UNIVERSITÉ DE GRENOBLE

THÈSE

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique et Mathématiques appliquées**

Arrêté ministériel : 7 août 2006

Présentée par
Cao Tri DO

Thèse dirigée par **Ahlame DOUZAL-CHOUAKRIA**,
codirigée par **Michèle ROMBAUT** et
co-encadré par **Sylvain MARIÉ**

préparée au sein du
Laboratoire d'Informatique de Grenoble (LIG)
dans **l'école doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique (MSTII)**

Metric Learning for Time Series Analysis

Thèse soutenue publiquement le **date de soutenance**,
devant le jury composé de:

Stéphane CANU
Laboratoire LITIS, Rapporteur
Marc SEBBAN
Laboratoire LAHC, Rapporteur
Patrick GALLINARI
Laboratoire LIP6, Examinateur
Gustavo CAMPS-VALLS
Laboratoire IPL, Examinateur
Ahlame DOUZAL-CHOUAKRIA
Laboratoire LIG, Directeur de thèse
Michèle ROMBAUT
Laboratoire GIPSA-Lab, Co-Directeur de thèse
Sylvain MARIÉ
Schneider Electric, Encadrant

UNIVERSITÉ DE GRENOBLE

ÉCOLE DOCTORALE MSTII

Description de complète de l'école doctorale

T H È S E

pour obtenir le titre de

docteur en sciences

de l'Université de Grenoble-Alpes

Mention : Informatique et Mathématiques appliquées

Présentée et soutenue par

Cao Tri DO

Metric Learning for Time Series Analysis

Thèse dirigée par Ahlame DOUZAL-CHOUAKRIA

préparée au Laboratoire d'Informatique de Grenoble (LIG)

soutenue le date de soutenance

Jury :

Rapporteurs :	Stéphane CANU	-	Laboratoire LITIS
	Marc SEBBAN	-	Laboratoire LAHC
Examinateur :	Patrick GALLINARI	-	Laboratoire LIP6
Examinateur :	Gustavo CAMPS-VALLS	-	Laboratoire IPL
Directeur :	Ahlame DOUZAL-CHOUAKRIA	-	Laboratoire LIG
Co-Directeur :	Michèle ROMBAUT	-	Laboratoire GIPSA-Lab
Encadrant :	Sylvain MARIÉ	-	Schneider Electric

Todo list

Acknowledgements

I would like to thanks:

- my directors
- my GIPSA collegues
- my AMA collegues
- my Schneider collegues
- my parents

Contents

List of Figures

List of Tables

Table of Acronyms

LIG	*Laboratoire d'Informatique de Grenoble*
AMA	*Apprentissage, Méthode et Algorithme*
GIPSA-Lab	*Grenoble Images Parole Signal Automatique Laboratoire*
AGPiG	*Architecture, Géométrie, Perception, Images, Gestes*
A4S	*Analytics for Solutions*
CIFRE	*Conventions Industrielles de Formation par la REcherche*
k-NN	*k-nearest neighbors*
SVM	*Support Vector Machines*
SVR	*Support Vector Regression*
cort	*Temporal correlation*
DTW	*Dynamic Time Warping*
SAX	*Symbolic Aggregate approXimation*
ARIMA	*AutoRegressive Integrated Moving Average*
ARMA	*AutoRegressive Moving Average*
IoT	*Internet of Things*
Acc	*Classification accuracy*
Err	*Classification error rate*
MAE	*Mean Absolute Error*
RMSE	*Root Mean Square Error*
MDS	*MultiDimensional Scaling*
FAQ	*Frequently Asked Questions*
M^{2}TML	*Multi-modal and Multi-scale Temporal Metric Learning*

Introduction

Motivation

The work of this PhD is in the context of a CIFRE[1] thesis with Schneider Electric and two public research laboratories, the LIG[2] and the GIPSA-lab[3]. Within Schneider Electric, the PhD took place in the Analytics for Solutions (A4S) team, part of the Technology and Strategy entity. Among the wide activities of the A4S team, in the context of system modeling (*e.g.*, building, sensor network, Internet of Things), two topics are at least studied: modeling by physical models (white/grey box) and modeling by machine learning algorithms (black-box). With the increase of the amount of data and sensors that collects data, modeling accurately systems through *a priori* equations (white/grey box) for some prediction tasks has become more and more difficult. Within the vast amount of applications in Schneider Electric, some applications involve in particular temporal data, *e.g.*, forecasting the energy consumption in a building, virtual sensors, fault detection. More generally, Schneider Electric, like many other companies and other diverse application domains (medicine, marketing, meteorology, etc.) has taken a growing interest these last decades in machine learning problems (classification, regression, clustering) that involves time series of one or several dimensions, of different samplings, of two or more classes, etc. A time series can be seen in signal processing and in control theory as the response of a dynamic system. Contrary to static data, time series are more challenging in the sense that the temporal aspect (*i.e.*, order of appearance of the observations) is an additional key information.

Problem statement and contributions

In this work, we focus on classification problems of monovariate time series (data of 1 dimension) with a fixed sampling rate and of same lengths. Among the wide variety of algorithms that exist in machine learning, some approaches (*e.g.*, k-Nearest Neighbors (k-NN)) classify samples using a concept of neighborhood based on the comparison between samples. In general, the concept of 'near' and 'far' between samples is expressed through a distance measure. Time series can be compared based not only on their amplitudes like static data but also on other characteristics, called modalities, such as their dynamic or frequency components. Many metrics for time series have been proposed in the literature such as the euclidean distance [Din+08], the temporal correlation [FDcG13], the Fourier-based distance [SS12a]. A detailed review of the major metrics is proposed in [MV14]. In general, the existing metrics involve one modality (characteristic) at the global scale (*i.e.*, implying systematically all the

[1] Conventions Industrielles de Formation par la REcherche
[2] Laboratoire d'Informatique de Grenoble
[3] Grenoble Images Parole Signal Automatique

time series observations). We believe also that the multi-scale aspect of time series (*i.e.*, implying a part of the observations), not present in static data, could enrich the definition of the existing metrics.

In this work, our objective is to learn a combined multi-modal and multi-scale time series metric for a robust k-NN classifier. The main contributions of the PhD are:

- The definition of a new space representation: the pairwise dissimilarity space where each pair of time series is embedded as a vector described by basic temporal metrics.
- The definition of basic temporal metrics that involves one modality at one specific scale.
- The learning of a multi-modal and multi-scale temporal metric for a large margin k-NN classifier of univariate time series.
- The definition of the general problem of learning a combined metric as a metric learning problem using the dissimilarity representation.
- The proposition of a framework based on Support Vector Machine (SVM) and a linear and non-linear solution to define the combined metric that satisfies at least the properties of a dissimilarity measure.
- The comparison of the proposed approach with standard metrics on a large number of public datasets.
- The analysis of the proposed approach to extract the discriminative features that are involved in the definition of the learned combined metric.

Organization of the manuscript

The first part makes a review of existing methods in machine learning and metrics for time series. The first chapter presents classical approaches in machine learning. We recall the general principle and framework in supervised learning and focus on two machine learning algorithms: the k-Nearest Neighbors (k-NN) and the Support Vector Machine (SVM) approach. In the second chapter, we review some basic terminology for time series and recall three types at least of metrics proposed for time series: amplitude-, behavior- and frequential-based.
The second part of the manuscript propose a Multi-modal and Multi-scale Temporal Metric Learning (M^2TML) approach for a robust k-NN classifier of time series. In the third chapter, we first review the concept of metric learning for static data and focus on a framework of metric learning for nearest neighbors classification proposed by Weinberger & Saul [WS09b]. Secondly, we present a new space representation, the pairwise dissimilarity space based on a multi-modal and multi-scale time series description and their corresponding basic metrics. Then, we formalize the general M^2TML optimization problem using the pairwise dissimilarity space representation. From the general formalization, we propose at least three different formalizations. The first and second proposition involve different regularizers, allowing to

learn an *a priori* linear or non-linear form of the combined metric. The third proposition presents a framework based on SVM and a solution to build the combined metric, in the linear and non-linear context, satisfying at least the properties of a dissimilarity measure. Finally, Chapter 4 presents the experiments conducted on a wide range of 30 public and challenging datasets, and discusses the results obtained.

Notations

$\mathbf{x}_i$	a vector sample / a time series
y_i	a label (discrete or continous)
$\hat{y}_i$	a predicted label (discrete or continous)
$X = \{\mathbf{x}_i, y_i\}_{i=1}^{n}$	a training set of $n \in \mathbb{N}$ labeled time series
$X_{Test} = \{\mathbf{x}_j, y_j\}_{j=1}^{m}$	a test set of $m \in \mathbb{N}$ labeled time series
d_A	amplitude-based distance
d_B	behavior-based distance
d_F	frequential-based distance
d_E	Euclidean distance
L_q	Minkovski q-norm
$\|\|\mathbf{x}\|\|_q$	q-norm of the vector $\mathbf{x}$
$cort$	temporal correlation
D	linear learned combined distance
$D_{\mathscr{H}}$	non-linear learned combined distance
κ	kernel function
$\phi(\mathbf{x}_i)$	embedding function from the original space to the Hilbert space
$\mathbf{x}_{ij}$	a pair of time series $\mathbf{x}_i$ and $\mathbf{x}_j$
y_{ij}	the pairwise label of $\mathbf{x}_{ij}$
t	time stamp/index with $t = 1, ..., q$
q	length of the time series (supposed fixed)
f	frequential index
F	length of the Fourier transform
ξ	slack variable
p	number of metric measure considered in the metric learning process
r	order of the temporal correlation
k	number of nearest neighbors
C	hyper-parameter of the SVM (trade-off)
α	parameter to control the size of the neighborhood
λ	parameter to control the push term
$\mathbf{w}$	weight vector
v	number of folds in cross-validation

Chapter 1

Related work

Contents

In this chapter, we recall some concepts of machine learning. First, we review the principles, the learning framework and the evaluation protocol in supervised learning. Then, we present the algorithms used in our work: k-Nearest Neighbors (k-NN) and Support Vector Machines (SVM).

1.1 Classification, Regression

In this section, we review some terminology used in machine learning. First, we recall the principle of machine learning. Then, we detail how to design a framework for supervised learning. After that, we present model evaluation. Finally, we review data normalization.

1.1.1 Machine learning principle

The idea of machine learning (also known 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 a 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. Then, during a 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 vector samples $\mathbf{x}_i \in \mathbb{R}^p$ and y_i their corresponding labels. The aim of supervised machine learning is to learn a relationship (model) f between the samples $\mathbf{x}_i$ and their labels y_i based on examples [Bis06]; [G. 06]; [OE73]. 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 $\hat{y}_j$ 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. 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 [Zhu07]. Note that when the labels y_i are totally unknown, learning f refers to a clustering problem (unsupervised learning) [JMF99]; [CHY96]. Semi-supervised and unsupervised learning problems are out of the scope of this work.

1.1.2 Model selection in supervised learning

A key objective of supervised learning algorithms is to build models f with good generalization capabilities, *i.e.*, models f that correctly predict the labels y_j of new unknown samples $\mathbf{x}_j$. There exists two types of errors committed by a classification or regression model f: training error and generalization error. **Training error** is the error on the training set and **generalization error** is the error on the testing set. 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 much can have a poorer generalization error than a model with a higher training error. Such situation is known as model overfitting (Fig. 1.1).

