

Third-year ISAE-SUPAERO engineering students
Research Area: Neuro & AI
December, 2020

Neuro & AI: Methods and Tools for Neuroergonomics

Introduction to Machine Learning

Nicolas Drougard¹

¹ISAE-SUPAERO DCAS, Toulouse, FRANCE

nicolas.drougard@isae-supaero.fr



Machine Learning in this third module (M3) of the NeuroAI domain

- 15 hours spread over 5 days: 7/01, 8/01, 14/01, 15/01 and 21/01.
- The main points:
 - Supervised Learning,
 - Preprocessing and dimensionality reduction,
 - Unsupervised Learning,
 - Theoretical Results,
 - Machine Learning in practice and for BCIs.

1 Introduction

- Artificial Intelligence, Machine Learning and Big Data?
- Basics in Probability Theory and Statistics

2 Supervised Learning

- Evaluation of a prediction function.
- Method and algorithms

3 Neural Networks and Deep learning

4 Mathematical tools for Machine Learning

5 Risks

- ▶ [F.C18] F.Chollet, *Deep learning with python*, MANNING, 2018, download link [here](#).
- ▶ [GS20] X. Gendre and F. Simatos, *Probabilités et statistique*, Tronc Commun Scientifique 1A, Formation Ingénieur ISAE-SUPAERO, 2020.
- ▶ [HMS20] G. Haine, D. Matignon, and M. Salaün, *Mathématiques déterministes*, Tronc Commun Scientifique 1A, Formation Ingénieur ISAE-SUPAERO, 2020.
- ▶ [HTF09] Trevor Hastie, Robert Tibshirani, and Jerome Friedman, *The elements of statistical learning: data mining, inference, and prediction*, Springer Science & Business Media, 2009, download link [here](#).

1 Introduction

- Artificial Intelligence, Machine Learning and Big Data?
- Basics in Probability Theory and Statistics

2 Supervised Learning

3 Neural Networks and Deep learning

4 Mathematical tools for Machine Learning

5 Risks

What is Artificial Intelligence?

Recent successes of AI applications

- ATTOL project with fully autonomous flight tests.



- Autonomous vehicles (*cf.* class of Caroline Chanel).



Recent successes of AI applications

- Deep fakes, filters.



- Artificially generated pictures <https://thispersondoesnotexist.com> (Generative Adversarial Networks, GANs).

Recent successes of AI applications

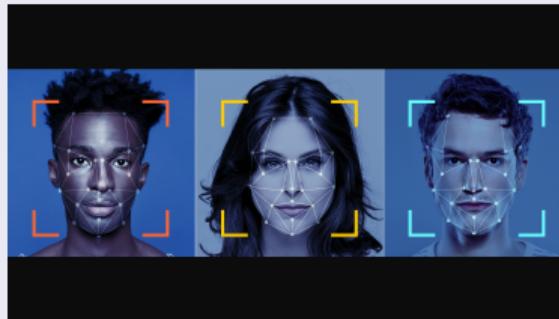
- Artificially generated artworks.



- Machine translation: google translate, deepl (based on linguee database).

Recent successes of AI applications

- Facial recognition, assessment of emotions in facial expressions.



- Personal assistant (e.g. siri, alexia, google home), music recognition (e.g. shazam).



Recent successes of AI applications

- Alphago



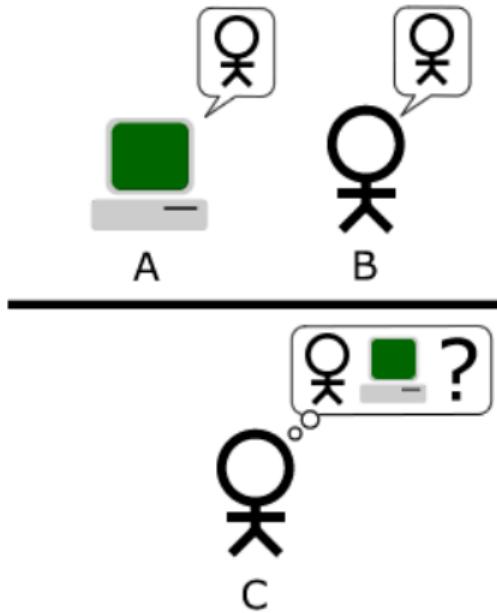
- recommendation systems, smart picture management, chatbots, fraud detection, speech synthesis, etc.

Attempt to define “Artificial Intelligence”

A research domain, or a set of methods, tools, formalisms, algorithms and theories, with the aim of making machines able to realize intelligent tasks generally performed by animals and humans.

- Example of **intelligent tasks generally performed by animals and humans**: using language, recognizing objects, people, emotions, playing games, learning from experience, deducing facts, creating art, piloting vehicles, etc.
- When allowing algorithms/machines to do these tasks, *i.e.* to act, we talk about **artificial agents**.
- Example of **automation tasks that are not considered to be AI**: compute the solution of a complex equation, a random or very large number, give a precise quantitative value to make a system stable or make a trajectory optimal using classical mechanics, ... not naturally performed by human/animals!

- **Turing test:** test of a machine's ability to exhibit intelligent behaviour equivalent to that of a human.



A field constantly evolving

- In the 1950s, mathematical functions $\xrightarrow{\text{transformation}}$ computer-executable instructions was artificial intelligence.
Today it is just a simple functionality of compilers.
- Same remark on route planners, that use AI planning algorithm like A^* (cf. class of Caroline Chanel).

Different visions

- AI is somewhere in the intersection between **Mathematics and Computer Science**.
- Note that until now, artificial agents are more or less complex algorithms, basically **optimizing criteria**.
- **Video Games:** AI can used to refer to a Non Player Characters (NPC).
- **Science Fiction:** AI can be used to refer to a fully autonomous virtual super-human able to understand and compute almost everything, to finally rule us or save the world.
- In **french**: Intelligence is often linked to a human quality, and even to consciousness, i.e. human beings more intelligent than animals.
- In **english**: Business Intelligence / Central Intelligence Agency (CIA). Intelligence is related to the tasks (intelligent tasks).
- In **recent years**: AI sometimes refers to *Neural Networks* and *Deep Learning*.

Short history

- Naturally **starts with computers**, roughly in the 1950's,
e.g. *Dynamic Programming* (R.Bellman) or Turing test.
- Then a lot of effort is made in the direction of **automatic reasoning** with expert system
managing knowledge, ontologies, set of rules, inferring facts, using modal logic etc.
Most impressive contributions: proofs of theorems, natural language processing (NLP)
and game opponents.
- Referred to **Symbolic AI**, or GOFAI (good old-fashioned artificial intelligence).
e.g. verification of accuracy & reliability of online aircraft control software (Airbus).
- Another AI field also emerges: **Machine learning**, or data-driven AI, coming from
Statistics, Computer Science, Connexionism (neural networks to model brains).

Machine Learning thanks to data

- Machine Learning (ML) is boosted by access to **more and more databases** (Internet, Big Data) and by the increasing available **computational power**.
- Trendiest framework in ML is **Deep Learning** (Neural Networks with a lot of layers).
- Deep Learning: hard to tune, hard to derive theoretical guarantees and needs a lot of data. Researchers in this field were perceived as alchemists until **recently** (around 2010).
- As a friend of yours put it so well: "*If you are not paying for it, you're not the customer; you're the product being sold.*".

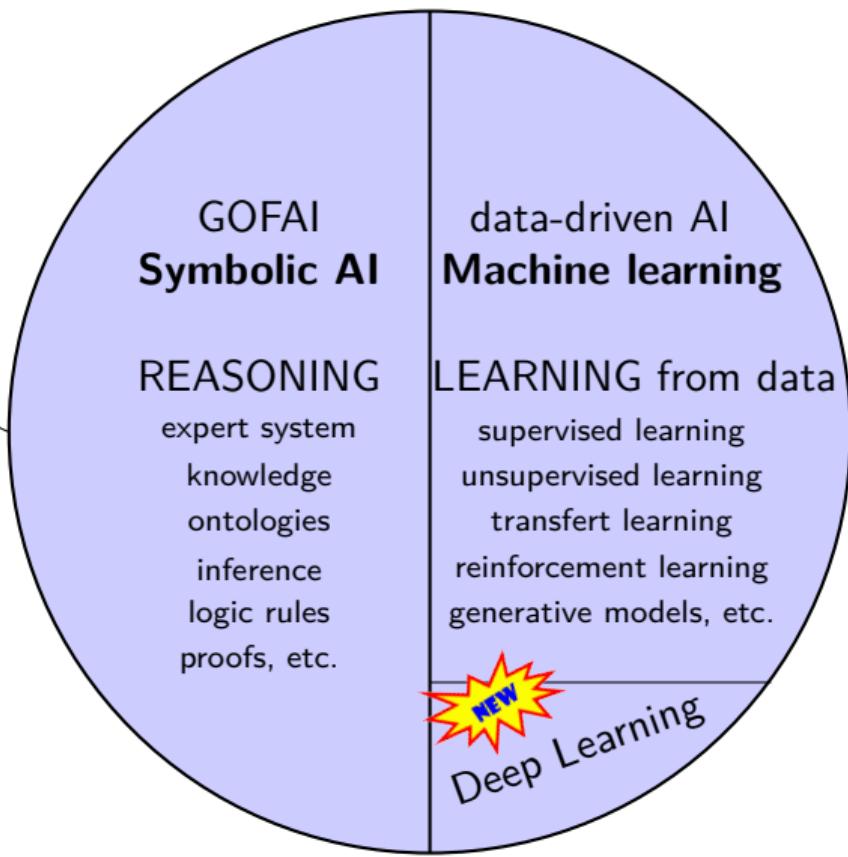


- Data allows to create models, to better understand and control future.

Representation of the research field of

Artificial Intelligence

Remark: the previous AI application examples were often a mix of GOFAI & Machine Learning.



Some remarks from AI leaders

- Yann LeCun: *Machines learn slower than we do.*



- Luc Julia: *It is not the intelligence that characterizes today's AI systems, but their ability to recognize through machine learning.*



Next challenges of AI

- Increase **trust** in AI: explainability, certification,
- **Speed** up learning,
- **Re-unite** GOFAI & Machine Learning,
- More generally, and as usual with AI, **reduce** the need for **human intervention**,
i.e. increase the autonomy of artificial agents.



Machine Learning

- Learning from data: dataset $d_n = \{x_i\}_{i=1}^n \in \mathcal{X}^n$, with $\dim(\mathcal{X}) = d$ (usually $\mathcal{X} = \mathbb{R}^d$). An element of d_n , $x_i \in \mathcal{X}$, is called a **sample**.

Matrix representation of the dataset: $d_n = \mathbb{X} = \begin{pmatrix} x_{1,1} & \dots & x_{1,d} \\ x_{2,1} & \dots & x_{2,d} \\ \vdots & & \vdots \\ x_{n,1} & \dots & x_{n,d} \end{pmatrix}.$

■ Supervised learning

- Settings: each sample x_i is associated with a target $y_i \in \mathcal{Y}$.
- Goal: compute a function (predictor) $c : \mathcal{X} \rightarrow \mathcal{Y}$ that predicts the target given a sample. Sample values are called **features**, and \mathcal{X} the feature space.
- Two main problems:
 - **classification** (\mathcal{Y} is finite, y_i are called **labels**),
 - and **regression** ($\mathcal{Y} = \mathbb{R}$).

Machine Learning

■ Unsupervised Learning

- Settings: samples are not associated with any target.
- Goal: anomaly/novelty detection, clustering, discovering structures in the data d_n .

■ Transfert Learning

- Settings: two different domains, *i.e.* two different couples (feature space, target space): source domain $(\mathcal{X}_S, \mathcal{Y}_S)$ and target domain $(\mathcal{X}_T, \mathcal{Y}_T)$.
- Goal: Use a dataset from source domain to help making predictions in the target domain.

■ Semi-supervised Learning

- Settings: the samples are not always associated with target.
- Goal: predict the target given a sample.

■ Self-supervised Learning

- Settings: the samples are associated with targets predicted by another predictor.
- Goal: predict the target given a sample.

Machine Learning

■ Reinforcement Learning

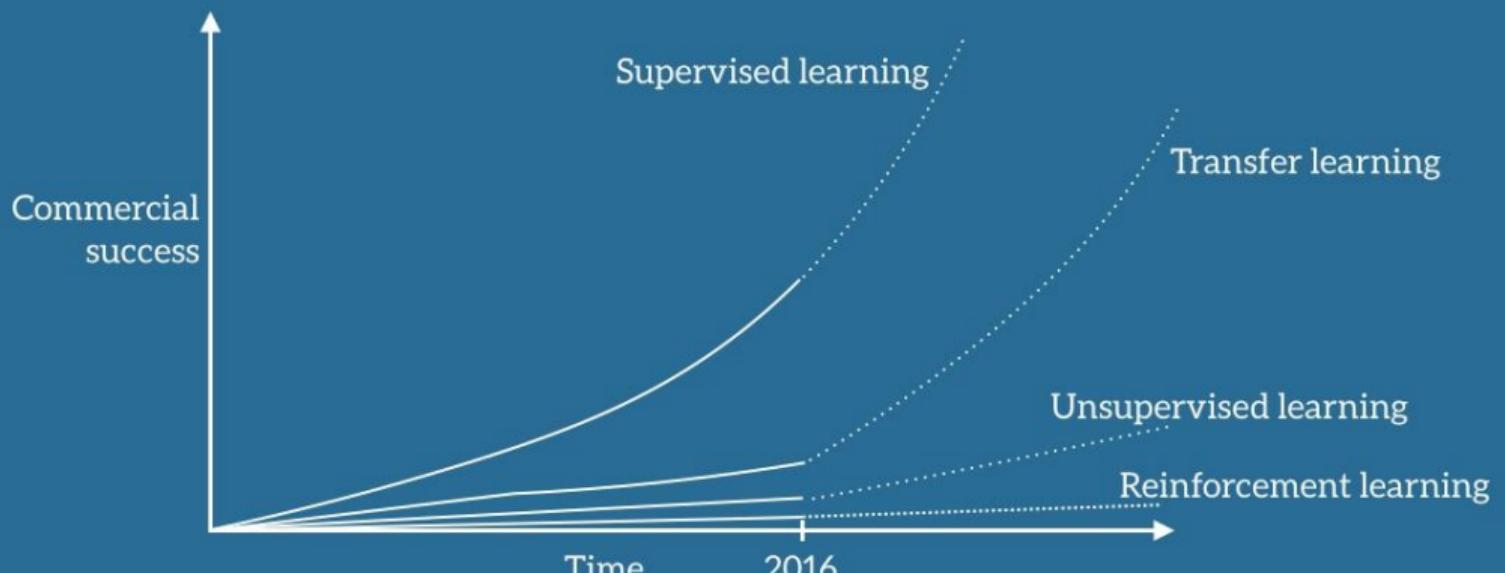
- Settings: environment simulator returning an observation $s_t \in \mathcal{S}$, and a reward $r_t \in \mathbb{R}$ for each action $a_t \in \mathcal{A}$ sent to it.
- Goal: compute an optimal strategy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ by interacting with the simulator, i.e. maximizing the expectation of the sum of future rewards, cf. class of Planning (C.Chanel) and Reinforcement Learning (N.Drougard).

■ Generative Models

- Settings: samples are not associated with any target.
- Goal: generation of samples that could be in the dataset d_n , i.e. that look like samples in the dataset.

In any of these Machine Learning problems, the computations on data, leading to desired outputs, are called **training**.

Drivers of ML success in industry



- Andrew Ng, NIPS 2016 tutorial

The Machine Learning library

- **Scikit-learn** (sklearn) is one of the most popular machine learning libraries.
- Written in python, designed to interoperate with NumPy and SciPy, first release in 2010.
- Sklearn defined the formalism
 - `model.fit(data)` to train on a dataset,
 - `model.predict(data)` to do prediction on new data,

used by most machine learning libraries (e.g. Deep Learning libraries pytorch, tensorflow, keras, etc.).



1 Introduction

- Artificial Intelligence, Machine Learning and Big Data?
- Basics in Probability Theory and Statistics

2 Supervised Learning

3 Neural Networks and Deep learning

4 Mathematical tools for Machine Learning

5 Risks

Random variable

- Random variable $X \in \mathcal{X}$, uncertainty model for each sample $x_i \in \mathcal{X}$.
- Distribution of the random variable defines the probability $\mathbb{P}(X \in A)$ for $A \subseteq \mathcal{X}$.
- Independant variables: X_1 and X_2 such that $\forall A, B \subseteq \mathcal{X}$,

$$\mathbb{P}((X_1, X_2) \in A \times B) = \mathbb{P}(X_1 \in A)\mathbb{P}(X_2 \in B).$$

Expectation and variance

- Expectation \mathbb{E} of a real-valued random variable X :
 - $\mathbb{E}[X] = \int_{\mathcal{X}} xf_X(x)dx$ if X is a continuous random variable,
 - $\mathbb{E}[X] = \sum_{x \in \mathcal{X}} x\mathbb{P}(X = x)$ if X is a discrete variable.
- \mathbb{E} is linear, i.e. $\mathbb{E}[aX_1 + bX_2] = a\mathbb{E}[X_1] + b\mathbb{E}[X_2]$, $\forall a, b \in \mathbb{R}$, X_1, X_2 random variables.
- Variance of a real-valued random variable X : $Var(X) = \mathbb{E}[(X - \mathbb{E}[X])^2]$.

Dataset distribution and likelihood

- To reason on the dataset before observation of it,
we also consider the random dataset $D_n = \{X_i\}_{i=1}^n$.
- If the samples are independent,
 - $f_{D_n}(d_n) = \prod_{i=1}^n f_{X_i}(x_i)$ if X continuous,
 - $p_{D_n}(d_n) = \prod_{i=1}^n p_{X_i}(x_i)$ if X discrete.
- When we assume that the variable distribution depends on parameters $\theta \in \Theta$
(e.g. $\theta = (\mu, \sigma)$ for $\mathcal{N}(\mu, \sigma^2)$, or $\theta = p$ for $Ber(p)$)
the Likelihood is the random variable $\mathcal{L}(\theta, X) = f_\theta(X)$, or $\mathcal{L}(\theta, X) = p_\theta(X)$ if discrete).
- $\mathcal{L}(\theta, D_n) = \prod_{i=1}^n \mathcal{L}(\theta, X_i)$ if variables X_i are independent.

Notations

- Random variable $X \sim P_X$ with values in \mathcal{X} .
- Random dataset $D_n = \{X_i\}_{i=1}^n$.
- Independant $X_i \Rightarrow D_n \sim P_X^n$.
- Consider a real random variable $\mathcal{X} = \mathbb{R}$.
- Cumulative Distribution Function (CDF): $F_X(t) = \mathbb{P}(X \leq t)$.
- Empirical CDF: $F_{X,n}(t) = \frac{1}{n} \sum_{i=1}^n \mathbb{1}_{\{X_i \leq t\}}$.
- $\forall t \in \mathbb{R}, F_{X,n}(t) \xrightarrow[n \rightarrow +\infty]{a.s.} F_X(t)$ (Strong Law of Large Numbers SLLN, see Theorem 3.1 [GS20]).
- Quantile function: $F_X^{-1}(s) = \inf \{t \in \mathbb{R} \mid F(t) \geq s\}$.

1 Introduction

2 Supervised Learning

- Evaluation of a prediction function.
- Method and algorithms

3 Neural Networks and Deep learning

4 Mathematical tools for Machine Learning

5 Risks

- **Input (features)**: X (random) or x (realization).
- **Output (targets)**: Y (random) or y (realization).
- **Random dataset**: $D_n = (X_i, Y_i)_{i=1}^n \in (\mathcal{X} \times \mathcal{Y})^n$
- usually, features $\mathcal{X} = \mathbb{R}^d$
- target for **classification**: $y \in \mathcal{Y} = \{y_i\}_{i=1}^m$. Binary classification: $\{y_+, y_-\}$ ($m = 2$).
- target for **regression**: $y \in \mathcal{Y} = \mathbb{R}$.
- **Classifier/regressor**: function $c : \mathcal{X} \rightarrow \mathcal{Y}$.
- **Dataset**, samples, realization of D_n : $d_n = (x_i, y_i)_{i=1}^n \in (\mathcal{X} \times \mathcal{Y})^n$.
- **Prediction**: $\hat{y}_i = c(x_i)$.

How to evaluate a Classification/Regression?

“Good classifiers” have a high score and low loss/error.

- Classification accuracy $\frac{1}{n} \sum_{i=1}^n \mathbf{1}_{\{\hat{y}_i = y_i\}} = 1 - \text{error rate} \approx \mathbb{E} [\mathbf{1}_{\{Y=c(X)\}}]$
 - ▶ sklearn: `metrics.accuracy_score(y_true, y_pred)`
 - & error rate = zero-one classification loss
 - ▶ sklearn: `metrics.zero_one_loss(y_true, y_pred)`
- Regression: mean squared error $\frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2$
 - ▶ sklearn: `metrics.mean_squared_error(y_true, y_pred)`

Some famous loss functions

- Hinge loss (used for SVM algorithm): $L_{Hinge}(y, v) = \max(0, 1 - sgn(y) \times v(x))$ with $sgn(y_+) = 1$, $sgn(y_-) = -1$, and the **decision function** $v(x) \in \mathbb{R}$ defining prediction by $\hat{y} = c(x) = \begin{cases} y_+ & v(x) > 0 \\ y_- & \text{otherwise} \end{cases}$.
A missclassification means $v(x)$ and $sgn(y)$ have opposite signs $\Rightarrow L_{Hinge}(y, v) \geq 1$:
 $\frac{1}{n} \sum_{i=1}^n L_{Hinge}(y_i, v(x_i)) \approx \mathbb{E}[L_{Hinge}(Y, v(X))]$ is an upper bound of the accuracy.
► `sklearn: metrics.hinge_loss(y_true, pred_decision)`
- Log (logistic, or cross-entropy) loss (used for logistic regression and neural networks):
 $L_{log}(y, \eta) = -\left(y \log(\eta(x)) + (1 - y) \log(1 - \eta(x))\right)$
with $y_+ = 1$, $y_- = -1$, and the **posterior probability function** $\eta(x)$ defining prediction by $\hat{y} = c(x) = \begin{cases} y_+ & \eta(x) > \frac{1}{2} \\ y_- & \text{otherwise} \end{cases}$. Opposite of the log-likelihood of the Bernoulli distribution with parameter $\eta(x) = \mathbb{P}(Y = y_+ | X = x)$ we want η to maximize.
► `sklearn: metrics.log_loss(y_true, y_pred)`

Binary classification (2-class)

- $\mathcal{Y} = \{y_+, y_-\}$
- Samples with $y = y_+$ are called positive: $Pos = \sum_{i=1}^n \mathbb{1}_{\{y_i=y_A\}}$.
- Samples with $y = y_-$ are called negative: $Neg = \sum_{i=1}^n \mathbb{1}_{\{y_i=y_B\}} = n - Pos$.

