DISCLAIMER

These notes are supposed to explain some major topics covered in the lessons.

While they may not provide an exhaustive overview, they strive to complement the information presented in the slides with some explanations taken from additional readings in various textbooks.

If someone finds some errors or have any suggestion, it's possible to contact me on mmeschini001@gmail.com

MACHINE LEARNING

Why machine learning? Many interesting problems are too complex to admit an algorithmic solution or even a complete description. For these problems only data are available. Machine learning is about using data to solve problems.

Machine learning jargon

MODEL: a mathematical or computational representation of the real world

TRAINING DATA: the portion of the dataset used to train the machine learning model

TEST DATA: the portion of the dataset used to evaluate the performance of a trained machine learning model

FEATURE/ATTRIBUTE/VARIABLES: an individual measurable property used to train the machine learning model

OBSERVATIONS/SAMPLES/ISTANCES: are individual data points that represent specific records. Typically, they correspond to a single entry

CLASS/LABEL: the output or target variable that a machine learning model is trying to predict

OVERFITTING: when a machine learning model learns the training data too well, capturing noise and irrelevant details, which can lead to poor generalization to new data (typically, too complex model)

UNDERFITTING: the opposite of overfitting, where a model is too simple to capture the underlying patterns in the data, resulting in poor performance

Country	Product	Units Sold	Manufacturing Pric	Sale Price	Gross Sales
Canada	Carretera	1618.5	$ 3.00	$ 20.00	$ 32,370.00
Germany	Carretera	1321	$ 3.00	$ 20.00	$ 26,420.00
France	Carretera	2178	$ 3.00	$ 15.00	$ 32,670.00
Germany	Carretera	888	$ 3.00	$ 15.00	$ 13,320.00
Mexico	Carretera	2470	$ 3.00	$ 15.00	$ 37,050.00
Germany	Carretera	1513	$ 3.00	$ 350.00	$ 529,550.00
Germany	Montana	921	$ 5.00	$ 15.00	$ 13,815.00
Canada	Montana	2518	$ 5.00	$ 12.00	$ 30,216.00
France	Montana	1899	$ 5.00	$ 20.00	$ 37,980.00
Germany	Montana	1545	$ 5.00	$ 12.00	$ 18,540.00
India	Lassi	$ 36,577.00	$ 4.99	$ 11.00	$ 250,000.00

There are three different types of machines learning:

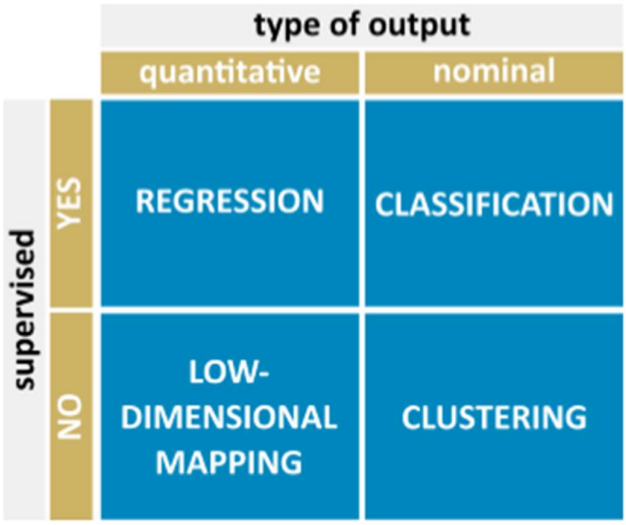
- **Supervised learning**: the algorithm "learns" from the training dataset by iteratively making predictions on the data and adjusting its parameters to give the correct answer. While supervised learning models tend to be more accurate than unsupervised learning models, they require upfront human intervention to label the data appropriately.
- **Unsupervised learning**: Unsupervised learning models, in contrast, work on their own to discover the inherent structure of unlabelled data. Note that they still require some human intervention for validating output variables.

- **Reinforcement learning:** algorithms work by training the agent through interaction with the environment. They learn to optimize their policy over time, improving their decision-making capabilities.

The types of quantity our model want to learn can be of two different types:

- Real values
- Categorical/nominal values: e.g. colour, name. (there isn't a natural ordering)

Depending on the model and the output variable types we can distinguish:



Possible scenarios

There are three possible scenarios:

1) Useful mostly for reasoning: we have a **learning machine** that is **fixed** and also the **data** are **fixed** (we have only the training set). The objective is to find the correct learners.
2) Not realizable: we **know the model that generates data** (the probabilities are known), so the objective is to find the optimal learner. Useful for theory.
3) Our usual situation: d**ata** are **stochastic**, the **learner** is **fixed** and chosen in advance. The objective is to find the best configuration of learning machine which is correct for any realization of the data.

The models will typically use **inductive learning.** This type of learning is based on examples and data, our objective is to learn a function **for every possible input**, given only a **finite set of examples input** -> **GENERALIZATION**

Induction works only if we have an a-priori knowledge or assumption. EX:

Problem	Bias
Classification	Points in the same class are close to each other
Regression	Nearby points don't have very different outputs
Clustering	There are k compact clusters in the data
Mapping	The data are organized in a low-dimensional shape within the d-dimensional data space

Probability

The probability quantifies the chance of an event or outcome occurring. It provides a way to express uncertainty to which something is likely to happen. Something that may or may not happen is called an **outcome (ω)**. All possible outcomes constitute the **sample space (Ω)**. An **event** refers to a specific outcome or a subset of **Ω.**

We defined as P(A) the probability of event A.

Axioms of probability:

1. $P(A) \geq 0$
2. $\sum_{i=1}^{N} P(\omega_i) = 1$, or $P(\Omega) = 1$
3. If A_1 and A_2 are mutually exclusive events (viewed as sets: if they are **disjoint** = have zero intersection), then $P(A_1 \textbf{ or } A_2) = P(A_1 \cup A_1) = P(A_1) + P(A_1)$

A **random variable (X)** is a numerical variable that doesn't have a fixed value but changes according to a given probability law. The **Cumulative distribution function (CDF)** of a random variable X is the function given by:

$$F_X(x) = \mathrm{P}(X \leq x) \quad \text{(Eq.1)}$$

Where the right-hand side represents the probability that the random variable X takes on a value less than or equal to x.

The **probability mass function** is a function that gives the probability that a **discrete** random variable is exactly equal to some value.

$$p_X(x) = P(X = x)$$

The **probability density function** is a function whose value at any given sample in the sample space can be interpreted as providing a relative likelihood that the value of the random variable would be equal to that sample.

A function $f_{\mathscr{X}}$ such that

$$P_{\mathscr{X}}(\hat{x}) = \int_{-\infty}^{\hat{x}} f_{\mathscr{X}}(x)\,\mathrm{d}x$$

is the **probability density function** of $\mathscr{X}$.

Then, we define the **conditional probability P(E | F)** as the probability of an event E given the knowledge that another event F has occurred.

Two events are **independent** if the outcome of one event does not influence the outcome of the second event.

Bayesian decision theory

Bayesian decision theory is a fundamental statistical approach to the problem of pattern classification. This approach is based on quantifying the trade-offs between various classification decisions using probability and the costs that accompany such decisions. It makes the assumption that the decision problem is posed in probabilistic terms, and that **all the relevant probability values are known.**

We define as **state of nature (ω)** *(t in professor slide)* the different possible states or scenarios that a system could be in (e.g. salmon and sea bass are two different state of nature). We define **a priori probability** as the probability of an event occurring based on prior knowledge. It is represented by the likelihood in the occurrence of an event before any new evidence is taken into account. Then, is important to define the **class-conditional probability density function** expressed as **P(x|ω)** that represents the distribution of the random variable x depending on the state of nature.

The probability density of finding a pattern that is in category $\boldsymbol{\omega_j}$ and has a feature value x, can be written in two ways: $\boldsymbol{P(\omega_j, x) = P(\omega_j|x) * P(x) = P(x|\omega_j) * P(\omega_j)}$, rearranging this leads us to the **Bayes' formula:**

$$\boxed{P(\omega_j|x) = \frac{p(x|\omega_j)P(\omega_j)}{p(x)},} \qquad posterior = \frac{likelihood \times prior}{evidence}.$$

Bayes' formula shows that by observing the value of x we can convert the prior probability $\boldsymbol{P(\omega_j)}$ to the **a posteriori probability $\boldsymbol{P(\omega_j|x)}$,** that is the probability of the state of nature being ω_j given that the feature value x has been measured.

Oss: the evidence can be viewed as a scale factor that guarantees that the posterior probability sums to one.

In this context, is useful to introduce the **decision rule** that represents a criterion (a set of boundaries) used to minimize the cost of our decision. To evaluate each decision, we use the **expected loss,** also called **risk.** The **loss function ($\boldsymbol{\lambda}$)** states exactly how costly each action is and is used to convert a probability determination into a decision. For a generic decision y_i the risk is:

$$R(y_i) = \sum_{j=1}^{c} \lambda(y_i, t_j) P(t_j)$$

Finally, the **conditional risk** represents the expected loss associated a decision under a specific state of nature.

$$R(y_i \,|\, \mathbf{x}) = \sum_{j=1}^{c} \lambda(y_i, t_j) P(t_j \,|\, \mathbf{x})$$

$R(y_i \,|\, \mathbf{x})$ is the **conditional risk** of decision y_i
when we have the **experimental observation** $\mathbf{x}$

Classification is a decision problem where there is no decision to take and we want only to recognize the state of nature. In short, the model tries to predict the correct class of a given input data. A classifier is a rule that **receives an observation x** and **outputs a class y(x).**

Naive Bayes classifier

A Naive Bayes classifier is a simple and probabilistic machine learning algorithm that is often used for classification tasks. It is based on Bayes' theorem and makes a "naïve" assumption that features used to describe data are conditionally independent.

