Fusion Methods for Time-Series Classification

A thesis submitted for the degree of

Doctor of Natural Science (Dr. rer. nat.)

by

Dipl.-Ing. Krisztian Antal Buza

Department of Computer Science
Information Systems and Machine Learning Lab (ISMLL)
UNIVERSITY OF HILDESHEIM

UPDATE 2
(27 AUG 2011)

2011

History

UPDATE 1 (18 AUG 2011) – Updated according to the formatting requests of the publisher (Peter Lang Verlag), e.g. Abstract and Erklärung are deleted, etc.

UPDATE 2 (27 AUG 2011) – Updated based on native-English proofreading.

Acknowledgments

It is hardly possible to recollect the names of all the persons who directly or indirectly inspired my research through discussions, conference talks, or in an other way. Therefore, the acknowledgments below are limited to the persons I directly cooperated with, who advised me, co-authored papers, with whom I had focused discussions on the topics related to my thesis.

First of all, I would like to thank to my supervisior, Lars Schmidt-Thieme in whose group I could spend four fruitful years at the University of Hildesheim. I acknowledge him for advising my work, for all the research talks, and his comments and suggestions. I would like to acknowledge Alexandros Nanopoulos for co-supervising my thesis. I thank him for his comments, remarks and suggestions on the papers we wrote.

I would like to acknowledge all my co-authors, especially Tomas Horváth for his remarks on the GRAMOFON framework. I would like to thank to Leandro Balby Marinho with whom we worked together on ontology induction, Philipp Cimiano and Sebastian Blohm with whom we explored relation extraction from natural language texts, Lucas Drumond and Timo Reuter furthermore Claudio Guiliano and Lorenza Romano with whom we developed clustering methods for images (according to events) and web pages (according to persons) respectively.

I would like to thank to Christine Preisach and Andre Busche for the beautiful years we worked together on the X-Media project, in which we explored aeroplane vibration analysis.

I would like to thank to Julia Koller for the discussions about electrocardiograph (ECG) signals and their medical applications.

Last, but not least, I would like to acknowledge Jessica Faruque for careful proof-reading and helping me with the “secrets” of the English language.

Finally, I thank to all the persons who indirectly inspired my work.

I acknowledge Prof. Eamonn Keogh for making time-series classification datasets available. I acknowledge Netflix for the datasets of the RMSE task of the Ensembling Challenge of the Australian Data Mining Conference 2009. I also acknowledge the UCI Machine Learning repository for the usage of their datasets [69].

Contents

Notations

For better readability the notations used throughout this book and the ones used in single chapters are listed separately.

Notations and abbreviations used throughout this book

x	a time series (instance of a dataset), or a segment of a time series
i, j	index variables
n, m	variables used to denote the size (length) of sets (sequences)
$x[i]$	the i-th value of the time series x
x_i	the i-th time series in a dataset of time series
$c(x)$	class label of time series x
$\mathcal{D}$	a labeled dataset (containing time series)
$\lvert\mathcal{D}\rvert$	Size of the data set $\mathcal{D}$ (number of instances contained in $\mathcal{D}$)
f	classifier
mod	remainder of the division (modulo), e.g. $12 \bmod 5 = 2$
$d(x_1, x_2)$	Distance of two time series x_1 and x_2.
$d_{EU}(x_1, x_2)$	Euclidean distance of two time series x_1 and x_2. (See Section 2.3.1)
$d^F_{EU}(x_1, x_2)$	Euclidean distance of Fourier-coefficients of two time series x_1 and x_2. (See Section 2.3.1)
$d^W_{EU}(x_1, x_2)$	Euclidean distance of the Haar Wavelet Transform of two time series x_1 and x_2. (See Section 2.3.1)
DTW	Dynamic Time Warping
$d_{DTW}(x_1, x_2)$	Dynamic Time Warping distance of two time series x_1 and x_2. (See Section 2.3.2)
$d_0^{DTW}(i, j)$	Entries of the DTW-matrix (See Section 2.3.2)
c_{el}^{DTW}	Cost of elongation in the DTW algorithm (See Section 2.3.2)
c_{tr}^{DTW}	Cost of transformation (mathing) in the DTW algorithm (See Section 2.3.2)
w^{DTW}	Warping window size in the DTW algorithm (See Section 2.3.2)

$\lfloor r \rfloor$	Floor(r) – The largest integer number r_0 so that $r_0 \leq r$ (r is a real number)
$\lceil r \rceil$	Ceil(r) – The smallest integer number r_0 so that $r_0 \geq r$ (r is a real number)
$g_N^k(x)$	Number of time series that have x among their k nearest neighbors
$g_G^k(x)$	Number of time series having x among their good k nearest neighbors
$g_B^k(x)$	Number of time series having x among their bad k nearest neighbors
$\mu_{g(x)}$	Mean of $g(x)$
$\sigma_{g(x)}$	Standard deviation of $g(x)$
$\mathcal{S}_{g(x)}$	Skewness of $g(x)$
n_m	Maximal number of allowed gaps in semi-continuous motifs
d_m	Maximal allowed length of a gaps in semi-continuous motifs

Notations used in Chapter 2

$\mathcal{T}$	set of all considered time series
$c_j^F(x)$	the j-th Fourier-coefficient of the time series x ($j \in \mathbb{Z}$)
$\mathcal{I}$	Imaginary unit: $\sqrt{-1}$
$DFT(x)$	Discrete Fourier Transform of x (See Section 2.2.1)
$FFT(x)$	Fast Fourier Transform of x (See Section 2.2.1)
$DWT(x)$	Discrete Wavelet Transform of x (See Section 2.2.2)
$HWT(x)$	Haar Wavelet Transform of x (See Section 2.2.2)
$SAX(x)$	Symbolic Aggregate Approximation of x (See Section 2.2.3)
ϵ^{EDR}	Similarity threshold for EDR (See Section 2.3.3
l_m	Predefined minimal length of motifs of interest (see Section 2.4.4)

Notations used in Chapter 3

$q(x,k)$	Quality score associated with a time series x and k-NN classifier
$M_{i,j}^*$	Meta models for individual quality estimation
p	Noise ratio

Notations used in Chapter 4

N	Number of selected instances
G_c	Coverage graph
V_G	Set of vertices of the graph G
E_G	Set of edges of the graph G

Notations used in Chapter 4 (cont.)

v	Vertex of the coverage graph
$C(v), C(S)$	Coverage of a vertex v and of a vertex-set S respectively
$X, \mathcal{F}$	X is a finite set, $\mathcal{F}$ is a family of subsets of X (Set-Coverage Problem in Section 4.2.1)
w	Weight of an edge (in a weighted coverage graph in Section 4.2.3)

Notations used in Chapter 5

$\mathcal{M}$	Regression model that is used to construct the fused distance measure
$\mathcal{I}(x_1, x_2)$	Indicator that shows whether two time series belong to the same class

Notations used in Chapter 6

$y(x)$	Numeric label of the instance x
$m(x)$	Model (regressor) that estimates the numeric label of instance (time series) x
m.predict($\mathcal{D}$)	vector containing the predictions of m for *all* the instances of dataset $\mathcal{D}$
$\mathcal{D}$.labels	vector containing the labels of *all* the instances of dataset $\mathcal{D}$
M	a set of models (regressors)
$\{p_i\}$	matrix consisting of column vectors $p_i = m_i$.predict(D) for all $m_i \in M$
$m^*(x)$	Meta-model (meta-regressor) that estimates the numeric label of instance (time series) x
N	number of considered models that is equal to the number of nodes in the model-network
n, ϵ	hyperparametes of GRAMOFON (see section 6.2.2)
C	complexity constant of a support vector machine (SVM)
e	exponent of the polynomial kernel used in an SVM
γ	hyperparameter of an RBF kernel used in an SVM

Notations used in Chapter 7

$\mathcal{D}_{SAX}$	Converted time series dataset (time series are converted to a sequence of discrete symbols with SAX)
Σ	Set of symbols
T_Σ	Taxonomic relation of the symbols in Σ
s	Mimimum support threshold

Part I

Preliminaries

Chapter 1

Introduction

Time-series classification is the theoretical background of many recognition tasks such as signature verification [77], [118], speech and handwriting recognition [124], [131], [153]. While, for example, we are writing a character on the touch screen of a mobile telephone or PDA, the device records the pen tip's position in consecutive moments of time, e.g. 100 times within a second. The pen tip's position can be described quantitatively by the horizontal and vertical coordinates of the point where the screen is currently touched. As the position is recorded in consecutive moments of time, as result, a sequence of measured numerical values (horizontal and vertical coordinates) is produced. Based on this information, the device is able to automatically recognize which character was written by the user.

Similar recognition tasks appear in numerous applications in various domains such as finance, medicine, biometrics, chemistry, astronomy, robotics, networking and industry[1] [95]. In medicine, for example, time-series classification can be applied in the analysis of images, brainwaves (EEG) and electrocardiograph (ECG) signals [21], [96], [116], [125].

The problem of time-series classification is challenging for many reasons. Two crucial requirements of successful classification are accuracy and recognition time. While in some of the above applications, one of these two basic criteria might be more important than the other – for example, in the case of person identification, accuracy might be more important than recognition time – in general, we require algorithms that fulfill both of them. Irregularities in the electrocardiograph signals should be recognized accurately *before* serious, irreversible deteriorations of the patient's health status; similarly, possible damages of an aircraft engine [35] should be found *before* the engine becomes unusable.

The above requirements, accuracy and recognition time, are often contradictory: the more accurate the recognition, usually, the longer the execution time. Keeping this well-known trade-off between quality and execution time in mind, I address both of these challenges in this work by aiming at speeding-up recognition

1 Section 2.7 surveys some of the most prominent tasks.

algorithms with minimal loss of accuracy and making recognition more accurate at tolerable overhead in execution time.

As a single model is often unable to perform recognition within the expected amount of time at the required accuracy, fusion (or hybridization) of various methods is widely used. Fusion, as shown in the chapters of Part II, can be achieved at various levels. In Chapter 3, I introduce a layer of meta models for *quality estimation* of recognition algorithms and use this estimation to make the recognition more accurate. In Chapter 6, I develop the *GRAMOFON ensemble framework* that utilizes error compensation in context of the stacking schema. In Chapter 4 I discuss *instance selection* that can be used as preprocessing technique in order to speed up the recognition and propose the selection of instances based on the hubness property. I propose *fusion of distance measures* in Chapter 5, while in Chapter 7, I extend *frequent pattern mining algorithms* and show how they can be incorporated in well-established recognition algorithms, classifiers.

While fusion is common for all the developed algorithms of the previously mentioned chapters, the contribution of this work is not limited to hybridization. For example, the frameworks that I introduce (individualized quality estimation and GRAMOFON) are interesting on their own, as well as the algorithms that I develop within these frameworks. Also the proof of NP-completeness of the instance selection problem is a result that is expected to have impact beyond the scope of fusion methods. A systematic overview of these contributions is provided below.

1.1 Outline

This book summarizes the results of my recent research. It is organized as follows. Chapter 2 defines the time-series classification problem, gives an overview of the concepts, techniques and algorithms of the literature that are most relevant from our point of view, and describes some of the real-life recognition tasks related to time-series classification. The aforementioned sections of Part II summarize major results of my research. Most of them were published in research papers at international scientific conferences [28], [29], [30], [31], [32], [33], [34], [36]. Some parts of the work were published in *Neurocomputing* [27] and as a book chapter [16]. Part III rounds off this book: before drawing conclusions, I discuss possible extensions of the proposed techniques, as well as some open questions. Next, the chapters of Part II are outlined.

Chapter 3: Individual Quality (IQ) Estimation

In Chapter 3, I develop the framework of *individual quality (IQ) estimation* [32]. After describing this in more detail, I deploy two new recognition algorithms, IQ-MAX and IQ-WV, within this framework. An earlier version of IQ estimation, called individualized error prediction, already achieved attention in the scientific community by winning the **best-paper award** of IEEE's renowned conference on Computational Science and Engineering [29]. As I point out, IQ estimation is not limited to time-series classification,therefore, I adapted and applied it for conventional vector classification tasks as well [33].

Chapter 4: INSIGHT: Instance Selection based on Graph-coverage and Hubness for Time series

In Chapter 4, I propose *Instance Selection based on Graph-coverage and Hubness for Time-series* [31], [34]. Instance selection is a technique that is often used in order to *speed up* time-series classification. The analysis focuses on the relation of instance selection to the recently observed phenomenon of *hubness*, which states that a *few* data instances tend to be frequently nearest neighbors of *many* other instances. In order to model the problem, I introduce the concept of coverage graphs and I prove that the instance selection problem is NP-hard in general. I point out, however, that for a special case which is most relevant for time-series classification, the problem can be solved computationally simply (in polynomial time). Therefore, justified by this discussion, I propose INSIGHT.

Chapter 5: Fusion of Time Series Distance Measures

Recent research [56] suggests that distance measures play crucial role in time-series classification. Different distance measures capture, however, different aspects of similarity: as the relevant aspects of similarity vary from application to application, none of them can be considered as the generally best distance measure. Therefore, I propose a framework for combining various distance measures, in order to capture relevant aspects of similarity. The fusion algorithm I propose in Chapter 5 produces a fused distance measure which adapts to the underlying application [30].

Chapter 6: GRAMOFON: General Model-selection Framework based on Networks

I propose the *General Model-selection Framework based on Networks* (GRAMOFON) in Chapter 6. Whenever many recognition algorithms are available for the same task, in order to achieve accurate recognition, instead of selecting the best one, usually it is worth combining different algorithms according to an ensemble schema. I observed, however, that combining *all* the algorithms (or the individually best ones) may be a sub-optimal choice. This observation motivated the development of the GRAMOFON framework, which supports the deployment of algorithms that select a subset of all the available algorithms in a more sophisticated way, in order to make the recognition more accurate. In Chapter 6, I propose GRAMOFON and four model selection algorithms, namely *Basic, BasicFast, RegOptMST, GraphOptMST*, that I deployed within the GRAMOFON framework [27], [28].

Chapter 7: Generalized Sequential Pattern Mining Algorithms

I generalized frequent pattern mining algorithms, ECLAT and Christian Borgelt's doubly recursive counting schema [19], [20] for the case of taxonomic sequential patterns [163] and used these patterns for time-series classification (Chapter 7). In joint work with Sebastian Blohm we used these algorithms for information extraction from the world wide web [16], [15].

Chapter 8: Further Related Applications

Together with my collaborators, Philipp Cimiano, Sebastian Blohm, Leandro Balby Marinho, Lucas Drumond, Timo Reuter, Lorenza Romano and Claudio Guiliano we used techniques similar to the proposed ones in various applications related to the world wide web. Together with Christine Preisach, Andre Busche, W.H. Leong, and Mark Walters, we aimed at detecting patterns in aeroplane engine vibration data. This is a crucial recognition task related to time series. These applications are outlined in Chapter 8.

In order to assist reproducibility and justify the relevance of the above methodical innovations in real-world applications, I performed experiments on publicly available real-world data from various domains. (I introduce these datasets and the corresponding recognition tasks in Section 2.7.) In most of the cases, I performed 10-fold-cross-validation and statistical significance tests in order to justify that my algorithms outperform their competitors. In total, the experiments presented here correspond to several years of CPU-time.[2]

2 I performed these experiments in parallel on several CPUs.

Chapter 2

Background

2.1 Basic Definitions and Problem Formulation

As mentioned in the Introduction, time-series classification is the common theoretical background of various recognition tasks. Keeping this in mind, as is common in machine learning, I will define the problem in a generic way.

Definition 1. *A* time series x *of length l, is a sequence of real numeric values:*

$$x = (x[0], ..., x[l-1]) \tag{2.1}$$

where x[j] denotes the j-th element of the time series x, $\forall j : x[j] \in \mathbb{R}, 0 \leq j < l$, thus: $x \in \mathbb{R}^n$.

Let $\mathcal{T}$ denote the set of all considered time series, $\mathcal{T} = \cup_{l=0}^{\infty} \mathbb{R}^l$. Some subsets of $\mathcal{T}$ are given, these subsets are called *classes*, and they are denoted as $C_1, ..., C_m$. Each time series $x_i \in \mathcal{T}$ belongs to exactly one of the classes:

$$\forall x_i \in \mathcal{T} : ((x_i \in C_1) \vee ... \vee (x_i \in C_m)) \wedge (x_i \in C_j \Rightarrow x_i \notin C_k, k \neq j) \tag{2.2}$$

For some time series $x_i \in \mathcal{T}$, however, it is unknown to *which* class they belong.

Definition 2. *A* labeled dataset $\mathcal{D} = \{(x_i, c(i))\}_{i=1}^n$, $\mathcal{D} \subset (\mathcal{T} \times \{1, ..., m\})$ *consists of n time series together with their class labels $c(i)$.*

In the above definition, for a time series x_i, the notation $c(i)$ stands for the class to which x_i belongs, e.g. $c(i) = 2 \Leftrightarrow x_i \in C_2$.

Time series in a labeled dataset are called *labeled time series*. Other time series, for which their classes are unknown, are called *unlabeled time series*. The time series in a (labeled) dataset $\mathcal{D}$ are also called *instances* of $\mathcal{D}$.

The *time-series classification problem* is the task of constructing a model that is able to assign new, unlabeled time series to their classes. The formal definition of this problem is developed in the next steps, I begin with the definition of a *classifier*:

Definition 3. *A* classifier *is a function f that maps time series to class labels is:*

$$f : \mathcal{T} \rightarrow \{1, ..., m\} \tag{2.3}$$

As classes are often associated with events in the future, the output of the classifier, $f(x)$, is often called *prediction* of the classifier for time series x. Suppose, for example, that patients are described by their medical time series. Some of theses patients will quickly recover (they belong to class 1), while others will not (they belong to class 2).

We say, a time series x is *correctly classified* by a classifier f, if $f(x)$, the output of the classifier, equals to the true class label of x.

Definition 4. *Given a classifier f and a labeled dataset $\mathcal{D}_{test}$, the* accuracy *of f is the portion of time series in $\mathcal{D}_{test}$ that are correctly classified:*

$$accuracy(f) = \frac{\Big|\{\forall(x_i, c(i)) \in \mathcal{D}_{test} | f(x_i) = c(i)\}\Big|}{|\mathcal{D}_{test}|} \tag{2.4}$$

The dataset $\mathcal{D}_{test}$ is called test data.

Definition 5. *Given a labeled dataset $\mathcal{D}_{train}$, called* training data*, and a set of classifiers, i.e. a set of functions $\mathbb{F}$ that map time series to classes*[1]*:*

$$\forall f \in \mathbb{F} : \quad f : \mathcal{T} \rightarrow \{1, ..., m\}, \tag{2.5}$$

the time-series classification problem *is defined as the following task: using $\mathcal{D}_{train}$, find the function $f \in \mathbb{F}$ that has maximal accuracy on an* unknown *test dataset $\mathcal{D}_{test}$.*

This definition is a conceptual description of the problem: according to this definition the test dataset is unknown when searching for the right classifier, therefore, one cannot directly find the optimal classifier based on the above definition.

In real-world applications, a recognition model is constructed by finding an appropriate classifier function using the given labeled dataset (training data) and then (possibly days, week or years later) the classifier is applied for new, unlabeled time series. While searching for the appropriate classifier function, we aim at finding that f which is *expected* to have the highest accuracy when classifying new time series. Estimation of this expected performance is challenging because the new, unlabeled time series are usually unknown during construction of the classifier.

In experimental settings, usually, the realistic scenario is simulated by using two disjoint labeled datasets $\mathcal{D}_{train}$ and $\mathcal{D}_{test}$: while searching for the classifier-function, only the time series and labels in $\mathcal{D}_{train}$ are accessed (this way $\mathcal{D}_{test}$ is

1 Note that the cardinality of $\mathbb{F}$ is not necessarily finite.

simulated to be unknown) and then, in a proceeding evaluation step, the found classifier is applied for the time series of $\mathcal{D}_{test}$ and the output of the classifier is compared to the true labels of $\mathcal{D}_{test}$. A more detailed description of the experimental protocol used in this book can be found in Section 2.6.

The process of constructing the recognition model by finding an appropriate function f out of the set of potential functions $\mathbb{F}$ is called *training* of the classifier.

