Exploring Machine Learning Techniques

Angelo Efoévi Koudou

Digital Learning Hub

Exploring Machine Learning Techniques

Agenda

- Types of machine learning problems (supervised, unsupervised semi-supervised, reinforcement learning)
- Linear regression and logistic regression
- Classification algorithms: K-NN, SVM, decision trees, random forests
- Clustering techniques: introduction to unsupervised learning methodes such as K-means and DBSCAN
- Model evaluation methods : cross-validation, confusion matrices, precision, recall, F1-score.

Learning outcomes

- Understand key concepts of machine learning
- Build and implement linear and logistic regression models
- Use classification algorithms as K-NN and SVM
- Understand decision trees and random forests
- Apply clustering techniques such as K-means and DBSCAN
- Evaluate models by using cross-validation and performance metrics
- Apply machine learning techniques to real datasets

Digital Learning Hub_

1. Introduction to Machine Learning

1.1 What is Machine Learning?

What Does It Mean for a Machine to Learn?

In the context of computer science, **machine learning** refers to the ability of a program to modify itself without such modification being explicitly programmed (Arthur Samuel, 1959).

In other words, there are two main paradigms:

- Traditional programming, where a predefined procedure and input data are used to produce an output.
Example: calculating the total amount spent by a customer based on their invoices.

- Machine learning, where data and corresponding outputs (or labels) are used to infer the procedure that can generate the outputs from new input data.
Example: analyzing a customer's invoices to predict which products they are most likely to purchase within the next month.

Ingredients of machine learning

Machine learning relies on two key components:

1. **Data**, which are the examples from which the algorithm learns;
2. **The learning algorithm**, which is the procedure run on the data to produce a model.
   The process of running a learning algorithm on a dataset is called **training**.

Both components are equally important:

- No learning algorithm can produce a good model from irrelevant data.
- A model trained with an inappropriate algorithm, even on relevant data, will not be of good quality.

In this course, we focus primarily on the **algorithms**, while keeping in mind that an essential part of a machine learner's or data scientist's work is **data preparation** — removing outliers, handling missing values, and so on.

Machine learning relies on mathematics, particularly statistics, for building models from data, and on computer science for data representation and the efficient implementation of optimization algorithms.

Connection with AI:
Machine learning is a part of AI. However, AI, defined as the set of techniques used to build machines capable of exhibiting behavior that can be considered intelligent, also draws on other sciences such as cognitive science, neurobiology, electronics, and many others.

Why Use Machine Learning?
To solve problems:

- that we do not know how to solve (as in the example of predicting purchases);
- that we know how to solve, but for which we cannot formulate a solution in algorithmic terms (for example, image recognition or natural language understanding);
- that we know how to solve, but with procedures that are too resource-intensive (for example, predicting interactions between large molecules).

Machine learning is therefore used when data are abundant, but knowledge is scarce or underdeveloped. It can also help humans learn, for example:

- Which features of past purchases allow us to predict future purchases?
- Which genes are involved in the development of a certain type of tumor, and how?
- Which regions of a brain image allow us to predict behavior?

1.2 Types of machine learning problems

1.2.1 Supervised learning

The goal is to learn how to make predictions from a list of labeled examples, that is, examples accompanied by the value to be predicted.

A supervised learning problem can be formalized as follows:

given *n* observations $x_1, x_2, \ldots, x_n$, belonging to the observation space $\mathcal{X}$, and their labels $y_1, y_2, \ldots, y_n$ belonging to the label space $\mathcal{Y}$, we assume that the labels can be obtained from the observations through a fixed but unknown function $\phi: \mathcal{X} \to \mathcal{Y}$:

$y_i = \phi(x_i) + \varepsilon_i$, where ε_i is a random noise.

The goal of supervised learning is to use the data to find a function $f: \mathcal{X} \to \mathcal{Y}$, called the **decision function**, such that $f(x)$ is a good approximation of $\phi(x)$ for all $x \in \mathcal{X}$.

The set $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ forms the **training set**.

Binary classification

In the case where there are only two possible labels, that is, $\mathcal{Y} = \{0,1\}$, we talk about a **binary classification** problem.
In some cases, it is mathematically simpler to use $\mathcal{Y} = \{-1,1\}$.

Examples :

- Identify whether an email is spam or not
- Identify whether a painting was made by Picasso or not
- Identify whether an image contains a giraffe or not
- Identify whether a molecule can treat depression or not
- Identify whether a financial transaction is fraudulent or not.

Multiclass classification

In the case where there are C classes, i.e. $\mathcal{Y} = \{1,2, \dots, C\}$, we talk about a **multiclass classification.**

Examples:

- Identifty in which language a text is written.
- Identify the facial expression from a predefined list of possibilities (anger, sadness, joy, etc.)
- Identify which species a plant belongs to, etc.

Regression

If $\mathcal{Y} = \mathbb{R}$, we talk about a regression problem.

Examples:

- Predicting the number of clicks on a link.
- Predicting the number of users of an online service at a given time.
- Predicting the price of a stock.
- Predicting the yield of a corn plant.

Structured Regression (or **structured output prediction**): a case where the label space is more structured than in the previous examples. This is the case, for example, in predicting images, graphs, or sequences. It allows formalizing many problems, such as machine translation or speech recognition (text-to-speech or speech-to-text, for example).

Cost functions

Solving a supervised learning problem consists in finding a function f whose predictions are as close as possible to the true labels, on the space $\mathcal{X}$. This is formalised by the notion of **cost function.**

A cost function $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, also called ***loss function***, is a function used to quantify the quality of a prediction.

The closer $f(x)$ to the true label y, the smaller $L\big(y, f(x)\big)$.

Cost functions for binary classification:

- 0/1 loss: $L\left(y, f(x)\right) = 1 \; if \; f(x) \neq y$, 0 otherwise, with $\mathcal{Y} = \{0,1\}$.
- Hinge loss : $L\left(y, f(x)\right) = 0$ if $yf(x) \geq 1$, $1 - yf(x)$ otherwise, with $\mathcal{Y} = \{-1,1\}$.
- Quadratic loss : $L\left(y, f(x)\right) = \left(1 - yf(x)\right)^2$ with $\mathcal{Y} = \{-1,1\}$.
- Logistic loss : $L\left(y, f(x)\right) = \ln(1 + e^{-yf(x)})$, $\mathcal{Y} = \{-1,1\}$.

Digital Learning Hub

Cross-entropy loss:

$$L\left(y, f(x)\right) = -y \ln f(x) - (1-y) \ln\left(1 - f(x)\right), \quad \mathcal{Y} = \{0,1\}.$$

Loss functions for regression :

- **Quadratic loss**: $L\left(y, f(x)\right) = \frac{1}{2}(y - f(x))^2$, $\mathcal{Y} = \mathbb{R}$.
- **Absolute loss:** $L\left(y, f(x)\right) = |y - f(x)|$.

