Classic machine learning algorithms

Johann Faouzi, Olivier Colliot

Chapter 2

Classic machine learning algorithms

Johann Faouzi[*1] **and Olivier Colliot**[1]

[1]Sorbonne Université, Institut du Cerveau - Paris Brain Institute - ICM, CNRS, Inria, Inserm, AP-HP, Hôpital de la Pitié-Salpêtrière, F-75013, Paris, France
[*]Corresponding author: e-mail address: johann.faouzi@gmail.com

Abstract

In this chapter, we present the main classic machine learning algorithms. A large part of the chapter is devoted to supervised learning algorithms for classification and regression, including nearest-neighbor methods, linear and logistic regressions, support vector machines and tree-based algorithms. We also describe the problem of overfitting as well as strategies to overcome it. We finally provide a brief overview of unsupervised learning methods, namely for clustering and dimensionality reduction. The chapter does not cover neural networks and deep learning as these will be presented in Chapters 3, 4, 5 and 6.

Keywords: machine learning, classification, regression, clustering, dimensionality reduction

1. Introduction

This chapter presents the main classic machine learning (ML) algorithms. There is a focus on supervised learning methods for classification and regression, but we also describe some unsupervised approaches. The chapter is meant to be readable by someone with no background in machine learning. It is nevertheless necessary to have some basic notions of linear

To appear in
O. Colliot (Ed.), *Machine Learning for Brain Disorders*, Springer

algebra, probabilities and statistics. If this is not the case, we refer the reader to chapters 2 and 3 of [1].

The rest of this chapter is organized as follows. Rather than grouping methods by categories (for instance classification or regression methods), we chose to present methods by increasing order of complexity. We first provide the notations in Section 2. We then describe a very intuitive family of methods, that of nearest neighbors (Section 3). We continue with linear regression (Section 4) and logistic regression (Section 5), the later being a classification technique. We subsequently introduce the problem of overfitting (Section 6) as well as strategies to mitigate it (Section 7). Section 8 describes support vector machines (SVM). Section 9 explains how binary classification methods can be extended to a multi-class setting. We then describe methods which are specifically adapted to the case of normal distributions (Section 10). Decision trees and random forests are described in Section 11. We then briefly describe some unsupervised learning algorithms, namely for clustering (Section 12) and dimensionality reduction (Section 13). The chapter ends with a description of kernel methods which can be used to extend linear algorithms to non-linear cases (Section 14). Box 1 summarizes the algorithms presented in this chapter, grouped by categories and then sorted in order of appearance.

2. Notations

Let n be the number of samples and p be the number of features. An input sample is thus a p-dimensional vector:

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_p \end{bmatrix}$$

An output sample is denoted by y. Thus, a sample is $(\boldsymbol{x}, y)$. The dataset of n samples can then be summarized as an $n \times p$ matrix $\boldsymbol{X}$ representing the input data and an n-dimensional vector $\boldsymbol{y}$ representing the target data:

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}^{(1)} \\ \vdots \\ \boldsymbol{x}^{(n)} \end{bmatrix} = \begin{bmatrix} x_1^{(1)} & \dots & x_p^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(n)} & \dots & x_p^{(n)} \end{bmatrix} \quad , \quad \boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

The input space is denoted by $\mathcal{I}$ and the set of training samples is denoted by $\mathcal{X}$.

In the case of regression, y is a real number. In the case of classification, y is a single label. More precisely, y can only take one of a finite set

Box 1: Main classic ML algorithms

- **Supervised learning**
  - **Classification**: nearest neighbors, logistic regression, support vector machine (SVM), naive Bayes, linear discriminant analysis (LDA), quadratic discriminant analysis, tree-based models (decision tree, random forest, extremely randomized trees).
  - **Regression**: nearest-neighbors, linear regression, support vector machine regression, tree-based models (decision tree, random forest, extremely randomized trees), kernel ridge regression.
- **Unsupervised learning**
  - **Clustering**: k-means, Gaussian mixture model.
  - **Dimensionality reduction**: principal component analysis (PCA), linear discriminant analysis (LDA), kernel principal component analysis.

of values called labels. The set of possible classes (i.e., labels) is denoted by $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_q\}$, with q being the number of classes. As the values of the classes are not meaningful, when there are only two classes, the classes are often called the positive and negative classes. In this case and also for mathematical reasons, without loss of generality, we assume the values of the classes to be $+1$ and -1.

3. Nearest-neighbor methods

One of the most intuitive approaches to machine learning is nearest neighbors. It is based on the following intuition: for a given input, its corresponding output is likely to be similar to the outputs of similar inputs. A real-life metaphor would be that, if a subject has similar characteristics than other subjects who were diagnosed with a given disease, then this subject is likely to also be suffering from this disease.

More formally, nearest-neighbor methods use the training samples from the neighborhood of a given point $\boldsymbol{x}$, denoted by $N(\boldsymbol{x})$, to perform prediction [2].

For regression tasks, the prediction is computed as a weighted mean of the target values in $N(\boldsymbol{x})$:

$$\hat{y} = \sum_{\boldsymbol{x}^{(i)} \in N(\boldsymbol{x})} w_i^{(\boldsymbol{x})} y^{(i)}$$

where $w_i^{(\boldsymbol{x})}$ is the weight associated to $\boldsymbol{x}^{(i)}$ to predict the output of $\boldsymbol{x}$, with $w_i^{(\boldsymbol{x})} \geq 0 \; \forall i$ and $\sum_i w_i^{(\boldsymbol{x})} = 1$.

For classification tasks, the predicted label corresponds to the label with the largest weighted sum of occurrences of each label:

$$\hat{y} = \arg\max_{\mathcal{C}} \sum_{\boldsymbol{x}^{(i)} \in N(\boldsymbol{x})} w_i^{(\boldsymbol{x})} \mathbf{1}_{y^{(i)} = \mathcal{C}_k}$$

A key parameter of nearest-neighbor methods is the *metric*, denoted by d, that is a mathematical function that defines dissimilarity. The metric is used to define the neighborhood of any point and can also be used to compute the weights.

3.1 Metrics

Many metrics have been defined for various types of input data such as vectors of real numbers, integers or booleans. Among these different types, vectors of real numbers are one of the most common types of

input data, for which the most commonly used metric is the Euclidean distance, defined as:

$$\forall \boldsymbol{x}, \boldsymbol{x}' \in \mathcal{I},\ \|\boldsymbol{x} - \boldsymbol{x}'\|_2 = \sqrt{\sum_{j=1}^{p} (x_j - x'_j)^2}$$

The Euclidean distance is sometimes referred to as the "ordinary" distance since it is the one based on the Pythagorean theorem and that everyone uses in their everyday lives.

3.2 Neighborhood

The two most common definitions of the neighborhood rely on either the number of neighbors or the radius around the given point. Figure 1 illustrates the differences between both definitions.

The k-nearest neighbor method defines the neighborhood of a given point $\boldsymbol{x}$ as the set of the k closest points to $\boldsymbol{x}$:

$$N(\boldsymbol{x}) = \{\boldsymbol{x}^{(i)}\}_{i=1}^{k} \quad \text{with} \quad d(\boldsymbol{x}, \boldsymbol{x}^{(1)}) \leq \ldots \leq d(\boldsymbol{x}, \boldsymbol{x}^{(n)})$$

The radius neighbor method defines the neighborhood of a given point $\boldsymbol{x}$ as the set of points whose dissimilarity to $\boldsymbol{x}$ is smaller than the given radius, denoted by r:

$$N(\boldsymbol{x}) = \{\boldsymbol{x}^{(i)} \in \mathcal{X} \,|\, d(\boldsymbol{x}, \boldsymbol{x}^{(i)}) < r\}$$

3.3 Weights

The two most common approaches to compute the weights are to use:

- uniform weights (all the weights are equal):

$$\forall i,\ w_i^{(\boldsymbol{x})} = \frac{1}{|N(\boldsymbol{x})|}$$

- weights inversely proportional to the dissimilarity:

$$\forall i,\ w_i^{(\boldsymbol{x})} = \frac{\frac{1}{d(\boldsymbol{x}^{(i)}, \boldsymbol{x})}}{\sum_j \frac{1}{d(\boldsymbol{x}^{(j)}, \boldsymbol{x})}} = \frac{1}{d(\boldsymbol{x}^{(i)}, \boldsymbol{x}) \sum_j \frac{1}{d(\boldsymbol{x}^{(j)}, \boldsymbol{x})}}$$

With uniform weights, every point in the neighborhood equally contributes to the prediction. With weights inversely proportional to the dissimilarity, closer points contribute more to the prediction than further points. Figure 2 illustrates the different decision functions obtained with uniform weights and weights inversely proportional to the dissimilarity for a 3-nearest neighbor classification model.

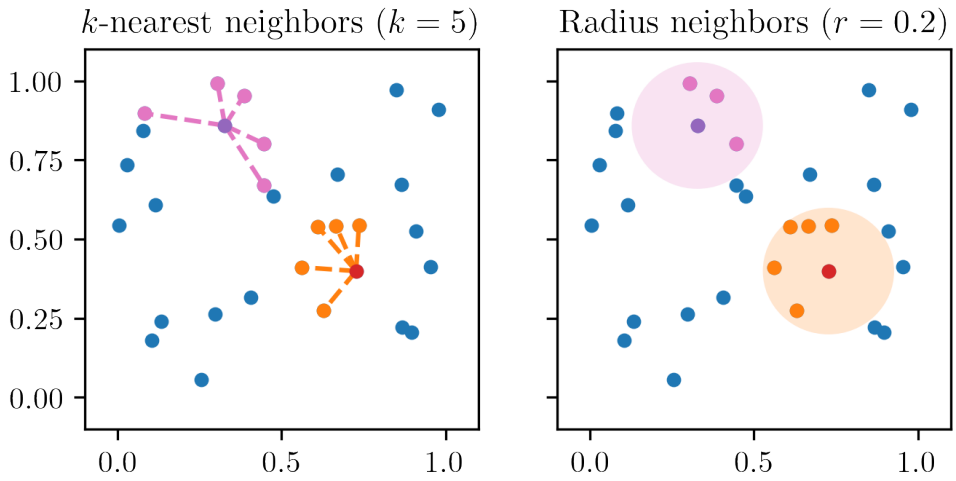

Figure 1: *Different definitions of the neighborhood. On the left, the neighborhood of a given point is the set of its 5 nearest neighbors. On the right, the neighborhood of a given point is the set of points whose dissimilarity is lower than the radius. For a given input, its neighborhood may be different depending on the definition used. The Euclidean distance is used as the metric in both examples.*

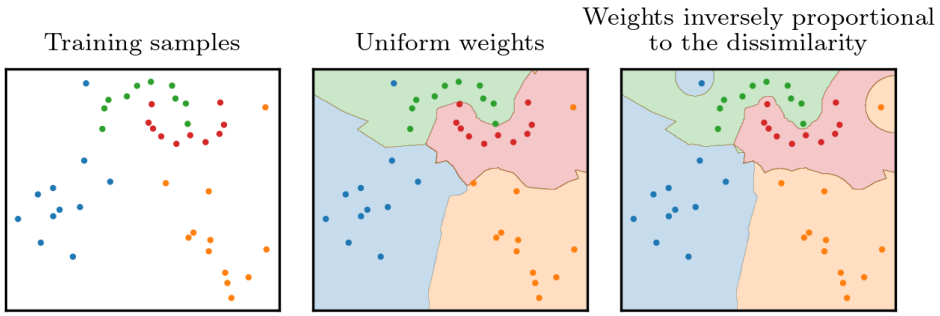

Figure 2: *Impact of the definition of the weights on the prediction function of a 3-nearest neighbor classification model. When the weights are inversely proportional to the dissimilarity, the classifier is more subject to outliers since the predictions in the close neighborhood of any input is mostly dedicated by the label of this input, independently of the number of neighbors used. With uniform weights, the prediction function tends to be smoother.*

3.4 Neighbor search

The brute-force method to compute the neighborhood for n points with p features is to compute the metric for each pair of inputs, which has a $\mathcal{O}(n^2p)$ algorithmic complexity (assuming that evaluating the metric for a pair of inputs has a complexity of $\mathcal{O}(p)$, which is the case for most metrics). However, it is possible to decrease this algorithmic complexity if the metric is a *distance*, that is if the metric d satisfies the following properties:

1. Non-negativity: $\forall \boldsymbol{a}, \boldsymbol{b}$, $d(\boldsymbol{a}, \boldsymbol{b}) \geq 0$
2. Identity: $\forall \boldsymbol{a}, \boldsymbol{b}$, $d(\boldsymbol{a}, \boldsymbol{b}) = 0$ if and only if $\boldsymbol{a} = \boldsymbol{b}$
3. Symmetry: $\forall \boldsymbol{a}, \boldsymbol{b}$, $d(\boldsymbol{a}, \boldsymbol{b}) = d(\boldsymbol{b}, \boldsymbol{a})$
4. Triangle Inequality: $\forall \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$, $d(\boldsymbol{a}, \boldsymbol{b}) + d(\boldsymbol{b}, \boldsymbol{c}) \geq d(\boldsymbol{a}, \boldsymbol{c})$

The key property is the *triangle inequality*, which has a simple interpretation: the shortest path between two points is a straight line. Mathematically, if $\boldsymbol{a}$ is far from $\boldsymbol{c}$ and $\boldsymbol{c}$ is close to $\boldsymbol{b}$ (i.e., $d(\boldsymbol{a}, \boldsymbol{c})$ is large and $d(\boldsymbol{b}, \boldsymbol{c})$ is small), then $\boldsymbol{a}$ is far from $\boldsymbol{b}$ (i.e., $d(\boldsymbol{a}, \boldsymbol{b})$ is large). This is obtained by rewriting the triangle inequality as follows:

$$\forall \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c},\ d(\boldsymbol{a}, \boldsymbol{b}) \geq d(\boldsymbol{a}, \boldsymbol{c}) - d(\boldsymbol{b}, \boldsymbol{c})$$

This means that it is not necessary to compute $d(\boldsymbol{a}, \boldsymbol{b})$ in this case. Therefore, the computational cost of a nearest neighbors search can be reduced to $\mathcal{O}(n \log(n) p)$ or better, which is a substantial improvement over the brute-force method for large n. Two popular methods that take advantage of this property are the *K-dimensional tree* structure [3] and the *ball tree* structure [4].

4. Linear regression

Linear regression is a regression model that linearly combines the features. Each feature is associated with a coefficient that represents the relative weight of this feature compared to the other features. A real-life metaphor would be to see the coefficients as the ingredients of a recipe: the key is to find the best balance (i.e., proportions) between all the ingredients in order to make the best cake.

Mathematically, a linear model is a model that linearly combines the features [5]:

$$f(\boldsymbol{x}) = w_0 + \sum_{j=1}^{p} w_j x_j$$

A common notation consists in including a 1 in $\boldsymbol{x}$ so that $f(\boldsymbol{x})$ can be written as the *dot product* between the vector $\boldsymbol{x}$ and the vector $\boldsymbol{w}$:

$$f(\boldsymbol{x}) = w_0 \times 1 + \sum_{j=1}^{p} w_j x_j = \boldsymbol{x}^\top \boldsymbol{w}$$

where the vector $\boldsymbol{w}$ consists of:

- the intercept (also known as bias) w_0, and
- the coefficients $(w_1, \ldots, w_p)$, where each coefficient w_j is associated with the corresponding feature x_j.

In the case of linear regression, $f(\boldsymbol{x})$ is the predicted output:

$$\hat{y} = f(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{w}$$

There are several methods to estimate the $\boldsymbol{w}$ coefficients. In this section, we present the oldest one which is known as *ordinary least squares regression*.

In the case of ordinary least squares regression, the cost function J is the sum of the squared errors on the training data (see Figure 3):

$$J(\boldsymbol{w}) = \sum_{i=1}^{n} \left(y^{(i)} - \hat{y}^{(i)}\right)^2 = \sum_{i=1}^{n} \left(y^{(i)} - \boldsymbol{x}^{(i)\top} \boldsymbol{w}\right)^2 = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2$$

One wants to find the optimal parameters $\boldsymbol{w}^\star$ that minimize the cost function:

$$\boldsymbol{w}^\star = \arg\min_{\boldsymbol{w}} J(\boldsymbol{w})$$

This optimization problem is *convex*, implying that any local minimum is a global minimum, and *differentiable*, implying that every local minimum has a null gradient. One therefore aims to find null gradients of the cost function:

$$\begin{aligned} & \nabla_{\boldsymbol{w}^\star} J = 0 \\ \implies & 2\boldsymbol{X}^\top \boldsymbol{X} \boldsymbol{w}^\star - 2\boldsymbol{X}^\top \boldsymbol{y} = 0 \\ \implies & \boldsymbol{X}^\top \boldsymbol{X} \boldsymbol{w}^\star = \boldsymbol{X}^\top \boldsymbol{y} \\ \implies & \boldsymbol{w}^\star = \left(\boldsymbol{X}^\top \boldsymbol{X}\right)^{-1} \boldsymbol{X}^\top \boldsymbol{y} \end{aligned}$$

Ordinary least square regression is one of the few machine learning optimization problem for which there exists a *closed formula*, i.e. the optimal solution can be computed using a finite number of standard operations such as addition, multiplication and evaluations of well-known functions.

Box 2: Linear regression

- **Main idea**: best hyperplane (i.e., line when $p = 1$, plane when $p = 2$) mapping the inputs and to the outputs.
- **Mathematical formulation**: linear relationship between the predicted output $\hat{y}$ and the input $\boldsymbol{x}$ that minimizes the sum of squared errors:

$$\hat{y} = w_0^\star + \sum_{j=1}^{n} w_j^\star x_j \quad \text{with} \quad \boldsymbol{w}^\star = \arg\min_{\boldsymbol{w}} \sum_{i=1}^{n} \left(y^{(i)} - \boldsymbol{x}^{(i)\top}\boldsymbol{w}\right)^2$$

- **Regularization**: can be penalized to avoid overfitting (ridge), to perform feature selection (lasso), or both (elastic-net). See Section 7.

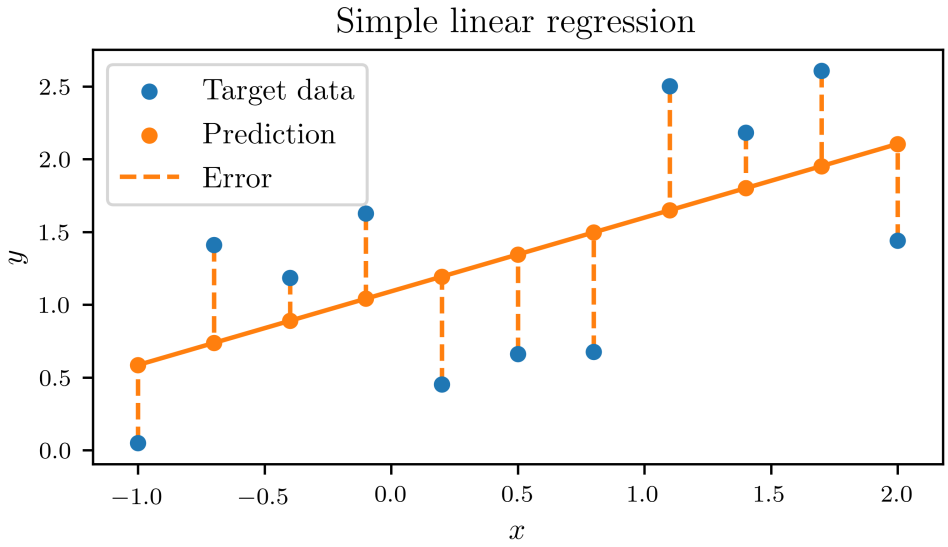

Figure 3: *Ordinary least squares regression. The coefficients (that is the intercept and the slope with a single predictor) are estimated by minimizing the sum of the squared errors.*

5. Logistic regression

Intuitively, linear regression consists in finding the line that best fits the data: the true output should be as close to the line as possible. For binary classification, one wants the line to separate both classes as well as possible: the samples from one class should all be in one subspace and the samples from the other class should all be in the other subspace, with the inputs being as far as possible from the line.