The accuracy of c that returns always y_+ (resp. y_-) is $\frac{Pos}{n}$ (resp. $\frac{Neg}{n}$).

Unbalanced dataset with $Pos > Neg$ (resp. $Pos < Neg$), $\Rightarrow c$'s accuracy can be good!!

E.g. if $Pos = 9 \times Neg$, the accuracy of $c(x) \stackrel{\text{const.}}{=} y_+$ is $\frac{Pos}{10 \times Neg} = \frac{9}{10} = 90\%$...

The balanced accuracy take this issue into account:

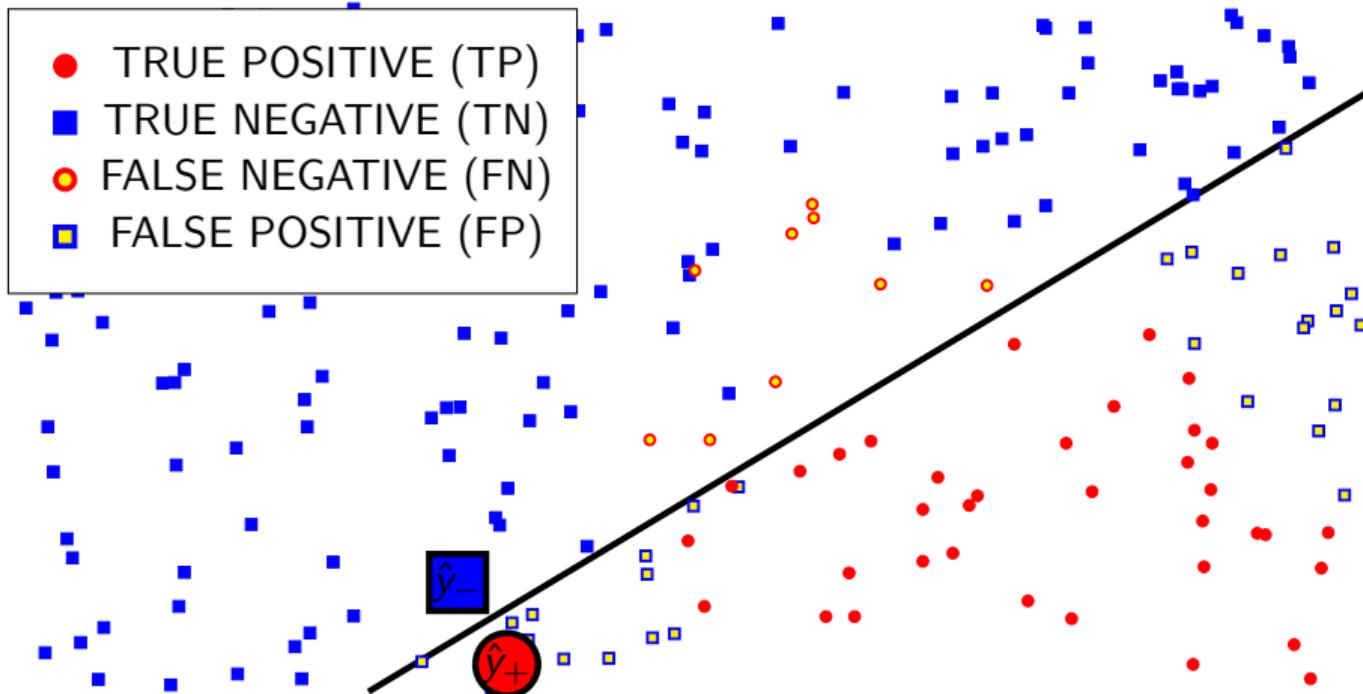
Balanced accuracy

$$\frac{1}{2} \left(\frac{1}{Pos} \sum_{i=1}^n \mathbb{1}_{\{\hat{y}_i=y_i=y_+\}} + \frac{1}{Neg} \sum_{i=1}^n \mathbb{1}_{\{\hat{y}_i=y_i=y_-\}} \right) = \frac{1}{2} \left(\frac{TP}{Pos} + \frac{TN}{Neg} \right) = \frac{1}{2} (TPR + TNR)$$

with $TP = \sum_{i=1}^n \mathbb{1}_{\{\hat{y}_i=y_i=y_+\}}$ the number of "True Positive", (TPR for "True Positive Rate"),
 $TN = \sum_{i=1}^n \mathbb{1}_{\{\hat{y}_i=y_i=y_-\}}$ the number of "True Negative", (TNR for "True Negative Rate").

► `sklearn.metrics.balanced_accuracy_score(y_true, y_pred)`

Binary classification (2-class)



Terminology: conditioning by reality y (samples)

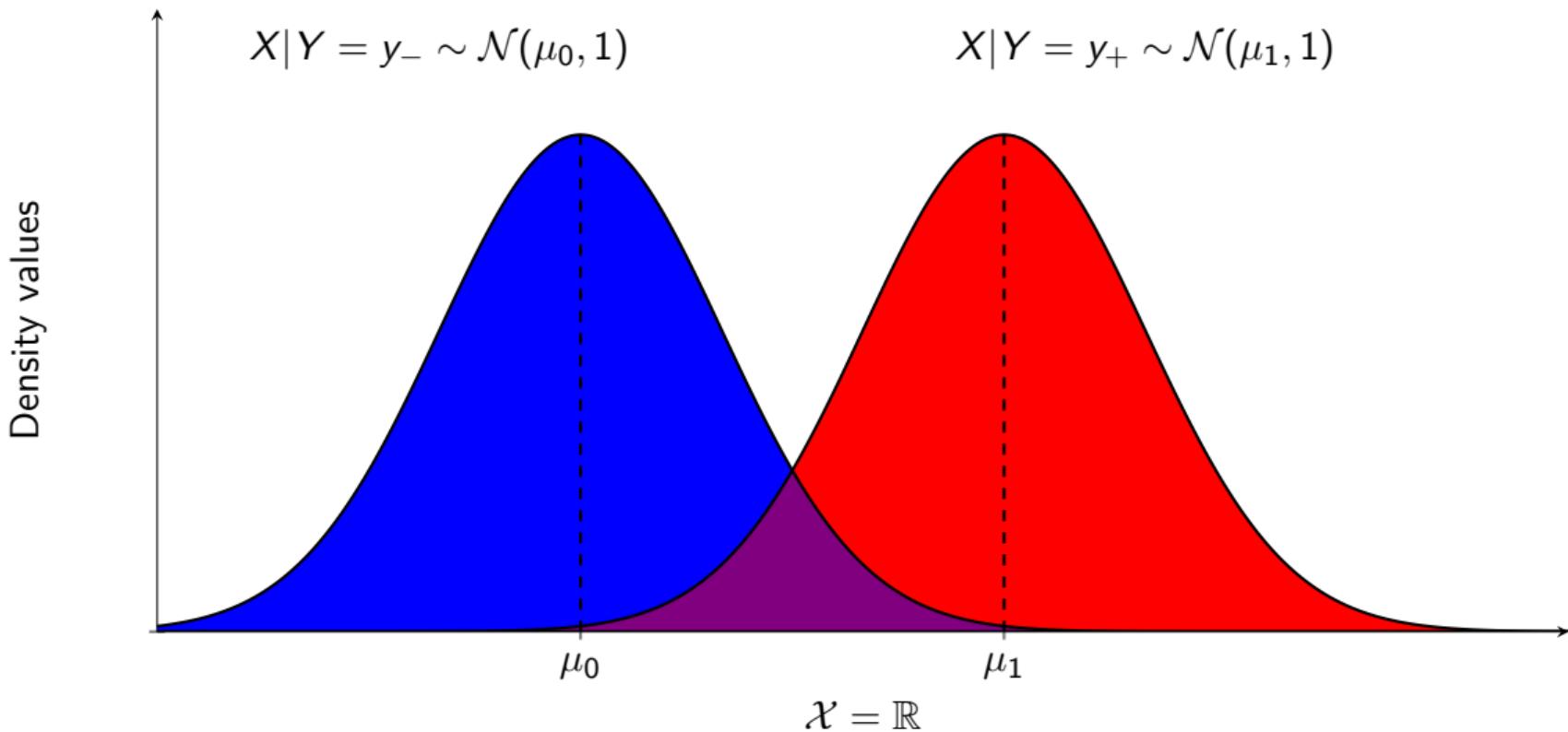
Proportions among

NEGATIVE samples y_- ■	POSITIVE samples y_+ ●
$TNR = \text{True Negative Rate}$ $= \text{specificity/selectivity}$ $= \frac{TN}{Neg} = \frac{\sum_{i=1}^n \mathbb{1}_{\{y_i=\hat{y}_i=y_-\}}}{\sum_{i=1}^n \mathbb{1}_{\{y_i=y_-\}}}$ $= 1 - \frac{FP}{Neg} = 1 - FPR$ $= 1 - \text{False Pos. Rate (or fall-out)}$	$TPR = \text{True Positive Rate}$ $= \text{sensitivity/recall/hit rate}$ $= \frac{TP}{Pos} = \frac{\sum_{i=1}^n \mathbb{1}_{\{y_i=\hat{y}_i=y_+\}}}{\sum_{i=1}^n \mathbb{1}_{\{y_i=y_+\}}}$ $= 1 - \frac{FN}{Pos} = 1 - FNR$ $= 1 - \text{False Neg. Rate (or miss rate)}$

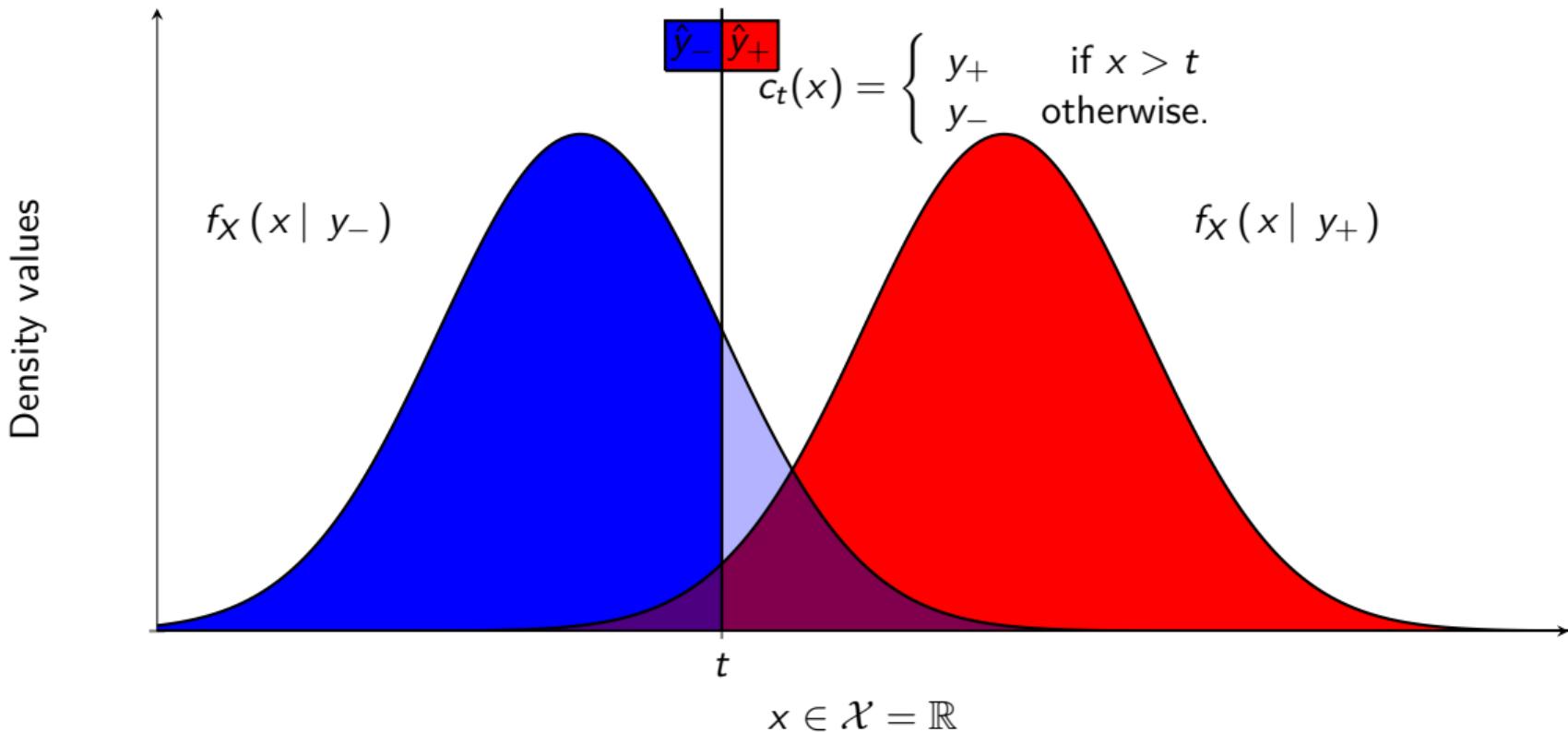
Links with statistical tests

H_0	H_1
$\approx p(\hat{y}_- y_-) \stackrel{\text{def.}}{=} \mathbb{P}(Y = y_- c(X) = y_-)$ $= 1 - p(\hat{y}_+ y_-)$ $= 1 - \alpha (\alpha = \text{type I error})$	$\approx p(\hat{y}_+ y_+) = \gamma (\text{power})$ $= 1 - p(\hat{y}_- y_+)$ $= 1 - \beta (\beta = \text{type II error})$

