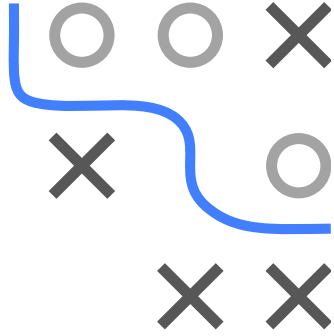
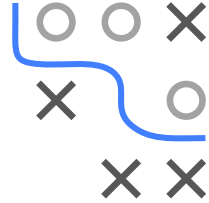


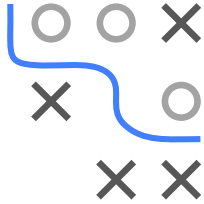


Introduction to Machine Learning

All slides

April 21, 2025

INTRODUCTION TO MACHINE LEARNING



ML Basics

Supervised Regression

Supervised Classification

Performance Evaluation

k-NN

Classification and Regression Trees (CART)

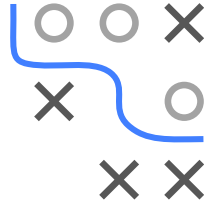
Random Forests

Neural Networks

Tuning

Nested Resampling

Introduction to Machine Learning



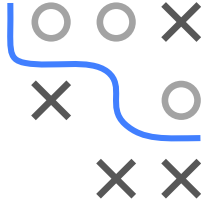
ML-Basics
What is Machine Learning?



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

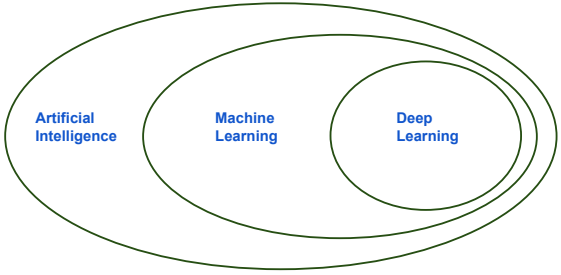
ML IS CHANGING OUR WORLD

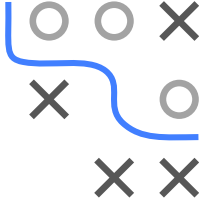


- Search engines learn your search preferences
- Recommender systems learn your taste in books, music, movies,...
- Algorithms do automatic stock trading
- Tools can accurately translate between many different languages
- DeepMind beats humans at Go
- Physicians are supported by personalized medicine
- LLMs revolutionize many fields (currently especially coding)
- Data-driven discoveries are made in physics, biology, genetics, astronomy, chemistry, neurology,...
- ...



AI, ML AND DL

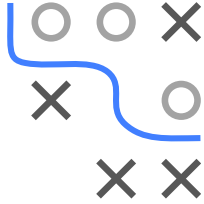




Many people are confused what these terms actually mean.

And what does all this have to do with statistics?

ARTIFICIAL INTELLIGENCE



- General term for very large and rapidly developing field.
- No strict definition, but often used when machines perform tasks that could only be solved by humans or are very difficult and assumed to require “intelligence”.
- Started in the 1940s – when the computer was invented. Turing and von Neumann immediately asked: If we can formalize computation, can we use that to formalize “thinking”?
- Includes ML, NLP, computer vision, robotics, planning, search, intelligent agents, ...
- Sometimes misused as a “hype” term for ML or ... basic data analysis.
- Or people refer to the fascinating developments in the area of foundation models



MACHINE LEARNING

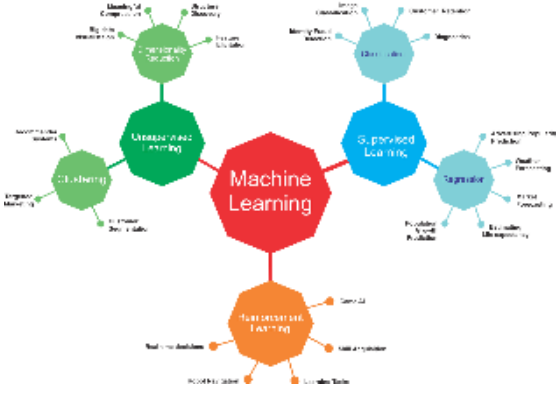
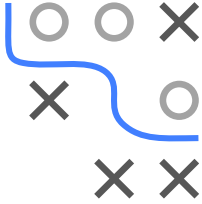


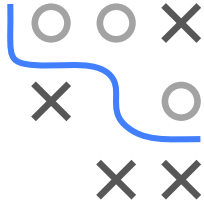
Image via `https://www.oreilly.com/library/view/java-deep-learning/9781788997454/assets/899ceaf3-c710-4675-ae99-33c76cd6ac2f.png`



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

DEEP LEARNING

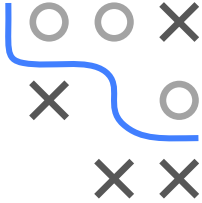
- Subfield of ML which studies neural networks.
- Artificial neural networks are roughly inspired by the human brain, but we treat them as useful, mathematical models.
- Studied for decades (start in the 1940/50s).Uses more layers, might use specific neurons, e.g., for images, many computational improvements to train on large data.
- Can be used on tabular data, but typical applications are images, texts or signals.
- Last 15-20 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

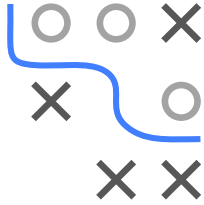


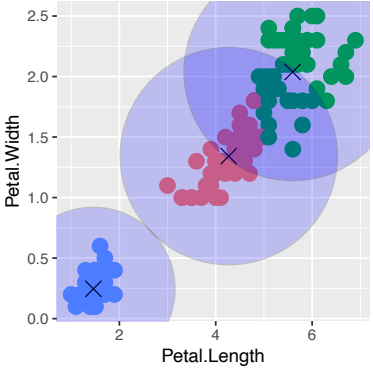
- Historically developed as different fields, but many methods and concepts are pretty much the same.
- ML: Rather accurate predictions with more complex models.
- Stats: More interpreting relationships and sound inference.
- Now: Both basically work on same problems with same tools.
- Communities are still divided.
- Often different terminology for the same concepts.
- Most parts of ML we could also call:
  Nonparametric statistics plus efficient numerical optimization.
- Personal opinion: Nowadays few practical differences, seeing differences instead of commonalities mainly holds you back



UNSUPERVISED LEARNING

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

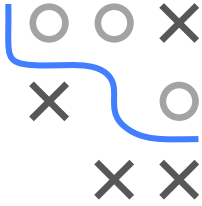


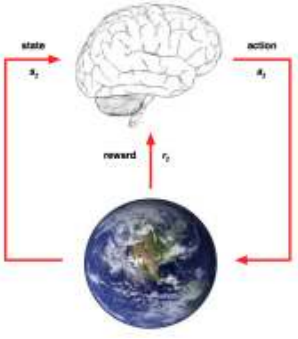


- Dimensionality reduction (PCA, autoencoders ...); compress information in $\mathcal{X}$
- Clustering: group similar observations
- Outlier detection, anomaly detection
- Association rules

REINFORCEMENT LEARNING

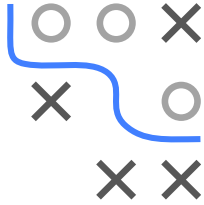
General-purpose framework. At each time step an *agent* interacts with an *environment*. It: observes state; receives reward; executes action.





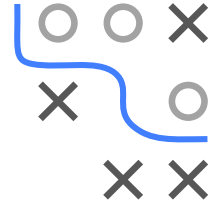
- Goal: Select actions to maximize future reward.
- Reward signals may be sparse, noisy and delayed.

WHAT COMES NEXT

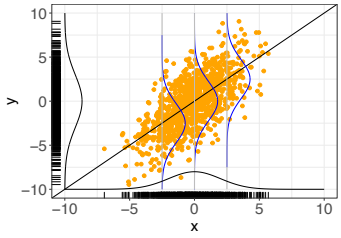


- **Supervised learning** for regression and classification: predict labels y through features $\mathbf{x}$, based on training data.
- First, we will go through fundamental concepts in supervised ML:
  - What kind of “data” do we learn from?
  - What is a “prediction model”?
  - How can we quantify “predictive performance”?
  - What is a “learning algorithm”
  - How can we operationalize learning?
- We will also introduce first concrete learning algorithms: Linear models, trees and forests.
- More complex stuff comes later.

Introduction to Machine Learning



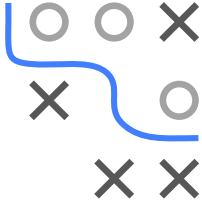
ML-Basics
Data



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 statistician Fisher, frequently used toy example.

- 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]

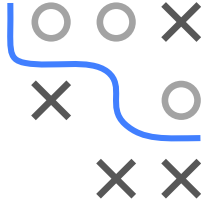


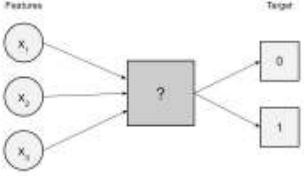
Source: `https://rpubs.com/vidhividhi/irisdataeda`

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

DATA IN SL

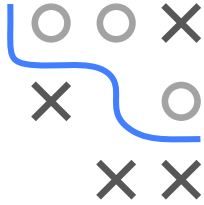
- Measurements on different aspects of *n* objects:
  - **Target**: output variable / goal of prediction
  - **Features**: properties that describe the object
- We assume some relationship between features and target, and want to discover that predictive rule.





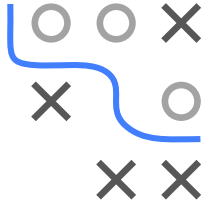
Features x				Target y
Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
4.3	3.0	1.1	0.1	setosa
5.0	3.3	1.4	0.2	setosa
7.7	3.8	6.7	2.2	virginica
5.5	2.5	4.0	1.3	versicolor

DATA TYPES



- Features and target may be of different data types
  - **Numerical** variables: $\in \mathbb{R}$
  - **Integer** variables: $\in \mathbb{Z}$
  - **Categorical** variables: $\in \{C_1, ..., C_g\}$
  - **Binary** variables: $\in \{0, 1\}$
- 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., trees can use categoricals without problems). For other 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.

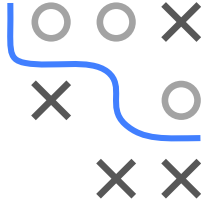
ENCODING FOR CATEGORICALS



- We expand the representation of a variable 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: $\boldsymbol{o}(x)_j = \mathbb{I}(x = j) \in \{0, 1\}$.
- Each entry of $\boldsymbol{o}(x) \in \{0, 1\}^{\tilde{k}}$ 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., $x \in \{a, b, c\} \mapsto \boldsymbol{o}(x) = (x_a, x_b, x_c)$, with $x_a = x_b = 0, x_c = 1$ and $\boldsymbol{o}(x) = (0, 0, 1)$ for $x = c$.
  - **Dummy encoding**: $\tilde{k} = k - 1$ dummies, so *at most one* element is 1, removing the redundancy of one-hot encoding (necessary for learners that require non-singular input matrices, such as in linear regression). E.g., $x \in \{a, b, c\} \mapsto \boldsymbol{o}(x) = (x_a, x_b)$ for reference category c, with $x_a = x_b = 0$ and $\boldsymbol{o}(x) = (0, 0)$ for $x = c$.
- We also often use this encoding for categorical targets.
- 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

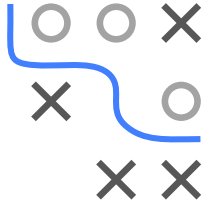
- We call the entries of the target column **labels**.
- We distinguish two types of data:
  - **Labeled** data: target is observed
  - **Unlabeled** data: target is unknown



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

NOTATION FOR DATA

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



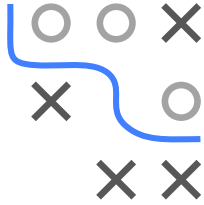
- $\mathcal{X}$: input space, usually $\mathcal{X} \subset \mathbb{R}^p$
- $\mathcal{Y}$: output / target space
- $\left(\mathbf{x}^{(i)}, y^{(i)} \right) \in \mathcal{X} \times \mathcal{Y}$: i-th observation
- $\mathbf{x}_j = \left(x_j^{(1)}, \ldots, x_j^{(n)} \right)^\top$: j-th feature vector

We also use

- $(\mathcal{X} \times \mathcal{Y})^n$, 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 $\mathbb{D}$.

$\Rightarrow$ We observe n objects, described by p features.

DATA-GENERATING PROCESS

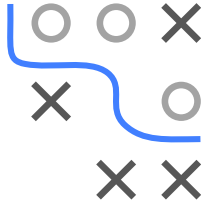


- We assume the observed data $\mathcal{D}$ to be generated by a process that can be characterized by some probability distribution

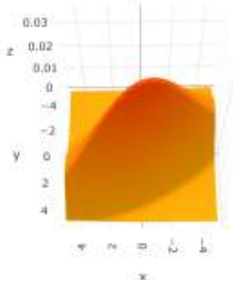
$$\mathbb{P}_{xy},$$

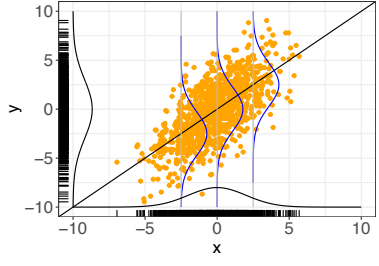
  defined on $\mathcal{X} \times \mathcal{Y}$.
- We denote the random variables following this distribution by lowercase **x** and y.
- We usually assume the true distribution to be **unknown**. Learning (part of) its structure to predict y is what ML is all about.

DATA-GENERATING PROCESS / 2



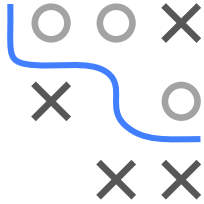
- We assume data to be drawn *i.i.d.* (**i**ndependent and **i**dentically **d**istributed) from $\mathbb{P}_{xy}$.
- This means: We assume that all samples are drawn from the same distribution and are mutually independent – the i-th realization does not depend on the other $n - 1$ ones.
- This is a strong assumption, not always realistic. More complex scenarios (e.g., time series) exist, but we focus on the i.i.d. scenario here.





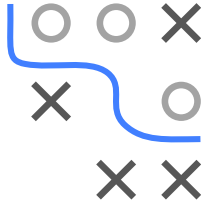
DATA-GENERATING PROCESS / 3

Remarks:

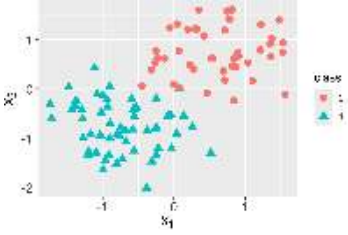


- With a slight abuse of notation we write random variables, e.g., $\mathbf{x}$ and y, in lowercase, as normal variables or function arguments. The context will make clear what is meant.
- Often, distributions are characterized by a parameter vector $\boldsymbol{\theta} \in \Theta$. We then write $p(\mathbf{x},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

Introduction to Machine Learning



ML-Basics
Supervised Tasks

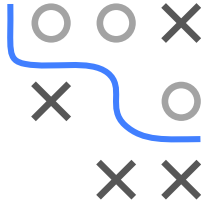


Learning goals

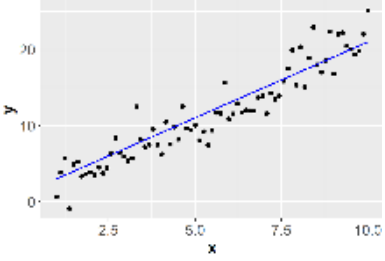
- Know definition and examples of supervised tasks
- Understand the difference between regression and classification

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: $\mathcal{Y} \subseteq \mathbb{R}$



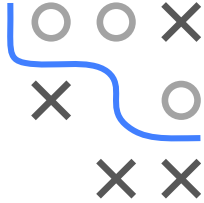
Classification: $\mathcal{Y} = \{C_1, ..., C_g\}$



PREDICT VS. EXPLAIN

We can distinguish two main reasons to learn this relationship:

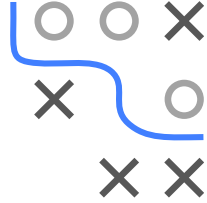
- **Learning to predict**. Here, usually, we don't care how our model is structured or whether we can understand it.
  We want an accurate predictor for new data.
- **Learning to explain**. Here, our model is only a means to a better understanding of the inherent relationship in the data.
  We might not use the learned model on new observations, but rather discuss its implications, in a scientific or social context. Of course, a model must "match" the data, we usually still measure this via predictive accuracy.



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



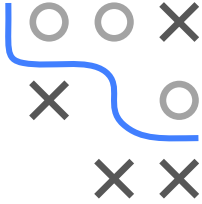
Features x				Target y
square footage of the house	number of bedrooms	swimming pool (yes/no)	...	house price in USD
1150	3	0	.	221 900
2570	3	1	.	555,000
770	2	0	.	100,000
1960	4	1	.	614,000



Rather *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



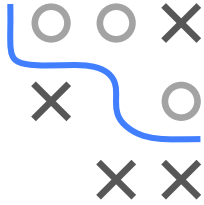
Features x					Target y
diagnosis category	admission type	gender	age	..	Length-of-stay in the hospital in days
heart disease	elective	male	75	..	48
injury	emergency	male	22	..	20
psychosis	newborn	female	0	..	8
pneumonia	urgent	female	67	..	55



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

CLASSIFICATION EXAMPLE: RISK CATEGORY

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

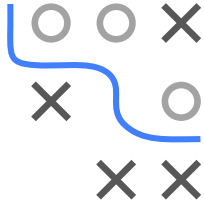


Features x				Target y
job type	age	smoker	...	risk group
company car	34	1	...	3
alcoholic sport	42	2	...	5
musician	23	1	...	1
white collar worker	40	0	...	2

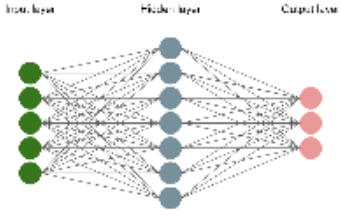


Probably *learn to predict*, but the company might be required to explain its predictions to its customers.

Introduction to Machine Learning



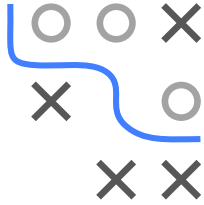
ML-Basics
Models & Parameters



Learning goals

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

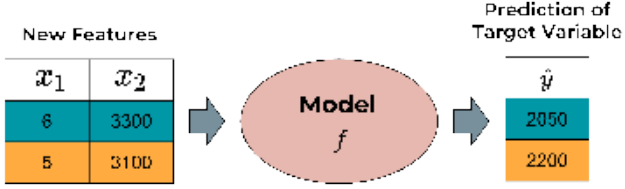
WHAT IS A MODEL?



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

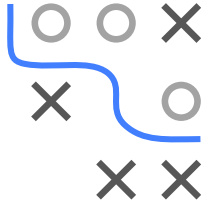
$$f : \mathcal{X} \to \mathbb{R}^g$$

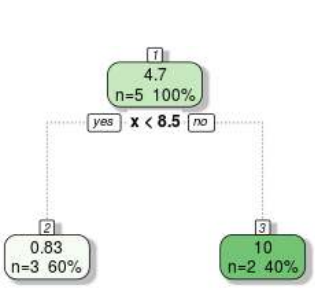
  is a function that maps feature vectors to predicted target values.
- In regression: $g = 1$; in classification, g is the number of classes, and output vectors are scores or class probabilities.

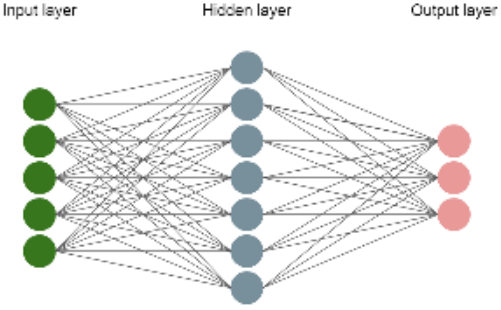


WHAT IS A MODEL? / 2

- f is meant to capture intrinsic patterns of the data, the underlying assumption being that these hold true for *all* data drawn from $\mathbb{P}_{xy}$.
- Models can range from super simple (e.g., linear, tree stumps) to very complex (e.g., DL). There are a lot of choices.





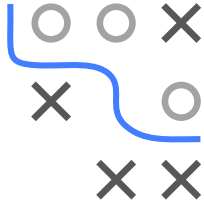


- ML requires **constraining** f to a certain type of functions.

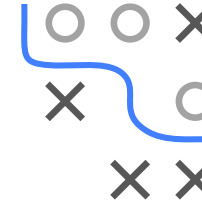
HYPOTHESIS SPACES

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

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



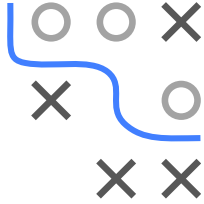
PARAMETRIZATION



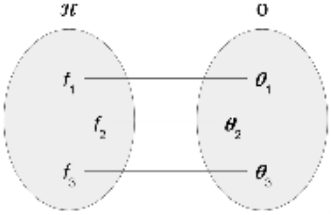
- All models within a hypothesis space share a common functional structure. We usually construct the space as **parametrized family of functions**.
- We collect all parameters in a **parameter vector** $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_d)$ from **parameter space** Θ.
- They are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

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

PARAMETRIZATION / 2



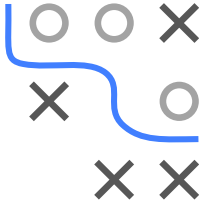
- Finding optimal model = finding optimal parameters.
- This allows us to operationalize our search for the best model as a search for the optimal value on a d-dimensional parameter surface.

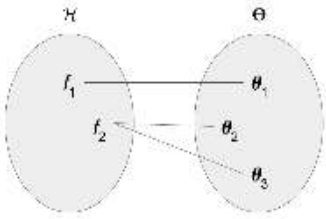


- θ might be scalar or very high-dimensional with thousands of parameters, depending on the complexity of our model.

PARAMETRIZATION / 3

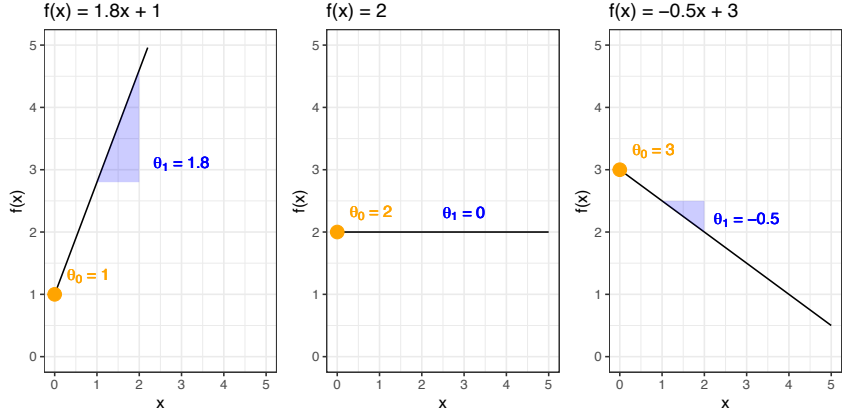
- Some parameter vectors, for some model classes, encode the same function; i.e., the parameter-to-model mapping could be non-injective.
- We call this a non-identifiable model.
- This shall not concern us here.





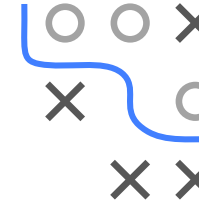
EXAMPLE: UNIVARIATE LINEAR FUNCTIONS

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

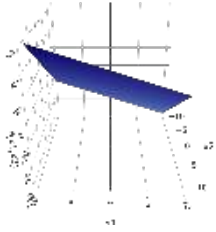


EXAMPLE: BIVARIATE QUADRATIC FUNCTIONS

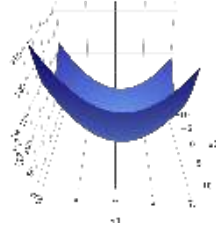
$$\mathcal{H} = \{f : f(\mathbf{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2, \boldsymbol{\theta} \in \mathbb{R}^6\},$$



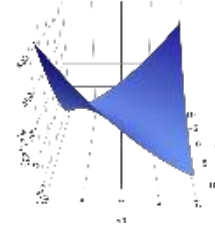
$f(x) = 3 + 2x_1 + 4x_2$



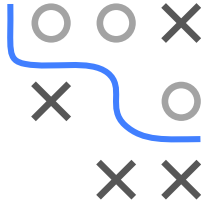
$f(x) = 3 + 2x_1 + 4x_2 + 1x_1^2 + 1x_2^2$



$f(x) = 3 + 2x_1 + 4x_2 + 1x_1^2 + 1x_2^2 + 4x_1x_2$



EXAMPLE: RBF NETWORK



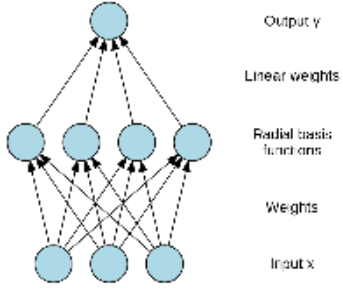
Radial basis function networks with Gaussian basis functions

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

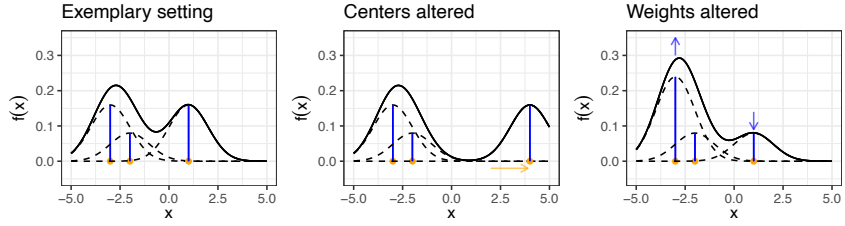
where

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

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



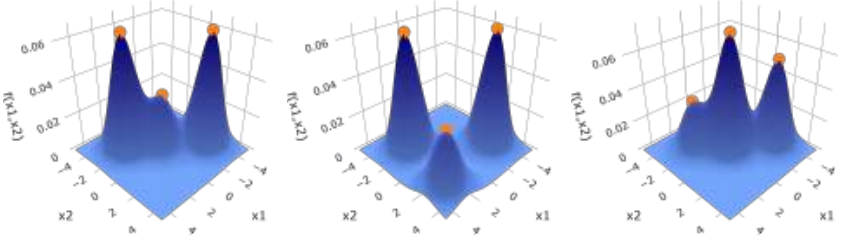
EXAMPLE: RBF NETWORK / 2



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

$a_1 = 0.4, a_2 = 0.2, a_3 = 0.4$
$c_1 = -3, c_2 = -2, c_3 = 4$

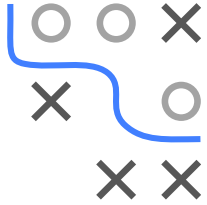
$a_1 = 0.6, a_2 = 0.2, a_3 = 0.2$
$c_1 = -3, c_2 = -2, c_3 = 1$



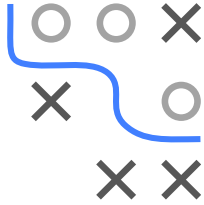
$a_1 = 0.4, a_2 = 0.2, a_3 = 0.4$
$c_1 = (2, -2), c_2 = (0, 0),$
$c_3 = (-3, 2)$

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

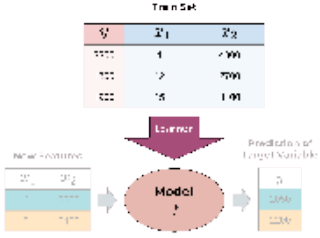
$a_1 = 0.2, a_2 = 0.45, a_3 = 0.35$
$c_1 = (2, -2), c_2 = (0, 0),$
$c_3 = (-3, 2)$



Introduction to Machine Learning



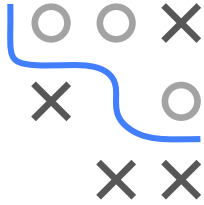
ML-Basics
Learner



Learning goals

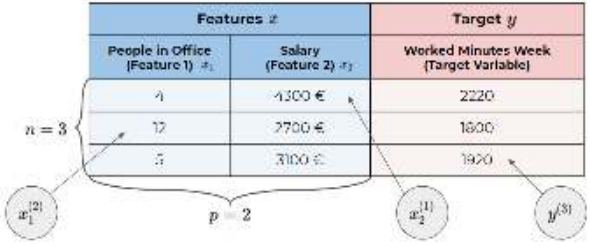
- Know formal definition of learner
- Understand that a learner receives training data and outputs the best model from $\mathcal{H}$

SUPERVISED LEARNING EXAMPLE



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

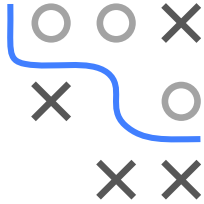
- It is a **regression** task since the target *productivity* is continuous.
- We collect data about worked minutes per week (*productivity*), how many people work in the same office as the employee in question, and the employee's salary.



SUPERVISED LEARNING EXAMPLE / 2

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:

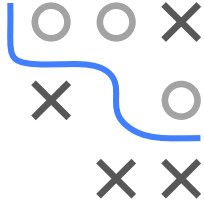


- The baseline productivity of an employee with salary 3000 and 7 people in the office is 1850 minutes.
- A decrease of 1 person in the office increases productivity by 30.
- An increase of the salary by 100 increases productivity by 10.

$\Rightarrow$ Obviously, this is neither feasible nor leads to a good model.

IDEA OF SUPERVISED LEARNING

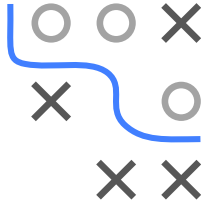
Goal: Identify the functional relationship that maps features to target.



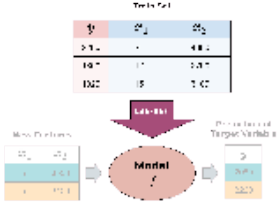
- **Supervised** learning means we use *labeled* data to learn model f.
- Later, we use model f to predict y for new *unlabeled* data.



LEARNER DEFINITION



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



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

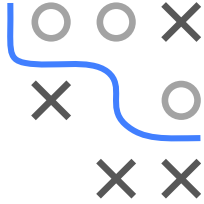
- Practically, we often construct a mapping

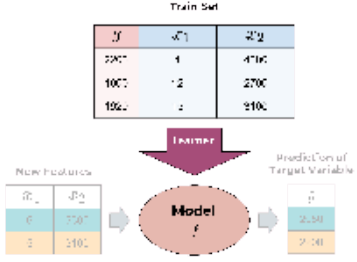
$$\mathcal{I} : \mathbb{D} \times \Lambda \rightarrow \Theta$$

LEARNER DEFINITION / 2

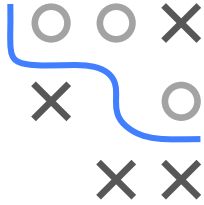
In pseudo-code:

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

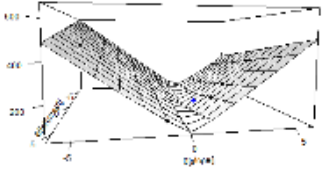




Introduction to Machine Learning



ML-Basics
Losses & Risk Minimization

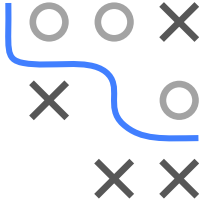


Learning goals

- Know concept of loss function
- Understand concept of theoretical and empirical risk
- Understand relationship between risk minimization and finding best model

HOW TO EVALUATE MODELS

- Training a learner = optimize over hypothesis space
- Find function that matches training data best
- We compare point-wise predicted outputs to observed labels

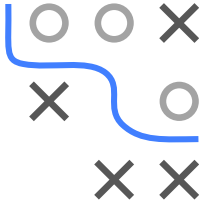


Features x		Target y		Prediction $\hat{y}$
People In Office (Feature 1) x_1	Salary (Feature 2) x_2	Worked Minutes Week (Target Variable)	$\overset{?}{\approx}$	Worked Minutes Week (Target Variable)
4	4300 €	2220		2388
12	7700 €	1800		1644
5	3100 €	1920		1870

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

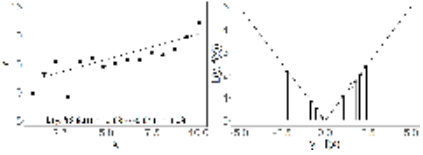
LOSS

Loss function $L(y, f(\mathbf{x}))$ quantifies point-wise how we measure errors in predictions for a single $\mathbf{x}$:

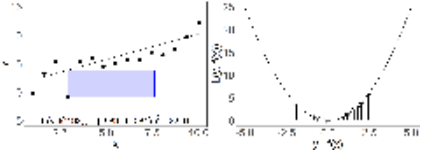


$$L : \mathcal{Y} \times \mathbb{R}^g \to \mathbb{R}.$$

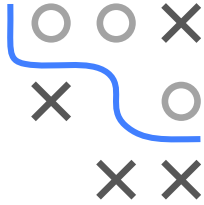
Regression: Could use absolute L1 loss $L(y, f(\mathbf{x})) = |y - f(\mathbf{x})|$



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



RISK OF A MODEL



- Theoretical **risk** of a candidate model f is the **expected loss**

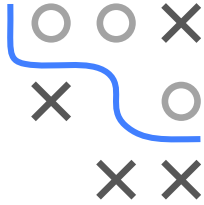
$$\mathcal{R}(f) := \mathbb{E}_{xy}[L(y, f(\mathbf{x}))] = \int L(y, f(\mathbf{x})) \, \mathrm{d}\mathbb{P}_{xy}$$

- Average error we incur when we use f on data from $\mathbb{P}_{xy}$
- Goal in ML: Find a hypothesis $f \in \mathcal{H}$ that **minimizes** this

Problems:

- $\mathbb{P}_{xy}$ is unknown
- Could estimate $\mathbb{P}_{xy}$ non-parametrically, e.g., by kernel density estimation, doesn't scale to higher dimensions
- Could efficiently estimate $\mathbb{P}_{xy}$, if we place assumptions on its form, e.g. cf. discriminant analysis

EMPIRICAL RISK

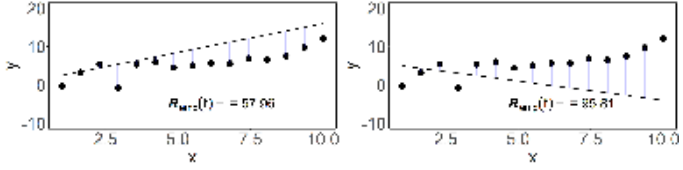


- Have n i.i.d. data from $\mathbb{P}_{xy}$, approximate expected risk empirically
- Just sum up all losses over training data

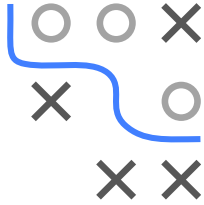
$$\mathcal{R}_{\text{emp}}(f) = \sum_{i=1}^{n} L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right)$$

- Associates one quality score with each $f \in \mathcal{H}$
- Encodes: How well does f fits training data
- Now we get very close to solve this by optimization

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



EMPIRICAL RISK / 2



- Can also define as average loss

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

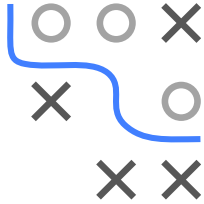
- Constant factor $\frac{1}{n}$ doesn't make a difference in optimization
- We usually use $\mathcal{R}_{\text{emp}}(f)$
- Since f is usually defined by **parameters** $\boldsymbol{\theta}$, this becomes:

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

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

EMPIRICAL RISK MINIMIZATION

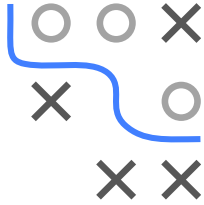
- Best model = smallest risk
- For finite $\mathcal{H}$: we could tabulate exhaustively



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

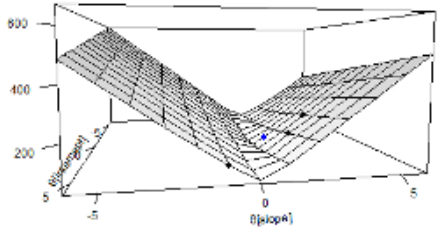
EMPIRICAL RISK MINIMIZATION

- But usually $\mathcal{H}$ is infinitely large
- Instead: Simply consider risk surface w.r.t. the parameters $\boldsymbol{\theta}$



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

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

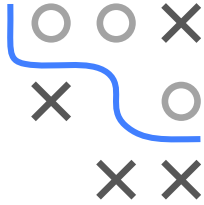


EMPIRICAL RISK MINIMIZATION / 2

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

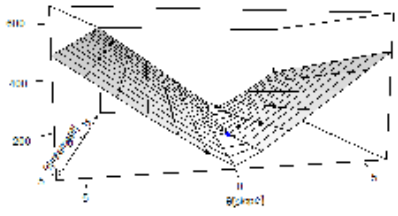
$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta} \in \Theta} \mathcal{R}_{\text{emp}}(\boldsymbol{\theta})$$

Usually we do this by numerical optimization



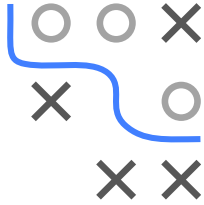
$$\mathcal{R}_{\text{emp}} : \mathbb{R}^d \to \mathbb{R}$$

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

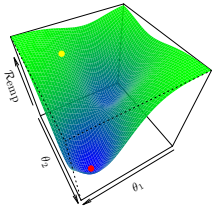


Kind of: Reduced "learning" to **numerical parameter optimization**
(Later we will learn that this is only part of the complete picture!)

Introduction to Machine Learning



ML-Basics
Optimization

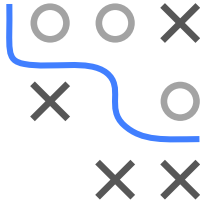


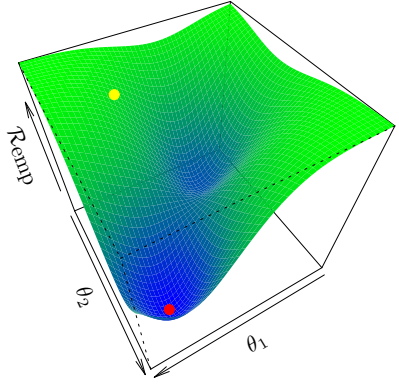
Learning goals

- 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

- Operationalize search for model f that matches training data best by looking for parametrization $\boldsymbol{\theta} \in \Theta$ with lowest risk $\mathcal{R}_{\text{emp}}(\boldsymbol{\theta})$.
- Traverse error surface downwards; often local search from some start point to minimum (hopefully).

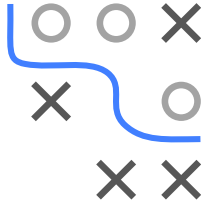




LEARNING AS PARAMETER OPTIMIZATION / 2

ERM optimization problem:

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



For **(global) minimum** $\hat{\boldsymbol{\theta}}$:

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

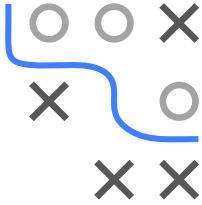
Does not imply that $\hat{\boldsymbol{\theta}}$ is unique.

- Best numerical optimizer depends on problem structure
- Continuous params? Uni-modal $\mathcal{R}_{\text{emp}}(\boldsymbol{\theta})$?
- Numerical optimization not our focus here, now

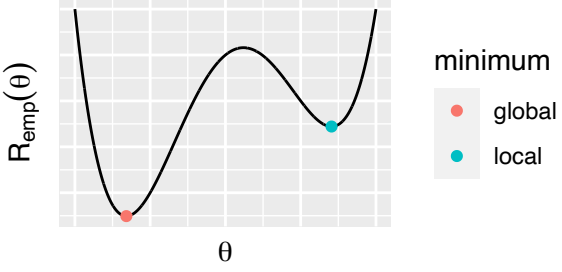
LOCAL MINIMA

Definition of **local minimum** $\hat{\boldsymbol{\theta}}$:

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



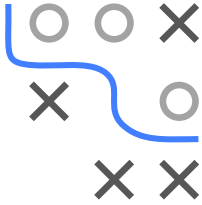
Clearly every global minimum is also a local minimum.
Finding local minimum is easier than global one.



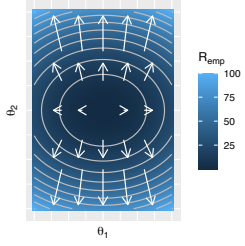
LOCAL MINIMA AND STATIONARY POINTS

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

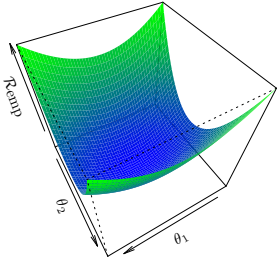
$$\frac{d\mathcal{R}_{\text{emp}}}{d\boldsymbol{\theta}}(\hat{\boldsymbol{\theta}}) = 0$$



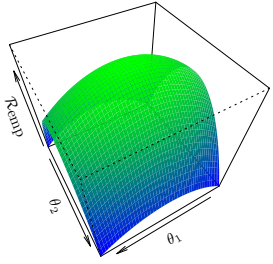
and Hessian at $\hat{\boldsymbol{\theta}}$ is positive definite.
Neg. gradient points into direction of fastest local decrease;
Hessian measures local curvature.



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



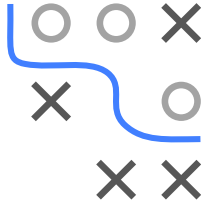
const. pos. def. Hessian



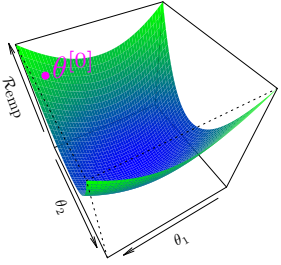
const. neg. def. Hessian

GRADIENT DESCENT

- Iteratively improve current candidate $\theta^{[t]}$
- Move in direction of neg. gradient, so direction of steepest descent
- Use step size / learning rate α

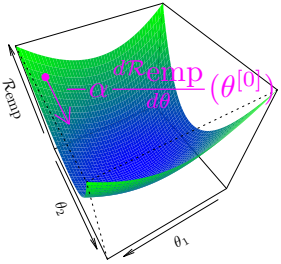


$$\theta^{[t+1]} = \theta^{[t]} - \alpha \frac{d\mathcal{R}_{\text{emp}}}{d\theta}(\theta^{[t]})$$

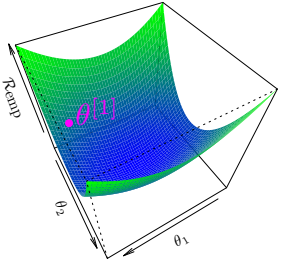


Random start $\theta^{[0]}$ with $\mathcal{R}_{\text{emp}}(\theta^{[0]}) = 76.25$.

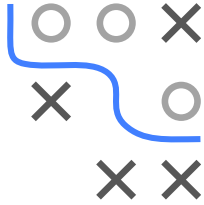
GRADIENT DESCENT - EXAMPLE



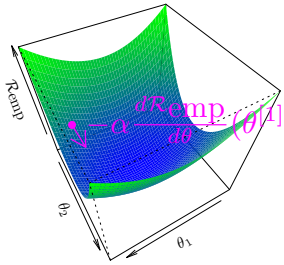
Direction of the neg. gradient at $\boldsymbol{\theta}^{[0]}$



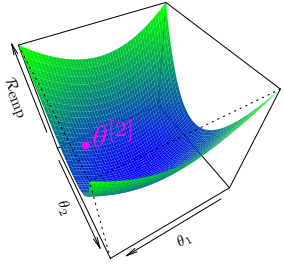
Arrive at $\boldsymbol{\theta}^{[1]}$ with $\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}^{[1]}) \approx 42.73$.
We improved:
$\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}^{[1]}) < \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}^{[0]})$



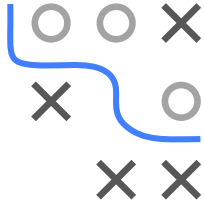
GRADIENT DESCENT - EXAMPLE



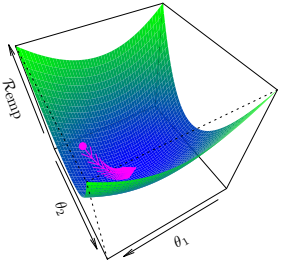
Now iterate, do the same at $\theta^{[1]}$



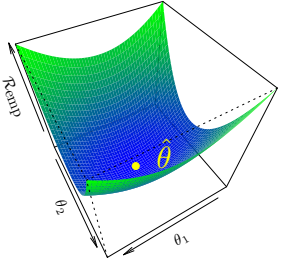
Now $\theta^{[2]}$ has risk $\mathcal{R}_{\text{emp}}(\theta^{[2]}) \approx 25.08$



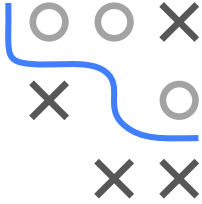
GRADIENT DESCENT - EXAMPLE



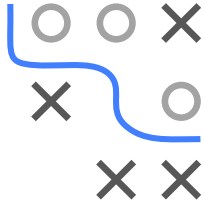
We iterate this until some form of convergence or termination



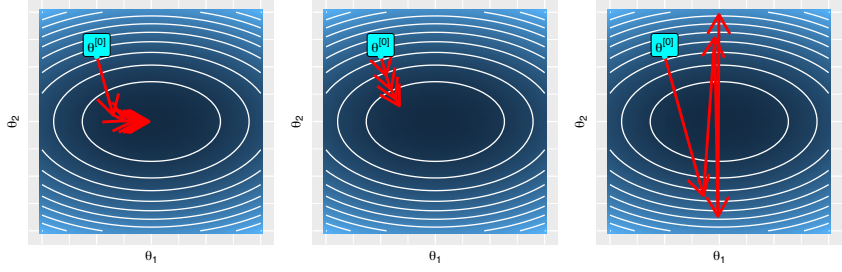
We arrive close to a stationary $\hat{\boldsymbol{\theta}}$ which is hopefully at least a local minimum



GRADIENT DESCENT - LEARNING RATE



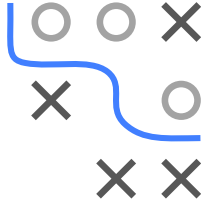
- Neg. gradient is direction that looks locally promising to reduce $\mathcal{R}_{\text{emp}}$.
- Hence weights components higher in which $\mathcal{R}_{\text{emp}}$ decreases more.
- However, the length of $-\frac{d}{d\boldsymbol{\theta}}\mathcal{R}_{\text{emp}}$ measures only the local decrease rate,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



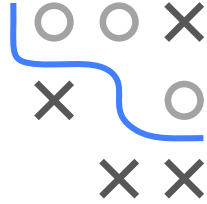
good convergence for α_1 slow convergence for $\alpha_2 (< \alpha_1)$ no convergence for $\alpha_3 (> \alpha_1)$

FURTHER TOPICS

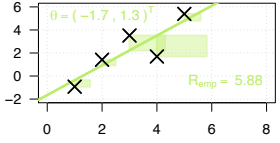
- Few models, e.g. linear regression, can be optimized analytically.
- GD is a so-called first-order method. Second-order methods (like Newton-Raphson) use the Hessian to refine the search direction for faster convergence.
- There exist many improvements of GD (momentum, ADAM), e.g., to smartly control the learn rate, to escape saddle points, to mimic second order behavior without computing the Hessian.
- If the gradient is not computed on the complete data, but instead on small, random batches, this is **stochastic gradient descent** (SGD). For large-scale problems, this is usually more efficient.



Introduction to Machine Learning



ML-Basics
Components of Supervised Learning

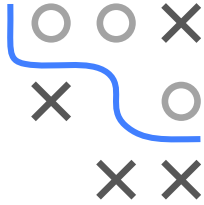


Learning goals

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

COMPONENTS OF SUPERVISED LEARNING

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

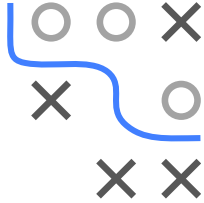


Learning = Hypothesis Space + Risk + Optimization

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

COMPONENTS OF SUPERVISED LEARNING

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

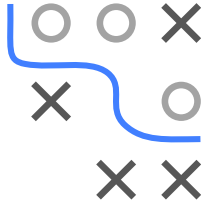


Learning = Hypothesis Space + Risk + Optim
Learning = Hypothesis Space + Loss (+ Regularization) + Optim

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

VARIETY OF LEARNING COMPONENTS

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

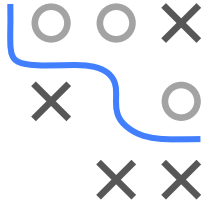


Hypothesis Space :
- Step functions
- Linear functions
- Sets of rules
- Neural networks
- Voronoi tesselations
- ...

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

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

SUPERVISED LEARNING, FORMALIZED



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

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

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

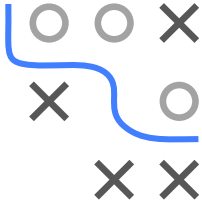
So the inducer mapping (including hyperparameters $\boldsymbol{\lambda} \in \Lambda$) is:

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

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

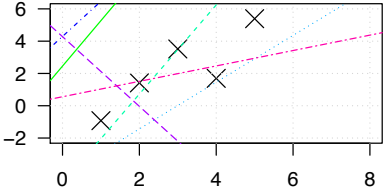
$$\mathcal{I} : \mathbb{D} \times \Lambda \to \Theta$$

EXAMPLE: LINEAR REGRESSION ON 1D



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

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

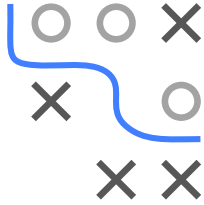


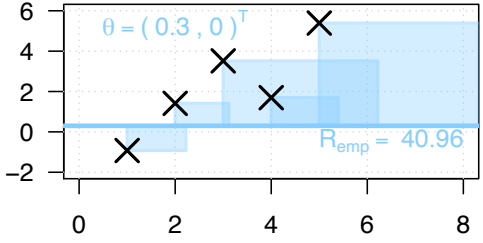
Design choice: We could add more flexibility by allowing polynomial effects or by using a spline basis.

EXAMPLE: LINEAR REGRESSION ON 1D / 2

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

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

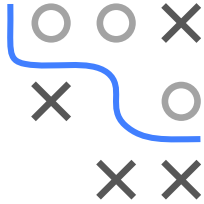


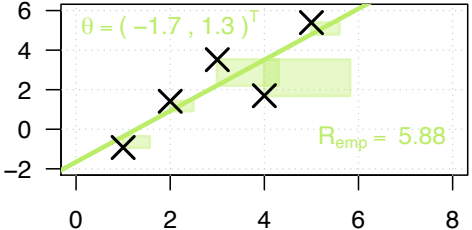


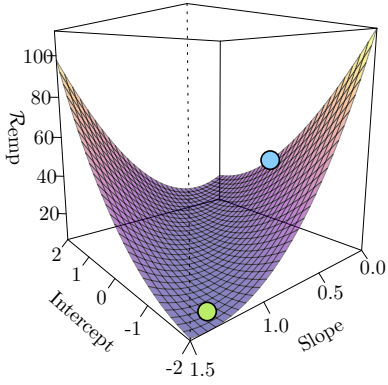
Design choice: Use absolute error / the $L1$ loss to create a more robust model which is less sensitive regarding outliers.

EXAMPLE: LINEAR REGRESSION ON 1D / 3

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



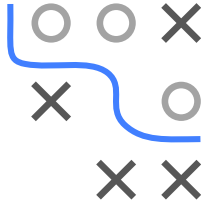




Design choice: We could use stochastic gradient descent to scale better to very large or out-of-memory data.

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

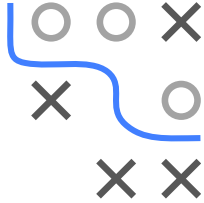
Classification and Regression Trees (CART)

Random Forests

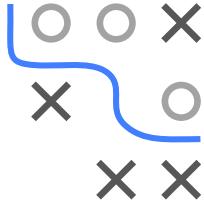
Neural Networks

Tuning

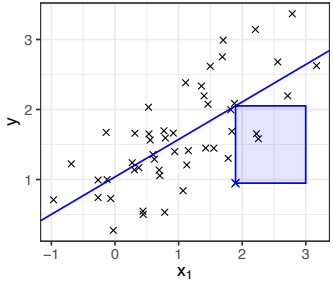
Nested Resampling



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

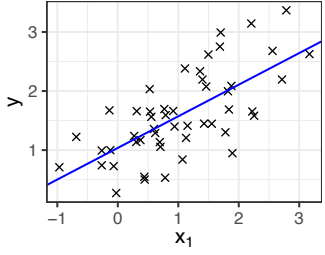
LINEAR REGRESSION

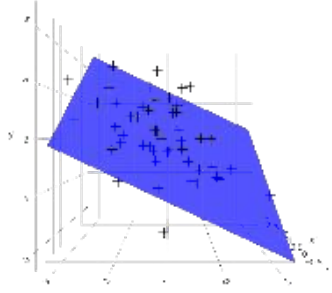
- Idea: predict $y \in \mathbb{R}$ as **linear** combination of features[1]:

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

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

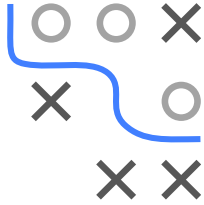
- Hypothesis space: $\mathcal{H} = \{f(\mathbf{x}) = \boldsymbol{\theta}^\top \mathbf{x} \mid \boldsymbol{\theta} \in \mathbb{R}^{p+1}\}$





[1] Actually special case of linear model, which is linear combo of *basis functions* of features $\rightsquigarrow$ Polynomial Regression Models

DESIGN MATRIX



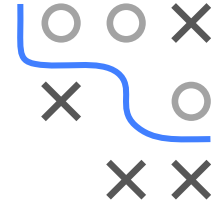
- Mismatch: $\boldsymbol{\theta} \in \mathbb{R}^{p+1}$ vs $\mathbf{x} \in \mathbb{R}^{p}$ due to intercept term
- Trick: pad feature vectors with leading 1, s.t.
  - $\mathbf{x} \mapsto \mathbf{x} = (1, x_1, \ldots, x_p)^\top$, and
  - $\boldsymbol{\theta}^\top \mathbf{x} = \theta_0 \cdot 1 + \theta_1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

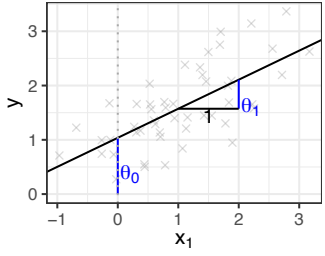
$$\hat{\mathbf{y}} = \mathbf{X}\boldsymbol{\theta} = \begin{pmatrix} 1 & x_1^{(1)} & \ldots & x_p^{(1)} \\ 1 & x_1^{(2)} & \ldots & x_p^{(2)} \\ \vdots & \vdots & & \vdots \\ 1 & x_1^{(n)} & \ldots & x_p^{(n)} \end{pmatrix} \begin{pmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_p \end{pmatrix} = \begin{pmatrix}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

- We will make use of this notation in other contexts

EFFECT INTERPRETATION

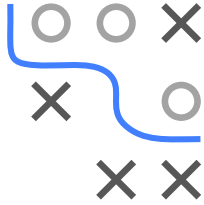
- Big plus of LM: immediately **interpretable** feature effects
- "Marginally increasing x_j by 1 unit increases y by θ_j units"
  $\rightsquigarrow$ *ceteris paribus* assumption: $x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_p$ fixed





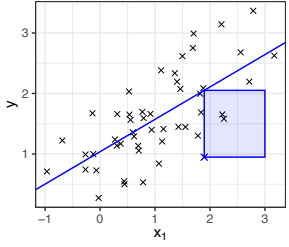

```
Call:
lm(formula = y ~ x_1, data = dt_univ)

Residuals:
     Min       1Q   Median       3Q      Max
-1.10346 -0.34727 -0.00766  0.31508  1.04204

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.03727    0.11368   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**

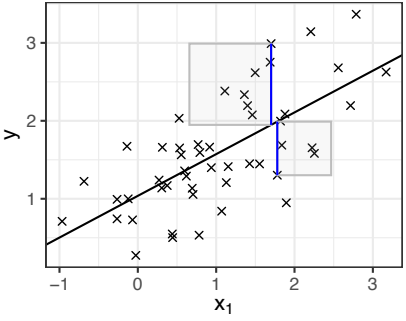
$$L\left(y, f(\mathbf{x})\right) = \left(y - f(\mathbf{x})\right)^2 \text{ or } L\left(y, f(\mathbf{x})\right) = 0.5 \cdot \left(y - f(\mathbf{x})\right)^2$$

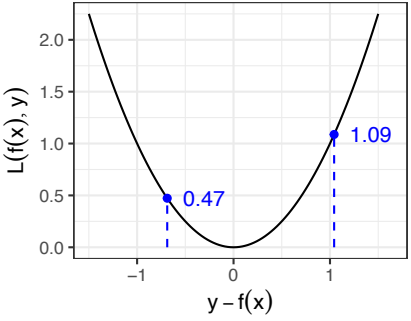


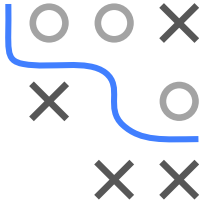
- Why penalize **residuals** $r = y - f(\mathbf{x})$ quadratically?
  - Easy to optimize (convex, differentiable)
  - Theoretically appealing (connection to classical stats LM)

LOSS PLOTS

We will often visualize loss effects like this:







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

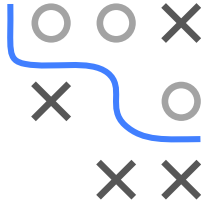
- Loss as function of residuals
  $\rightsquigarrow$ strength of penalty?
  $\rightsquigarrow$ symmetric?
- Highlighted: loss for residuals shown on LHS

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

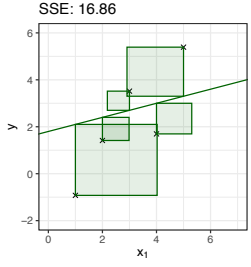


OPTIMIZATION

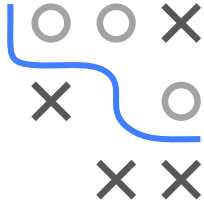
- Resulting risk equivalent to **sum of squared errors (SSE)**:

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

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



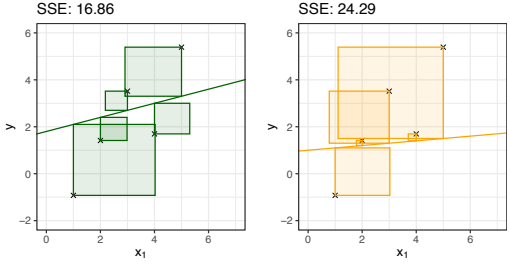
OPTIMIZATION



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

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

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

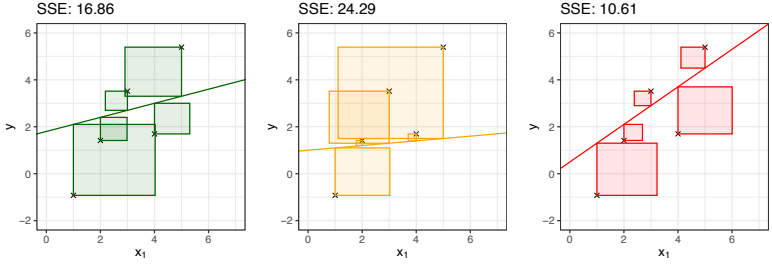


OPTIMIZATION

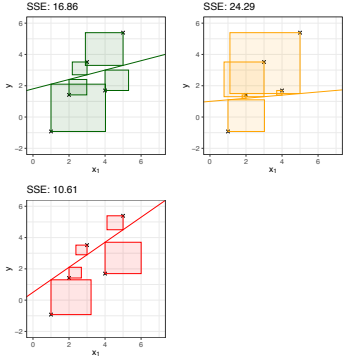
- Resulting risk equivalent to **sum of squared errors (SSE)**:

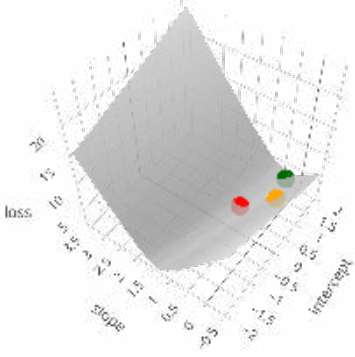
$$\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) = \sum_{i=1}^{n} \left(y^{(i)} - \boldsymbol{\theta}^\top \mathbf{x}^{(i)} \right)^2$$

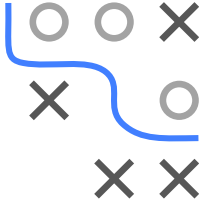
- Consider example with $n = 5 \rightsquigarrow$ different models with varying SSE



OPTIMIZATION

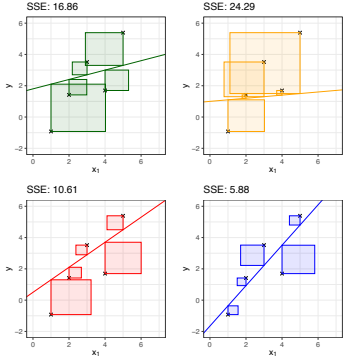


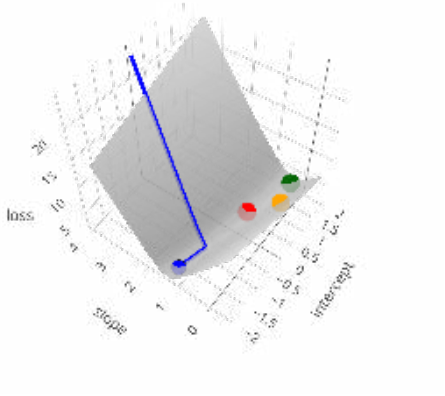


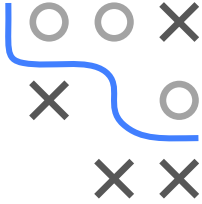


Intercept θ_0	Slope θ_1	SSE
1.80	0.30	16.86
1.00	0.10	24.29
0.50	0.80	10.61

OPTIMIZATION





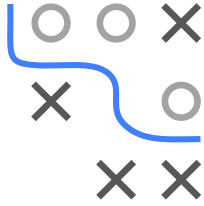


Intercept θ_0	Slope θ_1	SSE
1.80	0.30	16.86
1.00	0.10	24.29
0.50	0.80	10.61
-1.65	1.29	5.88

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

ANALYTICAL OPTIMIZATION

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



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

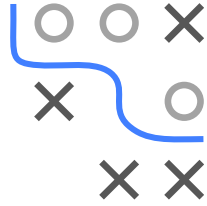
- Find via **normal equations**

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

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

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

STATISTICAL PROPERTIES



- LM with *L2* loss intimately related to classical stats LM
- Assumptions
  - $\mathbf{x}^{(i)}$ **iid** for $i \in \{1, \ldots, n\}$
  - **Homoskedastic** (equivariant) **Gaussian** errors

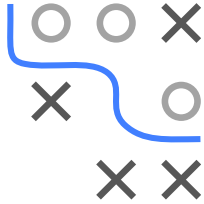
$$\mathbf{y} = \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\epsilon}, \ \boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2 \boldsymbol{I})$$

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

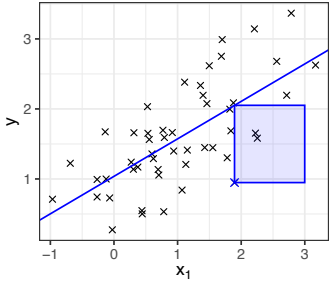
    $\rightsquigarrow$ multicollinearity destabilizes effect estimation
- If assumptions hold: statistical **inference** applicable
  - Hypothesis tests on significance of effects, incl. *p*-values
  - Confidence & prediction intervals via student-*t* distribution
  - Goodness-of-fit measure $R^2 = 1 - \text{SSE}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

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

Introduction to Machine Learning



Supervised Regression
Deep Dive: Proof OLS Regression

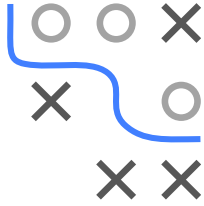


Learning goals

- Understand analytical derivation of OLS estimator for LM

ANALYTICAL OPTIMIZATION

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



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

- Find via **normal equations**

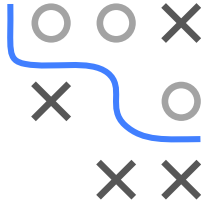
$$\frac{\partial \mathcal{R}_{\text{emp}}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = 0$$

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

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

ANALYTICAL OPTIMIZATION – PROOF

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

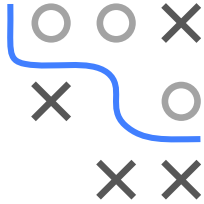


$$\begin{aligned}
0 &= \frac{\partial \mathcal{R}_{\text{emp}}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \quad \text{(sum notation)} \\
0 &= \frac{\partial}{\partial \boldsymbol{\theta}} \sum_{i=1}^{n} \epsilon_i^2 \quad \Big| \text{ sum \& chain rule} \\
0 &= \sum_{i=1}^{n} \frac{\partial \epsilon_i^2}{\partial \epsilon_i} \frac{\partial \epsilon_i}{\partial \boldsymbol{\theta}} \\
0 &= \sum_{i=1}^{n} 2\epsilon_i (-1) (\mathbf{x}^{(i)})^\top \\
0 &= \sum_{i=1}^{n} (y^{(i)} - \boldsymbol{\theta}^\top \mathbf{x}^{(i)}) (\mathbf{x}^{(i)})^\top \\
\boldsymbol{\theta}^\top \sum_{i=1}^{n} \mathbf{x}^{(i)} (\mathbf{x}^{(i)})^\top &= \sum_{i=1}^{n} y^{(i)} (\mathbf{x}^{(i)})^\top \quad \Big| \text{ transpose} \\
\left(\sum_{i=1}^{n} \underbrace{\mathbf{x}^{(i)} (\mathbf{x}^{(i)})^\top}_{\tilde{p} \times \tilde{p}} \right) \boldsymbol{\theta} &= \sum_{i=1}^{n} \mathbf{x}^{(i)} y^{(i)} \\
(\mathbf{X}^\top \mathbf{X}) \boldsymbol{\theta} &= \mathbf{X}^\top \mathbf{y}
\end{aligned}$$

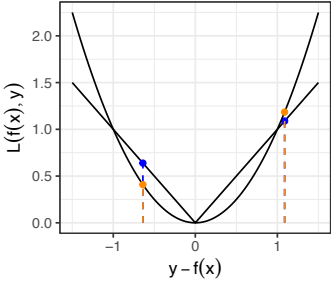
$$\begin{aligned}
0 &= \frac{\partial \mathcal{R}_{\text{emp}}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \quad \text{(matrix notation)} \\
0 &= \frac{\partial \|\boldsymbol{\epsilon}\|_2^2}{\partial \boldsymbol{\theta}} \\
0 &= \frac{\partial \boldsymbol{\epsilon}^\top \boldsymbol{\epsilon}}{\partial \boldsymbol{\theta}} \quad \Big| \text{ chain rule} \\
0 &= \frac{\partial \boldsymbol{\epsilon}^\top \boldsymbol{\epsilon}}{\partial \boldsymbol{\epsilon}} \cdot \frac{\partial \boldsymbol{\epsilon}}{\partial \boldsymbol{\theta}} \\
0 &= 2\boldsymbol{\epsilon}^\top \cdot (-1 \cdot \mathbf{X}) \\
0 &= (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^\top \mathbf{X} \\
0 &= \mathbf{y}^\top \mathbf{X} - \boldsymbol{\theta}^\top \mathbf{X}^\top \mathbf{X} \\
\boldsymbol{\theta}^\top \mathbf{X}^\top \mathbf{X} &= \mathbf{y}^\top \mathbf{X} \quad \Big| \text{ transpose} \\
\mathbf{X}^\top \mathbf{X} \boldsymbol{\theta} &= \mathbf{X}^\top \mathbf{y} \\
\boldsymbol{\theta} &= \underbrace{\underbrace{(\mathbf{X}^\top \mathbf{X})^{-1}}_{\tilde{p} \times \tilde{p}} \underbrace{\mathbf{X}^\top}_{\tilde{p} \times n} \underbrace{\mathbf{y}}_{n \times 1}}_{\tilde{p} \times 1}
\end{aligned}$$

NB: $\sum_{i=1}^{n} \mathbf{x}^{(i)} (\mathbf{x}^{(i)})^\top = \mathbf{X}^\top \mathbf{X}$ is easy to show (try it!) – and good to remember (this is basically the estimation of Cov(X))

Introduction to Machine Learning



Supervised Regression
Linear Models with $L1$ Loss

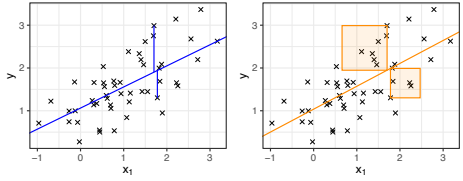


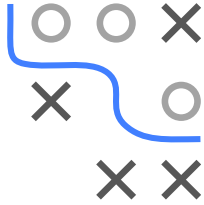
Learning goals

- Understand difference between $L1$ and $L2$ regression
- See how choice of loss affects optimization & robustness

ABSOLUTE LOSS

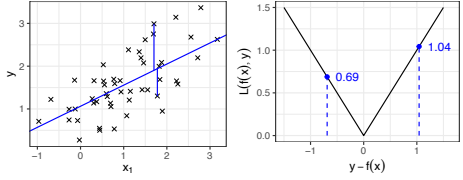
- *L*2 regression minimizes quadratic residuals – wouldn't **absolute** residuals seem more natural?



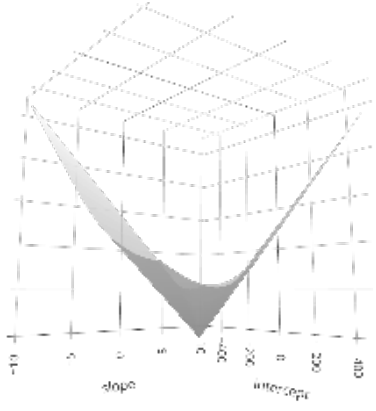


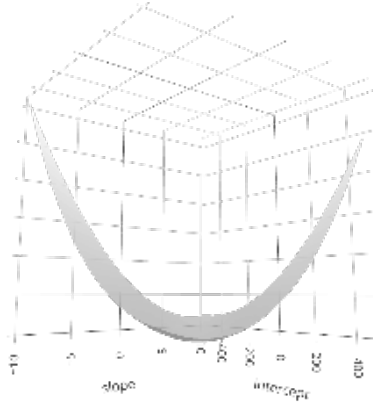
- *L*1 **loss** / **absolute error** / **least absolute deviation (LAD)**

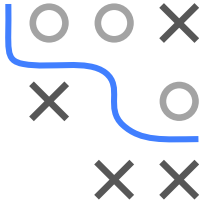
$$L(y, f(\mathbf{x})) = |y - f(\mathbf{x})|$$



*L*1 **VS** *L*2 **– LOSS SURFACE**





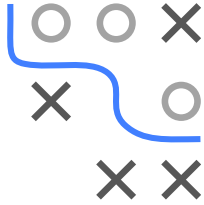


*L*1 loss (left) harder to optimize than *L*2 loss (right)

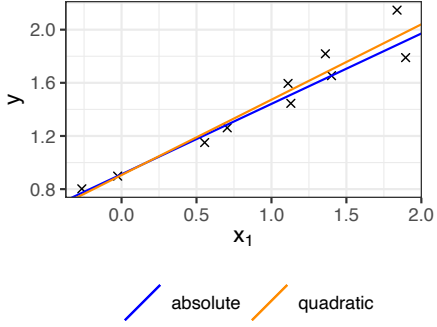
- Convex but **not differentiable** in $y - f(\mathbf{x}) = 0$
- No analytical solution

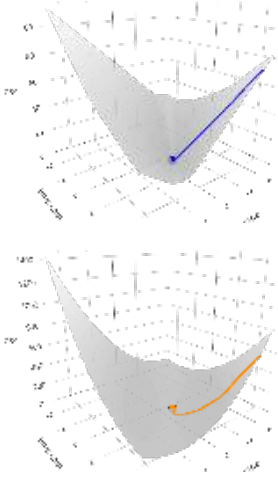
*L*1 **VS** *L*2 **– ESTIMATED PARAMETERS**

- Results of *L*1 and *L*2 regression often not that different
- Simulated data: $y^{(i)} = 1 + 0.5x_1^{(i)} + \epsilon^{(i)}, \quad \epsilon^{(i)} \overset{i.i.d}{\sim} \mathcal{N}(0, 0.01)$



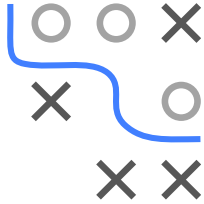
	intercept	slope
*L*1	0.91	0.53
*L*2	0.91	0.57

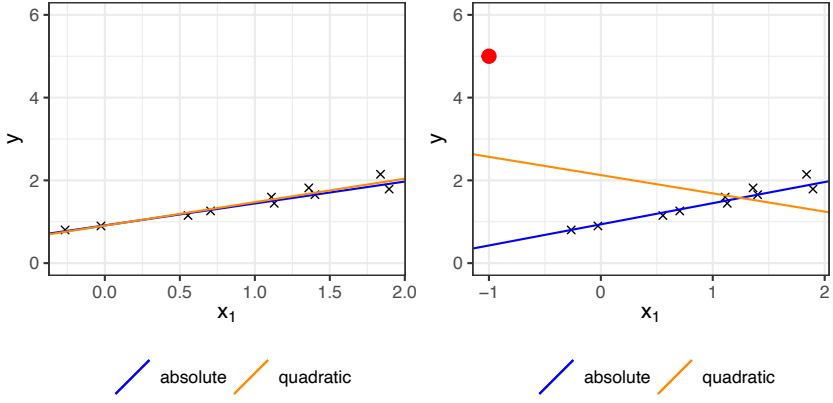




*L*1 **VS** *L*2 **– ROBUSTNESS**

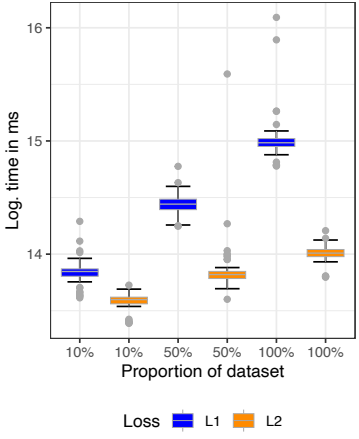
- *L*2 quadratic in residuals $\rightsquigarrow$ outlying points carry lots of weight
- E.g., 3$\times$ residual $\Rightarrow$ 9$\times$ loss contribution
- *L*1 more **robust** in presence of outliers (example ctd.):





*L*1 VS *L*2 – OPTIMIZATION COST

- Real-world `weather` problem $\rightsquigarrow$ predict mean temperature
- Compare **time** to fit *L*1 (`quantreg::rq()`) vs *L*2 (`lm::lm()`) for different dataset proportions (repeat 50$\times$)



Loss

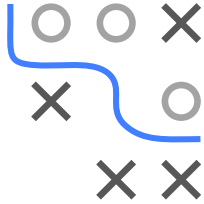
	Fitted: *L*1	Fitted: *L*2
Total *L*1 loss	8.98×10^4	8.99×10^4
Total *L*2 loss	5.83×10^6	5.81×10^6

Estimated coefficients

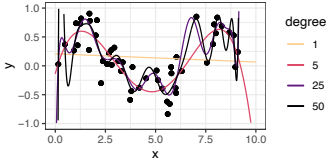
x_j	*L*1: $\hat{\theta}_j$	*L*2: $\hat{\theta}_j$
`Max_temperature`	0.553	0.563
`Min_temperature`	0.441	0.427
`Visibility`	0.026	0.041
`Wind_speed`	0.002	0.010
`Max_wind_speed`	−0.026	−0.039
`(Intercept)`	−0.380	−0.102

*L*1 **slower** to optimize!

Introduction to Machine Learning



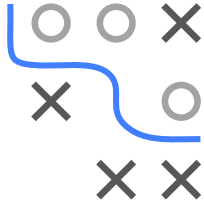
Supervised Regression
Polynomial Regression Models



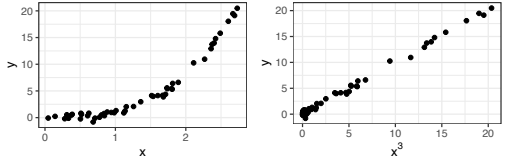
Learning goals

- 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



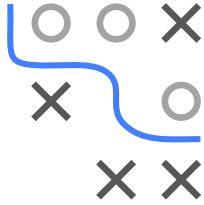
- Recall our definition of LM: model y as linear combo of features
- But: isn't that pretty **inflexible**?
- E.g., here, y does not seem to be a linear function of x...



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

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

THE LINEAR MODEL



- Recall what we previously defined as LM:

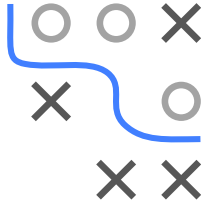
$$f(x) = \theta_0 + \sum_{j=1}^{p} \theta_j x_j = \theta_0 + \theta_1 x_1 + \cdots + \theta_p x_p \tag{1}$$

- Actually, just special case of "true" LM
- **The linear model** with **basis functions** ϕ_j:

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

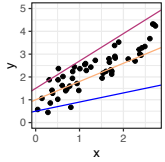
- In Eq. 1, we implicitly use identity trafo: $\phi_j = \text{id}_x : x \mapsto x \quad \forall j$
  $\rightsquigarrow$ we often say LM and imply $\phi_j = \text{id}_x$

THE LINEAR MODEL



- Are models like $f(\mathbf{x}) = \theta_0 + \theta_1 x^2$ **really linear**?
  - Certainly not in covariates:

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



$\boldsymbol{\theta} = (0.5, 0.4)^\top$
$\boldsymbol{\theta} = (1.0, 0.8)^\top$
$\boldsymbol{\theta} = (1.5, 1.2)^\top$

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

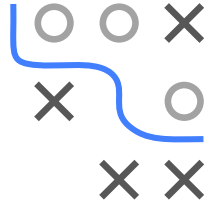
$$\begin{aligned} a \cdot f(x, \boldsymbol{\theta}) + b \cdot f(x, \boldsymbol{\theta}^*) &= a\theta_0 + b\theta_0^* + (a\theta_1 + b\theta_1^*)x^2 \\ &= f(x, a\boldsymbol{\theta} + b\boldsymbol{\theta}^*) \end{aligned}$$

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

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

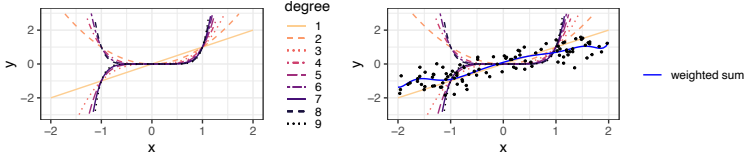
$\rightsquigarrow$ solution via normal equations as usual

POLYNOMIAL REGRESSION



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

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

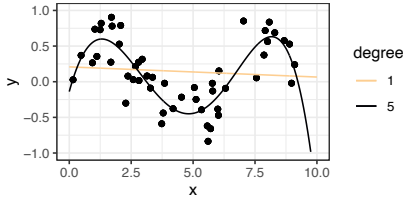


- How to estimate coefficients β_k?
  - Both LM & polynomials **linear** in their params $\rightsquigarrow$ merge
  - E.g., $f(\mathbf{x}) = \theta_0 + \theta_1 \phi^{(d)}(x) = \theta_0 + \sum_{k=1}^{d} \theta_{1,k} x^k$

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

POLYNOMIAL REGRESSION – EXAMPLES

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

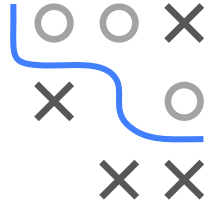


- Data-generating process:

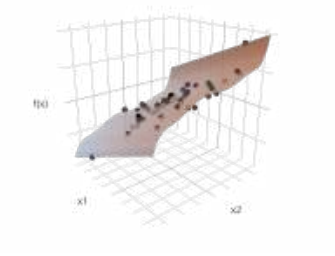
$$y = 0.5 \sin(x) + \epsilon,$$
$$\epsilon \sim \mathcal{N}(0, 0.3^2)$$

- Model:

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



Bivariate regression, $d = 7$



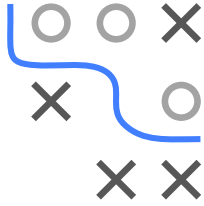
- Data-generating process:

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

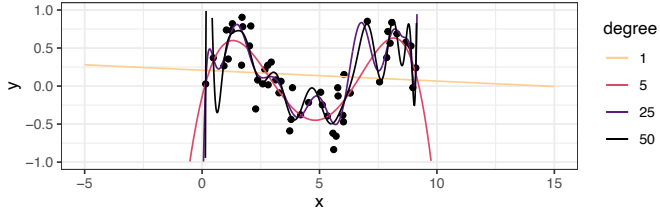
- Model:

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

COMPLEXITY OF POLYNOMIALS



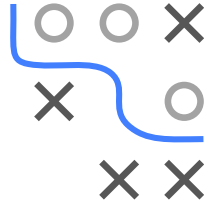
- Higher d allows to learn more complex functions
  $\rightsquigarrow$ richer hyp space / higher **capacity**



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

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

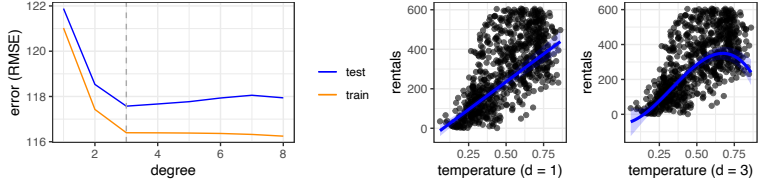
BIKE RENTAL EXAMPLE



- 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[2] confirms suspicion $\rightsquigarrow$ minimal for $d = 3$



d	θ_0	$\theta_{\text{temp},1}$	$\theta_{\text{temp},2}$	$\theta_{\text{temp},3}$	θ_{sSPRING}	θ_{sSUMMER}	θ_{sFALL}	θ_{hum}	**test error**
1	377.3	2778.2			101.0	57.0	80.1	-230.0	**121.9**
3	419.3	2645.8	-963.1	-430.9	71.9	75.8	56.6	-283.8	**117.6**

- Conclusion: flexible effects can improve fit/performance

[2] 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

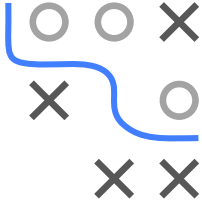
Classification and Regression Trees (CART)

Random Forests

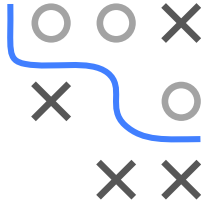
Neural Networks

Tuning

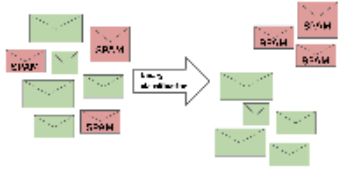
Nested Resampling



Introduction to Machine Learning



Classification Tasks

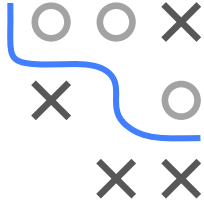


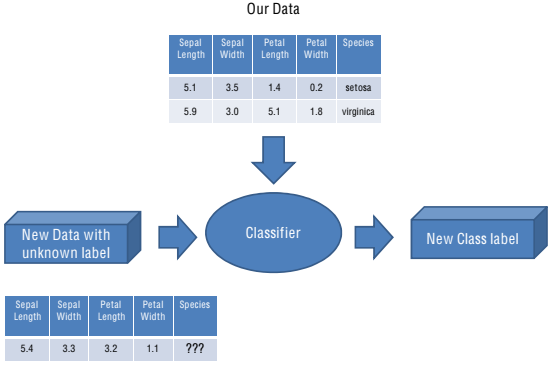
Learning goals

- Classification is supervised learning with categorical labels
- Binary vs. multiclass
- Some examples of classification tasks

CLASSIFICATION

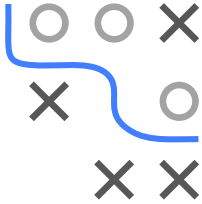
Learn functions that assign class labels to observation / feature vectors. Each observation belongs to exactly one class. The main difference to regression is that the target is categorical.

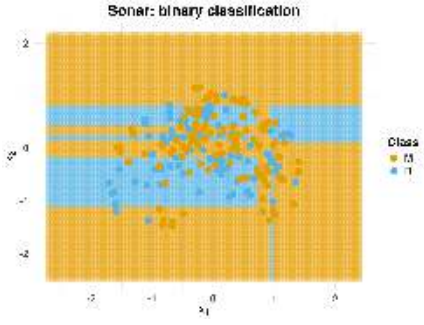




BINARY AND MULTICLASS TASKS

Tasks have a finite number of (unordered) classes.
They can be **binary** or **multiclass**.

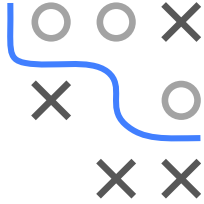


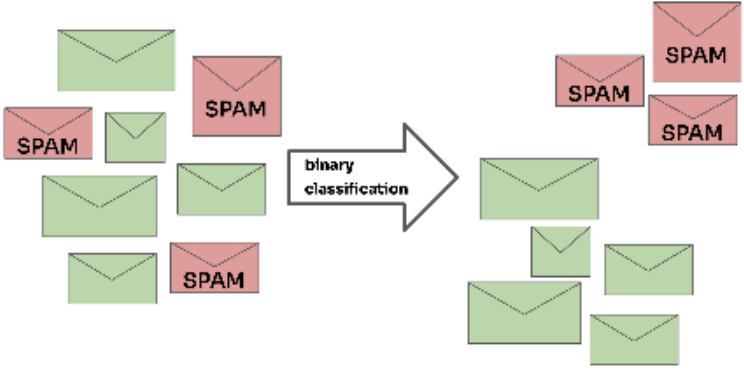




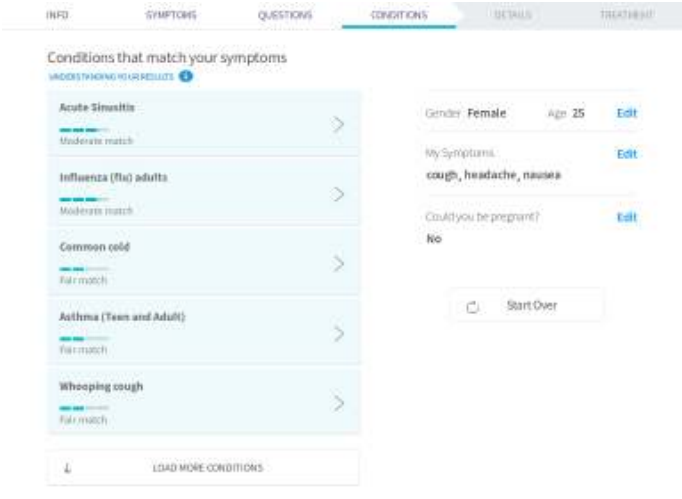
BINARY CLASSIFICATION TASK - EXAMPLES

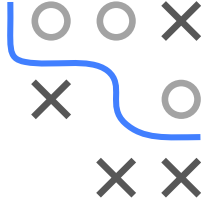
- Credit risk prediction, based on personal data and transactions
- Spam detection, based on textual features
- Churn prediction, based on customer behavior
- Predisposition for specific illness, based on genetic data





MULTICLASS TASK - MEDICAL DIAGNOSIS

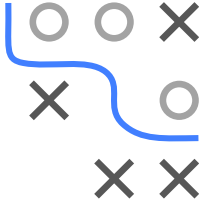




https://symptoms.webmd.com

MULTICLASS TASK - IRIS

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





Setosa



Versicolor



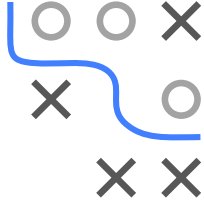
Virginica

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

MULTICLASS TASK - IRIS

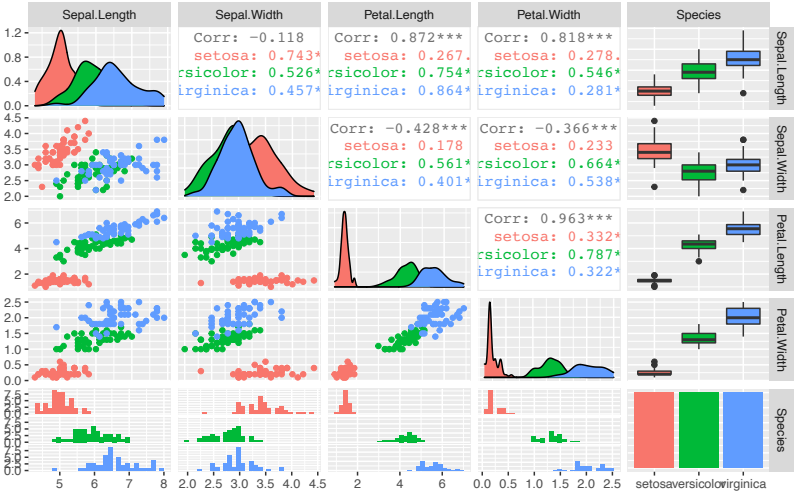
- 150 iris flowers
- Predict subspecies
- Based on sepal and petal length / width in [cm]

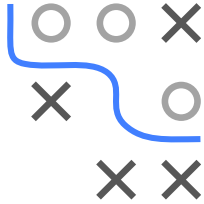




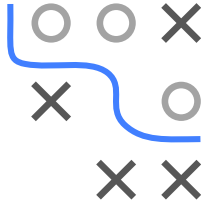
```
##       Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
##    1:          5.1         3.5          1.4         0.2    setosa
##    2:          4.9         3.0          1.4         0.2    setosa
##    3:          4.7         3.2          1.3         0.2    setosa
##    4:          4.6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