Mathematically, for binary classification tasks, a linear model is defined by an hyperplane splitting the input space into two subspaces such that each subspace is characteristic of one class. For instance, a line splits a plane into two subspaces in the two-dimensional case, while a plane splits a three-dimensional space into two subspaces. A hyperplane is defined by a vector $\boldsymbol{w} = (w_0, w_1, \ldots, w_p)$ and $f(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{w}$ corresponds to the *signed distance* between the input $\boldsymbol{x}$ and the hyperplane $\boldsymbol{w}$: in one subspace, the distance with any input is always positive, whereas in the other subspace, the distance with any input is always negative. Figure 4 illustrates the decision function in the two-dimensional case where both classes are linearly separable.

The sign of the signed distance corresponds to the decision function of a linear binary classification model:

$$\hat{y} = \text{sign}(f(\boldsymbol{x})) = \begin{cases} +1 & \text{if } f(\boldsymbol{x}) > 0 \\ -1 & \text{if } f(\boldsymbol{x}) < 0 \end{cases}$$

The logistic regression model is a probabilistic linear model that transforms the signed distance to the hyperplane into a probability using the sigmoid function [6], denoted by $\sigma(u) = \frac{1}{1+\exp(-u)}$.

Consider the linear model:

$$f(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{w} = w_0 + \sum_{i=j}^{p} w_j x_j$$

Then the probability of belonging to the positive class is:

$$P\left(\mathrm{y} = +1 | \mathbf{x} = \boldsymbol{x}\right) = \sigma(f(\boldsymbol{x})) = \frac{1}{1 + \exp\left(-f(\boldsymbol{x})\right)}$$

and that of belonging to the negative class is:

$$\begin{aligned} P\left(\mathrm{y} = -1 | \mathbf{x} = \boldsymbol{x}\right) &= 1 - P\left(\mathrm{y} = +1 | \mathbf{x} = \boldsymbol{x}\right) \\ &= \frac{\exp\left(-f(\boldsymbol{x})\right)}{1 + \exp\left(-f(\boldsymbol{x})\right)} \end{aligned}$$

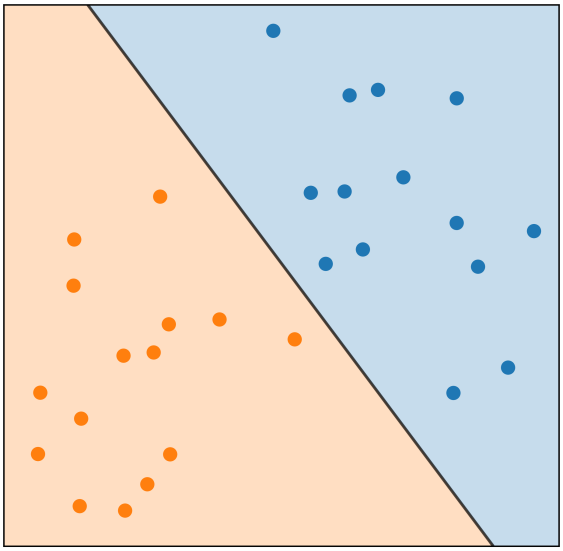

Figure 4: *Decision function of a logistic regression model. A logistic regression is a linear model, that is its decision function is linear. In the two-dimensional case, it separates a plane with a line.*

$$= \frac{1}{1 + \exp\left(f(\boldsymbol{x})\right)}$$
$$P\left(\mathrm{y} = -1 | \mathbf{x} = \boldsymbol{x}\right) = \sigma(-f(\boldsymbol{x}))$$

By applying the inverse of the sigmoid function, which is known as the logit function, one can see that the logarithm of the *odds ratio* is modeled as a linear combination of the features:

$$\log\left(\frac{P\left(\mathrm{y} = +1 | \mathbf{x} = \boldsymbol{x}\right)}{P\left(\mathrm{y} = -1 | \mathbf{x} = \boldsymbol{x}\right)}\right) = \log\left(\frac{P\left(\mathrm{y} = +1 | \mathbf{x} = \boldsymbol{x}\right)}{1 - P\left(\mathrm{y} = +1 | \mathbf{x} = \boldsymbol{x}\right)}\right) = f(\boldsymbol{x})$$

The $\boldsymbol{w}$ coefficients are estimated by maximizing the *likelihood* function, that is the function measuring the goodness of fit of the model to the training data:

$$L(\boldsymbol{w}) = \prod_{i=1}^{n} P\left(\mathrm{y} = y^{(i)} | \mathbf{x} = \boldsymbol{x}^{(i)}; \boldsymbol{w}\right)$$

For computational reasons, it is easier to maximize the *log-likelihood*, which is simply the logarithm of the likelihood:

$$\log(L(\boldsymbol{w})) = \sum_{i=1}^{n} \log\left(P\left(\mathrm{y} = y^{(i)} | \mathbf{x} = \boldsymbol{x}^{(i)}; \boldsymbol{w}\right)\right)$$

Box 3: Logistic regression

- **Main idea**: best hyperplane (i.e., line) that separates two classes.
- **Mathematical formulation**: the signed distance to the hyperplane is mapped into the probability to belong to the positive class using the sigmoid function:

$$f(\boldsymbol{x}) = w_0 + \sum_{j=1}^{n} w_j x_j$$

$$P\left(\mathrm{y} = +1 | \mathbf{x} = \boldsymbol{x}\right) = \sigma(f(\boldsymbol{x})) = \frac{1}{1 + \exp\left(-f(\boldsymbol{x})\right)}$$

- **Estimation**: likelihood maximization.
- **Regularization**: can be penalized to avoid overfitting (ℓ_2 penalty), to perform feature selection (ℓ_1 penalty), or both (elastic-net penalty).

$$\begin{aligned} &= \sum_{i=1}^{n} \log\left(\sigma\left(y^{(i)} f(\boldsymbol{x}^{(i)}; \boldsymbol{w})\right)\right) \\ &= \sum_{i=1}^{n} -\log\left(1 + \exp\left(y^{(i)} \boldsymbol{x}^{(i)\top} \boldsymbol{w}\right)\right) \\ \log(L(\boldsymbol{w})) &= -\sum_{i=1}^{n} \log\left(1 + \exp\left(y^{(i)} \boldsymbol{x}^{(i)\top} \boldsymbol{w}\right)\right) \end{aligned}$$

Finally, we can rewrite this maximization problem as a minimization problem by noticing that $\max_{\boldsymbol{w}} \log(L(\boldsymbol{w})) = -\min_{\boldsymbol{w}} -\log(L(\boldsymbol{w}))$:

$$\max_{\boldsymbol{w}} \log(L(\boldsymbol{w})) = -\min_{\boldsymbol{w}} \sum_{i=1}^{n} \log\left(1 + \exp\left(y^{(i)} \boldsymbol{x}^{(i)\top} \boldsymbol{w}\right)\right)$$

We can see that the $\boldsymbol{w}$ coefficients that maximize the likelihood are also the coefficients that minimize the sum of the *logistic loss* values, with the logistic loss being defined as:

$$\ell_{\text{logistic}}(y, f(\boldsymbol{x})) = \log\left(1 + \exp\left(y f(\boldsymbol{x})\right)\right) / \log(2)$$

Unlike for linear regression, there is no closed formula for this minimization. One thus needs to use an optimization method such as for instance

gradient descent which was presented in Section 3 of Chapter 1. In practice, more sophisticated approaches such as quasi-Newton methods and variants of stochastic gradient descent are often used.

6. Overfitting and regularization

The original formulations of ordinary least square regression and logistic regression are *unregularized* models, that is the model is trained to fit the training data as much as possible. Let us consider a real-life example as it is very similar to human learning. If a person learns by heart the content of a book, they are able to solve the exercises in the book, but unable to apply the theoretical concepts to new exercises or real-life situations. If a person only quickly reads through the book, they are probably unable to solve neither the exercises in the book nor new exercises.

The corresponding concepts are known as *overfitting* and *underfitting* in machine learning. Overfitting occurs when a model fits too well the training data and generalizes poorly to new data. Oppositely, underfitting occurs when a model does not capture well enough the characteristics of the training data, thus also generalizes poorly to new data.

Overfitting and underfitting are related to frequently used terms in machine learning: *bias* and *variance*. Bias is defined as the expected (i.e., mean) difference between the true output and the predicted output. Variance is defined as the variability of the predicted output. For instance, let us consider an algorithm predicting the age of a person from a picture. If the algorithm *always* underestimates or overestimates the age, then the algorithm is biased. If the algorithm makes *both large and small errors*, then the algorithm has a high variance.

Ideally, one would like to have a model with a small bias and a small variance. However, the bias of a model tends to increase when decreasing its variance, and the variance of the model tends to increase when decreasing its bias. This phenomenon is known as the *bias-variance trade-off*. Figure 5 illustrates this phenomenon. One can also notice it by computing the squared error between the true output y (fixed) and the predicted output $\hat{y}$ (random variable): its expected value is the sum of the squared bias of $\hat{y}$ and the variance of $\hat{y}$:

$$\begin{aligned}
\mathbb{E}\left[(y-\hat{y})^2\right] &= \mathbb{E}\left[y^2 - 2y\hat{y} + \hat{y}^2\right] \\
&= y^2 - 2y\mathbb{E}\left[\hat{y}\right] + \mathbb{E}\left[\hat{y}^2\right] \\
&= y^2 - 2y\mathbb{E}\left[\hat{y}\right] + \mathbb{E}\left[\hat{y}^2\right] + \mathbb{E}\left[\hat{y}\right]^2 - \mathbb{E}\left[\hat{y}\right]^2 \\
&= \left(\mathbb{E}\left[\hat{y}\right] - y\right)^2 + \mathbb{E}\left[\hat{y}^2\right] - \mathbb{E}\left[\hat{y}\right]^2
\end{aligned}$$

$$
\begin{aligned}
&= \left(\mathbb{E}\left[\hat{\mathrm{y}}\right] - y\right)^2 + \mathbb{E}\left[\hat{\mathrm{y}}^2 - \mathbb{E}\left[\hat{\mathrm{y}}\right]^2\right] \\
&= \left(\mathbb{E}\left[\hat{\mathrm{y}}\right] - y\right)^2 + \mathbb{E}\left[\hat{\mathrm{y}}^2 - 2\mathbb{E}\left[\hat{\mathrm{y}}\right]^2 + \mathbb{E}\left[\hat{\mathrm{y}}\right]^2\right] \\
&= \left(\mathbb{E}\left[\hat{\mathrm{y}}\right] - y\right)^2 + \mathbb{E}\left[\hat{\mathrm{y}}^2 - 2\hat{\mathrm{y}}\mathbb{E}\left[\hat{\mathrm{y}}\right] + \mathbb{E}\left[\hat{\mathrm{y}}\right]^2\right] \\
&= \left(\mathbb{E}\left[\hat{\mathrm{y}}\right] - y\right)^2 + \mathbb{E}\left[\left(\hat{\mathrm{y}} - \mathbb{E}\left[\hat{\mathrm{y}}\right]\right)^2\right] \\
\mathbb{E}\left[(y - \hat{\mathrm{y}})^2\right] &= \underbrace{\left(\mathbb{E}\left[\hat{\mathrm{y}}\right] - y\right)^2}_{\text{bias}^2} + \underbrace{\mathrm{Var}\left[\hat{\mathrm{y}}\right]}_{\text{variance}}
\end{aligned}
$$

7. Penalized models

Depending on the class of algorithms, there exist different strategies to tackle overfitting.

For neighbor methods, the number of neighbors used to define the neighborhood of any input and the strategy to compute the weights are the key hyperparameters to control the bias-variance trade-off. For models that are presented in the remaining sections of this chapter, we mention strategies to address the bias-variance trade-off in their respective sections. In this section, we present the most commonly used strategies for models whose parameters are optimized by minimizing a cost function defined as the mean loss values over all the training samples:

$$\min_{\boldsymbol{w}} J(\boldsymbol{w}) \quad \text{with} \quad J(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} \ell\left(y^{(i)}, f(\boldsymbol{x}^{(i)}; \boldsymbol{w})\right)$$

This is for instance the case of the linear and logistic regression methods presented in the previous sections.

7.1 Penalties

The main idea is to introduce a *penalty term* $\mathrm{Pen}(\boldsymbol{w})$ that will constraint the parameters $\boldsymbol{w}$ to have some desired properties. The most common penalties are the ℓ_2 penalty, the ℓ_1 penalty and the elastic-net penalty.

7.1.1. ℓ_2 penalty

The ℓ_2 penalty is defined as the squared ℓ_2 norm of the $\boldsymbol{w}$ coefficients:

$$\ell_2(\boldsymbol{w}) = \|\boldsymbol{w}\|_2^2 = \sum_{j=1}^{p} w_j^2$$

The ℓ_2 penalty forces each coefficient w_i not to be too large and makes the coefficients more robust to collinearity (i.e., when some features are approximately linear combinations of the other features).

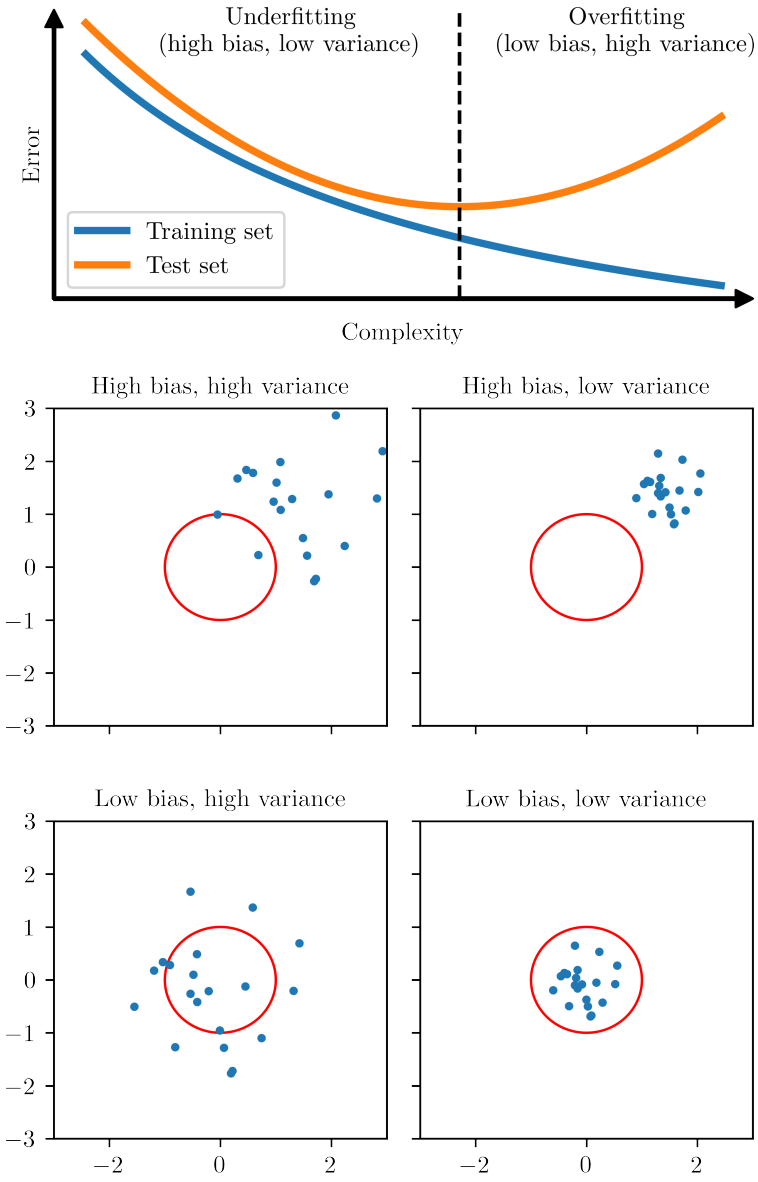

Figure 5: *Illustration of underfitting and overfitting. Underfitting occurs when a model is too simple and does not capture well enough the characteristics of the training data, leading to high bias and low variance. Oppositely, overfitting occurs when a model is too complex and learns the noise in the training data, leading to low bias and high variance.*

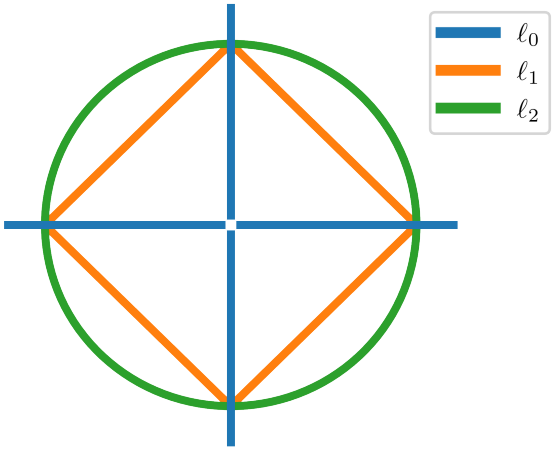

Figure 6: *Unit balls of the ℓ_0, ℓ_1 and ℓ_2 norms. For each norm, the set of points in $\mathbb{R}^2$ whose norm is equal to 1 is plotted. The ℓ_1 norm is the best convex approximation to the ℓ_0 norm. Note that the lines for the ℓ_0 norm extend to $-\infty$ and $+\infty$, but are cut for plotting reasons.*

7.1.2. ℓ_1 penalty

The ℓ_2 penalty forces the values of the parameters not to be too large, but does not incentive to make small values tend to zero. Indeed, the square of a small value is even smaller. When the number of features is large, or when interpretability is important, it can be useful to make the algorithm select the most important features. The corresponding metric is the ℓ_0 "norm" (which is not a proper norm in the mathematical sense), defined as the number of nonzero elements:

$$\ell_0(\boldsymbol{w}) = \|\boldsymbol{w}\|_0 = \sum_{j=1}^{p} \mathbf{1}_{w_j \neq 0}$$

However, the ℓ_0 "norm" is neither differentiable nor convex (which are useful properties to solve an optimization problem, but this is not further detailed for the sake of conciseness). The best convex differentiable approximation of the ℓ_0 "norm" is the the ℓ_1 norm (see Figure 6), defined as the sum of the absolute values of each element:

$$\ell_1(\boldsymbol{w}) = \|\boldsymbol{w}\|_1 = \sum_{j=1}^{p} |w_j|$$

7.1.3. Elastic-net penalty

Both the ℓ_2 and ℓ_1 penalties have their upsides and downsides. In order to try to obtain the best of penalties, one can add both penalties in the objective function. The combination of both penalties is known as the *elastic-net* penalty:

$$\mathrm{EN}(\boldsymbol{w}, \alpha) = \alpha\|\boldsymbol{w}\|_1 + (1-\alpha)\|\boldsymbol{w}\|_2^2$$

where $\alpha \in [0, 1]$ is a hyperparameter representing the proportion of the ℓ_1 penalty compared to the ℓ_2 penalty.

7.2 New optimization problem

A natural approach would be to add a constraint to the minimization problem:

$$\min_{\boldsymbol{w}} J(\boldsymbol{w}) \qquad \text{subject to} \qquad \mathrm{Pen}(\boldsymbol{w}) < c \tag{1}$$

which reads as "Find the optimal parameters that minimize the cost function J among all the parameters $\boldsymbol{w}$ that satisfy $\mathrm{Pen}(\boldsymbol{w}) < c$" for a positive real number c. Figure 7 illustrates the optimal solution of a simple linear regression task with different constraints. This figure also highlights the sparsity property of the ℓ_1 penalty (the optimal parameter for the horizontal axis is set to zero) that the ℓ_2 penalty does not have (the optimal parameter for the horizontal axis is small but different from zero).