The above definition of the time-series classification problem is generic, to allow for various recognition tasks depending on the semantics of the classes. For example, in case of handwriting recognition, the classes may correspond to the letters of the alphabet (one class for each letter), while in case of the analysis of electrocardiograph (ECG) signals, the classes may correspond to different diseases. In Section 2.7, I give an overview of the datasets and corresponding tasks considered in this book.

Instead of a single class label, advanced classifiers output a likelihood (or probability) for each class. Based on this information we can automatically assign an unlabeled time series to the most probable class, or we can use these class probabilities when fusing several classifiers.

2.1.1 Variants of Time-Series Classification

In the literature, there are some slightly different variants of the above-defined time-series classification problem. One of them is the early classification of time series that aims at determining the class as early as possible, i.e., without observing the entire time series. Therefore, in this case, the classifier only uses a prefix of the entire time series [195]. In case of multi-dimensional time series, such as recordings of brain waves (EEG), the elements of the time series are not single numbers, but vectors of numbers [72], [184]. When the time series are sampled unevenly and/or they contain missing values, interpolation techniques can be applied [45]. While the above definition of accuracy only focuses on whether or not the classification is correct, in many applications, different types of errors (Type I and Type II error) should be distinguished: e.g. in a medical recognition system, the case when (i) a healthy patient is recognized as an ill one (and consequently the doctor unnecessarily examines the patient), is substantially different from the case when (ii) an ill patient is recognized as healthy (and consequently the patient does not receive a treatment and her/his health status deteriorates).

Some works focus on domain-specific problems related to time-series classification, e.g. Botsch [22] developed recognition models for detection and categorization of car crashes. In such applications, time-series classification often needs to be combined with other techniques. Botsch, for example, distinguishes between two subproblems which can be interpreted as (i) segmentation of time series and (ii) classification of the segments. In this safety-critical application, besides accuracy, interpretability is crucial and in cases when the classifier is not sure, it is allowed to reject the input data, i.e., instead of an incorrect decision, the classifier does not output any class label in some cases.

2.1.2 Time-Series Classification and Vector Classification

Conventional classification problems, from the formal point of view, are similar to time-series classification: in both cases, each object to be classified is described by a vector of real numbers. In case of conventional classification problems, these vectors have the *same* length, while the length may *vary* in case of time-series classification. Regarding the semantics of the data, however, differences are more crucial and they substantially affect recognition algorithms: the elements of a time series correspond to measured values of the *same feature* in consecutive moments of time, and therefore these values are highly inter-correlated, in contrast, in case of conventional classification, elements of the vectors correspond to measured values of *different features*, and, in general, nothing can be assumed about their correlation. In case of conventional classification, the k-th value of the vector always corresponds to the same feature, while in case of time-series classification, a characteristic pattern may appear at (slightly) different positions of the time series, because it may be shifted and/or elongated.

In this book, in order to distinguish from *time-series classification*, I refer to conventional classification as *vector classification*.

2.2 Time Series Representations

One of the first questions one has to answer before constructing recognition models, or classifiers, is: *what is the best way to represent time series data?* In terms of Definition 1, the most natural representation is simply listing the numeric values of the time series. Whenever necessary for disambiguity, throughout this thesis, this representation is referred to as *raw time series*. Representing the data as raw time series is not necessarily the most suitable choice when we aim at solving the classification problem. In many tasks, specific properties of time series are relevant, such as frequency (e.g. in speech recognition) or presence of a particular pattern (if a disease have to be detected based on the electrocardiograph signal of a patient).

Due to the heterogeneous nature of the applications of time-series classification, various representations have been introduced ranging from Fourier and Wavelet Transforms [39], [65], [121], [193] over piecewise approximations [74], [94], [97], [111], [158], [125], [199] to Singular Value Decomposition [107]. I refer to [111] for an overview and categorization of time series representation. The following sections focus only on the ones that are used in this book later on.

2.2.1 Discrete Fourier Transform

Intuitively speaking, the Fourier Transformation decomposes a signal, i.e. time series in our case, as a sum of sinusoidal signals. While doing so, it captures which frequencies are present in the time series. Time series, defined as a sequence of numeric values (see Definition 1), are *time-discrete* signals, therefore, for this book, *Discrete Fourier Transformation* (DFT) is relevant.

DFT maps the time series $x = (x[0], ..., x[l-1])$ to l complex coefficients: $c_0^F, \ldots, c_{l-1}^F$. These coefficients are defined as

$$c_j^F(x) = \frac{1}{\sqrt{l}} \sum_{i=0}^{l-1} x[i] e^{-\frac{2\pi \mathcal{I} j i}{l}} \quad , \quad 0 \leq j \leq l-1 \tag{2.6}$$

where $\mathcal{I} = \sqrt{-1}$ (imaginary unit), $e \approx 2.718$, $\pi \approx 3.142$ and $x[i]$ denotes the i-th numeric value of the time series x. The *Discrete Fourier Transform* of x is defined as the *sequence* of all its Fourier-coefficients c_j^F:

$$DFT(x) = (c_0^F(x), ..., c_{l-1}^F(x)) \tag{2.7}$$

Although, in principle, one can calculate the Discrete Fourier Transformation of a time series according to Equation 2.6, there is a computationally more efficient method for this, called *Fast Fourier Transformation* (FFT). The FFT-algorithm is based on the property that the DFT of a discrete signal $x = (x[0], x[1], ..., x[l-2], x[l-1])$ of even length (i.e. $l = 2k, k \in \mathbb{Z}^+$) can be decomposed[2] as the function of the DFTs of the even-positioned and odd-positioned values of x:

$$DFT\ (x[0], x[1], ..., x[l-2], x[l-1])\ ) = \tag{2.8}$$
$$h\Big(DFT(\ (x[0], x[2], ..., x[l-2])\ ), DFT(\ (x[1], x[3], ..., x[l-1])\ )\Big)$$

The above decomposition only works for signals of even length. FFT applies this decomposition recursively, as long as it is possible (as long as the signal's length is even). Therefore, FFT is most efficient for signals which have a length of $2^k, k \in \mathbb{Z}^+$. For such signals FFT works in $\mathcal{O}(l \log l)$ time.

Algorithm 1 shows the pseudocode of FFT, while Figure 2.1 shows two examples of time series and their DFTs. In Figure 2.1 continuous lines denote the real parts of the complex numbers, while dashed lines denote the imaginary parts. The signal in the upper left is a simple sinusoidal signal, therefore its Fourier Transform (upper right) is zero everywhere, except the positions corresponding to the frequency of this sinusoidal signal. The signal in the bottom left consists of a dominant low-frequency component, a high-frequency component and some random noise. The low- and high-frequency components can be seen in its Fourier-Transform (bottom right). The peaks at the beginning and end correspond to low-frequency components, while the peaks at 16 and 48 correspond to the high-frequency components.

2 For more details see also http://www.ismll.uni-hildesheim.de/lehre/ip-08w/script/imageanalysis-2up-04-fourier-transform.pdf

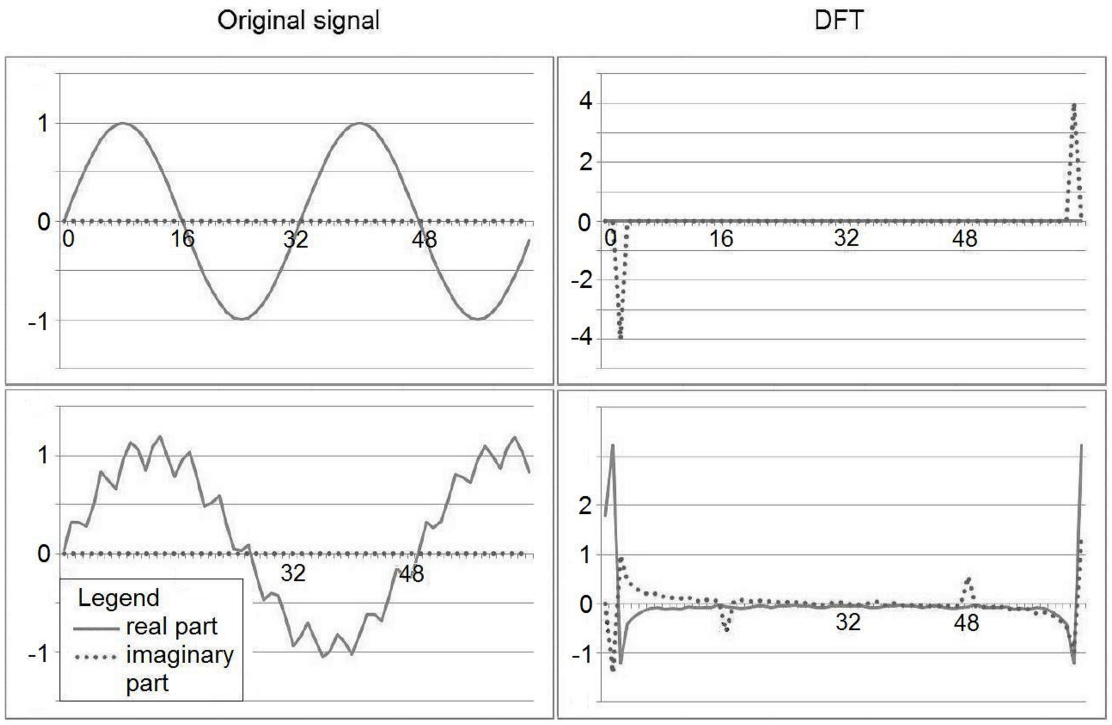


Figure 2.1: Real-valued time series (time-discrete signals) of length 64 (in the left) and their Discrete Fourier Transforms (in the right)

2.2.2 Discrete Wavelet Transform

By decomposing the time series as "sum" of sinusoidal signals, Fourier Transformation of a time series captures *global periodic* behavior. *Wavelets*, in contrast, aim at reflecting both *local* and *global* character of a time series [81].

Out of many variants of Discrete Wavelet Transform, in this book, I use only one, the so called Haar Wavelet Transform[3]. While performing this transformation, one produces a lower-resolution representation of a time series x by averaging its consecutive values. In order to keep all the information, detail-coefficients are also stored (that allow to restore the original time series). The procedure is repeated recursively as long as the length of the time series is longer than two. This is shown in Algorithm 2, where array a denotes the lower-resolution representation of the time series x, while c denotes the array containing the detail-coefficients.[4] For a detailed description, discussion and other variants of wavelets, the interested Reader is referred to [51], [115].

3 See http://www.ismll.uni-hildesheim.de/lehre/ip-08w/script/imageanalysis-2up-05-wavelets.pdf

4 Algorithm 2 works for time series that have a length of a power of two. If the length of time series is different from a power of two, a simple and commonly-used technique is to add some dummy values (e.g. zeros) at the end of the time series in order to ensure that its length is a power of two.

Algorithm 1 Fast Fourier Transformation (FFT)
(Based on the source in Footnote 2.)

```
Require: Time Series $x = (x[0], x[1], ..., x[l-2], x[l-1])$
Ensure: Discrete Fourier Transform of $x$
1: if $l$ is even then
2:     $a = FFT(\ (x[0], x[2], ..., x[l-2])\ )$     (recursive function call)
3:     $b = FFT(\ (x[1], x[3], ..., x[l-1])\ )$     (recursive function call)
4:     for $j = 0; j < l; j++$ do
5:         $a_0 = a[j \bmod (l/2)]$.realPart
6:         $a_1 = a[j \bmod (l/2)]$.imaginaryPart
7:         $b_0 = b[j \bmod (l/2)]$.realPart
8:         $b_1 = b[j \bmod (l/2)]$.imaginaryPart
9:         $c_j^F$.realPart= $\Big(a_0 + b_0 \cos(2\pi j/l) + b_1 \sin(2\pi j/l)\Big)/\sqrt{2}$
10:        $c_j^F$.imaginaryPart= $\Big(a_1 + b_1 \cos(2\pi j/l) - b_0 \sin(2\pi j/l)\Big)/\sqrt{2}$
11:    end for
12:    return $(c_1^F, ..., c_{l-1}^F)$
13: else
14:    return Discrete Fourier Transformation according to Equation 2.6
15: end if
```

2.2.3 Symbolic Aggregate Approximation

Symbolic Aggregate Approximation (SAX) aggregates and discretizes consecutive values of time series [111], [112]. Denoting the number of discrete values and length of the aggregated representation as v_{SAX} and l_{SAX} respectively, the procedure generating the Symbolic Aggregate Representation of a time series x can be summarized in three steps (see also Figure 2.2 for an example).

1. Normalization of x in order to ensure that x has a mean of zero and standard deviation of one. (Steps 1 ... 5 in Algorithm 3.)
2. Piecewise Aggregate Approximation of x — The entire time series x is divided into l_{SAX} equal sized, non-overlapping frames that cover the entire time series. For each such frame, the average of the respective values of x is calculated. The sequence of these averages is called Piecewise Aggregate Approximation. (Steps 6 ... 13 in Algorithm 3.)
3. Discretization — The above averages are mapped to discrete symbols 'A', 'B', 'C', etc. One aims at determining the breakpoints $\beta_0 \dots \beta_m$ used for this mapping in a way that the discretization technique produces each symbols with equal probability: "breakpoints are a sorted list of numbers (...) such that the area under a (...) Gaussian curve from β_i to $\beta_{i+1} = 1/m$ (β_0 and β_m are defined as $-\infty$ and ∞ respectively)" [111]. Assignment of discrete symbols is shown from Step 14 to Step 20 in Algorithm 3.

Algorithm 2 Haar Wavelet Transformation (HWT)
(Based on the source in Footnote 3.)

```
Require: Time Series $x = (x[0], x[1], ..., x[l-2], x[l-1])$    ( $l$ is even )
Ensure: Haar Wavelet Transform of $x$
1: for $j = 0...l/2 - 1$ do
2:     $a[j] = \frac{1}{\sqrt{2}}(x[2j] + x[2j+1])$
3:     $c[j] = \frac{1}{\sqrt{2}}(x[2j] - x[2j+1])$
4: end for
5: if $l > 2$ then
6:     return concat(HWT($a$),$c$)    (Concatenation of the sequences HWT($a$) and $c$)
7: else
8:     return ( $a[0]$, $c[0]$ )
9: end if
```

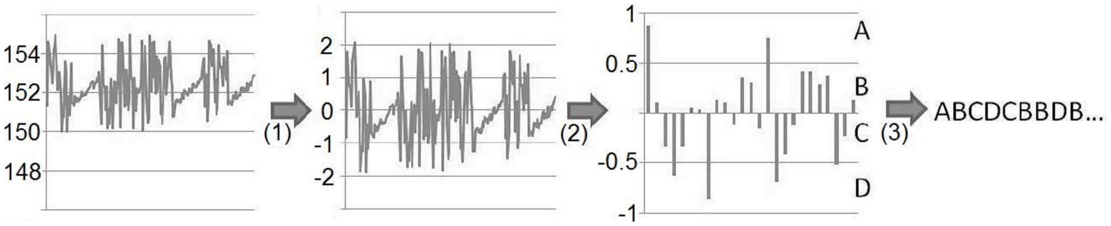


Figure 2.2: Symbolic Aggregate Approximation (SAX) of a time series: (1) normalization, (2) aggregation and (3) mapping to discrete symbols with $m_{SAX} = 4$.

The number of discrete symbols m, and therefore the number of breakpoints, is a parameter of SAX. In order to avoid ambiguity, in the remainder of the book I denote the number of breakpoints as m_{SAX}.

2.3 Distance Measures

A *time series distance measure* quantifies the difference between two time series. The *distance measure* (or analogously *similarity measure* or *kernel*) is a central component for many prominent time series classifiers, like shaplet-based decision trees for time series [198], nearest neighbor models or SVMs. This section reviews *distance measures*, while Section 2.4 is devoted to time series classifiers. Formally, a distance measure is a function $d(x_1, x_2)$:

Definition 6. *A* distance measure d *is a function that takes two time series* x_1 *and* x_2 *as input, and returns a numeric value:*

$$d(x_1, x_2) : \mathcal{T} \times \mathcal{T} \to \mathbb{R} \tag{2.9}$$

Algorithm 3 Symbolic Aggregate Approximation (SAX)

```
Require: Time Series $x = (x[0], x[1], \ldots, x[l-2], x[l-1])$,
    Number of discrete values $v_{SAX}$,   Length of aggregated time series $l_{SAX}$
Ensure: Symbolic Aggregate Approximation of $x$
 1: $a = (1/n) \sum_{i=0}^{i=l-1} x[i])$
 2: $s = \sqrt{(1/(n-1)) \sum_{i=0}^{i=l-1} (x[i]-a)^2}$
 3: for $j = 0 \ldots l-1$ do
 4:     $x[j] = (x[j]-a)/s$
 5: end for
 6: $n = l/l_{SAX}$   (Number of consecutive values that will be aggregated)
 7: for $j = 0 \ldots l_{SAX}-1$ do
 8:     $b = 0$
 9:     for $i = jn \ldots jn+n-1$ do
10:         $b = b + x[i]$
11:     end for
12:     $c[j] = b/n$
13: end for
14: for $j = 0 \ldots l_{SAX}-1$ do
15:     ( $\beta_0 ... \beta_m$ are breakpoints calculated as described in [111] )
16:     if $\beta_0 < c[j]$ and $c[j] \leq \beta_1$ then $d[j]$ = 'A'
17:     else if $\beta_1 < c[j]$ and $c[j] < \beta_2$ then $d[j]$ = 'B'
18:     else if $\beta_2 < c[j]$ and $c[j] < \beta_3$ then $d[j]$ = 'C'
19:     ...
20: end for
21: return $d$
```

The numerical value returned by $d(x_1, x_2)$ is the *distance* of x_1 and x_2. Intuitively speaking, the more dissimilar x_1 and x_2 are, the higher is the value of $d(x_1, x_2)$.

Most prominent distance measures for time series include *Euclidean Distance*, *Dynamic Time Warping* [14], [98], [101], [143], [153], DISSIM [70], *Edit Distance on Real Sequences* [41], *Edit Distance with Real Penalty* [40], *Sequence Weighted Alignment Model* (Swale) [122], *Spatial Assembling Distance* (SpADe) [42] and distance based on *Longest Common Subsequences* [181]. Vlachos et al. [182] introduced a rotation invariant distance measure, while Agrawal took noise, scaling and translation into account [2]. Caiado [37] proposed time series distance measures "based on the autocorrelations, partial and inverse autocorrelations, and periodogram ordinates" as well as "time and frequency domain based metrics for classification of time series with unequal lengths"[5]. Often, distance measures can

5 http://www.amazon.de/

be used with various time series representations resulting in a large number of variants: “there are over a dozen distance measures for similarity of time series data in the literature” [56]. The remainder of this section focuses on the ones that are used in this book later on.

2.3.1 Euclidean Distance

Definition 7. *The Euclidean Distance of two time series x_1 and x_2 of the same length l is defined as follows:*

$$d_{EU}(x_1, x_2) = \sqrt{\sum_{j=0}^{l-1} \Big(x_1[j] - x_2[j]\Big)^2} \tag{2.10}$$

The Euclidean Distance can not only be calculated with the raw representation of time series. The Euclidean Distance over the *Discrete Fourier Transform* of two time series, d^F_{EU}, can be calculated as (see also [1]):

$$d^F_{EU}(x_1, x_2) = \sqrt{\sum_{j=0}^{l-1} \Big(DFT(x_1)[j] - DFT(x_2)[j]\Big)^2} \tag{2.11}$$

Analogously, the Euclidean Distance over the Discrete *Haar Wavelet Transform* is:

$$d^W_{EU}(x_1, x_2) = \sqrt{\sum_{j=0}^{l-1} \Big(HWT(x_1)[j] - HWT(x_2)[j]\Big)^2} \tag{2.12}$$

2.3.2 Dynamic Time Warping

While calculating the Euclidean distance of two time series x_1 and x_2, the k-th element of x_1 is matched to the k-th element of x_2. In reality, however, when observing the same phenomenon several times, we cannot expect an event to happen (or a characteristic pattern to appear respectively) *always* at *exactly* the same time position, and the event’s duration can also vary slightly. Therefore, Dynamic Time Warping (DTW) captures the similarity of two time series’ shapes in a way that it allows for elongations: the k-th position of time series x_1 is not necessarily matched to the k-th position of time series x_2, but it can be matched to the k'-th position ($k' \neq k$) of x_2. This is illustrated in Figure 2.3. In order to allow for elongations, the same position of the first time series is allowed to be matched to several consecutive positions of the second time series and vice versa, e.g. in Figure 2.3, position 16 in the top right time series is matched to positions 23, 24, . . . , 33 in the bottom right time series.