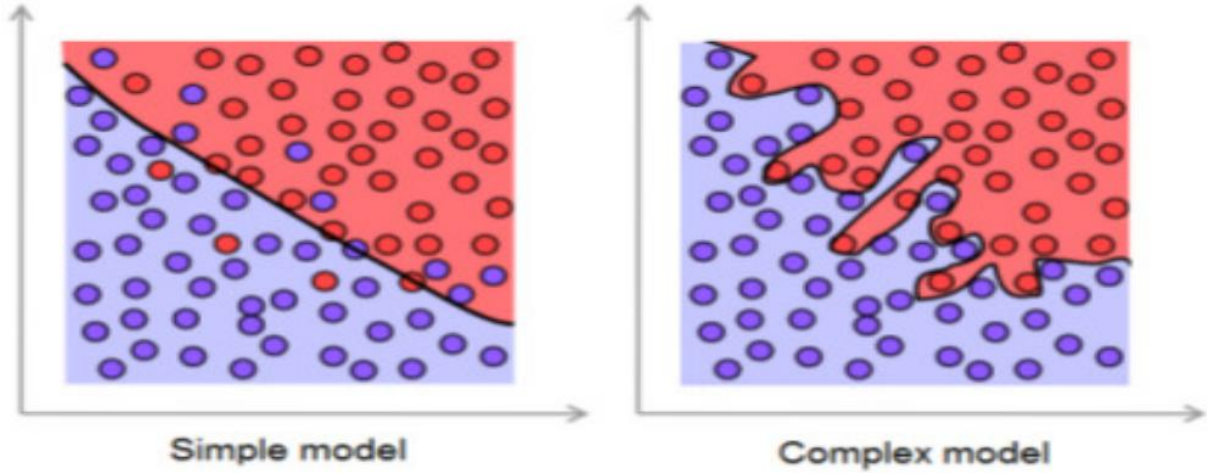
The quantity $\frac{1}{n}\sum_{i=1}^{n} L(y_i, h(x_i))$, where h belongs to a class of functions, is called **empirical risk.**

The aim is to find a function minimizing this risk among all functions in the considered class.

Digital Learning Hub

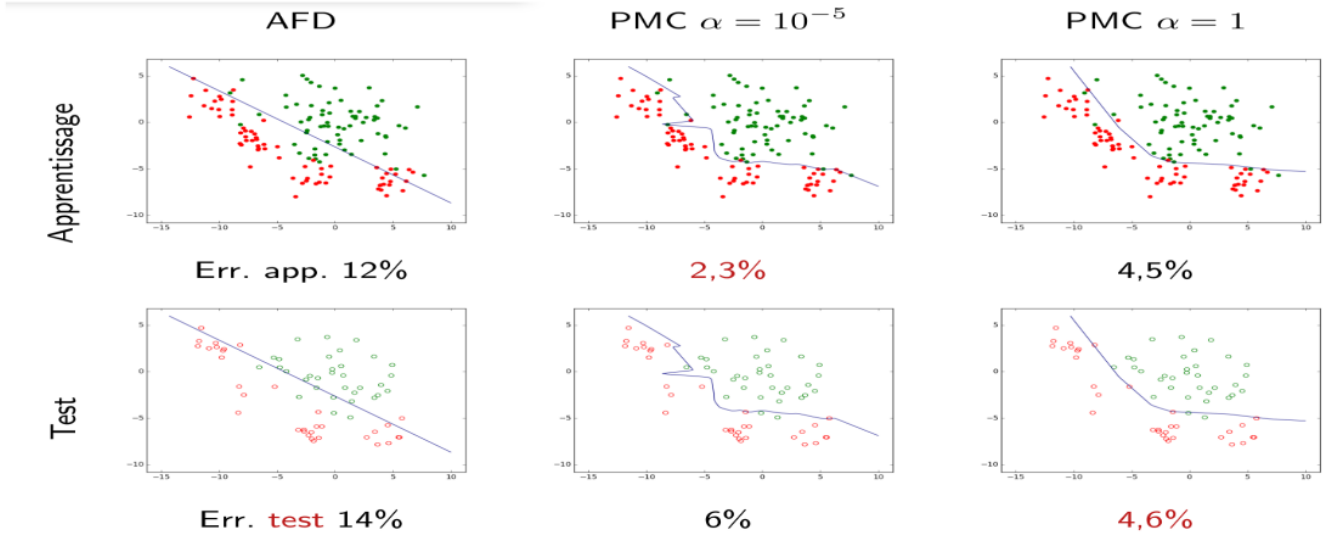
Generalization, Overfitting, Underfitting

- Evaluating a machine learning algorithm on the data it was trained on does not allow us to know how it will behave on new data — that is, its ability to generalize.
- **Generalization** refers to the ability of a model to make correct predictions on new data that were not used to build it.
- **Overfitting:** A model is said to overfit when, instead of capturing the underlying nature of the data to be labeled, it also models the noise and therefore fails to generalize properly. It is a model that is too complex, fitting the training data too closely and capturing their noise as well.
- **Underfitting:** A model is said to underfit when it is too simple to achieve good performance, even on the data used to train it.



Digital Learning Hub

The model with the lowest error on training set does not necessarily have the lowest error on test set.



Digital Learning Hub

1.2.2 Unsupervised Learning

In unsupervised learning, the data are not labeled. The goal is to model the observations in order to better understand them.

Clustering

Clustering, or partitioning, consists of identifying groups within the data.

Examples:

- Market segmentation: identifying groups of users or customers with similar behavior, for instance, to better target an advertising campaign.
- Image compression: grouping similar pixels together to represent them more efficiently.
- Identifying groups of patients with similar symptoms can help detect subtypes of a disease that may require different treatments.

Dimensionality Reduction
Dimensionality reduction is an important family of unsupervised learning problems. The goal is to find a representation of the data in a space of lower dimension than the one in which they were originally represented.
This makes it possible to reduce computation time and the memory required to store the data, and often also to improve the performance of a supervised learning algorithm subsequently trained on these data.

Density Estimation
This involves assuming that the dataset is a sample from a random variable, and then estimating the probability distribution of that random variable.

1.2.3 Semi-Supervised Learning

This involves learning labels from a partially labeled dataset. The main advantage of this approach is that it avoids having to label all the training examples.
For example, in the case of image classification, it is easy to obtain a database containing hundreds of thousands of images, but it can be tedious to have the specific label of interest for each one.

1.2.4 Reinforcement Learning

In reinforcement learning, the learning system can interact with its environment and perform actions. In return for these actions, it receives a reward: positive if the action was a good choice, or negative otherwise. The reward may sometimes come after a long sequence of actions, as is the case, for example, for a system learning to play Go or chess.
In this case, learning consists of defining a strategy that consistently maximizes the reward.

Practical Resources
Many freely available software and libraries allow the implementation of machine learning algorithms, for example:

- **Python library scikit-learn:** http://scikit-learn.org
- **Tools implemented in R:** http://cran.r-project.org/web/views/MachineLearning.html
- **Weka (Waikato Environment for Knowledge Analysis),** machine learning tools written in Java: https://www.cs.waikato.ac.nz/ml/weka/
- **Shogun,** interfaces for languages including Python, Octave, R, etc.: http://www.shogun-toolbox.org/
- Many machine learning algorithms are also implemented in **TensorFlow**, which is part of the specialized tools for deep learning developed in recent years: https://www.tensorflow.org

Datasets:
You can practice or test machine learning algorithms using many publicly available datasets:

- **University of California, Irvine (UCI) repository:** https://archive.ics.uci.edu/ml/index.php
- **Resources available on KDNuggets:** https://www.kdnuggets.com/datasets/index.html
- **The data science competition platform Kaggle:** https://www.kaggle.com

Further reading :

Alpaydin, Ethem (2020). *Introduction to Machine Learning*, (4th edition) MIT Press

Chloé-Agathe Azencott. *Introduction au machine learning,* Dunod. (french)

Virginie Mathivet. *Machine learning, implémentation en Python avec Scikit-learn*, eni. (french)

2. Linear regression, logistic regression

2.1 Linear regression

Objective : to describe a variable of interest Y (also called the **response variable** or **exogenous variable**) as a function of several explanatory variables $X_1, X_2, \ldots, X_p$ (also called **regressors** or **endogenous variables**).

Examples:

- Snow depth as a function of slope orientation, inclination, roughness, altitude, and latitude.
- Fuel consumption of a vehicle as a function of its weight, power, and engine displacement.
- Ozone concentration in the air as a function of temperature at different times of the day.

Multiple linear regression allows for:

- **Modeling:** constructing a model that best represents reality based on the available information;
- **Prediction:** estimating the value of the variable of interest given the values of the explanatory variables.