Although this approach is appealing due to its intuitiveness and the possibility to set the maximum possible penalty on the parameters $\boldsymbol{w}$, it leads to a minimization problem that is not trivial to solve. A similar approach consists in adding the regularization term in the cost function:

$$\min_{\boldsymbol{w}} J(\boldsymbol{w}) + \lambda \times \mathrm{Pen}(\boldsymbol{w}) \tag{2}$$

where $\lambda > 0$ is a hyperparameter that controls the weights of the penalty term compared to the mean loss values over all the training samples. This formulation is related to the Lagrangian function of the minimization problem with the penalty constraint.

This formulation leads to a minimization problem with no constraint which is much easier to solve. One can actually show that Equation 1 and Equation 2 are related: solving Equation 2 for a given λ, whose optimal solution is denoted by $\boldsymbol{w}_\lambda^\star$, is equivalent to solving Equation 1 for $c = \mathrm{Pen}(\boldsymbol{w}_\lambda^\star)$. In other words, solving Equation 2 for a given λ is equivalent to solving Equation 1 for c whose value is only known after finding the optimal solution of Equation 2.

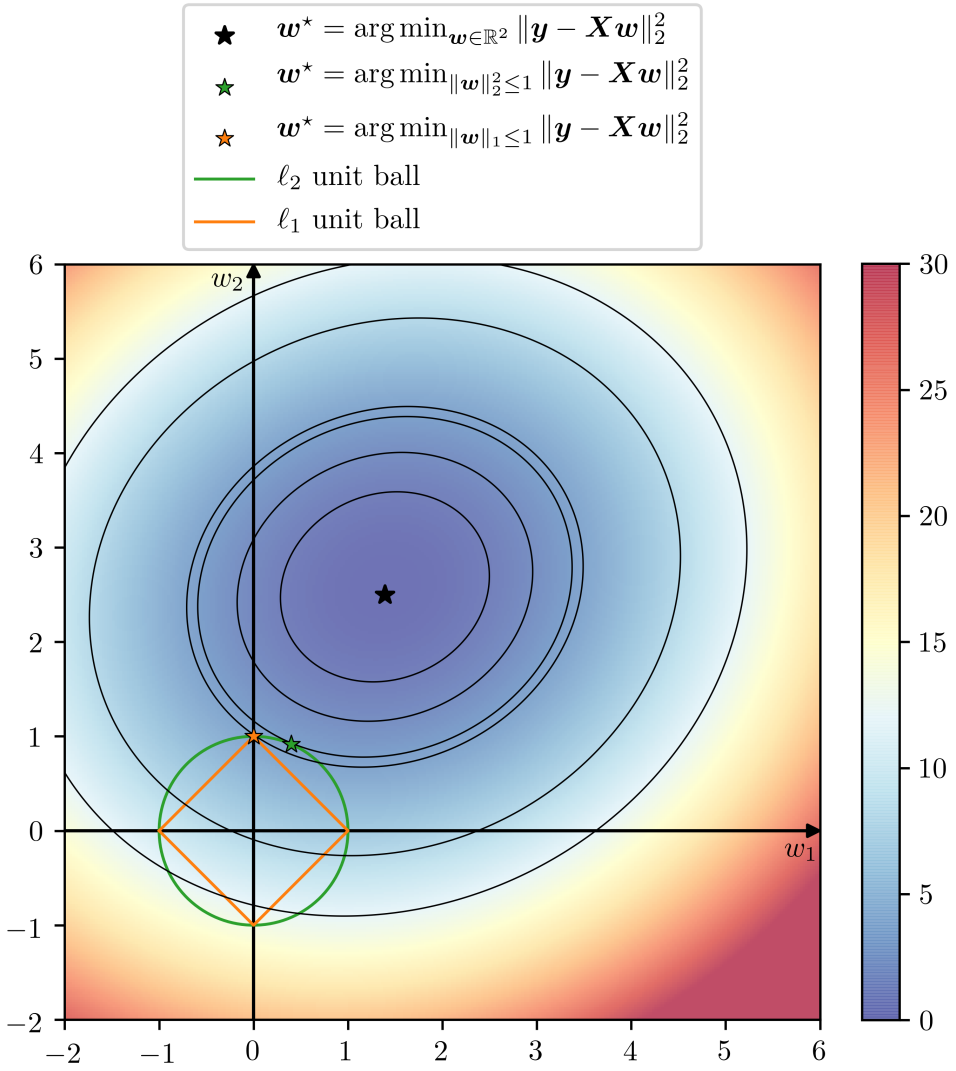

Figure 7: *Illustration of the minimization problem with a constraint on the penalty term. The plot represents the value of the loss function for different values of the two coefficients for a linear regression task. The black star indicates the optimal solution with no constraint. The green and orange stars indicate the optimal solutions when imposing a constraint on the ℓ_2 and ℓ_1 norms of the parameters $\boldsymbol{w}$ respectively.*

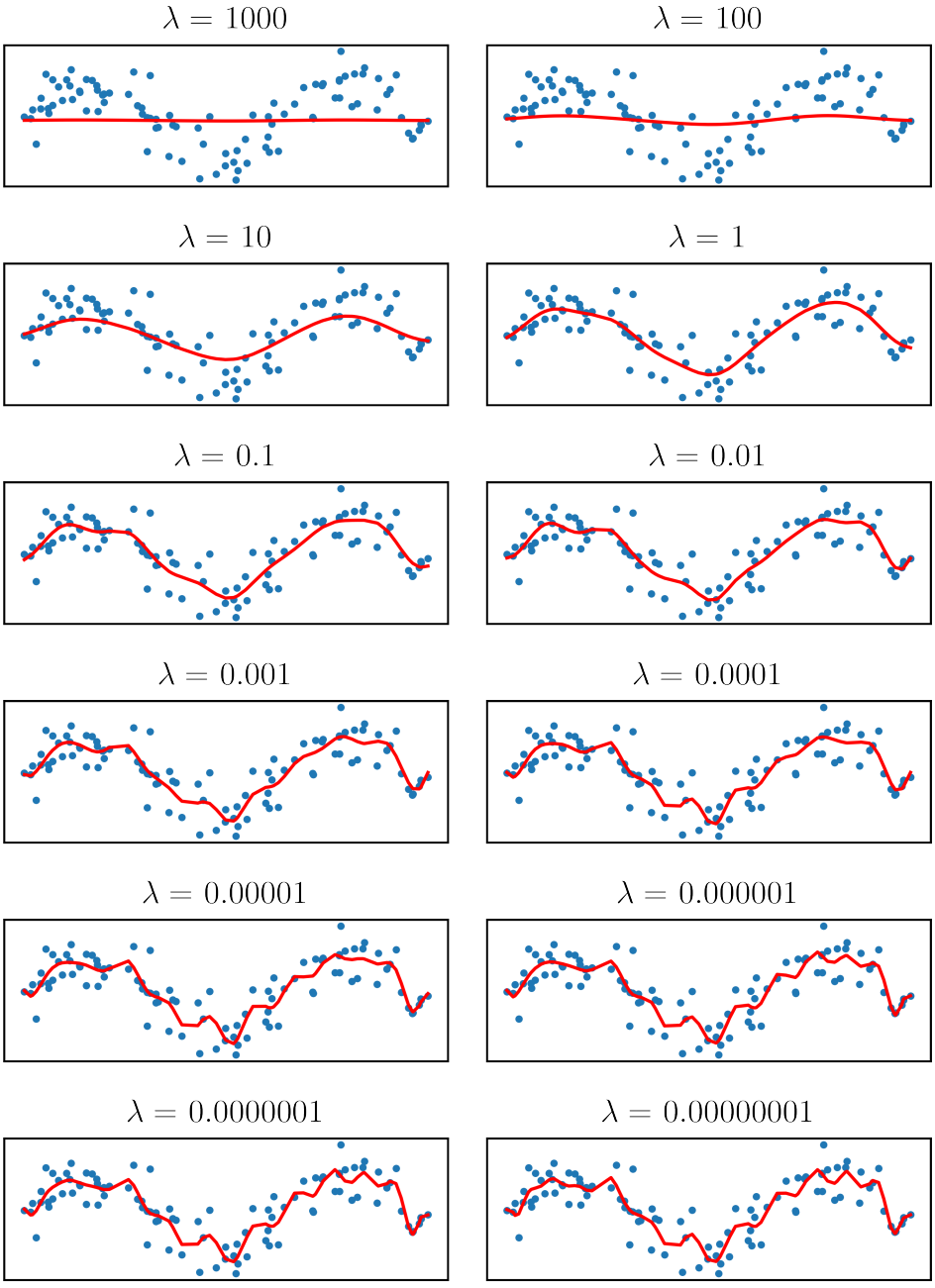

Figure 8: *Illustration of regularization. A kernel ridge regression algorithm is fitted on the training data (blue points) with different values of λ, which is the weight of the regularization in the cost function. The smaller values of λ, the smaller the weight of the ℓ_2 regularization. The algorithm underfits (respectively overfits) the data when the value of λ is too large (respectively low).*

Figure 8 illustrates the impact of the regularization term $\lambda \times \text{Pen}(\boldsymbol{w})$ on the prediction function of a kernel ridge regression algorithm (see Section 14 for more details) for different values of λ. For high values of λ, the regularization term is dominating the mean loss value, making the prediction function not fitting well enough the training data (underfitting). For small values of λ, the mean loss value is dominating the regularization term, making the prediction function fitting too well the training data (overfitting). A good balance between the mean loss value and the regularization term is required to learn the best function.

Since linear regression is one of the oldest and best-known models, the aforementioned penalties were originally introduced for linear regression:

- Linear regression with the ℓ_2 penalty is also known as ridge [7]:

$$\min_{\boldsymbol{w}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2 + \lambda \|\boldsymbol{w}\|_2^2$$

  As in ordinary least squares regression, there exists a closed formula for the optimal solution:

$$\boldsymbol{w}^\star = \left(\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{X}^\top \boldsymbol{y}$$

- Linear regression with the ℓ_1 penalty is also known as lasso [8]:

$$\min_{\boldsymbol{w}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2 + \lambda \|\boldsymbol{w}\|_1$$

- Linear regression with the elastic-net penalty is also known as elastic-net [9]:

$$\min_{\boldsymbol{w}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2 + \lambda\alpha \|\boldsymbol{w}\|_1 + \lambda(1-\alpha) \|\boldsymbol{w}\|_2^2$$

The penalties can also be added in other models such as logistic regression, support-vector machines, artificial neural networks, etc.

8. Support-vector machine

Linear and logistic regression take into account every training sample in order to find the best line, which is due to their corresponding loss functions: the squared error is zero only if the true and predicted outputs are equal, and the logistic loss is always positive. One could argue that the training samples whose outputs are "easily" well predicted are not relevant: only the training samples whose outputs are not "easily" well predicted or are wrongly predicted should be taken into account. The support vector machine (SVM) algorithm is based on this principle.

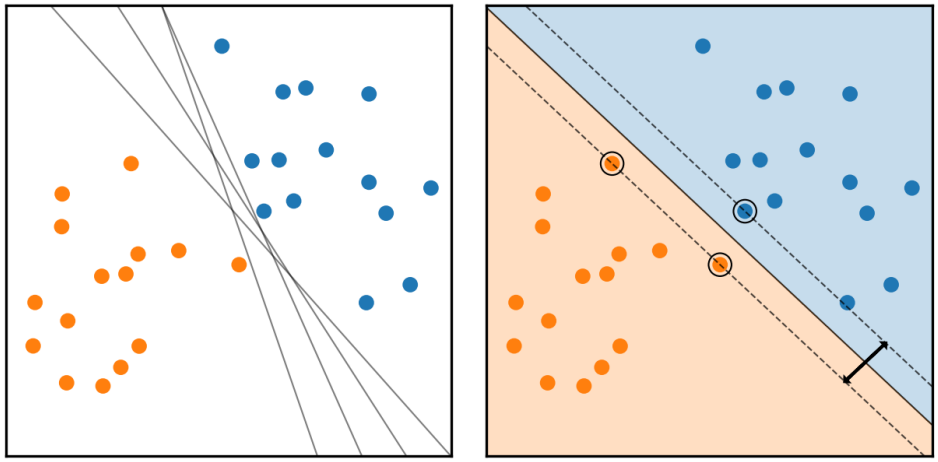

Figure 9: *Support vector machine classifier with linearly separable classes. When two classes are linearly separable, there exists an infinite number of hyperplanes separating them (left). The decision function of the support vector machine classifier is the hyperplane that maximizes the margin, that is the distance between the hyperplane and the closest points to the hyperplane (right). Support vectors are highlighted with a black circle surrounding them.*

8.1 Original formulation

The original support vector machine algorithm was invented in 1963 and was a linear binary classification algorithm [10]. Figure 9 illustrates the main concept of its original version. When both classes are linearly separable, there exists an infinite number of hyperplanes that separate both classes. The SVM algorithm finds the hyperplane that maximizes the *margin*, that is the distance between the hyperplane and the closest points of both classes to the hyperplane, while linearly separating both classes.

The SVM algorithm was later updated to non-separable classes [11]. Figure 10 illustrates the role of the margin in this case. The dashed lines correspond to the hyperplanes defined by the equations $\boldsymbol{x}^\top \boldsymbol{w} = +1$ and $\boldsymbol{x}^\top \boldsymbol{w} = -1$. The margin is the distance between both hyperplanes and is equal to $2/\|\boldsymbol{w}\|_2^2$. It defines which samples are included in the decision function of the model: a sample is included if and only if it is inside the margin, or outside the margin and misclassified. Such samples are called *support vectors* and are illustrated in Figure 10 with a black circle surrounding them. In this case, the margin can be seen a regularization term: the larger the margin is, the more support vectors are included in the decision function, the more regularized the model is.

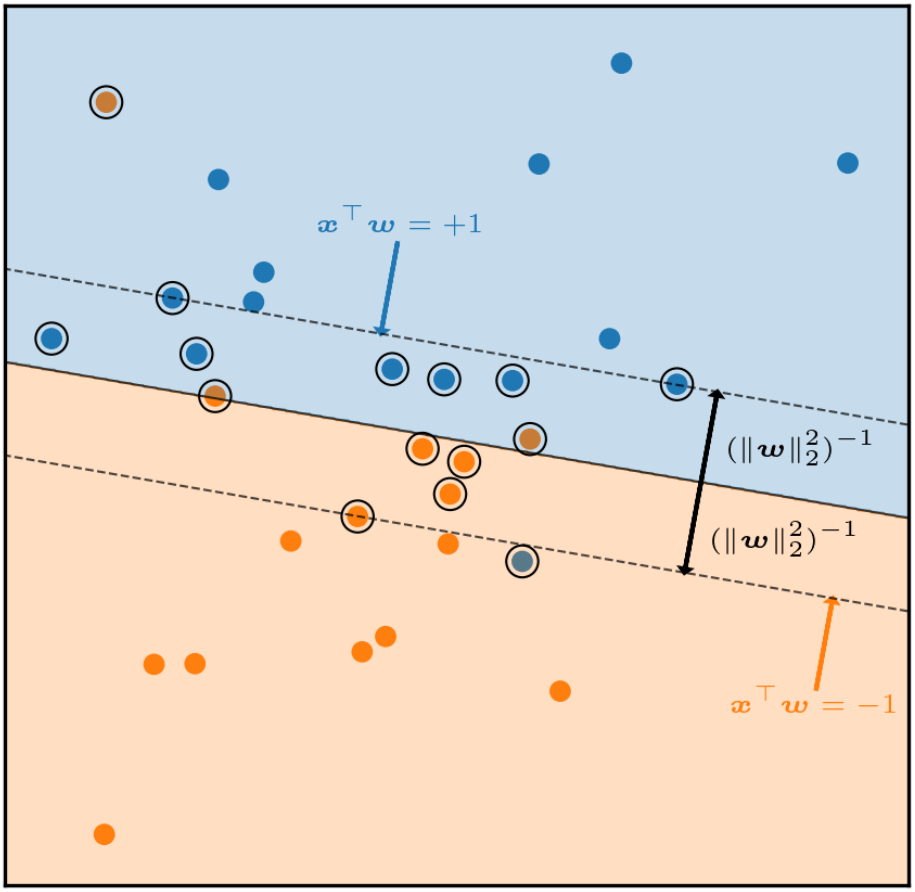

Figure 10: *Decision function of a support-vector machine classifier with a linear kernel when both classes are not strictly linearly separable. The support vectors are the training points within the margin of the decision function and the misclassified training points. The support vectors are highlighted with a black circle surrounding them.*

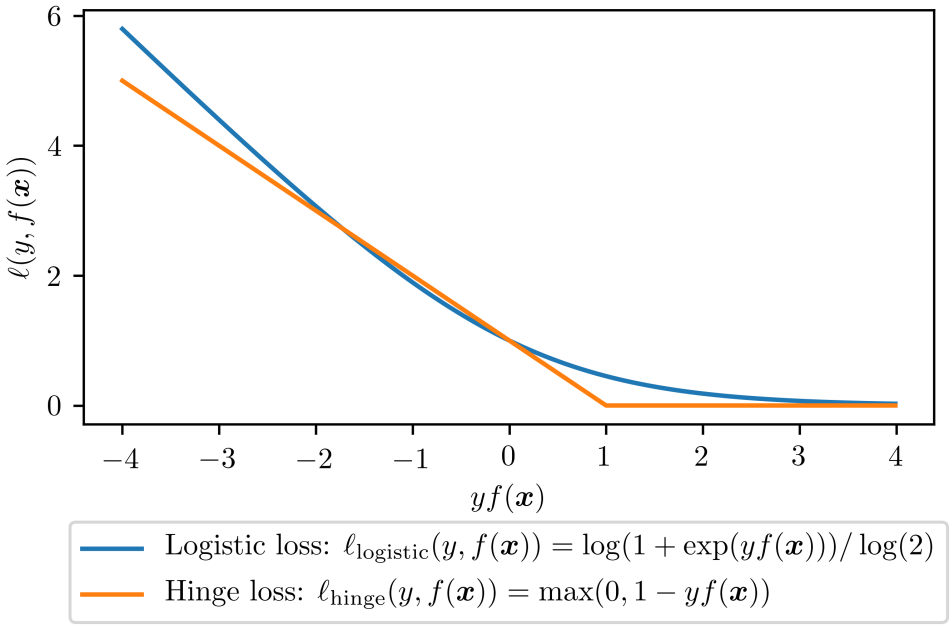

Figure 11: *Binary classification losses. The logistic loss is always positive, even when the point is accurately classified with high confidence (i.e., when $yf(\boldsymbol{x}) \gg 0$), whereas the hinge loss is equal to zero when the point is accurately classified with good confidence (i.e., when $yf(\boldsymbol{x}) \geq 1$).*

The loss function for the SVM algorithm is called the *hinge loss* and is defined as:

$$\ell_{\text{hinge}}(y, f(\boldsymbol{x})) = \max(0, 1 - yf(\boldsymbol{x}))$$

Figure 11 illustrates the curves of the logistic and hinge losses. The logistic loss is always positive, even when the point is accurately classified with high confidence (i.e., when $yf(\boldsymbol{x}) \gg 0$), whereas the hinge loss is equal to zero when the point is accurately classified with good confidence (i.e., when $yf(\boldsymbol{x}) \geq 1$). One can see that a sample $(\boldsymbol{x}, y)$ is a support vector if and only if $yf(\boldsymbol{x}) \geq= 1$, that is if and only if $\ell_{\text{hinge}}(y, f(\boldsymbol{x})) = 0$.

