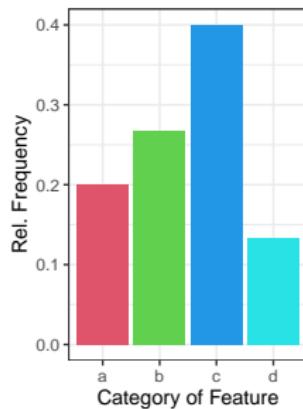
$$x_j \in \{a, b, c\} \leftarrow \mathcal{N} \rightarrow x_j \in \{d, e\}$$

- 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 large values of  $m$ .
- For regression with L2 loss and for binary classification, we can define clever shortcuts.

# CATEGORICAL FEATURES

For 0 – 1 responses, in each node:

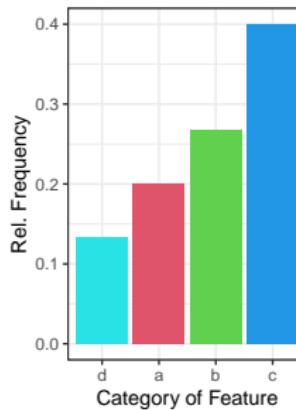
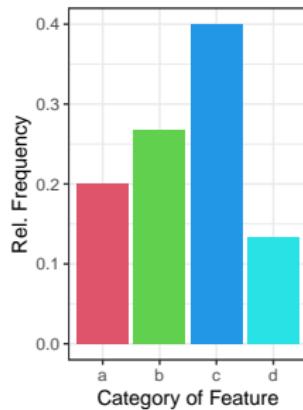
- 1 Calculate the proportion of 1-outcomes for each category of the feature in  $\mathcal{N}$ .



# CATEGORICAL FEATURES

For 0 – 1 responses, in each node:

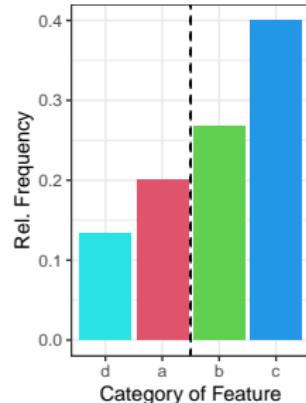
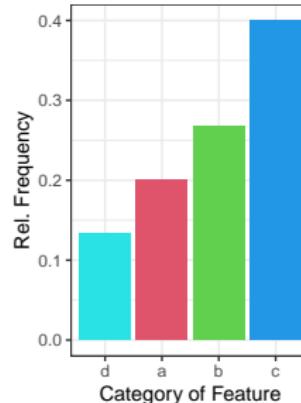
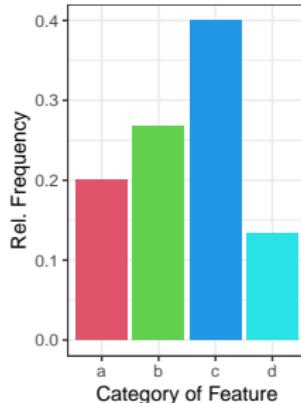
- ① Calculate the proportion of 1-outcomes for each category of the feature in  $\mathcal{N}$ .
- ② Sort the categories according to these proportions.



# CATEGORICAL 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.



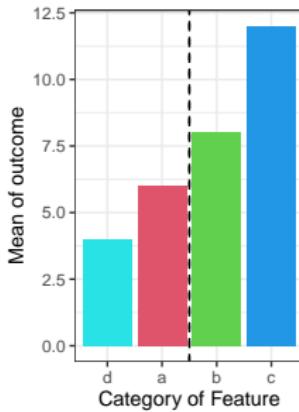
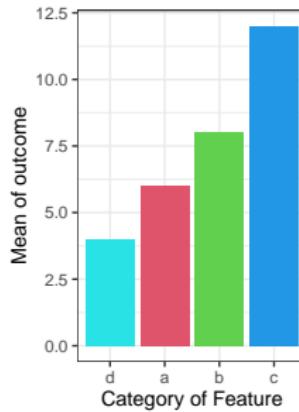
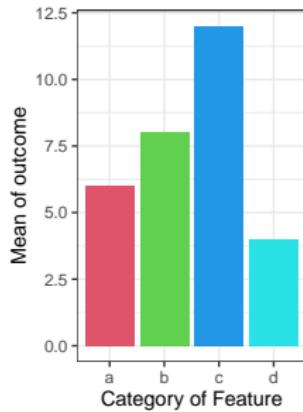
# CATEGORICAL FEATURES

- This procedure finds the optimal split.
- This result also holds for regression trees (with L2 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, Chapter 4
    - ▶ Ripley, 1996, pp. 213 et seqq.
  - for continuous responses:
    - ▶ Fisher, 1958
- There are only heuristics for the multiclass case ▶ Wright and König, 2019

# CATEGORICAL 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

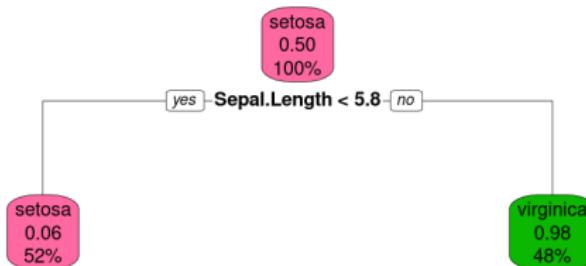


# MISSING FEATURE VALUES

- 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.)
- **Surrogate splits** can deal with missing values 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, and the feature value used in a split is missing, we use the surrogate split instead to decide to which child the data should be assigned.

# SURROGATE SPLITS

- Each surrogate split is a decision stump that tries to learn the actual splitting rule
- Consider this tree with the primary split w.r.t. Sepal.Length where we perform binary classification (setosa vs. virginica):



- Our surrogate split should optimize a splitting criterion w.r.t.  $\text{Sepal.Length} < 5.8$

# SURROGATE SPLITS

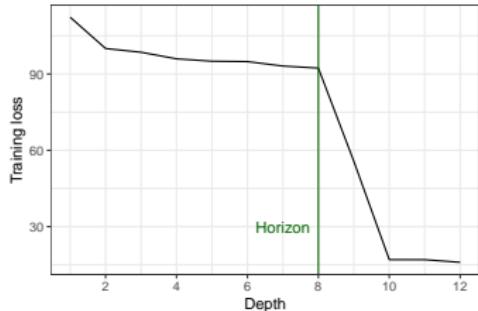
- Consider this subsample of the data used to fit the tree:

|    | Sepal.Length | ... | Petal.Width | Species   | Sepal.Length < 5.8 |
|----|--------------|-----|-------------|-----------|--------------------|
| 1  | 5.10         | ... | 0.20        | setosa    | TRUE               |
| 4  | 4.60         | ... | 0.20        | setosa    | TRUE               |
| 9  | 4.40         | ... | 0.20        | setosa    | TRUE               |
| 15 | 5.80         | ... | 0.20        | setosa    | FALSE              |
| 18 | 5.10         | ... | 0.30        | setosa    | TRUE               |
| 52 | 5.80         | ... | 1.90        | virginica | FALSE              |
| 57 | 4.90         | ... | 1.70        | virginica | TRUE               |
| 62 | 6.40         | ... | 1.90        | virginica | FALSE              |
| 77 | 6.20         | ... | 1.80        | virginica | FALSE              |
| 99 | 6.20         | ... | 2.30        | virginica | FALSE              |

- Add column that indicates whether Sepal.Length < 5.8
- Fit tree of depth 1 using all features but Sepal.Length to derive a split that explains Sepal.Length < 5.8 best  $\Rightarrow$  surrogate split
- Typically, software stores the best and a few more surrogate splits

# Introduction to Machine Learning

## CART: Stopping Criteria & Pruning

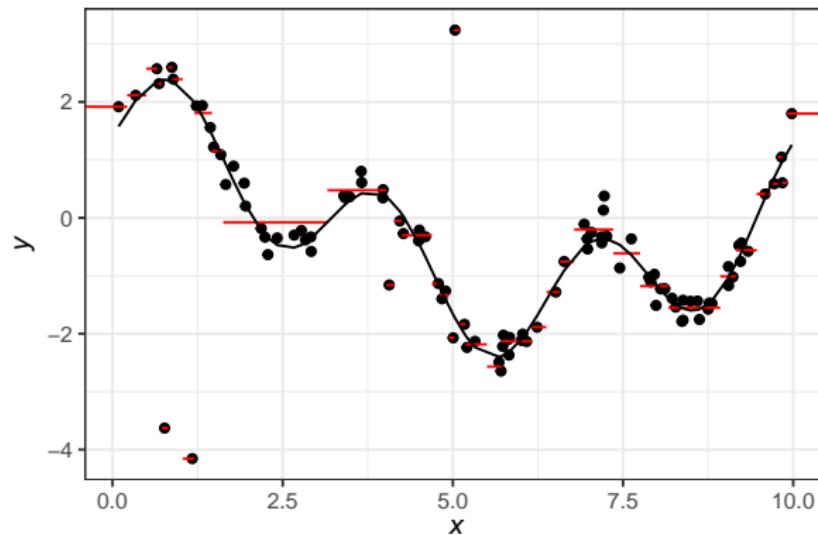


### Learning goals

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

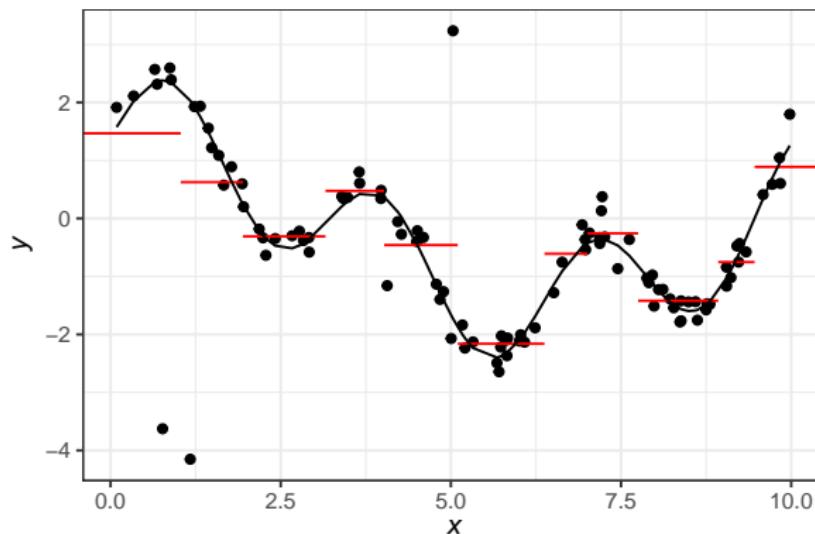
# OVERFITTING TREES

The recursive partitioning procedure used to grow a CART could run until every leaf only contains a single observation. Problem: Very complex trees will *overfit the training data*. At some point we should stop splitting nodes into ever smaller child nodes:



# OVERFITTING TREES

We can reduce overfitting to some extent with a less deep tree:



# STOPPING CRITERIA

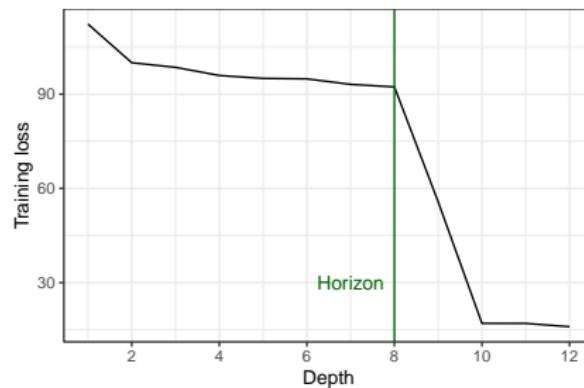
We can define different **stopping criteria**, e.g.: Don't split a node if

- a certain number of leaves if reached,
- it contains too few observations,
- splitting results in children with too few observations,
- splitting does not achieve a certain minimal improvement of the risk in the children, compared to the risk in the parent node,
- it already has the same target value (**pure node**) or identical feature values for all observations.

Selection of a stopping criterion and its concrete values are hyperparameters of CART.

# HORIZON EFFECT

It is 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*).



# PRUNING

We try to tackle the horizon effect 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 (see chapter on **tuning**).
- Alternative: Grow a large tree, then remove branches so that the resulting smaller tree has lower risk (“post-pruning”).
- Often, post-pruning is meant when referring to pruning.

## POST-PRUNING: CCP

- Prominent pruning method: Cost-complexity pruning (CCP)
- Idea: Grow a large tree and remove the least informative leaves
- CCP is steered with a regularization parameter  $\alpha$  that penalizes the number of leaves in a sub tree

$$\mathcal{R}_{\text{reg}}(T) = \sum_{m=1}^{|T|} \sum_{i: x^{(i)} \in Q_m} L(y^{(i)}, c_m) + \alpha |T|,$$

where  $|T|$  is the number of leaves of sub tree  $T$ ,  $Q_m$  is the subset of the feature space related to the  $m$ -th terminal node, with its prediction  $c_m$ , and  $T_0$  is the complete tree.

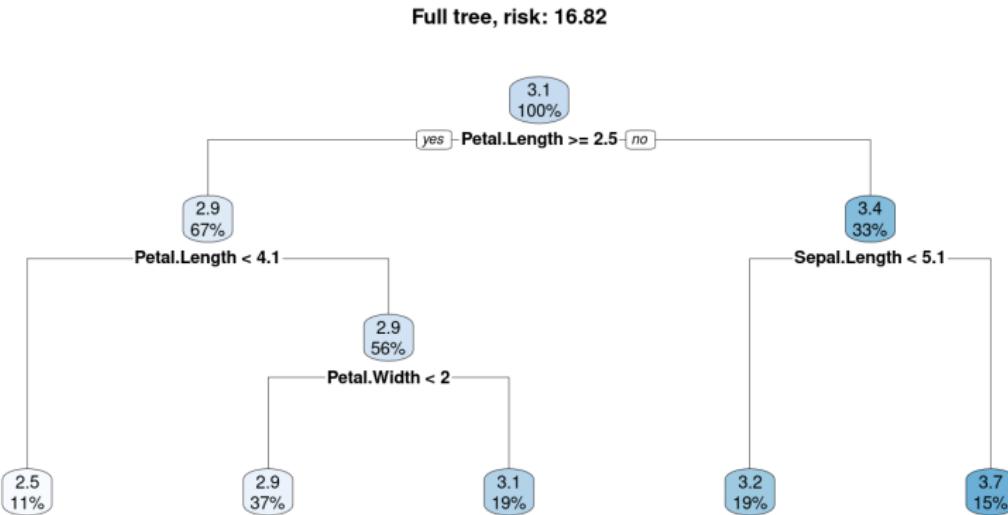
# CCP

CCP performs a greedy backward search:

- Computes  $\mathcal{R}_{\text{reg}}(T)$  with a fixed  $\alpha$  for all possible sub trees that can be created by replacing one internal node with a leaf.
- By replacing a node we also eliminate all subsequent nodes.
- We select the sub tree with lowest risk and repeat the procedure.
- We stop if pruning does not further reduce the risk.
- This is proven to result in the pruned tree with the lowest risk.
- For  $\alpha = 0$ , we would obviously select  $T_0$ .
- Hyperparameter  $\alpha$  is typically selected via cross-validation.
- Other prominent post-pruning methods include, e.g., reduced error pruning (REP) or pessimistic error pruning (PEP).

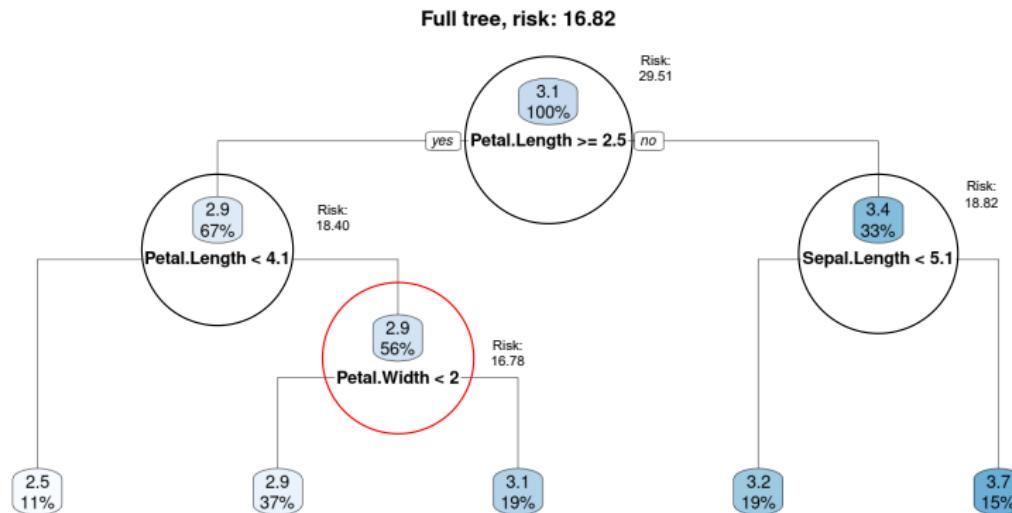
# CCP

We run the CCP algorithm step-by-step with  $\alpha = 1.2$ :



# CCP

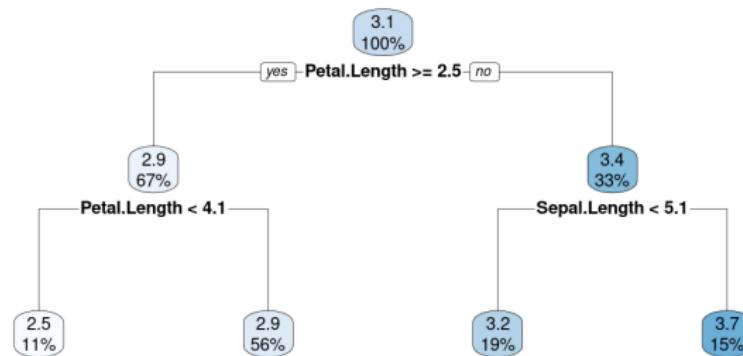
There are four possible nodes that we can eliminate to prune the tree. We take the one replacement that results in the lowest risk (red).



# CCP

The first pruned sub tree has a lower risk than the full tree. Thus, we prefer it over the full tree.

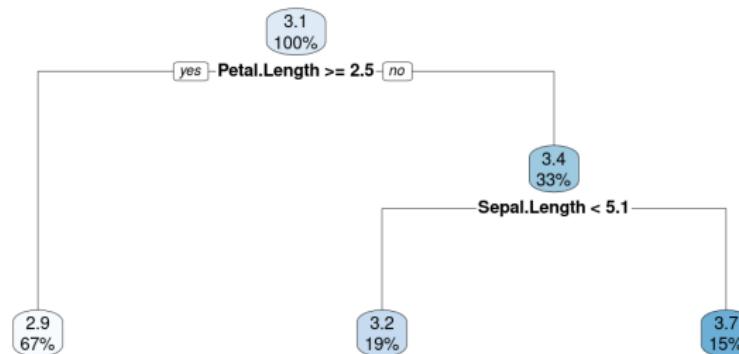
First pruned sub tree, risk: 16.78



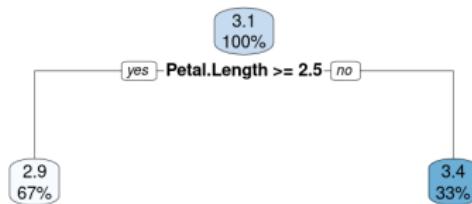
# CCP

From here on, the risk increases.

Second pruned sub tree, risk: 18.4



Third pruned sub tree, risk: 20.4



# CCP

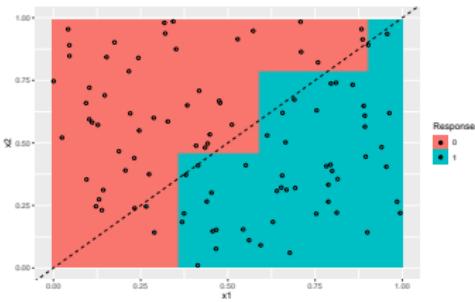
We select the first sub tree as it results in the lowest risk in the complete sequence of sub trees.

Fully pruned sub tree, risk: 29.51

3.1  
100%

# Introduction to Machine Learning

## CART: Advantages & Disadvantages



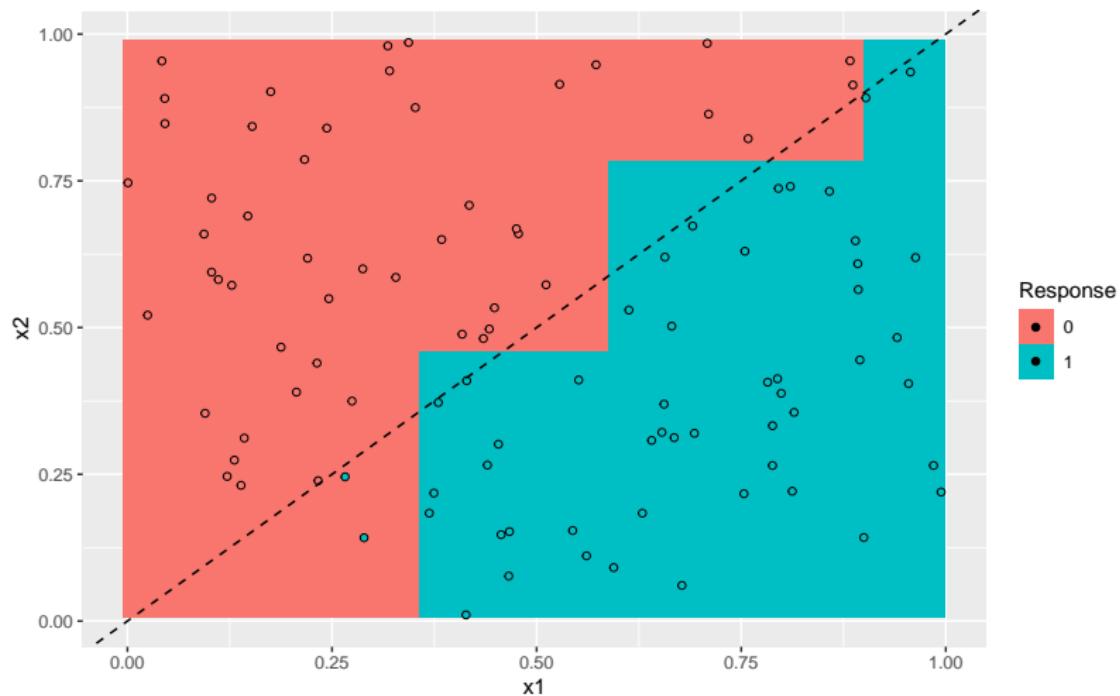
### Learning goals

- Understand the advantages and disadvantages of CART
- Know when and where CART are applied

# ADVANTAGES

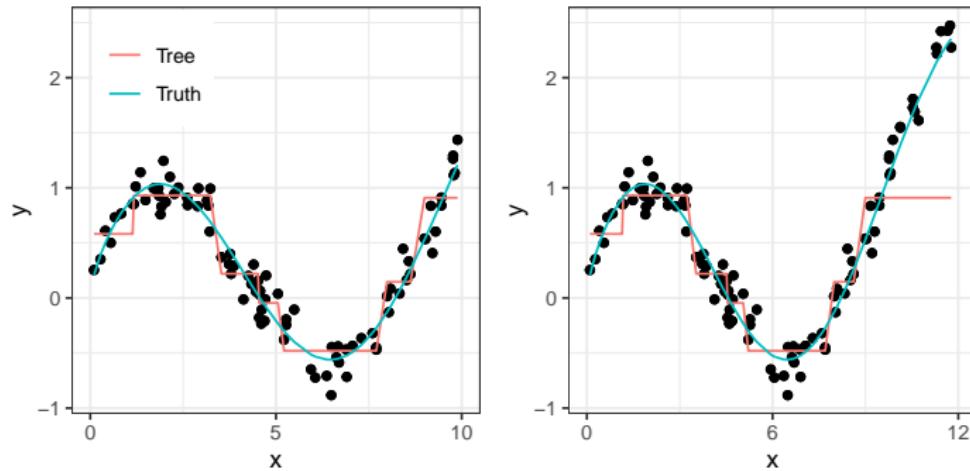
- Fairly easy to understand, interpret and visualize.
- Not much preprocessing required:
  - Automatic handling of non-numerical features
  - Automatic handling of missing values via surrogate splits
  - No problems with outliers in features
  - Monotone transformations do not affect the model fit: scaling becomes irrelevant
- Interaction effects between features are easily possible
- Can model discontinuities and non-linearities
- Performs automatic feature selection
- Relatively fast, scales well with larger data
- Flexibility through the 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 with fewer parameters.

# DISADVANTAGES: SMOOTH FUNCTIONS AND EXTRAPOLATION



Prediction functions of trees are never smooth as they are always step functions and do not extrapolate well beyond the training observations.

## DISADVANTAGE: INSTABILITY

- High instability (variance) of the trees
- Small changes in the data may lead to very different splits/trees
- This leads to a) less trust in interpretability b) is a reason why the prediction error of trees is usually comparably high

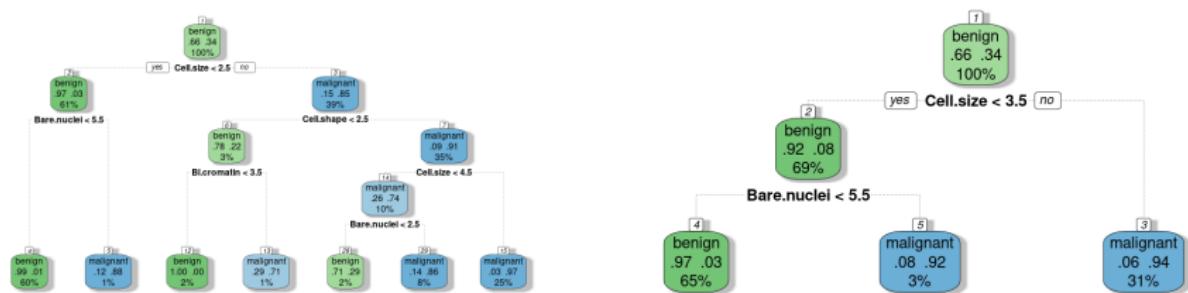
Consider the Wisconsin Breast Cancer data set with 699 observations on 9 features and binary target (“benign” vs. “malignant”). We fit two trees: (A) with the full data set and (B) where we eliminated a single observation. Results in label flip for 17 observations of the training data:

|           | benign | malignant |
|-----------|--------|-----------|
| benign    | 445    | 6         |
| malignant | 11     | 236       |

Rows: Predictions of (A), columns: Predictions of (B)

# DISADVANTAGE: INSTABILITY

The resulting decision trees look very different:



# CART IN PRACTICE

- Compared to other learners CART has suboptimal predictive performance, mainly because of the problems previously shown.
- However, most disadvantages can be overcome when trained in ensembles: bagging or random forests.
- Furthermore, trees are attractive tools if an interpretable model is desired or legally required.

# FURTHER TREE METHODOLOGIES

This lecture is mainly focused on Classification and Regression Trees (CART) ▶ Breiman, 1984

However, there are noteworthy other tree-based approaches as well:

- Automatic Interaction Detection (AID) ▶ Sonquist and Morgan, 1964 and Chi-squared Automatic Interaction Detection (CHAID) ▶ Kass, 1980 : Creates all possible cross-tabulations for each categorical predictor until the best outcome is achieved and no further splitting can be performed
- C4.5 ▶ Quinlan, 1993 : Not limited to binary splitting
- Unbiased Recursive Partitioning ▶ Hothorn et al., 2006 : Improves the variable selection algorithm

# INTRODUCTION TO MACHINE LEARNING

ML Basics

Supervised Regression

Supervised Classification

Performance Evaluation

k-NN

Classification and Regression Trees (CART)

**Random Forests**

Neural Networks

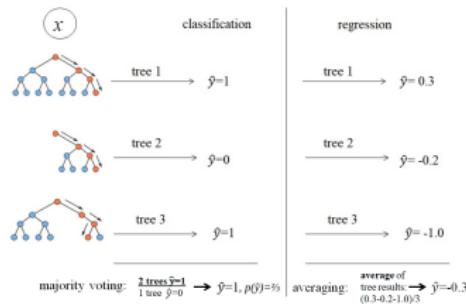
Tuning

Nested Resampling

# 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

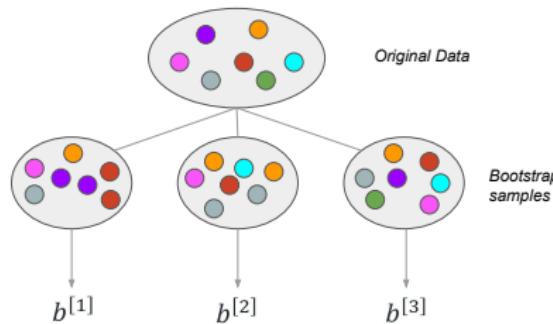
# BAGGING

- 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  $\mathcal{D}$ :

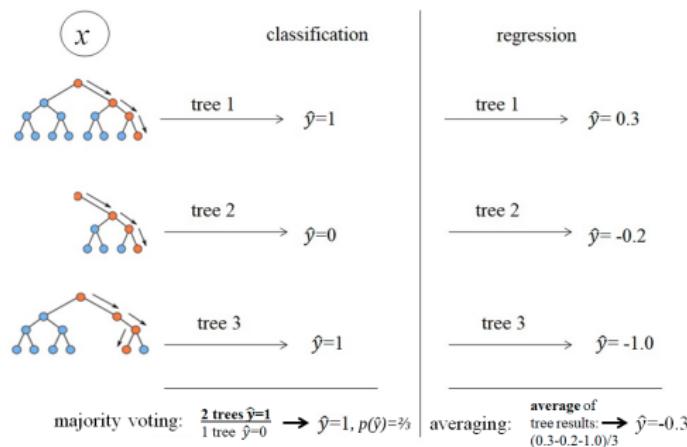
- Draw  $n$  observations from  $\mathcal{D}$  with replacement
- Fit the base learner on each of the  $M$  bootstrap samples to get models  $\hat{f}(x) = \hat{b}^{[m]}(\mathbf{x})$ ,  $m = 1, \dots, M$



# 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.

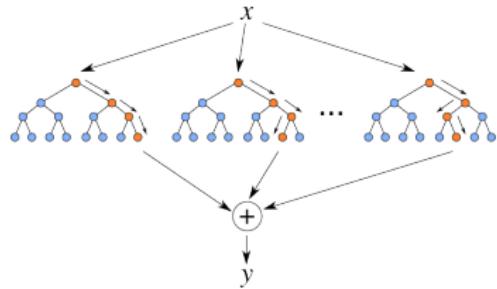
# BAGGING: SYNOPSIS

- 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

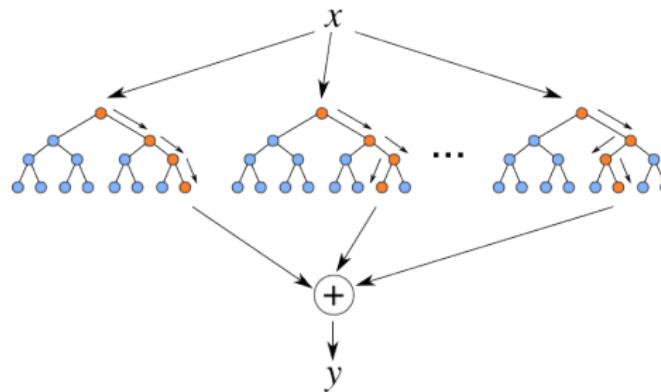


- Know how random forests are defined by extending the idea of bagging
- Understand that the goal is to decorrelate the trees
- Understand that the out-of-bag error is a way to obtain unbiased estimates of the generalization error during training