The model :

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots \beta_p X_p + \varepsilon,$$

where

Y is the **response variable** , $X_1, X_2, \ldots, X_p$ explanatory variables, ε is a random error,

$\beta_0, \beta_1, \ldots \beta_p$ are the parameters to estimate.

If we observe a sample of n individuals and denote by $x_{i,1}, x_{i,2}, \dots, x_{i,p}, Y_i$ the values of the different variables for individual i, then the model can be written as a system of n equations:

$$Y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \cdots \beta_p x_{ip} + \varepsilon_i, \text{ for } i = 1, \dots n.$$

This system of n équations can be expressed in matrix form as follows:

$$Y = X\beta + \varepsilon$$

with $Y = \begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{pmatrix}$, $\mathrm{X} = \begin{pmatrix} 1 & x_{1,1} & \dots & x_{1,p} \\ 1 & x_{2,1} & \dots & x_{2,p} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n,1} & \dots & x_{n,p} \end{pmatrix}$,

$\beta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{pmatrix}$ and $\varepsilon = \begin{pmatrix} \varepsilon_0 \\ \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{pmatrix}$.

The multiple linear regression model assumes that the random variables $\varepsilon_1, \varepsilon_2, \dots, \varepsilon_n$ have zero mean, are uncorrelated, and share the same variance σ^2 (**homoscedasticity**).

Digital Learning Hub

One seeks the vector $\hat{\beta}$ which minimizes the quantity

$$\|Y - X\beta\|^2 = \|\varepsilon\|^2 = (Y - X\beta)'(Y - X\beta)$$

$$= Y'Y - 2\beta'X'Y + \beta'X'X\beta.$$

Writing that the partial derivative of this expression with respect to each component of β is equal to 0, on obtains the equation

$$X'Y = X'X\hat{\beta}.$$

If the matrix $X'X$is invertible, then the solution is

$$\hat{\beta} = (X'X)^{-1}X'Y.$$

Regularized regression

- When the explanatory variables are highly correlated, or when their number exceeds the number of observations, the matrix $X'X$ is not invertible, and there is no unique solution to the linear regression problem obtained by least squares minimization.
- In such cases, the notion of **regularization** is introduced. Regularization consists in simultaneously controlling the model error and the values of the regression coefficients. The parameter vector is constrained to remain within a bounded domain.

Ridge Regression :

Given $\lambda > 0$, we seek the vector $\hat{\beta}$ which minimizes the quantity

$$\|Y - X\beta\|^2 + \lambda \|\beta\|^2.$$

The solution is given by

$$\hat{\beta} = (\lambda I_p + X'X)^{-1}X'Y.$$

This is equivalent to minimizing $\|Y - X\beta\|^2$ subject to the constraint $\|\beta\|^2 \leq t$,where $t > 0$.

Digital Learning Hub

Lasso regression:

In some applications, it may be reasonable to assume that the label to be predicted can be explained by only a limited number of variables.
It is therefore desirable to obtain a **parsimonious (sparse) model**, in which several coefficients are equal to zero.
The goal is then to drive some of the coefficients toward zero.

We thus seek the vector $\hat{\beta}$ that minimizes the following quantity:

$$\|Y - X\beta\|^2 + \lambda \sum_{j=0}^{p} \left|\beta_j\right| .$$

This is equivalent to minimizing $\|Y - X\beta\|^2$ subject to the constraint $\sum_{j=0}^{p} \left|\beta_j\right| \leq t$, where t>0.

LASSO is an acronym for **Least Absolute Shrinkage and Selection Operator**.
It uses the absolute values of the coefficients to **shrink** them, which enables the selection of variables whose coefficients are nonzero.

The LASSO does not have a closed-form solution and is solved using **numerical optimization methods** (for example, the **gradient descent algorithm**).

2.2 Logistic regression

The **logit function** is defined for $t \in]0,1[$ by

$$\text{logit}(p) = \ln \frac{t}{1-t}.$$

Its inverse function is the **logistic function** σ defined on $\mathbb{R}$ by

$$\sigma(u) = \frac{1}{1+e^{-u}}.$$

Given a dataset $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ of n observations in p dimensions and their labels in $\{0,1\}$, one assumes that $\mathcal{D}$ is a sample of a random vector (X, Y), with Y valued in $\{0,1\}$.

The logistic regression model assumes that

$$\text{logit}\big(P(Y = 1 \mid X = (X_1, X_2, \ldots X_p)\big) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p,$$

that is,

$$P(Y = 1 \mid X = (X_1, X_2, \ldots X_p) = \sigma\left(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots \beta_p X_p\right) = \frac{1}{1+e^{-(\beta_0+\beta_1 X_1+\beta_2 X_2+\cdots+\beta_p X_p)}}.$$

Digital Learning Hub

The aim is to find the parameters $\beta_0, \beta_1, \beta_2, \ldots, \beta_p$ that minimize the empirical risk

$$\frac{1}{n}\sum_{i=1}^{n} L_H(y_i, \sigma\left(\beta_0 + \beta_1 X_{i,1} + \beta_2 Y_{i,2} + \cdots + \beta_p X_{i,p}\right)),$$

where L_H is the cross-entropy loss function

$$L_H\left(y, f(x)\right) = -y ln\, f(x) - (1-y)\ln\left(1 - f(x)\right)\ .$$

Since there is no closed-form solution to this problem, one uses gradient descent.

3. Support vector machines (SVM)

Support Vector Machines (SVMs) are powerful machine learning algorithms.
Based on a linear algorithm proposed by Vapnik and Lerner in 1963, they were efficiently extended in the early 1990s to the learning of nonlinear models through the *kernel trick.*

3.1 The linearly separable case: Hard-margin SVM

Given $\mathrm{w} = (w_1, \ldots, w_d) \in \mathbb{R}^d$ and $b \in \mathbb{R}$, the *hyperplane* of $\mathbb{R}^d$ defined by (w, d) is the set of vectors $u \in \mathbb{R}^d$ such that $\langle w, u \rangle + b = 0$.

The notion of a hyperplane generalizes, in higher dimensions, the notion of a line in $\mathbb{R}^2$ $(d = 2)$ and of a plane in $\mathbb{R}^3$ $(d = 3)$.

w is then a vector normal to the hyperplane.

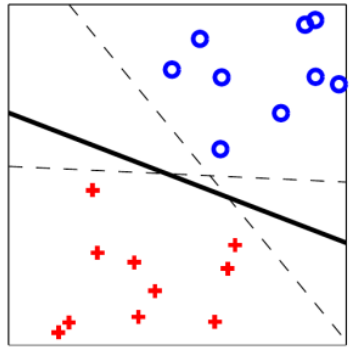
The Euclidean distance from a point $v \in \mathbb{R}^d$ to a hyperplane of equation $\langle w, u \rangle + b = 0$ is equal to

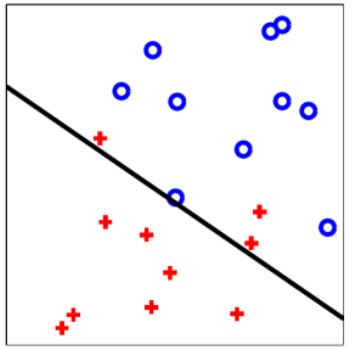
$$\frac{|\langle w, v \rangle + b|}{\|w\|},$$

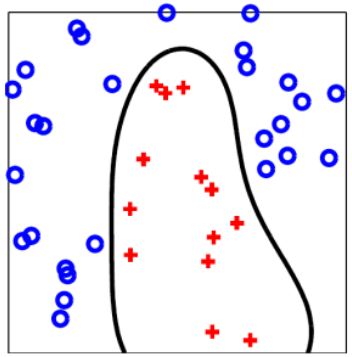
where $\|w\|$ is the Euclidean norm of w.