DTW is an edit distance [109]. This means that we can conceptually consider the calculation of the DTW distance of two time series x_1 and x_2 of length l_1

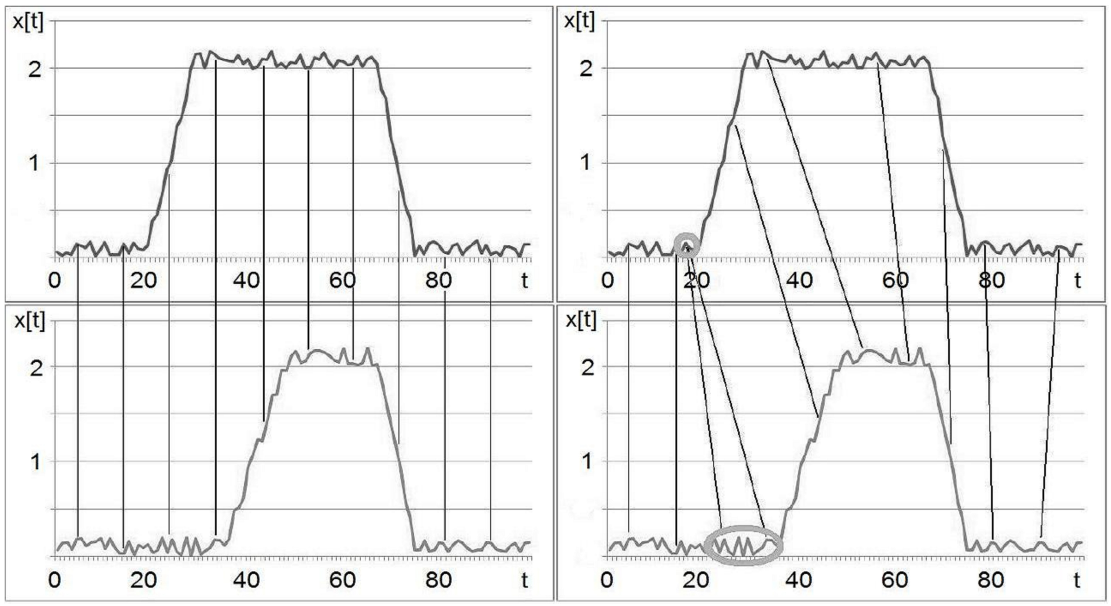


Figure 2.3: Euclidean Distance vs. Dynamic Time Warping: Euclidean Distance compares always the k-th positions of the both time series with each other (left), while DTW allows for elongation, and therefore when calculating the distance of two time series with DTW, the k-th position of the first time series is not necessarily matched to the k-th position of the second time series (right). This matching is shown by the roughly-vertical lines in both cases.

and l_2 respectively as the *process of transforming* x_1 into x_2. Suppose we have already transformed a prefix (possibly having length zero or l_1 in the extreme cases) of x_1 into a prefix (possibly having length zero or l_2 in the extreme cases) of x_2. Consider the *next* elements (the elements that directly follow the already-transformed prefixes) of x_1 and x_2. The following *editing steps* are possible, both of which being associated with a cost:

1. *replacement* of the next element of x_1 for the next element of x_2, in this case, the next element of x_1 is matched to the next element of x_2, *and*
2. *elongation* of an element: the next element of x_1 is matched to the last element of the already-matched prefix of x_2 or vice versa.

As result of the replacement step, both prefixes of the already-matched elements grow by one element (by the next elements of x_1 and x_2 respectively). In contrast, in an elongation step, one of these prefixes grows by one element, while the other prefix remains the same as before the elongation step.

The cost of transforming the entire time series x_1 into x_2 is the cost of *sum* of the costs of all the necessary editing steps. In general, there are many possibilities to transform x_1 into x_2, DTW calculates the one with minimal costs. This minimal cost serves as the distance between both time series. The details of the calculation of DTW are described next.

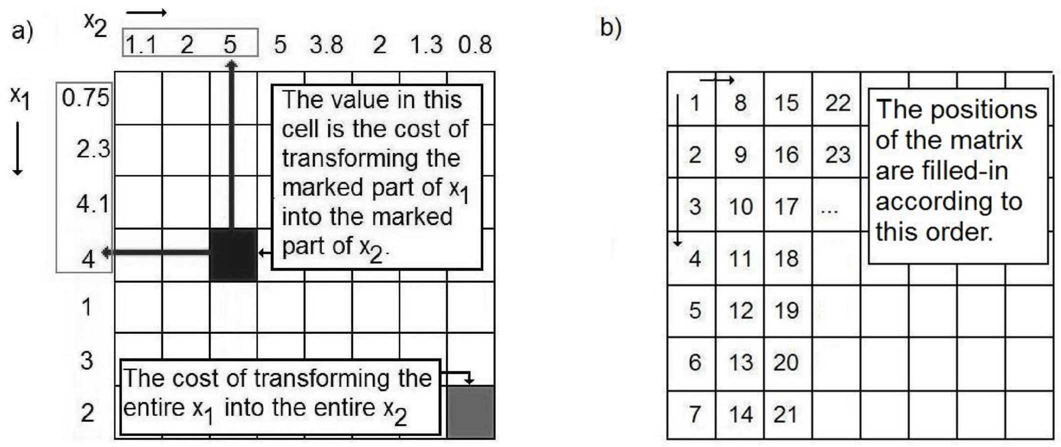

a)
x2
1.1 2 5 5 3.8 2 1.3 0.8
x1
0.75
2.3
4.1
4
1
3
2
The value in this cell is the cost of transforming the marked part of x1 into the marked part of x2.
The cost of transforming the entire x1 into the entire x2
b)
1 8 15 22
2 9 16 23
3 10 17 ...
4 11 18
5 12 19
6 13 20
7 14 21
The positions of the matrix are filled-in according to this order.

Figure 2.4: The DTW-matrix. While calculating the distance (transformation cost) between two time series x_1 and x_2, DTW fills-in the cells of a matrix. a) The values of time series $x_1 = (0.75, 2.3, 4.1, 4, 1, 3, 2)$ are enumerated on the left of the matrix from top to bottom. Time series x_2 is shown on the top of the matrix. A number in a cell corresponds to the distance (transformation cost) between two prefixes of x_1 and x_2. b) The order of filling the positions of the matrix.

DTW utilizes the dynamic programming approach [143], [144], [153]. Denoting the length of x_1 by l_1, and the length of x_2 by l_2, the calculation of the minimal transformation cost is done by filling the entries of an $l_1 \times l_2$ matrix. Each number in the matrix corresponds to the distance between a subsequence of x_1 and a subsequence of x_2. In particular, the number in the i-th row and j-th column[6], $d_0^{DTW}(i,j)$ corresponds to the distance between the subsequences $x_1' = (x_1[0], \ldots, x_1[i])$ and $x_2' = (x_2[0], \ldots, x_2[j])$. This is shown in Figure 2.4.

When we try to match the i-th position of x_1 and the j-th position of x_2, there are three possible cases: (i) elongation in x_1, (ii) elongation in x_2, and (iii) no elongation.

If there is no elongation, the prefix of x_1 up to the $(i-1)$-th position is matched (transformed) to the prefix of x_2 up to the $(j-1)$-th position, and the i-th position of x_1 is matched (transformed) to the j-th position of x_2.

Elongation in x_1 means that the i-th position of x_1 has already been matched to some positions of x_2, i.e., the prefix of x_1 up to the i-th position is matched (transformed) to the prefix of x_2 up to the $(j-1)$-th position, and the i-th position of x_1 is matched again, this time to the j-th position of x_2. This way the i-th position of x_1 is *elongated*, in the sense that it is allowed to match several positions of x_2. The elongation in x_2 can be described in an analogous way.

Out of these three possible cases, DTW selects the one that transforms the prefix $x_1' = (x_1[0], \ldots, x_1[i])$ into the prefix $x_2' = (x_2[0], \ldots, x_2[j])$ with minimal overall costs. Denoting the distance between the subsequences x_1' and x_2', i.e. the

6 Please note that the numbering of the columns and rows begin with zero, i.e., the very-first column/row of the matrix is called in this sense as the 0-th column/row.

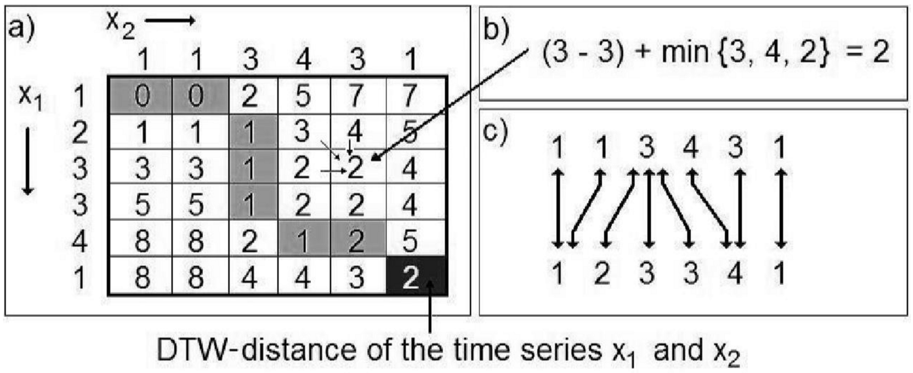


Figure 2.5: Example for the calculation of the DTW-matrix. a) The DTW-matrix calculated with $c_{tr}^{DTW}(v_A, v_B) = |v_A - v_B|$, $c_{el}^{DTW} = 0$. The time series x_1 and x_2 are shown on the left and top of the matrix respectively. b) The calculation of the value of a cell. c) The (implicitly) constructed mapping between the values of the both time series. The cells are leading to the minimum in Formula (2.13), i.e., the ones that allow for this mapping, are marked in the DTW-matrix.

value of the cell in the i-th row and j-th column, as $d_0^{DTW}(i,j)$, based on the above discussion, we can write:

$$d_0^{DTW}(i,j) = c_{tr}^{DTW}(x_1[i], x_2[j]) + \min \left\{ \begin{array}{l} d_0^{DTW}(i, j-1) + c_{el}^{DTW} \\ d_0^{DTW}(i-1, j) + c_{el}^{DTW} \\ d_0^{DTW}(i-1, j-1) \end{array} \right\} \tag{2.13}$$

In this formula, the first, second, and third terms of the minimum correspond to the above cases of elongation in x_1, elongation in x_2 and no elongation, respectively. The cost of matching (transforming) the i-th position of x_1 to the j-th position of x_2 is $c_{tr}^{DTW}(x_1[i], x_2[j])$. (If $x_1[i]$ and $x_2[j]$ are identical, the cost of this replacement is zero.) This cost is present in all the three above cases. In the cases, when elongation happens, there is an additional elongation cost denoted as c_{el}^{DTW}.

According to the principles of dynamic programming, Formula (2.13) can be calculated for all i, j in a column-wise fashion. First, we set $d_0^{DTW}(0,0) = c_{tr}^{DTW}(x_1[0], x_2[0])$. Then we begin calculating the very first column of the matrix ($j = 0$), followed by the next column corresponding to $j = 1$,etc. The cells of each column are calculated in order of their row-indexes: within one column, the cell in the row corresponding $i = 0$ is calculated first, followed by the cells corresponding to $i = 1$, $i = 2$, etc. (See Figure 2.4.) In some cases (in the very-first column and in the very-first cell of each row), in the min function of Formula (2.13), some of the terms are undefined (when $i - 1$ or $j - 1$ equals -1). In these cases, the minimum of the other (defined) terms are taken.

The DTW distance of x_1 and x_2, i.e. the cost of transforming the entire time series $x_1 = (x_1[0], x_1[1], \dots, x_1[l_1 - 1])$ into $x_2 = (x_2[0], x_2[1], \dots, x_2[l_2 - 1])$ is

$$d_{DTW}(x_1, x_2) = d_0^{DTW}(l_1 - 1, l_2 - 1) \tag{2.14}$$

An example for the calculation of DTW is shown in Figure 2.5.

Note that the described method implicitly constructs a mapping between the positions of the time series x_1 and x_2: by back-tracking which of the possible cases lead to the minimum in the Formula (2.13) in each step, i.e. which of the above discussed three possible cases lead to the minimal transformation costs in each step, we can reconstruct the mapping of positions between x_1 and x_2.

Dynamic Time Warping for Multivariate Time Series

DTW can be easily extended for the case of multivariate time series [183]. In order to do so, we only need to specify $c_{tr}^{DTW}(x_1[i], x_2[j])$ for the multi-dimensional case. As an example, we consider the case of *two-dimensional time series*, where the time series consits of two-dimensional vectors, the components of which are denoted with the postfixes a and b:

$$x = \Big((x[0].a, x[0].b), \quad (x[1].a, x[1].b), \quad \dots \quad , \quad (x[l-1].a, x[l-1].b) \Big) \tag{2.15}$$

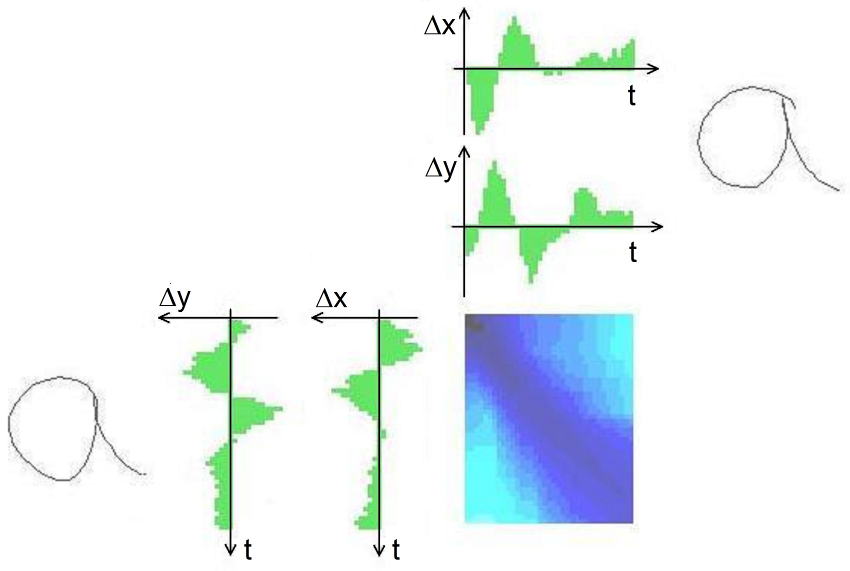
In this case $c_{tr}^{DTW}(x_1[i], x_2[j])$, the cost of transforming the two-dimensional vector $(x_1[i].a, x_1[i].b)$ into another two-dimensional vector $(x_2[j].a, x_2[j].b)$, can be specified as their Euclidean distance:

$$c_{tr}^{DTW}(x_1[i], x_2[j]) = \sqrt{(x_1[i].a - x_2[j].a)^2 + (x_1[i].b - x_2[j].b)^2} \tag{2.16}$$

Figure 2.6 shows an example for the calculation of two-dimensional DTW from the handwriting recognition domain. Handwriting is considered here as two-dimensional time series, i.e, time series of horizontal and vertical movements within short time frames. In order to keep the presentation compact, the numerical values of calculated numbers $d_0^{DTW}(i, j)$ are not shown, instead, the matrix is depicted as an image where light tones denote high values, and dark tones denote low ones. One can clearly see that two a's distance is much smaller than the distance between a and b.

Avoiding the computation of the entire DTW-matrix

For the final result of the distance calculation, the values close to the diagonal of the matrix are usually the most important ones (see Figure 2.5 for an illustration). Therefore, a simple, but effective way of speeding-up dynamic time warping is to restrict the calculations to the cells around the diagonal of the matrix [98], [153]. This means that one limits the elongations allowed when matching the both time series (see Figure 2.7).



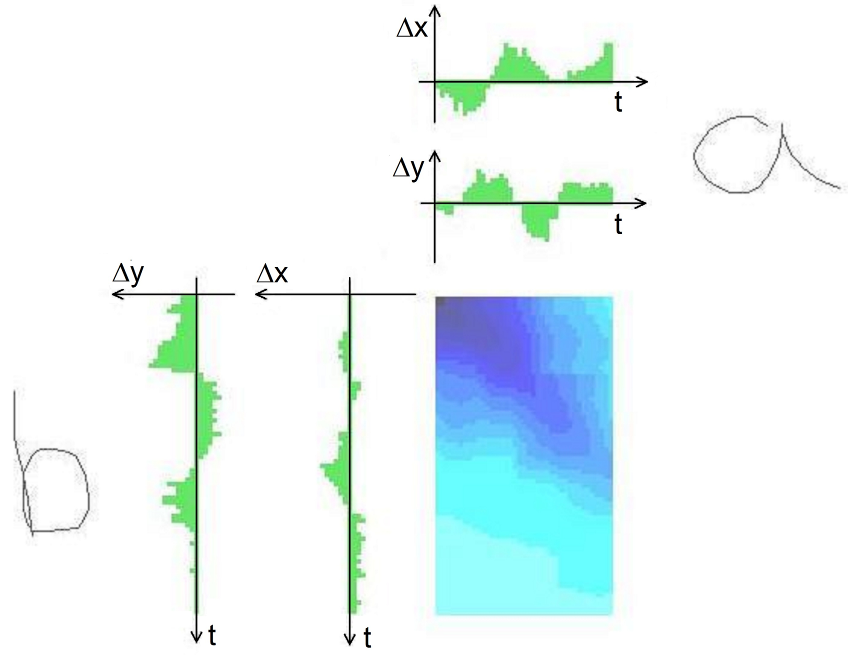


Figure 2.6: DTW for handwriting recognition. Handwriting is considered here as two-dimensional time series (time series of horizontal and vertical movements). The numerical values of calculated cells of the matrix are not shown, instead, the matrix is depicted as an image where light tones denote high values, and dark tones denote low ones. One can clearly see that the distance between two *a*-s is much smaller than the distance between *a* and *b*. (This illustration is adapted from [26].)

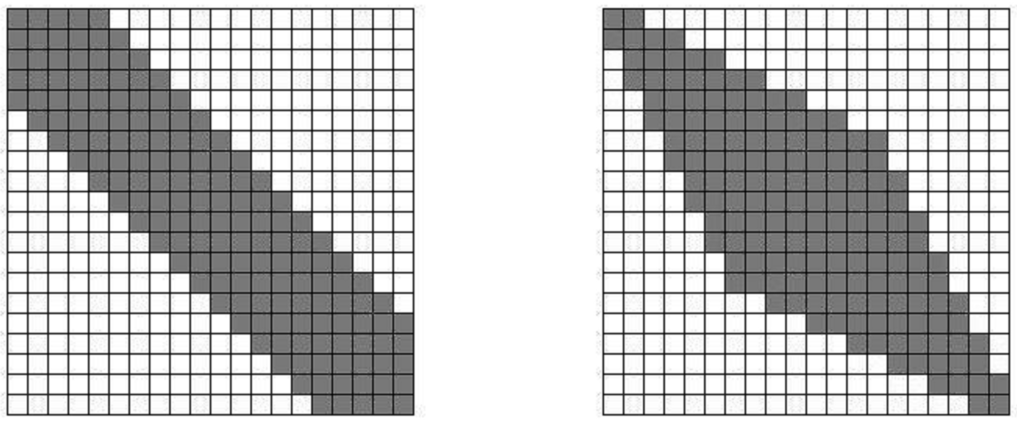


Figure 2.7: Limiting the size of the warping window: only the cells around the main diagonal of the matrix (marked cells) are calculated. In the left, the size of warping window is a constant of 20 %, while in the right the cells corresponding the Itakura Parallelogram are shown.

Restricting the warping window size to a pre-defined constant w^{DTW} (see in the left of Figure 2.7) means that only the cells of the matrix are calculated that at most w^{DTW} positions far from the main diagonal along the vertical direction:

$$d_0^{DTW}(i,j) \text{ is calculated} \Leftrightarrow |i-j| \leq w^{DTW} \tag{2.17}$$

The warping window size w^{DTW} is often expressed in percentage relative to the length of the time series. In this case, $w^{DTW} = 100\%$ means calculating the entire matrix, while $w^{DTW} = 0\%$ refers to the extreme case of not calculating any entries at all.

