Part I

Work positioning

The first part of the manuscript aims to position the work context. Our objective is the comparison and the classification or regression of time series. The first chapter considers time series as static vector data and presents classic Machine Learning algorithms. Most of these methods relies on the comparison of objects (time series in our case) through a distance measure. In the second chapter, to cope with the modalies inherent to time series (amplitude, behavior, frequential spectrum, etc.), we recall some basic metrics used to compare time series. We show that time series may be compared by several modalities and at different granularities. We finally cast that learning an adequate distance based on several modalities and several granularities is a key challenge nowadays to well classify time series by classic Machine Learning algorithms.

CHAPTER 1

Machine Learning: state of the art

Sommaire

In this chapter, we first present what a time series is. Then, we vectorize time series, consider them as static data and apply classic Machine Learning algorithms used for classification and regression. We review the protocol used in supervised learning to learn the best fitting of the hyper-parameters and how we evaluate and compare different algorithms performances. Finally, we give some more detailed insights on k-Nearest Neighbors (k-NN) and Support Vector Machine (SVM), used in our work.

1.1 Definition of a time series

Time series and more generally temporal data are data objects that frequently occur in physical sciences (meteorology, marine science, geophysics), marketing or process control [Cha04]. For physical systems, a time series of length N can be seen as a signal, sampled at a frequency f_e, in a temporal window $[0; \frac{N}{f_e}]$. From a mathematical perspective, a time series is a collection of a finite number of realized normalized observations made sequentially at discrete time instants $t = 1, ..., T$. Note that when $f_e = 1$, $T = N$.

Let $\mathbf{x}_i = (x_{i1}, x_{i2}, ..., x_{iT})$ be a univariate time series of length T. Each observation x_{it} is bounded (i.e., the infinity is not a valid value: $x_{it} \neq \pm\infty$). The time series $\mathbf{x}_i$ is said to be

univariate if the collection of observations x_{it} comes from the observations of one variable (i.e., it has been measured by one sensor, the temperature for example). When the observations are made at the same time from Q variables (several sensors such as the temperature, the pressure, etc.), the time series is said multivariate and is denoted $\mathbf{x}_i = (\mathbf{x}_{i,1}, ..., \mathbf{x}_{i,Q}) = (x_{i1,1}, ..., x_{iT,1}, x_{i1,2}, ..., x_{iT,2}, ..., x_{i1,Q}, ..., x_{iT,Q})$. For simplification purpose, we consider in the following univariate time series.

Time series can be found in various emerging applications such as sensor networks, smart buildings, social media networks or Internet of Things (IoT) [Naj+12]; [Ngu+12]; [YG08]. They are involved in many learning problems such as recognizing a human movement in a video, detect a particular operating mode, etc. [PAN+08]; [Ram+08]. In **clustering** problems, one would like to organize similar time series together into homogeneous groups. In **classification** problems, the aim is to assign time series to one of several predefined categories (e.g., different types of defaults in a machine). In **regression** problems, the objective is to predict a continuous value from observed time series (e.g., forecasting the measurement of a power meter from pressure and temperature sensors). Our work focus on classification and regression problems. However, due to their temporal and structured nature, time series constitute complex data to be analyzed by classic Machine Learning approaches.

To overcome this complexity, some authors propose to extract representative features from time series. Fig. 1.1 illustrates a model for time series proposed by Chatfield in [Cha04]. It states that a time series can be decomposed into 3 components: a trend, a cycle (periodic component) and a residual (irregular variations).

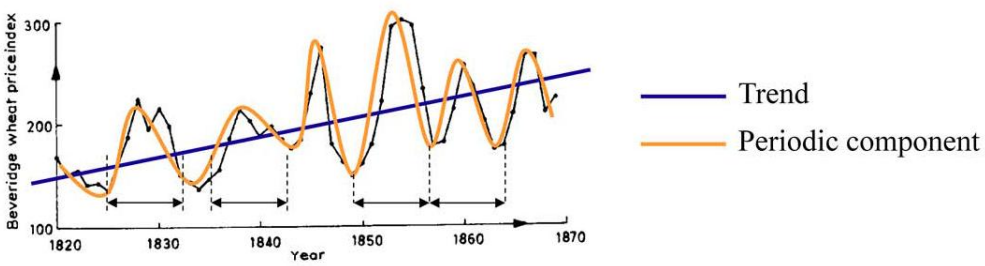


Figure 1.1: The Beveridge wheat price index is the average in nearly 50 places in various countries measured in successive years from 1500 to 1869. [1]

According to Chatfield, most time series exhibit a variation at a fixed period of time (seasonality) such as for example the seasonal variation of temperature. Beyond this cycle, there exists either or both a long term change in the mean (trend) that can be linear, quadratic, and a periodic (cyclic) component. In practice, these 3 features are rarely sufficient for the classification or regression of real time series. In our work, we propose to focus on the raw time series and do not try to extract global features from the time series.

Due to their complexity, other authors made the hypothesis of time independency between the observations x_{it}. They consider time series as a static vector data (samples) and use classic

[1] This time series can be downloaded from `http://www.york.ac.uk/depts/maths/data/ts/ts04.dat`

Machine Learning algorithms [Lia+12]; [CT01]; [HWZ13]; [HHK12].

1.2 Machine Learning framework

In this section, we review some basic terminology in Machine Learning. First, we recall the principle of statistical learning. Then, we detail how to design a framework in the case of supervised learning. After that, we recall how model evaluation is performed. Finally, we take an insight on data pre-processing.

1.2.1 Classification, regression

The idea of Machine Learning (also refer as Pattern Learning or Pattern Recognition) is to imitate with algorithms executed on computers, the ability of living beings to learn from examples. For instance, to teach a child how to read letters, we show him during the training phase labeled examples of letters ('A', 'B', 'C', etc.) written in different styles and fonts. We don't give him a complete and analytic description of the topology of the characters but labeled examples. After the training phase (testing phase), we want the child to be able to recognize and to label correctly the letters that have been seen during the training, and also to generalize to new instances [G. 06].

Let $X = \{\mathbf{x}_i, y_i\}_{i=1}^{n}$ be a training set of n samples $\mathbf{x}_i$ (time series in our case) and y_i their corresponding labels. The aim of Machine Learning is to learn a relation (model) f between the samples $\mathbf{x}_i$ and their labels y_i based on examples. This relationship can include static relationships, correlations, dynamic relationship, etc. After the training phase based on labeled examples $(\mathbf{x}_i, y_i)$, the model f has to be able to generalize on the testing phase, i.e., to give a correct prediction y_j for new instances $\mathbf{x}_j$ that haven't been seen during the training.

When y_i are class labels (e.g., class 'A', 'B', 'C' in the case of child's reading), learning the model f is a classification problem; when y_i is a continuous value (e.g., the energy consumption in a building), learning f is a regression problem. Both problems corresponds to supervised learning as $\mathbf{x}_i$ and y_i are known during the training phase [Bis06]; [G. 06]; [OE73]. For both problems, when a part of the labels y_i are known and an other part of y_i is unknown during training, learning f is a semi-supervised problem . Note that when the labels y_i are unknown, learning f refers to a clustering problem (unsupervised learning) [JMF99]; [CHY96], not managed in our work.

[biblio semi-supervisé]

1.2.2 Supervised learning framework

A key objective of learning algorithms is to build models f with good generalization abilities, i.e., models f that correctly predict the class labels y_j of new unknown samples $\mathbf{x}_j$. Fig. 1.3 shows a general approach for solving Machine Learning problems. In general, a dataset can

be divided into 3 sub-datasets (illustrated in Fig. 1.2):

- A **training set** $X = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ consisting of n samples $\mathbf{x}_i$ whose labels y_i are known. The training set is used to build the supervised model f.
- A **test set** $X_{Test} = \{(\mathbf{x}_j, y_j)\}_{j=1}^{m}$, which consists of m samples $\mathbf{x}_j$ which labels y_j are also known but the model f is applied to predict the label $\hat{y_j}$ of samples $\mathbf{x}_j$. The test is used to evaluate the performance of the learnt model between $\hat{y_j}$ and y_j.
- An **evaluation set** $X_{Eval} = \{(\mathbf{x}_l, y_l)\}_{l=1}^{L}$, which consists of L samples $\mathbf{x}_l$ with labels y_l are unknown. The evaluation set is in general a new dataset on which we would like to apply the learnt algorithm.

id	Attribute 1	Attribute 2	Attribute 3	True Class
1	Yes	Large	125	No
2	No	Medium	135	Yes
3	No	Small	256	No
4	Yes	Medium	320	No
5	Yes	Small	128	Yes
6	No	Large	852	Yes
7	No	Medium	963	Yes

Learning Set

Validation Set

Training Set

id	Attribute 1	Attribute 2	Attribute 3	True Class	Predicted Class
8	No	Large	566	No	?
9	No	Medium	456	Yes	?
10	Yes	Medium	321	No	?
11	No	Small	243	No	?
12	Yes	Small	863	Yes	?
13	No	Large	213	Yes	?
14	Yes	Large	132	Yes	?

Test Set

id	Attribute 1	Attribute 2	Attribute 3	True Class	Predicted Class
15	Yes	Large	874	?	?
16	No	Medium	541	?	?
17	No	Medium	236	?	?
18	No	Large	652	?	?
19	Yes	Small	324	?	?
20	Yes	Small	214	?	?
21	Yes	Medium	222	?	?

Evaluation Set

Figure 1.2: Division of a dataset into 3 datasets: training, test and evaluation.

The errors committed by a classification or regression model are divided into two types: training errors and generalization errors (testing errors). **Training error** is the number of misclassification errors in classification (Root Mean Square Error or other error measures used in regression) committed on training samples $\mathbf{x}_i$, whereas **generalization error** is the expected error of the model f on unseen samples $\mathbf{x}_j$. A good supervised model f must not only fit the training data X well, it must also accurately classify records it has never seen before (test set X_{Test}). In other words, a good model f must have low training error as well as low generalization error. This is important because a model that fits the training data too

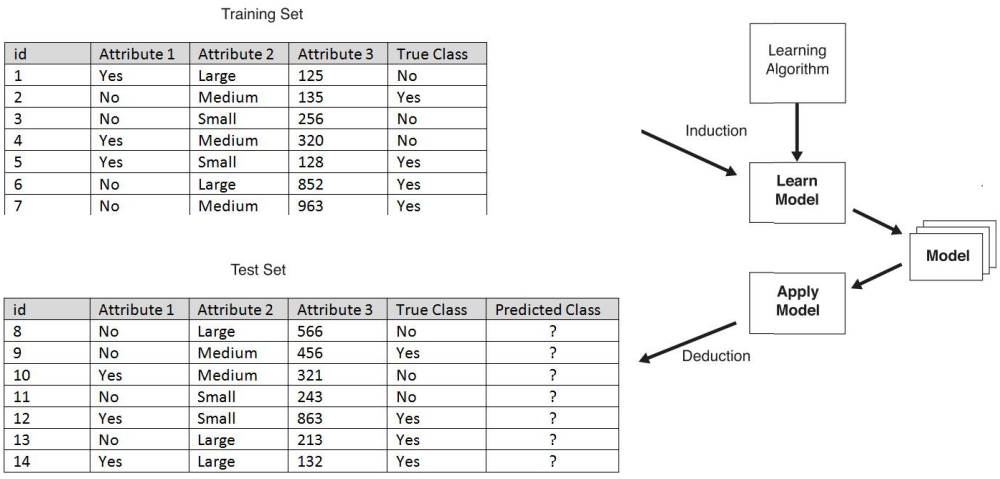


Figure 1.3: General framework for building a supervised (classification/regression) model. Example with 3 features and 2 classes ('Yes' and 'No').

well can have a poorer generalization error than a model with a higher training error. Such a situation is known as model overfitting (Fig. 1.4).

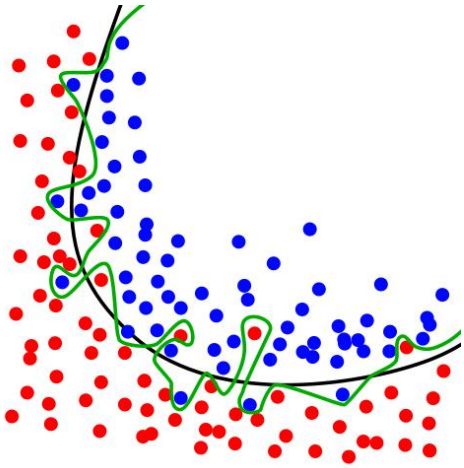


Figure 1.4: An example of overfitting in the case of classification. The objective is to separate blue points from red points. Black line shows a classifier f_1 with low complexity where as green line illustrates a classifier f_2 with high complexity. On training examples (blue and red points), the model f_2 separates all the classes perfectly but may lead to poor generalization on new unseen examples. Model f_1 is often preferred.

In most cases, learning algorithms requires to tune some hyper-parameters. For that, the training set can be divided into 2 sets: a learning and a validation set. Suppose we have two hyper-parameters to tune: C and γ. We make a grid search for each combination (C, γ) of the hyper-parameters, that is in this case a 2-dimensional grid (Fig. 1.5). For each combination (a cell of the grid), the model is learnt on the learning set and evaluated on the validation set.

At the end, the model with the lowest error on the validation set is retained. This process is refered as the model selection.

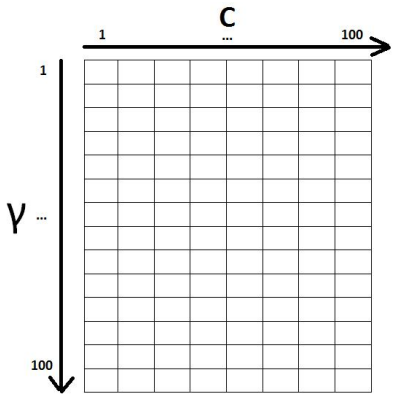


Figure 1.5: Example of a 2 dimensional grid search for parameters C and γ. It defines a grid where each cell of the grid contains a combination (C, γ). Each combination is used to learn the model and is evaluated on the validation set.