Naïve Bayes is a conditional probability model: it assigns probabilities $P(C_k|x_1, \dots, x_n)$ for each of the K possible classes C_k (= state of natures) given a problem instance to be classified represented by a vector $\boldsymbol{x} = (x_1, \dots, x_n)$ encoding some n features. From the Bayes' theorem

$$p(C_k \mid \mathbf{x}) = \frac{p(C_k)\, p(\mathbf{x} \mid C_k)}{p(\mathbf{x})}$$

In practice, there is interest only in the numerator of that fraction, because the denominator does not depend on C and the values of the features x_i are given, so that the denominator is effectively constant. The numerator is equivalent to the joint probability model

$$p(C_k, x_1, \dots, x_n)$$

which can be rewritten as follows, using the chain rule for repeated applications of the definition of conditional probability:

$$\begin{aligned} p(C_k, x_1, \dots, x_n) &= p(x_1, \dots, x_n, C_k) \\ &= p(x_1 \mid x_2, \dots, x_n, C_k)\, p(x_2, \dots, x_n, C_k) \\ &= p(x_1 \mid x_2, \dots, x_n, C_k)\, p(x_2 \mid x_3, \dots, x_n, C_k)\, p(x_3, \dots, x_n, C_k) \\ &= \cdots \\ &= p(x_1 \mid x_2, \dots, x_n, C_k)\, p(x_2 \mid x_3, \dots, x_n, C_k) \cdots p(x_{n-1} \mid x_n, C_k)\, p(x_n \mid C_k)\, p(C_k) \end{aligned}$$

Now the "naive" conditional independence assumptions come into play: assume that all features in $\mathbf{x}$ are mutually independent, conditional on the category C_k. Under this assumption,

$$p(x_i \mid x_{i+1}, \dots, x_n, C_k) = p(x_i \mid C_k)\,.$$

Thus, the joint model can be expressed as

$$\begin{aligned} p(C_k \mid x_1, \dots, x_n) \propto\ & p(C_k, x_1, \dots, x_n) \\ &= p(C_k)\, p(x_1 \mid C_k)\, p(x_2 \mid C_k)\, p(x_3 \mid C_k) \cdots \\ &= p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)\,, \end{aligned}$$

The decision rule for minimizing the probability of error is:

Decide ω_1 *if* $P(\omega_1|\, x) > P(\, \omega_2|x)$; *otherwise decide* ω_2

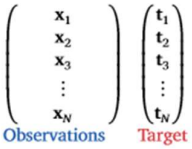
The loss function for this classifier il **zero-one:** $\lambda(y \mid t) = \begin{cases} 0 & \text{if } y = t \\ 1 & \text{if } y \neq t \end{cases}$

So, all types of errors have the same cost (=1) and correct classifications don't have a cost.

Linear Regression

When only data are available is possible to find the correct output using a regression model.
Regression means approximating a functional dependency based on measured data.

It is a supervised problem given that we have observations and targets.



We are looking for a linear model y(x) that predicts t given x.
Since the model is linear the form will be: $y(x) = w_0 + w_1 \cdot x$

From the training set is possible to estimate the parameters w. Difficulty is possible to find values that are good for all points, so the solution is to find two values that minimize the average error. The most popular estimation method is the **square error loss**, in which we pick the coefficients w to minimize the following loss function:

$$\lambda_{SE}(y,t) = (y-t)^2 \quad \xrightarrow{\text{So, our objective function will be}} \quad J_{MSE} = \frac{1}{N}\sum_{l=1}^{N}(y_l - t_l)^2$$

- This function when building a model (training) is a function of the **parameters** of the model and the data are fixed.
- When using a model (inferencing) is function of the **data**, while the parameters are now fixed.

The least squares solution of the one-dimensional linear regression problem (for the model $y(x) = w \cdot x$) is:

$$w = \frac{\sum_{l=1}^{N} x_l t_l}{\sum_{l=1}^{N} x_l^2}$$

While, for the model $y(x) = w_0 + w_1 \cdot x$, where is also present the offset, is:

$$\bar{x} = \frac{1}{N}\sum_{l=1}^{N} x_l \qquad \bar{t} = \frac{1}{N}\sum_{l=1}^{N} t_l$$

Note:

w_1 is called: slope, gain

w_0 is called intercept, offset, bias

$$w_1 = \frac{\sum_{l=1}^{N}(x_l - \bar{x})(t_l - \bar{t})}{\sum_{l=1}^{N}(x_l - \bar{x})^2}$$

$$w_0 = \bar{t} - w_1\bar{x}$$

Multidimensional linear regression problem

The data is now composed of d-dimensional vectors $\mathbf{x}_1 = [\, x_{1,1}, x_{1,2}, \ldots, x_{1,d}]$ and we can organize them into a N x d matrix. Now we also have d parameters:

$$X = \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \\ \vdots \\ \mathbf{x}_N \end{pmatrix} = \begin{pmatrix} x_{1,1} & x_{1,2} & \ldots & x_{1,d} \\ x_{2,1} & x_{2,2} & \ldots & x_{2,d} \\ x_{3,1} & x_{3,2} & \ldots & x_{3,d} \\ & & \vdots & \\ x_{N,1} & x_{N,2} & \ldots & x_{N,d} \end{pmatrix} \qquad \mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_d \end{pmatrix}$$

The linear model takes the d inputs of each observation and combines them by using the d parameters to produce one output:

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_N \end{pmatrix} = \begin{pmatrix} x_{1,1} & x_{1,2} & \ldots & x_{1,d} \\ x_{2,1} & x_{2,2} & \ldots & x_{2,d} \\ x_{3,1} & x_{3,2} & \ldots & x_{3,d} \\ & & \vdots & \\ x_{N,1} & x_{N,2} & \ldots & x_{N,d} \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_d \end{pmatrix} = \begin{pmatrix} x_{1,1}w_1 + x_{1,2}w_2 + & \ldots & +x_{1,d}w_d \\ x_{2,1}w_1 + x_{2,2}w_2 + & \ldots & +x_{2,d}w_d \\ x_{3,1}w_1 + x_{3,2}w_2 + & \ldots & +x_{3,d}w_d \\ & & \vdots \\ x_{N,1}w_1 + x_{N,2}w_2 + & \ldots & +x_{N,d}w_d \end{pmatrix} = X\mathbf{w}$$

The affine case let us incorporate the additive parameter w_0 by adding one constant column to the data matrix

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_N \end{pmatrix} = \begin{pmatrix} 1 & x_{1,1} & x_{1,2} & \ldots & x_{1,d} \\ 1 & x_{2,1} & x_{2,2} & \ldots & x_{2,d} \\ 1 & x_{3,1} & x_{3,2} & \ldots & x_{3,d} \\ & & & \vdots & \\ 1 & x_{N,1} & x_{N,2} & \ldots & x_{N,d} \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_d \end{pmatrix} = \begin{pmatrix} w_0 + x_{1,1}w_1 + x_{1,2}w_2 + & \ldots & +x_{1,d}w_d \\ w_0 + x_{2,1}w_1 + x_{2,2}w_2 + & \ldots & +x_{2,d}w_d \\ w_0 + x_{3,1}w_1 + x_{3,2}w_2 + & \ldots & +x_{3,d}w_d \\ & & \vdots \\ w_0 + x_{N,1}w_1 + x_{N,2}w_2 + & \ldots & +x_{N,d}w_d \end{pmatrix} = X\mathbf{w}$$

Our final goal is to make this model's prediction $\boldsymbol{y}$ as similar as possible to the measured outputs for each observation

$$\mathbf{t} = \begin{pmatrix} t_1 \\ t_2 \\ t_3 \\ \vdots \\ t_N \end{pmatrix}$$

The solution in this case is: $\mathbf{w} = (X^\intercal X)^{-1} X^\intercal \mathbf{t}$

Optimisation

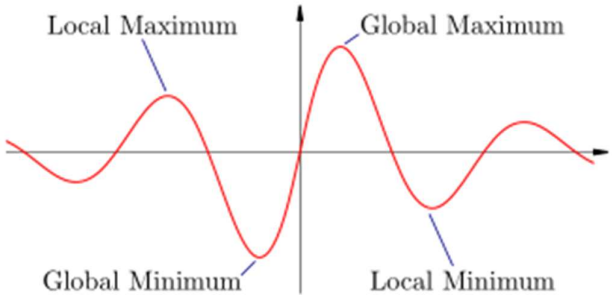
The target of optimisation is finding the extrema of an objective function J(**w**), where $J: S \subset \mathbb{R}^m \rightarrow \mathbb{R}$. An extremum is a point $\boldsymbol{w}^* \in S$ that may be a maximum or minimum.