MULTICLASS TASK - IRIS

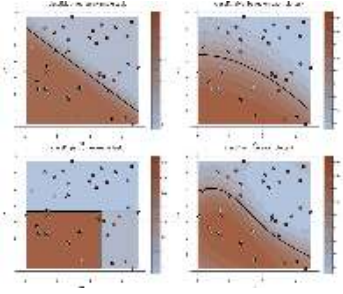




Introduction to Machine Learning



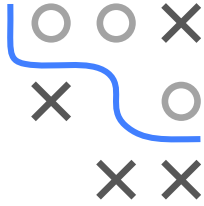
Classification
Basic Definitions



Learning goals

- Basic notation
- Hard labels vs. probabilities vs. scores
- Decision regions and boundaries
- Generative vs. discriminant approaches

NOTATION AND TARGET ENCODING



- In classification, we aim at predicting a discrete output

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

  with $2 \leq g < \infty$, given data $\mathcal{D}$
- For convenience, we often encode these classes differently
- **Binary case,** $g = 2$: Usually use $\mathcal{Y} = \{0, 1\}$ or $\mathcal{Y} = \{-1, +1\}$
- **Multiclass case,** $g \geq 3$: Could use $\mathcal{Y} = \{1, \ldots, g\}$, but often use **one-hot encoding** $o(y)$, i.e., g-length vector with $o_k(y) = \mathbb{I}(y = k) \in \{0, 1\}$:

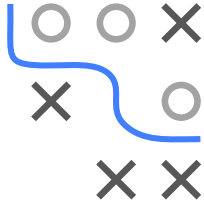
ID	Features	Species
1	...	Setosa
2	...	Setosa
3	...	Versicolor
4	...	Virginica
5	...	Setosa

one-hot encoding

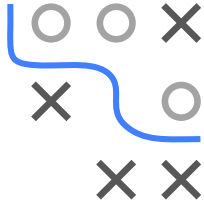
o(Species)
(1, 0, 0)
(1, 0, 0)
(0, 1, 0)
(0, 0, 1)
(1, 0, 0)

CLASSIFICATION MODELS

- While for regression the model $f : \mathcal{X} \to \mathbb{R}$ simply maps to the label space $\mathcal{Y} = \mathbb{R}$, classification is slightly more complicated.
- We sometimes like our models to output (hard) classes, sometimes probabilities, sometimes class scores. The latter 2 are vectors.
- The most basic / common form is the score-based classifier, this is why we defined models already as $f : \mathcal{X} \to \mathbb{R}^g$.
- To minimize confusion, we distinguish between all 3 in notation: $h(\mathbf{x})$ for hard labels, $\pi(\mathbf{x})$ for probabilities and $f(\mathbf{x})$ for scores
- Why all of that and not only hard labels? a) Scores / probabilities are more informative than hard class predictions; b) from an optimization perspective, it is much (!) easier to work with continuous values.



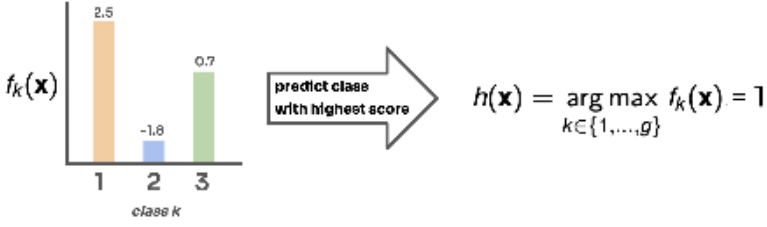
SCORING CLASSIFIERS



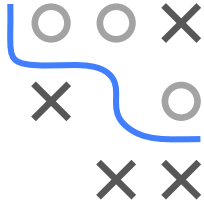
- Construct g **discriminant / scoring functions** $f_1, ..., f_g : \mathcal{X} \to \mathbb{R}$
- Predicted class is usually the one with max score

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

- For $g = 2$, a single discriminant function $f(\mathbf{x}) = f_1(\mathbf{x}) - f_{-1}(\mathbf{x})$ is sufficient (here, it's natural to label classes with $\{-1, +1\}$ and we used slight abuse of notation for the subscripts), class labels are constructed by $h(\mathbf{x}) = \text{sgn}(f(\mathbf{x}))$
- $|f(\mathbf{x})|$ or $|f_k(\mathbf{x})|$ is loosely called "confidence"



PROBABILISTIC CLASSIFIERS



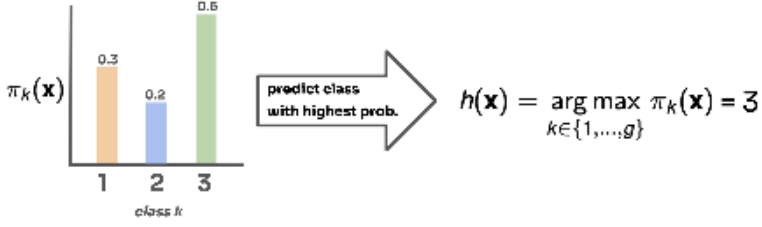
- Construct g **probability functions**

  $\pi_1, ..., \pi_g : \mathcal{X} \to [0, 1], \sum_{k=1}^{g} \pi_k(\mathbf{x}) = 1$

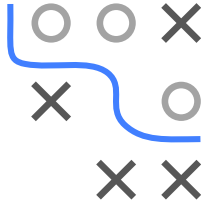
- Predicted class is usually the one with max probability

$$h(\mathbf{x}) = \underset{k \in \{1,...,g\}}{\arg\max}\ \pi_k(\mathbf{x})$$

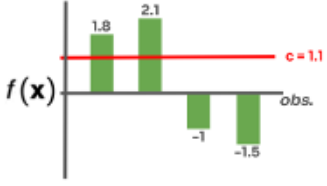
- For $g = 2$, single $\pi(\mathbf{x})$ is constructed, which models the predicted probability for the positive class (natural to encode $\mathcal{Y} = \{0, 1\}$)

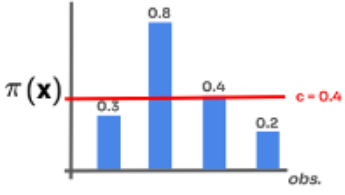


THRESHOLDING

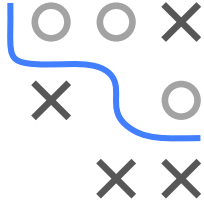


- For imbalanced cases or class with costs, we might want to deviate from the standard conversion of scores to classes
- Introduce basic concept (for binary case) and add details later
- Convert scores or probabilities to class outputs by thresholding: $h(\mathbf{x}) := [\pi(\mathbf{x}) \geq c]$ or $h(\mathbf{x}) := [f(\mathbf{x}) \geq c]$ for some threshold c
- Standard thresholds: $c = 0.5$ for probabilities, $c = 0$ for scores

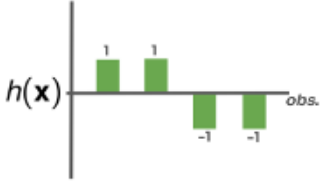


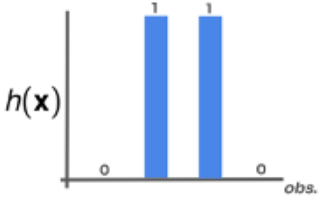


THRESHOLDING



- For imbalanced cases or class with costs, we might want to deviate from the standard conversion of scores to classes
- Introduce basic concept (for binary case) and add details later
- Convert scores or probabilities to class outputs by thresholding: $h(\mathbf{x}) := [\pi(\mathbf{x}) \geq c]$ or $h(\mathbf{x}) := [f(\mathbf{x}) \geq c]$ for some threshold c
- Standard thresholds: $c = 0.5$ for probabilities, $c = 0$ for scores

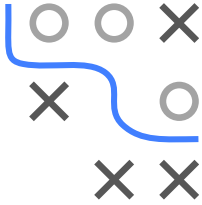


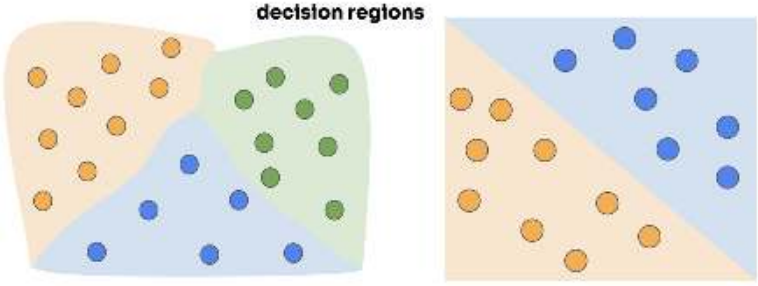


DECISION REGIONS

Set of points **x** where class k is predicted:

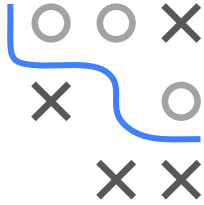
$$\mathcal{X}_k = \{\mathbf{x} \in \mathcal{X} : h(\mathbf{x}) = k\}$$





DECISION BOUNDARIES

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

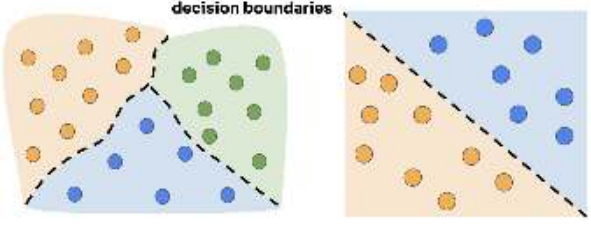


$$\{\mathbf{x} \in \mathcal{X} : \exists\, i \neq j \text{ s.t. } f_i(\mathbf{x}) = f_j(\mathbf{x}) \wedge f_i(\mathbf{x}), f_j(\mathbf{x}) \geq f_k(\mathbf{x})\ \forall k \neq i, j\}$$

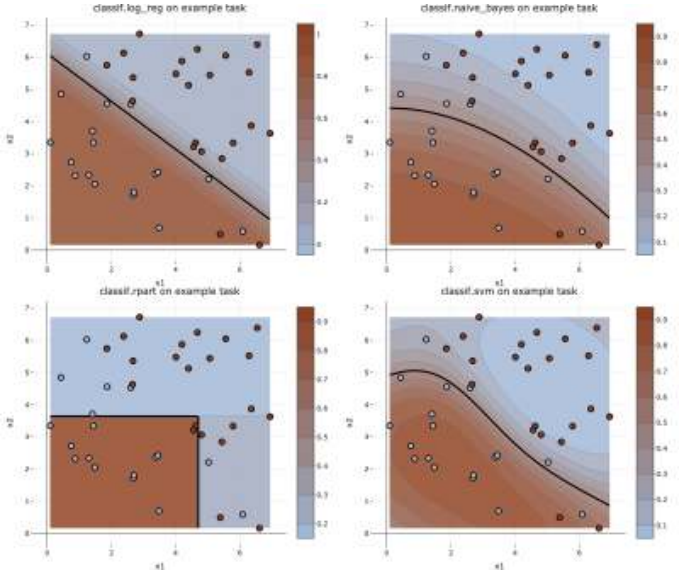
In binary case we can simply use the threshold:

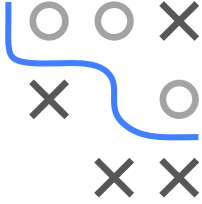
$$\{\mathbf{x} \in \mathcal{X} : f(\mathbf{x}) = c\}$$

$c = 0$ for scores and $c = 0.5$ for probs is consistent with the above.



DECISION BOUNDARY EXAMPLES

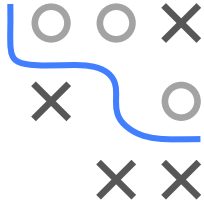




GENERATIVE APPROACH

Models class-conditional $p(\mathbf{x}|y = k)$, and employs Bayes' theorem:

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



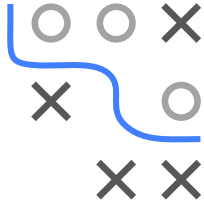
Prior probs $\pi_k = \mathbb{P}(y = k)$ can easily be estimated from training data as relative frequencies of each class:

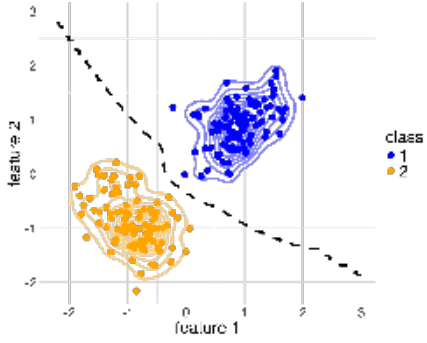
ID	Sex	Age	Class	Survived the Titanic
1	male	49	2nd	no
2	female	23	1st	yes
3	male	32	3rd	no
4	male	51	2nd	no
5	female	49	1st	yes

$$\hat{\pi} = \frac{2}{5}$$

GENERATIVE APPROACH

Decision boundary implicitly defined via the conditional distributions



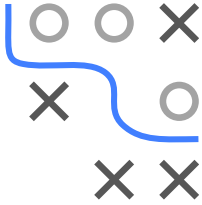


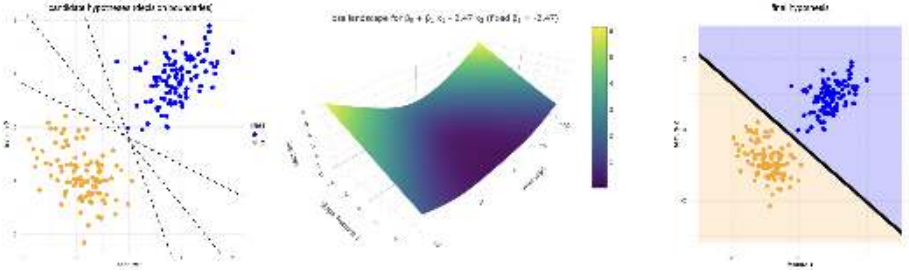
Examples are Naive Bayes, LDA and QDA.
NB: LDA and QDA have 'discriminant' in their name, but are generative!

DISCRIMINANT APPROACH

Here we optimize the discriminant functions (or better: their parameters) directly, usually via ERM:

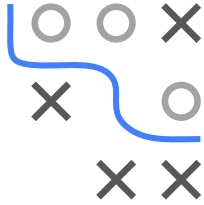
$$\hat{f} = \arg\min_{f \in \mathcal{H}} \mathcal{R}_{\text{emp}}(f) = \arg\min_{f \in \mathcal{H}} \sum_{i=1}^{n} L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right)$$



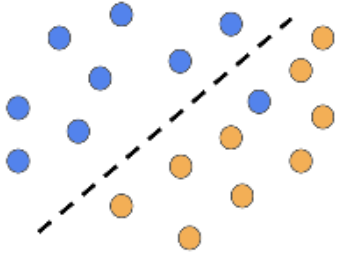


Examples are neural networks, logistic regression and SVMs

Introduction to Machine Learning



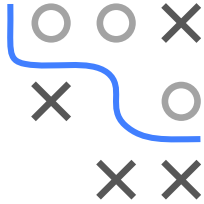
Classification
Linear Classifiers



Learning goals

- Linear classifier
- Linear decision boundaries
- Linear separability

LINEAR CLASSIFIERS



Important subclass of classification models.

Definition: If discriminant(s) $f_k(\mathbf{x})$ can be written as affine linear function(s) (possibly through a rank-preserving, monotone transformation g):

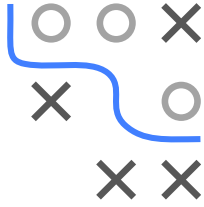
$$g(f_k(\mathbf{x})) = \boldsymbol{w}_k^\top \mathbf{x} + b_k,$$

we will call the classifier **linear**.

- $\boldsymbol{w}_k$ and b_k do not necessarily refer to parameters θ_k, although they often coincide; discriminant simply must be writable in an affine-linear way
- reasons for the transformation is that we only care about the position of the decision boundary

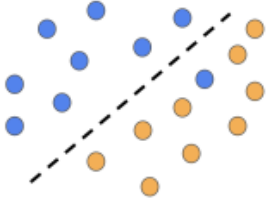
LINEAR DECISION BOUNDARIES

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:



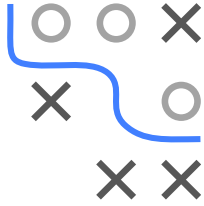
$$\begin{aligned} f_i(\mathbf{x}) &= f_j(\mathbf{x}) \\ g(f_i(\mathbf{x})) &= g(f_j(\mathbf{x})) \\ \boldsymbol{w}_i^\top \mathbf{x} + b_i &= \boldsymbol{w}_j^\top \mathbf{x} + b_j \\ (\boldsymbol{w}_i - \boldsymbol{w}_j)^\top \mathbf{x} + (b_i - b_j) &= 0 \end{aligned}$$

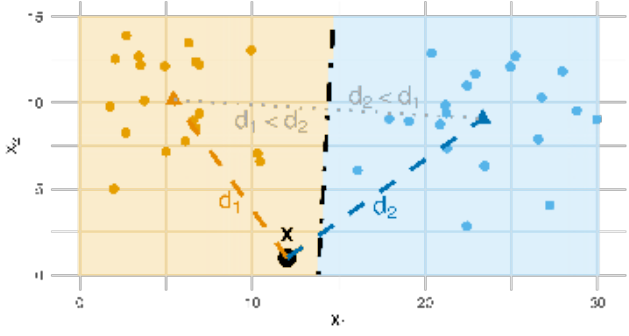
This represents a **hyperplane** separating two classes:



EXAMPLE: 2 CLASSES WITH CENTROIDS

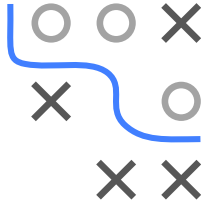
- Model binary problem with centroid $\boldsymbol{\mu_k}$ per class as "parameters"
- Don't really care how the centroids are estimated;
  could use class means, but the following doesn't depend on it
- Classify point **x** by assigning it to class *k* of nearest centroid





EXAMPLE: 2 CLASSES WITH CENTROIDS

Let's calculate the decision boundary:



$$d_1 = ||\mathbf{x}-\boldsymbol{\mu_1}||^2 = \mathbf{x}^\top\mathbf{x}-2\mathbf{x}^\top\boldsymbol{\mu_1}+\boldsymbol{\mu_1}^\top\boldsymbol{\mu_1} = \mathbf{x}^\top\mathbf{x}-2\mathbf{x}^\top\boldsymbol{\mu_2}+\boldsymbol{\mu_2}^\top\boldsymbol{\mu_2} = ||\mathbf{x}-\boldsymbol{\mu_2}||^2$$

Where d is measured using Euclidean distance. This implies:

$$-2\mathbf{x}^\top\boldsymbol{\mu_1} + \boldsymbol{\mu_1}^\top\boldsymbol{\mu_1} = -2\mathbf{x}^\top\boldsymbol{\mu_2} + \boldsymbol{\mu_2}^\top\boldsymbol{\mu_2}$$

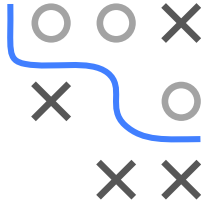
Which simplifies to:

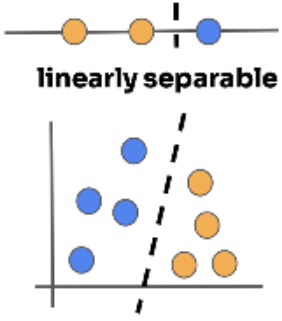
$$2\mathbf{x}^\top(\boldsymbol{\mu_2} - \boldsymbol{\mu_1}) = \boldsymbol{\mu_2}^\top\boldsymbol{\mu_2} - \boldsymbol{\mu_1}^\top\boldsymbol{\mu_1}$$

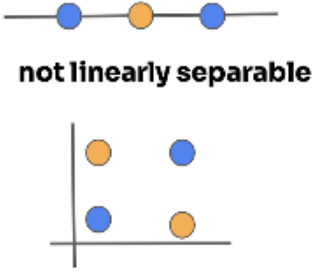
Thus, it's a linear classifier!

LINEAR SEPARABILITY

If there exists a linear classifier that perfectly separates the classes of some dataset, the data are called **linearly separable**.

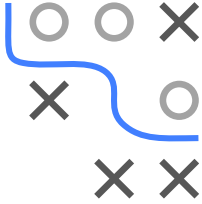


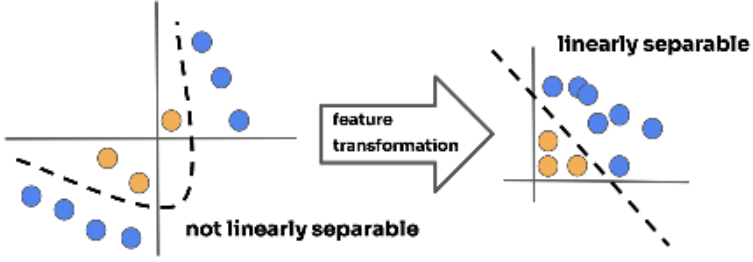




FEATURE TRANSFORMATIONS

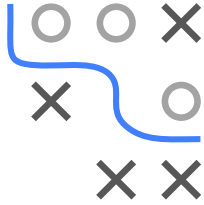
Note that linear classifiers can represent **non-linear** decision boundaries in the original input space if we use derived features like higher order interactions, polynomial features, etc.



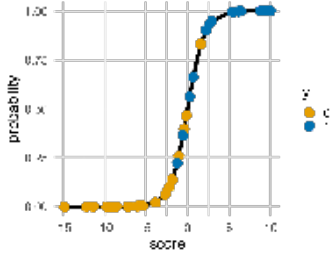


Here we used absolute values to find suitable derived features.

Introduction to Machine Learning



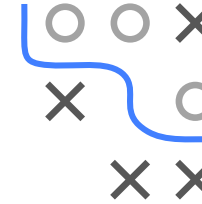
Classification
Logistic Regression



Learning goals

- Hypothesis space of LR
- Log-Loss derivation
- Intuition for loss
- LR as linear classifier

MOTIVATION



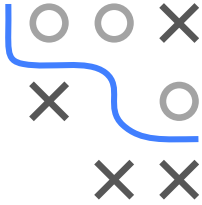
- Let's build a **discriminant** approach, for binary classification, as a probabilistic classifier $\pi(\mathbf{x} \mid \boldsymbol{\theta})$
- We encode $y \in \{0, 1\}$ and use ERM:

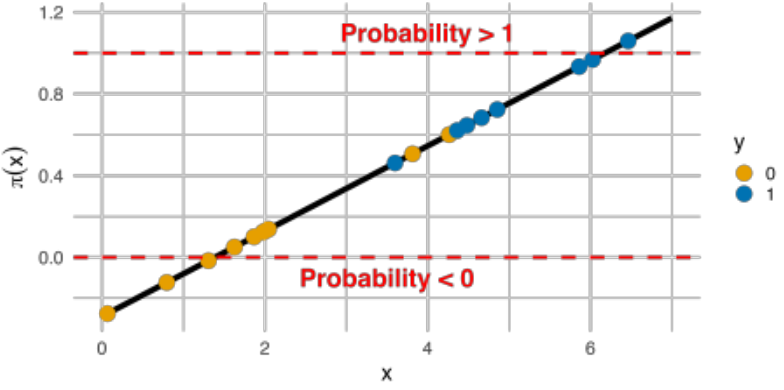
$$\arg\min_{\boldsymbol{\theta} \in \Theta} \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta} \in \Theta} \sum_{i=1}^{n} L\left(y^{(i)}, \pi\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)\right)$$

- We want to "copy" over ideas from linear regression
- In the above, our model structure should be "mainly" linear and we need a loss function

DIRECT LINEAR MODEL FOR PROBABILITIES

We could directly use an LM to model $\pi(\mathbf{x} \mid \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \mathbf{x}$.
And use L2 loss in ERM.

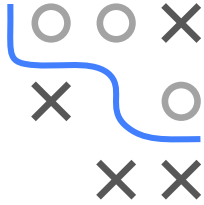




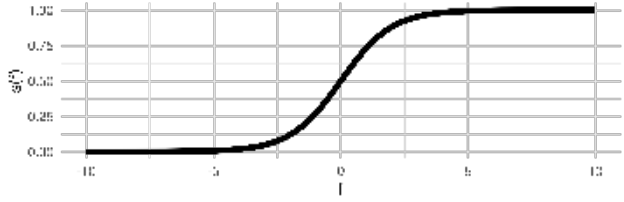
But: This obviously will result in predicted probabilities $\pi(\mathbf{x} \mid \boldsymbol{\theta}) \notin [0, 1]$!

HYPOTHESIS SPACE OF LR

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



$$\pi(\mathbf{x} \mid \boldsymbol{\theta}) = \frac{\exp\left(\boldsymbol{\theta}^\top \mathbf{x}\right)}{1 + \exp\left(\boldsymbol{\theta}^\top \mathbf{x}\right)} = \frac{1}{1 + \exp\left(-\boldsymbol{\theta}^\top \mathbf{x}\right)} = s\left(\boldsymbol{\theta}^\top \mathbf{x}\right) = s(f(\mathbf{x}))$$

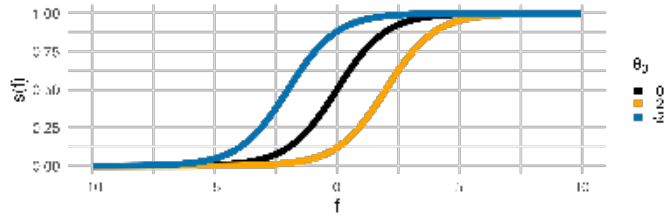


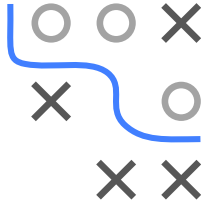
$\Rightarrow$ **Hypothesis space** of LR:

$$\mathcal{H} = \left\{ \pi : \mathcal{X} \to [0, 1] \mid \pi(\mathbf{x} \mid \boldsymbol{\theta}) = s(\boldsymbol{\theta}^\top \mathbf{x}) \mid \boldsymbol{\theta} \in \mathbb{R}^{p+1} \right\}$$

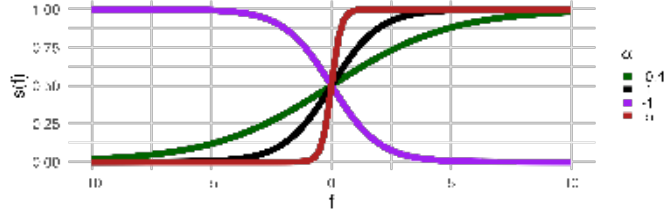
LOGISTIC FUNCTION

Intercept θ_0 shifts $\pi = s(\theta_0 + f) = \frac{\exp(\theta_0+f)}{1+\exp(\theta_0+f)}$ horizontally



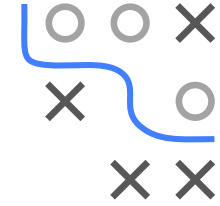


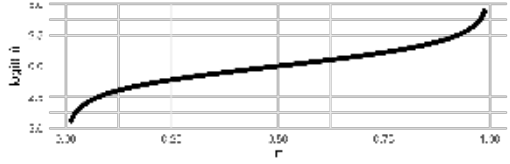
Scaling f like $s(\alpha f) = \frac{\exp(\alpha f)}{1+\exp(\alpha f)}$ controls slope and direction



THE LOGIT

The inverse $s^{-1}(\pi) = \log\left(\frac{\pi}{1-\pi}\right)$ where π is a probability is called **logit** (also called **log odds** since it is equal to the logarithm of the odds $\frac{\pi}{1-\pi}$)



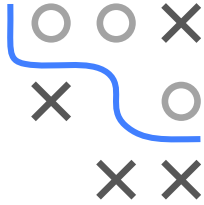


- Positive logits indicate probabilities > 0.5 and vice versa
- E.g.: if $p = 0.75$, odds are $3:1$ and logit is $log(3) \approx 1.1$
- Features $\mathbf{x}$ act linearly on logits, controlled by coefficients $\boldsymbol{\theta}$:

$$s^{-1}(\pi(\mathbf{x})) = \log\left(\frac{\pi(\mathbf{x})}{1-\pi(\mathbf{x})}\right) = \boldsymbol{\theta}^T x$$

DERIVING LOG-LOSS

We need to find a suitable loss function for **ERM**. We look at likelihood which multiplies up $\pi\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)$ for positive examples, and $1 - \pi\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)$ for negative.

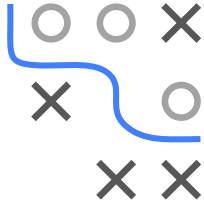


$$\mathcal{L}(\boldsymbol{\theta}) = \prod_{i \text{ with } y^{(i)}=1} \pi\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right) \prod_{i \text{ with } y^{(i)}=0} \left(1 - \pi\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)\right)$$

We can now cleverly combine the 2 cases by using exponents (note that only one of the 2 factors is not 1 and "active"):

$$\mathcal{L}(\boldsymbol{\theta}) = \prod_{i=1}^{n} \pi\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)^{y^{(i)}} \left(1 - \pi\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)\right)^{1-y^{(i)}}$$

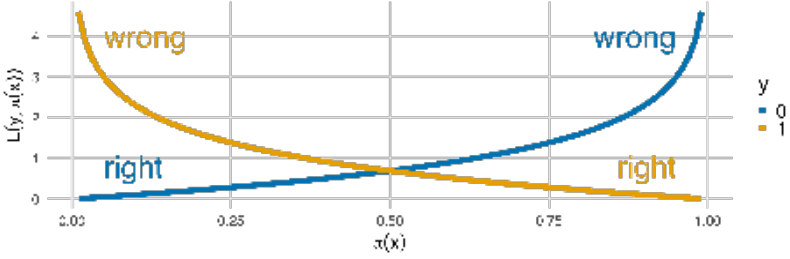
BERNOULLI / LOG LOSS



The resulting loss

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

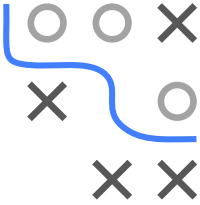
is called **Bernoulli, binomial, log** or **cross-entropy** loss

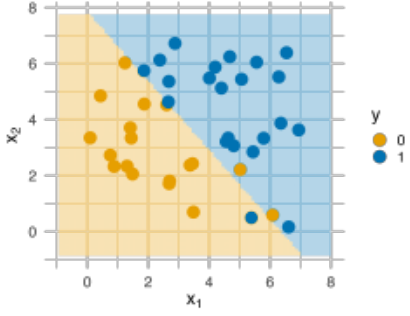


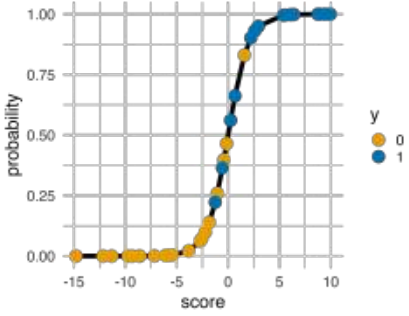
- Penalizes confidently wrong predictions heavily
- Is used for many other classifiers, e.g., in NNs or boosting

LOGISTIC REGRESSION IN 2D

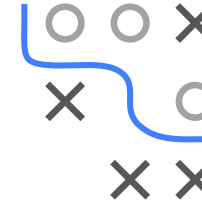
LR is a linear classifier, as $\pi(\mathbf{x} \mid \boldsymbol{\theta}) = s\left(\boldsymbol{\theta}^\top \mathbf{x}\right)$ and s is isotonic.







OPTIMIZATION



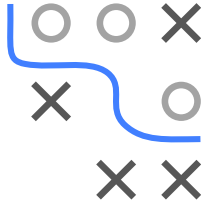
- Log-Loss is convex, under regularity conditions LR has a unique solution (because of its linear structure), but not an analytical one
- To fit LR we use numerical optimization, e.g., Newton-Raphson
- If data is linearly separable, the optimization problem is unbounded and we would not find a solution; way out is regularization
- Why not use least squares on $\pi(\mathbf{x}) = s(f(\mathbf{x}))$?
  Answer: ERM problem is not convex anymore :(
- We can also write the ERM as

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

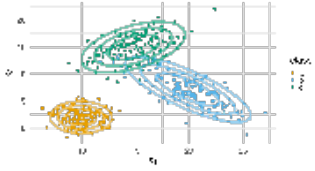
With $f(\mathbf{x} \mid \boldsymbol{\theta}) = \boldsymbol{\theta}^T\mathbf{x}$ and $L(y, f(\mathbf{x})) = -yf(\mathbf{x}) + \log(1 + \exp(f(\mathbf{x})))$

This combines the sigmoid with the loss and shows a convex loss directly on a linear function

Introduction to Machine Learning



Classification
Discriminant Analysis



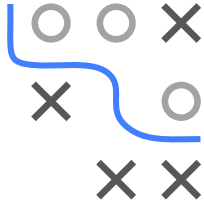
Learning goals

- LDA and QDA construction principle based on generative approach
- How are their parameters estimated
- Linear and quadratic decision boundaries

LINEAR DISCRIMINANT ANALYSIS

Generative approach, following Bayes' theorem:

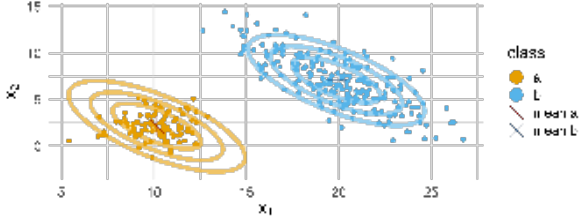
$$\pi_k(\mathbf{x}) \approx \mathbb{P}(y = k \mid \mathbf{x}) = \frac{\mathbb{P}(\mathbf{x}|y = k)\mathbb{P}(y = k)}{\mathbb{P}(\mathbf{x})} = \frac{p(\mathbf{x}|y = k)\pi_k}{\sum_{j=1}^{g} p(\mathbf{x}|y = j)\pi_j}$$



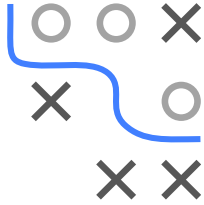
Assume that distribution $p(\mathbf{x}|y = k)$ per class is **multivariate Gaussian**:

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

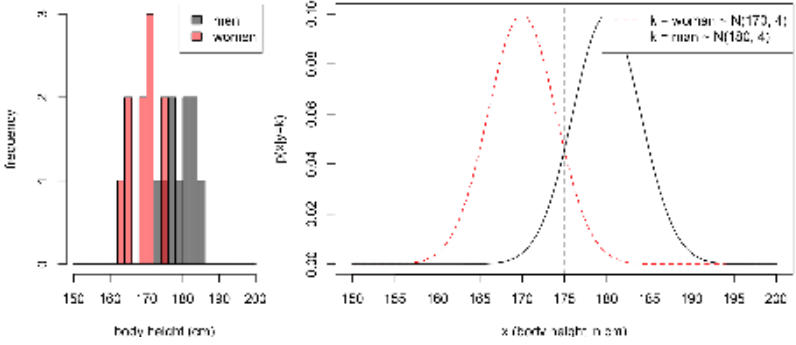
with **equal covariance structure**, so $\Sigma_k = \Sigma \quad \forall k$



UNIVARIATE EXAMPLE



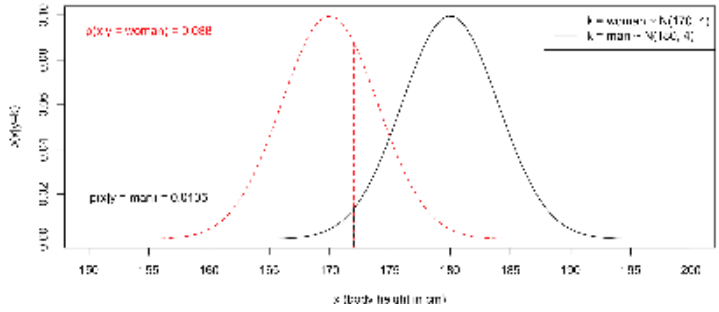
- Classify a new person as male or female based on their height (naive toy example, unrealistic in many ways)
- We will compute in the true DGP, so we assume we know all distributions and their params; we use the LDA setup



Optimal separation is located at the intersection (= decision boundary)!

UNIVARIATE EXAMPLE: EQUAL CLASS SIZES

Let's compute posterior probability that a 172 cm tall person is male

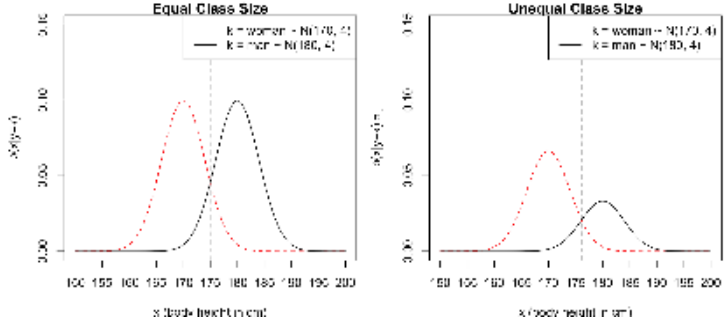


Assuming equal class sizes, prior probs π_k cancel out (since $\pi_{man} = \pi_{woman}$):

$$\mathbb{P}(y = \text{man} \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid y = \text{man})}{p(\mathbf{x} \mid y = \text{man}) + p(\mathbf{x} \mid y = \text{woman})} = \frac{0.0135}{0.0135 + 0.088} = 0.13$$

UNIVARIATE EXAMPLE: UNEQUAL CLASS SIZES

For unequal class sizes (e.g., $\pi_{woman} = 2\pi_{man}$), the prior probs matter and cause a shift of the decision boundary towards the smaller class

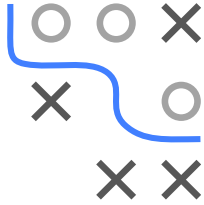


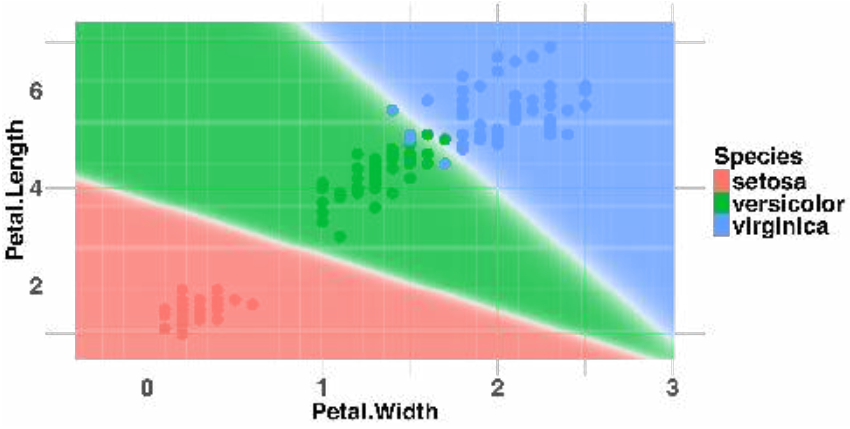
$$\mathbb{P}(y = \text{man} \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid y = \text{man})\pi_{man}}{p(\mathbf{x} \mid y = \text{man})\pi_{man} + p(\mathbf{x} \mid y = \text{woman})\pi_{woman}}$$

$$= \frac{0.0135 \cdot \frac{1}{3}}{0.0135 \cdot \frac{1}{3} + 0.088 \cdot \frac{2}{3}} = 0.0712$$

LDA AS LINEAR CLASSIFIER

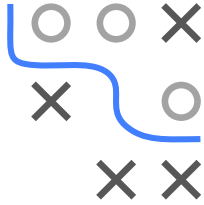
Because of the equal covariance structure of all class-specific Gaussians, the decision boundaries of LDA are always linear





LDA AS LINEAR CLASSIFIER

Can easily prove this by showing that posteriors can be written as affine-linear functions - up to rank-preserving transformation:



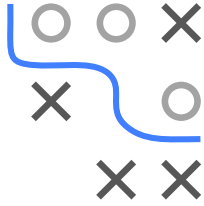
$$\pi_k(\mathbf{x}) = \frac{\pi_k \cdot p(\mathbf{x}|y=k)}{p(\mathbf{x})} = \frac{\pi_k \cdot p(\mathbf{x}|y=k)}{\sum_{j=1}^{g} \pi_j \cdot p(\mathbf{x}|y=j)}$$

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



$$
\begin{aligned}
& \pi_k \cdot p(\mathbf{x}|y=k) \\
\propto \quad & \pi_k \exp\left(-\tfrac{1}{2}\mathbf{x}^T\Sigma^{-1}\mathbf{x} - \tfrac{1}{2}\boldsymbol{\mu_k}^T\Sigma^{-1}\boldsymbol{\mu_k} + \mathbf{x}^T\Sigma^{-1}\boldsymbol{\mu_k}\right) \\
= \quad & \exp\left(\log \pi_k - \tfrac{1}{2}\boldsymbol{\mu_k}^T\Sigma^{-1}\boldsymbol{\mu_k} + \mathbf{x}^T\Sigma^{-1}\boldsymbol{\mu_k}\right) \exp\left(-\tfrac{1}{2}\mathbf{x}^T\Sigma^{-1}\mathbf{x}\right) \\
= \quad & \exp\left(w_{0k} + \mathbf{x}^T\boldsymbol{w_k}\right) \exp\left(-\tfrac{1}{2}\mathbf{x}^T\Sigma^{-1}\mathbf{x}\right) \\
\propto \quad & \exp\left(w_{0k} + \mathbf{x}^T\boldsymbol{w_k}\right)
\end{aligned}
$$

by defining $w_{0k} := \log \pi_k - \frac{1}{2}\boldsymbol{\mu_k}^T\Sigma^{-1}\boldsymbol{\mu_k}$ and $\boldsymbol{w_k} := \Sigma^{-1}\boldsymbol{\mu_k}$.

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

$$f_k(\mathbf{x}) = w_{0k} + \mathbf{x}^T\boldsymbol{w_k}$$

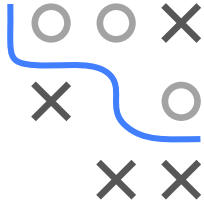
- The above is a little bit “lax” so lets carefully check
- We left out several (pos) multiplicative constants
- $\exp\left(-\frac{1}{2}\mathbf{x}^T\Sigma^{-1}\mathbf{x}\right)$ contains $\mathbf{x}$ but is the same for all classes
- $\log(at + b)$ is still isotonic for $a > 0$

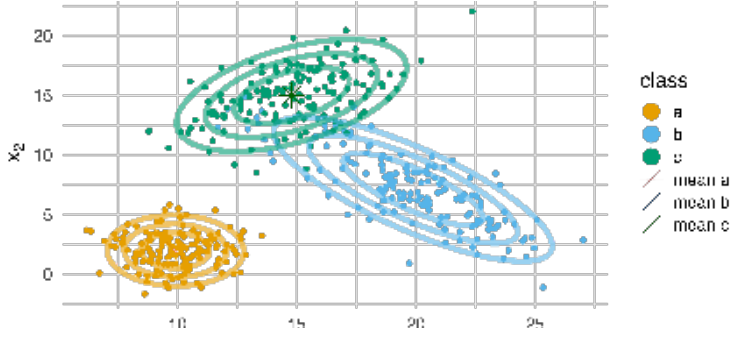
QUADRATIC DISCRIMINANT ANALYSIS

Doesn't assume equal covariances Σ_k per class, so generalizes LDA:

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

$\Rightarrow$ Better data fit but **requires estimation of more parameters** (Σ_k)!

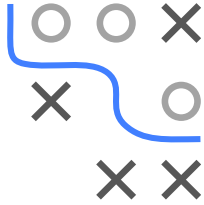


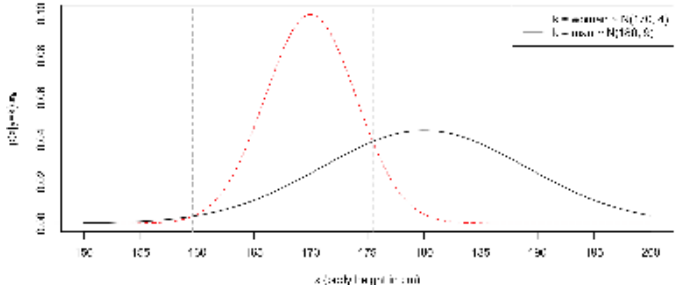


UNIVARIATE EXAMPLE WITH QDA

Different covariance matrices lead to multiple classification rules:

- $x < 159.6$ is being assigned to class *man*.
- $159.6 < x < 175.5$ is being assigned to class *woman*.
- $x > 175.5$ is being assigned to class *man*.

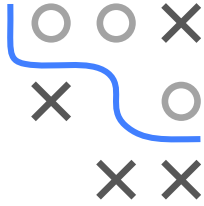




$\Rightarrow$ The separation function is quadratic, we learn a curved decision boundary (in 1D a little bit weird, as we learn an interval)

QDA DECISION BOUNDARIES

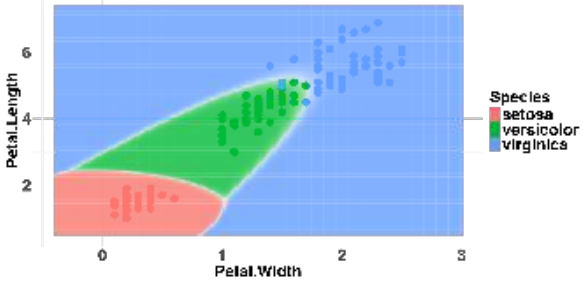
$$\begin{aligned}\pi_k(\mathbf{x}) &\propto \pi_k \cdot p(\mathbf{x}|y=k) \\ &\propto \pi_k |\Sigma_k|^{-\frac{1}{2}} \exp(-\frac{1}{2}\mathbf{x}^T\Sigma_k^{-1}\mathbf{x} - \frac{1}{2}\boldsymbol{\mu_k}^T\Sigma_k^{-1}\boldsymbol{\mu_k} + \mathbf{x}^T\Sigma_k^{-1}\boldsymbol{\mu_k})\end{aligned}$$



Taking log, we get a quadratic discriminant function in x:

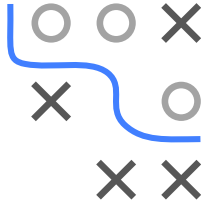
$$\log \pi_k - \frac{1}{2}\log|\Sigma_k| - \frac{1}{2}\boldsymbol{\mu_k}^T\Sigma_k^{-1}\boldsymbol{\mu_k} + \mathbf{x}^T\Sigma_k^{-1}\boldsymbol{\mu_k} - \frac{1}{2}\mathbf{x}^T\Sigma_k^{-1}\mathbf{x}$$

Allowing for curved decision boundaries:



PARAMETER ESTIMATION

Parameters $\boldsymbol{\theta}$ are estimated in a straightforward manner by:



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

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

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

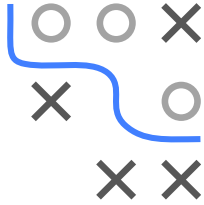
$$\hat{\Sigma} = \frac{1}{n - g} \sum_{k=1}^{g} \sum_{i:y^{(i)}=k} (\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}_{\boldsymbol{k}})(\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}_{\boldsymbol{k}})^T \quad \text{(LDA)}$$

As $\hat{\Sigma}_k, \hat{\Sigma}$ are $p \times p$ matrices (for p features), estimating all $\hat{\Sigma}_k$ involves $\frac{p(p+1)}{2} \cdot g$ parameters across g classes (vs. just $\frac{p(p+1)}{2}$ for LDA's $\hat{\Sigma}$)
(in addition to estimating priors and class means)

QDA PARAMETER ESTIMATION EXAMPLE

E.g., for a simple two-class, 2-dimensional dataset:

Class 1: $\mathbf{x}_1 = \begin{pmatrix}1\\2\end{pmatrix}, \mathbf{x}_2 = \begin{pmatrix}2\\3\end{pmatrix}$, Class 2: $\mathbf{x}_3 = \begin{pmatrix}6\\8\end{pmatrix}, \mathbf{x}_4 = \begin{pmatrix}7\\9\end{pmatrix}, \mathbf{x}_5 = \begin{pmatrix}8\\10\end{pmatrix}$



Class priors: $\hat{\pi}_1 = \frac{n_1}{n} = \frac{2}{5} = 0.4, \quad \hat{\pi}_2 = \frac{n_2}{n} = \frac{3}{5} = 0.6$

Class means: $\hat{\boldsymbol{\mu}}_1 = \frac{1}{2}(\mathbf{x}_1 + \mathbf{x}_2) = \begin{pmatrix}1.5\\2.5\end{pmatrix}, \quad \hat{\boldsymbol{\mu}}_2 = \frac{1}{3}(\mathbf{x}_3 + \mathbf{x}_4 + \mathbf{x}_5) = \begin{pmatrix}7\\9\end{pmatrix}$

Class covariances:

$$(\mathbf{x}_1 - \hat{\boldsymbol{\mu}}_1)(\mathbf{x}_1 - \hat{\boldsymbol{\mu}}_1)^\top = \begin{pmatrix}0.25 & 0.25\\0.25 & 0.25\end{pmatrix} = (\mathbf{x}_2 - \hat{\boldsymbol{\mu}}_1)(\mathbf{x}_2 - \hat{\boldsymbol{\mu}}_1)^\top$$

$$\Rightarrow \hat{\Sigma}_1 = \frac{1}{1}\left(\begin{pmatrix}0.25 & 0.25\\0.25 & 0.25\end{pmatrix} + \begin{pmatrix}0.25 & 0.25\\0.25 & 0.25\end{pmatrix}\right) = \begin{pmatrix}0.5 & 0.5\\0.5 & 0.5\end{pmatrix}$$

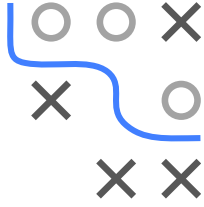
$$(\mathbf{x}_3 - \hat{\boldsymbol{\mu}}_2)(\mathbf{x}_3 - \hat{\boldsymbol{\mu}}_2)^\top = \begin{pmatrix}1 & 1\\1 & 1\end{pmatrix} = (\mathbf{x}_5 - \hat{\boldsymbol{\mu}}_2)(\mathbf{x}_5 - \hat{\boldsymbol{\mu}}_2)^\top,$$

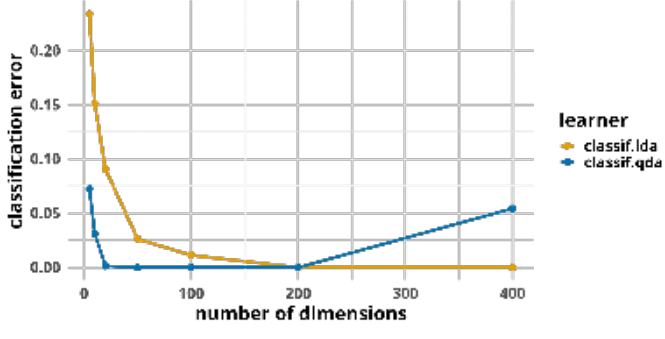
$$(\mathbf{x}_4 - \hat{\boldsymbol{\mu}}_2)(\mathbf{x}_4 - \hat{\boldsymbol{\mu}}_2)^\top = \begin{pmatrix}0 & 0\\0 & 0\end{pmatrix}$$

$$\Rightarrow \hat{\Sigma}_2 = \frac{1}{2}\left(\begin{pmatrix}1 & 1\\1 & 1\end{pmatrix} + \begin{pmatrix}0 & 0\\0 & 0\end{pmatrix} + \begin{pmatrix}1 & 1\\1 & 1\end{pmatrix}\right) = \begin{pmatrix}1 & 1\\1 & 1\end{pmatrix}$$

DISCRIMINANT ANALYSIS COMPARISON

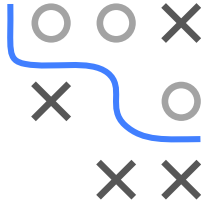
- We benchmark on simple toy data set(s)
- Normally distributed data per class, but unequal cov matrices
- And then increase dimensionality
- We might assume that QDA always wins here ...



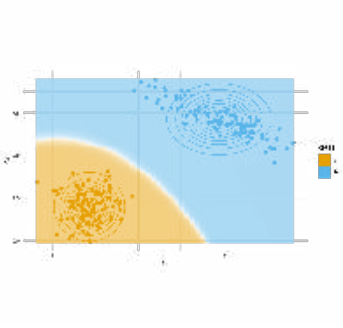


$\Rightarrow$ LDA might be preferable over QDA in higher dimensions!

Introduction to Machine Learning



Classification
Naive Bayes

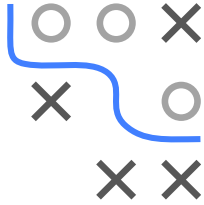


Learning goals

- Construction principle of NB
- Conditional independence assumption
- Numerical and categorical features
- Similarity to QDA, quadratic decision boundaries
- Laplace smoothing

NAIVE BAYES CLASSIFIER

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



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

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

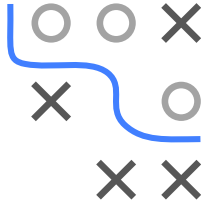
$$p(\mathbf{x}|y = k) = p((x_1, x_2, ..., x_p)|y = k) = \prod_{j=1}^{p} p(x_j|y = k).$$

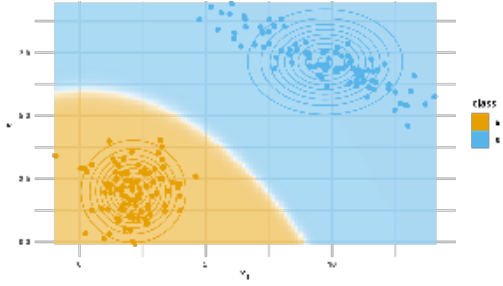
So we only need to specify and estimate the distributions $p(x_j|y = k)$, which is considerably simpler as these are univariate.

NUMERICAL FEATURES

Use univariate Gaussians for $p(x_j|y=k)$, and estimate $(\mu_{kj}, \sigma^2_{kj})$.

Because of $p(\mathbf{x}|y=k) = \prod_{j=1}^{p} p(x_j|y=k)$, joint conditional density is Gaussian with diagonal, non-isotropic covariances, and different across classes, so **QDA with diagonal covariances**.



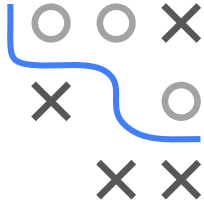


Note: In the above plot the data violates the NB assumption.

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 frequencies.

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



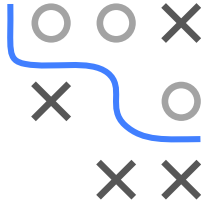
Because of the simple conditional independence structure, it is also very easy to deal with mixed numerical / categorical feature spaces.

ID	Class	Sex	Survived the Titanic
1	2nd	male	no
2	1st	male	yes
3	3rd	female	yes
4	1st	female	yes
5	2nd	female	yes
6	3rd	female	no

$$p(x_{\text{sex}}|y=\text{yes}) = p_{yes,sex,female}^{[x_{\text{sex}}=\text{female}]} \cdot p_{yes,sex,male}^{[x_{\text{sex}}=\text{male}]} = \frac{3}{4}^{[x_{\text{sex}}=\text{female}]} \cdot \frac{1}{4}^{[x_{\text{sex}}=\text{male}]}$$

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, e.g.:
$p_{\text{no, class, 1st}}^{[x_{class}=\text{1st}]} = 0$ (everyone from 1st class survived in the previous table)

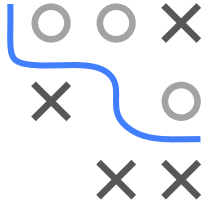


This is problematic because it will wipe out all information in the other probabilities when they are multiplied!

$$\pi_{no}(\text{class = 1st, sex = male}) = \frac{\hat{p}(x_{class}|y = no) \cdot \hat{p}(x_{sex}|y = no) \cdot \hat{\pi}_{no}}{\sum_{j=1}^{g} \hat{p}(\text{class = 1st, sex = male}|y = j)\hat{\pi}_j} = 0$$

LAPLACE SMOOTHING

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



- Add constant $\alpha > 0$ (e.g., $\alpha = 1$).
- For a categorical feature x_j with M_j possible values:

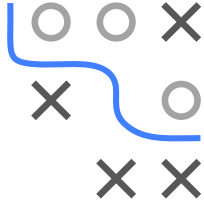
$$p_{kjm}^{[x_j=m]} = \frac{n_{kjm} + \alpha}{n_k + \alpha M_j} \quad \left(\text{instead of } p_{kjm}^{[x_j=m]} = \frac{n_{kjm}}{n_k}\right)$$

where:

- n_{kjm}: count of $x_j = m$ in class k,
- n_k: total counts in class k,
- M_j: number of possible distinct values of x_j.

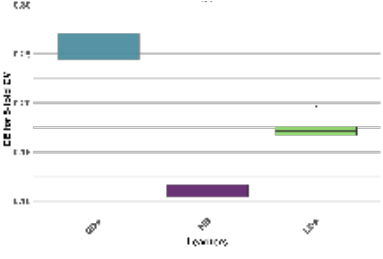
This ensures that our posterior probabilities are non-zero due to such effects, preserving the influence of all features in the model.

NAIVE BAYES: APPLICATION AS SPAM FILTER



- In the late 90s, NB became popular for e-mail spam detection
- Word counts were used as features to detect spam mails
- Independence assumption implies: occurrence of two words in mail is not correlated, this is often wrong;
  "viagra" more likely to occur in context with "buy"...
- In practice: often still good performance

Benchmarking QDA, NB and LDA on `spam`:



INTRODUCTION TO MACHINE LEARNING

ML Basics

Supervised Regression

Supervised Classification

Performance Evaluation

k-NN

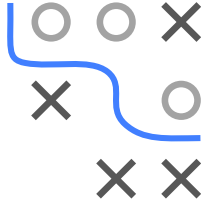
Classification and Regression Trees (CART)

Random Forests

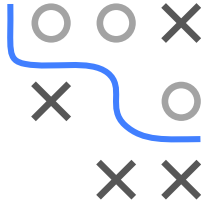
Neural Networks

Tuning

Nested Resampling



Introduction to Machine Learning



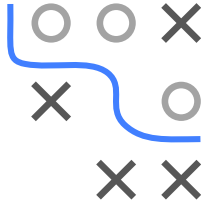
Evaluation
Generalization Error



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):

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

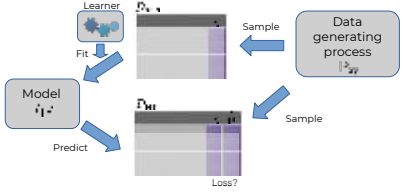
$$\min_{\theta \in \Theta} \sum_{i=1}^{n} L\left(y^{(i)}, f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)\right)$$

- 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

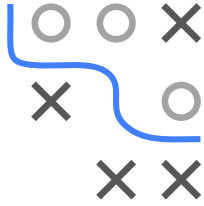
- GE for a fixed model: $\mathrm{GE}\left(\hat{f}, L\right) := \mathbb{E}\left[L\left(y, \hat{f}(\mathbf{x})\right)\right]$
  Expectation over a single, random test point $(\mathbf{x}, y) \sim \mathbb{P}_{xy}$.
- Estimator, **if a dedicated test set is available** (size m)

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

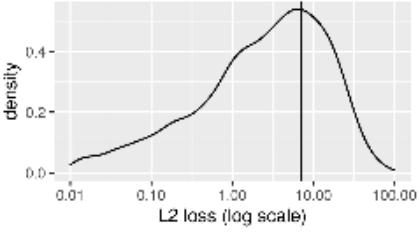


NB: Very often, no dedicated test-set is available, and what we describe here is not same as hold-out splitting (see later).

EXAMPLE: TEST LOSS AS RANDOM VARIABLE

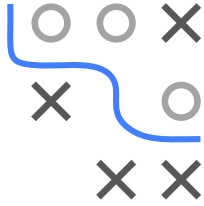


- For a fixed model and dedicated i.i.d. test set, we can easily approximate the complete test loss distribution $L(y, \hat{f}(\mathbf{x}))$.
- LM on `mlbench::friedman1` test problem
- With $n_{\text{train}} = 500$ we create a fixed model
- We feed 5000 fresh test points to model
- And plot the pointwise *L2* loss.



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

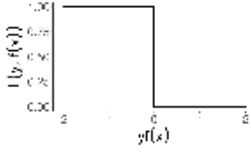
INNER VS OUTER LOSS



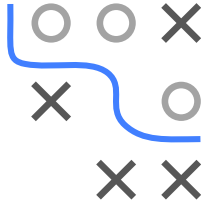
- Sometimes, we would like to evaluate our learner with a different loss L or metric ρ.
- Nomenclature: ERM and **inner loss**; evaluation and **outer loss**.
- Different losses, if computationally advantageous to deviate from outer loss of application; e.g., optimization faster with inner L2 or maybe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

Example: Linear binary classifier / Logistic regression.