# RANDOM FORESTS

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

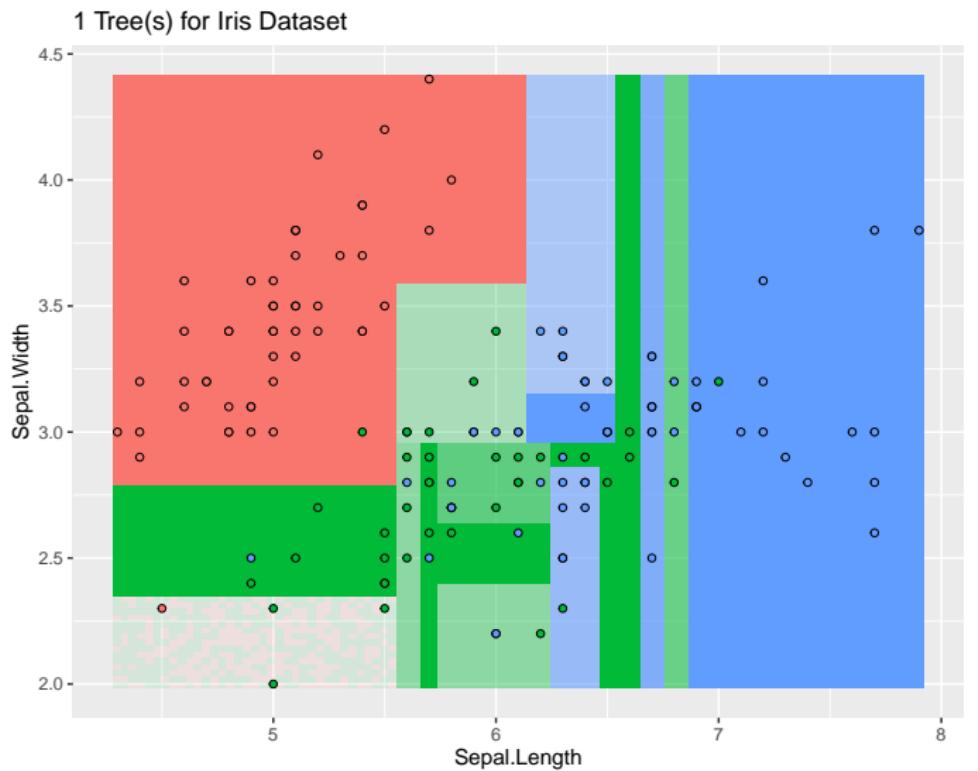
- Tree base learners on bootstrap samples of the data
- Uses **decorrelated** trees by randomizing splits (see below)
- Tree base learners are usually fully expanded, without aggressive early stopping or pruning, to **increase variance of the ensemble**



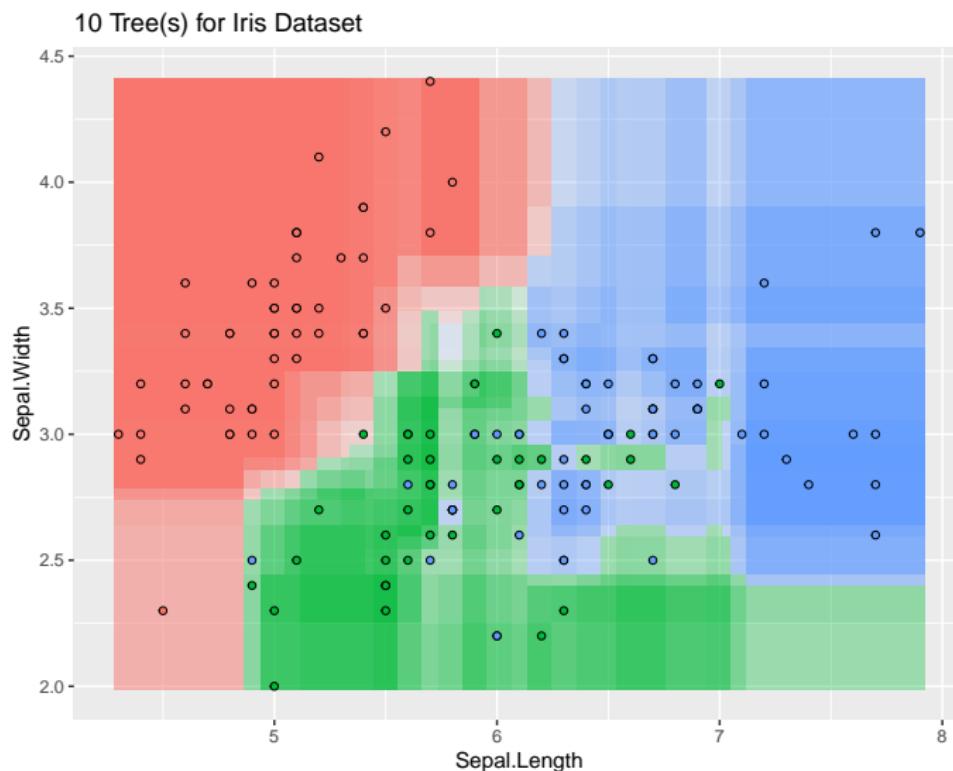
# RANDOM FEATURE SAMPLING

- 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  $mtry \leq p$  candidate features to consider for splitting. Recommended values:
  - Classification:  $mtry = \lfloor \sqrt{p} \rfloor$
  - Regression:  $mtry = \lfloor p/3 \rfloor$

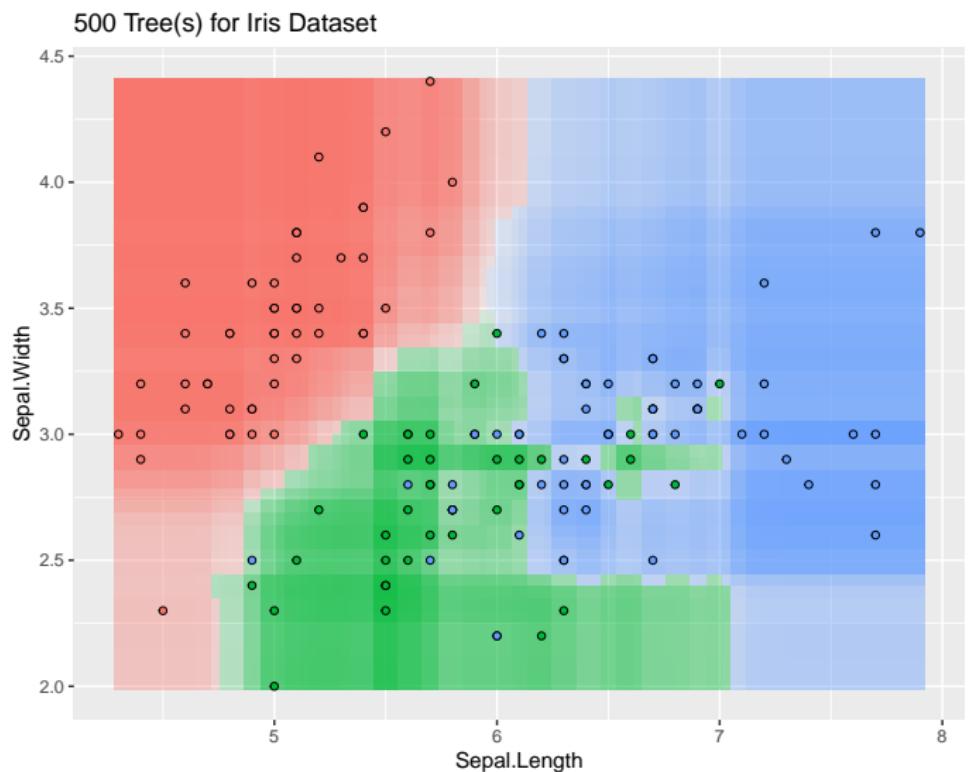
# EFFECT OF ENSEMBLE SIZE



# EFFECT OF ENSEMBLE SIZE

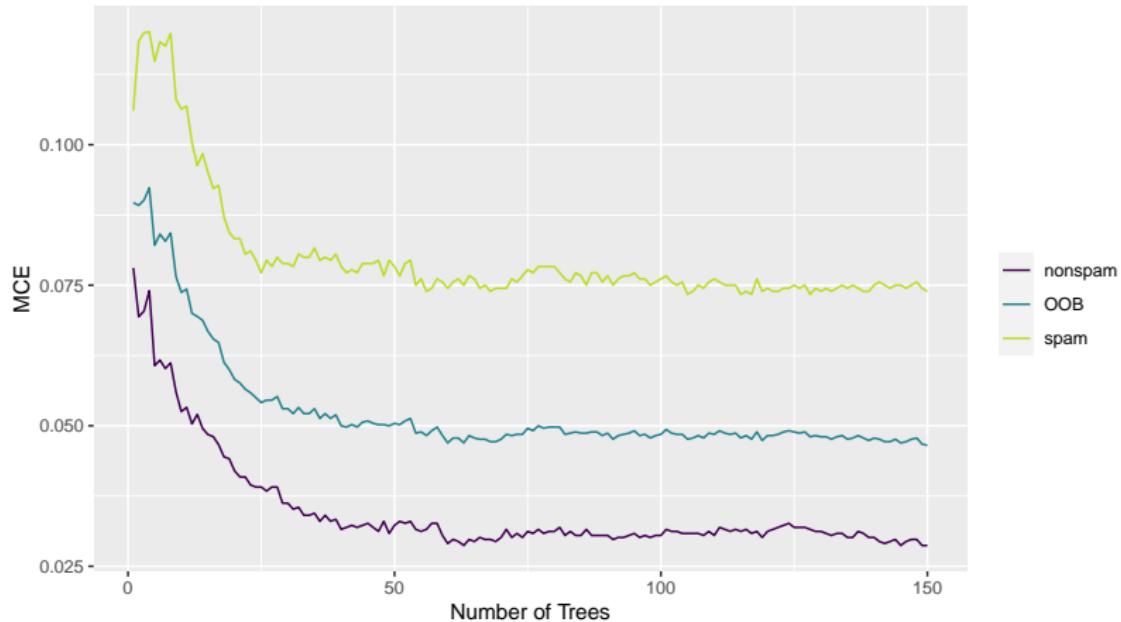


# EFFECT OF ENSEMBLE SIZE



# OUT-OF-BAG ERROR ESTIMATE

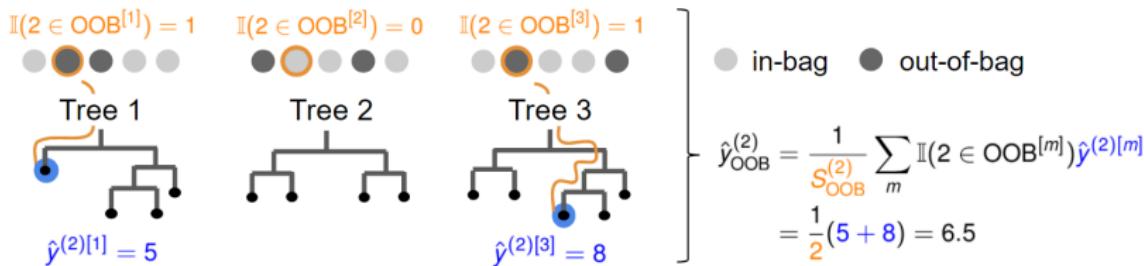
With the RF it is possible to obtain unbiased estimates of the generalization error directly during training, based on the out-of-bag observations for each tree:



# OUT-OF-BAG PREDICTIONS

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

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



# OUT-OF-BAG ERROR

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

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

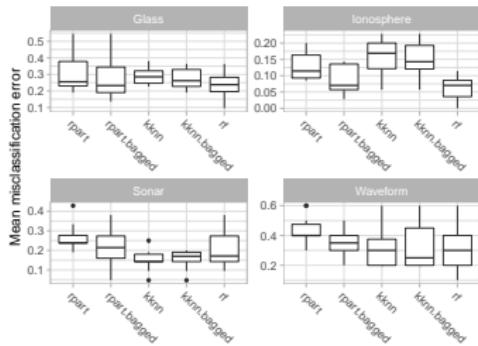
- Computing the probability of  $i$  being OOB in the  $m$ -th tree, we can see that the OOB error estimation is actually akin to 3-fold CV:

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

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

# Introduction to Machine Learning

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



### Learning goals

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

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

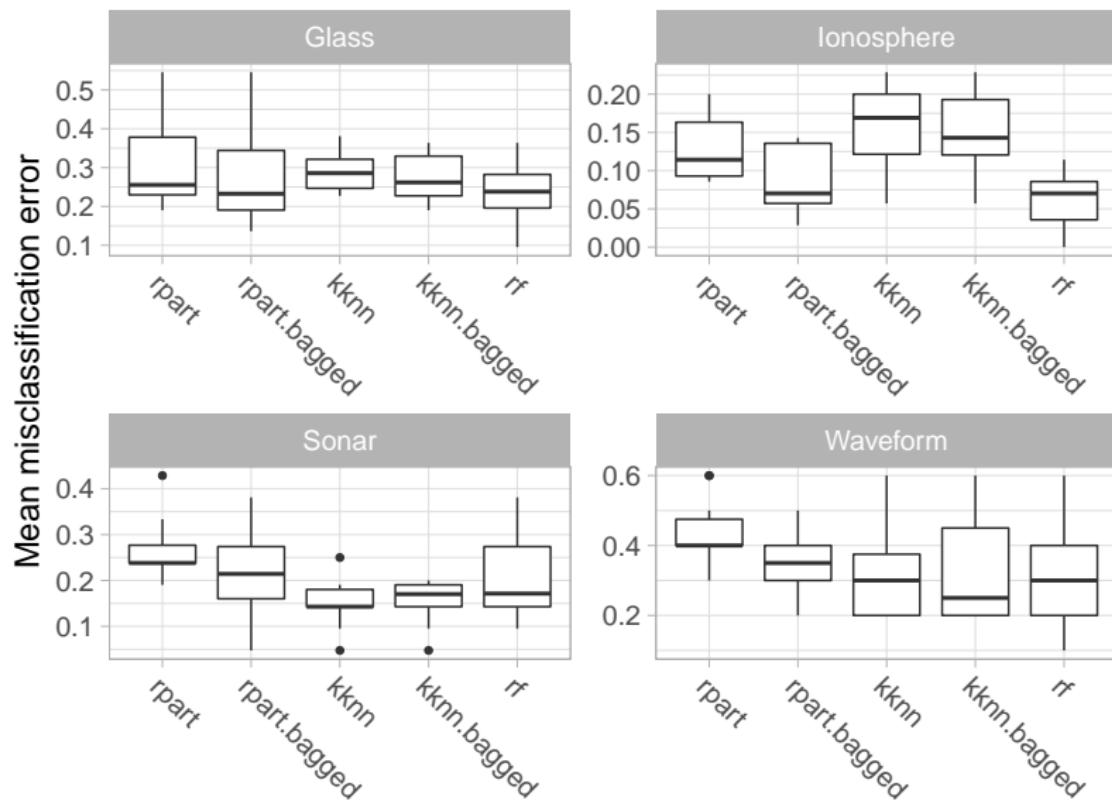
- Goal: Compare performance of random forest against (bagged) stable and (bagged) unstable methods
- Algorithms:
  - classification tree (CART, implemented in `rpart`,  
`max.depth: 30, min.split: 20, cp: 0.01`)
  - bagged classification tree using 50 bagging iterations  
(`bagged.rpart`)
  - k-nearest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

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

- Datasets from **mlbench**:

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

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



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

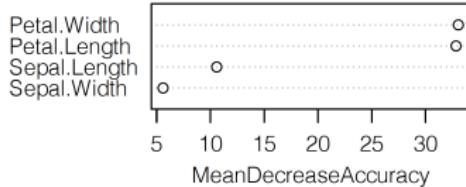
Bagging k-NN does not improve performance because:

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

# Introduction to Machine Learning

## Random Forests: Feature Importance

### Learning goals



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

# VARIABLE IMPORTANCE

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

# VARIABLE IMPORTANCE

---

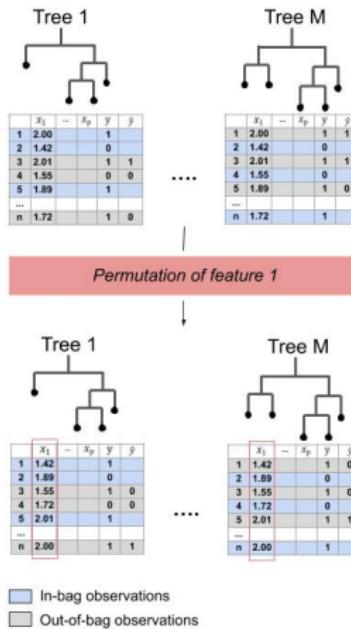
Measure based on improvement in split criterion