The optimal $\boldsymbol{w}$ coefficients for the original version are estimated by minimizing an objective function consisting of the sum of the *hinge loss* values and a ℓ_2 penalty term (which is inversely proportional to the margin):

$$\min_{\boldsymbol{w}} \sum_{i=1}^{n} \max(0, 1 - y^{(i)} \boldsymbol{x}^{(i)\top} \boldsymbol{w}) + \frac{1}{2C} \|\boldsymbol{w}\|_2^2$$

8.2 General formulation with kernels

The SVM algorithm was later updated to non-linear decision functions with the use of *kernels* [12].

In order to have a non-linear decision function, one could map the input space $\mathcal{I}$ into another space (often called the *feature space*), denoted by $\mathcal{G}$, using a function denoted by ϕ:

$$\begin{aligned} \phi \colon \mathcal{I} &\to \mathcal{G} \\ \boldsymbol{x} &\mapsto \phi(\boldsymbol{x}) \end{aligned}$$

The decision function would still be linear (with a dot product), but in the feature space:

$$f(\boldsymbol{x}) = \phi(\boldsymbol{x})^\top \boldsymbol{w}$$

Unfortunately, solving the corresponding minimization problem is not trivial:

$$\min_{\boldsymbol{w}} \sum_{i=1}^{n} \max\left(0, 1 - y^{(i)} \phi(\boldsymbol{x}^{(i)})^\top \boldsymbol{w}\right) + \frac{1}{2C} \|\boldsymbol{w}\|_2^2 \tag{3}$$

Nonetheless, two mathematical properties make the use of non-linear transformations in the feature space possible: the *kernel trick* and the *representer theorem*.

The kernel trick asserts that the dot product in the feature space can be computed using only the points from the input space and a *kernel function*, denoted by K:

$$\forall \boldsymbol{x}, \boldsymbol{x}' \in \mathcal{I},\ \phi(\boldsymbol{x})^\top \phi(\boldsymbol{x}') = K(\boldsymbol{x}, \boldsymbol{x}')$$

The representer theorem [13, 14] asserts that, under certain conditions on the kernel K and the feature space $\mathcal{G}$ associated with the function ϕ, any minimizer of Equation 3 admits the following form:

$$f = \sum_{i=1}^{n} \alpha_i K(\cdot, \boldsymbol{x}^{(i)})$$

where $\boldsymbol{\alpha}$ solves:

$$\min_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \max(0, 1 - y^{(i)} [\boldsymbol{K}\boldsymbol{\alpha}]_i) + \frac{1}{2C} \boldsymbol{\alpha}^\top \boldsymbol{K} \boldsymbol{\alpha}$$

where $\boldsymbol{K}$ is the $n \times n$ matrix consisting of the evaluations of the kernel on all the pairs of training samples: $\forall i, j \in \{1, \dots, n\}$, $K_{ij} = K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$.

Because the hinge loss is equal to zero if and only if $y f(\boldsymbol{x})$ is greater than or equal to 1, only the training samples $(\boldsymbol{x}^{(i)}, y^{(i)})$ such that $y^{(i)} f(\boldsymbol{x}^{(i)}) <$

Box 4: Support-vector machine

- **Main idea**: hyperplane (i.e., line) that maximizes the margin (i.e., the distance between the hyperplane and the closest inputs to the hyperplane).
- **Support vectors**: only the misclassified inputs and the inputs well classified but with low confidence are taken into account.
- **Non-linearity**: decision function can be non-linear with the use of non-linear kernels.
- **Regularization**: ℓ_2 penalty.

1 have a nonzero α_i coefficient. These points are the so-called support vectors and this is why they are the only training samples contributing to the decision function of the model:

$$\text{SV} = \{i \in \{1, \dots, n\} \,|\, \alpha_i \neq 0\}$$
$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i K(\boldsymbol{x}, \boldsymbol{x}^{(i)}) = \sum_{i \in \text{SV}} \alpha_i K(\boldsymbol{x}, \boldsymbol{x}^{(i)})$$

The kernel trick and the representer theorem show that it is more practical to work with the kernel K instead of the mapping function ϕ. Popular kernel functions include:

- the linear kernel:
$$K(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^\top \boldsymbol{x}'$$
- the polynomial kernel:
$$K(\boldsymbol{x}, \boldsymbol{x}') = \left(\gamma\, \boldsymbol{x}^\top \boldsymbol{x}' + c_0\right)^d \quad \text{with} \quad \gamma > 0,\, c_0 \geq 0,\, d \in \mathbb{N}^*$$
- the sigmoid kernel:
$$K(\boldsymbol{x}, \boldsymbol{x}') = \tanh\left(\gamma\, \boldsymbol{x}^\top \boldsymbol{x}' + c_0\right) \quad \text{with} \quad \gamma > 0,\, c_0 \geq 0$$
- the radial basis function (RBF) kernel:
$$K(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\gamma \left\|\boldsymbol{x} - \boldsymbol{x}'\right\|_2^2\right) \quad \text{with} \quad \gamma > 0$$

The linear kernel yields a linear decision function and is actually identical to the original formulation of the SVM algorithm (one can show that there is a mapping between the $\boldsymbol{\alpha}$ and $\boldsymbol{w}$ coefficients). Non-linear kernels allow for non-linear, more complex, decision functions. This is particularly useful when the data is not linearly separable, which is the most common use case. Figure 12 illustrates the decision function and the margin of a SVM classification model for four different kernels.

The SVM algorithm was also extended to regression tasks with the use of the *ε-insensitive loss*. Similarly to the hinge loss, which is equal to zero for points that are correctly classified and outside the margin, the ε-insensitive loss is equal to zero when the error between the true target value and the predicted value is not greater than ε:

$$\ell_{\varepsilon-\text{insensitive}}(y, f(\boldsymbol{x})) = \max(0, |y - f(\boldsymbol{x})| - \varepsilon)$$

The objective function for the SVM regression algorithm combines the values of ε-insensitive loss of the training points and the ℓ_2 penalty:

$$\min_{\boldsymbol{w}} \sum_{i=1}^{n} \max\left(0, \left|y^{(i)} - \phi(\boldsymbol{x}^{(i)})^\top \boldsymbol{w}\right| - \varepsilon\right) + \frac{1}{2C}\|\boldsymbol{w}\|_2^2$$

Figure 13 illustrates the curves of three regression losses. The squared error loss takes very small values for small errors and very high values for high errors, whereas the absolute error loss takes small values for small errors and high values for high errors. Both losses take small but non-zero values when the error is small. On the contrary, the ε-insensitive loss is null when the error is small and otherwise equal to the absolute error loss minus ε.

9. Multiclass classification

The classification algorithms that we presented so far, logistic regression and support-vector machines, are binary classification algorithms: they can only be used when there are only two possible outcomes. However, in practice, it is common to have more than two possible outcomes. For instance, differential diagnosis of brain disorders is often between several, and not only two, diseases.

Several strategies have been proposed to extend any binary classification algorithm to multiclass classification tasks. They all rely on transforming the multiclass classification task into several binary classification tasks. In this section, we present the most commonly used strategies: *one-vs-rest*, *one-vs-one* and *error correcting output code* [15]. Figure 14 illustrates the main ideas of these approaches. But first, we

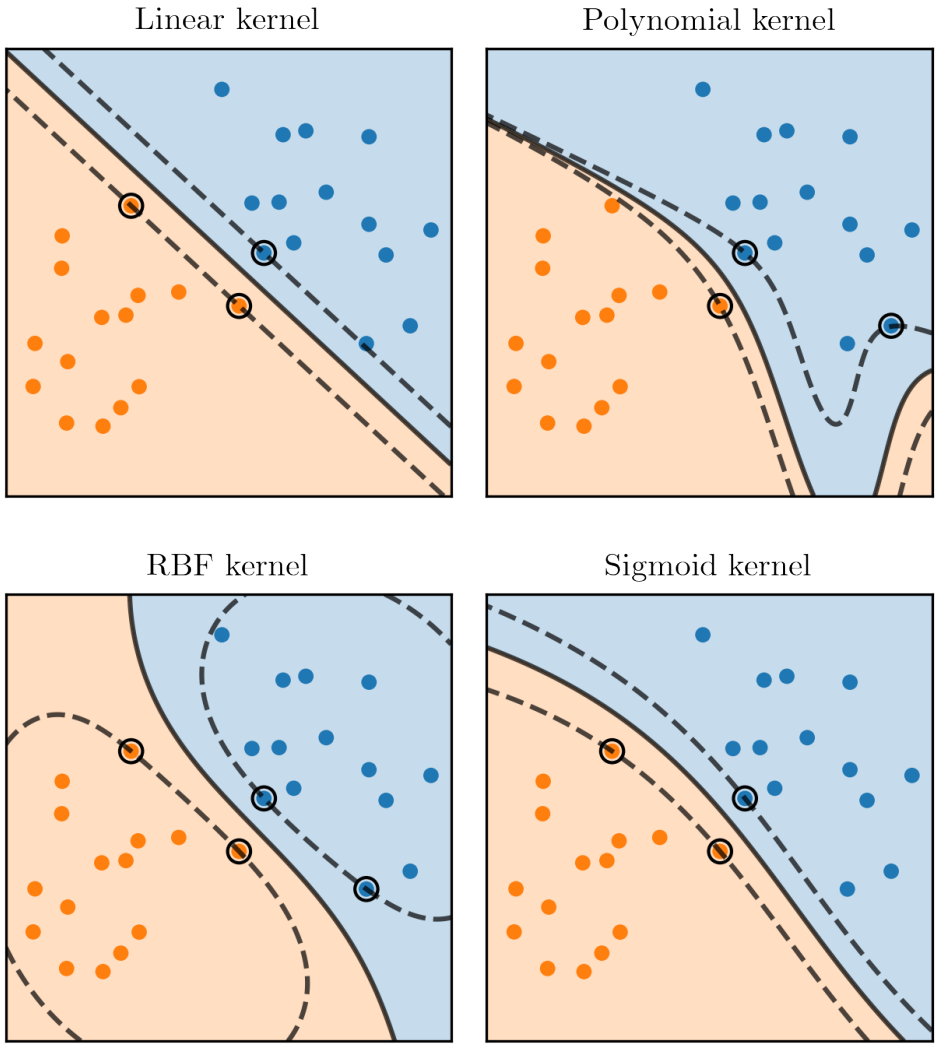

Figure 12: *Impact of the kernel on the decision function of a support vector machine classifier. A non-linear kernel allows for a non-linear decision function.*

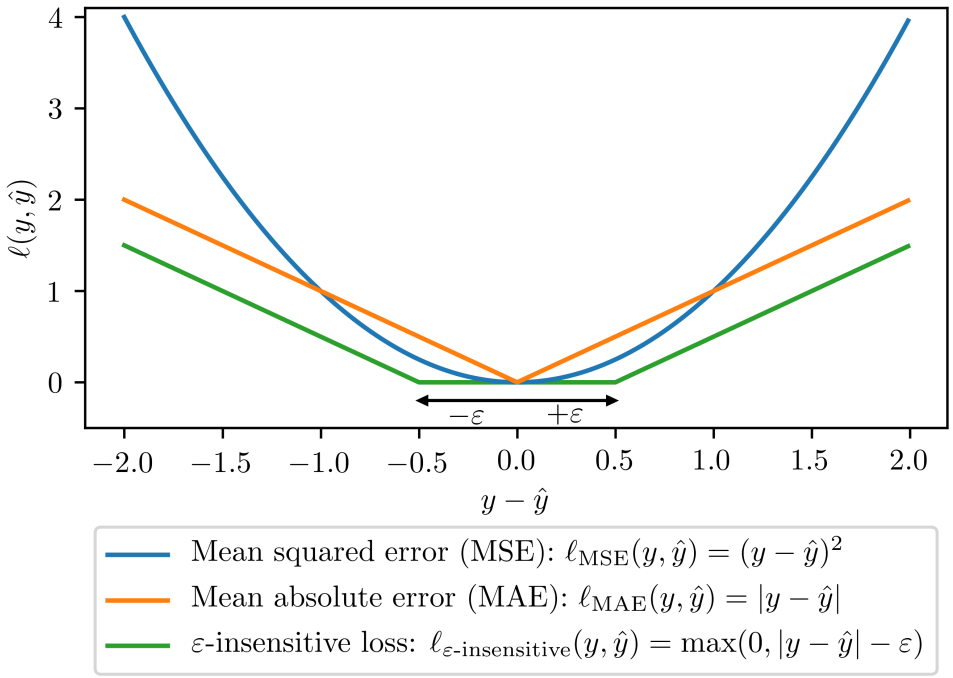

Figure 13: *Regression losses. The squared error loss takes very small values for small errors and very large values for large errors, whereas the absolute error loss takes small values for small errors and large values for large errors. Both losses take small but non-zero values when the error is small. On the contrary, the ε-insensitive loss is null when the error is small and otherwise equal the absolute error loss minus ε. When computed over several samples, the squared and absolute error losses are often referred to as mean squared error (MSE) and mean absolute error (MAE) respectively.*

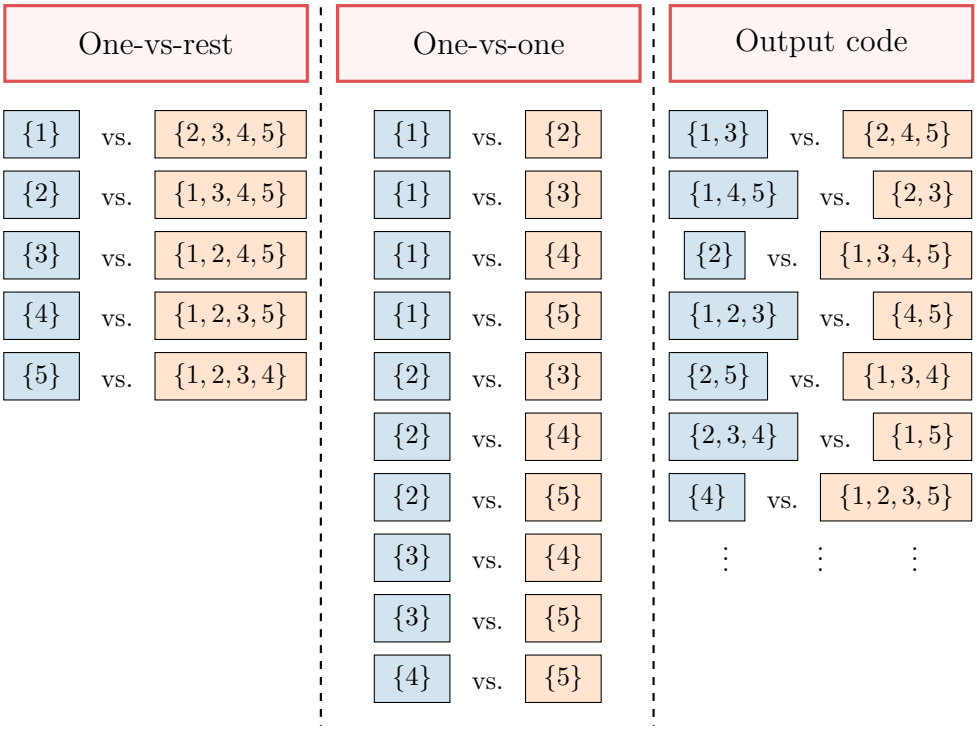

Figure 14: *Main approaches to convert a multiclass classification task into several binary classification tasks. In the one-vs-rest approach, each class is associated to a binary classification model that is trained to separate this class from all the other classes. In the one-vs-one approach, a binary classification algorithm is trained on each pair of classes. In the error correcting output code approach, the classes are (randomly) split into two groups and a binary classification algorithm is trained for each split.*

present a natural extension of logistic regression to multiclass classification tasks which is often referred to as *multinomial logistic regression* [5].

9.1 Multinomial logistic regression

For binary classification, logistic regression is characterized by a hyperplane: the signed distance to the hyperplane is mapped into the probability of belonging to the positive class using the sigmoid function. However, for multiclass classification, a single hyperplane is not enough to characterize all the classes. Instead, each class $\mathcal{C}_k$ is characterized by a hyperplane $\boldsymbol{w}_k$ and, for any input $\boldsymbol{x}$, one can compute the signed distance $\boldsymbol{x}^\top \boldsymbol{w}_k$ between the input $\boldsymbol{x}$ and the hyperplane $\boldsymbol{w}_k$. The signed

distances are mapped into probabilities using the softmax function, defined as $\text{softmax}\left(x_1, \ldots, x_q\right) = \left(\frac{\exp(x_1)}{\sum_{j=1}^{q} \exp(x_j)}, \ldots, \frac{\exp(x_q)}{\sum_{j=1}^{q} \exp(x_j)}\right)$, as follows:

$$\forall k \in \{1, \ldots, q\},\ P(\mathrm{y} = \mathcal{C}_k | \mathbf{x} = \boldsymbol{x}) = \frac{\exp\left(\boldsymbol{x}^\top \boldsymbol{w}_k\right)}{\sum_{j=1}^{q} \exp\left(\boldsymbol{x}^\top \boldsymbol{w}_j\right)}$$

The coefficients $(\boldsymbol{w}_k)_{1 \leq k \leq q}$ are still estimated by maximizing the likelihood function:

$$L(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_q) = \prod_{i=1}^{n} \prod_{k=1}^{q} P\left(\mathrm{y} = \mathcal{C}_k | \mathbf{x} = \boldsymbol{x}^{(i)}\right)^{\mathbf{1}_{y^{(i)} = \mathcal{C}_k}}$$

which is equivalent to minimizing the negative log-likelihood:

$$\begin{aligned}
&- \log(L(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_q)) \\
&= - \sum_{i=1}^{n} \sum_{k=1}^{q} \mathbf{1}_{y^{(i)} = \mathcal{C}_k} \log\left(P\left(\mathrm{y} = \mathcal{C}_k | \mathbf{x} = \boldsymbol{x}^{(i)}\right)\right) \\
&= \sum_{i=1}^{n} - \sum_{k=1}^{q} \mathbf{1}_{y^{(i)} = \mathcal{C}_k} \log\left(\frac{\exp\left(\boldsymbol{x}^{(i)\top} \boldsymbol{w}_k\right)}{\sum_{j=1}^{q} \exp\left(\boldsymbol{x}^{(i)\top} \boldsymbol{w}_j\right)}\right) \\
&= \sum_{i=1}^{n} \ell_{\text{cross-entropy}}\left(y^{(i)}, \text{softmax}\left(\boldsymbol{x}^{(i)\top} \boldsymbol{w}_1, \ldots, \boldsymbol{x}^{(i)\top} \boldsymbol{w}_q\right)\right)
\end{aligned}$$

where $\ell_{\text{cross entropy}}$ is known as the *cross-entropy loss* and is defined, for any label y and any vector of probabilities $(\pi_1, \ldots, \pi_q)$, as:

$$\ell_{\text{cross-entropy}}(y, (\pi_1, \ldots, \pi_q)) = - \sum_{k=1}^{q} \mathbf{1}_{y = \mathcal{C}_k} \pi_k$$

This loss is commonly used to train artificial neural networks on classification tasks and is equivalent to the logistic loss in the binary case.

Figure 15 illustrates the impact of the strategy used to handle a multiclass classification task on the decision function.