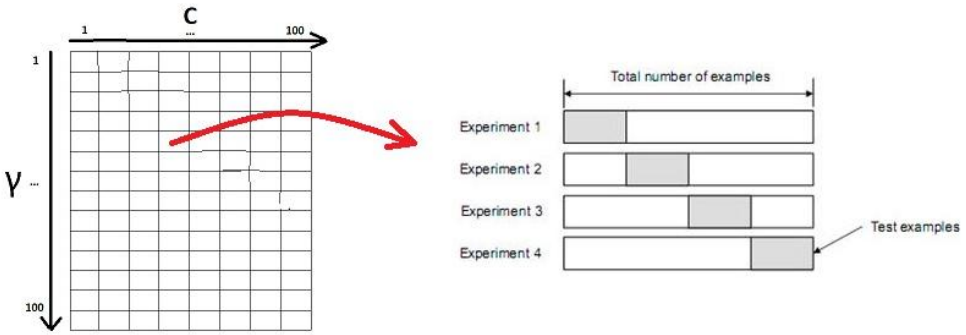


Figure 1.6: v-fold Cross-validation for one combination of parameters. For each of v experiments, use $v-1$ folds for training and a different fold for Testing, then the training error for this combination of parameter is the mean of all testing errors. This procedure is illustrated for $v = 4$.

An alternative is cross-validation with v folds, illustrated in Fig. 1.6. In this approach, we partition the training data into v equal-sized subsets. The objective is to evaluate the error for each combination of hyper-parameters. For each run, one fold is chosen for validation, while the $v-1$ rest folds are used as the learning set. We repeat the process for each fold, thus v times. Each fold gives one validation error and thus we obtain v errors. The total error for the current combination of hyper-parameters is obtained by summing up the errors for all v folds. When $v = n$, the size of training set, this approach is called leave-one-out or Jackknife. Each test set contains only one sample. The advantage is much data are used as possible for training. Moreover, the test sets are exclusive and they cover the entire data set. The drawback is that it is computationally expensive to repeat the procedure n times. Furthermore, since each test

set contains only one record, the variance of the estimated performance metric is usually high. This procedure is often used when n, the size training set, is small.

There exists other methods such as sub-sampling or bootstraps [OE73]; [G. 06]. We only use cross-validation in our experiments.

1.2.3 Model evaluation

1.2.3.a Classification evaluation

The performance of a classification model is based on the counts of test samples $\mathbf{x}_j$ correctly and incorrectly predicted by the model f. These counts are tabulated in a table called the confusion matrix. Table 1.1 illustrates the concept for a binary classification problem. Each cell f_{ij} the table stands for the number of samples from class i predicted to be of class j. Based on this matrix, the number of correct predictions made by the model is $\sum_{i=1}^{C} f_{ii}$, where C is the number of classes. Equivalently, the ratio of incorrect predictions is $1 - \sum_{i=1}^{C} f_{ii}$.

		Predicted class	
		Class = 1	Class = 0
Actual Class	Class = 1	f_{11}	f_{10}
	Class = 0	f_{01}	f_{00}

Table 1.1: Confusion matrix for a 2-class problem.

To summarize the information, it generally more convenient to use performance metrics such as the classification accuracy (Acc) or error rate (Err). This allows to compare several models with a single number. Note that $Err = 1 - Acc$.

$$Acc = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} = \frac{\sum_{i=1}^{C} f_{ii}}{\sum_{i,j=1}^{C} f_{ij}} \tag{1.1}$$

$$Err = \frac{\text{Number of wrong predictions}}{\text{Total number of predictions}} = \frac{\sum_{i,j=1,i\neq j}^{C} f_{ij}}{\sum_{i,j=1}^{C} f_{ij}} \tag{1.2}$$

Using these performance metrics allows to compare the performance of different classifiers f. It allows to determine in particular whether one learning algorithm outperforms another on a particular learning task on a given test dataset X_{Test}. However, depending on the size of the test dataset, the difference in error rate Err between two classifiers may not be statistically significant. Snedecor & Cochran proposed in 1989 a statistical test based on measuring the difference between two learning algorithms [Coc77]. It has been used by many researchers

[Die97]; [DHB95].

Let consider 2 classifiers f_A and f_B. We test these classifiers on the test set X_{Test} and denote p_A and p_B their respective error rates. The intuition of this statistical test is that when algorithm A classifies an example $\mathbf{x}_j$ from the test set X_{Test}, the probability of misclassification is p_A. Thus, the number of misclassification of m test examples is a binomial random variable with mean mp_A and variance $p_A(1-p_A)m$. The binomial distribution can be approximated by a normal distribution when m has a reasonable value (Law of large numbers). The difference between two independent normally distributed random variables is also normally distributed. Thus, the quantity $p_A - p_B$ is a normally distributed random variable. Under the null hypothesis (the two algorithm should have the same error rate), this will have a mean of zero and a standard error se of:

$$se = \sqrt{\frac{2p(1-p)}{m}} \tag{1.3}$$

where $p = \frac{p_A+p_B}{2}$ is the average of the two error probabilities. From this analysis, we obtain the statistic:

$$z = \frac{p_A - p_B}{\sqrt{2p(1-p)/m}} \tag{1.4}$$

which has (approximatively) a standard normal distribution. We can reject the null hypothesis if $|z| > Z_{0.975} = 1.96$ (for a 2-sided test with probability of incorrectly rejecting the null hypothesis of 0.05).

1.2.3.b Regression evaluation

As the concept of classes is restricted to classification problems, the performance of a regression model f is based on metrics that measure the difference between the predicted label $\hat{y}_j$ and the known label y_j. The Mean Absolute Error function (MAE) computes the mean absolute error, a risk metric corresponding to the expected value of the absolute error loss or L1-norm loss.

$$MAE(\hat{y}, y) = \frac{1}{m}\sum_{j=1}^{m}|\hat{y}_j - y_j| \tag{1.5}$$

A commonly used performance metrics is the Root Mean Squared Error function (RMSE) that computes the root of the mean square error, a risk metric corresponding to the expected value of the squared (quadratic) error loss.

$$RMSE(\hat{y}, y) = \sqrt{\frac{1}{m}\sum_{j=1}^{m}(\hat{y}_j - y_j)^2} \tag{1.6}$$

Many works relies on the R^2 function, the coefficient of determination. It provides a measure of how well future samples are likely to be predicted by the model.

$$R^2(\hat{y}, y) = 1 - \frac{\sum_{j=1}^{m}(\hat{y}_j - y_j)^2}{\sum_{j=1}^{m}(\bar{y} - y_j)^2} \tag{1.7}$$

where $\bar{y} = \sum_{j=1}^{m} y_j$ is the mean over the known labels y_j.

1.2.4 Data pre-processing

Real dataset are often subjected to noise or data scaling. Before applying any learning protocol, it is often necessary to pre-process the data: data scaling, data filtering (e.g., de-noising), etc. We focus on data scaling in our work.

Part 2 of Sarle's Neural Networks FAQ (1997) [2] explains the importance of theses considerations for neural network but they can be applied to any learning algorithms. The main advantage of scaling is to avoid attributes in greater numeric ranges to dominate those in smaller numeric ranges. Another advantage is to avoid numerical difficulties during the calculation. For example, in the case of SVM, because kernel values usually depend on the inner products of feature vectors, i.e. the linear kernel and the polynomial kernel, large attribute values might cause numerical problems.

In most cases, it is recommended to scale each attribute to the range [-1; +1] or [0; 1]. Many normalization methods have been proposed such as Min/Max normalization, Z-normalization or normalization of the log normalization . Let $X = \{\mathbf{x}_i, y_i\}_{i=1}^{n}$ be a training set, $\mathbf{x}_i$ being a sample described by T features $\mathbf{X}_1, \ldots, \mathbf{X}_T$. We define μ_j and σ_j as the mean and the standard deviation of a variable $\mathbf{X}_j$, applying the Z-normalized variable $\mathbf{X}_j^{norm}$ is given by:

références normalization

$$\mathbf{X}_j^{norm} = \frac{\mathbf{X}_j - \mu_j}{\sigma_j} \tag{1.8}$$

Note that the underlying assumption supposes that the variable $\mathbf{X}_j$ is normally distributed: data evolves between $[-\infty; +\infty]$ and are coming from a Gaussian process. In some cases, the data are skewed such as monetary amounts, incomes or distance measures. These data are often log-normally distributed, e.g., the log of the data is normally distributed (Fig. 1.7). The underlying idea is to take the log of the data ($\mathbf{X}_j^{log}$) to restore the symmetry, and then, to

[2] http://www.faqs.org/faqs/ai-faq/neural-nets/

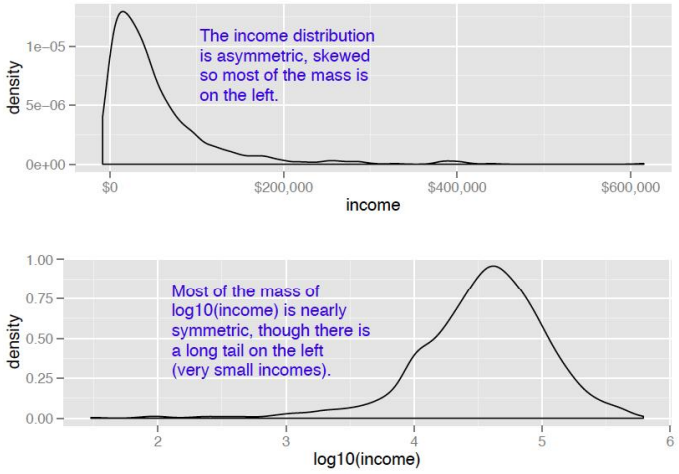


Figure 1.7: A nearly log-normal distribution, and its log [3]

apply a Z-normalization of this transformation:

$$\mathbf{X}_j^{log} = \ln(\mathbf{X}_j); \tag{1.9}$$

$$\mathbf{X}_j^{log,norm} = \frac{\mathbf{X}_j^{log} - \mu_j^{log}}{\sigma_j^{log}} \tag{1.10}$$

$$\mathbf{X}_j^{norm} = \exp(\mathbf{X}_j^{log,norm}) \tag{1.11}$$

where ln denotes the Natural Logarithm function, μ_j^{log} and σ_j^{log} the mean and the standard deviation of a variable $\mathbf{X}_j^{log}$.

Finally, we recall some precautions to the practitioner in the learning protocol, experimented by Hsu & al. in the context of Support Vector Machine (SVM) [HCL08]. First, training and testing data must be scaled using the same method. Second, training and testing data must not be scaled separately. Third, the whole dataset must not be scaled together at the same time. These often leads to poorer results. A proper way to do normalization is to scale the training data, store the parameters of the normalization (i.e. μ_i and σ_i for Z-normalization), then apply the same normalization to the testing data.

[3] source: `http://www.r-statistics.com/2013/05/log-transformations-for-skewed-and-wide-distributions-from-practical-data-science-with-r/`

1.3 Machine Learning algorithms

Many popular algorithms have been proposed in Machine Learning, in the context of supervised learning, such as the Deep Neural Network, the Decision Tree or the Relevance Vector Machine (RVM). We focus on k-Nearest Neighbors (k-NN) and Support Vector Machine (SVM). The interest of these two is that they are based on the comparison of samples (time series in our case) through a distance measure, notion detailed in the next chapters.

1.3.1 k-Nearest Neighbors (k-NN) classifier

A simple approach to classify samples is to consider that "close" samples have a great probability to belong to the same class. Given a test sample $\mathbf{x}_j$, one can decide that $\mathbf{x}_j$ belong to the same class of its nearest neighbor in the training set. More generally, we can consider the k nearest neighbors of $\mathbf{x}_j$. The class y_j of the test sample $\mathbf{x}_j$ is assigned with a voting scheme among them, i.e., using the majority of the class of nearest neighbors. This algorithm is refer as the k-Nearest Neighbors algorithm (k-NN) **Silverman1989**; [CH67]. Fig. 1.8 illustrates the concept for a neighborhood of $k = 3$ and $k = 5$.

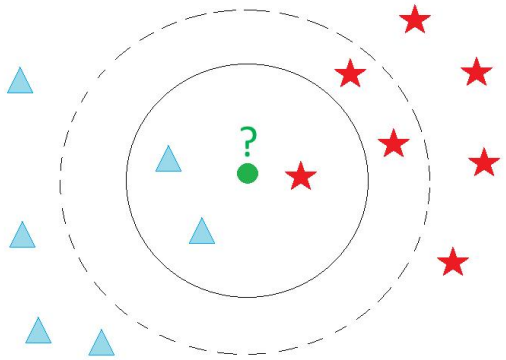


Figure 1.8: Example of k-NN classification. The test sample (green circle) is classified either to the first class (red stars) or to the second class (blue triangles). If $k = 3$ (solid line circle) it is assigned to the second class because there are 2 triangles and only 1 star inside the inner circle. If $k = 5$ (dashed line circle) it is assigned to the first class (3 stars vs. 2 triangles inside the outer circle).

In the k-NN algorithm, the notion of "closeness" between samples is based on the computation of a metric [4] D. Usually, for static data, standard metrics are the Euclidean distance, the Minkowski distance or the Mahalanobis distance[5]. Considering a training set X of n samples, solving the 1-NN classification problem is equivalent to solve the optimization problem:
For a new sample $\mathbf{x}_j$, $\forall i \in \{1...n\}$,

$$y_j = y_{i^*} \tag{1.12}$$

[4] A clarification of the terms metric, distance, dissimilarity, etc. will be given in Chapter 2. For now, we refer all of them as metrics.

[5] A recall of these metrics will be in Chapter 2.

where $i^* = \underset{i \in \{1...n\}}{\text{argmin}} D(\mathbf{x}_i, \mathbf{x}_j)$.

The k-NN algorithm can be extended to estimate continous labels (regression problems). The procedure is similar. The label y_j is defined as :

$$y_j = \sum_{i=1}^{k} y_i \tag{1.13}$$

where i corresponds to the index of the k-nearest neighbors **Altman1992** There exists other variants of the k-NN algorithms. In a weighed k-NN, the approach consists in weighting the k-NN decision by assigning to each neighbor $\mathbf{x}_i$ from an unknown sample $\mathbf{x}_j$, a weight w_i defined as a function of the distance $D(\mathbf{x}_i, \mathbf{x}_j)$ **Dudani1976** To cope with uncertainty or imprecision in the labeling of the training data $\mathbf{x}_i$, other authors propose in a fuzzy k-NN to determine the membership degree in each class of an unseen sample $\mathbf{x}_j$ by combining the memberships of its neighbors **Keller1985** Denoeux propose a framework based on Dempster-Shafer theory where the k-rule that takes into account the non-representativity of training data, the weighting rule and uncertainty in the labeling [Den95].