---

```
for features  $x_j, j = 1$  to  $p$  do
    for tree base learners  $\hat{b}^{[m]}, m = 1$  to  $M$  do
        Find all nodes  $\mathcal{N}$  in  $\hat{b}^{[m]}$  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 obs.

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

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

    Perform permutation  $\psi_j$  on  $x_j$  to distort  
    feature-target relation for  $x_j$ .

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

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

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

**end for**

    Estimate importance of  $j$ -th variable

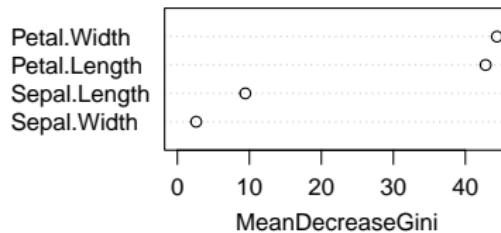
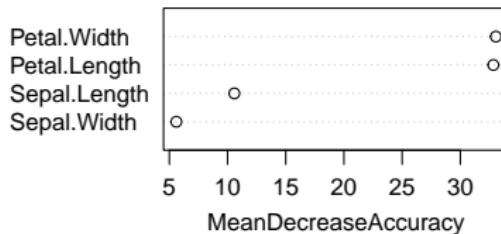
$$\widehat{VI}_j = \widehat{\text{err}}_{\text{OOB}, \psi_j} - \widehat{\text{err}}_{\text{OOB}}$$

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

**end for**

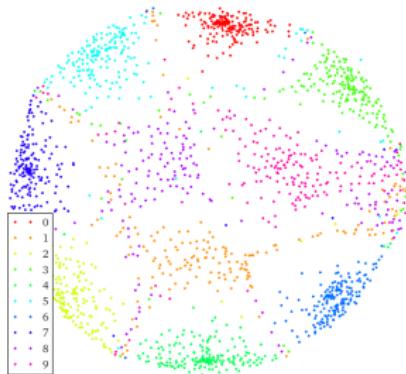
# VARIABLE IMPORTANCE

- Measure based on improvement in split criterion:  
MeanDecreaseGini (average total decrease in node impurities, measured by the Gini index)
- Measure based on permutations of OOB observations:  
MeanDecreaseAccuracy (average decrease in accuracy for predictions of OOB observations after permuting the  $j$ -th feature)



# 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

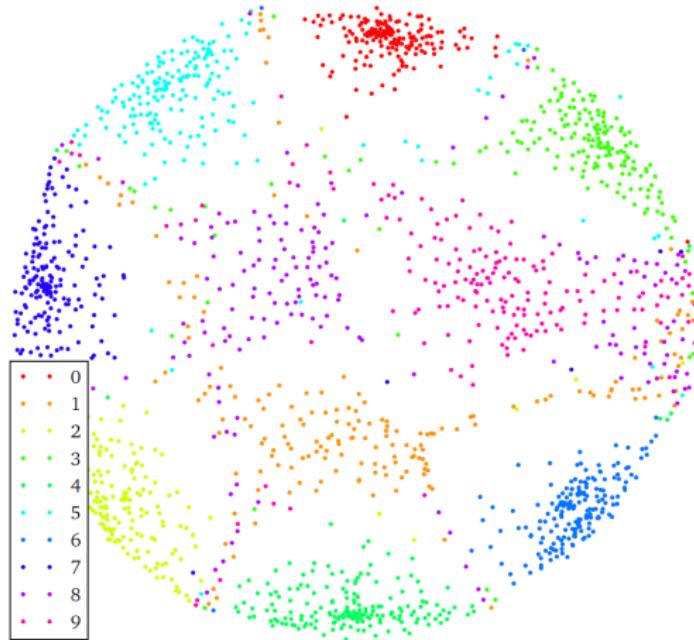
Steps 2 and 3 are then iterated a few times.

- 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

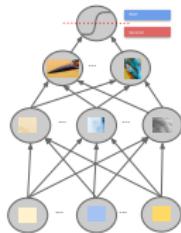
## **Neural Networks**

Tuning

Nested Resampling

# Deep Learning

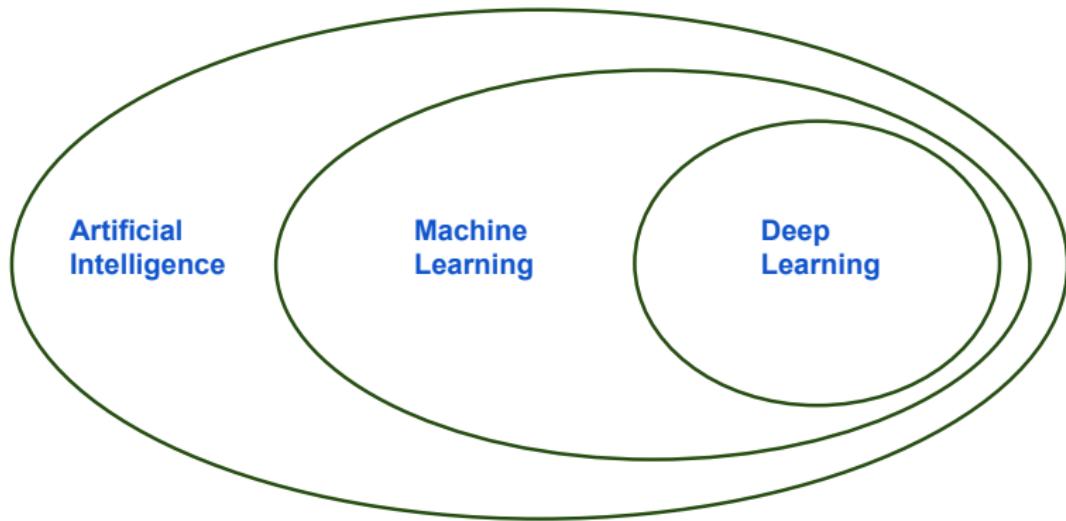
## Introduction



### Learning goals

- Relationship of DL and ML
- Concept of representation or feature learning
- Use-cases and data types for DL methods

# WHAT IS DEEP LEARNING



- Deep learning is a subfield of ML based on artificial neural networks.

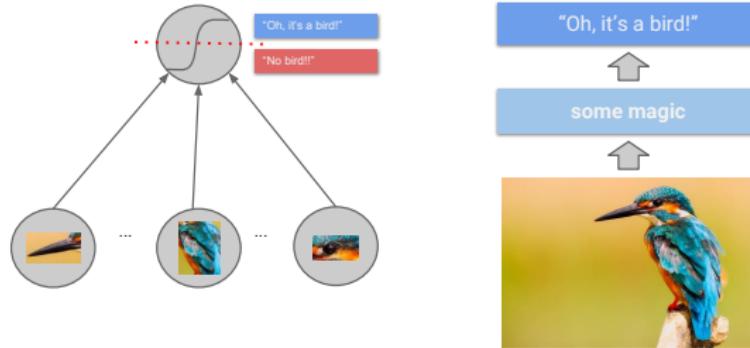
# DEEP LEARNING AND NEURAL NETWORKS

- Deep learning itself is not *new*:
  - Neural networks have been around since the 70s.
  - *Deep* neural networks, i.e., networks with multiple hidden layers, are not much younger.
- Why everybody is talking about deep learning now:
  - ❶ Specialized, powerful hardware allows training of huge neural networks to push the state-of-the-art on difficult problems.
  - ❷ Large amount of data is available.
  - ❸ Special network architectures for image/text data.
  - ❹ Better optimization and regularization strategies.

# IMAGE CLASSIFICATION WITH NEURAL NETWORKS

*"Machine learning algorithms, inspired by the brain, based on learning multiple levels of representation/abstraction."*

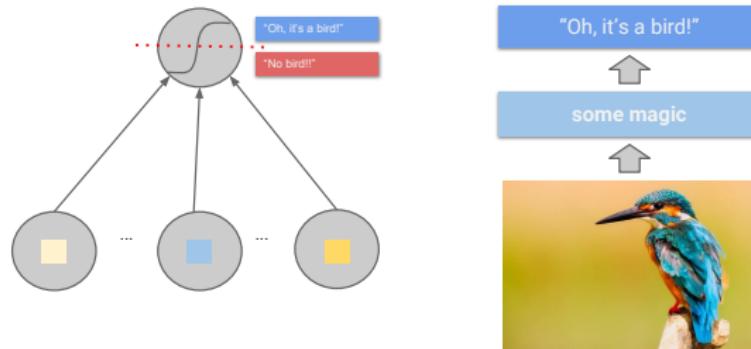
Y. Bengio



# IMAGE CLASSIFICATION WITH NEURAL NETWORKS

*"Machine learning algorithms, inspired by the brain, based on learning multiple levels of representation/abstraction."*

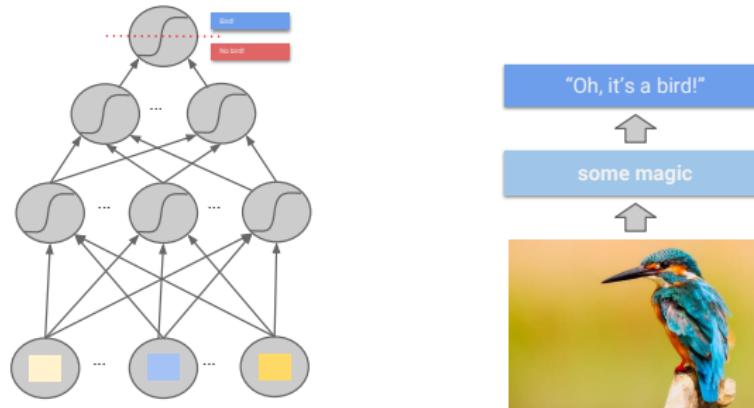
Y. Bengio



# IMAGE CLASSIFICATION WITH NEURAL NETWORKS

*"Machine learning algorithms, inspired by the brain, based on learning multiple levels of representation/abstraction."*

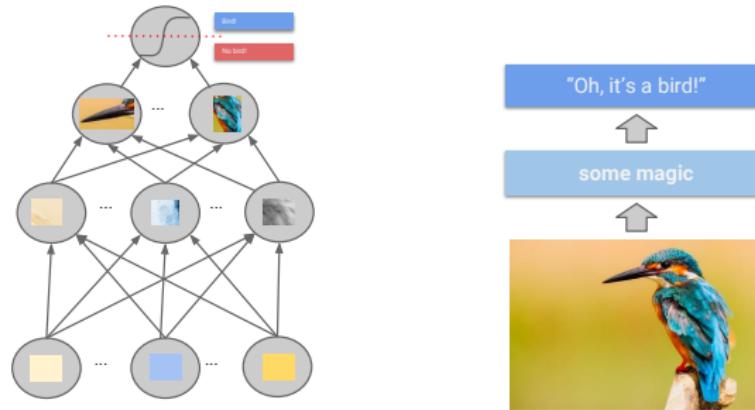
Y. Bengio



# IMAGE CLASSIFICATION WITH NEURAL NETWORKS

*“Machine learning algorithms, inspired by the brain, based on learning multiple levels of representation/abstraction.”*

Y. Bengio



# POSSIBLE USE-CASES

**Deep learning can be extremely valuable if the data has these properties:**

- It is high dimensional.
- Each single feature itself is not very informative but only a combination of them might be.
- There is a large amount of training data.

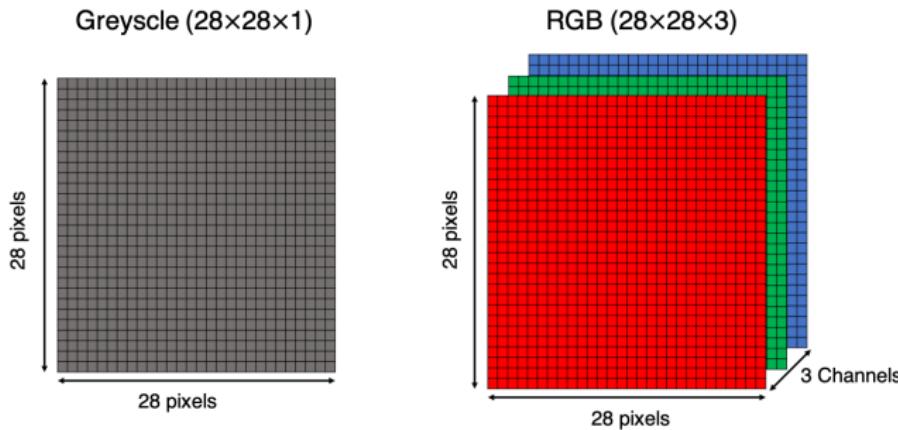
**This implies that for tabular data, deep learning is rarely the correct model choice.**

- Without extensive tuning, models like random forests or gradient boosting will outperform deep learning most of the time.
- One exception is data with categorical features with many levels.

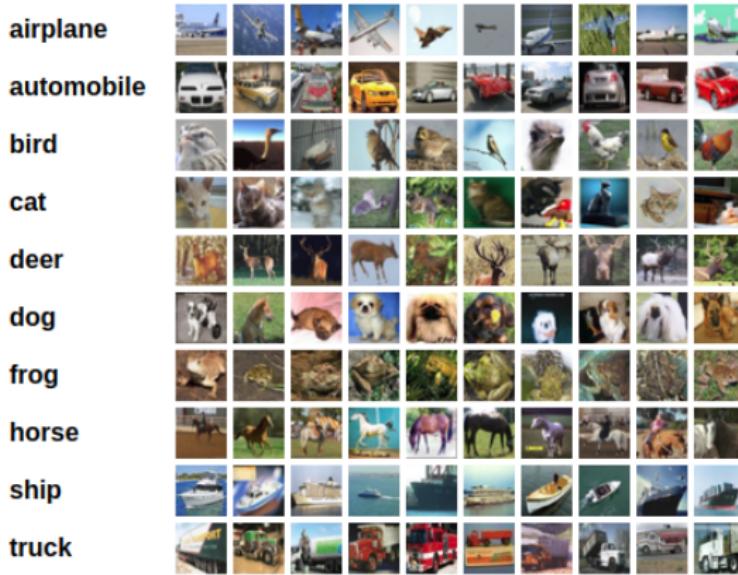
# POSSIBLE USE-CASE: IMAGES

- **High Dimensional:** A color image with  $255 \times 255$  (3 Colors) pixels already has 195075 features.
- **Informative:** A single pixel is not meaningful in itself.
- **Training Data:** Depending on applications huge amounts of data are available.

Architecture: **Convolutional Neural Networks (CNN)**



# POSSIBLE USE-CASE: IMAGES

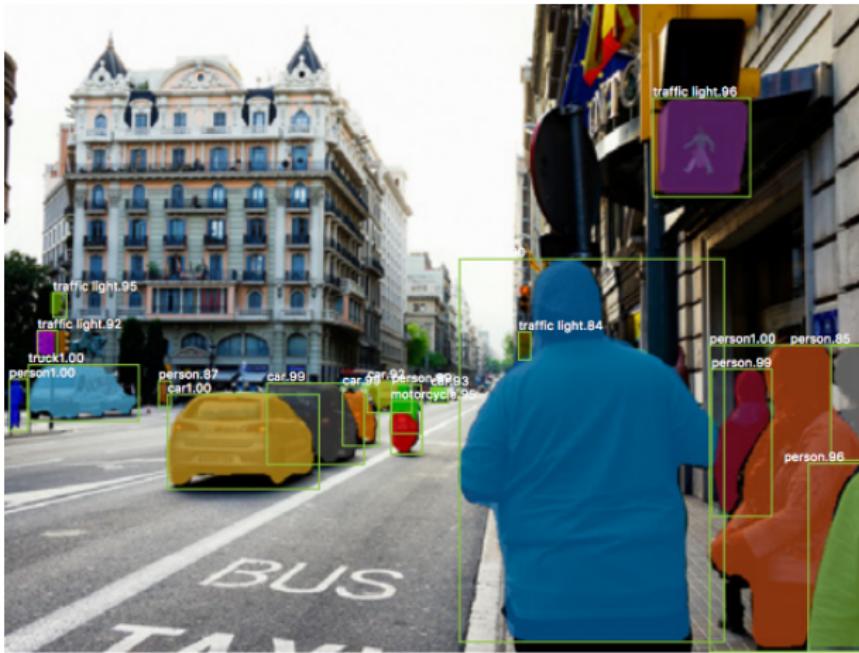


Credit: Alex Krizhevsky (2009)

**Image classification** tries to predict a single label for each image.

CIFAR-10 is a well-known dataset used for image classification. It consists of 60,000 32x32 color images containing one of 10 object classes, with 6000 images per class.

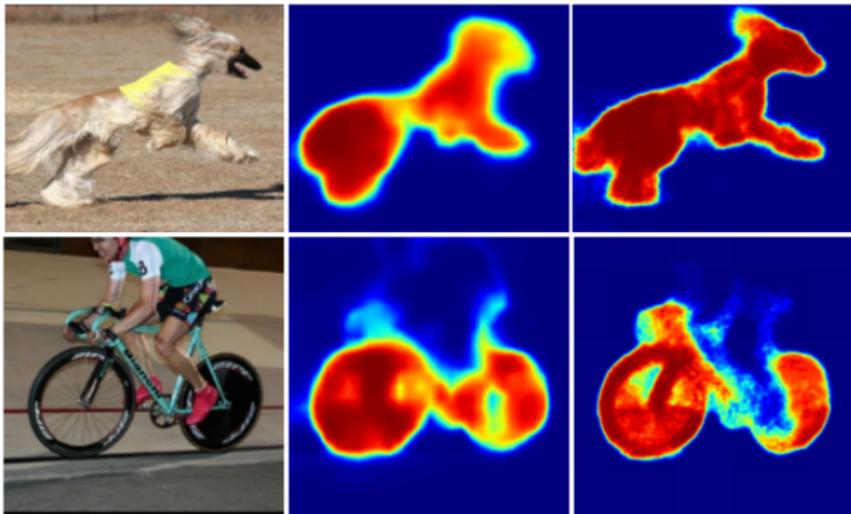
# POSSIBLE USE-CASE: IMAGES



Credit: Kaiming He (2017)

**Object Detection** Mask R-CNN is a general framework for instance segmentation, that efficiently detects objects in an image while simultaneously generating a high-quality segmentation mask for each instance.

# POSSIBLE USE-CASE: IMAGES



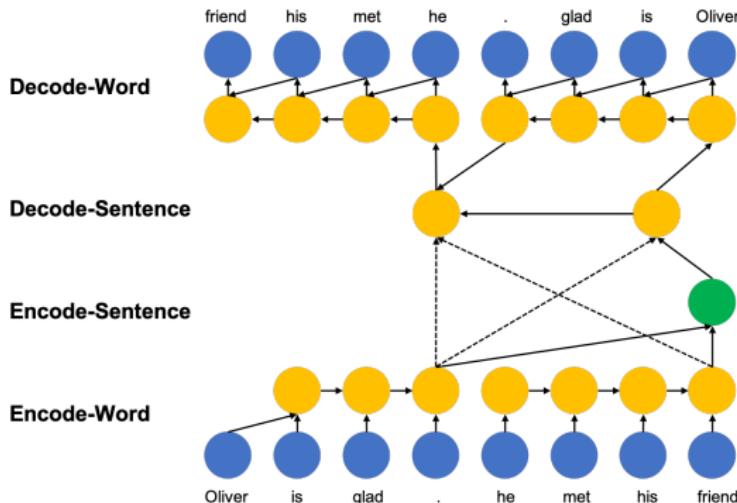
Credit: Hyeonwoo Noh (2015)

**Image segmentation** partitions the image into (multiple) segments.

# POSSIBLE USE-CASE: TEXT

- **High Dimensional:** Each word can be a single feature (300000 words in the German language).
- **Informative:** A single word does not provide much context.
- **Training Data:** Huge amounts of text data available.

Architecture: **Recurrent Neural Networks (RNN)**



# POSSIBLE USE-CASE: TEXT CLASSIFICATION



**Positive**



**Neutral**



**Negative**

Great job! Your customer support is fantastic.

Not bad, but it should be improved in the future.

The worst customer service I have ever seen.

**Sentiment Analysis** is the application of natural language processing to systematically identify the emotional and subjective information in texts.

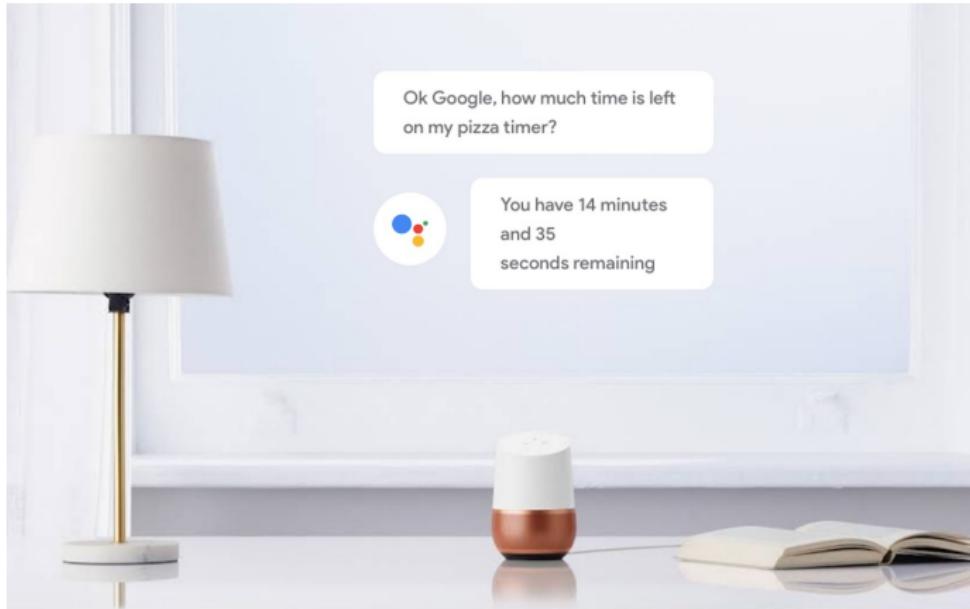
# POSSIBLE USE-CASE: TEXT

The image displays two separate instances of the Google Translate web interface. The top instance shows a translation from English ('He loves to eat') to German ('Er liebt es zu essen'). The bottom instance shows a translation from Norwegian ('Butikken er stengt') to English ('The store is closed'). Both interfaces include language selection dropdowns, microphone and speaker icons, and a copy/paste icon.

| From Language      | To Language | Text               | Edit                 |
|--------------------|-------------|--------------------|----------------------|
| English – detected | German      | He loves to eat    | Er liebt es zu essen |
| Norwegian          | English     | Butikken er stengt | The store is closed  |

**Machine Translation** (e.g. google translate) Neural machine translation exploits neural networks to predict the likelihood of a sequence of words, typically modeling entire sentences in a single integrated model.

# APPLICATIONS OF DEEP LEARNING: SPEECH

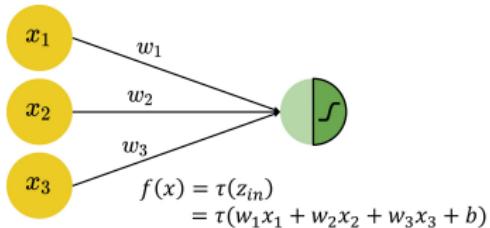


**Speech Recognition and Generation** (e.g. google assistant) Neural network extracts features from audio data for downstream tasks, e.g., to classify emotions in speech.

# Deep Learning

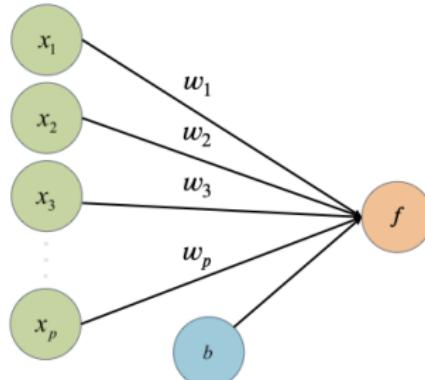
## Single Neuron / Perceptron

### Learning goals



- Graphical representation of a single neuron
- Affine transformations and non-linear activation functions
- Hypothesis spaces of a single neuron
- Typical loss functions

# A SINGLE NEURON



Perceptron with **input features**  $x_1, x_2, \dots, x_p$ , **weights**  $w_1, w_2, \dots, w_p$ , **bias term**  $b$ , and **activation function**  $\tau$ .

- The perceptron is a single artificial neuron and the basic computational unit of neural networks.
- It is a weighted sum of input values, transformed by  $\tau$ :

$$f(x) = \tau(w_1x_1 + \dots + w_px_p + b) = \tau(\mathbf{w}^T \mathbf{x} + b)$$

# A SINGLE NEURON

**Activation function**  $\tau$ : a single neuron represents different functions depending on the choice of activation function.

- The identity function gives us the simple **linear regression**:

$$f(x) = \tau(\mathbf{w}^T \mathbf{x}) = \mathbf{w}^T \mathbf{x}$$

- The logistic function gives us the **logistic regression**:

$$f(x) = \tau(\mathbf{w}^T \mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x})}$$

# A SINGLE NEURON

We consider a perceptron with 3-dimensional input, i.e.

$$f(\mathbf{x}) = \tau(w_1x_1 + w_2x_2 + w_3x_3 + b).$$

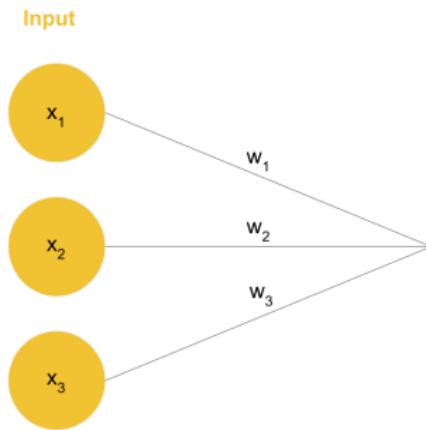
- Input features  $\mathbf{x}$  are represented by nodes in the “input layer”.



- In general, a  $p$ -dimensional input vector  $\mathbf{x}$  will be represented by  $p$  nodes in the input layer.

# A SINGLE NEURON

- Weights  $w$  are connected to edges from the input layer.



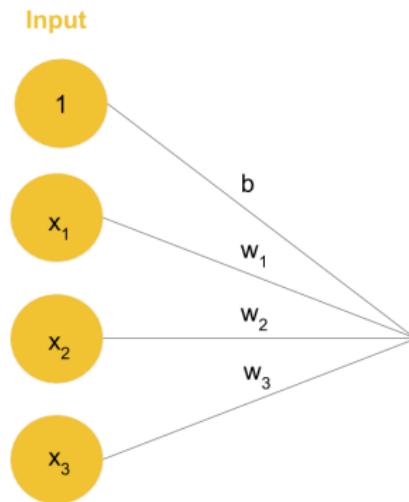
- The bias term  $b$  is implicit here. It is often not visualized as a separate node.

# A SINGLE NEURON

For an explicit graphical representation, we do a simple trick:

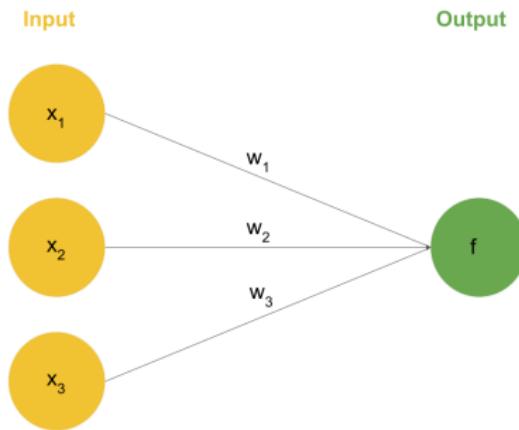
- Add a constant feature to the inputs  $\tilde{\mathbf{x}} = (1, x_1, \dots, x_p)^T$
- and absorb the bias into the weight vector  $\tilde{\mathbf{w}} = (b, w_1, \dots, w_p)$ .

The graphical representation is then:



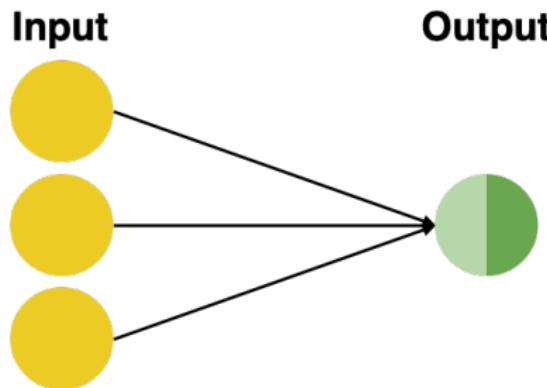
# A SINGLE NEURON

- The computation  $\tau(w_1x_1 + w_2x_2 + w_3x_3 + b)$  is represented by the neuron in the “output layer”.



# A SINGLE NEURON

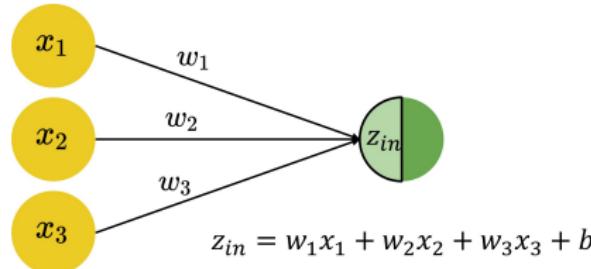
- You can picture the input vector being "fed" to neurons on the left followed by a sequence of computations performed from left to right. This is called a **forward pass**.



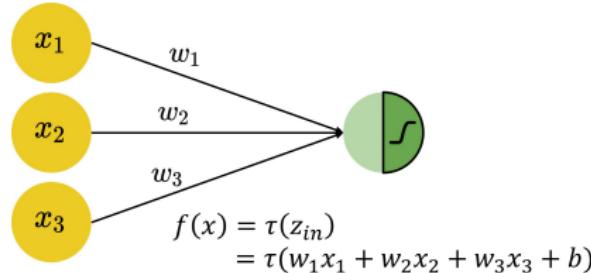
# A SINGLE NEURON

A neuron performs a 2-step computation:

- ❶ **Affine Transformation:** weighted sum of inputs plus bias.



- ❷ **Non-linear Activation:** a non-linear transformation applied to the weighted sum.

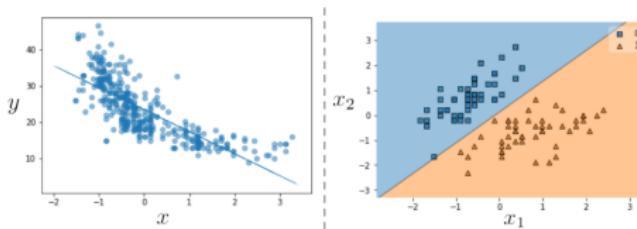


# A SINGLE NEURON: HYPOTHESIS SPACE

- The hypothesis space that is formed by single neuron is

$$\mathcal{H} = \left\{ f : \mathbb{R}^p \rightarrow \mathbb{R} \mid f(\mathbf{x}) = \tau \left( \sum_{j=1}^p w_j x_j + b \right), \mathbf{w} \in \mathbb{R}^p, b \in \mathbb{R} \right\}.$$

- If  $\tau$  is the logistic sigmoid or identity function,  $\mathcal{H}$  corresponds to the hypothesis space of logistic or linear regression, respectively.



**Figure:** Left: A regression line learned by a single neuron. Right: A decision-boundary learned by a single neuron in a binary classification task.

# A SINGLE NEURON: OPTIMIZATION

- To optimize this model, we minimize the empirical risk

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

where  $L(y, f(\mathbf{x}))$  is a loss function. It compares the network's predictions  $f(\mathbf{x})$  to the ground truth  $y$ .

- For regression, we typically use the L2 loss (rarely L1):

$$L(y, f(\mathbf{x})) = \frac{1}{2}(y - f(\mathbf{x}))^2$$

- For binary classification, we typically apply the cross entropy loss (also known as Bernoulli loss):

$$L(y, f(\mathbf{x})) = -(y \log f(\mathbf{x}) + (1 - y) \log(1 - f(\mathbf{x})))$$

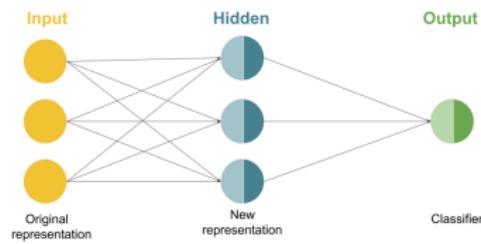
# A SINGLE NEURON: OPTIMIZATION

- For a single neuron and both choices of  $\tau$  the loss function is convex.
- The global optimum can be found with an iterative algorithm like gradient descent.