In most cases, learning algorithms require to tune some hyper-parameters. A first approach could consist in trying all the possible combinations of hyper-parameters values and keep the one with the lowest training error. However, as discussed above, the model with the lowest training error is not always the one with the best generalization error. To avoid overfitting, the training set can be divided into 2 sets: a learning and a validation set. Suppose that 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. 1.2). For each combination (a cell of the grid), the model is learned 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 referred as the **model selection.**

An alternative is **cross-validation** with v folds, illustrated in Fig. 1.3. 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

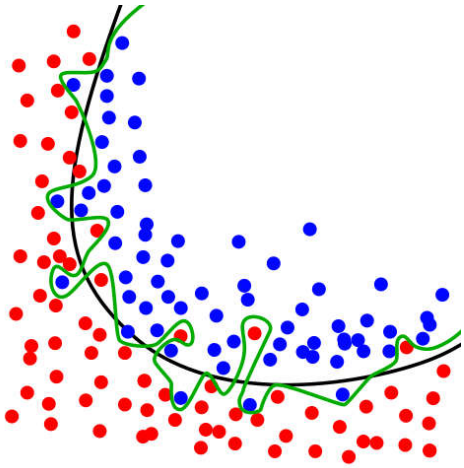


Figure 1.1: An example of overfitting in the case of classification. The objective is to separate blue points from red points. Black line shows a classifier f_1 with low complexity where as green line illustrates a classifier f_2 with high complexity. On training examples (blue and red points), the model f_2 separates all the classes perfectly but may lead to poor generalization on new unseen examples. Model f_1 is often preferred.

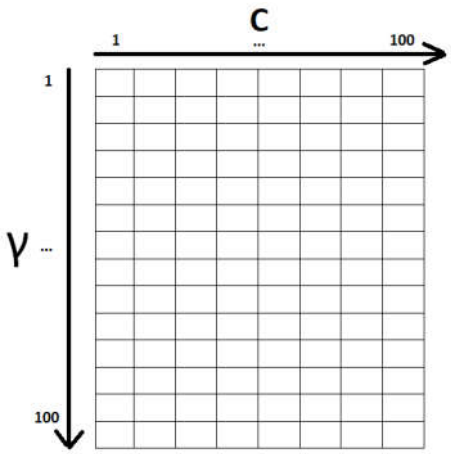


Figure 1.2: Example of a 2 dimensional grid search for parameters C and γ. It defines a grid where each cell of the grid contains a combination (C, γ). Each combination is used to learn the model and is evaluated on the validation set.

the $v-1$ remaining 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 that as much data as possible are used for training. Moreover, the validation 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 validation set contains only one record, the variance of the

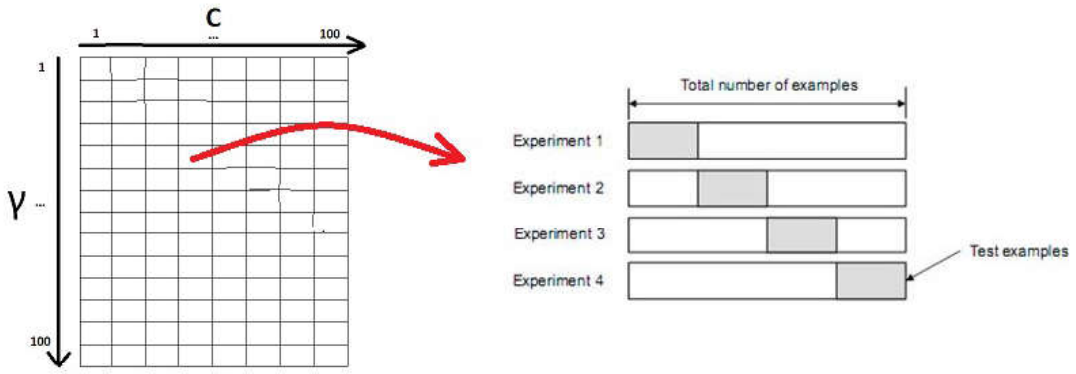


Figure 1.3: v-fold Cross-validation for one combination of parameters. For each of v experiments, use $v-1$ folds for training and a different fold for Testing, then the training error for this combination of parameter is the mean of all testing errors. This procedure is illustrated for $v = 4$.

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.

To sum up, Fig. 1.4 shows a general approach for solving machine learning problems. In general, a dataset can be divided into 3 sub-datasets (illustrated in Fig. 1.5):

- A **training set** $X = \{\mathbf{x}_i, y_i\}_{i=1}^{n}$, which consists of n samples $\mathbf{x}_i$ whose labels y_i are known. The training set is used to build the supervised model f. When the learning algorithm requires some hyper-parameters to be tuned, there exists a risk of model overfitting. To avoid such risks, the training set X has to be divided into two subsets :
  - A **learning set** which is used to build the supervised model f for each value of the hyper-parameters.
  - A **validation set** which is used to evaluate the supervised model f for each value of the hyper-parameter. The model f with the lowest error on the validation set is kept, thus ensuring that it has the best generalization abilities.
- A **test set** $X_{Test} = \{\mathbf{x}_j, y_j\}_{j=1}^{m}$, which consists of m samples $\mathbf{x}_j$ whose labels y_j are also known but are not used during the training step. The model f is applied to predict the label $\hat{y_j}$ of samples $\mathbf{x}_j$ to evaluate the performance of the learnt model by comparing $\hat{y_j}$ and y_j.
- An **operational set** $X_{op} = \{\mathbf{x}_l, y_l\}_{l=1}^{L}$, which consists of L samples $\mathbf{x}_l$ whose labels y_l are totally unknown. The operational set is in general a new dataset on which the learnt algorithm is applied.

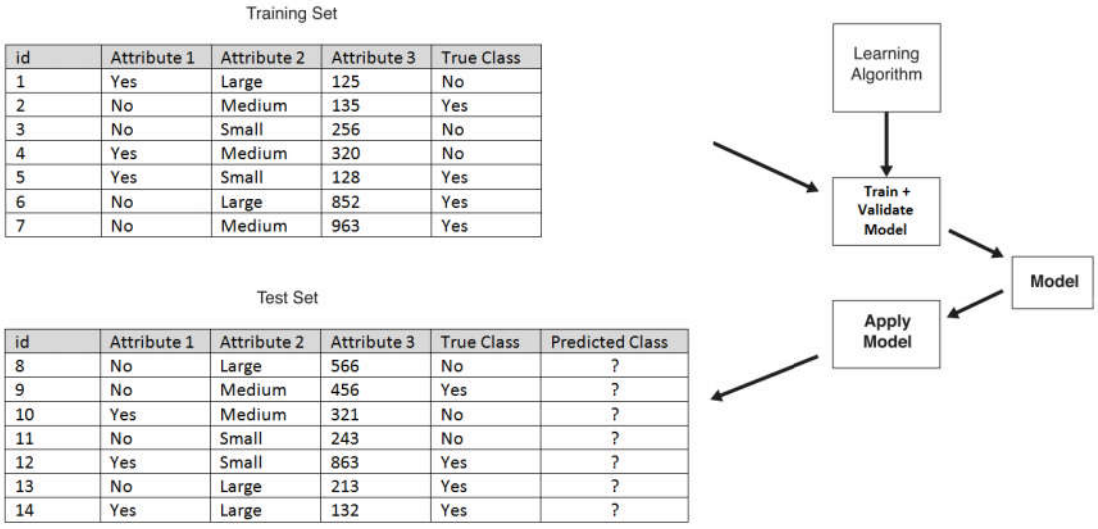


Figure 1.4: General framework for building a supervised (classification/regression) model. Example with 3 features and 2 classes ('Yes' and 'No').

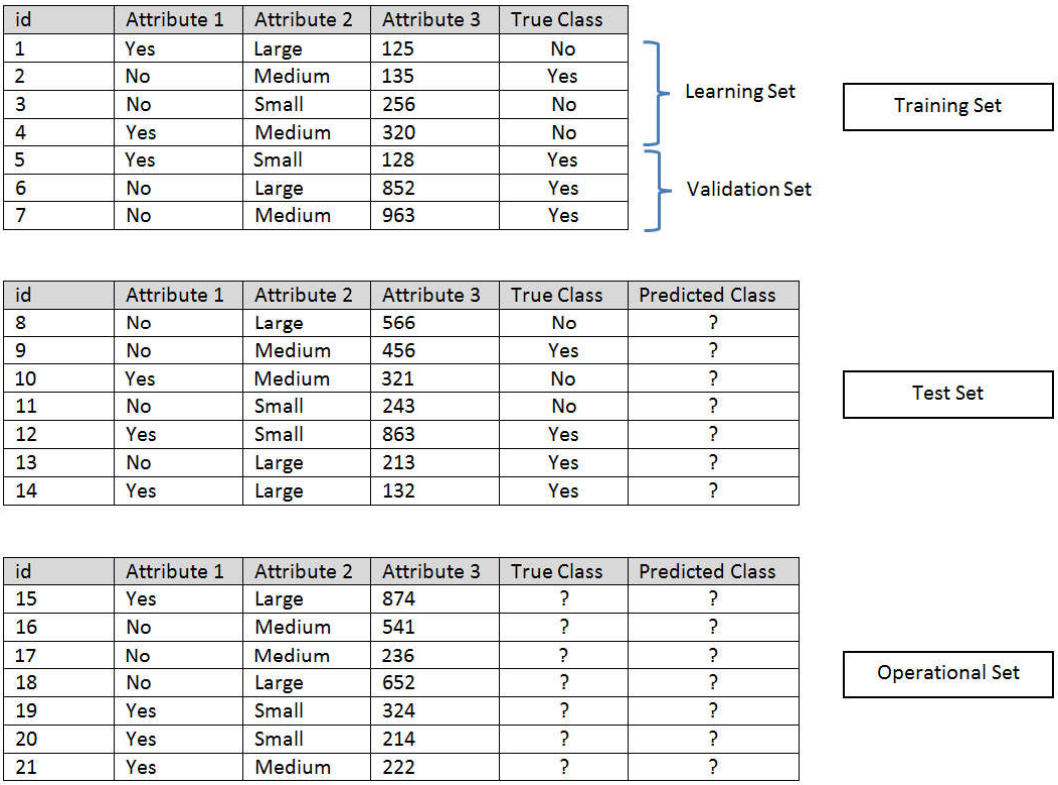


Figure 1.5: Division of a dataset into 3 datasets: training, test and operational.

1.1.3 Model evaluation

As seen in the previous section, model selection is inherently based on the ability to quantify its error on the training and validation sets. In this section, we recall how this error is computed for classification and regression problems.

1.1.3.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 g_{ij} of 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 $\sum_{i=1}^{C} g_{ii}$, where C is the number of classes.

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

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

For binary classification problems, g_{11} is the number of true positives, g_{10} is the number of false negatives, g_{01} is the number of false positives and g_{00} is the number of true negatives.

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

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

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

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 set 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 m_A (resp. m_B) of misclassification of m test examples made by classifier f_A (resp. f_B) 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 with a mean $m(p_A - p_B)$. Thus, the quantity $m_A - m_B$ is a normally distributed random variable. Under the null hypothesis (the two algorithms 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).

1.1.3.b Regression evaluation

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 L_1-norm loss.

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

A commonly used performance metrics is the Root Mean Squared Error function ($RMSE$) that computes the root of the mean square error:

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

Many works rely on the R^2 measure, the coefficient of determination [Nag91]. It provides

a measure of how well future samples are likely to be predicted by the model[1]. It can also be interpreted as a measure of how the model f is better than a constant model.

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

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

1.1.4 Data normalization

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

Let $X = \{\mathbf{x}_i, y_i\}_{i=1}^{n}$ be a training set, $\mathbf{x}_i$ being a sample described by p attributes $x_1, \ldots, x_p$. Part 2 of Sarle's Neural Networks FAQ (1997) [2] explains the importance of data normalization for neural networks but they can be applied to any learning algorithms. The main advantage of normalization 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 Support Vector Machine (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 [HCL08].

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 or Z-normalization [MU13]. Let μ_j and σ_j as the mean and the standard deviation of a variable x_j, applying the Z-normalized variable x_j^{norm} is given by:

$$x_j^{norm} = \frac{x_j - \mu_j}{\sigma_j} \tag{1.8}$$

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

[1] `http://scikit-learn.org/stable/modules/model_evaluation.html`
[2] `http://www.faqs.org/faqs/ai-faq/neural-nets/`

1.2 Machine learning algorithms

Many algorithms have been proposed in the context of supervised learning, such as Deep Neural Networks, Decision Trees, k-Nearest Neighbors (k-NN) or Support Vector Machine (SVM). In Decision Trees, the aim is to build a decision tree by recursively partitioning the sample space [Qui86]. The tree consists of nodes that splits the sample space into subspaces, and leaf nodes that are associated to classes. In Deep Neural Networks, most of the propositions aims to learn a representation of the data by extracting features using a cascade of layers [Lee+09], then to use a neural network algorithm to learn a model from the learned features. One main advantage of Decision Trees from Deep Neural Networks is that by using the features from the sample space, the model is still interpretable.
In this section, we focus and detail two approaches: k-Nearest Neighbors (k-NN) and Support Vector Machine (SVM).