Despite its simplicity, the k-NN algorithm presents many advantages and have been shown to be successful on time series classification problems [BMP02]; [Xi+06]; [Din+08]. One main advantage is that a 1-NN classifier can be used to evaluate and compare the efficacy of different metrics [Din+08]. First, the underlying metric is critical in the performance of the 1-NN classifier [TSK05]. Thus, the accuracy of the 1-NN classifier directly reflects the effectiveness of the metric. Second, 1-NN classifier is easy to implement and doesn't need to learn any hyper-parameters, which make it straightforward for anyone to reproduce results. All of this advantages allows one who want to evaluate a benchmark of metrics. Other methods to compare metrics exists such as clustering with small data sets which are not statistically significant, or compare the compactness of the metric [MP07]; [Vla+06]. The 1-NN algorithm will be used in our experiments to compare the performances different metrics used for time series.

However, the k-NN algorithm presents some disadvantages, mainly due to its computational complexity, both in space (storage of the training samples $\mathbf{x}_i$) and time (search of the neighbors) [OE73]. Suppose we have n labeled training samples in T dimensions, and find the closest neighbors to a test sample $\mathbf{x}_j$ ($k = 1$). In the most simple approach, we look at each stored samples $\mathbf{x}_i$ ($i = 1...n$) one by one, calculate its metric to $\mathbf{x}_i$ (D($\mathbf{x}_i$,$\mathbf{x}_j$)) and retain the index of the current closest one. For the standard Euclidean distance, each metric computation is $O(T)$ and thus the search is $O(Tn^2)$. Moreover, using standard metrics (such as the Euclidean distance) uses all the T dimensions in its computation and thus assumes that all dimensions have the same effect on the metric. This assumption may be wrong and can impact the classification performances. Wrong classification due to presence of many irrelevant dimensions is referred as the curse of dimensionality. The importance of defining adapted metrics for time series will be discussed in Chapter 2.

1.3.2 Support Vector Machine (SVM) algorithm

Support Vector Machine (SVM) is a state of the art classification method introduced in 1992 by Boser, Guyon, and Vapnik [BGV92]; [CV95]. The SVM classifier have demonstrate high accuracy, ability to deal with high-dimensional data, good generalization properties and interpretation for various applications from recognizing handwritten digits, to face identification, text categorization, bioinformatics and database marketing [Wan02]; [YL99]; [HHP01]; [SSB03]; [CY11]. SVMs belong to the category of kernel methods, algorithms that depends on the data only through dot-products [SS13].

As SVMs will be used in Chapter 4, we first present an intuition of maximum margin concept. We give the primal formulation of the SVM optimization problem. Then, by transforming the latter formulation into its dual form, the kernel trick can be applied to learn non-linear classifiers. Finally, we detail how we can interpret the obtained coefficients and how SVMs can be extended for regression problems.

Note that this section doesn't aim to give an detailed comprehension of SVMs. It aims to give an overview of the mathematical key points interpretation and comprehension of the method. For more informations, the reader can consult [SS13]; [CY11]; [CV95].

1.3.2.a Intuition

Let consider a classification problem with 2 classes ($y_i = \pm 1$). The objective is to learn a hyperplane, whose equations are $\mathbf{w}.\mathbf{x} + b = 0$[6], that can separate samples of class +1 from the ones of class -1. When the problem is linearly separable such as in Fig. 1.9, there exists an infinite number of hyperplanes. We denote $||\mathbf{w}||_2$, the L2-norm of the vector $\mathbf{w}$.

Vapnik & al. [CV95] propose to choose the separating hyperplane that maximizes the margin, e.g. the hyperplane that leaves as much distance as possible between the hyperplane and the closest examples of each class, called the support vectors. This distance is equal to $\frac{1}{||\mathbf{w}||_2}$. The hyperplanes passing through the support vectors of each class are referred as the canonical hyperplanes, and the region between the canonical hyperplanes is called the margin band (Fig. 1.10).

1.3.2.b Primal formulation

Finding $\mathbf{w}$ and b by maximizing the margin $\frac{1}{||\mathbf{w}||_2}$ is equivalent to minimizing the norm of $\mathbf{w}$ such that all samples from the training set are correctly classified:

$$\underset{\mathbf{w},b}{\operatorname{argmin}} \frac{1}{2}||\mathbf{w}||_2^2 \tag{1.14}$$

$$\textbf{s.t. } y_i(\mathbf{w}.\mathbf{x}_i + b) \geq 1 \tag{1.15}$$

[6]The scalar product between two vectors $\mathbf{x}$ and $\mathbf{y}$ is denoted: $\mathbf{x}.\mathbf{y} =< \mathbf{x}, \mathbf{y} >= \mathbf{x}'\mathbf{y}$

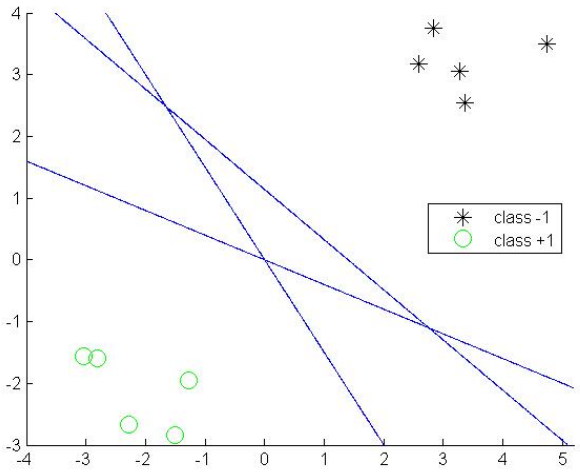


Figure 1.9: Example of linear classifiers in a 2-dimensional plot. For a set of points of classes +1 and -1 that are linearly separable, there exists an infinite number of separating hyperplanes corresponding to $\mathbf{w}.\mathbf{x} + b = 0$.

This is a constrained optimization problem in which we minimize an objective function (Eq. 1.14) subject to constraints (Eq. 1.15). This formulation is referred as the primal hard margin problem. Many real life datasets are subjected to noise and SVM can lead to poor generalization if it tries to fit to this noise, represented by the constraints in Eq. 1.15. The effects of noise can be reduced by introducing slack variables $\xi_i \geq 0$ to relax the optimization problem:

$$\underset{\mathbf{w},b}{\operatorname{argmin}} \left(\overbrace{\frac{1}{2}||\mathbf{w}||_2^2}^{Regularization} + C \overbrace{\sum_{i=1}^{n} \xi_i(\mathbf{w}; b; x_i; y_i)}^{Loss} \right) \tag{1.16}$$

$$\textbf{s.t.} \quad y_i(\mathbf{w}.\mathbf{x}_i + b) \geq 1 - \xi_i(\mathbf{w}; b; x_i; y_i) \tag{1.17}$$

$$\xi_i(\mathbf{w}; b; x_i; y_i) \geq 0 \tag{1.18}$$

where $C > 0$ is a penalty hyper-parameter.

This formulation is referred as the primal soft margin problem. It is quadratic programming optimization problem subjected to constraints. Thus, it is a convex problem: any local solutions is a global solution. The objective function in Eq. 1.16 is made of two terms. The first one, the regularization term, penalizes the complexity of the model and thus, controls the ability of the algorithm to generalize on new samples. The second one, the loss term, is an adaptation term to the data. The hyper-parameter C is a trade-off between the regularization and the loss term. When C tends to $+\infty$, the problem is equivalent to the primal hard margin problem. The hyper-parameter C is learnt during the training phase.

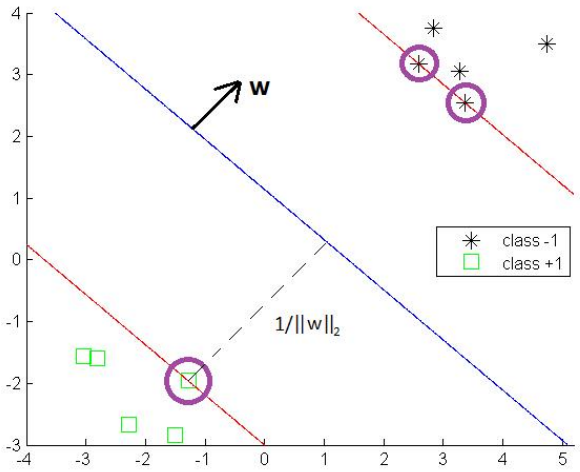


Figure 1.10: The argument inside the decision function of a classifier is $\mathbf{w}.\mathbf{x}+b$. The separating hyperplane corresponding to $\mathbf{w}.\mathbf{x}+b=0$ is shown as a line in this 2-dimensional plot. This hyperplane separates the two classes of data with points on one side labeled $y_i=+1$ ($\mathbf{w}.\mathbf{x}+b\geq 0$) and points on the other side labeled $y_i=-1$ ($\mathbf{w}.\mathbf{x}+b<0$). Support vectors are circled in purple and lies on the hyperplanes $\mathbf{w}.\mathbf{x}+b=+1$ and $\mathbf{w}.\mathbf{x}+b=-1$

For SVM, the two common loss functions ξ_i are $\max(1-y_i\mathbf{w}.\mathbf{x}_i,0)$ and $[\max(1-y_i\mathbf{w}.\mathbf{x}_i,0)]^2$. The former is referred to as L1-Loss and the latter is L2-Loss function. L2-loss function will penalize more slack variables ξ_i during training. Theorically, it should lead to less error in training and poorer generalization in most of the case.

An other thing to specify is the type of regularizer. For SVM, the two common regularizer are $||\mathbf{w}||_2$ and $||\mathbf{w}||_2^2$. The former is referred to as L1-Regularizer while the latter is L2-Regularizer. L1-Regularizer is used to obtain sparser models than L2-Regularizer. Thus, it can be used for variable selection. L2-Regularizer allows to transform the primal formulation into a dual form.

From this, for a binary classification problem, to classify a new sample $\mathbf{x}_j$, the decision function is:

$$f(\mathbf{x}_j)=sign(\mathbf{w}.\mathbf{x}_j+b) \tag{1.19}$$

1.3.2.c Dual formulation

From the primal formulation, using a L2-Regularizer, it is possible to have an equivalent dual form. This latter formulation allows samples $\mathbf{x}_i$ to appear in the optimization problem through dot-products only. Thanks to that, a kernel trick can be applied to extend the methods to learn non-linear classifiers.

First, to simplify the calculation development, let consider the hard margin formulation in

Eq. 1.16, 1.17 and 1.18 with a L1-Loss function. As a constrained optimization problem, the formulation is equivalent to the minimization of a Lagrange function $L(\mathbf{w}, b)$, consisting of the sum of the objective function and the n constraints multiplied by their respective Lagrange multipliers $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_n]'$:

$$\underset{\boldsymbol{\alpha}}{\operatorname{argmax}} \left(L(\mathbf{w}, b) = \frac{1}{2}(\mathbf{w}.\mathbf{w}) - \sum_{i=1}^{n} \alpha_i (y_i(\mathbf{w}.\mathbf{x}_i + b) - 1) \right) \tag{1.20}$$

s.t. $\forall i = 1...n$:

$$\alpha_i \geq 0 \tag{1.21}$$

$$y_i(\mathbf{w}.\mathbf{x}_i + b) - 1 \geq 0 \tag{1.22}$$

$$\alpha_i (y_i(\mathbf{w}.\mathbf{x}_i + b) - 1) = 0 \tag{1.23}$$

where $\alpha_i \geq 0$ are the Lagrange multipliers. In optimization theory, Eq. 1.21, 1.22 and 1.23 are called the Karush-Kuhn-Tucker (KKT) conditions. It corresponds to the set of conditions which must be satisfied at the optimum of a constrained optimization problem. The KKT conditions will play an important role in the interpretation of SVM in Section 1.3.2.e.

At the minimum value of $L(\mathbf{w}, b)$, we assume the derivatives with respect to b and $\mathbf{w}$ are set to zero:

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{n} \alpha_i y_i = 0$$

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = 0$$

that leads to:

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{1.24}$$

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \tag{1.25}$$

By substituting $\mathbf{w}$ into $L(\mathbf{w}, b)$ in Eq. 1.20, we obtain the dual formulation (*Wolfe dual*):

$$\underset{\boldsymbol{\alpha}}{\operatorname{argmax}} \left(\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i.\mathbf{x}_j) \right) \tag{1.26}$$

$$\textbf{s.t.} \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{1.27}$$

$$\alpha_i \geq 0 \tag{1.28}$$

The dual objective in Eq. 1.26 is quadratic in the parameters α_i. Adding the constraints in

Eq. 1.27 and 1.28, it is a constrained quadratic programming optimization problem (QP). Note that while the primal formulation is minimization, the equivalent dual formulation is maximization. It can be shown that the objective functions of both formulations reach the same value when the solution is found [CY11].
In the same spirit, considering the soft margin primal problem, it can be shown that it leads to the same formulation [CY11] (Eqs. 1.26 and 1.27), except that the Lagrange multipliers α_i are upper bounded by the trade-off C in the soft margin formulation:

$$0 \leq \alpha_i \leq C \tag{1.29}$$

The constraints in Eq. 1.29 are called the Box constraints [CY11]. From the optimal value of α_i, denoted α_i^*, it is possible to compute the weight vector $\mathbf{w}^*$ and the bias b^* at the optimality:

$$\mathbf{w}^* = \sum_{i=1}^{n} \alpha_i^* y_i \mathbf{x}_i \tag{1.30}$$

$$b^* = \sum_{i=1}^{n} (\mathbf{w}.\mathbf{x}_i - y_i) \tag{1.31}$$

At the optimality point, only a few number of datapoints have $\alpha_i^* > 0$ as shown as in Fig. 1.11. These samples are the vector supports. All other datapoints have $\alpha_i^* = 0$, and the decision function is independent of them. Thus, the representation is sparse.