- A single neuron with logistic sigmoid function trained with the Bernoulli loss yields the same result as logistic regression when trained until convergence.
- Note: In the case of regression and the L2-loss, the solution can also be found analytically using the “normal equations”. However, in other cases a closed-form solution is usually not available.

# Deep Learning

## Single hidden layer neural networks

### Learning goals



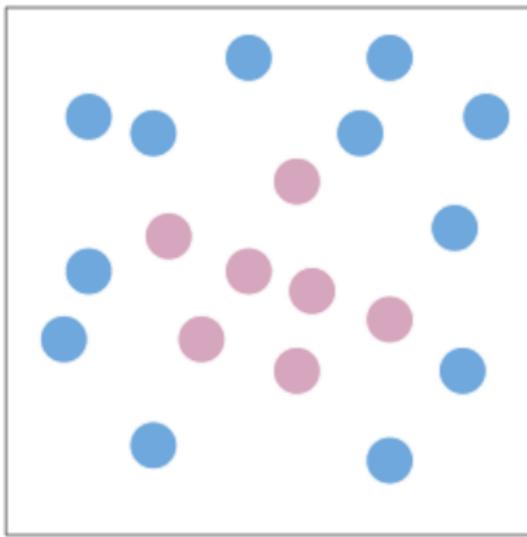
- Architecture of single hidden layer neural networks
- Representation learning/understanding the advantage of hidden layers
- Typical (non-linear) activation functions

# MOTIVATION

- The graphical way of representing simple functions/models, like logistic regression. Why is that useful?
- Because individual neurons can be used as building blocks of more complicated functions.
- Networks of neurons can represent extremely complex hypothesis spaces.
- Most importantly, it allows us to define the “right” kinds of hypothesis spaces to learn functions that are common in our universe in a data-efficient way (see Lin, Tegmark et al. 2016).

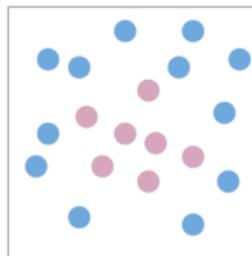
# MOTIVATION

Can a single neuron perform binary classification of these points?

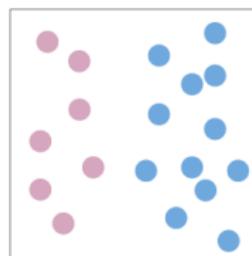


# MOTIVATION

- As a single neuron is restricted to learning only linear decision boundaries, its performance on the following task is quite poor:

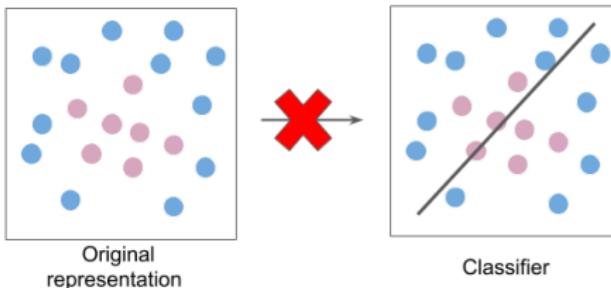


- However, the neuron can easily separate the classes if the original features are transformed (e.g., from Cartesian to polar coordinates):



# MOTIVATION

- Instead of classifying the data in the original representation,

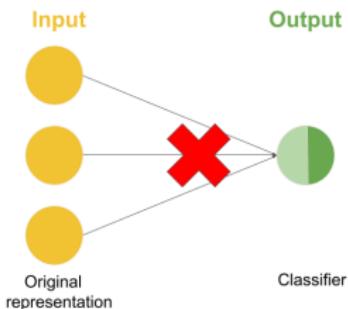


- we classify it in a new feature space.

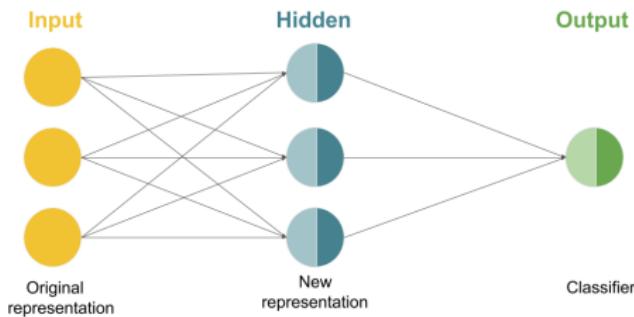


# MOTIVATION

- Analogously, instead of a single neuron,



- we use more complex networks.



# REPRESENTATION LEARNING

- It is *very* critical to feed a classifier the “right” features in order for it to perform well.
- Before deep learning took off, features for tasks like machine vision and speech recognition were “hand-designed” by domain experts. This step of the machine learning pipeline is called **feature engineering**.
- DL automates feature engineering. This is called **representation learning**.

# SINGLE HIDDEN LAYER NETWORKS

**Single neurons** perform a 2-step computation:

- ① **Affine Transformation**: a weighted sum of inputs plus bias.
- ② **Activation**: a non-linear transformation on the weighted sum.

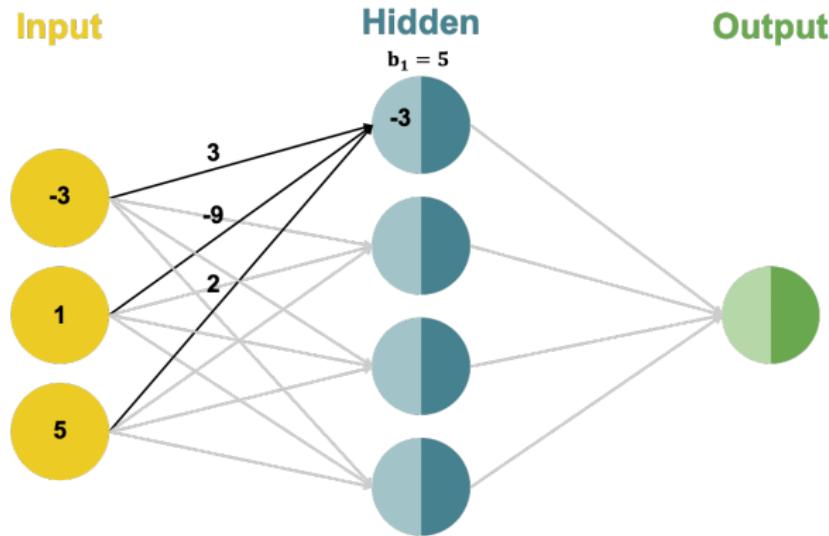
**Single hidden layer networks** consist of two layers (without input layer):

- ① **Hidden Layer**: having a set of neurons.
- ② **Output Layer**: having one or more output neurons.

- Multiple inputs are simultaneously fed to the network.
- Each neuron in the hidden layer performs a 2-step computation.
- The final output of the network is then calculated by another 2-step computation performed by the neuron in the output layer.

# SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

Each neuron in the hidden layer performs an **affine transformation** on the inputs:

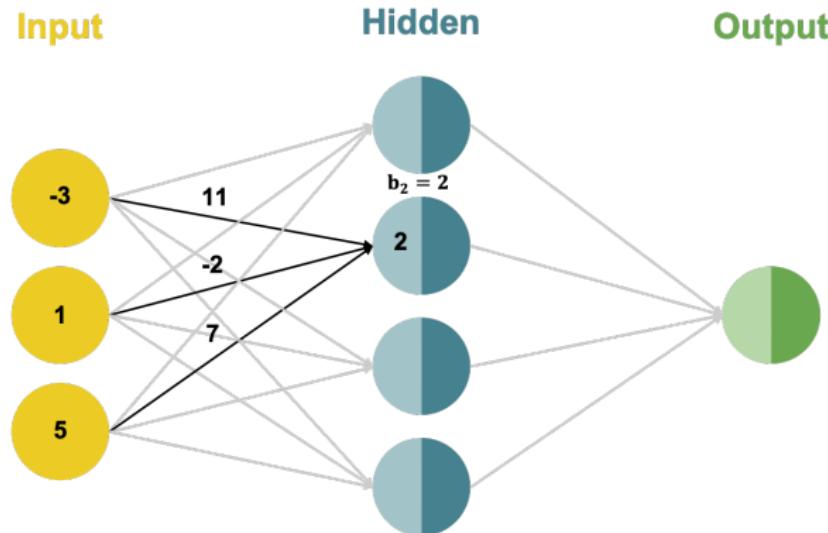


$$z_{in}^{(1)} = w_{11}x^{(1)} + w_{21}x^{(2)} + w_{31}x^{(3)} + b_1$$

$$z_{in}^{(1)} = 3 * (-3) + (-9) * 1 + 2 * 5 + 5 = -3$$

# SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

Each neuron in the hidden layer performs an **affine transformation** on the inputs:

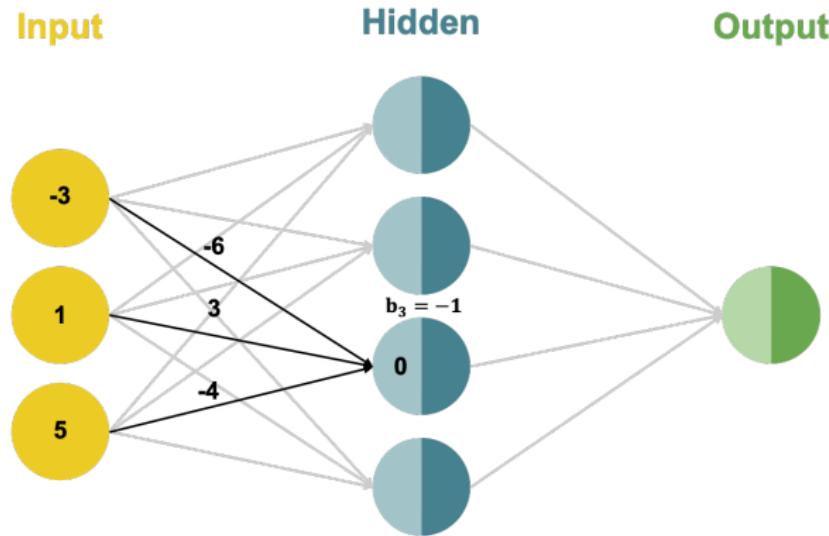


$$z_{\text{in}}^{(2)} = w_{12}x^{(1)} + w_{22}x^{(2)} + w_{32}x^{(3)} + b_2$$

$$z_{\text{in}}^{(2)} = 11 * (-3) + (-2) * 1 + 7 * 5 + 2 = 2$$

# SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

Each neuron in the hidden layer performs an **affine transformation** on the inputs:

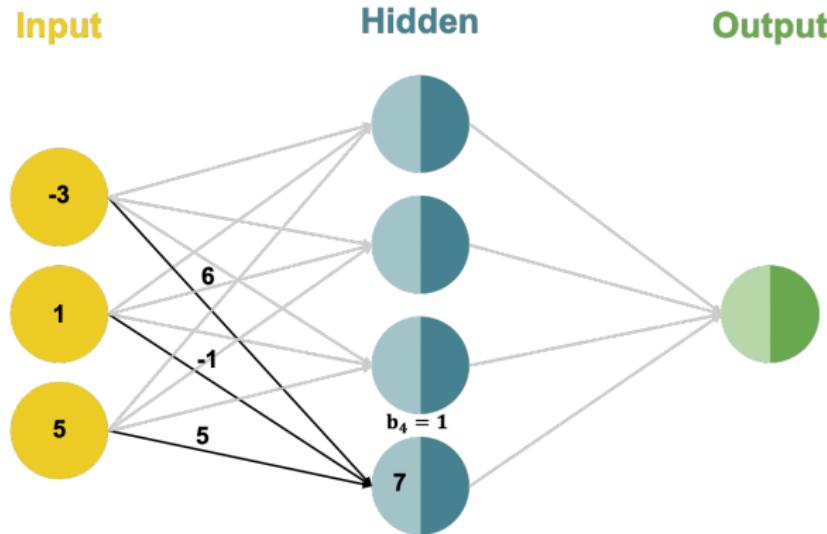


$$z_{in}^{(3)} = w_{13}x^{(1)} + w_{23}x^{(2)} + w_{33}x^{(3)} + b_3$$

$$z_{in}^{(3)} = (-6) * (-3) + 3 * 1 + (-4) * 5 - 1 = 0$$

# SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

Each neuron in the hidden layer performs an **affine transformation** on the inputs:

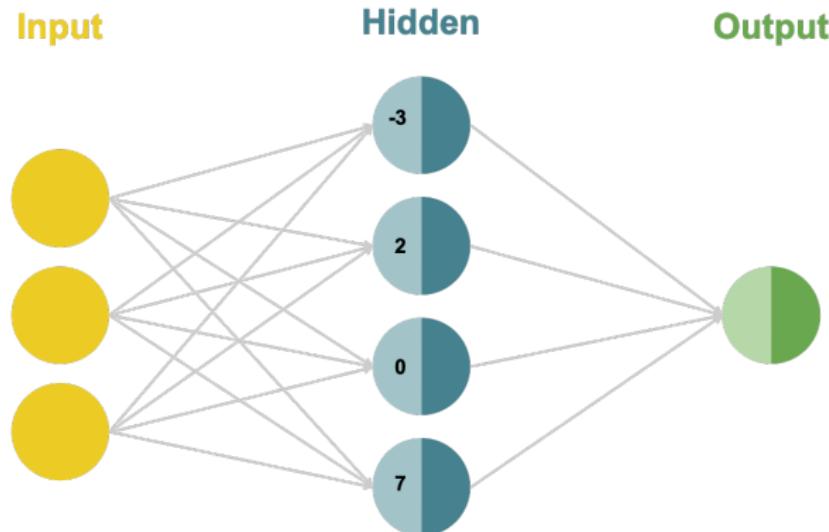


$$z_{in}^{(4)} = w_{14}x^{(1)} + w_{24}x^{(2)} + w_{34}x^{(3)} + b_4$$

$$z_{in}^{(4)} = 6 * (-3) + (-1) * 1 + 5 * 5 + 1 = 7$$

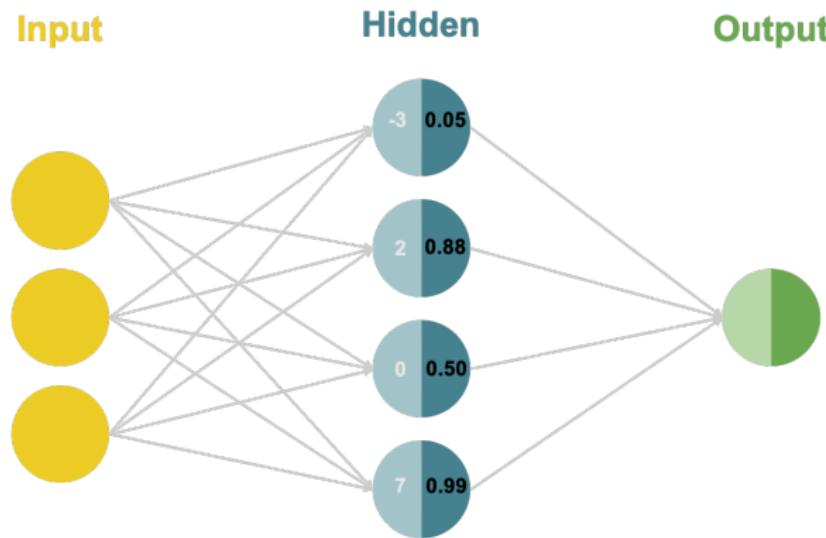
# SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

Each neuron in the hidden layer performs an **affine transformation** on the inputs:



# SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

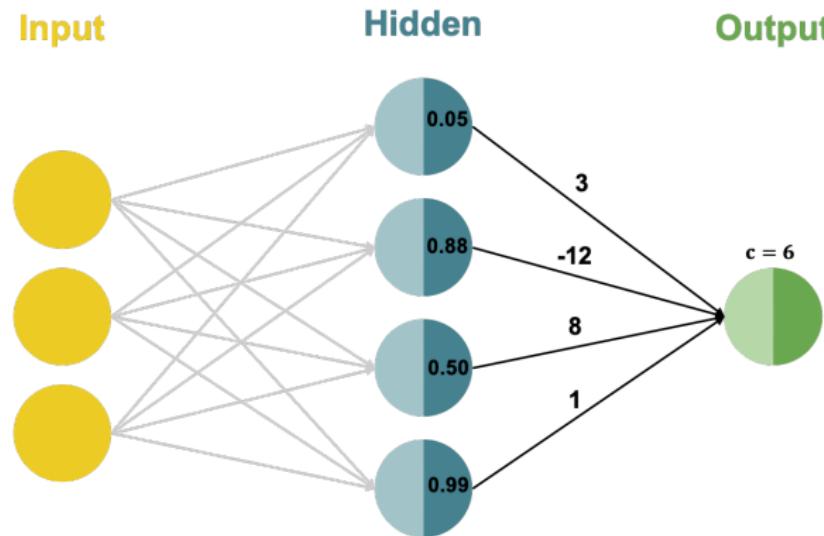
Each hidden neuron performs a non-linear **activation** transformation on the weight sum:



$$z_{\text{out}}^{(i)} = \sigma(z_{\text{in}}^{(i)}) = \frac{1}{1+e^{-z_{\text{in}}^{(i)}}}$$

# SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

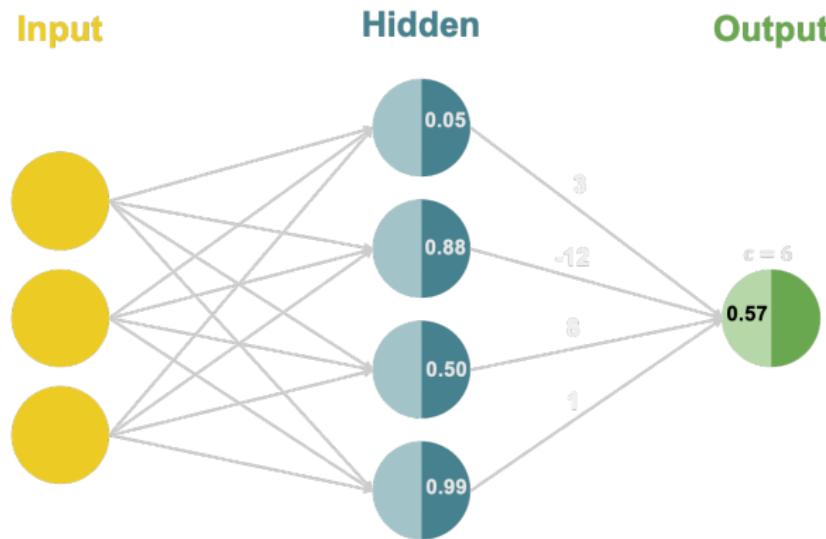
The output neuron performs an **affine transformation** on its inputs:



$$f_{\text{in}} = u_1 z_{\text{out}}^{(1)} + u_2 z_{\text{out}}^{(2)} + u_3 z_{\text{out}}^{(3)} + u_4 z_{\text{out}}^{(4)} + c$$

# SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

The output neuron performs an **affine transformation** on its inputs:

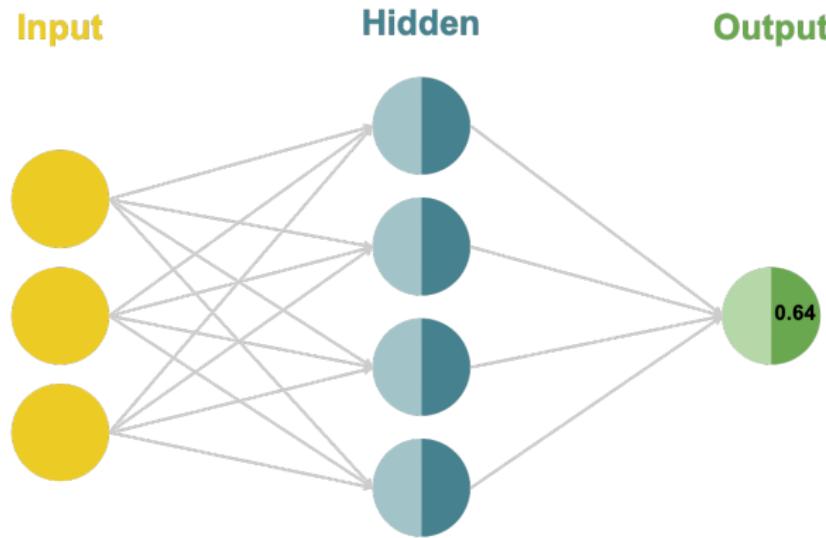


$$f_{in} = u_1 z_{out}^{(1)} + u_2 z_{out}^{(2)} + u_3 z_{out}^{(3)} + u_4 z_{out}^{(4)} + c$$

$$f_{in} = 3 * 0.05 + (-12) * 0.88 + 8 * 0.50 + 1 * 0.99 + 6 = 0.57$$

# SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

The output neuron performs a non-linear **activation** transformation on the weight sum:



$$f_{\text{out}} = \sigma(f_{\text{in}}) = \frac{1}{1+e^{f_{\text{in}}}}$$

$$f_{\text{out}} = \frac{1}{1+e^{0.57}} = 0.64$$

## HIDDEN LAYER: ACTIVATION FUNCTION

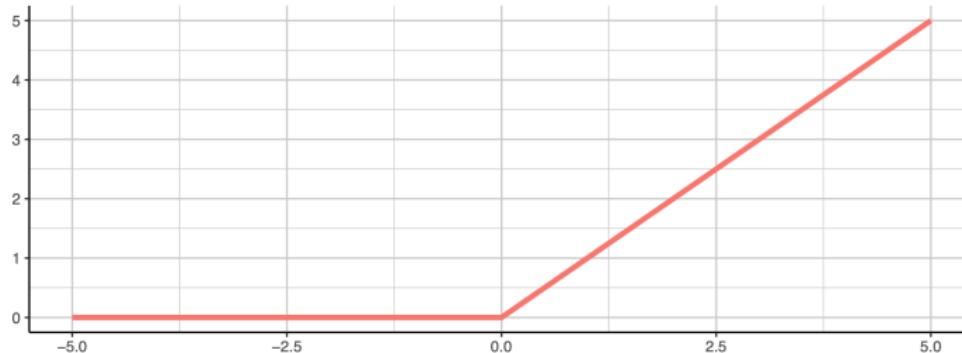
- If the hidden layer does not have a non-linear activation, the network can only learn linear decision boundaries.
- A lot of different activation functions exist.

# HIDDEN LAYER: ACTIVATION FUNCTION

## ReLU Activation:

- Currently the most popular choice is the ReLU (rectified linear unit):

$$\sigma(v) = \max(0, v)$$

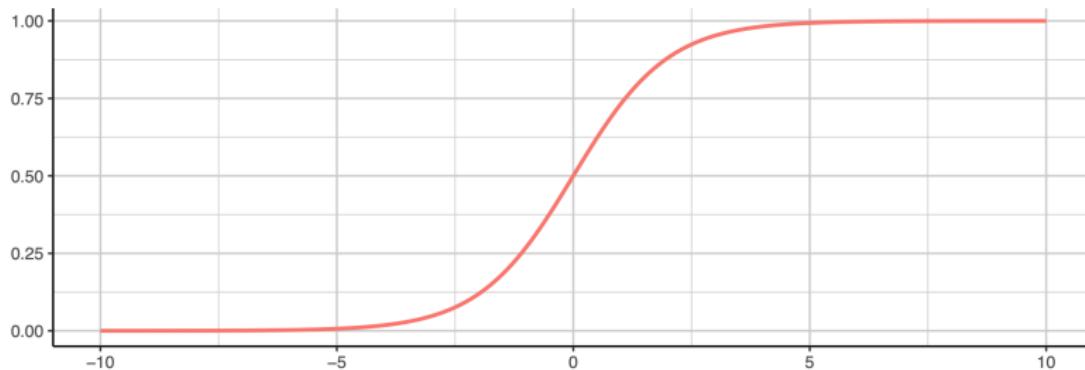


# HIDDEN LAYER: ACTIVATION FUNCTION

## Sigmoid Activation Function:

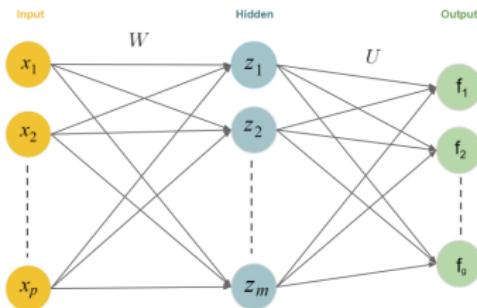
- The sigmoid function can be used even in the hidden layer:

$$\sigma(v) = \frac{1}{1 + \exp(-v)}$$



# Deep Learning

## Single Hidden Layer Networks for Multi-Class Classification



### Learning goals

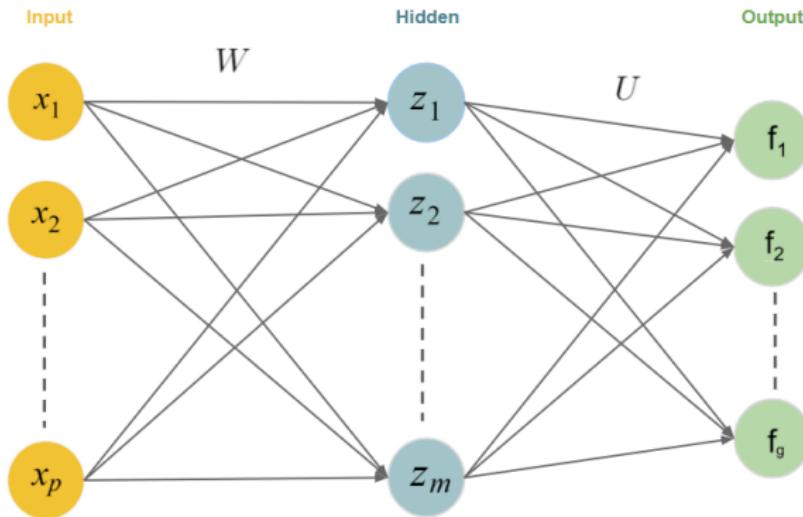
- Neural network architectures for multi-class classification
- Softmax activation function
- Softmax loss

# MULTI-CLASS CLASSIFICATION

- We have only considered regression and binary classification problems so far.
- How can we get a neural network to perform multiclass classification?

# MULTI-CLASS CLASSIFICATION

- The first step is to add additional neurons to the output layer.
- Each neuron in the layer will represent a specific class (number of neurons in the output layer = number of classes).



**Figure:** Structure of a single hidden layer, feed-forward neural network for  $g$ -class classification problems (bias term omitted).

# MULTI-CLASS CLASSIFICATION

## Notation:

- For  $g$ -class classification,  $g$  output units:

$$\mathbf{f} = (f_1, \dots, f_g)$$

- $m$  hidden neurons  $z_1, \dots, z_m$ , with

$$z_j = \sigma(\mathbf{W}_j^T \mathbf{x}), \quad j = 1, \dots, m.$$

- Compute linear combinations of derived features  $z$ :

$$f_{in,k} = \mathbf{U}_k^T \mathbf{z}, \quad \mathbf{z} = (z_1, \dots, z_m)^T, \quad k = 1, \dots, g$$

# MULTI-CLASS CLASSIFICATION

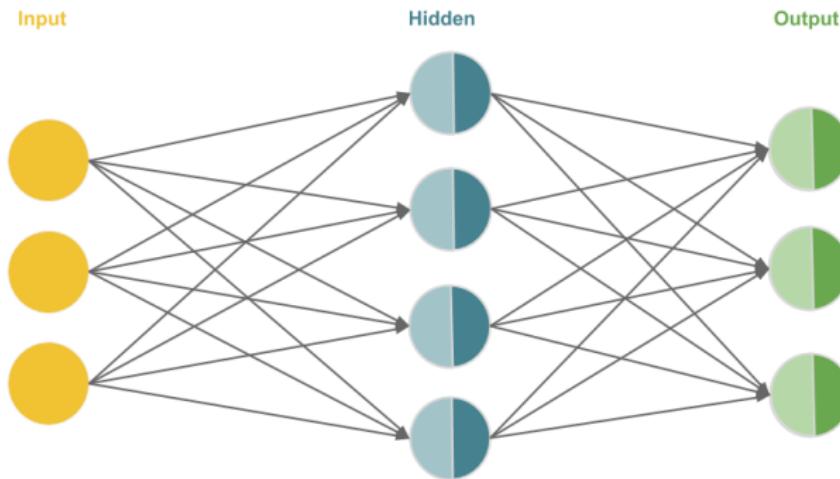
- The second step is to apply a **softmax** activation function to the output layer.
- This gives us a probability distribution over  $g$  different possible classes:

$$f_{out,k} = \tau_k(f_{in,k}) = \frac{\exp(f_{in,k})}{\sum_{k'=1}^g \exp(f_{in,k'})}$$

- This is the same transformation used in softmax regression!
- Derivative  $\frac{\partial \tau(\mathbf{f}_{in})}{\partial \mathbf{f}_{in}} = \text{diag}(\tau(\mathbf{f}_{in})) - \tau(\mathbf{f}_{in})\tau(\mathbf{f}_{in})^T$
- It is a “smooth” approximation of the argmax operation, so  $\tau((1, 1000, 2)^T) \approx (0, 1, 0)^T$  (picks out 2nd element!).

# MULTI-CLASS CLASSIFICATION: EXAMPLE

Forward pass (Hidden: Sigmoid, Output: Softmax).



$$\begin{pmatrix} 3 & -9 & 2 \\ 11 & -2 & 7 \\ -6 & 3 & -4 \\ 6 & -1 & 5 \end{pmatrix} \begin{pmatrix} 5 \\ 2 \\ -1 \\ 1 \end{pmatrix}$$

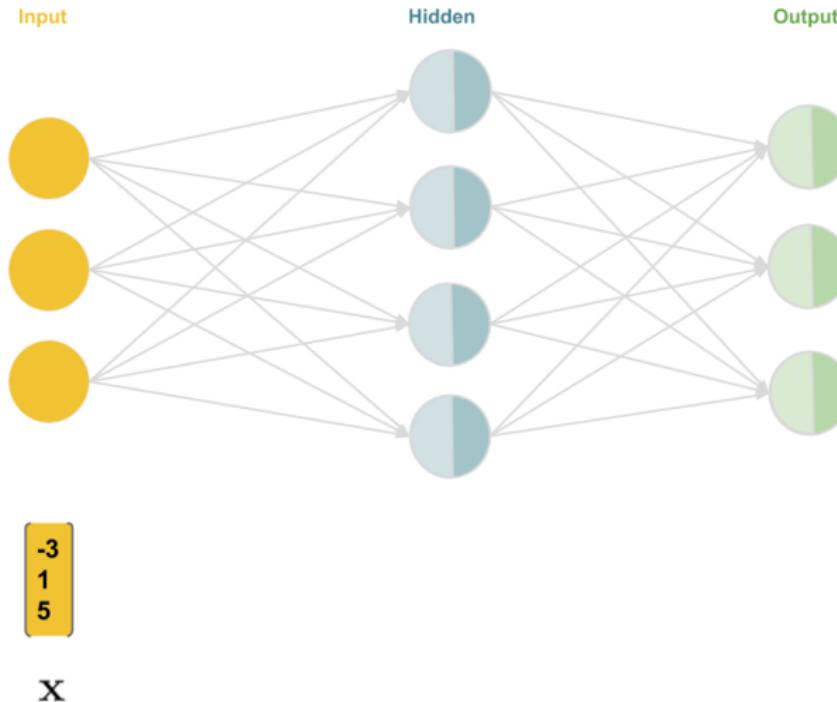
$$W^T \quad b$$

$$\begin{pmatrix} 3 & -12 & 8 & 1 \\ 2 & -3 & 9 & 1 \\ -5 & 1 & -1 & 7 \end{pmatrix} \begin{pmatrix} 6 \\ 0 \\ -8 \end{pmatrix}$$

$$U^T \quad c$$

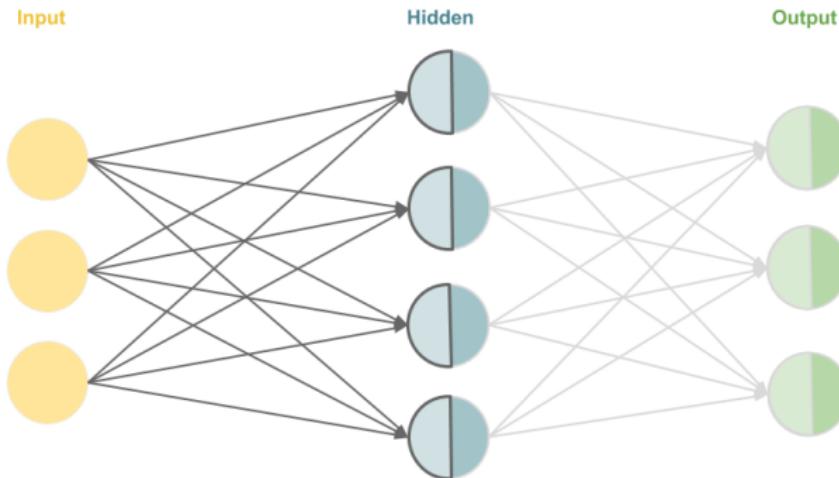
# MULTI-CLASS CLASSIFICATION: EXAMPLE

Forward pass (Hidden: Sigmoid, Output: Softmax).



# MULTI-CLASS CLASSIFICATION: EXAMPLE

Forward pass (Hidden: Sigmoid, Output: Softmax).

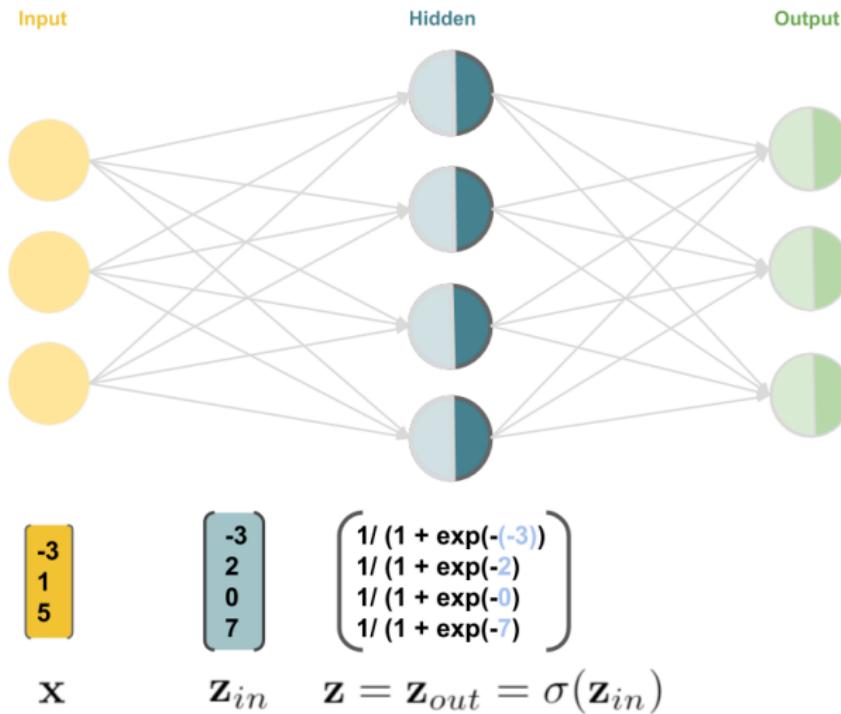


$$\begin{bmatrix} -3 \\ 1 \\ 5 \end{bmatrix} \quad \left. \begin{array}{l} (-3)^*3 + 1^*(-9) + 5^*2 + 5 \\ (-3)^*11 + 1^*(-2) + 5^*7 + 2 \\ (-3)^*(-6) + 1^*3 + 5^*(-4) + (-1) \\ (-3)^*6 + 1^*(-1) + 5^*5 + 1 \end{array} \right\}$$

$$\mathbf{x} \quad \mathbf{z}_{in} = \mathbf{w}^T \mathbf{x} + \mathbf{b}$$

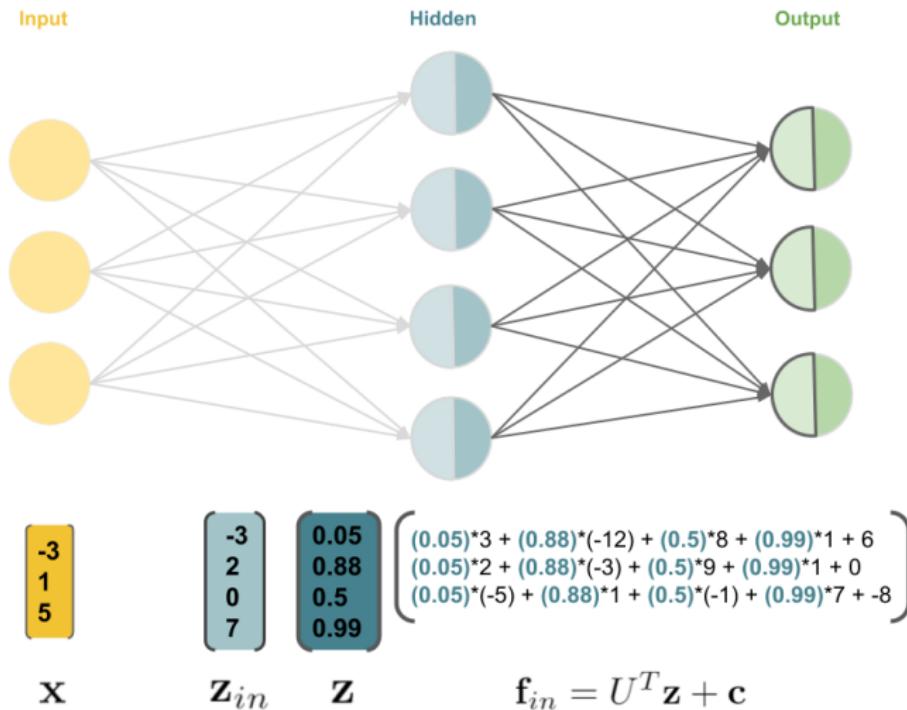
# MULTI-CLASS CLASSIFICATION: EXAMPLE

Forward pass (Hidden: Sigmoid, Output: Softmax).



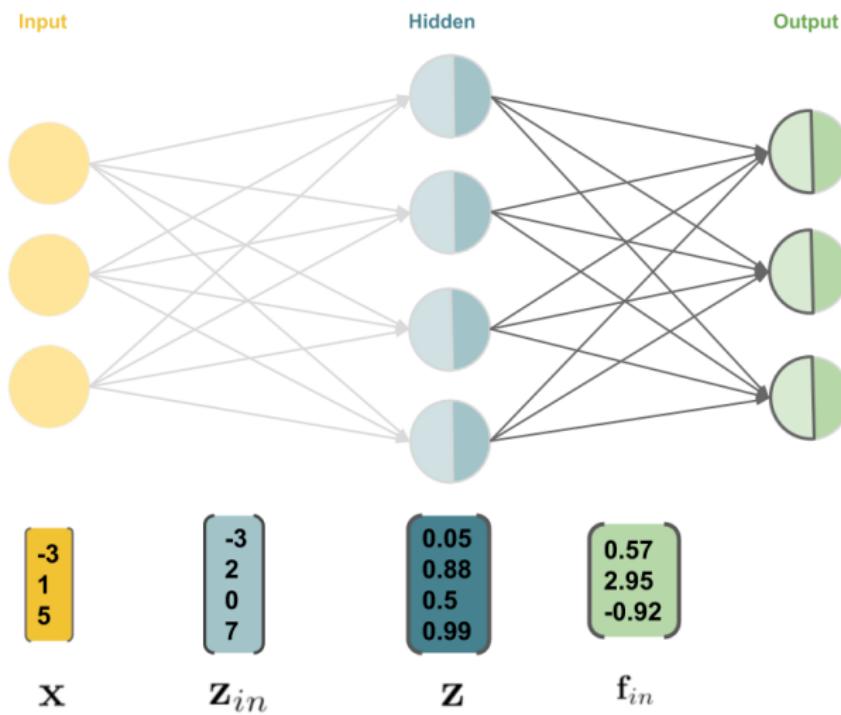
# MULTI-CLASS CLASSIFICATION: EXAMPLE

Forward pass (Hidden: Sigmoid, Output: Softmax).



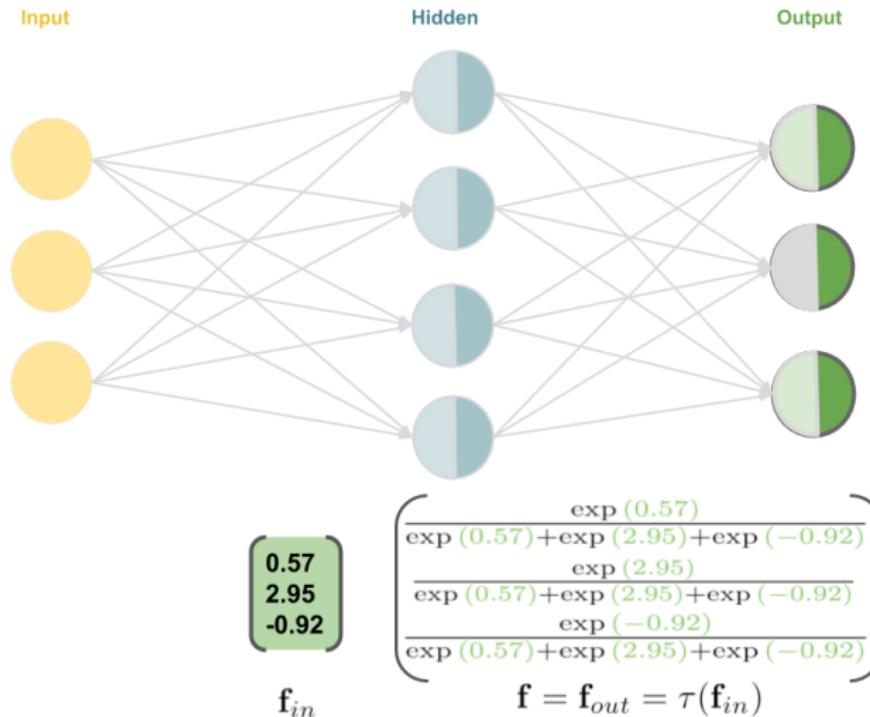
# MULTI-CLASS CLASSIFICATION: EXAMPLE

Forward pass (Hidden: Sigmoid, Output: Softmax).



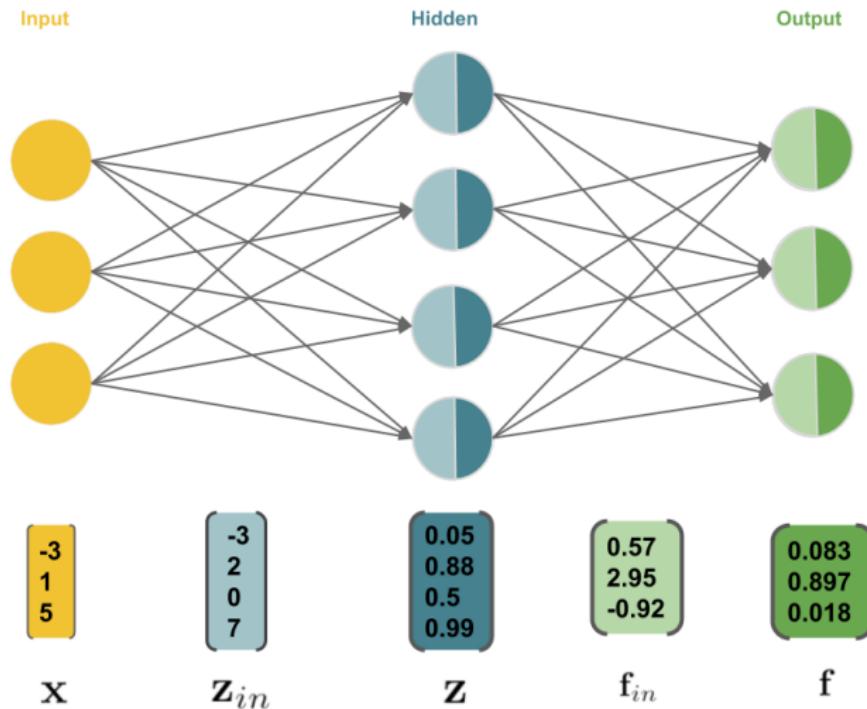
# MULTI-CLASS CLASSIFICATION: EXAMPLE

Forward pass (Hidden: Sigmoid, Output: Softmax).



# MULTI-CLASS CLASSIFICATION: EXAMPLE

Forward pass (Hidden: Sigmoid, Output: Softmax).



# OPTIMIZATION: SOFTMAX LOSS

- The loss function for a softmax classifier is

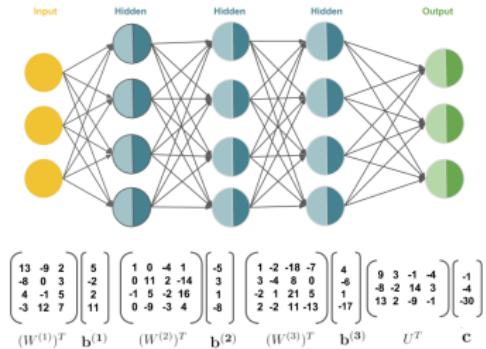
$$L(y, f(\mathbf{x})) = - \sum_{k=1}^g [y = k] \log \left( \frac{\exp(f_{in,k})}{\sum_{k'=1}^g \exp(f_{in,k'})} \right)$$

where  $[y = k] = \begin{cases} 1 & \text{if } y = k \\ 0 & \text{otherwise} \end{cases}$ .

- This is equivalent to the cross-entropy loss when the label vector  $\mathbf{y}$  is one-hot coded (e.g.  $\mathbf{y} = (0, 0, 1, 0)^T$ ).
- Optimization: Again, there is no analytic solution.

# Deep Learning

## MLP – Multi-Layer Feedforward Neural Networks



### Learning goals

- Architectures of deep neural networks
- Deep neural networks as chained functions

# FEEDFORWARD NEURAL NETWORKS

- We will now extend the model class once again, such that we allow an arbitrary amount / of hidden layers.
- The general term for this model class is (multi-layer) **feedforward networks** (inputs are passed through the network from left to right, no feedback-loops are allowed)

# FEEDFORWARD NEURAL NETWORKS

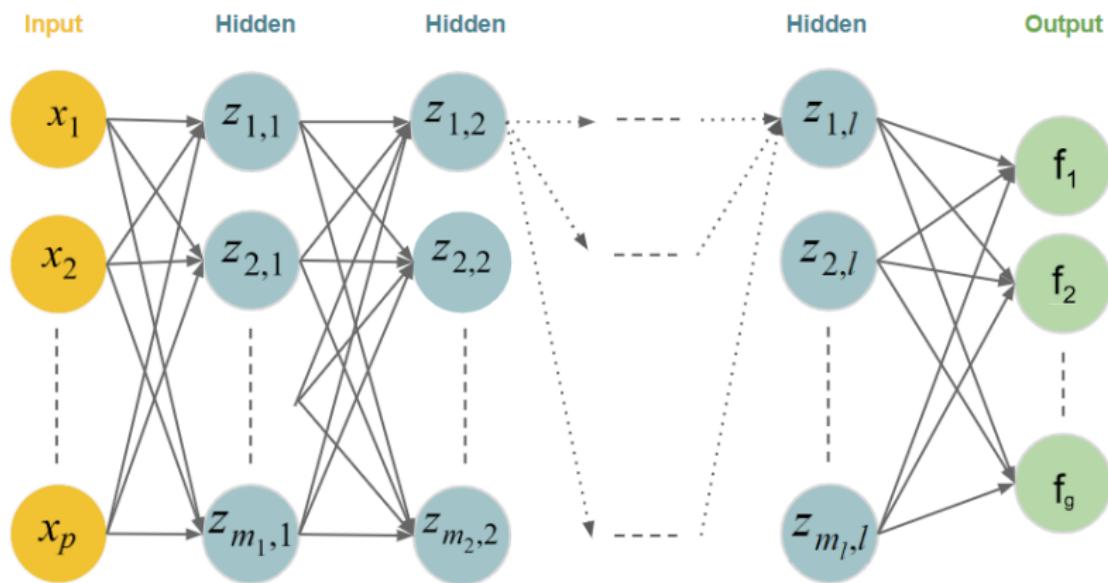
- We can characterize those models by the following chain structure:

$$f(\mathbf{x}) = \tau \circ \phi \circ \sigma^{(I)} \circ \phi^{(I)} \circ \sigma^{(I-1)} \circ \phi^{(I-1)} \circ \dots \circ \sigma^{(1)} \circ \phi^{(1)}$$