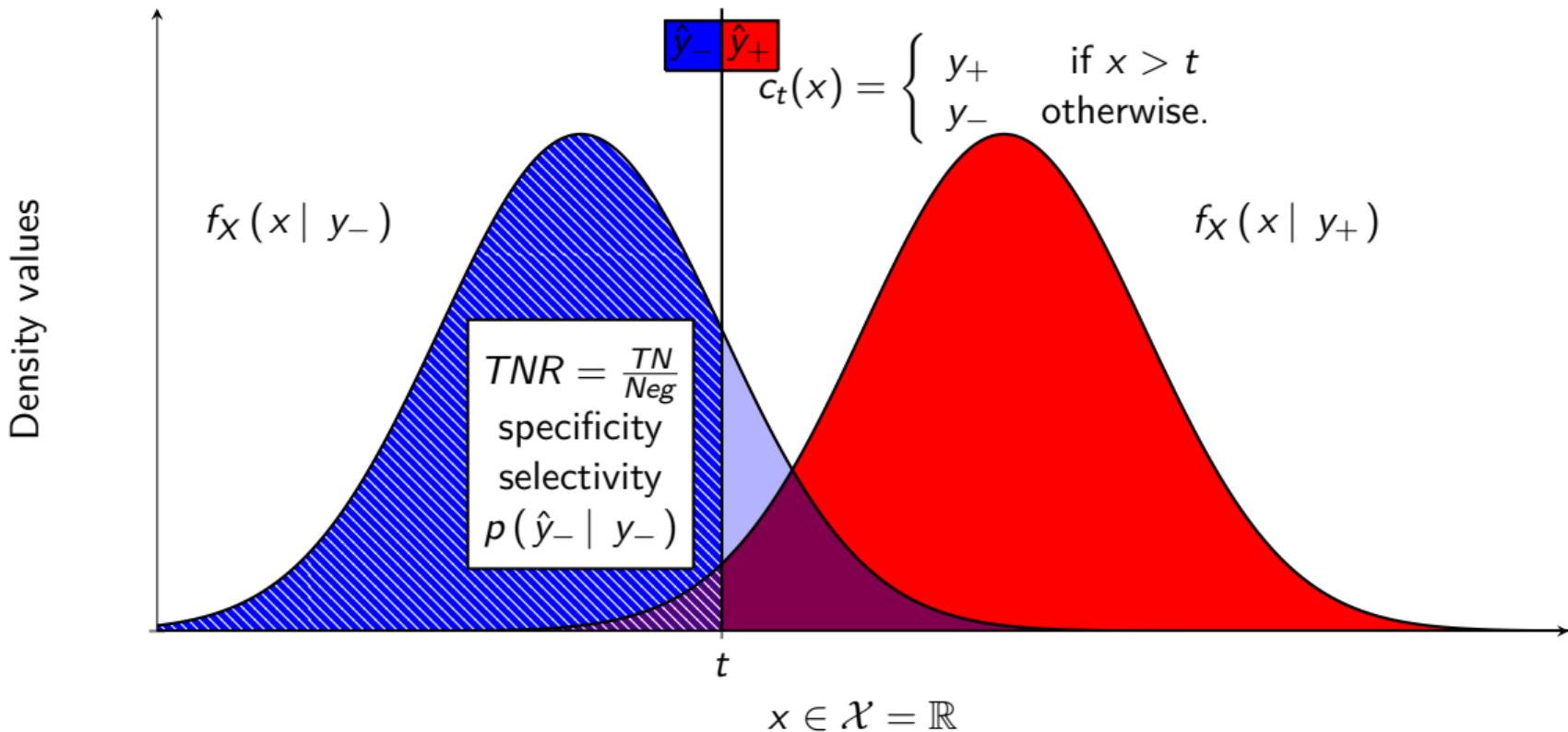
Visualization with densities



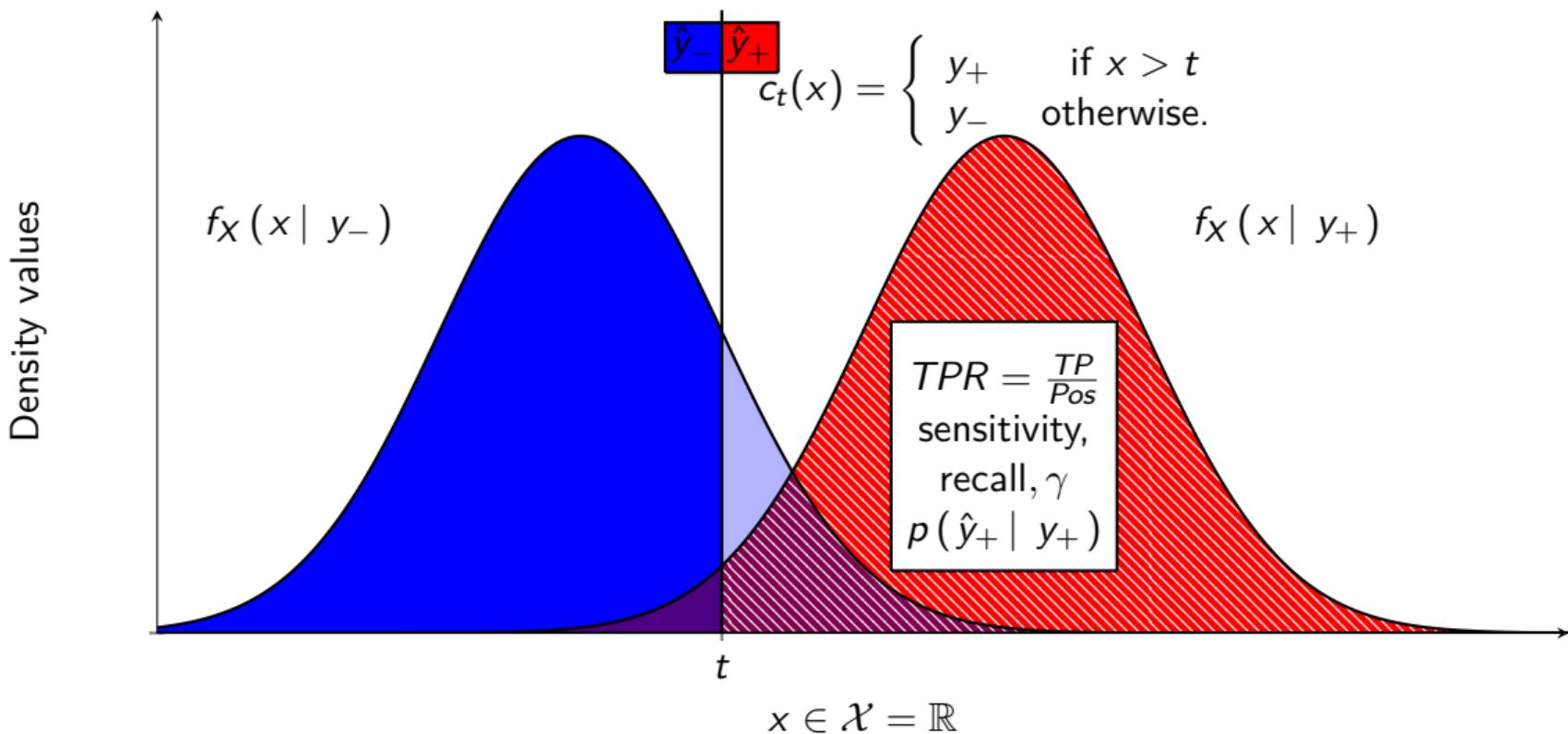
Visualization with densities



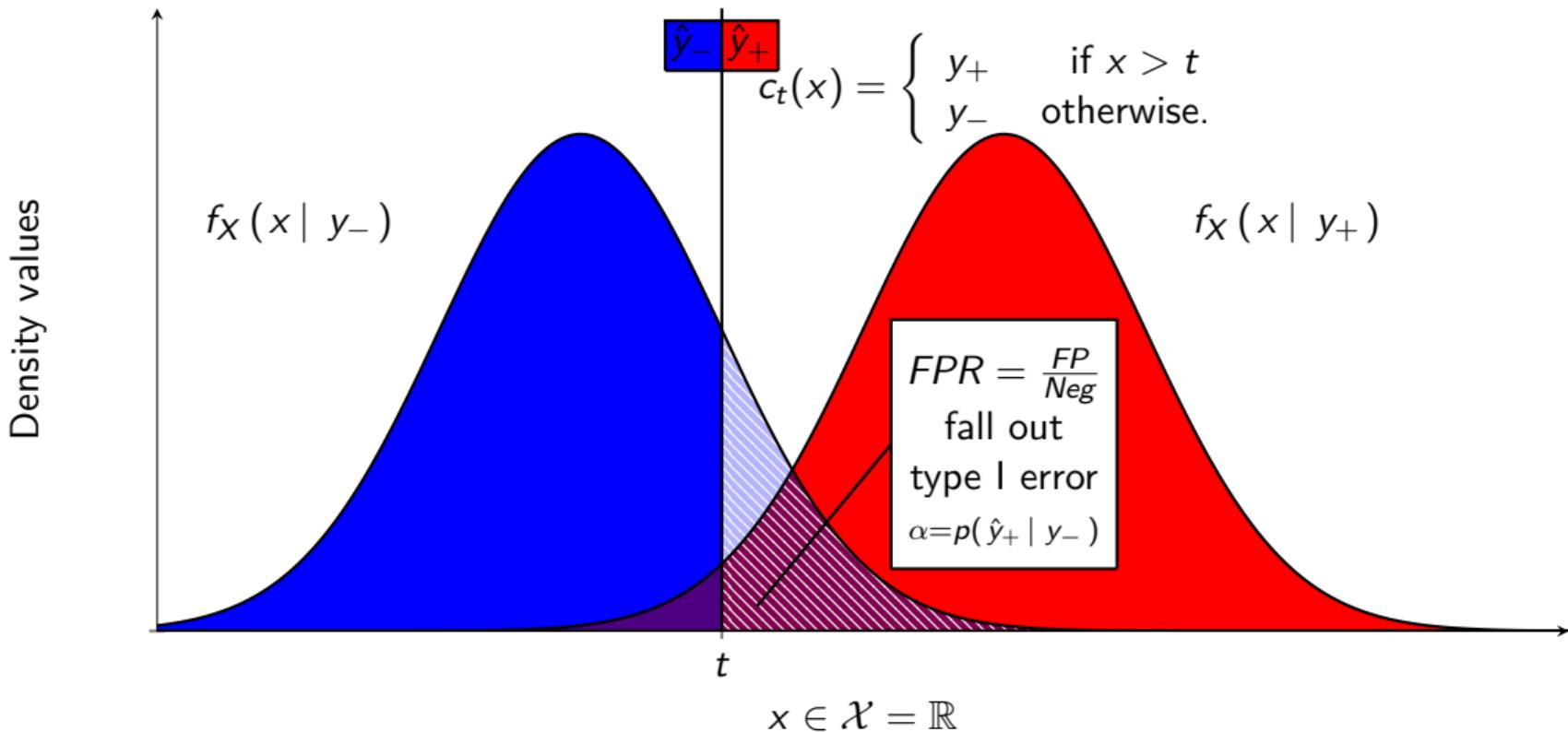
Visualization with densities



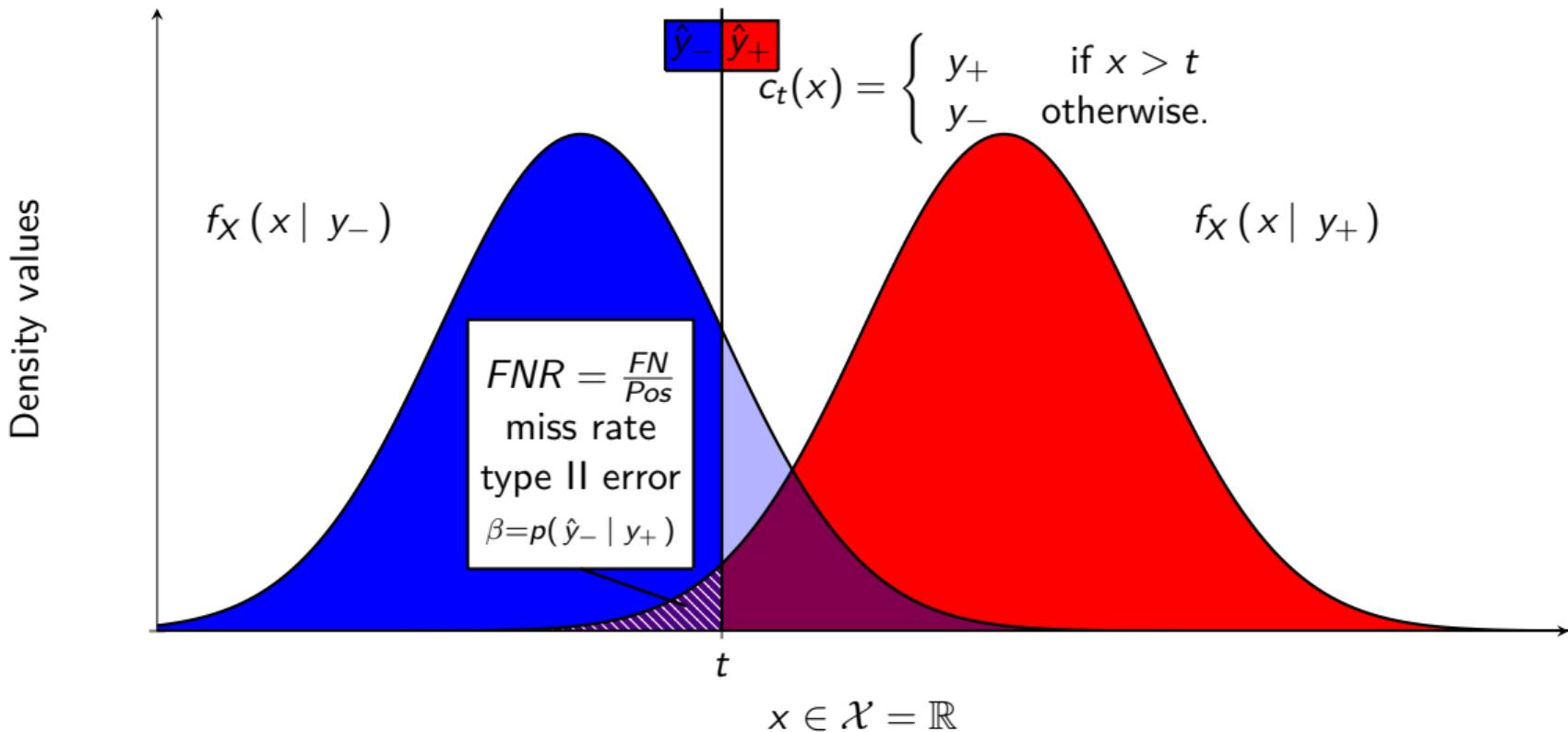
Visualization with densities



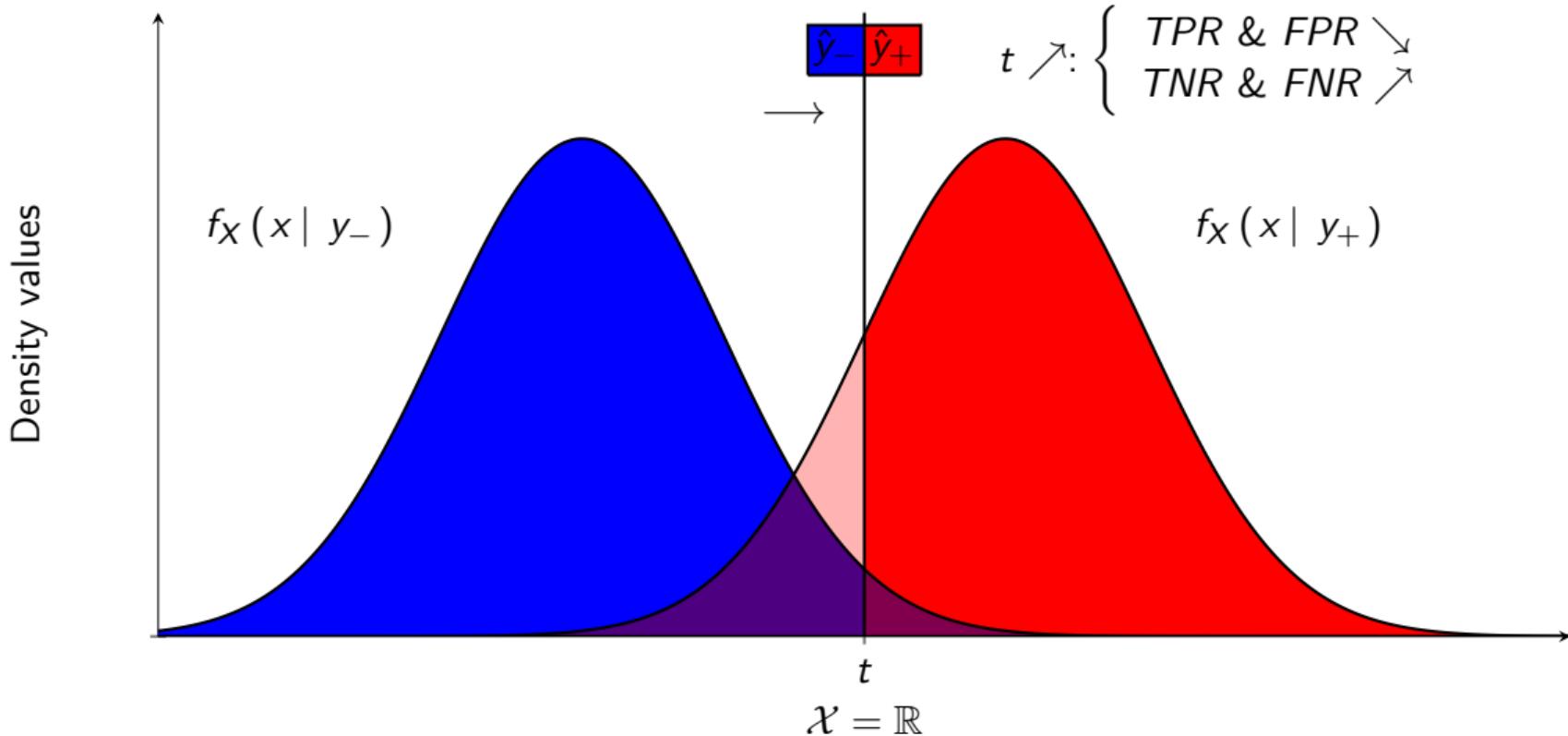
Visualization with densities



Visualization with densities



Visualization with densities



Consider a classifier $c_t(x) = \begin{cases} y_+ & \text{if } s(x) > t \\ y_- & \text{otherwise.} \end{cases}$ with $s : \mathcal{X} \rightarrow \mathbb{R}$ (called score) and $t \in \mathbb{R}$.

For instance, $s(x)$ can be an approximation of $\mathbb{P}(Y = y_+ | X = x) = \eta(x)$.

Note that when $t \nearrow$,

- $TP(t) \searrow, \lim_{t \rightarrow -\infty} TP(t) = Pos, \lim_{t \rightarrow +\infty} TP(t) = 0.$
- $FP(t) \nearrow, \lim_{t \rightarrow -\infty} FP(t) = Neg, \lim_{t \rightarrow +\infty} FP(t) = 0.$

ROC curve

Receiver operating characteristic (ROC): $(FPR(t), TPR(t))$

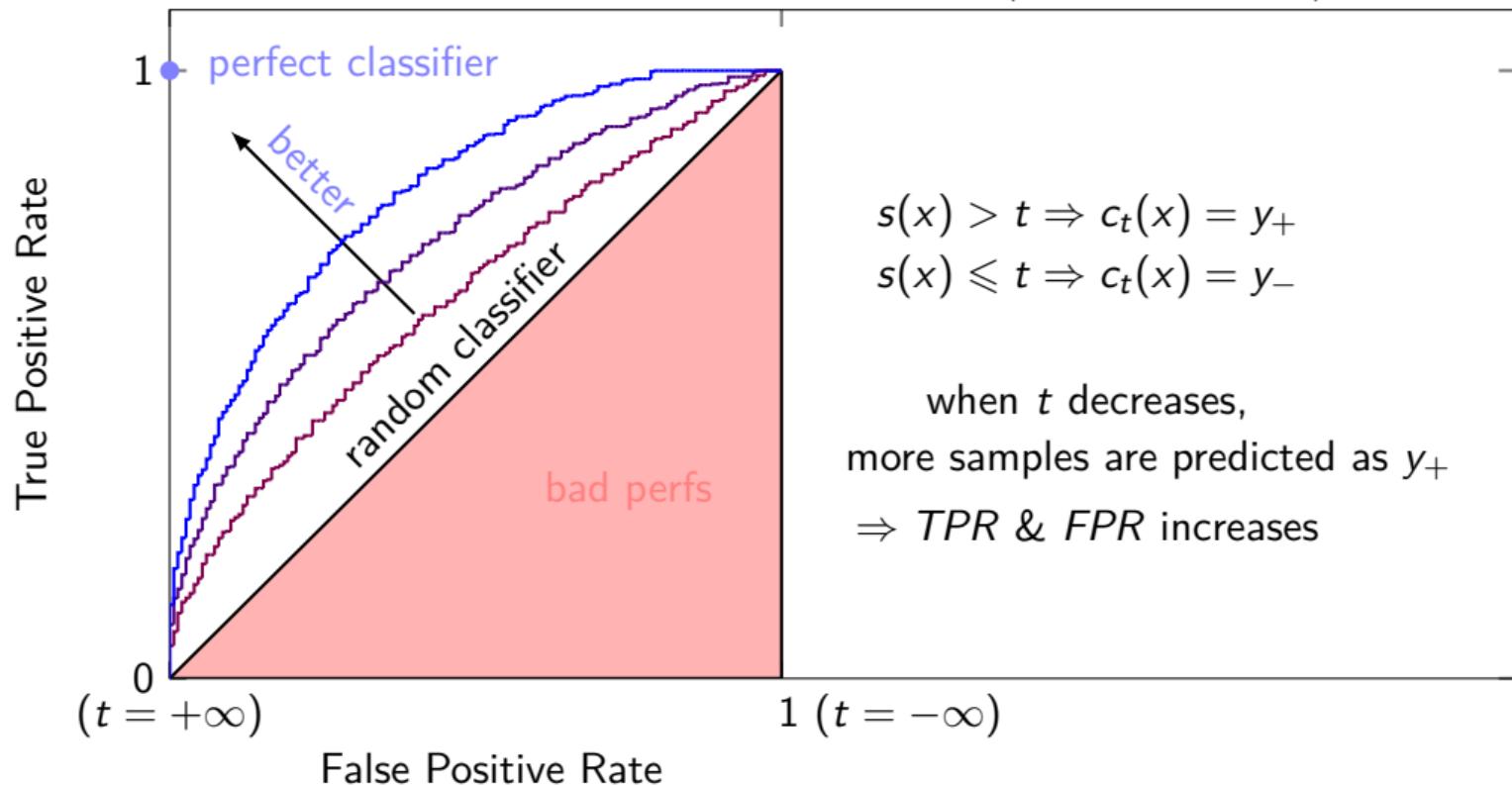
► `sklearn: metrics.roc_curve(y_true, y_score)`

Resulting performance metric: Area Under the ROC Curve (AUC).

► `sklearn: metrics.roc_auc_score(y_true, y_score)`

Evaluation of a prediction function.

Receiver operating characteristic (ROC): $(FPR(t), TPR(t))$



Terminology: conditioning by prediction \hat{y}

Proportions among

NEGATIVE predictions \hat{y}_- (above line)	POSITIVE predictions \hat{y}_+ (below line)
$NPV = \text{Negative Predictive Value}$ $= \frac{TN}{TN + FN} = \frac{\sum_{i=1}^n \mathbb{1}_{\{y_i = \hat{y}_i = y_-\}}}{\sum_{i=1}^n \mathbb{1}_{\{\hat{y}_i = y_-\}}}$ $\approx p(y_- \hat{y}_-)$ $\stackrel{\text{def.}}{=} \mathbb{P}(c(X) = y_- Y = y_-)$ $= 1 - \frac{FN}{TN + FN}$ $= 1 - FOR$ (for “ False Omission Rate ”)	$PPV = \text{Positive Predictive Value}$ = precision $= \frac{TP}{TP + FP} = \frac{\sum_{i=1}^n \mathbb{1}_{\{y_i = \hat{y}_i = y_+\}}}{\sum_{i=1}^n \mathbb{1}_{\{\hat{y}_i = y_+\}}}$ $\approx p(y_+ \hat{y}_+)$ $\stackrel{\text{def.}}{=} \mathbb{P}(c(X) = y_+ Y = y_+)$ $= 1 - \frac{FP}{TP + FP}$ $= 1 - FDR$ (for “ False Discovery Rate ”)

► `sklearn: metrics.precision_score(y_true, y_pred)`

Consider again $c_t(x) = \begin{cases} y_+ & \text{if } s(x) > t \\ y_- & \text{otherwise.} \end{cases}$ with $s : \mathcal{X} \rightarrow \mathbb{R}$ (called score) and $t \in \mathbb{R}$.

Note that when $t \searrow$, positive predictions $TP(t) + FP(t) \nearrow$.

The precision $PPV(t) = \frac{TP}{TP+FP}$ $\begin{cases} \bullet & \text{increases when } TP(t) \nearrow \\ \bullet & \text{decreases when } FP(t) \nearrow. \end{cases}$

Recall that when $t \searrow$ $TP(t) \nearrow$, $\lim_{t \rightarrow -\infty} TP(t) = Pos$, $\lim_{t \rightarrow +\infty} TP(t) = 0$.

Precision-Recall (PR) curve

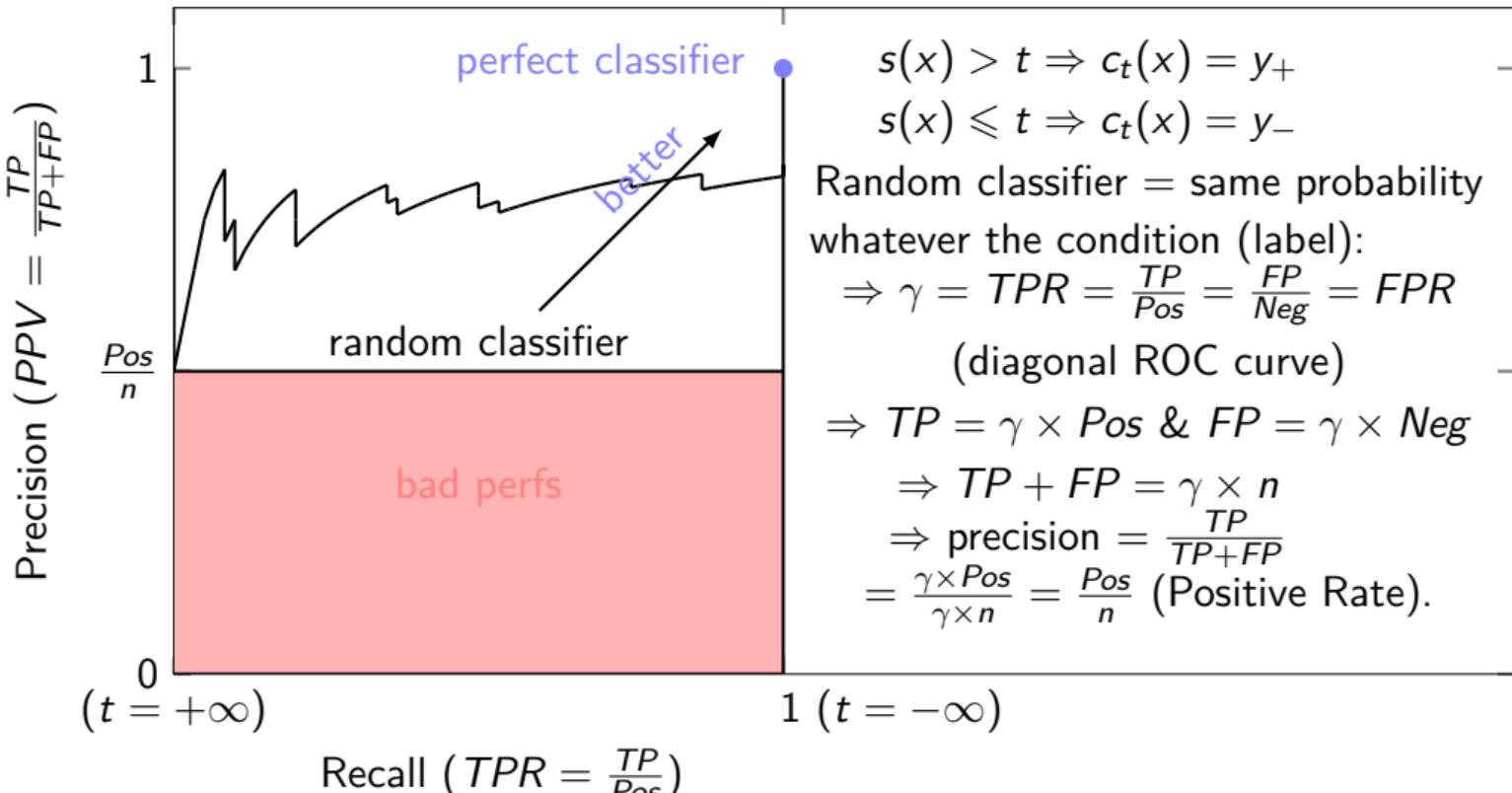
$t \mapsto (TPR(t), PPV(t))$. The y -axis of the ROC curve is now the x -axis.

The y -axis becomes the precision $PPV = \frac{TP}{TP+FP}$.