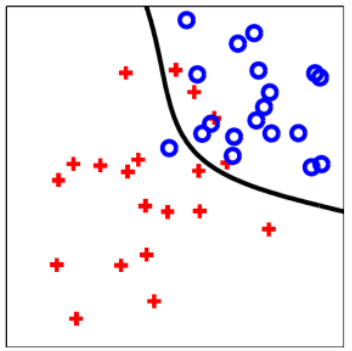
Digital Learning Hub

Linear separability









Digital Learning Hub

Consider a dataset $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ of n observations, with $x_i \in \mathbb{R}^d$ for all i, and their labels in $\{-1,1\}$.

A **binary classification problem** is said to be **linearly separable** if there exists a hyperplane that separates the two classes.
In this case, there actually exists an infinite number of separating hyperplanes.

The **margin** γ of a separating hyperplane is the distance between this hyperplane and the closest observation from the training set.
We seek the separating hyperplane that **maximizes the margin**.
There is then at least one negative observation (i.e., labeled -1) and one positive observation that are at a distance γ from the separating hyperplane.

We consider the hyperplanes H_+ and H_- parallel to H and located at distance γ from H.

The hyperplane H_+ contains at least one positive observation and H_- contains at least one negative observation.

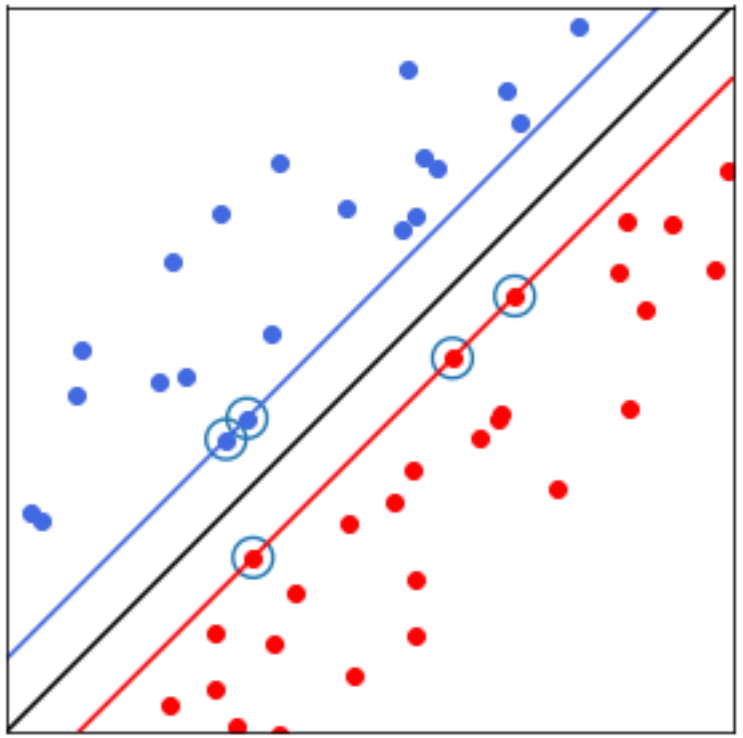
The separating hyperplane has an equation of the form $\langle w, u\rangle + b = 0$. Thus, the hyperplane$H_+$, which is parallel to H, has the equation $\langle w, u\rangle + b = c$, where c is a constant. This constant can be set to 1, since multiplying w by a constant does not change the equation of H.

The equation of H_- is therefore $\langle w, u\rangle + b = -1$.

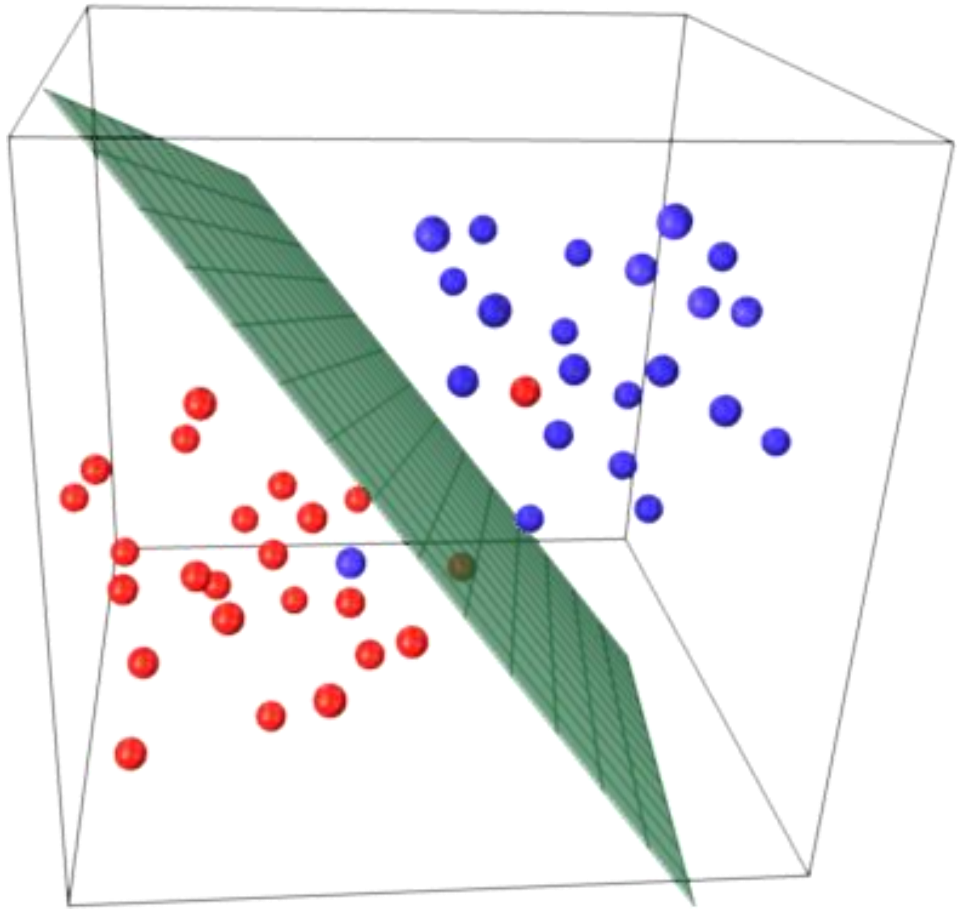
The **margin,** which is the distance between H and H_+, is then given by

$$\gamma = \frac{1}{\|w\|}.$$

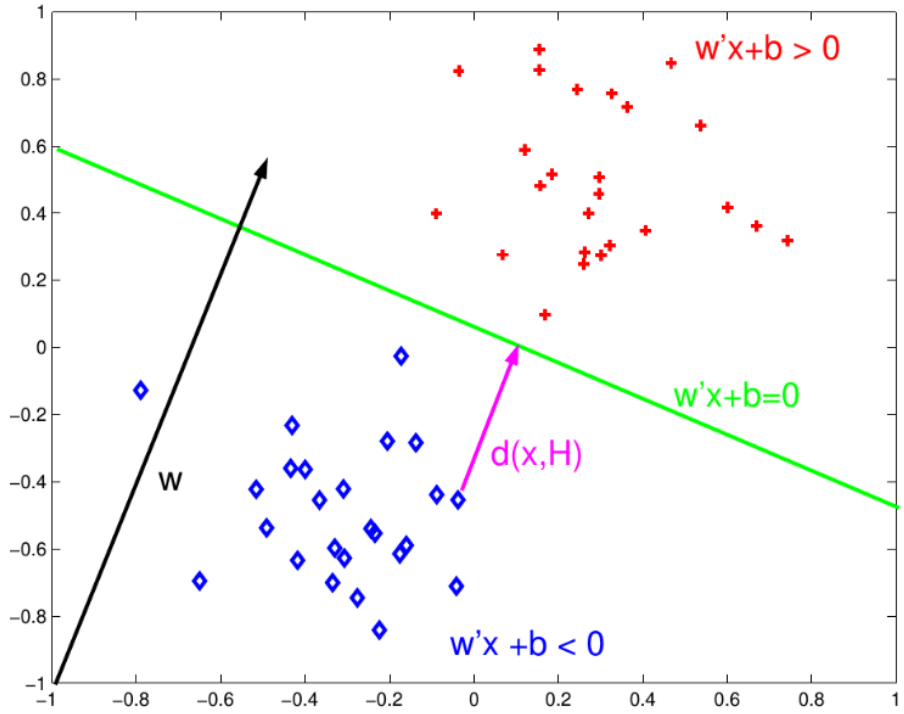
The **support vectors** are the training observations whose distance to the separating hyperplane is equal to γ. They "support" the hyperplanes H_+ and H_- , hence the name **Support Vector Machine**.