where  $\sigma^{(i)}$  and  $\phi^{(i)}$  are the activation function and the weighted sum of hidden layer  $i$ , respectively.  $\tau$  and  $\phi$  are the corresponding components of the output layer.

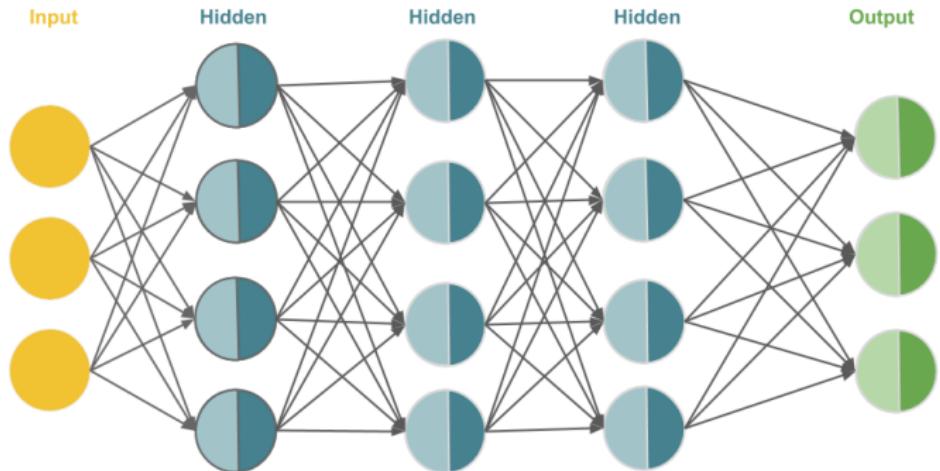
- Each hidden layer has:
  - an associated weight matrix  $\mathbf{W}^{(i)}$ , bias  $\mathbf{b}^{(i)}$ , and activations  $\mathbf{z}^{(i)}$  for  $i \in \{1 \dots I\}$ .
  - $\mathbf{z}^{(i)} = \sigma^{(i)}(\phi^{(i)}) = \sigma^{(i)}(\mathbf{W}^{(i)T} \mathbf{z}^{(i-1)} + \mathbf{b}^{(i)})$ , where  $\mathbf{z}^{(0)} = \mathbf{x}$ .
- Again, without non-linear activations in the hidden layers, the network can only learn linear decision boundaries.

# FEEDFORWARD NEURAL NETWORKS



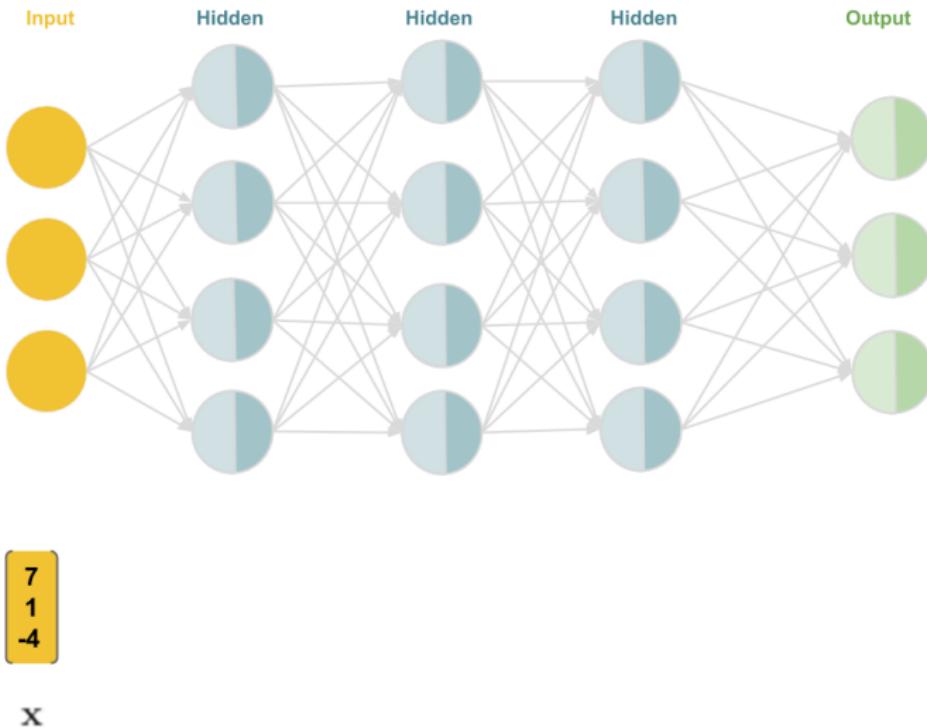
**Figure:** Structure of a deep neural network with  $l$  hidden layers (bias terms omitted).

# FEEDFORWARD NEURAL NETWORKS: EXAMPLE

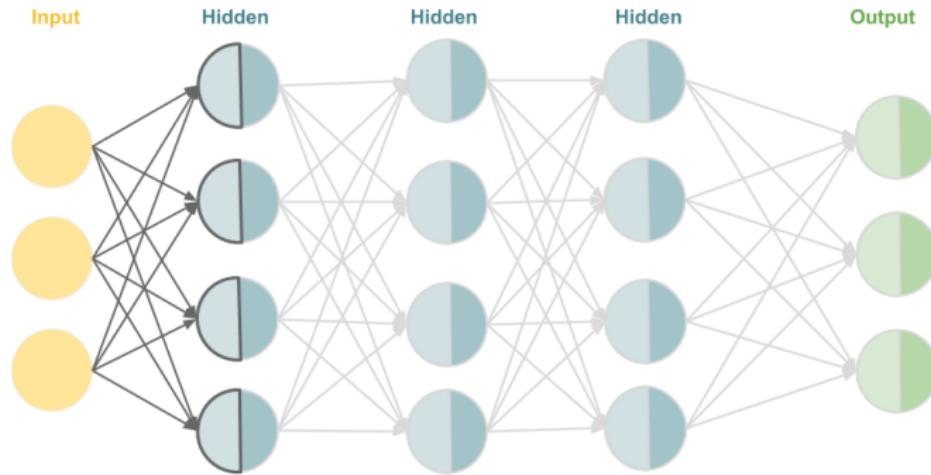


$$\begin{pmatrix} 13 & -9 & 2 \\ -8 & 0 & 3 \\ 4 & -1 & 5 \\ -3 & 12 & 7 \end{pmatrix} \begin{pmatrix} 5 \\ -2 \\ 2 \\ 11 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 & -4 & 1 \\ 0 & 11 & 2 & -14 \\ -1 & 5 & -2 & 16 \\ 0 & -9 & -3 & 4 \end{pmatrix} \begin{pmatrix} -5 \\ 3 \\ 1 \\ -8 \end{pmatrix} \quad \begin{pmatrix} 1 & -2 & -18 & -7 \\ 3 & -4 & 8 & 0 \\ -2 & 1 & 21 & 5 \\ 2 & -2 & 11 & -13 \end{pmatrix} \begin{pmatrix} 4 \\ -6 \\ 1 \\ -17 \end{pmatrix} \quad \begin{pmatrix} 9 & 3 & -1 & -4 \\ -8 & -2 & 14 & 3 \\ 13 & 2 & -9 & -1 \end{pmatrix} \begin{pmatrix} -1 \\ -4 \\ -30 \end{pmatrix}$$
$$(W^{(1)})^T \quad \mathbf{b}^{(1)} \quad (W^{(2)})^T \quad \mathbf{b}^{(2)} \quad (W^{(3)})^T \quad \mathbf{b}^{(3)} \quad U^T \quad \mathbf{c}$$

# FEEDFORWARD NEURAL NETWORKS: EXAMPLE



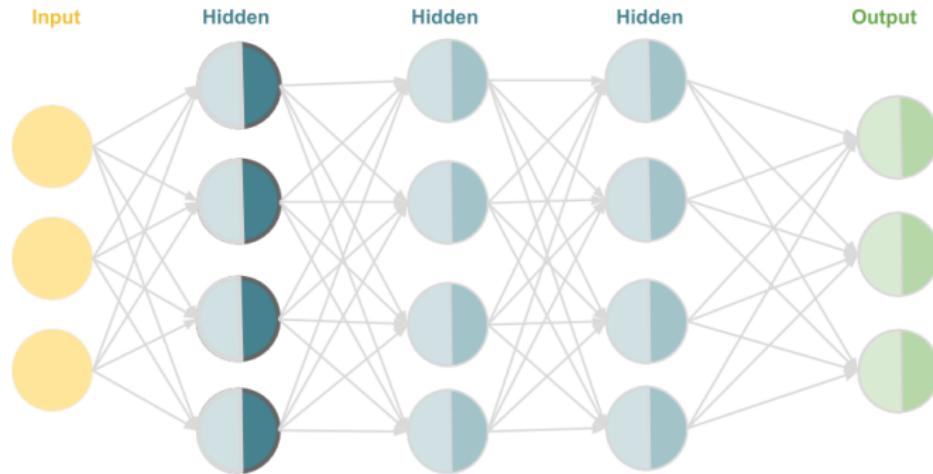
# FEEDFORWARD NEURAL NETWORKS: EXAMPLE



$$\begin{bmatrix} 7 \\ 1 \\ -4 \end{bmatrix} \left[ \begin{array}{l} 7*13 + 1*(-9) + (-4)*2 + 5 \\ 7*(-8) + 1*0 + (-4)*3 + (-2) \\ 7*4 + 1*(-1) + (-4)*5 + 2 \\ 7*(-3) + 1*12 + (-4)*7 + 11 \end{array} \right]$$

$$\mathbf{x} \quad \mathbf{z}_{in}^{(1)} = W^{(1)T} \mathbf{x} + \mathbf{b}^{(1)}$$

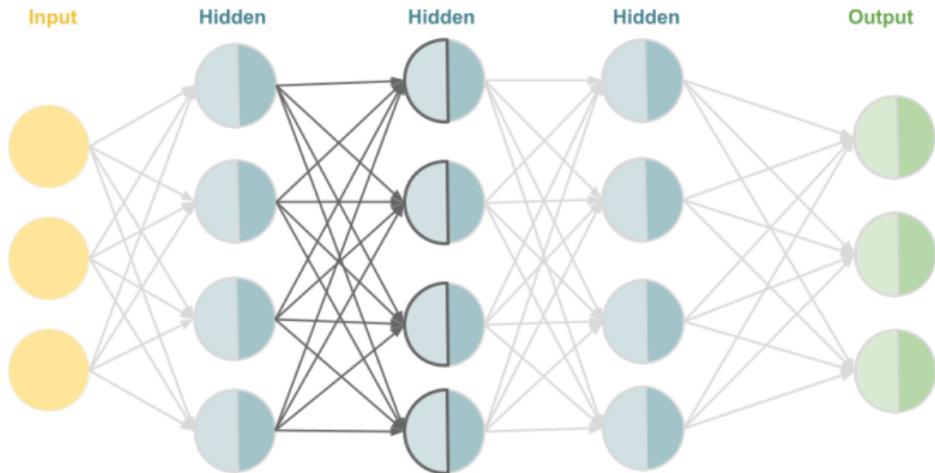
# FEEDFORWARD NEURAL NETWORKS: EXAMPLE



$$\mathbf{x} \quad \mathbf{z}_{in}^{(1)} \quad \mathbf{z}^{(1)} = \mathbf{z}_{out}^{(1)} = \sigma(\mathbf{z}_{in}^{(1)})$$

$\begin{bmatrix} 7 \\ 1 \\ -4 \end{bmatrix}$     $\begin{bmatrix} 79 \\ -70 \\ 9 \\ -26 \end{bmatrix}$     $\begin{bmatrix} \max(0, 79) \\ \max(0, -70) \\ \max(0, 9) \\ \max(0, -26) \end{bmatrix}$

# FEEDFORWARD NEURAL NETWORKS: EXAMPLE

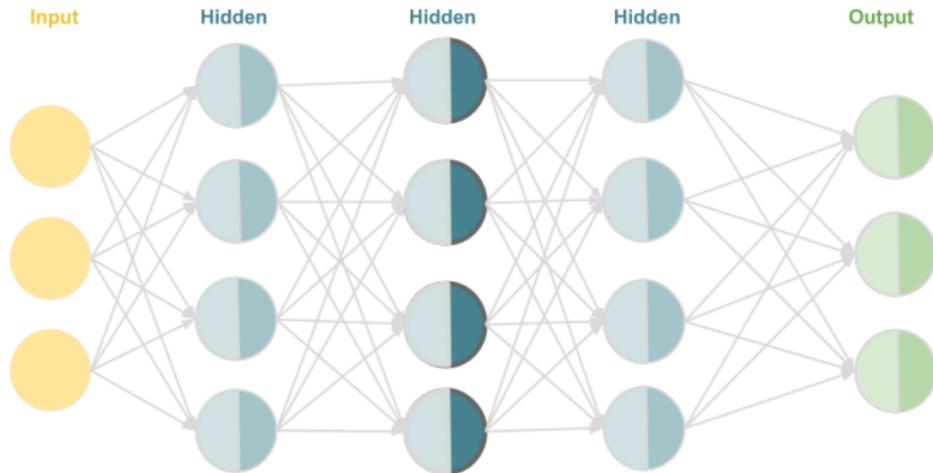


$$\mathbf{x} \quad \mathbf{z}_{in}^{(1)} \quad \mathbf{z}^{(1)} \quad \mathbf{z}_{in}^{(2)} = W^{(2)T} \mathbf{z}^{(1)} + \mathbf{b}^{(2)}$$

Below the input vector  $\mathbf{x}$  and the first hidden state  $\mathbf{z}_{in}^{(1)}$ , there are four boxes representing the initial values for the first hidden layer calculation:

|    |     |    |   |
|----|-----|----|---|
| 7  | 79  | 79 | $79 * 1 + 0 * 0 + 9 * (-4) + 0 * 1 + (-5)$    |
| 1  | -70 | 0  | $79 * 0 + 0 * 11 + 9 * 2 + 0 * (-14) + 3$     |
| -4 | 9   | 9  | $79 * (-1) + 0 * 5 + 9 * (-2) + 0 * 16 + 1$   |
|    | -26 | 0  | $79 * 0 + 0 * (-9) + 9 * (-3) + 0 * 4 + (-8)$ |

# FEEDFORWARD NEURAL NETWORKS: EXAMPLE

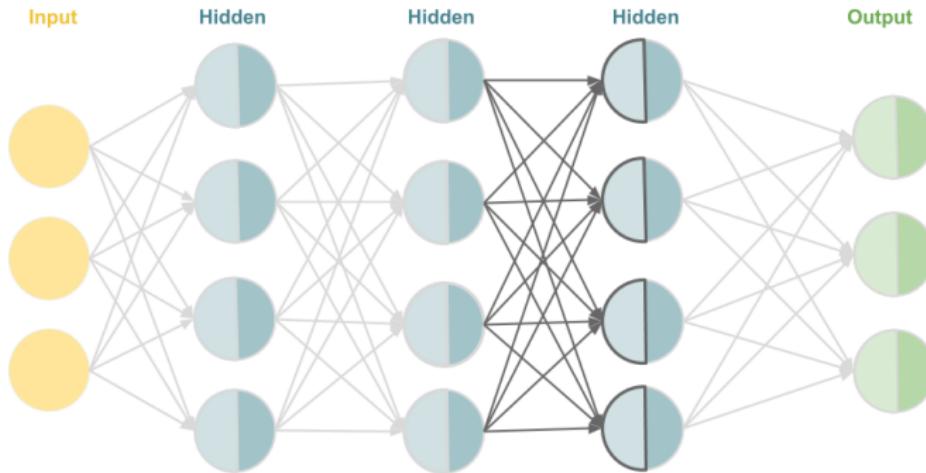


$$\begin{matrix} \mathbf{x} & \mathbf{z}_{in}^{(1)} & \mathbf{z}^{(1)} & \mathbf{z}_{in}^{(2)} & \mathbf{z}^{(2)} = \mathbf{z}_{out}^{(2)} = \sigma(\mathbf{z}_{in}^{(2)}) \end{matrix}$$

Below the input vector  $\mathbf{x}$ , the first hidden layer's input  $\mathbf{z}_{in}^{(1)}$ , the first hidden layer's output  $\mathbf{z}^{(1)}$ , the second hidden layer's input  $\mathbf{z}_{in}^{(2)}$ , and the second hidden layer's output  $\mathbf{z}^{(2)}$  (which is also the output  $\mathbf{z}_{out}^{(2)}$ ) are shown. The values in  $\mathbf{z}^{(1)}$  and  $\mathbf{z}^{(2)}$  are the results of applying the max operation to the corresponding values in  $\mathbf{z}_{in}^{(2)}$ .

$$\begin{aligned} \mathbf{z}^{(1)} &= \begin{pmatrix} 79 \\ -70 \\ 9 \\ -26 \end{pmatrix} \\ \mathbf{z}^{(2)} &= \begin{pmatrix} \max(0, 38) \\ \max(0, 21) \\ \max(0, -96) \\ \max(0, -36) \end{pmatrix} \end{aligned}$$

# FEEDFORWARD NEURAL NETWORKS: EXAMPLE

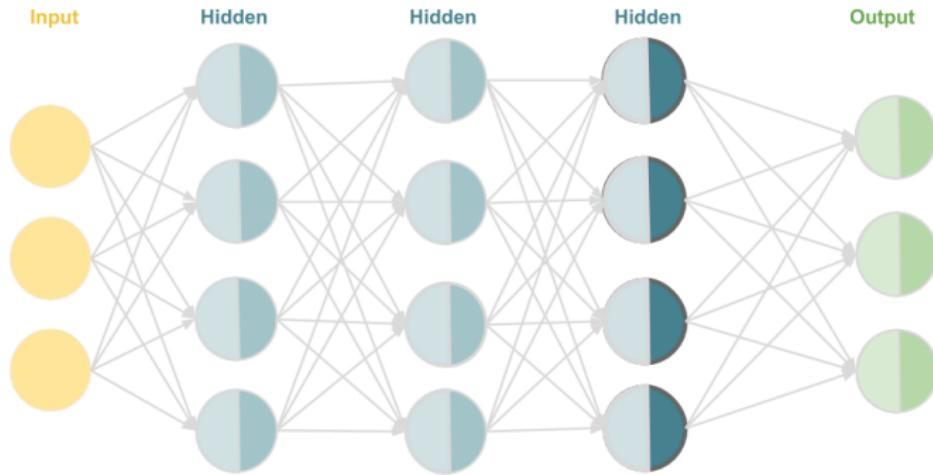


$$\mathbf{x} \quad \mathbf{z}_{in}^{(1)} \quad \mathbf{z}^{(1)} \quad \mathbf{z}_{in}^{(2)} \quad \mathbf{z}^{(2)} \quad \mathbf{z}_{in}^{(3)} = W^{(3)T} \mathbf{z}^{(2)} + \mathbf{b}^{(3)}$$

Below the input vector  $\mathbf{x}$  and hidden states  $\mathbf{z}_{in}^{(1)}, \mathbf{z}^{(1)}, \mathbf{z}_{in}^{(2)}, \mathbf{z}^{(2)}$ , the output vector  $\mathbf{z}_{in}^{(3)}$  is calculated as:

$$\left. \begin{aligned} & 38*1 + 21*(-2) + 0*(-18) + 0*(-7) + 4 \\ & 38*3 + 21*(-4) + 0*8 + 0*0 + (-6) \\ & 38*(-2) + 21*1 + 0*21 + 0*5 + 1 \\ & 38*2 + 21*(-2) + 0*11 + 0*(-13) + (-17) \end{aligned} \right\}$$

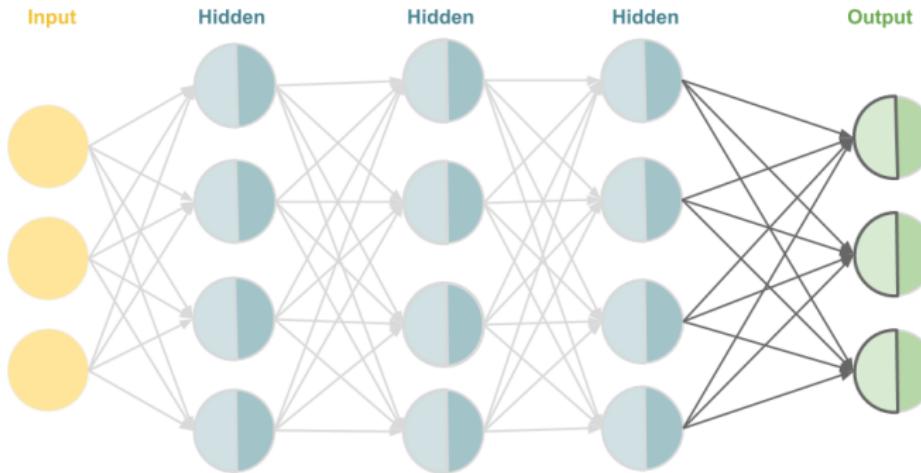
# FEEDFORWARD NEURAL NETWORKS: EXAMPLE



$$\mathbf{x} \quad \mathbf{z}_{in}^{(1)} \quad \mathbf{z}^{(1)} \quad \mathbf{z}_{in}^{(2)} \quad \mathbf{z}^{(2)} \quad \mathbf{z}_{in}^{(3)} \quad \begin{cases} \max(0, 0) \\ \max(0, 24) \\ \max(0, -54) \\ \max(0, 17) \end{cases}$$

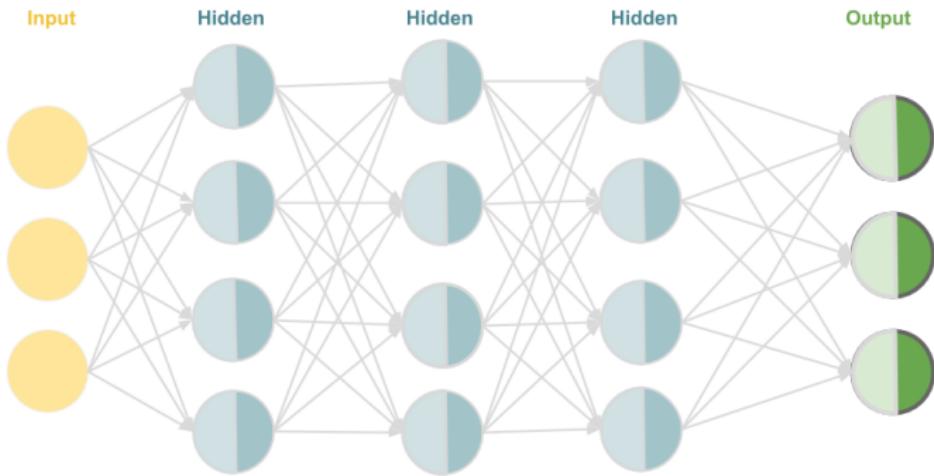
$$\mathbf{z}^{(3)} = \mathbf{z}_{out}^{(3)} = \sigma(\mathbf{z}_{in}^{(3)})$$

# FEEDFORWARD NEURAL NETWORKS: EXAMPLE



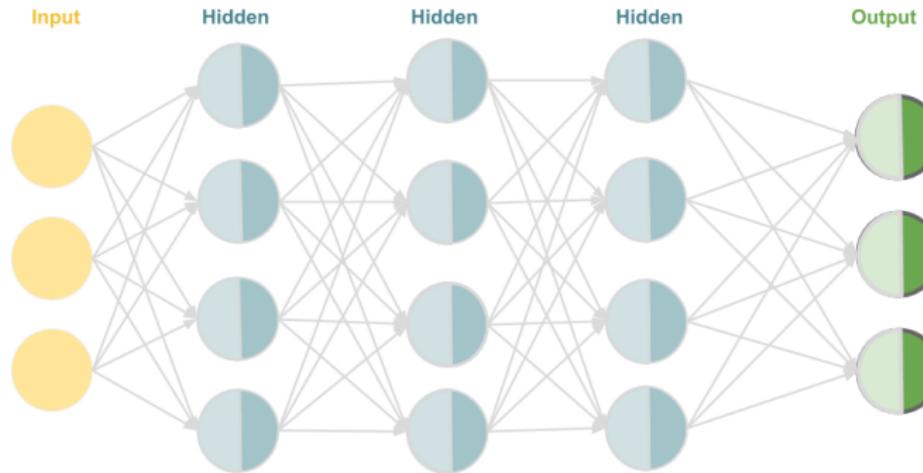
$$\mathbf{x} \quad \begin{pmatrix} 7 \\ 1 \\ -4 \end{pmatrix} \quad \begin{pmatrix} 79 \\ -70 \\ 9 \\ -26 \end{pmatrix} \quad \begin{pmatrix} 79 \\ 0 \\ 9 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 38 \\ 21 \\ -96 \\ -35 \end{pmatrix} \quad \begin{pmatrix} 38 \\ 21 \\ 0 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 24 \\ -54 \\ 17 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 24 \\ 0 \\ 17 \end{pmatrix} \quad \begin{pmatrix} 24*3 + 17*(-4) + (-1) \\ 24*(-2) + 17*3 + (-4) \\ 24*2 + 17*(-1) + (-30) \end{pmatrix}$$
$$\mathbf{z}_{in}^{(1)} \quad \mathbf{z}^{(1)} \quad \mathbf{z}_{in}^{(2)} \quad \mathbf{z}^{(2)} \quad \mathbf{z}_{in}^{(3)} \quad \mathbf{z}^{(3)} \quad \mathbf{f}_{in} = U^T \mathbf{z}^{(3)} + \mathbf{c}$$

# FEEDFORWARD NEURAL NETWORKS: EXAMPLE



|  |   |   |  |  |  |  |  |
|--|---|---|--|--|--|--|--|
| $x$  | $z_{in}^{(1)}$                                      | $z^{(1)}$                                       | $z_{in}^{(2)}$                                       | $z^{(2)}$  | $z_{in}^{(3)}$                                     | $z^{(3)}$  | $f_{in}$                                   |
| $\begin{matrix} 7 \\ 1 \\ -4 \end{matrix}$ | $\begin{matrix} 79 \\ -70 \\ 9 \\ -26 \end{matrix}$ | $\begin{matrix} 79 \\ 0 \\ 9 \\ 0 \end{matrix}$ | $\begin{matrix} 38 \\ 21 \\ -96 \\ -35 \end{matrix}$ | $\begin{matrix} 38 \\ 21 \\ 0 \\ 0 \end{matrix}$ | $\begin{matrix} 0 \\ 24 \\ -54 \\ 17 \end{matrix}$ | $\begin{matrix} 0 \\ 24 \\ 0 \\ 17 \end{matrix}$ | $\begin{matrix} 3 \\ -1 \\ 1 \end{matrix}$ |

# FEEDFORWARD NEURAL NETWORKS: EXAMPLE



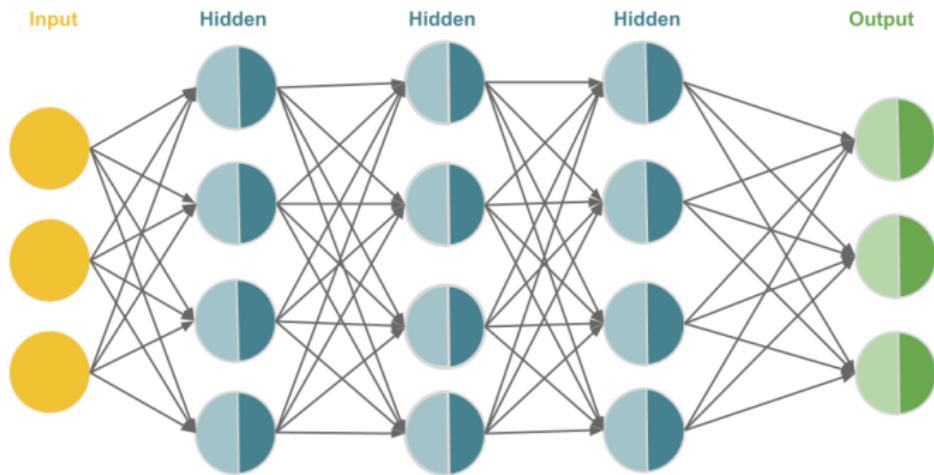
$$\begin{matrix} 3 \\ -1 \\ 1 \end{matrix}$$

$f_{in}$

$$\left\{ \begin{array}{l} \frac{\exp(3)}{\exp(3) + \exp(-1) + \exp(1)} \\ \frac{\exp(-1)}{\exp(3) + \exp(-1) + \exp(1)} \\ \frac{\exp(1)}{\exp(3) + \exp(-1) + \exp(1)} \end{array} \right.$$

$$f = f_{out} = \tau(f_{in})$$

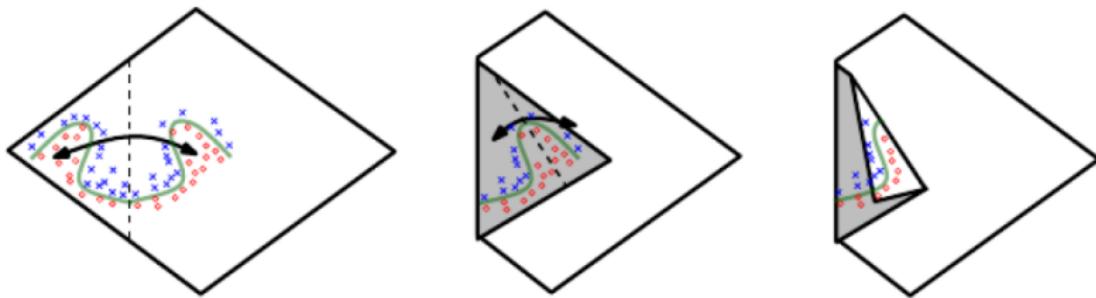
# FEEDFORWARD NEURAL NETWORKS: EXAMPLE



|  |   |   |  |  |  |  |  |   |
|--|---|---|--|--|--|--|--|---|
| $x$  | $z_{in}^{(1)}$  | $z^{(1)}$   | $z_{in}^{(2)}$   | $z^{(2)}$  | $z_{in}^{(3)}$                                       | $z^{(3)}$  | $f_{in}$                                     | $f$   |
| $\begin{bmatrix} 7 \\ 1 \\ -4 \end{bmatrix}$ | $\begin{bmatrix} 79 \\ -70 \\ 9 \\ -26 \end{bmatrix}$ | $\begin{bmatrix} 79 \\ 0 \\ 9 \\ 0 \end{bmatrix}$ | $\begin{bmatrix} 38 \\ 21 \\ -96 \\ -35 \end{bmatrix}$ | $\begin{bmatrix} 38 \\ 21 \\ 0 \\ 0 \end{bmatrix}$ | $\begin{bmatrix} 0 \\ 24 \\ -54 \\ 17 \end{bmatrix}$ | $\begin{bmatrix} 0 \\ 24 \\ 0 \\ 17 \end{bmatrix}$ | $\begin{bmatrix} 3 \\ -1 \\ 1 \end{bmatrix}$ | $\begin{bmatrix} 0.866 \\ 0.015 \\ 0.117 \end{bmatrix}$ |

# WHY ADD MORE LAYERS?

- Multiple layers allow for the extraction of more and more abstract representations.
- Each layer in a feed-forward neural network adds its own degree of non-linearity to the model.



**Figure:** An intuitive, geometric explanation of the exponential advantage of deeper networks formally (Montúfar et al. (2014)).

# DEEP NEURAL NETWORKS

Neural networks today can have hundreds of hidden layers. The greater the number of layers, the "deeper" the network. Historically DNNs were very challenging to train and not popular until the late '00s for several reasons:

- The use of sigmoid activations (e.g., logistic sigmoid and tanh) significantly slowed down training due to a phenomenon known as “vanishing gradients”. The introduction of the ReLU activation largely solved this problem.
- Training DNNs on CPUs was too slow to be practical. Switching over to GPUs cut down training time by more than an order of magnitude.
- When dataset sizes are small, other models (such as SVMs) and techniques (such as feature engineering) often outperform them.

# DEEP NEURAL NETWORKS

- The availability of large datasets and novel architectures that are capable of handling even complex tensor-shaped data (e.g. CNNs for image data), faster hardware, and better optimization and regularization methods made it feasible to successfully implement deep neural networks.
- An increase in depth often translates to an increase in performance on a given task. State-of-the-art neural networks, however, are much more sophisticated than the simple architectures we have encountered so far.

The term "**deep learning**" encompasses all of these developments and refers to the field as a whole.

# Deep Learning

## Brief History

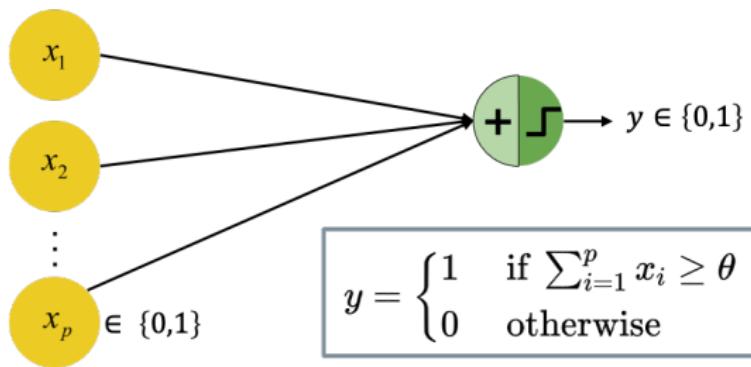


### Learning goals

- Predecessors of modern (deep) neural networks
- History of DL as a field

# A BRIEF HISTORY OF NEURAL NETWORKS

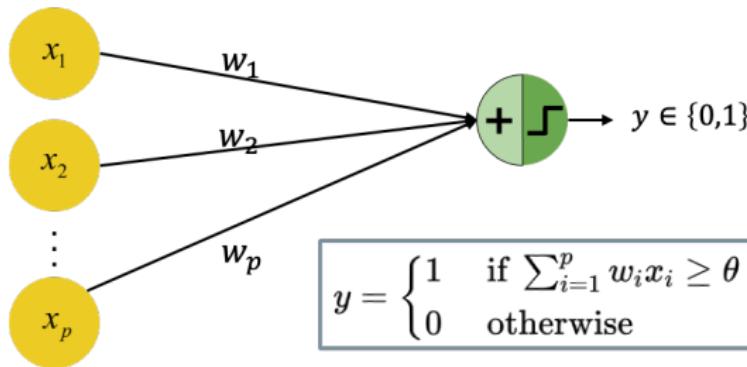
- **1943:** The first artificial neuron, the "Threshold Logic Unit (TLU)", was proposed by Warren McCulloch & Walter Pitts.



- The model is limited to binary inputs.
- It fires/outputs  $+1$  if the input exceeds a certain threshold  $\theta$ .
- The weights are not adjustable, so learning could only be achieved by changing the threshold  $\theta$ .

# A BRIEF HISTORY OF NEURAL NETWORKS

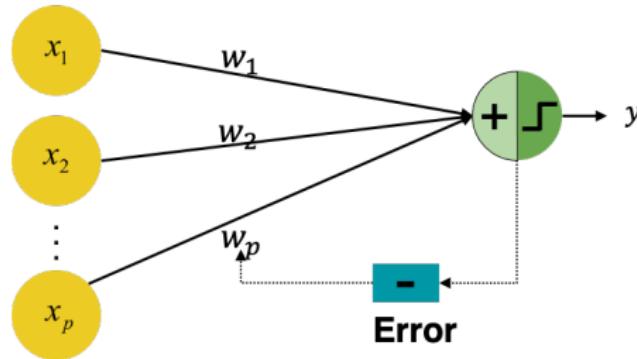
- 1957: The perceptron was invented by Frank Rosenblatt.



- The inputs are not restricted to be binary.
- The weights are adjustable and can be learned by learning algorithms.
- As for the TLU, the threshold is adjustable and decision boundaries are linear.

# A BRIEF HISTORY OF NEURAL NETWORKS

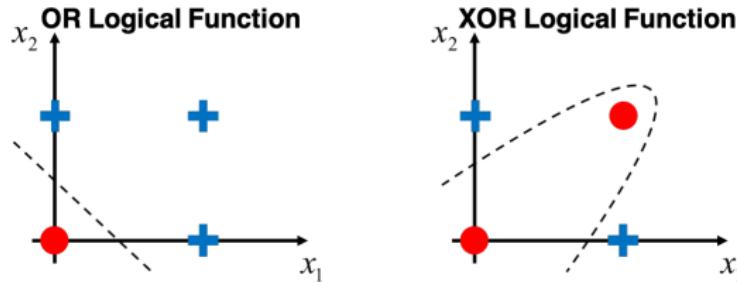
- **1960:** Adaptive Linear Neuron (ADALINE) was invented by Bernard Widrow & Ted Hoff; weights are now adjustable according to the weighted sum of the inputs.



- **1965:** Group method of data handling (also known as polynomial neural networks) by Alexey Ivakhnenko. The first learning algorithm for supervised deep feedforward multilayer perceptrons.

# A BRIEF HISTORY OF NEURAL NETWORKS

- 1969: The first “AI Winter” kicked in.