1.2.1 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 use the class y_i of its nearest neighbor $\mathbf{x}_i$ in the training set in order to predict its labels: $\hat{y}_j = y_i$.
More generally, we can consider the k nearest neighbors of $\mathbf{x}_j$. The class y_j of a 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 referred as the k-Nearest Neighbors algorithm (k-NN) [SJ89]; [CH67]. Fig. 1.6 illustrates the concept for a neighborhood of $k = 3$ and $k = 5$.

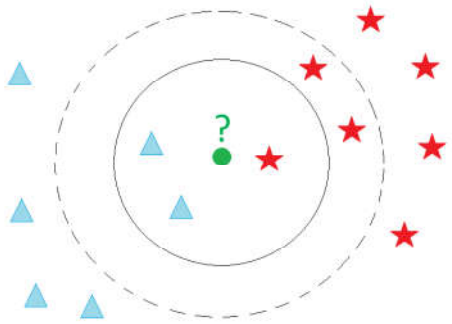


Figure 1.6: Example of k-NN classification. The test sample (green circle) is classified either to the first class (red stars) or to the second class (blue triangles). If $k = 3$ (solid line circle) it is assigned to the second class because there are 2 triangles and only 1 star inside the inner circle. 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 $\mathbf{x}_i$ is based on the computation of a metric[3] D. For static data, frequently used metrics are the Euclidean

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

distance, the Minkowski distance or the Mahalanobis distance. Considering a training set X of n samples, solving the 1-NN classification problem is equivalent to solve the following optimization problem: for a new sample $\mathbf{x}_j$, $\forall i \in \{1, \ldots, n\}$,

$$y_j = y_{i^*} \tag{1.9}$$

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

The k-NN algorithm can be extended to estimate continous labels (regression problems). In that case, the label y_j is defined as :

$$y_j = \frac{1}{k} \sum_{i=1}^{k} y_i \tag{1.10}$$

where i corresponds to the index of the k-nearest neighbors [Alt92]. 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 defined as a function of the distance $D(\mathbf{x}_i, \mathbf{x}_j)$ [Dud76]. To cope with uncertainty or imprecision in the labeling of the training data $\mathbf{x}_i$, other authors propose some variants such as the fuzzy k-NN or the belief k-NN. In a fuzzy k-NN, the membership degree in each class of an unseen sample $\mathbf{x}_j$ is obtained by combining the memberships of its neighbors [KGG85]. In a belief k-NN, the approach relies on the Dempster-Shafer theory to modify the belief concerning the class membership of a pattern, and quantify the uncertainty attached to each sample [Den95].

Despite its implementation simplicity, the k-NN algorithm has been shown to be successful on time series classification problems [BMP02]; [Xi+06]; [Din+08]. However, the k-NN algorithm presents some disadvantages, mainly due to its computational complexity, both in memory space (storage of the training samples $\mathbf{x}_i$) and time (search of the neighbors) [OE73]. Suppose that we have n labeled training samples in p 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 $\mathbf{x}_i$ ($i = 1, \ldots, n$) one by one, calculate its distance to $\mathbf{x}_j$ ($D(\mathbf{x}_i,\mathbf{x}_j)$) and retain the index of the current closest one. For the standard Euclidean distance, each metric computation is $O(p)$ and thus the search is $O(pn)$. Finally, note that using standard metrics (such as the Euclidean distance) in the k-NN relies on all p dimensions in the computation of the metric and thus assumes that all dimensions have the same effect on the metric. This assumption may be wrong and may impact the classification performances.

1.2.2 Support Vector Machine (SVM) algorithm

Support Vector Machine (SVM) is a classification method introduced in 1992 by Boser, Guyon, and Vapnik [BGV92]; [CV95] to solve at first linearly separable problems. The SVM classifier has demonstrated 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]. It allows thus to solve non-linear problems. This section gives a brief overview of the mathematical key points and interpretation of the method. For more information, the reader can consult [SS13]; [CY11]; [CV95].

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 a 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.

1.2.2.a Intuition

Let $\{\mathbf{x}_i, y_i\}_{i=1}^n$ be a set of n samples $\mathbf{x}_i \in \mathbb{R}^p$ and their labels $y_i = \pm 1$ (2 class-problem). The objective is to learn a hyperplane, whose equation is $\mathbf{w}^T\mathbf{x} + b = 0$[4], that can separate samples of class $+1$ from the ones of class -1. When the problem is linearly separable such as in Fig. 1.7, there exists an infinite number of valid hyperplanes.

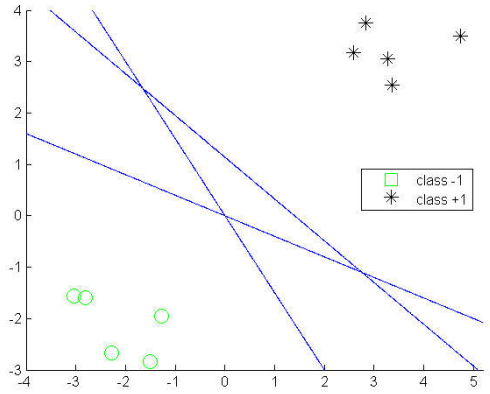


Figure 1.7: Example of linear classifiers (blue lines) in a 2-dimensional classification problem. For a set of samples of classes $+1$ (stars) and -1 (rectangle) that are linearly separable, there exists an infinite number of separating hyperplanes corresponding to $\mathbf{w}^T\mathbf{x} + b = 0$.

Vapnik & al. [CV95] propose to choose the separating hyperplane that maximizes the margin, *i.e.*, the hyperplane that leaves as much distance as possible between the hyperplane and the closest samples $\mathbf{x}_i$ of each class, called the **support vectors**. This distance is equal to $\frac{1}{||\mathbf{w}||_2}$.

[4] $\mathbf{w}^T$ denotes the transpose of the vector $\mathbf{w}$

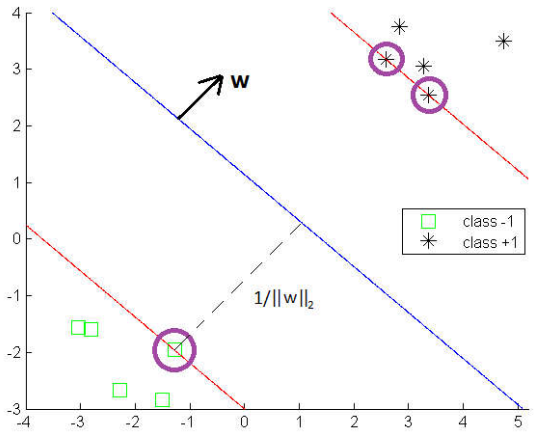


Figure 1.8: The argument inside the decision function of a classifier is $\mathbf{w}^T\mathbf{x}+b$. The separating hyperplane corresponding to $\mathbf{w}^T\mathbf{x} + b = 0$ is shown as a line in this 2-dimensional plot. This hyperplane separates the two classes of data with points on one side labeled $y_i = +1$ ($\mathbf{w}^T\mathbf{x}_i + b \geq 0$) and points on the other side labeled $y_i = -1$ ($\mathbf{w}^T\mathbf{x}_i + b < 0$). Support vectors are circled in purple and lies on the hyperplanes $\mathbf{w}^T\mathbf{x} + b = +1$ and $\mathbf{w}^T\mathbf{x} + b = -1$

We denote $||\mathbf{w}||_2$, the L_2-norm of the vector $\mathbf{w}$ and $||\mathbf{w}||_1$ the L_1-norm of $\mathbf{w}$:

$$||\mathbf{w}||_2 = \sqrt{\mathbf{w}^T\mathbf{w}} = \sqrt{\sum_{h=1}^{p} w_h^2} \tag{1.11}$$

$$||\mathbf{w}||_1 = \sum_{h=1}^{p} |w_h| \tag{1.12}$$

where $\mathbf{w} = [w_1, \ldots, w_p]^T$ denotes the weight vector.
The two 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. 1.8). From this, for a binary classification problem, to classify a new sample $\mathbf{x}_j$, the decision function is:

$$f(\mathbf{x}_j) = sign(\mathbf{w}^T\mathbf{x}_j + b) \tag{1.13}$$

1.2.2.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:

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

$$\textbf{s.t. } \forall i = 1, \ldots, n: \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \tag{1.15}$$

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**. When the problem is not linearly separable, slack variables $\xi_i \geq 0$ are introduced to relax the optimization problem:

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

$$\textbf{s.t. } \forall i = 1, \ldots, n:$$

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \tag{1.17}$$

$$\xi_i \geq 0 \tag{1.18}$$

where $C > 0$ is a penalty hyper-parameter. This formulation is referred as the **primal soft margin problem**. It is a 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, controlling 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 $+\infty$, all the slack variables ξ_i have to be equal to zero in order to not have an infinite loss term. The problem is thus equivalent to the primal hard margin problem. The hyper-parameter C is learnt during the training phase (Section 1.1.2).

1.2.2.c Dual formulation

From the primal formulation, 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. The 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 Eqs. 1.14, 1.15 and 1.18. 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 (Eq. 1.14) and the n constraints (Eq. 1.15) multiplied by their respective

Lagrange multipliers $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_n]^T$:

$$\underset{\boldsymbol{\alpha}}{\operatorname{argmax}} \left(L(\mathbf{w}, b) = \frac{1}{2}(\mathbf{w}^T\mathbf{w}) - \sum_{i=1}^{n} \alpha_i (y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1) \right) \tag{1.19}$$

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

$$\alpha_i \geq 0 \tag{1.20}$$

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 \geq 0 \tag{1.21}$$

$$\alpha_i(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1) = 0 \tag{1.22}$$

where $\alpha_i \geq 0$ are the Lagrange multipliers. In optimization theory, Eqs. 1.20, 1.21 and 1.22 are called the Karush-Kuhn-Tucker (KKT) conditions [Bis06]. 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 1.2.2.e.

At the maximum value of $L(\mathbf{w}, b)$, we assume that 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:

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{1.23}$$

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \tag{1.24}$$

By substituting $\mathbf{w}$ into $L(\mathbf{w}, b)$ in Eq. 1.19, we obtain the **dual formulation** (*Wolfe dual*):

$$\underset{\boldsymbol{\alpha}}{\operatorname{argmax}} \left(\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i^T \mathbf{x}_j) \right) \tag{1.25}$$

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

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{1.26}$$

$$\alpha_i \geq 0 \tag{1.27}$$

The dual objective in Eq. 1.25 is quadratic in the parameters α_i. Adding the constraints in Eqs. 1.26 and 1.27, it is a constrained quadratic programming optimization problem (QP). Note that while the primal formulation is a minimization problem, the equivalent dual for-

mulation is a maximization problem. It can be shown that the objective functions of both formulations (primal and dual) reach the same value when the solution is found [CY11].
In the same spirit, it can be shown that the soft margin primal problem leads to the same formulation to the ones in Eqs. 1.25 and 1.26, except that the Lagrange multipliers α_i are upper bounded by the trade-off C in the soft margin formulation [CY11]:

$$0 \leq \alpha_i \leq C \tag{1.28}$$

The constraints in Eq. 1.28 are called the Box constraints. 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:

$$\mathbf{w}^* = \sum_{i=1}^{n} \alpha_i^* y_i \mathbf{x}_i \tag{1.29}$$

$$b^* = \sum_{i=1}^{n} (\mathbf{w}^T \mathbf{x}_i - y_i) \tag{1.30}$$

At the optimality point, Eq. 1.22 leads $\alpha_i^* = 0$ for all datapoints that are well classified and that are not on the margin. Hence, only a few number of datapoints have $\alpha_i^* > 0$ as shown as in Fig. 1.9. These samples are the support vectors. All other datapoints have $\alpha_i^* = 0$, and the decision function is independent of them. Thus, the representation is said sparse.