Digital Learning Hub



Digital Learning Hub



Digital Learning Hub

Formulation of the hard-margin SVM:

Positive observations positives satisfy $\langle w, x_i \rangle + b \geq 1$.

Negative ones satisfy $\langle w, x_i \rangle + b \leq -1$.

Thus for each (x_i, y_i) in the training set, $y_i(\langle w, x_i \rangle + b) \geq 1$.

We seek to maximize $\frac{1}{\|w\|}$ under the constraints $y_i(\langle w, x_i \rangle + b) \geq 1, i = 1, \dots, n$.

The function which at w assigns $\|w\|$ is not differentiable at 0, so we formulate the problem as follows:

we call **hard-margin SVM** the problem that consists in finding $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$ which minimize $\|w\|^2$ under the constraints $y_i \langle w, x_i \rangle + b \geq 1, i = 1, \dots, n$.

This problem of convex optimization under constraints is equivalent to its **dual problem:**

Find $\alpha = (\alpha_1, \dots, \alpha_n) \in \mathbb{R}^n$ that maximizes

$$\sum_{i=1}^{n} \alpha_i - \sum_{i,j=1}^{n} \alpha_i \alpha_j \, y_i y_j \langle x_i, x_j \rangle$$

under constraints $\alpha_i \geq 0$ and $\sum_{i=0}^{n} \alpha_i y_i = 0, \quad i = 1, \dots, n.$

To solve the initial problem, we may begin by solving the dual. If α^* solves the dual, then the solution of the initial problem is

$w^* = \sum_{i=1}^{n} \alpha_i^* y_i x_i$ and $b^* = 1 - \min_{i:\, y_i = 1} \langle w^*, x_i \rangle.$

The decision function is then given by $f(x) = \sum_{i=1}^{n} \alpha_i^* y_i \langle x_i, x_j \rangle + b^*$, that is, a new observation x will be classified as positive if $f(x) \geq 0$ and as negative si $f(x) < 0$.

Digital Learning Hub

3.2 The Non–Linearly Separable Case: Soft-Margin SVM

In most situations, the data are not linearly separable.
In such cases, a **soft-margin** technique can be used, which allows for some misclassifications.

Idea: model potential classification errors using positive **slack variables** ξ_i associated with each observation (x_i, y_i). Two cases are distinguished:

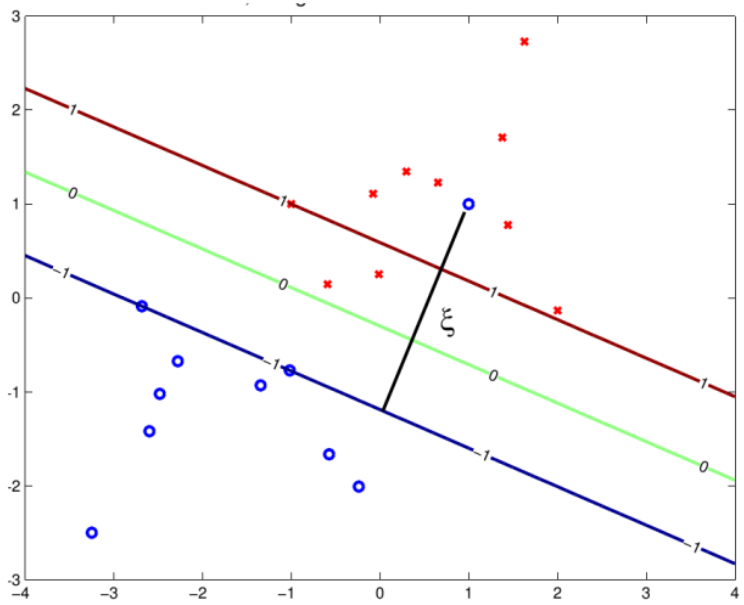
No error: $y_i\langle w, x_i\rangle + b \geq 1$, in which case $\xi_i = 0$.

Error: $y_i(\langle w, x_i\rangle + b) < 1$, and therefore $\xi_i = 1 - y_i(\langle w, x_i\rangle + b) > 0$.

Since the goal is to minimize both $\|w\|^2$ and the sum of the ξ_i values, a **regularization parameter** is introduced to balance the two objectives, resulting in a single optimization function.

The **soft-margin SVM** problem can thus be formulated as follows:

Find $w \in \mathbb{R}^d$, $b \in \mathbb{R}$ and $\xi = (\xi_1, \ldots, \xi_n) \in \mathbb{R}^n$ that minimize $\|w\|^2 + C\sum_{i=1}^{n} \xi_i$ subject to the constraints $y_i(\langle w, x_i\rangle + b) \geq 1 - \xi_i$, and $\xi_i \geq 0$, $i = 1, \ldots, n$.



Digital Learning Hub

3.3 Kernel SVM

It is often the case that a linear function is not suitable for separating the data.

Principle (Kernel Trick): transform the data into another space — generally of higher dimension — called the **feature space** (or **redescription space**), in which the data become linearly separable (or nearly so), and then apply the SVM algorithm to the transformed data.

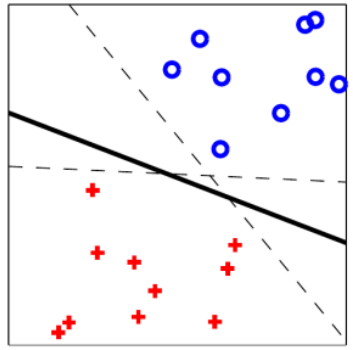
A transformation $\varphi : \mathbb{R}^d \rightarrow \mathcal{H}$ is used, where $\mathcal{H}$ is a **Hilbert space** (equipped with an inner product).

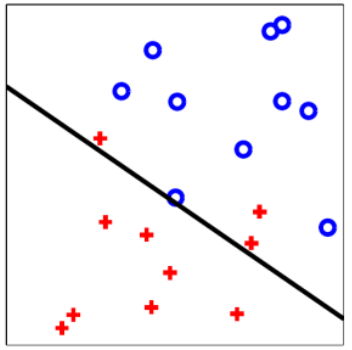
SVM in the Feature Space

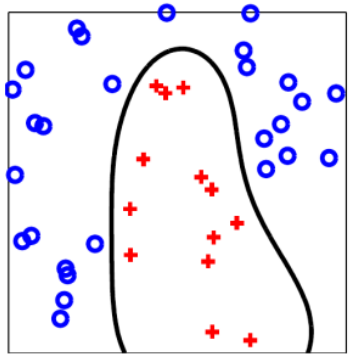
In this space, the optimization problem (for the hard-margin case) is formulated in the same way as before, but using the transformed data. We therefore seek to **maximize**:

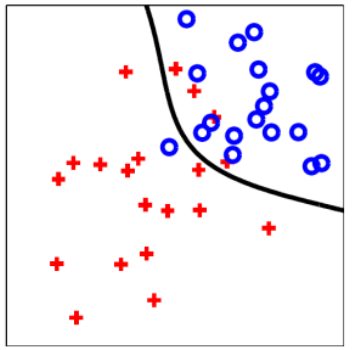
$\sum_{i=1}^{n} \alpha_i - \sum_{i,j=1}^{n} \alpha_i \alpha_j \, y_i y_j \langle \varphi(x_i), \varphi(x_j) \rangle$ subject to the previously stated constraints.