From this, to classify a new sample $\mathbf{x}_j$, the decision function for a binary classification problem is:

$$f(\mathbf{x}_j) = sign(\sum_{i=1}^{n} \alpha_i^* y_i (\mathbf{x}_i.\mathbf{x}_j) + b^*) \tag{1.32}$$

1.3.2.d Kernel trick

The concept of kernels was introduced by Aizerman & Al in 1964 to design potential functions in the context of pattern recognition [ABR64]. The idea was re-introduced in 1992 by Boser & al. for Support Vector Machine (SVM) and has been received a great number of improvements and extensions to symbolic objects such as text or graphs [BGV92].

One theoretical interesting property of SVM is that it has been shown that the generalization error bound does not depend on the dimensionality T of the space [SS13]. From the dual objective in Eq. 1.26, we note that the samples $\mathbf{x}_i$ are only involves in a dot-product. Therefore, we can map these samples $\mathbf{x}_i$ into a higher dimensional hyperspace, called the feature space, through the replacement:

$$(\mathbf{x}_i.\mathbf{x}_j) \rightarrow \Phi(\mathbf{x}_i).\Phi(\mathbf{x}_j) \tag{1.33}$$

where Φ is the mapping function. The intuition behind is that for many datasets, it is not

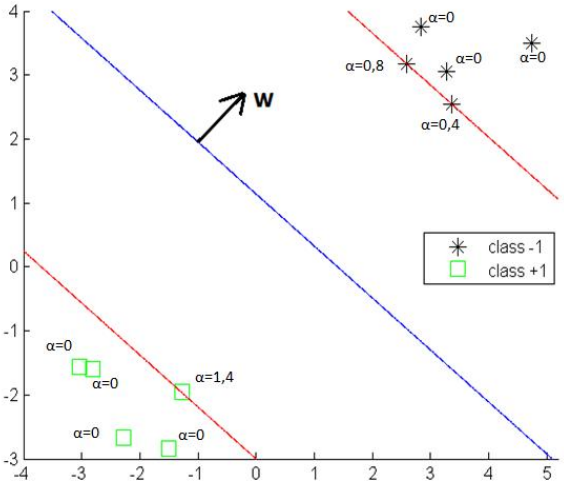


Figure 1.11: Obtained hyperplane after a dual resolution (full blue line). The 2 canonical hyperplanes (dash blue line) contains the support vectors whose $\alpha_i > 0$. Other points have their $\alpha_i = 0$ and the equation of the hyperplane is only affected by the support vectors.

possible to find a hyperplan that can separate the two classes in the input space if the problem is not linearly separable. However, by applying a transformation Φ, data might become linearly separable in a higher dimensional space (feature space). Fig. 1.12 illustrates the idea: in the original 2-dimensional space (left), the two classes can't be separated by a line. However, with a third dimension such that the $+1$ labeled points are moved forward and the -1 labeled moved back the two classes become separable.

In most of the case, the mapping function Φ does not need to be known since it will be defined by the choice of a kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i).\Phi(\mathbf{x}_j)$. We call Gram matrix G, the matrix containing all $K(\mathbf{x}_i, \mathbf{x}_j)$:

$$G = (K(\mathbf{x}_i, \mathbf{x}_j))_{1 \leq i,j \leq n} = \begin{pmatrix} K(\mathbf{x}_1, \mathbf{x}_1) & \dots & K(\mathbf{x}_1, \mathbf{x}_n) \\ \dots & & \dots \\ K(\mathbf{x}_n, \mathbf{x}_1) & \dots & K(\mathbf{x}_n, \mathbf{x}_n) \end{pmatrix}$$

Defining a kernel has to follow rules. One of these rules specifies that the kernel function has to define a proper inner product in the feature space. Mathematically, the Gram matrix has to be semi-definite positive (Mercer's theorem) [SS13]. These restricted feature spaces, containing an inner product are called Hilbert space.

Many kernels have been proposed in the literature such as the polynomial, sigmoid, exponential or wavelet kernels [SS13]. The most popular ones that we will use in our work are

[7] source: `http://users.sussex.ac.uk/~christ/crs/ml/lec08a.html`

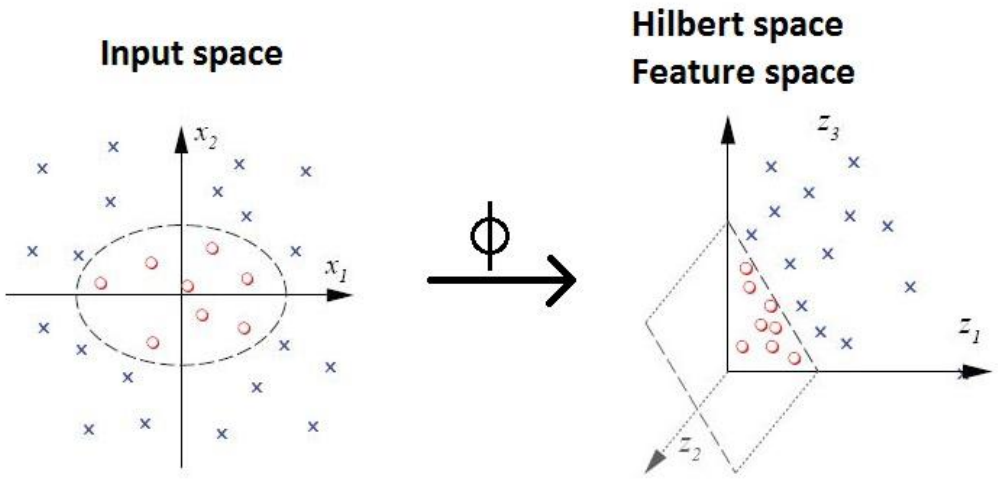


Figure 1.12: Left: in two dimensions these two classes of data are mixed together, and it is not possible to separate them by a line: the data is not linearly separable. Right: using a Gaussian kernel, these two classes of data (cross and circle) become separable by a hyperplane in feature space, which maps to the nonlinear boundary shown, back in input space.[7]

respectively the Linear and the Gaussian (or Radial Basis Function (RBF)) kernels:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i . \mathbf{x}_j \tag{1.34}$$

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\frac{||\mathbf{x}_j - \mathbf{x}_i||_2^2}{2\sigma^2}) = \exp(-\gamma||\mathbf{x}_j - \mathbf{x}_i||_2^2) \tag{1.35}$$

where $\gamma = \frac{1}{2\sigma^2}$ is the parameter of the Gaussian kernel and $||\mathbf{x}_j - \mathbf{x}_i||_2$ is the Euclidean distance between $\mathbf{x}_i$ and $\mathbf{x}_j$. Note that the Linear kernel is the identity transformation. In practice, for large scale problem (when T is high), using a Linear kernel is sufficient [FCH08].

The Gaussian kernel computed between a sample $\mathbf{x}_j$ and a support vector $\mathbf{x}_i$ is an exponentially decaying function in the input feature space. The maximum value of the kernel ($K(\mathbf{x}_i, \mathbf{x}_j)$=1) is attained at the support vector (when $\mathbf{x}_i = \mathbf{x}_j$). Then, the value of the kernel decreases uniformly in all directions around the support vector, with distance and ranges between zero and one. It can thus be interpreted as a similarity measure. Geometrically speaking, it leads to hyper-spherical contours of the kernel function as shown in Fig. 1.13 [8]. The parameter γ controls the decreasing speed of the sphere. In practice, this parameter is learnt during the training phase.

By applying the kernel trick to the soft margin formulation in Eq. 1.26, 1.27 and 1.29, the

[8] https://www.quora.com/Support-Vector-Machines/What-is-the-intuition-behind-Gaussian-kernel-in-SVM

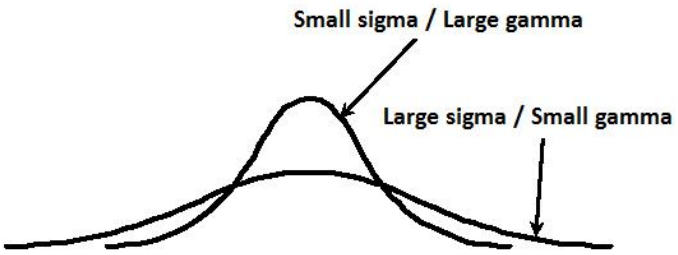


Figure 1.13: Illustration of the Gaussian kernel in the 1-dimensional input space for a small and large γ.

following optimization problem allows to learn non-linear classifiers:

$$\underset{\boldsymbol{\alpha}}{\text{argmax}} \left(\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i . \mathbf{x}_j) \right) \tag{1.36}$$

$$\textbf{s.t.} \quad \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{1.37}$$

$$0 \leq \alpha_i \leq C \tag{1.38}$$

The decision function f becomes:

$$f(\mathbf{x}_j) = sign(\sum_{i=1}^{n} \alpha_i^* y_i K(\mathbf{x}_i . \mathbf{x}_j) + b^*) \tag{1.39}$$

Note that in this case, we can't recover the weight vector $\mathbf{w}^*$. Let n_{SV} be the number of support vectors ($n_{SV} \leq n$). To recover b^*, we recall that for support vectors $\mathbf{x}_i$:

$$y_j \left(\sum_{i=1}^{n_{SV}} \alpha_i^* y_i K(\mathbf{x}_i, \mathbf{x}_j) + b^* \right) = 1 \tag{1.40}$$

From this, we can solve b^* using an arbitrarily chosen support vector $\mathbf{x}_i$:

$$b^* = \frac{1}{y_j} - \sum_{i=1}^{n_{SV}} \alpha_i^* y_i K(\mathbf{x}_i, \mathbf{x}_j) \tag{1.41}$$

1.3.2.e Complexity and interpretation

Complexity

Comment [CTD3]: réécrire + compléter avec Claude

As the objective is to provide an algorithm for both small and large datasets, let us examine the complexity of SVMs and in the computation of the Gram matrix G [BL07].

In the dual, suppose that we know which samples are not support vectors ($\alpha_i = 0$) and

which sample are bounded support vectors ($\alpha_i = C$). The R remaining support vectors are determined by a system of R linear equations. They represent the derivatives of the objective function and requires a number of operations proportional to R^3. Verifying that a vector α is a solution of the SVM problem involves computing the gradient of the dual and checking the optimality conditions. With n samples and n_{SV} support vectors, the number of operations required is equal to $n.n_{SV}$. When C gets large, few support vectors reach the upper bound, the cost is then $R^3 \approx n_{SV}^3$. The term $n.n_{SV}$ is usually larger. The final number of support vectors n_{SV} therefore is the critical component of the computational cost of solving the dual problem. Since the asymptotical number of support vectors n_{SV} grows linearly with the number of samples n, the computational cost of solving the SVM problem has both a quadratic and a cubic component. It grows at least like n^2 when C is small and n^3 when C gets large.

Computing the n^2 components of the kernel matrix $G = \{K(\mathbf{x}_i, \mathbf{x}_j)\}_{i=1}^n$ is a quadratic matter. Note that technical issues may arise in practice. For example, the kernel matrix does not fit in memory when n is large.

Interpretation in the primal

We recall that $\mathbf{x}_i$ is a univariate time series of length T. If we suppose time independency, the T observations x_{it} can be seen as attributes in the representation $\mathbb{R}^T$. Geometrically, the vector $\mathbf{w}$ represents the direction of the hyperplane (Fig. 1.14). The bias b is equal to the distance of the hyperplane to the origin point $\mathbf{x} = \mathbf{0}$[9]. The orthogonal projection of a sample $\mathbf{x}_i$ on the direction $\mathbf{w}$ is $P_{\mathbf{w}}(\mathbf{x}_i) = \mathbf{w}.\mathbf{x}_i$. In the soft margin problem, the slack variables ξ_i of the samples $\mathbf{x}_i$ that lies within the two canonical hyperplanes are equal to zero. Outside of these canonical hyperplanes, the slack variables $\xi_i > 0$ are equal to the distance to the hyperplane.

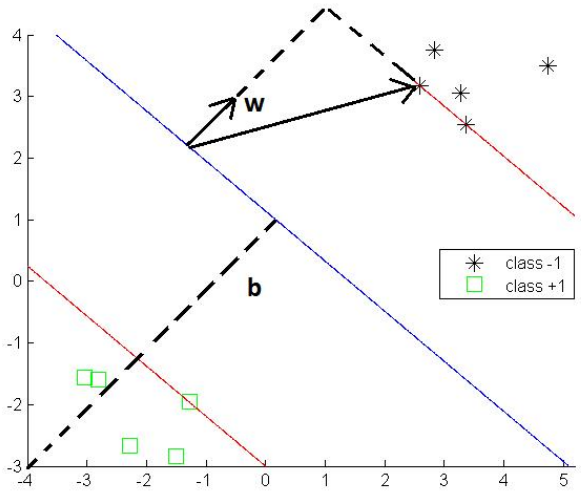


Figure 1.14: Geometric representation of SVM.

In the primal, the weight vector $\mathbf{w} = [w_1, \ldots, w_T]'$ contains as many elements as there are variables in the dataset, i.e., $\mathbf{w} \in \mathbb{R}^T$. The magnitude of each element in $\mathbf{w}$ denotes the

[9] $\mathbf{0}$ stands for the null vector: $\mathbf{0} = [0, \ldots, 0]^T$

importance of the corresponding variable for the classification problem. If the element of $\mathbf{w}$ for some variable is 0, these variables are not used for the classification problem.