From Eqs. 1.13 & 1.29, to classify a new sample $\mathbf{x}_j$, the decision function for a binary classification problem is:

$$f(\mathbf{x}_j) = sign\left(\sum_{i=1}^{n} \alpha_i^* y_i (\mathbf{x}_i^T \mathbf{x}_j) + b^*\right) \tag{1.31}$$

1.2.2.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].

From the dual objective in Eq. 1.25, we note that the samples $\mathbf{x}_i$ are only involved in a dot-product. Therefore, if we map these samples $\mathbf{x}_i$ into a higher dimensional hyperspace, called the feature space, we only need to know the dot product in the feature space:

$$(\mathbf{x}_i.\mathbf{x}_j) \rightarrow \Phi(\mathbf{x}_i).\Phi(\mathbf{x}_j) \tag{1.32}$$

where Φ is the mapping function.

The intuition behind using such mapping is that for many datasets, it is not possible to find

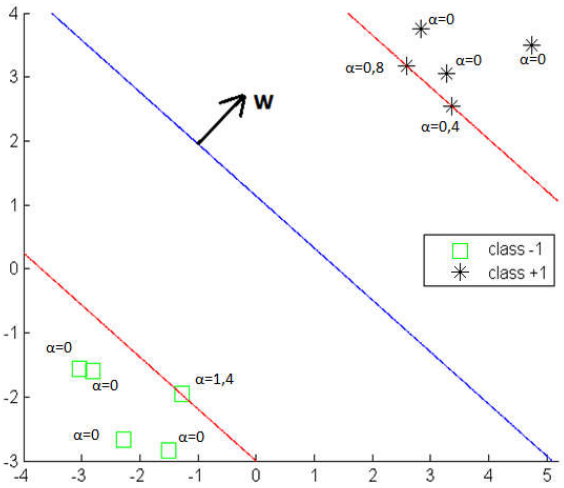


Figure 1.9: Hyperplane obtained after a dual resolution (blue line). The 2 canonical hyperplanes (red lines) contain the support vectors whose $\alpha_i > 0$. Other points have their $\alpha_i = 0$ and the equation of the hyperplane is only affected by the support vectors.

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. Fig. 1.10 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 $+1$ (circle) labeled points are moved forward and the -1 (cross) labeled moved back the two classes become separable.

In most of the case, the mapping function Φ does not need to be known since we only need the dot product $\Phi(\mathbf{x}_i).\Phi(\mathbf{x}_j)$. Therefore, we can use any kernel function κ such that: $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i).\Phi(\mathbf{x}_j)$. We call Gram matrix G, the matrix containing all $\kappa(\mathbf{x}_i, \mathbf{x}_j)$:

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

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 spaces.**

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

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

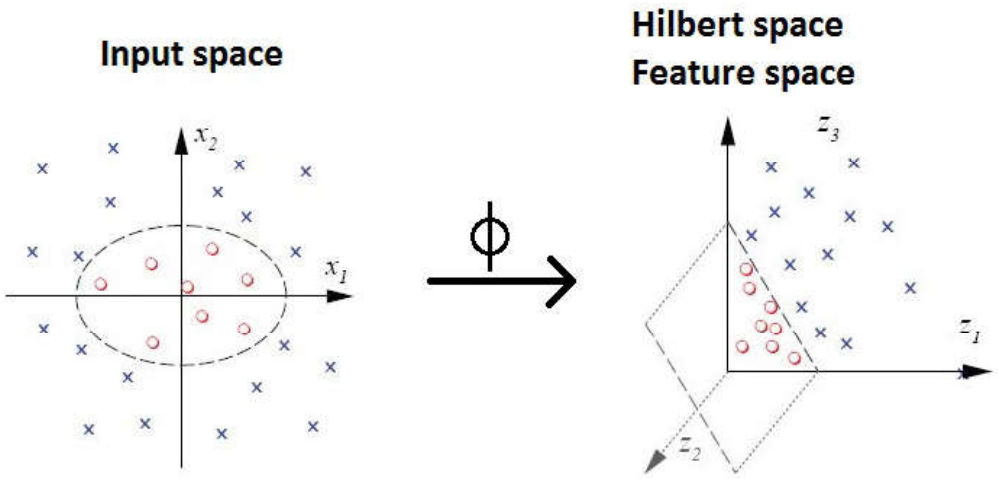


Figure 1.10: Left: in two dimensions the two classes of data (-1 for cross and +1 for circle) are mixed together, and it is not possible to separate them by a line: the data is not linearly separable. Right: using a kernel, these two classes of data become separable by a hyperplane in feature space, which maps to the nonlinear boundary shown, back in input space.[5]

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

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j \tag{1.33}$$

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{||\mathbf{x}_j - \mathbf{x}_i||_2^2}{2\sigma^2}\right) = \exp\left(-\gamma||\mathbf{x}_j - \mathbf{x}_i||_2^2\right) \tag{1.34}$$

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 the number of dimensions p 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 space. The maximum value of the kernel ($\kappa(\mathbf{x}_i, \mathbf{x}_j)$=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. 1.11 [6]. The parameter γ controls the decreasing speed of the sphere. In practice, this parameter is learned during the training phase (Section 1.1.2).

By applying the kernel trick to the soft margin formulation in Eqs. 1.25, 1.26 and 1.28,

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

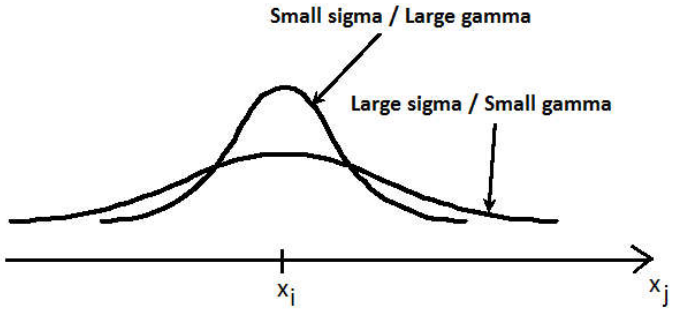


Figure 1.11: Illustration of the Gaussian kernel in the 1-dimensional input space for a small and large γ when $\mathbf{x}_i$ is fixed and $\mathbf{x}_j$ varies.

the following optimization problem allows to learn non-linear classifiers:

$$\underset{\boldsymbol{\alpha}}{\text{argmax}} \left(\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j) \right) \tag{1.35}$$

$$\textbf{s.t.} \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{1.36}$$

$$0 \leq \alpha_i \leq C \tag{1.37}$$

The decision function f becomes:

$$f(\mathbf{x}_j) = sign \left(\sum_{i=1}^{n} \alpha_i^* y_i \kappa(\mathbf{x}_i, \mathbf{x}_j) + b^* \right) \tag{1.38}$$

Let n_{SV} be the number of support vectors ($n_{SV} \leq n$). To recover b^*, we recall that for support vectors $\mathbf{x}_i$:

$$y_j \left(\sum_{i=1}^{n_{SV}} \alpha_i^* y_i \kappa(\mathbf{x}_i, \mathbf{x}_j) + b^* \right) = 1 \tag{1.39}$$

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

$$b^* = \frac{1}{y_j} - \sum_{i=1}^{n_{SV}} \alpha_i^* y_i \kappa(\mathbf{x}_i, \mathbf{x}_j) \tag{1.40}$$

Note that in this case, we can't recover the weight vector $\mathbf{w}^*$ but it is not useful here for the decision function.

1.2.2.e Interpretation

Interpretation in the primal

We recall that $\mathbf{x}_i$ is a sample in p dimensions: $x_1, \ldots, x_p$. Geometrically, the vector $\mathbf{w}$ represents the direction of the hyperplane and points towards the direction of positive decision function $f(\mathbf{x}) \geq 0$ (Fig. 1.12). The absolute value of the bias $|b|$ is equal to the distance of the hyperplane to the origin $\mathbf{x} = \mathbf{0}$[7] if the norm of the vector $\mathbf{w}$ is equal to 1. In the soft margin problem, the slack variables ξ_i can be interpreted as follows:

- $\xi_i = 0$ implies that $\mathbf{x}_i$ is correctly classified and is either on the margin or on the correct side of the margin.
- $0 < \xi_i \leq 1$ implies that $\mathbf{x}_i$ lies inside the margin, but on the correct side of the decision boundary.
- $\xi_i \geq 1$ implies that $\mathbf{x}_i$ lies on the wrong side of the decision boundary and is misclassified.

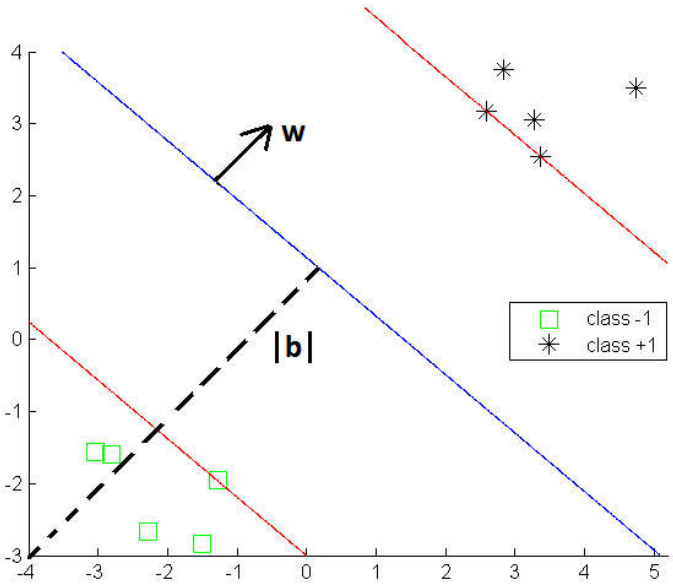


Figure 1.12: Geometric representation of SVM.

In the primal formulation, the weight vector $\mathbf{w} = [w_1, \ldots, w_p]^T$ contains as many elements as there are dimensions in the dataset, *i.e.*, $\mathbf{w} \in \mathbb{R}^p$. The magnitude of each element in $\mathbf{w}$ denotes the 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 $\mathbf{w}^T\mathbf{x} + b = 0$ shown in Fig. 1.13 with $p = 2$. Fig. 1.13(a) shows a hyperplane where elements of $\mathbf{w}$ are the same for both variables x_1 and x_2. The interpretation is that

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

both variables contribute equally for classification of objects into positive and negative. Fig. 1.13(b) shows a hyperplane where the element of $\mathbf{w}$ for x_1 is 1, while that for x_2 is 0. This is interpreted as that x_1 is important but x_2 is not. An opposite example is shown in Fig. 1.13(c) where x_2 is considered to be important but x_1 is not. Finally, Fig. 1.13(d) provides a 3-dimensional example ($p = 3$) where an element of $\mathbf{w}$ for x_3 is 0 and all other elements are equal to 1. The interpretation is that x_1 and x_2 are important but x_3 is not.