9.2 One-vs-rest

A strategy to transform a multiclass classification task into several binary classification tasks is to fit a binary classification algorithm for each class: the positive class is the given class, and the negative class consists of all the other classes merged into a single class. This strategy is known as *one-vs-rest*. The advantage of this strategy is that each class is characterized by a single model, so that it is possible to gain deeper knowledge about

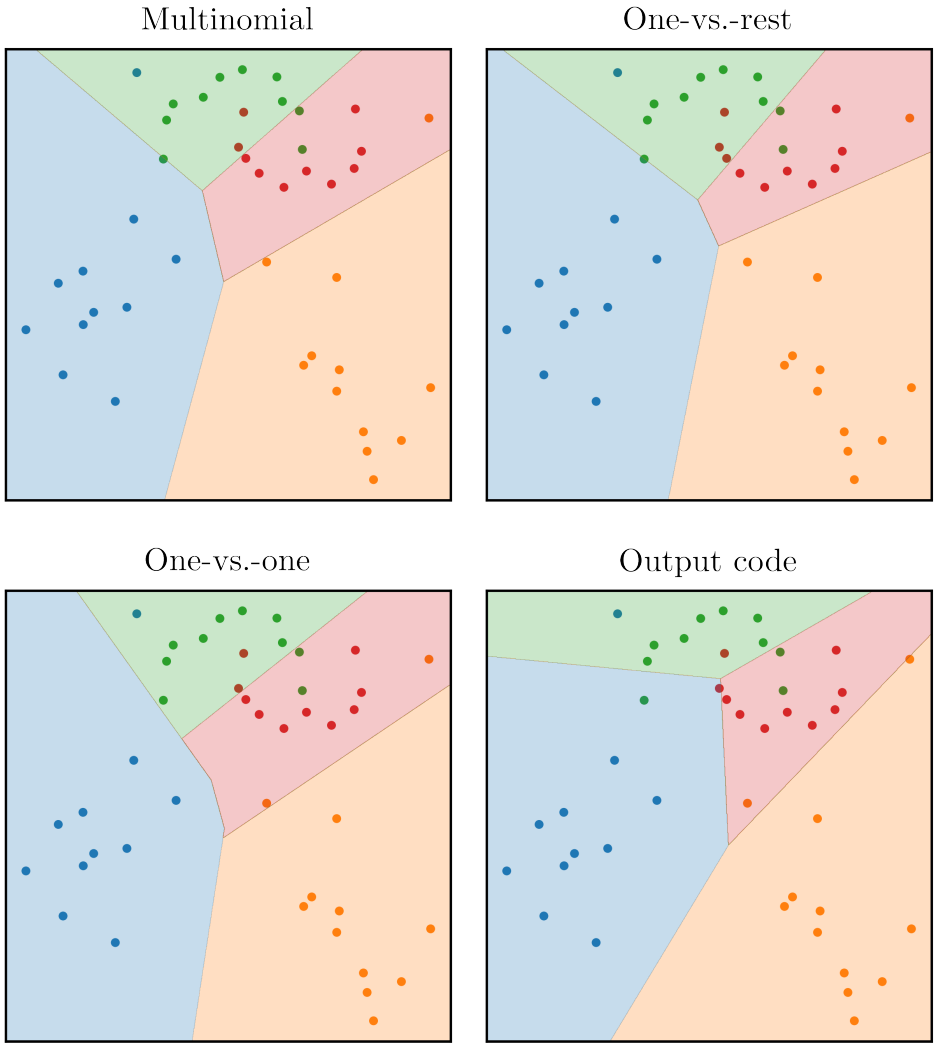

Figure 15: *Illustration of the impact of the strategy used to handle a multiclass classification task on the decision function of a logistic regression model.*

the class by inspecting its corresponding model. A consequence is that the predictions for new samples take into account the confidence of the models: the predicted class for a new input is the class for which the corresponding model is the most confident that this input belongs to its class. The one-vs-rest strategy is the most commonly used strategy and usually a good default choice.

9.3 One-vs-one

Another strategy is to fit a binary classification algorithm for each pair of classes: this strategy is known as *one-vs-one*. The advantage of this strategy is that the classes in each binary classification task are "pure", in the sense that different classes are never merged into a single class. However, the number of binary classification algorithms that needs to be trained is larger for the one-vs-one strategy ($\frac{1}{2}q(q-1)$) than for the one-vs-rest strategy (q). Nonetheless, for the one-vs-one strategy, the number of training samples in each binary classification tasks is smaller than the total number of samples, which makes training each binary classification algorithm usually faster. Another drawback is that this strategy is less interpretable compared to the one-vs-rest strategy, as the predicted class corresponds to the class obtaining the most votes (i.e., winning the most one-vs-one matchups), which does not take into account the confidence in winning each matchup.[1] For instance, winning a one-vs-one matchup with 0.99 probability gives the same result as winning the same matchup with 0.51 probability, i.e. one vote.

9.4 Error correcting output code

A substantially different strategy, inspired by the theory of error correction code, consists in merging a subset of classes into one class and the other subset into the other class, for each binary classification task. This data is often called the code book and can be represented as a matrix whose rows correspond to the classes and whose columns correspond to the binary classification tasks. The matrix consists only of -1 and $+1$ values that represent the corresponding label for each class and for each binary task.[2] For any input, each binary classifier returns the score (or probability) associated with the positive class. The predicted class for this input is the class whose corresponding vector is the most similar to the vector of scores, with similarity being assessed with the Euclidean

[1] The confidences are actually taken into account but only in the event of a tie.

[2] The values are 0 and 1 when the classification algorithm does not return scores but only probabilities.

distance (the lower, the more similar). There exist advanced strategies to define the code book, but it has been argued than a random code book usually gives as good results as a sophisticated one [16].

10. Decision functions with normal distributions

Normal distributions are popular distributions because they are commonly found in nature. For instance, the distribution of heights and birth weights of human beings can be approximated using normal distributions. Moreover, normal distributions are particularly easy to work with from a mathematical point of view. For these reasons, a common model consists in assuming that the training input vectors are independently sampled from normal distributions.

A possible classification model consists in assuming that, for each class, all the corresponding inputs are sampled from a normal distribution with mean vector $\boldsymbol{\mu}_k$ and covariance matrix $\boldsymbol{\Sigma}_k$:

$$\forall i \text{ such that } y^{(i)} = \mathcal{C}_k,\ \boldsymbol{x}^{(i)} \sim \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

Using the probability density function of a normal distribution, one can compute the probability density of any input $\boldsymbol{x}$ associated to the distribution $\mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ of class $\mathcal{C}_k$:

$$p_{\mathbf{x}|\mathrm{y}=\mathcal{C}_k}(\boldsymbol{x}) = \frac{1}{\sqrt{(2\pi)^p |\boldsymbol{\Sigma}_k|}} \exp\left(-\frac{1}{2}[\boldsymbol{x} - \boldsymbol{\mu}_k]^\top \boldsymbol{\Sigma}_k^{-1} [\boldsymbol{x} - \boldsymbol{\mu}_k]\right)$$

With such a probabilistic model, it is easy to compute the probability that a sample belongs to class $\mathcal{C}_k$ using Bayes rule:

$$P(\mathrm{y} = \mathcal{C}_k | \mathbf{x} = \boldsymbol{x}) = \frac{p_{\mathbf{x}|\mathrm{y}=\mathcal{C}_k}(\boldsymbol{x}) P(\mathrm{y} = \mathcal{C}_k)}{p_{\mathbf{x}}(\boldsymbol{x})}$$

With normal distributions, it is mathematically easier to work with log-

probabilities:

$$
\begin{aligned}
&\log P(\mathrm{y} = \mathcal{C}_k | \mathbf{x} = \boldsymbol{x}) \\
&= \log p_{\mathbf{x}|\mathrm{y}=\mathcal{C}_k}(\boldsymbol{x}) + \log P(\mathrm{y} = \mathcal{C}_k) - \log p_{\mathbf{x}}(\boldsymbol{x}) \\
&= -\frac{1}{2}[\boldsymbol{x} - \boldsymbol{\mu}_k]^\top \boldsymbol{\Sigma}_k^{-1}[\boldsymbol{x} - \boldsymbol{\mu}_k] - \frac{1}{2}\log|\boldsymbol{\Sigma}_k| + \log P(\mathrm{y} = \mathcal{C}_k) \\
&\quad - \frac{p}{2}\log(2\pi) - \log p_{\mathbf{x}}(\boldsymbol{x}) \\
&= -\frac{1}{2}\boldsymbol{x}^\top \boldsymbol{\Sigma}_k^{-1}\boldsymbol{x} + \boldsymbol{x}^\top \boldsymbol{\Sigma}_k^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\boldsymbol{\mu}_k^\top \boldsymbol{\Sigma}_k^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\log|\boldsymbol{\Sigma}_k| + \log P(\mathrm{y} = \mathcal{C}_k) \\
&\quad - \frac{p}{2}\log(2\pi) - \log p_{\mathbf{x}}(\boldsymbol{x})
\end{aligned}
\tag{4}
$$

It is also possible to make further assumptions on the covariance matrices that lead to different models. In this section, we present the most commonly used ones: Naive Bayes, linear discriminant analysis and quadratic discriminant analysis. Figure 16 illustrates the covariance matrices and the decision functions for these models in the two-dimensional case.

10.1 Naive Bayes

The Naive Bayes model assumes that, conditionally to each class $\mathcal{C}_k$, the features are independent and have the same variance σ_k^2:

$$\forall k,\ \boldsymbol{\Sigma}_k = \sigma_k^2 \boldsymbol{I}_p$$

Equation 4 can thus be further simplified:

$$
\begin{aligned}
&\log P(\mathrm{y} = \mathcal{C}_k | \mathbf{x} = \boldsymbol{x}) \\
&= -\frac{1}{2\sigma_k^2}\boldsymbol{x}^\top \boldsymbol{x} + \frac{1}{\sigma_k^2}\boldsymbol{x}^\top \boldsymbol{\mu}_k - \frac{1}{2\sigma_k^2}\boldsymbol{\mu}_k^\top \boldsymbol{\mu}_k - \log \sigma_k + \log P(\mathrm{y} = \mathcal{C}_k) \\
&\quad - \frac{p}{2}\log(2\pi) - \log p_{\mathbf{x}}(\boldsymbol{x}) \\
&= \boldsymbol{x}^\top \boldsymbol{W}_k \boldsymbol{x} + \boldsymbol{x}^\top \boldsymbol{w}_k + w_{0k} + s
\end{aligned}
$$

where:

- $\boldsymbol{W}_k = -\frac{1}{2\sigma_k^2}\boldsymbol{I}_p$ is the matrix of the quadratic term for class $\mathcal{C}_k$,
- $\boldsymbol{w}_k = \frac{1}{\sigma_k^2}\boldsymbol{\mu}_k$ is the vector of the linear term for class $\mathcal{C}_k$,
- $w_{0k} = -\frac{1}{2\sigma_k^2}\boldsymbol{\mu}_k^\top \boldsymbol{\mu}_k - \log \sigma_k + \log P(\mathrm{y} = \mathcal{C}_k)$ is the intercept for class $\mathcal{C}_k$, and

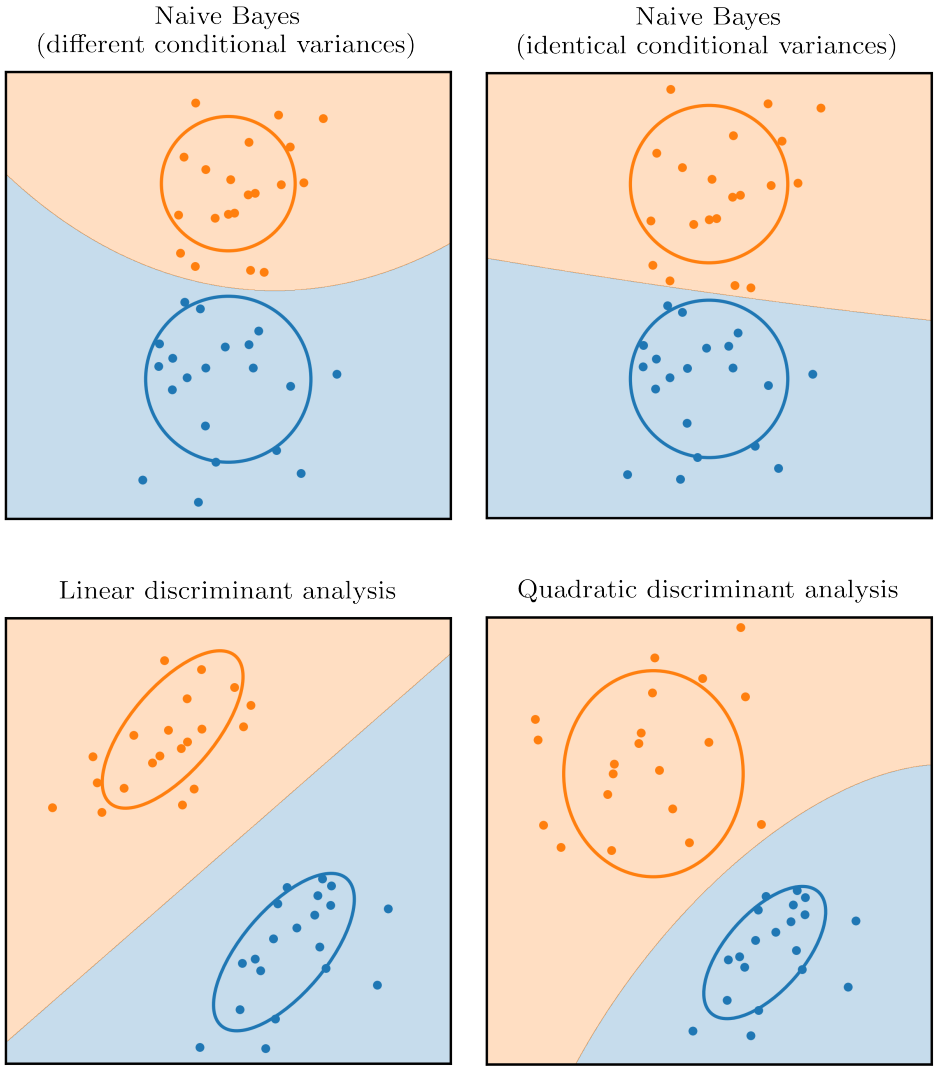

Figure 16: *Illustration of decision functions with normal distributions. A two-dimensional covariance matrix can be represented as an ellipse. In the Naive Bayes model, the features are assumed to be independent and to have the same variance conditionally to the class, leading to covariance matrices being represented as circles. When the covariance matrices are assumed to be identical, the decision functions are linear instead of quadratic.*

- $s = -\frac{p}{2}\log(2\pi) - \log p_{\mathbf{x}}(\boldsymbol{x})$ is a term that does not depend on class $\mathcal{C}_k$.

Therefore, Naive Bayes is a quadratic model. The probabilities for input $\boldsymbol{x}$ to belong to each class $\mathcal{C}_k$ can then easily be computed:

$$P(\mathrm{y} = \mathcal{C}_k|\mathbf{x} = \boldsymbol{x}) = \frac{\exp\left(\boldsymbol{x}^\top \boldsymbol{W}_k \boldsymbol{x} + \boldsymbol{x}^\top \boldsymbol{w}_k + w_{0k}\right)}{\sum_{j=1}^{k} \exp\left(\boldsymbol{x}^\top \boldsymbol{W}_j \boldsymbol{x} + \boldsymbol{x}^\top \boldsymbol{w}_j + w_{0j}\right)}$$

With the Naive Bayes model, it is relatively common to have the conditional variances σ_k^2 to all be equal:

$$\forall k, \boldsymbol{\Sigma}_k = \sigma_k^2 \boldsymbol{I}_p = \sigma^2 \boldsymbol{I}_p$$

In this case, Equation 4 can be even further simplified:

$$\begin{aligned}
&\log P(\mathrm{y} = \mathcal{C}_k|\mathbf{x} = \boldsymbol{x}) \\
&= -\frac{1}{2\sigma^2}\boldsymbol{x}^\top \boldsymbol{x} + \frac{1}{\sigma^2}\boldsymbol{x}^\top \boldsymbol{\mu}_k - \frac{1}{2\sigma^2}\boldsymbol{\mu}_k^\top \boldsymbol{\mu}_k - \log \sigma_k + \log P(\mathrm{y} = \mathcal{C}_k) \\
&\quad - \frac{p}{2}\log(2\pi) - \log p_{\mathbf{x}}(\boldsymbol{x}) \\
&= \boldsymbol{x}^\top \boldsymbol{w}_k + w_{0k} + s
\end{aligned}$$

where:

- $\boldsymbol{w}_k = \frac{1}{\sigma^2}\boldsymbol{\mu}_k$ is the vector of the linear term for class $\mathcal{C}_k$,
- $w_{0k} = -\frac{1}{2\sigma^2}\boldsymbol{\mu}_k^\top \boldsymbol{\mu}_k + \log P(\mathrm{y} = \mathcal{C}_k)$ is the intercept for class $\mathcal{C}_k$, and
- $s = -\frac{1}{2\sigma^2}\boldsymbol{x}^\top \boldsymbol{x} - \log \sigma - \frac{p}{2}\log(2\pi) - \log p_{\mathbf{x}}(\boldsymbol{x})$ is a term that does not depend on class $\mathcal{C}_k$.

In this case, Naive Bayes becomes a linear model.

10.2 Linear discriminant analysis

Linear discriminant analysis (LDA) makes the assumption that all the covariance matrices are identical but otherwise arbitrary:

$$\forall k, \boldsymbol{\Sigma}_k = \boldsymbol{\Sigma}$$

Therefore, Equation 4 can be further simplified:

$$\begin{aligned}
&\log P(\mathrm{y} = \mathcal{C}_k|\mathbf{x} = \boldsymbol{x}) \\
&= -\frac{1}{2}[\boldsymbol{x} - \boldsymbol{\mu}_k]^\top \boldsymbol{\Sigma}^{-1}[\boldsymbol{x} - \boldsymbol{\mu}_k] - \frac{1}{2}\log|\boldsymbol{\Sigma}| + \log P(\mathrm{y} = \mathcal{C}_k)
\end{aligned}$$

$$
\begin{aligned}
&\quad - \frac{p}{2}\log(2\pi) - \log p_{\mathbf{x}}(\boldsymbol{x}) \\
&= -\frac{1}{2}\left(\boldsymbol{x}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{x} - \boldsymbol{x}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k - \boldsymbol{\mu}_k^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{x} + \boldsymbol{\mu}_k^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k\right) \\
&\quad - \frac{1}{2}\log|\boldsymbol{\Sigma}| + \log P(\mathrm{y} = \mathcal{C}_k) - \frac{p}{2}\log(2\pi) - \log p_{\mathbf{x}}(\boldsymbol{x}) \\
&= -\boldsymbol{x}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\boldsymbol{x}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{x} - \frac{1}{2}\boldsymbol{\mu}_k^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k + \log P(\mathrm{y} = \mathcal{C}_k) - \frac{1}{2}\log|\boldsymbol{\Sigma}| \\
&\quad - \frac{p}{2}\log(2\pi) - \log p_{\mathbf{x}}(\boldsymbol{x}) \\
&= \boldsymbol{x}^\top\boldsymbol{w}_k + w_{0k} + s
\end{aligned}
$$

where:

- $\boldsymbol{w}_k = \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k$ is the vector of coefficients for class $\mathcal{C}_k$,
- $w_{0k} = -\frac{1}{2}\boldsymbol{\mu}_k^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k + \log P(\mathrm{y} = \mathcal{C}_k)$ is the intercept for class $\mathcal{C}_k$, and
- $s = -\frac{1}{2}\boldsymbol{x}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{x} - -\frac{1}{2}\log|\boldsymbol{\Sigma}| - \frac{p}{2}\log(2\pi) - \log p_{\mathbf{x}}(\boldsymbol{x})$ is a term that does not depend on class $\mathcal{C}_k$.

Therefore, linear discriminant analysis is a linear model. When $\boldsymbol{\Sigma}$ is diagonal, linear discriminant analysis is identical to Naive Bayes with identical conditional variances.