Not all classification problems can be solved using linear models

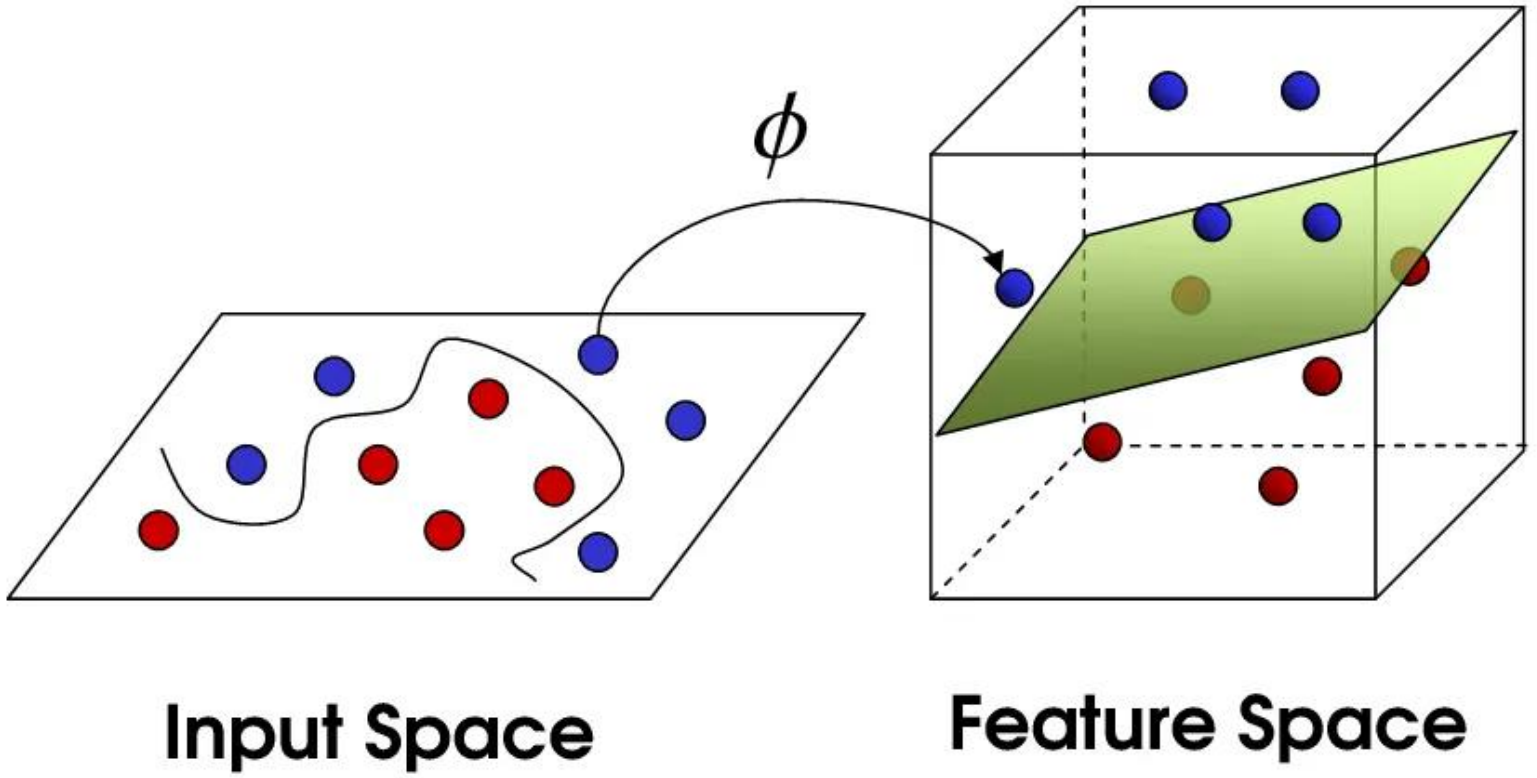








Digital Learning Hub


ϕ
Input Space
Feature Space

The advantage of this approach is that it does not require explicit knowledge of the function φ but only the expression of the inner product $\langle \varphi(x_i), \varphi(x_j) \rangle$ for $x_i , x_j \in \mathbb{R}^d$.

A **kernel** is any function k of two variables that can be written in the form $k(u, v) = \langle \varphi(u), \varphi(v) \rangle$, where φ maps into a **Hilbert space** $\mathcal{H}$.

Examples of kernels:

- **Quadratic kernel:** $k(u, v) = (\langle u, v \rangle + c)^2 , \ c \geq 0.$
- **Polynomial kernel:** $k(u, v) = (\langle u, v \rangle + c)^d, \ c \geq 0.$
- **Gaussian radial basis function (RBF) kernel** with bandwidth $\sigma > 0$ *:*

$$k(u, v) = \exp\left(-\frac{\|u-v\|^2}{2\sigma^2}\right).$$

Digital Learning Hub

4. Model evaluation methods

It is well known that no machine learning algorithm can perform well on all learning problems.
For a given problem, it is essential to test several approaches in order to select the **optimal model**.
Several criteria may influence this choice — not only **prediction quality**, but also **computational resources**, which can be a limiting factor in practice.

4.1 Empirical Estimation of the Generalization Error

4.1.1 Test set

For a given dataset, the **training set** refers to the portion of the data used to train a predictive model, while the **test set** is the portion used to evaluate its performance.

4. 1. 2 Validation set

Suppose we wish to choose among K models. Each model can be trained on the training dataset, resulting in K decision functions $f_1, \dots, f_k$.

We can then compute the error of each model on the test set and select the model that achieves the lowest test error.

However, since the test set has been used to select the model, it **can no longer be used** to estimate the **generalization error** of the chosen model.

Digital Learning Hub

- Hence, the solution is to divide the dataset into **three parts**:
- a **training set**, on which the K algorithms are trained;
- a **validation set**, on which the K resulting models are evaluated in order to select the final model;
- a **test set**, on which the generalization error of the selected model is evaluated.

It is therefore important to **distinguish between model selection and model evaluation**.

Performing both on the same data can lead to an **underestimation of the generalization error** and to **overfitting** of the chosen model.

Once a model has been selected, it can be **retrained on the union of the training and validation sets** to build the final model.

4.1.3 Cross-validation

For a dataset $\mathcal{D}$ and integer K, ***K*-fold cross-validation** is the procedure that consists of:

- partitionning $\mathcal{D}$ into K subsets of approximately equal size, $\mathcal{D}_1, \dots, \mathcal{D}_K$;
- for each $k = 1, \dots, K$, training the model on $\mathcal{D} \setminus \mathcal{D}_k$ and evaluating it on $\mathcal{D}_k$.

Each labeled observation in $\mathcal{D}$ thus belongs to exactly **one test set** and to(K-1) training sets.

This procedure therefore provides a prediction for each observation in $\mathcal{D}$.

Two evaluation strategies can then be used:

- **Assess the overall prediction quality** on the entire dataset $\mathcal{D}$.
- **Evaluate the performance** of the K predictors on their respective test subsets $\mathcal{D}_k$, and **average** their results.