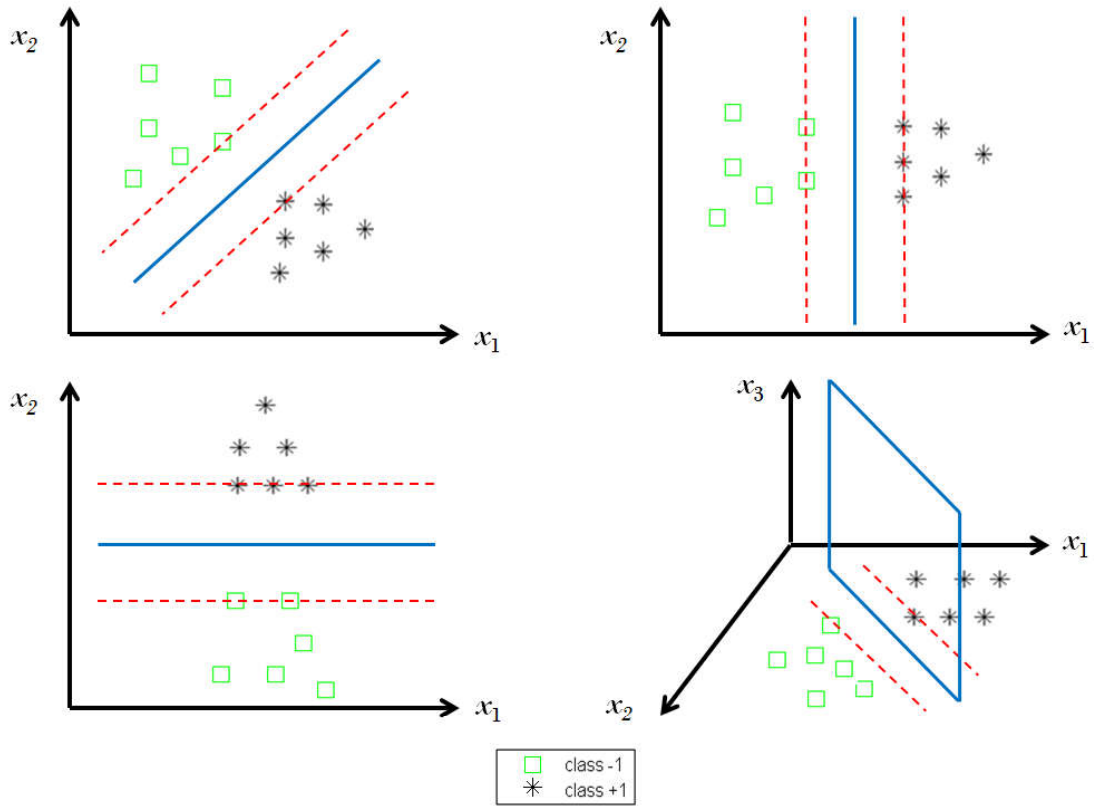


Figure 1.13: Example of several SVMs and how to interpret the weight vector $\mathbf{w}$

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

Interpretation in the dual

As a constrained optimization, the dual form satisfies the Karush-Kuhn-Tucker (KKT) conditions (Eqs. 1.20, 1.21 and 1.22). We recall Eq. 1.22:

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

From this, for every datapoint $\mathbf{x}_i$, either $\alpha_i^* = 0$ or $y_i(\mathbf{w}^T\mathbf{x}_i + b) = 1$. Any datapoint with $\alpha_i^* = 0$ do not appear in the sum of the decision function f in Eq. 1.31 or 1.38. 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 vectors. 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 the coefficient α_i^* for a sample $\mathbf{x}_i$ is, the more the sample $\mathbf{x}_i$ impacts on the decision function f. However, an unusually high value of α_i^* among the samples can lead to two interpretations: either this point is a critical point to the decision, or 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.

1.2.2.f Variants of SVM

From the primal formulation of SVM (Eqs. 1.14 & 1.15), some works investigate the effect of modifications in the regularization and loss term [HCL08].

First, the two common regularizers are $||\mathbf{w}||_1$ and $||\mathbf{w}||_2$. The former is referred to as L_1-Regularizer while the latter is L_2-Regularizer. L_1-Regularizer is used to obtain sparser models than L_2-Regularizer, *i.e.*, the vector $\mathbf{w}$ will contain many elements w_i that will equal to zero. Thus, it can be used for variable selection.

Secondly, the two common loss functions ξ_i are $\max(1 - y_i(\mathbf{w}^T\mathbf{x}_i + b), 0)$ and $[\max(1 - y_i(\mathbf{w}^T\mathbf{x}_i + b), 0)]^2$. The former is referred to as L_1-Loss and the latter is L_2-Loss function. L_2-loss function will penalize more slack variables ξ_i during training and would be more sensitive to outliers. Theorically, it should lead to less error in training and poorer generalization in most of the case [HCL08]. In general, L_1-Loss is preferred.

1.2.2.g 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 $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$. 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. 1.14.

The ϵ-insensitive error function E_ϵ is defined by:

$$E_\epsilon(f(\mathbf{x}_i) - y_i) = \begin{cases} 0 & \text{if} \quad |f(\mathbf{x}_i) - y_i| < \epsilon \\ |f(\mathbf{x}_i) - y_i| - \epsilon & \text{otherwise} \end{cases} \tag{1.41}$$

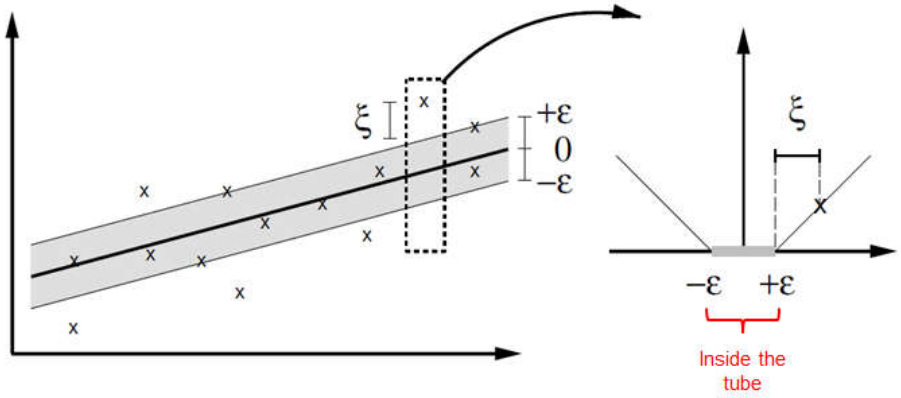


Figure 1.14: Illustration of SVM regression (left), showing the regression curve with the ϵ-insensitive "tube" (right). Samples $\mathbf{x}_i$ above the ϵ-tube have $\xi_1 > 0$ and $\xi_1 = 0$, points below the ϵ-tube have $\xi_2 = 0$ and $\xi_2 > 0$, and points inside the ϵ-tube have $\xi = 0$.

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

$$\underset{\mathbf{w},b}{\operatorname{argmin}} \left(\overbrace{\frac{1}{2}||\mathbf{w}||_2^2}^{Regularization} + C \overbrace{\sum_{i=1}^{n}(\xi_{i_1} + \xi_{i_2})}^{Loss} \right) \tag{1.42}$$

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

$$y_i - (\mathbf{w}^T \mathbf{x}_i + b) \geq \epsilon - \xi_{i_1} \tag{1.43}$$

$$(\mathbf{w}^T \mathbf{x}_i + b) - y_i \geq \epsilon - \xi_{i_2} \tag{1.44}$$

$$\xi_{i_1} \geq 0 \tag{1.45}$$

$$\xi_{i_2} \geq 0 \tag{1.46}$$

The slack variables are divided into 2 kind of 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:

$$\underset{\boldsymbol{\alpha}}{\operatorname{argmax}} \left(\sum_{i=1}^{n} y_i(\alpha_{i_1} - \alpha_{i_2}) - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} (\alpha_{i_1} - \alpha_{i_2})(\alpha_{j_1} - \alpha_{j_2})(\mathbf{x}_i . \mathbf{x}_j) \right) \tag{1.47}$$

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

$$\sum_{i=1}^{n} \alpha_{i_1} = \sum_{i=1}^{n} \alpha_{i_2} \tag{1.48}$$

$$0 \leq \alpha_{i_1} \leq C \tag{1.49}$$

$$0 \leq \alpha_{i_2} \leq C \tag{1.50}$$

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

$$f(\mathbf{x}_j) = \mathbf{w}^T \mathbf{x}_j + b \tag{1.51}$$

$$f(\mathbf{x}_j) = \sum_{i=1}^{n} (\alpha_{i_1}^* - \alpha_{i_2}^*)(\mathbf{x}_i . \mathbf{x}_j) + b \tag{1.52}$$

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

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

1.3 Conclusion of the chapter

This chapter reviews the different steps in a machine learning framework: data normalization, model selection and model evaluation. We focus on two machine learning algorithms: the k-Nearest Neighbors (k-NN) and the Support Vector Machine (SVM). In the following, we consider the k-NN as our classifier. The SVM will be used for its large margin concept, a key part of one of our algorithms.

Our objective being the learning of a metric that optimizes the performances of the k-NN classifier, we review in the next section some metrics proposed for time series.

Chapter 2

Time series metrics

Contents

In this chapter, we first present the definition of time series. Then, we recall the general properties of a metric and introduce some metrics proposed for time series. In particular, we focus on amplitude-based, behavior-based and frequential-based metrics. As real time series are subject to varying size and delays, we recall the concept of alignment and dynamic programming. Finally, we present some proposed combined metrics for time series.

2.1 Definition of a time series

Time series are data that can be frequently 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, detecting 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). Due to their temporal and structured nature, time series constitute complex data to be analyzed by classic machine learning approaches.

For physical systems, a time series of duration T can be seen as a signal, sampled at a frequency f_e, in a temporal window $[0;T]$. From a mathematical perspective, a time series of length q is a collection of a finite number of observations made sequentially at discrete time instants $t = 1, ..., q$. Note that $q = Tf_e$.

Let $\mathbf{x}_i = (x_{i1}, x_{i2}, ..., x_{iq})$ be a univariate time series of length q. 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} $(t = 1, ..., q)$ comes from the observation of one variable (*e.g.*, the temperature measured by one sensor). When simultaneous observation of p variables (several sensors such as the temperature, the pressure, etc.) are made at the same time, the time series is said to be multivariate. From this, one possible representation would be (Fig. 2.1, red circle):

$$\mathbf{x}_i = (x_{i1,1}, ..., x_{i1,p}, x_{i2,2}, ..., x_{i2,2}, ..., x_{iq,p})$$

where $\mathbf{x}_{it,j}$ is the observation of the time series $\mathbf{x}_i$ at the time instant t along the variable x_j. An other possible representation could consider a multivariate time series $\mathbf{x}_i$ as the union of multiple univariate time series (Fig. 2.1, green circle). In this case:

$$\mathbf{x}_i = (\mathbf{x}_{i,1}, ..., \mathbf{x}_{i,p}) = (x_{i1,1}, ..., x_{iQ,1}, x_{i1,2}, ..., x_{i1,p}, ..., x_{iq,p})$$

where $\mathbf{x}_{i,j} = (\mathbf{x}_{i1,j}, \ldots, \mathbf{x}_{iq,j})$. For simplification purpose, we only consider univariate time series in the following.

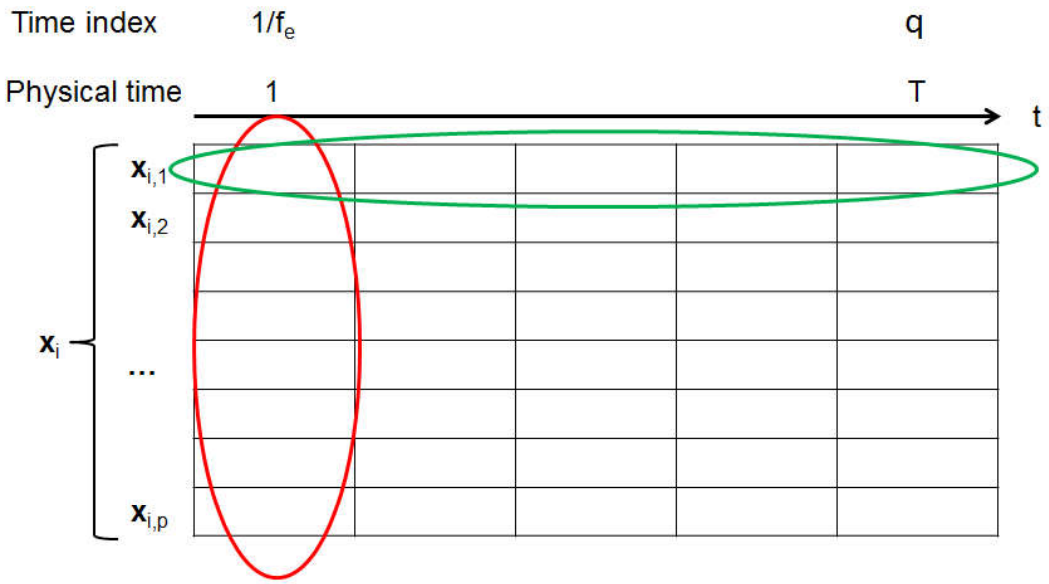


Figure 2.1: Two examples of representation of multivariate time series

Some authors propose to extract representative features from time series. Fig. 2.2 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).