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



SET-BASED PERFORMANCE METRICS



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

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

- 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, $\boldsymbol{F}$ is prediction matrix

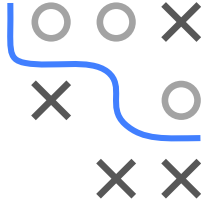
$$\boldsymbol{F} = \begin{bmatrix} \hat{f}(\mathbf{x}^{(1)}) \\ \dots \\ \hat{f}(\mathbf{x}^{(m)}) \end{bmatrix} \in \mathbb{R}^{m \times g}$$

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

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

MODEL GE VS. LEARNER GE

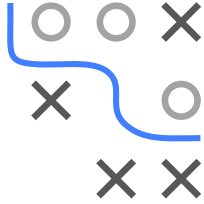
To clear up a major point of confusion (or totally confuse you):



- 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

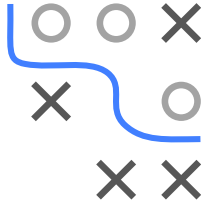
$$\mathrm{GE}(\mathcal{I}, \boldsymbol{\lambda}, n_{\mathrm{train}}, \rho) := \lim_{n_{\mathrm{test}} \to \infty} \mathbb{E}\left[\rho\left(\mathbf{y}, \boldsymbol{F}_{\mathcal{D}_{\mathrm{test}}, \mathcal{I}(\mathcal{D}_{\mathrm{train}}, \boldsymbol{\lambda})}\right)\right]$$



- Quality of models when fitted with $\mathcal{I}_{\boldsymbol{\lambda}}$ on n_{train} points from $\mathbb{P}_{xy}$.
- Expectation **both** over $\mathcal{D}_{\mathrm{train}}$ and $\mathcal{D}_{\mathrm{test}}$, 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

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



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

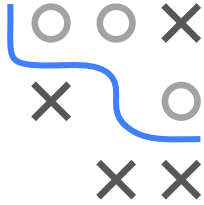
$$\mathrm{GE}(\mathcal{I}, \boldsymbol{\lambda}, n_{\text{train}}, \rho_L) := \mathbb{E}\left[L(y, \mathcal{I}(\mathcal{D}_{\text{train}}, \boldsymbol{\lambda})(\mathbf{x}))\right]$$

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

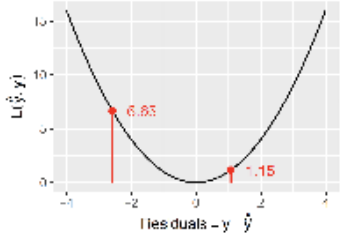
$$\mathrm{GE}\left(\hat{f}, L\right) := \mathrm{GE}(\mathcal{I}, \boldsymbol{\lambda}, n_{\text{train}}, \rho_L | \mathcal{D}_{\text{train}})$$

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

Introduction to Machine Learning



Evaluation
Measures for Regression

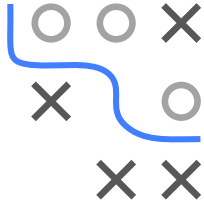


Learning goals

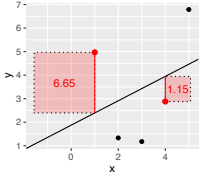
- Know the definitions of mean squared error (MSE) and mean absolute error (MAE)
- Understand the connections of MSE and MAE to L2 and L1 loss
- Know the definition of Spearman's ρ
- Know the definitions of R^2 and generalized R^2

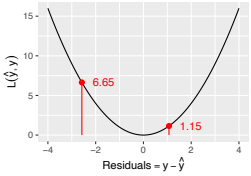
MEAN SQUARED ERROR (MSE)

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



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



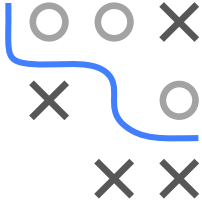


Similar measures:

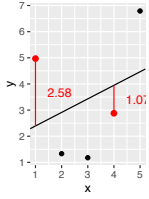
- Sum of squared errors: $\rho_{SSE}(\mathbf{y}, \boldsymbol{F}) = \sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2$
- Root MSE (orig. scale): $\rho_{RMSE}(\mathbf{y}, \boldsymbol{F}) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2}$

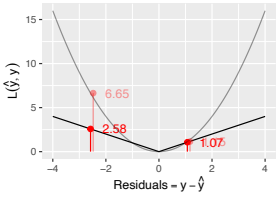
MEAN ABSOLUTE ERROR

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



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



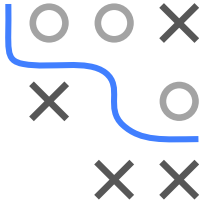


Similar measures:

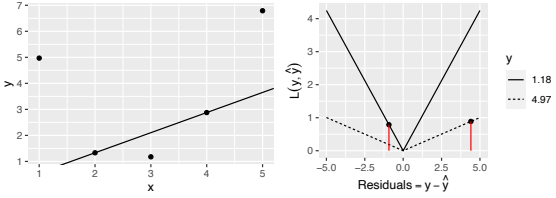
- Median absolute error (for even more robustness)

MEAN ABSOLUTE PERCENTAGE ERROR

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



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

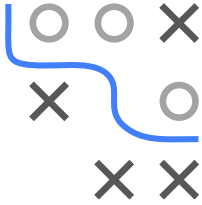


Similar measures:

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

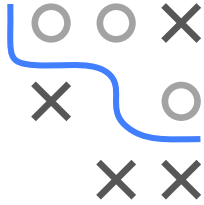
R^2

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

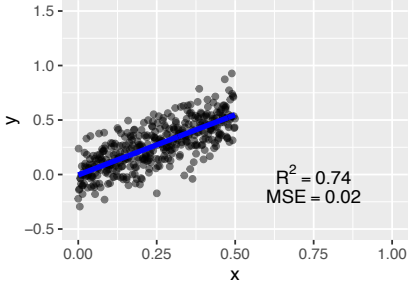


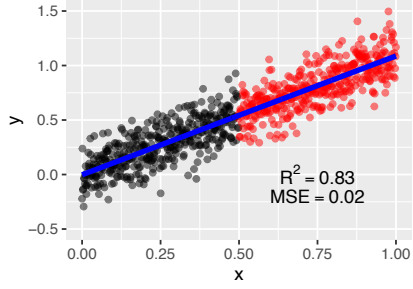
- Well-known classical measure for LMs – on train data.
- "Fraction of variance explained" by the model.
- How much SSE of constant baseline is reduced when we use more complex model?
- $\rho_{R^2} = 1$: all residuals are 0, we predict perfectly,
- $\rho_{R^2} = 0.9$: LM reduces SSE by factor of 10.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

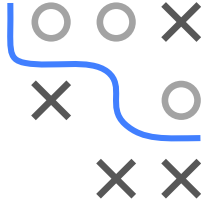




- Fit does not improve, but R^2 goes up.
- But: Invariant w.r.t. to linear scaling of y, MSE is not.

GENERALIZED R^2 FOR ML

$$1 - \frac{Loss_{ComplexModel}}{Loss_{SimplerModel}}.$$

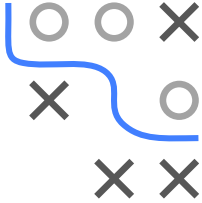


- E.g., model vs constant, LM vs non-linear model, tree vs forest, model with fewer features vs model with more, ...
- We could use arbitrary measures.
- In ML we would rather evaluate on test set.
- Can then become negative, e.g., for SSE and constant baseline, if our model fairs worse on the test set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

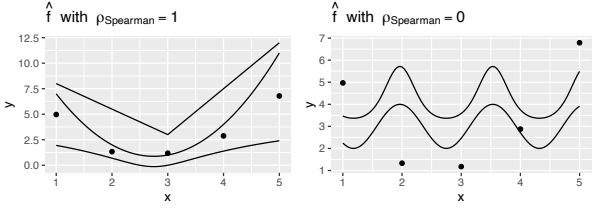
SPEARMAN'S ρ

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

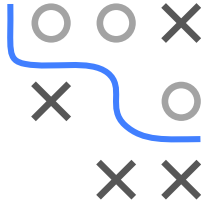
$$\rho_{\text{Spearman}}(\mathbf{y}, \boldsymbol{F}) = \frac{\text{Cov}(\text{rg}(\mathbf{y}), \text{rg}(\hat{\mathbf{y}}))}{\sqrt{\text{Var}(\text{rg}(\mathbf{y}))} \cdot \sqrt{\text{Var}(\text{rg}(\hat{\mathbf{y}}))}} \in [-1, 1],$$



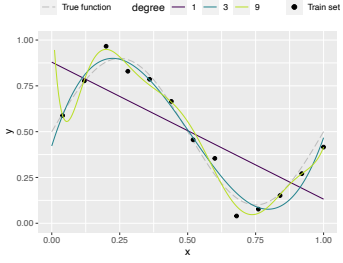
- Very robust against outliers
- A value of 1 or -1 means that $\hat{\mathbf{y}}$ and $\mathbf{y}$ have a perfect monotonic relationship.
- Invariant under monotone transformations of $\hat{\mathbf{y}}$



Introduction to Machine Learning



Evaluation
Training Error

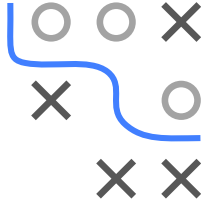


Learning goals

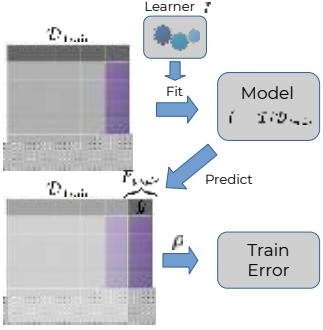
- Understand the definition of training error
- Understand why train error is unreliable for models of higher complexity when overfitting can occur

TRAINING ERROR

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



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

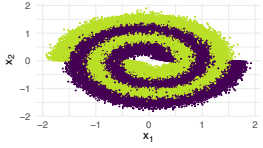


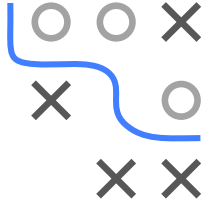
A.k.a. apparent error or resubstitution error.

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").



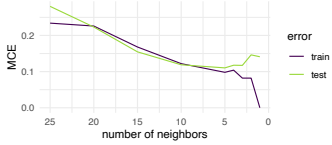


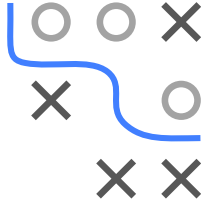
We increase train size, and see how gap between train error and GE closes.

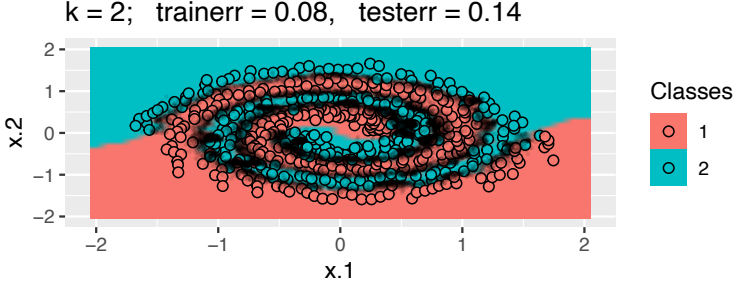


EXAMPLE 1: KNN

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



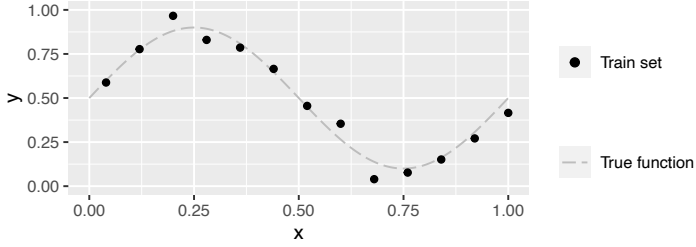


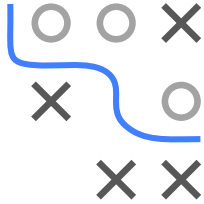


Black region are misclassifications from large test test.

EXAMPLE 2: POLYNOMIAL REGRESSION

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



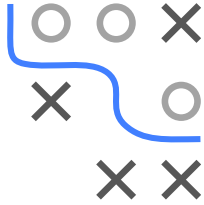


We fit a dth-degree polynomial:

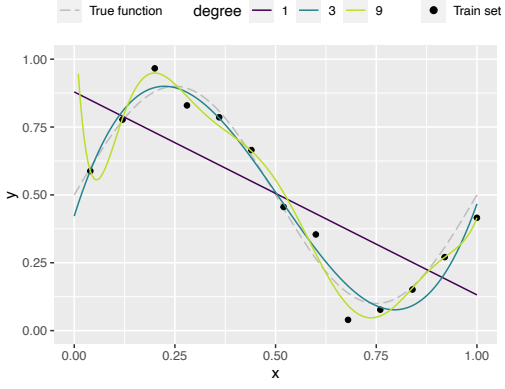
$$f(\mathbf{x} \mid \boldsymbol{\theta}) = \theta_0 + \theta_1 \mathbf{x} + \cdots + \theta_d \mathbf{x}^d = \sum_{j=0}^{d} \theta_j \mathbf{x}^j.$$

EXAMPLE 2: POLYNOMIAL REGRESSION

Simple model selection problem: Which d?



Visual inspection vs quantitative MSE on training set:

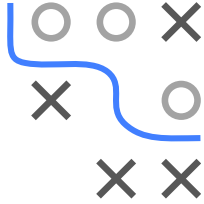


- $d = 1$: MSE = 0.036: clearly underfitting
- $d = 3$: MSE = 0.003: pretty OK
- $d = 9$: MSE = 0.001: clearly overfitting

Using the train error chooses overfitting model of maximal complexity.

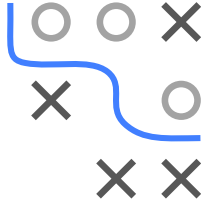
TRAIN ERROR CAN EASILY BECOME 0

- 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.



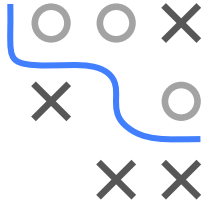
CLASSICAL STATISTICAL GOF MEASURES

- **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?

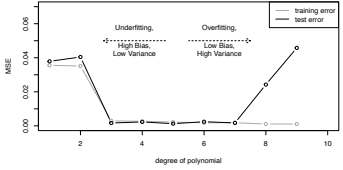


Out-of-sample testing is probably always a good idea!

Introduction to Machine Learning



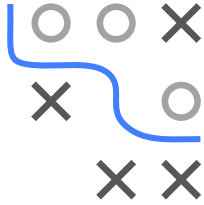
Evaluation
Test Error



Learning goals

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

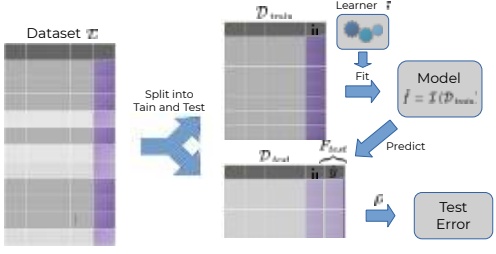
TEST ERROR AND HOLD-OUT SPLITTING



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

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

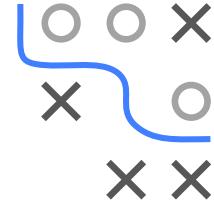
- Partition data, e.g., 2/3 for train and 1/3 for test.

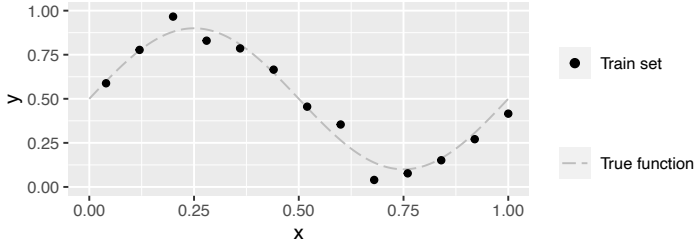


A.k.a. holdout splitting.

EXAMPLE: POLYNOMIAL REGRESSION

Previous example:

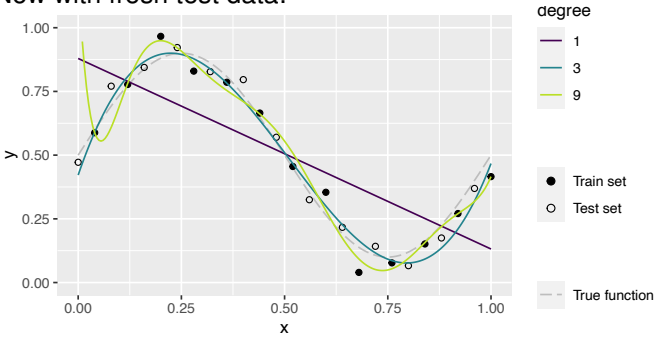


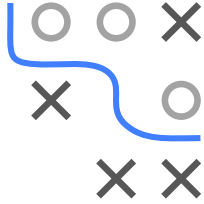


$$f(\mathbf{x} \mid \boldsymbol{\theta}) = \theta_0 + \theta_1 \mathbf{x} + \cdots + \theta_d \mathbf{x}^d = \sum_{j=0}^{d} \theta_j \mathbf{x}^j.$$

EXAMPLE: POLYNOMIAL REGRESSION

Now with fresh test data:



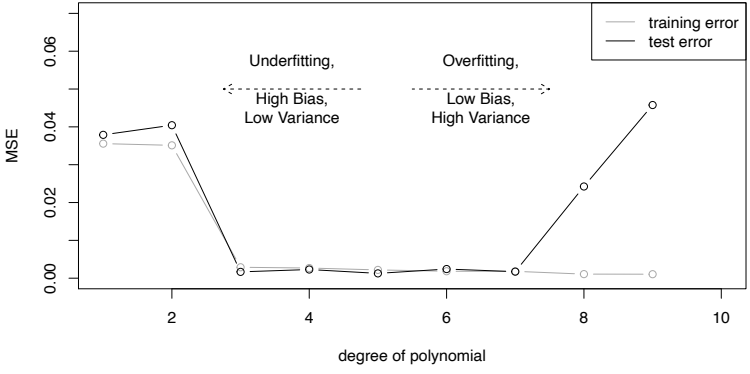


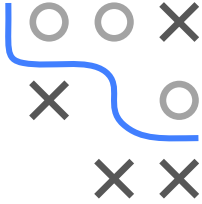
- $d = 1$: MSE = 0.038: clearly underfitting
- $d = 3$: MSE = 0.002: pretty OK
- $d = 9$: MSE = 0.046: clearly overfitting

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:

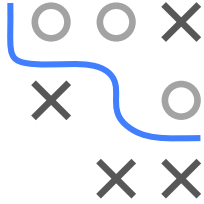




Increasing model complexity tends to cause

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

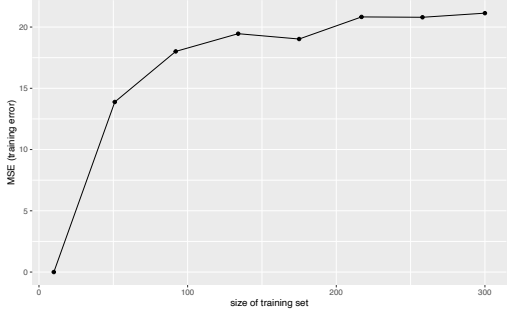
TRAINING VS. TEST ERROR



- Boston Housing data
- Polynomial regression (without interactions)

The training error...

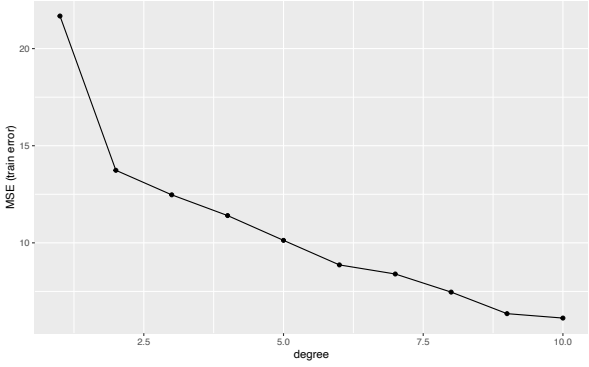
- decreases with smaller training set size as it becomes easier for the model to learn all observed patterns perfectly.

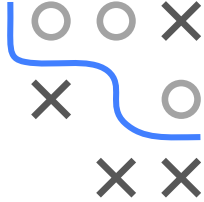


TRAINING VS. TEST ERROR

The training error...

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

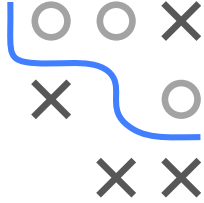


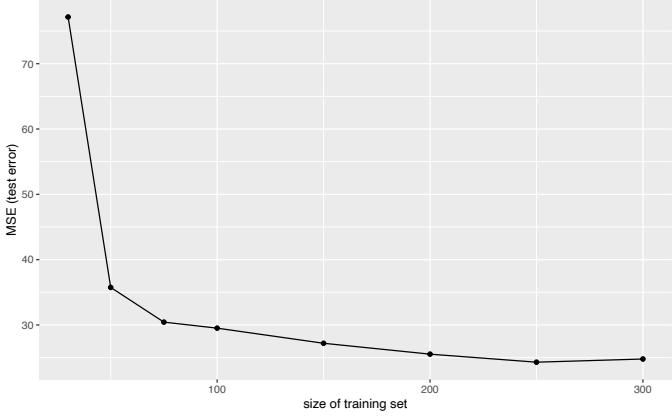


TRAINING VS. TEST ERROR

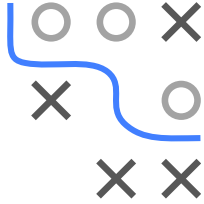
The test error...

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



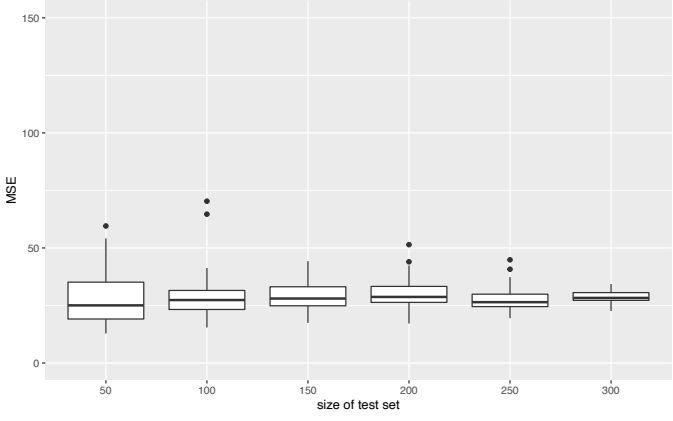


TRAINING VS. TEST ERROR

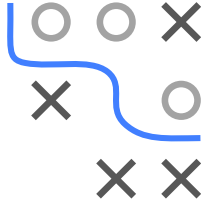


The test error...

- will have higher variance with smaller test set size.

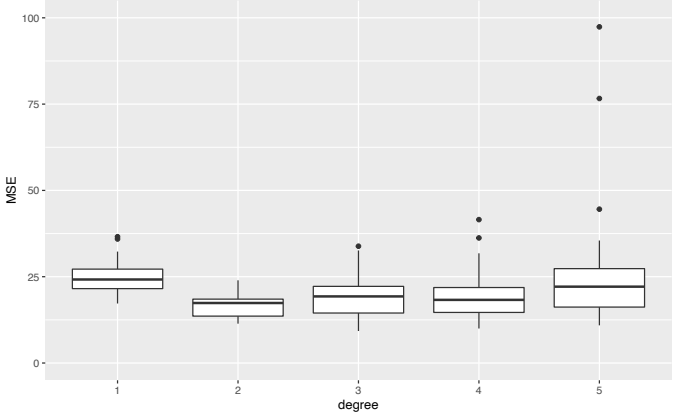


TRAINING VS. TEST ERROR

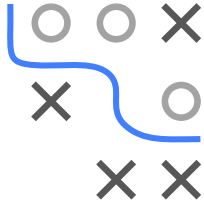


The test error...

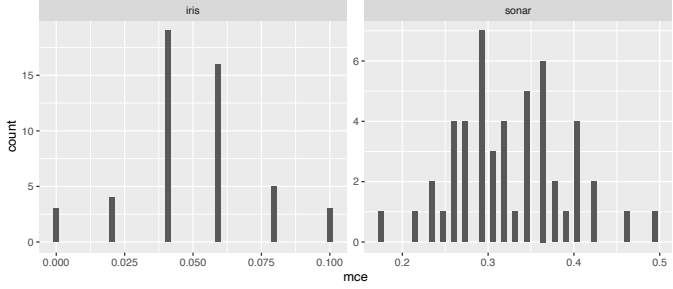
- will have higher variance with increasing model complexity.



BIAS AND VARIANCE



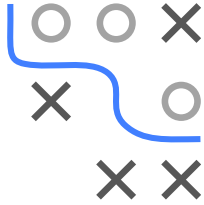
- Test error is a good estimator of GE, given a) we have enough data b) test data is representative i.i.d.
- Estimates for smaller test sets can fluctuate considerably – this is why we use resampling in such situations.
  Repeated $\frac{2}{3}$ / $\frac{1}{3}$ holdout splits:
  `iris` ($n = 150$) and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

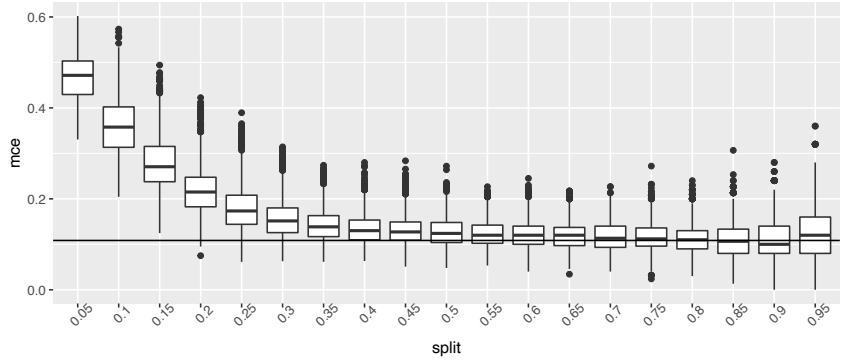
- Smaller training set $\rightarrow$ poor fit, pessimistic bias in $\widehat{\mathrm{GE}}$.
- Smaller test set $\rightarrow$ high variance.

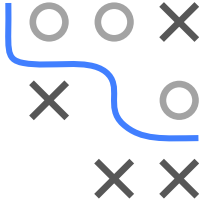


Experiment:

- `spirals` data ($sd = 0.1$), with CART tree.
- Goal: estimate real performance of a model with $|\mathcal{D}_{\text{train}}| = 500$.
- Split rates $s \in \{0.05, 0.10, ..., 0.95\}$ 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

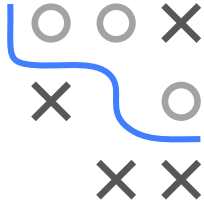


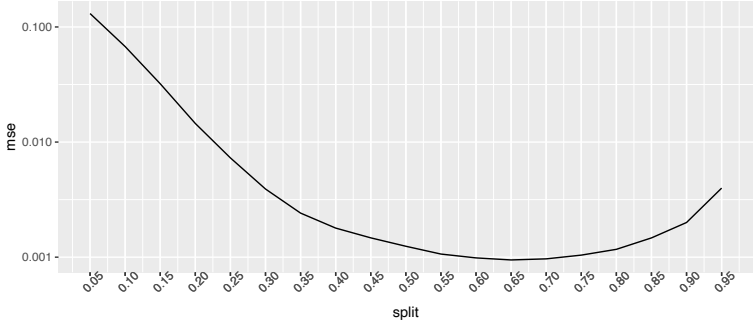


- Clear pessimistic bias for small training sets – we learn a much worse model than with 500 observations.
- But increase in variance when test sets become smaller.

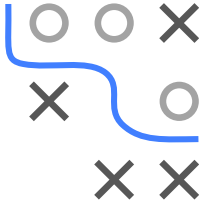
BIAS-VARIANCE OF HOLD-OUT – EXPERIMENT

- 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.

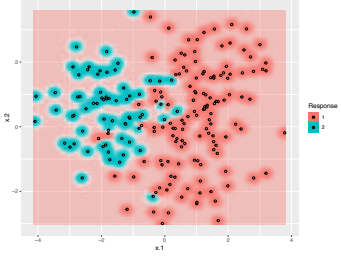




Introduction to Machine Learning



Evaluation
Overfitting and Underfitting

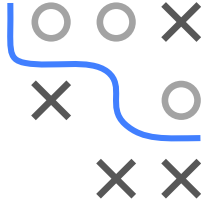


Learning goals

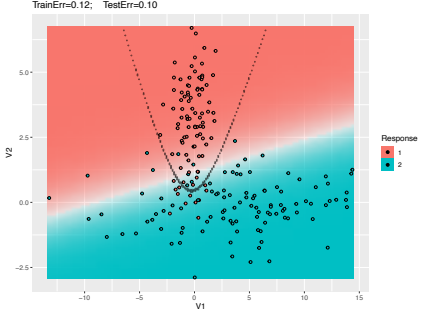
- Understand definitions of overfitting and underfitting

UNDERFITTING

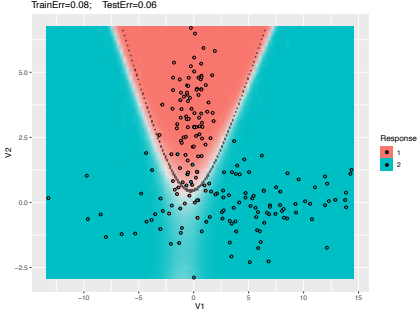
- Occurs if model does not reflect true shape of underlying function
- Hence, predictions will be less good as they could be
- High train error and high test error
- Hard to detect, as we don't know what the Bayes error is for a task



Underfitted model

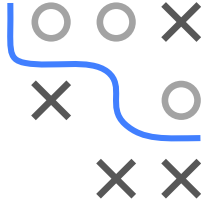


Appropriate model

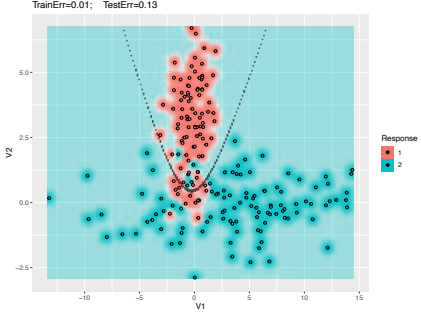


OVERFITTING

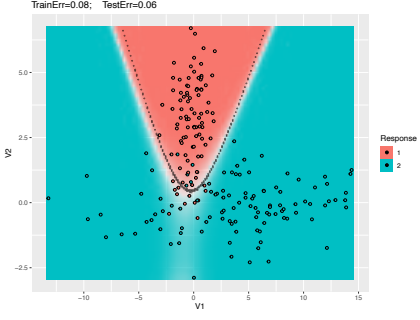
- Overfitting occurs when the model reflects noise or artifacts in training data, which do not generalize
- Small train error, at cost of test high error
- Hence, predictions of overfitting models cannot be trusted - but proper ML evaluation workflows should make it visible



Overfitted model

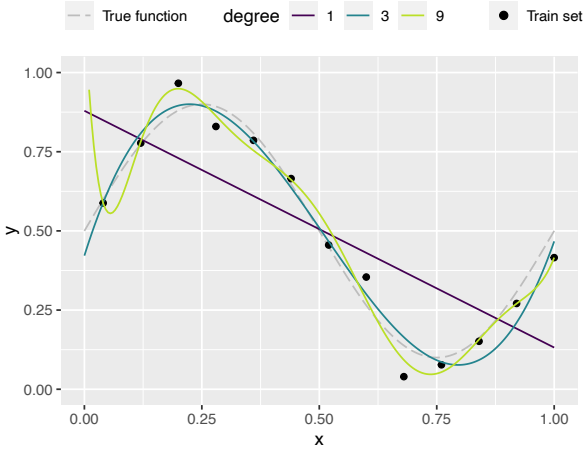


Appropriate model

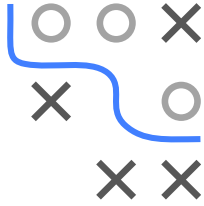


UNDER- AND OVERFITTING IN REGRESSION

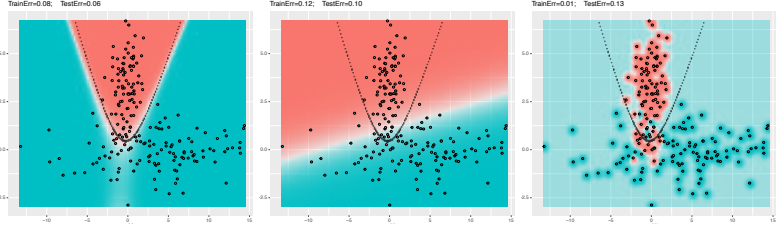
- Poly-Regression, on data from sinusoidal function
- LM underfits, high-d overfits



MATHEMATICAL DEFINITIONS



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

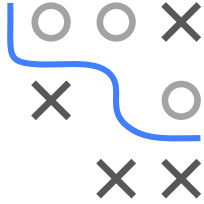


NB: Now, RHS is both UF and OF, let's say OF has "prio".

OVERFITTING TRADE-OFFS

The potential for overfitting is influenced by:

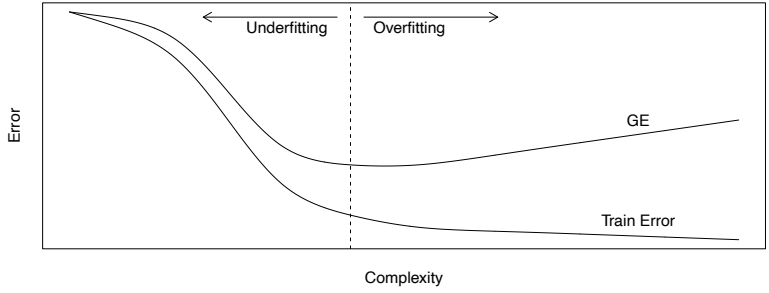
- Complexity of hypothesis space
- Amount of training data
- Dimensionality of feature space
- Irreducible noise

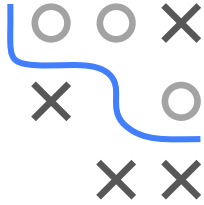


Implications:

- 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

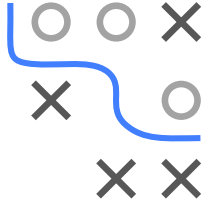




- Common U-shape of GE if complexity or train-rounds go up.
- Optimal level of complexity:
  Simplest model for which GE is not significantly outperformed
- We could also call "Point of OF" the point where GE goes up.

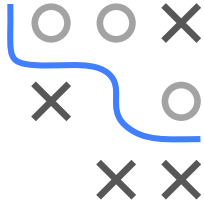
AVOIDING OVERFITTING

- 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.

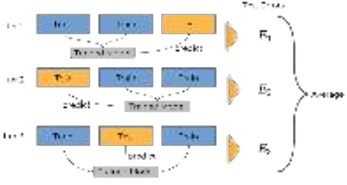


All of the above are methods of regularization, which we study in a dedicated chapter.

Introduction to Machine Learning



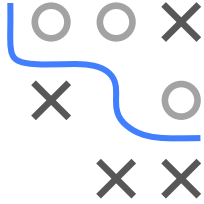
Evaluation
Resampling 1



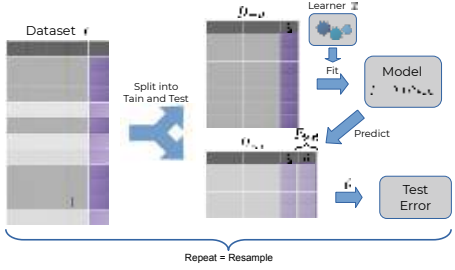
Learning goals

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

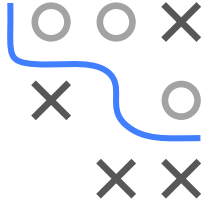
RESAMPLING



- **Goal**: estimate $\mathrm{GE}(\mathcal{I}, \boldsymbol{\lambda}, n, \rho_L) = \mathbf{E}\left[L(y, \mathcal{I}(\mathcal{D}_{\text{train}}, \boldsymbol{\lambda})(\mathbf{x}))\right]$.
- 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 $\mathrm{GE}(\mathcal{I}, \boldsymbol{\lambda}, n_{\text{train}}, \rho)$ as a proxy for $\mathrm{GE}(\mathcal{I}, \boldsymbol{\lambda}, n, \rho)$)
- And reduce var from small testsets via averaging over repetitions.



RESAMPLING STRATEGIES



- Represent train and test sets by index vectors::
  $J_{\text{train}} \in \{1, \ldots, n\}^{n_{\text{train}}}$ and $J_{\text{test}} \in \{1, \ldots, n\}^{n_{\text{test}}}$
- Resampling strategy = collection of splits:

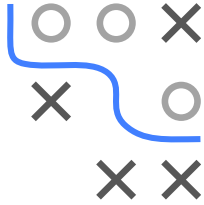
$$\mathcal{J} = \left((J_{\text{train},1}, J_{\text{test},1}), \ldots, (J_{\text{train},B}, J_{\text{test},B}) \right).$$

- Resampling estimator:

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

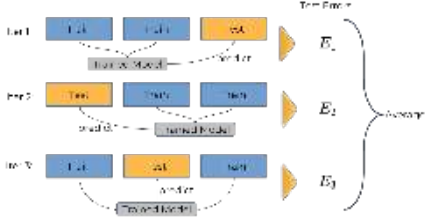
- Aggregation agr is typically "mean" and $n_{\text{train}} \approx n_{\text{train},1} \approx \cdots \approx n_{\text{train},B}$.

CROSS-VALIDATION



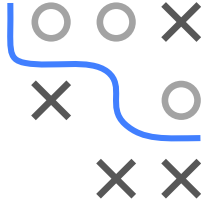
- Split the data into k roughly equally-sized partitions.
- Each part is test set once, join $k-1$ parts for training.
- Obtain k test errors and average.
- Fraction $(k-1)/k$ is used for training, so 90% for 10CV
- Each observation is tested exactly once.

Example: 3-fold CV

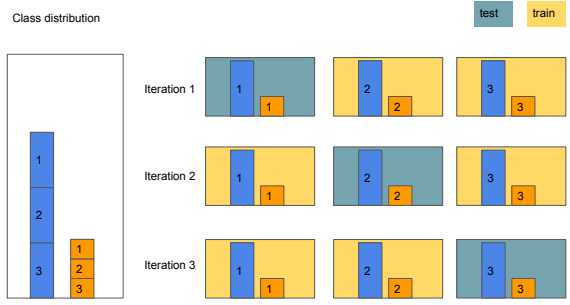


CROSS-VALIDATION - STRATIFICATION

- 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

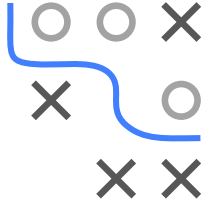


Example: stratified 3-fold cross-validation

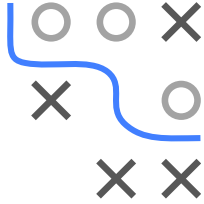


CROSS-VALIDATION

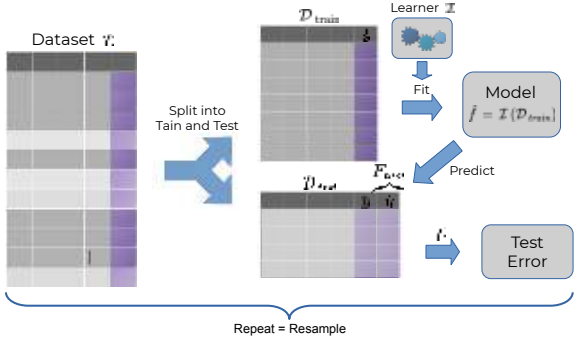
- 5 or 10 folds are common.
- $k = n$ is known as "leave-one-out" CV (LOO-CV)
- Bias of $\widehat{\mathrm{GE}}$: The more folds, the smaller. LOO nearly unbiased.
- LOO has high var, better many folds for small data but not LOO
- Repeated CV (avg over high-fold CVs) good for for small data.



SUBSAMPLING

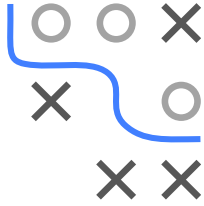


- Repeated hold-out with averaging, a.k.a. Monte Carlo CV.
- Typical choices for splitting: $\frac{4}{5}$ or $\frac{9}{10}$ for training.

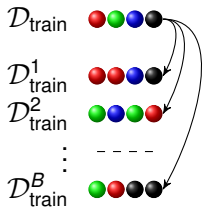


- Smaller subsampling rate = larger pessimistic bias
- More reps = smaller var

BOOTSTRAP



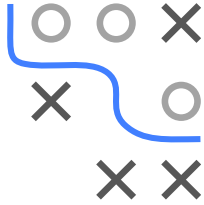
- Draw B trainsets of size n with replacement from orig $\mathcal{D}$
- Testsets = Out-Of-Bag points: $\mathcal{D}^b_{\text{test}} = \mathcal{D} \setminus \mathcal{D}^b_{\text{train}}$

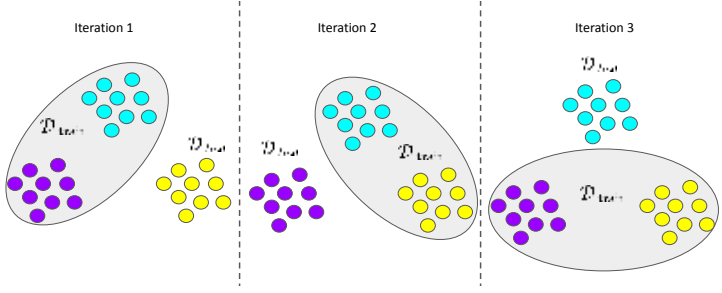


- Similar analysis as for subsampling
- Trainsets contain about 2/3 unique points:
  $1 - \mathbb{P}((\mathbf{x}, y) \notin \mathcal{D}_{\text{train}}) = 1 - \left(1 - \frac{1}{n}\right)^n \overset{n\to\infty}{\longrightarrow} 1 - \frac{1}{e} \approx 63.2\%$
- Replicated train points can lead to problems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

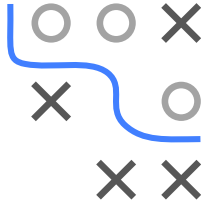
LEAVE-ONE-OBJECT-OUT

- 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 $\widehat{\mathrm{GE}}$
- CV on objects, or leave-one-object-out

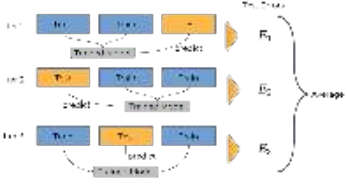




Introduction to Machine Learning



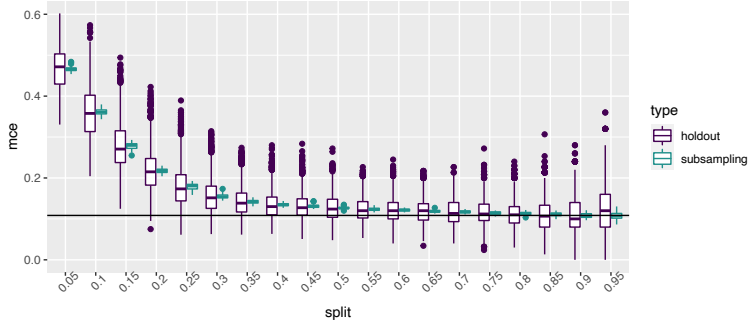
Evaluation
Resampling 2

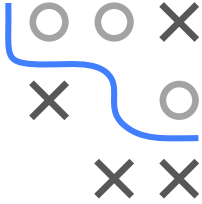


Learning goals

- 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

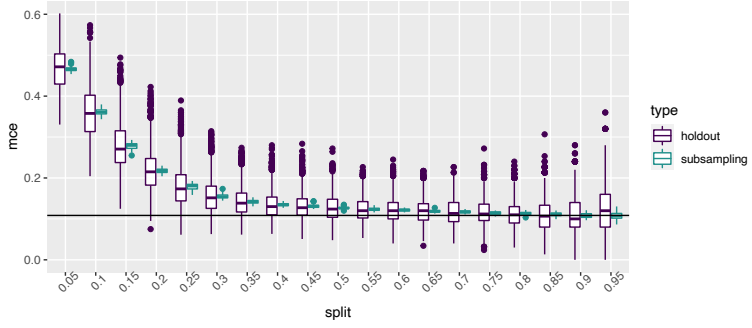
BIAS-VARIANCE ANALYSIS FOR SUBSAMPLING

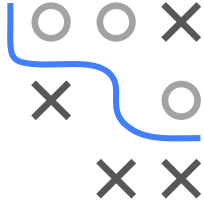




- Reconsider bias-var experiment for holdout (maybe re-read)
- Split rates $s \in \{0.05, 0.1, ..., 0.95\}$ with $|\mathcal{D}_{\text{train}}| = s \cdot 500$.
- Holdout vs. subsampling with 50 iters
- 50 replications

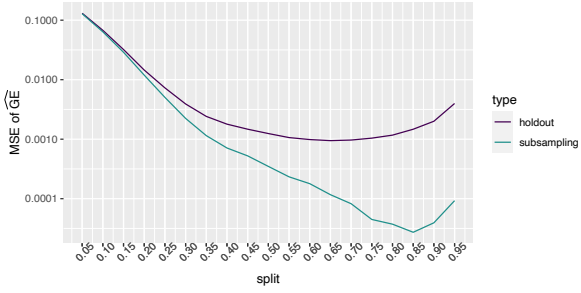
BIAS-VARIANCE ANALYSIS FOR SUBSAMPLING

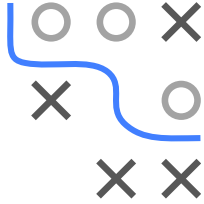




- Both estimators are compared to "real" MCE (black line)
- SS same pessimistic bias as holdout for given s, but much less var

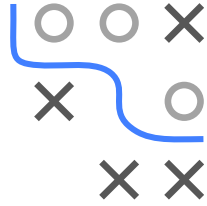
BIAS-VARIANCE ANALYSIS FOR SUBSAMPLING





- MSE of $\widehat{\mathrm{GE}}$ 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

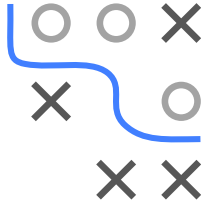


- Goal: estimate $\mathrm{GE}\left(\hat{f}\right) = \mathbb{E}\left[L\left(y, \hat{f}(\mathbf{x})\right)\right]$ via

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

  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 $\widehat{\mathrm{GE}}\left(\hat{f}\right)$ with Gaussian.
- $\mathbb{E}[\widehat{\mathrm{GE}}\left(\hat{f}\right)] = \mathbb{E}[L\left(y, \hat{f}(\mathbf{x})\right)] = \mathrm{GE}\left(\hat{f}\right)$
- $\mathbb{V}[\widehat{\mathrm{GE}}\left(\hat{f}\right)] = \frac{1}{m}\mathbb{V}[L\left(y, \hat{f}(\mathbf{x})\right)]$
- So $\widehat{\mathrm{GE}}\left(\hat{f}\right)$ is unbiased estimator of $\mathrm{GE}\left(\hat{f}\right)$, 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 $\mathrm{GE}(\mathcal{I}, n)$ (surrogate for $\mathrm{GE}\left(\hat{f}\right)$ when $\hat{f}$ is fit on full $\mathcal{D}$, with $|\mathcal{D}| = n$) via resampling based estim $\widehat{\mathrm{GE}}(\mathcal{I}, n_{\text{train}})$

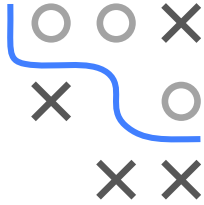
$$\widehat{\mathrm{GE}}(\mathcal{I}, \mathcal{J}, \rho, \boldsymbol{\lambda}) = \mathrm{agr}\Big(\rho\Big(\mathbf{y}_{J_{\text{test},1}}, \boldsymbol{F}_{J_{\text{test},1}, \mathcal{I}(\mathcal{D}_{\text{train},1}, \boldsymbol{\lambda})}\Big), \\ \vdots \\ \rho\Big(\mathbf{y}_{J_{\text{test},B}}, \boldsymbol{F}_{J_{\text{test},B},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

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

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

- NB1: In above, as always for $\mathrm{GE}(\mathcal{I})$, 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



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

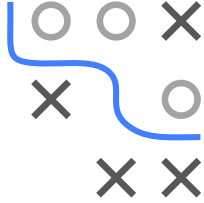
$$\mathbf{E}\left[\frac{1}{m} \sum_{(\mathbf{x},y) \in \mathcal{D}_{\text{test}}} L(y, \mathcal{I}(\mathcal{D}_{\text{train}})(\mathbf{x}))\right] = \mathrm{GE}(\mathcal{I}, n_{\text{train}})$$

$\Rightarrow$

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

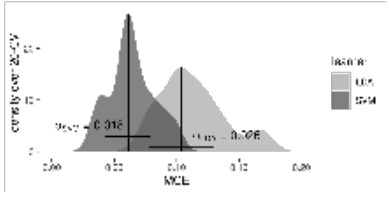
NO INDEPENDENCE OF CV RESULTS

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



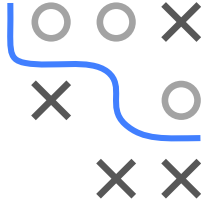
DON'T DO THIS AT HOME

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.



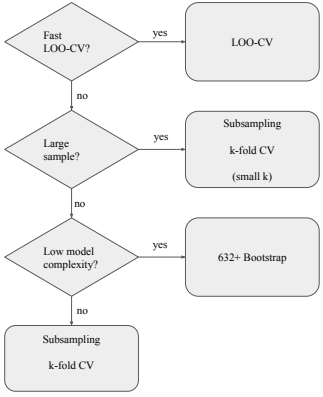
LDA vs SVM on `spam` classification problem, performance estimation via 20-CV w.r.t. MCE.

NO INDEPENDENCE OF CV RESULTS

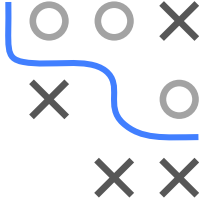


- $\mathbb{V}[\widehat{\mathrm{GE}}]$ of CV is a difficult combination of
  - average variance as we estim on finite trainsets
  - covar from test errors, as models result from overlapping trainsets
  - covar due to the dependence of trainsets and test obs appear in trainsets
- Naively using the empirical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

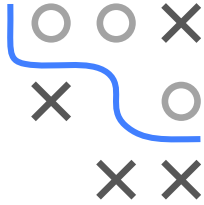
SHORT GUIDELINE



- 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

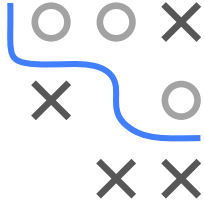
	True Class y	
Pred. $\hat{y}$	True Positive (TP)	False Positive (FP)
	False Negative (FN)	True Negative (TN)

Learning goals

- 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:

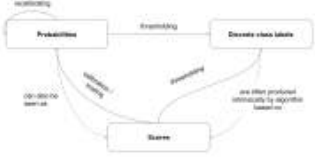
$$\boldsymbol{F} = \left(\hat{o}_k^{(i)}\right)_{i \in \{1,\ldots,m\}, k \in \{1,\ldots,g\}} \in \mathbb{R}^{m \times g},$$

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

2. Class probabilities:

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

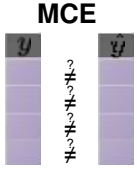
$\rightarrow$ These form the basis for evaluation.

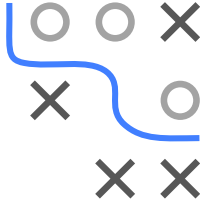


LABELS: MCE & ACC

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

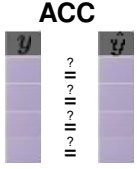
$$\rho_{MCE} = \frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \neq \hat{y}^{(i)}] \in [0, 1].$$





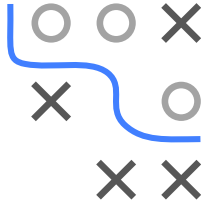
Accuracy (ACC) is defined in a similar fashion for correct classifications:

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



- 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**:

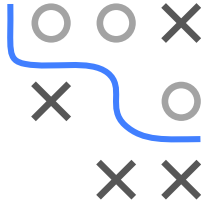
- true classes in columns,
- predicted classes in rows.

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

True classes

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

LABELS: CONFUSION MATRIX



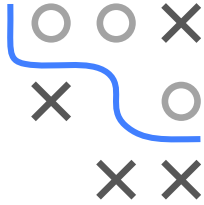
- In binary classification, we typically call one class "positive" and the other "negative".
- The positive class is the more important, often smaller one.

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

e.g.,

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

LABELS: COSTS



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

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

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

Predict if a person is solvent (yes / no).
Should the bank lend them the money?

Examplary costs:
Loss in event of default: 10,000 EUR
Income through interest paid: 100 EUR

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

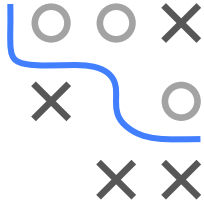
LABELS: COSTS

Cost matrix

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

Confusion matrix

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



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

$$\begin{aligned} \textit{Costs} &= \frac{1}{n}\sum_{i=1}^{n} C[y^{(i)}, \hat{y}^{(i)}] \\ &= \frac{1}{100}\left(100 \cdot 7 + 0 \cdot 70 + 10.000 \cdot 3 + 0 \cdot 20\right) = 307 \end{aligned}$$

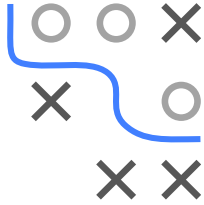
- If the bank gives everyone a credit, the costs are at:

$$\textit{Costs} = \frac{1}{100}\left(100 \cdot 0 + 0 \cdot 77 + 10.000 \cdot 23 + 0 \cdot 0\right) = 2.300$$

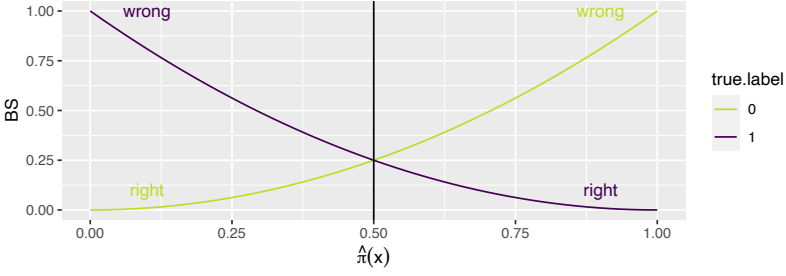
PROBABILITIES: BRIER SCORE

Measures squared distances of probabilities from the true class labels:

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

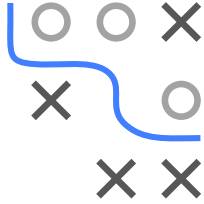


- Fancy name for MSE on probabilities.
- Usual definition for binary case; $y^{(i)}$ must be encoded as 0 and 1.



PROBABILITIES: BRIER SCORE / 2

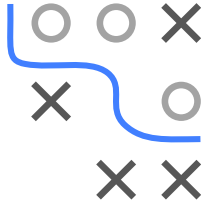
$$\rho_{BS,MC} = \frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{g} \left(\hat{\pi}_k^{(i)} - o_k^{(i)} \right)^2$$



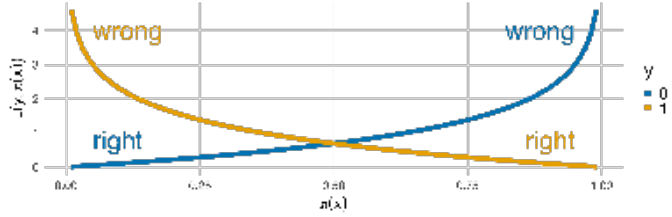
- Original by Brier, works also for multiple classes.
- $o_k^{(i)} = [y^{(i)} = k]$ marks the one-hot-encoded class label.
- For the binary case, $\rho_{BS,MC}$ is twice as large as ρ_{BS}: in $\rho_{BS,MC}$, we sum the squared difference for each observation regarding both class 0 **and** class 1, not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

PROBABILITIES: LOG-LOSS

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

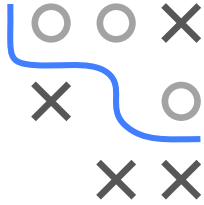


$$\rho_{LL} = \frac{1}{m}\sum_{i=1}^{m}\left(-y^{(i)}\log\left(\hat{\pi}^{(i)}\right) - \left(1-y^{(i)}\right)\log\left(1-\hat{\pi}^{(i)}\right)\right).$$



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

Introduction to Machine Learning



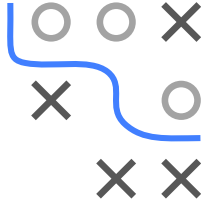
Evaluation
ROC Basics

		True Class y		
		$+$	$-$	
Pred. $\hat{y}$	$+$	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 = $\frac{TP+TN}{TOTAL}$

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

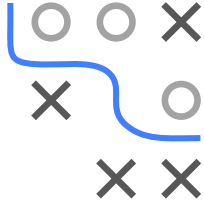
CLASS IMBALANCE

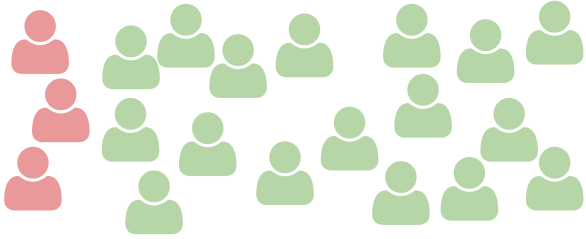


- Assume a binary classifier diagnoses a serious medical condition.
- Label distribution is often **imbalanced**, i.e, not many people have the disease.
- Evaluating on mce is often inappropriate for scenarios with imbalanced labels:
  - Assume that only 0.5 % have the disease.
  - Always predicting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

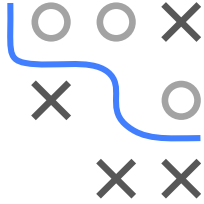
CLASS IMBALANCE

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





IMBALANCED COSTS

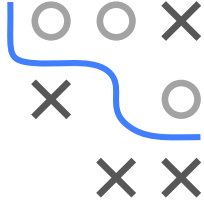


- Another point of view is **imbalanced costs**.
- In our example, classifying a sick patient as healthy should incur a much higher cost than classifying a healthy patient as sick.
- The costs depend a lot on what happens next: we can well assume that our system is some type of screening filter, and 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.

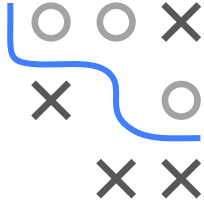




- Problem: if we were able to specify costs precisely, we could evaluate or even optimize on them.
- This important subfield of ML is called **cost-sensitive learning**, which we will not cover in this lecture unit.
- Unfortunately, users find it notoriously hard to come up with precise cost figures 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.

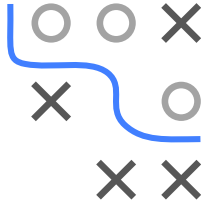




http://media.iwm.org.uk/iwm/mediaLib//39/media-39665/large.jpg

LABELS: ROC METRICS

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

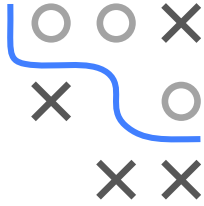


		True Class y		
		$+$	$-$	
Pred.	$+$	TP	FP	$\rho_{PPV} = \frac{\text{TP}}{\text{TP+FP}}$
$\hat{y}$	$-$	FN	TN	$\rho_{NPV} = \frac{\text{TN}}{\text{FN+TN}}$
		$\rho_{TPR} = \frac{\text{TP}}{\text{TP+FN}}$	$\rho_{TNR} = \frac{\text{TN}}{\text{FP+TN}}$	$\rho_{ACC} = \frac{\text{TP+TN}}{\text{TOTAL}}$

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

LABELS: ROC METRICS

Example:

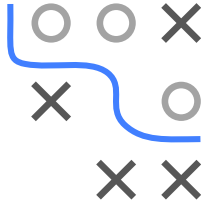


		Actual Class y		
		Positive	Negative	
$\hat{y}$ Pred.	Positive	**True Positive** (TP) = 20	**False Positive** (FP) = 180	Positive predictive value = TP / (TP + FP) = 20 / (20 + 180) = **10%**
	Negative	**False Negative** (FN) = 10	**True Negative** (TN) = 1820	Negative predictive value = TN / (FN + TN)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

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

MORE METRICS AND ALTERNATIVE TERMINOLOGY

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

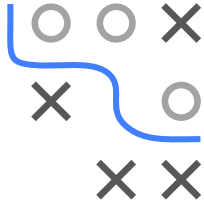


		True condition			
	Total population	Condition positive	Condition negative	Prevalence = Σ Condition positive / Σ Total population	Accuracy (ACC) = (Σ True positive + Σ True negative) / Σ Total population
Predicted condition	Predicted condition positive	True positive, Power	False positive, Type I error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

▸ Clickable version/picture source

▸ Interactive diagram

LABELS: F_1 MEASURE



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

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

1. Maximizing positive predictive value
2. Maximizing true positive rate

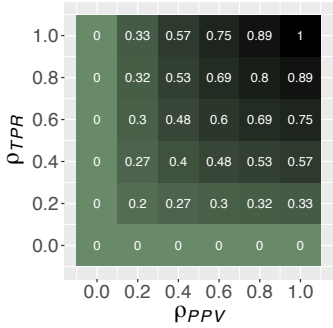
ρ_{F_1} is the harmonic mean of ρ_{PPV} and ρ_{TPR}:

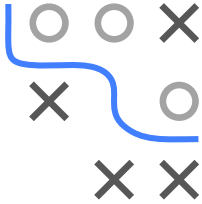
$$\rho_{F_1} = 2 \cdot \frac{\rho_{PPV} \cdot \rho_{TPR}}{\rho_{PPV} + \rho_{TPR}}$$

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

LABELS: F_1 MEASURE

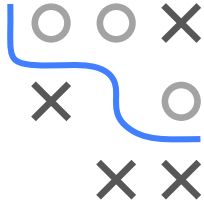
F_1 score for different combinations of ρ_{PPV} & ρ_{TPR}.
$\rightarrow$ Tends more towards the lower of the two combined values.



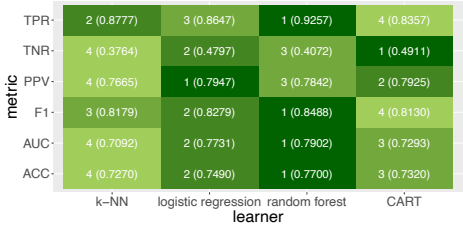


- A model with $\rho_{TPR} = 0$ (no positive instance predicted as positive) or $\rho_{PPV} = 0$ (no true positives among the predicted) has $\rho_{F_1} = 0$.
- Always predicting "negative": $\rho_{F_1} = 0$.
- Always predicting "positive":
  $\rho_{F_1} = 2 \cdot \rho_{PPV} / (\rho_{PPV} + 1) = 2 \cdot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

WHICH METRIC TO USE?

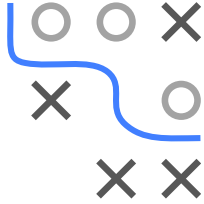


- As we have seen, there is a plethora of methods.
  $\rightarrow$ This leaves practitioners with the question of which to use.
- Consider a small benchmark study.
  - We let k-NN, logistic regression, a classification tree, and a random forest compete on classifying the `credit risk` data.
  - The data 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):

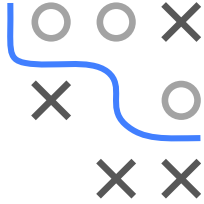


WHICH METRIC TO USE?

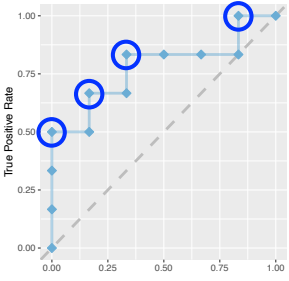
- We need not expect overly large discrepancies in general, but neither will we always see an unambiguous picture.
- Different metrics emphasize different aspects of performance.
  $\rightarrow$ The choice should be made in the domain context.
- For practitioners it is vital to understand what should 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

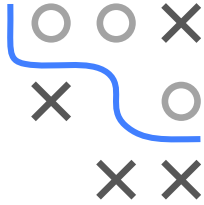


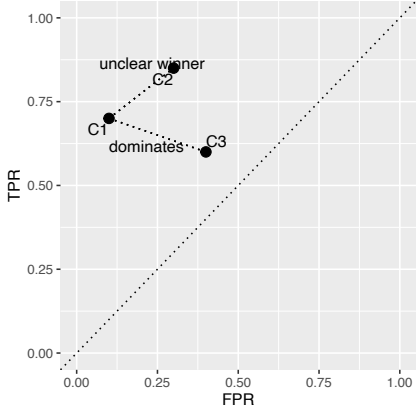
Learning goals

- 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.





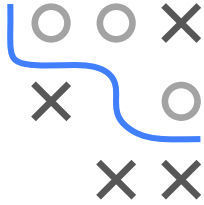
		True Class y	
		$+$	$-$
Pred. $\hat{y}$	$+$	TP	FP
	$-$	FN	TN

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

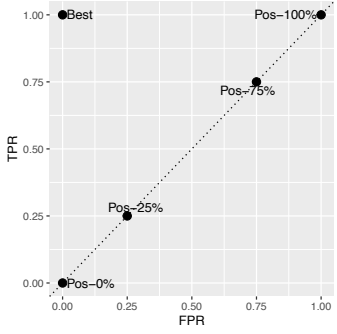
$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

LABELS: ROC SPACE

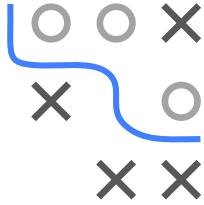
- The best classifier lies on the top-left corner, where FPR equals 0 and TPR is maximal.
- The diagonal is worst as it corresponds to a classifier producing random labels (with different proportions).



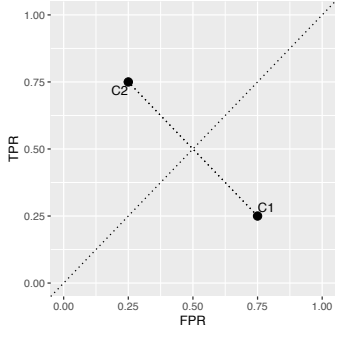
- If each positive x will be randomly classified with 25% as "pos", TPR $= 0.25$.
- If we assign each negative x randomly to "pos", FPR $= 0.25$.



LABELS: ROC SPACE

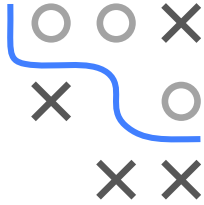


- In practice, we should never obtain a classifier below the diagonal.
- Inverting the predicted labels ($0 \mapsto 1$ and $1 \mapsto 0$) will result in a reflection at the diagonal.
  $\Rightarrow \text{TPR}_{\text{new}} = 1 - \text{TPR}$ and $\text{FPR}_{\text{new}} = 1 - \text{FPR}$.



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

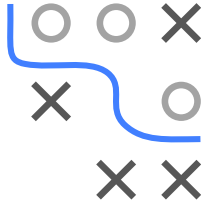
$\text{MCE} = 45/150 = 0.3$
$\text{TPR} = 0.8$
$\text{FPR} = 0.5$

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

FROM PROBABILITIES TO LABELS: ROC CURVE

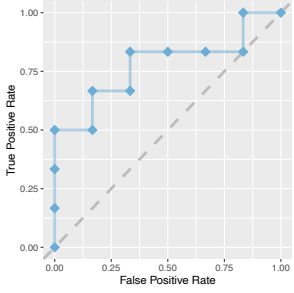
Remember: Both probabilistic and scoring classifiers can output classes by thresholding:

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



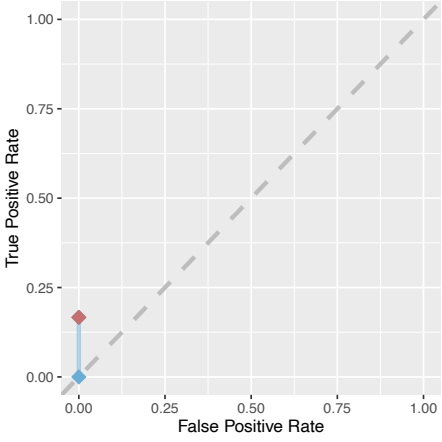
To draw a ROC curve:

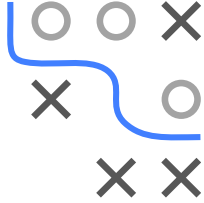
1. Rank test observations on decreasing score.
2. Start with $c = 1$, so we start in $(0, 0)$; we predict everything as negative.
3. Iterate through all possible thresholds c and proceed for each observation x as follows:
   - If x is positive, move TPR $1/n_+$ up, as we have one TP more.
   - If x is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



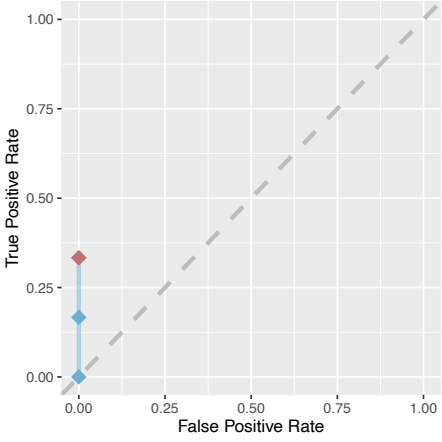


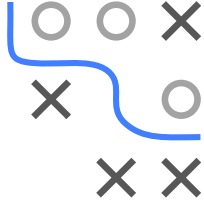
$c = 0.9$
$\rightarrow$ TPR = 0.167
$\rightarrow$ FPR = 0



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



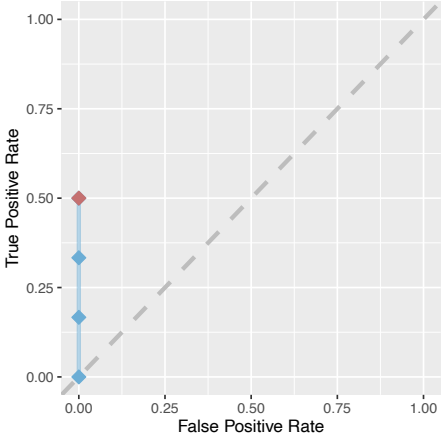


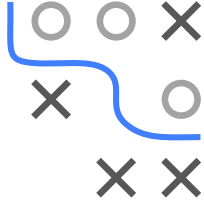
$c = 0.85$
$\rightarrow$ TPR = 0.333
$\rightarrow$ FPR = 0



DRAWING ROC CURVES

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



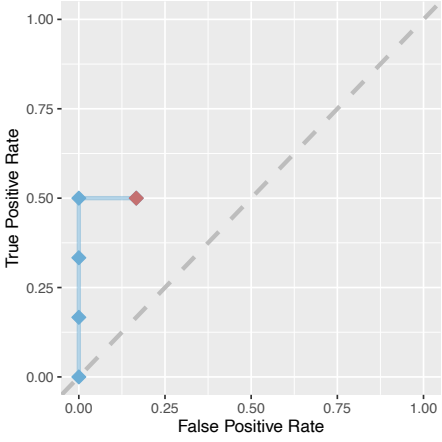


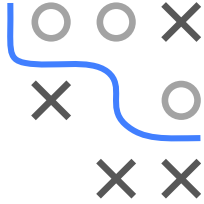
$c = 0.66$
$\rightarrow$ TPR = 0.5
$\rightarrow$ FPR = 0



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



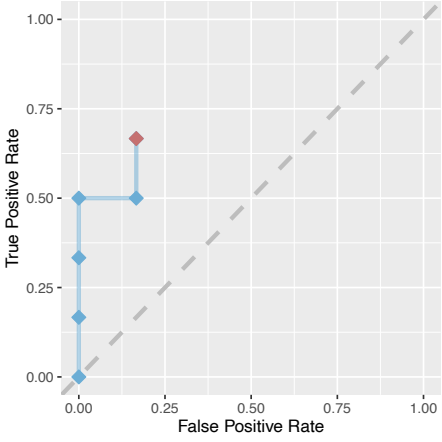


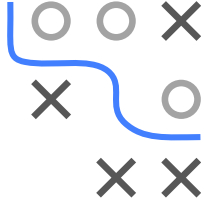
$c = 0.6$
$\rightarrow$ TPR = 0.5
$\rightarrow$ FPR = 0.167



DRAWING ROC CURVES

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





$c = 0.55$

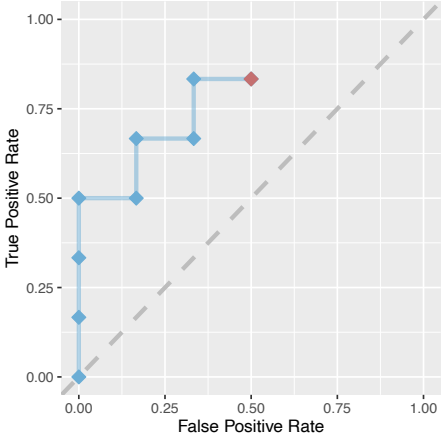
$\rightarrow$ TPR = 0.667

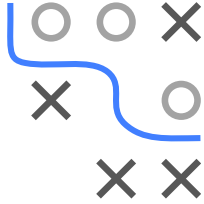
$\rightarrow$ FPR = 0.167



DRAWING ROC CURVES

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





$c = 0.3$

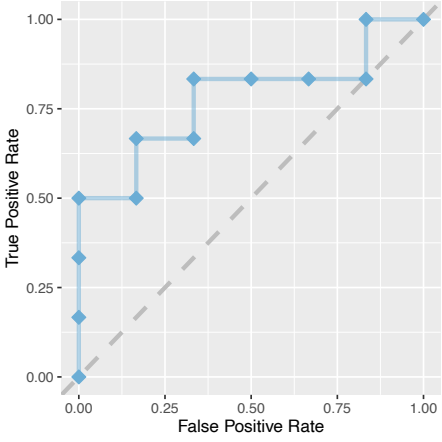
$\rightarrow$ TPR = 0.833

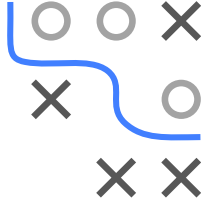
$\rightarrow$ FPR = 0.5



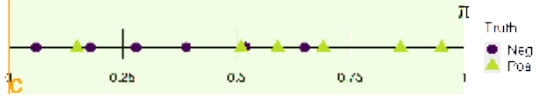
DRAWING ROC CURVES

#	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



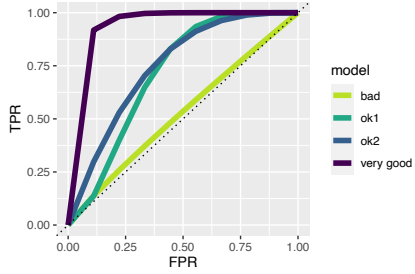


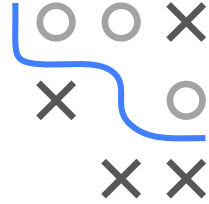
$c = 0$
$\rightarrow$ TPR = 1
$\rightarrow$ FPR = 1



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.

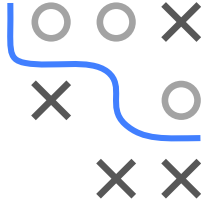




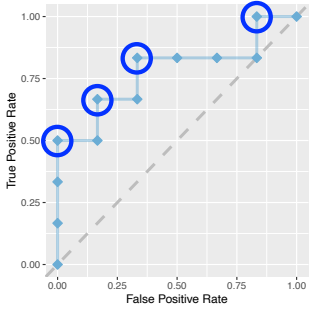
- Small thresholds will very liberally predict the positive class, and result in a potentially higher FPR, but also higher TPR.
- High thresholds will very conservatively predict the positive class, and result in a lower FPR and TPR.
- As we have not defined the trade-off between false positive and false negative costs, we cannot easily select the "best" threshold.
  $\rightarrow$ 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.