An advantage of this second approach is that it allows for the estimation of the **standard deviation of the performance metrics**, providing insight into the **variability of the model's predictions** depending on the training data.

Cross-validation is said to be **stratified** if the average of the labels of the observations is roughly the same in each of the subsets $\mathcal{D}_k$.

In the case of a classification problem, this means that the proportion of examples from each class is the same in each $\mathcal{D}_k$.

Leave-one-out: Case where the number of folds is equal to the number of observations.
Drawback: the trained models are very similar and differ little from a model trained on the entire dataset.

4.1.4 Bootstrap

For a dataset $\mathcal{D}$ and an interger B, *bootstrap* refers to the procedure of creating B samples $\mathcal{D}_1, \ldots, \mathcal{D}_B$ from $\mathcal{D}$, each obtained by drawing *n* examples from $\mathcal{D}$ **with replacement**.

Each example may therefore appear several times, or not at all, in any of the B samples.

4.2 Hyperparameter Optimization

A **hyperparameter** is a parameter of the learning algorithm (not of the model itself), such as the regularization coefficient in linear regression or the degree of the polynomial kernel in kernel SVM.

To determine the optimal value of one or more hyperparameters, *cross-validation* is often used. This requires defining a list of K hyperparameter values to be evaluated.

- Several strategies are possible:
- **Grid Search:** The most commonly used method. For each hyperparameter, a list of values to test is defined, and the generalization error of each possible combination of hyperparameter values is empirically evaluated.
  Drawback: may overlook some relevant hyperparameter values.
- **Random Search:** For each hyperparameter, a range of values to test is defined, and the generalization error of K combinations of hyperparameter values, drawn uniformly from these ranges, is empirically evaluated.

4.3 Performance Metrics

There are many ways to evaluate the predictive performance of a supervised learning model.

4.3.1 Confusion Matrix

Given a classification problem with C classes, the **confusion matrix** is a CxC matrix whose entry in row *m* and column *k* represents the number of examples belonging to class *m* for which the label *k* was predicted.

In the case of binary classification, the confusion matrix takes the following form:

		True class	
		0	1
Predicted class	0	True negatives(TN)	False Negatives(FN)
	1	False positives (FP)	True positives(TP)

4.3.2 Accuracy

Accuracy measures the proportion of correctly classified instances among all predictions.
It is defined as the ratio between the number of correctly predicted examples and the total number of examples.
Although accuracy is an intuitive and widely used metric, it may be misleading in cases of class imbalance.

4.3.3 Precision, Recall and F_1-Score

For binary classification problems, the performance of a model can be further characterized using *precision* and *recall*:

- **Precision (Positive Predictive Value):** the proportion of correctly predicted positive examples among all examples predicted as positive: Précision $= \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{F}P}$.
- **Recall (Sensitivity or True Positive Rate):** the proportion of correctly predicted positive examples among all actual positive examples: Recall $= \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$.

- The **F_1-score** provides a single metric that balances precision and recall. It is defined as the harmonic mean of the two measures:

$$F_1 - \text{score} = 2\frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = \frac{2\,\text{TP}}{2\text{TP} + \text{FP} + \text{FN}}$$

4.3.4 ROC Curve and AUC

The *Receiver Operating Characteristic (ROC) curve* represents the trade-off between the true positive rate (sensitivity) and the false positive rate (1 − specificity) for different decision thresholds.
The *Area Under the Curve (AUC)* summarizes this trade-off as a single scalar value.
A model with an AUC close to 1 indicates strong discriminative ability, whereas an AUC close to 0.5 suggests random performance.

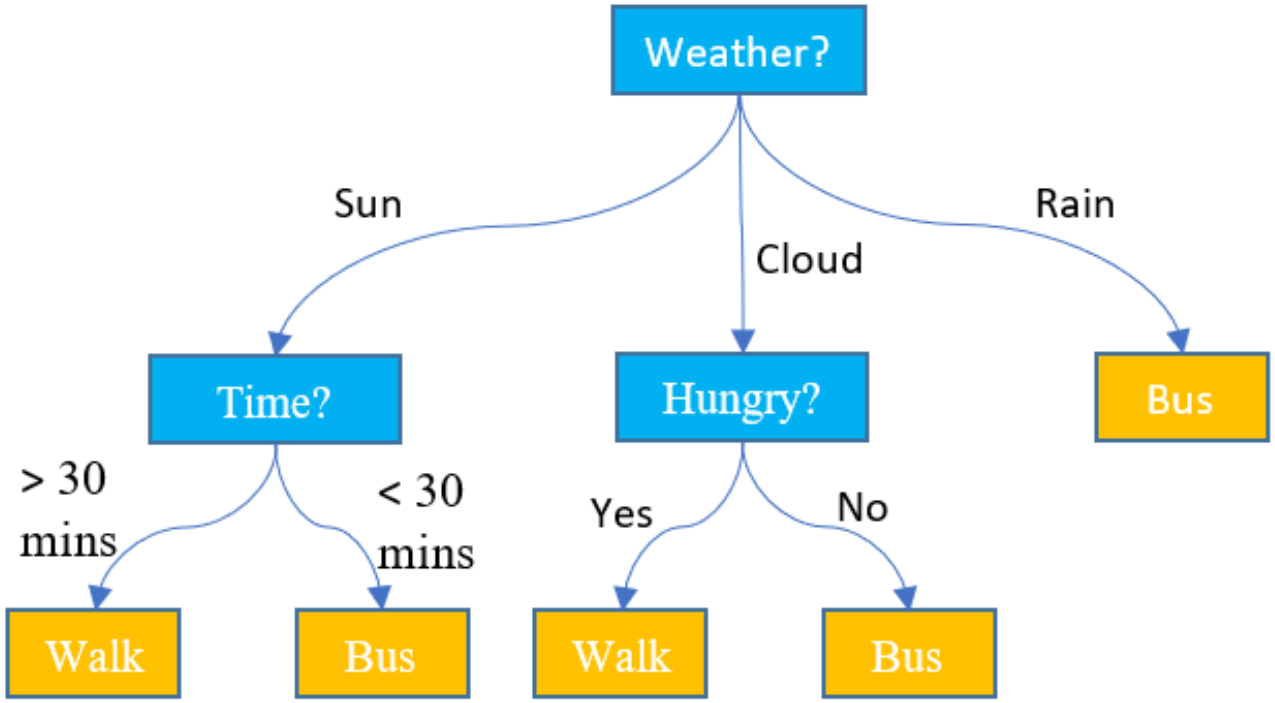
5. Decision trees, random forests, K-nearest neighbors

5.1 Decision Trees

Decision trees are among the simplest machine learning algorithms. Nevertheless, they can achieve very strong performance, particularly when used within ensemble methods. It is not uncommon to see such algorithms ranking among the top performers in Kaggle competitions.

A decision tree is, as its name suggests, a *tree* in the mathematical sense — a set of nodes connected by edges and leading to leaves.

At each node, a test is performed on the value of a variable. Depending on the result, one proceeds to one of the child nodes. This process is repeated until a node with no descendants is reached, which is then called a *leaf*.
The leaf indicates the predicted class of the processed data instance.


Weather?
Sun
Cloud
Rain
Time?
Hungry?
Bus
> 30 mins
< 30 mins
Yes
No
Walk
Bus
Walk
Bus

Several algorithms can be used to construct decision trees. Their general principles are similar.

The algorithm implemented in Python's *scikit-learn* library is the **CART** (Classification and Regression Tree) algorithm, developed by Breiman and co-authors in 1984.

At the output of the tree, some algorithms provide only the predicted class (for example, "accept" or "reject"). Others return *class membership probabilities*.
For instance, if a leaf contains 8 observations belonging to class 1 and 2 observations belonging to class 2, instead of simply predicting "class 1," the algorithm will output a probability of 0.8 for class 1 and 0.2 for class 2.