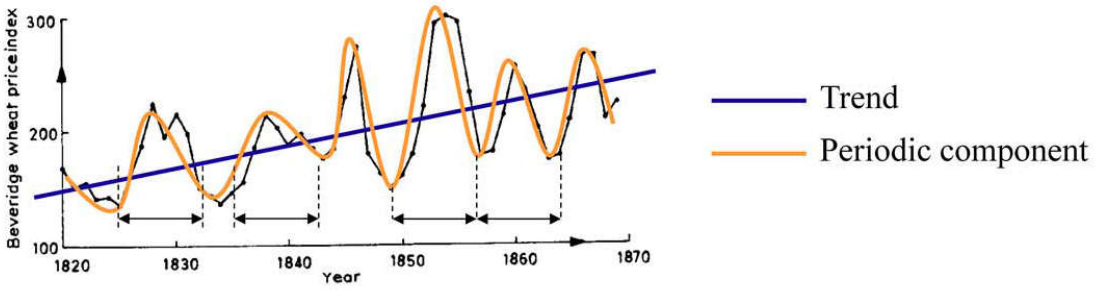


Figure 2.2: The Beveridge wheat price index is the average in nearly 50 places in various countries measured in successive years from 1500 to 1869[1].

According to Chatfield, most time series exhibit either or both a long term change in the mean (trend) and a periodic (cyclic) component. The trend can be linear, quadratic, etc. The cyclic component is a variation at a fixed period of time (seasonality) such as for example the seasonal variation of temperature. In practice, the 3 features (trend, cycle, residuals) are rarely sufficient for the classification or regression of real time series.

Other authors made the hypothesis of time independency between the observations x_{it}. They consider time series as a static vector data and use classic machine learning algorithms [Lia+12]; [CT01]; [HWZ13]; [HHK12]. Our work focuses on classification problems, and on time series comparison through metrics.

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

2.2 Properties and representation of a metric

A mapping $D : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}^+$ over a vector space $\mathbb{R}^p$ is called a **metric** or a distance if for all vectors $\forall\ \mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_l \in \mathbb{R}^p$, it satisfies the properties [DD09]:

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. $D(\mathbf{x}_i, \mathbf{x}_j) + D(\mathbf{x}_j, \mathbf{x}_l) \geq D(\mathbf{x}_i, \mathbf{x}_l)$ (triangular inequality)

We call reflexivity, the property: $\mathbf{x}_i = \mathbf{x}_j \Rightarrow D(\mathbf{x}_i, \mathbf{x}_j) = 0$.

A mapping D that satisfies at least properties 1, 2 and reflexivity is called a **dissimilarity,** and the one that satisfies at least properties 1, 2, 4 a **pseudo-metric**. A metric, a dissimilarity and a pseudo metric can be both interpretated as a measure of how "different" two samples are: if a sample $\mathbf{x}_i$ is expected to be closer to $\mathbf{x}_j$ than to $\mathbf{x}_l$, then $D(\mathbf{x}_i, \mathbf{x}_j) \leq D(\mathbf{x}_i, \mathbf{x}_l)$. A mapping $S : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$ is called a **similarity** on $\mathbb{R}^p$ if S satisfies the properties of positivity, symmetry and the inequality: $S(\mathbf{x}_i, \mathbf{x}_j) \leq S(\mathbf{x}_i, \mathbf{x}_i)$. To simplify the discussion in the following, we refer to pseudo-metric and dissimilarity as metrics, pointing out the distinction only when necessary.

Metric can be represented in two ways (Fig. 2.3). First, in Fig. (a), data points (samples $\mathbf{x}_i$ and $\mathbf{x}$) can be fixed and the distance sphere is shown for each metric (*e.g.*, the Manhattan, Euclidean and Infinite distance). Secondly, by fixing a data point $\mathbf{x}_i$, the distance sphere is fixed and the data point $\mathbf{x}$ changes according to the considered distance at hand ($\mathbf{x}_{Manhattan}$ for the Manhattan distance, $\mathbf{x}_{Euclidean}$ for the Euclidean distance, $\mathbf{x}_{Infinite}$ for the Infinite distance). For example, the latter representation is used by Weinberger and Saul in [WS09a] to illustrate the effect of an initial Euclidean distance and a learned Mahalanobis metric to purify the neighborhood of data points. This concept will be explored in Chapter 3.

Given the pairwise dissimilarities between samples, some algorithms such as MultiDimensional Scaling (MDS) [CA80] or Isomap [GZZ05] have been proposed to visualize the proximity between samples in a dataset. Briefly, a MDS algorithm aims to place each sample in a P-dimensional space (in general, $P = 2$ or 3) such that the between-object distances are preserved as well as possible. Classical MDS takes an input matrix giving dissimilarities between pairs of samples and outputs a coordinate for each sample whose configuration minimizes a loss function called stress. An example of applications is given in Fig. 2.4: the distances between pairs of cities is given and the aim is to reconstruct a two dimensional map that reproduces the best the given distances. More generally, we can take benefice of this algorithm for any other type of data (samples, time series, etc.) if the given dissimilarity matrix is given.

[2]source: `http://www.bristol.ac.uk/media-library/sites/cmm/migrated/documents/chapter3.pdf`

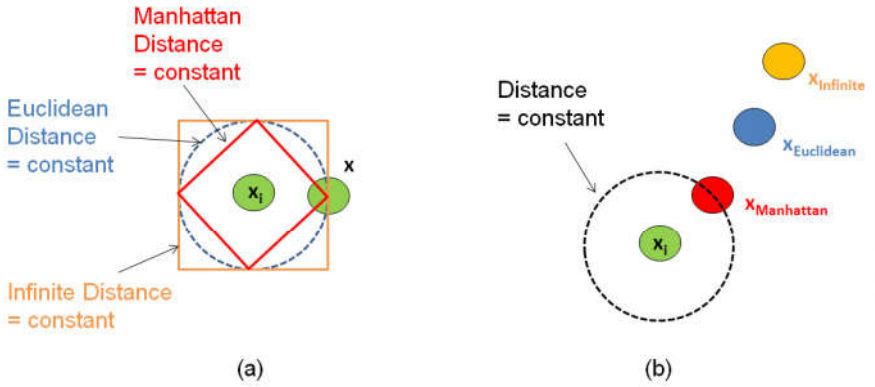
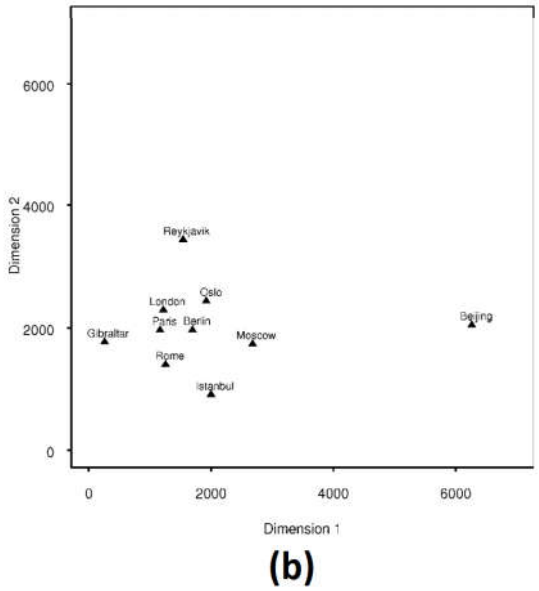


Figure 2.3: Example of metric representation: (a) Data points are fixed and the distance sphere is shown for each metric given a constant. (b) The distance sphere is fixed and the data points **x** are moving according to each distance.

	London	Berlin	Oslo	Moscow	Paris	Rome	Beijing	Istanbul	Gibraltar	Reykjavik
London	–									
Berlin	570	–								
Oslo	710	520	–							
Moscow	1550	1000	1020	–						
Paris	210	540	830	1540	–					
Rome	890	730	1240	1470	680	–				
Beijing	5050	4570	4360	3600	5100	5050	–			
Istanbul	1550	1080	1520	1090	1040	850	4380	–		
Gibraltar	1090	1450	1790	2410	960	1030	6010	1870	–	
Reykjavik	1170	1480	1080	2060	1380	2040	4900	2560	2050	–

(a)



(b)

Figure 2.4: Example of MDS. (a) Distances between ten cities in miles. (b) Two dimensional plot of the ten cities from a classical MDS.[2]

2.3 Unimodal metrics for time series

In the following, we suppose that time series have the same duration T and have been regularly sampled at the frequency f_e. Therefore, they have the same length $q = Tf_e$. Let $\mathbf{x}_i = (x_{i1}, x_{i2}, ..., x_{iq})$ and $\mathbf{x}_j = (x_{i1}, x_{i2}, ..., x_{iq})$ be two univariate time series of length q.

2.3.1 General review of unimodal metrics for time series

Defining and evaluating metrics for time series has become an active area of research for a wide variety of problems in machine learning [Din+08]; [Naj+12]. For example, as explained in [MV14], a crucial question in clustering is to determine what we mean by "similar" samples, *i.e.*, defining a suitable metrics between two samples. The idea is not restricted to clustering and can be naturally extended to other machine learning problems (supervised, semi-supervised). Due to their temporal nature, time series may be compared based on different characteristics, called **modalities**, such as their amplitude, shape or frequency. Contrary to static data, time series may be also subjected to temporal specificities such as delays. From the surge of research, one can identify at least three categories: metrics in the time domain, metrics in a feature-extracted domain, metrics based on models.

The first category (metrics in the time domain) aims to compare the closeness of time series observations in the temporal domain. Without being exhaustive, many variants cover the Minkowski distance or the Frechet distance [Mau06] for amplitude comparison, and the correlation-based distances [DCA11]; [Ben+09] for behavior comparison. The second category (metrics in a feature extracted domain) aims to project the time series in an other domain, such as the frequential domain or the symbolic representation, and to compute a distance in the projected domain. For frequential-based distance, many measures have been proposed such as the periodogram-based distance [CCP06] or the dissimilarity measure based on the discrete wavelet transform [Cha+08]. Based on the symbolic representation SAX (Symbolic Aggregate approXimation), a dissimilarity measure between symbols have been proposed in [Lin+03]. In the third category (metrics based on models), the aim is to assume a model of the time series and to compute the dissimilarity between the evaluated models. Most of the propositions assume that time series are generated by either an ARIMA (AutoRegressive Integrated Moving Average) or ARMA (AutoRegressive Moving Average) process [KGP01]; [Mar00]. Some work have considered alternative models such as the Markov chains [RSC02] or the hidden Markov models [Smy97]. Based on the ARIMA or ARMA model, a number of metrics have been proposed, such as the Piccolo distance [Pic90] or the Maharaj distance [Mah96], which evaluate the distance between the parameters of the estimated model.

In the following, we focus on three types of metrics for time series, two in the time domain (amplitude-based and behavior-based distance) and one in the frequential domain (frequential-based distance).

2.3.2 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]:

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

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], defined as a dissimilarity measure weighted by a matrix $\mathbf{M}$:

$$d_M(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{M}^{-1} (\mathbf{x}_i - \mathbf{x}_j) \tag{2.2}$$

If the covariance matrix $\mathbf{M}$ is the identity matrix, the Mahalanobis distance is equal to the Euclidean distance. If the covariance matrix $\mathbf{M}$ is diagonal, then the resulting distance measure is called the normalized Euclidean distance:

$$d_M(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{t=1}^{q} \frac{(x_{it} - x_{jt})^2}{m_t}} \tag{2.3}$$

where m_t is the variance of the x_{it} and x_{jt} over the sample set. Note that this is equivalent to normalize each feature: $x'_{it} = x_{it}/\sqrt{m_t}$ and use the Euclidean distance on the normalized features x'_{it}. In the following of the work, we consider the standard Euclidean distance d_E as the amplitude-based distance d_A.