- ▶ `sklearn: metrics.precision_recall_curve(y_true, probas_pred)`
- ▶ `sklearn: metrics.recall_score(y_true, y_pred)`

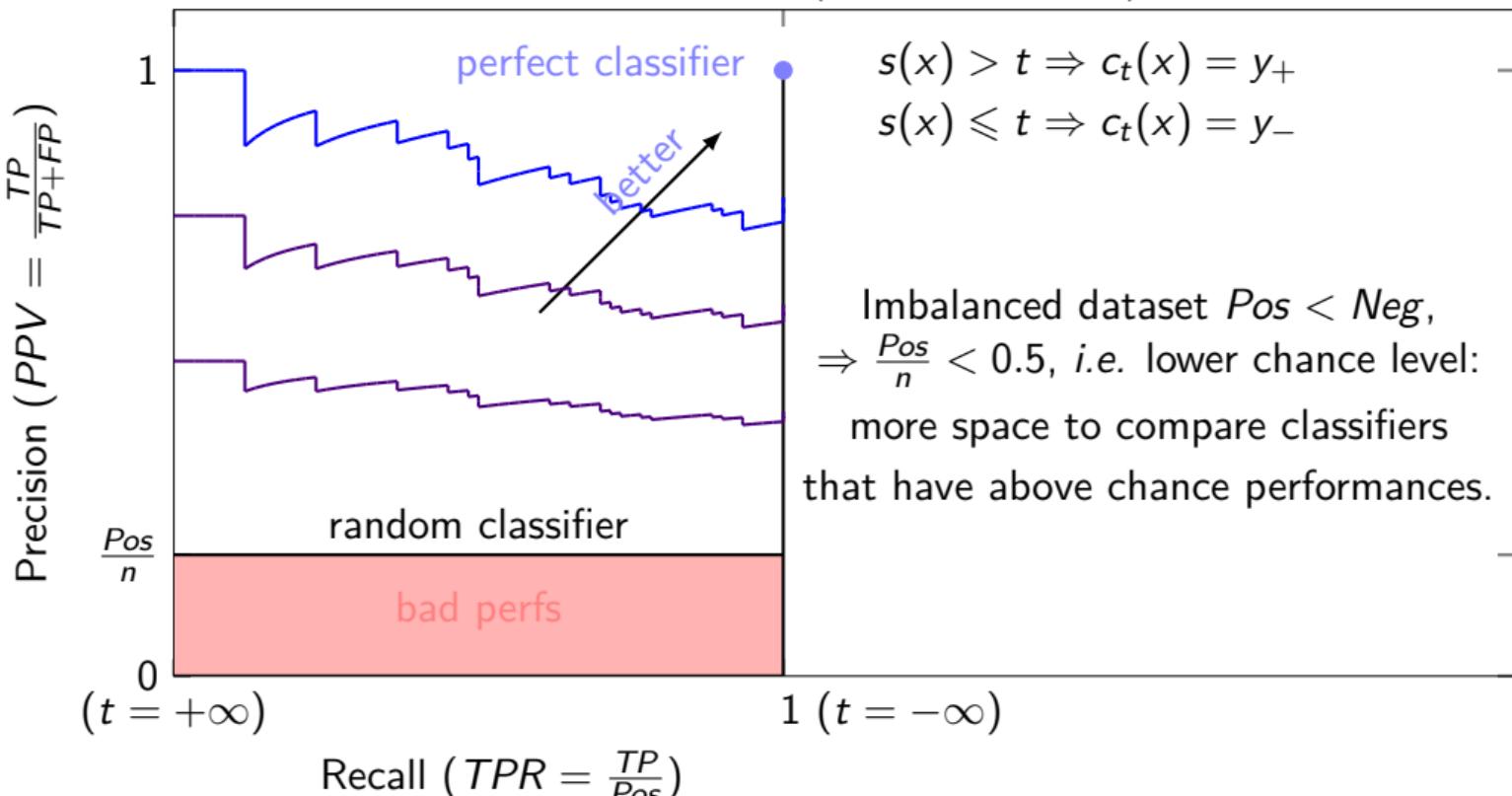
Evaluation of a prediction function.

Precision Recall (PR): $(TPR(t), PPV(t))$



Evaluation of a prediction function.

Precision Recall (PR): $(TPR(t), PPV(t))$



Resulting performance metrics

- Area Under the Curve (AUC) just like for the ROC curve:
 - ▶ `sklearn: metrics.auc(x,y)`
- Average precision: $AP = \sum_{i=1}^n (R_{i+1} - R_i)P_{i+1}$
with $(R_i)_{i=1}^{n+1}$ the non-decreasing values of Recall $\in \left\{ \frac{i}{Pos} \mid i \in \{1, \dots, Pos\} \right\}$ when $t \searrow$,
and P_n the associated Precision values. Same idea as AUC, but less optimistic:
 - ▶ `sklearn: metrics.average_precision_score(y_true)`
- F-measure or balanced F-score (F1 score)
= harmonic mean of precision and recall: $2 \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$
 - ▶ `sklearn: metrics.f1_score(y_true, y_pred)`
 - ▶ `sklearn: metrics.precision_recall_fscore_support`
- statistics on the accuracy distribution, and many other metrics:
 - ▶ `sklearn: Classification metrics`

- **Confusion matrix** $\begin{pmatrix} TPR & FPR \\ FNR & TNR \end{pmatrix}$ with T/F = True/False, P/N = Positive/Negative and R = Rate.
- Generalization for multi-class labels $\mathcal{Y} = \{\lambda_1, \dots, \lambda_m\}$: confusion matrix $\forall (i, j) \in \mathcal{Y}$,
$$M_{i,j} = \hat{p}(\hat{y} = \lambda_i \mid y = \lambda_j) = \frac{\sum_{k=1}^n \mathbb{1}_{\{\hat{y}_k = \lambda_i, y_k = \lambda_j\}}}{\sum_{k=1}^n \mathbb{1}_{\{y_k = \lambda_j\}}}.$$
 - ▶ `sklearn: metrics.confusion_matrix(y_true, y_pred)`

1 Introduction

2 Supervised Learning

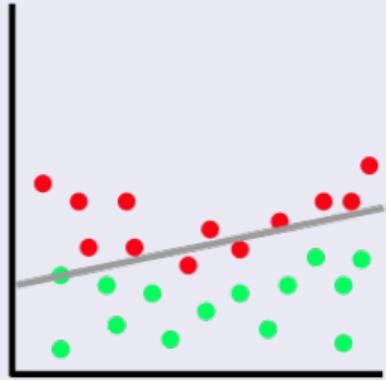
- Evaluation of a prediction function.
- Method and algorithms

3 Neural Networks and Deep learning

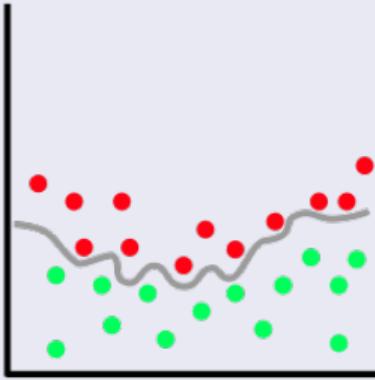
4 Mathematical tools for Machine Learning

5 Risks

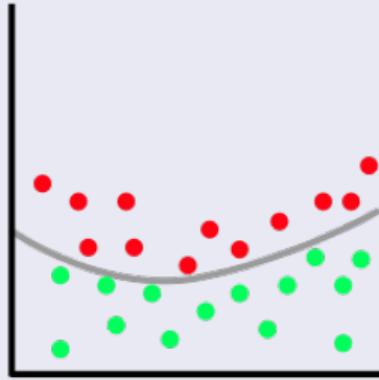
Overfitting/Underfitting: classification



Underfitting

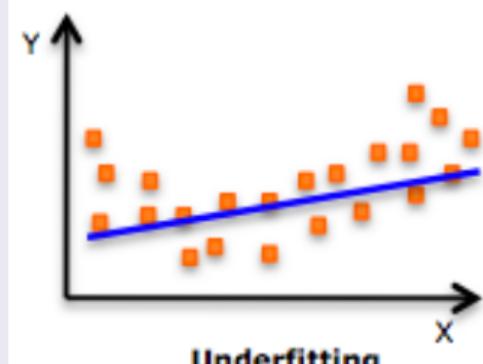


Overfitting

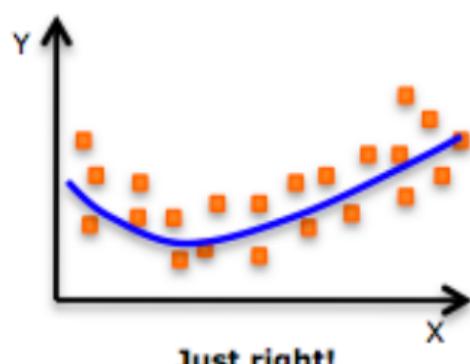


Balanced

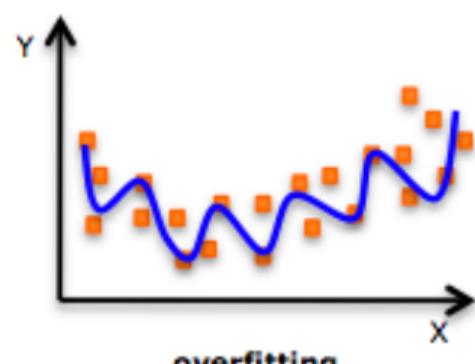
Overfitting/Underfitting: regression



Underfitting

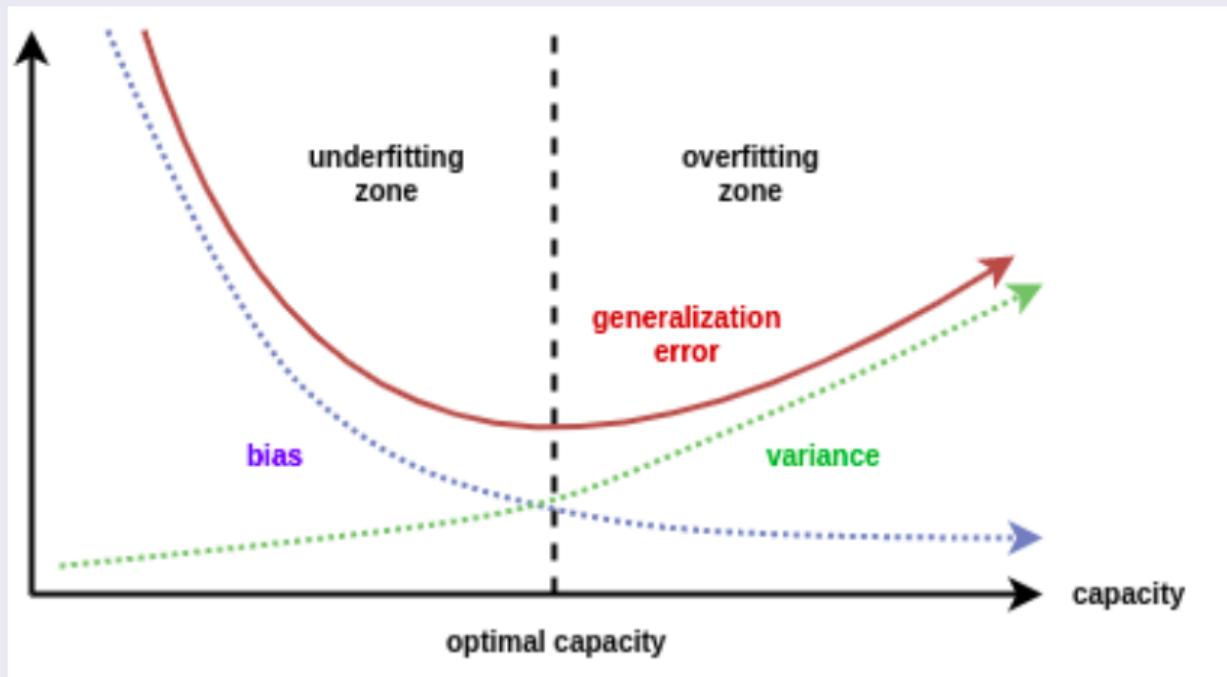


Just right!

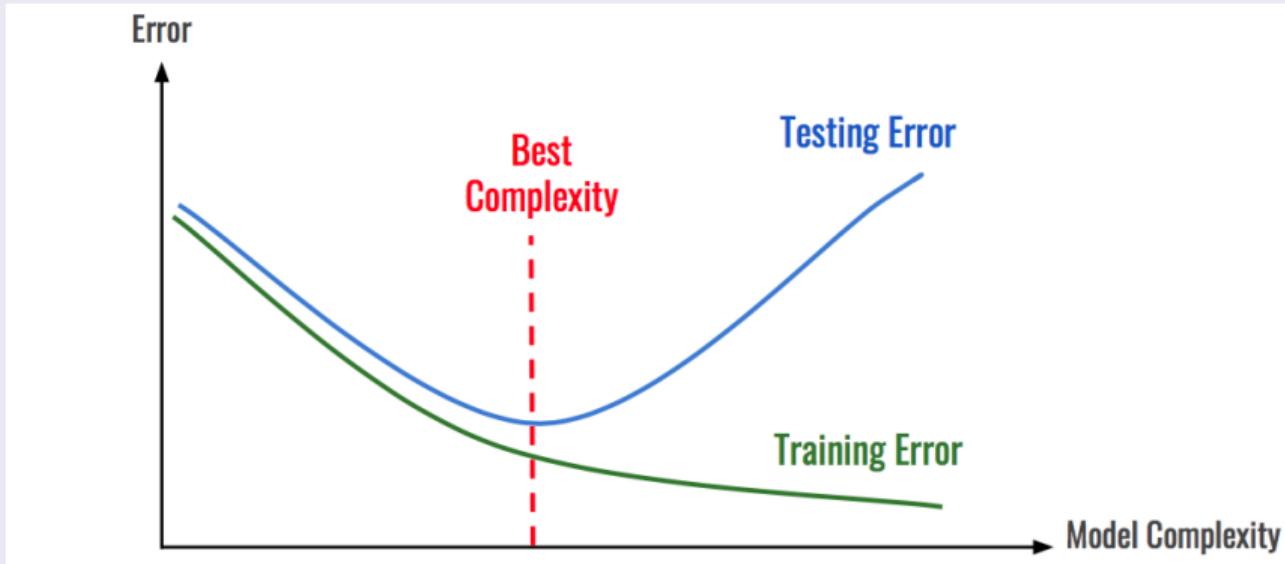


overfitting

Overfitting/Underfitting



Overfitting/Underfitting



Overfitting/Underfitting

Overfitting is avoided by

- penalizing the model complexity (typically the number of parameters)
- if training is sequential, stop when the error on the **testing set** increases.

General process

- training (optimization)
- validation (meta optimization, cross-validation)
- testing (final evaluation)

DATASET d_n

Training set (d_{train}) compute c with algorithm \mathcal{A} for some meta-parameters $m \in \mathcal{M}$: $c_m = \mathcal{A}(d_{train}, m), c_m : \mathcal{X} \rightarrow \mathcal{Y}$.	Validation set (d_{val}) Evaluate c_m on d_{val} : $error_m = loss(c_m, d_{val})$.	Testing set (d_{test}) for final evaluation only: $error_{m^*} = loss(c_{m^*}, d_{test})$ with $c_{m^*} = \mathcal{A}(d_{train} + d_{val}, m^*)$
	Cross-validation	
best metaparameter $m^* \in \mathcal{C}$ according to the average error		

Linear Regression

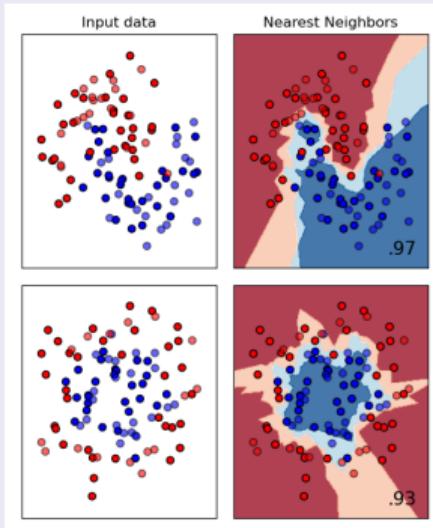
General Linear Model:

$$Y = \mathbb{X}\theta = \begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{pmatrix} = \begin{pmatrix} x_1^{(1)} & \dots & x_1^{(d)} \\ x_2^{(1)} & \dots & x_2^{(d)} \\ \vdots & & \vdots \\ x_n^{(1)} & \dots & x_n^{(d)} \end{pmatrix} \begin{pmatrix} \theta_1 \\ \vdots \\ \theta_d \end{pmatrix} + \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}$$

- the MLE is the linear least square parameters $\widehat{\theta}_n = (\mathbb{X}^T \mathbb{X})^{-1} \mathbb{X}^T Y$.
- Given the target $y \in \mathbb{R}^n$, the regression is $c(x) = x^T \widehat{\theta}_n$ with $\widehat{\theta}_n = (\mathbb{X}^T \mathbb{X})^{-1} \mathbb{X}^T y$, for feature vector $x \in \mathcal{X}$.
 - ▶ `sklearn: linear_model.LinearRegression`

k nearest neighbors (k -NN)

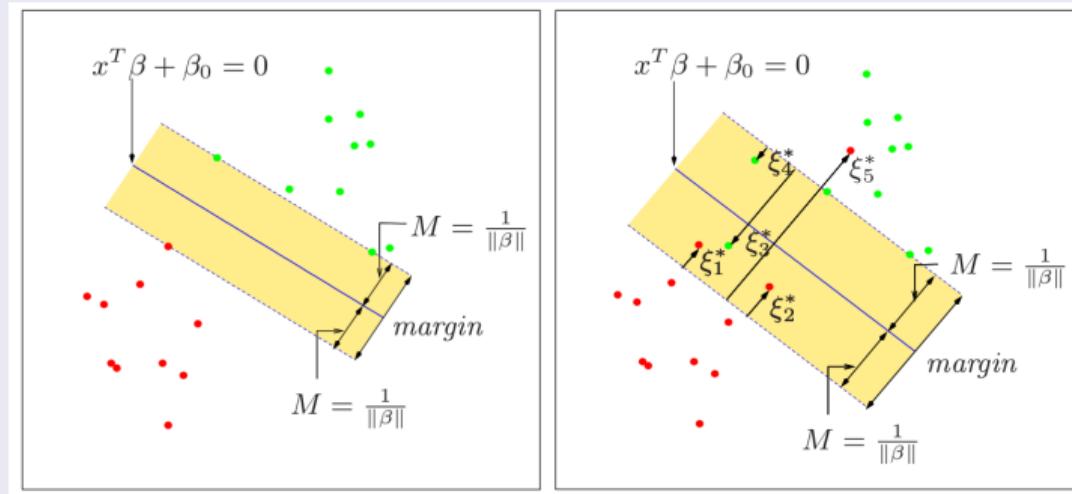
- Let x be a feature vector. We want to predict the associated label y .
- Let $N_k(x)$ be the k closest feature vectors in the dataset
- Prediction: $c(x) = \arg \max_y \# \{y_i \mid x_i \in N_k(x)\}$.



► `sklearn: neighbors.KNeighborsClassifier`

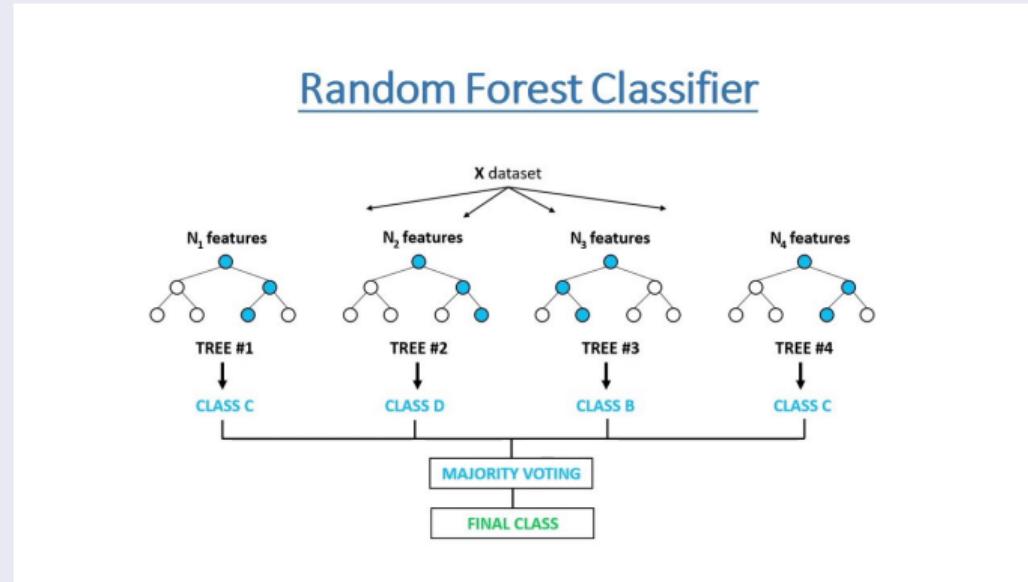
Support Vector Machine (SVM)

- Maximizing the margin



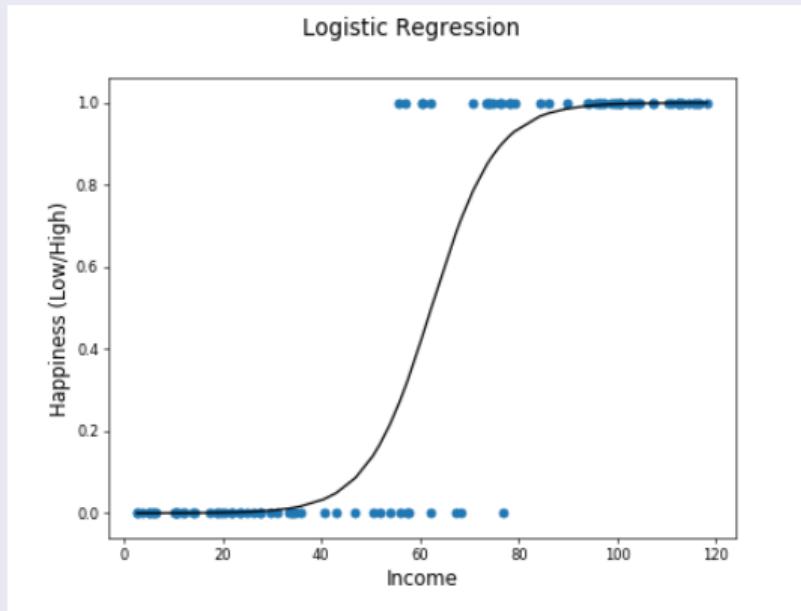
► `sklearn: svm.SVC`

Random Forest (RF)



► `sklearn: ensemble.RandomForestClassifier`

Logistic Regression



► `sklearn: linear_model.LogisticRegression`

1 Introduction

2 Supervised Learning

3 Neural Networks and Deep learning

4 Mathematical tools for Machine Learning

5 Risks

In the past few years, artificial intelligence (AI) has been a subject of intense media hype. Machine learning, deep learning, and AI come up in countless articles, often outside of technology-minded publications. We're promised a future of intelligent chatbots, self-driving cars, and virtual assistants—a future sometimes painted in a grim light and other times as utopian, where human jobs will be scarce and most economic activity will be handled by robots or AI agents. For a future or current practitioner of machine learning, it's important to be able to recognize the signal in the noise so that you can tell world-changing developments from overhyped press releases. [F.C18]