  - Marvin Minsky & Seymour Papert proved that a perceptron cannot solve the XOR-Problem (linear separability).
  - Less funding ⇒ Standstill in AI/DL research.



- 1985: Multilayer perceptron with backpropagation by David Rumelhart, Geoffrey Hinton, and Ronald Williams.
  - Efficiently compute derivatives of composite functions.
  - Backpropagation was developed already in 1970 by Linnainmaa.

# A BRIEF HISTORY OF NEURAL NETWORKS

- 1985: The second “AI Winter” kicked in.
  - Overly optimistic expectations concerning potential of AI/DL.
  - The phrase “AI” even reached a pseudoscience status.
  - Kernel machines and graphical models both achieved good results on many important tasks.
  - Some fundamental mathematical difficulties in modeling long sequences were identified.



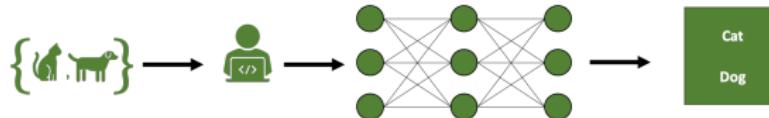
Credit: <https://emerj.com/ai-executive-guides/will-there-be-another-artificial-intelligence-winter-probably-not/>

# A BRIEF HISTORY OF NEURAL NETWORKS

- 2006: Age of deep neural networks began.

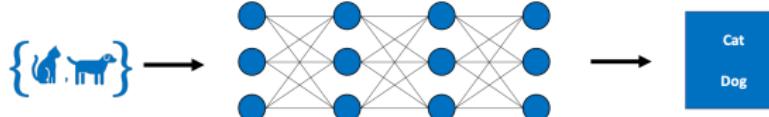
- Geoffrey Hinton showed that a deep belief network could be efficiently trained using *greedy layer-wise pretraining*.
- This wave of research popularized the use of the term deep learning to emphasize that researchers were now able to train deeper neural networks than had been possible before.
- At this time, deep neural networks outperformed competing AI systems based on other ML technologies as well as hand-designed functionality.

## Machine Learning



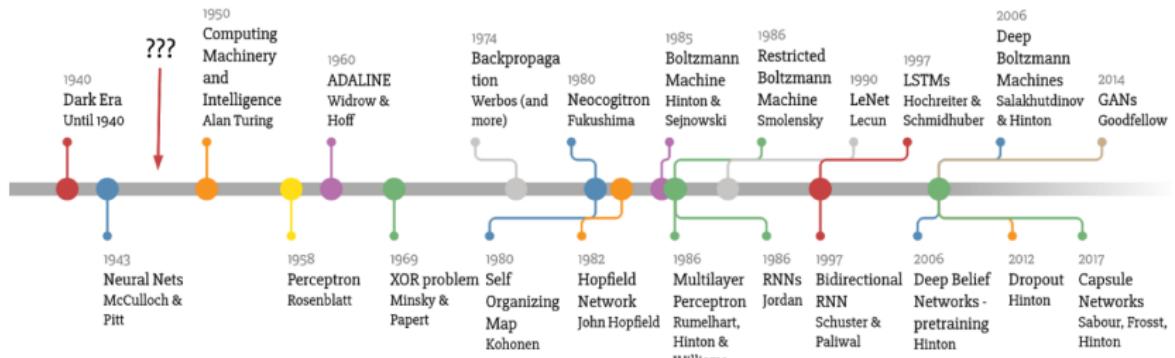
---

## Deep Learning



# A BRIEF HISTORY OF NEURAL NETWORKS

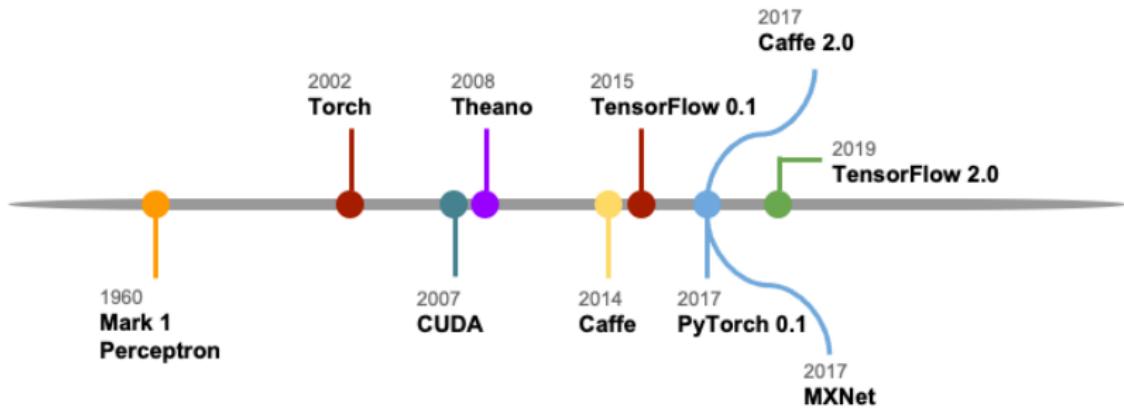
## Deep Learning Timeline



Credit: <https://towardsdatascience.com/a-weird-introduction-to-deep-learning-7828803693b0>

# A BRIEF HISTORY OF NEURAL NETWORKS

## History of DL Tools



# A BRIEF HISTORY OF NEURAL NETWORKS



**Figure:** IBM Supercomputer

- Watson is a question-answering system capable of answering questions posed in natural language, developed in IBM's DeepQA project.
- In 2011, Watson competed on *Jeopardy!* against champions Brad Rutter and Ken Jennings, winning the first place prize of \$1 million.

# A BRIEF HISTORY OF NEURAL NETWORKS



**Figure:** Google self driving car (Waymo)

- Google's development of self-driving technology began on January 17, 2009, at the company's secretive X lab.
- By January 2020, 20 million miles of self-driving on public roads had been completed by Waymo.

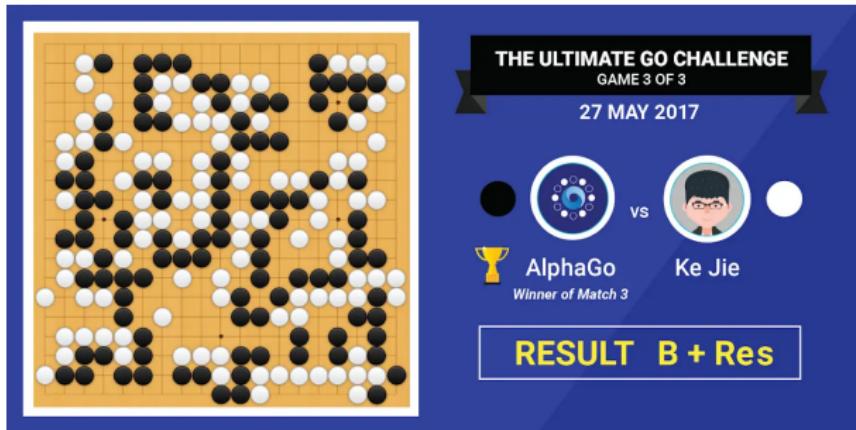
# A BRIEF HISTORY OF NEURAL NETWORKS



Credit: DeepMind

- **AlphaFold** is a deep learning system, developed by Google DeepMind, for determining a protein's 3D shape from its amino-acid sequence.
- In 2018 and 2020, AlphaFold placed first in the overall rankings of the Critical Assessment of Techniques for Protein Structure Prediction (CASP).

# A BRIEF HISTORY OF NEURAL NETWORKS



Credit: DeepMind

- **AlphaGo**, originally developed by DeepMind, is a deep learning system that plays the board game Go. In 2017, the Master version of AlphaGo beat Ke Jie, the number one ranked player in the world at the time.
- While there are several extensions to AlphaGo (e.g., Master AlphaGo, AlphaGo Zero, AlphaZero, and MuZero), the main idea is the same: search for optimal moves based on knowledge acquired by machine learning.

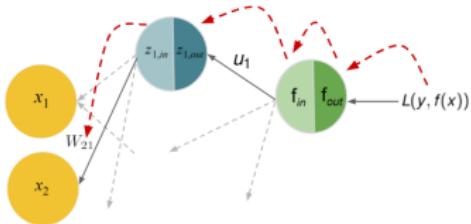
# A BRIEF HISTORY OF NEURAL NETWORKS



- **Generative Pre-trained Transformer 3 (GPT-3)** is the third generation of the GPT model, introduced by OpenAI in May 2020, to produce human-like text.
- There are 175 billion parameters to be learned by the algorithm, but the quality of the generated text is so high that it is hardly possible to distinguish it from a human-written text.

# Deep Learning

## Basic Backpropagation 1



### Learning goals

- Forward and backward passes
- Chain rule
- Details of backprop

# BACKPROPAGATION: BASIC IDEA

We would like to run ERM by GD on:

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

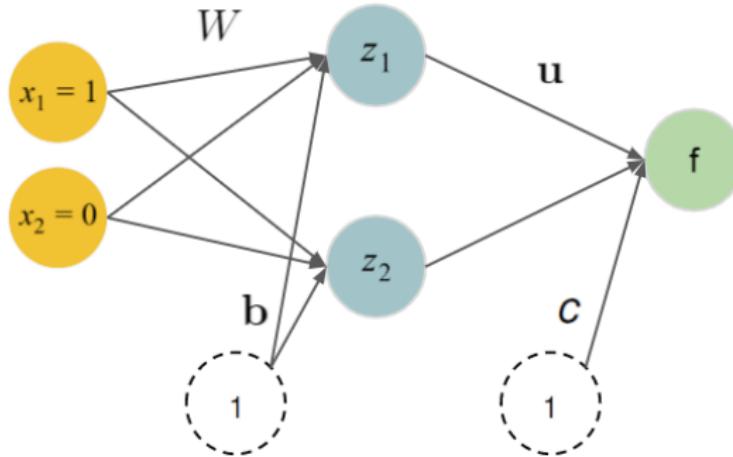
Backprop training of NNs runs in 2 alternating steps, for one  $\mathbf{x}$ :

- ① **Forward pass:** Inputs flow through model to outputs. We then compute the observation loss. We covered that.
- ② **Backward pass:** Loss flows backwards to update weights so error is reduced, as in GD.

We will see: This is simply (S)GD in disguise, cleverly using the chain rule, so we can reuse a lot of intermediate results.

# XOR EXAMPLE

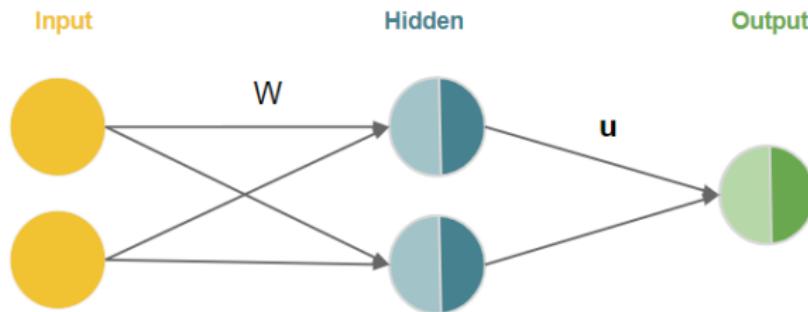
- As activations (hidden and outputs) we use the logistic.
- We run one FP and BP on  $\mathbf{x} = (1, 0)^T$  with  $y = 1$ .
- We use L2 loss between 0-1 labels and the predicted probabilities. This is a bit uncommon, but computations become simpler.



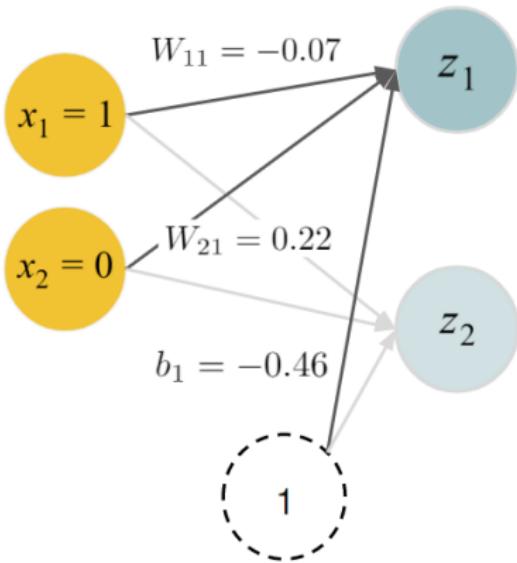
Note: We will only show rounded decimals.

# FORWARD PASS

- We will divide the FP into four steps:
  - the inputs of  $z_i$ :  $\mathbf{z}_{i,in}$
  - the activations of  $z_i$ :  $\mathbf{z}_{i,out}$
  - the input of  $f$ :  $\mathbf{f}_{in}$
  - and finally the activation of  $f$ :  $\mathbf{f}_{out}$



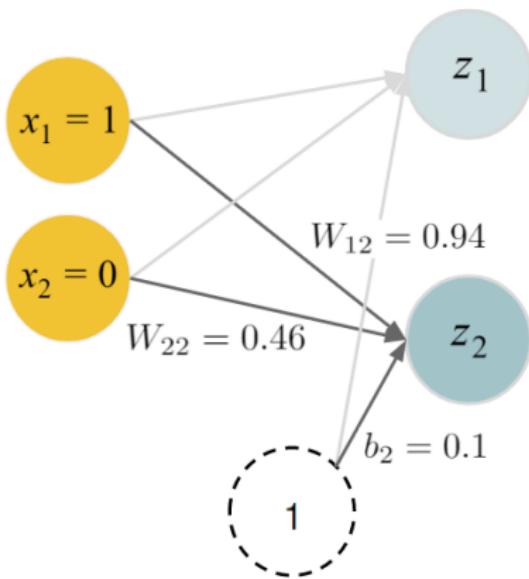
# FORWARD PASS



$$z_{1,in} = \mathbf{W}_1^T \mathbf{x} + b_1 = 1 \cdot (-0.07) + 0 \cdot 0.22 + 1 \cdot (-0.46) = -0.53$$

$$z_{1,out} = \sigma(z_{1,in}) = \frac{1}{1 + \exp(-(-0.53))} = 0.3705$$

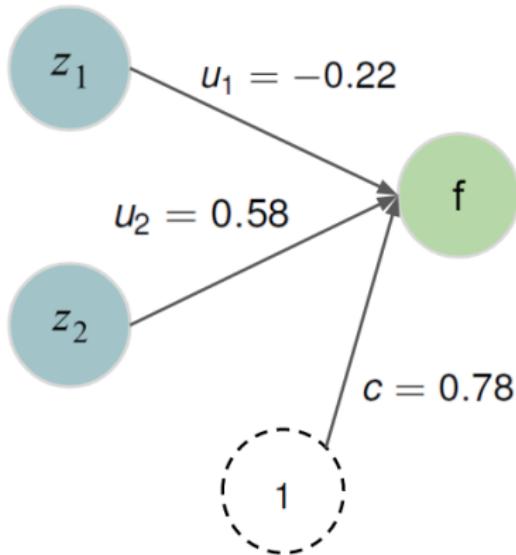
# FORWARD PASS



$$z_{2,in} = \mathbf{W}_2^T \mathbf{x} + b_2 = 1 \cdot 0.94 + 0 \cdot 0.46 + 1 \cdot 0.1 = 1.04$$

$$z_{2,out} = \sigma(z_{2,in}) = \frac{1}{1 + \exp(-1.04)} = 0.7389$$

# FORWARD PASS



$$f_{in} = \mathbf{u}^T \mathbf{z} + c = 0.3705 \cdot (-0.22) + 0.7389 \cdot 0.58 + 1 \cdot 0.78 = 1.1122$$

$$f_{out} = \tau(f_{in}) = \frac{1}{1 + \exp(-1.1122)} = 0.7525$$

# FORWARD PASS

- The FP predicted  $f_{out} = 0.7525$
- Now, we compare the prediction  $f_{out} = 0.7525$  and the true label  $y = 1$  using the L2-loss:

$$\begin{aligned}L(y, f(\mathbf{x})) &= \frac{1}{2}(y - f(\mathbf{x}^{(i)} | \boldsymbol{\theta}))^2 = \frac{1}{2}(y - f_{out})^2 \\&= \frac{1}{2}(1 - 0.7525)^2 = 0.0306\end{aligned}$$

- The calculation of the gradient is performed backwards (starting from the output layer), so that results can be reused.

# BACKWARD PASS

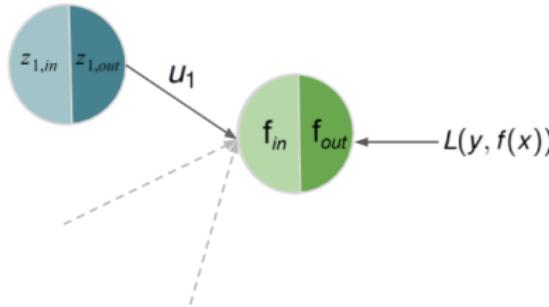
The main ingredients of the backward pass are:

- to reuse the results of the forward pass  
(here:  $z_{i,in}$ ,  $z_{i,out}$ ,  $f_{in}$ ,  $f_{out}$ )
- reuse the **intermediate results** from the chain rule
- the derivative of the activations and some affine functions

# BACKWARD PASS

- Let's start to update  $u_1$ . We recursively apply the chain rule:

$$\frac{\partial L(y, f(\mathbf{x}))}{\partial u_1} = \frac{\partial L(y, f(\mathbf{x}))}{\partial f_{out}} \cdot \frac{\partial f_{out}}{\partial f_{in}} \cdot \frac{\partial f_{in}}{\partial u_1}$$

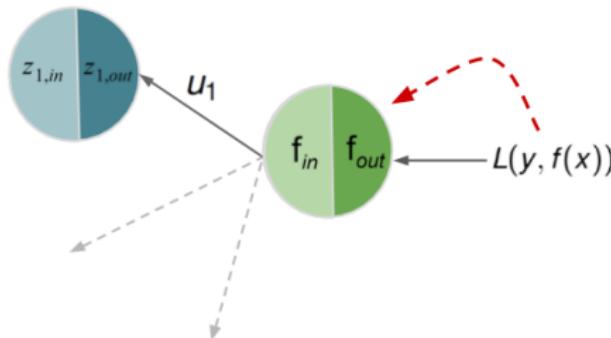


**Figure:** Snippet from our NN, with backward path for  $u_1$ .

# BACKWARD PASS

- 1st step: The derivative of L2 is easy; we know  $f_{out}$  from FP.

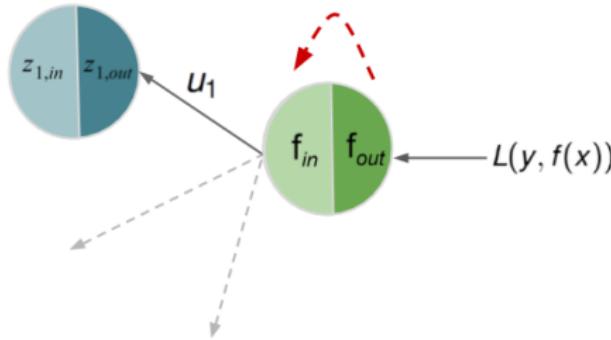
$$\begin{aligned}\frac{\partial L(y, f(\mathbf{x}))}{\partial f_{out}} &= \frac{d}{\partial f_{out}} \frac{1}{2} (y - f_{out})^2 = -\underbrace{(y - f_{out})}_{\hat{=} \text{residual}} \\ &= -(1 - 0.7525) = -0.2475\end{aligned}$$



# BACKWARD PASS

- 2nd step.  $f_{out} = \sigma(f_{in})$ , use rule for  $\sigma'$ , use  $f_{in}$  from FP.

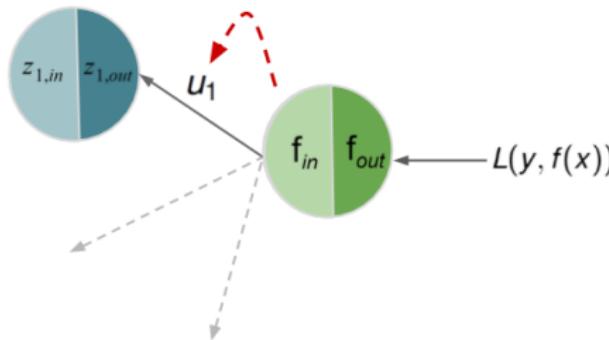
$$\begin{aligned}\frac{\partial f_{out}}{\partial f_{in}} &= \sigma(f_{in}) \cdot (1 - \sigma(f_{in})) \\ &= 0.7525 \cdot (1 - 0.7525) = 0.1862\end{aligned}$$



# BACKWARD PASS

- 3rd step. Derivative of the linear input is easy; use  $z_{1,out}$  from FP.

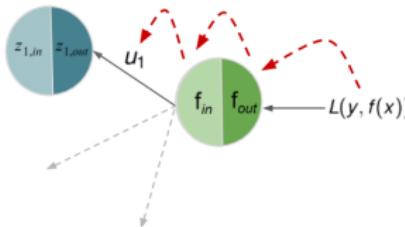
$$\frac{\partial f_{in}}{\partial u_1} = \frac{\partial(u_1 \cdot z_{1,out} + u_2 \cdot z_{2,out} + c \cdot 1)}{\partial u_1} = z_{1,out} = 0.3705$$



# BACKWARD PASS

- Plug it together:

$$\begin{aligned}\frac{\partial L(y, f(\mathbf{x}))}{\partial u_1} &= \frac{\partial L(y, f(\mathbf{x}))}{\partial f_{out}} \cdot \frac{\partial f_{out}}{\partial f_{in}} \cdot \frac{\partial f_{in}}{\partial u_1} \\ &= -0.2475 \cdot 0.1862 \cdot 0.3705 = -0.0171\end{aligned}$$



- With LR  $\alpha = 0.5$ :

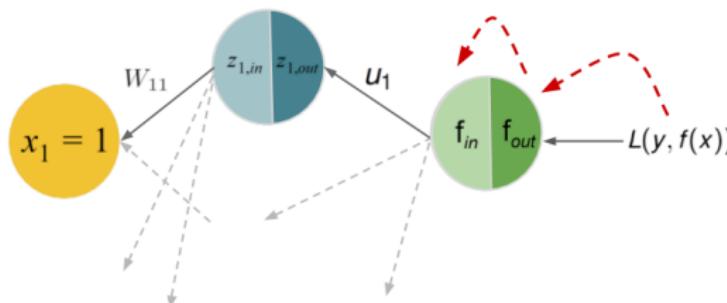
$$\begin{aligned}u_1^{[new]} &= u_1^{[old]} - \alpha \cdot \frac{\partial L(y, f(\mathbf{x}))}{\partial u_1} \\ &= -0.22 - 0.5 \cdot (-0.0171) = -0.2115\end{aligned}$$

# BACKWARD PASS

- Now for  $W_{11}$ :

$$\frac{\partial L(y, f(\mathbf{x}))}{\partial W_{11}} = \frac{\partial L(y, f(\mathbf{x}))}{\partial f_{out}} \cdot \frac{\partial f_{out}}{\partial f_{in}} \cdot \frac{\partial f_{in}}{\partial z_{1,out}} \cdot \frac{\partial z_{1,out}}{\partial z_{1,in}} \cdot \frac{\partial z_{1,in}}{\partial W_{11}}$$

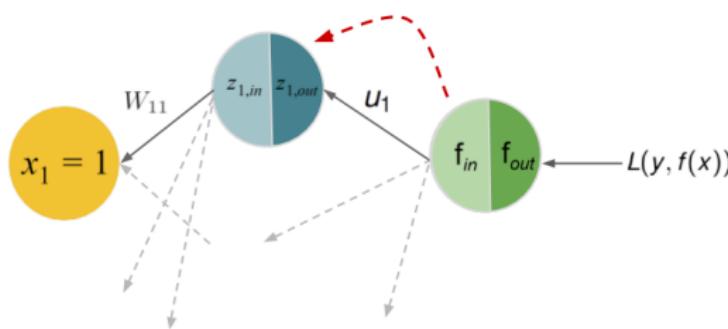
- We know  $\frac{\partial L(y, f(\mathbf{x}))}{\partial f_{out}}$  and  $\frac{\partial f_{out}}{\partial f_{in}}$  from BP for  $u_1$ .



# BACKWARD PASS

- $f_{in} = u_1 \cdot z_{1,out} + u_2 \cdot z_{2,out} + c \cdot 1$  is linear, easy and we know  $u_1$ :

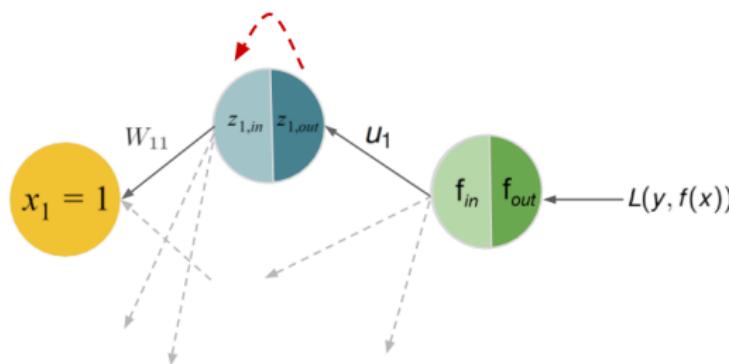
$$\frac{\partial f_{in}}{\partial z_{1,out}} = u_1 = -0.22$$



# BACKWARD PASS

- Next. Use rule for  $\sigma'$  and FP results:

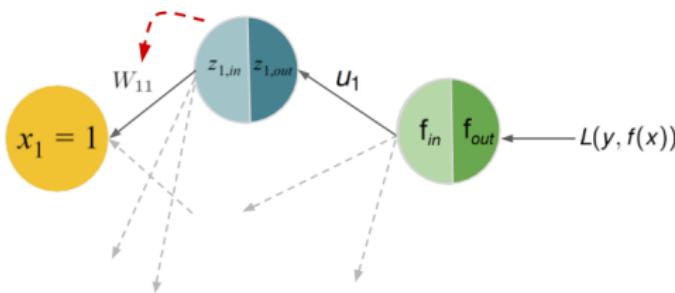
$$\begin{aligned}\frac{\partial z_{1,out}}{\partial z_{1,in}} &= \sigma(z_{1,in}) \cdot (1 - \sigma(z_{1,in})) \\ &= 0.3705 \cdot (1 - 0.3705) = 0.2332\end{aligned}$$



# BACKWARD PASS

- $z_{1,in} = x_1 \cdot W_{11} + x_2 \cdot W_{21} + b_1 \cdot 1$  is linear and depends on inputs:

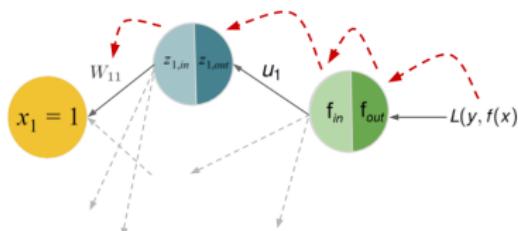
$$\frac{\partial z_{1,in}}{\partial W_{11}} = x_1 = 1$$



# BACKWARD PASS

- Plugging together:

$$\begin{aligned}\frac{\partial L(y, f(\mathbf{x}))}{\partial W_{11}} &= \frac{\partial L(y, f(\mathbf{x}))}{\partial f_{out}} \cdot \frac{\partial f_{out}}{\partial f_{in}} \cdot \frac{\partial f_{in}}{\partial z_{1,out}} \cdot \frac{\partial z_{1,out}}{\partial z_{1,in}} \cdot \frac{\partial z_{1,in}}{\partial W_{11}} \\ &= (-0.2475) \cdot 0.1862 \cdot (-0.22) \cdot 0.2332 \cdot 1 \\ &= 0.0024\end{aligned}$$



- Full SGD update:

$$\begin{aligned}W_{11}^{[new]} &= W_{11}^{[old]} - \alpha \cdot \frac{\partial L(y, f(\mathbf{x}))}{\partial W_{11}} \\ &= -0.07 - 0.5 \cdot 0.0024 = -0.0712\end{aligned}$$

# RESULT

- We can do this for all weights:

$$W = \begin{pmatrix} -0.0712 & 0.9426 \\ 0.22 & 0.46 \end{pmatrix}, b = \begin{pmatrix} -0.4612 \\ 0.1026 \end{pmatrix},$$

$$u = \begin{pmatrix} -0.2115 \\ 0.5970 \end{pmatrix} \text{ and } c = 0.8030.$$

- Yields  $f(\mathbf{x} \mid \theta^{[new]}) = 0.7615$  and loss  $\frac{1}{2}(1 - 0.7615)^2 = 0.0284$ .
- Before, we had  $f(\mathbf{x} \mid \theta^{[old]}) = 0.7525$  and higher loss 0.0306.

Now rinse and repeat. This was one training iter, we do thousands.

# INTRODUCTION TO MACHINE LEARNING

ML Basics

Supervised Regression

Supervised Classification

Performance Evaluation

k-NN

Classification and Regression Trees (CART)

Random Forests

Neural Networks

**Tuning**

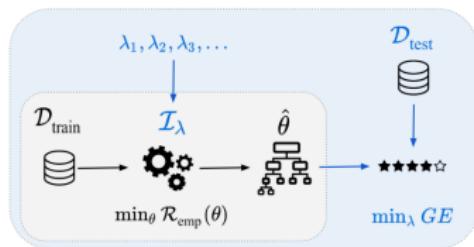
Nested Resampling

# Introduction to Machine Learning

## Hyperparameter Tuning - Introduction

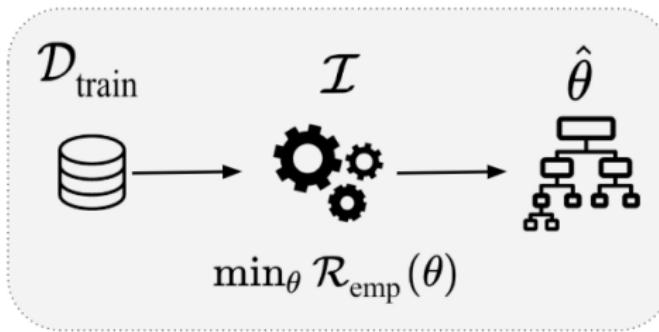