The probabilities for input $\boldsymbol{x}$ to belong to each class $\mathcal{C}_k$ can then easily be computed:

$$
P(\mathrm{y} = \mathcal{C}_k|\mathbf{x} = \boldsymbol{x}) = \frac{\exp\left(\boldsymbol{x}^\top\boldsymbol{w}_k + w_{0k}\right)}{\sum_{j=1}^{k}\exp\left(\boldsymbol{x}^\top\boldsymbol{w}_j + w_{0j}\right)}
$$

10.3 Quadratic discriminant analysis

Quadratic discriminant analysis makes no assumption on the covariance matrices $\boldsymbol{\Sigma}_k$ that can all be arbitrary. Equation 4 can be written as:

$$
\begin{aligned}
&\log P(\mathrm{y} = \mathcal{C}_k|\mathbf{x} = \boldsymbol{x}) \\
&= -\frac{1}{2}\boldsymbol{x}^\top\boldsymbol{\Sigma}_k^{-1}\boldsymbol{x} + \boldsymbol{x}^\top\boldsymbol{\Sigma}_k^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\boldsymbol{\mu}_k^\top\boldsymbol{\Sigma}_k^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\log|\boldsymbol{\Sigma}_k| + \log P(\mathrm{y} = \mathcal{C}_k) \\
&\quad - \frac{p}{2}\log(2\pi) - \log p_{\mathbf{x}}(\boldsymbol{x}) \\
&= \boldsymbol{x}^\top\boldsymbol{W}_k\boldsymbol{x} + \boldsymbol{x}^\top\boldsymbol{w}_k + w_{0k} + s
\end{aligned}
$$

where:

- $\boldsymbol{W}_k = -\frac{1}{2}\boldsymbol{\Sigma}_k^{-1}$ is the matrix of the quadratic term for class $\mathcal{C}_k$,

- $\boldsymbol{w}_k = \boldsymbol{\Sigma}_k^{-1}\boldsymbol{\mu}_k$ is the vector of the linear term for class $\mathcal{C}_k$,
- $w_{0k} = -\frac{1}{2}\boldsymbol{\mu}_k^\top \boldsymbol{\Sigma}_k^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\log|\boldsymbol{\Sigma}_k| + \log P(\mathrm{y} = \mathcal{C}_k)$ is the intercept for class $\mathcal{C}_k$, and
- $s = -\frac{p}{2}\log(2\pi) - \log p_{\mathbf{x}}(\boldsymbol{x})$ is a term that does not depend on class $\mathcal{C}_k$.

Therefore, quadratic discriminant analysis is a quadratic model.

The probabilities for input $\boldsymbol{x}$ to belong to each class $\mathcal{C}_k$ can then easily be computed:

$$P(\mathrm{y} = \mathcal{C}_k|\mathbf{x} = \boldsymbol{x}) = \frac{\exp\left(\boldsymbol{x}^\top \boldsymbol{W}_k \boldsymbol{x} + \boldsymbol{x}^\top \boldsymbol{w}_k + w_{0k}\right)}{\sum_{j=1}^{k} \exp\left(\boldsymbol{x}^\top \boldsymbol{W}_j \boldsymbol{x} + \boldsymbol{x}^\top \boldsymbol{w}_j + w_{0j}\right)}$$

11. Tree-based algorithms

11.1 Decision tree

Binary decisions based on conditional statements are frequently used in everyday life because they are intuitive and easy to understand. Figure 17 illustrates a general approach when someone is ill. Depending on conditional statements (severity of symptoms, ability to quickly consult a specialist), the decision (consult your general practitioner or a specialist, or call for emergency services) is different. Models with such an architecture are often used in machine learning and are called *decision trees*.

A decision tree is an algorithm containing only conditional statements and can be represented with a tree [17]. This graph consists of:

- decision nodes for all the conditional statements,
- branches for the potential outcomes of each decision node, and
- leaf nodes for the final decision.

Figure 18 illustrates a decision tree and its corresponding decision function. For a given sample, the final decision is obtained by following its corresponding path, starting at the root node.

A decision tree recursively partitions the feature space in order to group samples with the same labels or similar target values. At each node, the objective is to find the best (feature, threshold) pair so that both subsets obtained with this split are the most *pure*, that is homogeneous. To do so, the best (feature, threshold) pair is defined as the pair that minimizes an *impurity* criterion.

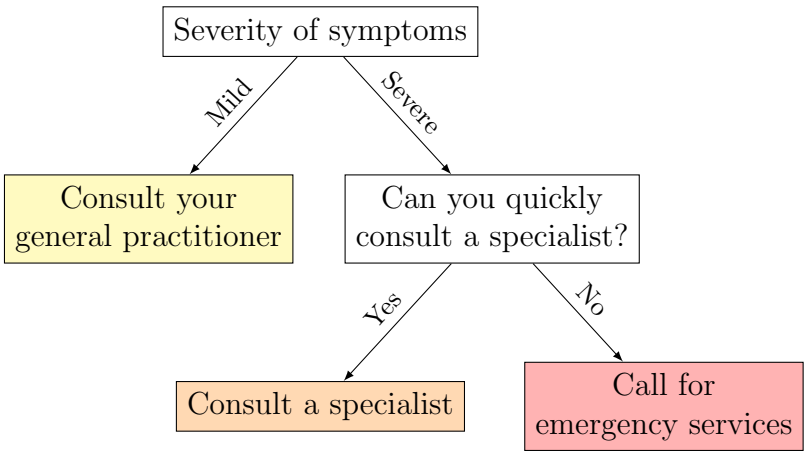

Figure 17: *A general thought process when being ill. Depending on conditional statements (severity of symptoms, ability to quickly consult a specialist), the decision (consult your general practitioner or a specialist, or call for emergency services) is different.*

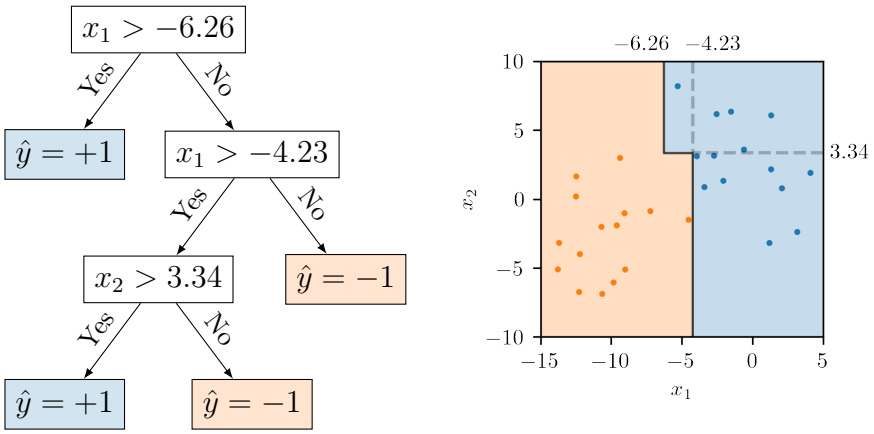

Figure 18: *A decision tree: (left) the rules learned by the decision tree, and (right) the corresponding decision function.*

Let $\mathcal{S} \subseteq \mathcal{X}$ be a subset of training samples. For classification tasks, the distribution of the classes, that is the proportion of each class, is used to measure the purity of the subset. Let p_k be the proportion of samples from class $\mathcal{C}_k$ in a given partition:

$$p_k = \frac{1}{|\mathcal{S}|} \sum_{y \in \mathcal{S}} \mathbf{1}_{y=\mathcal{C}_k}$$

Popular impurity criterion for classification tasks include:

- Gini index: $\sum_k p_k(1 - p_k)$
- Entropy: $-\sum_k p_k \log(p_k)$
- Misclassification: $1 - \max_k p_k$

Figure 19 illustrates the values of the Gini index and the entropy for a single class $\mathcal{C}_k$ and for different proportions of samples p_k. One can see that the entropy function takes larger values than the Gini index, especially for $p_k < 0.8$. Since the sum of the proportions is equal to 1, most classes only represent a small proportion of the samples. Therefore, a simple interpretation is that entropy is more discriminative against heterogeneous subsets than the Gini index. Misclassification only takes into account the proportion of the most common class and tends to be even less discriminative against heterogeneous subsets than both entropy and Gini index.

For regression tasks, the mean error from a reference value (such as the mean or the median) is often used as the impurity criterion:

- Mean squared error: $\frac{1}{|\mathcal{S}|} \sum_{y \in \mathcal{S}} (y - \bar{y})^2 \quad \text{with} \quad \bar{y} = \frac{1}{|\mathcal{S}|} \sum_{y \in \mathcal{S}} y$
- Mean absolute error: $\frac{1}{|\mathcal{S}|} \sum_{y \in \mathcal{S}} |y - \text{median}_{\mathcal{S}}(y)|$

Theoretically, a tree can grow until every leaf node is perfectly pure. However, such a tree would have a lot of branches and would be very complex, making it prone to overfitting. Several strategies are commonly used to limit the size of the tree. One approach consists in growing the tree with no restriction, then *pruning* the tree, that is replacing subtrees with nodes [17]. Other popular strategies to limit the complexity of the tree are usually applied while the tree is grown, and include setting:

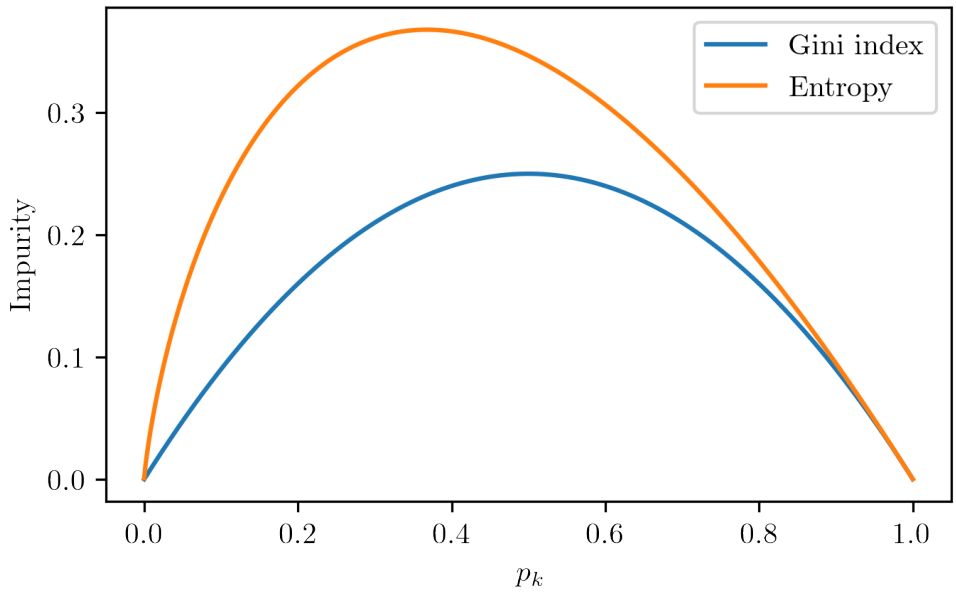

Figure 19: *Illustration of Gini index and entropy. The entropy function takes larger values than the Gini index, especially for $p_k < 0.8$, thus is more discriminative against heterogeneous subsets (when most classes only represent only a small proportion of the samples) than Gini index.*

- a maximum depth for the tree,
- a minimum number of samples required to be at an internal node,
- a minimum number of samples required to split a given partition,
- a maximum number of leaf nodes,
- a maximum number of features considered (instead of all the features) to find the best split,
- a minimum impurity decrease to split an internal node.

11.2 Random forest

One limitation of decision trees is their simplicity. Decision trees tend to use a small fraction of the features in their decision function. In order to use more features in the decision tree, growing a larger tree is required, but large trees tend to suffer from overfitting, that is having a low bias but a high variance. One solution to decrease the variance without much increasing the bias is to build an ensemble of trees with randomness, hence the name *random forest* [18].

Box 5: Random forest

- **Random forest**: ensemble of decision trees with randomness introduced to build different trees.
- **Decision tree**: algorithm containing only conditional statements and represented with a tree.
- **Regularization**: maximum depth for each tree, minimum number of samples required to split a given partition, etc.

In a bid to have trees that are not perfectly correlated (thus building actually different trees), each tree is built using only a subset of the training samples obtained with random sampling. Moreover, for each decision node of each tree, only a subset of the features are considered to find the best split.

The final prediction is obtained by averaging the predictions of each tree. For classification tasks, the predicted class is either the most commonly predicted class (hard-voting) or the one with the highest mean probability estimate (soft-voting) across the trees. For regression tasks, the predicted value is usually the mean of the predicted values across the trees.

11.3 Extremely randomized trees

Even though random forests involve randomness in sampling both the samples and the features, trees inside a random forest tend to be correlated, thus limiting the variance decrease. In order to decrease even more the variance of the model (while possibly increasing its bias) by growing less correlated trees, *extremely randomized trees* introduce more randomness [19]. Instead of looking for the best split among all the candidate (feature, threshold) pairs, one threshold is drawn at random for each candidate feature and the best of these randomly generated thresholds is chosen as the splitting rule.

12. Clustering

So far, we have presented classic machine learning algorithms for classification and regression, which are the main components of supervised learning. Each input $\boldsymbol{x}^{(i)}$ had an associated output $y^{(i)}$. In this section

we present clustering, which is an unsupervised machine learning task. In unsupervised learning, only the inputs $\boldsymbol{x}^{(i)}$ are available, with no associated outputs. As the ground truth is not available, the objective is to extract information from the input data without supervising the learning process with the output data.

Clustering consists in finding groups of samples such that:

- samples from the same group are similar, and
- samples from different groups are different.

For instance, clustering can be used to identify disease subtypes for heterogeneous diseases such as Alzheimer's disease and Parkinson's disease.

In this section, we present two of the most common clustering methods: the k-means algorithm and the Gaussian mixture model.

12.1 k-means

The k-means algorithm divides a set of n samples, denoted by $\mathcal{X}$, into a set of k disjoint clusters, each denoted by $\mathcal{X}_j$, such that $\mathcal{X} = \{\mathcal{X}_1, \dots, \mathcal{X}_k\}$.

Figure 20 illustrates the concept of this algorithm. Each cluster $\mathcal{X}_j$ is characterized by its *centroid*, denoted by $\boldsymbol{\mu}_j$, that is the mean of the samples in this cluster:

$$\boldsymbol{\mu}_j = \frac{1}{|\mathcal{X}_j|} \sum_{\boldsymbol{x}^{(i)} \in \mathcal{X}_j} \boldsymbol{x}^{(i)}$$

The centroids fully define the set of clusters because each sample is assigned to the cluster whose centroid is the closest.

The k-means algorithm aims at finding centroids that minimize the *inertia*, also known as *within-cluster sum-of-squares criterion*:

$$\min_{\{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_k\}} \sum_{j=1}^{k} \sum_{\boldsymbol{x}^{(i)} \in \mathcal{X}_j} \|\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_j\|_2^2$$

The original algorithm used to find the centroids is often referred to as the *Lloyd's algorithm* [20] and is presented in Algorithm 1. After initializing the centroids, a two-step loop is repeated until convergence (when the centroids are identical for two consecutive iterations) consisting of:

1. the *assignment step*, where the clusters are updated based on the current centroids, and
2. the *update step*, where the centroids are updated based on the current clusters.

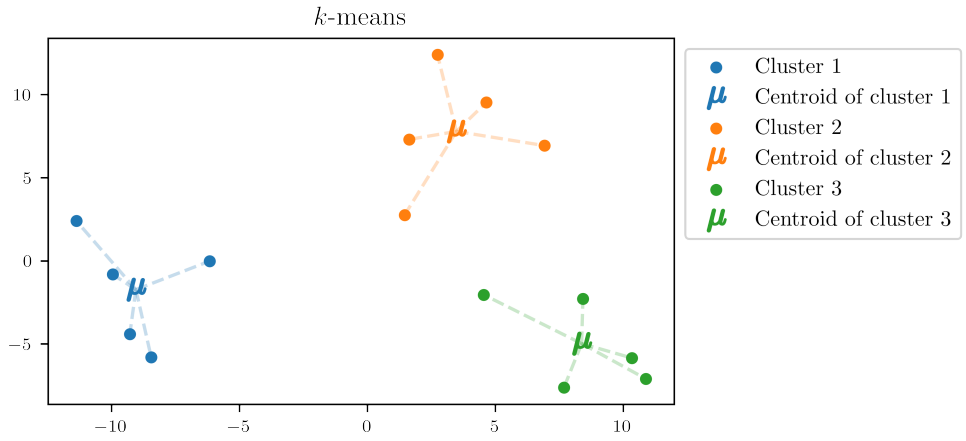

Figure 20: *Illustration of the k-means algorithm. The objective of the algorithm is to find the centroids that minimize the within-cluster sum-of-squares criterion. In this example, the inertia is approximately equal to 184.80 and is the lowest possible inertia, meaning that the represented centroids are optimal.*

When clusters are well-defined, a point from a given cluster is likely to stay in this cluster. Therefore, the assignment step can be sped up thanks to the triangle inequality by keeping track of lower and upper bounds for distances between points and centers, at the cost of higher memory usage [21].

Even though the k-means algorithm is one of the simplest and most used clustering algorithms, it has several downsides that should be kept in mind.

First, the number of clusters k is a hyperparameter. Setting a value much different from the actual number of clusters may yield poor clusters.

Second, the inertia is not a convex function. Although Lloyd's algorithm is guaranteed to converge, it may converge to a local minimum that is not a global minimum. Figure 21 illustrates the convergence to such centroids. Several strategies are often applied to address this issue, including sophisticated centroid initialization [22] and running the algorithm numerous times and keeping the best run (i.e., the one yielding the lowest inertia).

Third, inertia makes the assumption that the clusters are convex and isotropic. The k-means algorithm may yield poor results when this assumption does not hold, such as with elongated clusters or manifolds with irregular shapes.

Fourth, the Euclidean distance tends to be inflated (i.e., the ratio of the distances of the nearest and farthest neighbors to a given target is

Algorithm 1: *Lloyd's algorithm (a.k.a. naive k-means algorithm).*

Result: Centroids $\{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\}$

Initialize the centroids $\{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\}$;

while *not converged* **do**

Assignment step: Compute the clusters (i.e., assign each sample to its nearest centroid):

$$\forall j \in \{1, \ldots, k\},\ \mathcal{X}_j = \{\boldsymbol{x}^{(i)} \in \mathcal{X} \mid \|\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_j\|_2^2 = \min_l \|\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_l\|_2^2\}$$

Update step: Compute the centroids of the updated clusters:

$$\forall j \in \{1, \ldots, k\},\ \boldsymbol{\mu}_j = \frac{1}{|\mathcal{X}_j|} \sum_{\boldsymbol{x}^{(i)} \in \mathcal{X}_j} \boldsymbol{x}^{(i)}$$

close to 1) in high-dimensional spaces, making inertia a poor criterion in such spaces [23]. One can alleviate this issue by running a dimensionality reduction algorithm such as principal component analysis prior to the k-means algorithm.

12.2 Gaussian mixture model

A mixture model makes the assumption that each sample is generated from a mixture of several independent distributions.