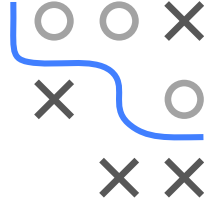


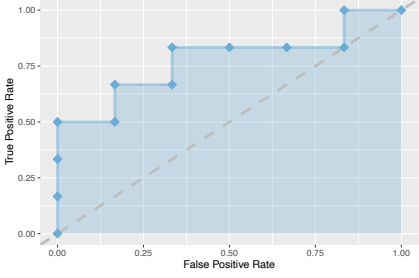
- Identify non-dominated points
- Assess TPR / FPR
- Decide which combo is best for task
- Pick associated threshold



AUC: AREA UNDER ROC CURVE

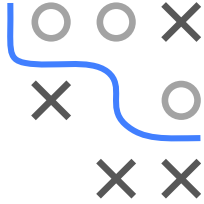
- AUC $\in [0, 1]$ is a single metric to evaluate scoring classifiers – independent of the chosen threshold.
  - AUC = 1: perfect classifier
  - AUC = 0.5: random, non-discriminant classifier
  - AUC = 0: perfect, with inverted labels





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	0.46
0	0.3
0	0.1

Choose a random positive

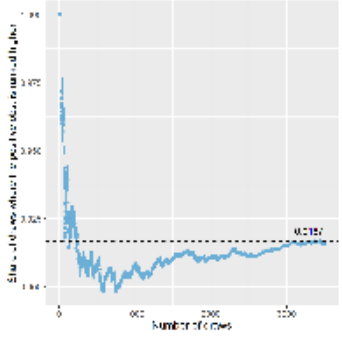
1	0.76

Choose a random negative

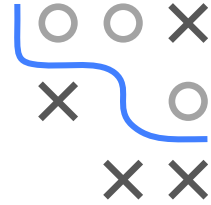
0	0.3

AUC = 0.9167

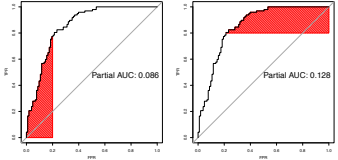
- Classifier ranks the positive higher than the negative
- This happens with a mean probability of 0.9167



Introduction to Machine Learning



Evaluation
Partial AUC

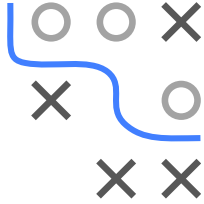


Learning goals

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

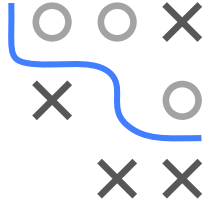
PARTIAL AUC

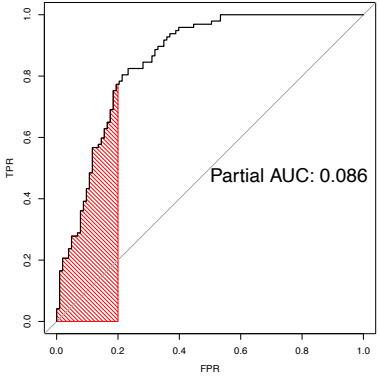
- TPR and FPR often treated asymmetrically in biomed contexts
- TPR = disease detection, is crucial
- But low FPR needed to avoid unnecessary treatments
- Common solution: Fix either TPR or FPR to a required value and optimize the other, but not easy to select exact point

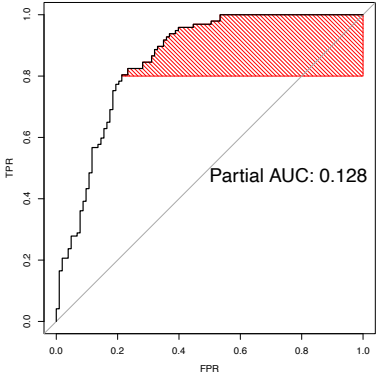


PARTIAL AUC

- 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



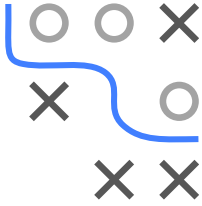




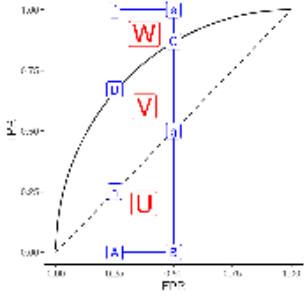
CORRECTED PARTIAL AUC

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

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

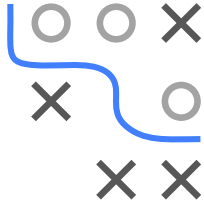


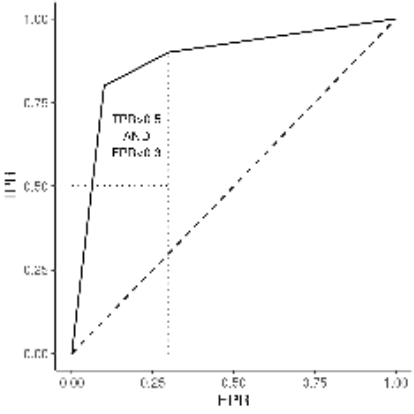
- pAUC is V+U = "A-B-C-D"
- pAUC_{min} is pAUC of random classifier, so U = "A-B-g-h"
- pAUC_{max} is U+V+W = "A-B-e-f"
- Compute percentage of V in V+W
- Rescale so random=0.5; optimal=1



2WAY PARTIAL AUC

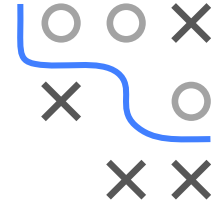
- Can also limit both TPR and FPR
- 2way pAUC = compute area under 2way limited segment





Introduction to Machine Learning

Evaluation
Multi-Class AUC



AUC(pos|neg) = AUC(1|3)

	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)

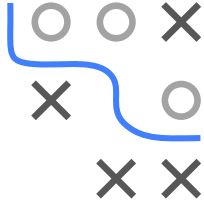
	Y			
neg	1	0.7	0.2	0.1
neg	1	0.5	0.3	0.2
~~2~~	~~0.3~~	~~0.5~~	~~0.2~~	
~~2~~	~~0.4~~	~~0.5~~	~~0.1~~	
pos	3	0.6	0.1	0.3
pos	3	0.1	0.1	0.8

Learning goals

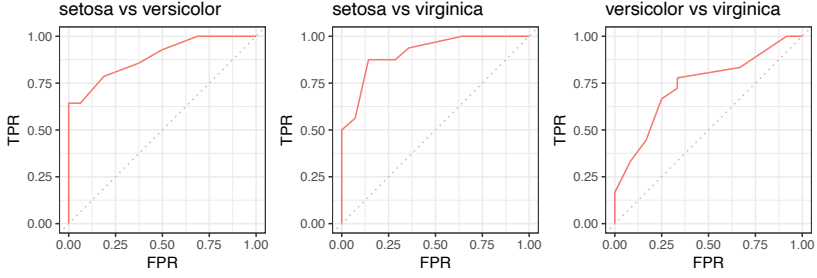
- Understand that generalizing AUC to multi-class is not trivial
- Learn how multi-class AUC can be derived

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

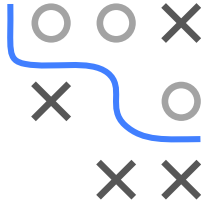


Example: 1-vs-1 on `iris`



MULTI-CLASS AUC

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



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

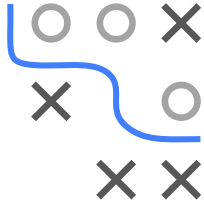
AUC(pos|neg) = AUC(3|1)

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

1. Subset pred rows to true classes 1 and 3
2. Use $k = 3$ as pos and $\ell = 1$ as neg class
3. Compute standard AUC with $\hat{\pi}_3$ as scores
4. 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: $\text{AUC}(k \mid \ell) \neq \text{AUC}(\ell \mid k)$
- **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)

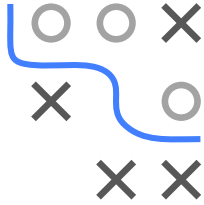
	Y	$\hat{\pi}_1$	$\hat{\pi}_2$	$\hat{\pi}_3$
neg	1	0.7	0.2	0.1
neg	1	0.5	0.3	0.2
	2	0.3	0.5	0.2
	2	0.4	0.5	0.1
pos	3	0.6	0.1	0.3
pos	3	0.1	0.1	0.8

MULTI-CLASS AUC

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

- For all class pairs, compute AUC($k \mid \ell$).

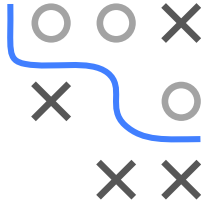
$$\text{AUC}_{MC} = \frac{1}{g(g-1)} \sum_{k \neq \ell} \text{AUC}(k|\ell) \in [0, 1].$$



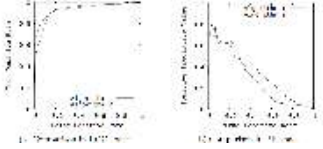
Comments:

- 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

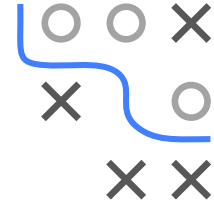


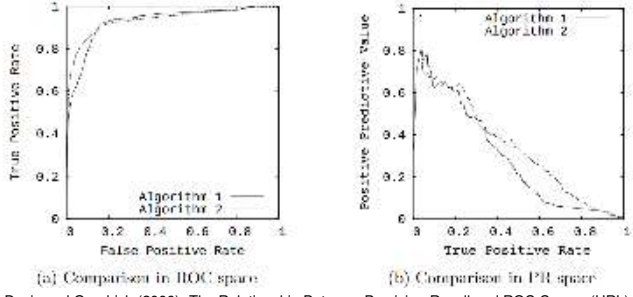
Learning goals

- 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



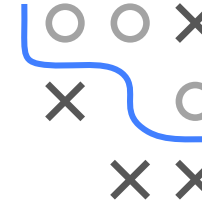


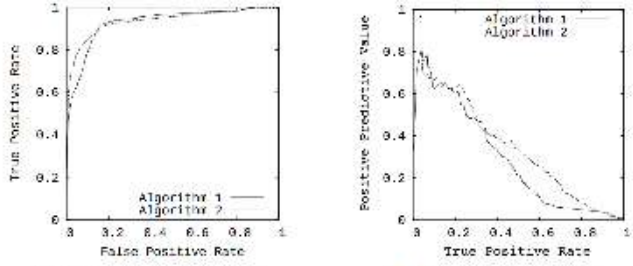
(a) Comparison in ROC space (b) Comparison in PR space

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



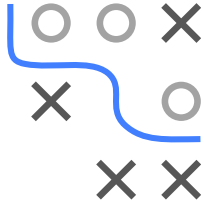


[a] Comparison in ROC space [b] Comparison in PR space

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

$\text{TPR} = 10/11$
$\text{FPR} = 0.001$
$\text{PPV} = 10/11$

FP=100:

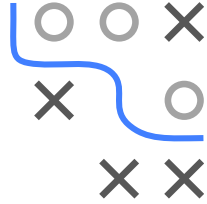
	True +1	True -1
Pred. +1	100	100
Pred. -1	10	9900
Total	110	10000

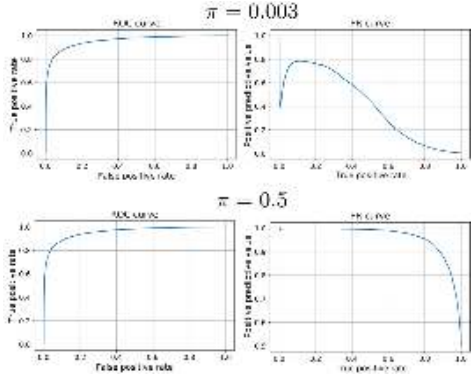
$\text{TPR} = 10/11$
$\text{FPR} = 0.01$
$\text{PPV} = 1/2$

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

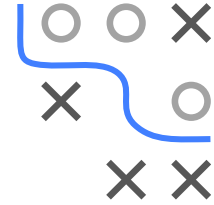


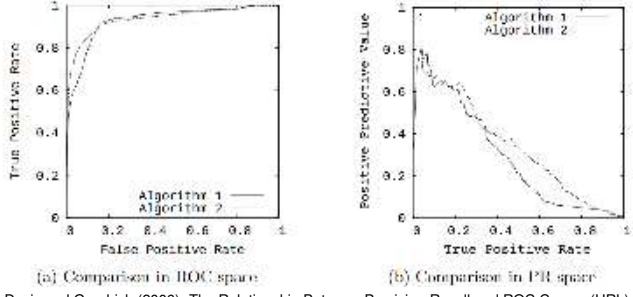


Wissam Siblini et. al. (2004): Master your Metrics with Calibration (URL).

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)

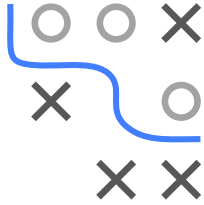




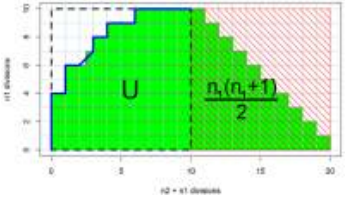
(a) Comparison in ROC space (b) Comparison in PR space

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

Introduction to Machine Learning



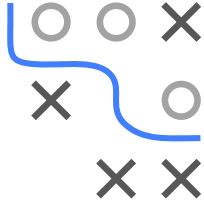
Evaluation
AUC & Mann-Whitney-U Test



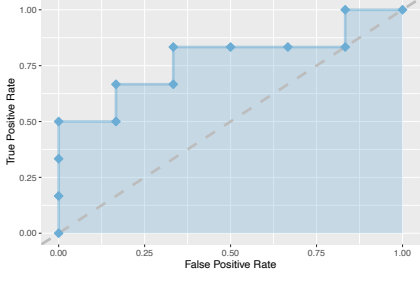
Learning goals

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

AUC AS A RANK-BASED METRIC



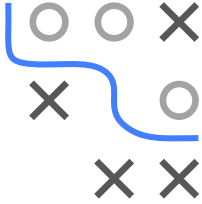
- The AUC metric is intimately related to the **Mann-Whitney-U test**, also known as **Wilcoxon rank-sum test**.
- This connection is best understood viewing the AUC from a slightly different angle: it is, in effect, a **rank-based** metric.
- Recall that, constructing the ROC curve, we count TP and 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 AUC = 1.



Truth	Score
1	0.9
1	0.76
1	0.7
0	0.5
1	0.46
0	0.3
0	0.1

Choose a random positive

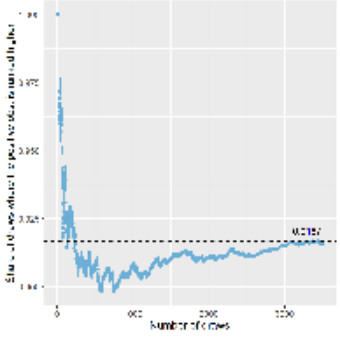
1	0.76

Choose a random negative

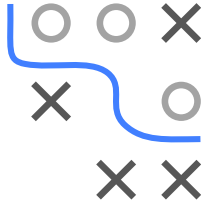
0	0.3

AUC = 0.9167

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



MANN-WHITNEY-U TEST

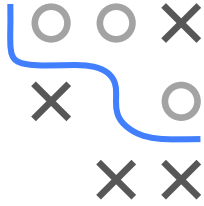


- The Mann-Whitney-U test is a **non-parametric hypothesis test** on the difference in location between two samples X_1, X_2 of sizes n_1 and n_2, respectively.
- Under the null, X_1 and X_2 follow the same (unknown) distribution $\mathbb{P}$, and for any pair of observations $x_{1,1} \in X_1$, $x_{2,1}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

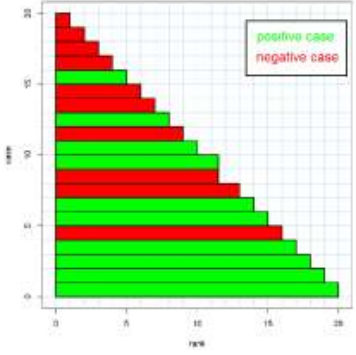
$$U = \frac{1}{n_1 n_2} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \mathbb{I}[x_{1,i} > x_{2,j}] = R_1 - \frac{n_1(n_1+1)}{2}$$

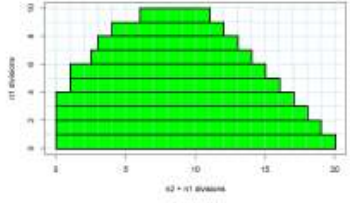
- For large samples, U is approximately normally distributed.

AUC & MANN-WHITNEY-U TEST

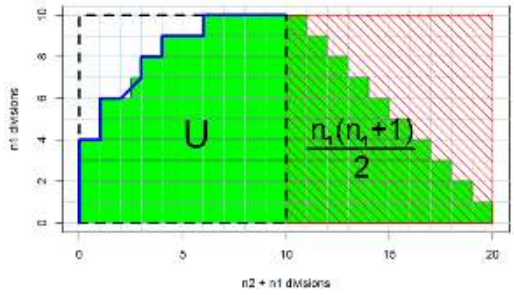


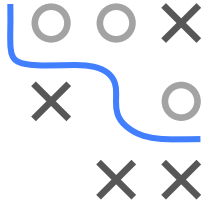
- We can directly interpret the AUC in the light of the U statistic.
- In order to see this, plot the ranks of all the scores as a stack of horizontal bars, and color them by label.
- Next, keep only the green ones, and slide them horizontally to get a nice even stairstep on the right edge:





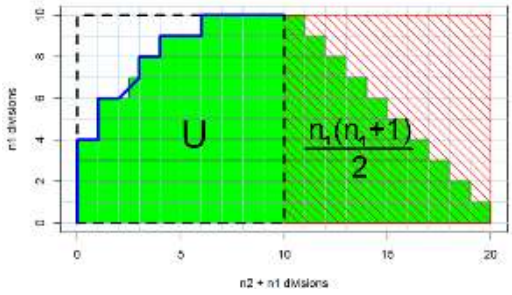
AUC & MANN-WHITNEY-U TEST

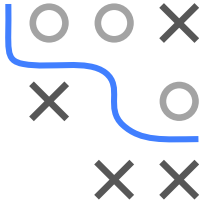




- Denoting the respective numbers of cases as n_+ and n_-, we have:
  $U = R_+ - \frac{n_+(n_+ + 1)}{2}$.
- The area of the green bars on the right is equal to $\frac{n_+(n_+ + 1)}{2}$.

AUC & MANN-WHITNEY-U TEST





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

$\Rightarrow$ 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

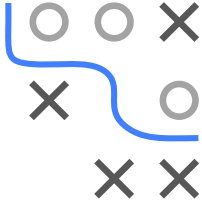
Classification and Regression Trees (CART)

Random Forests

Neural Networks

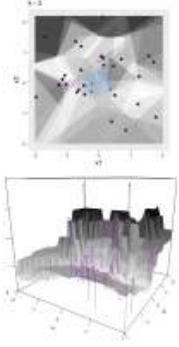
Tuning

Nested Resampling



Introduction to Machine Learning

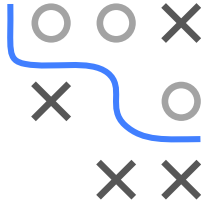
k-Nearest Neighbors



Learning goals

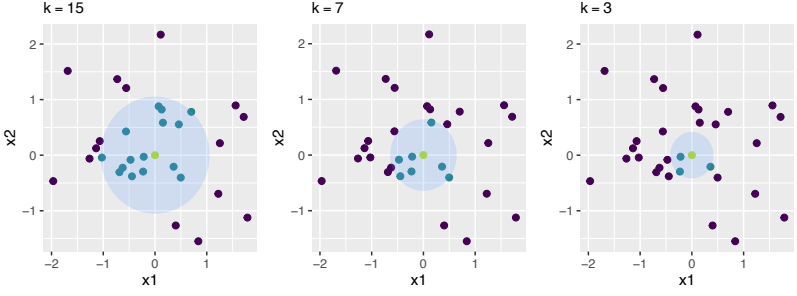
- Understand the basic idea of k-NN for regression and classification
- Understand that k-NN is a non-parametric, local model
- Know different distance measures for different scales of feature variables

K-NEAREST-NEIGHBORS

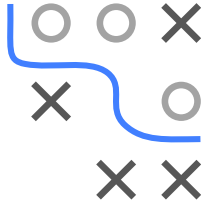


- k-**NN** can be used for regression and classification.
- Generates "similar" predictions for $\mathbf{x}$ to its k closest neighbors.
- "Closeness" requires a distance or similarity measure.
- The subset of $\mathcal{D}_{\text{train}}$ that is at least as close to $\mathbf{x}$ as its k-th closest neighbor $\mathbf{x}^{(k)}$ in $\mathcal{D}_{\text{train}}$ is called the k-**neighborhood** $N_k(\mathbf{x})$ of $\mathbf{x}$:

$$N_k(\mathbf{x}) = \{\mathbf{x}^{(i)} \in \mathcal{D}_{\text{train}} \mid d(\mathbf{x}^{(i)}, \mathbf{x}) \leq d(\mathbf{x}^{(k)}, \mathbf{x})\}$$



DISTANCE MEASURES



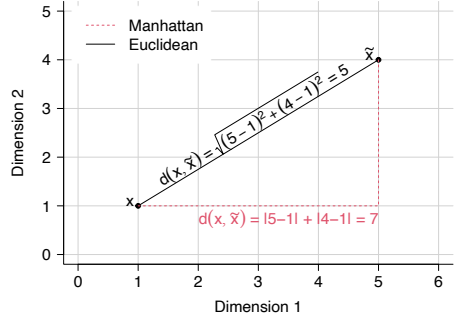
- Popular for numerical features: **Minkowski** distances of the form

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

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

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

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

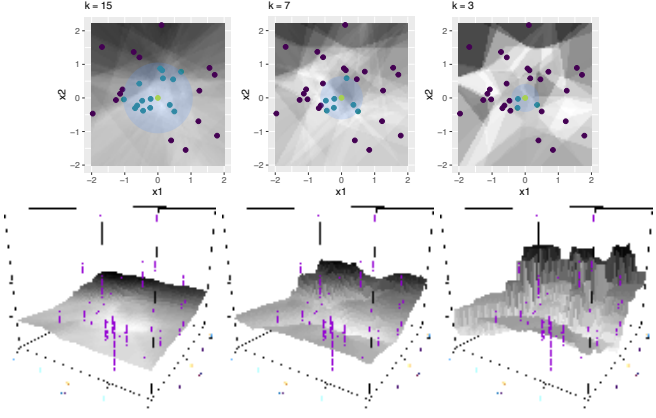


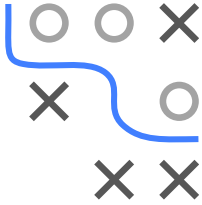
PREDICTION - REGRESSION

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

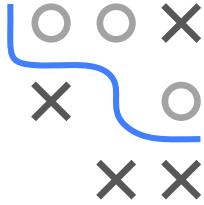




PREDICTION - CLASSIFICATION

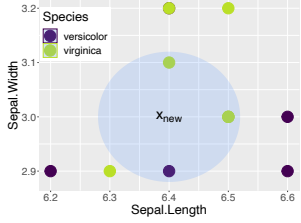
For classification in g groups, a majority vote is used:

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



And posterior probabilities can be estimated with:

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



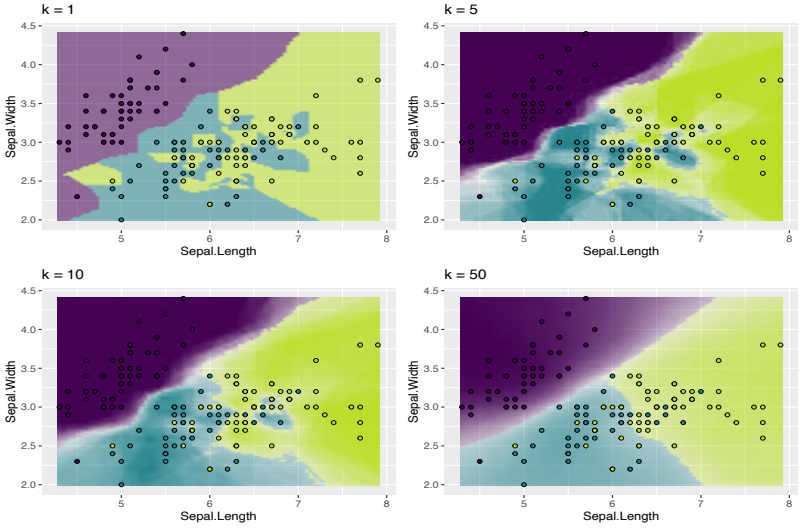
	SL	SW	Species	dist
52	6.4	3.2	versicolor	0.200
59	6.6	2.9	versicolor	0.224
75	**6.4**	**2.9**	**versicolor**	**0.100**
76	6.6	3.0	versicolor	0.200
98	6.2	2.9	versicolor	0.224
104	6.3	2.9	virginica	0.141
105	**6.5**	**3.0**	**virginica**	**0.100**
111	6.5	3.2	virginica	0.224
116	6.4	3.2	virginica	0.200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

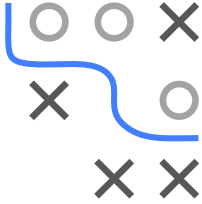
Example with subset of iris data ($k = 3$)

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

$\hat{h}(\mathbf{x}_{new}) = virginica$

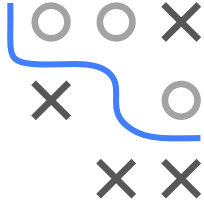
K-NN: FROM SMALL TO LARGE *K*





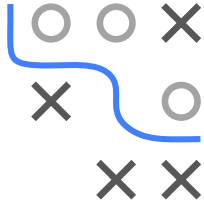
Complex, local model vs smoother, more global model

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

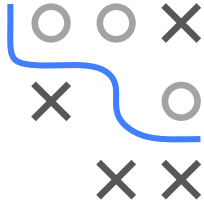


- **Standardization:** Features in k-NN are usually standardized or normalized. If two features have values on a very different range, most distances would place a higher importance on the one with a larger range, leading to an imbalanced influence of that feature.
- **Importance:** Sometimes one feature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

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

- 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.

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

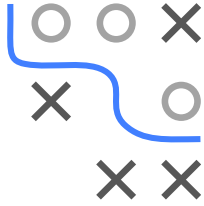
- $\delta_{x_j,\tilde{x}_j}$ is 0 or 1. It's 0 if *j*-th feature is *missing* in at least one observation, or when the feature is asymmetric binary (where "1" is more important than "0") and both values are zero. Otherwise 1.
- $d_{gower}(x_j, \tilde{x}_j)$: For nominals it's 0 if both values are equal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

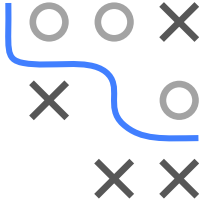
Classification and Regression Trees (CART)

Random Forests

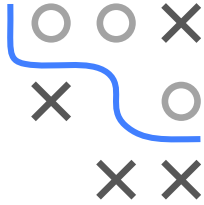
Neural Networks

Tuning

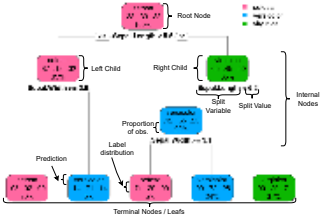
Nested Resampling



Introduction to Machine Learning



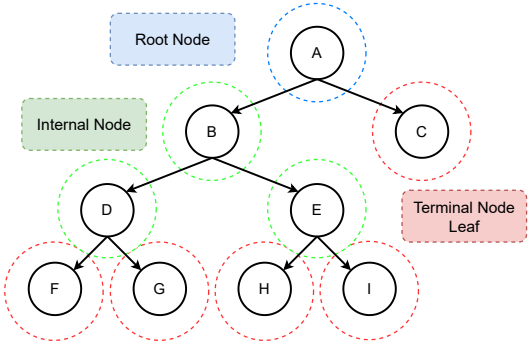
CART
Predictions with CART

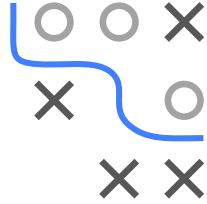


Learning goals

- 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

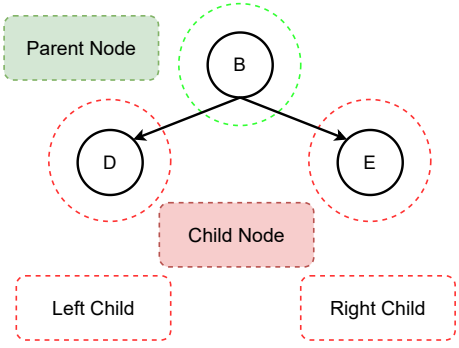
BINARY TREES

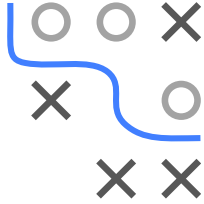




- Binary trees represent a top-down hierarchy with binary splits
- A tree is composed of different node types: a) the root node, b) internal nodes, and c) terminal nodes (also leaves).

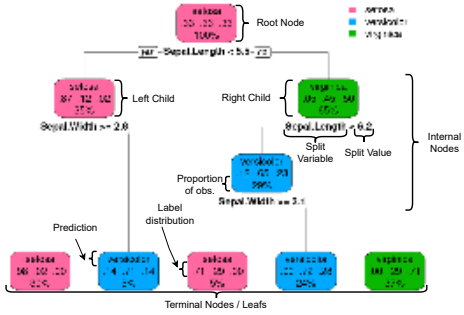
BINARY TREES

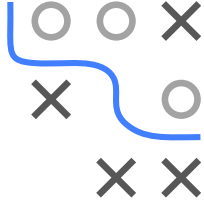




- Nodes have relative relationships, they can be:
  - Parent nodes
  - Child nodes
- Root nodes don't have parents – leaves don't have children

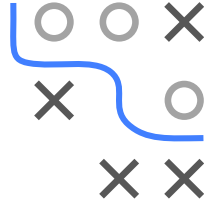
CLASSIFICATION TREES



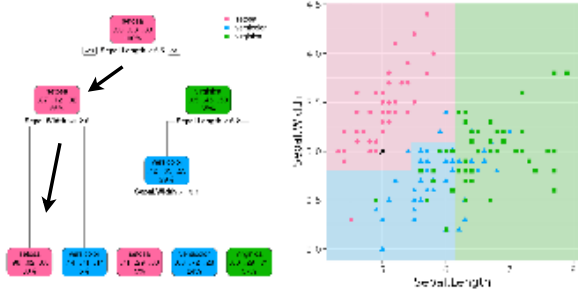


- Classification trees use the structure of a binary tree
- Binary splits are constructed top-down in a *data optimal* way
- Each split is a threshold decision for a single feature
- Each node contains the training points which follow its path
- Each leaf contains a constant prediction

CLASSIFICATION TREE MODEL AND PREDICTION

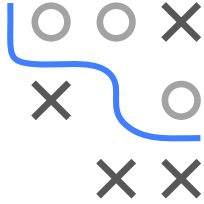


- When predicting new data (here*: `Sepal.Length` = 5, `Sepal.Width` = 3) we use the learned split points and pass an observation through the tree
- Each observation is assigned to exactly one leaf
- Classification trees can make hard-label predictions (here: `setosa`) or predict probabilities (here: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

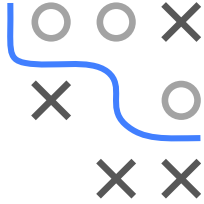


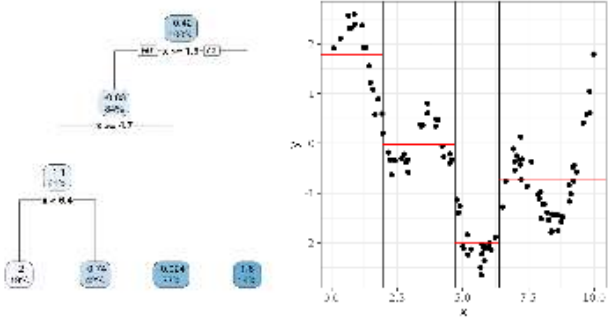
Hard label prediction	Label distribution	`Sepal.Width`	`Sepal.Length`
`setosa`	0.98, 0.02, 0.00	≥ 2.8	< 5.5
`versicolor`	0.14, 0.71, 0.14	< 2.8	< 5.5
`setosa`	0.71, 0.29, 0.00	≥ 3.1	≥ 5.5 & < 6.2
`versicolor`	0.00, 0.72, 0.28	< 3.1	≥ 5.5 & < 6.2
`virginica`	0.00, 0.29, 0.71	–	≥ 6.2



REGRESSION TREE MODEL AND PREDICTION

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

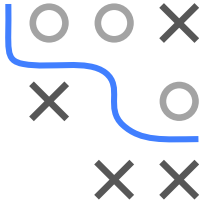




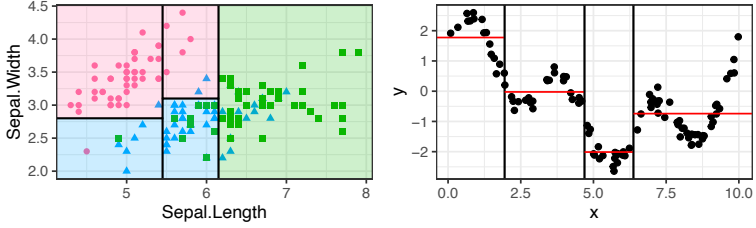
TREE AS AN ADDITIVE MODEL

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

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

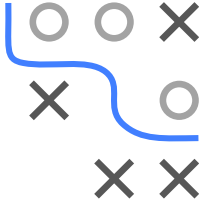


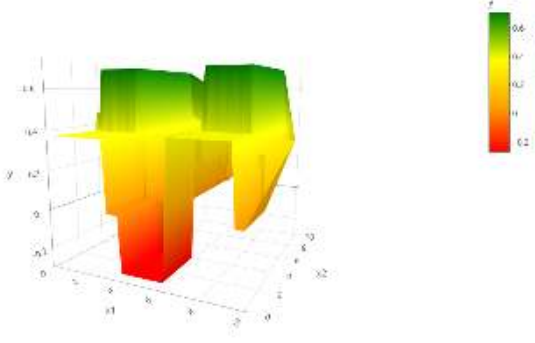
where a tree with M leaf nodes defines M "rectangles" Q_m.
c_m is the predicted numerical response, class label or class distribution in the respective leaf node.



TREE AS AN ADDITIVE MODEL

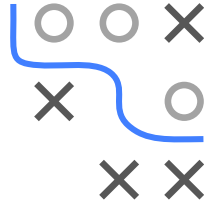
A 2D regression example:



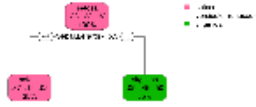


(For binary classification with probabilities, 2D surface looks similar.)

Introduction to Machine Learning



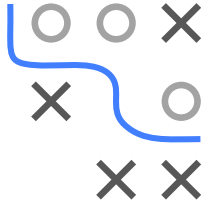
CART
Growing a Tree



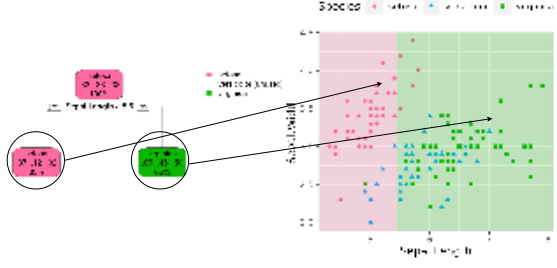
Learning goals

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

TREE GROWING

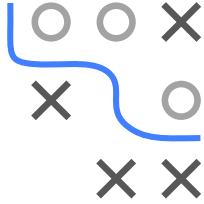


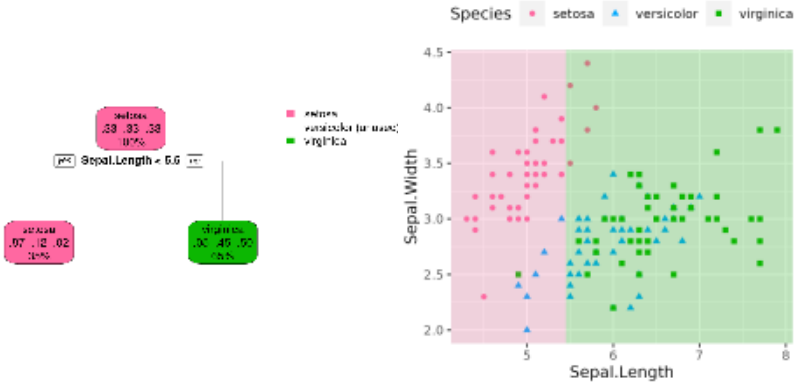
- We start with an empty tree, a root node that contains all the data. Trees are then grown by recursively applying **greedy** optimization to each node $\mathcal{N}$.
- Greedy means we do an **exhaustive search**: Ideally, all possible splits of $\mathcal{N}$ on all possible points t for all features 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

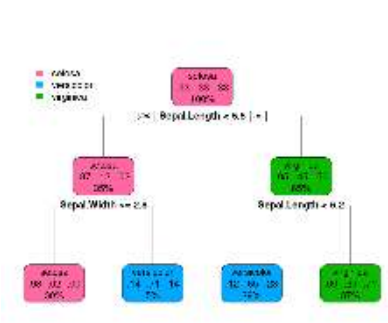
1. Start with a root node of all data.
2. Search for feature and split point that minimizes the empirical risk in child nodes – makes label distribution more homogenous.

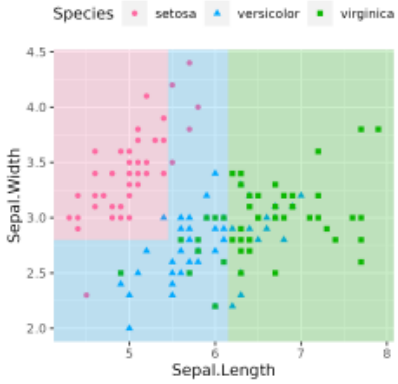


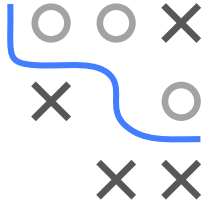


TREE GROWING

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

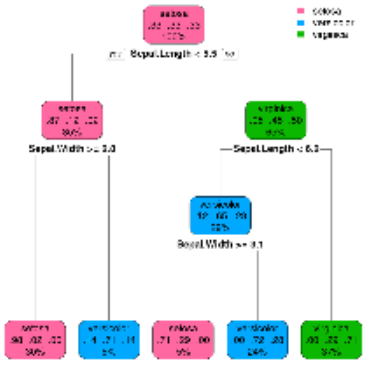


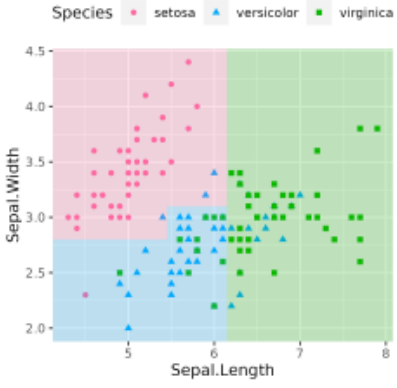


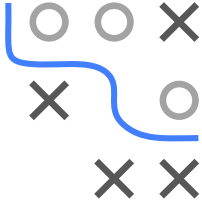


TREE GROWING

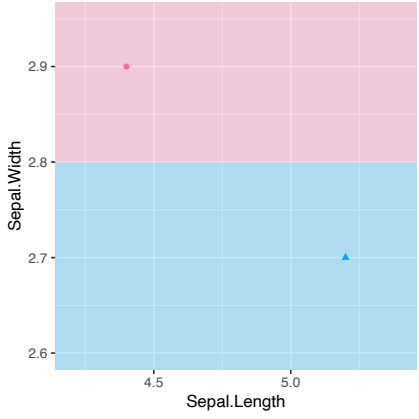
4. Repeat until we reach a stop criterion, e.g., until each leaf cannot be split further.

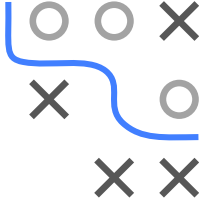






SPLIT PLACEMENT

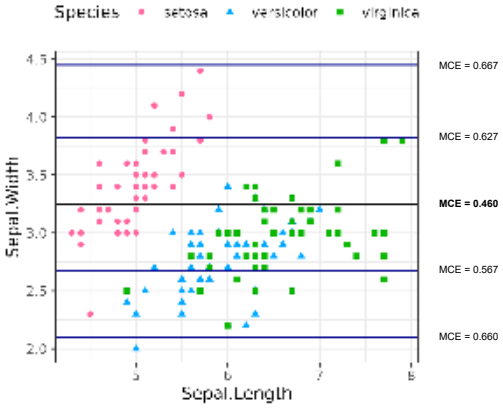


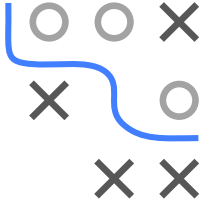


Splits are usually placed at the mid-point of the observations they split: the large margin to the next closest observations makes better generalization on new, unseen data more likely.

FINDING THE SPLIT

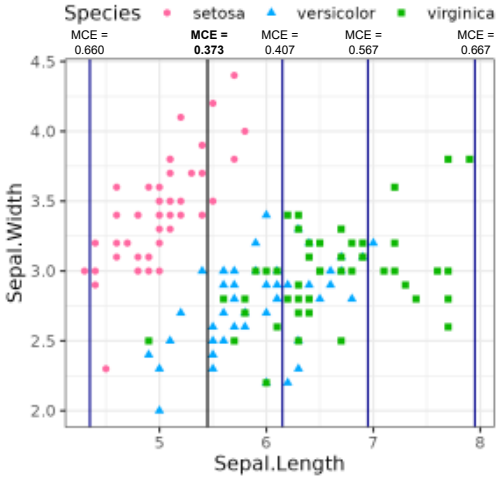
Assume we split the data so that the misclassification error (MCE) is minimal through the splitting.
First, we check a set of potential splits for `Sepal.Width`

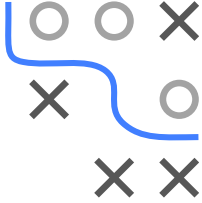




FINDING THE SPLIT

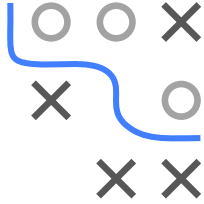
Then we check a set of potential splits for `Sepal.Length`

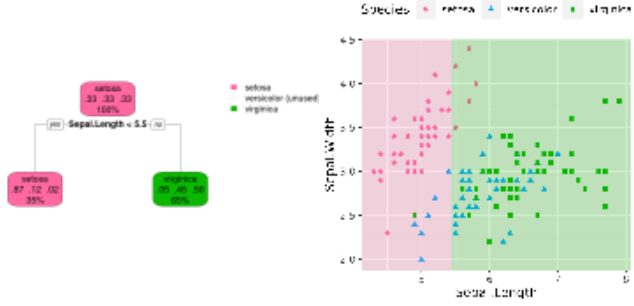




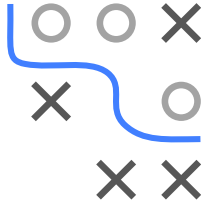
FINDING THE SPLIT

- We take the split with lowest MCE: `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.

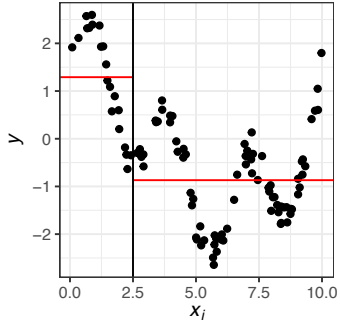




Introduction to Machine Learning



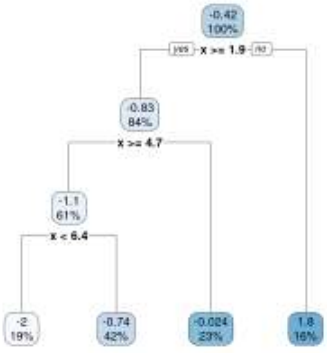
CART
Splitting Criteria for Regression

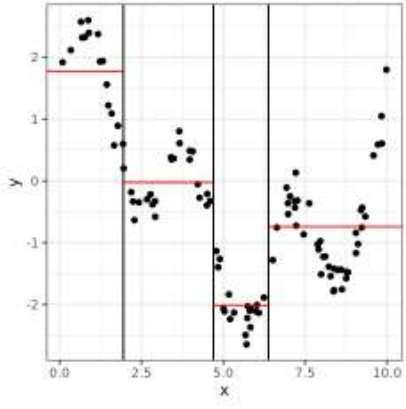


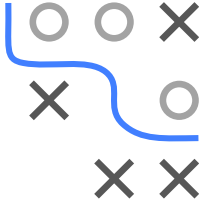
Learning goals

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

SPLITTING CRITERIA



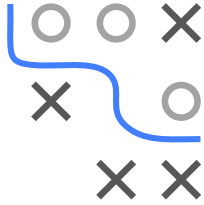




How to find good splitting rules? $\Longrightarrow$ **Empirical Risk Minimization**

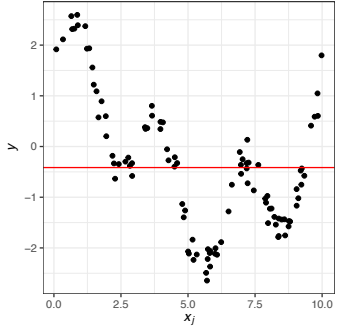
OPTIMAL CONSTANTS IN LEAVES

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



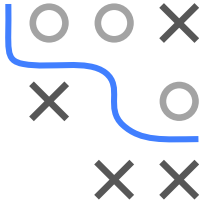
For each child $\mathcal{N}$, predict with optimal constant, e.g., the mean $c_{\mathcal{N}} = \frac{1}{|\mathcal{N}|} \sum_{(\mathbf{x},y) \in \mathcal{N}} y$ for the L_2 loss, i.e., $\mathcal{R}(\mathcal{N}) = \sum_{(\mathbf{x},y) \in \mathcal{N}} (y - c_{\mathcal{N}})^2$.

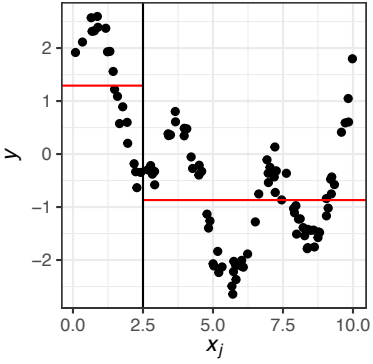
Root node:

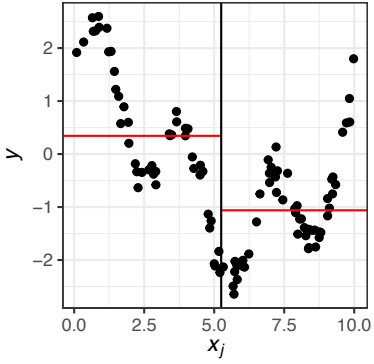


OPTIMAL CONSTANTS IN LEAVES

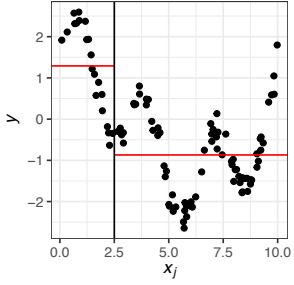
Which of these two splits is better?

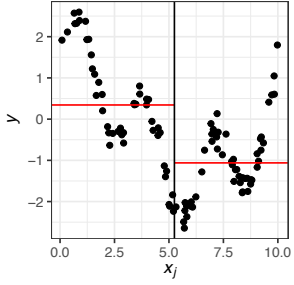


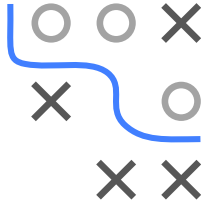




RISK OF A SPLIT







$\mathcal{R}(\mathcal{N}_1) = 23.4, \mathcal{R}(\mathcal{N}_2) = 72.4$ $\mathcal{R}(\mathcal{N}_1) = 78.1, \mathcal{R}(\mathcal{N}_2) = 46.1$

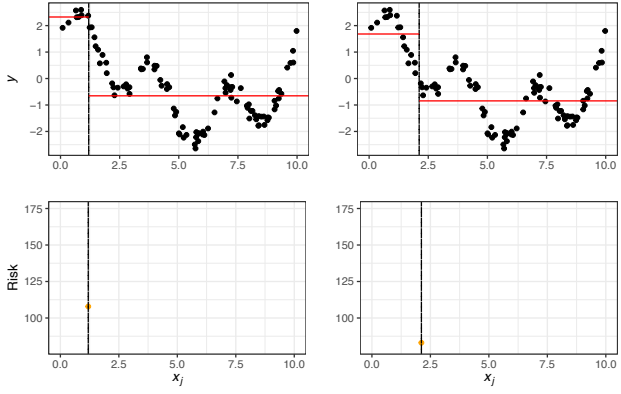
The total risk is the sum of the individual losses:

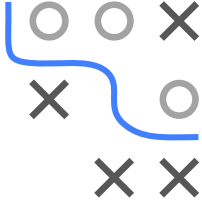
$23.4 + 72.4 = 95.8$ $78.0 + 46.1 = 124.1$

Based on the SSE, we prefer the first split.

SEARCHING THE BEST SPLIT

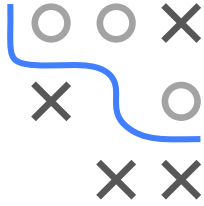
Let's find the best split for this feature by tabulating results.

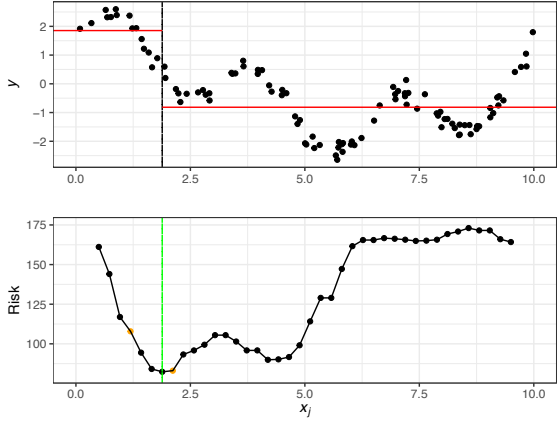




SEARCHING THE BEST SPLIT

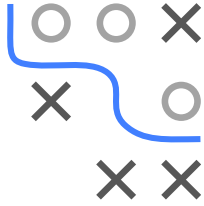
Let's iterate – quantile-wise or over all points.





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:

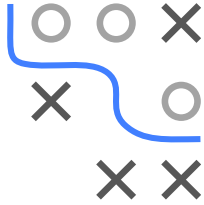
$$\mathcal{R}(\mathcal{N}) = \sum_{(\mathbf{x},y)\in\mathcal{N}} L(y, c_{\mathcal{N}})$$

- 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

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

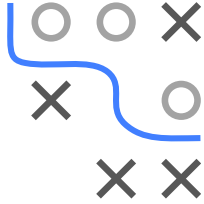
- Finding the best way to split $\mathcal{N}$ into $\mathcal{N}_1, \mathcal{N}_2$ means solving

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

FORMALIZATION

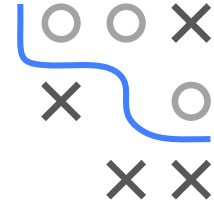
- $\mathcal{R}(\mathcal{N}, j, t) = \mathcal{R}(\mathcal{N}_1) + \mathcal{R}(\mathcal{N}_2)$, makes sense if $\mathcal{R}$ is a simple sum
- If we use averages, we have to reweight the terms to obtain a global average w.r.t. $\mathcal{N}$ as the children have different sizes

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

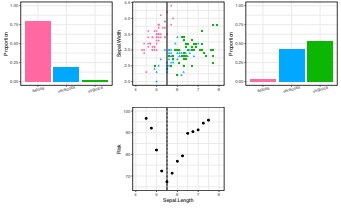


- We mention this for clarity, as quite a few texts contain only the (more complicated) weighted formula without clear explanation

Introduction to Machine Learning



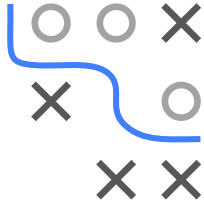
CART
Splitting Criteria for Classification



Learning goals

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

OPTIMAL CONSTANT MODELS



As losses in classification, we typically use:

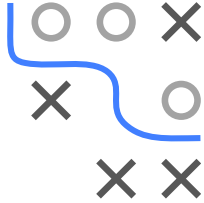
- (Multi-class) Brier score $L(y, \pi) = \sum_{k=1}^{g} (\pi_k - o_k(y))^2$,
  a.k.a. L_2 loss on probabilities
- (Multi-class) Log loss $L(y, \pi) = -\sum_{k=1}^{g} o_k(y) \log(\pi_k)$,
  as in logistic regression

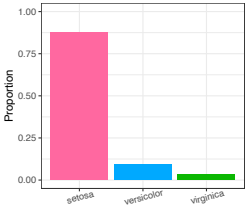
Optimal constant predictions (in a node) for both losses are simply the proportions of the contained classes:

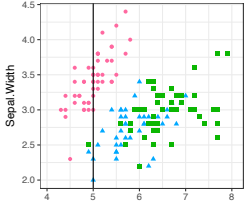
$$c_{\mathcal{N}} = (\hat{\pi}_1^{(\mathcal{N})}, \ldots, \hat{\pi}_g^{(\mathcal{N})}) \quad \text{with}$$
$$\hat{\pi}_k^{(\mathcal{N})} = \frac{1}{|\mathcal{N}|} \sum_{(\mathbf{x},y) \in \mathcal{N}} \mathbb{I}(y = k) \quad \forall k \in \{1, \ldots, g\}$$

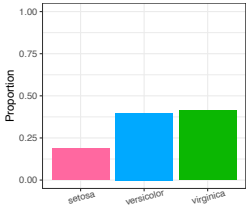
FINDING THE BEST SPLIT

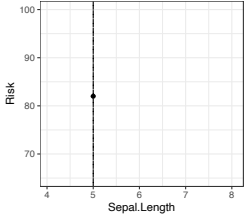
Let's compute the Brier score for all splits, with optimal constant probability vectors in both children





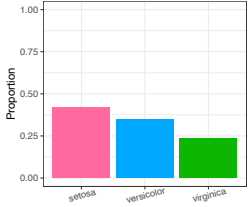


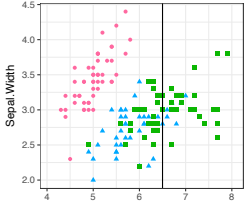


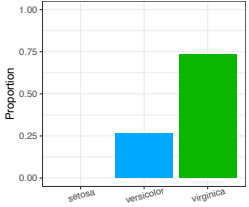


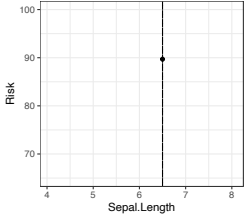
FINDING THE BEST SPLIT

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



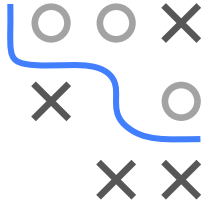


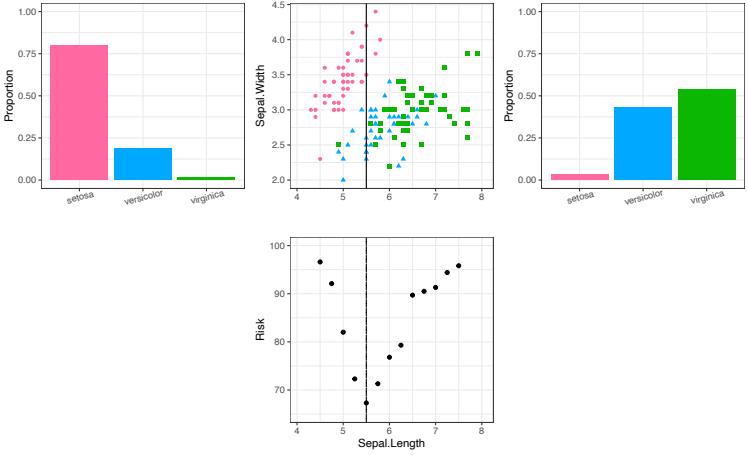




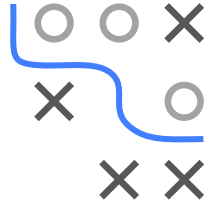
FINDING THE BEST SPLIT

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

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

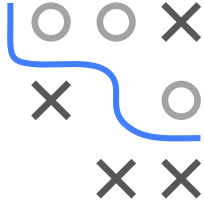
- Log loss is equivalent to entropy

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

- 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:

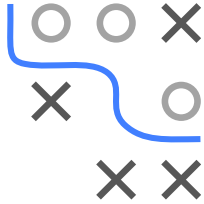
	class 0	class 1
$\mathcal{N}_1$	300	100
$\mathcal{N}_2$	100	300

Split 2:

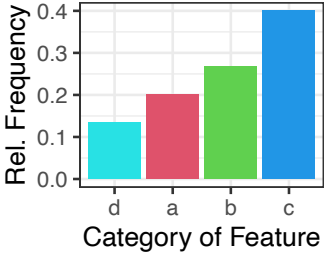
	class 0	class 1
$\mathcal{N}_1$	400	200
$\mathcal{N}_2$	0	200

- 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

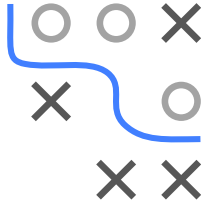


Learning goals

- 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.

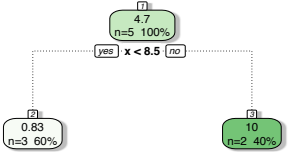


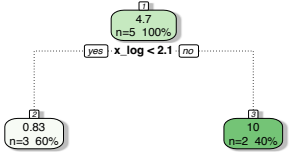
Original data

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

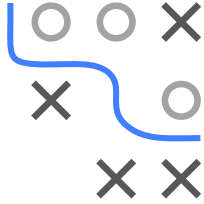


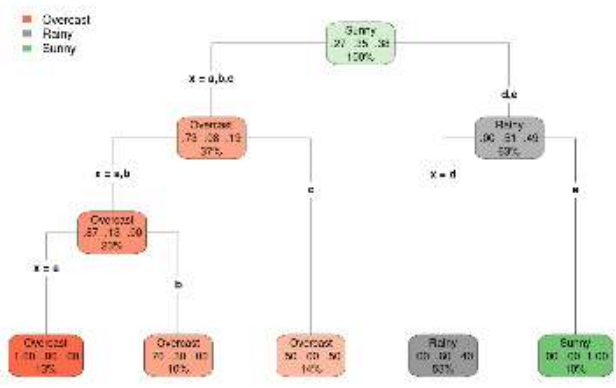


CATEGORICAL FEATURES

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

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

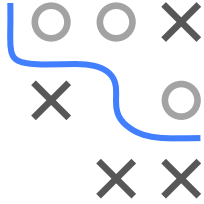




CATEGORICAL FEATURES

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

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

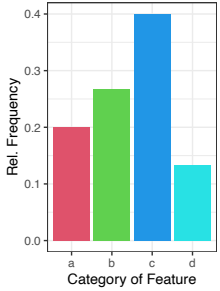


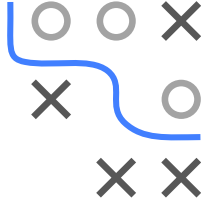
- For a feature with m levels, there are about 2^m different possible partitions of the m values into two groups ($2^{m-1} - 1$ because of symmetry and empty groups).
- Searching over all these becomes prohibitive for large values of m.
- For regression with L2 loss and for binary classification, we 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 $\mathcal{N}$.

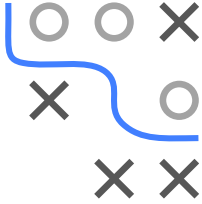


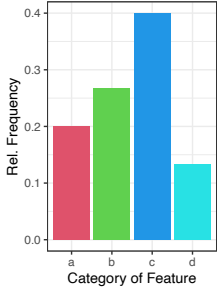


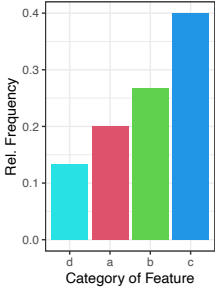
CATEGORICAL FEATURES

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.



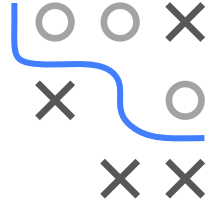


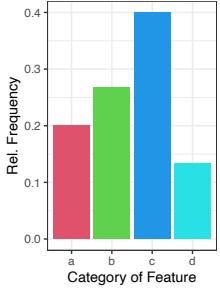


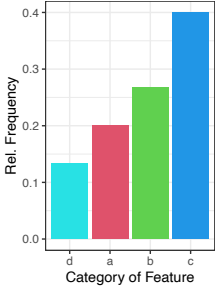
CATEGORICAL FEATURES

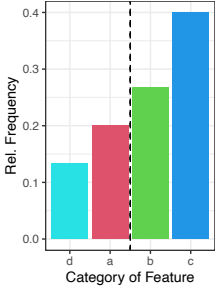
For $0 - 1$ responses, in each node:

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.

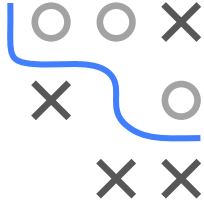








CATEGORICAL FEATURES

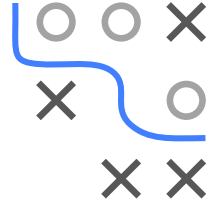


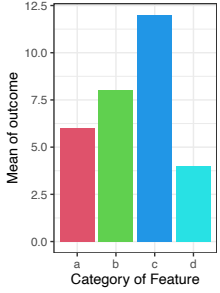
- This procedure finds the optimal split.
- This result also holds for regression trees (with L2 loss) if the levels of the feature are ordered by increasing mean of the target
- The proofs are not trivial and can be found here:
  - for 0-1 responses:
    - ▸ Breiman, 1984, Chapter 4
    - ▸ Ripley, 1996, pp. 213 et seqq.
  - for continuous responses:
    - ▸ Fisher, 1958
- There are only heuristics for the multiclass case ▸ Wright and König, 2019

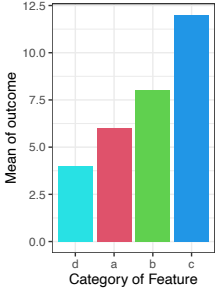
CATEGORICAL FEATURES

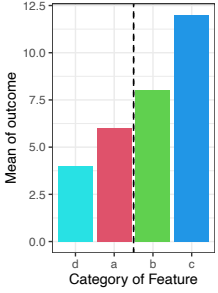
For continuous responses, in each node:

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



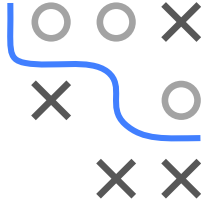




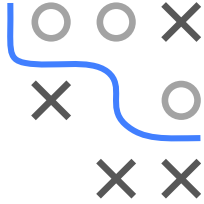


MISSING FEATURE VALUES

- 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

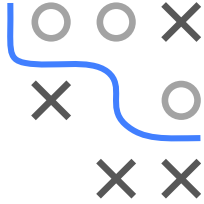


- Each surrogate split is a decision stump that tries to learn the actual splitting rule
- Consider this tree with the primary split w.r.t. `Sepal.Length` where we perform binary classification (`setosa` vs. `virginica`):



- Our surrogate split should optimize a splitting criterion w.r.t. `Sepal.Length < 5.8`

SURROGATE SPLITS

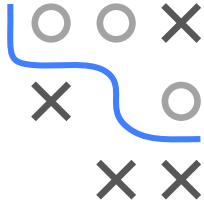


- Consider this subsample of the data used to fit the tree:

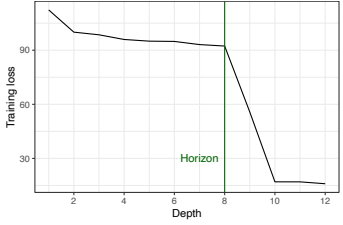
	Sepal.Length	...	Petal.Width	Species	Sepal.Length $<$ 5.8
1	5.10	...	0.20	setosa	TRUE
4	4.60	...	0.20	setosa	TRUE
9	4.40	...	0.20	setosa	TRUE
15	5.80	...	0.20	setosa	FALSE
18	5.10	...	0.30	setosa	TRUE
52	5.80	...	1.90	virginica	FALSE
57	4.90	...	1.70	virginica	TRUE
62	6.40	...	1.90	virginica	FALSE
77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

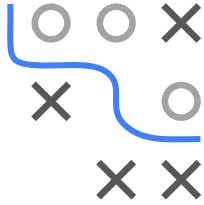


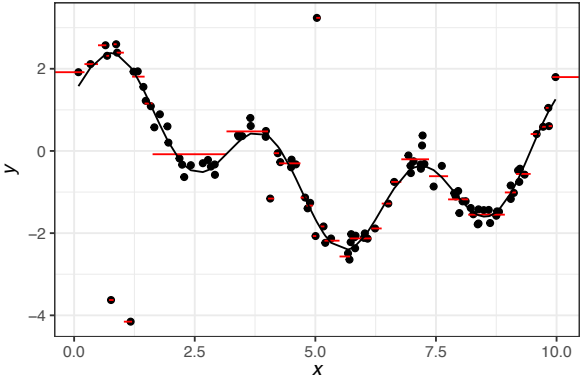
Learning goals

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

OVERFITTING TREES

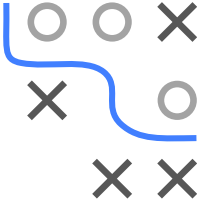
The recursive partitioning procedure used to grow a CART could run until every leaf only contains a single observation. Problem: Very complex trees will *overfit the training data.* At some point we should stop splitting nodes into ever smaller child nodes:

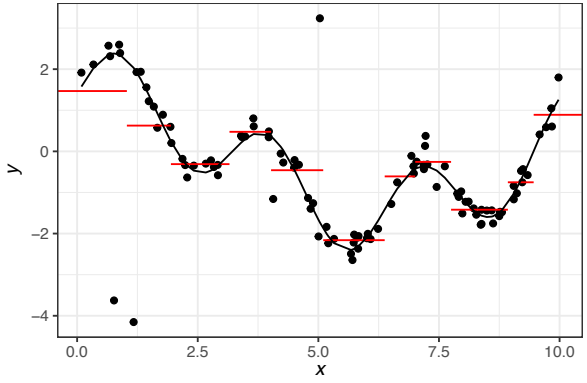




OVERFITTING TREES

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



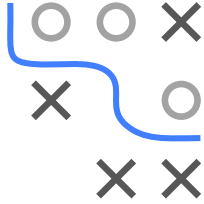


STOPPING CRITERIA

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

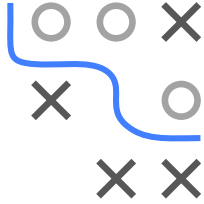
- a certain number of leaves if reached,
- it contains too few observations,
- splitting results in children with too few observations,
- splitting does not achieve a certain minimal improvement of the risk in the children, compared to the risk in the parent node,
- it already has the same target value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

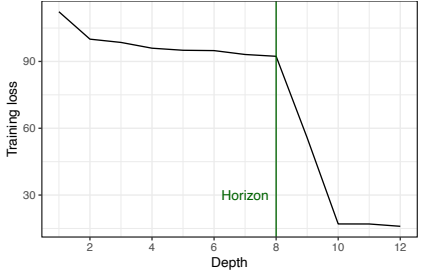
Selection of a stopping criterion and its concrete values are hyperparameters of CART.



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*).

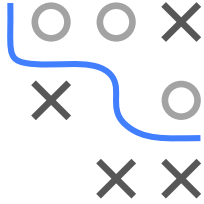




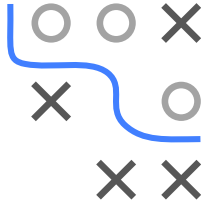
PRUNING

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



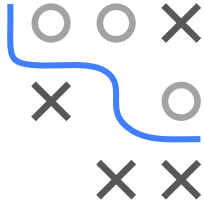
- Prominent pruning method: Cost-complexity pruning (CCP)
- Idea: Grow a large tree and remove the least informative leaves
- CCP is steered with a regularization parameter α that penalizes the number of leaves in a sub tree

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

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

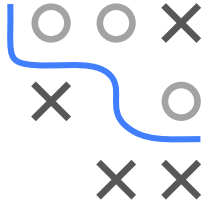
CCP performs a greedy backward search:

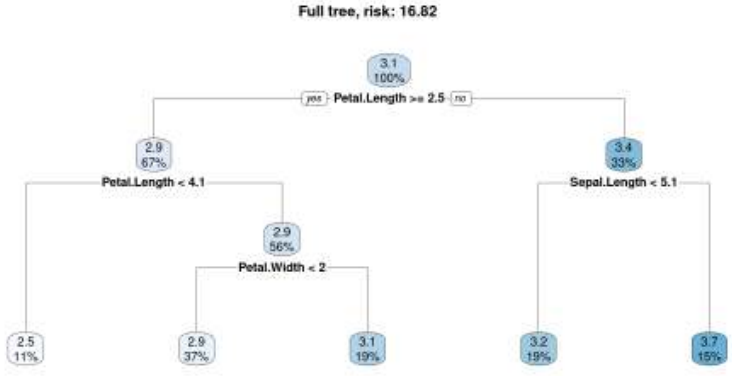


- Computes $\mathcal{R}_{\text{reg}}(T)$ with a fixed α for all possible sub trees that can be created by replacing one internal node with a leaf.
- By replacing a node we also eliminate all subsequent nodes.
- We select the sub tree with lowest risk and repeat the procedure.
- We stop if pruning does 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 α is typically selected via cross-validation.
- Other prominent post-pruning methods include, e.g., reduced error pruning (REP) or pessimistic error pruning (PEP).

CCP

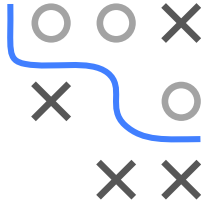
We run the CCP algorithm step-by-step with $\alpha = 1.2$:

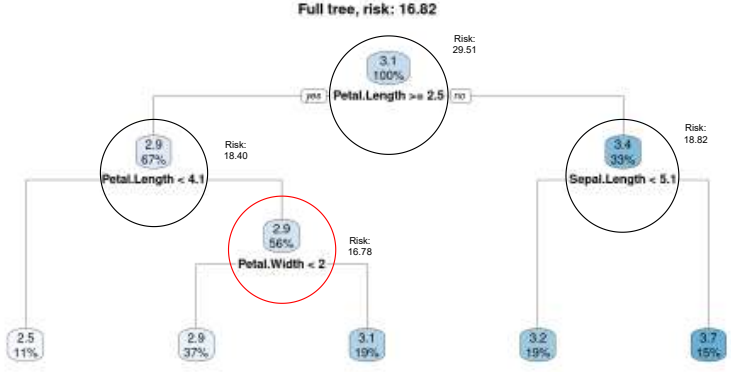




CCP

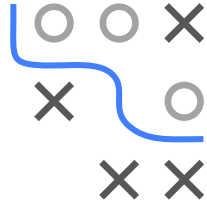
There are four possible nodes that we can eliminate to prune the tree.
We take the one replacement that results in the lowest risk (red).

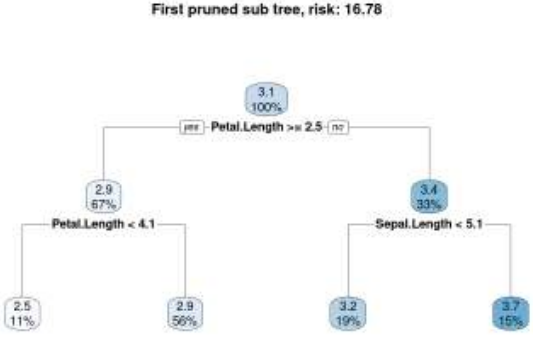




CCP

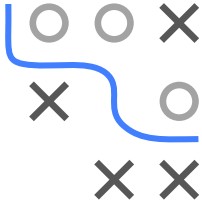
The first pruned sub tree has a lower risk than the full tree. Thus, we prefer it over the full tree.

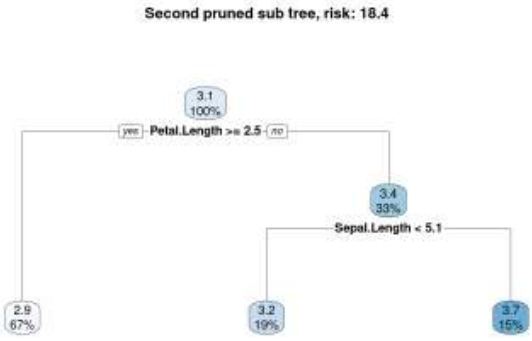




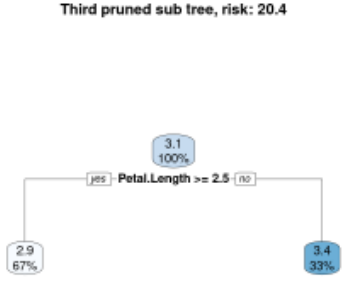
CCP

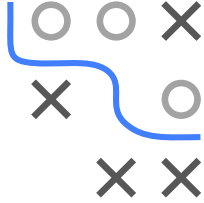
From here on, the risk increases.





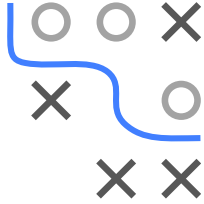
CCP


Third pruned sub tree, risk: 20.4
3.1
100%
yes
Petal.Length >= 2.5
no
2.9
67%
3.4
33%



CCP

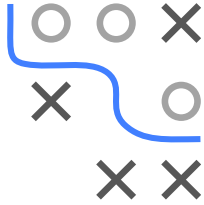
We select the first sub tree as it results in the lowest risk in the complete sequence of sub trees.



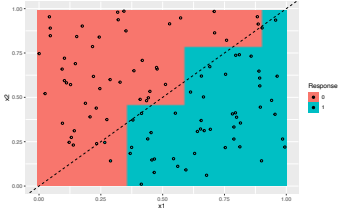
Fully pruned sub tree, risk: 29.51



Introduction to Machine Learning



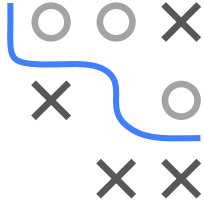
CART
Advantages & Disadvantages



Learning goals

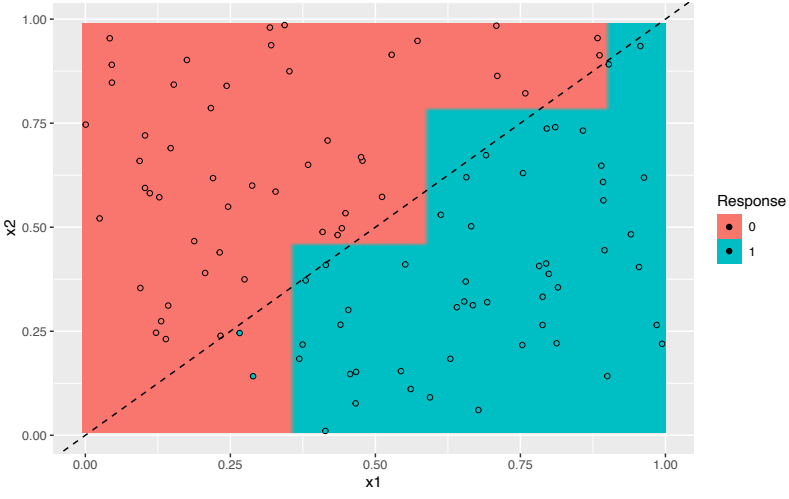
- Understand the advantages and disadvantages of CART
- Know when and where CART are applied

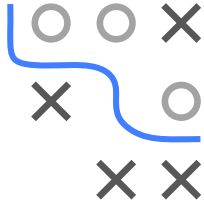
ADVANTAGES



- 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.

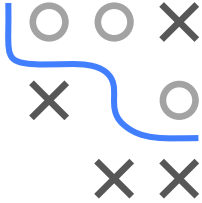
DISADVANTAGE: LINEAR DEPENDENCIES

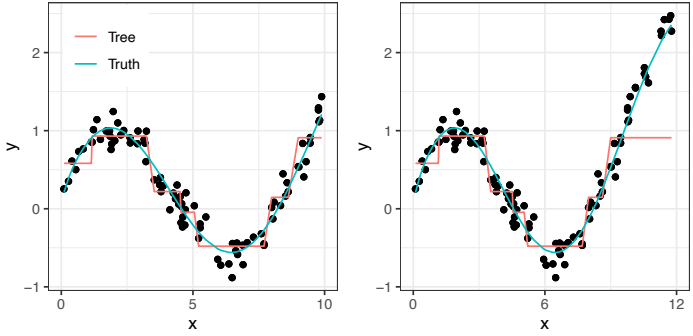




Linear dependencies must be modeled over several splits. Logistic regression would model this easily with fewer parameters.