Recent research shows that setting w^{DTW} to a relatively small value such as 5%, does *not* negatively affect the accuracy of the classification. In contrast, the experiments in [143], [145] imply that such relatively small warping windows lead to more accurate classification than calculation of the entire matrix. Interpreting this results in terms of elongations, one can conclude that allowing for moderate elongation is beneficial, however, allowing for too much elongation (and therefore matching distant parts of time series) might harm the final quality of recognition.

More advanced techniques to avoid the computation of the entries of the entire matrix, include the Itakura Parallelogram [86], [98], and learned constraints [144]. According to the Itakura Parallelogram, close to the corners of the matrix, only a few cells are calculated, while many cells are calculated in the middle of the matrix (see Figure 2.7). Ratanamahatana and Keogh applied learned constraints [144] in order to adapt the warping window size *locally*, i.e. it may be different e.g. at the beginning and end of time series. Besides avoiding the calculation of many cells, these adaptations aim to optimize the classification accuracy.

Indexing

In many problems, such as similarity search or nearest neighbor classification, for a given time series x we aim at finding the *most similar* time series out of a set of time series $\mathcal{D}$. Indexing [38], [78] aims at *efficiently* solving this task. In order to do so, several lower-bounding techniques were proposed [98], [100], [104], [200]. A lower-bound of DTW, $d^{lb}_{DTW}(x_1, x_2)$, is an approximation of the true DTW-distance, $d_{DTW}(x_1, x_2)$, of two time series x_1 and x_2, so that $d^{lb}_{DTW}(x_1, x_2)$ is guaranteed to be less than or equal to the true DTW-distance:

$$d^{lb}_{DTW}(x_1, x_2) \leq d_{DTW}(x_1, x_2) \tag{2.18}$$

Computationally cheap lower-bounds were proposed in [98], [100], [104], [200] which allowed to substantially speed up the search process for the most similar time series. As an example, suppose, we are searching for the time series that is most similar to x, and the current best candidate, i.e., the time series found to be the closest one up to the current step of the search process, is x_0. In the current step, we examine whether another time series x_1 is closer to x than x_0. If the computationally cheap lower-bound approximates that the distance of x_1 to x is larger than the distance of x_0 to x, we do not need to calculate the computationally expensive true distance of x_1 and x because:

$$d_{DTW}(x_1, x) \geq d^{lb}_{DTW}(x_1, x) > d_{DTW}(x_0, x) \tag{2.19}$$

If, however, $d^{lb}_{DTW}(x_1, x) \leq d_{DTW}(x_0, x)$, we need to calculate the computationally expensive true distance $d_{DTW}(x_1, x)$.

Due to the above described filtering, the execution time for finding the time series being most similar to a given time series x, can be considerable for large time series datasets, since it can be affected by the significant computational requirements posed by the need to calculate the true DTW distance between x and *several* time series in the dataset $\mathcal{D}$ ($\mathcal{O}(|\mathcal{D}|)$ in the worst case, where $|\mathcal{D}|$ is the number of time series contained in $\mathcal{D}$). Furthermore, in case of large time series datasets, the lower-bound – which is computationally inexpensive for a single pair of time series – needs to be calculated for *large number of* time series pairs causing considerable computational expenses.

Therefore, indexing can be considered complementary to other speed-up techniques such as instance selection (see Section 2.4.2) or the limiting the warping window size, because these techniques can be applied together with indexing in order to achieve the required execution time.

Further Recent Results on DTW

Recent research on dynamic time warping, besides the ones already mentioned, aimed at making DTW more accurate and robust against noise: e.g. the derivation of time series was proposed in [102] as a preprocessing step before calculating the

DTW-distances. Ratanamahatana and Keogh [143], [145] gave a thorough analysis that altered our understanding of the properties of DTW. In their comprehensive experiments on 38 datasets from various domains, Ding et al. [56] compared DTW against other distance measures and showed that DTW is exceptionally hard to beat. This is also justified by recent applications built on DTW, such as [146]. In the context of the car crash detection and categorization problem, Botsch modified DTW in order to capture, in addition to the similarity in time series shapes, the duration of time series too [22].

Standard Parameter Settings for Dynamic Time Warping

In the standard settings used in this book, the cost of elongation, c_{el}^{DTW}, is set to zero:

$$c_{el}^{DTW} = 0 \tag{2.20}$$

The cost of transformation (matching), denoted as c_{tr}^{DTW}, depends on what value is replaced by what: if the numerical value v_A is replaced by v_B, the cost of this step is:

$$c_{tr}^{DTW}(v_A, v_B) = |v_A - v_B| \tag{2.21}$$

I set the warping window size to $w^{DTW} = 5\%$, which is justified by the analysis in [143], [145]. Whenever the opposite is not explicitly stated, I used the above standard settings for DTW.

2.3.3 Edit Distance on Real Sequences

As another distance measure for time series, Chen et al. [41] introduced the *Edit Distance on Real Sequences* (EDR). EDR between two time series $x_1 = (x_1[0], \dots, x_1[l_1 - 1])$ and $x_2 = (x_2[0], \dots, x_2[l_2 - 1])$ was defined as "the *number* of insert, delete, or replace operations that are needed to change" [41] x_1 into x_2. While calculating EDR, two elements, $x_1[i]$ and $x_2[j]$, are considered to *match*, i.e., none of the above operations is necessary to transform $x_1[i]$ into $x_2[j]$, if $|x_1[i] - x_2[j]| \le \epsilon^{EDR}$, where ϵ^{EDR} is a parameter, called *similarity threshold.*

The generic description of DTW in Section 2.3.2 allows to interpret Edit Distance on Real Sequences as a variant[7] of DTW with warping window size $w^{DTW} = 100\%$, elongation cost $c_{el}^{DTW} = 1$, and transformation cost

$$c_{tr}^{DTW}(v_A, v_B) = \begin{cases} 0 & \text{if } |v_A - v_B| \le \epsilon^{EDR} \\ 1 & \text{otherwise} \end{cases} \tag{2.22}$$

7 Chen and Ng [40] already pointed out the analogy between DTW, EDR and ERP. They described all these distance measures with recursive formulae of similar type, while my descriptions here and in the next section are more procedural.

A further difference is that instead of Equation (2.13), EDR uses (2.23), according to which c_{tr}^{DTW} is only added if there is no elongation:

$$d_0^{DTW}(i,j) = \min \left\{ \begin{array}{c} d_0^{DTW}(i,j-1) + c_{el}^{DTW} \\ d_0^{DTW}(i-1,j) + c_{el}^{DTW} \\ d_0^{DTW}(i-1,j-1) + c_{tr}^{DTW}(x_1[i], x_2[j]) \end{array} \right\} \tag{2.23}$$

The initialization of the recursive EDR-formula in [41] can be interpreted as follows. EDR initializes

$$d_0^{DTW}(-1,-1) = 0 \tag{2.24}$$

$$d_0^{DTW}(i,-1) = i+1 \ \text{ for } \ 0 \le i \le l_1 - 1 \tag{2.25}$$

and

$$d_0^{DTW}(-1,j) = j+1 \ \text{ for } \ 0 \le j \le l_2 - 1 \tag{2.26}$$

This way, all the terms of the minimum in Formula (2.23) are defined for all the positions $0 \le i \le l_1 - 1$, $0 \le j \le l_2 - 1$, therefore the first column and first row of the DTW-matrix (corresponding to $i = 0$ and $j = 0$ respectively) can be calculated in the same way as the other cells of the matrix (instead of taking the minimum only of the defined elements when calculating the first row or the first column of the matrix).

2.3.4 Edit Distance with Real Penalty

Using their respective standard settings, neither DTW nor EDR fulfills triangle inequality in general. Following the example in [40], this is illustrated in Figure 2.8. As described in [40], for EDR, this problem originates from Equation (2.22), where the similarity threshold ϵ^{EDR} is used to decide whether two values match. Although DTW has been shown to "loosely" satisfy triangle inequality for speech applications [180], after verifying this observation on 24 benchmark datasets from various domains, Chen and Ng report: "It appears that this observation is not true in general, as on average nearly 30% of all the triplets do not satisfy the triangle inequality. (...) The key reason why DTW [with the standard settings] does not satisfy the triangle inequality is that, when a gap needs to be added, it replicates the previous element" [40], i.e., its allows to match the same element several times in case of an elongation, see Figure 2.5 c) for an example. When the same element $x_1[i]$ is matched several times, the costs of transforming $x_1[i]$ into several elements $x_2[j], x_2[j+1], \ldots$ are accumulated.

Edit Distance with Real Penalty (ERP) was designed to satisfy triangle inequality. Therefore, in contrast to EDR it avoids the usage of the similarity threshold ϵ^{EDR}, instead, in line with DTW, as the cost of matching two elements, ERP applies the difference of their values. In contrast to DTW, however, ERP avoids the accumulation of costs in case of elongations.

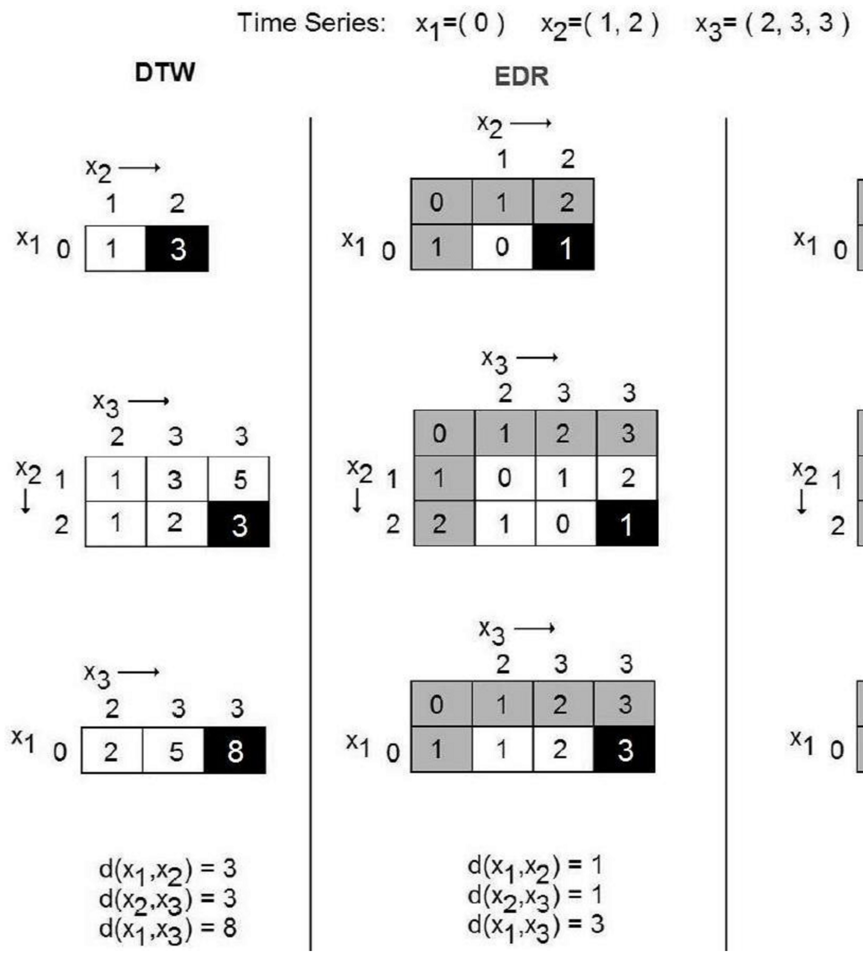


Figure 2.8: Distance of time series x_1, x_2 and x_3 with DTW (standard settings), EDR and ERP. Gray marking denotes the additional cells (the "minus first" rows and columns). As shown, DTW and EDR do not fulfill the triangle inequality. (The example in [40] is adapted in this illustration.)

One can formally describe ERP as an instance of DTW as follows. The warping window size w^{DTW} is set to 100%. In line with the standard settings of DTW, $c_{tr}^{DTW}(v_A, v_B) = |v_A - v_B|$. Instead of Equation (2.13), ERP uses (2.27), according to which, in line with EDR, c_{tr}^{DTW} is only added if there is no elongation, therefore the above mentioned undesired accumulation of transformation costs is avoided:

$$d_0^{DTW}(i,j) = \min \left\{ \begin{array}{c} d_0^{DTW}(i, j-1) + c_{el,1}^{DTW} \\ d_0^{DTW}(i-1, j) + c_{el,2}^{DTW} \\ d_0^{DTW}(i-1, j-1) + c_{tr}^{DTW}(x_1[i], x_2[j]) \end{array} \right\} \tag{2.27}$$

In contrast to EDR, instead of a uniform elongation cost, ERP applies the value of the "inserted" element as an elongation cost: $c_{el,1}^{DTW} = x_2[j]$, $c_{el,2}^{DTW} = x_1[i]$. In terms of transformation costs, this can be interpreted as follows: whenever there is an elongation, it is considered as the insertion of an element, the cost of this step is considered to be the value of the inserted element.

Similarly to EDR, the initialization of the Chen's recursive ERP-formula [40] can be interpreted as an additional column and an additional row of the DTW-matrix (corresponding $i = -1$ and $j = -1$ respectively) :

$$d_0^{DTW}(-1,-1) = 0 \tag{2.28}$$

$$d_0^{DTW}(i,-1) = \sum_{i'=0}^{i'=i} x_1[i] \quad \text{for} \quad 0 \leq i \leq l_1 - 1 \tag{2.29}$$

$$d_0^{DTW}(-1,j) = \sum_{j'=0}^{j'=j} x_2[j] \quad \text{for} \quad 0 \leq j \leq l_2 - 1 \tag{2.30}$$

This way, all the terms of the minimum in Formula (2.27) are defined for all the positions $0 \leq i \leq l_1 - 1$, $0 \leq j \leq l_2 - 1$, therefore the first column and first row of the DTW-matrix (corresponding to $i = 0$ and $j = 0$ respectively) can be calculated in the same way as the other cells of the matrix (instead of taking the minimum only of the defined elements when calculating the first row or the first column of the matrix).

2.3.5 Distance based on Longest Common Subsequences

Originally, Vlachos et al. [181] defined a distance measure based on the length of the Longest Common Subsequences (LCSS) for trajectories (two-dimensional time series). In this section, their definition is adapted for the case of one-dimensional time series.

For a time series $x = (x[0], \ldots, x[l-1])$, denote HEAD$(x)$ its first $l-1$ elements: HEAD$(x) = (x[0], \ldots, x[l-2])$. With this notation, the length of the Longest Common Subsequences (LCSS) of two time series x_1 and x_2 is defined as follows:

$$\text{LCSS}(x_1,x_2) = \begin{cases} 0 \quad \text{if } l_1 = 0 \text{ or } l_2 = 0 \ (x_1 \text{ or } x_2 \text{ is empty}) \\[1ex] 1 + \text{LCSS}(\text{HEAD}(x_1), \text{HEAD}(x_2)) \\ \quad \text{if } |x_1[l_1-1] - x_2[l_2-1]| < \epsilon^{LCSS} \text{ and } |l_1 - l_2| < w^{LCSS} \\[1ex] \max\Big(\text{LCSS}(\text{HEAD}(x_1), x_2), \text{LCSS}(x_1, \text{HEAD}(x_2))\Big) \\ \quad \text{otherwise} \end{cases} \tag{2.31}$$

where the parameters ϵ^{LCSS} and w^{LCSS} stand for the similarity threshold and the warping window size, respectively. Similarly to DTW, based on the above formula, LCSS of two time series $x_1 = (x_1[0], \ldots, x_1[l_1-1])$ and $x_2 = (x_2[0], \ldots, x_2[l_2-1])$

can be calculated by filling the entries of an $l_1 \times l_2$ matrix according to the dynamic programming schema.

Based on LCSS, the distance of two time series $x_1 = (x_1[0], \dots, x_1[l_1 - 1])$ and $x_2 = (x_2[0], \dots, x_2[l_2 - 1])$ can be defined as follows:

$$d_{LCSS}(x_1, x_2) = \frac{\text{LCSS}(x_1, x_2)}{\min(l_1, l_2)} \tag{2.32}$$

2.3.6 DISSIM

Originally, the DISSIM distance measure was defined for two-dimensional time series (also called as trajectories) as follows: "The Dissimilarity [...] between trajectories Q and T being valid during [a particular] period [...] is defined as the definite integral of the function of time of the Euclidean distance between the two trajectories during the same period." [70]

Up to now, we considered time series as a discrete sequence of real numbers. However, in order to be able to specify DISSIM for one-dimensional time series, we need to define values of a time series x for *continuous* positions, i.e. for positions *between* two discrete positions. As so far, we denote values at discrete (integer) positions with $x[t], t \in \mathbb{Z}$. For the positions between two such position, we can use the linear approximation

$$x[t] = x[\lfloor t \rfloor] + (t - \lfloor t \rfloor)(x[\lceil t \rceil] - x[\lfloor t \rfloor]). \tag{2.33}$$

Then, for the case of two one-dimensional time series $x_1 = (x_1[0], \dots, x_1[l-1])$ and $x_2 = (x_2[0], \dots, x_2[l-1])$, the above definition of DISSIM simplifies to:

$$d_{DISSIM}(x_1, x_2) = \int_{t=0}^{t=l-1} \mid x_1[t] - x_2[t] \mid dt \tag{2.34}$$

The lower the value of this integral is, the more similar are the two time series x and y. DISSIM is illustrated in Figure 2.9.

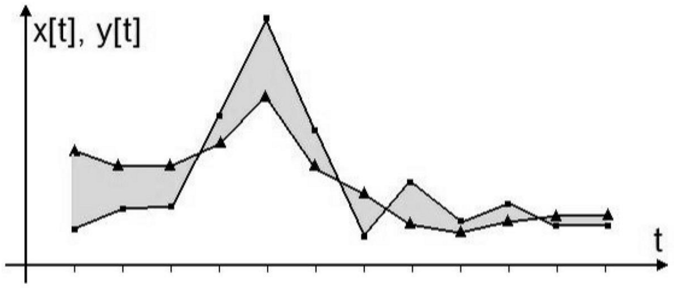


Figure 2.9: DISSIM calculates the area between two time series.

2.4 Time-Series Classification Algorithms

By reason of the increasing interest in time-series classification, various approaches have been introduced including neural networks [93], [130], Bayesian networks [168] to hidden markov models [103], [114], [139], genetic algorithms, support vector machines [62], methods based on random forests and generalized radial basis functions [22] as well as frequent pattern mining [74]. Modular neural networks and their applications to time-series classification, for example, are detailed in [130]. However, the k-nearest neighbor (k-NN) classifier has been shown to be competitive (if not superior) to many other state-of-the-art models [56], [99], [146]. A description of this classifier follows.

Suppose, we are given an unlabeled (or test) time series x. The k-NN classifier searches in the labeled training dataset for those k time series that are most similar to x. These k most similar time series are called the k nearest neighbors of x. The k-NN classifier considers the k nearest neighbors, and takes the majority vote of their labels and assigns this label to x: e.g. if $k = 3$ and two of the nearest neighbors of x belong to class C_1, while one of the nearest neighbors of x belongs to class C_2, then this 3-NN classifier recognizes x as an instance belonging to the class C_1. Besides experimental evidence [56], [99], [146], there are theoretical results about the properties and optimality of nearest neighbor classifiers which are reviewed e.g. in [52].

The applied distance measure plays a crucial role in the nearest neighbor classification of time series. Usually, nearest-neighbor classification of time series uses Dynamic Time Warping (DTW), because it is an elastic distance measure, i.e., it is robust w.r.t. elongation in the time series. As described in Section 2.3.2 in detail, recent works aimed at making DTW more accurate and scalable [101], [144], DTW has been examined in depth (a thorough summary of results can be found at [143], [145]), whereas Ding et al. found no other distance measure that significantly outperforms DTW [56].

2.4.1 Local adaptations of the k-NN classifier

Local adaptations of the k-NN classifier were explored by several authors: Prekopcsak et al. [49], for example, applied class-based attribute weighting. From the point of view of my research, most relevant works deal with the choice of parameter k for the k-NN classifier that is known to affect the bias-variance trade-off [81]: smaller values of k may lead to overfitting, whereas larger values of k increase the bias and in this case the model may capture only global tendencies. Recent studies [142] indicate that significant improvement in the accuracy of the k-NN time-series classification can be attained with k being larger than 1. This is due to intrinsic characteristics in time series data sets, such as the mixture between the different classes, the dimensionality, and the skewness in the distribution of error (i.e., the existence of so called “bad hubs”, described in Section 2.4.3 that account for a surprisingly large fraction of the total error). Whenever the data has homogeneous characteristics, parameter k can be chosen based on the classifier's performance on a hold-out subset of the training data.