Single layer perceptron a.k.a. *single hidden layer back-propagation network*

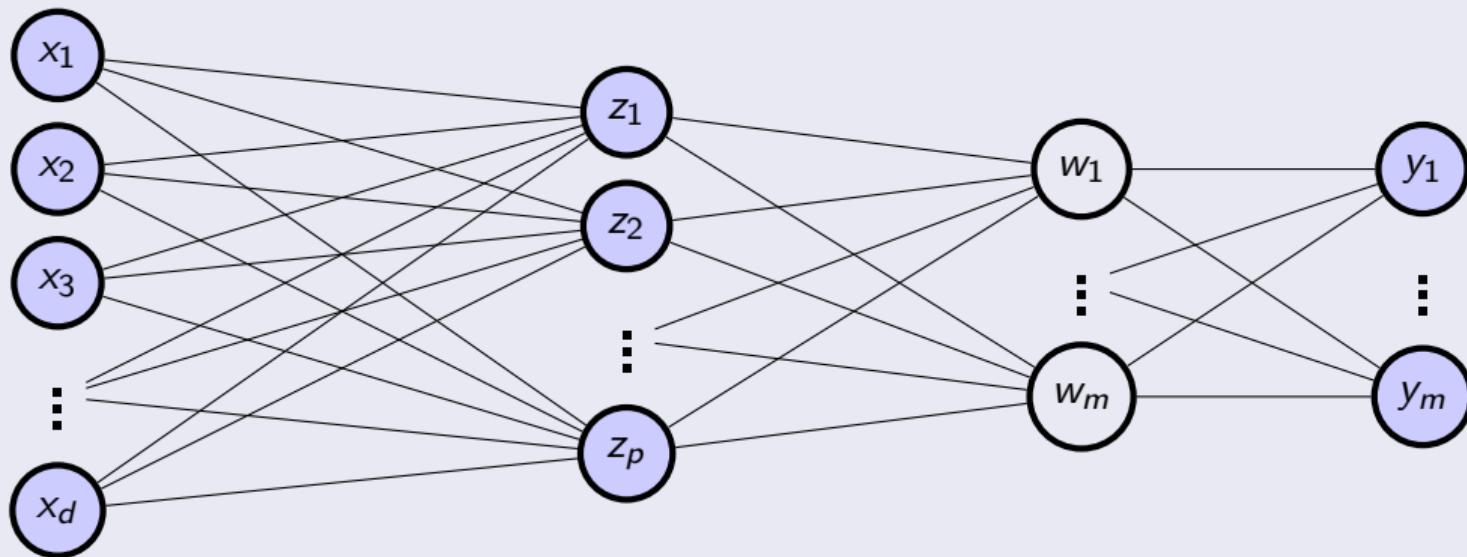
inputs/features	$x \in \mathbb{R}^d$
neurons/hidden layer	$z_i = \sigma(\alpha_{i0} + \alpha_i^T x) \quad \forall i \in \{1, \dots, p\}$
network outputs	$w_j = \beta_{j0} + \beta_j^T z \quad \forall j \in \{1, \dots, m\}$
prediction/outputs	$\hat{y}_j = \tau_j(w) \quad \forall j \in \{1, \dots, m\}$

with

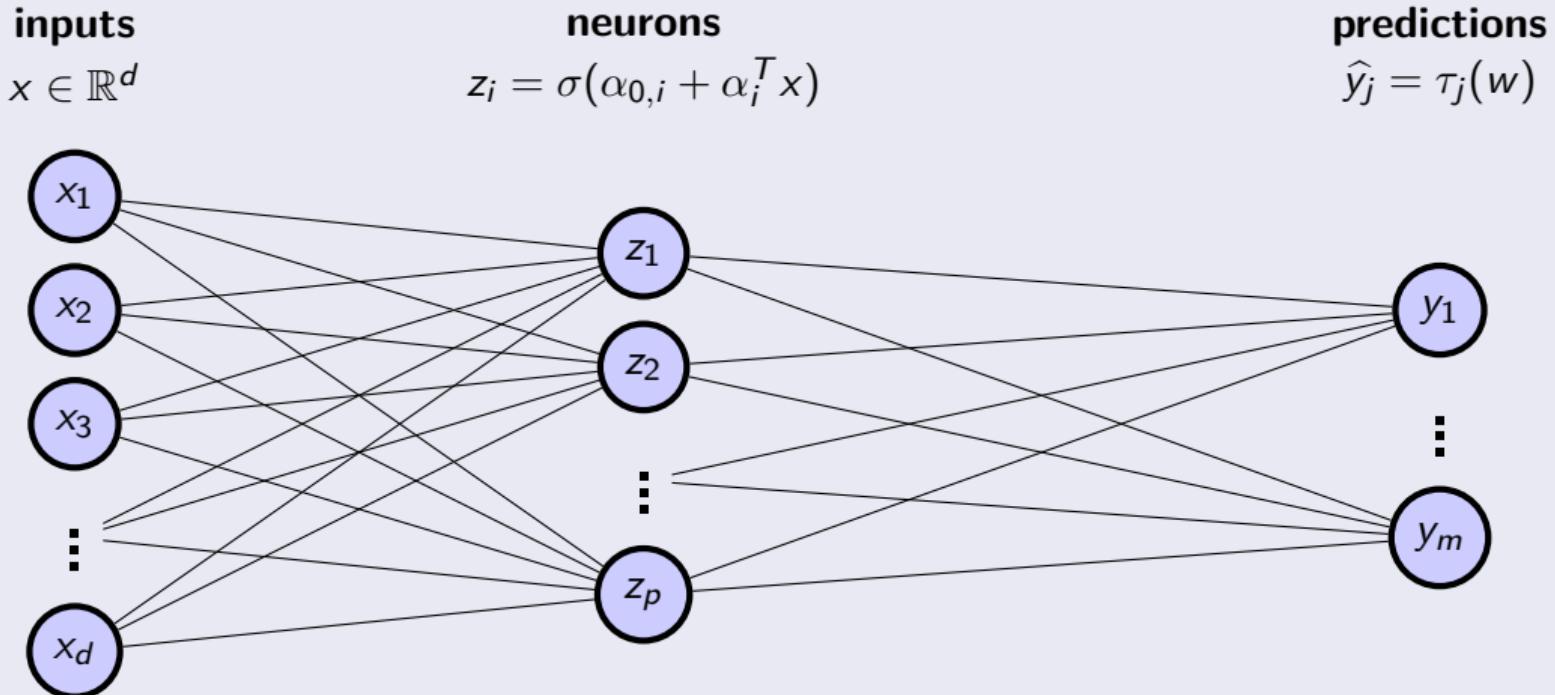
- $\alpha_{i0} \in \mathbb{R}$ and $\beta_{j0} \in \mathbb{R}$ **biases**,
- $\alpha_i \in \mathbb{R}^d$ and $\beta_j \in \mathbb{R}^p$ **weights**,
- $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ non-linear **activation function**,
- and $\tau : \mathbb{R}^m \rightarrow \mathbb{R}^m$ the **output function**.

Single layer perceptron

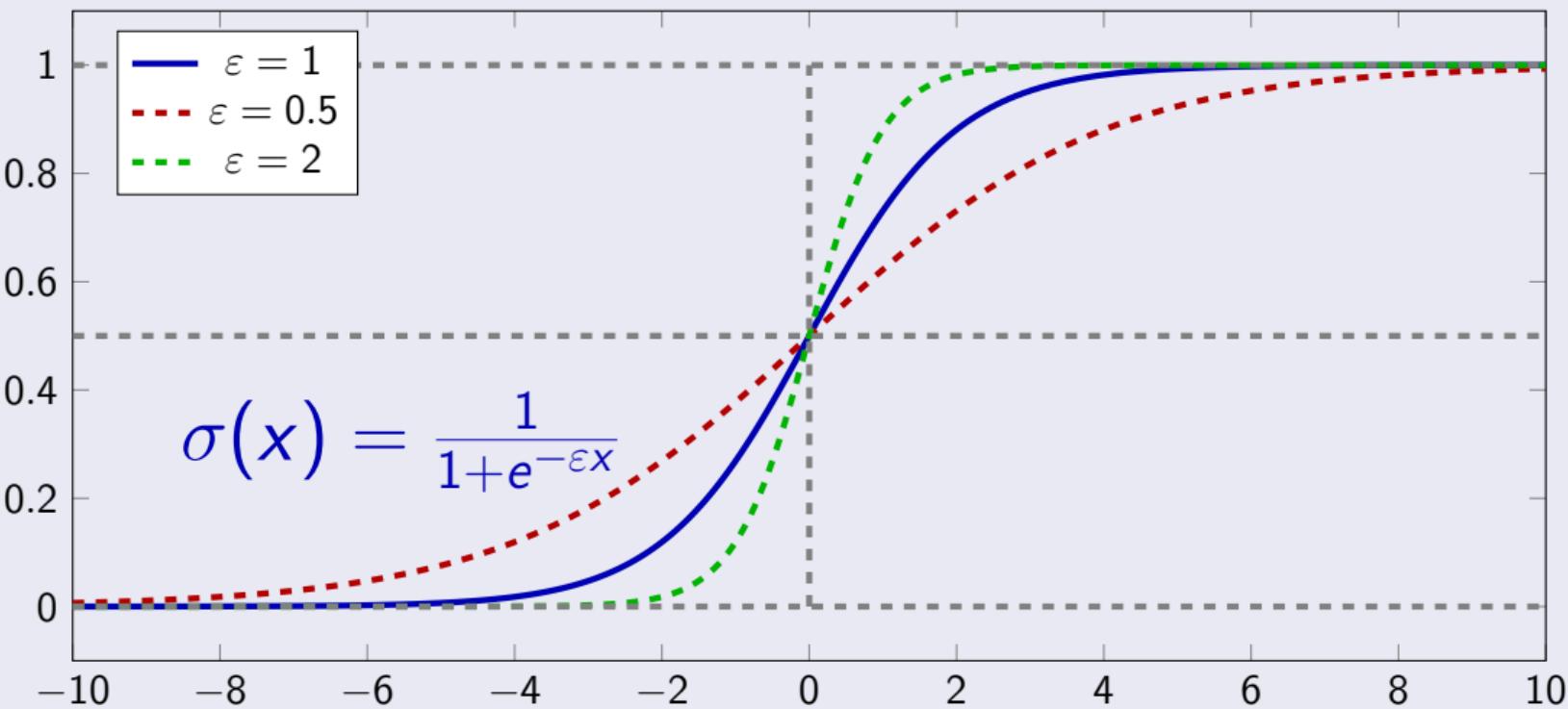
inputs	neurons	network outputs	predictions
$x \in \mathbb{R}^d$	$z_i = \sigma(\alpha_{i0} + \alpha_i^T x)$	$w_j = \beta_{j0} + \beta_j^T z$	$\hat{y}_j = \tau_j(w)$



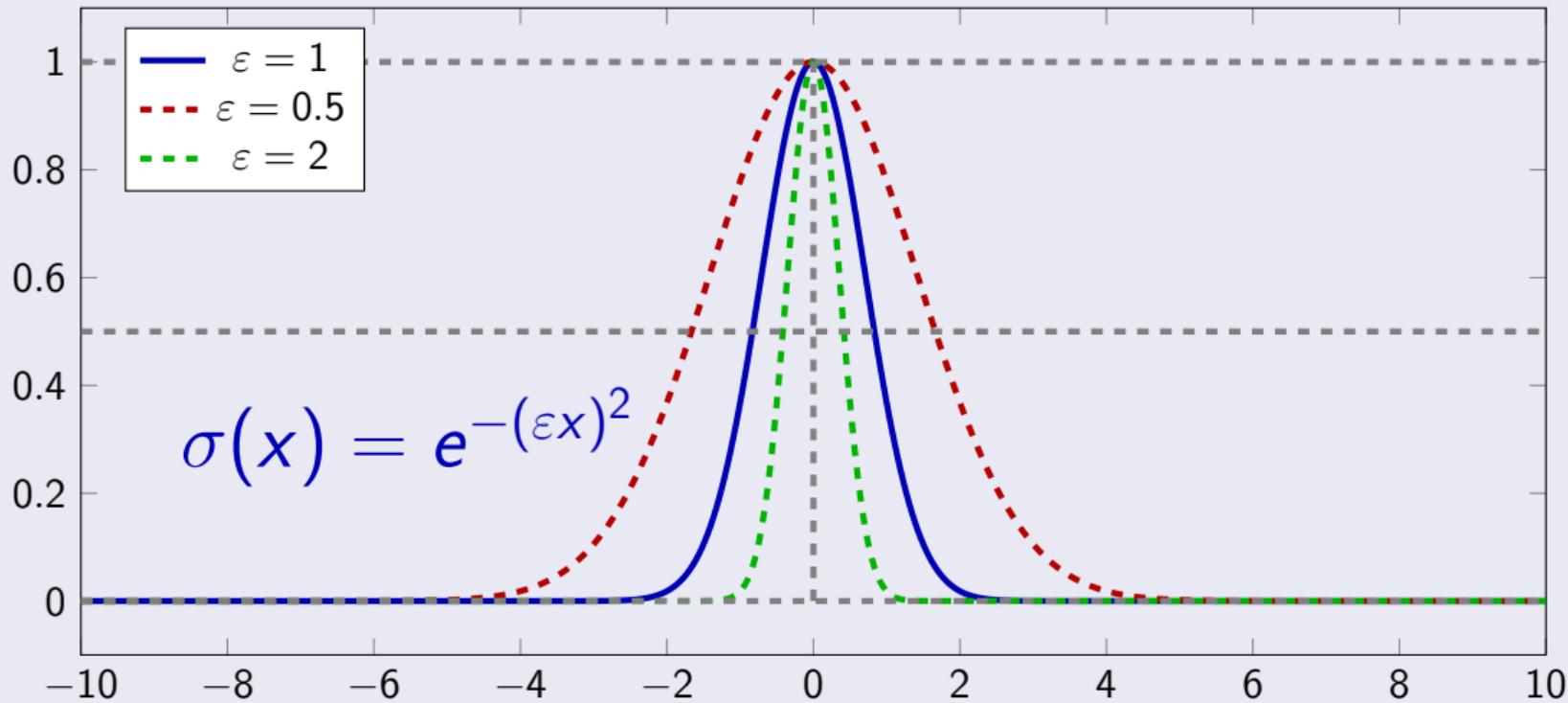
Single layer perceptron



Typical activation function: the sigmoid



Another activation function: the gaussian radial basis function



Typical output functions

- the **softmax function**

$$\forall j \in \{1, \dots, m\} \quad \tau_j : \mathbb{R}^m \rightarrow \mathbb{R}$$

$$\tau_j(w) = \frac{e^{w_j}}{\sum_{k=1}^m e^{w_k}} \in]0, 1[$$

i.e. the prediction $\hat{y} = \tau(w)$ can be seen as a **probability distribution** on \mathcal{Y} :
→ actual prediction is then $\arg \max_{1 \leqslant j \leqslant m} \hat{y}_j$.

- the **identity function**

$$\hat{y} = w \in \mathbb{R}^m,$$

i.e. the prediction \hat{y} is the network output w .

Fitting Neural Networks

Network's parameters θ consist in:

- $\{\alpha_{0,i}, \alpha_{1,i}, \dots, \alpha_{d,i}\} \forall i \in \{1, \dots, p\}$ i.e. $p(d+1)$ weights,
- and $\{\beta_{0,i}, \beta_{1,i}, \dots, \beta_{p,i}\} \forall i \in \{1, \dots, m\}$ i.e. $m(p+1)$ weights.

Usual loss functions (errors measures)

- regression: squared error $L(\theta) = \sum_{i=1}^n \|y_i - \hat{y}_i\|^2 = \sum_{i=1}^n \sum_{j=1}^m (y_{ij} - \hat{y}_{ij})^2$.
- classification: squared error or cross-entropy (deviance, logistic loss)
$$L(\theta) = -\sum_{i=1}^n L_{log}(y_i, \hat{y}_i) = -\sum_{i=1}^n \sum_{j=1}^m y_{ij} \log(\hat{y}_{ij}).$$

Generic approach to minimize $L(\theta)$:

→ gradient descent, called **back-propagation**.

Avoid overfitting using:

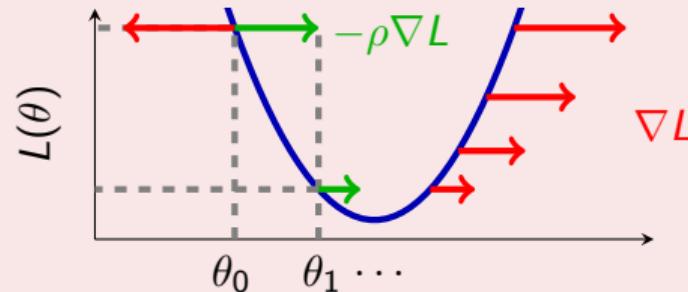
- a penalty term,
- or an early stop.

Gradient descent

Problem: minimize a regular function $L : \begin{cases} \mathbb{R}^q & \rightarrow \mathbb{R} \\ \theta & \mapsto L(\theta) \end{cases}$.

Solution: follow the (opposite of) the gradient!

$$L(\theta) \geq L\left(\theta - \rho \frac{d}{d\theta} L(\theta)\right) = L\left(\theta - \rho \nabla L(\theta)\right),$$



with ρ called *step size* or *learning rate* in ML.

If L is strictly convex, gradient methods lead to a point θ^* close to the global minimum
cf. Section 5.6.2.1 of [HMS20].

Back-propagation

Compute a better θ with gradient descent

→ computation of the gradient w.r.t. the parameters $\theta = \{\alpha, \beta\}$
using the **chain rule** (cf. Theorem 3.7 in [HMS20]).

Reminder: $(\nabla_{\theta} L(\tilde{\theta}))^T = \frac{\partial}{\partial \theta} L(\tilde{\theta}) = \left(\frac{\partial}{\partial \theta_1} L(\tilde{\theta}), \frac{\partial}{\partial \theta_2} L(\tilde{\theta}), \dots, \frac{\partial}{\partial \theta_{\dim(\theta)}} L(\tilde{\theta}) \right)$

Remember that $z_i = \sigma(\alpha_{0i} + \alpha_i^T x), \quad \forall i \in \{1, \dots, p\},$

$$w_j = \beta_{0j} + \beta_j^T z, \quad \forall j \in \{1, \dots, m\},$$

$$\hat{y}_j = \tau_j(w), \quad \forall j \in \{1, \dots, m\},$$

$$L(\theta) = \sum_{i=1}^n L(y_i, \hat{y}_i). \quad \text{with } y_i \in \mathbb{R}^m \text{ and } \hat{y}_i \in \mathbb{R}^m.$$

Gradient computation: loss

$$\begin{aligned} (\nabla_{\theta} L)^T &= \frac{\partial}{\partial \theta} \sum_{i=1}^n L(y_i, \hat{y}_i) = \frac{\partial}{\partial \theta} \sum_{i=1}^n \sum_{j=1}^m L(y_{ij}, \hat{y}_{ij}) = \sum_{i=1}^n \sum_{j=1}^m \frac{\partial}{\partial \theta} L(y_{ij}, \hat{y}_{ij}) \\ (\text{chain rule}) &= \sum_{i=1}^n \sum_{j=1}^m \frac{\partial}{\partial \hat{y}_{ij}} L(y_{ij}, \hat{y}_{ij}) \frac{\partial}{\partial \theta} \hat{y}_{ij} \end{aligned}$$

Gradient computation: loss

$$(\nabla_{\theta} L)^T = \sum_{i=1}^n \sum_{j=1}^m \frac{\partial}{\partial \widehat{y}_{ij}} L(y_{ij}, \widehat{y}_{ij}) \frac{\partial}{\partial \theta} \widehat{y}_{ij}$$

Squared error

$$\frac{\partial}{\partial \widehat{y}_{ij}} L(y_{ij}, \widehat{y}_{ij}) = \frac{\partial}{\partial \widehat{y}_{ij}} (y_{ij} - \widehat{y}_{ij})^2 = -2(y_{ij} - \widehat{y}_{ij}).$$

Gradient computation: loss

$$(\nabla_{\theta} L)^T = \sum_{i=1}^n \sum_{j=1}^m \frac{\partial}{\partial \widehat{y}_{ij}} L(y_{ij}, \widehat{y}_{ij}) \frac{\partial}{\partial \theta} \widehat{y}_{ij}$$

Cross entropy

$$\frac{\partial}{\partial \widehat{y}_{ij}} L(y_{ij}, \widehat{y}_{ij}) = \frac{\partial}{\partial \widehat{y}_{ij}} y_{ij} \log(\widehat{y}_{ij}) = y_{ij} \frac{\partial}{\partial \widehat{y}_{ij}} \log(\widehat{y}_{ij}) = \frac{y_{ij}}{\widehat{y}_{ij}}.$$

Gradient computation: loss

$$(\nabla_{\theta} L)^T = \sum_{i=1}^n \sum_{j=1}^m \frac{\partial}{\partial \hat{y}_{ij}} L(y_{ij}, \hat{y}_{ij}) \frac{\partial}{\partial \theta} \hat{y}_{ij}$$

So now, we fix a sample $\hat{y} \in \mathbb{R}^m$, and we have to deal with $\frac{\partial}{\partial \theta} \hat{y}_i = \frac{\partial}{\partial \theta} \tau_i(w)$,
where $\hat{y}_i \in \mathbb{R}$ and $i \in \{1, \dots, m\}$.

Gradient computation: β

- $\forall i \in \{1, \dots, m\}, \quad w_i = \beta_{i0} + \beta_i^T z = \beta_{i0} + \sum_{k=1}^p \beta_{ik} z_k$
 - $\frac{\partial}{\partial \beta_{i0}} w_i = \frac{\partial}{\partial \beta_{i0}} \beta_{i0} = 1, \text{ and } \frac{\partial}{\partial \beta_{kj}} w_i = 0 \text{ if } k \neq i,$
 - $\frac{\partial}{\partial \beta_{ij}} w_i = \frac{\partial}{\partial \beta_{ij}} \beta_{ij} z_j = z_j, \text{ and } \frac{\partial}{\partial \beta_{kj}} w_i = 0 \text{ if } k \neq i.$
- $\forall i \in \{1, \dots, m\}, \quad \hat{y}_i = \tau_i(w)$
 - $\forall k \in \{1, \dots, m\}, \quad \frac{\partial}{\partial \beta_{k0}} \hat{y}_i = \frac{\partial}{\partial \beta_{k0}} \tau_i(w) = \nabla \tau_i(w)^T \frac{\partial}{\partial \beta_{k0}} w = \frac{\partial}{\partial w_k} \tau_i(w),$
 - $\forall j \in \{1, \dots, p\} \quad \frac{\partial}{\partial \beta_{kj}} \hat{y}_i = \frac{\partial}{\partial \beta_{kj}} \tau_i(w) = \nabla \tau_i(w)^T \frac{\partial}{\partial \beta_{kj}} w = \frac{\partial}{\partial w_k} \tau_i(w) z_j.$

Gradient computation: β

- We have $\forall i \in \{1, \dots, m\}, \forall k \in \{1, \dots, m\}, \forall j \in \{1, \dots, p\}$,

$$\frac{\partial}{\partial \beta_{k0}} \hat{y}_i = \frac{\partial}{\partial w_k} \tau_i(w),$$

$$\frac{\partial}{\partial \beta_{kj}} \hat{y}_i = \frac{\partial}{\partial w_k} \tau_i(w) z_j.$$

Let's compute the partial derivatives with respect to α components.