In order to visualize the above interpretation of the weight vector $\mathbf{w}$, let us examine several hyperplanes $\mathbf{w}.\mathbf{x}+b=0$ shown in Fig. 1.15 with $T=2$. Figure (a) shows a hyperplane where elements of $\mathbf{w}$ are the same for both variables $\mathbf{x}_1$ and $\mathbf{x}_2$. The interpretation is that both variables contribute equally for classification of objects into positive and negative. Figure (b) shows a hyperplane where the element of $\mathbf{w}$ for $\mathbf{x}_1$ is 1, while that for $\mathbf{x}_2$ is 0. This is interpreted as that $\mathbf{x}_1$ is important but $\mathbf{x}_2$ is not. An opposite example is shown in figure (c) where $\mathbf{x}_2$ is considered to be important but $\mathbf{x}_1$ is not. Finally, figure (d) provides a 3-dimensional example ($T=3$) where an element of $\mathbf{w}$ for $\mathbf{x}_3$ is 0 and all other elements are equal to 1. The interpretation is that $\mathbf{x}_1$ and $\mathbf{x}_2$ are important but $\mathbf{x}_3$ is not.

figure à refaire

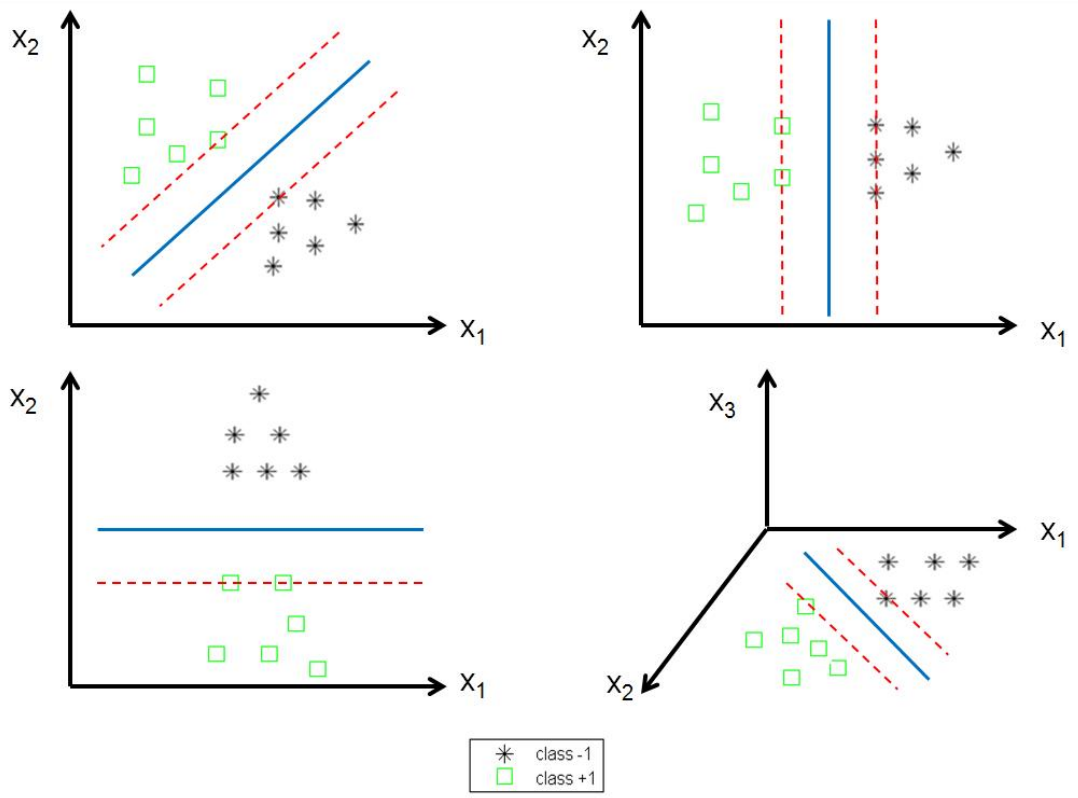


Figure 1.15: Example of several SVMs and how to interpret the weight vector $\mathbf{w}$

Another way to interpret how much a variable contributes in the vector $\mathbf{w}$ is to express the contribution in percentage. To do that, if the variables $\mathbf{X}_j$ of the time series are normalized before learning the SVM model, they evolves in the same range. Thus, the ratio $\frac{w_j}{||\mathbf{w}||_2}.100$ defines the percentage of contribution for each variable $\mathbf{X}_j$ in the SVM model.

Interpretation in the dual

As a constrained optimization, the dual form satisfies the Karush-Kuhn-Tucker (KKT) con-

ditions (Eq. 1.21, 1.22 and 1.23). We recall Eq. 1.23:

$$\alpha_i(y_i(\mathbf{w}.\mathbf{x}_i + b) - 1) = 0$$

From this, for every datapoint $\mathbf{x}_i$, either $\alpha_i^* = 0$ or $y_i(\mathbf{w}.\mathbf{x}_i + b) = 1$. Any datapoint with $\alpha_i^* = 0$ do not appear in the sum of the decision function f in Eq. 1.32 or 1.39. Hence, they play no role for the classification decision of a new sample $\mathbf{x}_j$. The others $\mathbf{x}_i$ such that $\alpha_i^* > 0$ corresponds to the support vector. Looking at the distribution of α_i^* allows also to have either a better understanding of the datasets, or either to detect outliers. The higher is the coefficient α_i^* for a sample $\mathbf{x}_i$, the more the sample $\mathbf{x}_i$ impacts on the decision function f. However, unusual high value of α_i^* among the samples can lead to two interpretations: either this point is a critical point to the decision, either this point is an outlier. In the soft margin formulation, by constraining α_i^* to be inferior to C (Box constraints) the effect of outliers can be reduced and controlled.

1.3.2.f Extensions of SVM

SVM has received many interests in recent years. Many extensions has been developed such as ν-SVM, asymmetric soft margin SVM or multiclass SVM [KU02]; [CS01]. One interesting extension is the extension of Support Vector Machine to regression problems, also called Support Vector Regression (SVR). The objective is to find a linear regression model $f(\mathbf{x}) = \mathbf{w}.\mathbf{x} + b$. To preserve the property of sparseness, the idea is to consider an ϵ-insensitive error function. It gives zero error if the absolute difference between the prediction $f(\mathbf{x}_i)$ and the target y_i is less than ϵ where $\epsilon > 0$ penalize samples that are outside of a ϵ-tube as shown as in Fig. 1.16.

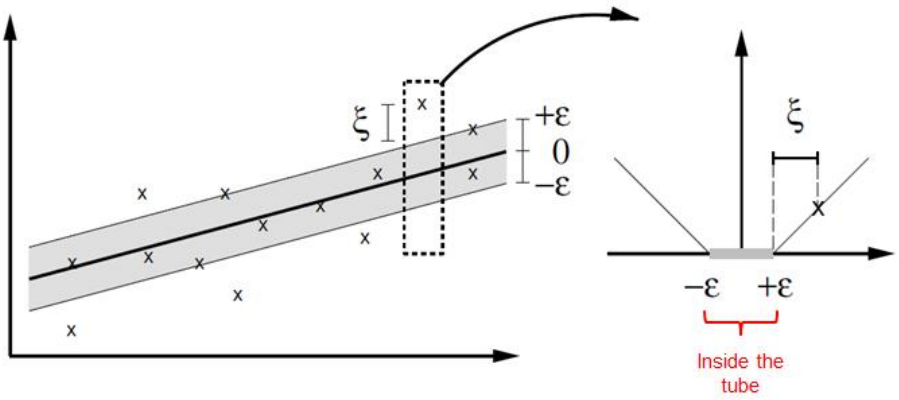


Figure 1.16: Illustration of SVM regression (left), showing the regression curve with the ϵ-insensitive "tube" (right). Samples $\mathbf{x}_i$ above the ϵ-tube have $\xi_1 > 0$ and $\xi_1 = 0$, points below the ϵ-tube have $\xi_2 = 0$ and $\xi_2 > 0$, and points inside the ϵ-tube have $\xi = 0$.

An example of ϵ-insensitive error function E_ϵ is given by,

$$E_\epsilon(f(\mathbf{x}_i) - y_i) = \begin{cases} 0 & \text{if} \quad |f(\mathbf{x}_i) - y_i| < \epsilon \\ |f(\mathbf{x}_i) - y_i| - \epsilon & \text{otherwise} \end{cases} \tag{1.42}$$

The soft margin optimization problem in its primal form is formalized as:

$$\underset{\mathbf{w},b}{\operatorname{argmin}} \left(\overbrace{\frac{1}{2}||\mathbf{w}||_2^2}^{Regularization} + C \overbrace{\sum_{i=1}^{n}(\xi_{i_1} + \xi_{i_2})}^{Loss} \right) \tag{1.43}$$

s.t. $\forall i = 1 \dots n$:

$$y_i - (\mathbf{w}.\mathbf{x}_i + b) \geq \epsilon - \xi_{i_1} \tag{1.44}$$

$$(\mathbf{w}.\mathbf{x}_i + b) - y_i \geq \epsilon - \xi_{i_2} \tag{1.45}$$

$$\xi_{i_1} \geq 0 \tag{1.46}$$

$$\xi_{i_2} \geq 0 \tag{1.47}$$

The slacks variables are divided into 2 slacks variables, one for samples above the decision function f (ξ_{i_1}), and one for samples under the decision function f (ξ_{i_2}). As for SVM, it is possible to have a dual formulation:

$$\underset{\boldsymbol{\alpha}}{\operatorname{argmax}} \left(\sum_{i=1}^{n} y_i(\alpha_{i_1} - \alpha_{i_2}) - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(\alpha_{i_1} - \alpha_{i_2})(\alpha_{j_1} - \alpha_{j_2})(\mathbf{x}_i.\mathbf{x}_j) \right) \tag{1.48}$$

s.t. $\forall i = 1 \dots n$:

$$\sum_{i=1}^{n} \alpha_{i_1} = \sum_{i=1}^{n} \alpha_{i_2} \tag{1.49}$$

$$0 \leq \alpha_{i_1} \leq C \tag{1.50}$$

$$0 \leq \alpha_{i_2} \leq C \tag{1.51}$$

As in SVM, we obtain three possible decision functions for a new sample $\mathbf{x}_j$, respectively in its primal, dual, and non-linear form:

$$f(\mathbf{x}_j) = \mathbf{w}.\mathbf{x}_j + b \tag{1.52}$$

$$f(\mathbf{x}_j) = \sum_{i=1}^{n}(\alpha_{i_1}^* - \alpha_{i_2}^*)(\mathbf{x}_i.\mathbf{x}_j) + b \tag{1.53}$$

$$f(\mathbf{x}_j) = \sum_{i=1}^{n}(\alpha_{i_1}^* - \alpha_{i_2}^*)K(\mathbf{x}_i.\mathbf{x}_j) + b \tag{1.54}$$

More informations about the calculation development can be found in [Bis06].

1.3.3 Other classification algorithms

Partie non encore rédigée. A faire à la fin.

- Positionner les travaux par rapport aux autres méthodes d'apprentissage supervisé
- S'intéresser au Deep neural network (à la mode en ce moment)
- RVM, Decision Tree,
- Ne pas trop développer
- Dans notre cas, on ne s'intéressera pas à ce type d'algorithmes (type deep learning) car il ne repose pas sur une notion de distance et les features qui sont trouvés ne sont pas interprétables

1.4 Conclusion of the chapter

To make the classification or regression of time series, a commonly hypothesis is to consider time series as static data and then to apply classical machine learning algorithms, such as a k-Nearest Neighbors (k-NN) or Support Vector Machine (SVM) approach. For that, the practitioner has to be careful on the design of his learning framework: data must be separated into a training and testing set, data have to be normalized depending on their distributions, cross-validation must be operated on the training set to learn the best fitting of the hyper-parameters and finally, performance metrics and statistical tests should be used to compare the performances of different classifiers.

In the following, we consider the k-NN as our classifier. The SVM will be used in our work for its generalization properties thanks to the large margin concept. A key aspect in k-NN relies on the comparison of time series through metrics. Assuming that time series can be reduced to flat data may be too restrictive. Under such hypothesis and using a standard Euclidean distance, time series are only compared on their amplitude at the same time. However, time series are more complex. They may exhibit similar behavior or share similar frequential spectrum. Thus, there is a need to consider time series as an ordered object and to define adapted metrics for time series.

CHAPTER 2

Time series metrics and Metric Learning

Sommaire

In this chapter, we review different metrics for time series. In classification problems, time series are expected to be similar if they belong to the same class. The concept of similarity among time series is directly linked to the concept of metrics.

In the following, we consider time series as an object. They may be compared either on all their observations x_{it}, a part of them or in a window. We first recall the properties of a metric. Then, we review three types of metrics (amplitude-based, behavior-based, frequential-based) and kernels adapted to time series. As real time series are subjected to varying delays, we recall the concept of alignment and dynamic programming. We show how these metrics can be used to define either metrics that can capture local characteristics of time series, or metrics that combine multiple modalities. Finally, as k-NN performances is directly impacted by the choice of the metric, we review some insights on Metric Learning investigated in the case of static data.

2.1 Properties of a metric

Defining and evaluating adapted metrics for time series has become an active area of research for a wide variety of problems in Machine Learning [Din+08]; [Naj+12]. In the following, we suppose that time series have the same lengths T and have been sampled at the same sampling frequency f_e. Let $\mathbf{x}_i = (x_{i1}, x_{i2}, ..., x_{iT})$ and $\mathbf{x}_j = (x_{i1}, x_{i2}, ..., x_{iT})$ be two univariate time series of length T.

A mapping $D : \mathbb{R}^T \times \mathbb{R}^T \rightarrow \mathbb{R}^+$ over a vector space $\mathbb{R}^T$ is called a metric or a distance if for all vectors $\forall \mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_l$, it satisfies the properties:

1. $D(\mathbf{x}_i, \mathbf{x}_j) \geq 0$ (positivity)
2. $D(\mathbf{x}_i, \mathbf{x}_j) = D(\mathbf{x}_j, \mathbf{x}_i)$ (symmetry)
3. $D(\mathbf{x}_i, \mathbf{x}_j) = 0 \Leftrightarrow \mathbf{x}_i = \mathbf{x}_j$ (distinguishability)
4. $D(\mathbf{x}_i, \mathbf{x}_j) D(\mathbf{x}_j, \mathbf{x}_l) \leq D(\mathbf{x}_i, \mathbf{x}_l)$ (triangular inequality)

A mapping D that satisfies at least properties 1, 2, 3 is called a dissimilarity, and the one that satisfies at least properties 1, 2, 4 a pseudo-metric. Note that for a metric, a dissimilarity and a pseudo metric, if a time series $\mathbf{x}_i$ is expected to be closer to $\mathbf{x}_j$ than to $\mathbf{x}_l$, then $D(\mathbf{x}_i, \mathbf{x}_j) \leq D(\mathbf{x}_i, \mathbf{x}_l)$. On the contrary, the mapping D is called a similarity when the time series $\mathbf{x}_i$ is expected to be closer to $\mathbf{x}_j$ than to $\mathbf{x}_l$ and then $D(\mathbf{x}_i, \mathbf{x}_j) \geq D(\mathbf{x}_i, \mathbf{x}_l)$. To simplify the discussion in the following, we refer to pseudo-metric and dissimilarity as metrics, pointing out the distinction only when necessary.