In *scikit-learn*, the user can either obtain the class with the highest probability using the *predict* method, or retrieve the probability distribution over all classes using *predict_proba.*

Choosing the Split Point
When constructing the tree, the algorithm must determine which variable to use next and what value to select as the *split point*.
The general principle is to choose the split that increases the *purity* of the resulting leaves — that is, each leaf should contain observations that are as homogeneous as possible with respect to the target variable.

Splitting Criteria
At each node of the decision tree, the algorithm must decide on which variable to split and at what threshold.
The objective is to create subsets that are as *homogeneous* as possible with respect to the target variable.
Several criteria can be used to measure the quality of a split, the most common being **information gain** (based on *entropy*) and the **Gini index**.

Entropy

Entropy measures the impurity or disorder within a dataset.

For a classification problem with C classes, the entropy of a node N is defined as:

$$H(N) = 1 - \sum_{c=1}^{C} p_c(N) \ \log_2 \ (p_c(N)),$$

where $p_c(N)$ is the proportion of examples belonging to class c in the node N.

A node is considered *pure* (i.e., all examples belong to the same class) when its entropy equals zero. If a node contains the same proportion of examples of each class, then its entropy is $\log_2 C$, therefore equal to 1 in the case of binary classification.

Gini Index

Another commonly used measure of impurity is the **Gini index**, defined as:

$$G(N) = \sum_{c=1}^{C} p_c(N)\left(1 - p_c(N)\right) = 1 - \sum_{c=1}^{C} p_c(N)^2 .$$

A node is pure when its Gini index equals zero. If a node contains the same proportion of examples of each class, then its Gini index is $1 - \frac{1}{C}$, therefore equal to $\frac{1}{2}$ in the case of binary classification.

In practice, the Gini index and entropy-based measures often produce similar trees.

Once the impurity measure has been selected, the **information gain** is computed for each potential split.
This gain is used to compare the impurity of the parent node with that of its child nodes.
The variable that results in the highest gain is chosen for the split.

For **numerical variables**, potential *splitting points* are pre-defined.
For **categorical variables**, each distinct value (or category) serves as a possible split point.
When the dataset contains a large number of variables, the number of possible tests at each step can become very large.
In this case, two common strategies are used:

- **“best”** — all possible split points are tested, and the one yielding the best gain is selected;
- **“random”** — a subset of potential split points is chosen randomly.

In *scikit-learn*, the impurity criterion is specified using the *criterion* parameter “Gini” or “entropy”, and the splitting strategy is controlled via the *splitter* parameter “best” or “random”.

Stopping Criteria
In the absence of additional constraints, the construction of a decision tree continues until each leaf is *pure* (i.e., contains only examples from a single class) or contains only one observation.
However, this usually leads to *overfitting*.

Several **stopping criteria** are therefore used. In *scikit-learn*, the most common parameters are:

- Maximum depth (*max_depth*) sets the maximum allowable depth of the tree.
- Minimum number of samples required to split a node (*min_samples_split*) : a node becomes a leaf if it contains fewer samples than this threshold.
- Minimum number of samples required in a leaf (min_samples_leaf). if a split would result in a leaf containing fewer samples than this number, the split is canceled, and the node is left as a leaf.
- Maximum number of features to consider at each split (max_features) : limits how many features are evaluated when searching for the best split.
- Maximum number of leaves (*max_leaf_nodes*).

- Minimum gain required for a split to be performed (*min_impurity_decrease) :* a split will only be made if it results in a reduction of impurity greater than or equal to this value.
- Minimum impurity required to allow a split (*min_impurity_split) :* defines the minimum impurity threshold a node must have before it can be split.

Il importe de choisir quels critères d'arrêt permettent les meilleurs résultats sur un dataset donné.

It is important to determine which combination of stopping criteria yields the best results for a given dataset.
In the case of **multiclass classification**, the tree must contain at least as many leaves as there are classes.

5.2 Random forests

Decision trees offer several advantages, but they are often too simple to achieve high predictive performance.

Random forests are an ensemble learning method based on the idea of building multiple decision trees on different subsets of the data. These trees can be trained in parallel, which helps reduce computation time. By aggregating the predictions of many trees, random forests generally achieve much better performance than individual trees.

In *scikit-learn*, the **RandomForestClassifier** class can be used to construct a forest, train it, make predictions, and evaluate performance metrics. Its parameters are similar to those of decision trees, with the addition of one specifying the number of trees **n_estimators.**

5.3 K-Nearest Neighbors (KNN)

- K-Nearest Neighbors (KNN) is a non-parametric method. The training dataset itself serves as the model.
  Each new observation is classified by assigning it the majority label among the *K* nearest training samples in the case of a classification problem,
  or by taking the average of the labels of its *K* nearest neighbors in the case of a regression problem.
- *K* is a hyperparameter that must be determined.
  If *K* is too small, only a few neighbors are selected, and the algorithm becomes highly sensitive to outliers (overfitting).
  If *K* is too large, the boundaries between classes tend to become blurred.

The value of *K* is often determined through cross-validation or set approximately to $\sqrt{n}$, where *n* is the number of training examples. Even values of *K* are generally avoided to prevent ties.

In *scikit-learn*, theKNeighborsClassifier class is used for this method, and data should be normalized beforehand.

XGBoost **(eXtreme Gradient Boosting)**

XGBoost is one of the most popular algorithms used by data scientists when working with numerical data.

Similar to random forests, it builds many decision trees. However, unlike random forests, these trees are built **sequentially** — each new tree focuses on the errors made by the previous ones.
This iterative process allows the model to gradually improve its performance by correcting previous mistakes.

Documentation: https://xgboost.readthedocs.io/en/latest/

6. Unsupervised Machine Learning Algorithms

Unsupervised Learning Methods:

- **Clustering:** the goal is to group data into clusters to better analyze or understand them.
- **Dimensionality reduction:** transforming a dataset with many features into one that contains only the most important features explaining the studied phenomenon.
- **Recommender systems:** suggesting products or services to users based on historical data.
- **Association rules:** identifying relationships between events to create rules such as *"If A and B occur, then C happens in x% of cases."*

Digital Learning Hub

6.1 The K-means algorithm

This algorithm is based on the distances between observations. The number of clusters must be chosen in advance, and the following steps are performed:

1. Randomly choose the centroids (cluster centers).
2. Assign each point to the centroid that is closest to it.
3. Compute the barycenter (mean position) of each cluster thus formed.
4. Move each centroid to its new barycenter position.
5. Repeat steps 2–4 until the results stabilize.

Drawbacks:

- The number of clusters must be specified by the user. In some cases, it is difficult to estimate, and several trials with different values may be required.
- The results often depend on the initial positions of the centroids. To improve reliability, it is common to run the algorithm multiple times and compare the outcomes.
- It is important to calculate the **inertia**, which is the sum of the squared distances between each point and its nearest centroid. Multiple models can then be compared, and the one with the lowest inertia should be selected.

6.2 The DBSCAN algorithm

Distance-based algorithms have two main drawbacks:

- The number of clusters must be specified.
- Non-convex clusters cannot be detected.

To overcome these issues, **density-based algorithms** can be used, such as **DBSCAN (Density-Based Spatial Clustering of Applications with Noise)**.

DBSCAN works by creating a neighborhood around each point. If a point has at least a certain number of neighbors (*min_samples*) within a specified distance (*eps*), it is considered a **core point** of a cluster. Points that do not belong to any cluster are treated as **noise**.

In *scikit-learn*, the algorithm is implemented in the DBSCAN classThe two key parameters are:

- eps : the radius that defines the neighborhood;
- min_samples : the minimum number of points required in the neighborhood for a point to be considered a cluster core.

Please give us feedback about the course

Digital Learning Hub