In complex time series data sets, however, the intrinsic characteristics, such as the ones mentioned above, may vary from region to region. For this reason, approaches performing local adaptation of the k-NN classifier should be developed. A locally adaptive distance measure was proposed by Hastie and Tibshirani [80], while Domeniconi and Gunopulos [57] used SVMs to define a local measure of feature relevance, i.e., feature weights depending on the location of a data point to be classified. In [58] adaptive nearest neighbor classification in high-dimensional spaces was studied. In contrast to these works, my approaches based on individual quality (IQ) estimation (called IQ-MAX and IQ-WV in Chapter 3) adapt by selecting the proper value of k (IQ-MAX) and by combining several classifiers in a localized way (IQ-WV).

Ougiaroglou et al. [126] presented three early-break heuristics for k-NN which can be interpreted as adapting the number of nearest neighbors. These heuristics aimed at speeding-up k-NN. In contrast, in Chapter 3, I focus on making nearest neighbor classification more accurate using the principled framework of IQ estimation. My instance selection method described in Chapter 4 aims at speeding-up k-NN classification. This is orthogonal to the work of Ougiaroglou et al. [126], as both techniques can be applied together in order to achieve higher speed-up.

Methods for quality estimation (also known as error estimation) are usually applied globally in order to estimate the overall performance of a classification model [88], [120]. In my proposed approach, however, as described in Chapter 3, the focus is on individualized quality estimation. This is similar to learning the residuals, i.e., the difference between predicted and actual labels. Duffy and Helmbold followed this direction and incorporated residuals into boosting of regression models [60]. In contrast to this work, I do not focus on boosting. Similar to my work, Tuda et al. [178] proposed an individualized approach for estimating the leave-one-out error of vector classification with support vector machines (SVM) and linear programming machines (LPM). Unlike this work, my proposed approach performs general IQ estimation (not just for leave one out). More importantly, my approach *exploits* IQ estimation in order to improve accuracy of classification and I do not treat it as a *per se* task, as done in [178].

A set of earlier approaches to localized quality estimation for the k-NN classifier was proposed by Wettschereck and Dietterich [187]. However, these approaches were based solely on heuristics such as using different k values per class or per cluster (after clustering the training set). My proposed framework of IQ estimation is more principled and more generic than these simple approaches: I distinguish between the quality estimation step and classification step, my framework supports systematic usage of the estimated quality, and my framework allows various classification and regression models. Furthermore, while predicting a class, my framework can involve an arbitrary number of models and an arbitrary number of meta models, whereas Wettschereck and Dietterich [187] used a fixed number of models (mostly just one selected model) at the elementary level, and they used heuristics instead of meta models.

Furthermore, the aforementioned works concerned with classification of *vectors*, while I focus on *time-series classification*.

Kernel methods can be interpreted as local adaptations of k-NN classifiers where the influence of a training instance x_0 to the classification of an unlabeled (or test) instance x is determined by the distance between x and x_0 and its ratio to the distances between x and the other training instances. In contrast, in my methods IQ-MAX and IQ-WV (Chapter 3), the distances are only used to determine the nearest neighbors. More importantly, the meta-layer models estimate the classification error for each test instance *individually*: in case of IQ-MAX, a single k is selected in an extremely localized fashion, i.e., *individually* for each test instance to be classified, while in case of IQ-WV, k-NN models are combined in an individualized way: the weights of the elementary k-NN models are determined for each test instance separately.

Therefore, IQ-MAX and IQ-WV perform extremely local adaptations of k-NN classifiers by adapting the model for *each* test instance. This is different from hyperparameter search, where e.g. grid-search or local search is applied in order to find weights or proper values of k and other parameters, and then the found weights and parameter-values are used *universally*, i.e., for the classification of *all* the test instances. Furthermore, note that the proposed framework of IQ estimation is neither limited to finding proper values of parameters for each instance individually, nor to the context of k-NN classification: even though in context of time-series classification I used k-NNs, in general, arbitrary classifiers of various types (such as SVMs, decision trees, HMMs, etc.) can be combined in various ways by selecting different meta-level decision methods.

2.4.2 Speeding-up time-series classification

Attempts to speed up DTW-based nearest neighbor classification fall into 4 major categories: i) speeding-up the calculation of the distance of two time series (by e.g. limiting the warping window size), ii) indexing, iii) reducing the length of the time series used, and iv) instance selection. As the former two were already described in Section 2.3.2, this section focuses on the latter two.

Reduction of the length of time series

A simple way to speed up time-series classification is to reduce the length of time series by aggregating consecutive values into a single number [111], [112], [158], which reduces the overall length of time series and thus makes their processing faster. A more advanced method proposed by Salvador [155] uses multiple resolutions in order to speed up the computation, while Sakurai et al. [154] combined several ideas such as lower bounding, early stopping and aggregation of consecutive values (with different resolutions).

Instance Selection

Instance selection (also known as *numerosity reduction* or *prototype selection*) aims at discarding most of the training time series while keeping only the most informative ones, which are then used to classify unlabeled instances. While instance selection is well explored for general nearest-neighbor classification, see e.g. [4], [23], [76], [89], [113], there are just a few works for the case of time series. Xi et al. [194] present the FastAWARD approach and show that it outperforms state-of-the-art, general-purpose instance selection techniques applied for time series.

FastAWARD follows an iterative procedure for discarding time series: in each iteration, the rank of all the time series is calculated and the one with lowest rank is discarded (see the Naive_Rank_Reduction function in Algorithm 4). Thus, each iteration corresponds to a particular number of kept time series. Furthermore, Xi et al. argue that the optimal warping window size depends on the number of kept time series. Therefore, FastAWARD calculates the optimal warping window size dependent on the number of kept time series (see the main function, called FastAWARD in Algorithm 4).

2.4.3 Hubs in Time Series Data

The presence of hubs, i.e., that some few objects tend to be much more frequently neighbors than the remaining ones, has been observed for many natural and artificial networks, such as protein-interaction networks or the internet [9]. As this book focuses on time-series classification, in this section, hubness is described from this point of view.

For classification, the property of hubness has recently been explored [140], [141], [142]. This property states that for data with high (intrinsic) dimensionality, like most of the time series data[8], some objects tend to become nearest neighbors much more frequently than others. In order to express hubness in a more precise way, for a time series dataset $\mathcal{D}$ one can define the *k-occurrence* of a time series $x \in \mathcal{D}$, denoted by $g_N^k(x)$, as the number of time series in $\mathcal{D}$ having x among their k nearest neighbors. With the term *hubness* we refer to the phenomenon that the distribution of $g_N^k(x)$ becomes significantly skewed to the right. We can measure this skewness, denoted by $\mathcal{S}_{g_N^k(x)}$, with the standardized third moment of $g_N^k(x)$:

$$\mathcal{S}_{g_N^k(x)} = \frac{E[(g_N^k(x) - \mu_{g_N^k(x)})^3]}{\sigma^3_{g_N^k(x)}} \tag{2.35}$$

8 In case of time series, consecutive values are strongly interdependent, thus instead of the length of time series, we have to consider the *intrinsic* dimensionality [142].

Algorithm 4 Simplified pseudocode of FastAWARD
(This pseudocode is based on [194], see that paper for more details.)

```
Require: Time Series Dataset $\mathcal{D}$, Number of selected instances $n$
Ensure: Dataset containing the selected time series $\mathcal{D}^*$,
    Optimal warping window size $w_{opt}^{DTW}$

1: function FastAWARD(Time Series Dataset $\mathcal{D}$,
   Number of selected instances $n$)
2:     Determine the best warping window size on $\mathcal{D}$ and initialize $w$ to this value
3:     (This algorithm works with the relative warping window size, thus $0 \leq w \leq 100\%$ here.)
4:     $\mathcal{S}$ = Naive_Rank_Reduction($\mathcal{D}$)
5:     while $|\mathcal{S}| > n$ do
6:         number_of_selected_instances = $|\mathcal{S}|$
7:         Remove worst (lowest ranked) instance from $\mathcal{S}$
8:         if warping window size of $w + 1\%$ leads to better accuracy (on training
9:         data) then $w \leftarrow w + 1\%$
10:    end for
11:    return $\mathcal{S}$ together with $w$
12:        (The remaining instances are the selected ones, $w$ is the optimal
13:        warping window size for this amount of selected instances.)
14: end

15: function Naive_Rank_Reduction(Time Series Dataset $\mathcal{D}$)
16:    Remove any duplicate instances from $\mathcal{D}$
17:    Leave-one-out 1-NN classification on $\mathcal{D}$
18:    OrderedSet $\mathcal{S} = \{\}$
19:    for (loop_num = 0,  loop_num < $|\mathcal{D}|$, loop_num++)
20:        Calculate $g_{Xi}(x)$ (see Equation 4.3) for each time series $x \in \mathcal{D} \setminus \mathcal{S}$
21:        Rank all the time series $x \in \mathcal{D} \setminus \mathcal{S}$ based on $g_{Xi}(x)$,
22:                resolve ties based on distances
23:        Select the worst (lowest ranked) instance,
24:                discard it from $\mathcal{D}$ and push it (at the end of) $\mathcal{S}$
25:    end for
26: return $\mathcal{S}$
27: end
```

where $\mu_{g_N^k(x)}$ and $\sigma_{g_N^k(x)}$ are the mean and standard deviation of $g_N^k(x)$. When $\mathcal{S}_{g_N^k(x)}$ is higher than zero, the corresponding distribution is skewed to the right and starts presenting a long tail.

In the presence of labeled data, we distinguish between *good hubness* and *bad hubness*: we say that the time series x' is a *good k-nearest neighbor* of the time series x, if (i) x' is one of the k-nearest neighbors of x, and (ii) both have the *same*

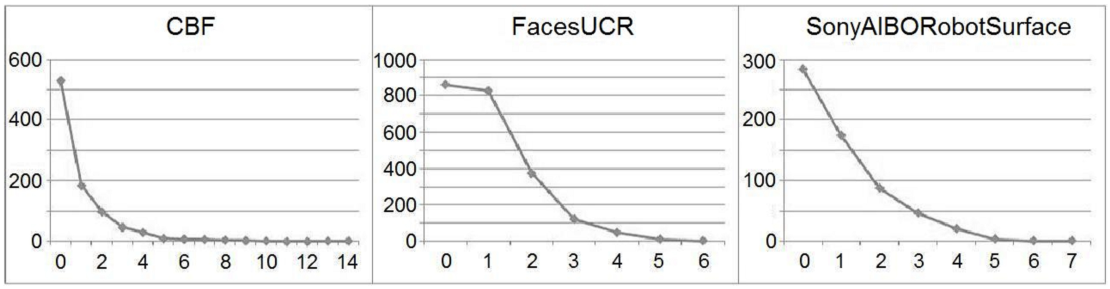


Figure 2.10: Distribution of good hubs for some time series datasets. The horizontal axis correspond to the values of $g^1_G(x)$, while on the vertical axis one can see how many instances have that value.

class labels. Similarly: we say that the time series x' is a *bad k-nearest neighbor* of the time series x, if (i) x' is one of the k-nearest neighbors of x, and (ii) they have *different* class labels. This allows us to define *good (bad) k-occurrence* of a time series x, $g^k_G(x)$ (and $g^k_B(x)$ respectively), which is the number of other time series that have x as one of their good (bad respectively) k-nearest neighbors. For time series, both distributions $g^k_G(x)$ and $g^k_B(x)$ are usually skewed, as it is exemplified in Figure 2.10, which depicts the distribution of $g^1_G(x)$ for some time series data sets (from the collection described 2.7). As shown, the distributions have long tails in which the good hubs occur.

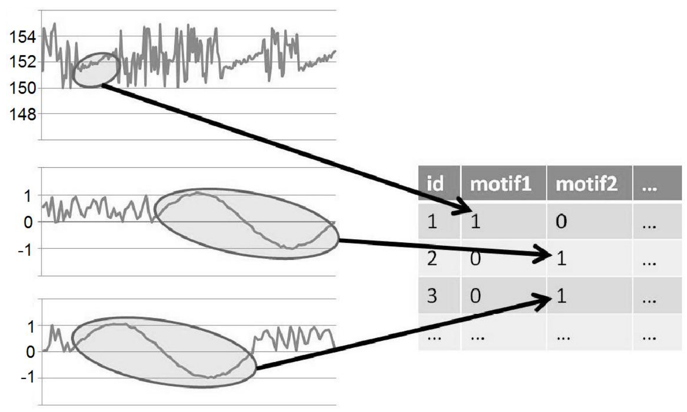
We say that a time series x is a good (bad) hub, if $g^k_G(x)$ (and $g^k_B(x)$ respectively) is exceptionally large for x. For the nearest neighbor classification of time series, the skewness of good occurrence is of major importance, because of two reasons:

1. On one hand, some few time series are responsible for large portion of the overall error: *bad hubs* tend to *misclassify* a surprisingly large number of other time series, this problem is discussed and alleviated by the application of an appropriate weighting schema in [142].

2. On the other hand, a few time series, the *good hubs*, are able to *correctly* classify most of the other time series.

Therefore, one has to take into account the presence of good and bad hubs in time series datasets. In Chapter 4, I discuss hub-based score functions for the problem of instance selection that is described in Section 2.4.2. In the Chapter 4, I also prove a theorem regarding the optimality of hub-based instance selection.

Note that in other domains, distributions have been found that are very similar to the ones shown in Figure 2.10 [9], [10], [53], [179]. Therefore, this property can be considered as a generic one, which further justifies its exploitation in the domain of time-series classification.



id	motif1	motif2	...
1	1	0	...
2	0	1	...
3	0	1	...
...	...	...	...

Figure 2.11: Representation of time series as attribute vectors using motif features

2.4.4 Classification based on Feature Extraction

Besides nearest neighbor classification, an alternative approach for classifying time series is based on feature extraction [123]. Simple features are for example the *average*, *minimum* or *maximum* of a time series x. In general, a feature is a value that aims at capturing a characteristic property of a time series x. Usually, one selects some features (such as the ones mentioned above), and then calculates all of the selected features for all the time series of the dataset $\mathcal{D}$. Therefore, each time series $x \in \mathcal{D}$ can be represented as a vector of features: suppose, for example, that one selected average as the first feature, then for each time series $x \in \mathcal{D}$ the first component of the corresponding vector is the average of x.

On one hand, converting time series into vectors has the advantage that conventional, well-studied vector classifiers, such as SVMs [108], decision trees [61] or Naive Bayes [67] can be used for the time-series classification problem. On the other hand, however, finding good features that are descriptive and therefore allow for distinguishing between classes, is a difficult task in general: whether a feature is a good one or not, depends highly on the underlying application. Features for a given application can be constructed based on domain-specific background knowledge [22].

In contrast, a widely studied set of generic features are based on *motifs* [61], [67], [108]. A motif is a characteristic pattern. In the most simple case, for each motif there is a binary feature, which indicates whether the corresponding motif appeared in the time series or not (see Figure 2.11). Alternatively, a feature can indicate the total number and/or average length of motifs occurring in a time series [67]. In order to construct motif-based features, motifs need to be found first. This is detailed in the following section.

Motif Discovery

The *task of motif discovery* in time series has many variants in the literature. Yankov et al. [197] and Patel et al. [127] define the task of motif discovery regarding one long time series: the target of their work is to identify recurrent patterns, i.e.,

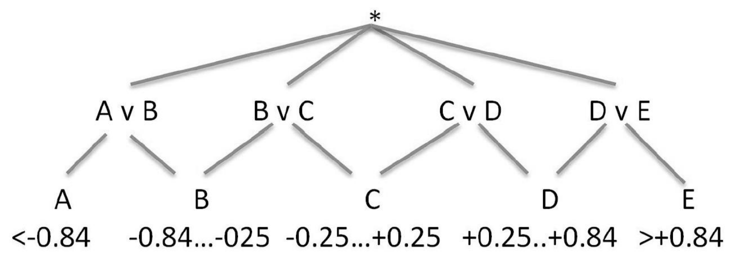


Figure 2.12: Specific and generic wild-cards build a taxonomy of symbols: suppose that the time series are represented as a sequence of discrete symbols A,B,C,D,E (see Section 2.2.3). One can define additional symbols, so called *wild-cards* that can match several symbols: e.g. $*$ matches any symbol, while the symbol "$A \vee B$" matches both symbols A and B.

approximately repeated parts of the given time series. In contrast, Futschik and Carlisle [71] are concerned with a set of time series. They aim at finding global patterns: they cluster times series, and calculate the "compromise" time series for each cluster. Such a "compromise" time series is regarded as a representative pattern of the time series in the cluster. Jensen at al. [90] and Ferreira at al. [68] also use clustering, however, in a more local fashion: they do not cluster the whole sequences, but subsequences of them.

Predefining a (minimal) length l_m for motifs, scanning the database, and enumerating (almost) all the subsequences of the given length l_m is common in the biological domain [68], [90], [127], [197]. However, this may not be efficient enough, especially if complex motifs with gaps and/or taxonomical wild-cards are to be discovered (see Figure 2.12). For noise-robust motif detection (without wild-cards) Buhler and Tompa used random projections [25]. A more sophisticated solution is based on the property called *anti-monotonicity*, which was originally observed in the frequent itemset mining community [3]. Anti-monotonicity states that subpatterns of a frequent pattern are also frequent. State-of-the-art frequent sequence mining algorithms, see e.g. [17], [73], are based on anti-monotonicity which suggests, roughly speaking, that one should discover short motifs first and grow them step by step together to longer ones. Approaches based on anti-monotonicity avoid processing of many redundant (i.e. non-motif) subsequences. They have been researched intensively, resulting in highly efficient implementations [19], [20]. There are different algorithms for databases of different character: for example, in case if many short motifs are expected, they can efficiently be discovered ("grown together") in a breadth-first search manner as in [18], whereas if long motifs are expected, approaches utilizing a depth-first search schema could be preferable, such as PrefixSpan [128].

In this book, by *motifs* I mean approximately repeated *local patterns*. In Chapter 7, I propose two algorithms based on *frequent pattern mining* techniques in order to discover motifs in a *set of time series*. Both of my approaches use the symbolic aggregate approximation of time series (see Section 2.2.3).

Table 2.1: A systematic overview of selected related works on pattern mining

	Flat Patterns	Generalized Patterns
Set Motifs (item sets)	[3], [19], [20]	[84], [133], [165], [166]
Sequential Motifs (Contiguous and non-contiguous)	[18], [43]	[73], [163]
Semi-contiguous Sequential Motifs	[67]	**Chapter 7**

Given the symbolic aggregate approximation of the time series of a dataset $\mathcal{D}$, different classes of motifs can be defined regarding generality and character:

- Regarding **generality**, I distinguish (i) *flat patterns* (without wild-cards), and (ii) patterns with wild-cards, called *generalized* patterns.

- Regarding **character**, I distinguish between (i) *set motifs* (the order of symbols is omitted), (ii) *sequential motifs* (contiguous and non-contiguous sequences), and (iii) *semi-contiguous sequential motifs*, which is a common generalization of contiguous and non-contiguous motifs: in semi-contiguous motifs maximal n_m *gaps* are allowed, each of these gaps are allowed to have *maximal length* of d_m. (For $d_m = n_m = 0$ semi-contiguous motifs are identical with contiguous motifs; for $d_m = n_m = \infty$ they are the same as non-contiguous motifs.)

Table 2.1 provides a systematic overview of related works on pattern mining.

For the task of discovering flat patterns, optimization techniques (recursive counting and recursion pruning) were introduced in [19] and [20]. In Chapter 7, I generalize these techniques for the case of semi-continuous sequential patterns. Similarly to my work, Ferreira and Azevedo [67] allowed gaps in motifs, however, they did not use taxonomic wild-cards. My work also differs from theirs in the algorithm used to discover motifs: they enumerated all subsequences (of a given length), whereas I build my algorithms on frequent pattern mining techniques.