2.2 Unimodal metrics for time series

A large number of distance measures have been proposed in the literature [MV14]. Contrary to static data, time series may exibit modalities and specificities due to their temporal nature (e.g., value, shape, frequency, delay, temporal locality). In this section, we review 3 categories of time series metrics used in our work: amplitude-based, frequential-based and behavior-based.

2.2.1 Amplitude-based metrics

The most usual comparison measures are amplitude-based metrics, where time series are compared in the temporal domain on their amplitudes regardless of their behaviors or frequential characteristics. Among these metrics, there are the commonly used Euclidean distance that

compares elements observed at the same time [Din+08]:

$$d_E(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{t=1}^{T} (x_{it} - x_{jt})^2} \tag{2.1}$$

Note that the Euclidean distance is a particular case of the Minkowski L_p norm ($p = 2$). An other amplitude-based metric is the Mahalanobis distance [PL12]:

$$d_M(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i - \mathbf{x}_j)' \mathbf{M} (\mathbf{x}_i - \mathbf{x}_j) \tag{2.2}$$

In particular, when $\mathbf{M}$ is a diagonal matrix, the previous formula becomes:

$$M = \begin{pmatrix} M_1 & & & \\ & \dots & & 0 \\ & & M_t & \\ & 0 & & \dots \\ & & & M_T \end{pmatrix} \tag{2.3}$$

$$d_M(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{t=1}^{T} M_t (x_{it} - x_{jt})^2} \tag{2.4}$$

In practice, the M_t coefficients are often set as the variance of the value on the corresponding dimension. Intuitively, each dimension difference $(x_{it} - x_{jt})$ is weighed by a factor M_t. It is also known as the weighted Euclidean distance [McN02]. In the following of the work, we consider the standard Euclidean distance d_E as the amplitude-based distance d_A.

Comment [MR4]: Plus les données sont imprécises, moins elle est importante

In the example of Fig. 2.1, the aim is to determined which time series ($\mathbf{x}_2$ or $\mathbf{x}_3$) is the closest to $\mathbf{x}_1$. The amplitude-based distance d_A states that $\mathbf{x}_2$ is closer to $\mathbf{x}_1$ than $\mathbf{x}_3$ since $d_A(\mathbf{x}_1, \mathbf{x}_2) = 7.8816 < d_A(\mathbf{x}_1, \mathbf{x}_3) = 31.2250$.

2.2.2 Frequential-based metrics

The second category, commonly used in signal processing, relies on comparing time series based on their frequential properties (e.g. Fourier Transform, Wavelet, Mel-Frequency Cepstral Coefficients [SS12]; [TC98]; [BM67]). In our work, we limit the frequential comparison to Discrete Fourier Transform [Lhe+11], but other frequential properties can be used as well. Thus, for time series comparison, first the time series $\mathbf{x}_i$ are transformed into their Fourier representation $\tilde{\mathbf{x}}_i = [\tilde{x}_{i1}, ..., \tilde{x}_{iF}]$, with $\tilde{x}_{if}$ the complex component at frequential index f. The Euclidean distance is then used between their respective complex number modules $\tilde{x}_{if}$, noted $|\tilde{x}_{if}|$:

$$d_F(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{f=1}^{F} (|\tilde{x}_{if}| - |\tilde{x}_{jf}|)^2} \tag{2.5}$$

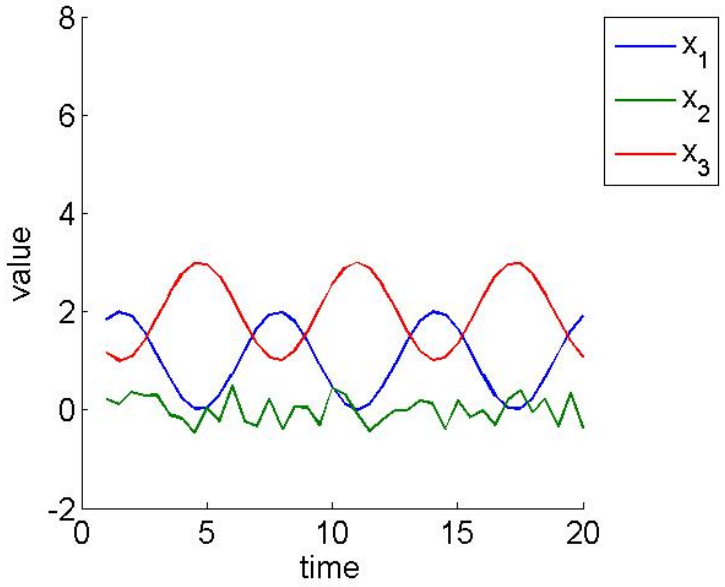
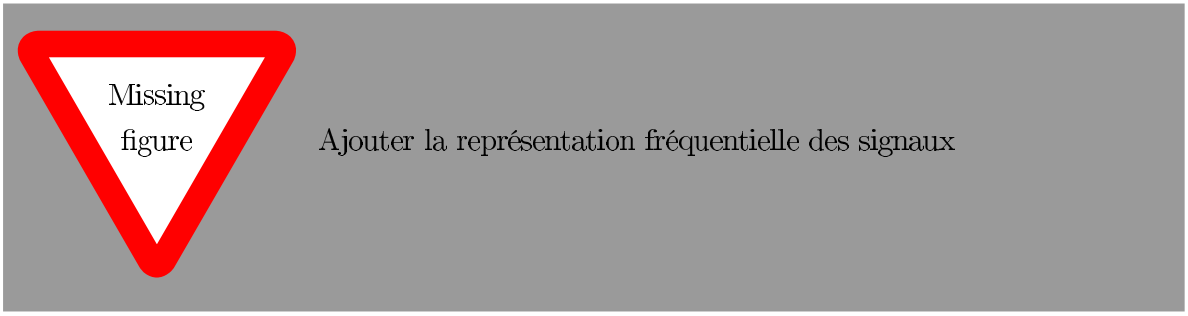


Figure 2.1: 3 toy time series. Time series in blue and red are two sinusoïdal signals. Time series in green is a random signal.

In the example of Fig. 2.1, the frequential-based distance d_F states that the time series $\mathbf{x}_3$ is closer to $\mathbf{x}_1$ than $\mathbf{x}_2$ since $d_F(\mathbf{x}_1, \mathbf{x}_3) = 0.8519 < d_F(\mathbf{x}_1, \mathbf{x}_2) = 0.9250$. This can be illustrated in the Frequency domain (Fig. **??**)



2.2.3 Behavior-based metrics

The third category of metrics aims to compare time series based on their shape or behavior despite the range of their amplitudes. By time series of similar behavior, it is generally intended that for all temporal window $[t, t']$, they increase or decrease simultaneously with the same growth rate. On the contrary, they are said of opposite behavior if for all $[t, t']$, if one time series increases, the other one decreases and (vise-versa) with the same growth rate in absolute value. Finally, time series are considered of different behaviors if they are not similar, nor opposite. Many applications refer to the Pearson correlation [AT10]; [Ben+09] for

behavior comparison. A generalization of the Pearson correlation is introduced in [DCA11]:

$$cort_r(\mathbf{x}_i, \mathbf{x}_j) = \frac{\sum (x_{it} - x_{it'})(x_{jt} - x_{jt'})}{\sqrt{\sum (x_{it} - x_{it'})^2}\sqrt{\sum (x_{jt} - x_{jt'})^2}} \tag{2.6}$$

where $|t-t'| \leq r$, $r \in [1, ..., T-1]$. The parameter r can be tuned or fixed a priori. It measures the importance of noise in data. For non-noisy data, low orders r is generally sufficient. For noisy data, the practitioner can either use de-noising data technics (Kalman or Wiener filtering [Kal60]; [Wie42]), or fix a high order r.

The temporal correlation *cort* computes the sum of growth rate between $\mathbf{x}_i$ and $\mathbf{x}_j$ between all pairs of values observed at $[t, t']$ for $t' \leq t+r$ (r-order differences). The value $cort_r(\mathbf{x}_i, \mathbf{x}_j) = 1$ means that $\mathbf{x}_i$ and $\mathbf{x}_j$ have similar behavior. The value $cort_r(\mathbf{x}_i, \mathbf{x}_j) = -1$ means that $\mathbf{x}_i$ and $\mathbf{x}_j$ have opposite behavior. Finally, $cort_r(\mathbf{x}_i, \mathbf{x}_j) = 0$ expresses that their growth rates are stochastically linearly independent (different behaviors).

When $r = 1$, Eq. (2.6) leads to the temporal correlation coefficient *cort* [DCA11]. When $r = T - 1$, it leads to the Pearson correlation. As $cort_r$ is a similarity measure, it can be transformed into a dissimilarity measure:

$$d_B(\mathbf{x}_i, \mathbf{x}_j) = \frac{1 - cort_r(\mathbf{x}_i, \mathbf{x}_j)}{2} \tag{2.7}$$

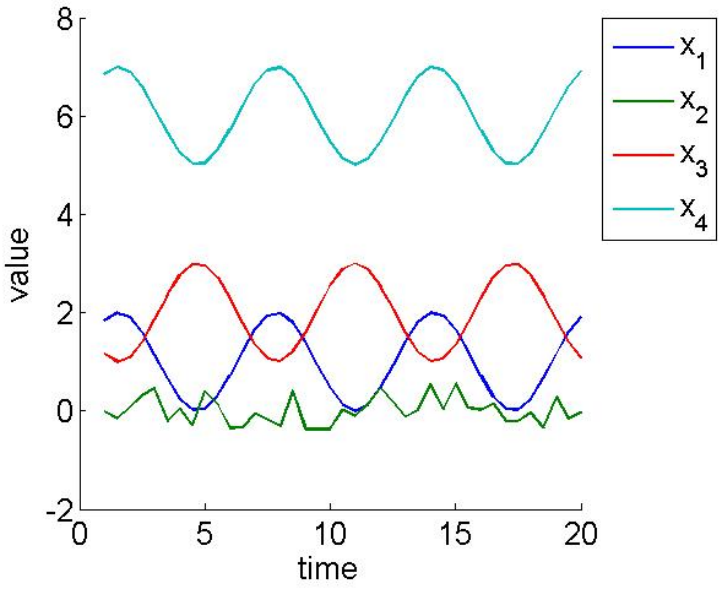


Figure 2.2: The signal from Fig. 2.1 and a signal $\mathbf{x}_4$ which is signal $\mathbf{x}_1$ and an added translation. Based on behavior comparison, $\mathbf{x}_4$ is the closest to $\mathbf{x}_1$.

Now considering Fig. 2.2

$$d_B(\mathbf{x}_1, \mathbf{x}_2) = 0.477$$
$$d_B(\mathbf{x}_1, \mathbf{x}_3) = 1$$
$$d_B(\mathbf{x}_1, \mathbf{x}_4) = 0$$

2.2.4 Other metrics and Kernels for time series

A faire à la fin, pas urgent

- Il existe dans la littérature de nombreuses autres métriques pour les séries temporelles (laisser la porte ouverte).
- Certaines métriques sont utilisées dans le domaine temporelle
- D'autres métriques sont utilisés dans d'autres représentations (Wavelet, etc.)
- Certaines combinent la représentation temporelles et fréquentielles (Représentation spectrogramme en temps-fréquence)
- Se baser sur l'article "TSclust : An R Package for Time Series Clustering".
- Fermer le cadre : dans la suite de notre travail, on ne va pas les utiliser mais elles pourront être intégrées dans le framework qui suivra au chapitre suivant

2.3 Time series alignment and dynamic programming approach

In some applications, time series needs to be compared at different time t (i.e. energy data [Naj+12]) whereas in others, comparing time series on the same time t is essential (i.e. gene expression [DCN07]). When time series are asynchronous (i.e. varying delays or dynamic changes), they must be aligned before any analysis process. The asynchronous effects can be of various natures: time shifting (phase shift in signal processing), time compression or time dilatation. For example, in the case of voice recognition (Fig. 2.3), it is straightforward that a same sentence said by two different speakers will produce different time series: one speaker may speak faster than the other; one speaker may take more time on some vowels, etc.

Comment [MR5]: Modifier figure. enlever 'one' et mettre la même échelle temporelle

To cope with delays and dynamic changes, dynamic programming approach has been introduced [BC94]. An alignment $\boldsymbol{\pi}$ of length $|\boldsymbol{\pi}| = m$ between two time series $\mathbf{x}_i$ and $\mathbf{x}_j$ of length T is defined as the set of m ($T \leq m \leq 2T-1$) couples of aligned elements of $\mathbf{x}_i$ to m elements of $\mathbf{x}_j$:

$$\boldsymbol{\pi} = ((\pi_i(1), \pi_j(1)), (\pi_i(2), \pi_j(2)), \ldots, (\pi_i(m), \pi_j(m))) \tag{2.8}$$

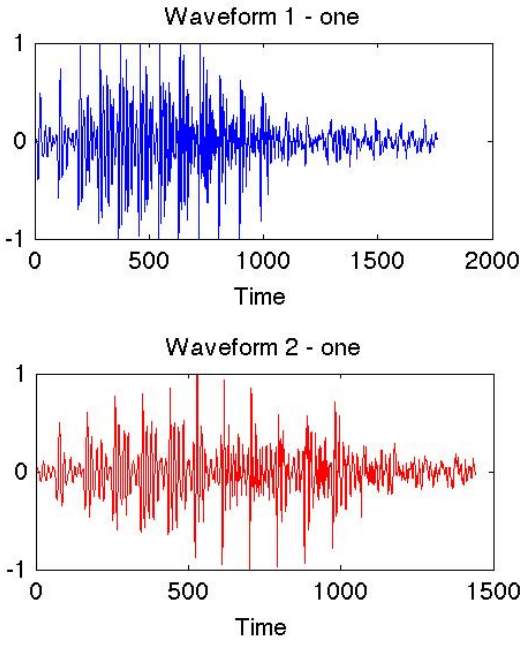


Figure 2.3: Example of a same sentence said by two different speakers. Time series are shifted, compressed and dilatated in the time.

where the applications π_i and π_j defined from $\{1, ..., m\}$ to $\{1, ..., T\}$ obey the following boundary monotonicity conditions:

$$1 = \pi_i(1) \leq \pi_i(2) \leq ... \leq \pi_i(m) = T \tag{2.9}$$
$$1 = \pi_j(1) \leq \pi_j(2) \leq ... \leq \pi_j(m) = T \tag{2.10}$$

$\forall l \in \{1, ..., m\}$,

$$\pi_i(l+1) \leq \pi_i(l) + 1 \tag{2.11}$$
and $$\pi_j(l+1) \leq \pi_j(l) + 1 \tag{2.12}$$
and $$(\pi_i(l+1) - \pi_i(l)) - (\pi_j(l+1) - \pi_j(l)) \geq 1. \tag{2.13}$$

Intuitively, an alignment $\boldsymbol{\pi}$ defines a way to associate elements of two time series. Alignments can be described by paths in the $T \times T$ grid that crosses the elements of $\mathbf{x}_i$ and $\mathbf{x}_j$ (Fig. 2.4). We denote $\boldsymbol{\pi}$ a valid alignment and A, the set of all possible alignments between $\mathbf{x}_i$ and $\mathbf{x}_j$ ($\boldsymbol{\pi} \in A$). To find the best alignment $\boldsymbol{\pi}^*$ between two time series $\mathbf{x}_i$ and $\mathbf{x}_j$, the Dynamic Time Warping (DTW) algorithm has been proposed [KR04]; [SC].