A point $\boldsymbol{w}^*$ is a **minimum** if there is a neighbourhood $R \subseteq S$, where the following holds:

$$J(\boldsymbol{w}) \geq J(\boldsymbol{w}^*)\ \forall \boldsymbol{w}\ \in R$$

A point $\boldsymbol{w}^*$ is a **maximum** if $\leq$ is used.

An extremum is **relative/local** if R is strictly contained is S, or in another words if it is a minimum only in a local neighbourhood. An extremum is **absolute/global** if R = S. Finally, an extremum is **isolated** if we can draw a sphere around $\boldsymbol{w}^*$ and there are no other points there with function values greater (for a maximum) or smaller (for a minimum) than the value at that point.



Typical objective function in machine learning

- **Cost function:** which is inversely related to how much we like a solution (higher cost, less quality of the solution)
- **Risk** (expected loss): $J(\mathbf{w}) = \int_{\mathscr{X}} \lambda(\mathbf{t}, \mathbf{y}(\mathbf{w}))\, p(\mathbf{x}) d\mathbf{x}$
- **Likelihood:** which is a common function that we have to maximize

We define as **optimal solution** an extremum $\boldsymbol{w}^*$ of *J,* and the optimal value will be $J(\boldsymbol{w}^*)$. Then, we can have a **feasible solution** which is any point **w** that satisfies all hypotheses of the optimisation problem. Moreover, another important aspect is the convexity of a function.

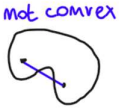
Convex sets

A set $S \subset \mathbb{R}^m$ is convex if and only if, for any $\theta \in [0,1]$,

$$\forall \mathbf{v}, \mathbf{w} \in S \Rightarrow \theta \mathbf{v} + (1-\theta)\mathbf{w} \in S$$

more generally if for any $\theta_1 > 0, \ldots, \theta_n > 0$ such that $\sum_k \theta_k = 1$

$$\forall \mathbf{v}_1, \ldots, \mathbf{v}_n \in S \Rightarrow \sum_k \theta_k \mathbf{v}_k \in S$$ **Convex combination**

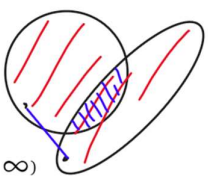


Properties:

- $\mathbf{v} \in \mathbb{R}^m$ (a single point) is convex
- $\emptyset = \{\}$ (the empty set) is convex
- $\mathbb{R}^m$ is convex

and if S_1 and S_2 are convex, then

- $S_1 \cap S_2$ is convex
- $S_1 \cup S_2$ IS NOT NECESSARILY convex $(o + o = \infty)$



↳ Union can be not convex

Convex functions

A function $J: S \subseteq \mathbb{R}^m \rightarrow \mathbb{R}$ is convex if S is a convex set and if $\forall \mathbf{v}, \mathbf{w} \in S$, and with $0 \leq \theta \leq 1$:

$$J(\theta \mathbf{v} + (1-\theta)\mathbf{w}) \leq \theta J(\mathbf{v}) + (1-\theta) J(\mathbf{w})$$

i.e., if its epigraph is a convex set

more generally if for any $\theta_1 > 0, \ldots, \theta_n > 0$ such that $\sum_k \theta_k = 1$

$$\forall \mathbf{v}_1, \ldots, \mathbf{v}_n \in S \Rightarrow J\left(\sum_k \theta_k \mathbf{v}_k\right) \leq \sum_k \theta_k J(\mathbf{v}_k)$$

J is concave if $-J$ is convex

The convexity is a good thing because it ensures the **uniqueness of extrema** and **the convergence of iterative algorithms.** While, for non-convex problems we cannot be sure whether an extremum is absolute or relative.

Types of optimisation problems

The optimisation problems can be of different types:

- **Discrete**: if $S \in \mathbb{N}^m$, or more generally there are countable variables
- **Continuous**: $S \in \mathbb{R}^m$
- **Constrained:** if we want to optimize an objective function with respect to some variables in the presence of constraints on those variables

  Constraints are expressed by **constraint functions**:
  - **Equality constraints:** $f_1(\mathbf{w}) = 0, f_2(\mathbf{w}) = 0, \dots f_k(\mathbf{w}) = 0$
  - **Inequality constraints:** $f_1(\mathbf{w}) \geq 0, f_2(\mathbf{w}) \geq 0, \dots f_k(\mathbf{w}) \geq 0$
- **Unconstrained**
- **Smooth:** if the objective function is differentiable
- **Nonsmooth:** if the objective function is not differentiable
- **Linear** (typically needs a constraint, otherwise the objective can increase/decrease indefinitely)
- **Convex/Nonconvex**

Useful for this type of problems is the gradient, that indicates the direction of maximum increase.

$$\nabla J(\mathbf{w}) = \begin{bmatrix} \frac{\partial J(\mathbf{w})}{\partial w_1} \\ \frac{\partial J(\mathbf{w})}{\partial w_2} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial w_m} \end{bmatrix}$$

- Derivative → rate of growth of a function of a scalar variable
- Negative sign → decreasing

- Gradient *length (norm)* → rate of maximum growth
- *Direction → direction of maximum growth*

Then, the Hessian matrix $H_J(\mathbf{w}) : \mathbb{R}^m \to \mathbb{R}^m \times \mathbb{R}^m$ s.t. $H_J(\mathbf{w})_{ij} = \frac{\partial^2 J(\mathbf{w})}{\partial w_i \partial w_j}$

$$H = \begin{pmatrix} \frac{\partial^2 J}{\partial w_1^2} & \frac{\partial^2 J}{\partial w_1 \partial w_2} & \frac{\partial^2 J}{\partial w_1 \partial w_3} & \cdots & \frac{\partial^2 J}{\partial w_1 \partial w_m} \\ \frac{\partial^2 J}{\partial w_2 \partial w_1} & \frac{\partial^2 J}{\partial w_2^2} & \frac{\partial^2 J}{\partial w_2 \partial w_3} & \cdots & \frac{\partial^2 J}{\partial w_2 \partial w_m} \\ \frac{\partial^2 J}{\partial w_3 \partial w_1} & \frac{\partial^2 J}{\partial w_3 \partial w_2} & \frac{\partial^2 J}{\partial w_3^2} & \cdots & \frac{\partial^2 J}{\partial w_2 \partial w_m} \\ & & \vdots & & \\ \frac{\partial^2 J}{\partial w_m \partial w_1} & \frac{\partial^2 J}{\partial w_m \partial w_2} & \frac{\partial^2 J}{\partial w_m \partial w_3} & \cdots & \frac{\partial^2 J}{\partial w_m^2} \end{pmatrix}$$

The matrix H is **symmetric**

Finally, the **Taylor polynomials** centred around w_0:

$$J(w) \approx J(w_0) + J'(w)\,|_{w=w_0}\,(w - w_0) + \frac{1}{2} J''(w)\,|_{w=w_0}\,(w - w_0)^2$$

Or for the multidimensional case:

$$J(\mathbf{w}) \approx J(\mathbf{w}_0) + \nabla J(\mathbf{w})\,|_{\mathbf{w}=\mathbf{w}_0}\,(\mathbf{w} - \mathbf{w}_0) + \frac{1}{2}(\mathbf{w} - \mathbf{w}_0)^\intercal H\,|_{\mathbf{w}=\mathbf{w}_0}\,(\mathbf{w} - \mathbf{w}_0)$$

All these tools are useful to characterize the extremum.

To check whether a given point $\mathbf{w}$ is a minimum...

1. Necessary conditions of extremum: gradient $= 0$ in the point of interest
2. Necessary and sufficient condition of local convexity: Hessian $\succeq 0$ at the point of interest
3. Sufficient conditions of local minimum: gradient $= 0$ and Hessian $\succeq 0$ at the point of interest
4. Necessary and sufficient condition of convexity: Hessian $\succeq 0$ everywhere
5. Sufficient conditions of global minimum: gradient $= 0$ and Hessian $\succeq 0$ everywhere

Basic optimisation methods

An important observation is that in general we don't know everything about the objective function (realistic problems are typically too complex). On the following part, it will be proposed some situations and some possible solutions (algorithms) to solve the problems.

- Case 1: we only know the function $J(\boldsymbol{w})$, so it's not possible to use Taylor expansion and the only feasible solution is to **direct search** for minimum. Is possible to use different techniques: simulated annealing, genetic algorithms, particle swarm optimisation, ant colony optimisation. PRO: simple, CONS: we need infinite time available.
- Case 2: we know the function $J(\boldsymbol{w})$ and the gradient $\nabla J(\boldsymbol{w})$, in this case we can use first-order Taylor expansions and then find minimum in local region.

  ### Gradient descent algorithm

  1. Initialize: set $i = 0$; select some $\mathbf{w}_0 = \mathbf{w}_{\text{start}}$
  2. Compute $\nabla J(\mathbf{w}_i)$
  3. Select the appropriate step size η_i
  4. Compute the step $\Delta\mathbf{w}_i = -\eta_i \nabla J(\mathbf{w}_i)$
  5. Perform step $\mathbf{w}_{i+1} \leftarrow \mathbf{w}_i + \Delta\mathbf{w}_i$
  6. Compute convergence test. If necessary, iterate from step 2.