Gradient computation: α

First, the neurons $\{z_i\}_{i=1}^p$:

$$\blacksquare \forall i \in \{1, \dots, p\}, \quad z_i = \sigma(\alpha_{i0} + \alpha_i^T x) = \sigma\left(\alpha_{i0} + \sum_{k=1}^d \alpha_{ik} x_k\right)$$

$$\frac{\partial}{\partial \alpha_{i0}} z_i = \sigma'(\alpha_{i0} + \alpha_i^T x) \frac{\partial}{\partial \alpha_{i0}} \alpha_{i0} = \sigma'(\alpha_{i0} + \alpha_i^T x),$$

$$\forall j \in \{1, \dots, d\}, \quad \frac{\partial}{\partial \alpha_{ij}} z_i = \sigma'(\alpha_{i0} + \alpha_i^T x) \frac{\partial}{\partial \alpha_{ij}} \alpha_{ij} x_j = \sigma'(\alpha_{i0} + \alpha_i^T x) x_j,$$

$$\frac{\partial}{\partial \alpha_{kj}} z_i = \frac{\partial}{\partial \alpha_{kj}} z_i = 0 \text{ if } k \neq i.$$

Now, what are the partial derivatives of w with respect to α components?

Gradient computation: α

- We have $\forall i \in \{1, \dots, p\}$, $\frac{\partial}{\partial \alpha_{i0}} z_i = \sigma'(\alpha_{i0} + \alpha_i^T x)$,
- $\forall j \in \{1, \dots, d\}$, $\frac{\partial}{\partial \alpha_{ij}} z_i = \sigma'(\alpha_{i0} + \alpha_i^T x)x_j$ and $\frac{\partial}{\partial \alpha_{k0}} z_i = \frac{\partial}{\partial \alpha_{kj}} z_i = 0$ if $k \neq i$.
- $\forall i \in \{1, \dots, m\}$, $w_i = \beta_{i0} + \beta_i^T z = \beta_{i0} + \sum_{l=1}^p \beta_{il} z_l$
- $\forall k \in \{1, \dots, p\}$, $\frac{\partial}{\partial \alpha_{k0}} w_i = \sum_{l=1}^p \beta_{il} \frac{\partial}{\partial \alpha_{k0}} z_l = \beta_{ik} \frac{\partial}{\partial \alpha_{k0}} z_k = \beta_{ik} \sigma'(\alpha_{k0} + \alpha_k^T x)$,
- $\forall j \in \{1, \dots, d\}$, $\frac{\partial}{\partial \alpha_{kj}} w_i = \sum_{l=1}^p \beta_{il} \frac{\partial}{\partial \alpha_{ij}} z_l = \beta_{ik} \frac{\partial}{\partial \alpha_{kj}} z_k = \beta_{ik} \sigma'(\alpha_{k0} + \alpha_k^T x)x_j$.

Now, what are the partial derivatives of \hat{y} with respect to α components?

Gradient computation: α

- We have $\forall i \in \{1, \dots, m\}$, $\forall k \in \{1, \dots, p\}$, $\forall j \in \{1, \dots, d\}$,

$$\frac{\partial}{\partial \alpha_{k0}} w_i = \beta_{ik} \sigma'(\alpha_{k0} + \alpha_k^T x), \text{ and } \frac{\partial}{\partial \alpha_{kj}} w_i = \beta_{ik} \sigma'(\alpha_{k0} + \alpha_k^T x) x_j.$$

- $\forall i \in \{1, \dots, m\}$, $\hat{y}_i = \tau_i(w)$

$$\begin{aligned} \forall k \in \{1, \dots, p\}, \frac{\partial}{\partial \alpha_{k0}} \hat{y}_i &= \frac{\partial}{\partial \alpha_{k0}} \tau_i(w) = \nabla \tau_i(w)^T \frac{\partial}{\partial \alpha_{k0}} w \\ &= \sum_{l=1}^m \frac{\partial}{\partial w_l} \tau_i(w) \frac{\partial}{\partial \alpha_{k0}} w_l = \sum_{l=1}^m \frac{\partial}{\partial w_l} \tau_i(w) \beta_{lk} \sigma'(\alpha_{k0} + \alpha_k^T x), \end{aligned}$$

$$\begin{aligned} \forall j \in \{1, \dots, d\}, \frac{\partial}{\partial \alpha_{kj}} \hat{y}_i &= \frac{\partial}{\partial \alpha_{kj}} \tau_i(w) = \nabla \tau_i(w)^T \frac{\partial}{\partial \alpha_{kj}} w \\ &= \sum_{l=1}^m \frac{\partial}{\partial w_l} \tau_i(w) \frac{\partial}{\partial \alpha_{kj}} w_l = \sum_{l=1}^m \frac{\partial}{\partial w_l} \tau_i(w) \beta_{lk} \sigma'(\alpha_{k0} + \alpha_k^T x) x_j. \end{aligned}$$

Gradient computation: α

- We have $\forall i \in \{1, \dots, m\}, \forall k \in \{1, \dots, p\}, \forall j \in \{1, \dots, d\}$,

$$\frac{\partial}{\partial \alpha_{k0}} \hat{y}_i = \left(\sum_{l=1}^m \frac{\partial}{\partial w_l} \tau_i(w) \beta_{lk} \right) \sigma'(\alpha_{k0} + \alpha_k^T x),$$

$$\frac{\partial}{\partial \alpha_{kj}} \hat{y}_i = \left(\sum_{l=1}^m \frac{\partial}{\partial w_l} \tau_i(w) \beta_{lk} \right) \sigma'(\alpha_{k0} + \alpha_k^T x) x_j.$$

Here is where we see backpropagation.

Put it all together: $\frac{\partial}{\partial \theta} \hat{y}_i = (\nabla_{\theta} \hat{y}_i)^T$

■ $\forall i \in \{1, \dots, m\}, \forall k \in \{1, \dots, m\}, \forall j \in \{1, \dots, p\},$

$$\frac{\partial}{\partial \beta_{k0}} \hat{y}_i = \frac{\partial}{\partial w_k} \tau_i(w) \stackrel{\text{def}}{=} \delta_k^\beta,$$

$$\frac{\partial}{\partial \beta_{kj}} \hat{y}_i = \frac{\partial}{\partial w_k} \tau_i(w) z_j = \delta_k^\beta z_j.$$

■ $\forall i \in \{1, \dots, m\}, \forall k \in \{1, \dots, p\}, \forall j \in \{1, \dots, d\},$

$$\frac{\partial}{\partial \alpha_{k0}} \hat{y}_i = \left(\sum_{l=1}^m \frac{\partial}{\partial w_l} \tau_i(w) \beta_{lk} \right) \sigma'(\alpha_{k0} + \alpha_k^T x) \stackrel{\text{def}}{=} \delta_k^\alpha,$$

$$\frac{\partial}{\partial \alpha_{kj}} \hat{y}_i = \left(\sum_{l=1}^m \frac{\partial}{\partial w_l} \tau_i(w) \beta_{lk} \right) \sigma'(\alpha_{k0} + \alpha_k^T x) x_j = \delta_k^\alpha x_j.$$

Put it all together: $\frac{\partial}{\partial \theta} \hat{y}_i = (\nabla_{\theta} \hat{y}_i)^T$

■ $\forall i \in \{1, \dots, m\}, \quad \forall k \in \{1, \dots, m\}, \forall j \in \{1, \dots, p\},$

$$\frac{\partial}{\partial \beta_{k0}} \hat{y}_i = \frac{\partial}{\partial w_k} \tau_i(w) \stackrel{\text{def}}{=} \delta_k^\beta,$$

$$\frac{\partial}{\partial \beta_{kj}} \hat{y}_i = \frac{\partial}{\partial w_k} \tau_i(w) z_j = \delta_k^\beta z_j.$$

■ $\forall i \in \{1, \dots, m\}, \quad \forall k \in \{1, \dots, p\}, \forall j \in \{1, \dots, d\},$

$$\frac{\partial}{\partial \alpha_{k0}} \hat{y}_i = \left(\sum_{l=1}^m \delta_l^\beta \beta_{lk} \right) \sigma'(\alpha_{k0} + \alpha_k^T x) \stackrel{\text{def}}{=} \delta_k^\alpha,$$

$$\frac{\partial}{\partial \alpha_{kj}} \hat{y}_i = \left(\sum_{l=1}^m \delta_l^\beta \beta_{lk} \right) \sigma'(\alpha_{k0} + \alpha_k^T x) x_j = \delta_k^\alpha x_j.$$

$$\Rightarrow \delta_k^\alpha = \left(\sum_{l=1}^m \delta_l^\beta \beta_{lk} \right) \sigma'(\alpha_{k0} + \alpha_k^T x) \text{ a function of } \left\{ \delta_l^\beta \right\}_{l=1}^m.$$

Put it all together: $\frac{\partial}{\partial \theta} \hat{y}_i = (\nabla_{\theta} \hat{y}_i)^T$

■ $\forall i \in \{1, \dots, m\}, \quad \forall k \in \{1, \dots, m\}, \forall j \in \{1, \dots, p\},$

$$\frac{\partial}{\partial \beta_{k0}} \hat{y}_i = \frac{\partial}{\partial w_k} \tau_i(w) \stackrel{\text{def}}{=} \delta_k^\beta,$$

$$\frac{\partial}{\partial \beta_{kj}} \hat{y}_i = \frac{\partial}{\partial w_k} \tau_i(w) z_j = \delta_k^\beta z_j.$$

■ $\forall i \in \{1, \dots, m\}, \quad \forall k \in \{1, \dots, p\}, \forall j \in \{1, \dots, d\},$

$$\frac{\partial}{\partial \alpha_{k0}} \hat{y}_i = \left(\sum_{l=1}^m \delta_l^\beta \beta_{lk} \right) \sigma'(\alpha_{k0} + \alpha_k^T x) \stackrel{\text{def}}{=} \delta_k^\alpha,$$

$$\frac{\partial}{\partial \alpha_{kj}} \hat{y}_i = \left(\sum_{l=1}^m \delta_l^\beta \beta_{lk} \right) \sigma'(\alpha_{k0} + \alpha_k^T x) x_j = \delta_k^\alpha x_j.$$

$$\Rightarrow \delta_k^\alpha = \left(\sum_{l=1}^m \delta_l^\beta \beta_{lk} \right) \sigma'(\alpha_{k0} + \alpha_k^T x) \text{ a function of } \left\{ \delta_l^\beta \right\}_{l=1}^m.$$

Backpropagation equations

$$\forall k \in \{1, \dots, p\} \quad \delta_k^\alpha = \left(\sum_{l=1}^m \delta_l^\beta \beta_{lk} \right) \sigma'(\alpha_{k0} + \alpha_k^T x).$$

A two-pass algorithm

- Forward pass: computation of \hat{y}
- Backward pass:
 - computation of δ^β
 - computation of δ^α from δ^β
- Computation of $\nabla_\theta L$ with δ^α and δ^β .
- Gradient descent i.e. update of the weights.

Computation of $\nabla_{\theta} L$

$$\frac{\partial}{\partial \beta_{k0}} L(\theta) = \sum_{i=1}^n \sum_{j=1}^m \frac{\partial}{\partial \widehat{y}_{ij}} L(y_{ij}, \widehat{y}_{ij}) \frac{\partial}{\partial \beta_{k0}} \widehat{y}_{ij} = \sum_{i=1}^n \sum_{j=1}^m \frac{\partial}{\partial \widehat{y}_{ij}} L(y_{ij}, \widehat{y}_{ij}) \delta_k^\beta$$

$$\frac{\partial}{\partial \beta_{kl}} L(\theta) = \sum_{i=1}^n \sum_{j=1}^m \frac{\partial}{\partial \widehat{y}_{ij}} L(y_{ij}, \widehat{y}_{ij}) \frac{\partial}{\partial \beta_{kl}} \widehat{y}_{ij} = \sum_{i=1}^n \sum_{j=1}^m \frac{\partial}{\partial \widehat{y}_{ij}} L(y_{ij}, \widehat{y}_{ij}) \delta_k^\beta z_l$$

$$\frac{\partial}{\partial \alpha_{k0}} L(\theta) = \sum_{i=1}^n \sum_{j=1}^m \frac{\partial}{\partial \widehat{y}_{ij}} L(y_{ij}, \widehat{y}_{ij}) \frac{\partial}{\partial \alpha_{k0}} \widehat{y}_{ij} = \sum_{i=1}^n \sum_{j=1}^m \frac{\partial}{\partial \widehat{y}_{ij}} L(y_{ij}, \widehat{y}_{ij}) \delta_k^\alpha$$

$$\frac{\partial}{\partial \alpha_{kl}} L(\theta) = \sum_{i=1}^n \sum_{j=1}^m \frac{\partial}{\partial \widehat{y}_{ij}} L(y_{ij}, \widehat{y}_{ij}) \frac{\partial}{\partial \alpha_{kl}} \widehat{y}_{ij} = \sum_{i=1}^n \sum_{j=1}^m \frac{\partial}{\partial \widehat{y}_{ij}} L(y_{ij}, \widehat{y}_{ij}) \delta_k^\alpha x_l$$

Update of the weights

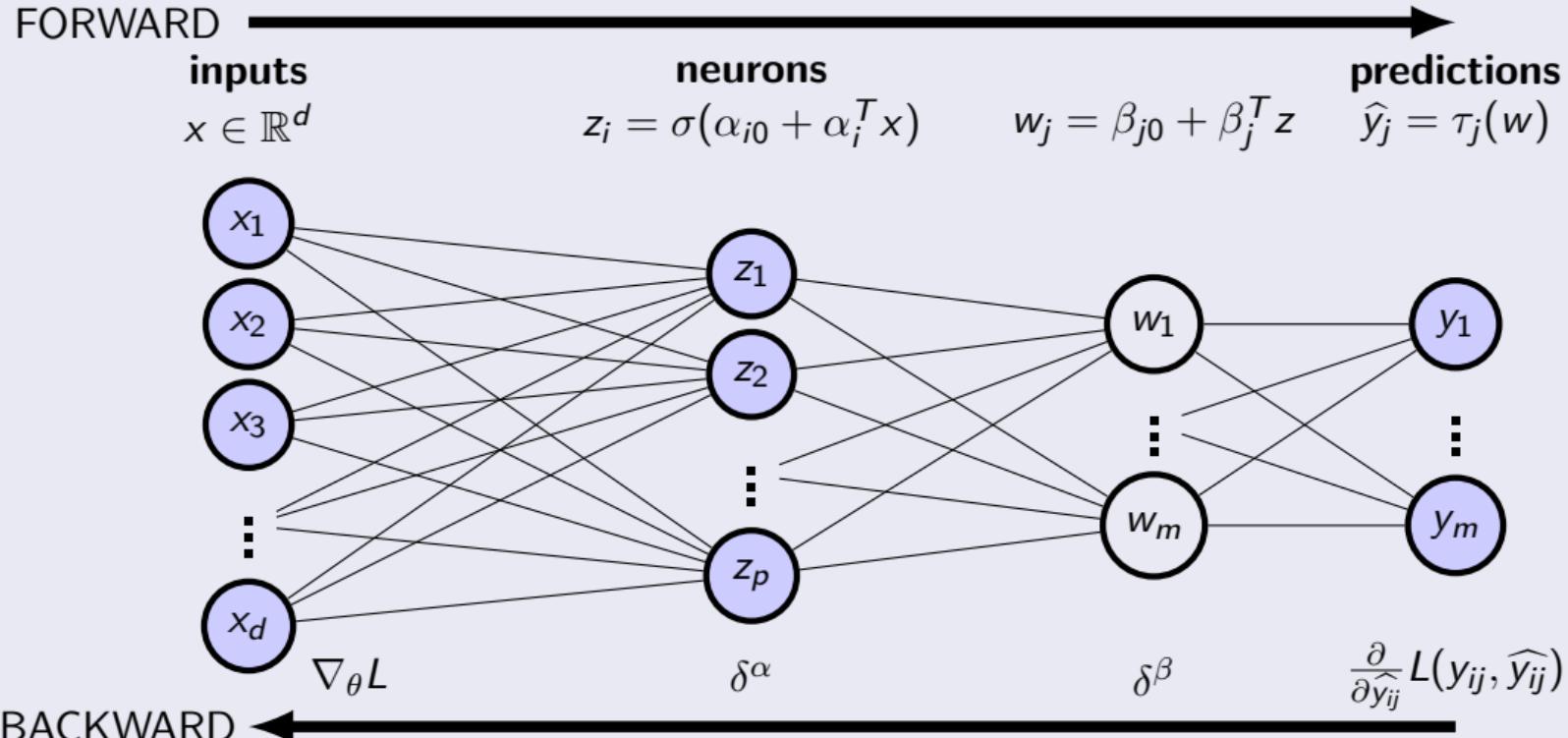
$$\beta_{k0}^{new} = \beta_{k0} - \rho \frac{\partial}{\partial \beta_{k0}} L(\theta)$$

$$\beta_{kj}^{new} = \beta_{kj} - \rho \frac{\partial}{\partial \beta_{kj}} L(\theta)$$

$$\alpha_{k0}^{new} = \alpha_{k0} - \rho \frac{\partial}{\partial \alpha_{k0}} L(\theta)$$

$$\alpha_{kj}^{new} = \alpha_{kj} - \rho \frac{\partial}{\partial \alpha_{kj}} L(\theta)$$

Forward/Backward pass



Advantages of back-propagation

- simple and local nature
- information shared only with connected units \Rightarrow efficient implementation on parallel architecture computer (e.g. GPU).

Batch versus Stochastic/Online learning

- We used the entire dataset (batch) to compute the gradient.
- We can also use a random subset (Stochastic Gradient Descent).
- In this case, the learning rate ρ should tend to zero. \Rightarrow efficient implementation on parallel architecture computer (e.g. GPU).

Advantages of back-propagation

- simple and local nature
- information shared only with connected units \Rightarrow efficient implementation on parallel architecture computer (e.g. GPU).

Batch versus Stochastic/Online learning

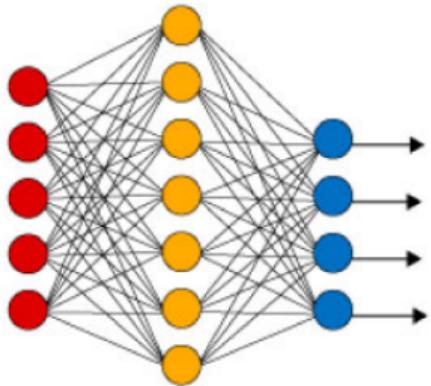
- We used the entire dataset (batch) to compute the gradient.
- We can also use a random subset (Stochastic Gradient Descent).
- In this case, the learning rate ρ should tend to zero. \Rightarrow efficient implementation on parallel architecture computer (e.g. GPU).

Issues

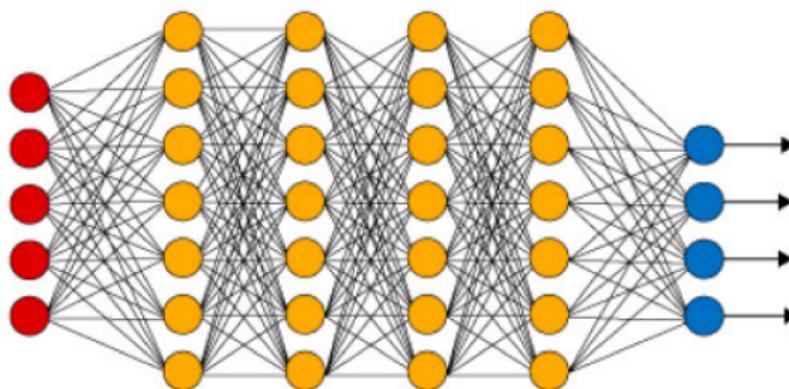
- starting values
- overfitting (penalizes weights, early stoping)
- design number of hidden units and layes
- $\theta \rightarrow L(\theta)$ is not strictly convex
⇒ convergence of the gradient descent is not ensured!