Let k be the number of distributions. Each distribution F_j is characterized by its probability of being picked, denoted by π_j, and its density p_j with parameters $\boldsymbol{\theta}_j$, denoted by $p_j(\cdot, \boldsymbol{\theta}_j)$. Let $\boldsymbol{\Delta} = (\Delta_1, \ldots, \Delta_k)$ be a vector-valued random variable such that:

$$\sum_{j=1}^{k} \Delta_j = 1 \quad \text{and} \quad \forall j \in \{1, \ldots, k\},\ P(\Delta_j = 1) = 1 - P(\Delta_j = 0) = \pi_j$$

and $(\mathrm{x}_1, \ldots, \mathrm{x}_k)$ be independent random variables such that $\mathrm{x}_j \sim F_j$. The samples are assumed to be generated from a random variable x with density p_{x} such that:

$$\mathrm{x} = \sum_{j=1}^{k} \Delta_j \mathrm{x}_j$$

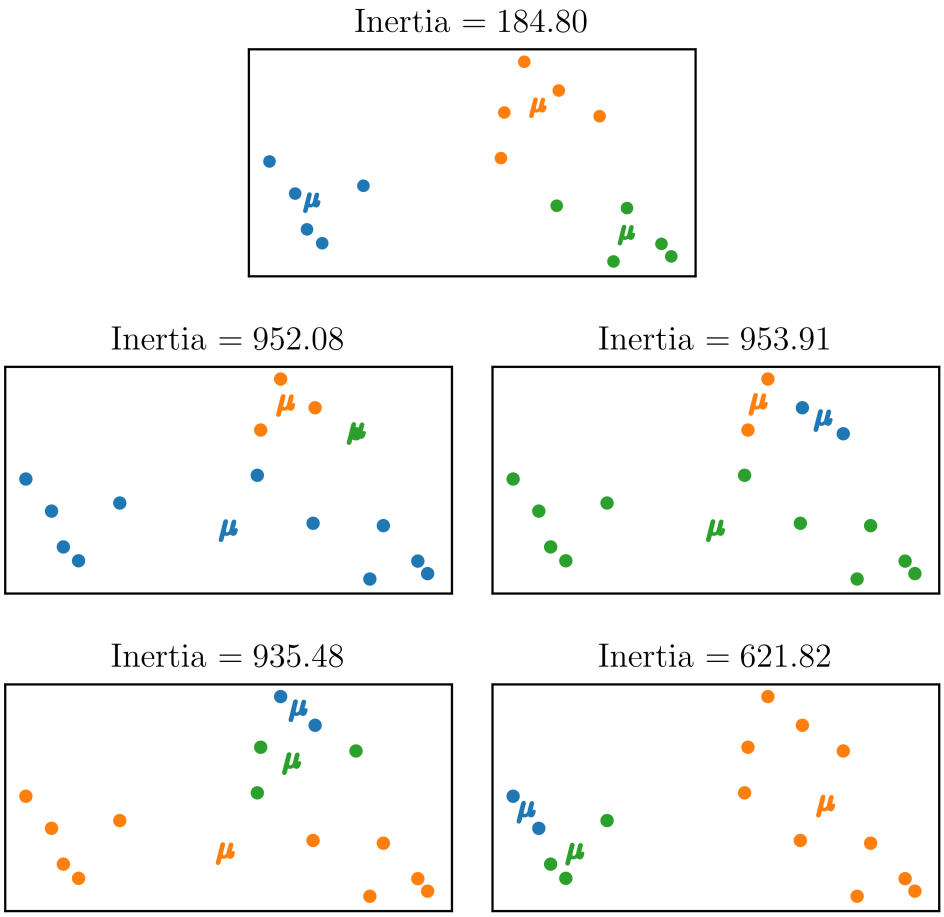

Figure 21: *Illustration of the convergence of the k-means algorithm to bad local minima. In the upper figure, the algorithm converged to a global minimum because the inertia is equal to the minimum possible value (184.80), thus the obtained clusters are optimal. In the four other figures, the algorithm converged to a local minima that are not global minima because the inertias are higher than the minimum possible value, thus the obtained clusters are suboptimal.*

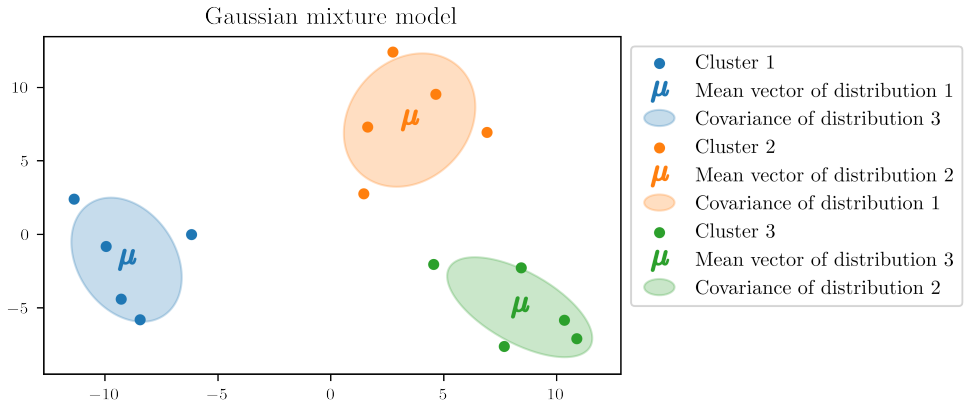

Figure 22: *Gaussian mixture model. For each estimated distribution, the mean vector and the ellipsis consisting of all the points within one standard deviation of the mean are plotted.*

$$\forall \boldsymbol{x} \in \mathcal{X},\ p_{\mathrm{x}}(\boldsymbol{x}, \boldsymbol{\theta}) = \sum_{j=1}^{k} \pi_j p_j(\boldsymbol{x}; \boldsymbol{\theta}_j)$$

A Gaussian mixture model is a particular case of a mixture model in which each distribution F_j is a Gaussian distribution with mean vector $\boldsymbol{\mu}_j$ and covariance matrix $\boldsymbol{\Sigma}_j$:

$$\forall j \in \{1, \ldots, k\},\ F_j = \mathcal{N}(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$$

Figure 22 illustrates the learned distributions from a Gaussian mixture model.

The objective is to find the parameters $\boldsymbol{\theta}$ that maximize the likelihood, with $\boldsymbol{\theta} = \left(\{\boldsymbol{\mu}_j\}_{j=1}^k, \{\boldsymbol{\Sigma}_j\}_{j=1}^k, \{\pi_j\}_{j=1}^k\right)$:

$$L(\boldsymbol{\theta}) = \prod_{i=1}^{n} p_X(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$

For computational reasons, it is easier to maximize the log-likelihood:

$$\log(L(\boldsymbol{\theta})) = \sum_{i=1}^{n} \log(p_X(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})) = \sum_{i=1}^{n} \log\left(\sum_{j=1}^{k} \pi_j p_j(\boldsymbol{x}; \boldsymbol{\theta}_j)\right)$$

Because the density $p_X(\cdot, \boldsymbol{\theta})$ is a weighted sum of Gaussian densities, the expression cannot be further simplified.

In order to solve this maximization problem, an algorithm called *Expectation–Maximization* (EM) is often applied [24]. Algorithm 2 describes the main concepts of this algorithm. After initializing the parameters of each distribution, a two-step loop is repeated until convergence (when the parameters are stable over consecutive loops):

Algorithm 2: *Expectation–Maximization algorithm for Gaussian mixture models.*

Result: Mean vectors $\{\boldsymbol{\mu}_j\}_{j=1}^k$, covariance matrices $\{\boldsymbol{\Sigma}_j\}_{j=1}^k$ and probabilities $\{\pi_j\}_{j=1}^k$

Initialize the mean vectors $\{\boldsymbol{\mu}_j\}_{j=1}^k$, covariance matrices $\{\boldsymbol{\Sigma}_j\}_{j=1}^k$ and probabilities $\{\pi_j\}_{j=1}^k$;

while *not converged* **do**

E-step: Compute the posterior probability $\gamma_i(j)$ for each sample $\boldsymbol{x}^{(i)}$ to have been generated from distribution F_j:

$$\forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, k\}, \ \gamma_i(j) = \frac{\pi_j p_j(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}_j, \boldsymbol{\Sigma}_j)}{\sum_{l=1}^k \pi_l p_j(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}_l, \boldsymbol{\Sigma}_l)}$$

M-step: Update the parameters of each distribution F_j:

$$\forall j \in \{1, \dots, k\}, \ \boldsymbol{\mu}_j = \frac{\sum_{i=1}^n \gamma_i(j) \boldsymbol{x}^{(i)}}{\sum_{i=1}^n \gamma_i(j)}$$

$$\forall j \in \{1, \dots, k\}, \ \boldsymbol{\Sigma}_j = \frac{\sum_{i=1}^n \gamma_i(j) [\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_j][\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_j]^\top}{\sum_{i=1}^n \gamma_i(j)}$$

$$\forall j \in \{1, \dots, k\}, \ \pi_j = \frac{1}{n} \sum_{i=1}^n \gamma_i(j)$$

- the *expectation step*, in which the probability for each sample $\boldsymbol{x}^{(i)}$ to have been generated from distribution F_j is computed, and
- the *maximization step*, in which the probability and the parameters of each distribution are updated.

Because it is impossible to know which samples have been generated by each distribution, it is also impossible to directly maximize the log-likelihood, which is why we compute its *expected value* using the posterior probabilities, hence the name *expectation step*. The second step simply consists in maximizing the expected log-likelihood, hence the name *maximization step*.

Lloyd's and EM algorithms have a lot of similarities. In the first step, the assignment step assigns each sample to its closest cluster, whereas the expectation step computes the probability for each sample to have been generated from each distribution. In the second step, the update

step computes the centroid of each cluster as the mean of the samples in a given cluster, while the maximization step updates the probability and the parameters of each distribution as a weighted average over all the samples. For these reasons, the k-means algorithm is often referred to as a *hard-voting* clustering algorithm, as opposed to the Gaussian mixture model which is referred to as a *soft-voting* clustering algorithm.

The Gaussian mixture model has several advantages over the k-means algorithm.

First, the use of normal distribution densities instead of Euclidean distances dwindles the inflation issue in high-dimensional spaces. Second, the Gaussian mixture model includes covariance matrices, allowing for clusters with elliptical shapes, while the k-means algorithm only include centroids, forcing clusters to have circular shapes.

Nonetheless, the Gaussian mixture model also has several drawbacks, sharing a few with the k-means algorithm.

First, the number of distributions k is a hyperparameter. Setting a value much different from the actual number of clusters may yield poor clusters. Second, the log-likelihood is not a concave function. Like Lloyd's algorithm, the EM algorithm is guaranteed to converge but it may converge to a local maximum that is not a global maximum. Several strategies are often applied to address this issue, including sophisticated centroid initialization [22] and running the algorithm numerous times and keeping the best run (i.e., the one yielding the highest log-likelihood). Third, the Gaussian mixture model has more parameters than the k-means algorithm. Therefore, it usually requires more samples to accurately estimate its parameters (in particular the covariance matrices) than the k-means algorithm.

13. Dimensionality reduction

Dimensionality reduction consists in finding a good mapping from the input space into a space of lower dimension. Dimensionality reduction can either be unsupervised or supervised.

13.1 Principal component analysis

For exploratory data analysis, it may be interesting to investigate the variances of the p features and the $\frac{1}{2}p(p-1)$ covariances or correlations. However, as the value of p increases, this process becomes growingly tedious. Moreover, each feature may explain a small proportion of the total variance. It may be more desirable to have another representation of the data where a small number of features explain most of the total

variance, in other words to have a coordinate system adapted to the input data.

Principal component analysis (PCA) consists in finding a representation of the data through *principal components* [25]. The principal components are a sequence of unit vectors such that the i-th vector is the best approximation of the data (i.e., maximizing the explained variance) while being orthogonal to the first $i-1$ vectors.

Figure 23 illustrates principal component analysis when the input space is two-dimensional. On the upper figure, the training data in the original space is plotted. Both features explain about the same amount of the total variance, although one can clearly see that both features are strongly correlated. Principal component analysis identifies a new Cartesian coordinate system based on the input data. On the lower figure, the training data in the new coordinate system is plotted. The first dimension explains much more variance than the second dimension.

13.1.1. Full decomposition

Mathematically, given an input matrix $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ that is centered (i.e., the mean value of each column $\boldsymbol{X}_{:,j}$ is equal to zero), the objective is to find a matrix $\boldsymbol{W} \in \mathbb{R}^{p \times p}$ such that:

- $\boldsymbol{W}$ is an orthogonal matrix, i.e. its columns are unit vectors and orthogonal to each other,
- the new representation of the input data, denoted by $\boldsymbol{T}$, consists of the coordinates in the Cartesian coordinate system induced by $\boldsymbol{W}$ (whose columns form an orthogonal basis of $\mathbb{R}^p$ with the Euclidean dot product):
$$\boldsymbol{T} = \boldsymbol{X}\boldsymbol{W}$$
- each column of $\boldsymbol{W}$ maximizes the explained variance.

Each column $\boldsymbol{w}_i = \boldsymbol{W}_{:,i}$ is a principal component. Each input vector $\boldsymbol{x}$ is transformed into another vector $\boldsymbol{t}$ using a linear combination of each feature with the weights from the $\boldsymbol{W}$ matrix:

$$\boldsymbol{t} = \boldsymbol{x}^\top \boldsymbol{W}$$

The first principal component $\boldsymbol{w}^{(1)}$ is the unit vector that maximizes the explained variance:

$$\boldsymbol{w}_1 = \underset{\|\boldsymbol{w}\|=1}{\arg\max} \left\{ \sum_{i=1}^{n} \boldsymbol{x}^{(i)\top} \boldsymbol{w} \| \right\}$$

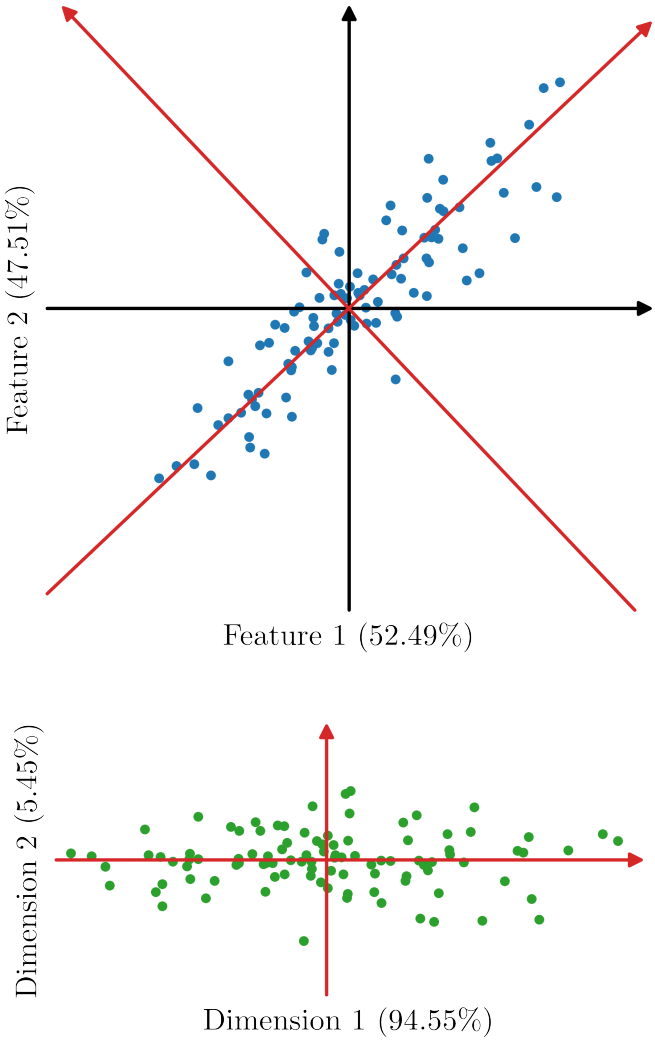

Figure 23: *Illustration of principal component analysis. On the upper figure, the training data in the original space (blue points with black axes) is plotted. Both features explain about the same amount of the total variance, although one can clearly see a linear pattern. The principal component analysis algorithm learns a new Cartesian coordinate system based on the input data (red axes). On the lower figure, the training data in the new coordinate system is plotted (green points with red axes). The first dimension explains much more variance than the second dimension.*

$$
\begin{aligned}
&= \arg\max_{\|\boldsymbol{w}\|=1} \{\|\boldsymbol{X}\boldsymbol{w}\|\} \\
&= \arg\max_{\|\boldsymbol{w}\|=1} \left\{\boldsymbol{w}^\top \boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{w}\|\right\} \\
\boldsymbol{w}_1 &= \arg\max_{\boldsymbol{w}\in\mathbb{R}^p} \left\{ \frac{\boldsymbol{w}^\top \boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{w}}{\boldsymbol{w}^\top \boldsymbol{w}} \right\}
\end{aligned}
$$

As $\boldsymbol{X}^\top \boldsymbol{X}$ is a positive semi-definite matrix, a well known result from linear algebra is that $\boldsymbol{w}^{(1)}$ is the eigenvector associated with the largest eigenvalue of $\boldsymbol{X}^\top \boldsymbol{X}$.

The k-th component is found by subtracting the first $k-1$ principal components from $\boldsymbol{X}$:

$$
\hat{\boldsymbol{X}}_k = \boldsymbol{X} - \sum_{s=1}^{k-1} \boldsymbol{X}\boldsymbol{w}^{(s)}\boldsymbol{w}^{(s)\top}
$$

and then finding the unit vector that explains the maximum variance from this new data matrix:

$$
\boldsymbol{w}_k = \arg\max_{\|\boldsymbol{w}\|=1} \left\{ \|\hat{\boldsymbol{X}}_k \boldsymbol{w}\| \right\} = \arg\max_{\boldsymbol{w}\in\mathbb{R}^p} \left\{ \frac{\boldsymbol{w}^\top \hat{\boldsymbol{X}}_k^\top \hat{\boldsymbol{X}}_k \boldsymbol{w}}{\boldsymbol{w}^\top \boldsymbol{w}} \right\}
$$

One can show that the eigenvector associated with the k-th largest eigenvalue of the $\boldsymbol{X}^\top \boldsymbol{X}$ matrix maximizes the quantity to be maximized.

Therefore, the matrix $\boldsymbol{W}$ is the matrix whose columns are the eigenvectors of the $\boldsymbol{X}^\top \boldsymbol{X}$ matrix, sorted by descending order of their associated eigenvalues.

13.1.2. Truncated decomposition

Since each principal component iteratively maximizes the remaining variance, the first principal components explain most of the total variance, while the last ones explain a tiny proportion of the total variance. Therefore, keeping only a subset of the ordered principal components usually gives a good representation of the input data.

Mathematically, given a number of dimensions l, the new representation is obtained by truncating the matrix of principal components $\boldsymbol{W}$ to only keep the first l columns, resulting in the submatrix $\boldsymbol{W}_{:,:l}$:

$$
\tilde{\boldsymbol{T}} = \boldsymbol{X}\boldsymbol{W}_{:,:l}
$$

Figure 24 illustrates the use of principal component analysis as dimensionality reduction. The Iris flower dataset consists of 50 samples for each of three iris species (setosa, versicolor and virginica) for which four features were measured: the length and the width of the sepals and petals, in centimeters. The projection of each sample on the first two principal components is shown in this figure.