- Case 3: we know the function $J(\boldsymbol{w})$**,** the gradient $\nabla J(\boldsymbol{w})$ and the Hessian H. It's possible to use the second-order Taylor polynomial and then the Newton-Raphson method.
  This method attempts to solve the problem of finding the minimum of a function by constructing a sequence $\{x_k\}$ from an initial guess x_0 that converges to a minimum by using a sequence of second-order Taylor approximations of J around the iterates.
  - Start at x_0
  - At each step τ compute the update as

$$x_{\tau+1} = x_\tau - \frac{J'(x_\tau)}{J''(x_\tau)}$$

  PRO: simple, much faster than gradient descent. CONS: need to compute the Hessian

Statistics

Statistics consists in collecting, organizing, analyzing and interpreting data. Statistical models are mathematical and are based on probability theory. It is also defined as the science of using empirical data to create models based on probability.

Terminology

- **Model**, an approximation of reality that retains only those characteristics that are useful for a certain purpose.
- **Population:** a collection of people, items, events about which we want to make inferences.
- **Sample:** a subset of the population.
- **Observation:** An individual element of the sample.

An important assumption is that our samples will be independent and identically distributed.

Statistical estimates

They typically measure central tendency, assuming that there is some "middle value" around which random fluctuations occur.

- **Expectation,** used under scenario 2: know the model that generates data

  continuous **x:** $\mathrm{E}\{X\} = \int_X \mathbf{x} f_x(\mathbf{x})\, d\mathbf{x}$ discrete **x:** $\mathrm{E}\{X\} = \sum_i \mathbf{x}_i F_X(\mathbf{x}_i)$

- **Mean (empirical expectation),** used under scenarios 1 and 3

  $\mathrm{mean}\{X\} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_i$

- **Median:** the middle value after sorting x in increasing order

  1, 3, 3, **6**, 7, 8, 9 — Median = **6**

  2, 2, 3, **5**, **6**, 8, 8, 9 — Median = (5 + 6) ÷ 2 = **5,5**

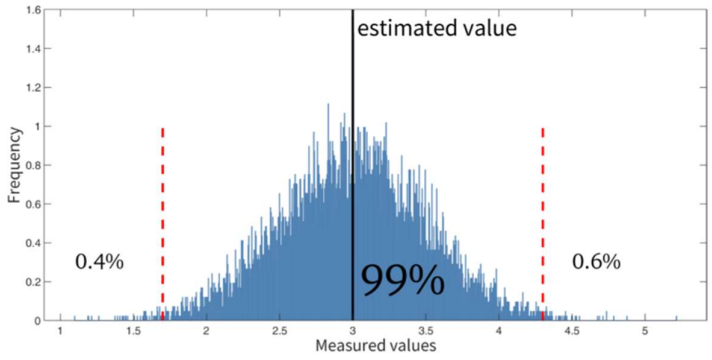
- **Mode:** the value that occurs more frequently
- **Variance:** how much the values vary around the mean

  theoretical $\sigma^2 = \mathrm{E}\{(\mathbf{x} - E\{X\})^2\}$ empirical $\sigma^2 = \frac{1}{N}\sum_{i=1}^{N} (\mathbf{x}_i - \mathrm{mean}\{X\})^2$

- **Standard deviation:** $\sigma = \sqrt{\sigma^2}$
- **Percentile:** measure the position of a specific value within a data set. First, the data are sorted in ascending order, then the percentile of a given data point is computed by finding what percentage of data are lower than that point. For example, the 50th percentile is the value below which 50% of data are lower
- **Confidence intervals:** represent the probability that a sample will be in each interval. Mathematically: given ϵ, the probability that the actual values will differ from the estimated location by more than ϵ is: $\delta = \Pr(|\hat{\mu} - \mu| > \epsilon)$

  $[\hat{\mu} - \epsilon, \hat{\mu} + \epsilon]$ is a **confidence interval** at **level** $1 - \delta$

Example



In this case: $\epsilon = 1.3$ and $\delta = 0.99$

So, with probability 99% the values will be between 1.7 and 4.3

Non-parametric models

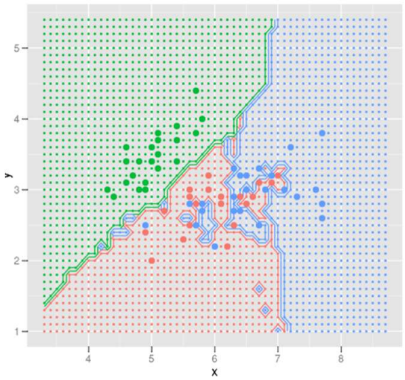
Non-parametric models are a class of models that do not make strong assumptions about the underlying data distribution or the functional form of the relationships between variables. Unlike parametric models, which have a fixed number of parameters and assume a specific data distribution (e.g., linear regression assumes a linear relationship between variables), non-parametric models are more flexible and adaptive. They can capture complex patterns and are particularly useful when data relationships are not well-understood or when assumptions about linearity and distribution are not met. The negative side is that the complexity (model size, number of parameters) is not pre-defined, but depends on the data.

Nearest-neighbour classifier

- We are given a training set $X = \{x_1, x_2, \dots, x_n\}$.
- Then we have a point x^* that we want to classify.
- We search for the nearest point in the training set: $q = \arg\min_i ||x_i - x^*||$; the output will be the class of the nearest point: $y = t_q$

Properties

+ $O(1)$ "learning" complexity (just store the training set!)
− $O(nd)$ classification complexity
+ (sort of) works even for regression, $t \in \mathbb{R}$
+ Theoretical guarantee: For $n \to \infty$, error rate $\leq 2 \cdot$ Bayes error
− May "overfit", i.e., decision regions may have jagged borders



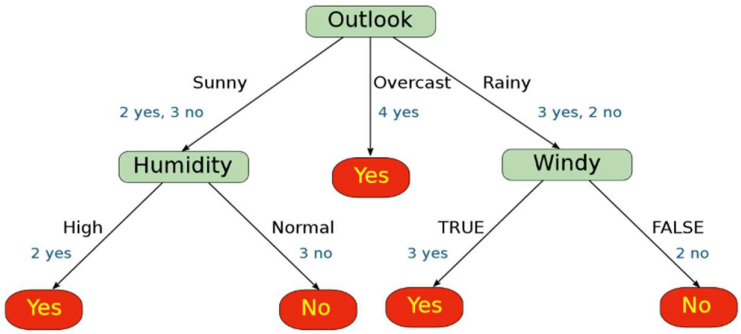
The **K-Nearest-neighbour classifier** is another version of the algorithm in which instead of only one point we select k nearest point. So, $\{n_1, \dots, n_k\} = topk\ ||x_i - x^*||$ and the output is $y = mode\{t_{n_1}, \dots, t_{n_k}\}$ -> it is a majority voting algorithm. Other variants use weighted nearest neighbours, condensed NN and approximated NN.

Decision trees

Assumption: categorical features

- Select one feature.
- Partition of the training set according to its possible values (**levels**).
- **For each** of the parts of the training set so obtained (one per level):
  1. If it contains data from one class only, then decide for that class.
  2. Else if it contains data from more than one class and all features have been used, then decide for the majority class.
  3. Else apply again the same procedure using a different feature.

Outlook	Temperature	Humidity	Windy	Play
overcast	hot	high	FALSE	yes
overcast	cool	normal	TRUE	yes
overcast	mild	high	TRUE	yes
overcast	hot	normal	FALSE	yes
rainy	mild	high	FALSE	yes
rainy	cool	normal	FALSE	yes
rainy	cool	normal	TRUE	no
rainy	mild	normal	FALSE	yes
rainy	mild	high	TRUE	no
sunny	hot	high	FALSE	no
sunny	hot	high	TRUE	no
sunny	mild	high	FALSE	no
sunny	cool	normal	FALSE	yes
sunny	mild	normal	TRUE	yes



Root Node: The top node in the tree, representing the initial input data.
Internal Nodes: These nodes represent questions or conditions on which the data are split. Internal nodes have branches to other internal nodes or leaf nodes.
Leaf Nodes: The terminal nodes that provide the final output or decision. In classification, leaf nodes represent class labels, and in regression, they represent predicted numerical values.

Properties

+ Interpretable (only one feature at a time)
+ Very quick learning
– Suboptimal (a **greedy** algorithm)
– Splits on features with many values are more likely
– With few data splits may be on irrelevant features (noise)
– Not suitable for quantitative (e.g., real-valued) features

Some improvements are the pruning of the tree according to some criterion (for irrelevant splits) and the use for real-valued features by divide value range into sub-ranges (quantization) and use that sub-ranges as categorical features.

DEF: an **ensemble classifier** is composed of **many classifiers**, a **learning procedure** that encourages diversity and an **aggregation rule.**

Random forest

A set of random forest is a set of random trees, trained on randomly resampled training sets. It is a type of **ensemble classifier**. The model is based on the following algorithm:

- Receive one training set X of cardinality n patterns (n observations).
- Create b new training sets of size n by randomly sampling from X with replacement.
- Train one random tree on each of the b new training sets.
- Inference: given one query pattern, classify the pattern with each tree and make the final decision by majority voting among the b individual decisions.