Simple Neural Networks versus Deep Learning

Simple Neural Network



Deep Learning Neural Network

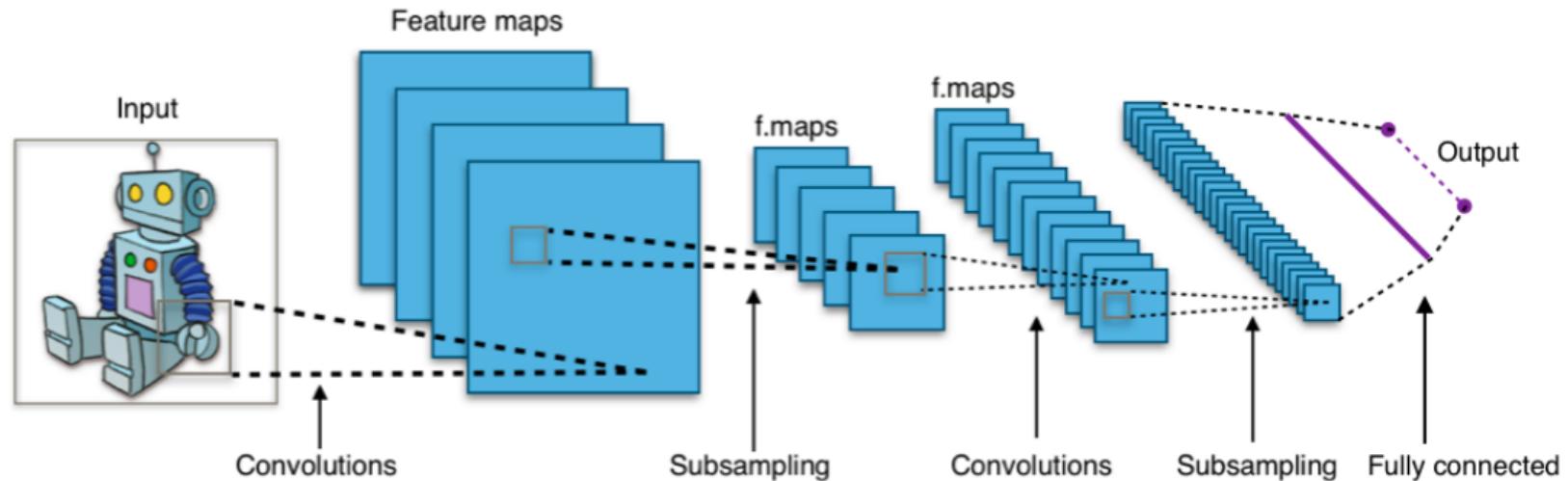


● Input Layer

● Hidden Layer

● Output Layer

Convolutional Neural Network



- the most famous libraries are pytorch, keras, tensorflow
- Pro:
 - no feature extraction (end to end learning, learning from the raw data)
- Cons:
 - no guarantees of convergence in a global minimum
 - large amount of training data needed
 - large number of hyperparameters (layer/units number, activation functions...)
 - "black box" model, "not very useful for producing an understandable model for the data"
[HTF09]

1 Introduction

2 Supervised Learning

3 Neural Networks and Deep learning

4 Mathematical tools for Machine Learning

5 Risks

Markov's Inequality

Let Z be a non negative random variable and $\varepsilon > 0$ a positive real number. Then,

$$\mathbb{P}(Z \geq \varepsilon) \leq \frac{\mathbb{E}[Z]}{\varepsilon}.$$

Proof

$$\begin{aligned} Z \geq 0 &\Rightarrow \varepsilon \cdot \mathbb{1}_{\{Z \geq \varepsilon\}} \leq Z \\ &\Rightarrow \varepsilon \cdot \mathbb{P}(Z \geq \varepsilon) \leq \mathbb{E}[Z] \\ &\Rightarrow \mathbb{P}(Z \geq \varepsilon) \leq \frac{\mathbb{E}[Z]}{\varepsilon} \end{aligned}$$

(see Theorem 1.3.11 in Section 1.3.4 of [GS20]).

Chebyshev's inequality

Let Z be a random variable and $\varepsilon > 0$ a positive real number. Then,

$$\mathbb{P}(|Z - \mathbb{E}[Z]| \geq \varepsilon) \leq \frac{\text{Var}(Z)}{\varepsilon^2}.$$

Proof

By using Markov's inequality on the non negative random variable $(Z - \mathbb{E}[Z])^2$,

$$\mathbb{P}(|Z - \mathbb{E}[Z]| \geq \varepsilon) = \mathbb{P}\left((Z - \mathbb{E}[Z])^2 \geq \varepsilon^2\right) \leq \frac{\mathbb{E}[(Z - \mathbb{E}[Z])^2]}{\varepsilon^2} = \frac{\text{Var}(Z)}{\varepsilon^2}.$$

□

(see also [GS20]).

Hoeffding's inequality

Let $(X_i)_{i=1}^n$ be n independent random variables. Assume that $\forall i \in \{1, \dots, n\}$, $\exists (a_i, b_i) \in \mathbb{R}^2$ such that $a_i \leq X_i \leq b_i$. Then, $\forall \varepsilon > 0$,

$$\mathbb{P}\left(\sum_{i=1}^n (X_i - \mathbb{E}[X_i]) > \varepsilon\right) \leq \exp\left(-\frac{2\varepsilon^2}{\sum_{i=1}^n (b_i - a_i)^2}\right).$$

In order to show this inequality, we'll need another result called **Hoeffding's Lemma**...

Hoeffding's Lemma

Let Z be a centered random variable (i.e. $\mathbb{E}[Z] = 0$) such that $a \leq Z \leq b$.

$$\text{Then, } \forall \lambda > 0, \quad \mathbb{E} [e^{\lambda Z}] \leq \exp \left(\frac{\lambda^2(b-a)^2}{8} \right).$$

Proof – 1

Let's consider the function $\Psi : \lambda \mapsto \ln(\mathbb{E}[e^{\lambda Z}])$ with $\lambda > 0$. Note that $\omega \mapsto e^{\lambda Z(\omega)}$ and $\omega \mapsto \frac{\partial^k}{\partial \lambda^k} e^{\lambda Z(\omega)} = Z(\omega)^k e^{\lambda Z(\omega)} \forall k \in \mathbb{N}$ are bounded. Then they are integrable (\mathbb{P} is a finite measure) and the theorem of *differentiation under the integral sign* [HMS20] gives:

$$\frac{d^k}{d\lambda^k} \mathbb{E}[e^{\lambda Z}] = \mathbb{E}\left[\frac{\partial^k}{\partial \lambda^k} e^{\lambda Z}\right] = \mathbb{E}[Z^k e^{\lambda Z}].$$

$$\Rightarrow \Psi'(\lambda) = \frac{d}{d\lambda} \ln(\mathbb{E}[e^{\lambda Z}]) = \frac{d}{d\lambda} (\mathbb{E}[e^{\lambda Z}]) \frac{1}{\mathbb{E}[e^{\lambda Z}]} = \frac{\mathbb{E}[Z e^{\lambda Z}]}{\mathbb{E}[e^{\lambda Z}]}$$

Hoeffding's inequality

Hoeffding's Lemma: $\mathbb{E} [e^{\lambda Z}] \leq \exp \left(\frac{\lambda^2(b-a)^2}{8} \right).$

Proof – 2

$$\begin{aligned}\Psi : \lambda \mapsto \ln \left(\mathbb{E} [e^{\lambda Z}] \right) \text{ with } \lambda > 0. \quad \Rightarrow \Psi'(\lambda) = \frac{d}{d\lambda} \left(\mathbb{E} [e^{\lambda Z}] \right) \frac{1}{\mathbb{E} [e^{\lambda Z}]} = \frac{\mathbb{E} [Ze^{\lambda Z}]}{\mathbb{E} [e^{\lambda Z}]} \\ \Rightarrow \Psi''(\lambda) = \frac{d}{d\lambda} \left(\frac{\mathbb{E} [Ze^{\lambda Z}]}{\mathbb{E} [e^{\lambda Z}]} \right) = \frac{\frac{d}{d\lambda} \left(\mathbb{E} [Ze^{\lambda Z}] \right) \mathbb{E} [e^{\lambda Z}] - \mathbb{E} [Ze^{\lambda Z}] \frac{d}{d\lambda} \left(\mathbb{E} [e^{\lambda Z}] \right)}{\mathbb{E} [e^{\lambda Z}]^2} \\ = \frac{\mathbb{E} [Z^2 e^{\lambda Z}] \mathbb{E} [e^{\lambda Z}] - \mathbb{E} [Ze^{\lambda Z}]^2}{\mathbb{E} [e^{\lambda Z}]^2} \\ = \mathbb{E} \left[Z^2 \frac{e^{\lambda Z}}{\mathbb{E} [e^{\lambda Z}]} \right] - \left(\mathbb{E} \left[Z \frac{e^{\lambda Z}}{\mathbb{E} [e^{\lambda Z}]} \right] \right)^2\end{aligned}$$

Hoeffding's inequality

Hoeffding's Lemma: $\mathbb{E} [e^{\lambda Z}] \leq \exp\left(\frac{\lambda^2(b-a)^2}{8}\right).$

Proof – 3

$$\Psi : \lambda \mapsto \ln\left(\mathbb{E} [e^{\lambda Z}]\right) \text{ with } \lambda > 0. \quad \Rightarrow \Psi''(\lambda) = \mathbb{E} \left[Z^2 \frac{e^{\lambda Z}}{\mathbb{E}[e^{\lambda Z}]} \right] - \left(\mathbb{E} \left[Z \frac{e^{\lambda Z}}{\mathbb{E}[e^{\lambda Z}]} \right] \right)^2.$$

Note that $\omega \mapsto \frac{e^{\lambda Z(\omega)}}{\mathbb{E}[e^{\lambda Z}]}$ is a density on (Ω, \mathbb{P}) (non negative & $\int_{\Omega} \frac{e^{\lambda Z(\omega)}}{\mathbb{E}[e^{\lambda Z}]} d\mathbb{P}(\omega) = \frac{\mathbb{E}[e^{\lambda Z}]}{\mathbb{E}[e^{\lambda Z}]} = 1$).

Let's consider the probability measure \mathbb{Q}_{λ} defined by $\forall A \subset \Omega$, $\mathbb{Q}_{\lambda}(A) = \int_A \frac{e^{\lambda Z(\omega)}}{\mathbb{E}[e^{\lambda Z}]} d\mathbb{P}(\omega)$.

$$\Rightarrow \Psi''(\lambda) = \int_{\Omega} Z(\omega)^2 d\mathbb{Q}_{\lambda}(\omega) - \left(\int_{\Omega} Z(\omega) d\mathbb{Q}_{\lambda}(\omega) \right)^2 = \mathbb{E}_{\mathbb{Q}_{\lambda}} [Z^2] - \mathbb{E}_{\mathbb{Q}_{\lambda}} [Z]^2 = \text{Var}_{\mathbb{Q}_{\lambda}} (Z).$$

Note also that $a \leq Z \leq b \Rightarrow \frac{a-b}{2} \leq Z - \frac{b+a}{2} \leq \frac{b-a}{2} \Rightarrow |Z - \frac{b+a}{2}| \leq \frac{b-a}{2}$.

Then $\forall \lambda > 0$, $\Psi''(\lambda) = \text{Var}_{\mathbb{Q}_{\lambda}} \left(Z - \frac{b+a}{2} \right) \leq \mathbb{E}_{\mathbb{Q}_{\lambda}} \left[\left(Z - \frac{b+a}{2} \right)^2 \right] \leq \left(\frac{b-a}{2} \right)^2$.

Hoeffding's Lemma: $\mathbb{E} [e^{\lambda Z}] \leq \exp \left(\frac{\lambda^2(b-a)^2}{8} \right).$

Proof – 4

$$\Psi : \lambda \mapsto \ln \left(\mathbb{E} [e^{\lambda Z}] \right) \text{ with } \lambda > 0. \quad \Rightarrow \Psi''(\lambda) \leq \left(\frac{b-a}{2} \right)^2.$$

- $\Psi'(0) = \frac{\mathbb{E}[Z e^{0 \times Z}]}{\mathbb{E}[e^{0 \times Z}]} = \mathbb{E}[Z] = 0$ (Z is centered) $\Rightarrow \Psi'(\lambda) = \int_0^\lambda \Psi''(u) du \leq \lambda \frac{(b-a)^2}{4}.$
- $\Psi(0) = \ln \left(\mathbb{E} [e^{0 \times Z}] \right) = \ln(1) = 0$, thus

$$\Psi(\lambda) = \int_0^\lambda \Psi'(u) du \leq \int_0^\lambda \frac{u(b-a)^2}{4} du = \frac{\lambda^2(b-a)^2}{8}.$$

□

Hoeffding's inequality

Let $(X_i)_{i=1}^n$ be n independent random variables. Assume that $\forall i \in \{1, \dots, n\}$, $\exists (a_i, b_i) \in \mathbb{R}^2$ such that $a_i \leq X_i \leq b_i$. Then, $\forall \varepsilon > 0$,

$$\mathbb{P}\left(\sum_{i=1}^n (X_i - \mathbb{E}[X_i]) > \varepsilon\right) \leq \exp\left(-\frac{2\varepsilon^2}{\sum_{i=1}^n (b_i - a_i)^2}\right).$$

Proof – 1

Let's consider the centered random variable $Z_i = X_i - \mathbb{E}[X_i]$. Then $\forall \lambda > 0$,

$$\begin{aligned} \mathbb{P}\left(\sum_{i=1}^n Z_i > \varepsilon\right) &= \mathbb{P}\left(e^{\lambda \sum_{i=1}^n Z_i} > e^{\lambda \varepsilon}\right) \stackrel{\text{(Markov's ineq.)}}{\leq} \frac{\mathbb{E}\left[e^{\lambda \sum_{i=1}^n Z_i}\right]}{e^{\lambda \varepsilon}} = \\ &= \frac{\mathbb{E}\left[\prod_{i=1}^n e^{\lambda Z_i}\right]}{e^{\lambda \varepsilon}} = \frac{\prod_{i=1}^n \mathbb{E}\left[e^{\lambda Z_i}\right]}{e^{\lambda \varepsilon}} \quad (\text{indep. vars., Prop. 1.20 [GS20]}) \end{aligned}$$

Hoeffding's inequality

Hoeffding's inequality: $\mathbb{P}(\sum_{i=1}^n (X_i - \mathbb{E}[X_i]) > \varepsilon) \leq \exp\left(-\frac{2\varepsilon^2}{\sum_{i=1}^n (b_i - a_i)^2}\right).$

Proof – 2

$$\mathbb{P}\left(\sum_{i=1}^n Z_i > \varepsilon\right) \leq \frac{\prod_{i=1}^n \mathbb{E}[e^{\lambda Z_i}]}{e^{\lambda \varepsilon}} \stackrel{\text{(Hoeffding's Lemma)}}{\leq} \frac{\prod_{i=1}^n \exp\left(\frac{\lambda^2(b_i - a_i)^2}{8}\right)}{e^{\lambda \varepsilon}} = \exp\left(-\lambda \varepsilon + \frac{\lambda^2}{8} \sum_{i=1}^n (b_i - a_i)^2\right)$$

Note that $\lambda \mapsto -\lambda \varepsilon + \frac{\lambda^2}{8} \sum_{i=1}^n (b_i - a_i)^2$ is minimal for $\lambda_* = \frac{4\varepsilon}{\sum_{i=1}^n (b_i - a_i)^2}$.

$$\Rightarrow \mathbb{P}(\sum_{i=1}^n Z_i > \varepsilon) \leq \exp\left(-\lambda_* \varepsilon + \frac{\lambda_*^2}{8} \sum_{i=1}^n (b_i - a_i)^2\right)$$

$$= \exp\left(-\frac{4\varepsilon^2}{\sum_{i=1}^n (b_i - a_i)^2} + \frac{16\varepsilon^2}{8 \sum_{i=1}^n (b_i - a_i)^2}\right) = \exp\left(-\frac{2\varepsilon^2}{\sum_{i=1}^n (b_i - a_i)^2}\right).$$

□

Hoeffding's inequality

Let $(X_i)_{i=1}^n$ be n independent random variables. Assume that $\forall i \in \{1, \dots, n\}$, $\exists (a_i, b_i) \in \mathbb{R}^2$ such that $a_i \leq X_i \leq b_i$. Then, $\forall \varepsilon > 0$,

$$\mathbb{P}\left(\sum_{i=1}^n (X_i - \mathbb{E}[X_i]) > \varepsilon\right) \leq \exp\left(-\frac{2\varepsilon^2}{\sum_{i=1}^n (b_i - a_i)^2}\right).$$

Corollary

$$\mathbb{P}\left(\left|\sum_{i=1}^n (X_i - \mathbb{E}[X_i])\right| > \varepsilon\right) \leq 2 \exp\left(-\frac{2\varepsilon^2}{\sum_{i=1}^n (b_i - a_i)^2}\right).$$

Corollary

$$\mathbb{P} \left(\left| \sum_{i=1}^n (X_i - \mathbb{E}[X_i]) \right| > \varepsilon \right) \leq 2\delta \quad \text{with} \quad \delta = \exp \left(- \frac{2\varepsilon^2}{\sum_{i=1}^n (b_i - a_i)^2} \right).$$

Proof

$(-X_i)_{i=1}^n$ are also n independent random variables with $\forall i \in \{1, \dots, n\}$, $-b_i \leq -X_i \leq -a_i$ and $(-b_i, -a_i) \in \mathbb{R}^2$. Since $(a_i - b_i)^2 = (b_i - a_i)^2$, we get $\mathbb{P}(-\sum_{i=1}^n (X_i - \mathbb{E}[X_i]) > \varepsilon) \leq \delta$. Finally, since the events are incompatible,

$$\mathbb{P} \left(\left| \sum_{i=1}^n (X_i - \mathbb{E}[X_i]) \right| > \varepsilon \right) = \mathbb{P} \left(\sum_{i=1}^n (X_i - \mathbb{E}[X_i]) > \varepsilon \right) + \mathbb{P} \left(-\sum_{i=1}^n (X_i - \mathbb{E}[X_i]) > \varepsilon \right) \leq 2\delta.$$

□

Another way to write the inequality

Let $(X_i)_{i=1}^n$ be n independent random variables. Assume that $\forall i \in \{1, \dots, n\}$, $\exists (a_i, b_i) \in \mathbb{R}^2$ such that $a_i \leq X_i \leq b_i$. Then, $\forall \varepsilon > 0$,

$$\mathbb{P}\left(\left|\frac{1}{n} \sum_{i=1}^n (X_i - \mathbb{E}[X_i])\right| > \varepsilon\right) \leq 2\delta \quad \text{with} \quad \delta = \exp\left(-\frac{2n^2\varepsilon^2}{\sum_{i=1}^n (b_i - a_i)^2}\right).$$

by simply replacing the variable $\varepsilon \rightarrow n\varepsilon$.

1 Introduction

2 Supervised Learning

3 Neural Networks and Deep learning

4 Mathematical tools for Machine Learning

5 Risks

Optimal classifier

- X and Y are random variables distributed according to \mathbb{P} .
- We look for a **classifier** that predicts Y , denoted by $c : \mathcal{X} \rightarrow \mathcal{Y} = \{y_+, y_-\}$.
- The probability of error, or **risk** is $R(c) = \mathbb{P}(c(X) \neq Y) = \mathbb{E} \left[\mathbb{1}_{\{c(X) \neq Y\}} \right]$

$$\begin{aligned} &= \mathbb{E} \left[\mathbb{1}_{\{c(X)=y_+\}} \mathbb{1}_{\{Y=y_-\}} + \mathbb{1}_{\{c(X)=y_-\}} \mathbb{1}_{\{Y=y_+\}} \right] \\ &= \mathbb{E} \left[\mathbb{1}_{\{c(X)=y_+\}} \mathbb{E} \left[\mathbb{1}_{\{Y=y_-\}} \mid X \right] + \mathbb{1}_{\{c(X)=y_-\}} \mathbb{E} \left[\mathbb{1}_{\{Y=y_+\}} \mid X \right] \right] \end{aligned}$$

$(\mathbb{E}[Y] = \mathbb{E}[\mathbb{E}[Y \mid X]] \& \mathbb{E}[\phi(X)Y \mid X] \stackrel{a.s}{=} \phi(X)\mathbb{E}[Y \mid X]$ see Prop. 1.31 in [GS20])

$$\begin{aligned} &= \mathbb{E} \left[\mathbb{1}_{\{c(X)=y_+\}} \mathbb{P}(Y = y_- \mid X) + \mathbb{1}_{\{c(X)=y_-\}} \mathbb{P}(Y = y_+ \mid X) \right] \\ &= \mathbb{E} \left[\mathbb{1}_{\{c(X)=y_+\}} (1 - \eta(X)) + \mathbb{1}_{\{c(X)=y_-\}} \eta(X) \right] \end{aligned}$$

with $\eta(X) = \mathbb{P}(Y = y_+ \mid X) \in [0, 1]$.

$$R(c) = \mathbb{E} \left[\mathbb{1}_{\{c(X)=y_+\}} (1 - \eta(X)) + \mathbb{1}_{\{c(X)=y_-\}} \eta(X) \right]$$

Since $0 \leq \eta(X) \leq 1$, the optimal classifier $c : \mathcal{X} \rightarrow \mathcal{Y}$, i.e. minimizing $R(c)$, should be

- equal to y_+ when $\eta(X) > \frac{1}{2}$
- equal to y_- otherwise.

t is called the **Bayes classifier** or **target function**:

$$t(x) = \begin{cases} y_+ & \text{if } \eta(x) > \frac{1}{2}, \\ y_- & \text{if } \eta(x) \leq \frac{1}{2}. \end{cases}$$

The associated risk is then called the **Bayes risk**

$$R^* = \inf_{c:\mathcal{X} \rightarrow \mathcal{Y}} R(c) = R(t) = \mathbb{E} \left[\min \{ \eta(X), 1 - \eta(X) \} \right].$$

Note on Bayes term.

- Thomas Bayes (1701-1761)
- $\mathbb{P}(A|B) = \frac{\mathbb{P}(B|A)\mathbb{P}(A)}{\mathbb{P}(B)}$.
- prior $\mathbb{P}(A)$, posterior $\mathbb{P}(A|B)$.
- no data, information in the prior, parameters have a probability distribution
- prior knowledge on the parameters $p_{prior}(\theta)$, θ becomes a random variable.
- the set of distributions of the dataset $\mathbb{P}_\theta(D_n = d_n)$ **parameterized by** $\theta \in \Theta$ considered as the **conditional probability distribution given** $\theta \in \Theta$: $\mathbb{P}(D_n = d_n | \theta)$.
- $p_{post}(\theta) = \mathbb{P}(\theta | D_n = d_n) = \frac{\mathbb{P}_\theta(D_n = d_n)}{\int_\theta \mathbb{P}_\theta(D_n = d_n) p_{prior}(\theta) d\theta}$



Bayes' rule

$$\mathbb{P}(Hypothesis | Data) = \frac{\mathbb{P}(Data | Hypothesis) \mathbb{P}(Hypothesis)}{\mathbb{P}(Data)}$$

- Bayesian's use Bayes' Rule to **update beliefs in hypotheses in response to data**.
- $\mathbb{P}(Hypothesis | Data)$ is the **posterior distribution**,
- $\mathbb{P}(Hypothesis)$ is the **prior distribution**,
- $\mathbb{P}(Data | Hypothesis)$ is the **likelihood**, and
- $\mathbb{P}(Data)$ is a normalizing constant sometimes called the **evidence**.