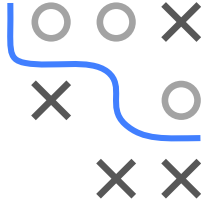
DISADVANTAGES: SMOOTH FUNCTIONS AND EXTRAPOLATION





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

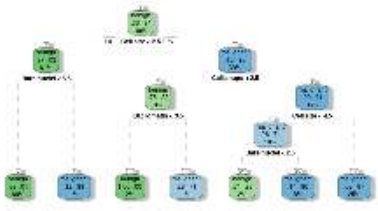
Consider the Wisconsin Breast Cancer data set with 699 observations on 9 features and binary target (“benign” vs. “malignant”). We fit two trees: (A) with the full data set and (B) where we eliminated a single observation. Results in label flip for 17 observations of the training data:

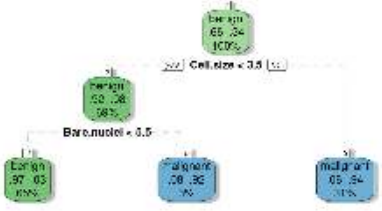
	benign	malignant
benign	445	6
malignant	11	236

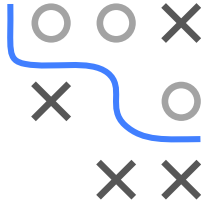
Rows: Predictions of (A), columns: Predictions of (B)

DISADVANTAGE: INSTABILITY

The resulting decision trees look very different:

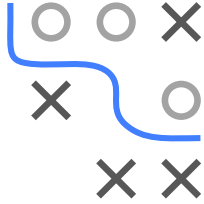




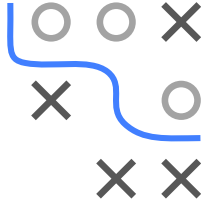


CART IN PRACTICE

- 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

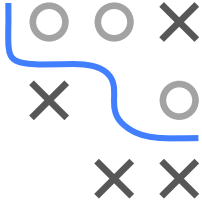
Classification and Regression Trees (CART)

Random Forests

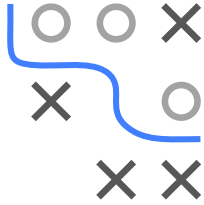
Neural Networks

Tuning

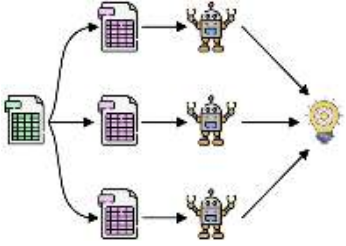
Nested Resampling



Introduction to Machine Learning



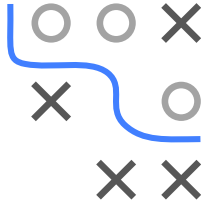
Random Forest
Bagging Ensembles



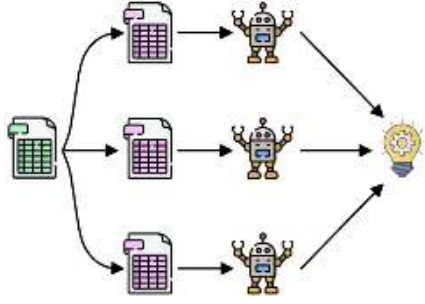
Learning goals

- Understand idea of bagging
- Be able to explain the connection between bagging and bootstrap
- Understand why bagging improves predictive performance

BAGGING



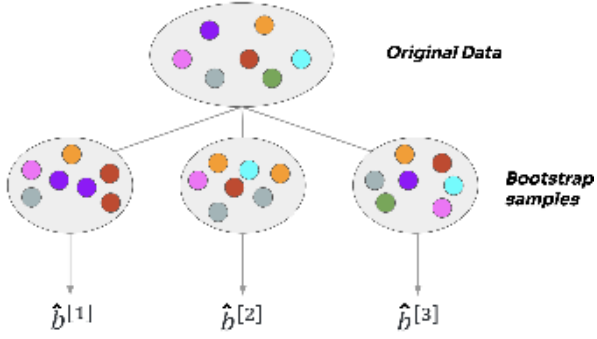
- Bagging is short for **B**ootstrap **Agg**regation
- **Ensemble method**, combines models into large “meta-model”; ensembles usually better than single **base learner**
- Homogeneous ensembles always use same BL class (e.g. CART), heterogeneous ensembles can use different classes
- Bagging is homogeneous



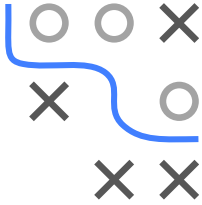
TRAINING BAGGED ENSEMBLES

Train BL on M **bootstrap** samples of training data $\mathcal{D}$:

- Draw n observations from $\mathcal{D}$ with replacement
- Fit BL on each bootstrapped data $\mathcal{D}^{[m]}$ to obtain $\hat{b}^{[m]}$

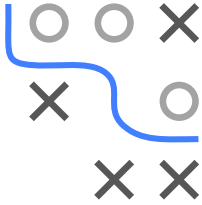


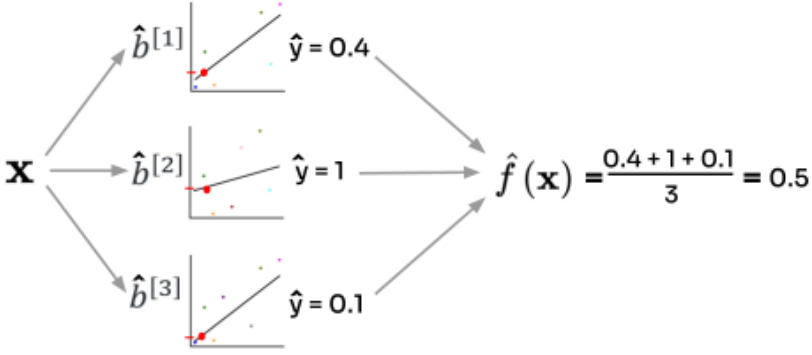
- Data sampled in one iter called “in-bag” (IB)
- Data not sampled called “out-of-bag” (OOB)



PREDICTING WITH A BAGGED ENSEMBLE

Average predictions of *M* fitted models for ensemble:
(here: regression)





BAGGING PSEUDO CODE

Bagging algorithm: Training

1: **Input:** Dataset $\mathcal{D}$, type of BLs, number of bootstraps M
2: **for** $m = 1 \to M$ **do**
3: Draw a bootstrap sample $\mathcal{D}^{[m]}$ from $\mathcal{D}$
4: Train BL on $\mathcal{D}^{[m]}$ to obtain model $\hat{b}^{[m]}$
5: **end for**

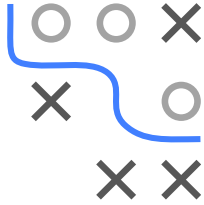
Bagging algorithm: Prediction

1: **Input:** Obs. $\mathbf{x}$, trained BLs $\hat{b}^{[m]}$ (as scores $\hat{f}^{[m]}$, hard labels $\hat{h}^{[m]}$ or probs $\hat{\pi}^{[m]}$)
2: Aggregate/Average predictions

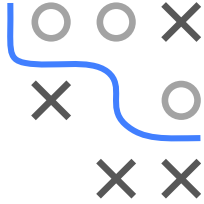
$$\hat{f}(\mathbf{x}) = \frac{1}{M} \sum_{m=1}^{M} \left(\hat{f}^{[m]}(\mathbf{x}) \right) \quad \text{(regression / decision score, use } \hat{f}_k \text{ in multi-class)}$$

$$\hat{h}(\mathbf{x}) = \underset{k \in \mathcal{Y}}{\arg\max} \sum_{m=1}^{M} \mathbb{I}\left(\hat{h}^{[m]}(\mathbf{x}) = k \right) \quad \text{(majority voting)}$$

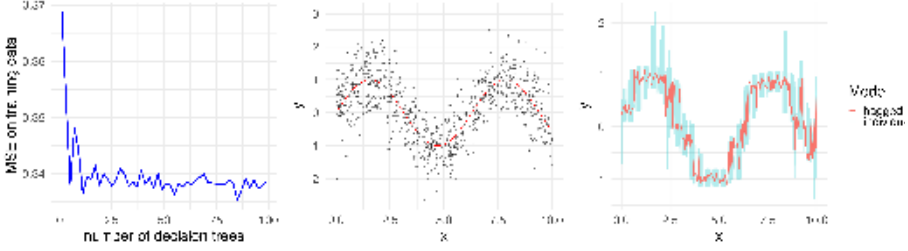
$$\hat{\pi}_k(\mathbf{x}) = \begin{cases} \frac{1}{M} \sum_{m=1}^{M} \hat{\pi}_k^{[m]}(\mathbf{x}) & \text{(probabilities through averaging)} \\ \frac{1}{M} \sum_{m=1}^{M} \mathbb{I}\left(\hat{h}^{[m]}(\mathbf{x}) = k \right) & \text{(probabilities through class frequencies)} \end{cases}$$



WHY/WHEN DOES BAGGING HELP?



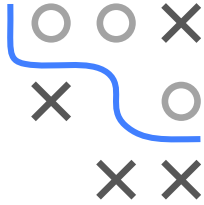
- Bagging reduces the variability of predictions by averaging the outcomes from multiple BL models
- It is particularly effective when the errors of a BL are mainly due to (random) variability rather than systematic issues

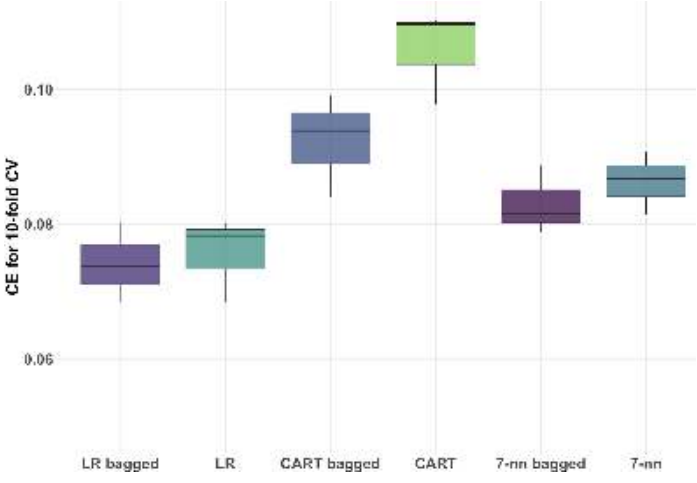


- Increasing **nr. of BLs** improves performance, up to a point, optimal ensemble size depends on inducer and data distribution

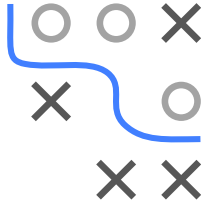
MINI BENCHMARK

Bagged ensembles with 100 BLs each on `spam`:
Bagging seems especially helpful for less stable learners like CART





Introduction to Machine Learning



Random Forest Basics



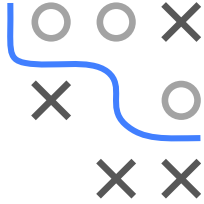
Learning goals

- Know how random forests are defined by extending the idea of bagging
- Understand general idea to decorrelate trees
- Understand effects of hyperparameters
- RFs and overfitting

MOTIVATION

CARTs offer several appealing features:

- **Interpretability**: Easy to understand and explain
- **Invariance to rank-preserving transformations**: E.g., unaffected by scaling or shifting of features
- **Versatility**: Work on categorical and numerical data
- **Robustness to missing values**: Can work with missings



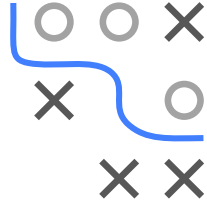
Despite these benefits, CARTs are not without drawbacks:

▸ Hastie, Tibshirani, and Friedman 2009

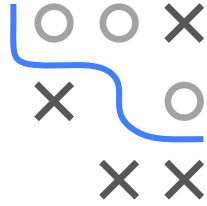
> *"Trees have one aspect that prevents them from being the ideal tool for predictive learning, namely inaccuracy."*

RANDOM FORESTS ▸ Breiman 2001

- RFs use **bagging with CARTs as BLs**
- **Random feature sampling** decorrelates the base learners
- **Fully expanded trees** further increase variability of trees



INTUITION BEHIND DECORRELATION



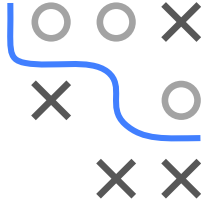
- Since bootstrap samples are similar, models $\hat{b}^{[m]}$ are correlated, affecting the variance of an ensemble $\hat{f}$
- We would like variance to go down linearly with ensemble size, but because of correlation we cannot really expect that
- Assuming $\text{Var}(\hat{b}^{[m]}) = \sigma^2$, $\text{Corr}(\hat{b}^{[m]}, \hat{b}^{[j]}) = \rho$, semi-formal analysis, without proper analysis of prediction error:

$$\text{Var}\left(\hat{f}\right) = \text{Var}\left(\frac{1}{M}\sum_{m=1}^{M}\hat{b}^{[m]}\right) = \frac{1}{M^2}\left(\sum_{m=1}^{M}\text{Var}(\hat{b}^{[m]}) + 2\sum_{m<j}\text{Cov}(\hat{b}^{[m]}, \hat{b}^{[j]}\right)$$

$$= \frac{1}{M^2}\left(M\sigma^2 + 2\frac{M(M-1)}{2}\rho\sigma^2\right) = (1-\rho)\frac{\sigma^2}{M} + \rho\sigma^2$$

- Ensemble variance is “convex-combo of linear-reduction and no-reduction, controlled by ρ”
- Maybe we can decorrelate trees, to reduce ensemble variance? And get less prediction error?

RANDOM FEATURE SAMPLING

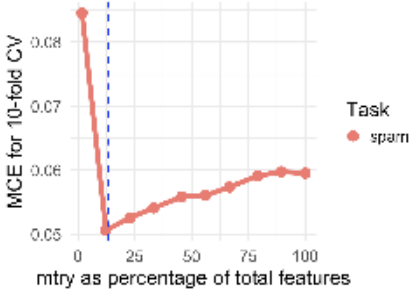


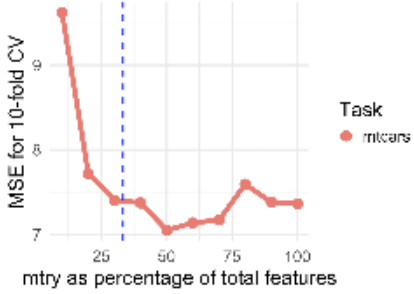
RFs decorrelate trees with a simple randomization:

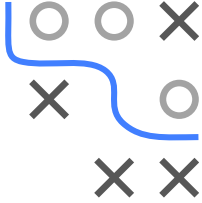
- For each node of tree, randomly draw $\texttt{mtry} \leq p$ features ($\texttt{mtry}$ = name in some implementations)
- Only consider these features for finding the best split
- Careful: Our previous analysis was simplified! The more we decorrelate by this, the more random the trees become! This also has negative effects!



EFFECT OF FEATURE SAMPLING





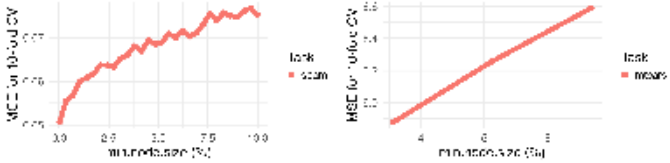


- Optimal `mtry` typically larger for regression than for classification
- Good defaults exist, but still most relevant tuning param
- Rule of thumb:
  - Classification: $\texttt{mtry} = \lfloor \sqrt{p} \rfloor$
  - Regression: $\texttt{mtry} = \lfloor p/3 \rfloor$

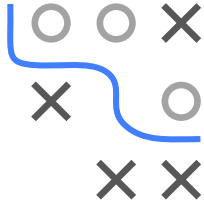
TREE SIZE

In addition to `mtry`, RFs have two other important HPs:

- Min. nr. of obs. in each decision tree node
  Default (ranger): `min.node.size = 5` ▸ Breiman 2001



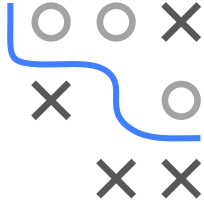
- Depth of each tree
  Default (ranger): `maxDepth` $= \infty$
- There are more alternative HPs to control depth of tree:
  minimal risk reduction, size of terminal nodes, etc.

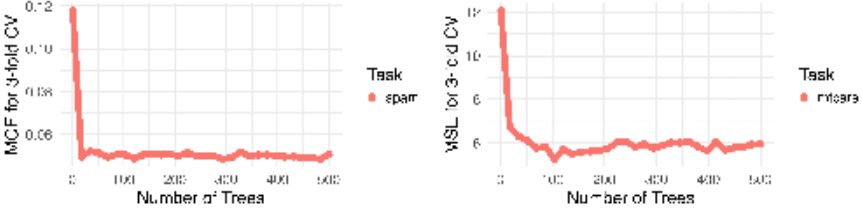


RF usually use fully expanded trees, without aggressive early stopping or pruning, to further **increase variability of each tree**. ▸ Louppe 2015

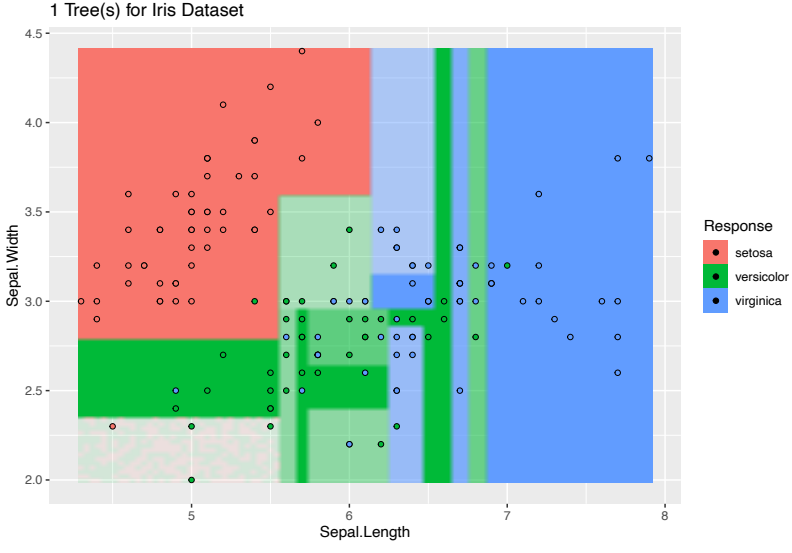
ENSEMBLE SIZE

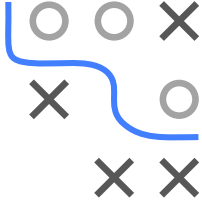
- RFs usually better if ensemble is large ▸ Breiman 2001
- But: Increases computational costs, and diminishing returns
- 100 or 500 is a sensible default
- Can also inspect the OOB error (see later)



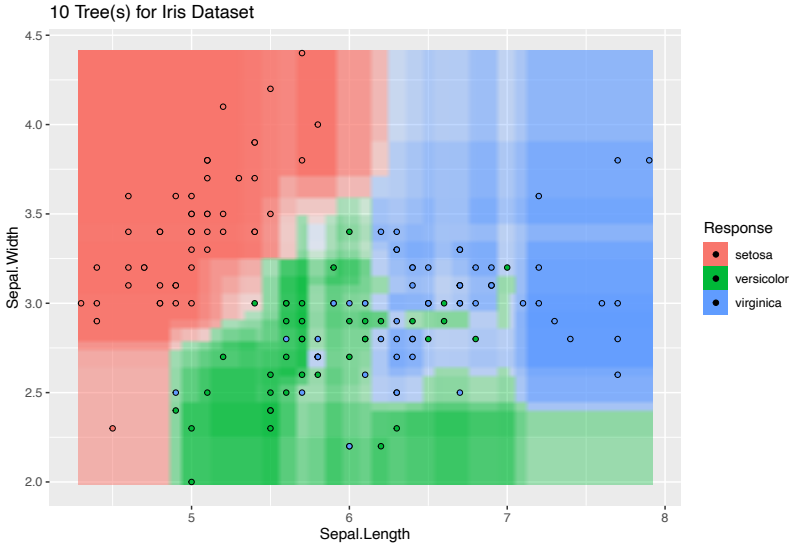


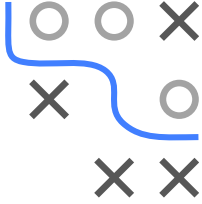
EFFECT OF ENSEMBLE SIZE



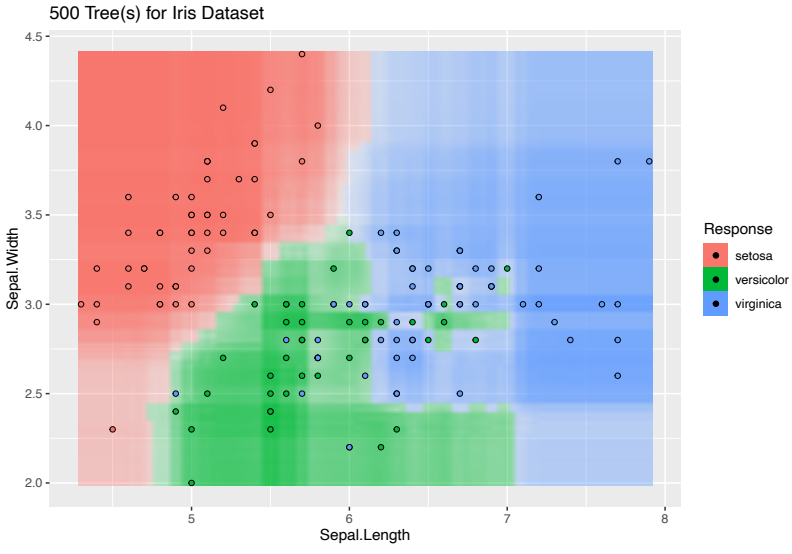


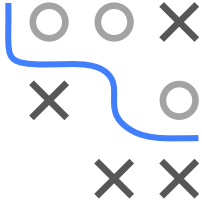
EFFECT OF ENSEMBLE SIZE





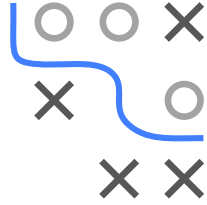
EFFECT OF ENSEMBLE SIZE

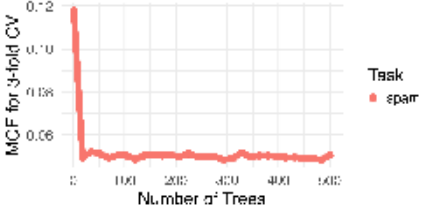


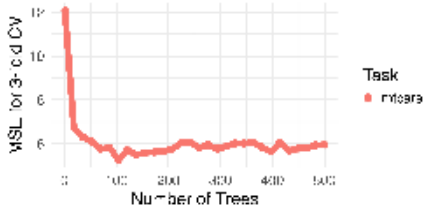


CAN RF OVERFIT? ▸ Probst and Boulesteix 2018

- Just like any other learner, RFs **can** overfit!
- However, RFs generally **less** prone to overfitting than individual CARTs.
- Overly complex trees can *still* lead to overfitting!
  If most trees capture noise, so does the RF.
- But randomization and averaging helps.



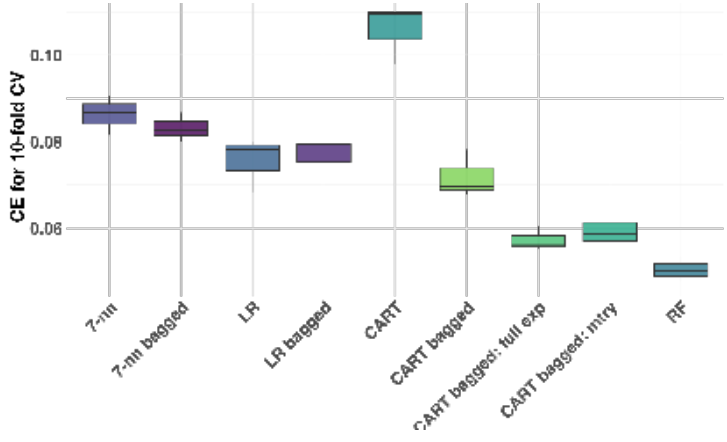


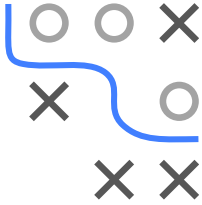


Since each tree is trained *individually and without knowledge of previously trained trees*, increasing `ntrees` generally reduces variance **without increasing the chance of overfitting!**

RF IN PRACTICE

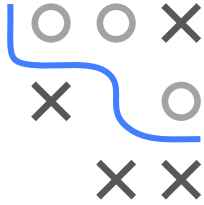
Benchmarking bagged ensembles with 100 BLs each on `spam` versus RF (`ntrees` $= 100$, `mtry` $= \sqrt{p}$, `minnode` $= 1$), we see how well RF performs!





$\Rightarrow$ RFs combine the benefits of random feature selection and fully expanded trees.

DISCUSSION



Advantages:

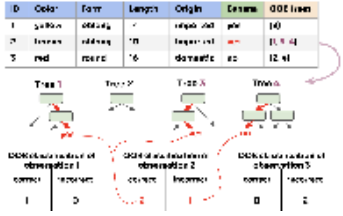
- Most advantages of trees also apply to RF: not much preprocessing required, missing value handling, etc.
- Easy to parallelize
- Often work well (enough)
- Works well on high-dimensional data
- Works well on data with irrelevant “noise” variables

Disadvantages:

- 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 can be computationally demanding for large ensembles

Introduction to Machine Learning

Random Forest
Out-of-Bag Error Estimate



Learning goals

- Understand the concept of out-of-bag and in-bag observations
- Learn how out-of-bag error provides an estimate of the generalization error during training

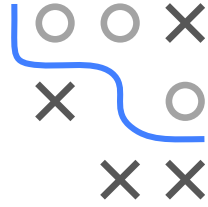
OUT-OF-BAG VS IN-BAG OBSERVATIONS

ID	Color	Form	Length	Origin	Banana
1	yellow	oblong	14	imported	yes
2	brown	oblong	10	imported	yes
3	red	round	16	domestic	no

Bootstrapping to train tree 1

ID	Color	Form	Length	Origin	Banana
1	yellow	oblong	14	imported	yes
3	red	round	16	domestic	no
3	red	round	16	domestic	no

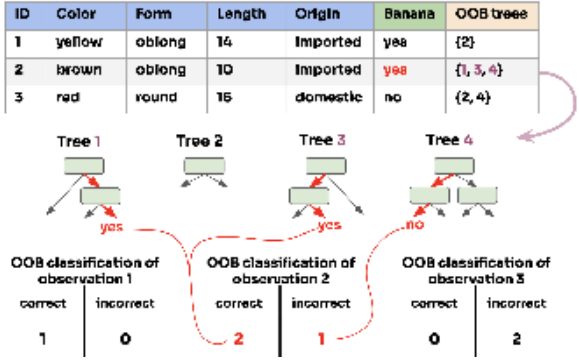




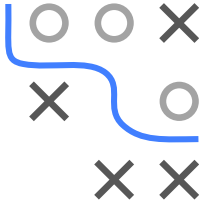
- IB observations for *m*-th bootstrap:
  $\text{IB}^{[m]} = \{i \in \{1, \ldots, n\} | \left(\mathbf{x}^{(i)}, y^{(i)}\right) \in \mathcal{D}^{[m]}\}$
- OOB observations for *m*-th bootstrap:
  $\text{OOB}^{[m]} = \{i \in \{1, \ldots, n\} | \left(\mathbf{x}^{(i)}, y^{(i)}\right) \notin \mathcal{D}^{[m]}\}$
- Nr. of trees where *i*-th observation is OOB:
  $s_{\text{OOB}}^{(i)} = \sum_{m=1}^{M} \mathbb{I}(i \in \text{OOB}^{[m]})$.

OUT-OF-BAG ERROR ESTIMATE

Predict i-th observation with all trees $\hat{b}^{[m]}$ for which it is OOB:

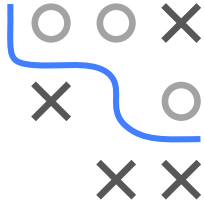


ID	Color	Form	Length	Origin	Banana	OOB trees
1	yellow	oblong	14	imported	yes	{2}
2	brown	oblong	10	imported	yes	{1, 3, 4}
3	red	round	16	domestic	no	{2, 4}



OOB prediction $\hat{\pi}^{(2)}_{\text{OOB}} = 2/3$. Evaluating all OOB predictions with some loss function L or set-based metric ρ estimates the GE.
As we do not violate the **untouched test set principle**, $\widehat{\text{GE}}$ is not *optimistically* biased.

OUT-OF-BAG ERROR PSEUDO CODE



Out-Of-Bag error estimation

1: **Input:** $\text{OOB}^{[m]}, \hat{b}^{[m]} \forall m \in \{1, \ldots, M\}$
2: **for** $i = 1 \rightarrow n$ **do**
3: Compute the ensemble OOB prediction for observation i, e.g., for regression:

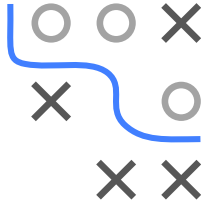
$$\hat{f}^{(i)}_{\text{OOB}} = \frac{1}{S_{\text{OOB}^{(i)}}} \sum_{m=1}^{M} \mathbb{I}(i \in \text{OOB}^{[m]}) \cdot \hat{f}^{[m]}(\mathbf{x}^{(i)})$$

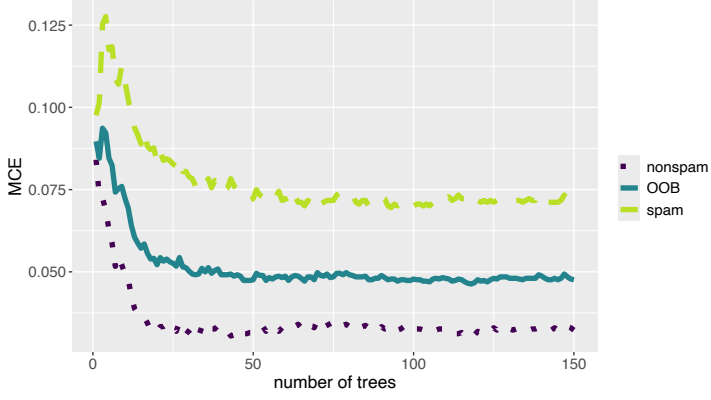
4: **end for**
5: Average losses over all observations:

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

USING THE OUT-OF-BAG ERROR ESTIMATE

- Gives us a (proper) estimator of GE, computable during training
- Can even compute this for all smaller ensemble sizes (after we fitted *M* models)

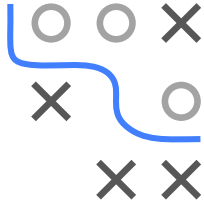




OOB ERROR: COMPARABILITY, BEST PRACTICE

OOB Size: The probability that an observation is out-of-bag (OOB) is:

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



$\Rightarrow$ similar to holdout or 3-fold CV (1/3 validation, 2/3 training)

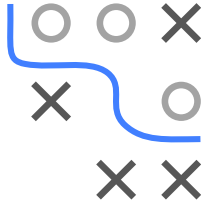
Comparability Issues:

- **OOB error** rather unique to RFs / bagging
- To compare models, we often still use CV, etc., to be consistent

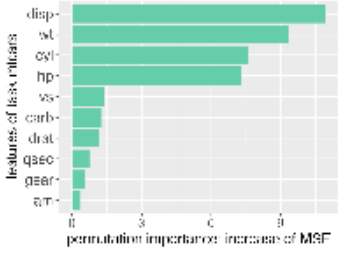
Use the OOB Error for:

- Get first impression of RF performance
- Select ensemble size
- Efficiently evaluate different RF hyperparameter configurations

Introduction to Machine Learning



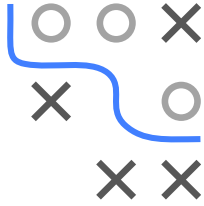
Random Forest
Feature Importance



Learning goals

- Understand that the goal of feature importance is to enhance interpretability of RF
- Understand FI based on feature permutation
- Understand FI based on improvement in splits

PERMUTATION FEATURE IMPORTANCE

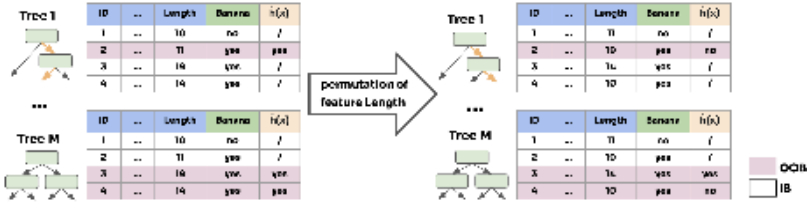


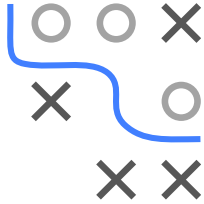
RFs improve accuracy by aggregating multiple decision trees but **lose interpretability** compared to a single tree. **Feature importance** mitigates this problem.

- How much does performance *decrease*, if feature is removed / rendered useless?
- We permute values of considered feature
- Removes association between feature and target, keeps marginal distribution
- Can obtain $\widehat{\mathrm{GE}}$ of RF (without and with permuted features) by predicting OOB data, to **efficiently compute FI during training**
- Avoids not only new models (if feature would be removed) but can already use "OOB test data" during training

ID	Color	Form	Origin	Length	Banana
1	yellow	round	domestic	10	no
2	brown	oblong	imported	11	yes
3	green	oblong	imported	19	yes
4	yellow	oblong	domestic	14	yes

PERMUTATION IMPORTANCE

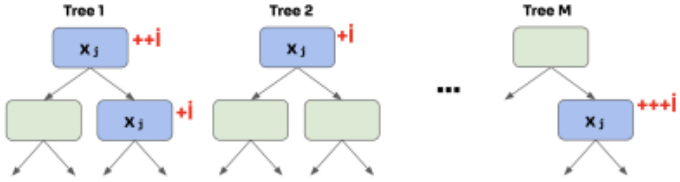


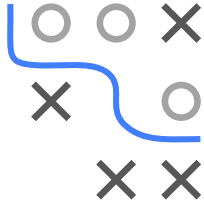


1: Calculate $\widehat{\mathrm{GE}}_{\mathrm{OOB}}$ using set-based metric ρ
2: **for** features x_j, $j = 1 \to p$ **do**
3: **for** Some statistical repetitions **do**
4: Distort feature-target relation: permute x_j with ψ_j
5: Compute all n OOB-predictions for permuted feature data, obtain all $\hat{f}^{(i)}_{\mathrm{OOB},\psi_j}$
6: Arrange predictions in $\hat{\boldsymbol{F}}_{\mathrm{OOB},\psi_j}$; Compute $\widehat{\mathrm{GE}}_{\mathrm{OOB},j} = \rho(\mathbf{y}, \hat{\boldsymbol{F}}_{\mathrm{OOB},\psi_j})$
7: Estimate importance of j-th feature: $\widehat{\mathrm{FI}}_j = \widehat{\mathrm{GE}}_{\mathrm{OOB},j} - \widehat{\mathrm{GE}}_{\mathrm{OOB}}$
8: **end for**
9: Average obtained $\widehat{\mathrm{FI}}_j$ values over reps
10: **end for**

IMPURITY IMPORTANCE

Alternative: Add up all *improvements* in splits where feature x_j is used.

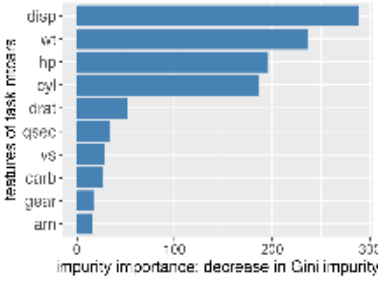


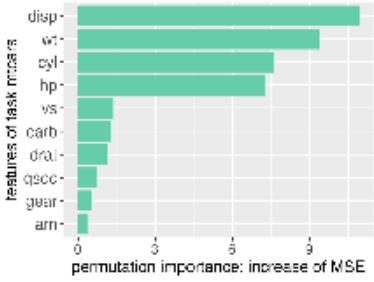


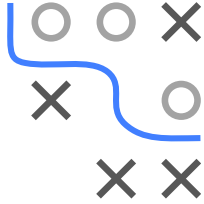
1: **for** features x_j, $j = 1 \to p$ **do**
2: **for** all models $\hat{b}^{[m]}$, $m = 1 \to M$ **do**
3: Find all splits in $\hat{b}^{[m]}$ on x_j
4: Extract improvement / risk reduction for these splits
5: Sum them up
6: **end for**
7: Add up improvements over all trees for FI of x_j
8: **end for**

IN PRACTICE / OUTLOOK

Let's compare both FI variants on `mtcars`:

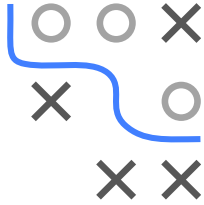




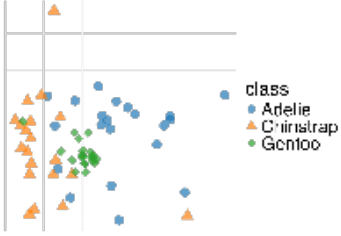


- Both methods are **biased toward features with more levels** (i.e., continuous or categoricals with many categories) ▸ Strobl et al. 2007
- More advanced versions exist
- PFI and FI have been generalized, see our lecture on IML!

Introduction to Machine Learning



Random Forest Proximities

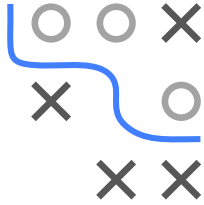


Learning goals

- Understand how RF can be used to define proximities of observations
- Know how proximities can be used for visualization, outlier detection and imputation

PROXIMITIES

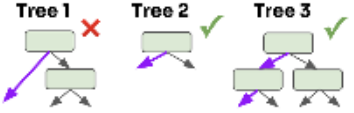
RFs have built-in similarity measure for pairs of observations:



ID	Color	Form	Length
1	yellow	oblong	14

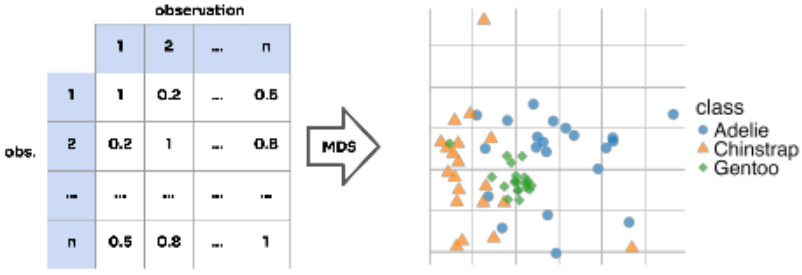


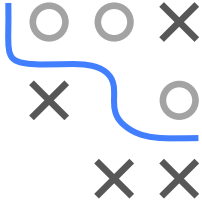
ID	Color	Form	Length
2	brown	oblong	10



- After training, push all observations through each tree
- To calculate prox $\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right)$: Percentage of how often both points are placed in **same terminal node of a tree**
- Here: prox $\left(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}\right) = 2/3$
- All proximities are arranged in symmetric $n \times n$ matrix

VISUALIZING PROXIMITIES



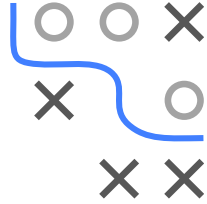


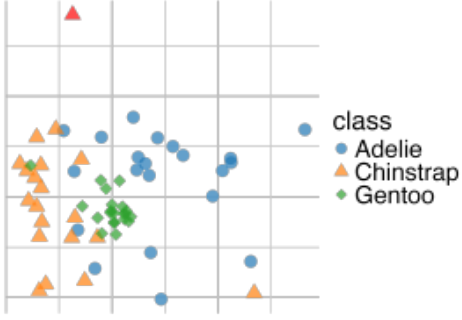
Can visualize the proximity matrix by projecting it into lower-dim. space, e.g., via multidim. scaling (might have to turn proximities into distances)

- Samples from same class usually form **identifiable clusters**
- **Offers some error-inspection**, e.g., Adelie has high within-class variance and has overlaps with other classes

OUTLIER DETECTION

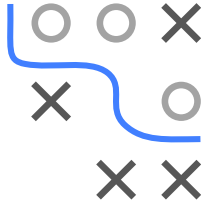
- Can also be used to **locate outliers**
- Or mislabeled points, especially in manually labeled data sets





IMPUTING MISSING DATA

ID	Color	Form	Origin	Length
1	yellow	round	domestic	14
2	brown	oblong	imported	???
3	brown	oblong	imported	19
4	???	round	domestic	14

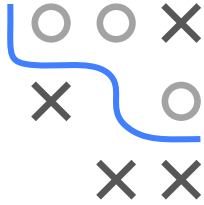


1. Replace missings per feature by median (of available values)
2. Compute proximities (NB: data has changed)
3. Replace missings in $\mathbf{x}^{(i)}$ by weighted average of non-missings; weights proportional to proximities

Steps 2 and 3 are iterated a few times.

IMPUTING MISSING DATA

ID	Color	Form	Origin	Length
1	yellow	round	domestic	14
2	brown	oblong	imported	14
3	brown	oblong	imported	19
4	brown	round	domestic	14



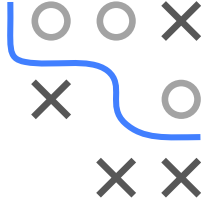
1. Replace missings per feature by median (of available values)
2. Compute proximities (NB: data has changed)
3. Replace missings in $\mathbf{x}^{(i)}$ by weighted average of non-missings; weights proportional to proximities

Steps 2 and 3 are iterated a few times.

IMPUTING MISSING DATA

ID	Color	Form	Origin	Length
1	yellow	round	domestic	14
2	brown	oblong	imported	17
3	brown	oblong	imported	19
4	brown	round	domestic	14

weighted average
using proximities



1. Replace missings per feature by median (of available values)
2. Compute proximities (NB: data has changed)
3. Replace missings in $\mathbf{x}^{(i)}$ by weighted average of non-missings; weights proportional to proximities

Steps 2 and 3 are iterated a few times.

INTRODUCTION TO MACHINE LEARNING

ML Basics

Supervised Regression

Supervised Classification

Performance Evaluation

k-NN

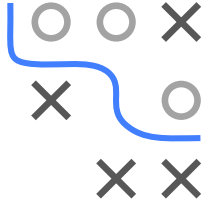
Classification and Regression Trees (CART)

Random Forests

Neural Networks

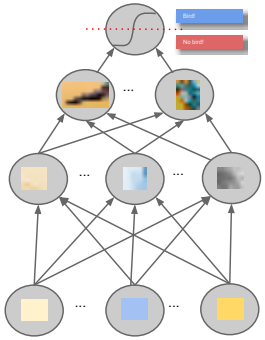
Tuning

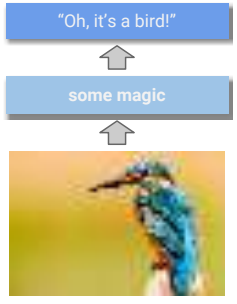
Nested Resampling



Deep Learning

Introduction

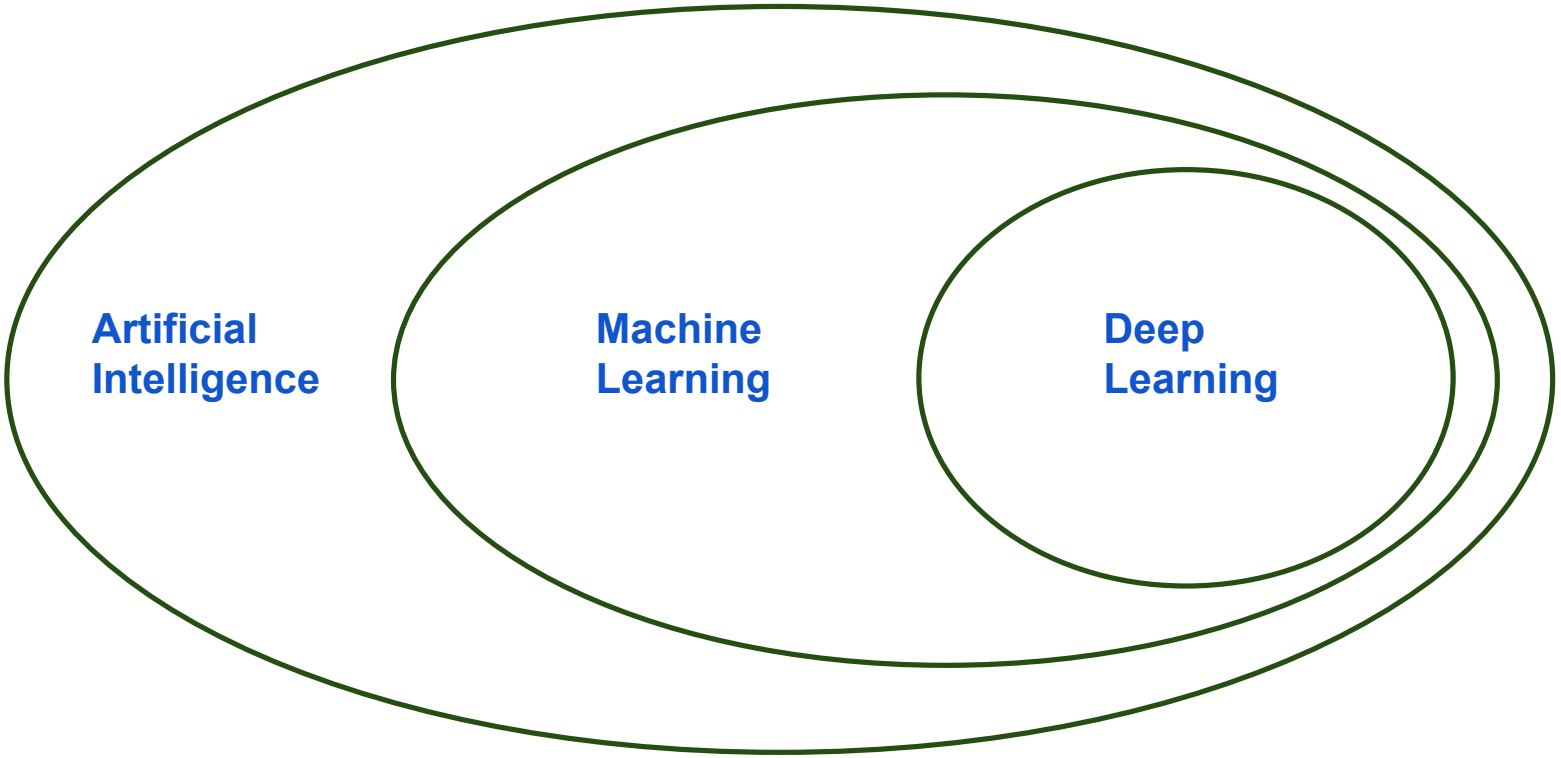




Learning goals

- 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 subield of ML based on artificial neural networks.

DEEP LEARNING AND NEURAL NETWORKS

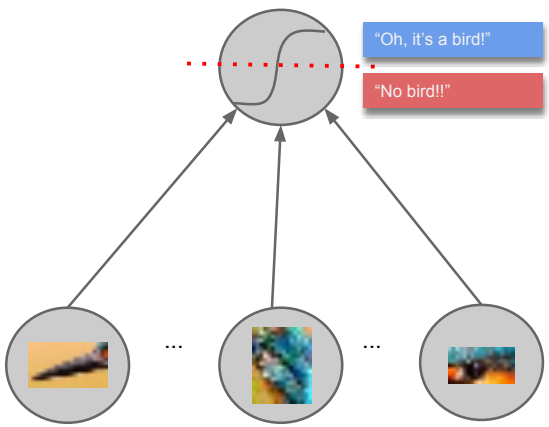
- Deep learning itself is not *new*:
  - Neural networks have been around since the 70s.
  - *Deep* neural networks, i.e., networks with multiple hidden layers, are not much younger.

- Why everybody is talking about deep learning now:
  1. Specialized, powerful hardware allows training of huge neural networks to push the state-of-the-art on difficult problems.
  2. Large amount of data is available.
  3. Special network architectures for image/text data.
  4. 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



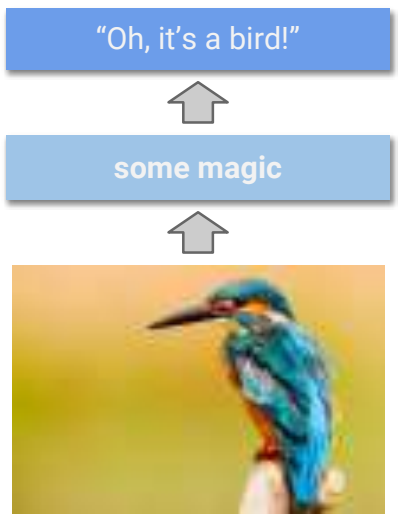
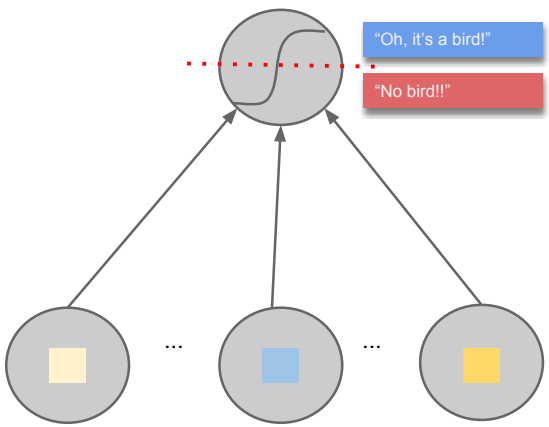


IMAGE CLASSIFICATION WITH NEURAL NETWORKS

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

Y. Bengio



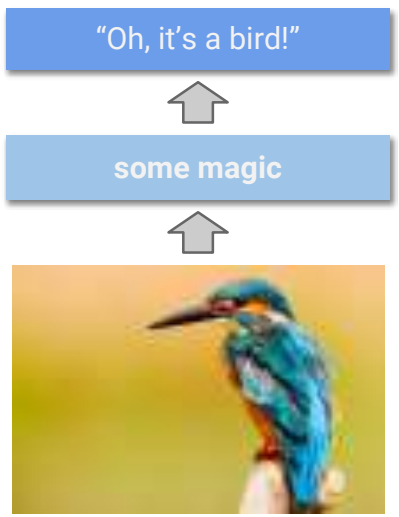
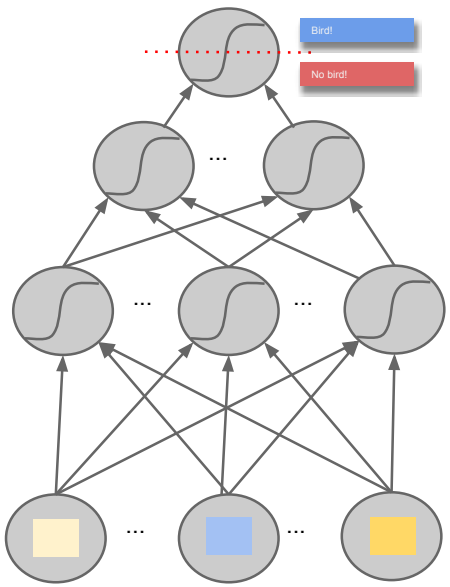


IMAGE CLASSIFICATION WITH NEURAL NETWORKS

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

Y. Bengio



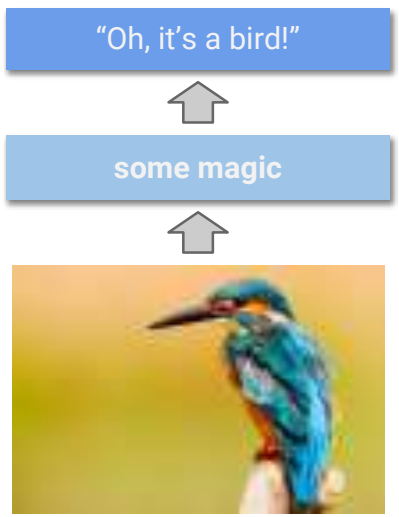
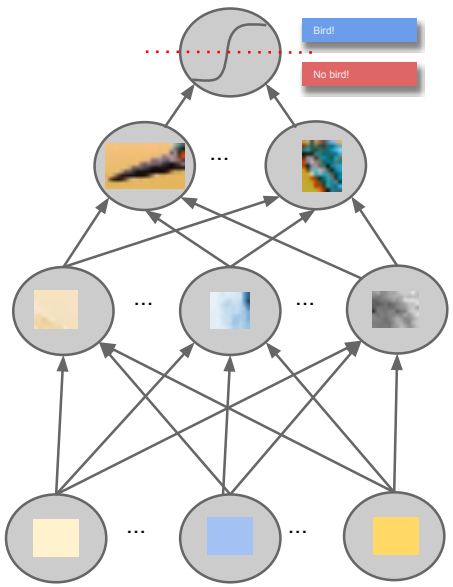
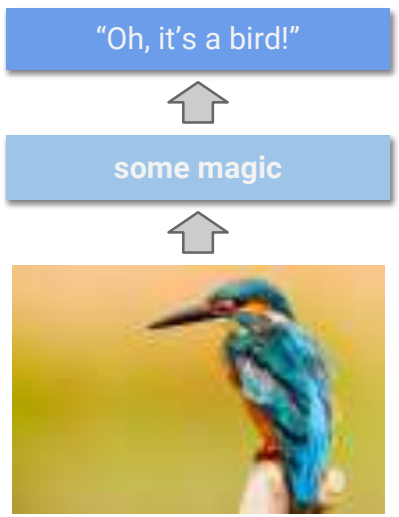


IMAGE CLASSIFICATION WITH NEURAL NETWORKS

"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.

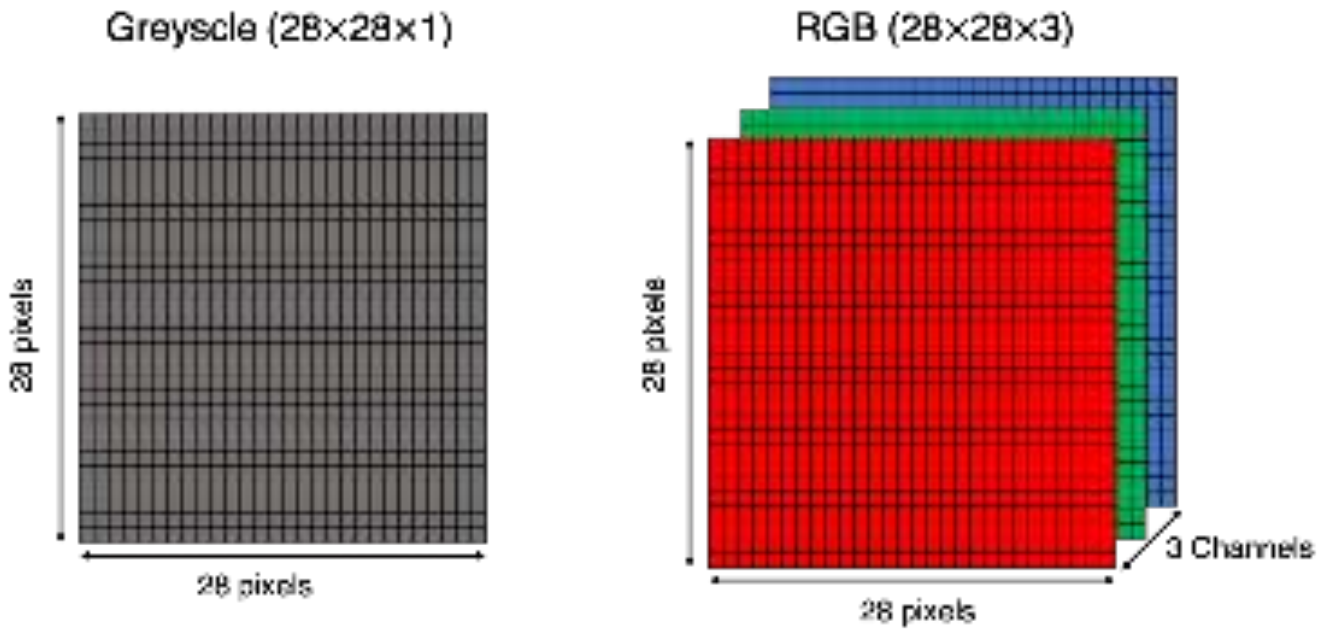
This implies that for tabular data, deep learning is rarely the correct model choice.

- 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 255×255 (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: **C**onvolutional **N**eural **N**etworks (CNN)



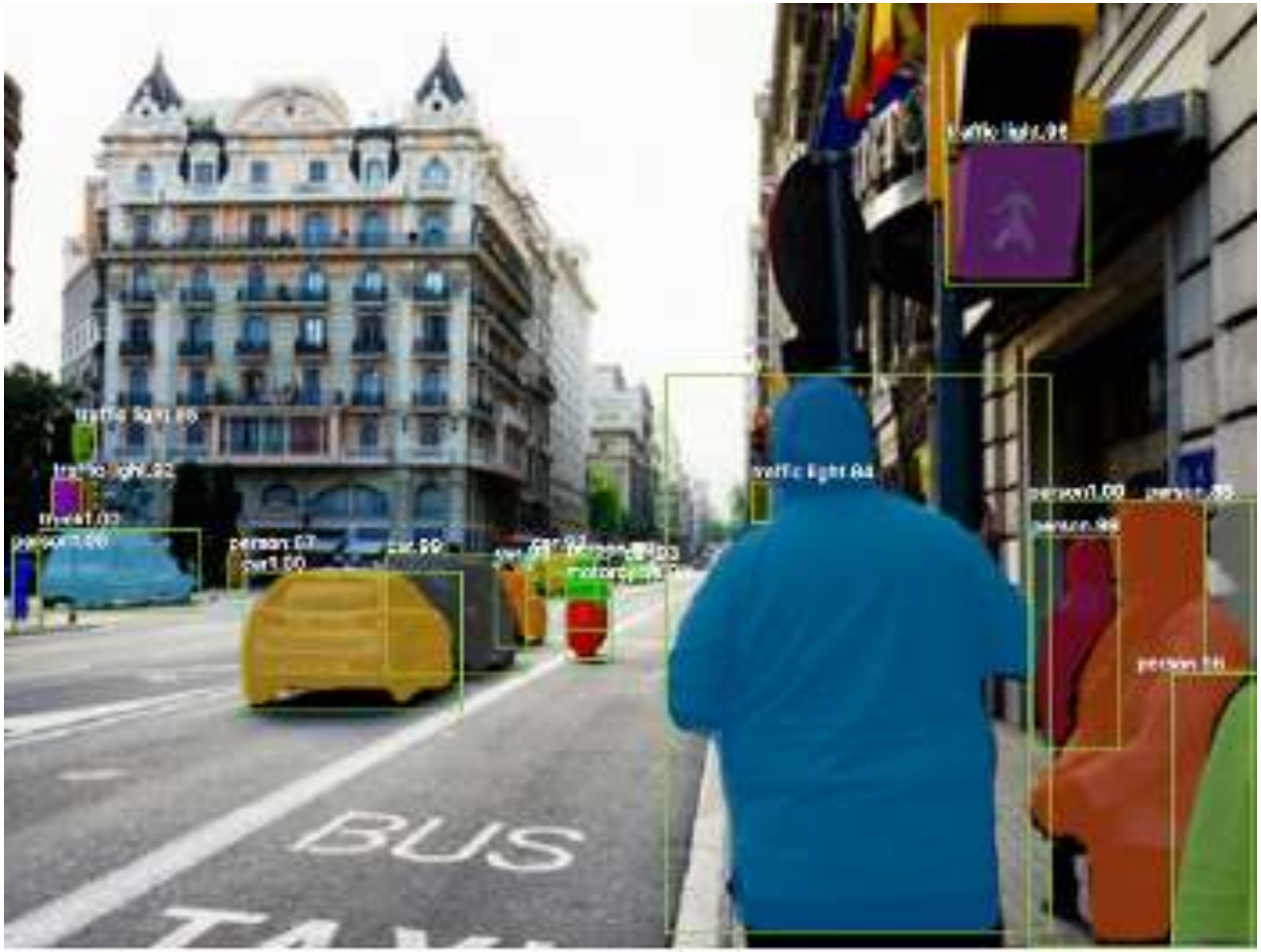
POSSIBLE USE-CASE: IMAGES



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 32*x*32 color images containing one of 10 object classes, with 6000 images per class.

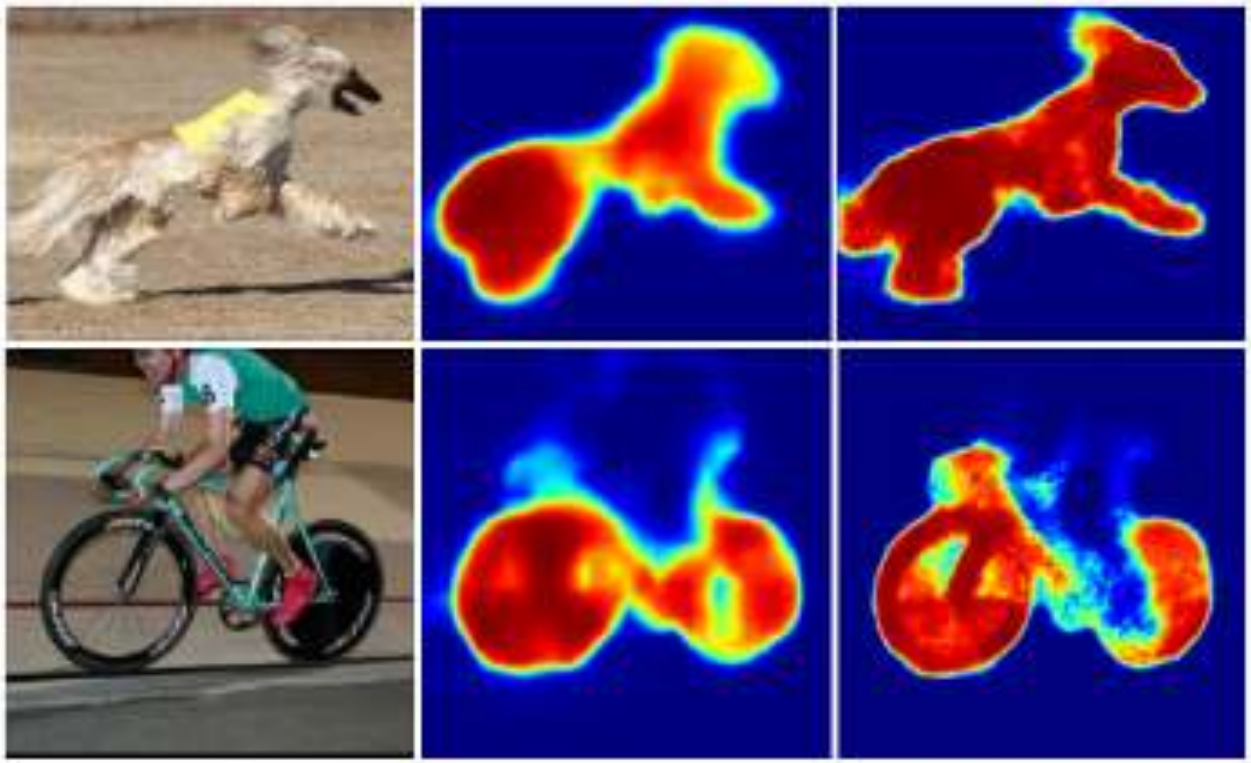
POSSIBLE USE-CASE: IMAGES



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



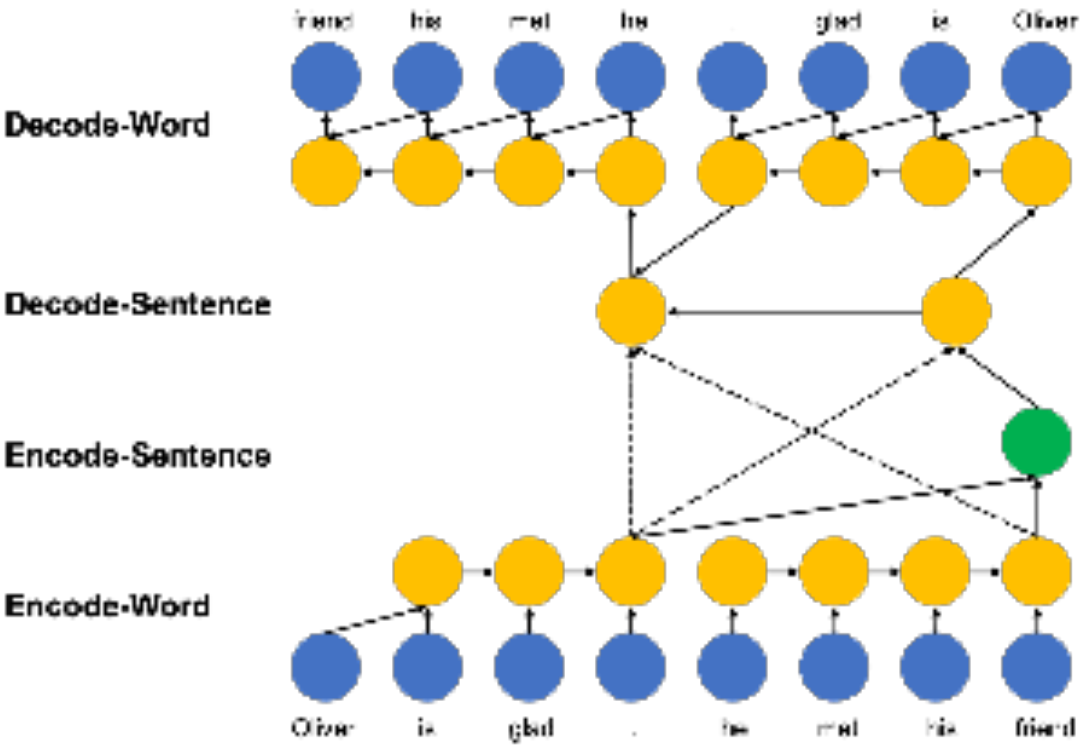
Credit: Hyeonwoo Noh (2015)

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: **R**ecurrent **N**eural **N**etworks (RNN)



POSSIBLE USE-CASE: TEXT CLASSIFICATION



Positive

Great job! Your customer support is fantastic.



Neutral

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



Negative

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



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.

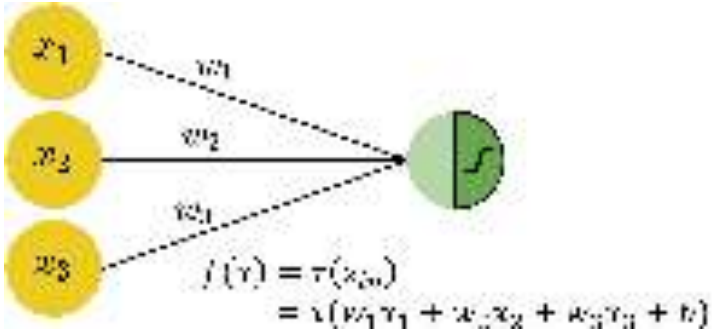
APPLICATIONS OF DEEP LEARNING: SPEECH



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

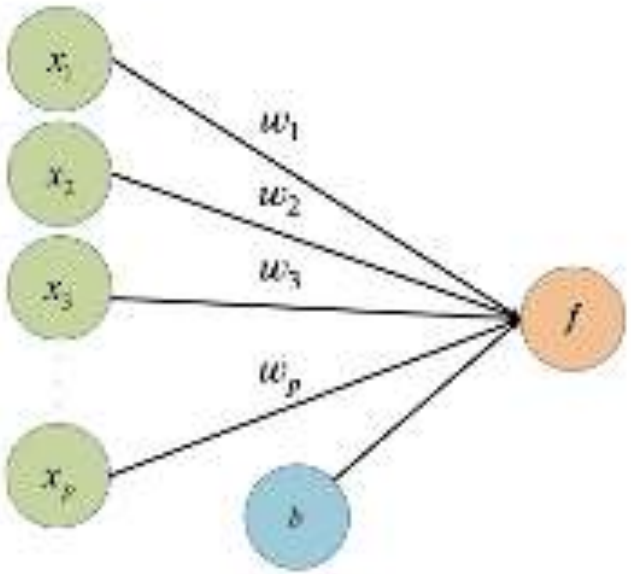
Single Neuron / Perceptron



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** $x_1, x_2, \ldots, x_p$, **weights** $w_1, w_2, \ldots, w_p$, **bias term** b, and **activation function** τ.

- 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 τ:

$$f(x) = \tau(w_1 x_1 + \ldots + w_p x_p + b) = \tau(\mathbf{w}^T \mathbf{x} + b)$$

A SINGLE NEURON

Activation function τ: a single neuron represents different functions depending on the choice of activation function.

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

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

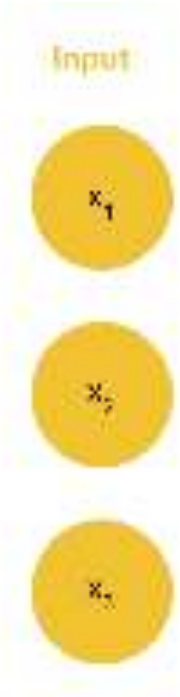
- The logistic function gives us the **logistic regression**:

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

A SINGLE NEURON

We consider a perceptron with 3-dimensional input, i.e.
$f(\mathbf{x}) = \tau(w_1 x_1 + w_2 x_2 + w_3 x_3 + b)$.

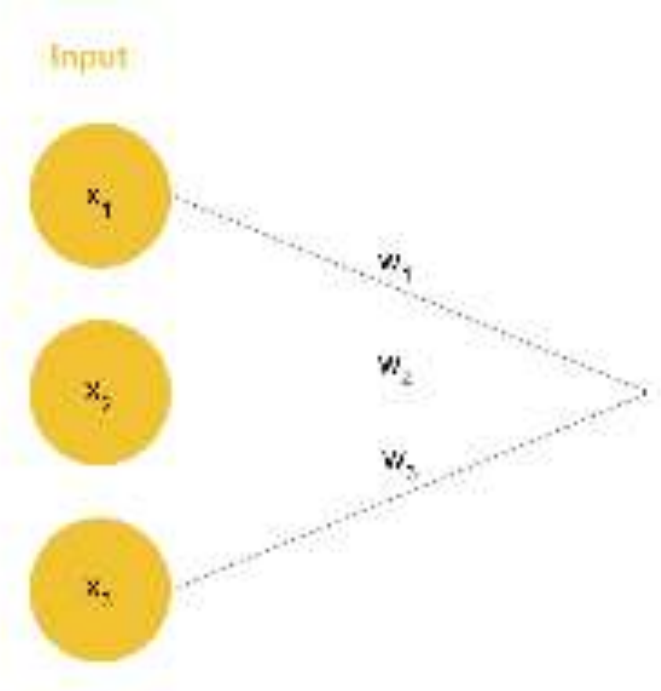
- Input features **x** are represented by nodes in the "input layer".



- In general, a p-dimensional input vector **x** will be represented by p nodes in the input layer.

A SINGLE NEURON

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



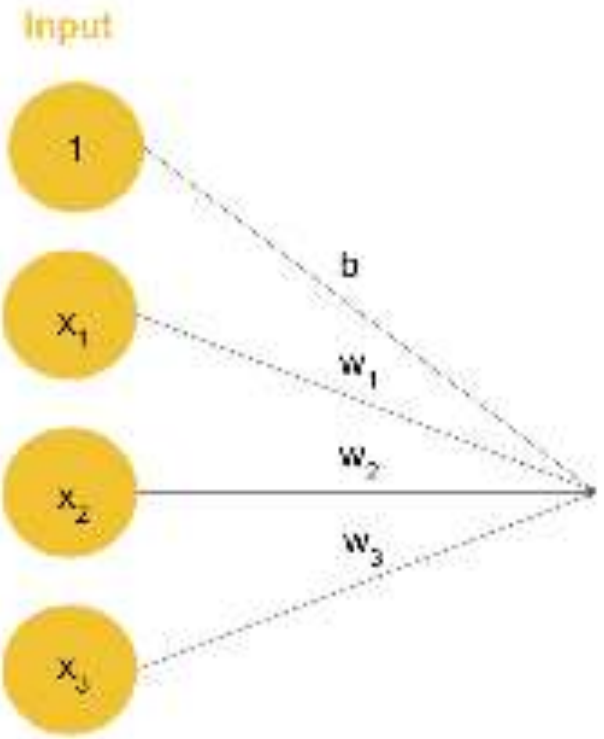
- The bias term *b* is implicit here. It is often not visualized as a separate node.

A SINGLE NEURON

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

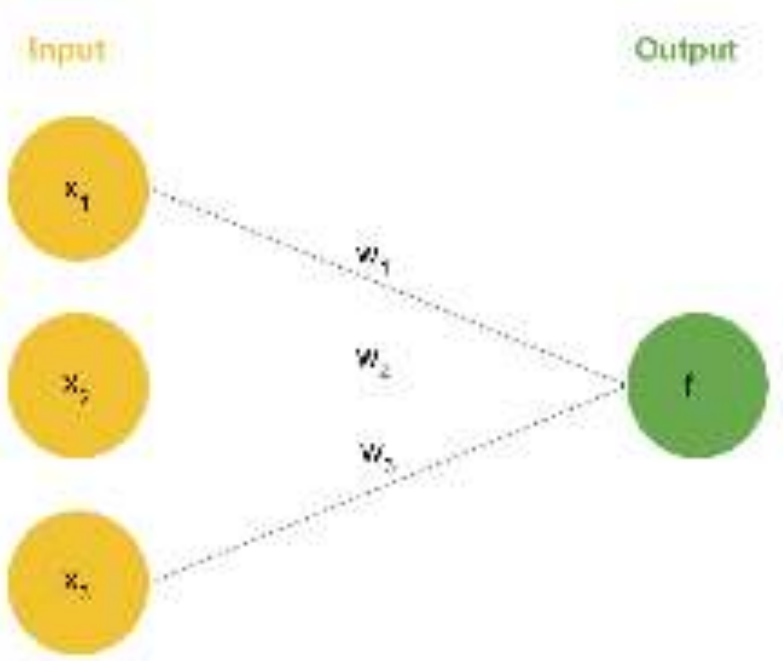
- Add a constant feature to the inputs $\tilde{\mathbf{x}} = (1, x_1, ..., x_p)^T$
- and absorb the bias into the weight vector $\tilde{\boldsymbol{w}} = (b, w_1, ..., w_p)$.

The graphical representation is then:



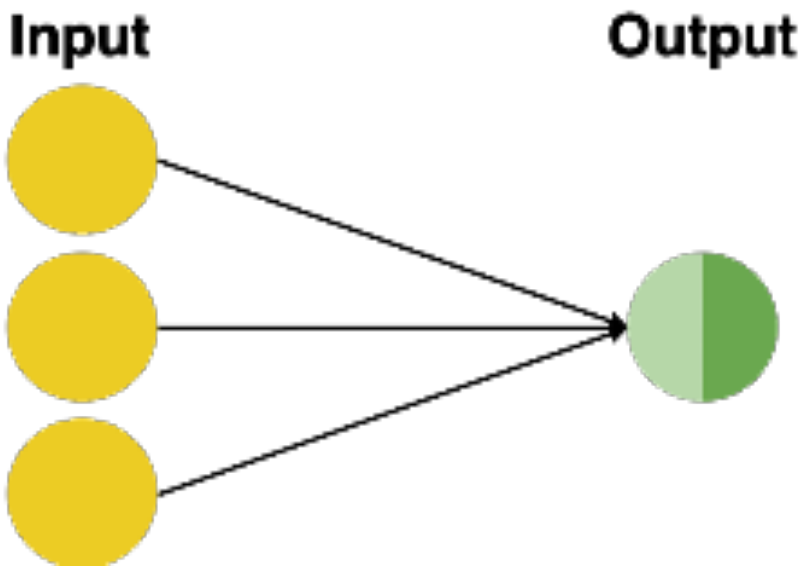
A SINGLE NEURON

- The computation $\tau(w_1 x_1 + w_2 x_2 + w_3 x_3 + b)$ is represented by the neuron in the “output layer”.