Random resampling with replacement (Bootstrap)

The second point of the algorithm is explained by this: given one training set X of cardinality n, a bootstrap sample is another training set $X^{(i)}$ obtained by randomly sampling with replacement n times from X.

$$X = [A\,B\,C\,D\,E]$$

$$X^{(1)} = [A\,A\,C\,D\,E]$$
$$X^{(2)} = [B\,C\,C\,D\,E]$$
$$X^{(3)} = [A\,A\,B\,D\,D]$$
$$X^{(4)} = [A\,B\,C\,E\,E]$$
$$X^{(5)} = [A\,A\,A\,C\,C]$$

OSS: The decision tree induction rule will tend to select always the most discriminating features, so trees will resemble each other and will not be very independent. To solve this problem is possible to random sampling also for features.

Evaluation of classifiers

It is possible to decompose the expected square error objective J_{mse} as the sum of three terms:

$$J_{mse} = B + V + N$$

- **Bias (B):** the expected difference between the error of the best model and the error of our model *(it suggests problem with model)*
- **Variance (V):** the variance of our model's error *(it suggests problem with data)*
- **Noise (N):** a residual, irreducible quantity.

The bias can be lowered by using a **more adaptable** model -> can cause **overfitting**
The variance can be lowered by using a **less adaptable** model -> cause **underfitting**

So, to sum up, low bias = a good performance on the training set. While a low variance = reliable performance in inference (good generalisation). The statistical problem of model selection consists in choosing the model that generalises best. There are two possible strategies:

1. Evaluate quality empirically (ex post) and use this for designing a new and better learner.
2. Estimate quality theoretically (ex ante) and use this for designing the learner once for all.

Empirical estimates of generalization

Typically, the designer trains the machine using a **training set**, then he evaluates the quality using an independent **validation set**, if the quality is not sufficient, the structure of the model is changed. Finally, the costumer evaluates the model on the **test set.** An important parameter to

considers is the size of the datasets, because small training set leads to overfitting, while with small validation or test set it's not possible to be sure about the performance of the model.

A way to obtain more data to use is given by the resampling methods. Resampling means generating new samples from a given available sample and estimates the confidence intervals, from this is possible to generalize.

Another way is the **cross-validation:** the samples are split into a training set and a test set, then the model is evaluated. This can be done during iterative training, when the test value of the model starts to grow, we are overfitting (stop the procedure).

An alternative to this method is the **leave-one-out cross-validation,** in this case the samples are split in n training sets of size n-1 and corresponding n test sets of size 1. The model is then evaluated on each test obtaining the results for each test. The final estimate of our model is the average of all these one-point estimate. (possible to use the leave-k-out)

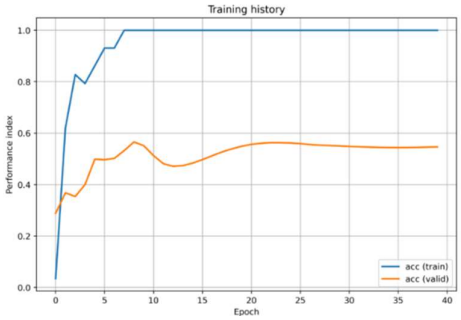


Figure 1:example of cross-validation

Leave-one-out validation algorithm

Input: Data set X of n rows
1: **for** l: 1…n **do**
2: Remove $\mathbf{x}_l$ from X, obtaining training set X_l with $n-1$ rows
3: Train learning machine M_l
4: Compute validation cost $R(\mathbf{x}_l)$ using machine M_l
5: Add $R(\mathbf{x}_l)$ to set S
6: **end for**
7: Train final learning machine M on X
8: Compute statistics on S (average cost, standard deviation of cost, min/max/percentiles of cost, …)
Output: Machine M characterized by statistics over S

Finally, the last method is the **bootstrapping.**

Now is possible to consider indexes to measure the classification quality. Some of them are:

- **Contingency tables**: each entry of the matrix indicates the number of co-occurrences of pairs of events from events set $E_1, E_2, \dots, E_k$ and $F_1, F_2, \dots, F_h$. They are useful to summarize and analyse the relationship between two or more categorical variables.

$$\begin{array}{c} \\ F_1 \\ F_2 \\ \vdots \\ F_h \end{array} \begin{array}{c} \begin{array}{cccc} E_1 & E_2 & \dots & E_k \end{array} \\ \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1k} \\ c_{21} & c_{22} & \dots & c_{2k} \\ & \vdots & & \\ c_{h1} & c_{h2} & \dots & c_{hk} \end{bmatrix} \end{array}$$

Each row and column indicate one category.
The entry c_{ij} represents the number of experiments where events E_i and F_j have been observed together. It's possible to find the frequencies using: $f_{ij} = c_{ij}/N$

- **Confusion matrix (C):** it is a contingency table for **classification outputs** vs **actual classes.**

		Output class: Non-bot	Output class: Bot	
Target class	**Non-bot**	56.3%	3.7%	*93.8 %*
	Bot	0.8%	39.1%	*98.0%*
		98.5%	*91.3%*	**95.5%**

Figure 2:Example with classification model for bot vs human visitors in a web site

The entry c_{ij} represents the number of times that the classifier decided w_j when in fact the true class was w_i

With the confusion matrix is possible to find the number of correct classifications $n_{correct} = trC$ and then the accuracy $f_{correct} = \frac{n_{correct}}{n}$.

The dichotomic case (only two classes)

		Output 0	Output 1
Target	0	NR CORRECT NEG	NR FALSE POS
	1	NR FALSE NEG	NR CORRECT POS

- Cases 00 and 11 are correct classifications
- Case 01 is called **false positive**
- Case 10 is called **false negative**

Note: in statistics event 01 is also called error of the first type and event 10 is called error of the second type

In this situation we can't trust the accuracy alone, in fact suppose to have the following example

Recognize publications written by T. Bayes in the whole Web

There is only one document by Bayes and (suppose) 10^{12} other documents on the Web.

Case 1: All correctly recognized.

$$\text{Accuracy} = \frac{10^{12}+1}{10^{12}+1} = 1$$

Case 2: We always classify as "not-by-Bayes".

$$\text{Accuracy} = \frac{10^{12}}{10^{12}+1} = \frac{1000000000000}{1000000000001} \approx 1$$

❗ We got one of the two classes completely wrong (100% error), but the accuracy was almost perfect. So, it's useful to introduce new indexes:

- **Sensitivity/true positive rate:** probability to correct recognize class 1

$$\frac{\text{prob. true pos}}{\text{prob. true pos} + \text{prob. false neg}} = \frac{P(11)}{P(11)+P(10)} \approx \frac{c_{11}}{c_{11}+c_{10}}$$

- **Specificity/true negative rate:** probability to correctly recognize class 0

$$\frac{\text{prob. true neg}}{\text{prob. true neg} + \text{prob. false pos}} = \frac{P(00)}{P(00)+P(01)} \approx \frac{c_{00}}{c_{00}+c_{01}}.$$

A good classifier has both high sensitivity and high specificity.

- **Precision/positive predictive values:** fraction of positive outputs that was actually positive

$$\frac{P(11)}{P(11)+P(01)} \approx \frac{c_{11}}{c_{11}+c_{01}}$$

- **Recall = sensitivity**
- **F measure:** it is a combined index that express with a single number the evaluation

$$F = 2\frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = \frac{2P(11)^2}{2P(11)+P(10)+P(01)} \approx \frac{2c_{11}^2}{2c_{11}+c_{10}+c_{01}}.$$

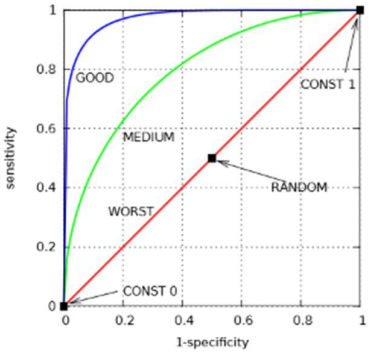
A generalized version exists:

$$F_\beta = (1+\beta^2)\frac{\text{precision} \times \text{recall}}{\beta^2\text{precision} + \text{recall}}$$

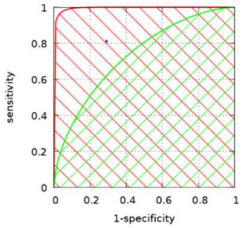
OSS: Neither precision/recall nor F consider the probability of being correct on the negative class. This is an advantage when it is easy to be correct on the negative class.

Receiver operating characteristic curve (ROC)

It is a graphic indicator for a binary classifier whose operation depends on some parameter $\boldsymbol{\theta}$ (ex: a threshold) and changing this parameter changes the performance of the classifier itself. We plot sensitivity as the abscissa and false positive rate (1-specificity) as the ordinate for varying values of θ.



A good classifier is near the top-left corner, this is also the one enclosing the largest area.
The **Area Under the Curve (AUC)** is another quality index to synthesize the information in one value.



NOTE: these indexes cannot directly be used as objective function because they cannot be expressed as a loss function. For instance:

$$F1 = \frac{2c_{11}^2}{2c_{11} + c_{10} + c_{01}} = \sum_{l=1}^{n} \lambda(y_l, t_l) \quad \Rightarrow \quad \lambda(y_l, t_l) = \; ???$$

NOTE: if I want to evaluate my solution according to two incompatible criteria the problem becomes multi-objective and some external criteria will be needed.

Neural networks

A neural network is a machine learning model inspired by the structure and functioning of the human brain. It is a computational model that consists of interconnected nodes, also known as artificial neurons or perceptrons, organized into layers. These networks are very **flexible** and can be trained to perform a wide range of tasks, including pattern recognition, classification, regression, and more.