Schmidt-Thieme [156] presents a generic framework that allows to deploy frequent pattern mining algorithms for various pattern types. In terms of [156], the patterns used in my work can be interpreted as patterns of higher order. Out of the pattern types considered in [156], most closely related to my work is the type *sequences of sequences* because a sequential pattern containing taxonomic wild-cards can be considered as a sequence of sequences. This is illustrated with the following example. Suppose, we are given the sequential pattern $(A, D, A \vee B, *, E)$ containing symbols of the taxonomy shown in Figure 2.12. Instead of each symbol we can write the path that leads to that symbol in the taxonomy: ($(*, A \vee B, A)$, $(*, D \vee E, D)$, $(*, A \vee B)$, $(*)$, $(*, D \vee E, E)$). Provided that the semi-continuous semantic of the patterns is taken into account while counting the support, this interpretation allows to apply algorithms for mining frequent *sequences of sequences* as presented in the 5th Chapter of [156]. However, in the case considered in my work the "internal" sequences (i.e. the sequences that correspond to symbols of the taxonomy) have the special property that they are paths of the taxonomy, and therefore a symbol can only be followed by an other symbol that is the specification of the previous symbol according to the taxonomy. In contrast to [156], in Chapter 7 this property is taken into account and exploited for efficient pattern mining algorithms (see *taxonomic antimonotone* property in Chapter 7).

2.5 Fusion Methods

As a single model is usually unable to solve complex recognition tasks at the required accuracy, fusion techniques are popularly used, see e.g. [87], [129], [135], [160], [171], [191], [192], [202]. Fusion can be achieved various ways: in [35], for example, together with my collaborators, we solved different subtasks of the problem with separate models: we applied image processing methods in order to recognize lines in aeroplane vibration diagrams, and in a subsequent task we used a matching technique in order to label these lines. Whether such "separation by task" is possible (and effective) highly depends on the concrete application.

In contrast, one can provide more generic schemes, according to which a (large) number of models solve the same classification task and their solutions are integrated either using some heuristics or with a meta-model. This resulting structure is often called an *ensemble of classifiers*. Fundamental statistical, computational and representational reasons, why ensembles work better than single models, were discussed in [55].

In practice, besides the simple voting schemes, most prominent ensemble methods include boosting, bagging [11], [54] and stacking [173] (also called *stacked generalization* [189]). In case of boosting, a series of classifiers are trained: one classifier in each iteration. While doing so, in each iteration, priority is given to the correct classification of those instances that were misclassified by the models constructed in the previous iterations. This is achieved by assigning higher weights to previously misclassified instances. In contrast, in case of bagging, different samples of the entire training data are used to train the models. Stacking is similar to bagging, however, the variety of classifiers is usually achieved by using different models, such as nearest neighbor approaches, decision trees, Bayesian approaches, support vector machines, neural networks. Then, based on the outputs of these models, a meta-model decides the class label. Variety of models of the ensemble can be achieved not only by using different model types, but also by choosing different hyperparameters or (similarly to bagging) different subsets of the entire training data.

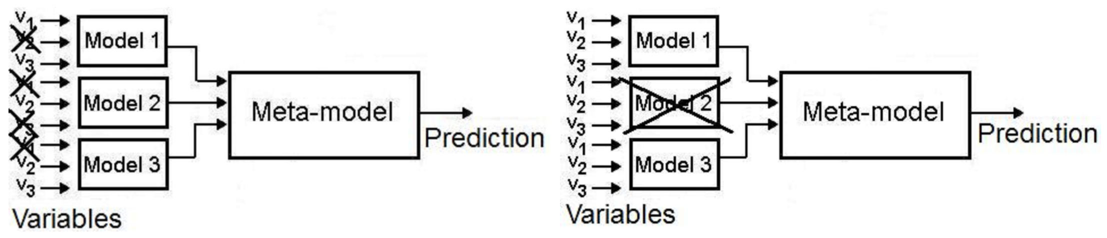


Figure 2.13: Feature selection in ensembles: at the elementary level (variable selection, left) and at the meta-level (model selection, right)

The focus of Chapter 6 is on a generic framework in order to describe *model selection strategies*. In context of stacking, model selection is feature selection at the meta-level (and variable selection is feature selection at the elementary level). This is shown in Figure 2.13. In the studied context, related works include feature selection at the elementary level [85], [110]. Some more closely related works study feature selection at the meta level, e.g. Bryll et al. [24] applies a ranking of models and selects the best models to participate in the ensemble.

Other, less closely related works include the technique developed by Zhou et al. [203], [204], who employed a genetic algorithm to find meta-level weights and selected models based on these weights. They found that the ensemble of the selected models outperformed the ensemble of all of the models. Yang et al. [196] compared model selection and model weighting strategies for ensembles of Naive-Bayes-extensions, called "Super-Parent-One-Dependence Estimators" (SPODE) [186]. All of these works focus on specific models: Zhou et al. [204] are concerned with neural networks and decision trees [203], whereas Yang et al. focused on SPODE Ensembles [196]. In contrast to them, in Chapter 6, I develop a general framework that operates with various models and meta-models.

The model selection approach by Tsymbal et al. [177] is also essentially different from mine: they select (dynamically) those models that deliver the best predictions individually. In contrast, I view the task more globally by taking interactions of models into account and thus supporting less greedy strategies.

Bacauskiene et al. [7] applied a genetic algorithm for finding the ensemble settings both at the elementary level (hyper-parameters and variable selection) and at the meta-level. However, due to their high computational costs, genetic algorithms are impractical in the case of large number of models.

Based on weighted voting of heterogeneous classifiers, Tsoumakas et al. [175], [176] developed "selective fusion", while in context of the boosting schema [138], in their pioneering work, Margineantu and Dietterich [117] observed that many classifiers of an ensemble can be pruned without harming its accuracy. In Chapter 6, I follow a similar direction, but in contrast to them, I consider the more sophisticated stacking schema.

Several authors designed and applied ensembles of classifiers for time-series classification [157], [201]. In contrast to these works, in Chapter 5, I aim at fusing *distance measures*, instead of working at the level of classifiers' outputs.

One of the core components of my framework presented in Chapter 5 is a model for pairwise decisions about whether or not two time series belong to the same class. Similar models were applied in context of record linkage (also called object identification) [44], [149]. Both of these works, however, aimed at finding groups of equivalent items. In contrast to them, in Chapter 5, I work with time series. I use multilayer perceptrons and linear regressions instead of support vector machines, and, most importantly, I focus on classification. In a joint work with my collaborators we used similar approaches for clustering web pages and images [150], [152].

Fusion of distance measures is also related to multiple kernel learning [162]. As opposed to [162], I consider time series in a simple and generic framework.

For time-series classification, Preisach [134] combined DTW with relational classifiers. She used DTW together with an appropriate threshold to build a so-called similarity graph [134]. Then, she used relational classifiers for semi-supervised classification of time series. All the techniques I propose in Part II of this book are substantially different from the ones used by Preisach: as opposed to [134], I neither build on relational classification, nor use the semi-supervised training protocol. Furthermore, I propose time series classifiers based on *individual quality estimation* (Chapter 3), exploit the presence of hubs (Chapter 4), fuse distance measures (Chapter 5), exploit mutual error compensation (Chapter 6) and extend sequential pattern mining algorithms (Chapter 7). In contrast, Preisach [134] explored none of these techniques for time-series classification.

2.6 Evaluation of time series classifiers

Throughout this book, in many experiments, I performed 10-fold-cross-validation. For this reason, I describe this protocol. Cross-validation protocols and other evaluations protocols are surveyed in [5] and [188] respectively.

While performing 10-fold-cross-validation, the entire dataset is divided into 10 disjoint splits by putting each instance (time series) into exactly one split. Out of the 10 splits, one is reserved as *test* data while the remaining 9 splits constitute the *training* data. As described in Section 2.1, the training data is used to construct the classifier. Then, in a proceeding evaluation phase, the classifier is used to determine the class labels of the test data. Then, the class labels output by the classifier are compared to the true class labels in order to allow for a quantitative evaluation of the quality of the classifier. The whole process of classifier construction and evaluation is repeated 10 times, in each of the 10 rounds a different data split serves as test data.

In order to quantitatively measure the quality of the constructed classifier, in most of the experiments, I use accuracy and classification error. If the classifier recognizes the label of an instance x correctly, we say that x is *classified correctly.* Accuracy (see also Section 2.1) is the portion of correctly classified instances. In contrast, *classification error* is the portion of instances that were misclassified (incorrectly classified). Therefore:

$$\text{accuracy} = 1 - \text{classification error} \tag{2.36}$$

2.7 Time-Series Classification Tasks and Benchmark Data

As already mentioned, time-series classification finds applications in various domains such as handwriting and speech recognition, finance, medicine, biometrics, chemistry, astronomy, robotics, networking and industry [95].

In their fundamental work, Ding et al. [56] used a large collection of 38 benchmark datasets for the evaluation of time series distance measures and classifiers. This collection (or subsets of that) was used by many authors, see e.g. [49], [142], [143]. In order to assist reproducibility and comparability, I decided to use this collection of datasets in the experiments I performed. The basic properties of these datasets are summarized in Table 2.2. These basic properties include (i) the size of the dataset (number of contained time series), (ii) the length of time series in the dataset and (iii) number of classes.

Most of these datasets are available in the UCR repository.[9] Datasets that are not (yet) contained in the UCR repository are described at this URL:
http://users.eecs.northwestern.edu/˜hdi117/listfile/VLDB08_datasets.ppt .

In this section, I would like to illustrate how time-series classification can be applied in real-world scenarios. Though this demonstration focuses on the context of *electrocardiography* (Section 2.7.1), core components of many other practical tasks can be formulated as time-series classification problems. For *handwriting recognition*, this was already shown in Figure 2.6, datasets associated with related tasks are 50words, Haptics, Symbols and WordsSynonyms. As another example, Section 2.7.2 describes how *shape recognition* in images can be solved via time-series classification and lists the datasets associated with such tasks.

2.7.1 Analysis of Electrocardiograph Signals

Electrocardiographs (ECG) are used in various ways in clinical practice: in the most simple case, directly after the ECG has been recorded, the doctor analyses it and makes the diagnosis [34]. In other cases, due to the general health state

9 http://www.cs.ucr.edu/˜eamonn/time_series_data/

Table 2.2: Basic characteristics of the datasets used in the experiments

Dataset name	Size	Length of time series	Number of classes
50words	905	270	50
Adiac	781	176	37
Beef	60	470	5
Car	120	577	4
CBF	930	128	3
ChlorineConcentration	4307	166	3
CinC	1420	1639	4
Coffee	56	286	2
DiatomSizeReduction	322	345	4
ECG200	200	96	2
ECGFiveDays	884	136	2
FaceFour	112	350	4
FacesUCR	2250	131	14
Fish	350	463	7
GunPoint	200	150	2
Haptics	463	1092	5
InlineSkate	650	1882	7
ItalyPowerDemand	1096	24	2
Lighting2	121	637	2
Lighting7	143	319	7
Mallat	2400	1024	8
MedicalImages	1141	99	10
Motes	1272	84	2
OliveOil	60	570	4
OSULeaf	442	427	6
Plane	210	144	7
SonyAIBORobotSurface	621	70	2
SonyAIBORobotSurfaceII	980	65	2
StarLightCurves	9236	1024	3
SwedishLeaf	1125	128	15
Symbols	1020	398	6
SyntheticControl	600	60	6
Trace	200	275	4
TwoLeadECG	1162	82	2
TwoPatterns	5000	128	4
Wafer	7164	152	2
WordsSynonyms	905	270	25
Yoga	3300	426	2

of the patient or when the abnormality can only be observed at a previously unknown time (in some types of arrhythmias and ischemias), the ECG signal is being recorded *continuously* for a longer time period (intensive care monitoring or monitoring with a mobile device called Holter monitor).

In such cases, automatic recognition of abnormalities in ECG signals may substantially support doctors' work. When an out-patient is wearing a mobile ECG recorder, and this device detects serious abnormalities, it can warn the patient or call an emergency service automatically. If a nurse takes care for several patients and the ECG signal becomes abnormal, the ECG recording device displays a warning so that the doctor can be called in advance. Recognizing a disease *soon* and *accurately*, either by human experts or automatically, is a difficult task: a retrospective study [75] showed that, e.g., the infants admitted to a neonatal intensive care unit had abnormal heart beating patterns 24 hours before the doctor diagnosed them with sepsis.

When an ECG signal is recorded for one day for an out-patient, the record reflects approximately one hundred-thousand heart beats. Therefore, deep analysis of the *entire* signal, due to its length, is usually impossible by human experts. Rather, the doctor focuses on the most important parts of the signal, which can be positions where an event happens (something changes) or abnormalities appear. While the ECG is being recorded with a mobile device, the patient can press a button in specific cases such as sickness, going to bed, taking pills, etc. "A special mark will be then placed into the record so that the doctors or technicians can quickly pinpoint these areas when analyzing the signal."[10] Some abnormalities, however, may be left unnoticed by the patient and therefore the system will not record any marking points corresponding to that parts of the signal. In these cases, the entire signal can be scanned and automatically analyzed by computers that produce suggestions to medical experts for abnormal parts of the signal.

In order to allow for such an analysis, the task can be formulated as a time-series classification problem. Our original definition of time-series classification (in Section 2.1) was generic, allowing for several variants of the problem, e.g.:

1. If we want to find *when* abnormalities appear in a long (e.g. 24 hours) ECG-recording, each time series in the above definition corresponds to a short segment (e.g. 1 heartbeat) of the long signal. In this case, there are two classes: *normal* signal segments belong to the first class, while *abnormal* signal segments belong to the second class. Whenever a segment is recognized as abnormal, this is a candidate for more accurate examination by human experts. If the classifier outputs a probability for each segment of being abnormal, the segments can be ranked according to this probability so that the most serious segments can be checked first by the human expert.
2. Another task is to find *where*, in which part of the patient's heart, the abnormality (e.g. an infarct) happened. This is possible as signals of different

10 http://en.wikipedia.org/wiki/Holter_Monitor

leads of the ECG correspond to the electrical activity of different parts of the heart.[11] Therefore, the task is to find in which signal(s) the abnormality is expressed. In this case, one class corresponds to the expression of the abnormality, while the signals where the abnormality is not expressed, belong to the other class.

3. If we aim at recognizing the type of abnormality, we can define several classes, one class for each disease and an additional class for normal signals.

Of course, we can combine all the above tasks (by e.g. using several classifiers), so that the result of the automatic recognition is a list of items describing *when* (at what time, at which position of the signal), *where* (at which part of the patient's heart) and *what* kind of abnormality was likely to happen. Such lists of abnormality-candidates can support human expert's diagnostic work.

Semi-automatic detection of irregularities in ECG signals has been explored by several researchers. An early approach was proposed by Bortolan and Willems who used neural networks for ECG classification [21]. Olszewski [125] examined feature extraction techniques for ECG classification, Melagni and Bazi used support vector machines [119], Syed and Chia presented an approach based on approximately conserved heart rate sequences [167], while Keogh et al. [96] applied a similarity-based, *unsupervised*, nearest-neighbor-like method in order to find "unusual" (and therefore likely irregular) segments of ECG signals.

Next, electrocardiograph datasets of the UCR Time-Series classification archive (see also Table 2.2) are described.

ECG200

The ECG200 dataset contains 200 ECG signals, each of them consisting of 96 measured values (each time series reflects 1 heartbeat). Out of the 200 time series, 133 are labeled as normal while the remaining 67 are labeled as abnormal [125]. Time series are segments of a long ECG signal, therefore the experiments on this dataset simulate the scenario when the automatic recognition system is supposed to support the doctor while she or he is searching for abnormal parts of a long ECG signal (first task listed above).

TwoLeadECG

The TwoLeadECG dataset contains 1162 ECG signals of length 82 (each time series reflects 1 heartbeat). In the TwoLeadECG dataset, only two different leads of the ECG are considered. Each signal originates from one out of these two leads. An abnormality, infarct, is expressed with different intensity in these both leads.

11 From our point of view, each lead of the ECG is a signal that reflects the electrical activity of a certain part of the heart, different leads correspond different parts. See also: http://en.wikipedia.org/wiki/Electrocardiography

As the two classes correspond to two different leads of the ECG, experiments on this dataset simulate the second scenario listed above, when we aim at finding in which part of the patient's heart the abnormality occurred.

ECGFiveDays

The ECGFiveDays dataset contains 884 ECG signals of length 136 taken from a 67 year old male. Two classes correspond to two different days, 12th and 17th November 1990.

CinC

CinC (or CinC_ECG_Torso) contains 1420 ECG signals of length 1639. Four classes of the data correspond to four different persons.

2.7.2 Shape Recognition in Images

One way of processing image data, and recognizing shapes is based on converting images into time series, then time-series classification algorithms can be applied, see e.g. [185]. "A two-dimensional shape, such as an arrowhead, can be converted to a one-dimensional pseudo time series by tracing the boundary of the shape and recording the local angle."[197]

The datasets *FacesUCR*, *FaceFour* and *OSULeaf* were produced this way. For FacesUCR and FaceFour, photographs of graduate students were taken and converted into time series, the task is to recognize faces based on their head profiles. OSULeaf was constructed based on the images available at the Herbarium Website[12].

2.8 Summary

In this chapter, the time-series classification problem was described. I emphasized that this problem finds applications in various domains, and some of the real-world tasks related to time-series classification were described in more detail. From the research point of view, an interesting recent observation is that simple nearest neighbor models together with the DTW distance measure work surprisingly well, and in general they outperform many other state-of-the-art approaches. Therefore, in the proceeding chapters I present techniques that I developed in order to make the aforementioned model faster and more accurate.

12 http://web.engr.oregonstate.edu/~tgd/leaves/dataset/herbarium

Part II

Fusion Methods for Time-Series Classification

Chapter 3

Individual Quality Estimation

As described in Section 2.4, k-NN classifiers have been shown to be very successful in the time series domain. In complex time series data sets, however, the intrinsic characteristics, such as the dimensionality and the skewness in the distribution of error (see Section 2.4.3), may vary from region to region. As a consequence, a single, global choice for k (≥ 1) can become suboptimal, since each individual region of the dataset may require a different value of k. Therefore, the motivation of my study is to investigate how to perform k-NN time-series classification in a more fine grained fashion that adapts to the varying characteristics among different regions by avoiding the restriction of a single value of k.

In order to address the above problem in a principled way, and allow for accurate k-NN classification of complex time series data, I propose *individual quality* (IQ) *estimation*. IQ estimation is a mechanism that considers a range of values for k and estimates for each time series x that has to be classified, the local quality of each k-NN classifier. Local quality is defined as the likelihood of correct classification of x by the k-NN classifier. This way, a quality score $q(x, k)$ is assigned to each pair $(x, k\text{-NN})$ of a time series x and a k-NN classifier.

This information is then used by a meta level decision method that combines the predictions of k-NN classifiers. In my first approach, IQ-MAX, for each time series x to be classified, the meta level decision method selects a value of k that maximizes the estimated quality. As the quality estimation is done for each time series x individually, for different time series, different values of k can be selected. In my second approach, IQ-WV, the outputs of the different k-NN classifiers are weighted according to their estimated quality.

Since I propose the classification of time series based on a quality score estimated individually for each of them, the approach is called time-series classification based on *individual quality (IQ) estimation.*[1] IQ estimation, in particular the calculation of the quality score, is performed by regression models that are trained in order to make accurate estimations for the quality of the k-NN classifier.

1 I called an early-version of the approach as *individualized error prediction* in [29].

My contribution described in this Chapter can be summarized as follows:

1. I introduce IQ estimation. This is a general technique that can be applied in context of many different classification problems and algorithms, i.e., not just for the k-NN classification of time series. In particular, besides its use in time-series classification, I will show that IQ estimation may also be beneficial for regression problems involving conventional vector data.

2. I propose a novel method for IQ estimation and apply it in order to estimate k-NN classifiers' quality for the task of classifying time series.

3. In the IQ-estimation framework, I propose two approaches, IQ-MAX and IQ-WV that use the output of IQ estimation in two different ways in order to make time-series classification more accurate.

4. I perform a thorough experimental evaluation with 35 commonly used benchmark datasets. The results indicate significant improvement in accuracy attained by the proposed approaches when compared to the widely used 1-NN classifier and to the k-NN classifier that determines a single optimal k ($k \geq 1$).

This Chapter is organized as follows: Section 3.1 presents a motivating example, Section 3.2 outlines IQ estimation, whereas Section 3.3 describes both proposed approaches IQ-MAX and IQ-WV. Section 3.4 presents the results of the experimental evaluation. As IQ estimation is not limited to time-series classification, I develop the IQ-Reg method for regression problems over conventional vector data, which is presented in Section 3.5.