DTW requires to choose a cost function φ to be optimised, such as a dissimilarity function (d_A, d_B, d_F, etc.). Classical DTW uses the Euclidean distance d_A (Eq. 2.1) as the cost

function [BC94]. The warp path $\boldsymbol{\pi}$ is optimized for the chosen cost function φ:

$$\boldsymbol{\pi}^* = \underset{\boldsymbol{\pi} \in A}{\operatorname{argmin}} \frac{1}{|\boldsymbol{\pi}|} \sum_{(t,t') \in \boldsymbol{\pi}} \varphi(x_{it}, x_{jt'}) \tag{2.14}$$

When the cost function φ is a similarity measure, the optimization involves maximization instead of minimization. When other constraints are applied on $\boldsymbol{\pi}$, Eq. (2.14) leads to other variants of DTW (Sakoe-Shiba [SC78], Itakura parallelogram [RJ93]). Finally, the warped signals $\mathbf{x}_{i,\boldsymbol{\pi}}$ and $\mathbf{x}_{j,\boldsymbol{\pi}}$ are defined as:

$$\mathbf{x}_{i,\boldsymbol{\pi}} = (x_{i\pi_i(1)}, ..., x_{i\pi_i(m)}) \tag{2.15}$$

$$\mathbf{x}_{j,\boldsymbol{\pi}} = (x_{j\pi_j(1)}, ..., x_{j\pi_j(m)}) \tag{2.16}$$

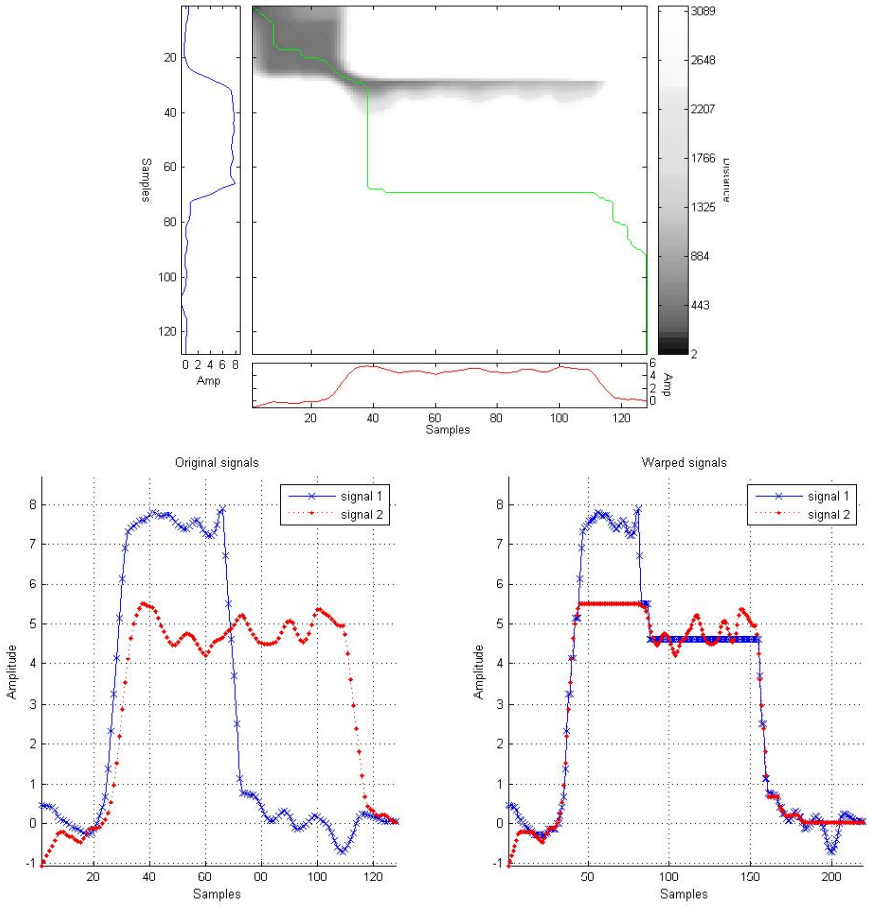


Figure 2.4: Example of DTW grid between 2 time series $\mathbf{x}_i$ and $\mathbf{x}_j$ (top) and the signals before and after warping (bottom). On the DTW grid, the two signals can be represented on the left and bottom of the grid. The optimal path $\boldsymbol{\pi}^*$ is represented in green line and show to associate elements of $\mathbf{x}_i$ to element of $\mathbf{x}_j$. Background show in grey scale the value of the considered metric (amplitude-based distance d_A in classical DTW)

The previous metric (amplitude-based d_A, behavior-based d_B) can be then computed on the warped signals $\mathbf{x}_{i,\boldsymbol{\pi}^*}$ and $\mathbf{x}_{j,\boldsymbol{\pi}^*}$. In the following, we suppose that the best alignment $\boldsymbol{\pi}^*$ is found. For simplification purpose, we refer $\mathbf{x}_{i,\boldsymbol{\pi}^*}$ and $\mathbf{x}_{j,\boldsymbol{\pi}^*}$ as $\mathbf{x}_i$ and $\mathbf{x}_j$.

2.4 Multi-scale comparison

In some applications, time series may exhibit similarities among the classes based on local patterns in the signal. Fig. 2.5 illustrates a toy example (UMD dataset) in which the time series of different classes seems to be similar on a global scale. However, at a more locally scale, a characteristic bell (up or down) at the beginning or at the end of the time series allows to differentiate the classes. Also, in massive time series datasets, computing the metric on all time series elements x_{it} might become time consuming. Computing the metric on a smaller part of the signal and not all the time series elements makes the metric computation faster.

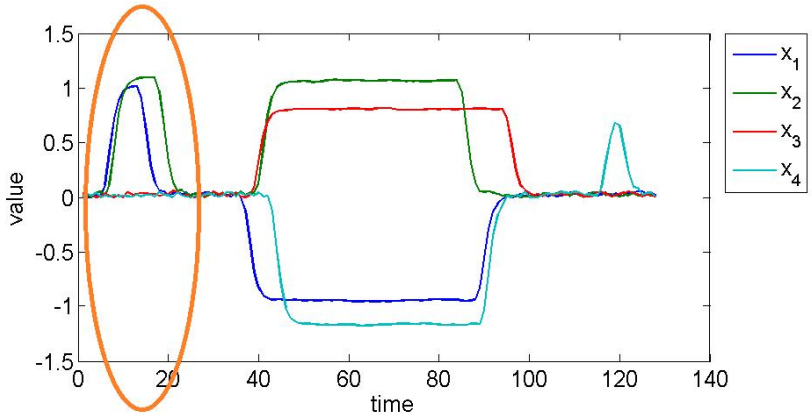


Figure 2.5: Example of 4 time series from the BME dataset, made of 3 classes : Begin, Middle and End. The 'Up' class has a characteristic bell at the beginning of the time series. The 'End' class has a characteristic bell at the end of the time series. The 'Middle' class has no characteristic bell. Orange circle show the region of interest of these bells for the class 'Begin'. This region is local and standard global metric fails to show these characteristics.

Localizing patterns of interest in huge time series datasets has become an active area of search in many applications including diagnosis and monitoring of complex systems, biomedical data analysis, and data analysis in scientific and business time series . A large number of ref methods have been proposed covering the extraction of local features from temporal windows [BC94] or the matching of queries according to a reference sequence [FRM94]. Our work will focus on the computation of local metrics.

It can be noted that the distance measures (d_A[1], d_F, d_B) in Eqs. 2.1, 2.5 and 2.7 implies systematically the total time series elements x_{it} and thus, restricts the distance measures to capture local temporal differences. In our work, we provide a multi-scale framework for time

[1]We recall that d_A is the Euclidean distance d_E in our work.

series comparison. Many methods exist in the literature such as the sliding window or the dichotomy ref. We detailed here the latter one.

A multi-scale description is obtained by repeatedly segmenting a time series expressed at a given temporal scale to induce its description at a more locally level. Many approaches have been proposed assuming fixed either the number of the segments or their lengths. In our work, we fix the number of segments and consider a binary segmentation. Let $I = [a; b]$ be a temporal interval of size $(b - a)$. For a strict division (no overlapping), the dichotomy process divide I into two equal intervals at $\frac{b-a}{2}$: the left one I_L and the right I_R one. We add a parameter α that allows to overlap the two intervals I_L and I_R, covering discriminating subsequences in the central region of I (around $\frac{b-a}{2}$) and thus avoiding 'border effects':

$$I = [a; b] \tag{2.17}$$
$$I_L = [a; a + \alpha(b - a)] \tag{2.18}$$
$$I_R = [a - \alpha(b - a); b] \tag{2.19}$$

For $\alpha = 0.6$, the overlap covers 10% of the size of the interval I. A multi-scale description is then obtained on computing the usual time series metrics (d_A, d_B, d_F) on the resulting segments I, I_L and I_R and by repeating the process on I_L and I_R. For a multi-scale amplitude-based comparison based on binary segmentation, Figure 2.6 shows the set of involved amplitude-based measures d_A^{Is}:

$$d_A^{Is}(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{t \in Is} (x_{it} - x_{jt})^2} \tag{2.20}$$

The local behaviors- and frequential- based measures d_B^{Is} and d_F^{Is} are obtained similarly.

2.5 Combined metrics for time series

In most classification problems, it is not known a priori if time series of a same class exhibits same characteristics based on their amplitude, behavior or frequential components alone. In some cases, several components (amplitude, behavior and/or frequential) may be implied.

A first technic considers a classifier for each p metric and combines the decision of the p resulting classifiers. This methods is referred as post-fusion ref, not considered in our work. Other propositions show the benefit of involving both behavior and amplitude components through a combination function. They combines the unimodal metrics together to obtain a single metric used after that in a classifier. This is called pre-fusion. The most classical combination functions combines the unimodal metrics (mainly d_A and d_B) through linear and geometric functions:

$$D_{Lin}(\mathbf{x}_i, \mathbf{x}_j) = \alpha d_B(\mathbf{x}_i, \mathbf{x}_j) + (1 - \alpha) d_A(\mathbf{x}_i, \mathbf{x}_j) \tag{2.21}$$
$$D_{Geom}(\mathbf{x}_i, \mathbf{x}_j) = (d_B(\mathbf{x}_i, \mathbf{x}_j))^{\alpha} (d_A(\mathbf{x}_i, \mathbf{x}_j))^{1-\alpha} \tag{2.22}$$

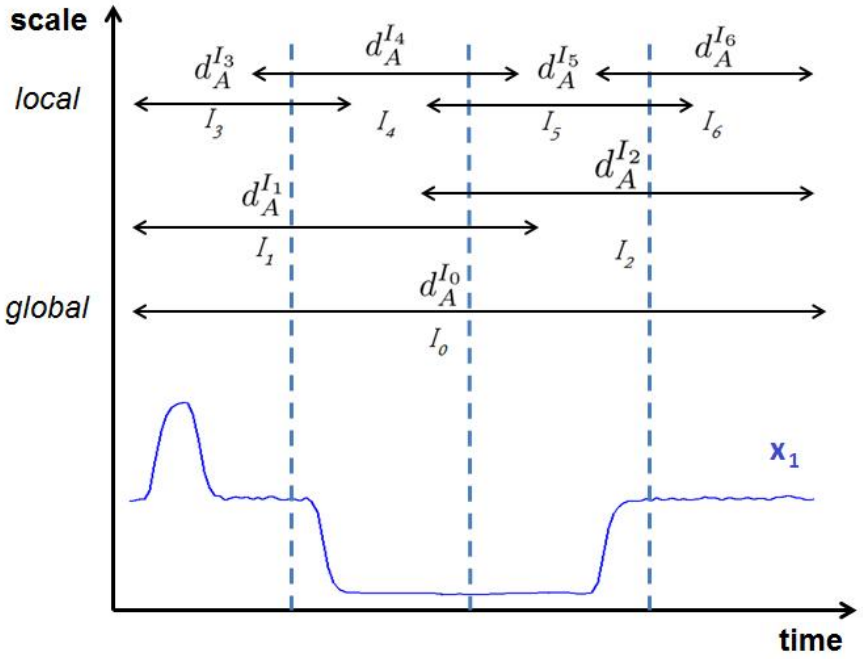


Figure 2.6: Multi-scale amplitude-based measures d_A^{Is}

where $\alpha \in [0;1]$ defines the trade-off between the amplitude d_A and the behavior d_B components, and is thus application dependent. In general, it is learned through a grid search procedure. Without being restrictive, these combinations can be extended to take into account more unimodal metrics.
More specific work on d_A and *cort* propose to combine the two unimodal metrics through a sigmoid combination function [DCA11]:

$$D_{Sig}(\mathbf{x}_i, \mathbf{x}_j) = \frac{2d_A(\mathbf{x}_i, \mathbf{x}_j)}{1 + \exp(\alpha cort_r(\mathbf{x}_i, \mathbf{x}_j))} \tag{2.23}$$

where α is a parameter that defines the compromise between behavior and amplitude components. When α is fixed to 0, the metric only includes the value proximity component. For $\alpha \geq 6$, the metric completely includes the behavior proximity component.