A more straight definition is this one: neural networks (ANNs) are networks of several interconnected units, each with a simple behaviour, often **without** (or with limited) memory capable of a single input-to-output transformation. There are usually several connections, and their properties are summarized in parameters called **weights.**

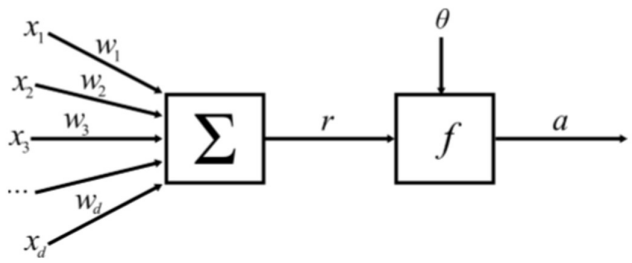
Models of a neuron

The linear models for regression and classification, already seen, take the form:

$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x})\right)$$

Where $f(\cdot)$ is a nonlinear activation function in the case of classification and is identity in the case of regression

A simple model of a neuron can be then built as this one:



- **x** are the **inputs** of the model
- **w** are a set of parameters called **weights**
- **r** is the net input, $r = \boldsymbol{w} \cdot \boldsymbol{x} = \sum_{i=1}^{d} w_i x_i$
- **f** is the **activation function**
- $\boldsymbol{\theta}$ is the **bias**
- **a** is the **activation value**

The output of this function is called **hidden unit**

Some examples of activation functions are:

Heaviside step (defined on $(-\infty, +\infty) \to \{0, +1\}$):

$$f(r) = \begin{cases} +1 & r \geq 0 \\ 0 & r < 0 \end{cases}$$

(ties broken arbitrarily)

variously indicated as $\mathbf{1}(r)$ or $\theta(r)$ or $\Theta(r)$ or $\mathscr{H}(r)$ or $u(r)$

Signum function (defined on $(-\infty, +\infty) \to \{-1, +1\}$):

$$f(r) = \begin{cases} +1 & r \geq 0 \\ -1 & r < 0 \end{cases}$$

indicated as $\operatorname{sgn}(r)$ or $\operatorname{sign}(r)$

Sigmoid (defined on $(-\infty, +\infty) \to (0, +1)$):

$$f(r) = \frac{1}{1 + e^{-r}}$$

Usually indicated as $\sigma(r)$

Hyperbolic tangent (defined on $(-\infty, +\infty) \to (-1, +1)$):

$$f(r) = \frac{1 - e^{-r}}{1 + e^{-r}}$$

Usually indicated as tanh

Discontinuous -> not differentiable

Continuous and smooth -> differentiable

Typically, for regression problems the activation function is the identity, for multiple binary classifications problems is used the sigmoid and finally for multiclass problems is used a softmax activation function of the form: $f(r_i) = \frac{e^{r_i}}{\sum_{j=1}^{k} e^{r_j}}$ (multi-neuron activations, where k indicates the neuron)

It's possible to combine many units to form one collective output.

Summary of activation function types

Type	Meaning	Examples
Single-unit	One $r \to$ one a	Heaviside step (binary, positive only)
		Signum (binary, symmetric)
		Logistic sigmoid (continuous, differentiable, positive only)
		Hyperbolic tangent (continuous, differentiable, symmetric)
		ReLU (continuous, nondifferentiable, positive only)
		Softplus ((continuous, differentiable, positive only)
Multi-unit	Multiple $r \to$ multiple a	Max (binary, positive only)
		Softmax (continuous, differentiable, positive only)
Assembly	Multiple $r \to$ one a	Pooling (many types)

Graphical conventions

Symbol	Meaning
●	Input unit – Stores a scalar input without processing it
x ●	Input unit with value specified
◯	Active unit (a formal neuron) – Takes one or more inputs, produces one output value
⟶	Connection – Inputs a scalar value on the non-arrow side and outputs it on the arrow side
x⟶	Connection with input specified
w⟶	Weighted connection – Output is input multiplied by weight w

Input-to-output signal flow may be drawn left-to-right or top-to-bottom, depending on graphical convenience.

Single-unit neural networks

Rosenblatt's perceptron

The Rosenblatt perceptron is a type of artificial neuron or a simple binary classifier introduced by Frank Rosenblatt in the late 1950s. This model has 3 layers:

- Sensory units or input retina (S-units), each unit is connected to up to 40 of the A-units
- Association units (A-units)
- Output layer of perceptrons, called response units (R-units)

The S-units are connected to the A-units randomly and then the A-units are connected to the R-units with adjustable weights. The perceptron learning algorithm is:

- Weights modification are computed upon receiving each individual pattern
- The linear threshold has an activation function $f(a) = sign(a)$ and bias w_0

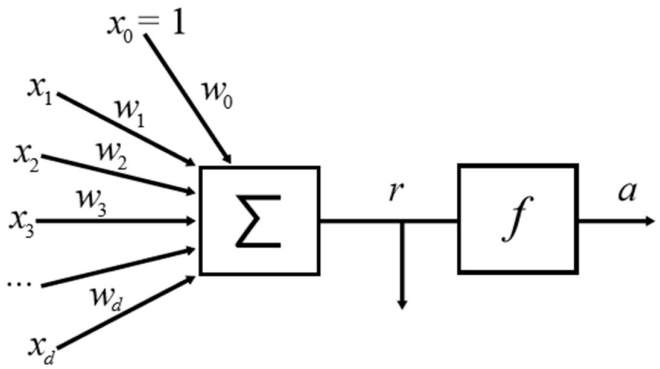
(init) Perform initializations
1. INPUT: Read input pattern number l, $\mathbf{x}_l$
2. Compute output: $r_l = \mathbf{w} \cdot \mathbf{x}_l$, $a_l = f(r_l)$
3. Compute output error: $\delta_l = \frac{1}{2}(t_l - a_l)$
4. Compute correction: $\Delta \mathbf{w}_l = \eta \delta_l \mathbf{x}_l$
5. Apply correction (weight update): $\mathbf{w}_{l+1} = \mathbf{w}_l + \Delta \mathbf{w}_l$
6. Increase l (*)
7. Go to INPUT

(*) modulo training set cardinality

Theorem

If the training set is linearly separable, the perceptron learning procedure will find a separating hyperplane for it in a finite number of steps

Adaline



Schematic according to our conventions.

r is a continuous function of weights and is used in the learning process rather than a. The evaluation of the error is based on the **Least Mean Squares algorithm.**

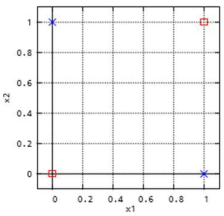
The Adaline training follows this algorithm:

At time step l the Adaline with weights w_l receives the pattern $\boldsymbol{x}$. Then the weights vector is changed by summing an updating step Δw_l s.t: $w_{l+1} = w_l + \Delta w_l$. This updating step is proportional to the product: $error\ on\ the\ output \cdot input\ vector$ -> $(t - r) \cdot \boldsymbol{x}$
In particular, $\Delta \boldsymbol{w_l} = \eta(t - r) \cdot \boldsymbol{x}$, If shorten $\delta = t - r$ -> $\Delta \boldsymbol{w_l} = \eta \delta \boldsymbol{x}$ (delta rule)
NOTE: the value of η is predefined. (proof of the formula above on the slides)

Ways to overcome the linear separability problem

The single neuron "networks" seen so far only implement a linear threshold classifier. Now we want something more powerful that can overcome the linear separability problem. There are three methods that can be used. Suppose to have this problem and we want to classify it

x_1	x_2	t
0	0	0
1	0	1
0	1	1
1	1	0



There are some possible way to solve this problem:

- **Feature engineering**: employ the knowledge of the problem to build new features

x_1	x_2	$x_3 = x_1 x_2$	t
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0

$$w_0 + x_1 w_1 + x_2 w_2 + x_3 w_3 = t$$

$$\begin{cases} w_0 + 0 \cdot w_1 + 0 \cdot w_2 + 0 \cdot w_3 = 0 \\ w_0 + 1 \cdot w_1 + 0 \cdot w_2 + 0 \cdot w_3 = 1 \\ w_0 + 0 \cdot w_1 + 1 \cdot w_2 + 0 \cdot w_3 = 1 \\ w_0 + 1 \cdot w_1 + 1 \cdot w_2 + 1 \cdot w_3 = 0 \end{cases} \qquad \mathbf{w} = [0\ 1\ 1\ -2]$$

- **Nonlinear neuron model:** employ the knowledge to transform the discriminant function

$$r = \mathbf{w} \cdot \mathbf{x} + \mathbf{x}^T V \mathbf{x} \qquad V = \begin{bmatrix} 0 & v \\ v & 0 \end{bmatrix}$$

- **Multi-layer network:** it's a universal solution ⇒ $r = x_0 w_0 + x_1 w_1 + x_2 w_2 + x_1 x_2 (2v)$