3.1 Motivating Example

As mentioned above, setting a single value of k for the k-NN time series classifier, can lead to sub-optimal accuracy, because of varying characteristics among different regions of the data. I investigate this phenomenon in more detail by first presenting a motivating example for the simple setting of binary classification of a 2-dimensional dataset.[2]

Figure 6.1 depicts a set of labeled instances from two classes that are denoted by triangles and circles. The density in the class of triangles (upper region) is larger than in the class of circles (lower region). We consider two test instances, denoted as '1' and '2' that have to be classified. We also assume that the ground-truth considers test instance '1' as a triangle, whereas '2' as a circle. For '1', its 1-NN is a circle. Thus, the 1-NN method classifies '1' incorrectly. Using the

2 In this example, in order to ease the presentation with an illustrative figure, I use a 2-dimensional dataset, thus I depart for the moment from the examination of time series data that are in general high-dimensional.

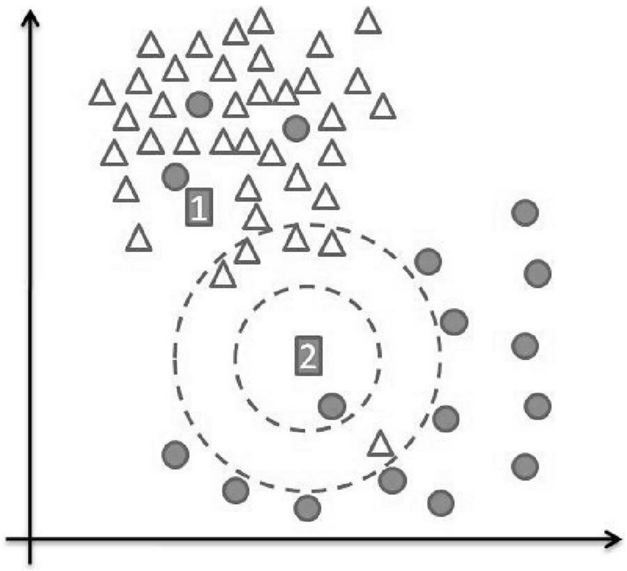


Figure 3.1: The optimal choice of the number of nearest neighbors is not unique for the entire data, but it may be different from region to region. In case of the classification of the unlabeled instance denoted by '1', $k > 1$ (e.g., $k = 3$) is required; whereas for '2' we should choose $k = 1$. (We assume that the ground-truth considers test instance '1' as a triangle, whereas '2' as a circle.)

k-NN classifier with $k > 1$ (e.g., in the range between 3 and 6), we can overcome this problem. However, the selection of a single k from the above range results in incorrect classification of test instance '2'. Due to the lower density in the circles' class, by setting k in the range between 3 and 6, we detect neighbors of '2' whose majority belongs to the triangles' class (we assumed '2' is a circle). This can be observed in Figure 6.1, where the large dashed cycle around '2' shows that among all its 6-NN, only 1 belongs to the circles' class. Thus, unlike for '1', $k = 1$ is a good choice for '2', because its 1-NN (shown inside the smaller dashed cycle) has the correct class.

The exemplified problem is amplified with time series data due to higher dimensionality and complexity in the latter. Therefore, I propose to estimate the likelihood of correct classification (the quality) of the k-NN classifiers on an individualized basis, i.e., separately for each test instance to be classified I aim at estimating the performance of each classifier. Based on this information, I want to choose the classifiers having the best estimated quality and combine their outputs. Following this approach in the example of Figure 6.1, besides the k-NN classifier, an additional model is needed which will allow for predicting that $k_1 = 3$, $k_1 = 4$, $k_1 = 5$ and $k_1 = 6$ are good choices (i.e. the likelihood of correct classification is high) when classifying instance '1', whereas $k_2 = 1$ is an appropriate choice for the classification of instance '2'. In the following I outline how the proposed approach can be developed.

3.2 Outline of IQ Estimation

The schema of the proposed mechanism for individualized quality (IQ) estimation for k-NN classifiers is depicted in Figure 3.2. This mechanism for IQ estimation

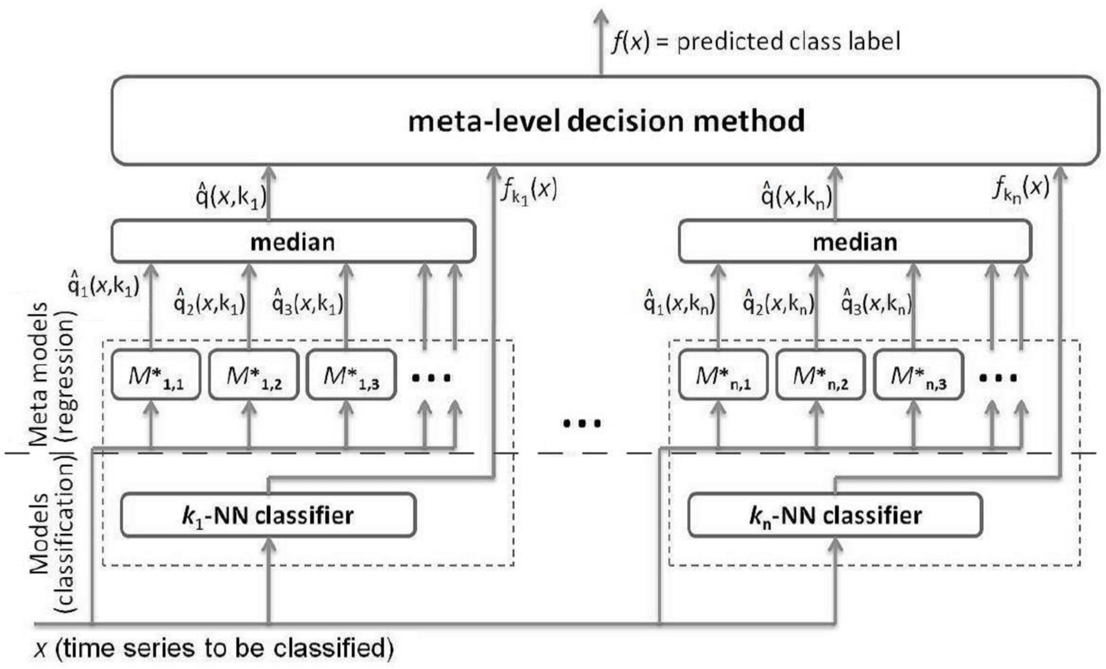


Figure 3.2: Classification based on IQ estimation

considers a range of values for k.[3] This examined range of n values for k is denoted as $\{k_i\}_{i=1}^{n}$. For each k_i-NN classifier and for each time series x that has to be classified, we estimate the local quality: the quality score $\hat{q}(x, k_i)$ denotes the estimated likelihood that the k_i-NN classifier ($1 \leq i \leq n$) correctly classifies x.

For IQ estimation, i.e. for the calculation of the estimated quality scores $\hat{q}(x, k_i)$, I introduce a second layer of models. The models of this second layer are referred to as *meta models*. They are denoted as $M^*_{i,j}$ in Figure 3.2. These meta models are regression models that are trained to predict the likelihood of correct classification of each considered k_i-NN classifier. For each k_i-NN classifier, several meta models are trained, and the median of their outputs is used as the quality score. Instead of the median, one could also use the average, however, I decided to use the median because it is generally known to be more stable than the average.

3.2.1 IQ-MAX

In my first approach, IQ-MAX, for each time series x to be classified, $k^* \in \{k_i\}_{i=1}^{n}$ is selected so that k^* maximizes estimated quality:

$$k^* = \arg\max_{k_i} \{\hat{q}(x, k_i)\} \qquad 1 \leq i \leq n \tag{3.1}$$

Finally, the k^*-NN classifier is used to classify x. This is shown in Figure 3.3.

3 Although this range is user-defined, its determination is much simpler and intuitive compared to selecting a single k. This will be asserted by our experimental results, which indicate that the range $1 - 10$ was appropriate for all examined benchmark datasets.

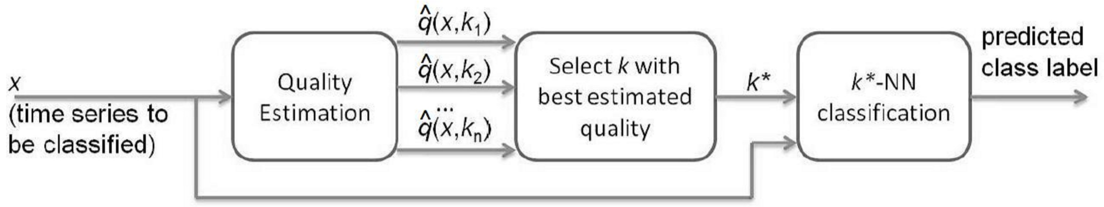


Figure 3.3: Summary of IQ-MAX approach for k-NN classification

3.2.2 IQ-WV

In my second approach, IQ-WV, the labels predicted by the k_i-NN classifiers are combined according to a weighted voting schema. The quality scores are used as weights. Formally, this approach can be described as follows. Consider the set of k_i-NN classifiers (models) that output class label c as the predicted class label for a time series x. Let M_c^x denote the set of k_i values for such k_i-NN classifiers. Denoting the class label predicted by the k_i-NN classifier for the time series x as $f_{k_i}(x)$, one can write:

$$M_c^x = \{k_i | f_{k_i}(x) = c\} \tag{3.2}$$

The weight of class label c when classifying time series x, denoted as w_c^x, can be calculated as the sum of the quality scores associated to those k_i-NN classifiers that predict c as class label:

$$w_c^x = \sum_{k_i \in M_c^x} \hat{q}(x, k_i) \tag{3.3}$$

As final result of the classification of time series x, the class label having maximal weight is selected:

$$f(x) = \arg\max_c \{w_c^x\} \tag{3.4}$$

Note that in terms of Figure 3.2, IQ-MAX and IQ-WV differ only in the meta-level decision method. In the case of IQ-MAX, this meta-level decision method consists of selecting that k_i-NN classifier which is expected to be the best. In the case of IQ-WV, the meta-level decision method is realized by weighted voting using $q(x, k_i)$ as weights of the respective predicted class labels $f_{k_i}(x)$.

A concrete algorithm for times-series classification is developed in the following section, by specifying the secondary models that perform IQ estimation.

3.3 IQ Estimation for Time-Series Classification

The proposed mechanism for classification based on Individual Quality Estimation involves two types of models:

- *Primary models* (for simplicity, I refer to them as *models*, whenever it is not confusing) – they classify time series using the k-NN approach (with the DTW distance).

- *Meta models* – they estimate the quality of the primary models, see $M^*_{i,j}$ in Figure 3.2.

In order to train the meta models, the original training data, $\mathcal{D}$, is partitioned into two disjoint subsets $\mathcal{D}_1$ and $\mathcal{D}_2$ (i.e., $\mathcal{D}_1 \cup \mathcal{D}_2 = \mathcal{D}, \mathcal{D}_1 \cap \mathcal{D}_2 = \emptyset$). $\mathcal{D}_2$ is called hold-out set. For each time series $x \in \mathcal{D}_2$, and for each examined value of k_i in the range $\{k_i\}_{i=1}^n$, the k_i-NN classifier uses $\mathcal{D}_1$ to classify x. Based on the class label of x that is given in $\mathcal{D}_2$, one determines if the k_i-NN classifier (for each $1 \leq i \leq n$) has correctly classified x. Time series x is associated with a quality score $q(x, k_i)$ that is:

$$q(x, k_i) = \begin{cases} 1 & \text{if } k_i\text{-NN classifies } x \text{ correctly} \\ 0 & \text{otherwise} \end{cases} \qquad x \in \mathcal{D}_2 \tag{3.5}$$

Thus, from the hold-out set $\mathcal{D}_2$, n new data sets $\mathcal{D}'_i$, $1 \leq i \leq n$ are generated. Each $\mathcal{D}'_i$ contains all time series of the hold-out set $\mathcal{D}_2$ along with their associated quality scores (1 or 0) for the corresponding k_i-NN classifier:

$$\mathcal{D}'_i = \Big\{\forall x \in \mathcal{D}_2 : \Big(x, q(x, k_i)\Big)\Big\} \tag{3.6}$$

Next, each generated $\mathcal{D}'_i$ acts as the training set for the corresponding meta models. Thus, based on the associated quality scores in each $\mathcal{D}'_i$, the corresponding meta models are trained as regression models in order to be able to predict the quality score of the k_i-NN classifier (i.e., the corresponding primary model) for new time series.

The described process is shown in Figure 3.4, while the schema of the architecture of the approach is depicted in Figure 3.2. The quality of each primary k_i-NN classifier is estimated by n' meta models, denoted as $M^*_{i,j}$, $1 \leq j \leq n'$.

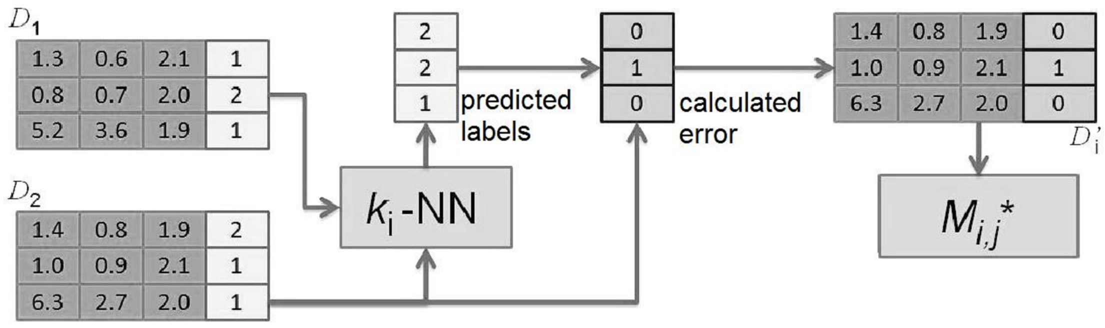


Figure 3.4: Training procedure for Individual Quality Estimation

The estimation output by the meta model $M^*_{i,j}$ is denoted as $\hat{q}_j(x, k_i)$. As already mentioned,these quality estimations are aggregated by taking their median:

$$\hat{q}(x, k_i) = \underset{\forall j}{\text{median}}\ \hat{q}_j(x, k_i) \tag{3.7}$$

Each meta model $M^*_{i,j}$ is implemented as a k'_j-NN *regression model* based on the DTW distance (k'_j is used in order to distinguish from k_i that is used in the primary k_i-NN classification models). The secondary level prediction of $M^*_{i,j}$ for a new time series $x^* \notin \mathcal{D}$ is calculated as

$$\hat{q}_j(x^*, k_i) = \frac{\sum\limits_{x_n \in \mathcal{N}(x^*)} q(x_n, k_i)}{k'_j} \tag{3.8}$$

where $\mathcal{N}(x^*) \subset \mathcal{D}'_i$ is the set of k'_j nearest neighbors of x^* and $q(x_n, k_i)$ is the associated quality score of each $x_n \in \mathcal{N}(x)$ and the k_i-NN classifier.

3.3.1 Efficient Implementation

The training of meta models is performed in an off-line fashion, i.e., the process of partitioning the train data into $\mathcal{D}_1$ and $\mathcal{D}_2$ and generating meta level training sets $\mathcal{D}'_i$ is performed off-line, independently of the (online) classification of unlabeled (or test) time series.

Classification of new time series

Regarding the (online) time needed to classify a time series, first note that DTW, as described in Section 2.3.2, can be calculated rapidly and the nearest neighbors can be found efficiently using recent indexing techniques. More importantly, please note that the schema presented in Figure 3.2 is a conceptual description of the approach; in order to implement it efficiently, one can exploit an interesting property of nearest neighbor classification and regression which is described below. Suppose we want to classify a time series x using its $k_1 < \ldots < k_n$ nearest neighbors.[4] For this task, most of the computational costs are spent for finding the nearest neighbors. However:

1. While classifying x with k_n nearest neighbors, with minimal additional overhead one can also produce the classification results for the other cases, because the sets of $k_1, \ldots, k_{n-1}$ nearest neighbors of x, denoted as $\mathcal{N}_{k_1}(x), \ldots, \mathcal{N}_{k_{n-1}}(x)$, are subsets of the set of k_n nearest neighbors: $\mathcal{N}_{k_1}(x) \subset \mathcal{N}_{k_n}(x), \ldots, \mathcal{N}_{k_{n-1}}(x) \subset \mathcal{N}_{k_n}(x)$. Therefore, the nearest neighbors required for k_1-NN, ..., k_{n-1}-NN classifications can be found quickly among the k_n nearest neighbors.

4 In this discussion it is assumed that each k_i is much smaller than the number of *all the* time series in the train data: $\forall k_i : k_i << |\mathcal{D}|$.

2. Furthermore: if the time series in two data sets are identical, only their class labels differ, and we want to classify a new time series x^*, we need to determine the nearest neighbors of x^* only *once*. This can be exploited in our case, because the training sets of all the meta models consist of the same time series (only their meta-level class labels differ).

Taking both of the above observations into account, the whole meta level, containing in total $n \times n'$ nearest neighbor regression models, can be implemented at approximately the same computational costs as one single nearest neighbor regression model with $k' = \max\{k'_1, \ldots, k'_{n'}\}$. Similarly, all the primary level models together can be implemented at approximately the same computational costs as one single nearest neighbor classifier with $k = \max\{k_1, \ldots, k_n\}$. Therefore, the total classification time required for primary and meta models is approximately twice as the classification time required by one single nearest neighbor classifier.

Training of the models

Regarding the training procedure and the respective off-line (training) time of our approach, the computationally expensive part of the calculations consists of the classification of the time series of the hold-out dataset $\mathcal{D}_2$. The same is done in case if one searches for a globally optimal k for the k-NN classifier. Therefore, the execution time of the (off-line) training procedure of our approach is approximately the same as the time required for finding a globally optimal k for the k-NN classifier using the hold-out data set $\mathcal{D}_2$.

Summarizing the discussion in this section, please note that, despite its complex schema, my approach can be implemented efficiently. Assuming such an implementation, the execution times do not drastically differ from that of one single a k-NN classifier: the online time necessary to classify new time series only increases by a small factor of two, while the offline (training) time is approximately the same as the time required to find a globally optimal k for one single k-NN classifier.

3.4 Experimental Evaluation

In order to assist reproducibility, I provide a detailed description of the configuration of the experiments I performed.

Methods. I compare the proposed methods, denoted as IQ-MAX and IQ-WV, against two baselines: the 1-NN classifier and the k-NN classifier that selects a global k using a hold-out set from the training data. The latter baseline uses the same hold-out set as the proposed method, examines the same range of values for k, and selects the one that produces the smallest average error for all time series in the hold-out set. As distance measure, all examined methods use DTW with standard settings (see Section 2.3.2).

Datasets. Out of all the 38 datasets introduced in Section 2.7, I examined 35 datasets: 3 of them were excluded (Coffee, Beef and OliveOil) due to their small sizes (less than 100 time series). The names of the remaining data sets are listed in the first column of Table 3.3.

Parameters. At the primary level of my both proposed methods, I use k-NN classifiers with all k values in the range $1 - 10$.

For both of the proposed approaches, I implement the meta models as k'_j-NN regressors as described in Section 3.3. For IQ-MAX, in oder to keep the approach simple, I use a single value of $k' = 5$ at the meta level. The experimental results show that this was appropriate for all the examined benchmark data sets.[5] In case of IQ-WV, I used a range of $1 - 10$ as k' values.

Comparison protocol. I measure the misclassification error using 10-fold cross validation, with the exception of three data sets (FaceFour, Lighting2, and Lighting7) for which I used the leave-one-out protocol due to their small size (less than 200 instances). In each round of the 10-fold cross validation, out of the 9 training splits, I used 5 to train the primary models ($\mathcal{D}_1$), the rest 4 splits served as hold-out data ($\mathcal{D}_2$).[6] For classifying test data, i.e., *after* training IQ-MAX and IQ-VW and selecting the best k for k-NN, all training splits can be used by the primary models.