Fig.2.7 illustrates the value of the resulting combined metrics (D_{Lin}, D_{Geom} and D_{Sig}) in 2-dimensional space using contour plots for different values of the trade-off α. For small value of α (=0), the three metrics only includes d_A. For high value of α (=1), D_{Lin} and D_{Geom} only includes d_B. For $\alpha = 6$, D_{Sig} doesn't include completely *cort*. Note that these combinations are fixed and defined independently from the analysis task at hand. Moreover, in the case of D_{Sig}, only two variables are taking into account in these combined metrics and the component $cort_r$ can be seen as a penalizing factor of d_A. It doesn't represent a real compromise between value and behavior components. Finally, by adding metrics, the grid search to find the best parameters can become time consuming.

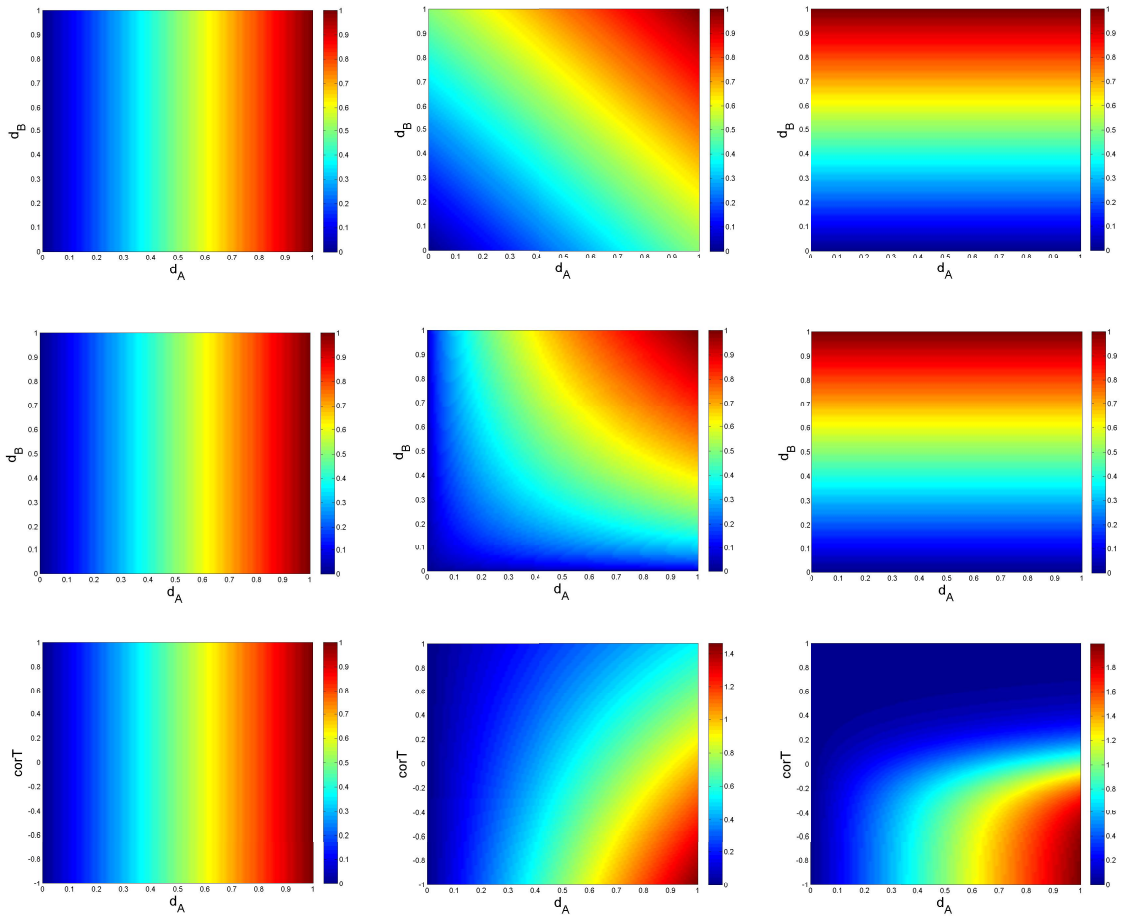


Figure 2.7: Contour plot of the resulting combined metrics: D_{Lin} (1^{st} line), D_{Geom} (2^{nd} line) and D_{Sig} (3^{rd} line), for different value of α (D_{Sig}: $\alpha = 0; 1; 6$ and D_{Lin} and D_{Geom}: $\alpha = 0; 0.5; 1$). For D_{Sig}, the first and second dimensions are respectively the amplitude-based metrics d_A and the temporal correlation $corT$; for D_{Lin} and D_{Geom}, they correspond to d_A and the behavior-based metric d_B.

Comment [MR6]: Ajouter sur la figure la valeur de α et le nom des métriques pour une meilleure visibilité

2.6 Metric Learning

The problem in this PhD is to define a metric that can be learnt in order to optimize the performance of the k-NN classifier. In this section, we first make a state of the art of Metric Learning propositions. Then, we focus on the framework proposed by Weinberger & Saul for Large Margin Nearest Neighbor (LMNN) classification [WS09].

2.6.1 State of the art

In the case of static data, many work have demonstrated that k-NN classification performances depends highly on the considered metric and can be improved by learning an appropriate metric [She+02]; [Gol+04]; [CHL05]. Metric Learning can be defined as a process that aims to learn a distance from labeled examples by making closer samples that are expected to be similar, and far away those expected to be dissimilar.

A faire, le positionnement du Metric Learning avec l'aide du papier PRL

2.6.2 Large Margin Nearest Neighbors (LMNN)

Let $\mathbf{X} = \{\mathbf{x}_i, y_i\}_{i=1}^{N}$ be a set of N static vector samples, $\mathbf{x}_i \in \mathbb{R}^p$, p being the number of descriptive features and y_i the class labels. Weinberger & Saul proposed in [WS09] an approach to learn a dissimilarity metric D for a large margin k-NN in the case of static data.

Large Margin Nearest Neighbor (LMNN) approach is based on two intuitions: first, each training sample $\mathbf{x}_i$ should have the same label y_i as its k nearest neighbors; second, training samples with different labels should be widely separated. For this, the concept of **target** and **imposters** for each training sample $\mathbf{x}_i$ is introduced. Target neighbors of $\mathbf{x}_i$, noted $j \rightsquigarrow i$, are the k closest $\mathbf{x}_j$ of the same class ($y_j = y_i$), while imposters of $\mathbf{x}_i$, denoted, $l \nrightarrow i$, are the $\mathbf{x}_l$ of different class ($y_l \neq y_i$) that invade the perimeter defined by the farthest targets of $\mathbf{x}_i$. Mathematically, for a sample $\mathbf{x}_i$, an imposter $\mathbf{x}_l$ is defined by an inequality related to the targets $\mathbf{x}_j$: $\forall l, \exists j \in j \rightsquigarrow i/$

$$D(\mathbf{x}_i, \mathbf{x}_l) \leq D(\mathbf{x}_i - \mathbf{x}_j) + 1 \tag{2.24}$$

Geometrically, an imposter is a sample that invades the target neighborhood plus one unit margin as illustrated in Fig. 2.8. Note that the target neighborhood is defined with respect to an initial metric. Without prior knowledge, L2 norm is often used. Metric Learning by LMNN aims to minimize the number of impostors invading the target neighborhood. By adding a margin of safety of one, the model is ensured to be robust to small amounts of noise in the training sample (large margin). The learned metric D pulls the targets and pushes the imposters as shown in Fig. 2.8.

LMNN approach learns a Mahalanobis distance D for a robust k-NN. We recall that the k-NN decision rule will correctly classify a sample if its k nearest neighbors share the same label (Section 1.3.1). The objective of LMNN is to increase the number of samples with this property by learning a linear transformation $\mathbf{L}$ of the input space ($\mathbf{x}_i = \mathbf{L}.\mathbf{x}_i$) before applying the k-NN classification:

$$D(\mathbf{x}_i, \mathbf{x}_j) = ||\mathbf{L}(\mathbf{x}_i, \mathbf{x}_j)||_2^2 \tag{2.25}$$

Commonly, the squared distances can be expressed in terms of the square matrix:

$$\mathbf{M} = \mathbf{L}'\mathbf{L} \tag{2.26}$$

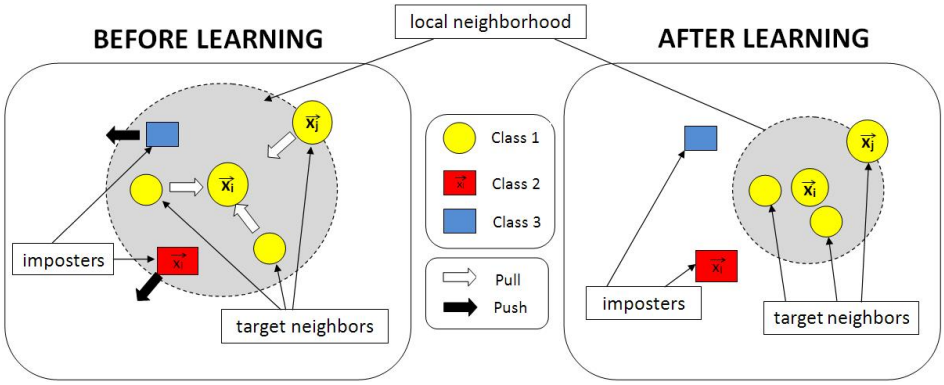


Figure 2.8: Pushed and pulled samples in the $k = 3$ target neighborhood of $\mathbf{x}_i$ before (left) and after (right) learning. The pushed (vs. pulled) samples are indicated by a white (vs. black) arrows (Weinberger & Sault [WS09]).

It is proved that any matrix $\mathbf{M}$ formed as below from a real-valued matrix $\mathbf{L}$ is positive semidefinite (i.e., no negative eigenvalues) [WS09]. Using the matrix $\mathbf{M}$, squared distances can be expressed as:

$$D(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i - \mathbf{x}_j)\mathbf{M}(\mathbf{x}_i - \mathbf{x}_j) \tag{2.27}$$

Learning the linear transformation $\mathbf{L}$ is thus equivalent to learn the corresponding Mahalanobis metric D parametrized by $\mathbf{M}$. This equivalence leads to two different approaches to metric learning: we can either estimate the linear transformation $\mathbf{L}$, or estimate a positive semidefinite matrix $\mathbf{M}$. LMNN solution refers on the latter one.

Mathematically, it can be formalized as an optimization problem involving two competiting terms for each sample $\mathbf{x}_i$: one term penalizes large distances between nearby inputs with the same label (pull), while the other term penalizes small distances between inputs with different labels (push). For every $\mathbf{x}_i$, this implies a minimization problem:

$$\begin{aligned} &\min_{w,\xi} \underbrace{\sum_{i,j \rightsquigarrow i} D(\mathbf{x}_i, \mathbf{x}_j)}_{pull} + C \underbrace{\sum_{i,j \rightsquigarrow i,l} \frac{1 + y_{il}}{2} . \xi_{ijl}}_{push} \\ &\text{s.t. } \forall j \rightsquigarrow i, y_l \neq y_i, \\ &D(\mathbf{x}_i, \mathbf{x}_l) - D(\mathbf{x}_i, \mathbf{x}_j) \geq 1 - \xi_{ijl} \\ &\xi_{ijl} \geq 0 \end{aligned} \tag{2.28}$$

where C is a trade-off between the push and pull term and $y_{il} = -1$ if $y_i = y_l$ (same class) and $+1$ otherwise (different classes). Generally, the parameter C is tuned via cross validation and grid search. Similarly to Support Vector Machine (SVM) approach, slack variables ξ_{ijl} are introduced to balance the effect of noise in real data.

Many parallels can be made between LMNN and SVM: both are convex optimization problem based on a regularized and a loss term. Thus, some authors extends the approach

to work in non-linear feature spaces by using the "kernel trick" . LMNN differs from SVM in which k-NN classification replaces linear classification and LMNN requires no modification for multiclass problems.

ref

2.7 Conclusion of the chapter

To cope with modalities inherent to time series, we review in this chapter several unimodal metrics dedicated to time series. Depending on the considered modality (amplitude, behavior, frequency), adapted metrics for time series have been proposed in the literature such as the Euclidean distance d_A, the Temporal correlation d_B or the Fourier-based distance d_F.

In practice, real time series may be subjected to delays. They need to be re-aligned before any analysis task. For that, the Dynamic Time Warping (DTW) algorithm has been used usually. To capture local characteristics, the previous metrics (d_A, d_B, d_F) can be computed on smaller intervals. Many strategies exist such as the dichotomy or the sliding window.

However, all of these metrics only include one modality and at a particular scale. In general, several modalities may be implied in real data. Some authors proposed to combine temporal metrics together. They mainly combine the Euclidean distance d_A and the Temporal correlation d_B. As k-NN performances is impacted by the choice of the metric, other work propose in the case of static data to learn the metric in order to optimize the k-NN classification. In the following, we extend this framework to learn a combined metric for large margin k-NN classification of time series.

Conclusion of Part I

In order to make the classification or regression of time series, a lot of technics exist in the literature. Our work focus on k-NN classifier and the SVM will be used in the following for its large margin concept. We note that the k-Nearest Neighbors algorithm is based on the comparison of time series through distance measures.

Considering time series as static data lead to the only comparison based on their amplitude and the same time instant. To take into account other specifities of time series (behavior, frequential components), other metrics (e.g., the temporal correlation d_B, the frequential-based distance d_F, etc.) and other methods (Dynamic Time Warping DTW, dichotomy) have been proposed to cope with temporal characteristics.

Learning an adequate metric is a key challenge to well classify time series. Inspired by Metric Learning work for static data, we propose in the following a framework to learn a Multi-modal and Multi-scale Metric for a robust nearest neighbor classifier of time series.