NOT

x_1	a
0	1
1	0

$\mathbf{w} = [0, -1]$

AND

x_1	x_2	a
0	0	0
1	0	0
0	1	0
1	1	1

$\mathbf{w} = [-1.5, 1, 1]$

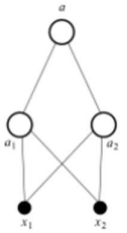
OR

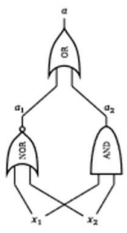
x_1	x_2	a
0	0	0
1	0	1
0	1	1
1	1	1

$\mathbf{w} = [-.5, 1, 1]$

Weights:
Bottom left unit (a_1): $\mathbf{w}_1 = [-0.5, 1, 1]$ (OR)
Bottom right unit (a_2): $\mathbf{w} = [1.5, -1, -1]$ (NAND)
Top unit (a): $\mathbf{w} = [-1.5, 1, 1]$ (AND)

x_1	x_2	a_1	a_2	a
0	0	0	1	1
1	0	0	0	0
0	1	0	0	0
1	1	1	0	1

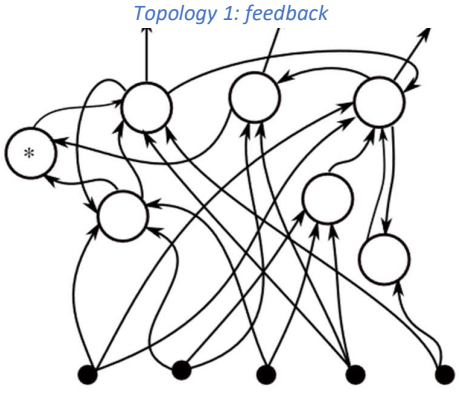




Network topologies

We want now to analyse some multi-layer network topologies

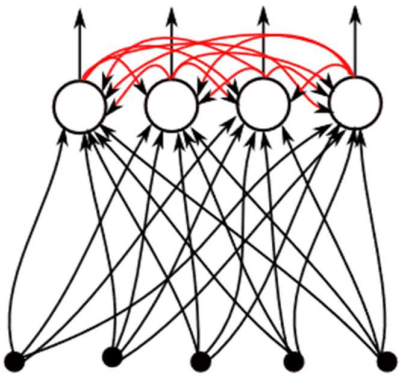
Topology 1: feedback



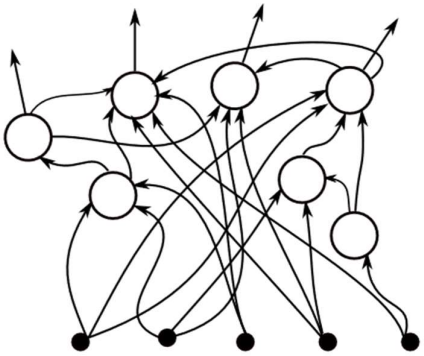
Units may be **visible** or **hidden** (*)

Hidden unit: it is neither on the input layer nor on the output

Topology 2: feedback with lateral connections

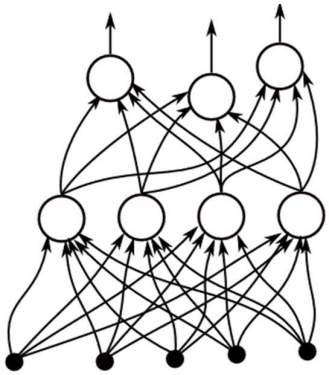


Topology 3: feedforward



Cycles are forbidden

Topology 4: multi-layer



Least general: multi-layer

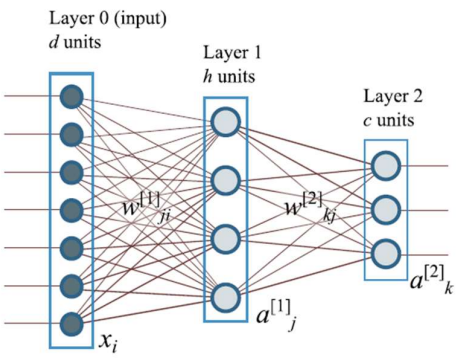
The layered network is the simplest possible connectivity that has the universal approximation

Definition

A feed-forward network with a single **hidden layer** containing a finite number of neurons can approximate continuous functions on compact subsets of $\mathbb{R}^d$

When applying a learning algorithm, a multi-layer networks will adapt its weights to approximate a given input-output mapping.

Layers are usually made of homogeneous units (same activation function), but layers need not be homogeneous to each other.



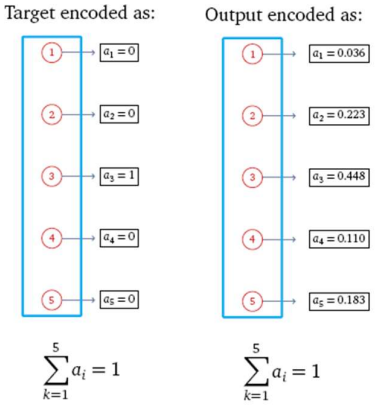
Layer 1 is called active layer. Typically, there are different types of layer:

- Input layer
- Normalisation layers
- Active layers
- Softmax layers
- Pooling layers

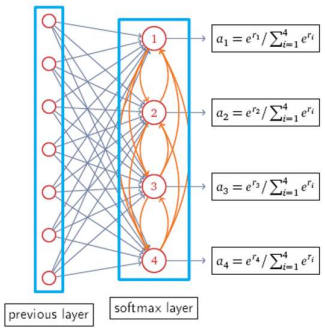
Multi-class classification

In the case of multi-class problem (c > 2), one output unit is not sufficient and we have to use c units. We use the one-hot encoding: at each label is assigned a unique binary value and all values are mutually exclusive. In this way the classification obeys to the probabilistic constraint $\sum_{j=1}^{c} a_j = 1$. The target and output vectors are of length c.

Now the problem is the following one: suppose to have 5 classes and that we want to classify class



We can interpret one-hot encoding as a probability distribution over classes and we can select the class with maximum output: $\arg\max a_k$.



Using the softmax is possible to find which unit has the max value

Since we are learning a probability distribution with the network this approach is called ***logistic regression***

Cross-entropy loss function

Cross-entropy is a measure of the difference between two probability distributions for a given random variable or set of events. Cross entropy loss measures the difference between the discovered probability distribution of a machine learning classification model and the predicted distribution.

- **Binary cross-entropy,** in this case it is given by: $\lambda(t,a) = -(t \cdot \log a + (1-t) \cdot \log(1-a))$, where t can be either 1 or 0 and a is the output probability of the model.
- **Multi class cross-entropy,** in this case the loss is given by: $\lambda(t,a) = -\sum_{i=1}^{c} t_i \cdot \log a_i$, where t_i is the actual probability of the class and a_i is the output probability of the model for that class.

This loss function essentially penalizes the model more when it makes a confident wrong prediction.

Training multi-layer networks

With learning we want to optimize a suitable cost function. Most methods are variations of gradient descent: $w_{t+1} = w_t - \eta \nabla \mathrm{J}(\mathrm{w})$. We need to have $J = E[\lambda(y(w))]$ differentiable, so λ differentiable w.r.t the output and the output differentiable w.r.t the weights. But, even if J is differentiable, for hidden units we cannot compute a loss $\lambda(a,t)$ which requires both the action value and the target value (we need another way to do this).

Error back-propagation algorithm

The first step is to compute the forward propagation, in which input data are passed through the neural network, and predictions are made. Initially, these predictions might not be accurate. After making predictions, we compare them to the actual, desired outputs. The goal is to quantify how far off our predictions are from the true values, finding the error. Then, it's computed the back-propagation on the error to adjust the weights and biases. Using the notation above, backpropagation attempts to minimize the following error function with respect to the neural network's weights:

$$E(X,\theta) = \frac{1}{2N} \sum_{i=1}^{N} (y_i - t_i)^2$$

This expression represents the mean squared error over all individual input-output pairs, where *N* is the number of pairs, $y_i$is the predicted output, and t_i is the actual output for the *i-th* pair. To achieve this minimization, back-propagation calculates the partial derivative of the error function with respect to each weight w_{ij}^k. This is denoted as $\frac{\partial E}{\partial w_{ij}^k}$ and it signifies how much the error would change if the corresponding weight is adjusted.

Since the error function can be expressed as a sum over individual error terms for each input-output pair, the derivative can be computed with respect to each pair separately and then aggregated.

$$\frac{\partial w_{ij}^k}{\partial E(X,\theta)} = \frac{1}{N} \sum_{d=1}^{N} \frac{\partial w_{ij}^k}{\partial E_d}$$

This process allows the neural network to adjust its weights based on the contribution of each pair to the overall error, facilitating the refinement of the model's predictions during training.

The derivation of the back-propagation algorithm begins by applying the chain rule to the error function partial derivative: $$\frac{\partial E}{\partial w_{ijk}} = \frac{\partial a_{jk}}{\partial w_{ijk}} \cdot \frac{\partial E}{\partial a_{jk}}$$

Here, a_j^k is the activation of node *j* in layer *k* before it is passed to the nonlinear activation function to generate the output. This decomposition of the partial derivative basically says that the change in the error function due to a weight is a product of the change in the error function *E* due to the activation a_j^k times the change in the activation a_j^k due to the weight w_{ij}^k. Recalling that is possible to write:

$$a_{ik} = b_{ik} + \sum_{j=1}^{r_{k-1}} w_{ji}^k o_j^{k-1} = \sum_{j=0}^{r_{k-1}} w_{ji}^k o_j^{k-1} \qquad \text{And that} \qquad \delta_{jk} \equiv \frac{\partial a_{jk}}{\partial E}$$

Obtaining: $$\frac{\partial w_{ijk}}{\partial E} = \delta_{jk} o_{ik-1}$$

Thus, the partial derivative of a weight is a product of the error term δ_{jk} at node *j* in layer *k* and the output o_j^{k-1} of node *i* in layer *k-1*. This makes intuitive sense since the weight w_{ij}^k connects the output of node *i* in layer *k-1* to the input of node *j* in layer *k* in the computation graph.