### Learning goals

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



# MOTIVATING EXAMPLE

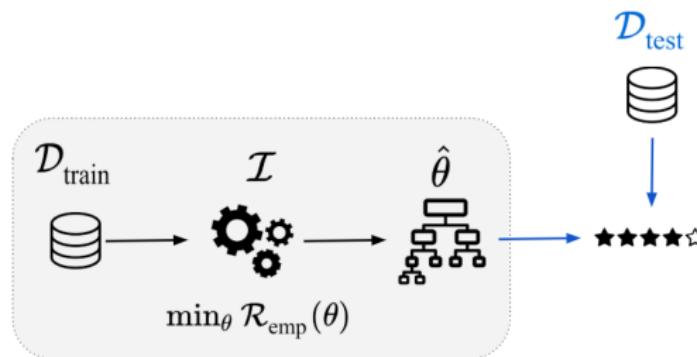
- Given a data set, we want to train a classification tree.
- We feel that a maximum tree depth of 4 has worked out well for us previously, so we decide to set this hyperparameter to 4.
- The learner ("inducer")  $\mathcal{I}$  takes the input data, internally performs **empirical risk minimization**,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



# MOTIVATING EXAMPLE

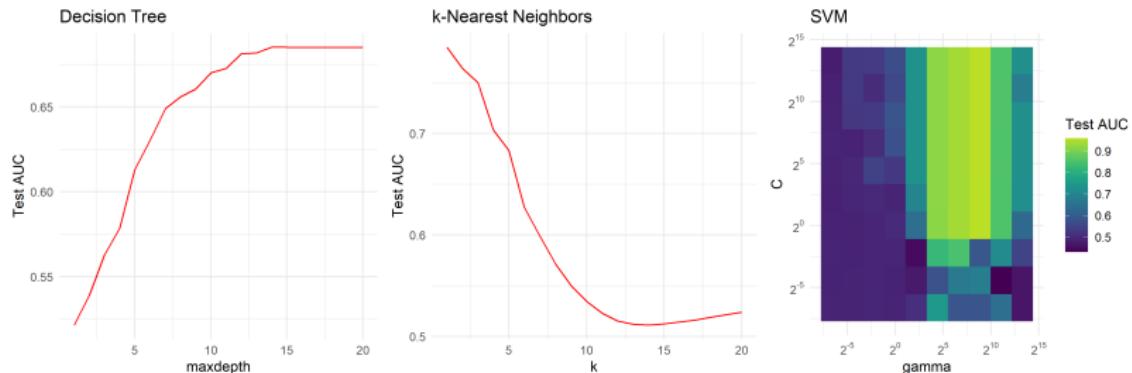
- We are **actually** interested in the **generalization performance**  $\widehat{GE}(\hat{f})$  of the estimated model on new, previously unseen data.
- We estimate the generalization performance by evaluating the model  $\hat{f} = \mathcal{I}(\mathcal{D}_{\text{train}}, \lambda)$  on a test set  $\mathcal{D}_{\text{test}}$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

$$\widehat{GE}_{\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}}}(\mathcal{I}, \lambda, n_{\text{train}}, \rho) = \rho \left( \mathbf{y}_{\mathcal{D}_{\text{test}}}, \mathbf{F}_{\mathcal{D}_{\text{test}}, \hat{f}} \right)$$



# 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.
- Consider a simulation example of 3 ML algorithms below, where we use the dataset *mlbench.spiral* and 10,000 testing points. As can be seen, variating hyperparameters can lead to big difference in model's generalization performance.

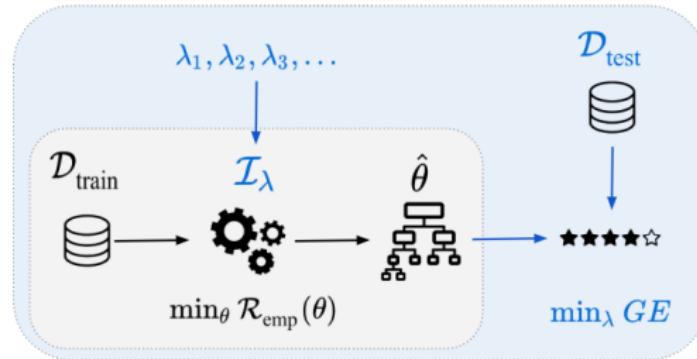


# MOTIVATING EXAMPLE

For our example this could mean:

- Data too complex to be modeled by a tree of depth 4
- Data much simpler than we thought, a tree of depth 4 overfits

- ⇒ Algorithmically try out different values for the tree depth. For each maximum depth  $\lambda$ , we have to train the model **to completion** and evaluate its performance on the test set.
- We choose the tree depth  $\lambda$  that is **optimal** w.r.t. the generalization error of the model.



# MODEL PARAMETERS VS. HYPERPARAMETERS

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

**Model parameters**  $\theta$  are optimized during training. They are an **output** of the training.

Examples:

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

# MODEL PARAMETERS VS. HYPERPARAMETERS

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

The process of finding the best hyperparameters is called **tuning**.

Examples:

- Maximum depth of a tree
- $k$  and which distance measure to use for  $k$ -NN
- Number and maximal order of interactions to be included in a linear regression model
- Number of optimization steps if the empirical risk minimization is done via gradient descent

# TYPES OF HYPERPARAMETERS

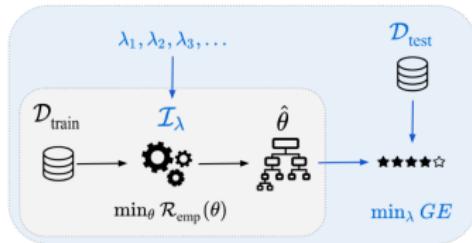
We summarize all hyperparameters we want to tune in a vector  $\lambda \in \Lambda$  of (possibly) mixed type. HPs can have different types:

- 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

- Definition of HPO objective and components
- Understand its properties
- What makes tuning challenging

# HYPERPARAMETER OPTIMIZATION

**Hyperparameters (HP)**  $\lambda$  are parameters that are *inputs* to learner  $\mathcal{I}$  which performs ERM on training data set to find optimal **model parameters**  $\theta$ . HPs can influence the generalization performance in a non-trivial and subtle way.

**Hyperparameter optimization (HPO) / Tuning** is the process of finding a well-performing hyperparameter configuration (HPC)  $\lambda \in \tilde{\Lambda}$  for an learner  $\mathcal{I}_\lambda$ .

# OBJECTIVE AND SEARCH SPACE

Search space  $\tilde{\Lambda} \subset \Lambda$  with all optimized HPs and ranges:

$$\tilde{\Lambda} = \tilde{\Lambda}_1 \times \tilde{\Lambda}_2 \times \cdots \times \tilde{\Lambda}_l$$

where  $\tilde{\Lambda}_i$  is a bounded subset of the domain of the  $i$ -th HP  $\Lambda_i$ , and can be either continuous, discrete, or categorical.

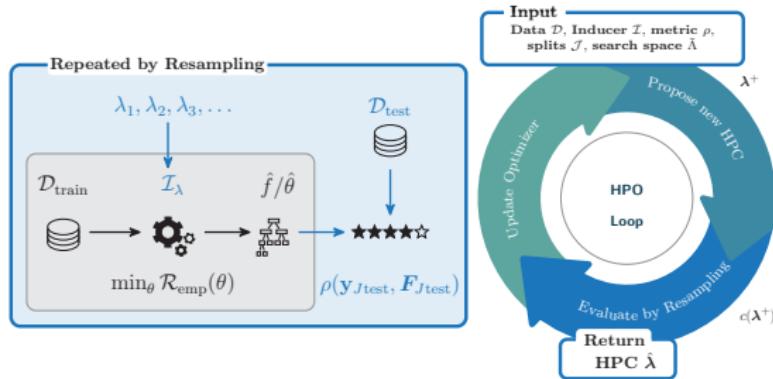
The general HPO problem is defined as:

$$\lambda^* \in \arg \min_{\lambda \in \tilde{\Lambda}} c(\lambda) = \arg \min_{\lambda \in \tilde{\Lambda}} \widehat{GE}(\mathcal{I}, \mathcal{J}, \rho, \lambda)$$

with  $\lambda^*$  as theoretical optimum, and  $c(\lambda)$  is short for estim. gen. error when  $\mathcal{I}$ , resampling splits  $\mathcal{J}$ , performance measure  $\rho$  are fixed.

# OBJECTIVE AND SEARCH SPACE

$$\lambda^* \in \arg \min_{\lambda \in \tilde{\Lambda}} c(\lambda) = \arg \min_{\lambda \in \tilde{\Lambda}} \widehat{\text{GE}}(\mathcal{I}, \mathcal{J}, \rho, \lambda)$$



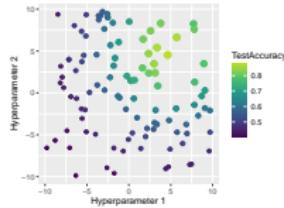
- Evals are stored in **archive**  
 $\mathcal{A} = ((\lambda^{(1)}, c(\lambda^{(1)})), (\lambda^{(2)}, c(\lambda^{(2)})), \dots)$ , with  
 $\mathcal{A}^{[t+1]} = \mathcal{A}^{[t]} \cup (\lambda^+, c(\lambda^+))$ .
- We can define tuner as function  $\tau : (\mathcal{D}, \mathcal{I}, \tilde{\Lambda}, \mathcal{J}, \rho) \mapsto \hat{\lambda}$

# WHY IS TUNING SO HARD?

- Tuning is usually **black box**: No derivatives of the objective are available. We can only eval the performance for a given HPC via a computer program (CV of learner on data).
- Every evaluation can require multiple train and predict steps, hence it's **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 can have non-metric, complicated structure.
- Many standard optimization algorithms cannot handle these properties.

# Introduction to Machine Learning

## Hyperparameter Tuning - Basic Techniques



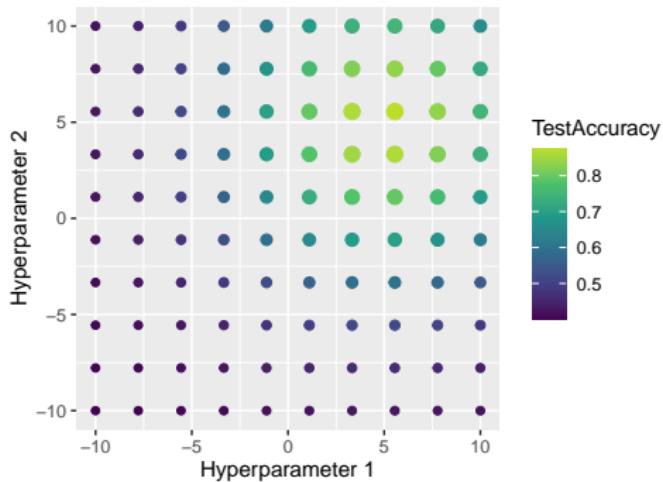
### Learning goals

- 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

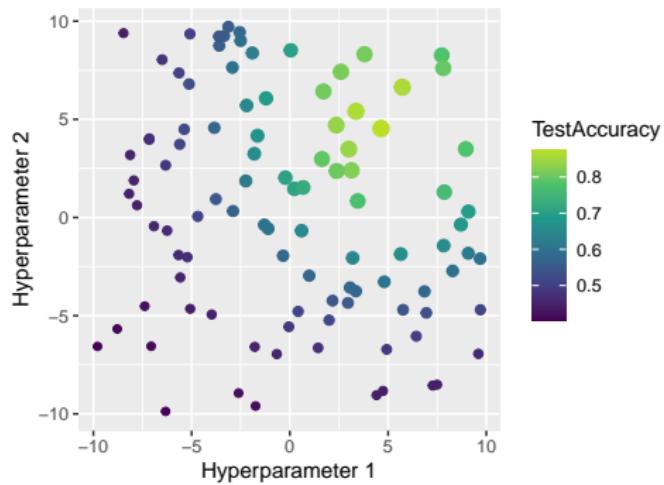
## Disadvantages

- 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

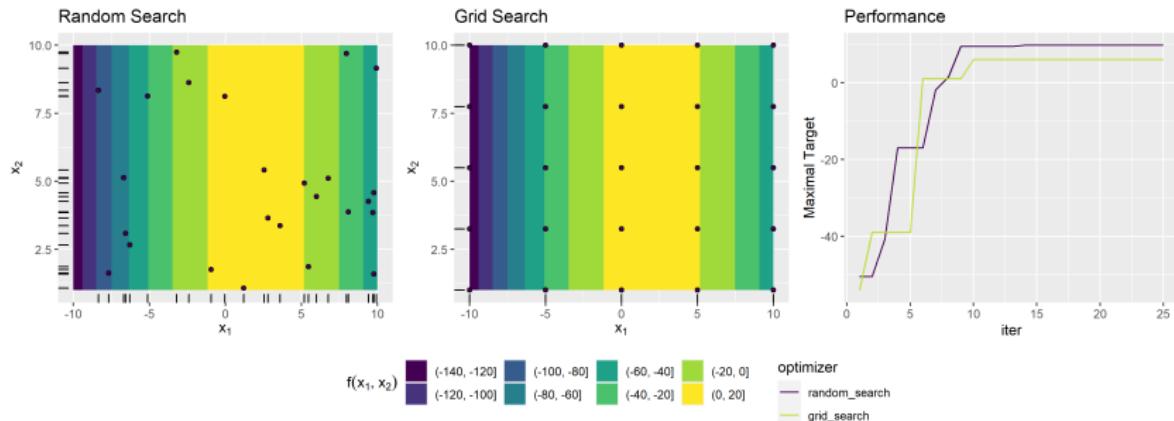
- Like grid search: very easy to implement, all parameter types possible, trivial parallelization
- Anytime algorithm: can stop the search whenever our budget for computation is exhausted, or continue until we reach our performance goal.
- No discretization: each individual parameter is tried with a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

# RANDOM SEARCH VS. GRID SEARCH

We consider a maximization problem on the function  $f(x_1, x_2) = g(x_1) + h(x_2) \approx g(x_1)$ , i.e. in order to maximize the target,  $x_1$  should be the parameter to focus on.

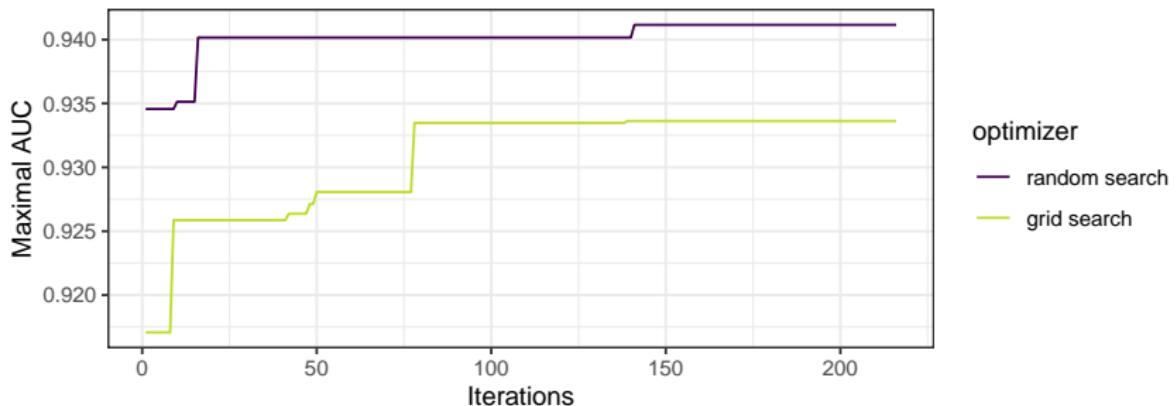


⇒ In this setting, random search is more superior as we get a better coverage for the parameter  $x_1$  in comparison with grid search, where we only discover 5 distinct values for  $x_1$ .

# TUNING EXAMPLE

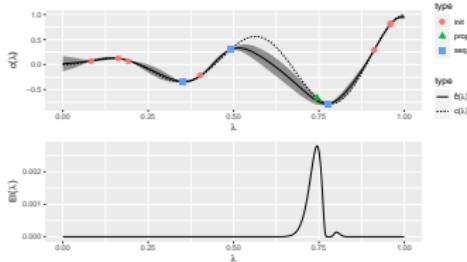
Tuning random forest with grid search/random search and 5CV on the sonar data set for AUC:

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



# Introduction to Machine Learning

## Hyperparameter Tuning - Advanced Tuning Techniques

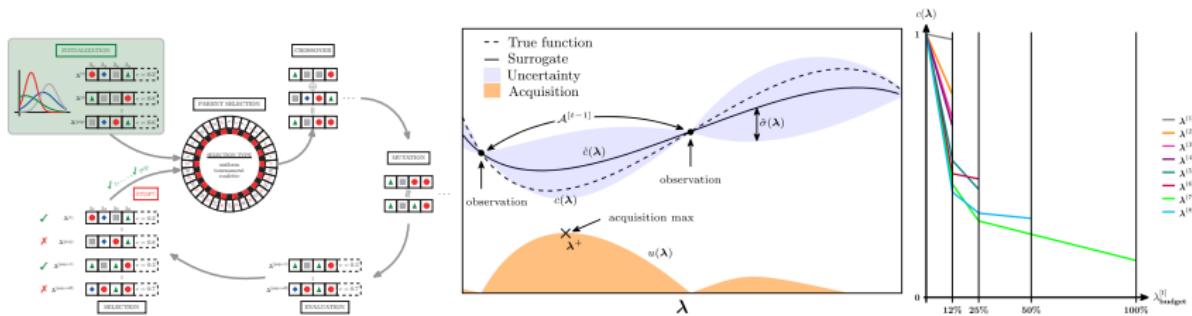


### Learning goals

- Basic idea of evolutionary algorithms
- and Bayesian Optimization
- and hyperband

# HPO – MANY APPROACHES

- Evolutionary algorithms
- Bayesian / model-based optimization
- Multi-fidelity optimization, e.g. Hyperband

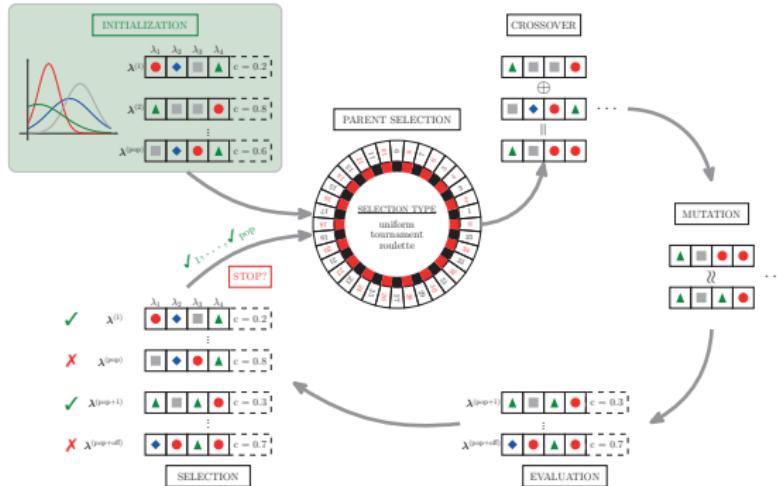


HPO methods can be characterized by:

- how the exploration vs. exploitation trade-off is handled
- how the inference vs. search trade-off is handled

Further aspects: Parallelizability, local vs. global behavior, handling of noisy observations, multifidelity and search space complexity.

# EVOLUTIONARY STRATEGIES



- Are a class of stochastic population-based optimization methods inspired by the concepts of biological evolution
- Are applicable to HPO since they do not require gradients
- Mutation is the (randomized) change of one or a few HP values in a configuration.
- Crossover creates a new HPC by (randomly) mixing the values of two other configurations.

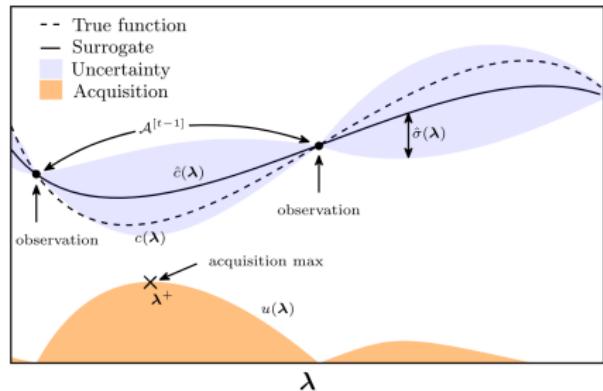
# BAYESIAN OPTIMIZATION

BO sequentially iterates:

- ➊ Approximate  $\lambda \mapsto c(\lambda)$  by (nonlin) regression model  $\hat{c}(\lambda)$ , from evaluated configurations (archive)

- ➋ Propose candidates via optimizing an acquisition function that is based on the surrogate  $\hat{c}(\lambda)$

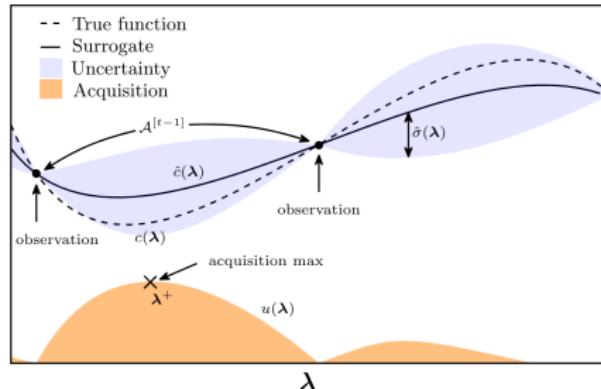
- ➌ Evaluate candidate(s) proposed in 2, then go to 1
- Important trade-off: **Exploration** (evaluate candidates in under-explored areas) vs. **exploitation** (search near promising areas)



# BAYESIAN OPTIMIZATION

## Surrogate Model:

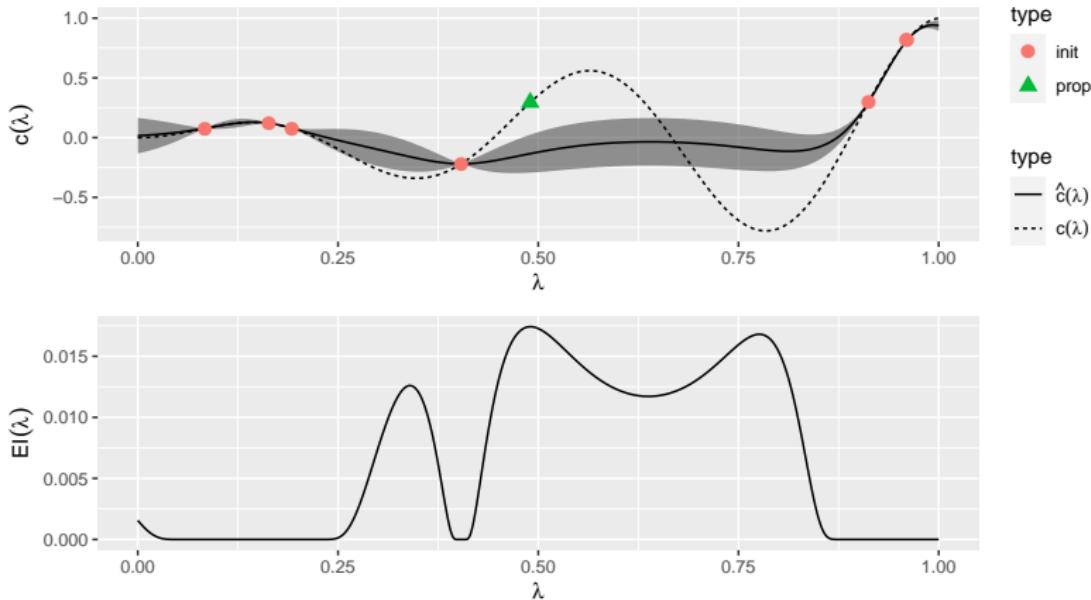
- Probabilistic modeling of  $C(\lambda) \sim (\hat{c}(\lambda), \hat{\sigma}(\lambda))$  with posterior mean  $\hat{c}(\lambda)$  and uncertainty  $\hat{\sigma}(\lambda)$ .
- Typical choices for numeric spaces are Gaussian Processes; random forests for mixed spaces



## Acquisition Function:

- Balance exploration (high  $\hat{\sigma}$ ) vs. exploitation (low  $\hat{c}$ ).
- Lower confidence bound (LCB):  $a(\lambda) = \hat{c}(\lambda) - \kappa \cdot \hat{\sigma}(\lambda)$
- Expected improvement (EI):  $a(\lambda) = \mathbb{E} [\max \{c_{\min} - C(\lambda), 0\}]$  where ( $c_{\min}$  is best cost value from archive)
- Optimizing  $a(\lambda)$  is still difficult, but cheap(er)

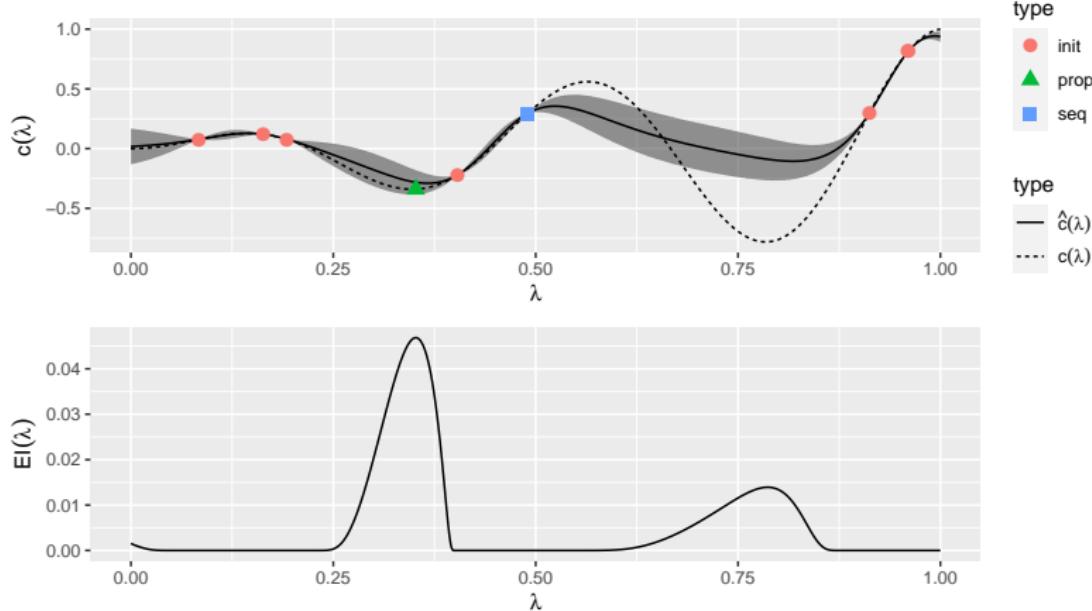
# BAYESIAN OPTIMIZATION



Upper plot: The surrogate model (black, solid) models the *unknown* relationship between input and output (black, dashed) based on the initial design (red points).

Lower plot: Mean and variance of the surrogate model are used to derive the expected improvement (EI) criterion. The point that maximizes the EI is proposed (green point).

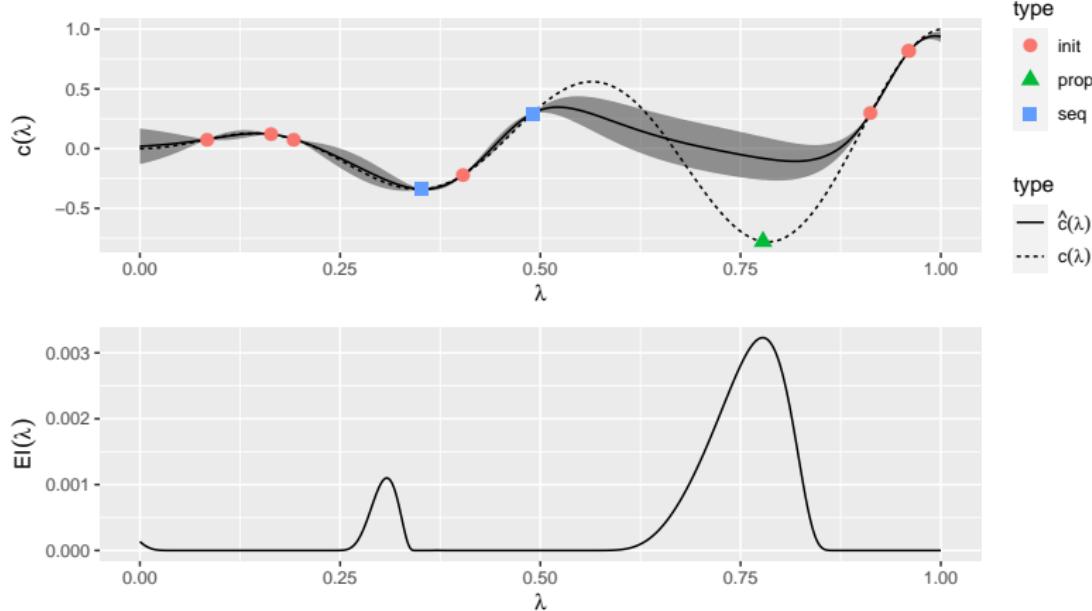
# BAYESIAN OPTIMIZATION



Upper plot: The surrogate model (black, solid) models the *unknown* relationship between input and output (black, dashed) based on the initial design (red points).

Lower plot: Mean and variance of the surrogate model are used to derive the expected improvement (EI) criterion. The point that maximizes the EI is proposed (green point).

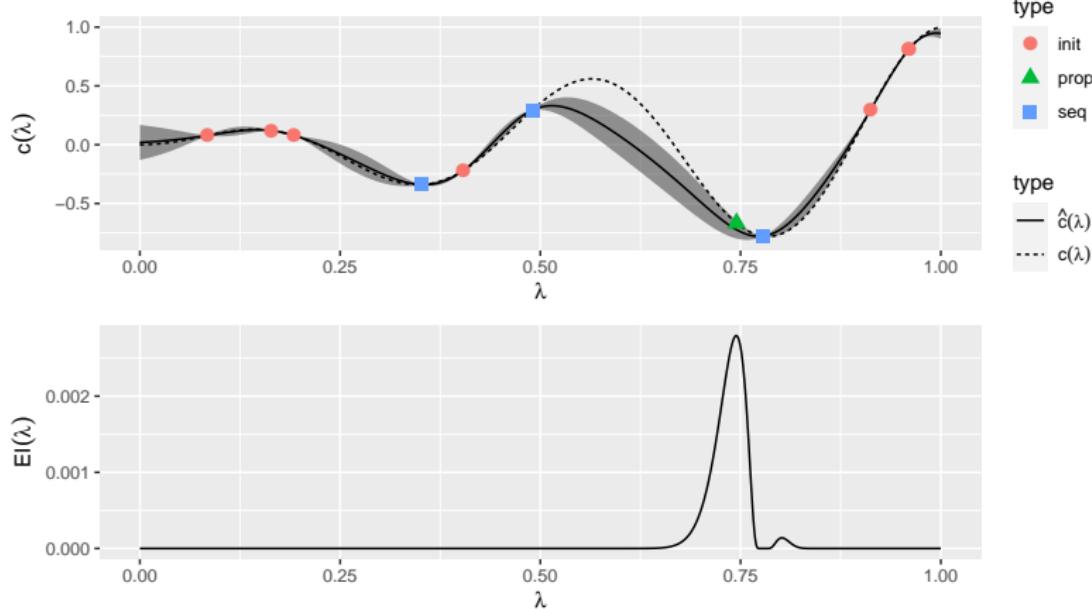
# BAYESIAN OPTIMIZATION



Upper plot: The surrogate model (black, solid) models the *unknown* relationship between input and output (black, dashed) based on the initial design (red points).

Lower plot: Mean and variance of the surrogate model are used to derive the expected improvement (EI) criterion. The point that maximizes the EI is proposed (green point).

# BAYESIAN OPTIMIZATION

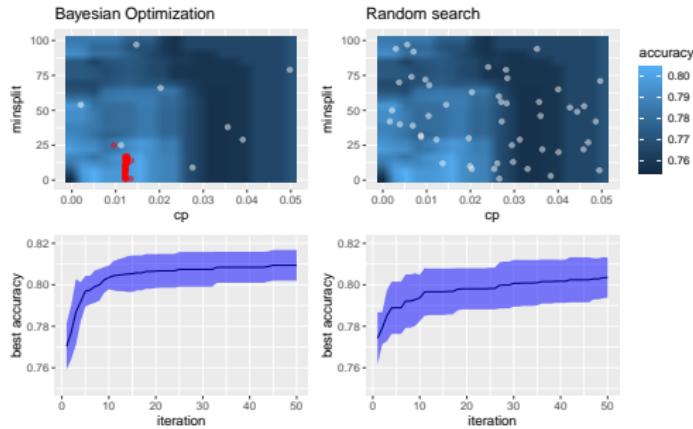


Upper plot: The surrogate model (black, solid) models the *unknown* relationship between input and output (black, dashed) based on the initial design (red points).

Lower plot: Mean and variance of the surrogate model are used to derive the expected improvement (EI) criterion. The point that maximizes the EI is proposed (green point).

# BAYESIAN OPTIMIZATION

Since we use the sequentially updated surrogate model predictions of performance to propose new configurations, we are guided to “interesting” regions of  $\Lambda$  and avoid irrelevant evaluations:



**Figure:** Tuning complexity and minimal node size for splits for CART on the titanic data (10-fold CV maximizing accuracy).

Left panel: BO, 50 configurations; right panel: random search, 50 iterations.