Deterministic case

Let's suppose $(X, Y) = (X, f(X))$ with $X \sim \mathbb{P}$ and $f : \mathcal{X} \rightarrow \mathcal{Y}$.

$$\begin{aligned}\eta(X) &= \mathbb{P}(Y = y_+ \mid X) = \mathbb{E} \left[\mathbb{1}_{\{Y=y_+\}} \mid X \right] \\ &= \mathbb{E} \left[\mathbb{1}_{\{f(X)=y_+\}} \mid X \right] \\ &= \mathbb{1}_{\{f(X)=y_+\}}.\end{aligned}$$

$\Rightarrow R^* = \mathbb{E} [\min \{\eta(X), 1 - \eta(X)\}] = 0$, i.e. the Bayes risk is equal to zero.

General case

We can then define the **noise level** $s(x) = \min \{\eta(x), 1 - \eta(x)\}$,
and the Bayes risk can be rewritten $R^* = \mathbb{E}[s(X)]$.

- **Input (features)**: X (random) or x (realization).
- **Output (targets)**: Y (random) or y (realization).
- **Random dataset**: $D_n = (X_i, Y_i)_{i=1}^n \in (\mathcal{X} \times \mathcal{Y})^n$
- usually, features $\mathcal{X} = \mathbb{R}^d$
- set of targets/labels $\{y_k\}_{k=1}^m$. Binary classification: $\{y_+, y_-\}$ ($m = 2$).

Empirical risk

$$R_n(c) = \frac{1}{n} \sum_{i=1}^n \mathbb{1}_{\{c(X_i)=Y_i\}}.$$

Loss class

$$\mathcal{F} = \left\{ f : (x, y) \mapsto \mathbb{1}_{\{c(x) \neq y\}} \mid c \in \mathcal{C} \right\} :$$

Let's denote $f_c(x, y) = \mathbb{1}_{\{c(x) \neq y\}}$:

$$R(c) - R_n(c) = \mathbb{E}[f_c(X, Y)] - \frac{1}{n} \sum_{i=1}^n f_c(X_i, Y_i).$$

Some definitions

- \mathcal{C} is the set of **candidate functions** or **classifiers**: if $c \in \mathcal{C}$, $c : \mathcal{X} \rightarrow \mathcal{Y}$.
- c^* denotes the classifier that minimizes the **theoretical risk**:

$$c^* \in \arg \min_{c \in \mathcal{C}} R(c).$$

Note that if the Bayes classifier t is in \mathcal{C} , then $R(c^*) = R^* \stackrel{\text{def}}{=} R(t) = \mathbb{E} \left[\mathbb{1}_{\{t(x) \neq y\}} \right]$.

- c_n denotes the classifier that minimizes the **empirical risk**:

$$c_n \in \arg \min_{c \in \mathcal{C}} R_n(c).$$

First result

$\forall c \in \mathcal{C}, R_n(c) \xrightarrow{n \rightarrow +\infty} R(c)$ almost surely.

Proof

Strong law of the large numbers (see Theorem 3.1 in [GS20]) applied on integrable (bounded) independent random variables $(f_c(X_i, Y_i))_{i=1}^n$:

$$\mathbb{P} \left(\lim_{n \rightarrow +\infty} \frac{1}{n} \sum_{i=1}^n f_c(X_i, Y_i) = \mathbb{E}[f_c(X, Y)] \right) = 1,$$

i.e.

$$\mathbb{P} \left(\lim_{n \rightarrow +\infty} R_n(c) = R(c) \right) = 1.$$

Property: Hoeffding's inequality

Let $(X_i)_{i=1}^n$ be n independent random variables. Assume that $\forall i \in \{1, \dots, n\}$, $\exists (a_i, b_i) \in \mathbb{R}^2$ such that $a_i \leq X_i \leq b_i$. Then, $\forall \varepsilon > 0$,

$$\mathbb{P}\left(\left|\frac{1}{n} \sum_{i=1}^n (X_i - \mathbb{E}[X_i])\right| > \varepsilon\right) \leq 2 \exp\left(-\frac{2n^2\varepsilon^2}{\sum_{i=1}^n (b_i - a_i)^2}\right).$$

Let us apply this inequality with f_c : $(f_c(X_i, Y_i))_{i=1}^n$ that are independent random variables such that $0 \leq f_c(X_i, Y_i) \leq 1$: $\mathbb{P}\left(\left|\frac{1}{n} \sum_{i=1}^n f_c(X_i, Y_i) - \mathbb{E}[f_c(X, Y)]\right| > \varepsilon\right) \leq 2e^{-2n\varepsilon^2}$.

Result: probabilistic bound on the risk estimation error

$$\mathbb{P}(|R_n(c) - R(c)| > \varepsilon) \leq 2e^{-2n\varepsilon^2} \quad \& \quad \mathbb{P}(R_n(c) - R(c) > \varepsilon) \leq e^{-2n\varepsilon^2}.$$

Bound for a classifier's risk

Probabilistic bound on the risk estimation error

$$\mathbb{P}(|R_n(c) - R(c)| > \varepsilon) \leq 2e^{-2n\varepsilon^2} \quad \& \quad \mathbb{P}(R(c) - R_n(c) > \varepsilon) \leq e^{-2n\varepsilon^2}.$$

By writing $\delta = 2e^{-2n\varepsilon^2}$, we have $\varepsilon = \sqrt{\frac{\ln(\frac{2}{\delta})}{2n}}$ (on the right $\varepsilon = \sqrt{\frac{\ln(\frac{1}{\delta})}{2n}}$), so, we can write:

- with probability at most δ , $|R_n(c) - R(c)| > \sqrt{\frac{\ln(\frac{2}{\delta})}{2n}}$, or also:
- with probability at least $1 - \delta$, $|R_n(c) - R(c)| < \sqrt{\frac{\ln(\frac{2}{\delta})}{2n}}$.

Bound on the true risk

In practice, knowing n and $R_n(c)$, we have $\forall \delta \in]0, 1[$ a bound on the true risk with probability $1 - \delta$:

$$R(c) \leq R_n(c) + \sqrt{\frac{\ln\left(\frac{1}{\delta}\right)}{2n}}.$$

Bound for a finite set of candidate functions

Let us consider a finite set of candidate classifiers \mathcal{C} with $\#\mathcal{C} = N < +\infty$.

$$\begin{aligned}\mathbb{P}(\exists c \in \mathcal{C} \mid |R_n(c) - R(c)| > \varepsilon) &= \mathbb{P}(\cup_{c \in \mathcal{C}} \{|R_n(c) - R(c)| > \varepsilon\}) \\ &\leq \sum_{c \in \mathcal{C}} \mathbb{P}(|R_n(c) - R(c)| > \varepsilon) \\ &\leq 2N e^{-2n\varepsilon^2}.\end{aligned}$$

Then, with $\delta = 2N e^{-2n\varepsilon^2}$, $\varepsilon = \sqrt{\frac{\ln\left(\frac{2N}{\delta}\right)}{2n}}$, so,

with probability at least $1 - \delta$, $\forall c \in \mathcal{C}$,

$$|R_n(c) - R(c)| \leq \sqrt{\frac{\ln(N) + \ln\left(\frac{2}{\delta}\right)}{2n}} \quad \text{and} \quad R(c) \leq R_n(c) + \sqrt{\frac{\ln(N) + \ln\left(\frac{1}{\delta}\right)}{2n}}.$$

The extra $\ln(N)$ accounts for the fact that we want N bounds to hold simultaneously.

Bound on the estimation error

We know that, with probability $1 - \delta$, $\forall c \in \mathcal{C}$, $R_n(c) - R(c)$ is bounded. But we would like to know how to bound the actual risk of our estimation of the optimal classifier $c_n \in \arg \min_{c \in \mathcal{C}} R_n(c)$.

Approximation and estimation errors

$$R(c_n) - R^* = \underbrace{R(c^*) - R^*}_{\text{approximation error}} + \underbrace{R(c_n) - R(c^*)}_{\text{estimation error}}.$$

- The **approximation error** $R(c^*) - R^*$ decreases when the size of the set of candidates \mathcal{C} increases. If it contains the Bayes classifier ($t \in \mathcal{C}$), this error is equal to zero.
- What about the **estimation error** $R(c_n) - R(c^*)$?
i.e. the error due to the minimization of the empirical risk $R_n(c) = \frac{1}{n} \sum_{i=1}^n \mathbb{1}_{\{c(X_i) \neq Y_i\}}$ instead of the actual one $R(c) = \mathbb{E} [\mathbb{1}_{\{c(X) \neq Y\}}]$?

Bound on the estimation error

Approximation and estimation errors

$$R(c_n) - R^* = \underbrace{R(c^*) - R^*}_{\text{approximation error}} + \underbrace{R(c_n) - R(c^*)}_{\text{estimation error}}.$$

$$\begin{aligned} R(c_n) &= R(c_n) - R(c^*) + R(c^*) \\ &\leq R(c_n) - R(c^*) + R(c^*) + R_n(c^*) - R_n(c_n) \quad (\text{since } R_n(c^*) - R_n(c_n) \geq 0) \\ &\leq |R(c_n) - R_n(c_n)| + |R_n(c^*) - R(c^*)| + R(c^*) \leq 2 \sup_{c \in \mathcal{C}} |R(c) - R_n(c)| + R(c^*). \end{aligned}$$

Bound for the estimation error

$$\text{With probability at least } 1 - \delta, \quad R(c_n) - R(c^*) \leq 2 \sqrt{\frac{\ln(N) + \ln\left(\frac{2}{\delta}\right)}{2n}}.$$

While $R(c^*)$ decreases with the size of the class \mathcal{C} , the bound increases ($\#\mathcal{C} = N$).

Infinite case

Let us consider a countable set of classifiers $\mathcal{C} = (c_i)_{i \in \mathbb{N}}$. Let us assume that we know the probability distribution on the classifiers computed by the considered algorithm: $\forall c \in \mathcal{C}$, $p(c) \in [0, 1]$. We take the risk $\delta(c) = \delta p(c)$, $\forall c \in \mathcal{C}$.

$$\begin{aligned}\mathbb{P} \left(\exists c \in \mathcal{C} \mid |R_n(c) - R(c)| > \sqrt{\frac{\ln\left(\frac{2}{\delta(c)}\right)}{2n}} \right) &= \mathbb{P} \left(\cup_{c \in \mathcal{C}} \left\{ |R_n(c) - R(c)| > \sqrt{\frac{\ln\left(\frac{2}{\delta(c)}\right)}{2n}} \right\} \right) \\ &\leq \sum_{c \in \mathcal{C}} \mathbb{P} \left(|R_n(c) - R(c)| > \sqrt{\frac{\ln\left(\frac{2}{\delta(c)}\right)}{2n}} \right) \leq \sum_{c \in \mathcal{C}} \delta(c) = \sum_{c \in \mathcal{C}} \delta p(c) = \delta\end{aligned}$$

A bound depending on a distribution on classifiers $p(c) > 0$ and $\sum_{c \in \mathcal{C}} p(c) = 1$.

$$\text{With probability at least } 1 - \delta, |R_n(c) - R(c)| \leq \sqrt{\frac{\ln\left(\frac{2}{\delta(c)}\right)}{2n}} = \sqrt{\frac{\ln\left(\frac{1}{p(c)}\right) + \ln\left(\frac{2}{\delta}\right)}{2n}}.$$

Note that we get back the finite case's bound by using a uniforme probability $p(c) = \frac{1}{N}$.



Figure: Vladimir Vapnik & Yann Lecun

Vapnik-Chervonenkis

For any $\delta > 0$, with probability at least $1 - \delta$,

$$\forall c \in \mathcal{C}, R(c) \leq R_n(c) + 2\sqrt{2 \frac{\ln(S_{\mathcal{C}}(2n)) + \ln\left(\frac{2}{\delta}\right)}{n}}.$$

with $S_{\mathcal{C}}(n) = \sup_{(x_i, y_i)_{i=1}^n \in (\mathbb{R} \times \{y_1, y_2\})^n} \# \{(f_c(x_i, y_i))_{i=1}^n \in \{0, 1\}^n \mid c \in \mathcal{C}\}$, the maximum number of ways into which n points can be classified by the function class \mathcal{F} .

Since $S_{\mathcal{C}}(n) = \#A$ with $A \subset \{0, 1\}^n$, $S_{\mathcal{C}}(n) \leq 2^n$.

VC dimension

Largest n such that $S_{\mathcal{C}}(n) = 2^n$

Examples

The set of half-planes in \mathbb{R}^d can shatter a set of $d + 1$ points but no set of $d + 2$ points: the VC dimension is $d + 1$

Lemma – Symmetrization

For any $t > 0$, such that $nt^2 \geq 2$,

$$\mathbb{P} \left(\sup_{c \in \mathcal{C}} (R(c) - R_n(c)) \geq t \right) \leq 2\mathbb{P} \left(\sup_{c \in \mathcal{C}} (R_n(c) - R'_n(c)) \geq \frac{t}{2} \right),$$

where $R'_n(c) = \frac{1}{n} \sum_{i=1}^n \mathbb{1}_{\{c(X'_i) \neq Y'_i\}}$ is computed with an extra dataset $(X'_i, Y'_i)_{i=1}^n$ (called ghost dataset) independent from $(X_i, Y_i)_{i=1}^n$ (used for $R_n(c)$).

Proof – 1

For $c \in \mathcal{C}$, “ $R(c) - R_n(c) \geq t$ and $R'_n(c) - R(c) \geq -\frac{t}{2}$ ” $\Rightarrow R'_n(c) - R_n(c) \geq \frac{t}{2}$,

$$\text{i.e. } \left\{ R(c) - R_n(c) \geq t \right\} \cap \left\{ R(c) - R'_n(c) \leq -\frac{t}{2} \right\} \subseteq \left\{ R'_n(c) - R_n(c) \geq \frac{t}{2} \right\},$$

or also $\mathbb{1}_{\{R(c) - R_n(c) \geq t\}} \mathbb{1}_{\{R(c) - R'_n(c) \leq -\frac{t}{2}\}} \leq \mathbb{1}_{\{R'_n(c) - R_n(c) \geq \frac{t}{2}\}}.$

Lemma – Symmetrization

$$\mathbb{P} \left(\sup_{c \in \mathcal{C}} (R(c) - R_n(c)) \geq t \right) \leq 2\mathbb{P} \left(\sup_{c \in \mathcal{C}} (R_n(c) - R'_n(c)) \geq \frac{t}{2} \right).$$

Proof – 2

For $c \in \mathcal{C}$, $\mathbb{1}_{\{R(c) - R_n(c) \geq t\}} \mathbb{1}_{\{R(c) - R'_n(c) \leq \frac{t}{2}\}} \leq \mathbb{1}_{\{R'_n(c) - R_n(c) \geq \frac{t}{2}\}}.$

Let us assume that c depends on $(X_i, Y_i)_{i=1}^n$ but not on the ghost dataset $(X'_i, Y'_i)_{i=1}^n$. By applying the conditional expectation, conditioned by the variables $(X_i, Y_i)_{i=1}^n$,

$$\mathbb{1}_{\{R(c) - R_n(c) \geq t\}} \mathbb{P} \left(R(c) - R'_n(c) \leq \frac{t}{2} \mid (X_i, Y_i)_{i=1}^n \right) \leq \mathbb{P} \left(R'_n(c) - R_n(c) \geq \frac{t}{2} \mid (X_i, Y_i)_{i=1}^n \right),$$

since $\mathbb{E}[f(X)Z \mid X] = f(X)\mathbb{E}[Z \mid X]$ (see Prop. 1.31 in [GS20]).

Lemma – Symmetrization

$$\mathbb{P} \left(\sup_{c \in \mathcal{C}} (R(c) - R_n(c)) \geq t \right) \leq 2\mathbb{P} \left(\sup_{c \in \mathcal{C}} (R_n(c) - R'_n(c)) \geq \frac{t}{2} \right).$$

Proof – 3

$$\mathbb{1}_{\{R(c) - R_n(c) \geq t\}} \mathbb{P}(R(c) - R'_n(c) \leq \frac{t}{2} \mid (X_i, Y_i)_{i=1}^n) \leq \mathbb{P}(R'_n(c) - R_n(c) \geq \frac{t}{2} \mid (X_i, Y_i)_{i=1}^n).$$

In addition, using Chebyshev's inequality (slide 95)

$$\begin{aligned} & \mathbb{P}(R(c) - R'_n(c) > \frac{t}{2} \mid (X_i, Y_i)_{i=1}^n) \\ & \leq \mathbb{P}(|R(c) - R'_n(c)| \geq \frac{t}{2} \mid (X_i, Y_i)_{i=1}^n) \leq \frac{\text{Var}(R'_n(c))}{\left(\frac{t}{2}\right)^2} = \frac{4\text{Var}(\mathbb{1}_{\{c(X') \neq Y'\}})}{nt^2} \leq \frac{1}{nt^2} \end{aligned}$$

(since the variance of a random variable with range [0, 1] is less than $\frac{1}{4}$: $\mathbb{E}[X^2] - (\mathbb{E}[X])^2 \leq \mathbb{E}[X](1 - \mathbb{E}[X]) \leq \frac{1}{4}$).

$$\Rightarrow \mathbb{1}_{\{R(c) - R_n(c) \geq t\}} \left(1 - \frac{1}{nt^2}\right) \leq \mathbb{P}\left(R'_n(c) - R_n(c) \geq \frac{t}{2} \mid (X_i, Y_i)_{i=1}^n\right).$$

Lemma – Symmetrization

For any $t > 0$, such that $nt^2 \geq 2$,

$$\mathbb{P} \left(\sup_{c \in \mathcal{C}} (R(c) - R_n(c)) \geq t \right) \leq 2\mathbb{P} \left(\sup_{c \in \mathcal{C}} (R_n(c) - R'_n(c)) \geq \frac{t}{2} \right).$$

Proof – 4

$$\mathbb{1}_{\{R(c) - R_n(c) \geq t\}} \left(1 - \frac{1}{nt^2} \right) \leq \mathbb{P} (R'_n(c) - R_n(c) \geq \frac{t}{2} \mid (X_i, Y_i)_{i=1}^n).$$

$$\text{In addition, } nt^2 \geq 2 \Rightarrow \left(1 - \frac{1}{nt^2} \right) \geq \frac{1}{2}$$

$$\Rightarrow \mathbb{1}_{\{R(c) - R_n(c) \geq t\}} \leq 2\mathbb{P} (R'_n(c) - R_n(c) \geq \frac{t}{2} \mid (X_i, Y_i)_{i=1}^n).$$

By applying the expectation $\mathbb{E}[.]$,

$$\Rightarrow \mathbb{P}(R(c) - R_n(c) \geq t) \leq 2\mathbb{P} \left(R'_n(c) - R_n(c) \geq \frac{t}{2} \right).$$

We get the “ $\sup_{c \in \mathcal{C}}$ ” by using increasing sequences of events $\{R'_n(c) - R_n(c) \geq \frac{t}{2}\}$ and $\{R(c) - R_n(c) > t\}$ (Prop. 1.1 in [GS20]) whose limits are the “ $\sup_{c \in \mathcal{C}}$ ” with classifiers depending on $(X_i, Y_i, X'_i, Y'_i)_{i=1}^n$ on the right, and depending on $(X_i, Y_i)_{i=1}^n$ on the left. \square

Vapnik-Chervonenkis

For any $\delta > 0$, with probability at least $1 - \delta$,

$$\forall c \in \mathcal{C}, R(c) \leq R_n(c) + 2\sqrt{2 \frac{\ln(S_{\mathcal{C}}(2n)) + \ln\left(\frac{2}{\delta}\right)}{n}}.$$

with $S_{\mathcal{C}}(n)$ the maximum number of ways into which n points can be classified by \mathcal{C} :

$$S_{\mathcal{C}}(n) = \max_{(x_i, y_i)_{i=1}^n} \#\mathcal{F}\left((x_i, y_i)_{i=1}^n\right) \text{ with } \mathcal{F}\left((x_i, y_i)_{i=1}^n\right) = \left\{ (f_c(x_i, y_i))_{i=1}^n \in \{0, 1\}^n \mid c \in \mathcal{C} \right\}.$$

Proof – 1

$$\begin{aligned} \text{Note that } & \left\{ \sup_{c \in \mathcal{C}} (R_n(c) - R'_n(c)) \geq \frac{t}{2} \right\} = \left\{ \sup_{c \in \mathcal{C}} \frac{1}{n} \sum_{i=1}^n (f_c(X_i, Y_i) - f_c(X'_i, Y'_i)) \geq \frac{t}{2} \right\} \\ & = \left\{ \exists c \in \mathcal{C} \mid \frac{1}{n} \sum_{i=1}^n (f_c(X_i, Y_i) - f_c(X'_i, Y'_i)) \geq \frac{t}{2} \right\} = \cup_{c \in \mathcal{C}} \left\{ (R_n(c) - R'_n(c)) \geq \frac{t}{2} \right\} \\ & = \cup_{f \in \mathcal{F}\left((X_i, Y_i)_{i=1}^n, (X'_i, Y'_i)_{i=1}^n\right)} \left\{ \frac{1}{n} \sum_{i=1}^n (f_i - f_{i+n}) \geq \frac{t}{2} \right\}. \end{aligned}$$

Vapnik-Chervonenkis

With probability $1 - \delta$, $\forall c \in \mathcal{C}$, $R(c) \leq R_n(c) + 2\sqrt{2\frac{\ln(S_{\mathcal{C}}(2n)) + \ln(\frac{2}{\delta})}{n}}$.

Proof – 2

By using the Symmetrization's Lemma (slide 124),

$$\begin{aligned} \mathbb{P}\left(\sup_{c \in \mathcal{C}} (R(c) - R_n(c)) \geq t\right) &\leq 2\mathbb{P}\left(\sup_{c \in \mathcal{C}} (R_n(c) - R'_n(c)) \geq \frac{t}{2}\right) \\ &= 2\mathbb{P}\left(\bigcup_{f \in \mathcal{F}((X_i, Y_i)_{i=1}^n, (X'_i, Y'_i)_{i=1}^n)} \left\{ \frac{1}{n} \sum_{i=1}^n (f_i - f_{i+n}) \geq \frac{t}{2} \right\}\right) \\ &\leq 2\mathbb{P}\left(\bigcup_{f \in \mathcal{F}((X_i, Y_i)_{i=1}^n, (X'_i, Y'_i)_{i=1}^n)} \left\{ \frac{1}{n} \sum_{i=1}^n (f_i - f_{i+n}) \geq \frac{t}{2} \right\}\right) \end{aligned}$$

□

Institut Supérieur de l'Aéronautique et de l'Espace

10 avenue Édouard Belin – BP 54032
31055 Toulouse Cedex 4 – France
Phone: +33 5 61 33 80 80

www.isae-sup Aero.fr