The Error Back-Propagation learning algorithm

For each input pattern x:

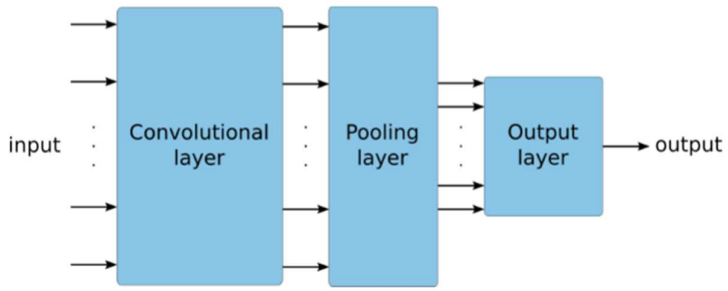
1. **Forward propagation (activation):**
   - $r_j = \sum_{i=0}^{d} w[1]_{ji} x_i$
   - $a[1]_j = \sigma(r_j)$
   - $r_k = \sum_{j=0}^{h} w[2]_{kj} a[1]_j$
   - $a[2]_k = \sigma(r_k)$
2. **Back propagation (error and correction):**
   - $\delta_k = \lambda'(t_k, a[2]_k)\sigma'(r_k)$
   - $\Delta w[2]_{kj} = -\eta \delta_k a[1]_j$
   - $\delta_j = \left[\frac{1}{c}\sum_{k=1}^{c} \delta_k \sigma'(r_k) w[2]_{kj}\right]$
   - $\Delta w[1]_{ji} = -\eta \delta_j x_i$
3. - BY PATTERN: immediately update $\mathbf{w} = \mathbf{w} + \Delta\mathbf{w}$
   - BY EPOCH: accumulate $\Delta\mathbf{w}_{tot} = \Delta\mathbf{w}_{tot} + \Delta\mathbf{w}$ and wait until end of the epoch, then apply $\Delta\mathbf{w}_{tot}$

Convolutional Neural Networks (CNNs)

They are used especially when dealing with large-dimensionality natural signals. They have two strong simplifications:

1. A fixed structure and a fixed reduced connectivity pattern
2. Weights on the hidden layer are tied or shared

Furthermore, the use pooling and non-linear output (usually ReLU). The architecture:



The **convolutional layer** mathematically implements a discrete convolution followed by nonlinearity.
The **pooling layer** is used after the convolutional layer to reduce dimensionality, indeed it takes a set of units and replace the n output values with one scalar.
The **output layer** is a fully-connected layer, that takes as inputs all the outputs of the pooling layer.
The model of the neuron is $r = \boldsymbol{x} \cdot \boldsymbol{w}$, $a = a(r)$ and the activation nonlinearity is usually softmax.

Training CNNs

Training is normally done with stochastic gradient descent using minibatches and use **regularization methods** to speed up the process, simplifying the objective function making it smoother and with less local minima (easier to train). These methods are:

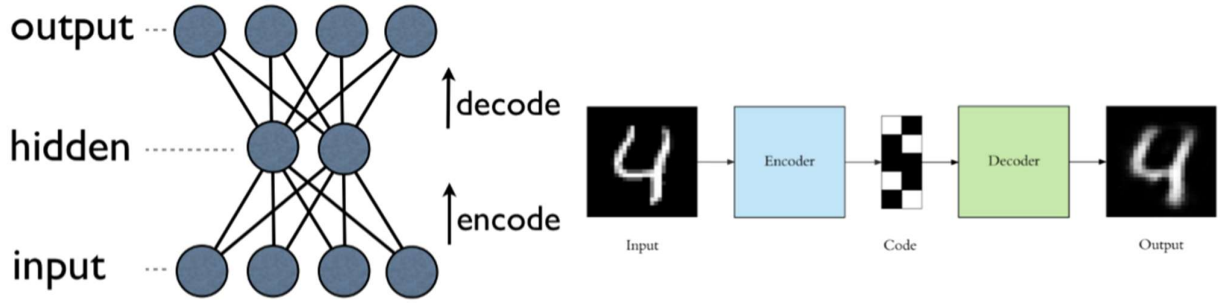
- Early stopping: measure the validation error during training and stops when it starts to increase
- Reducing network capacity, using smaller networks, shared weights
- Weight decay

Autoencoders

They are special case of a multi-layer perceptron characterised by two aspects:

1. **Structure:** the number of units in the input layer is equal to the number of units in the output layer, and these are greater than the number of hidden units.
2. **Learned task:** an autoencoder is trained to approximate the identity function.

This is not a classifier and the interesting is the pattern on the hidden layer (and not the output value)



Unsupervised learning

A typical unsupervised learning problem is **Clustering**, which means finding groups in data. There are two different types of clustering:

- **Crisp (hard) clustering:** each data item belongs to a single cluster
- **Fuzzy (soft) clustering:** each data item belongs to different clusters in different proportions

In a dataset comprising n experimental observations, clusters are characterised by **prototypes/centroids**. These representative points exhibit the property of being the mean of all the points within their respective clusters. The number k of clusters (and therefore of centroids) must be specified in advance -> k-means

K-means algorithm

- Step 0: centroids are initialized
- Step 1: each data point is put in its own cluster, the criterion is the nearest centroid
- Step 2: centroids are adjusted to actually be in the barycentre of the cluster they represent
- Repeat steps 1 and 2 until nothing changes anymore

Input: Training set X, number of clusters k

0 Initialize k centroids $\mathbf{y}_1, \ldots, \mathbf{y}_k$, for instance randomly.
Create an indicator vector $\mathbf{u}_l$ for each data point.

1 For each data point $\mathbf{x}_l$:
- 1.1 Compute the distance $d_{lj} = \|\mathbf{x}_l - \mathbf{y}_j\|$ to each centroid $\mathbf{y}_j$
- 1.2 Select $j^* = \arg\min_j d_{lj}$
- 1.3 Set

$$\mathbf{u}_l = [\ \underset{1}{0}\ \underset{2}{0}\ \ldots\ \underset{j^*}{1}\ \ldots\ \underset{k-1}{0}\ \underset{k}{0}\ ]$$

2 For each centroid $\mathbf{y}_j$:
- 2.1 Select all data points that have $\mathbf{u}_{l,j} = 1$
- 2.2 Move centroid $\mathbf{y}_j$ to their geometric mean

3 If centroids have changed w.r.t. last iteration, goto 1

The k-means objective function is given by: $J = \sum_{l=1}^{n} |x_l - y_j|^2$, where y_j is the nearest centroid to x_l

Pros: very simple algorithm, guaranteed not to diverge, works with high-dimensional data

Cons: local optimization only, since J is not convex, always finds k clusters

To improve the algorithm, it is used the k-means++ initialization. In this case as initial configuration of centroids are chosen points that are far away from each other.

Fuzzy c-means

This is an extension of the traditional k-means clustering algorithm that allows for fuzzy or soft assignments of data points to clusters. The membership function is used to evaluate the presence of points to cluster. Membership of point x_l to cluster with centroid y_j is a function of distance, and given by:

$$u_{lj} = \frac{1}{\sum_{k=1}^{c} \left(\frac{|x_l - y_j|}{|x_l - y_k|} \right)^{\frac{2}{m-1}}}$$

Where m is a fuzzification parameter. In this case centroids are weighted by: $y_j = \frac{\sum_{l=1}^{n} u_{lj} x_l}{\sum_{l=1}^{n} u_{lj}}$

Input: Training set X, number of clusters c, fuzzification parameter m;
stopping thresholds ϵ (on memberships), maxiter (on iterations).

0 Initialize c centroids $\mathbf{y}_1, \ldots, \mathbf{y}_c$, for instance randomly.
Create an indicator vector $\mathbf{u}_l$ for each data point.

1 For each data point $\mathbf{x}_l$:
- 1.1 Compute the distance $d_{lj} = \|\mathbf{x}_l - \mathbf{y}_j\|$ to each centroid $\mathbf{y}_j$
- 1.2 Compute the components $[\ u_{l1}, u_{l2}, \ldots, u_{lc}\ ]$ of the membership vector $\mathbf{u}_l$ as:

$$u_{lj} = \frac{1}{\sum_{k=1}^{c} \left(\frac{\|\mathbf{x}_l - \mathbf{y}_j\|}{\|\mathbf{x}_l - \mathbf{y}_k\|} \right)^{\frac{2}{m-1}}}$$

2 Compute each centroid $\mathbf{y}_j$ as:

$$\mathbf{y}_j = \frac{\sum_{l=1}^{n} u_{lj} \mathbf{x}_l}{\sum u_{lj}}$$

3 If the average centroid change $\|\mathbf{u}(t) - \mathbf{u}(t-1)\| > \epsilon$ from last iteration $t-1$ to this iteration t, and number of iterations $\leq$ maxiter, goto 1