A SINGLE NEURON

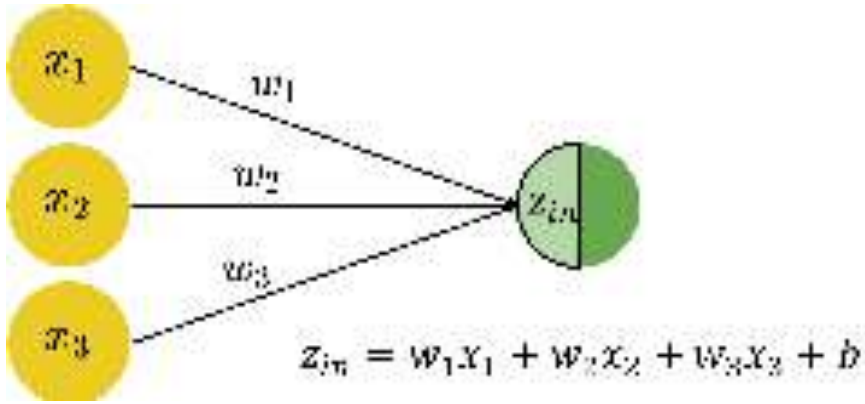
- You can picture the input vector being "fed" to neurons on the left followed by a sequence of computations performed from left to right. This is called a **forward pass**.



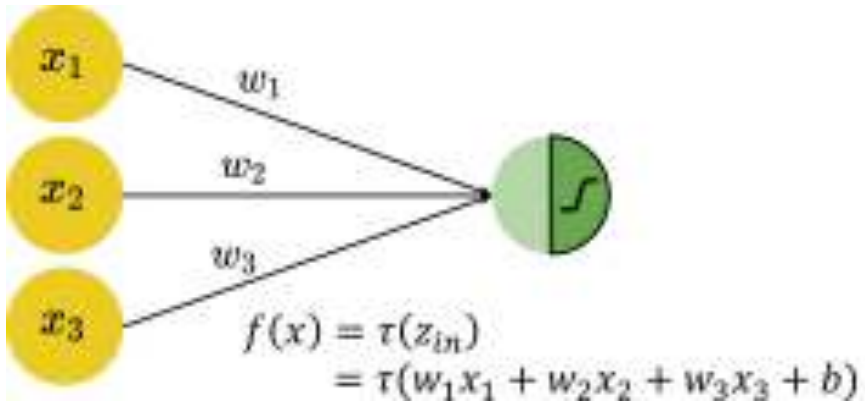
A SINGLE NEURON

A neuron performs a 2-step computation:

1. **Affine Transformation:** weighted sum of inputs plus bias.



2. **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

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

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

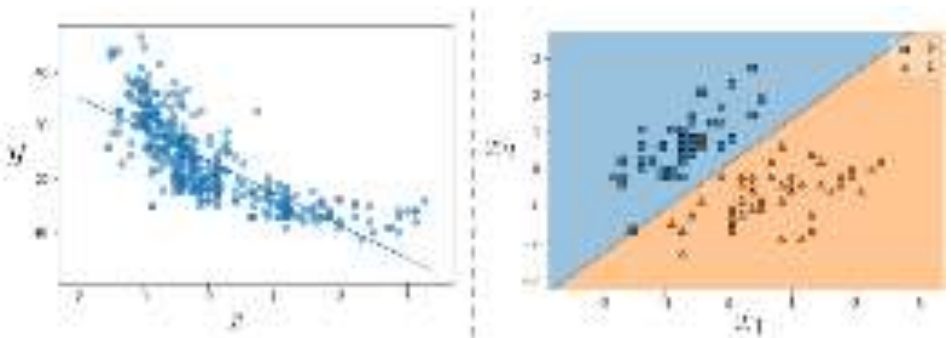


Figure: *Left*: A regression line learned by a single neuron. *Right*: A decision-boundary learned by a single neuron in a binary classification task.

A SINGLE NEURON: OPTIMIZATION

- To optimize this model, we minimize the empirical risk

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

  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):

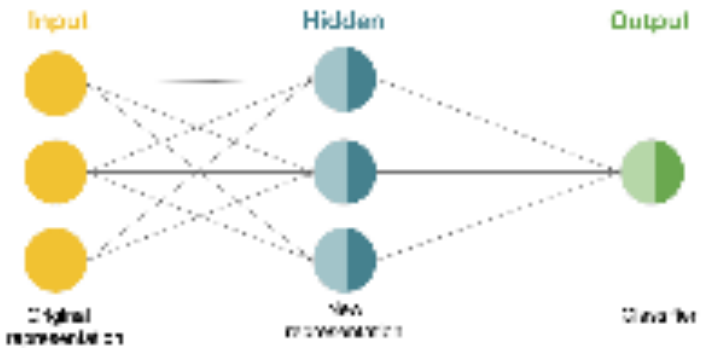
$$L(y, f(\mathbf{x})) = -(y \log f(\mathbf{x}) + (1 - y) \log(1 - f(\mathbf{x})))$$

A SINGLE NEURON: OPTIMIZATION

- For a single neuron and both choices of τ 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

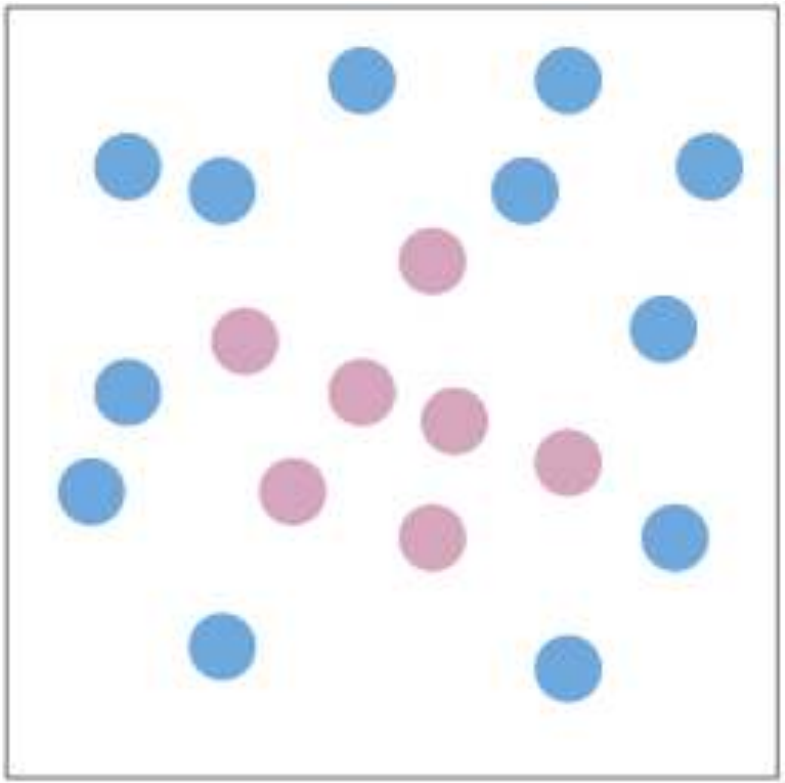
- Architecture of single hidden layer neural networks
- Representation learning/ understanding the advantage of hidden layers
- Typical (non-linear) activation functions

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).

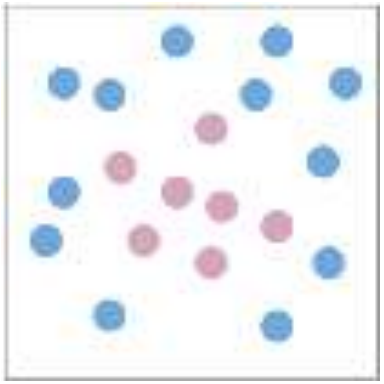
MOTIVATION

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:

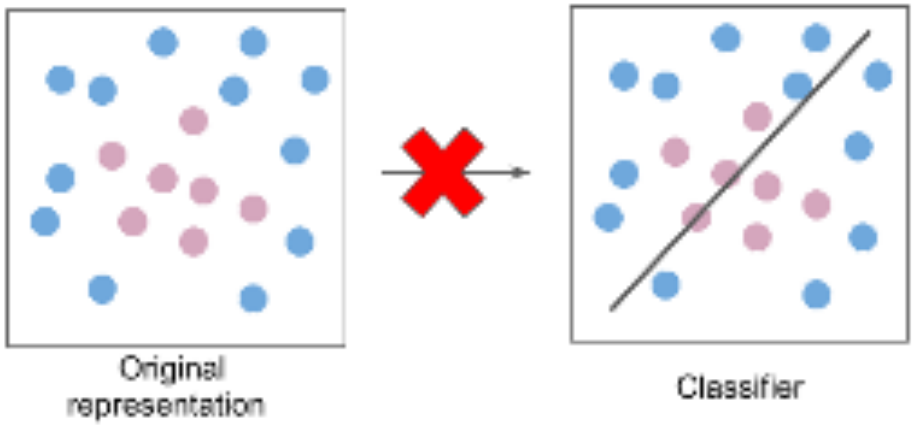


- However, the neuron can easily separate the classes if the original features are transformed (e.g., from Cartesian to polar coordinates):

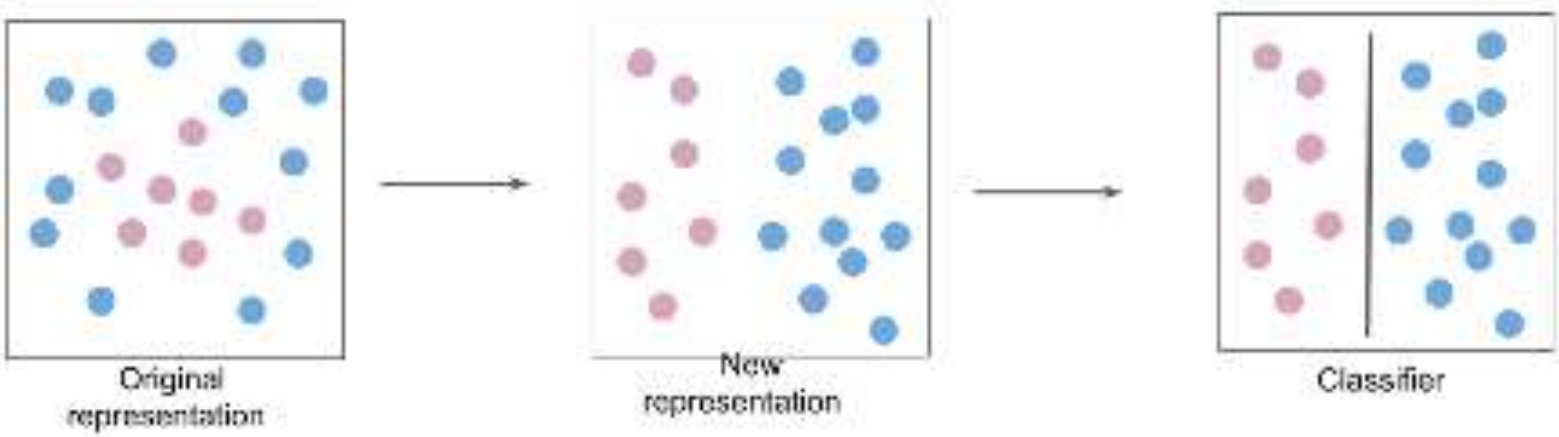


MOTIVATION

- Instead of classifying the data in the original representation,

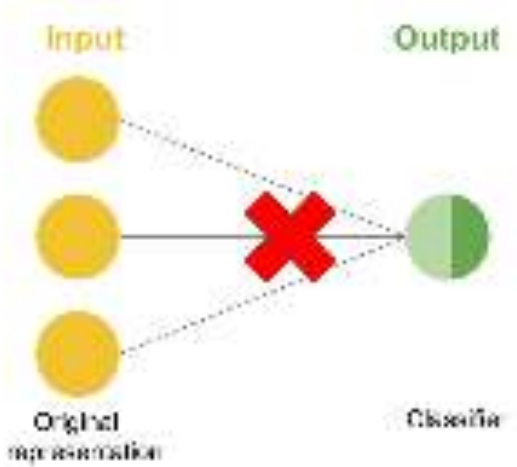


- we classify it in a new feature space.

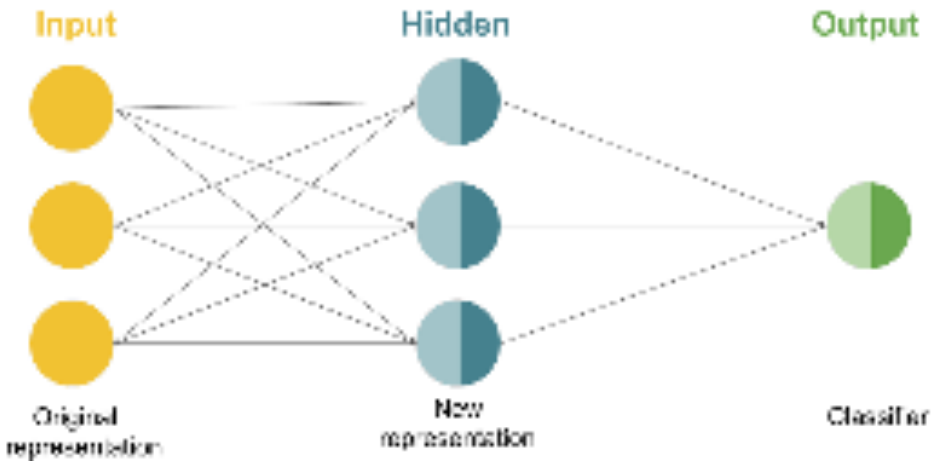


MOTIVATION

- 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.

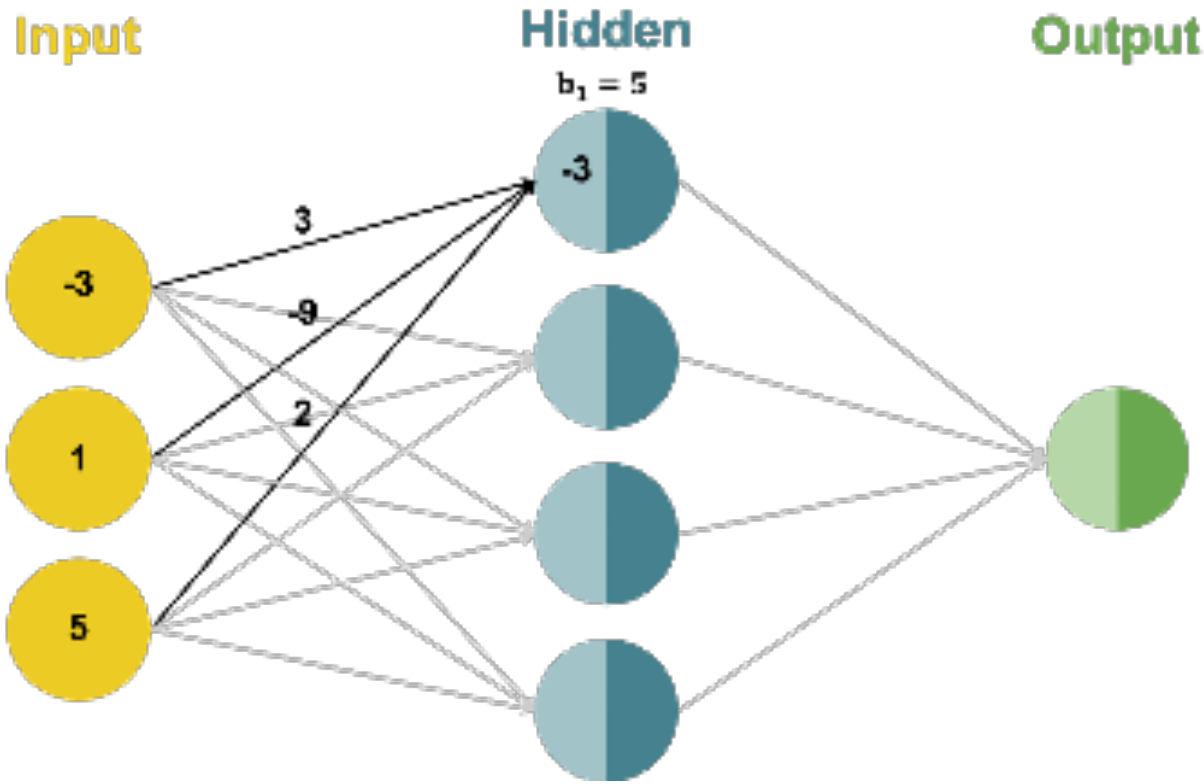
Single hidden layer networks consist of two layers (without input layer):

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

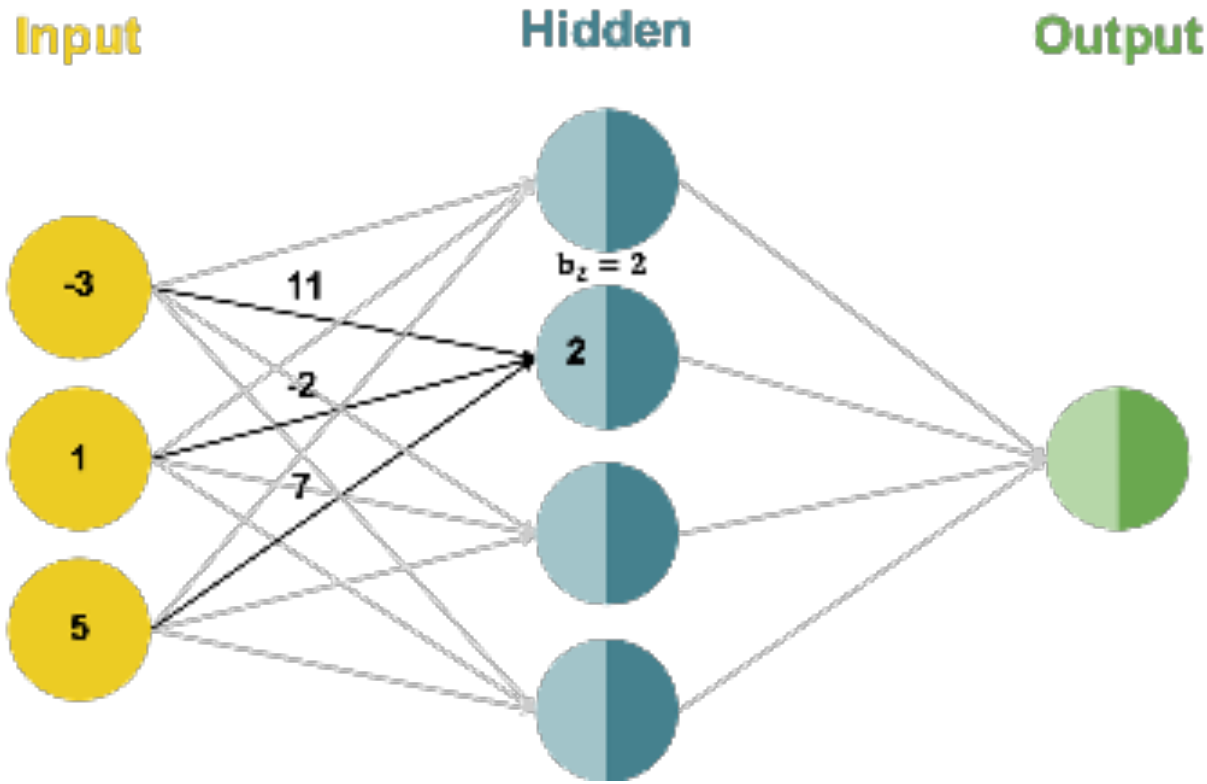
Each neuron in the hidden layer performs an **affine transformation** on the inputs:



$$z_{\text{in}}^{(1)} = w_{11}x^{(1)} + w_{21}x^{(2)} + w_{31}x^{(3)} + b_1$$
$$z_{\text{in}}^{(1)} = 3 * (-3) + (-9) * 1 + 2 * 5 + 5 = -3$$

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

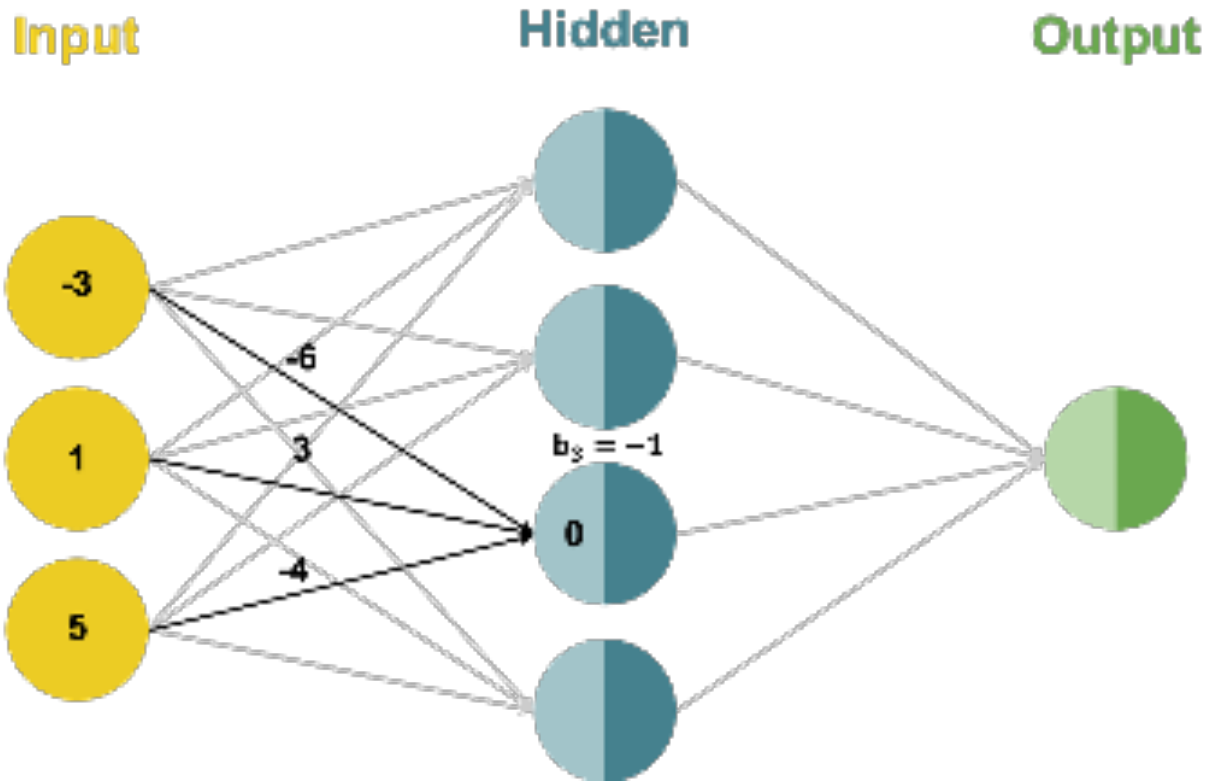
Each neuron in the hidden layer performs an **affine transformation** on the inputs:



$$z_{\text{in}}^{(2)} = w_{12}x^{(1)} + w_{22}x^{(2)} + w_{32}x^{(3)} + b_2$$
$$z_{\text{in}}^{(2)} = 11 * (-3) + (-2) * 1 + 7 * 5 + 2 = 2$$

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

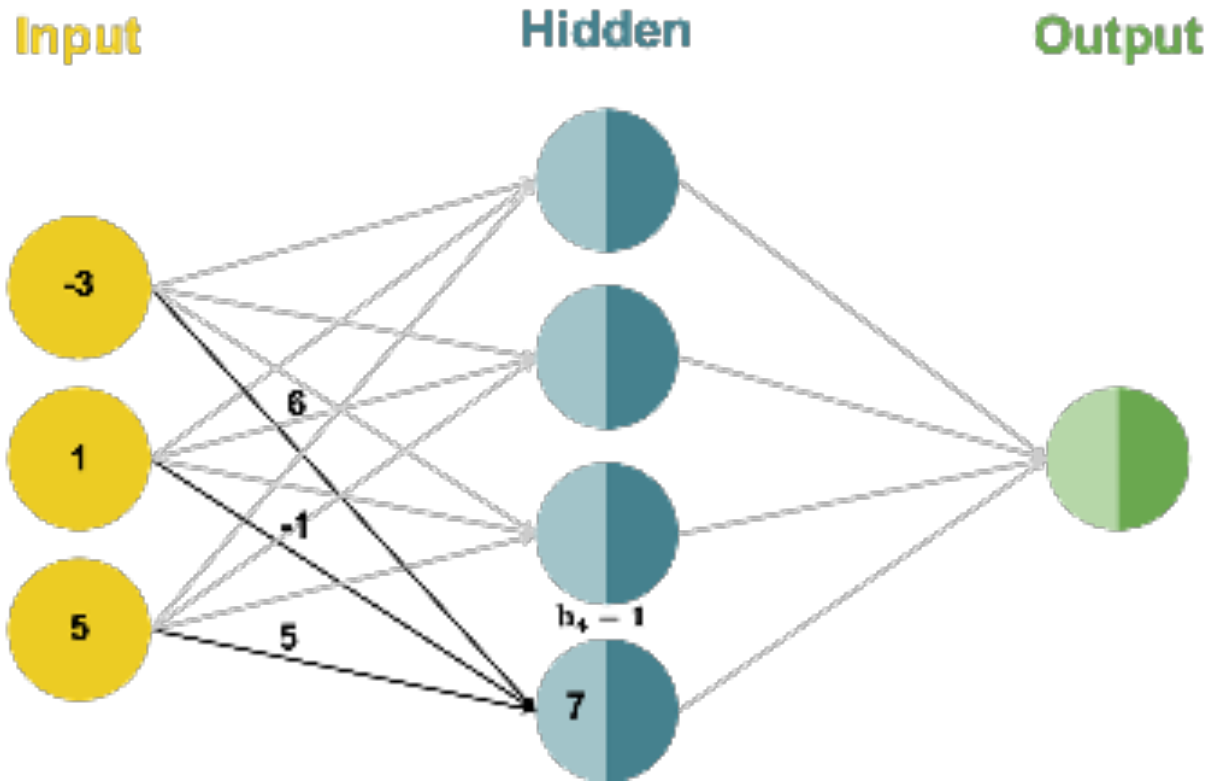
Each neuron in the hidden layer performs an **affine transformation** on the inputs:



$$z_{\text{in}}^{(3)} = w_{13}x^{(1)} + w_{23}x^{(2)} + w_{33}x^{(3)} + b_3$$
$$z_{\text{in}}^{(3)} = (-6) * (-3) + 3 * 1 + (-4) * 5 - 1 = 0$$

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

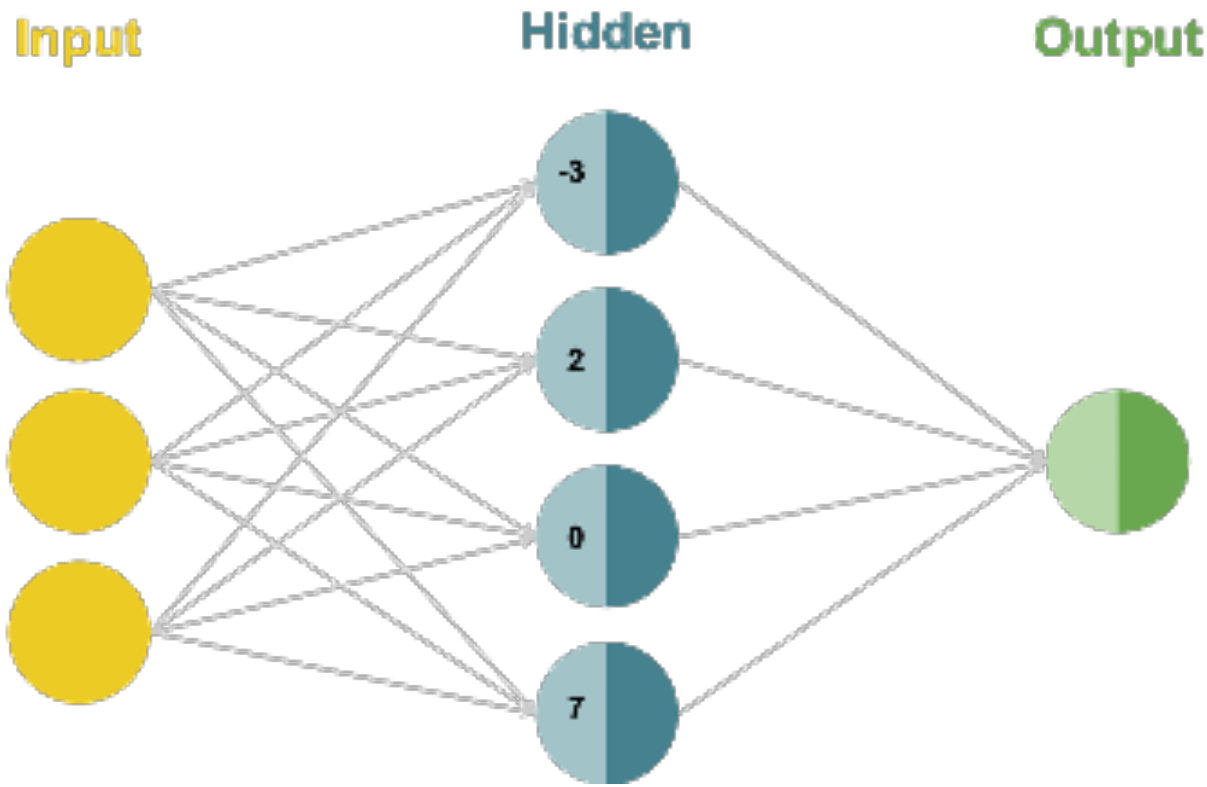
Each neuron in the hidden layer performs an **affine transformation** on the inputs:



$$z_{\text{in}}^{(4)} = w_{14}x^{(1)} + w_{24}x^{(2)} + w_{34}x^{(3)} + b_4$$
$$z_{\text{in}}^{(4)} = 6 * (-3) + (-1) * 1 + 5 * 5 + 1 = 7$$

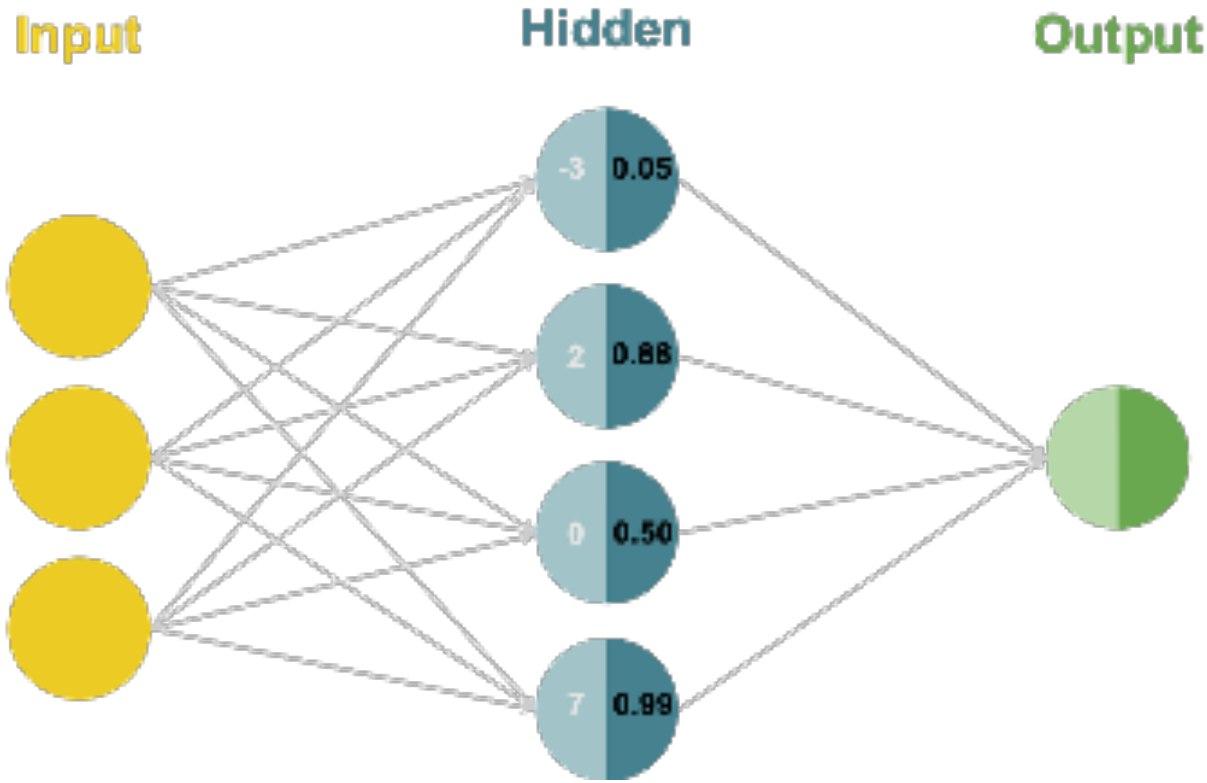
SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

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



SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

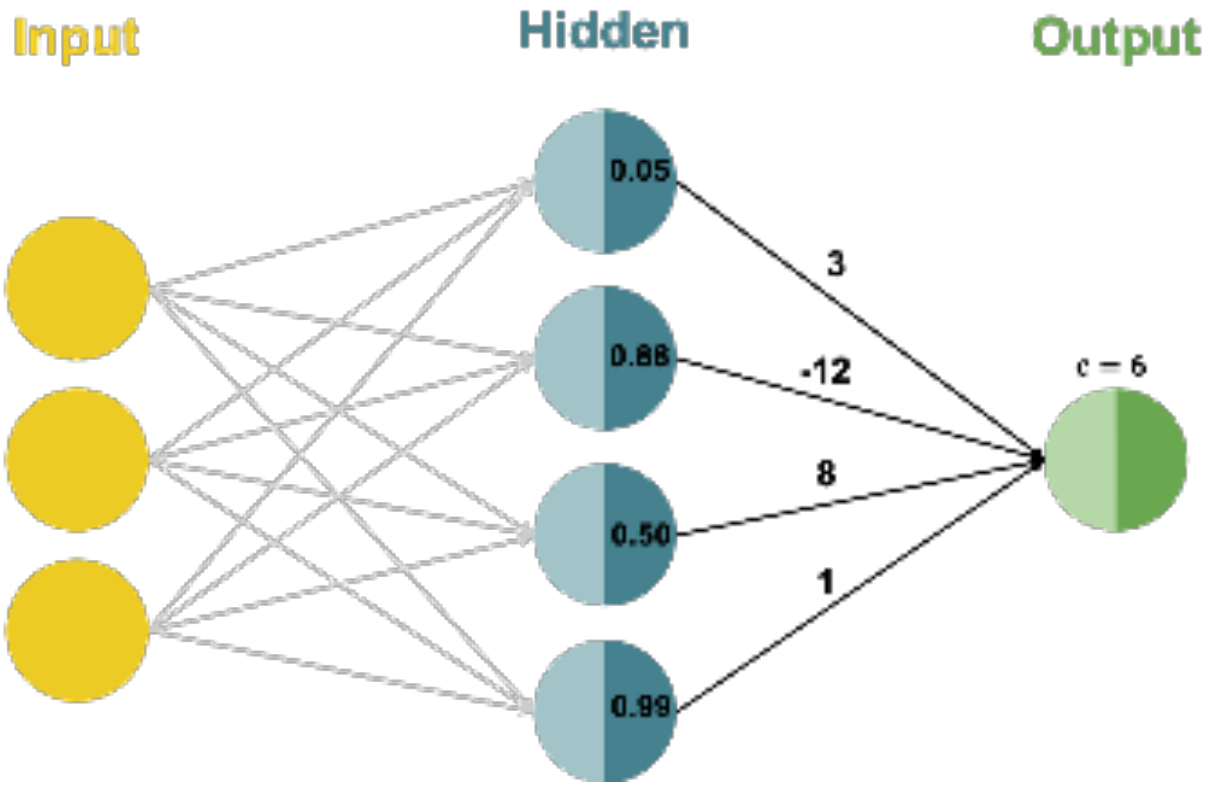
Each hidden neuron performs a non-linear **activation** transformation on the weight sum:



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

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

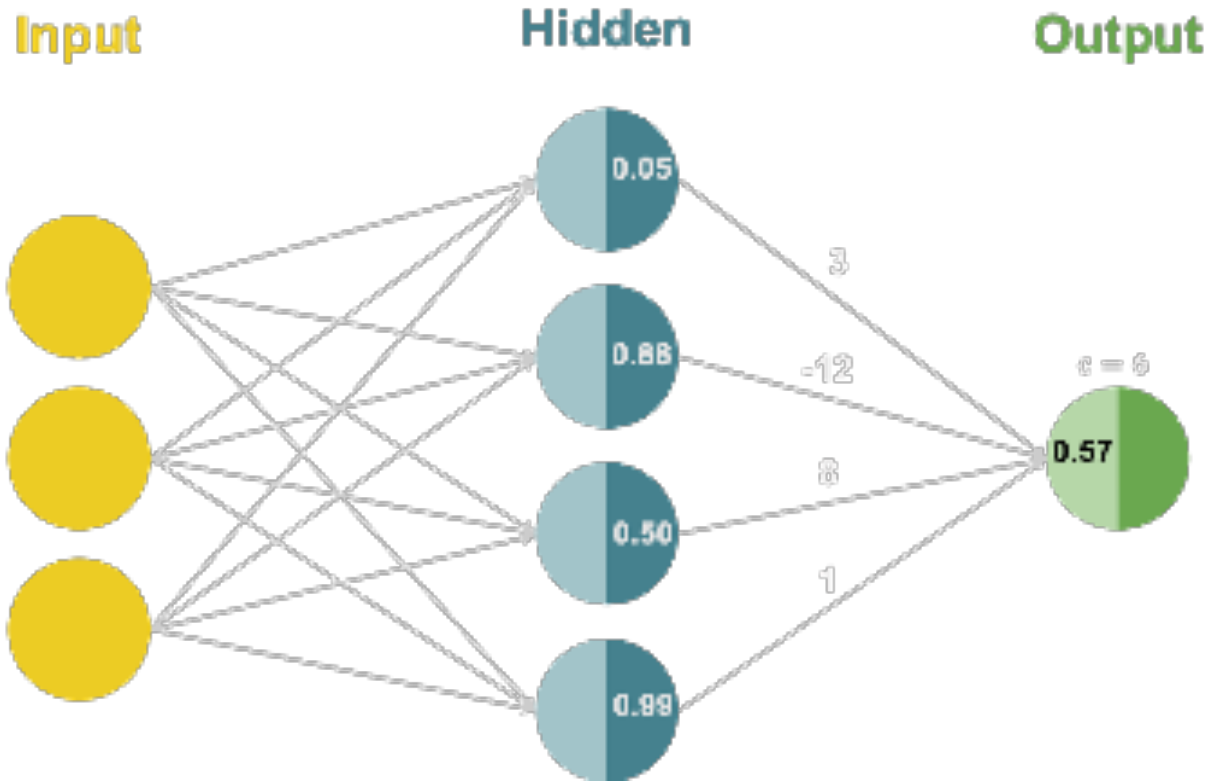
The output neuron performs an **affine transformation** on its inputs:



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

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

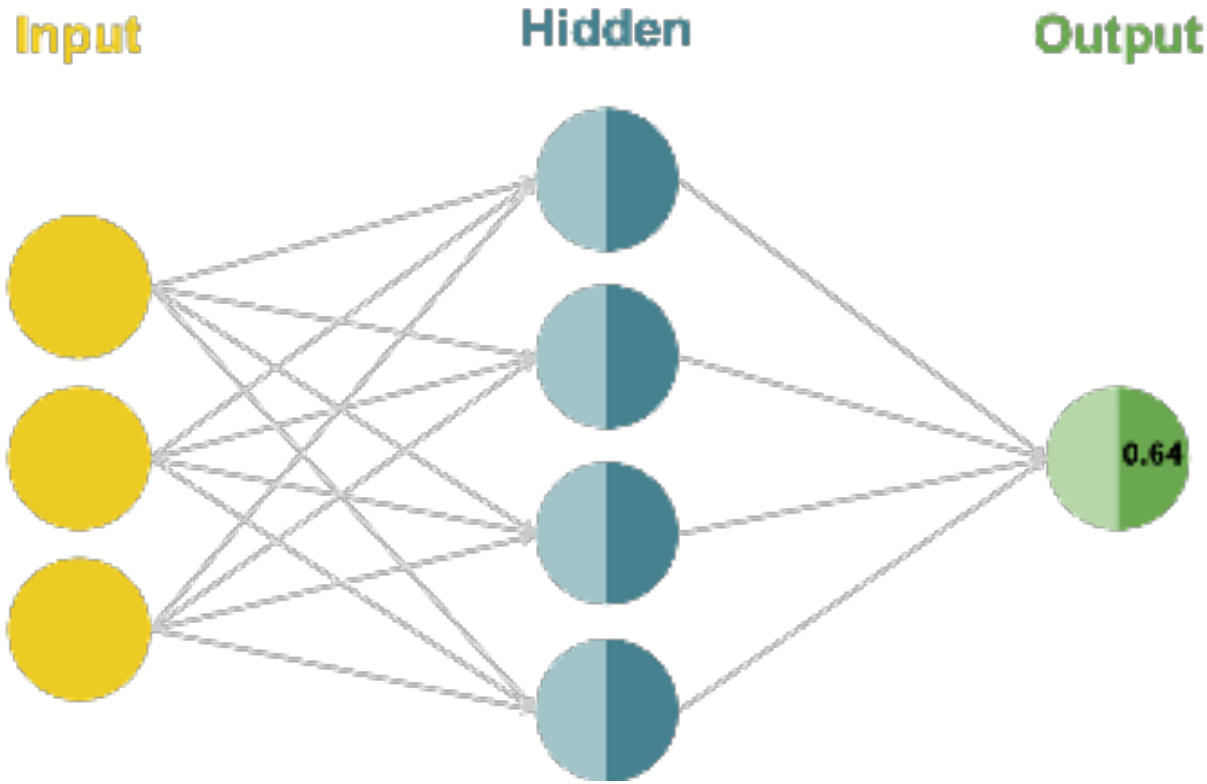
The output neuron performs an **affine transformation** on its inputs:



$$f_{\text{in}} = u_1 z_{\text{out}}^{(1)} + u_2 z_{\text{out}}^{(2)} + u_3 z_{\text{out}}^{(3)} + u_4 z_{\text{out}}^{(4)} + c$$
$$f_{\text{in}} = 3 * 0.05 + (-12) * 0.88 + 8 * 0.50 + 1 * 0.99 + 6 = 0.57$$

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

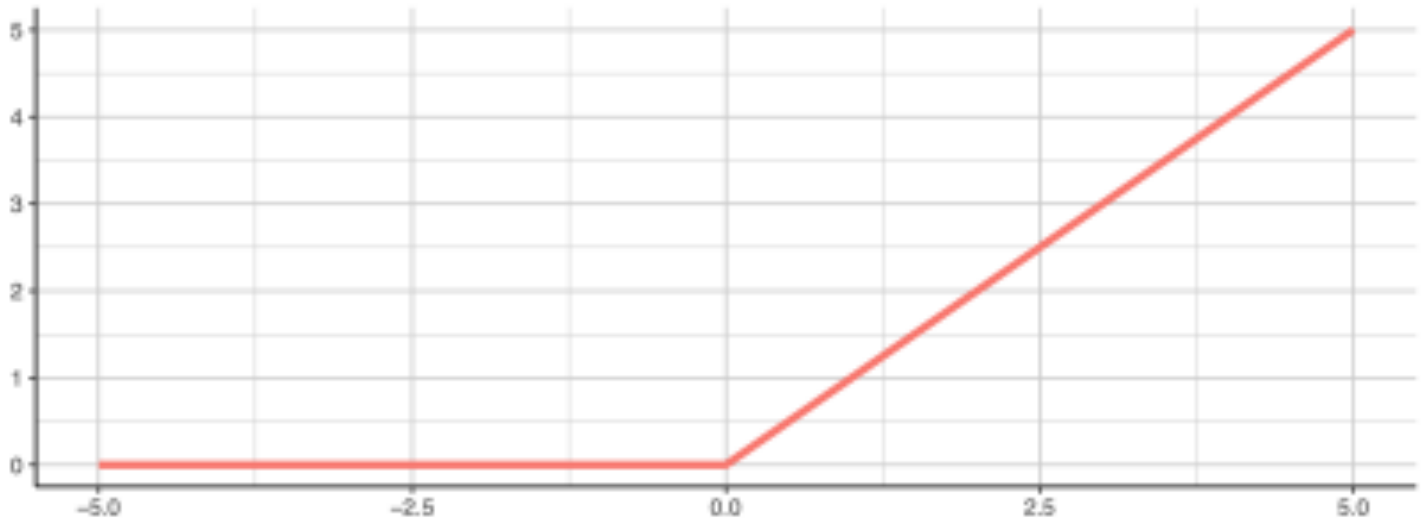
- If the hidden layer does not have a non-linear activation, the network can only learn linear decision boundaries.
- A lot of different activation functions exist.

HIDDEN LAYER: ACTIVATION FUNCTION

ReLU Activation:

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

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

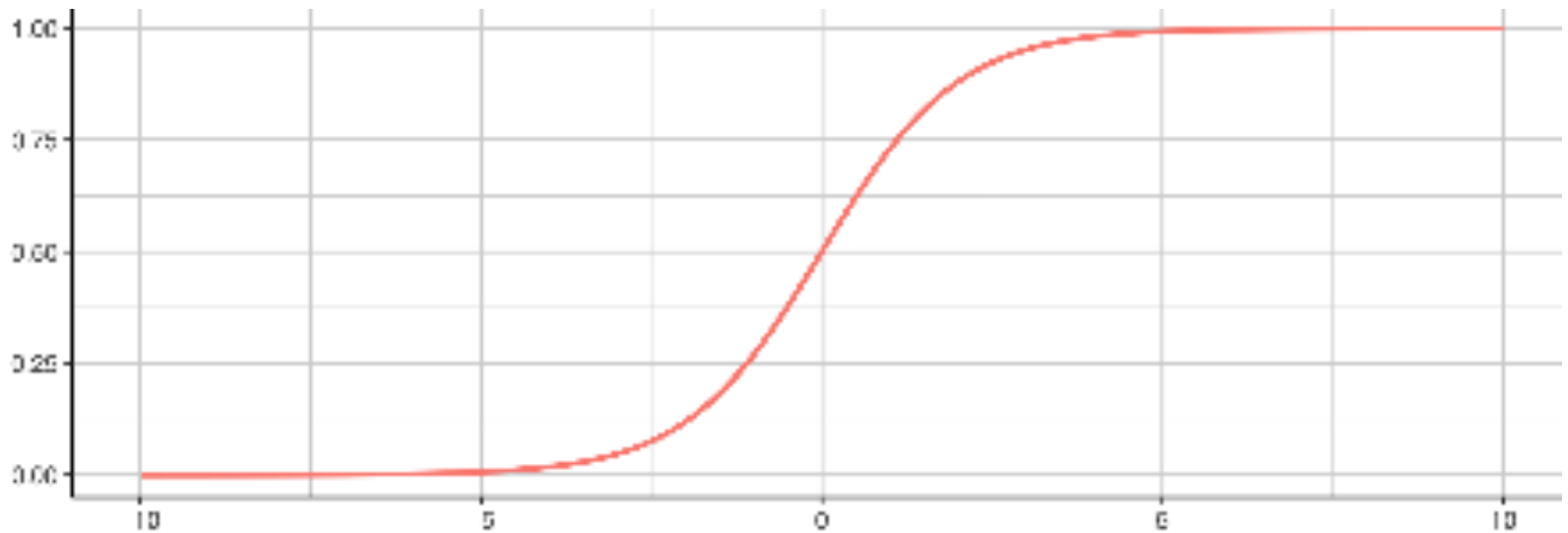


HIDDEN LAYER: ACTIVATION FUNCTION

Sigmoid Activation Function:

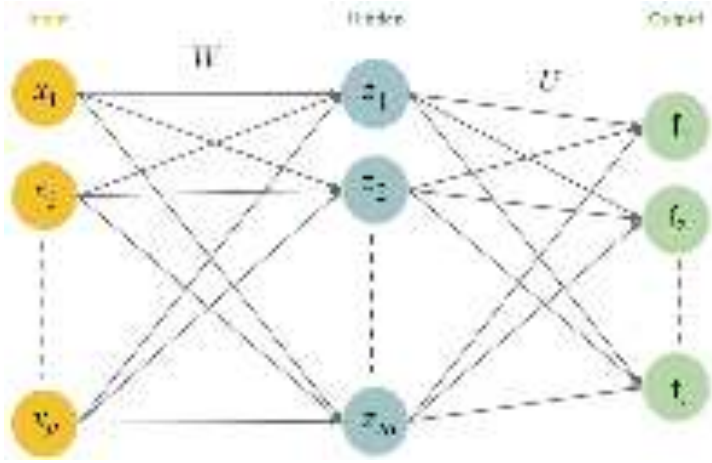
- The sigmoid function can be used even in the hidden layer:

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



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).

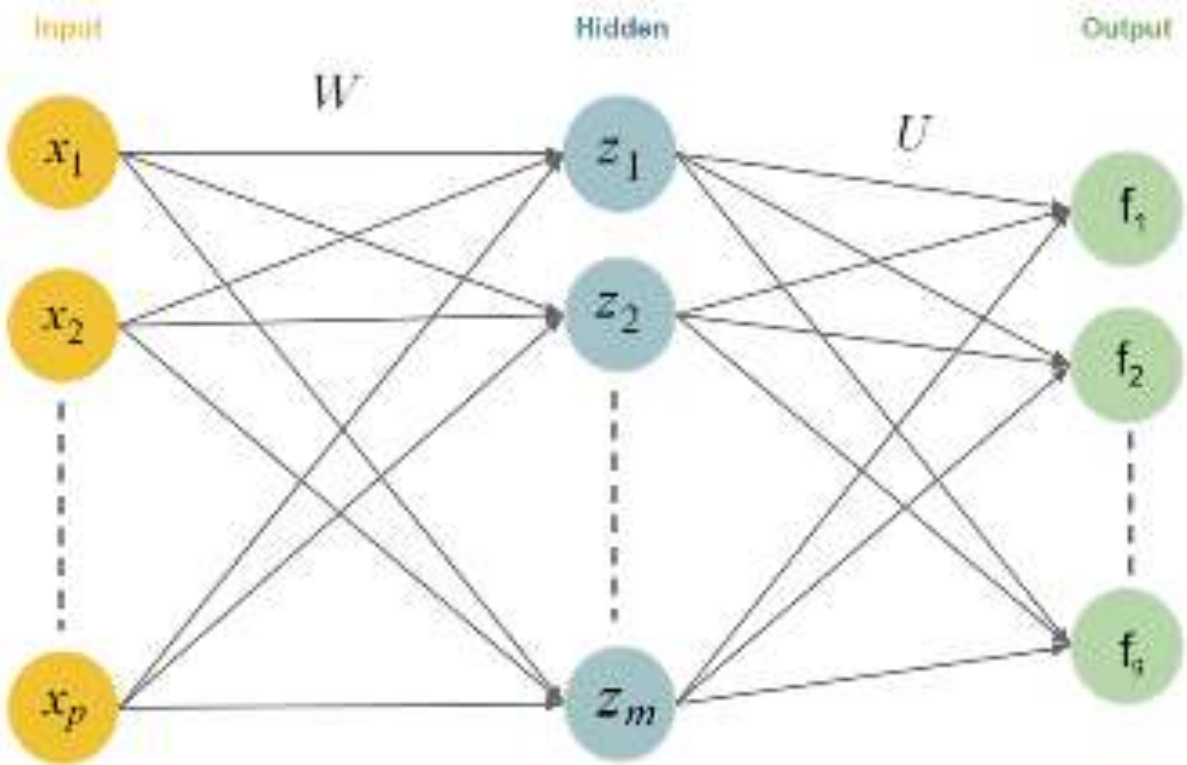


Figure: Structure of a single hidden layer, feed-forward neural network for g-class classification problems (bias term omitted).

MULTI-CLASS CLASSIFICATION

Notation:

- For g-class classification, g output units:

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

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

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

- Compute linear combinations of derived features z:

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

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:

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

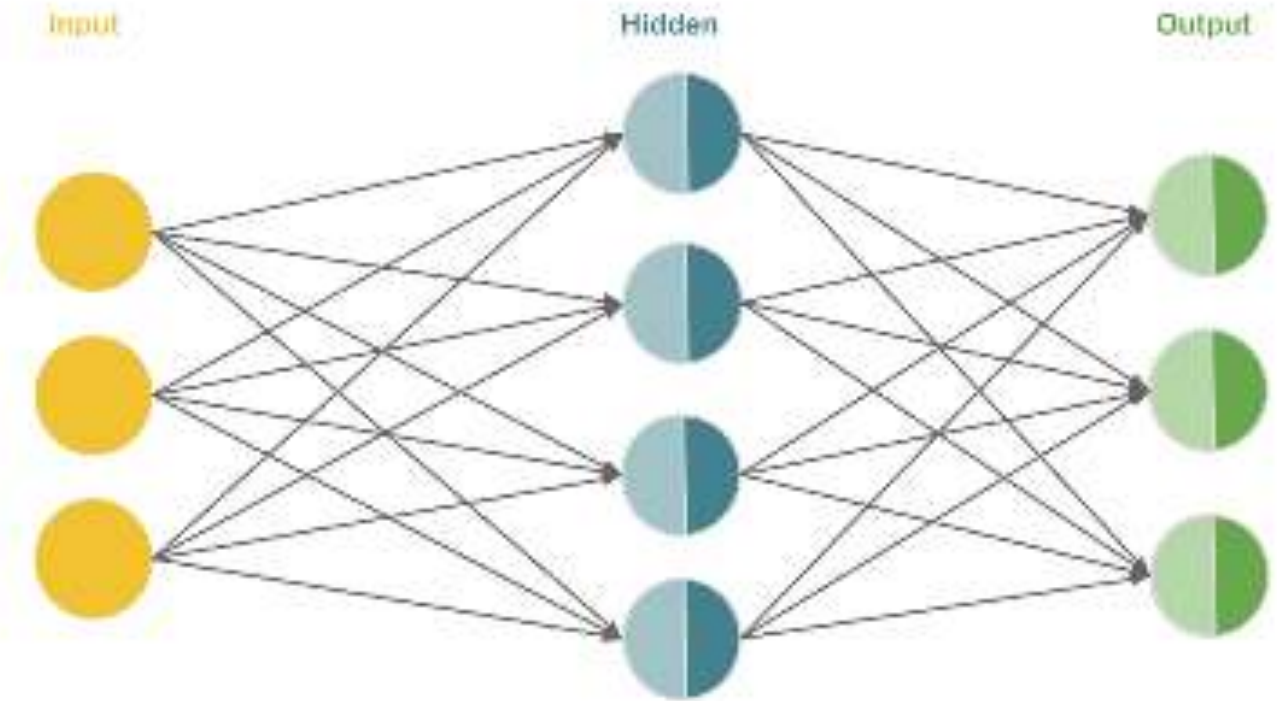
- This is the same transformation used in softmax regression!

- 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).



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

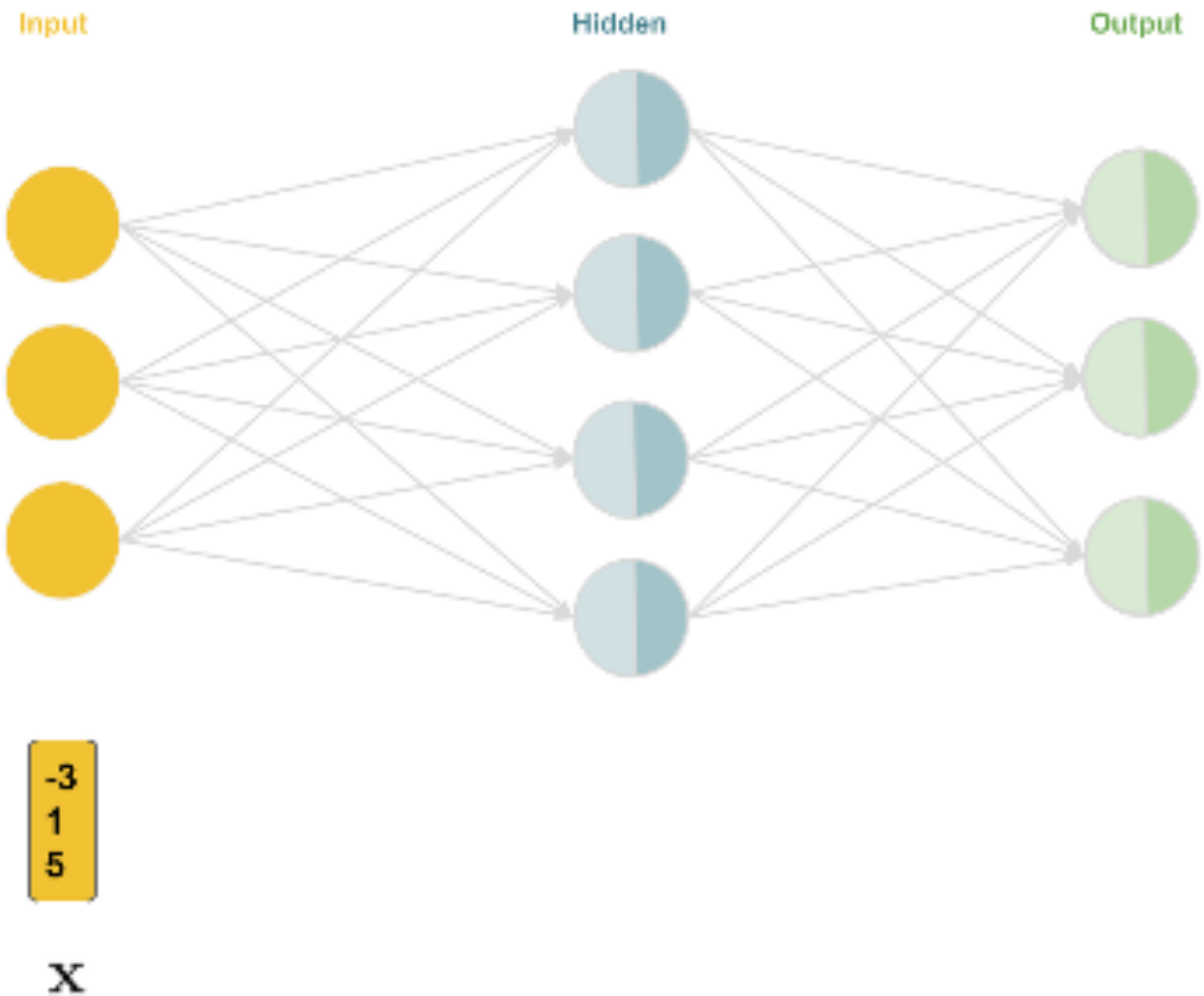
$W^T \quad \mathbf{b}$

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

$U^T \quad \mathbf{c}$

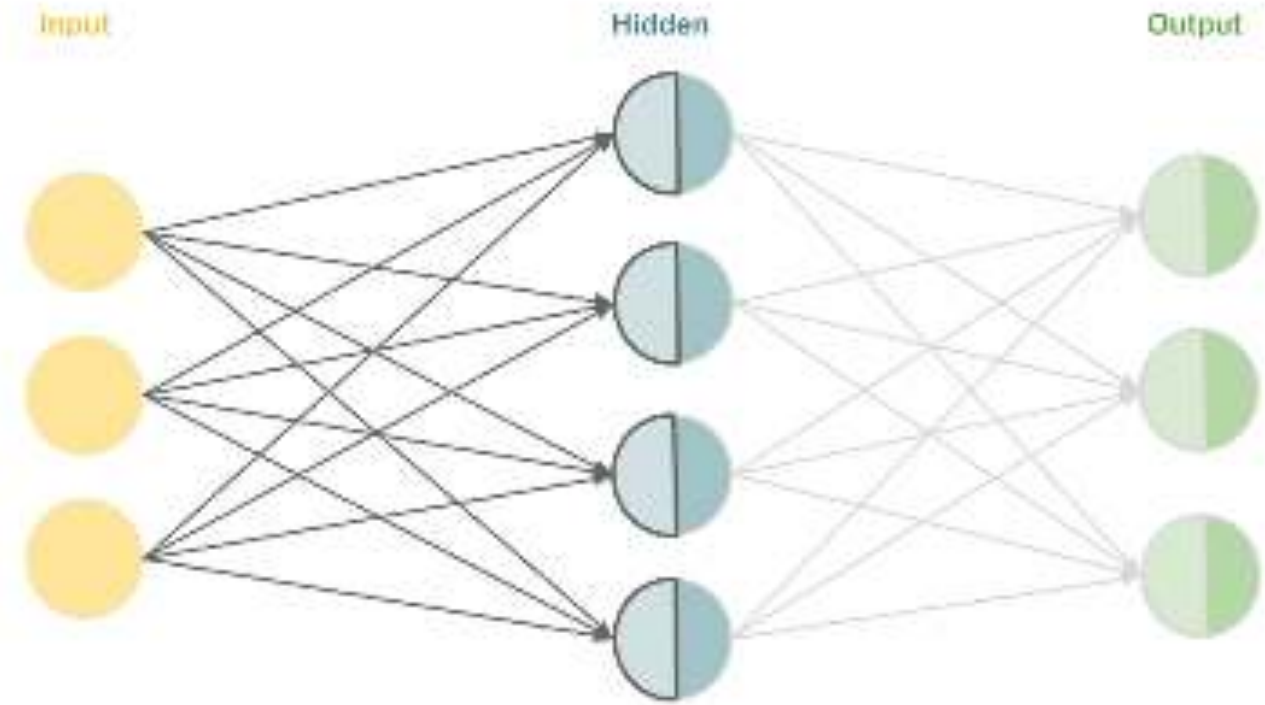
MULTI-CLASS CLASSIFICATION: EXAMPLE

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



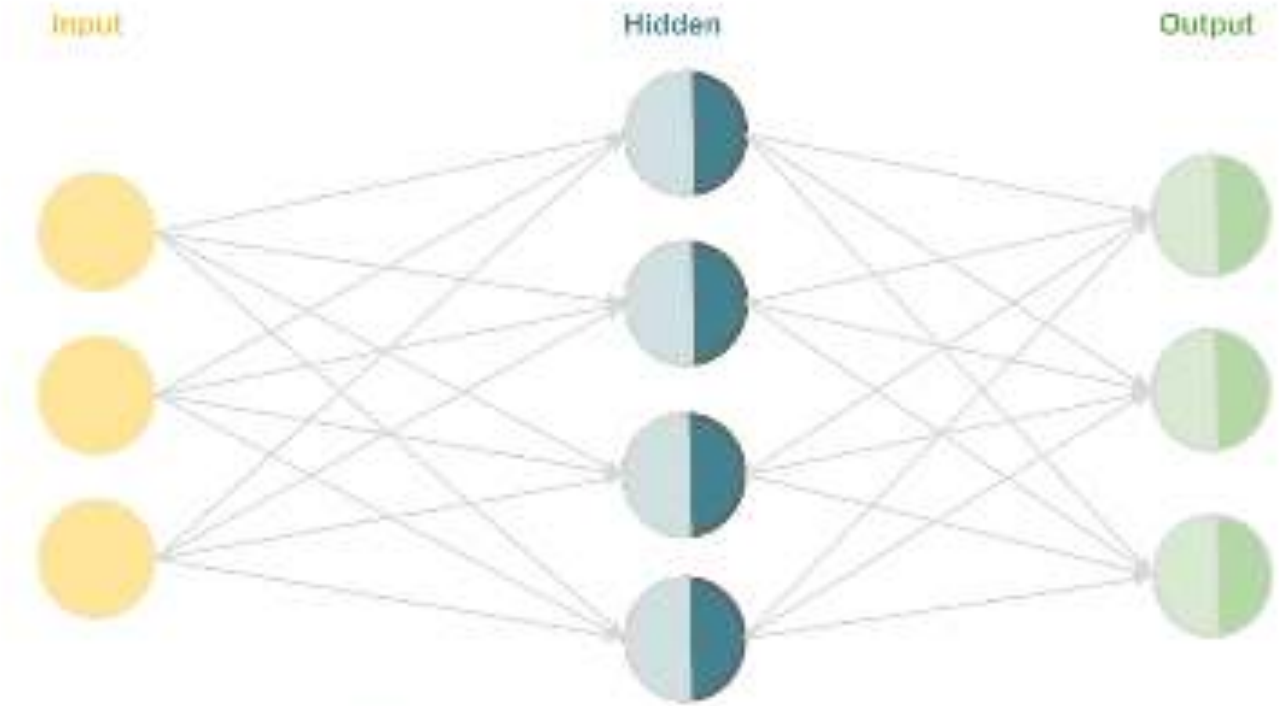
MULTI-CLASS CLASSIFICATION: EXAMPLE

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



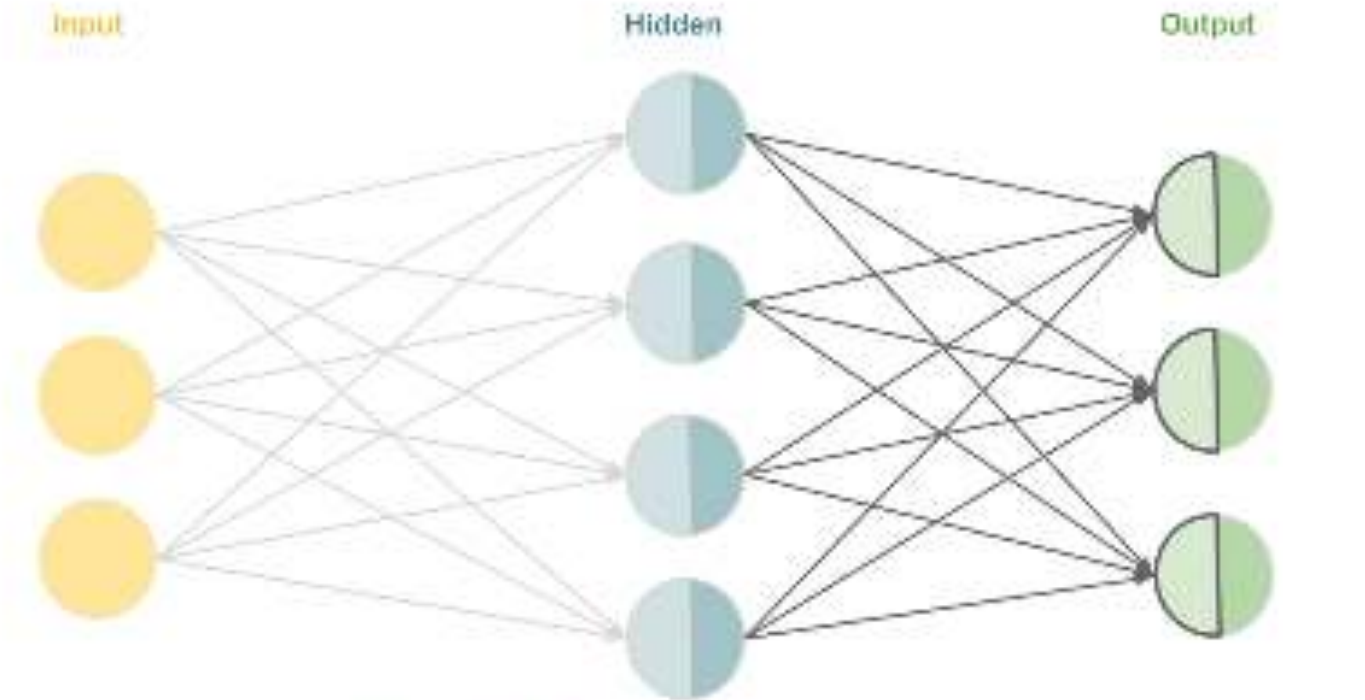
MULTI-CLASS CLASSIFICATION: EXAMPLE

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



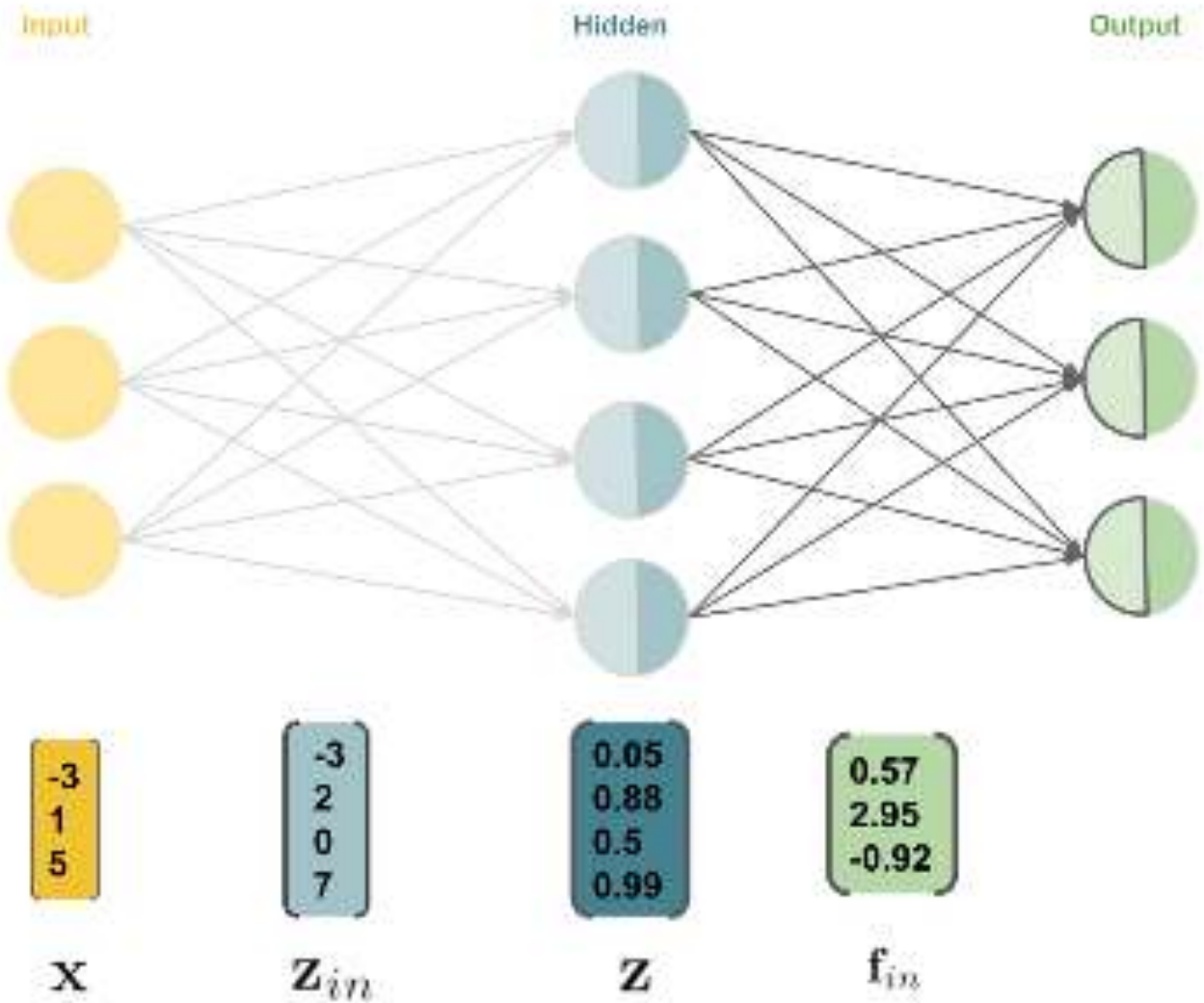
MULTI-CLASS CLASSIFICATION: EXAMPLE

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



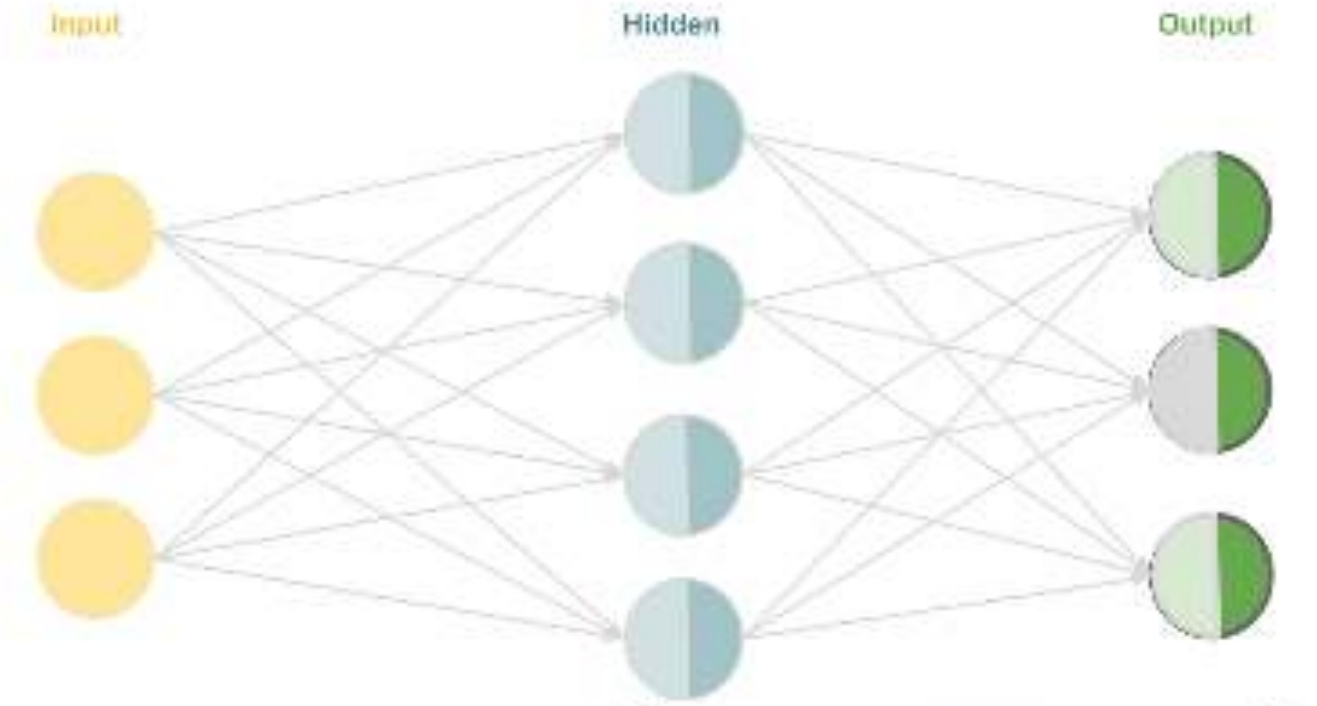
MULTI-CLASS CLASSIFICATION: EXAMPLE

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



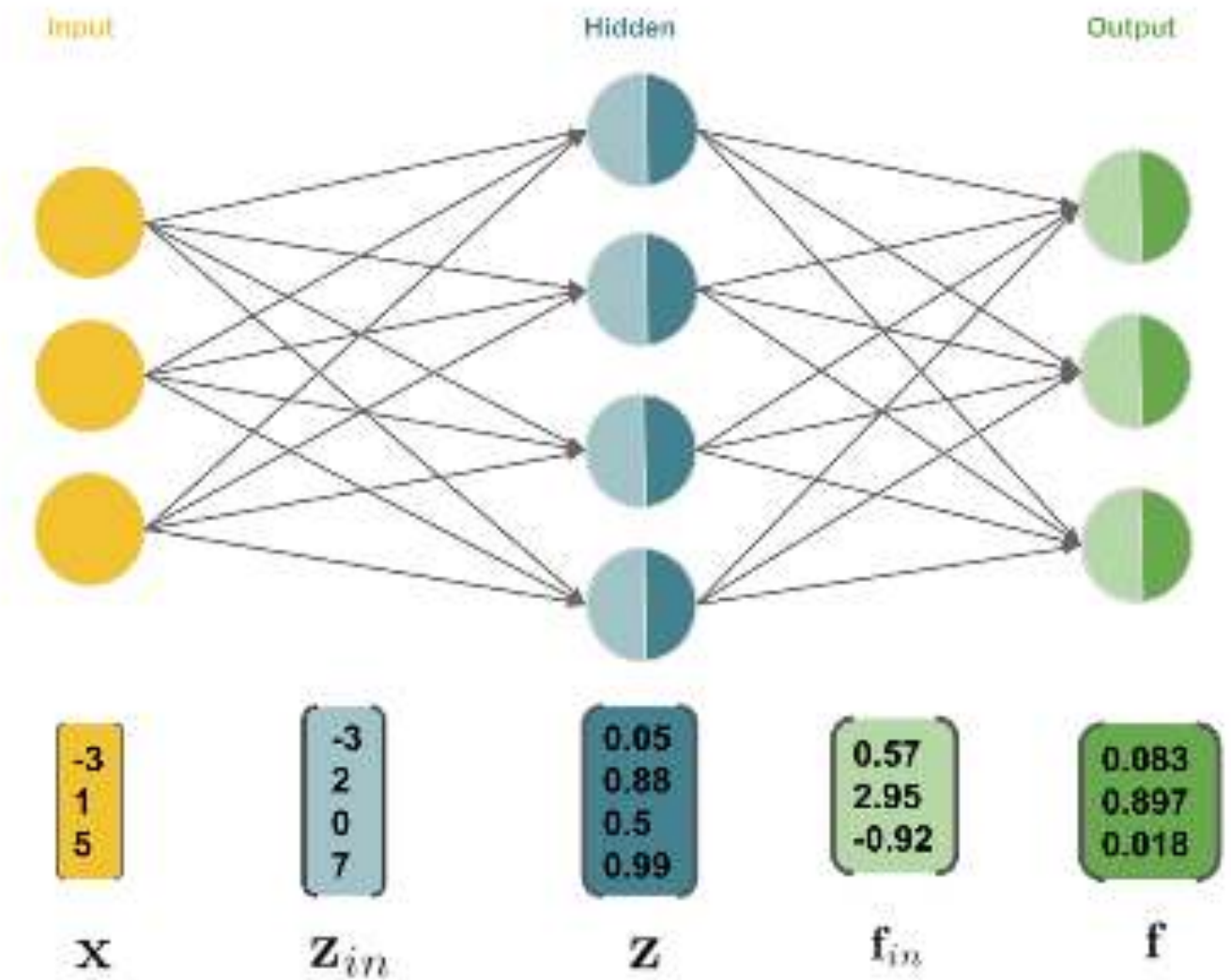
MULTI-CLASS CLASSIFICATION: EXAMPLE

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

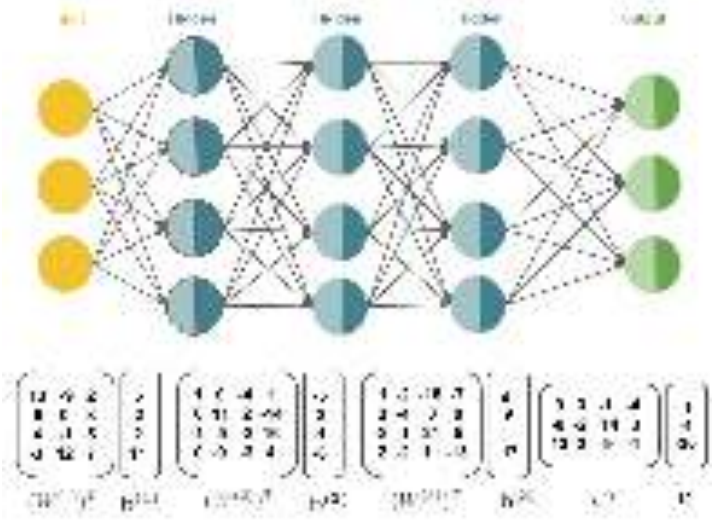
$$L(y, f(\mathbf{x})) = -\sum_{k=1}^{g} [y = k] \log \left(\frac{\exp(f_{in,k})}{\sum_{k'=1}^{g} \exp(f_{in,k'})} \right)$$