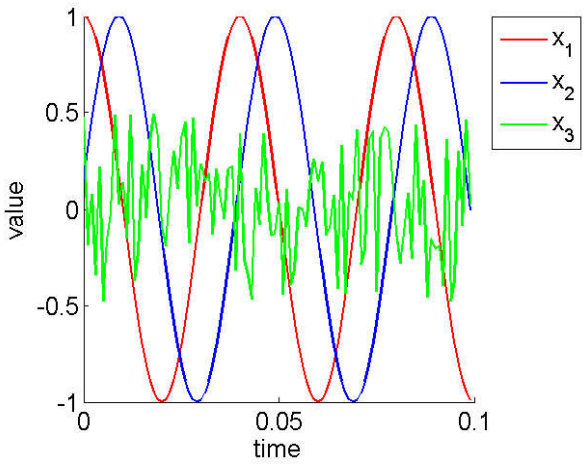


Figure 2.5: 3 toy time series in the temporal domain. Time series in blue and red are two sinusoidal signals. Time series in green is a random signal.

In the example of Fig. 2.5, let's try to determine 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 $\mathbf{x}_1$ is closer to $\mathbf{x}_3$ than to $\mathbf{x}_2$ since $d_A(\mathbf{x}_1, \mathbf{x}_3) = 24.1909 < d_A(\mathbf{x}_1, \mathbf{x}_2) = 29.0818$.

2.3.3 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 [SS12b]; [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 components at frequential index $f = \{1, \ldots, F\}$. The Euclidean distance is then used between their respective complex number modules $|\tilde{x}_{if}|$:

$$d_F(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{f=1}^{F} (|\tilde{x}_{if}| - |\tilde{x}_{jf}|)^2} \tag{2.4}$$

In the example of Fig. 2.5, recall that the Euclidean distance d_A states that $\mathbf{x}_1$ is closer to $\mathbf{x}_3$ than $\mathbf{x}_2$. However, in the frequency domain (Fig. 2.6), the frequential-based distance d_F states that $\mathbf{x}_1$ is closer to $\mathbf{x}_2$ than to $\mathbf{x}_3$ since $d_F(\mathbf{x}_1, \mathbf{x}_2) = 0.0835 < d_F(\mathbf{x}_1, \mathbf{x}_3) = 1.0124$.

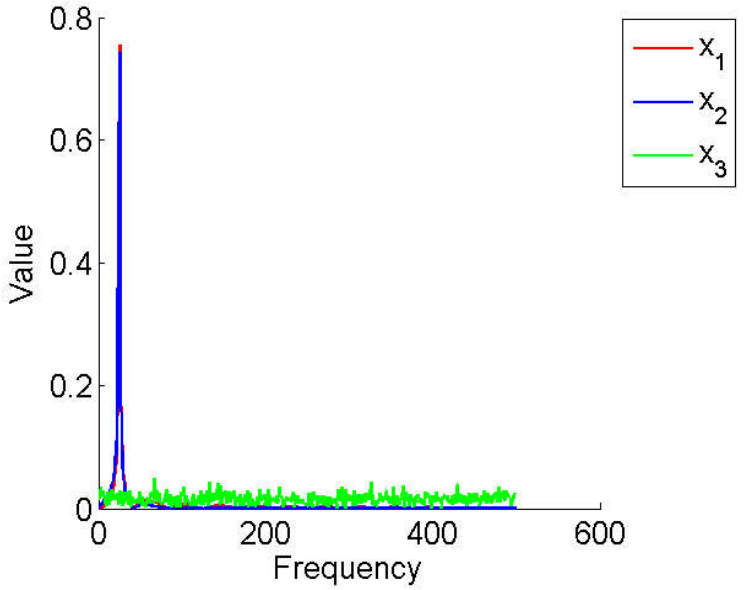


Figure 2.6: 3 toy time series in the frequency domain: blue and red are the spectrum of the Fourier transform of two sinusoidal signals; green is the spectrum of the Fourier transform of a random signal.

2.3.4 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]:

$$cort_r(\mathbf{x}_i, \mathbf{x}_j) = \frac{\sum_{t,t'=1}^{q} (x_{it} - x_{it'})(x_{jt} - x_{jt'})}{\sqrt{\sum_{t,t'=1}^{q} (x_{it} - x_{it'})^2} \sqrt{\sum_{t,t'=1}^{q} (x_{jt} - x_{jt'})^2}} \tag{2.5}$$

where $|t - t'| \leq r$, $r \in [1, ..., q-1]$. 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, *i.e.*, there exists some constant c such that $\mathbf{x}_i = \mathbf{x}_j + c$. The value $cort_r(\mathbf{x}_i, \mathbf{x}_j) = -1$ means that $\mathbf{x}_i$ and $\mathbf{x}_j$ have opposite behavior, *i.e.*, there exists some constant c such that $\mathbf{x}_i = -\mathbf{x}_j + c$. Finally, $cort_r(\mathbf{x}_i, \mathbf{x}_j) = 0$ expresses that their growth rates are stochastically linearly independent (different behaviors).

When $r = q - 1$, it leads to the Pearson correlation. As $cort_r$ is a similarity measure, it can be transformed into a dissimilarity measure:

$$d_B(\mathbf{x}_i, \mathbf{x}_j) = \frac{1 - cort_r(\mathbf{x}_i, \mathbf{x}_j)}{2} \tag{2.6}$$

Considering Fig. 2.7, the behavior-based metric d_B states that $\mathbf{x}_1$ is closer to $\mathbf{x}_4$ than to $\mathbf{x}_2$ or $\mathbf{x}_3$ since $d_B(\mathbf{x}_1, \mathbf{x}_2) = 0.477$, $d_B(\mathbf{x}_1, \mathbf{x}_3) = 1$ and $d_B(\mathbf{x}_1, \mathbf{x}_4) = 0$.

2.4 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

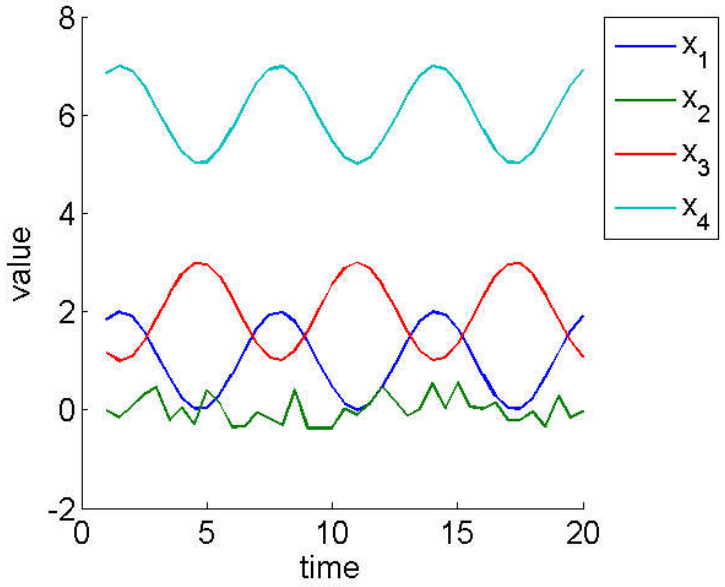


Figure 2.7: The signal from Fig. 2.5 and a signal $\mathbf{x}_4$ which is signal $\mathbf{x}_1$ and an added translation. Based on behavior comparison, $\mathbf{x}_4$ is the closest to $\mathbf{x}_1$.

of various natures: time shifting (phase shift in signal processing), time compression or time dilatation. For example, in the case of voice recognition (Fig. 2.8), 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.

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

$$\boldsymbol{\pi}_{ij} = ((\pi_i(1), \pi_j(1)), (\pi_i(2), \pi_j(2)), \ldots, (\pi_i(m), \pi_j(m))) \tag{2.7}$$

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

$$1 = \pi_i(1) \leq \pi_i(2) \leq ... \leq \pi_i(m) = q \tag{2.8}$$
$$1 = \pi_j(1) \leq \pi_j(2) \leq ... \leq \pi_j(m) = q \tag{2.9}$$

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

$$\pi_i(l+1) \leq \pi_i(l) + 1 \tag{2.10}$$
and $$\pi_j(l+1) \leq \pi_j(l) + 1 \tag{2.11}$$
and $$(\pi_i(l+1) - \pi_i(l)) - (\pi_j(l+1) - \pi_j(l)) \geq 1. \tag{2.12}$$

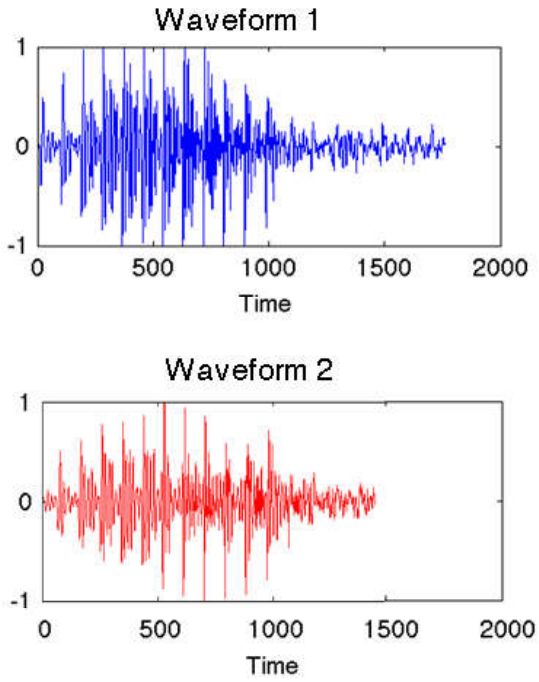


Figure 2.8: Example of a same sentence said by two different speakers. Time series are shifted, compressed and dilatated in the time.

In the following, we denote $\boldsymbol{\pi} = \boldsymbol{\pi}_{ij}$ to simplify the notation. Intuitively, an alignment $\boldsymbol{\pi}$ defines a way to associate elements of two time series. Alignments can be described by paths in the $q \times q$ grid that crosses the elements of $\mathbf{x}_i$ and $\mathbf{x}_j$ (Fig. 2.9). We denote $\boldsymbol{\pi}$ a valid alignment and A_{ij}, 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.). Standard 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:

$$\boldsymbol{\pi}^* = \underset{\boldsymbol{\pi} \in A_{ij}}{\operatorname{argmin}} \frac{1}{|\boldsymbol{\pi}|} \sum_{(t,t') \in \boldsymbol{\pi}} \varphi(x_{it}, x_{jt'}) \tag{2.13}$$

When the cost function φ is a similarity measure (Section 2.2), the optimization involves maximization instead of minimization. When other constraints are applied on $\boldsymbol{\pi}$, Eq. (2.13) 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:

$$\mathbf{x}_{i,\boldsymbol{\pi}^*} = (x_{i\pi_i(1)}, ..., x_{i\pi_i(m)}) \tag{2.14}$$

$$\mathbf{x}_{j,\boldsymbol{\pi}^*} = (x_{j\pi_j(1)}, ..., x_{j\pi_j(m)}) \tag{2.15}$$