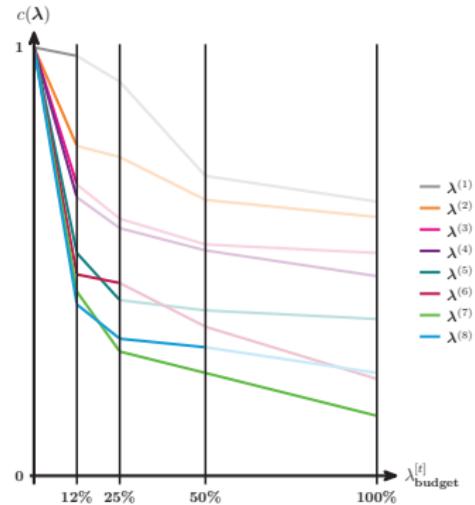
Top panel: one run (initial design of BO is white); bottom panel: mean  $\pm$  std of 10 runs.

# MULTIFIDELITY OPTIMIZATION

- Prerequisite: Fidelity HP  $\lambda_{\text{fid}}$ , i.e., a component of  $\lambda$ , which influences the computational cost of the fitting procedure in a monotonically increasing manner
- Methods of multifidelity optimization in HPO are all tuning approaches that can efficiently handle a  $\mathcal{I}$  with a HP  $\lambda_{\text{fid}}$
- The lower we set  $\lambda_{\text{fid}}$ , the more points we can explore in our search space, albeit with much less reliable information w.r.t. their true performance.
- We assume to know box-constraints of  $\lambda_{\text{fid}}$ , so  $\lambda_{\text{fid}} \in [\lambda_{\text{fid}}^{\text{low}}, \lambda_{\text{fid}}^{\text{upp}}]$ , where the upper limit implies the highest fidelity returning values closest to the true objective value at the highest computational cost.

# SUCCESSIVE HALVING

- Races down set of HPCs to the best
- Idea: Discard bad configurations early
- Train HPCs with fraction of full budget (SGD epochs, training set size); the control param for this is called **multi-fidelity HP**
- Continue with better  $1/\eta$  fraction of HPCs (w.r.t  $\widehat{GE}$ ); with  $\eta$  times budget (usually  $\eta = 2, 3$ )
- Repeat until budget depleted or single HPC remains



# MULTIFIDELITY OPTIMIZATION – HYPERBAND

## Problem with SH

- Good HPCs could be killed off too early,  
depends on evaluation schedule

## Solution: Hyperband

- Repeat SH with different start budgets  $\lambda_{\text{fid}}^{[0]}$   
and initial number of HPCs  $p^{[0]}$
- Each SH run is called bracket
- Each bracket consumes ca. the same budget

For  $\eta = 4$

| bracket 3 |                              |             |
|-----------|------------------------------|-------------|
| $t$       | $\lambda_{\text{fid}}^{[t]}$ | $p_3^{[t]}$ |
| 0         | 1                            | 82          |
| 1         | 4                            | 20          |
| 2         | 16                           | 5           |
| 3         | 64                           | 1           |

| bracket 2 |                              |             |
|-----------|------------------------------|-------------|
| $t$       | $\lambda_{\text{fid}}^{[t]}$ | $p_2^{[t]}$ |
| 0         | 4                            | 27          |
| 1         | 16                           | 6           |
| 2         | 64                           | 1           |

| bracket 1 |                              |             |
|-----------|------------------------------|-------------|
| $t$       | $\lambda_{\text{fid}}^{[t]}$ | $p_1^{[t]}$ |
| 0         | 16                           | 10          |
| 1         | 64                           | 2           |

| bracket 0 |                              |             |
|-----------|------------------------------|-------------|
| $t$       | $\lambda_{\text{fid}}^{[t]}$ | $p_0^{[t]}$ |
| 0         | 64                           | 5           |

## MORE TUNING ALGORITHMS:

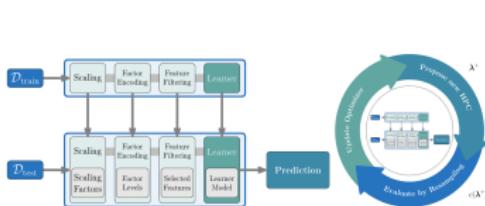
Other advanced techniques besides model-based optimization and the hyperband algorithm are:

- Stochastic local search, e.g., simulated annealing
- Genetic algorithms / CMAES
- Iterated F-Racing
- Many more ...

For more information see *Hyperparameter Optimization: Foundations, Algorithms, Best Practices and Open Challenges*, Bischl (2021)

# Introduction to Machine Learning

## Hyperparameter Tuning - Pipelines and AutoML

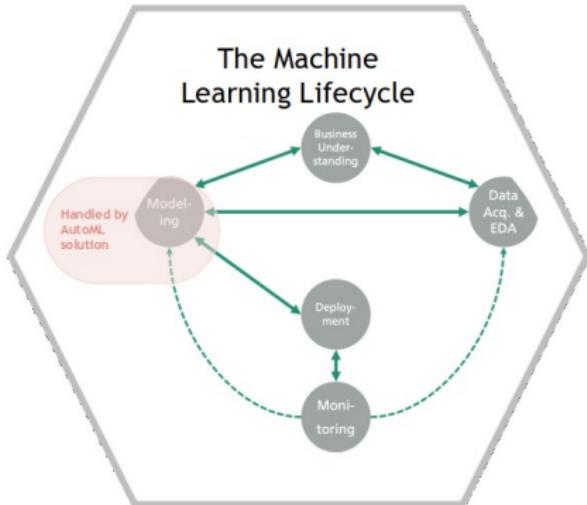


### Learning goals

- Pipelines as connected steps of learnable operations
- Sequential pipeline
- Pipelines and DAGs

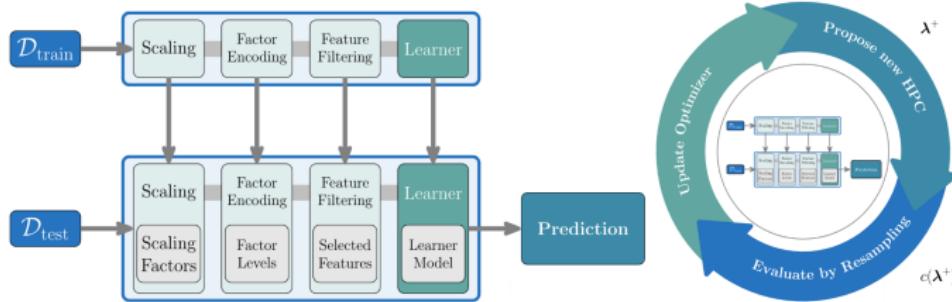
# CASE FOR AUTOML

- More and more tasks are approached via data driven methods.
- Data scientists often rely on trial-and-error.
- The process is especially tedious for similar, recurring tasks.
- Not the entire machine learning lifecycle can be automated.



# PIPELINES AND AUTOML

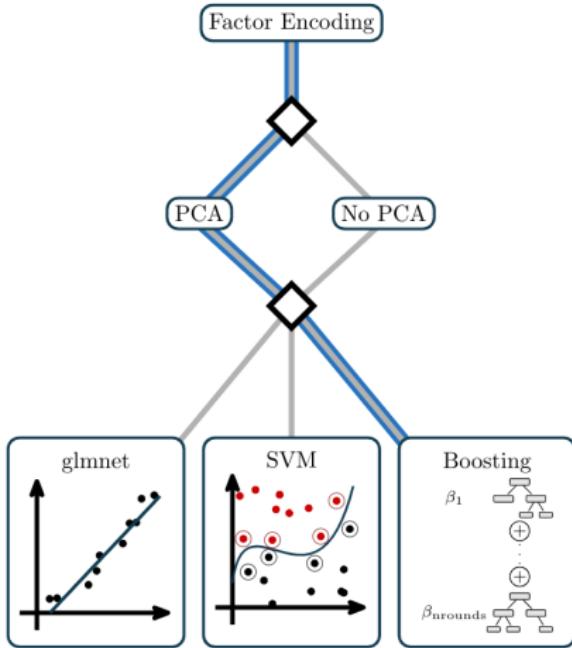
- ML typically has several data transformation steps before model fit
- If steps are in succession, data flows through sequential pipeline
- NB: Each node has a train and predict step and learns params
- And usually has HPs



**Pipelines are required to embed full model building into CV to avoid overfitting and biased evaluation!**

# PIPELINES AND AUTOML

- Further flexibility by representing pipeline as DAG
- Single source accepts  $\mathcal{D}_{\text{train}}$ , single sink returns predictions
- Each node represents a preprocessing operation, a learner, a postprocessing operation or controls data flow
- Can be used to implement ensembles, operator selection,  
...



# PIPELINES AND AUTOML

- HPs of pipeline are the joint set of all HPs of its contained nodes:

$$\tilde{\Lambda} = \tilde{\Lambda}_{\text{op},1} \times \cdots \times \tilde{\Lambda}_{\text{op},k} \times \tilde{\Lambda}_{\mathcal{I}}$$

- HP space of a DAG is more complex:  
Depending on branching / selection  
different nodes and HPs are active  
→ **hierarchical search space**

Search Space  $\tilde{\Lambda}$

| Name                  | Type | Bounds/Values            | Trafo  |
|-----------------------|------|--------------------------|--------|
| encoding              | C    | one-hot, impact          |        |
| ◊ pca                 | C    | PCA, no PCA              |        |
| ◊ learner             | C    | glmnet, SVM,<br>Boosting |        |
| <hr/>                 |      |                          |        |
| if learner = glmnet   |      |                          |        |
| s                     | R    | [−12, 12]                | $2^x$  |
| alpha                 | R    | [0, 1]                   | —      |
| <hr/>                 |      |                          |        |
| if learner = SVM      |      |                          |        |
| cost                  | R    | [−12, 12]                | $2^x$  |
| gamma                 | R    | [−12, 12]                | $2^x$  |
| <hr/>                 |      |                          |        |
| if learner = Boosting |      |                          |        |
| eta                   | R    | [−4, 0]                  | $10^x$ |
| nrounds               | I    | {1, ..., 5000}           | —      |
| max_depth             | I    | {1, ..., 20}             | —      |

A graph that includes many preprocessing steps and learner types can be flexible enough to work on a large number of data sets

**Combining such graph with an efficient tuner is key in AutoML**

# AUTOML – CHALLENGES

- Most efficient approach?
- How to integrate human a-priori knowledge?
- How can we best (computationally) transfer “experience” into AutoML? Warmstarts, learned search spaces, etc.
- Multi-Objective goals, including model interpretability
- AutoML as a process is too much of a black-box, hurts adoption.

# INTRODUCTION TO MACHINE LEARNING

ML Basics

Supervised Regression

Supervised Classification

Performance Evaluation

k-NN

Classification and Regression Trees (CART)

Random Forests

Neural Networks

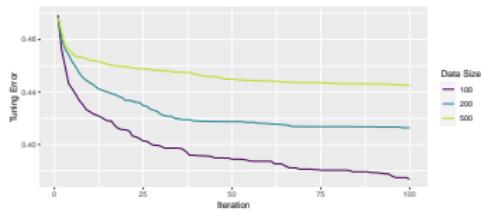
Tuning

**Nested Resampling**

# Introduction to Machine Learning

## Nested Resampling Motivation

### Learning goals



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

# MOTIVATION

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

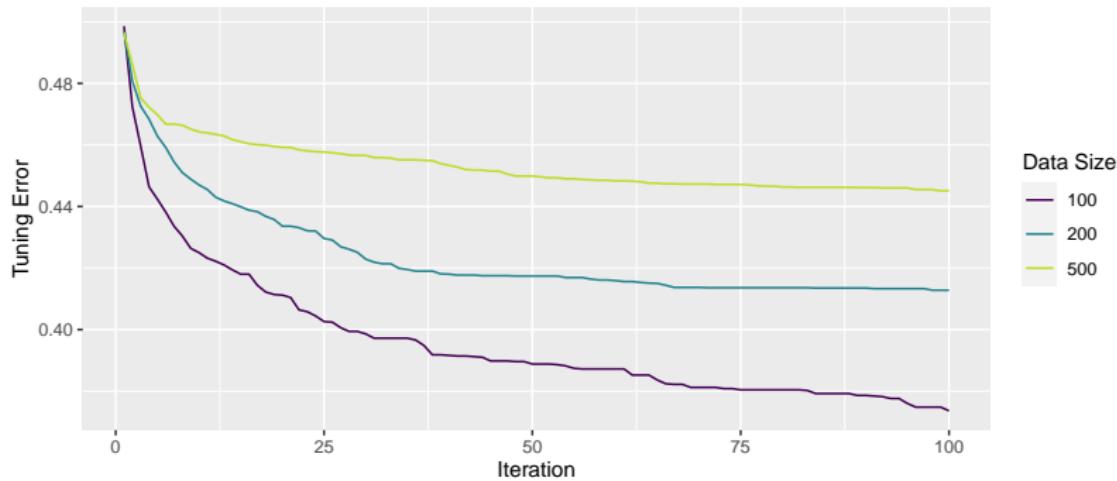
## Problem

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

# INSTRUCTIVE AND PROBLEMATIC EXAMPLE

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

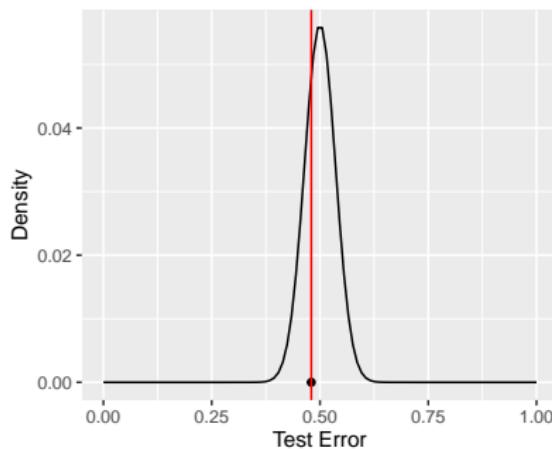
# INSTRUCTIVE AND PROBLEMATIC EXAMPLE



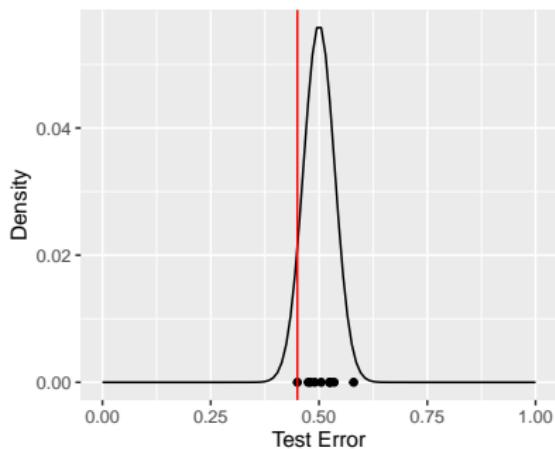
- Plotted is the best “tuning error” (i.e. the performance of the model with fixed  $\lambda$  as evaluated by the cross-validation) after  $k$  tuning iterations.
- We have performed the experiment for different sizes of learning data that were cross-validated.

# INSTRUCTIVE AND PROBLEMATIC EXAMPLE

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



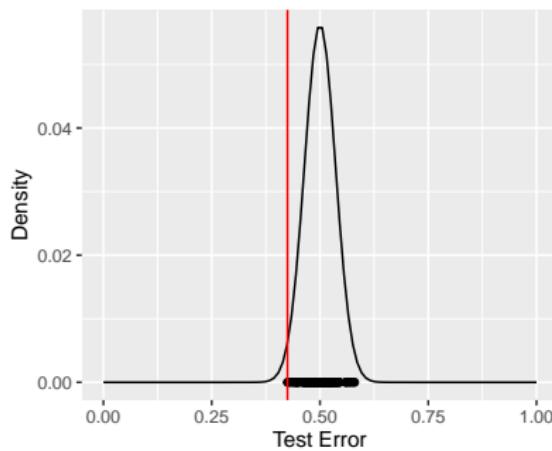
$n = 200$ ; #runs = 10; best err = 0.45



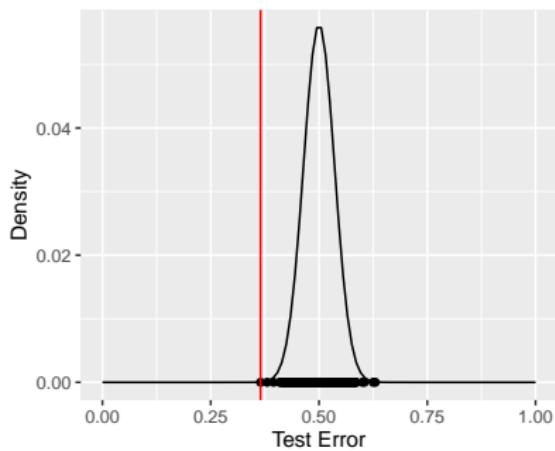
- For 1 experiment, the CV score will be nearly 0.5, as expected
- We basically sample from a (rescaled) binomial distribution when we calculate error rates
- And multiple experiment scores are also nicely arranged around the expected mean 0.5

# INSTRUCTIVE AND PROBLEMATIC EXAMPLE

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



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



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

# UNTOUCHED TEST SET PRINCIPLE

Countermeasure: simulate what actually happens in model application.

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

# UNTOUCHED TEST SET PRINCIPLE

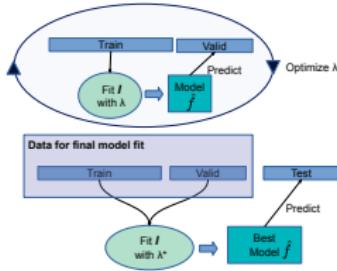
- For steps that themselves require resampling (e.g., hyperparameter tuning) this results in **nested resampling**, i.e., resampling strategies for both
  - tuning: an inner resampling loop to find what works best based on training data
  - outer evaluation on data not used for tuning to get honest estimates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

# Introduction to Machine Learning

## Training - Validation - Test

### Learning goals

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



# TUNING PROBLEM

Remember:

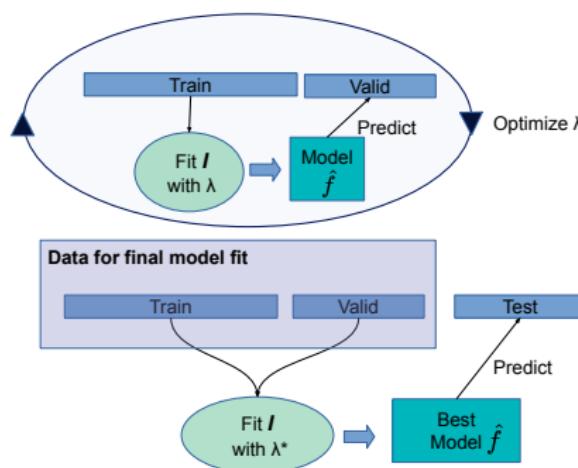
We need to

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

# TRAIN - VALIDATION - TEST

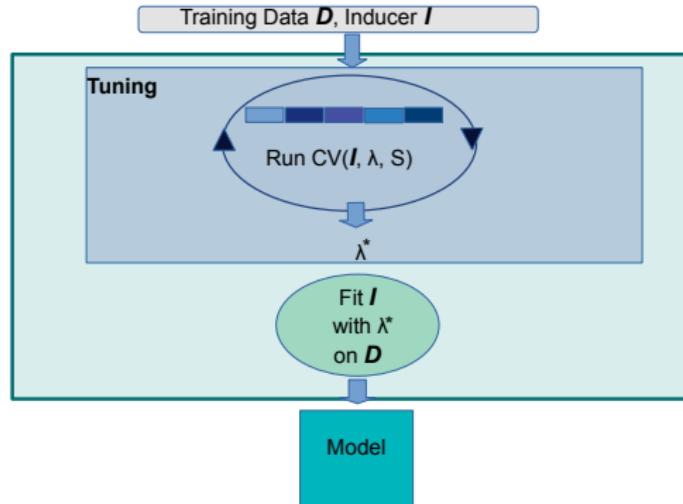
Simplest method to achieve this: a 3-way split

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



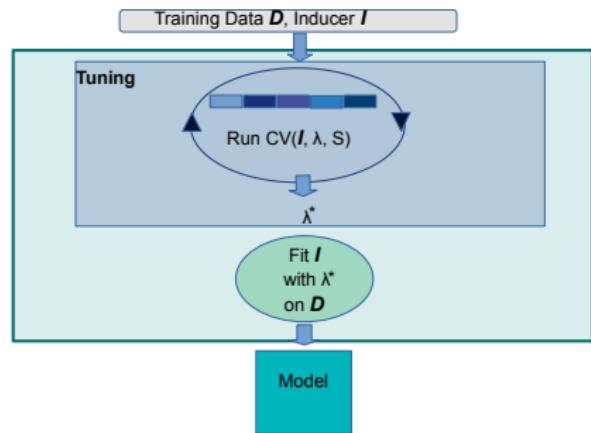
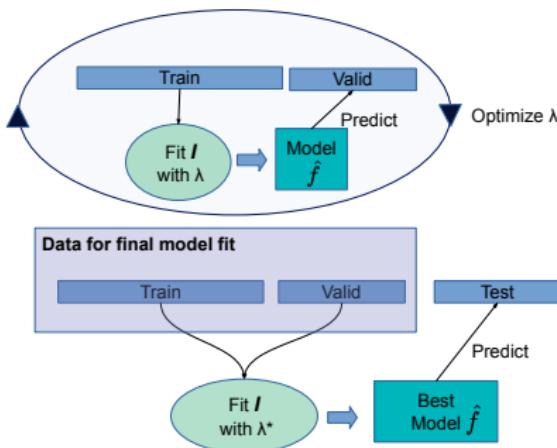
# TUNING AS PART OF MODEL BUILDING

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



# TUNING AS PART OF MODEL BUILDING

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

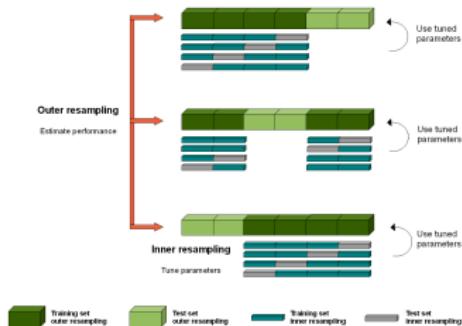


# Introduction to Machine Learning

## Nested Resampling

### Learning goals

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



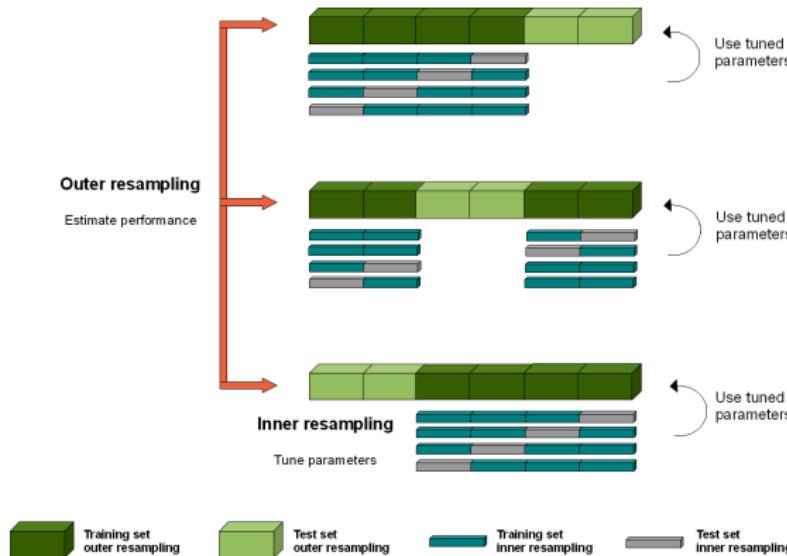
# NESTED RESAMPLING

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

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

# NESTED RESAMPLING

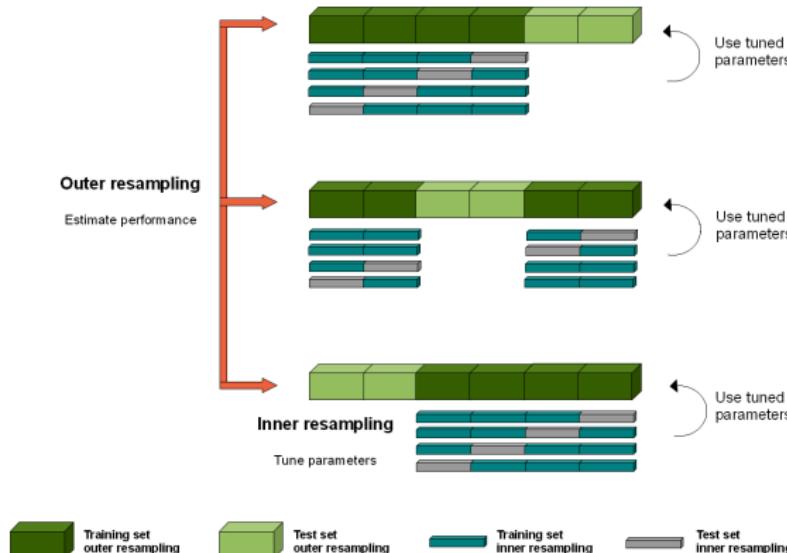
Assume we want to tune over a set of candidate HP configurations  $\lambda_i; i = 1, \dots$  with 4-fold CV in the inner resampling and 3-fold CV in the outer loop. The outer loop is visualized as the light green and dark green parts.



# NESTED RESAMPLING

In each iteration of the outer loop we:

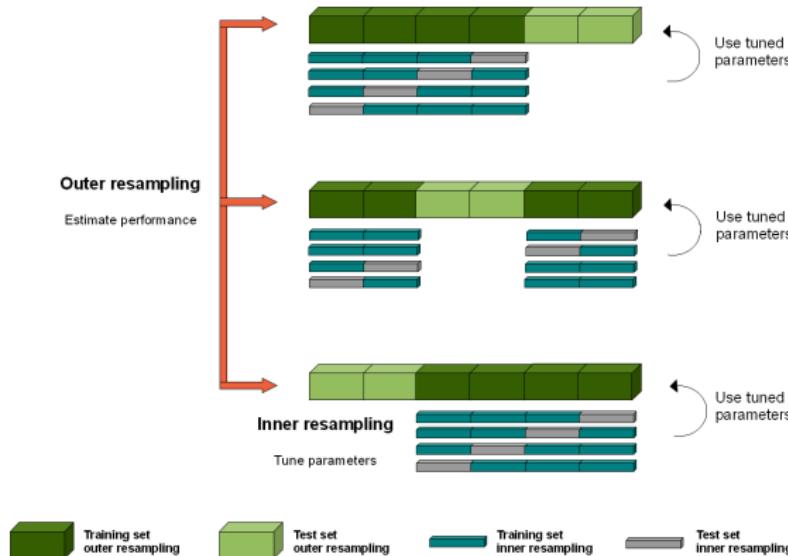
- Split off the light green testing data
- Run the tuner on the dark green part of the data, e.g., evaluate each  $\lambda_i$  through fourfold CV on the dark green part



# NESTED RESAMPLING

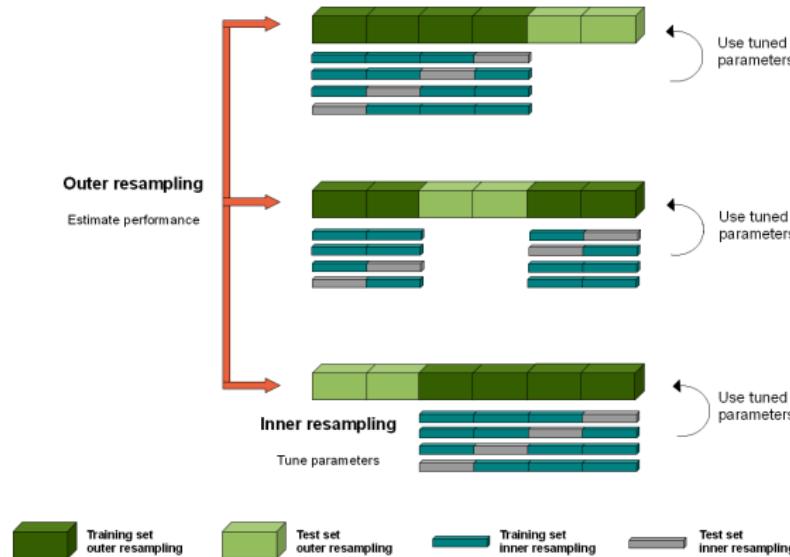
In each iteration of the outer loop we:

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



# NESTED RESAMPLING

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



# NESTED RESAMPLING - INSTRUCTIVE EXAMPLE

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