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 $\boldsymbol{y}$ 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 l 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:

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

  where $\sigma^{(i)}$ and $\phi^{(i)}$ are the activation function and the weighted sum of hidden layer i, respectively. τ and ϕ 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 $i \in \{1 \ldots l\}$.
  - $\mathbf{z}^{(i)} = \sigma^{(i)}(\phi^{(i)}) = \sigma^{(i)}(\mathbf{W}^{(i)T}\mathbf{z}^{(i-1)} + \mathbf{b}^{(i)})$, 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

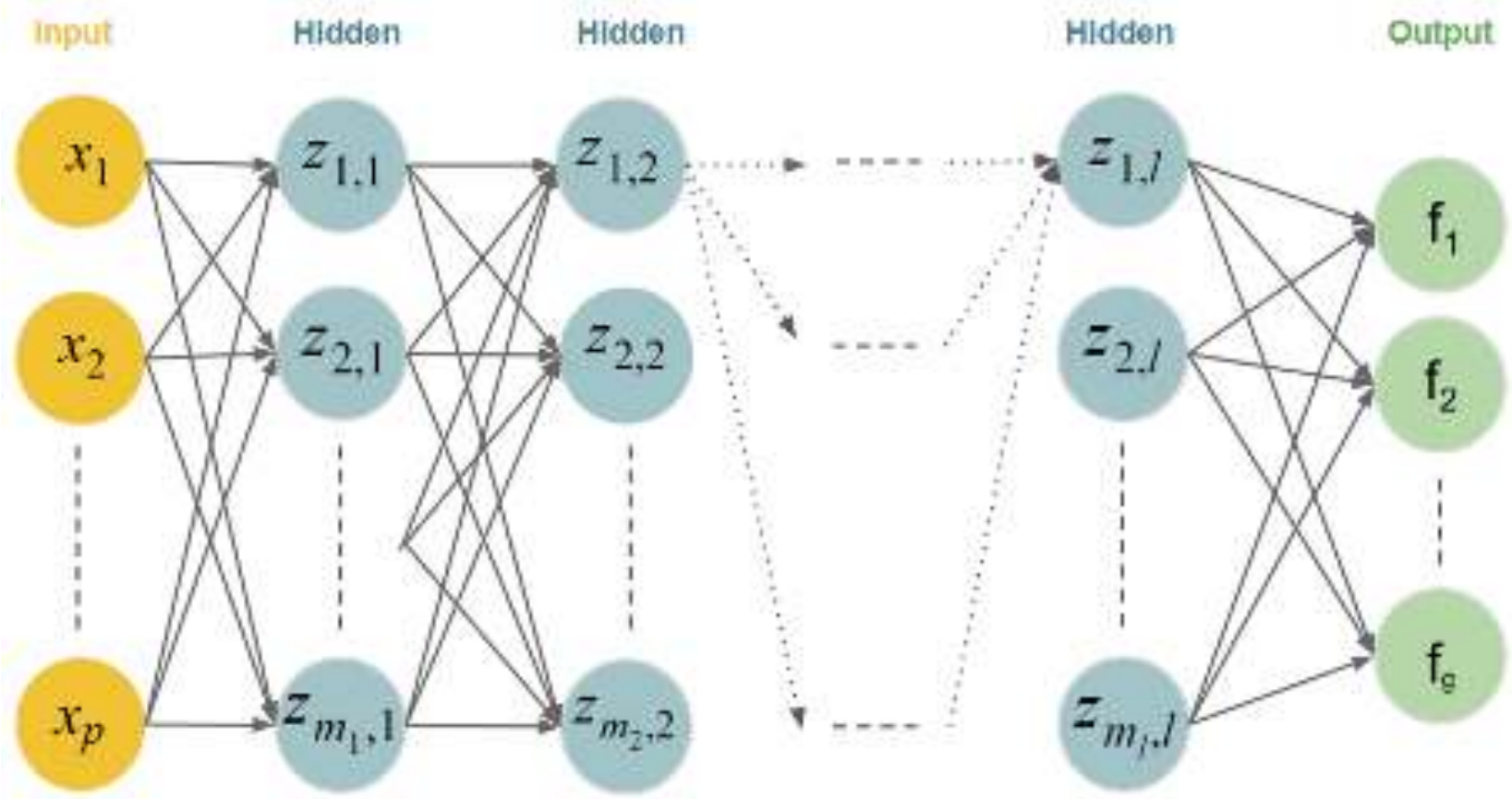
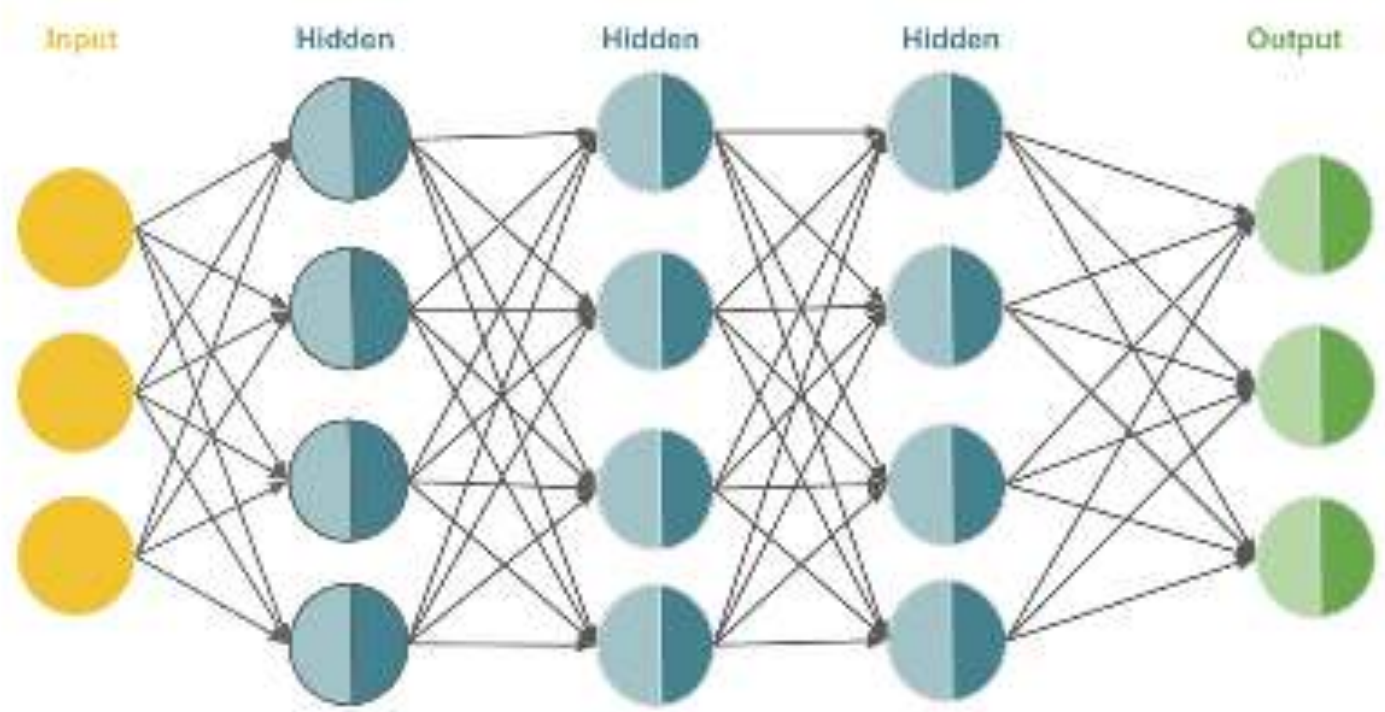


Figure: Structure of a deep neural network with *l* hidden layers (bias terms omitted).

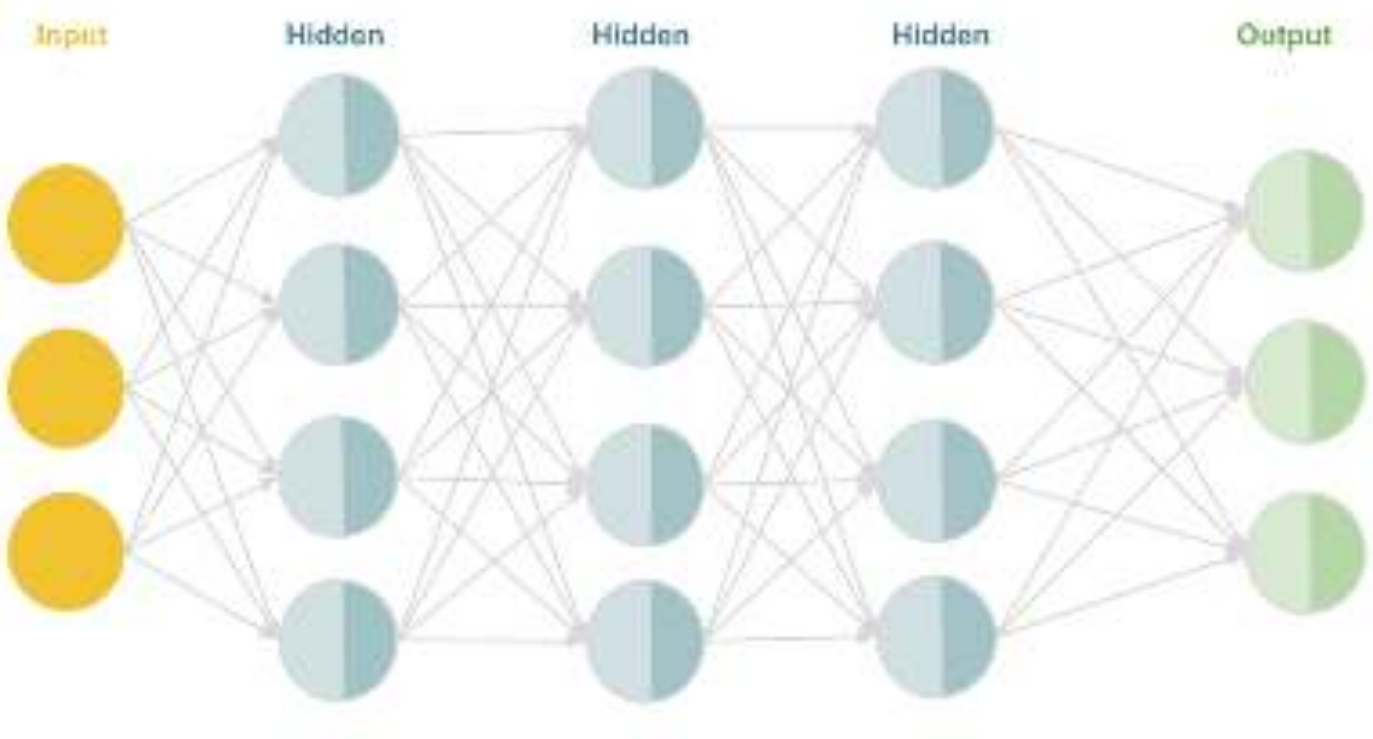
FEEDFORWARD NEURAL NETWORKS: EXAMPLE



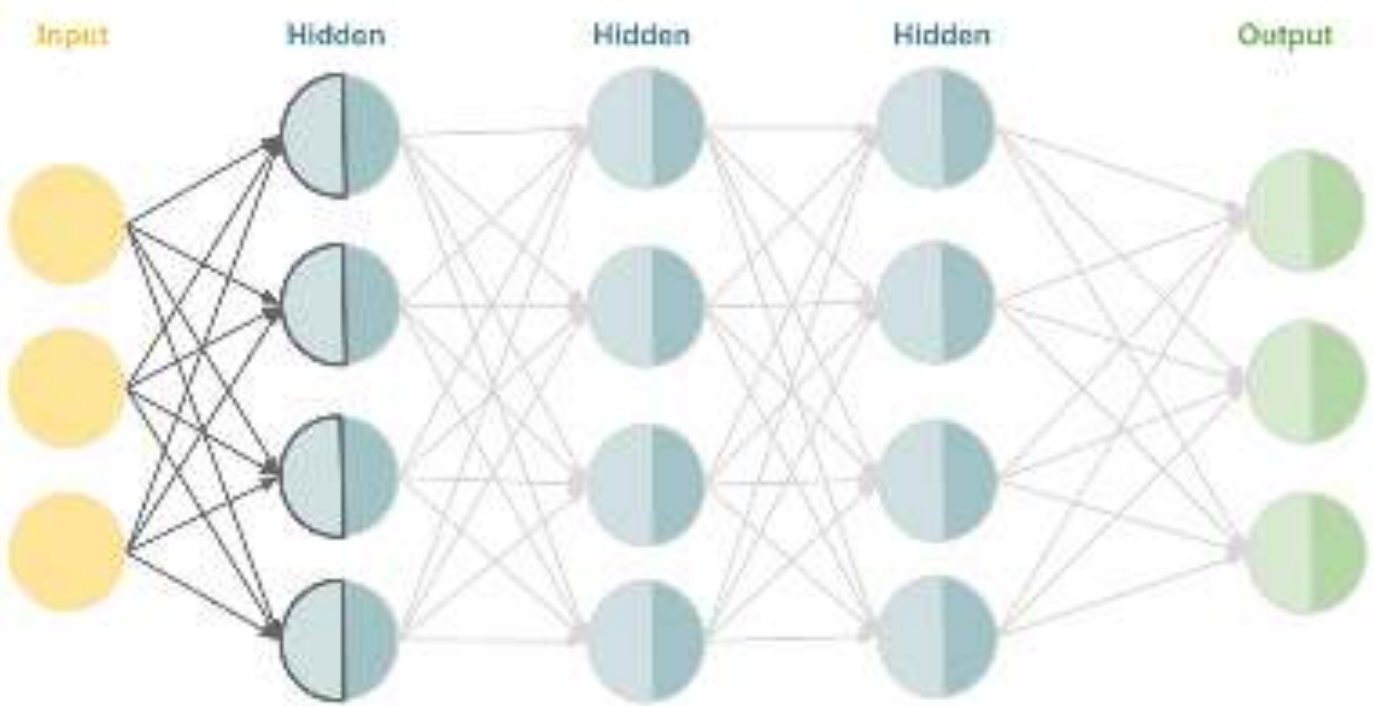
$$\begin{pmatrix} 13 & -9 & 2 \\ -8 & 0 & 3 \\ 4 & -1 & 5 \\ -3 & 12 & 7 \end{pmatrix} \begin{pmatrix} 5 \\ -2 \\ 2 \\ 11 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 & -4 & 1 \\ 0 & 11 & 2 & -14 \\ -1 & 5 & -2 & 16 \\ 0 & -9 & -3 & 4 \end{pmatrix} \begin{pmatrix} -5 \\ 3 \\ 1 \\ -8 \end{pmatrix} \quad \begin{pmatrix} 1 & -2 & -18 & -7 \\ 3 & -4 & 8 & 0 \\ -2 & 1 & 21 & 5 \\ 2 & -2 & 11 & -13 \end{pmatrix} \begin{pmatrix} 4 \\ -6 \\ 1 \\ -17 \end{pmatrix} \quad \begin{pmatrix} 9 & 3 & -1 & -4 \\ -8 & -2 & 14 & 3 \\ 13 & 2 & -9 & -1 \end{pmatrix} \begin{pmatrix} -1 \\ -4 \\ -30 \end{pmatrix}$$

$$(W^{(1)})^T \quad \mathbf{b}^{(1)} \quad (W^{(2)})^T \quad \mathbf{b}^{(2)} \quad (W^{(3)})^T \quad \mathbf{b}^{(3)} \quad U^T \quad \mathbf{c}$$

FEEDFORWARD NEURAL NETWORKS: EXAMPLE



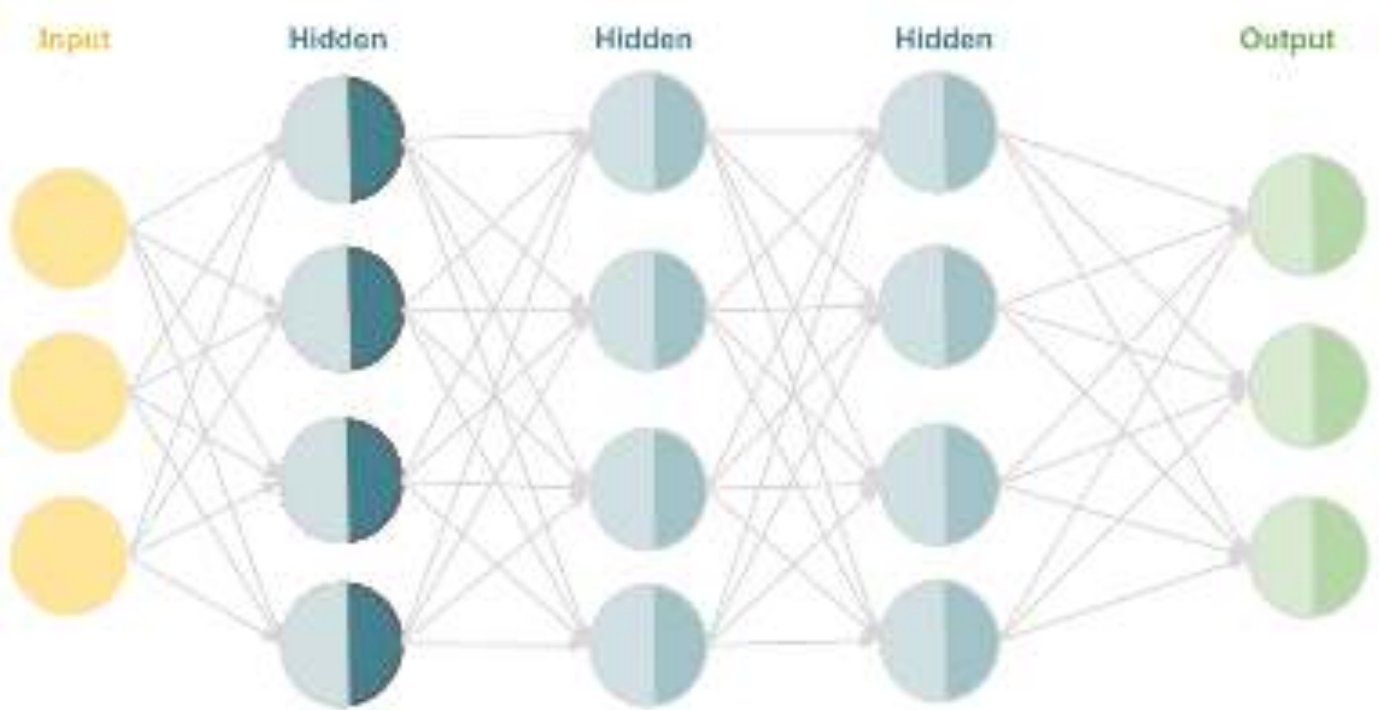
FEEDFORWARD NEURAL NETWORKS: EXAMPLE



$$\begin{matrix} 7 \\ 1 \\ -4 \end{matrix} \begin{pmatrix} 7*13 + 1*(-9) + (-4)*2 + 5 \\ 7*(-8) + 1*0 + (-4)*3 + (-2) \\ 7*4 + 1*(-1) + (-4)*5 + 2 \\ 7*(-3) + 1*12 + (-4)*7 + 11 \end{pmatrix}$$

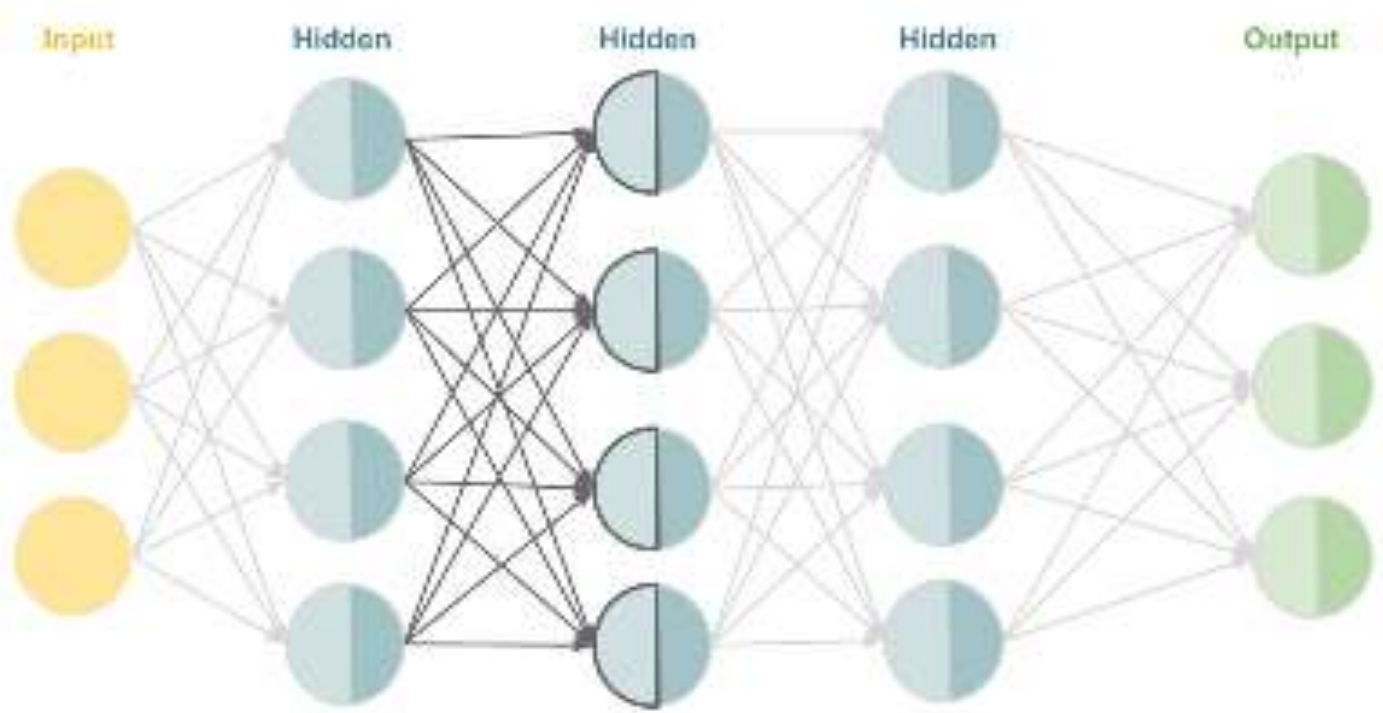
$\mathbf{x}$ $\quad \mathbf{z}_{in}^{(1)} = W^{(1)T}\mathbf{x} + \mathbf{b}^{(1)}$

FEEDFORWARD NEURAL NETWORKS: EXAMPLE



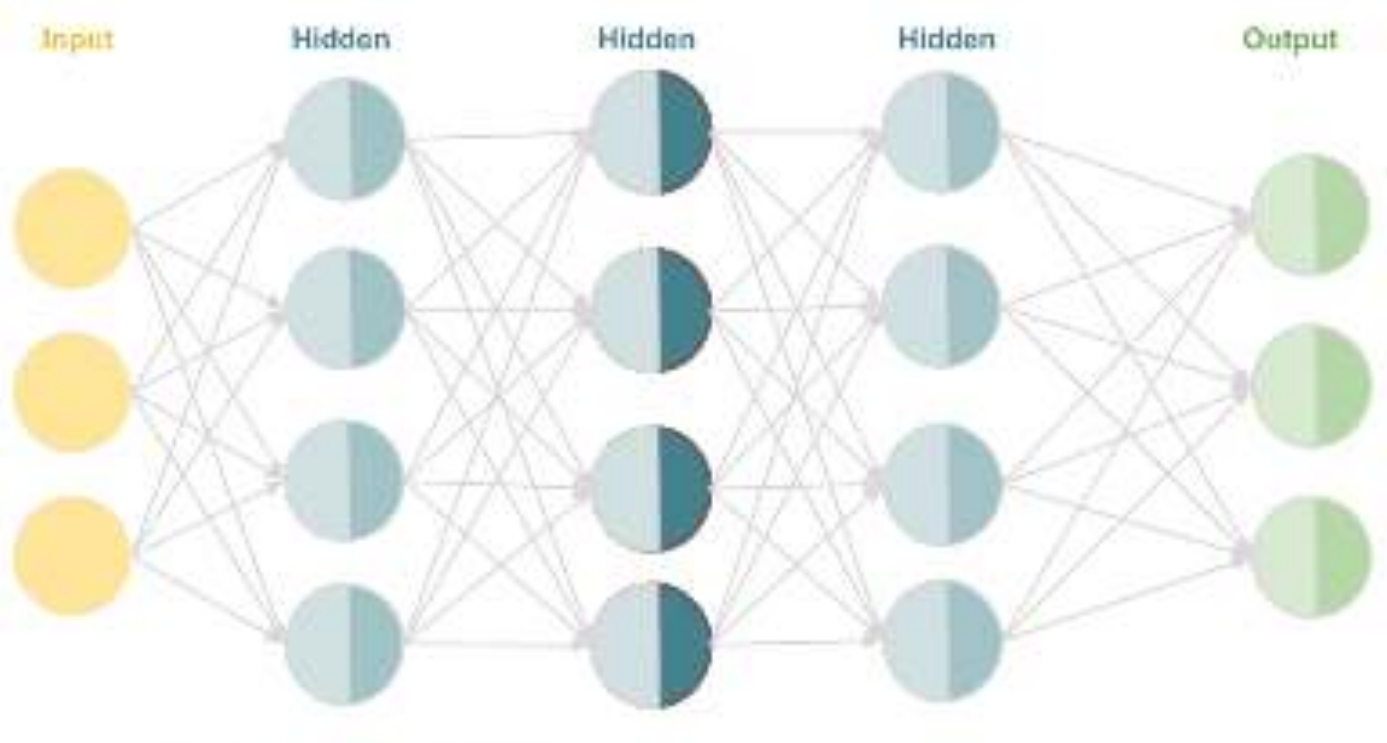
$\mathbf{x}$ $\mathbf{z}_{in}^{(1)}$ $\mathbf{z}^{(1)} = \mathbf{z}_{out}^{(1)} = \sigma(\mathbf{z}_{in}^{(1)})$

FEEDFORWARD NEURAL NETWORKS: EXAMPLE

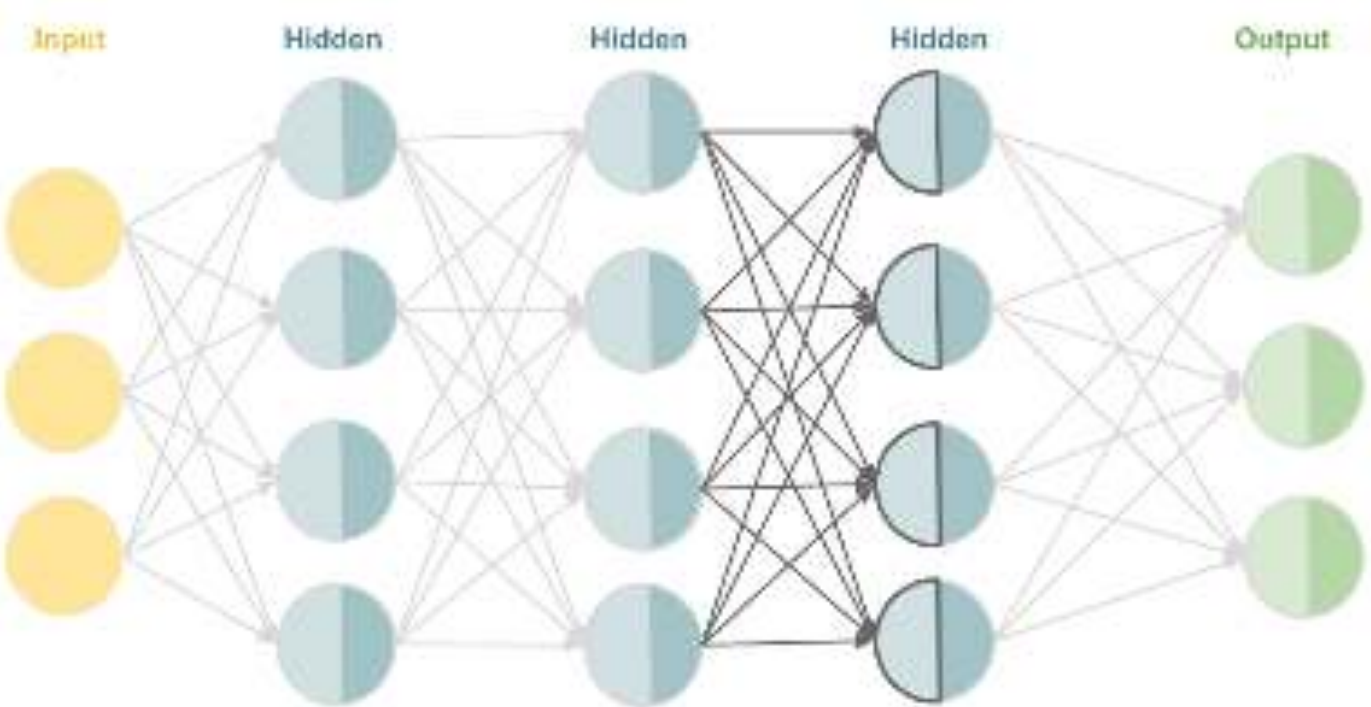


$\mathbf{x}$ $\mathbf{z}_{in}^{(1)}$ $\mathbf{z}^{(1)}$ $\mathbf{z}_{in}^{(2)} = W^{(2)T}\mathbf{z}^{(1)} + \mathbf{b}^{(2)}$

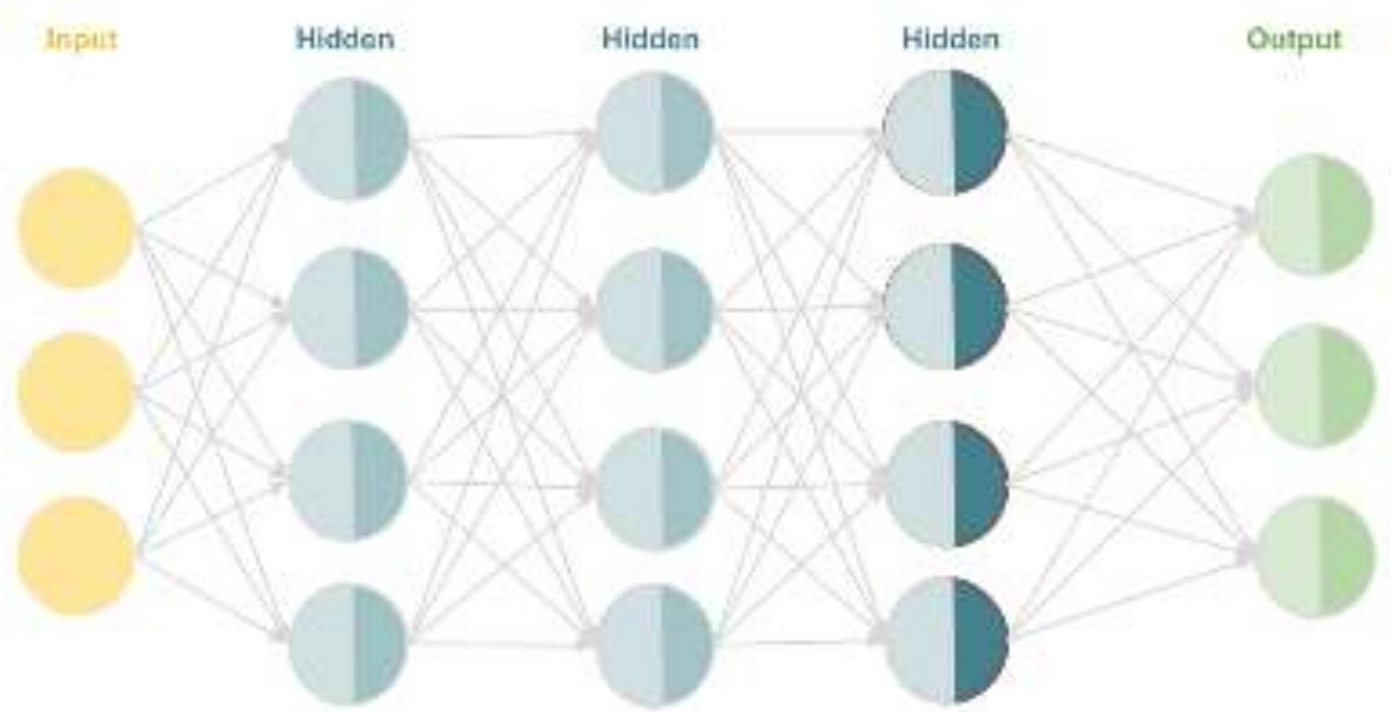
FEEDFORWARD NEURAL NETWORKS: EXAMPLE



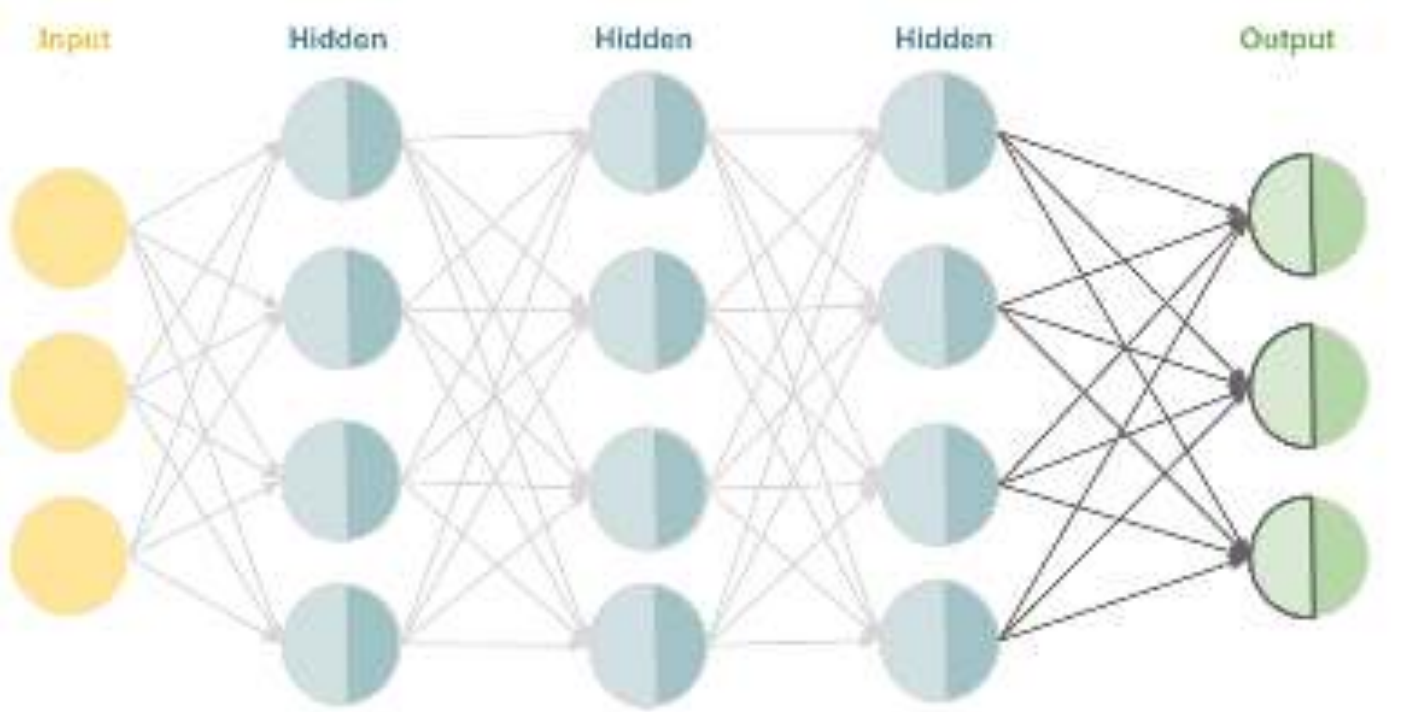
FEEDFORWARD NEURAL NETWORKS: EXAMPLE



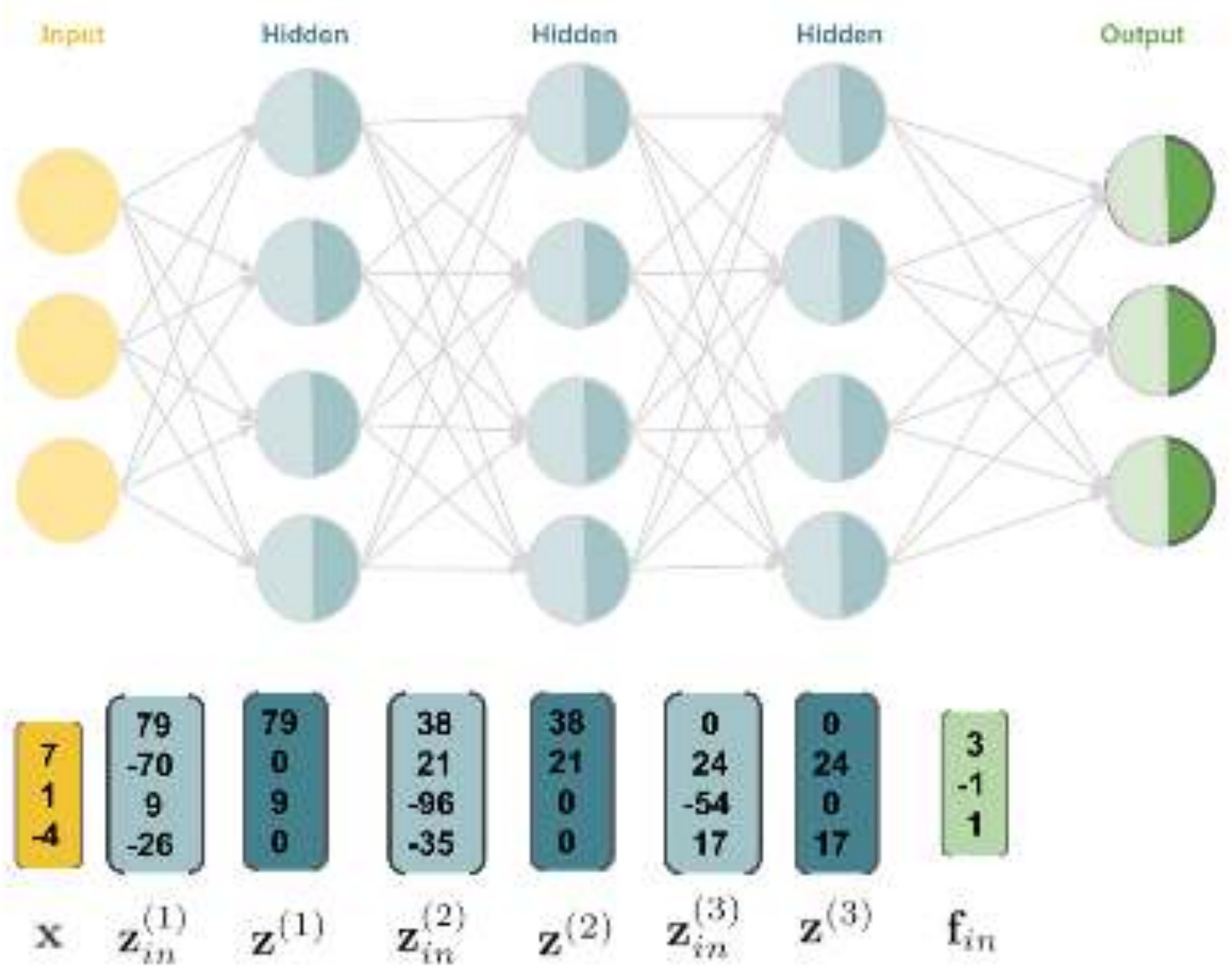
FEEDFORWARD NEURAL NETWORKS: EXAMPLE



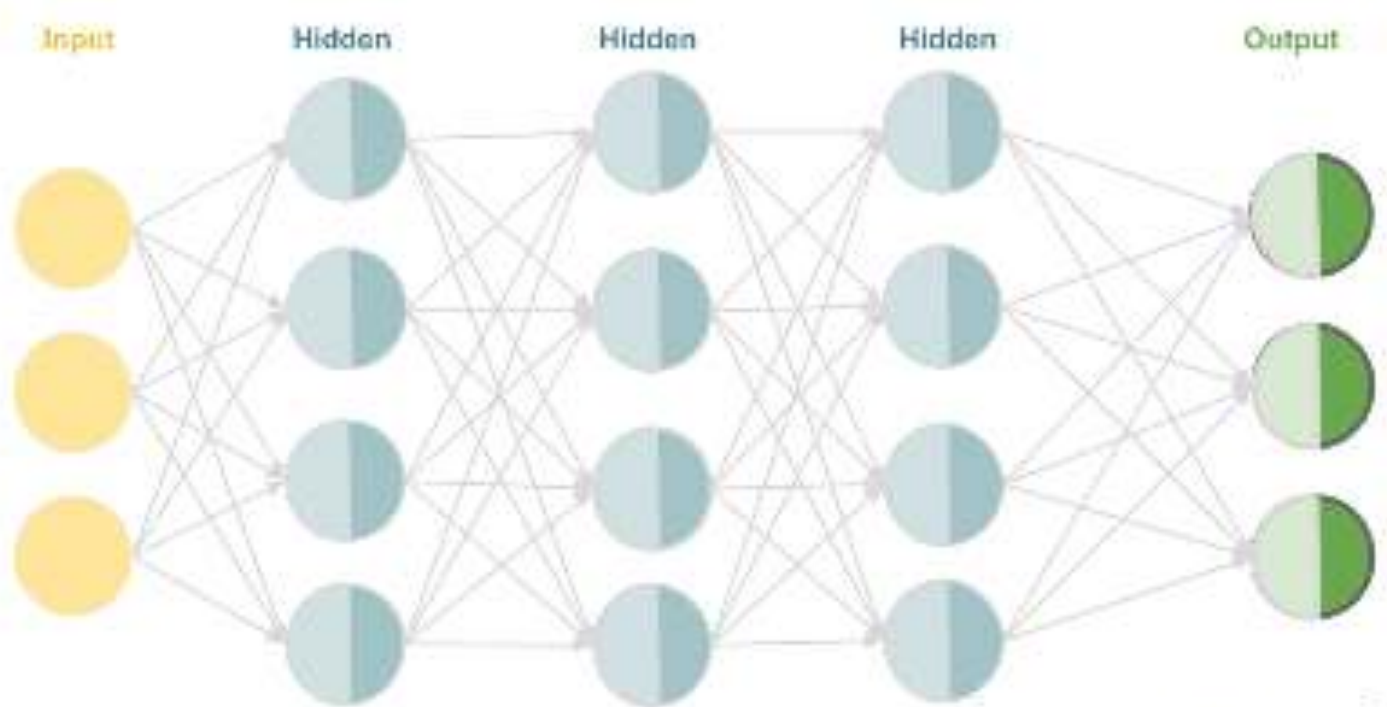
FEEDFORWARD NEURAL NETWORKS: EXAMPLE



FEEDFORWARD NEURAL NETWORKS: EXAMPLE



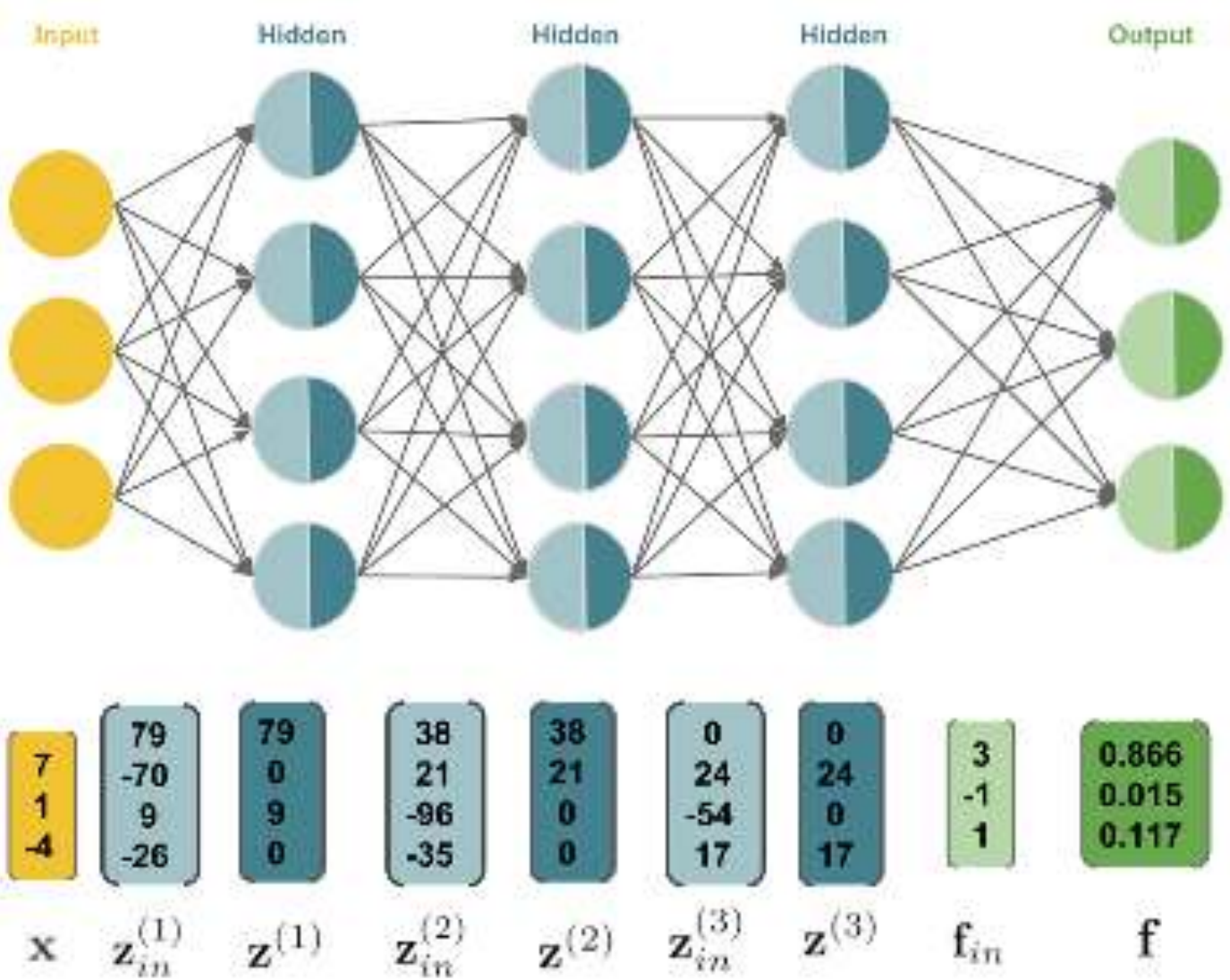
FEEDFORWARD NEURAL NETWORKS: EXAMPLE



$$\mathbf{f}_{in} = \begin{matrix} 3 \\ -1 \\ 1 \end{matrix}$$

$$\mathbf{f} = \mathbf{f}_{out} = \tau(\mathbf{f}_{in}) = \begin{matrix} \frac{\exp(3)}{\exp 3+\exp(-1)+\exp(1)} \\ \frac{\exp(-1)}{\exp 3+\exp(-1)+\exp(1)} \\ \frac{\exp(1)}{\exp 3+\exp(-1)+\exp(1)} \end{matrix}$$

FEEDFORWARD NEURAL NETWORKS: EXAMPLE



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-lnearity to the model.

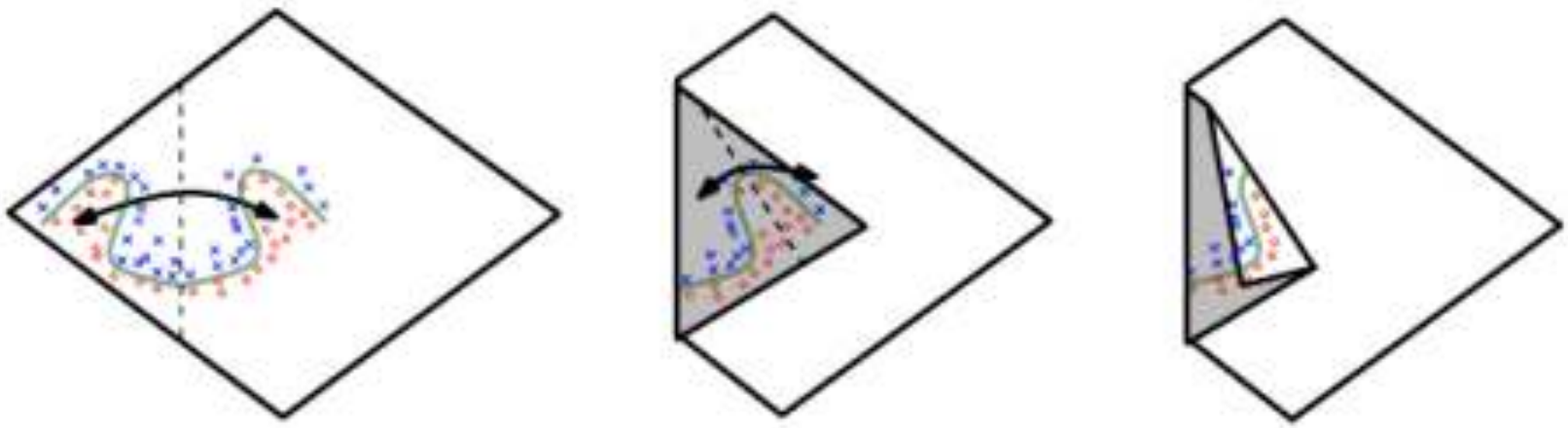


Figure: An intuitive, geometric explanation of the exponential advantage of deeper networks formally (Montúfar et al. (2014)).

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

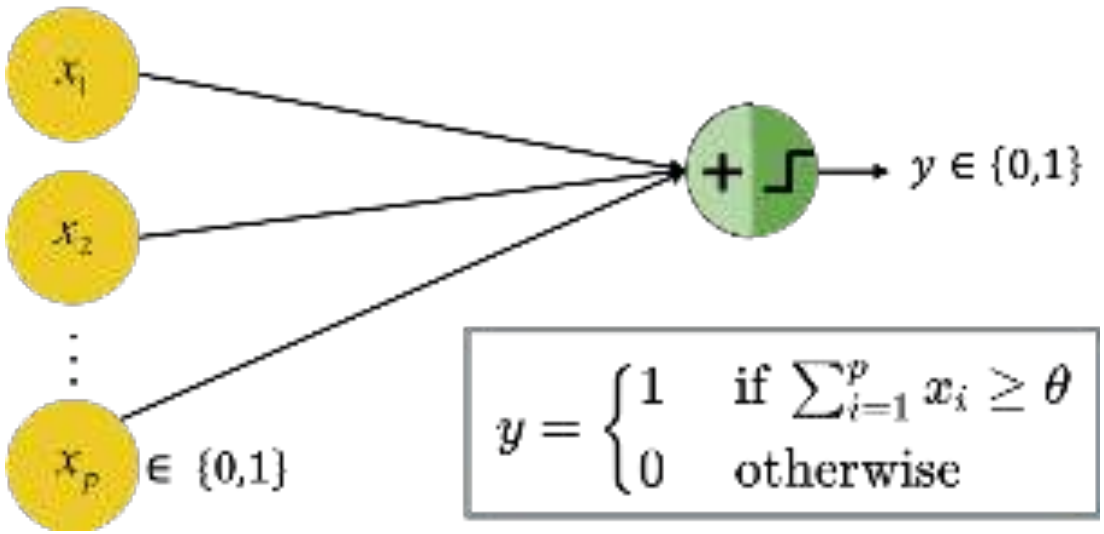


Learning goals

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

A BRIEF HISTORY OF NEURAL NETWORKS

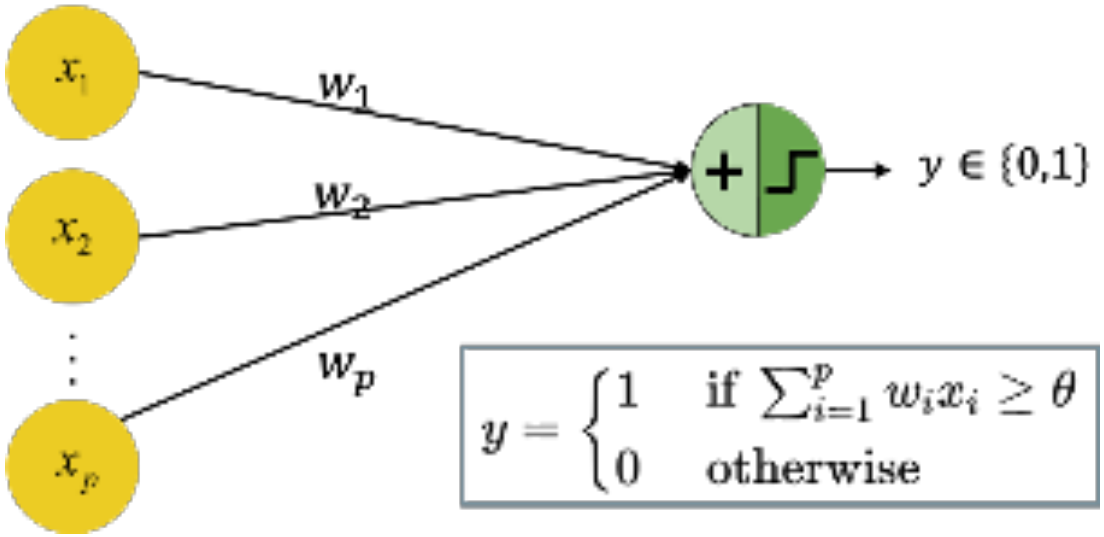
- **1943:** The first artificial neuron, the "Threshold Logic Unit (TLU)", was proposed by Warren McCulloch & Walter Pitts.



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

A BRIEF HISTORY OF NEURAL NETWORKS

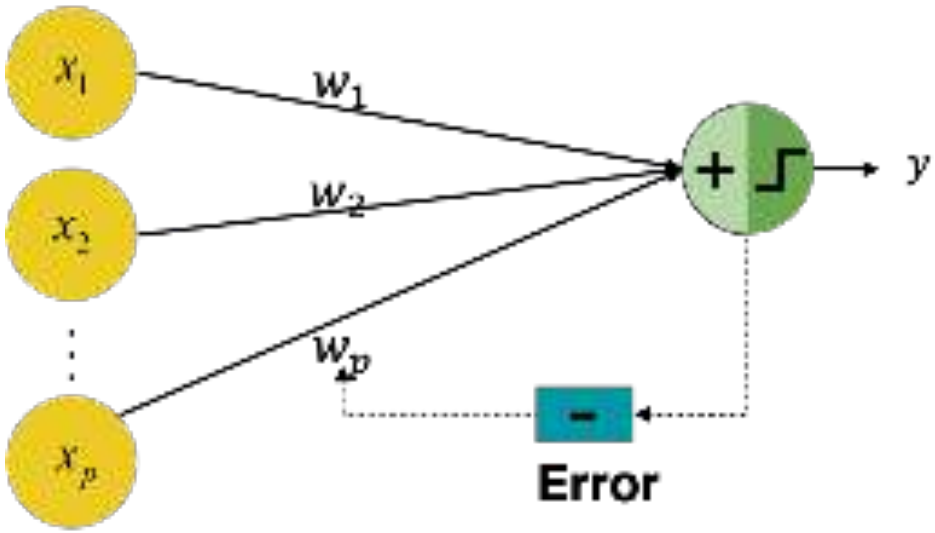
- **1957:** The perceptron was invented by Frank Rosenblatt.



  - 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

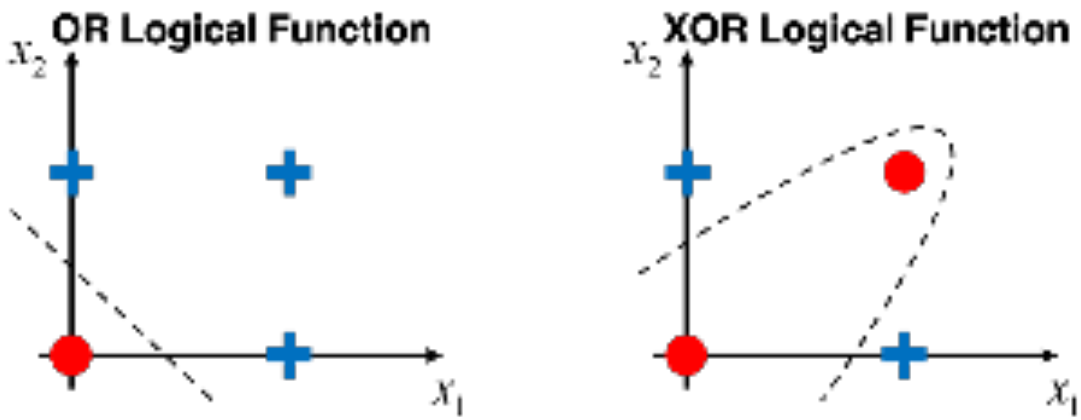
- **1960:** Adaptive Linear Neuron (ADALINE) was invented by Bernard Widrow & Ted Hoff; weights are now adjustable according to the weighted sum of the inputs.



- **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 $\Rightarrow$ 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

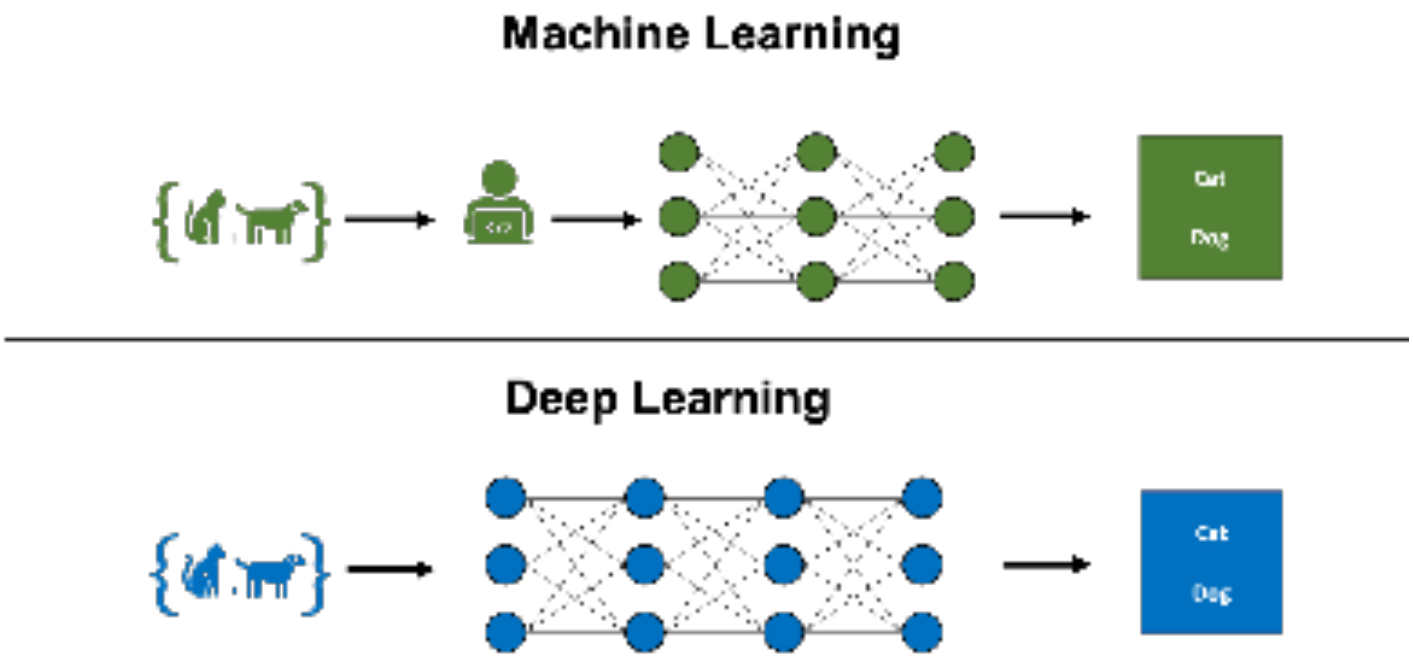
- **1985:** The second “AI Winter” kicked in.
  - Overly optimistic expectations concerning potential of AI/DL.
  - The phrase “AI” even reached a pseudoscience status.
  - Kernel machines and graphical models both achieved good results on many important tasks.
  - Some fundamental mathematical difficulties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



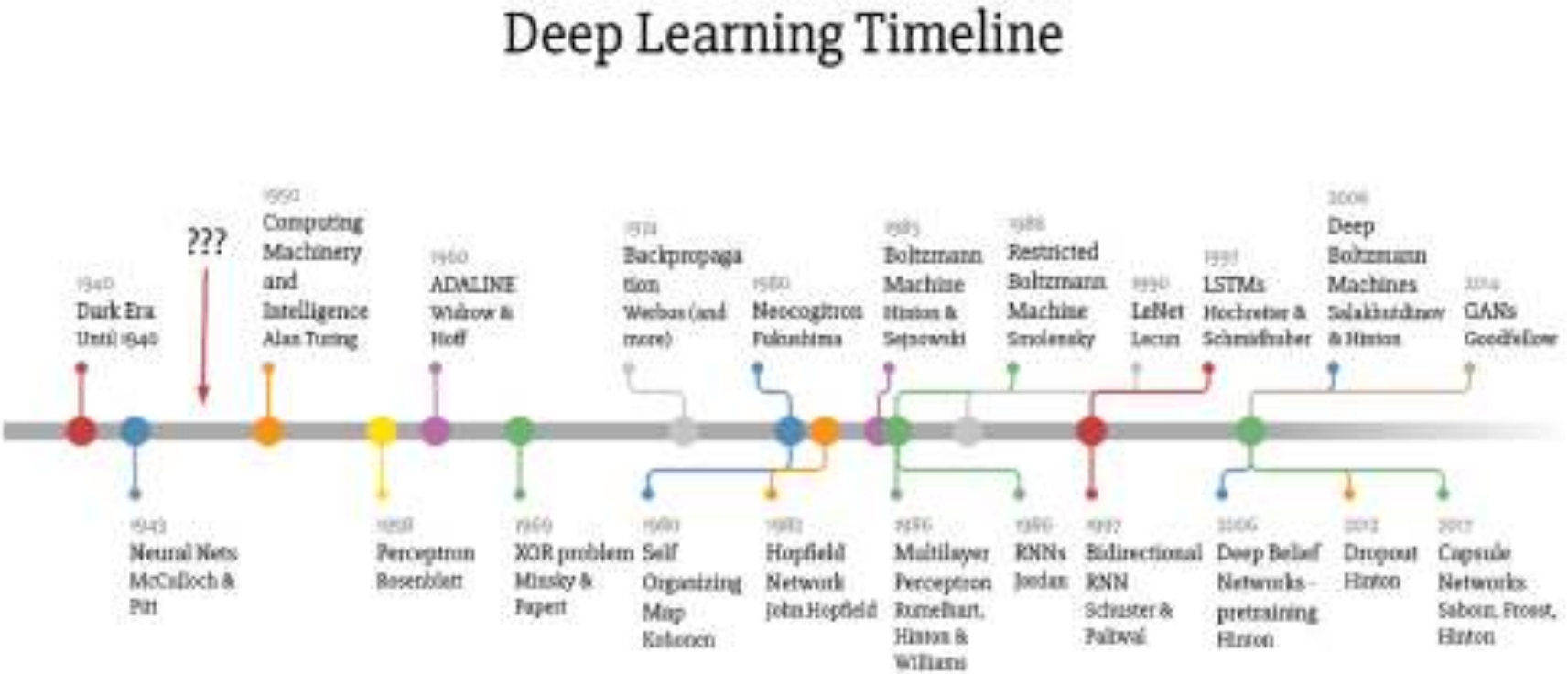
Credit: https://emerj.com/ai-executive-guides/will-there-be-another-artificial-intelligence-winter-probably-not/

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.

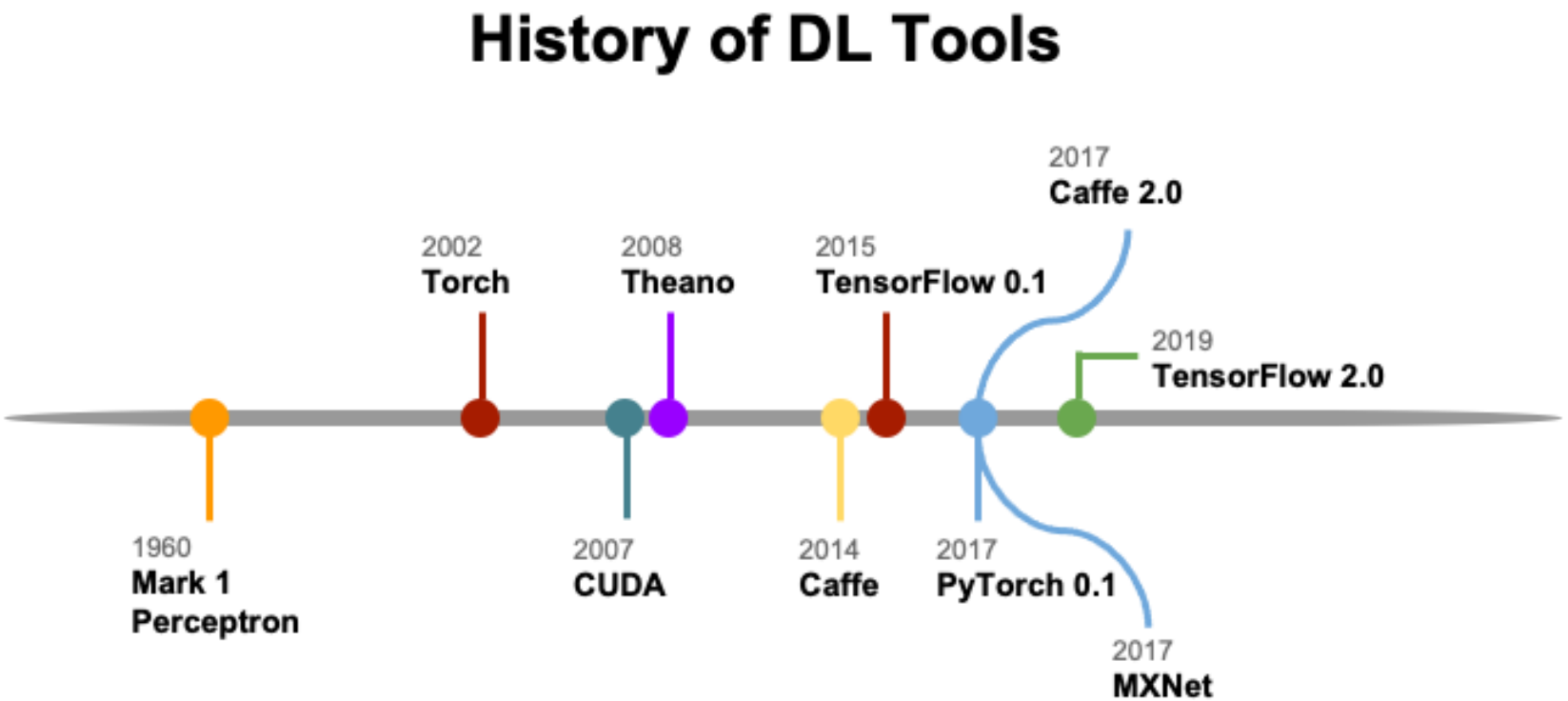


A BRIEF HISTORY OF NEURAL NETWORKS



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

A BRIEF HISTORY OF NEURAL NETWORKS



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.

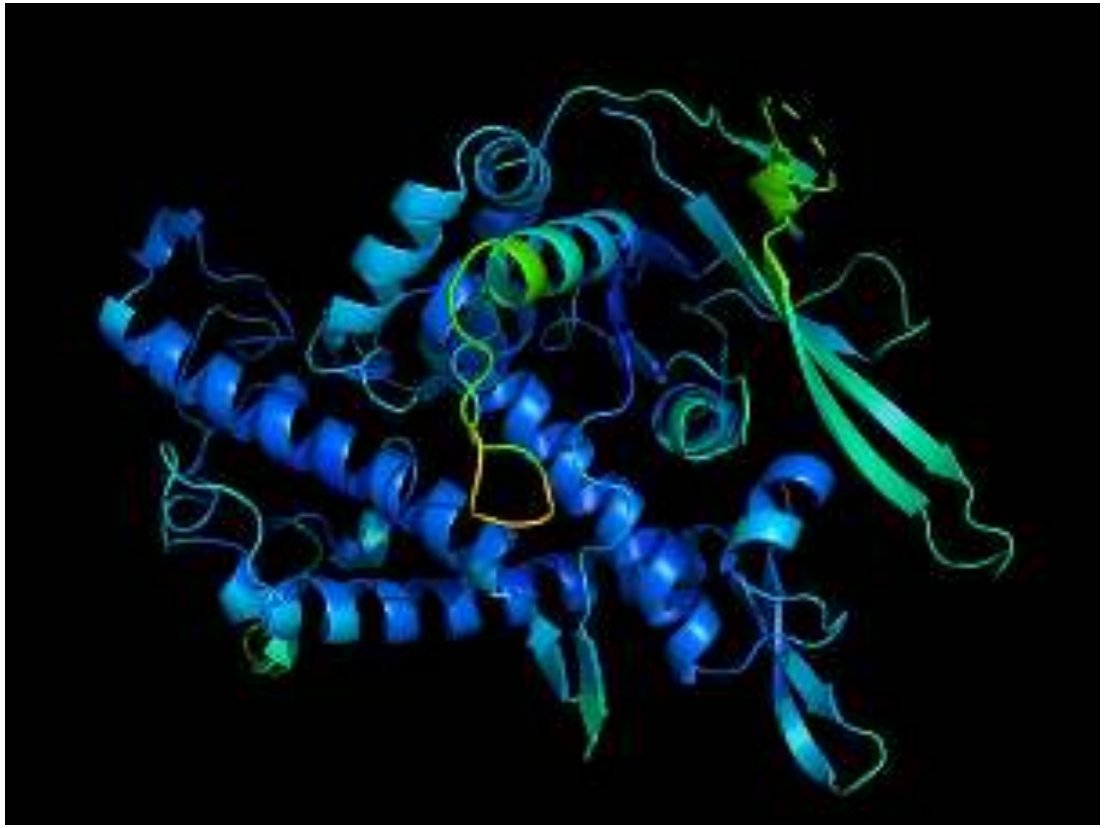
A BRIEF HISTORY OF NEURAL NETWORKS



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.