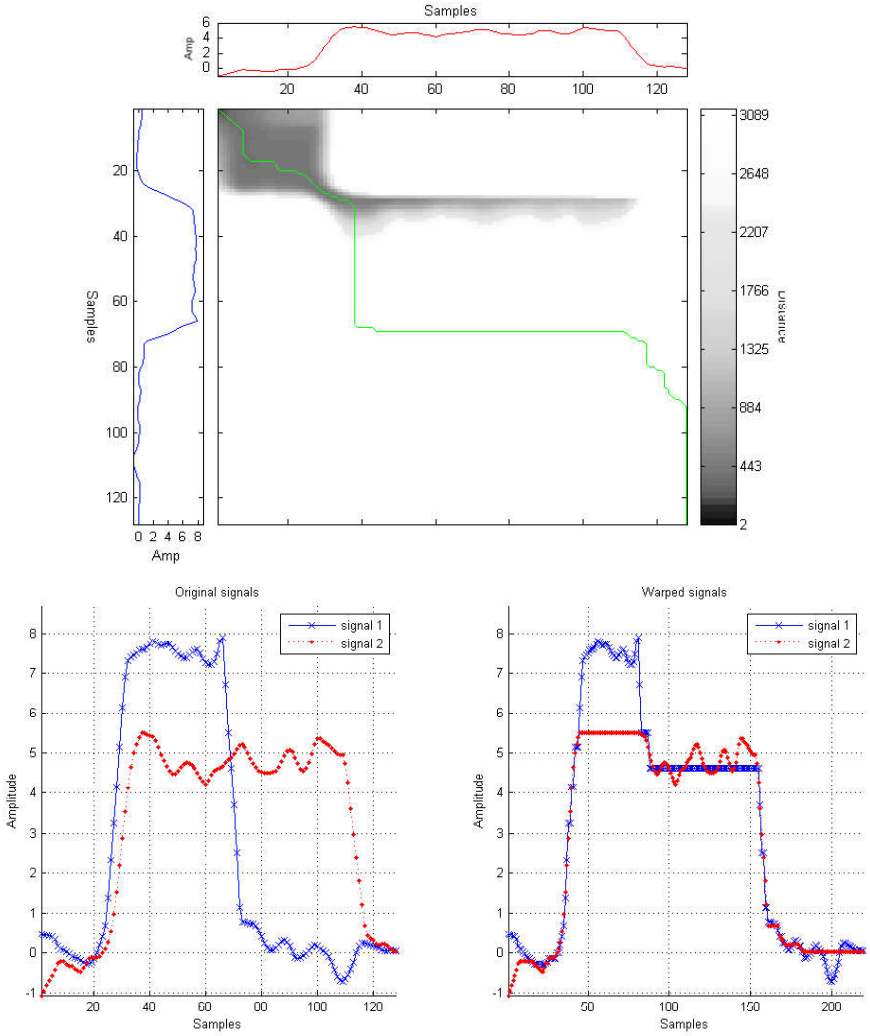


Figure 2.9: Example of DTW grid between 2 time series $\mathbf{x}_i$ and $\mathbf{x}_j$ (top) and the signals before and after warping (bottom). On the DTW grid, the two signals can be represented on the left and bottom of the grid. The optimal path $\boldsymbol{\pi}^*$ is represented in green line and shows how 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)

Once an optimal alignment $\boldsymbol{\pi}^*$ has been found, and whatever cost function φ have been chosen to find it, the metric presented in Section 2.3 (amplitude-based d_A, behavior-based d_B, frequential-based d_F) 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 to $\mathbf{x}_{i,\boldsymbol{\pi}^*}$ and $\mathbf{x}_{j,\boldsymbol{\pi}^*}$ as $\mathbf{x}_i$ and $\mathbf{x}_j$.

2.5 Combined metrics for time series

2.5.1 Combination functions

In most classification problems, it is not known *a priori* if time series of a same class exhibits same modalities (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 [Zha+13], 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 basic combination functions that we could use combines two unimodal metrics through a linear or geometric function. For example, with d_A and d_B, we obtain:

$$D_{Lin}(\mathbf{x}_i, \mathbf{x}_j) = \beta d_B(\mathbf{x}_i, \mathbf{x}_j) + (1-\beta) d_A(\mathbf{x}_i, \mathbf{x}_j) \tag{2.16}$$

$$D_{Geom}(\mathbf{x}_i, \mathbf{x}_j) = (d_B(\mathbf{x}_i, \mathbf{x}_j))^{\beta} (d_A(\mathbf{x}_i, \mathbf{x}_j))^{1-\beta} \tag{2.17}$$

where $\beta \in [0; 1]$ defines the trade-off between the amplitude d_A and the behavior d_B components, and is thus application dependent. For example, in classification problems, this parameter can be learned through a grid search procedure (Section 1.1.2). 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 components through a sigmoid combination function [DCA11]:

$$D_{Sig}(\mathbf{x}_i, \mathbf{x}_j) = \frac{2 d_A(\mathbf{x}_i, \mathbf{x}_j)}{1 + \exp(\beta cort_r(\mathbf{x}_i, \mathbf{x}_j))} = \frac{2 d_A(\mathbf{x}_i, \mathbf{x}_j)}{1 + \exp(\beta(1 - 2 d_B(\mathbf{x}_i, \mathbf{x}_j)))} \tag{2.18}$$

where β is a parameter that defines the compromise between behavior and amplitude components.

Fig. 2.10 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 β. For small value of β ($\beta = 0$), the three metrics only includes d_A. For high value of β ($\beta = 1$), D_{Lin} and D_{Geom} only includes d_B. For $\beta = 6$ and for small values of d_B, D_{Sig} mostly includes d_B while for large value of d_B, D_{Sig} mostly includes d_A.
Note that these combinations are fixed and defined independently from the analysis task at hand. Moreover, in the case of D_{Sig}, the component $cort_r$ can be seen as a penalizing factor of d_A. Finally, one could extend D_{Lin} and D_{Geom} by adding metrics, but that would imply to add parameters. The grid search to find the best parameters would become time consuming.

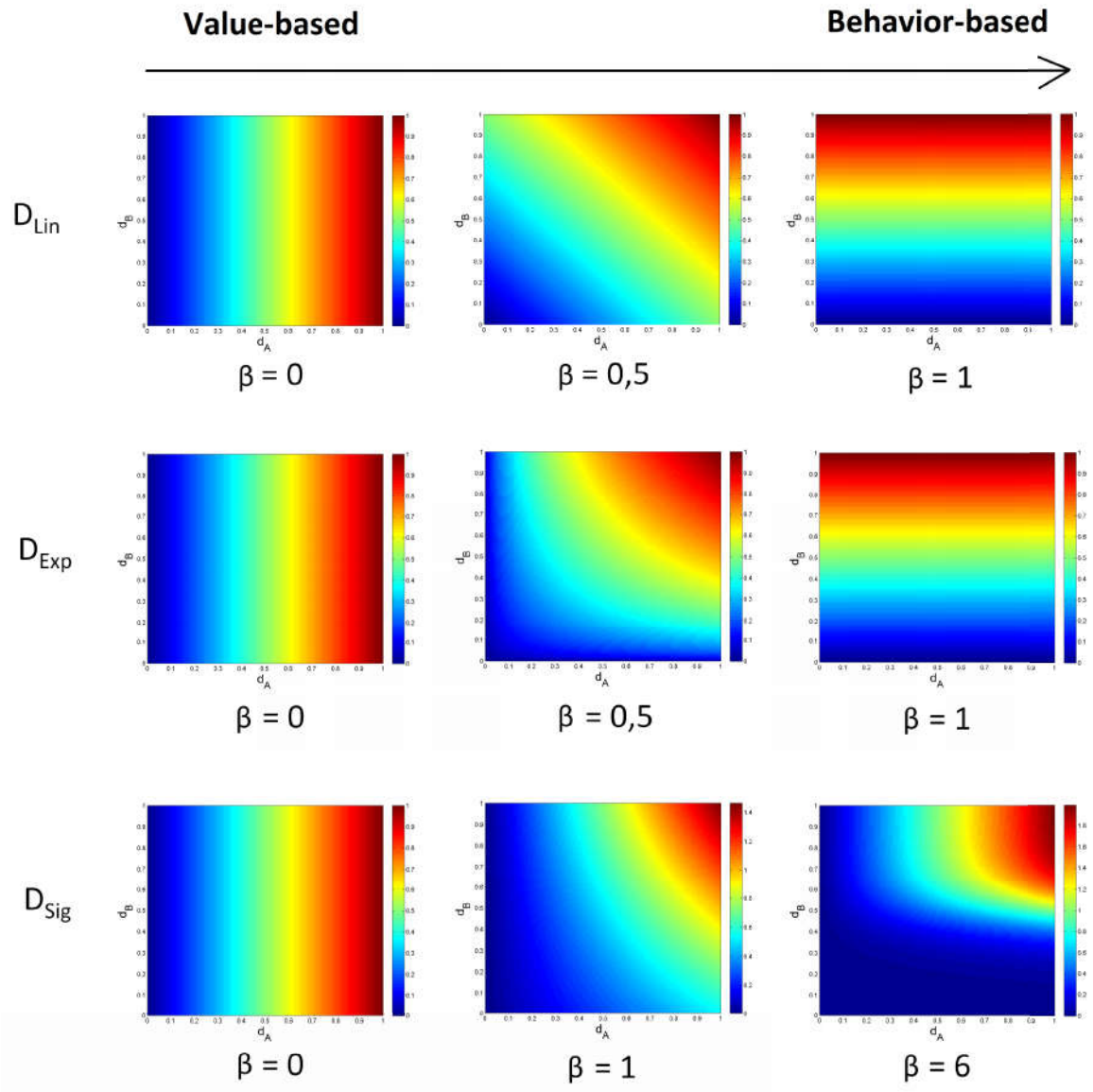


Figure 2.10: Contour plot of the resulting combined metrics: D_{Lin} (1^{st} line), D_{Geom} (2^{nd} line) and D_{Sig} (3^{rd} line), for different values of β. For the three combined metrics, the first and second dimensions are respectively the amplitude-based metrics d_A and the behavior-based metric d_B.

2.5.2 Impact of normalization

When combining several metrics into a single metric, it is necessary to normalize the metrics involved in the combination. Classically, to normalize data, Z-normalization is used (Section 1.1.4). In that case, we suppose that the variables x_j are 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 distances. These data may be log-normally

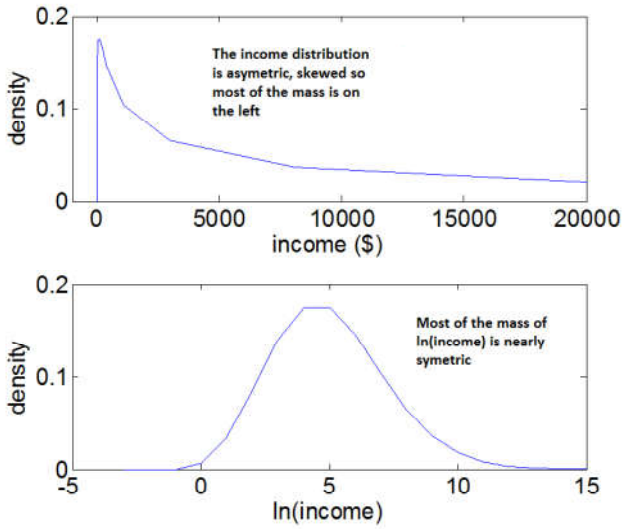


Figure 2.11: A nearly log-normal distribution, and its log transform[3]

distributed, *e.g.*, the log of the data is normally distributed (Fig. 2.11). Some works proposes to take the log of the data (x_j^{ln}) to restore the symmetry, and then, to apply a Z-normalization of this transformation [ZMP14]:

$$x_j^{ln} = \ln(x_j); \tag{2.19}$$

$$x_j^{ln,norm} = \frac{x_j^{ln} - \mu_j^{ln}}{\sigma_j^{ln}} \tag{2.20}$$

$$x_j^{norm} = \exp(x_j^{ln,norm}) \tag{2.21}$$

where ln denotes the Natural Logarithm function, μ_j^{ln} and σ_j^{ln} the mean and the standard deviation of a variable x_j^{ln}.

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

2.6 Conclusion of the chapter

To cope with modalities (characteristics) inherent to time series (amplitude, behavior, frequency, etc.), we review in this chapter several unimodal metrics for time series, in particular, the Euclidean distance d_A, the behavior-based distance d_B and the Fourier-based distance d_F. In practice, real time series may be subjected to delays and need to be re-aligned before any analysis task. For that, the Dynamic Time Warping (DTW) algorithm is used in practice. However, the unimodal metrics (d_A, d_B, d_F) only include one modality. In general, several modalities may be implied and some combined metric have been proposed, but the propositions are often limited to two modalities and the metrics are defined independently from the analysis task.

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, inspired from these ideas, we propose framework to learn a combined metric for a large margin k-NN classifier of time series.