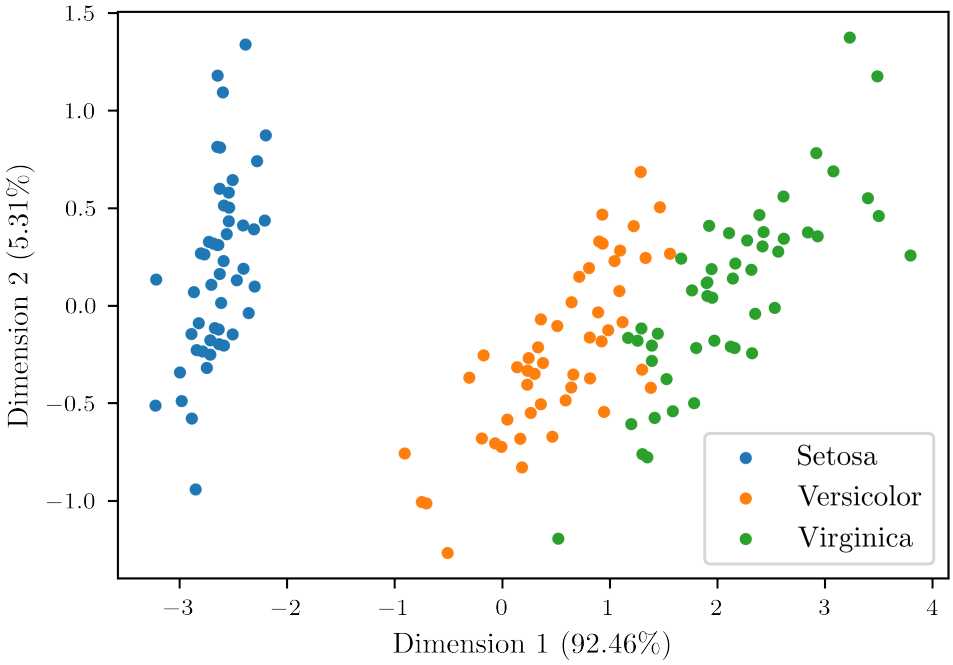

Figure 24: *Illustration of principal component analysis as a dimensionality reduction technique. The Iris flower dataset consists of 50 samples for each of three iris species (setosa, versicolor and virginica) for which four features were measured: the length and the width of the sepals and petals, in centimeters. The projection of each sample on the first two principal components is shown in this figure. The first dimension explains most of the variance (92.46%).*

13.2 Linear discriminant analysis

In Section 10, we introduced linear discriminant analysis (LDA) as a classification method. However, it can also be used as a supervised dimensionality reduction method. LDA fits a multivariate normal distribution for each class $\mathcal{C}_k$, so that each class is characterized by its mean vector $\boldsymbol{\mu}_k \in \mathbb{R}^p$ and has the same covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{p \times p}$. However, a set of k points lies in a space of dimension at most $k-1$. For instance, a set of 2 points lies on a line, while a set of 3 points lies on a plane. Therefore, the subspace induced by the k mean vectors $\boldsymbol{\mu}_k$ can be used as dimensionality reduction.

There exists another formulation of linear discriminant analysis which is equivalent and more intuitive for dimensionality reduction. Linear discriminant analysis aims to find a linear projection so that the classes are separated as much as possible (i.e., projections of samples from a same class are close to each other, while projections of samples from different classes are far from each other).

Mathematically, the objective is to find the matrix $\boldsymbol{W} \in \mathbb{R}^{p \times l}$ (with $l \leq k-1$) that maximizes the between-class scatter while also minimizing the within-class scatter:

$$\max_{\boldsymbol{W}} \operatorname{tr}\left(\left(\boldsymbol{W}^\top \boldsymbol{S}_w \boldsymbol{W}\right)^{-1}\left(\boldsymbol{W}^\top \boldsymbol{S}_b \boldsymbol{W}\right)\right)$$

The within-class scatter matrix $\boldsymbol{S}_w$ summarizes the diffusion between the mean vector $\boldsymbol{\mu}_k$ of class $\mathcal{C}_k$ and all the inputs $\boldsymbol{x}^{(i)}$ belonging to class $\mathcal{C}_k$, over all the classes:

$$\boldsymbol{S}_w = \sum_{k=1}^{q} \sum_{y^{(i)}=\mathcal{C}_k} [\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_k][\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_k]^\top$$

The between-class scatter matrix $\boldsymbol{S}_b$ summarizes the diffusion between all the mean vectors:

$$\boldsymbol{S}_b = \sum_{k=1}^{q} n_k [\boldsymbol{\mu}_k - \boldsymbol{\mu}][\boldsymbol{\mu}_k - \boldsymbol{\mu}]^\top$$

where n_k is the proportion of samples belonging to class $\mathcal{C}_k$ and $\boldsymbol{\mu} = \sum_{k=1}^{q} n_k \boldsymbol{\mu}_k = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}^{(i)}$ is the mean vector over all the input vectors.

One can show that the $\boldsymbol{W}$ matrix consists of the first l eigenvectors of the matrix $\boldsymbol{S}_w^{-1}\boldsymbol{S}_b$ with the corresponding eigenvalues being sorted in descending order. Just as in principal component analysis, the corresponding eigenvalues can be used to determine the contribution of each

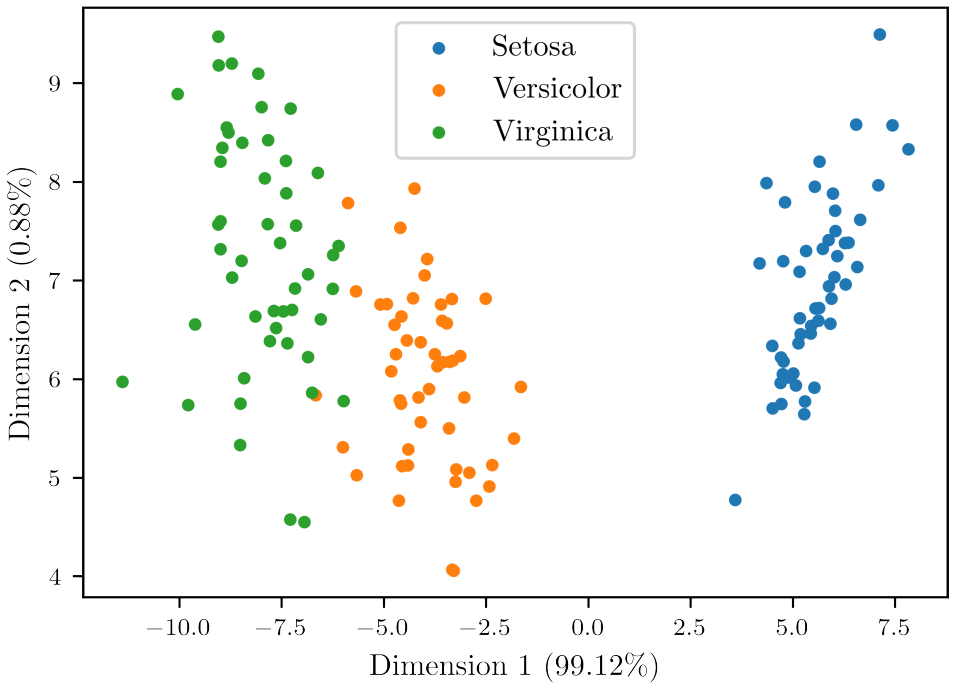

Figure 25: *Illustration of linear discriminant analysis as a dimensionality reduction technique. The Iris flower dataset consists of 50 samples for each of three iris species (setosa, versicolor and virginica) for which four features were measured: the length and the width of the sepals and petals, in centimeters. The projection of each sample on the learned two-dimensional space is shown in this figure.*

dimension. However, the criterion for linear discriminant analysis is different from the one from principal component analysis: it is to maximizing the separability of the classes instead of maximizing the explained variance.

Figure 25 illustrates the use of linear discriminant analysis as a dimensionality reduction technique. We use the same Iris flower dataset as in Figure 24 illustrating principal component analysis. The projection of each sample on the learned two-dimensional space is shown and one can see that the first (horizontal) axis is more discriminative of the three classes with linear discriminant analysis than with principal component analysis.

14. Kernel methods

Kernel methods allow for generalizing linear models to non-linear models with the use of kernel functions.

As mentioned in Section 8, the main idea of kernel methods is to first map the input data from the original input space to a feature space, and then perform dot products in this feature space. Under certain assumptions, an optimal solution of the minimization problem of the cost function admits the following form:

$$f = \sum_{i=1}^{n} \alpha_i K(\cdot, \boldsymbol{x}^{(i)})$$

where K is the kernel function which is equal to the dot product in the feature space:

$$\forall \boldsymbol{x}, \boldsymbol{x}' \in \mathcal{I},\ K(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^\top \phi(\boldsymbol{x}')$$

As this term frequently appears, we denote by $\boldsymbol{K}$ the $n \times n$ symmetric matrix consisting of the evaluations of the kernel on all the pairs of training samples:

$$\forall i, j \in \{1, \ldots, n\},\ K_{ij} = K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$$

In this section we present the extension of two models previously introduced in this chapter, ridge regression and principal component analysis, with kernel functions.

14.1 Kernel ridge regression

Kernel ridge regression combines ridge regression with the kernel trick, and thus learns a linear function in the space induced by the respective kernel and the training data [2]. For non-linear kernels, this corresponds to a non-linear function in the original input space.

Mathematically, the objective is to find the function f with the following form:

$$f = \sum_{i=1}^{n} \alpha_i K(\cdot, \boldsymbol{x}^{(i)})$$

that minimizes the sum of squared errors with a ℓ_2 penalization term:

$$\min_f \sum_{i=1}^{n} \left(y^{(i)} - f(\boldsymbol{x}^{(i)}\right)^2 + \lambda \|f\|^2$$

The cost function can be simplified using the specific form of the possible functions:

$$\begin{aligned}
&\sum_{i=1}^{n}(y^{(i)} - f(\boldsymbol{x}^{(i)})^2 + \lambda\|f\|^2 \\
&= \sum_{i=1}^{n}\left(y^{(i)} - \sum_{j=1}^{n}\alpha_j k(\boldsymbol{x}^{(j)}, \boldsymbol{x}^{(i)})\right)^2 + \lambda\left\|\sum_{i=1}^{n}\alpha_i K(\cdot, \boldsymbol{x}^{(i)})\right\|^2 \\
&= \sum_{i=1}^{n}\left(y^{(i)} - \boldsymbol{\alpha}^\top \boldsymbol{K}_{:,i}\right)^2 + \lambda\boldsymbol{\alpha}^\top\boldsymbol{K}\boldsymbol{\alpha} \\
&= \|\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\alpha}\|_2^2 + \lambda\boldsymbol{\alpha}^\top\boldsymbol{K}\boldsymbol{\alpha}
\end{aligned}$$

Therefore, the minimization problem is:

$$\min_{\boldsymbol{\alpha}} \|\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\alpha}\|_2^2 + \lambda\boldsymbol{\alpha}^\top\boldsymbol{K}\boldsymbol{\alpha}$$

for which a solution is given by:

$$\boldsymbol{\alpha}^\star = (\boldsymbol{K} + \lambda\boldsymbol{I})^{-1}\boldsymbol{y}$$

Figure 8 illustrates the prediction function of a kernel ridge regression algorithm with a radial basis function kernel. The prediction function is non-linear as the kernel is non-linear.

14.2 Kernel principal component analysis

As mentioned in Section 13, principal component analysis consists in finding the linear orthogonal subspace in the original input space such that each principal component explains the most variance. The optimal solution is given by the first eigenvectors of $\boldsymbol{X}^\top\boldsymbol{X}$ with the corresponding eigenvalues being sorted in descending order.

With kernel principal component analysis, the objective is to find the linear orthogonal subspace in the feature space such that each principal component in the feature space explains the most variance [26]. The solution is given by the first l eigenvectors $(\boldsymbol{\alpha}_k)_{1\leq k\leq l}$ of the $\boldsymbol{K}$ matrix with the corresponding eigenvalues being sorted in descending order. The eigenvectors are normalized in order to be unit vectors in the feature space.

Finally, the projection of any input $\boldsymbol{x}$ in the original space on the k-th component can be computed as:

$$\phi(\boldsymbol{x})^\top\boldsymbol{\alpha}_k = \sum_{i=1}^{n}\alpha_{ki}K(\boldsymbol{x}, \boldsymbol{x}^{(i)})$$

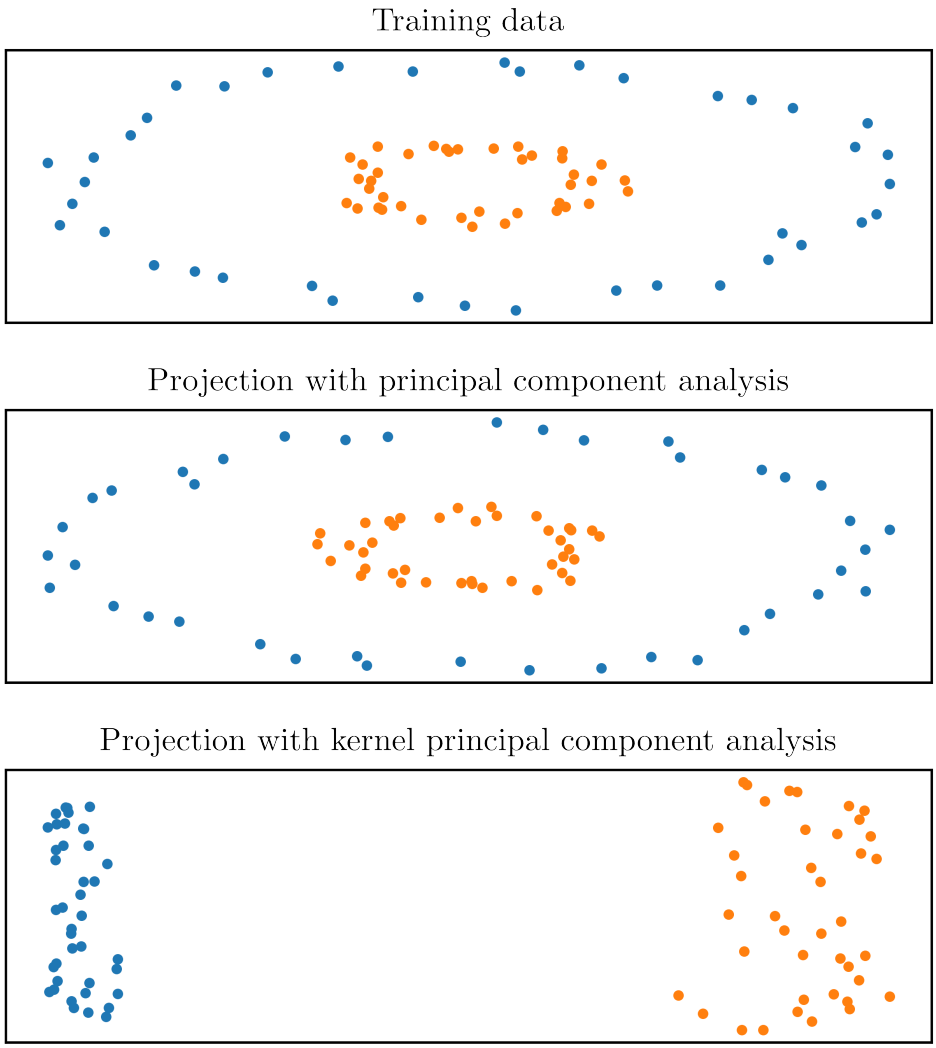

Figure 26: *Illustration of kernel principal component analysis. Some non-linearly separable training data is plotted (top). The projected training data using principal component analysis remains non-linearly separable (middle). The projected training data using kernel principal component analysis (with a non-linear kernel) becomes linearly separable (bottom).*

Figure 26 illustrates the projection of some non-linearly separable classification data with principal component analysis and with kernel principal component analysis with a non-linear kernel. The projected input data becomes linearly separable using kernel principal component analysis, whereas the projected input data using (linear) principal component analysis remains non-linearly separable.

15. Conclusion

In this chapter, we described the main classic machine learning methods. Due to space constraints, the description of some of them was brief. The reader who seeks more details can refer to [5, 6]. All these approaches are implemented in the scikit-learn Python library [27]. A common point of the approaches presented in this chapter is that they use as input a set of given or pre-extracted features. On the contrary, deep learning approaches often provide an end-to-end learning setup within which the features are learned. These techniques are covered in Chapters 3 to 6.

Acknowledgments

The authors would like to thank Hicham Janati for his fruitful remarks. The authors would like to acknowledge the extensive documentation of the scikit-learn Python package, in particular its user guide, for the relevant information and references provided. We used the NumPy [28], matplotlib [29] and scikit-learn [27] Python packages to generate all the figures. This work was supported by the French government under management of Agence Nationale de la Recherche as part of the "Investissements d'avenir" program, reference ANR-19-P3IA-0001 (PRAIRIE 3IA Institute) and reference ANR-10-IAIHU-06 (Agence Nationale de la Recherche-10-IA Institut Hospitalo-Universitaire-6), and by the European Union H2020 programme (grant number 826421, project TVB-Cloud).

References

[1] Goodfellow I, Bengio Y, Courville A (2016) Deep Learning. MIT Press, http://www.deeplearningbook.org

[2] Murphy KP (2012) Machine Learning: A Probabilistic Perspective. The MIT Press, Cambridge, MA

[3] Bentley JL (1975) Multidimensional binary search trees used for associative searching. Communications of the ACM 18(9):509–517

[4] Omohundro SM (1989) Five Balltree Construction Algorithms. Tech. rep., International Computer Science Institute

[5] Bishop CM (2006) Pattern Recognition and Machine Learning. Springer-Verlag

[6] Hastie T, Tibshirani R, Friedman J (2009) The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition, 2nd edn. Springer Series in Statistics, Springer-Verlag, New York

[7] Tikhonov AN, Arsenin VY, John F (1977) Solutions of Ill Posed Problems. John Wiley & Sons Inc, Washington : New York

[8] Tibshirani R (1996) Regression Shrinkage and Selection via the Lasso. Journal of the Royal Statistical Society Series B (Methodological) 58(1):267–288

[9] Zou H, Hastie T (2005) Regularization and variable selection via the elastic net. Journal of the Royal Statistical Society: Series B (Statistical Methodology) 67(2):301–320

[10] Vapnik VN, Lerner A (1963) Pattern recognition using generalized portrait method. Automation and Remote Control 24:774–780

[11] Cortes C, Vapnik V (1995) Support-vector networks. Machine Learning 20(3):273–297

[12] Boser BE, Guyon IM, Vapnik VN (1992) A training algorithm for optimal margin classifiers. In: Proceedings of the fifth annual workshop on Computational learning theory, Association for Computing Machinery, Pittsburgh, Pennsylvania, USA, COLT '92, pp 144–152

[13] Aizerman MA, Braverman EA, Rozonoer L (1964) Theoretical foundations of the potential function method in pattern recognition learning. In: Automation and Remote Control, 25, pp 821–837

[14] Schölkopf B, Herbrich R, Smola AJ (2001) A Generalized Representer Theorem. In: Computational Learning Theory, Springer, pp 416–426

[15] Aly M (2005) Survey on multiclass classification methods

[16] James G, Hastie T (1998) The Error Coding Method and PICTs. Journal of Computational and Graphical Statistics 7(3):377–387

[17] Breiman L, Friedman J, Stone CJ, Olshen RA (1984) Classification and Regression Trees. Taylor & Francis

[18] Breiman L (2001) Random Forests. Machine Learning 45(1):5–32

[19] Geurts P, Ernst D, Wehenkel L (2006) Extremely randomized trees. Machine Learning 63(1):3–42

[20] Lloyd S (1982) Least squares quantization in PCM. IEEE Transactions on Information Theory 28(2):129–137

[21] Elkan C (2003) Using the triangle inequality to accelerate k-means. In: Proceedings of the Twentieth International Conference on International Conference on Machine Learning, pp 147–153

[22] Arthur D, Vassilvitskii S (2007) k-means++: the advantages of careful seeding. In: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, pp 1027–1035

[23] Aggarwal CC, Hinneburg A, Keim DA (2001) On the Surprising Behavior of Distance Metrics in High Dimensional Space. In: International Conference on Database Theory, Springer, pp 420–434

[24] Dempster AP, Laird NM, Rubin DB (1977) Maximum Likelihood from Incomplete Data via the EM Algorithm. Journal of the Royal Statistical Society Series B (Methodological) 39(1):1–38

[25] Jolliffe IT (2002) Principal Component Analysis, 2nd edn. Springer Science & Business Media

[26] Schölkopf B, Smola AJ, Müller KR (1999) Kernel principal component analysis. In: Advances in kernel methods: support vector learning, MIT Press, pp 327–352

[27] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, et al (2011) Scikit-learn: Machine learning in python. the Journal of machine Learning research 12:2825–2830

[28] Harris CR, Millman KJ, van der Walt SJ, Gommers R, Virtanen P, Cournapeau D, Wieser E, Taylor J, Berg S, Smith NJ, et al (2020) Array programming with numpy. Nature 585(7825):357–362

[29] Hunter JD (2007) Matplotlib: A 2d graphics environment. Computing in science & engineering 9(03):90–95