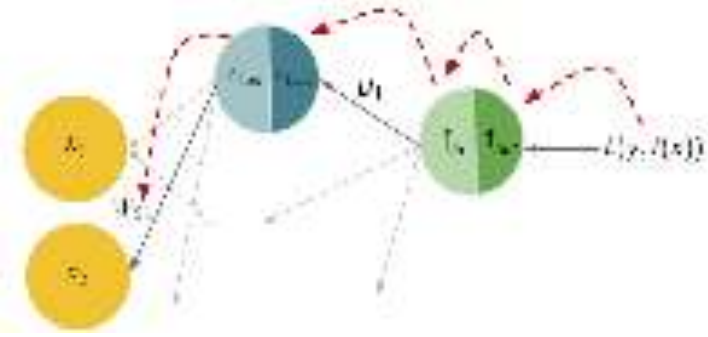
A BRIEF HISTORY OF NEURAL NETWORKS



- **Generative Pre-trained Transformer 3 (GPT-3)** is the third generatation 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:

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

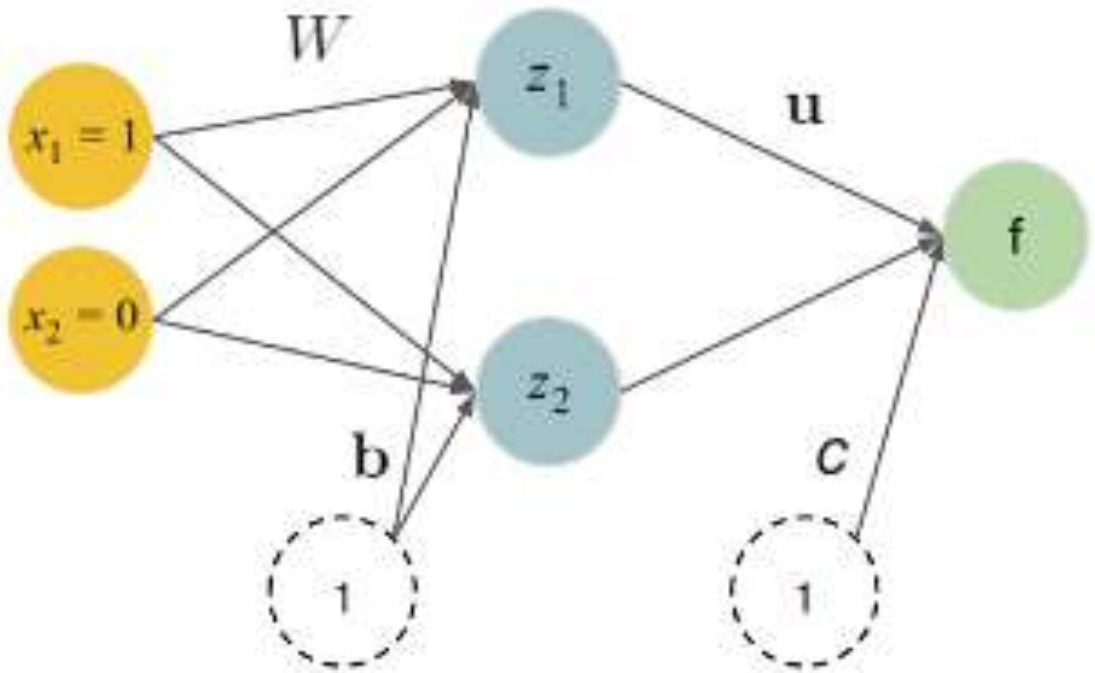
Backprop training of NNs runs in 2 alternating steps, for one $\mathbf{x}$:

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

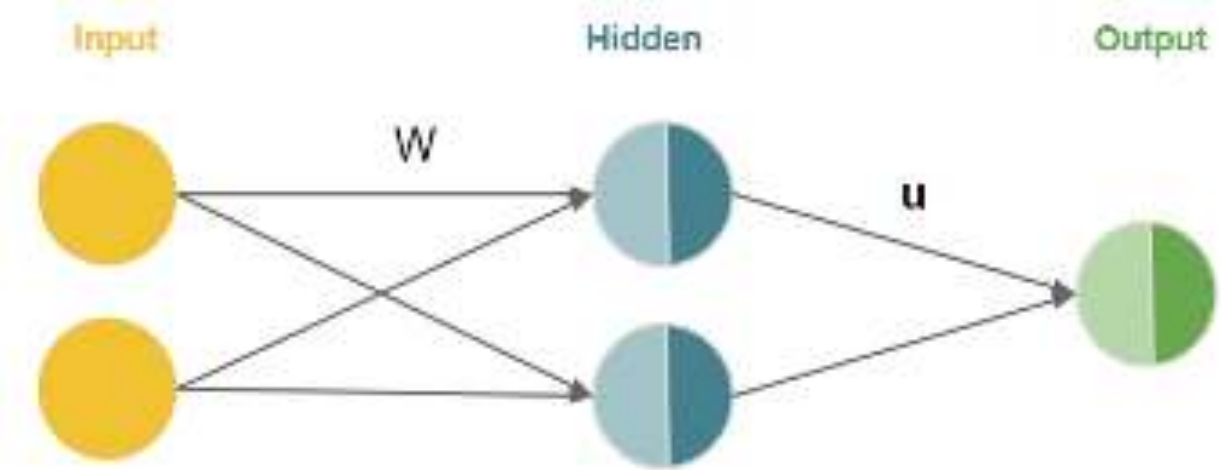
- As activations (hidden and outputs) we use the logistic.
- We run one FP and BP on $\mathbf{x} = (1, 0)^T$ with $y = 1$.
- We use L2 loss between 0-1 labels and the predicted probabilities. This is a bit uncommon, but computations become simpler.



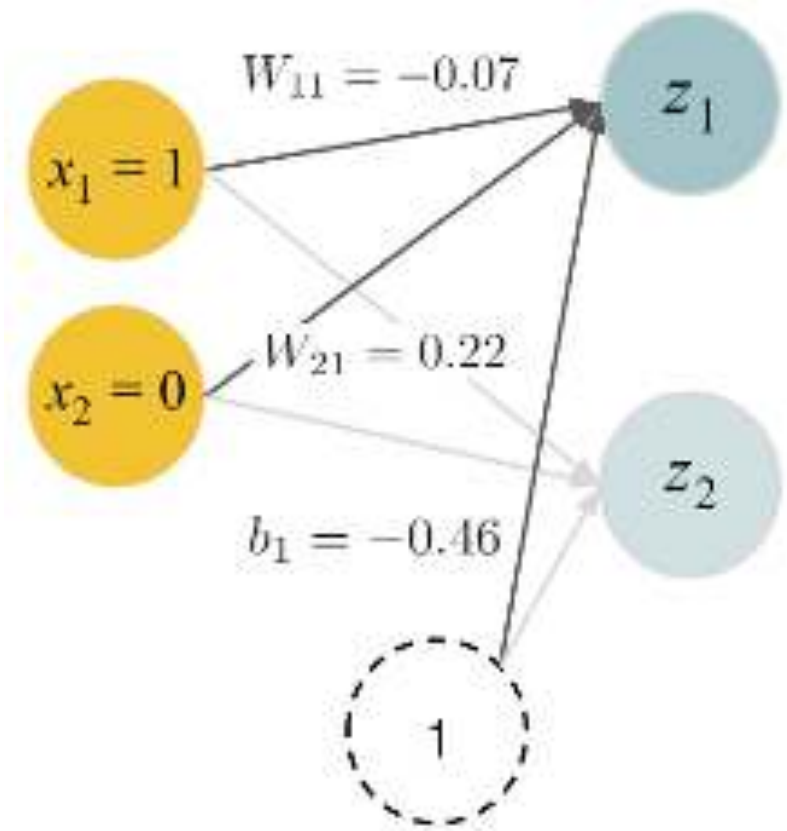
Note: We will only show rounded decimals.

FORWARD PASS

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



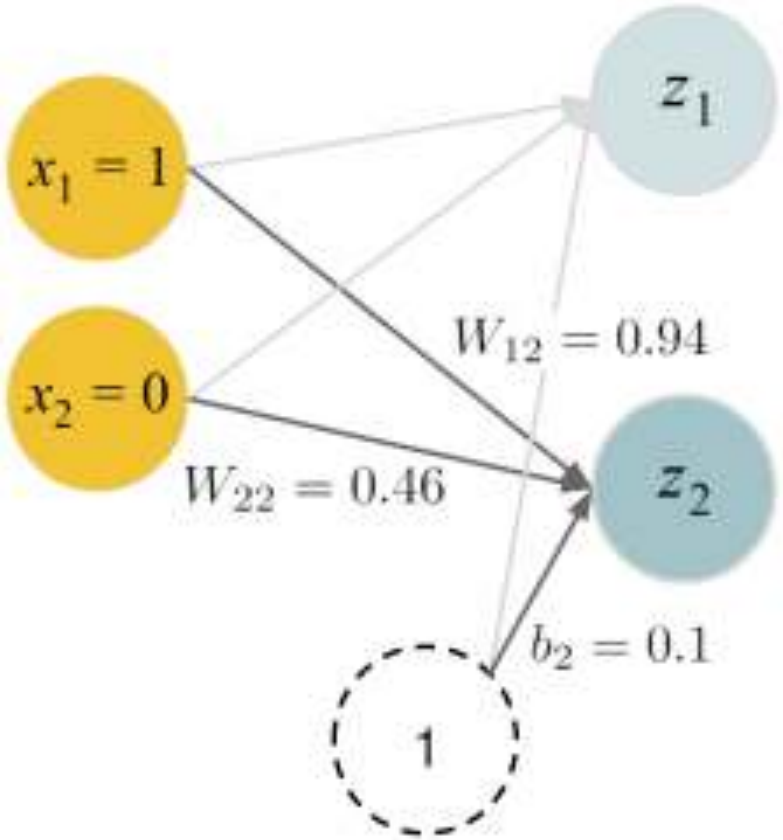
FORWARD PASS



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

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

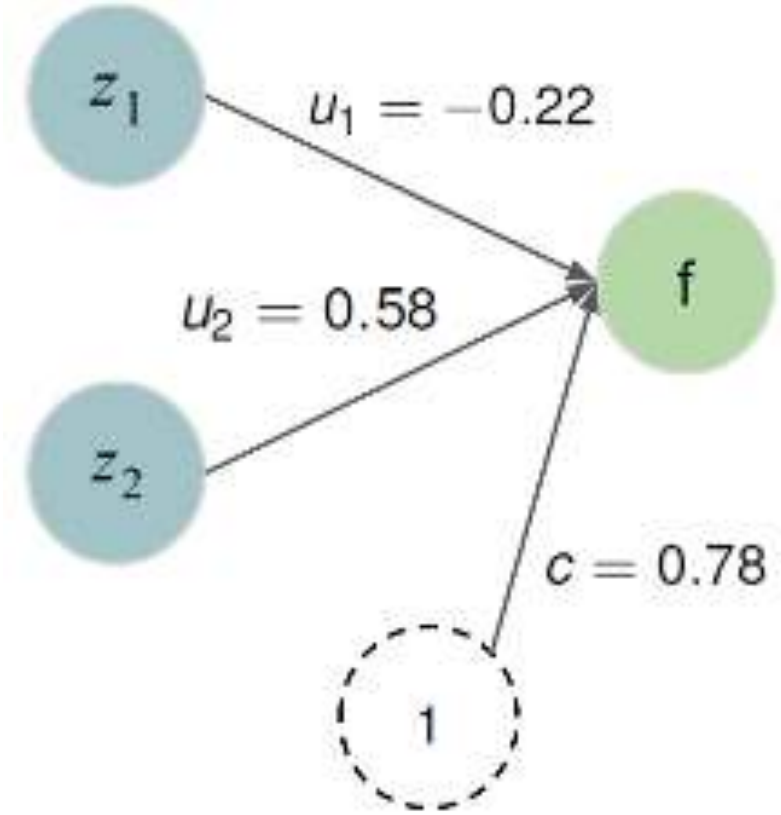
FORWARD PASS



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

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

FORWARD PASS



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

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

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:

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

- 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:

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

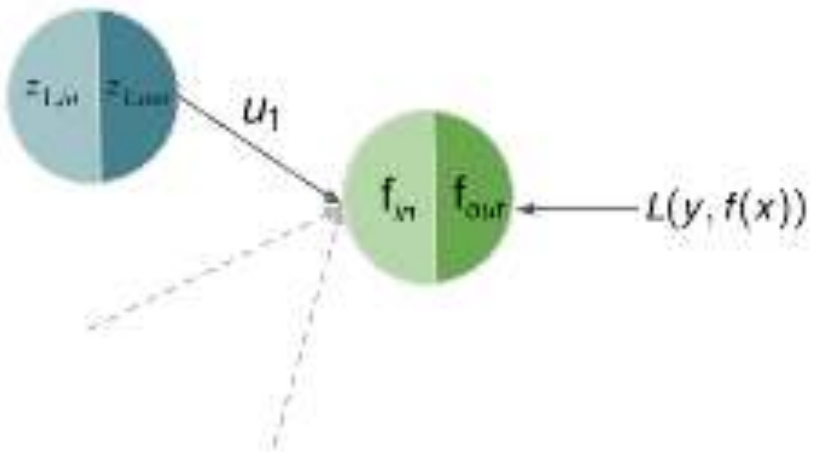
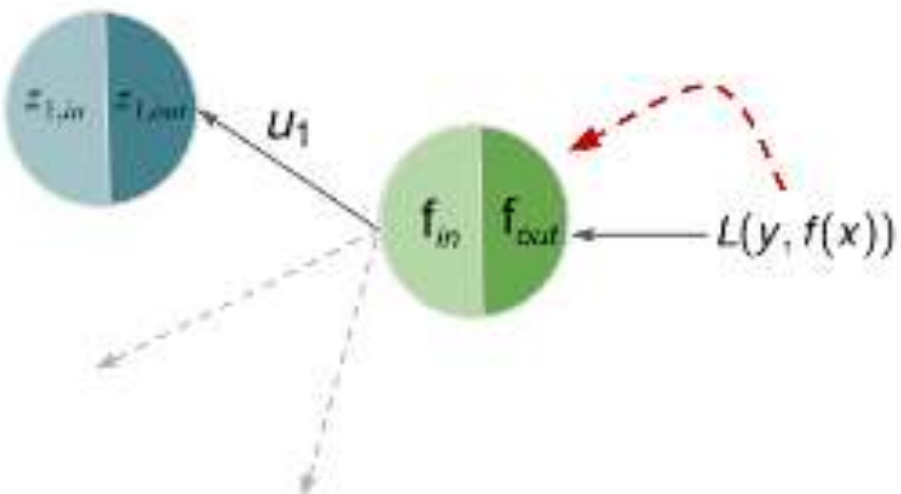


Figure: Snippet from our NN, with backward path for u_1.

BACKWARD PASS

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

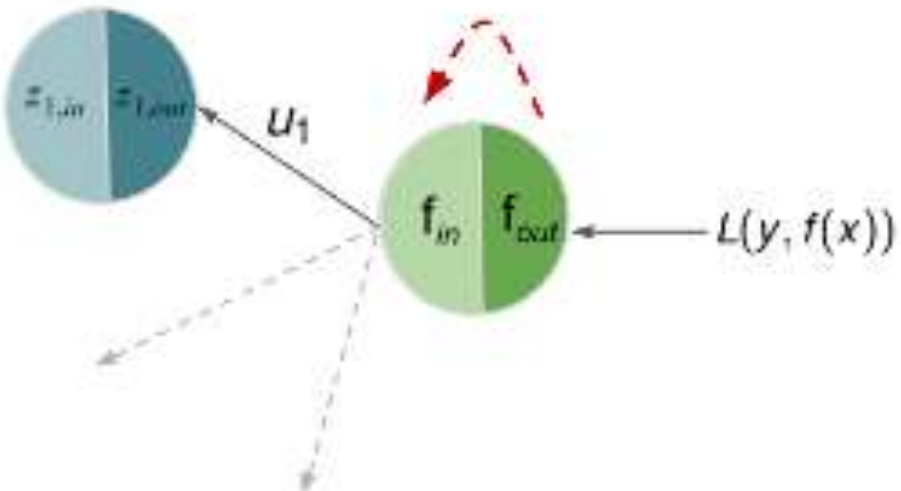
$$\begin{aligned}\frac{\partial L(y, f(\mathbf{x}))}{\partial f_{out}} &= \frac{d}{\partial f_{out}} \frac{1}{2}(y - f_{out})^2 = -\underbrace{(y - f_{out})}_{\hat{=}\text{residual}} \\ &= -(1 - 0.7525) = -0.2475\end{aligned}$$



BACKWARD PASS

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

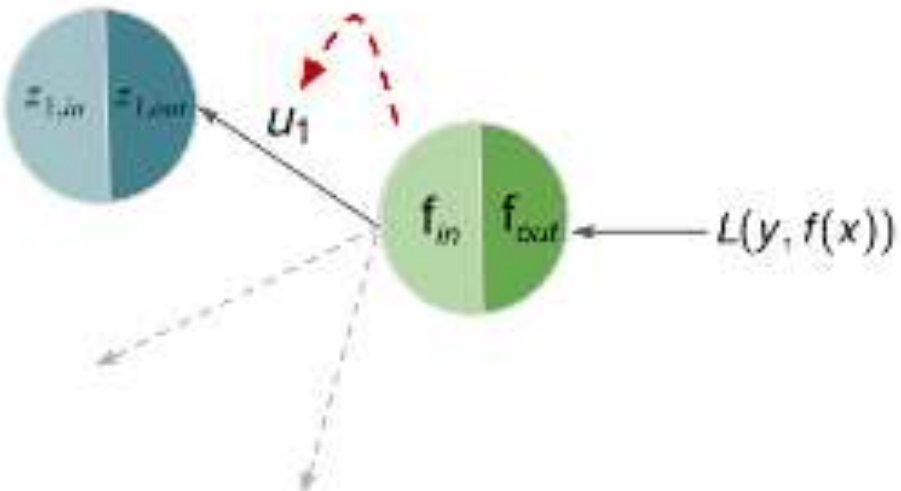
$$\begin{aligned} \frac{\partial f_{out}}{\partial f_{in}} &= \sigma(f_{in}) \cdot (1 - \sigma(f_{in})) \\ &= 0.7525 \cdot (1 - 0.7525) = 0.1862 \end{aligned}$$



BACKWARD PASS

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

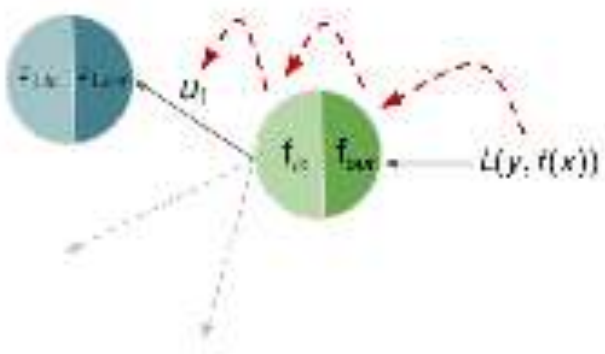
$$\frac{\partial f_{in}}{\partial u_1} = \frac{\partial(u_1 \cdot z_{1,out} + u_2 \cdot z_{2,out} + c \cdot 1)}{\partial u_1} = z_{1,out} = 0.3705$$



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 $\alpha = 0.5$:

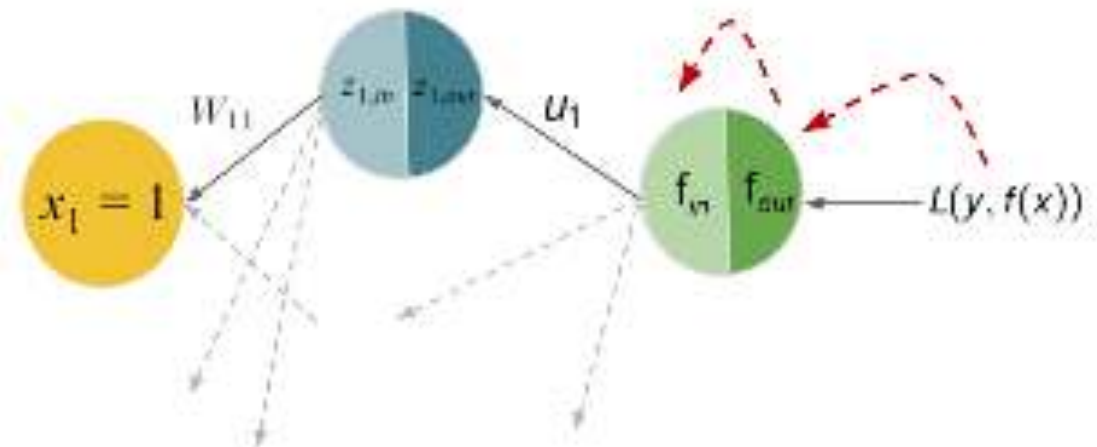
$$\begin{aligned}u_1^{[new]} &= u_1^{[old]} - \alpha \cdot \frac{\partial L(y, f(\mathbf{x}))}{\partial u_1} \\ &= -0.22 - 0.5 \cdot (-0.0171) = -0.2115\end{aligned}$$

BACKWARD PASS

- Now for W_{11}:

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

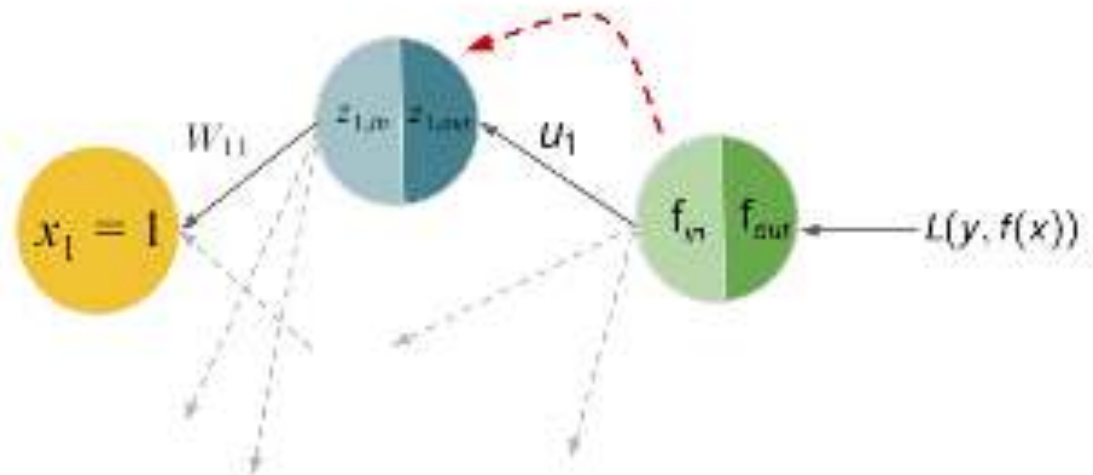
- We know $\frac{\partial L(y,f(\mathbf{x}))}{\partial f_{out}}$ and $\frac{\partial f_{out}}{\partial f_{in}}$ from BP for u_1.



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 :

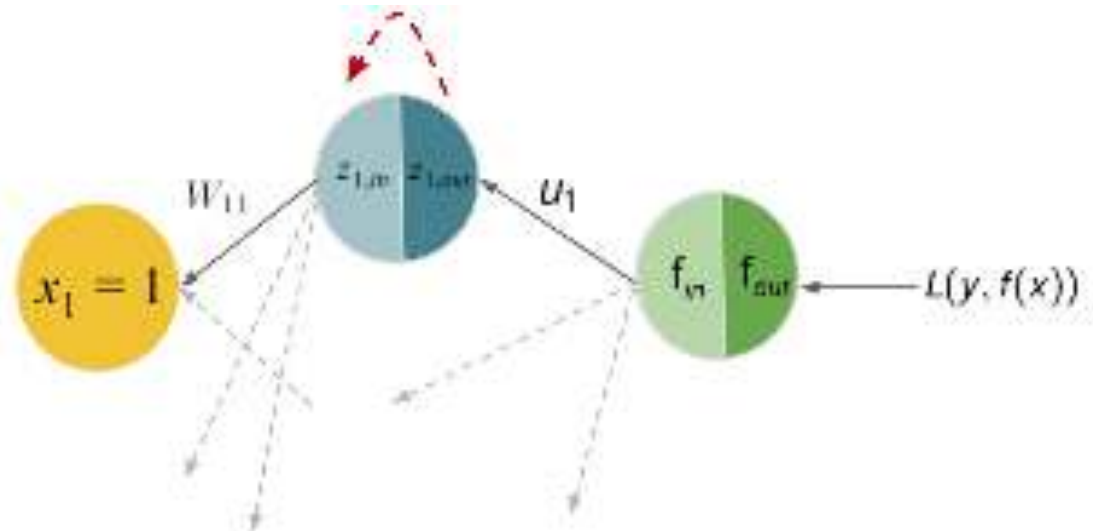
$$\frac{\partial f_{in}}{\partial z_{1,out}} = u_1 = -0.22$$



BACKWARD PASS

- Next. Use rule for σ' and FP results:

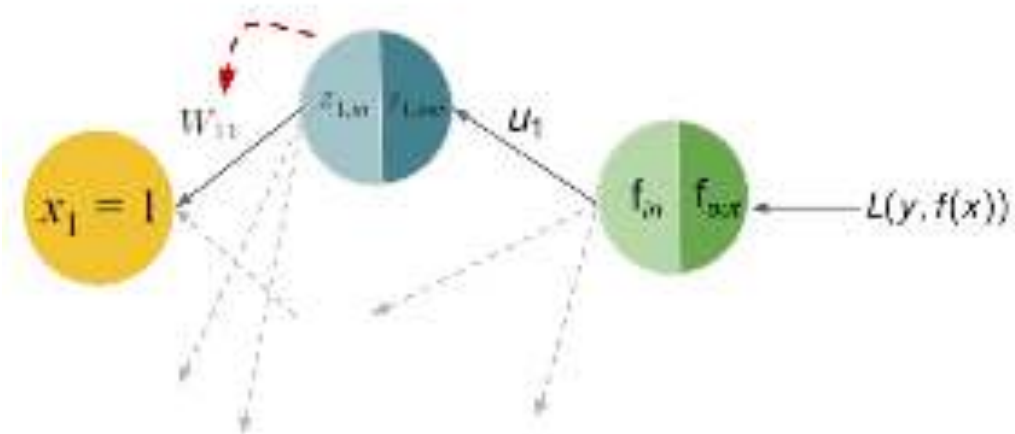
$$\begin{aligned} \frac{\partial z_{1,out}}{\partial z_{1,in}} &= \sigma(z_{1,in}) \cdot (1 - \sigma(z_{1,in})) \\ &= 0.3705 \cdot (1 - 0.3705) = 0.2332 \end{aligned}$$



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:

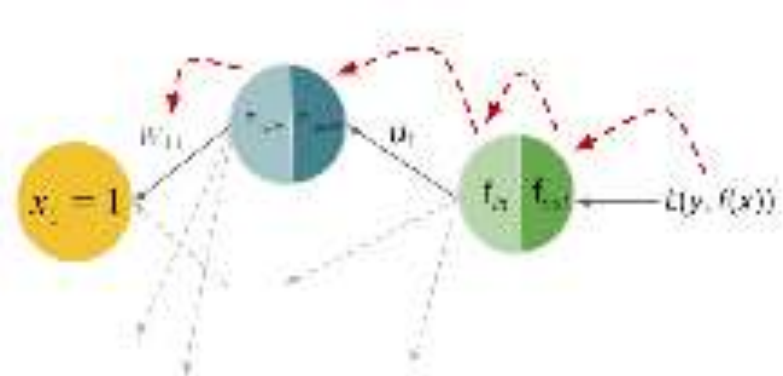
$$\frac{\partial z_{1,in}}{\partial W_{11}} = x_1 = 1$$



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 $f(\mathbf{x} \mid \boldsymbol{\theta}^{[new]}) = 0.7615$ and loss $\frac{1}{2}(1 - 0.7615)^2 = 0.0284$.
- Before, we had $f(\mathbf{x} \mid \boldsymbol{\theta}^{[old]}) = 0.7525$ 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

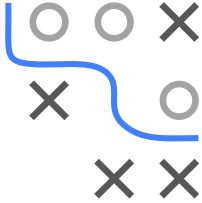
Classification and Regression Trees (CART)

Random Forests

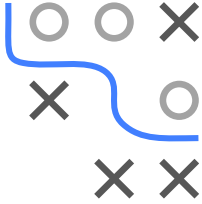
Neural Networks

Tuning

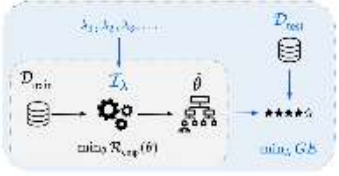
Nested Resampling



Introduction to Machine Learning



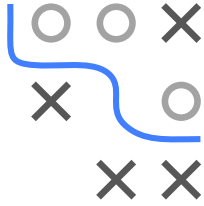
Hyperparameter Tuning
Introduction



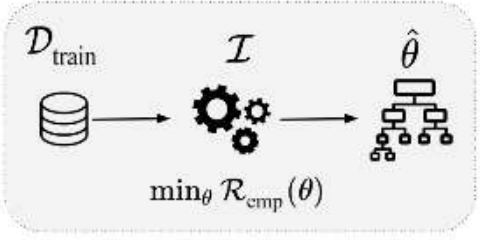
Learning goals

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

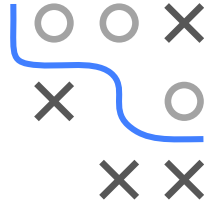
MOTIVATING EXAMPLE



- 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.

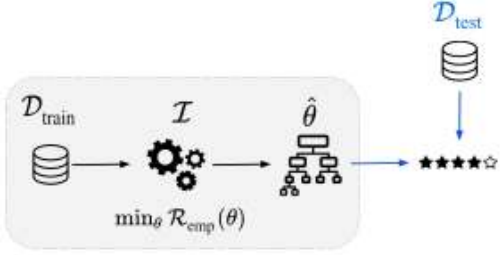


MOTIVATING EXAMPLE

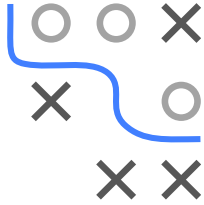


- We are **actually** interested in the **generalization performance** $\mathrm{GE}\left(\hat{f}\right)$ of the estimated model on new, previously unseen data.
- We estimate the generalization performance by evaluating the model $\hat{f} = \mathcal{I}(\mathcal{D}_{\text{train}}, \boldsymbol{\lambda})$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

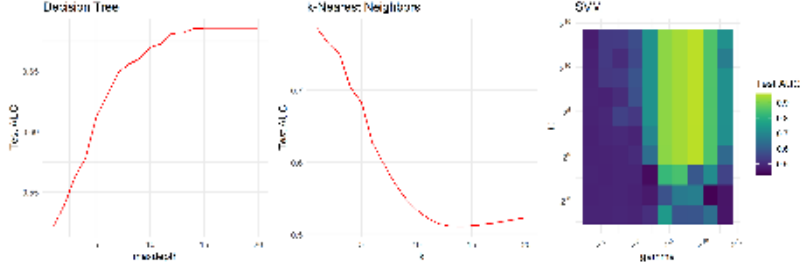
$$\widehat{\mathrm{GE}}_{\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}}}(\mathcal{I}, \boldsymbol{\lambda}, n_{\text{train}}, \rho) = \rho\left(\mathbf{y}_{\mathcal{D}_{\text{test}}}, \boldsymbol{F}_{\mathcal{D}_{\text{test}}, \hat{f}}\right)$$



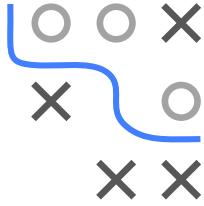
MOTIVATING EXAMPLE



- 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

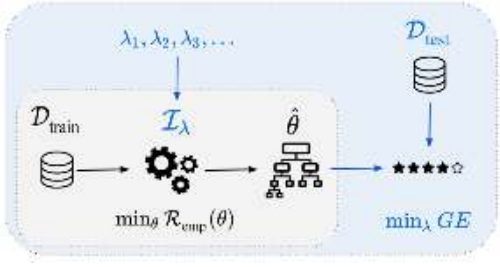


For our example this could mean:

- Data too complex to be modeled by a tree of depth 4
- Data much simpler than we thought, a tree of depth 4 overfits

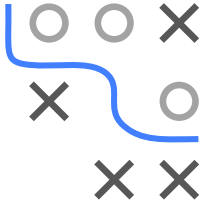
$\Longrightarrow$ Algorithmically try out different values for the tree depth. For each maximum depth λ, we have to train the model **to completion** and evaluate its performance on the test set.

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



MODEL PARAMETERS VS. HYPERPARAMETERS

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

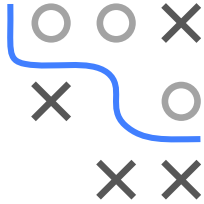


Model parameters θ are optimized during training. They are an **output** of the training.

Examples:

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

MODEL PARAMETERS VS. HYPERPARAMETERS



In contrast, **hyperparameters** (HPs) λ 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:

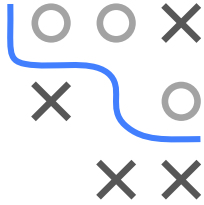
- Maximum depth of a tree
- k and which distance measure to use for k-NN
- Number and maximal order of interactions to be included in a linear regression model
- Number of optimization steps if the empirical risk minimization is done via gradient descent

TYPES OF HYPERPARAMETERS

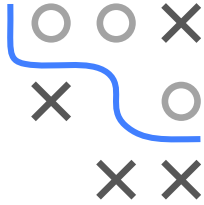
We summarize all hyperparameters we want to tune in a vector $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ of (possibly) mixed type. HPs can have different types:

- 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?

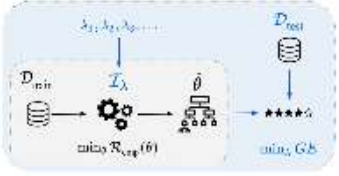
Hyperparameters are often **hierarchically dependent** on each other, e.g., *if* we use a kernel-density estimate for Naive Bayes, what is its width?



Introduction to Machine Learning



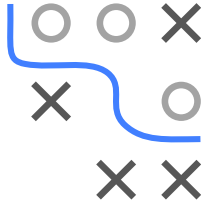
Hyperparameter Tuning
Problem Definition



Learning goals

- Definition of HPO objective and components
- Understand its properties
- What makes tuning challenging

HYPERPARAMETER OPTIMIZATION



Hyperparameters (HP) $\boldsymbol{\lambda}$ are parameters that are *inputs* to learner $\mathcal{I}$ which performs ERM on training data set to find optimal **model parameters** $\boldsymbol{\theta}$. HPs can influence the generalization performance in a non-trivial and subtle way.

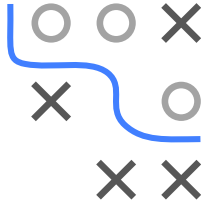
Hyperparameter optimization (HPO) / **Tuning** is the process of finding a well-performing hyperparameter configuration (HPC) $\boldsymbol{\lambda} \in \tilde{\boldsymbol{\Lambda}}$ for an learner $\mathcal{I}_{\boldsymbol{\lambda}}$.

OBJECTIVE AND SEARCH SPACE

Search space $\tilde{\boldsymbol{\Lambda}} \subset \boldsymbol{\Lambda}$ with all optimized HPs and ranges:

$$\tilde{\boldsymbol{\Lambda}} = \tilde{\boldsymbol{\Lambda}}_1 \times \tilde{\boldsymbol{\Lambda}}_2 \times \cdots \times \tilde{\boldsymbol{\Lambda}}_I$$

where $\tilde{\boldsymbol{\Lambda}}_i$ is a bounded subset of the domain of the i-th HP $\boldsymbol{\Lambda}_i$, and can be either continuous, discrete, or categorical.



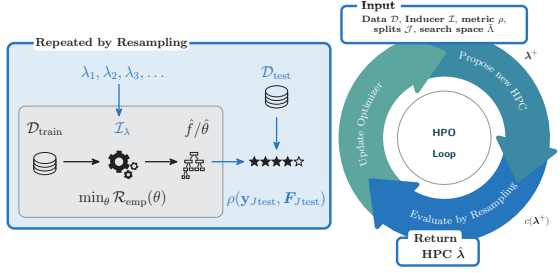
The general HPO problem is defined as:

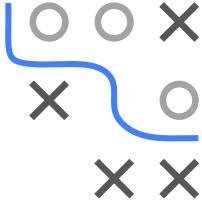
$$\boldsymbol{\lambda}^* \in \arg\min_{\boldsymbol{\lambda} \in \tilde{\boldsymbol{\Lambda}}} c(\boldsymbol{\lambda}) = \arg\min_{\boldsymbol{\lambda} \in \tilde{\boldsymbol{\Lambda}}} \widehat{\mathrm{GE}}(\mathcal{I}, \mathcal{J}, \rho, \boldsymbol{\lambda})$$

with $\boldsymbol{\lambda}^*$ as theoretical optimum, and $c(\boldsymbol{\lambda})$ is short for estim. gen. error when $\mathcal{I}$, resampling splits $\mathcal{J}$, performance measure ρ are fixed.

OBJECTIVE AND SEARCH SPACE

$$\boldsymbol{\lambda}^* \in \underset{\boldsymbol{\lambda} \in \tilde{\boldsymbol{\Lambda}}}{\arg\min}\, c(\boldsymbol{\lambda}) = \underset{\boldsymbol{\lambda} \in \tilde{\boldsymbol{\Lambda}}}{\arg\min}\, \widehat{\mathrm{GE}}(\mathcal{I}, \mathcal{J}, \rho, \boldsymbol{\lambda})$$

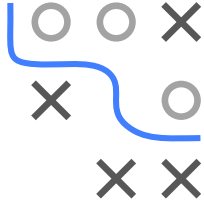




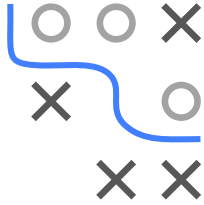
- Evals are stored in **archive**
  $\mathcal{A} = ((\boldsymbol{\lambda}^{(1)}, c(\boldsymbol{\lambda}^{(1)})), (\boldsymbol{\lambda}^{(2)}, c(\boldsymbol{\lambda}^{(2)})), \ldots)$, with
  $\mathcal{A}^{[t+1]} = \mathcal{A}^{[t]} \cup (\boldsymbol{\lambda}^+, c(\boldsymbol{\lambda}^+))$.
- We can define tuner as function $\tau : (\mathcal{D}, \mathcal{I}, \tilde{\boldsymbol{\Lambda}}, \mathcal{J}, \rho) \mapsto \hat{\boldsymbol{\lambda}}$

WHY IS TUNING SO HARD?

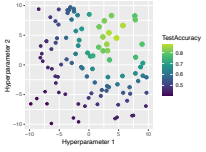
- Tuning is usually **black box**: No derivatives of the objective are availabe. We can only eval the performance for a given HPC via a computer program (CV of learner on data).
- Every evaluation can require multiple train and predict steps, hence it's **expensive**.
- Even worse: the answer we get 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

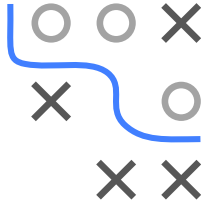


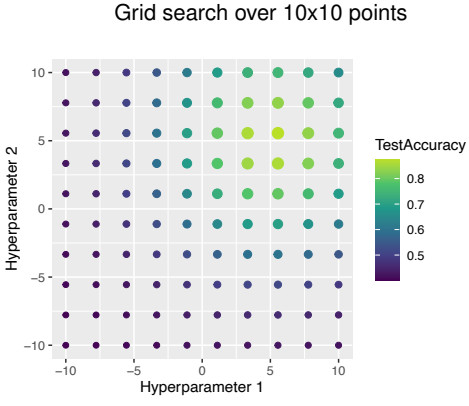
Learning goals

- 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

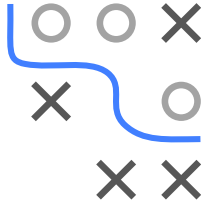




GRID SEARCH

Advantages

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

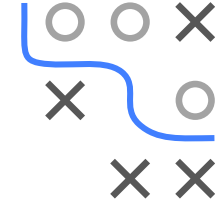


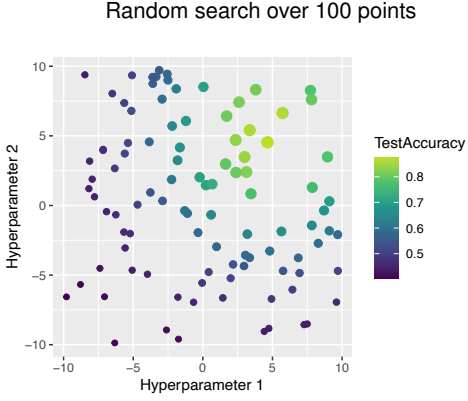
Disadvantages

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

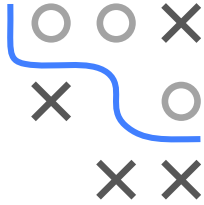
RANDOM SEARCH

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





RANDOM SEARCH



Advantages

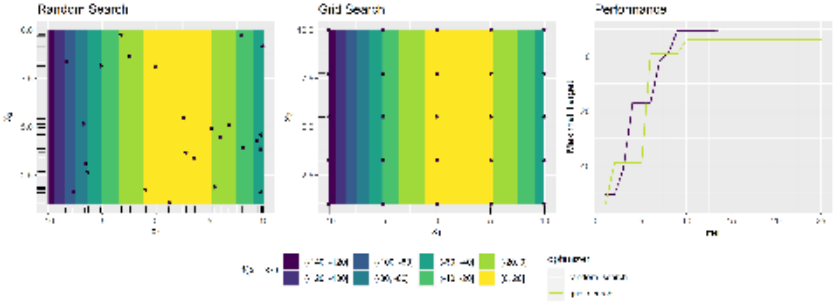
- Like grid search: very easy to implement, all parameter types possible, trivial parallelization
- Anytime algorithm: can stop the search whenever our budget for computation is exhausted, or continue until we reach our performance goal.
- No discretization: each individual parameter is tried with a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

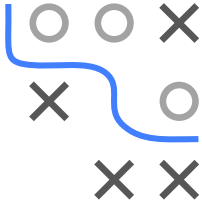
Disadvantages

- 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.

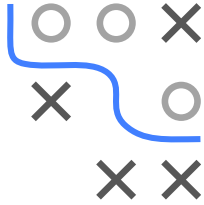




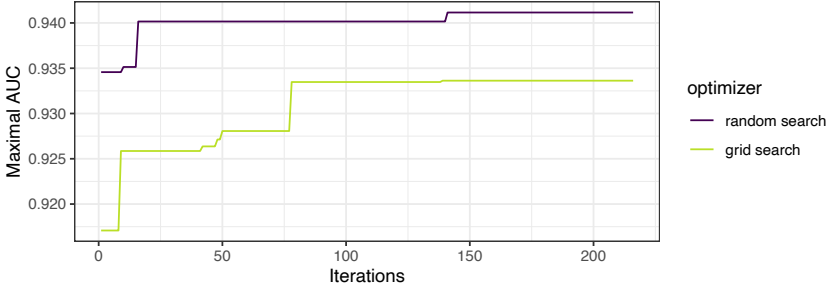
$\Rightarrow$ In this setting, random search is more superior as we get a better coverage for the parameter x_1 in comparison with grid search, where we only discover 5 distinct values for x_1.

TUNING EXAMPLE

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

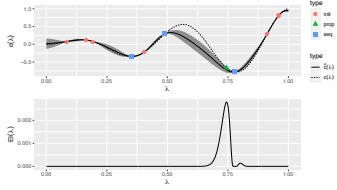


Hyperparameter	Type	Min	Max
`num.trees`	integer	3	500
`mtry`	integer	5	50
`min.node.size`	integer	10	100



Introduction to Machine Learning

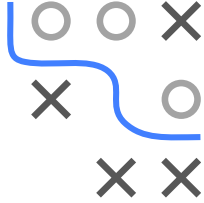
Hyperparameter Tuning
Advanced Tuning Techniques



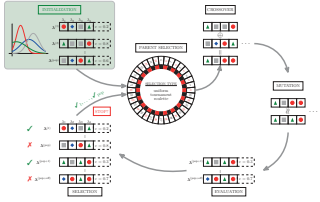
Learning goals

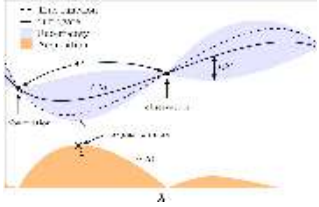
- Basic idea of evolutionary algorithms
- and Bayesian Optimization
- and hyperband

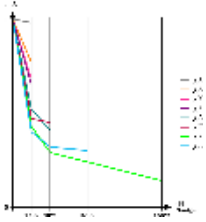
HPO – MANY APPROACHES



- Evolutionary algorithms
- Bayesian / model-based optimization
- Multi-fidelity optimization, e.g. Hyperband





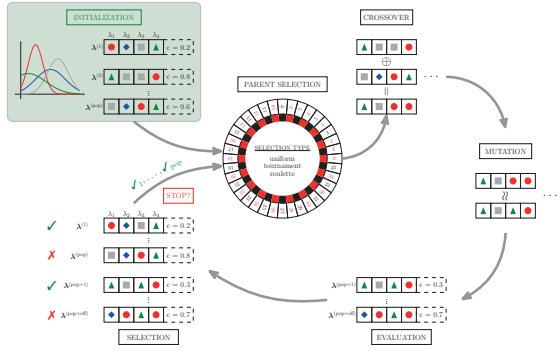


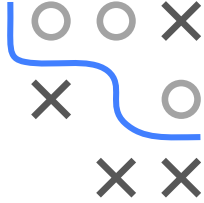
HPO methods can be characterized by:

- 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

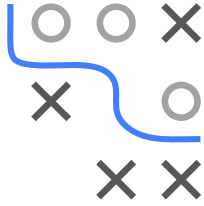




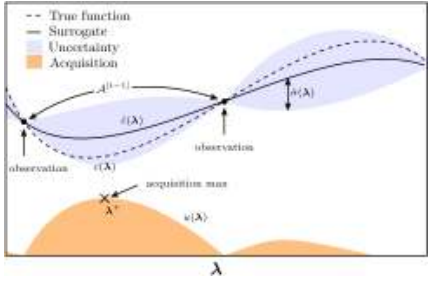
- Are a class of stochastic population-based optimization methods inspired by the concepts of biological evolution
- Are applicable to HPO since they do not require gradients
- Mutation is the (randomized) change of one or a few HP values in a configuration.
- Crossover creates a new HPC by (randomly) mixing the values of two other configurations.

BAYESIAN OPTIMIZATION

BO sequentially iterates:

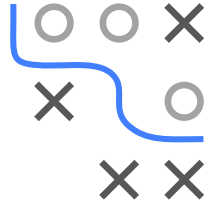


1. **Approximate** $\boldsymbol{\lambda} \mapsto c(\boldsymbol{\lambda})$ by (nonlin) regression model $\hat{c}(\boldsymbol{\lambda})$, from evaluated configurations (archive)
2. **Propose candidates** via optimizing an acquisition function that is based on the surrogate $\hat{c}(\boldsymbol{\lambda})$
3. **Evaluate** candidate(s) proposed in 2, then go to 1



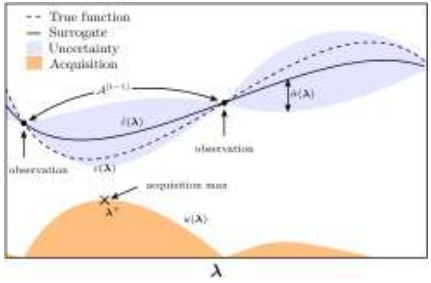
Important trade-off: **Exploration** (evaluate candidates in under-explored areas) vs. **exploitation** (search near promising areas)

BAYESIAN OPTIMIZATION



Surrogate Model:

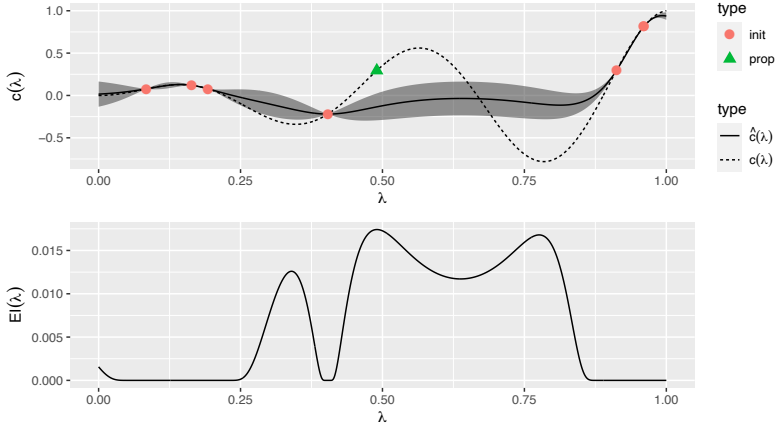
- Probabilistic modeling of $C(\boldsymbol{\lambda}) \sim (\hat{c}(\boldsymbol{\lambda}), \hat{\sigma}(\boldsymbol{\lambda}))$ with posterior mean $\hat{c}(\boldsymbol{\lambda})$ and uncertainty $\hat{\sigma}(\boldsymbol{\lambda})$.
- Typical choices for numeric spaces are Gaussian Processes; random forests for mixed spaces

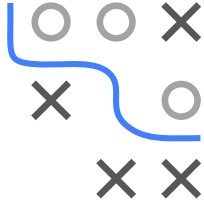


Acquisition Function:

- Balance exploration (high $\hat{\sigma}$) vs. exploitation (low $\hat{c}$).
- Lower confidence bound (LCB): $a(\boldsymbol{\lambda}) = \hat{c}(\boldsymbol{\lambda}) - \kappa \cdot \hat{\sigma}(\boldsymbol{\lambda})$
- Expected improvement (EI): $a(\boldsymbol{\lambda}) = \mathbb{E}\left[\max\left\{c_{\min} - C(\boldsymbol{\lambda}), 0\right\}\right]$ where ($c_{\min}$ is best cost value from archive)
- Optimizing $a(\boldsymbol{\lambda})$ is still difficult, but cheap(er)

BAYESIAN OPTIMIZATION

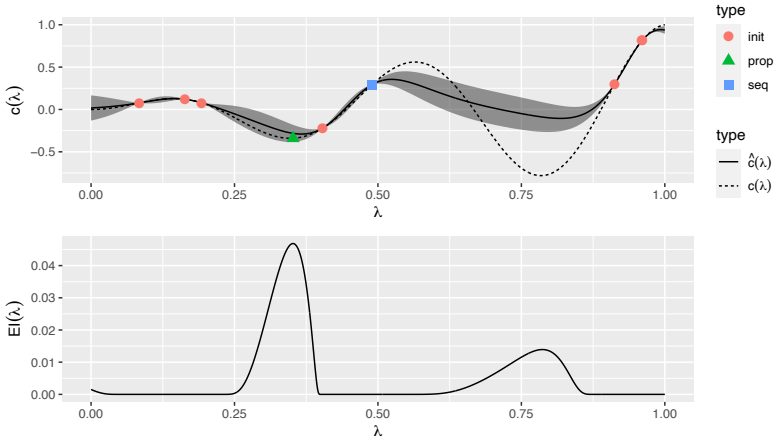


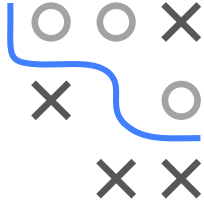


Upper plot: The surrogate model (black, solid) models the *unknown* relationship between input and output (black, dashed) based on the initial design (red points).

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

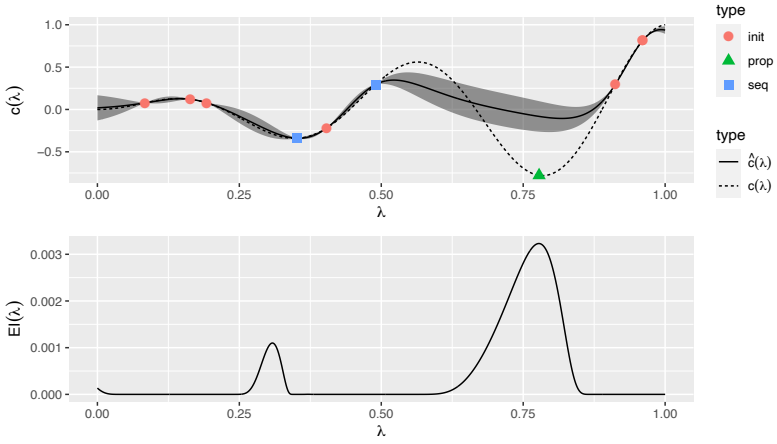


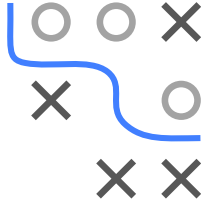


Upper plot: The surrogate model (black, solid) models the *unknown* relationship between input and output (black, dashed) based on the initial design (red points).

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

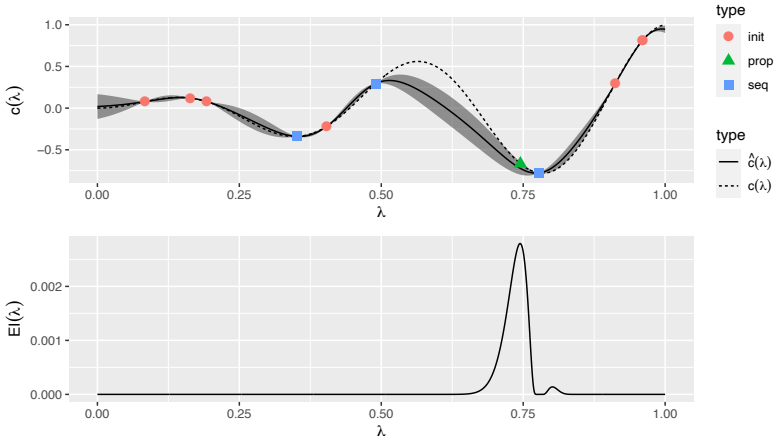


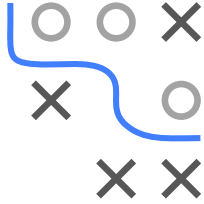


Upper plot: The surrogate model (black, solid) models the *unknown* relationship between input and output (black, dashed) based on the initial design (red points).

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



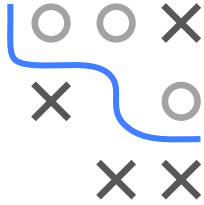


Upper plot: The surrogate model (black, solid) models the *unknown* relationship between input and output (black, dashed) based on the initial design (red points).

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 $\mathbf{\Lambda}$ and avoid irrelevant evaluations:



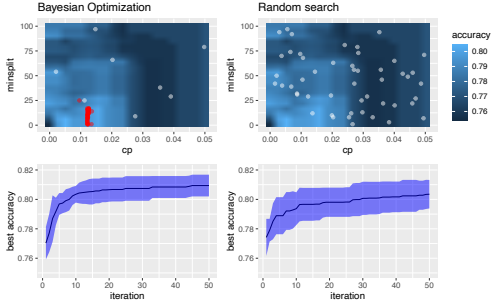
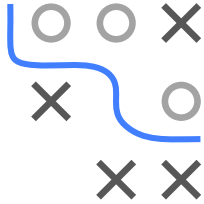


Figure: Tuning complexity and minimal node size for splits for CART on the `titanic` data (10-fold CV maximizing accuracy).
Left panel: BO, 50 configurations; right panel: random search, 50 iterations.
Top panel: one run (initial design of BO is white); bottom panel: mean $\pm$ std of 10 runs.

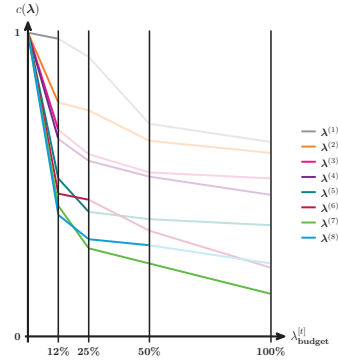
MULTIFIDELITY OPTIMIZATION

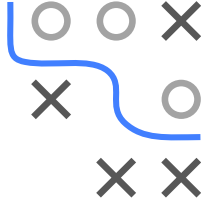


- Prerequiste: Fidelity HP λ_{fid}, i.e., a component of $\boldsymbol{\lambda}$, which influences the computational cost of the fitting procedure in a monotonically increasing manner
- Methods of multifidelity optimization in HPO are all tuning approaches that can efficiently handle a $\mathcal{I}$ with a HP λ_{fid}
- The lower we set λ_{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 λ_{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 $\widehat{\mathrm{GE}}$); with η 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

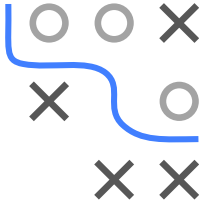
t	$\lambda_{\text{fid}}^{[t]}$	$p_2^{[t]}$
0	4	27
1	16	6
2	64	1

bracket 1

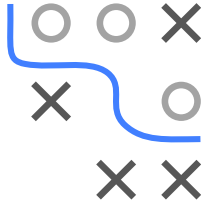
t	$\lambda_{\text{fid}}^{[t]}$	$p_1^{[t]}$
0	16	10
1	64	2

bracket 0

t	$\lambda_{\text{fid}}^{[t]}$	$p_0^{[t]}$
0	64	5



MORE TUNING ALGORITHMS:

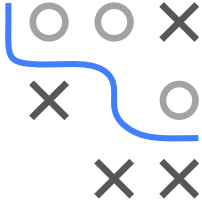


Other advanced techniques besides model-based optimization and the hyperband algorithm are:

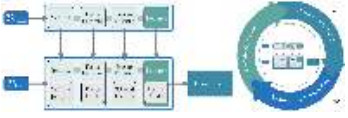
- Stochastic local search, e.g., simulated annealing
- Genetic algorithms / CMAES
- Iterated F-Racing
- Many more . . .

For more information see *Hyperparameter Optimization: Foundations, Algorithms, Best Practices and Open Challenges*, Bischl (2021)

Introduction to Machine Learning



Hyperparameter Tuning
Pipelines and AutoML

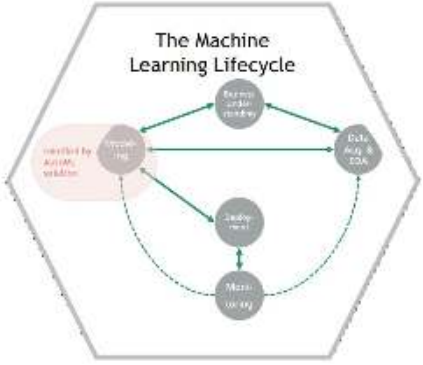


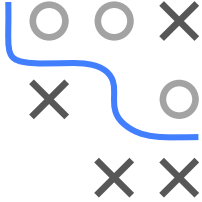
Learning goals

- Pipelines as connected steps of learnable operations
- Sequential pipeline
- Pipelines and DAGs

CASE FOR AUTOML

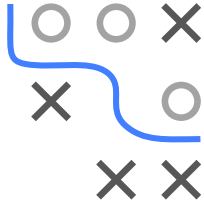
- More and more tasks are approached via data driven methods.
- Data scientists often rely on trial-and-error.
- The process is especially tedious for similar, recurring tasks.
- Not the entire machine learning lifecycle can be automated.

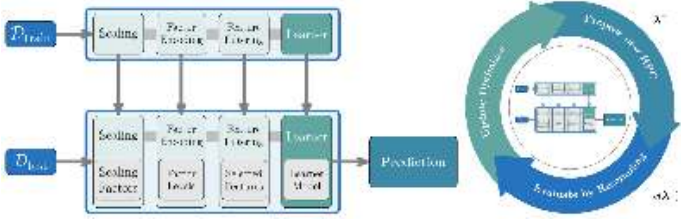




PIPELINES AND AUTOML

- 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

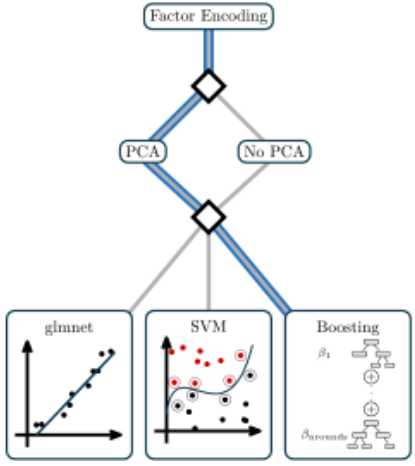


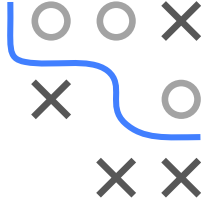


Pipelines are required to embed full model building into CV to avoid overfitting and biased evaluation!

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, ...



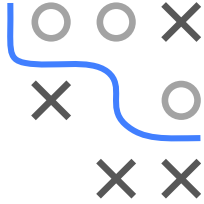


PIPELINES AND AUTOML

- HPs of pipeline are the joint set of all HPs of its contained nodes:
  $\tilde{\mathbf{\Lambda}} = \tilde{\mathbf{\Lambda}}_{\text{op},1} \times \cdots \times \tilde{\mathbf{\Lambda}}_{\text{op},k} \times \tilde{\mathbf{\Lambda}}_{\mathcal{I}}$
- HP space of a DAG is more complex: Depending on branching / selection different nodes and HPs are active
  $\rightarrow$ **hierarchical search space**

Search Space A

Name	Type	Bounds/Values	Trafo
encoding	C	one-hot, impact	
◇ pca	C	PCA, no PCA	
◇ learner	C	glmnet, SVM, Boosting	
if learner = glmnet			
s	R	$[-12, 12]$	2^x
alpha	R	$[0, 1]$	
if learner = SVM			
cost	R	$[-12, 12]$	2^x
gamma	R	$[-12, 12]$	2^x
if learner = Boosting			
eta	R	$[-4, 0]$	10^x
nrounds	I	$\{1, \ldots, 5000\}$	–
max_depth	I	$\{1, \ldots, 20\}$	–

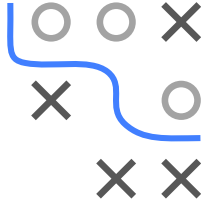


A graph that includes many preprocessing steps and learner types can be flexible enough to work on a large number of data sets

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 intepretability
- 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

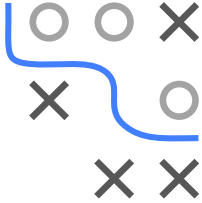
Classification and Regression Trees (CART)

Random Forests

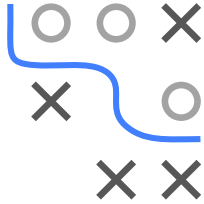
Neural Networks

Tuning

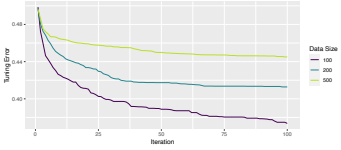
Nested Resampling



Introduction to Machine Learning



Nested Resampling Motivation

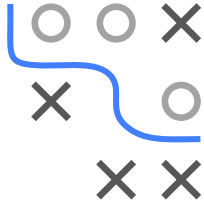


Learning goals

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

MOTIVATION

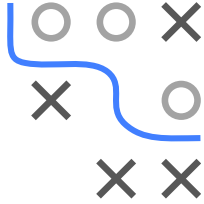
Selecting the best model from a set of potential candidates (e.g., different classes of learners, different hyperparameter settings, different feature sets, different preprocessing, ...) is an important part of most machine learning problems.



Problem

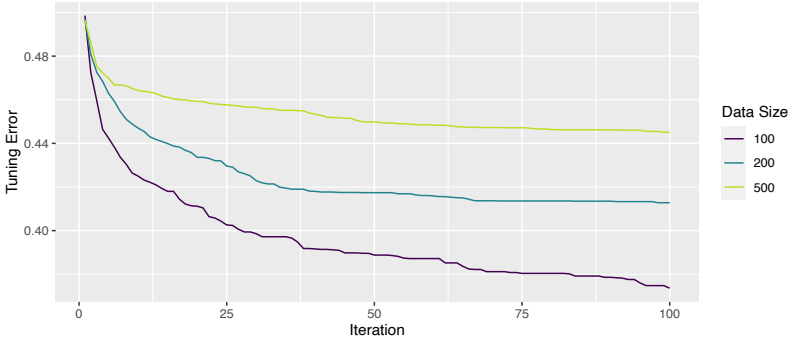
- We cannot evaluate our finally selected learner on the same resampling splits that we have used to perform model selection for it, e.g., to tune its hyperparameters.
- By repeatedly evaluating the learner on the same test set, or the same CV splits, information about the test set “leaks” into our evaluation.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

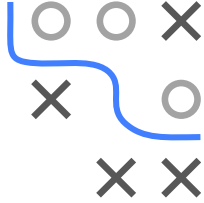
INSTRUCTIVE AND PROBLEMATIC EXAMPLE



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

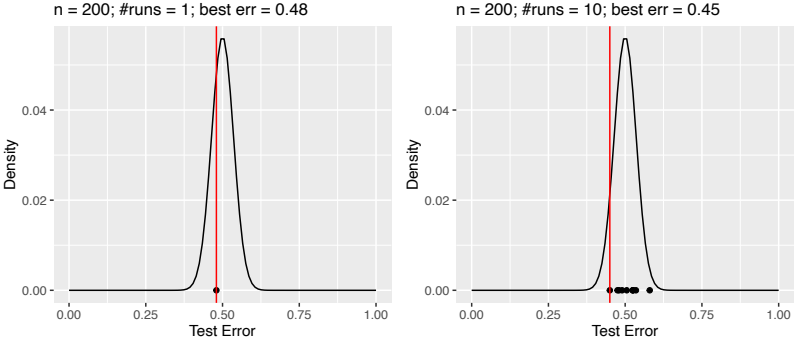
INSTRUCTIVE AND PROBLEMATIC EXAMPLE

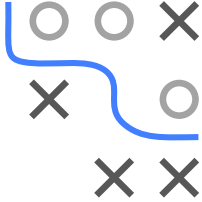




- Plotted is the best “tuning error” (i.e. the performance of the model with fixed λ as evaluated by the cross-validation) after k tuning iterations.
- We have performed the experiment for different sizes of learning data that were cross-validated.

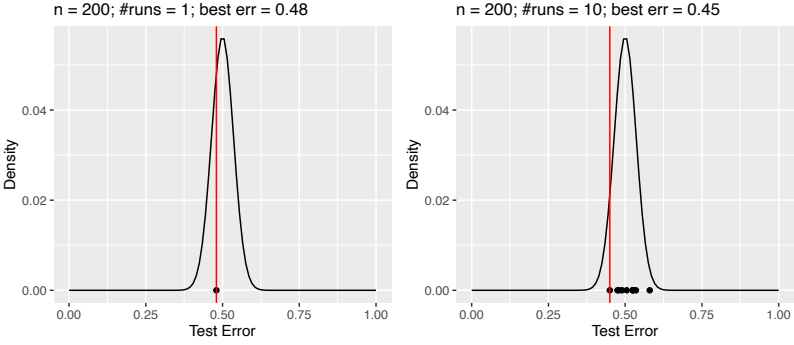
INSTRUCTIVE AND PROBLEMATIC EXAMPLE

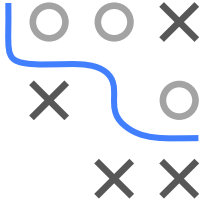




- For 1 experiment, the CV score will be nearly 0.5, as expected
- We basically sample from a (rescaled) binomial distribution when we calculate error rates
- And multiple experiment scores are also nicely arranged around the expected mean 0.5

INSTRUCTIVE AND PROBLEMATIC EXAMPLE

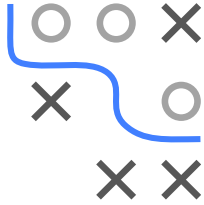




- But in tuning we take the minimum of those! So we don't really estimate the "average performance" anymore, we get an estimate of "best case" performance instead.
- The more we sample, the more "biased" this value becomes.

UNTOUCHED TEST SET PRINCIPLE

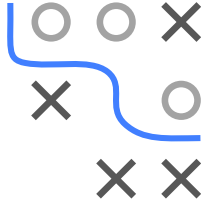
Countermeasure: simulate what actually happens in model application.



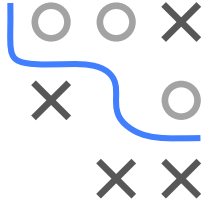
- All parts of the model building (including model selection, preprocessing) should be embedded in the model-finding process **on the training data**.
- The test set should only be touched once, so we have no way of "cheating". The test data set is only used once *after* a model is completely trained, 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

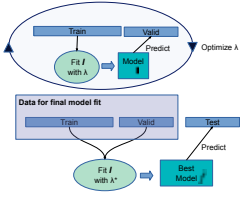
- For steps that themselves require resampling (e.g., hyperparameter tuning) this results in **nested resampling**, i.e., resampling strategies for both
  - tuning: an inner resampling loop to find what works best based on training data
  - outer evaluation on data not used for tuning to get honest estimates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



Introduction to Machine Learning



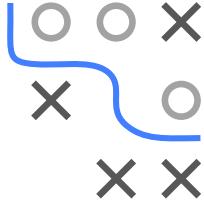
Nested Resampling
Training - Validation - Test



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



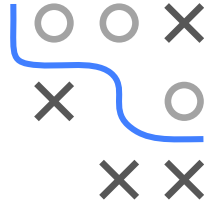
Remember:

We need to

- **select an optimal learner**
- without compromising the **accuracy of the performance estimate** for that learner

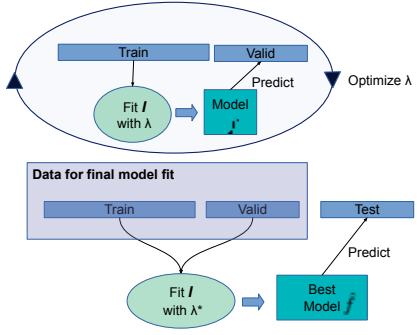
– for that we need an **untouched test set**!

TRAIN - VALIDATION - TEST



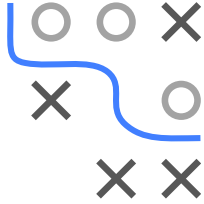
Simplest method to achieve this: a 3-way split

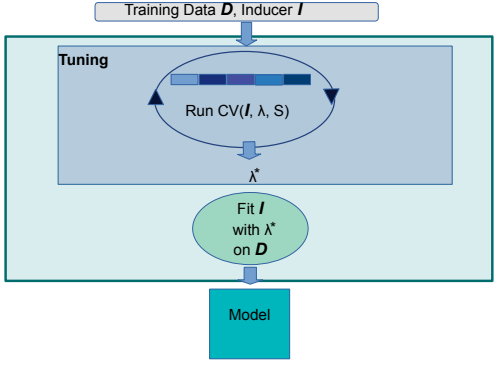
- During tuning, a learner is trained on the **training set**, evaluated on the **validation set**
- After the best model configuration $\boldsymbol{\lambda}^*$ has been selected, we re-train on the joint (training+validation) set and evaluate the model's performance on the **test set**.



TUNING AS PART OF MODEL BUILDING

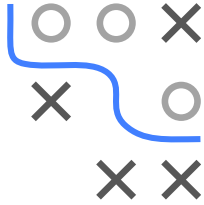
- Effectively, the tuning step is now simply part of a more complex training procedure.
- We could see this as removing the hyperparameters from the inputs of the algorithm and making it “self-tuning”.

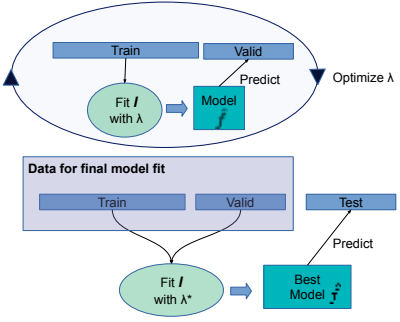


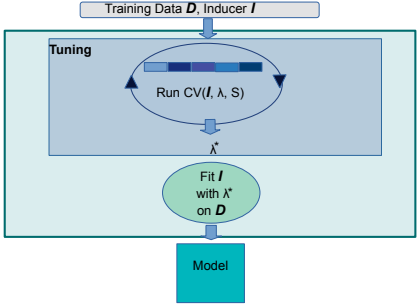


TUNING AS PART OF MODEL BUILDING

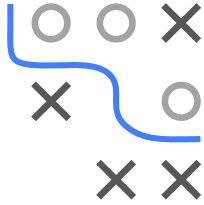
More precisely: the combined training & validation set is actually the training set for the “self-tuning” endowed algorithm.



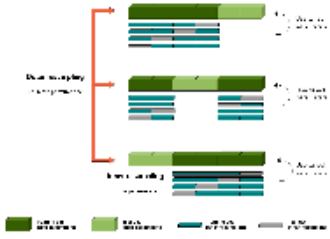




Introduction to Machine Learning



Nested Resampling
Nested Resampling Procedure

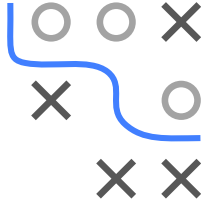


Learning goals

- 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

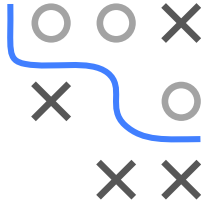
Just like we can generalize hold-out splitting to resampling to get more reliable estimates of the predictive performance, we can generalize the training/validation/test approach to **nested resampling**.

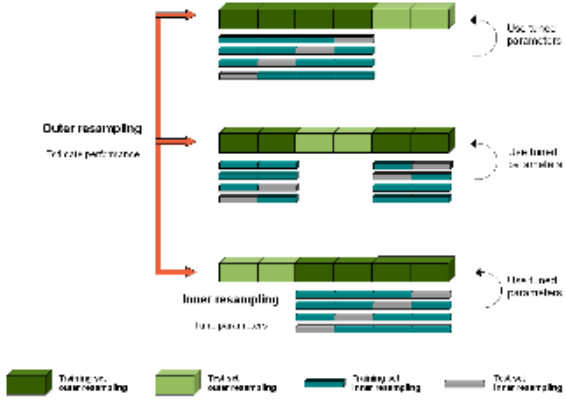


This results in two nested resampling loops, i.e., resampling strategies for both tuning and outer evaluation.

NESTED RESAMPLING

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

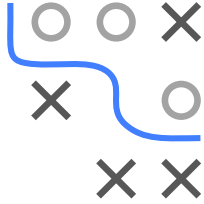


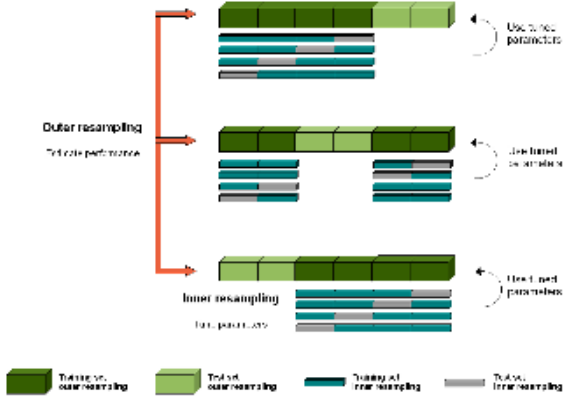


NESTED RESAMPLING

In each iteration of the outer loop we:

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

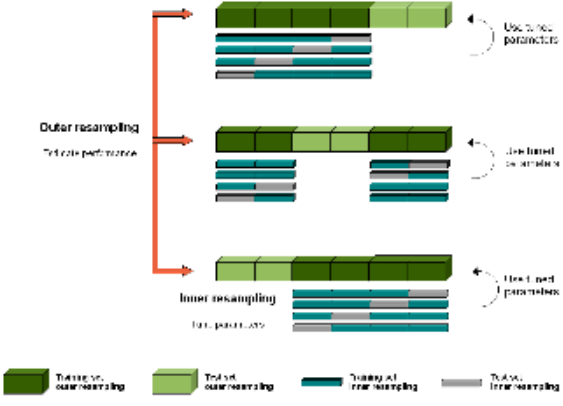




NESTED RESAMPLING

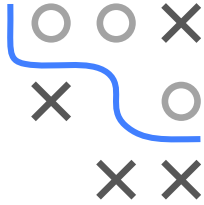
In each iteration of the outer loop we:

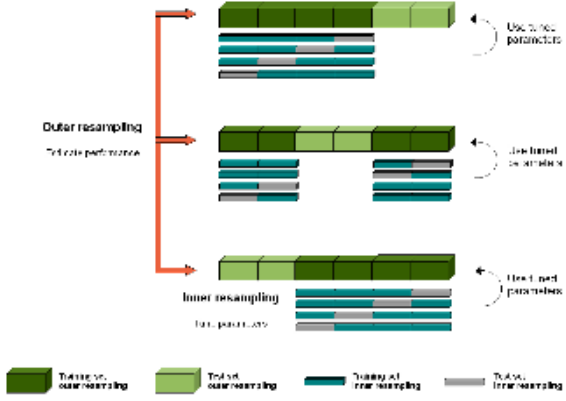
- Return the winning λ^* that performed best on the grey inner test sets
- Re-train the model on the full outer dark green train set
- Evaluate it on the outer light green test set



NESTED RESAMPLING

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





NESTED RESAMPLING - INSTRUCTIVE EXAMPLE

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