After using the above evaluation procedure, I made a striking observation about the performance of all examined methods (proposed and baselines): in the majority of data sets, the misclassification error was rather low (less than 5%). In order to have a challenging comparison with nontrivial classification, I choose to affect intrinsic characteristics of the data sets. According to the findings in [142], time series datasets usually have high intrinsic dimensionality and, thus, some of their instances tend to misclassify a surprisingly large number of other instances when using the k-NN classifier ($k \geq 1$). These instances are called "bad hubs" and are responsible for a very large fraction of the total error. For this reason, for each time series x in a dataset, I measured two quantities: the badness $f_B^1(x)$ of x and the goodness $f_G^1(x)$ of x. As described in Section 2.4.3, $f_B^1(x)$ ($f_G^1(x)$, respectively) is the total number of time series in the data set, which have x as their first nearest neighbor while having different (same, respectively) class label compared to the class label of x. For each data set, I sort all time series according to the $f_1^G(x) - f_B^1(x)$ quantity in descending order. Then I change the label of first p percent of the time series in this ranking (p varies in range 0-10%).[7] Since

5 Note that I also experimented with other single k' values for IQ-MAX. For $k' \geq 5$ I observed similar results, whereas for small values of k', such as 1 or 2, I observed worse performance.

6 Ratios other than the examined 5-4, gave similar results. In case of leave-one-out, the training data was split according to 5 to 4 proportion into $\mathcal{D}_1$ and $\mathcal{D}_2$.

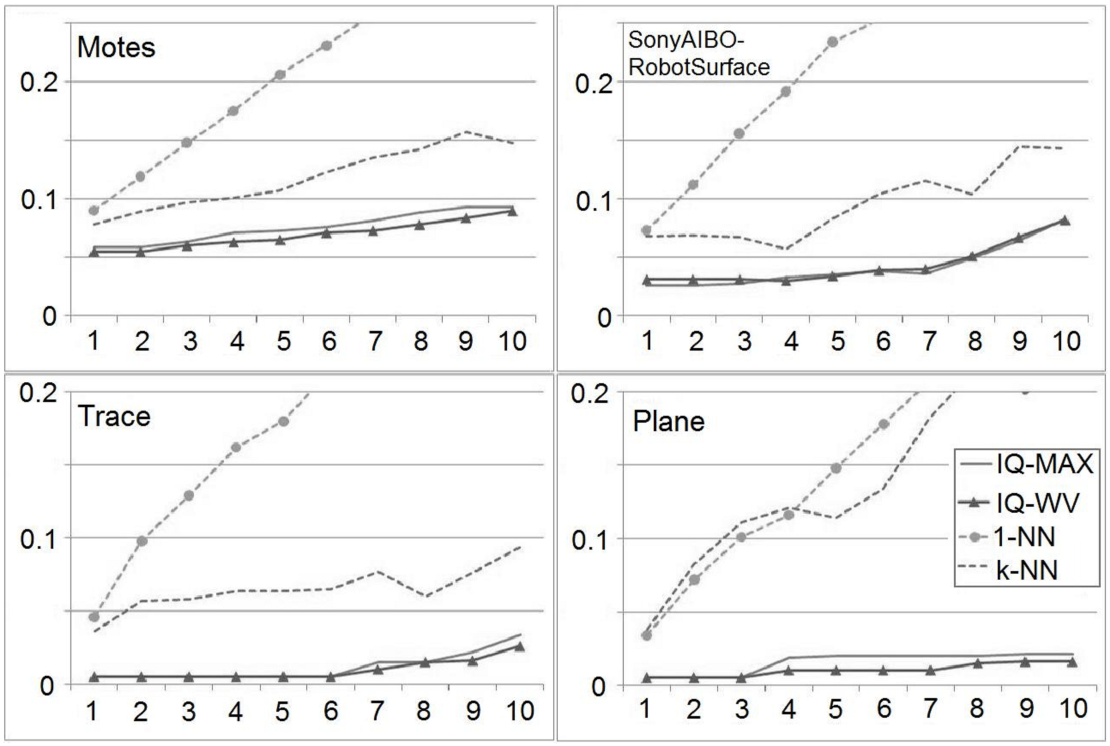


Figure 3.5: Classification error (vertical axis) depending on the noise level p (horizontal axis) for some data sets

the above procedure results in data sets that have stronger "bad hubs" and a less clear separation between classes, the comparison among the examined methods both becomes more challenging and can better characterize the robustness of the methods.

3.4.1 Experimental Results

The results on classification error are summarized in Table 3.2, Table 3.3 and Table 3.4. For brevity, the tables only report results at $p = 0\%$, $p = 1\%$ and 5% noise[8], however, I observed similar tendencies at all the examined noise ratios. For four data sets, Motes, SonyAIBORobotSurface, Trace and Plane Figure 3.5 shows the classification error for all the examined values of p.

Whenever IQ-MAX and/or IQ-WV significantly outperform(s) one or both of the baselines, IQ-MAX and/or IQ-WV is/are marked bold in Table 3.2, Table 3.3 and Table 3.4. Additionally, I provide two symbols in the form $\pm/\pm$ in order to denote the result of statistical-significance test (t-test at 0.05 level) against 1-NN

7 The time series whose labels were changed by this procedure, are assigned to an additional class (not included in the original data set). In order to keep the experimental evaluation meaningful, the time series with changed labels were excluded from the test set.

8 For simplicity, I refer to the above-described enrichment in bad hubs as *noise*.

and k-NN, respectively, where + denotes significance and − its absence. Similarly, if one of the baselines significantly outperforms IQ-MAX or IQ-WV, it is marked bold and I provide the result of statistical-significance test (again in form of $\pm/\pm$) against IQ-MAX and IQ-WV.

Table 3.1 summarizes these results by reporting the number of cases, per noise level and in total, when IQ-MAX and IQ-WV wins/looses against the baselines, 1-NN and k-NN. (In parentheses I report in how many cases the differences are statistically significant).

As shown, in the vast majority of the cases both IQ-MAX and IQ-WV outperform the competitors, often significantly; whereas when they loose, the difference is usually not significant. Note that in several cases, the errors of my IQ estimation based methods are an order of magnitude lower than the error of 1-NN and k-NN: see e.g. TwoLeadECG at $p = 1\%$ (for both IQ-MAX and IQ-VW) and at $p = 5\%$ (for IQ-WV) in Table 3.3 and Table 3.4, furthermore GunPoint and Trace at $p = 5\%$ (for both IQ-MAX and IQ-VW) in Table 3.4. As expected, IQ-WV, which is more sophisticated than IQ-MAX, generally performs better than IQ-MAX, e.g. at $p = 5\%$ noise IQ-VW is superior to IQ-MAX on 29 datasets, whereas IQ-MAX is better than IQ-VW in only 2 cases.

3.4.2 Execution Time

Even for the large data sets, I observed the execution times of my methods to be reasonable, e.g. for IQ-MAX, I measured 12.9, 19.8 and 6.8 minutes off-line (training) times (on a Xeon 2.3 GHz processor) for the data sets Wafer, TwoPatterns and ChlorineConcentration respectively. Note that this off-line time refers to the time required for training which has to be performed only *once*. The same off-line time was necessary for k-NN to find the globally optimal k. This is because training is dominated by the classification of the hold-out set $\mathcal{D}_2$ in both cases.

Table 3.1: Number IQ-MAX's and IQ-WV's wins/looses against 1-NN and k-NN. (In parentheses I report in how many cases wins/looses are statistically significant.)

	$p = 1$ %	$p = 5$ %	total
IQ-MAX wins against 1-NN	29 (20)	34 (29)	63 (49)
IQ-MAX looses against 1-NN	5 (1)	1 (0)	6 (1)
IQ-MAX wins against k-NN	30 (15)	29 (9)	59 (24)
IQ-MAX looses against k-NN	5 (1)	5 (1)	10 (2)
IQ-WV wins against 1-NN	31 (21)	35 (31)	66 (52)
IQ-WV looses against 1-NN	3 (1)	0	3 (1)
IQ-WV wins against k-NN	31 (15)	34 (22)	65 (37)
IQ-WV looses against k-NN	4 (1)	0	4 (1)

Table 3.2: Classification error at $p = 0\%$ noise

Dataset	IQ-MAX	IQ-WV	1-NN	k-NN
50 Words	0.229	0.233	**0.203** -/+	**0.203** -/+
Adiac	0.370	0.374	**0.344** +/+	**0.344** +/+
Car	0.275	0.267	0.250	0.258
CBF	0.000	0.000	0.000	0.000
ChlorineConcentration	0.048	0.055	**0.004** +/+	**0.004** +/+
CinC	0.003	0.001	0.000	0.000
DiatomSizeReduction	0.006	0.006	0.003	0.003
ECG200	0.135	0.125	0.115	0.120
ECGFiveDays	0.012	0.010	0.009	0.009
FaceFour	0.054	0.063	0.045	0.054
FacesUCR	0.028	0.027	**0.015** +/+	**0.015** +/+
Fish	0.229	0.217	**0.203** +/-	0.200
GunPoint	0.010	0.010	0.010	0.015
Haptics	**0.477** +/-	**0.471** +/-	0.547	0.512
InlineSkate	0.457	0.432	0.429	0.448
ItalyPowerDemand	0.038	0.033	0.041	0.042
Lighting2	0.182	0.149	0.091	0.107
Lighting7	0.245	0.210	0.210	0.224
Mallat	0.014	0.012	0.013	0.013
MedicalImages	0.211	0.201	0.197	0.203
Motes	0.057	0.054	0.053	0.064
OSULeaf	0.303	0.292	**0.251** +/-	**0.251** +/-
Plane	0.005	0.005	0.000	0.000
SonyAIBORobotS.	0.027	0.029	0.021	0.021
SonyAIBORobotS.II	0.035	0.031	**0.019** +/+	**0.019** +/+
StarLightCurves	0.075	**0.071** -/+	0.073	0.086
Symbols	0.023	0.024	0.019	0.019
SyntheticControl	0.018	0.018	0.017	0.015
SwedishLeaf	0.167	0.166	0.159	0.171
Trace	0.005	0.005	0.000	0.000
TwoLeadECG	0.001	0.001	0.001	0.001
TwoPatterns	0.000	0.000	0.000	0.000
Wafer	0.003	0.003	0.003	0.003
WordsSynonyms	0.217	0.217	**0.194** -/+	**0.194** -/+
Yoga	0.068	0.071	**0.055** +/+	**0.056** -/+

Table 3.3: Classification error at $p = 1\%$ noise

Dataset	IQ-MAX	IQ-WV	1-NN	k-NN
50 Words	0.239	0.241	0.249	0.242
Adiac	0.373	0.377	0.381	0.384
Car	0.279	0.270	0.278	0.303
CBF	**0.004** +/+	**0.001** +/+	0.106	0.047
ChlorineConcentration	0.053	0.058	**0.021** +/+	**0.021** +/+
CinC	**0.003** +/+	**0.001** +/+	0.033	0.011
DiatomSizeReduction	**0.006** +/+	**0.006** +/+	0.031	0.038
ECG200	0.136	0.126	0.171	0.156
ECGFiveDays	**0.013** +/+	**0.010** +/+	0.041	0.045
FaceFour	0.063	0.072	0.108	0.072
FacesUCR	**0.029** +/+	**0.026** +/+	0.059	0.039
Fish	0.228	**0.219** +/-	0.254	0.239
GunPoint	**0.010** -/+	**0.010** -/+	0.036	0.061
Haptics	**0.490** +/-	**0.482** +/-	0.582	0.532
InlineSkate	0.469	**0.442** -/+	0.461	0.483
ItalyPowerDemand	**0.038** +/+	**0.034** +/+	0.087	0.081
Lighting2	0.192	0.142	0.133	0.125
Lighting7	0.254	0.211	0.254	0.289
Mallat	**0.014** +/-	**0.012** +/-	0.055	0.018
MedicalImages	0.212	0.203	0.228	0.234
Motes	**0.059** +/+	**0.055** +/+	0.090	0.078
OSULeaf	0.320	0.301	0.287	0.292
Plane	**0.005** +/+	**0.005** +/+	0.034	0.038
SonyAIBORobotS.	**0.026** +/+	**0.031** +/-	0.073	0.068
SonyAIBORobotS.II	**0.034** +/+	**0.032** +/+	0.063	0.067
StarLightCurves	**0.076** +/-	**0.071** +/-	0.119	0.073
Symbols	**0.023** +/-	**0.024** +/-	0.061	0.031
SyntheticControl	**0.020** +/-	**0.018** +/-	0.076	0.017
SwedishLeaf	**0.170** +/+	**0.169** +/+	0.206	0.197
Trace	0.005	0.005	0.046	0.036
TwoLeadECG	**0.001** +/+	**0.001** +/+	0.041	0.052
TwoPatterns	**0.001** +/+	**0.001** +/+	0.065	0.007
Wafer	**0.003** +/-	**0.003** +/-	0.042	0.004
WordsSynonyms	0.224	0.220	0.238	0.241
Yoga	**0.071** +/+	**0.072** +/+	0.099	0.114

Table 3.4: Classification error at $p = 5\%$ noise. The last column shows the overall standard deviation of the secondary models for all k' values in the range $5 \leq k' \leq 10$.

Dataset	IQ-MAX	IQ-WV	1-NN	k-NN	$\sigma_{k'}$
50 Words	**0.270** +/-	**0.254** +/-	0.388	0.260	0.021
Adiac	**0.415** +/-	**0.411** +/+	0.508	0.451	0.018
Car	**0.310** +/-	**0.283** +/+	0.416	0.353	0.051
CBF	**0.043** +/-	**0.034** +/+	0.328	0.057	0.044
ChlorineConc.	0.077	0.073	0.075	0.075	0.021
CinC	**0.008** +/-	**0.004** +/+	0.143	0.021	0.048
DiatomSizeRed.	**0.010** +/+	**0.010** +/+	0.141	0.049	0.058
ECG200	**0.150** +/-	**0.124** +/-	0.313	0.134	0.073
ECGFiveDays	**0.020** +/+	**0.017** +/+	0.164	0.136	0.041
FaceFour	0.075	0.075	0.234	0.112	n/a
FacesUCR	**0.044** +/-	**0.033** +/+	0.193	0.046	0.028
Fish	**0.244** +/-	**0.244** +/-	0.386	0.280	0.042
GunPoint	**0.016** +/+	**0.011** +/+	0.162	0.176	0.107
Haptics	**0.540** +/-	**0.533** +/-	0.681	0.553	0.026
InlineSkate	0.523	**0.504** +/+	0.562	0.570	0.025
ItalyPowerDemand	**0.059** +/-	**0.047** +/-	0.237	0.060	0.030
Lighting2	0.209	0.157	0.270	0.209	n/a
Lighting7	0.279	0.243	0.426	0.338	n/a
Mallat	**0.019** +/+	**0.017** +/+	0.178	0.034	0.030
MedicalImages	**0.228** +/-	**0.211** +/+	0.339	0.256	0.023
Motes	**0.073** +/+	**0.065** +/+	0.206	0.107	0.038
OSULeaf	0.363	**0.308** +/+	0.402	0.345	0.028
Plane	**0.020** +/+	**0.010** +/+	0.148	0.114	0.093
SonyAIBOR.S.	**0.035** +/+	**0.034** +/+	0.234	0.083	0.047
SonyAIBOR.S.II	**0.037** +/-	**0.034** +/+	0.212	0.119	0.031
StarLightCurves	**0.096** +/-	**0.089** +/+	0.253	0.098	0.003
Symbols	**0.029** +/-	**0.031** +/-	0.196	0.036	0.040
SyntheticControl	**0.028** +/-	**0.035** +/-	0.227	0.058	0.057
SwedishLeaf	**0.189** +/+	**0.181** +/+	0.328	0.216	0.023
Trace	**0.005** +/-	**0.005** +/-	0.180	0.064	0.074
TwoLeadECG	**0.005** +/+	**0.002** +/+	0.175	0.025	0.042
TwoPatterns	**0.014** +/-	**0.012** +/+	0.236	0.019	0.026
Wafer	**0.006** +/-	**0.005** +/-	0.160	**0.005** +/-	0.025
WordsSynonyms	**0.270** +/-	**0.257** +/+	0.379	0.287	0.032
Yoga	**0.085** +/-	**0.080** +/+	0.223	0.115	0.020

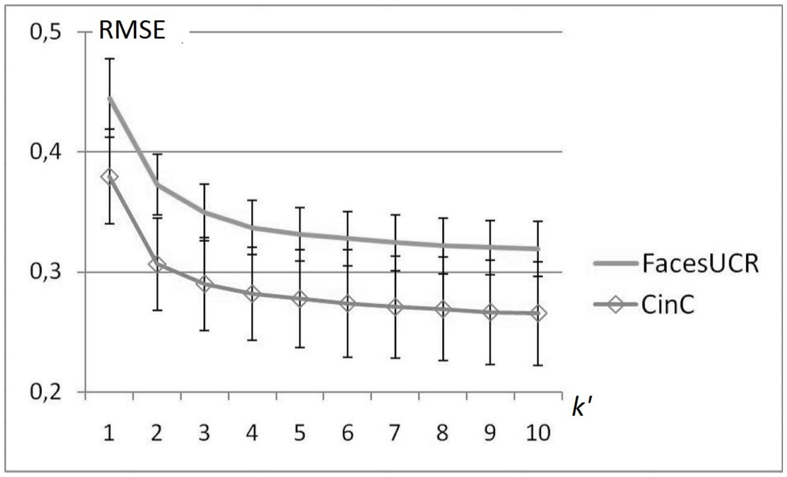


Figure 3.6: RMSE of the meta models for various values of k' for FacesUCR and CinC at $p = 5$ % noise. (Bars show standard deviations.)

For IQ-MAX the on-line time required to classify a new time series was 0.22, 0.51 and 0.23 seconds (for the above mentioned datasets). For IQ-WV, I measured similar execution times which justify the expectations based on the discussion in Section 3.3.1. Therefore, my approaches are able to maintain fast classification of new time series.

3.4.3 Meta model's quality

I also examined the error of the quality estimation performed by the meta models. In order to gain insight in to the role of the number of nearest neighbors at the meta level, k', Figure 3.6 depicts the root mean squared error (RMSE)[9] of the meta models as function of k' for two characteristic datasets. Increasing k' initially leads to improvement of the meta models. However, after a point ($k' = 5$ in the examined cases) the error of the meta models becomes stable, i.e. it does not change significantly. This tendency is similar in the range $5 \leq k' \leq 10$ for all data sets. The difference of the meta models' performance between $k' = 5$ and $k' = 10$ is very small in general, which is shown in Figure 3.7. Further, the last column of Table 3.4 shows for each dataset the overall standard deviation of the secondary models for all k' values in the range $5 \leq k' \leq 10$. The resulting small values in all cases indicate the stability of the approach w.r.t. k'. This justifies the use of $k' = 5$ for the IQ-MAX approach.

9 RMSE is defined in Section 6.1. Given a primary model and a meta model, the RMSE of the meta model describes how well the meta model estimates the quality of the primary model. Smaller values correspond more accurate meta models. While calculating the RMSE of the meta model, for each time series x in the test data, $\hat{q}(x, k_i)$, the quality estimation output by the meta model is compared to the true quality of the primary model, $q(x, k_i)$ which is 1 if the primary model classifies x correctly and 0 otherwise. The reported results are aggregated over 10 folds and various primary models.

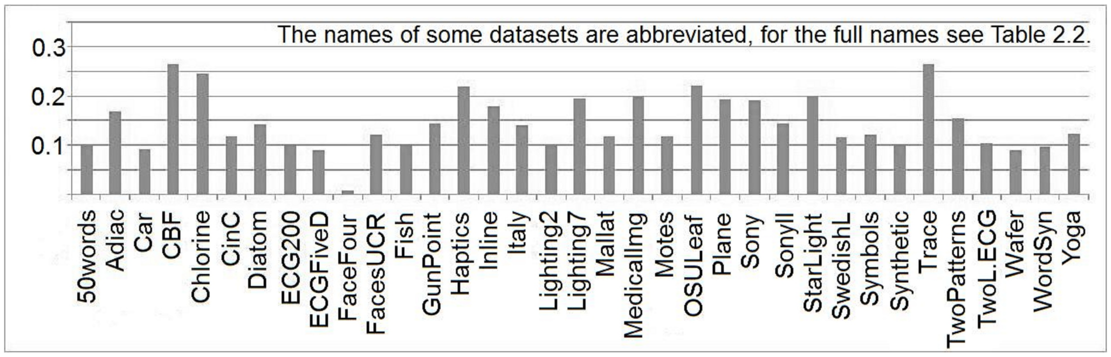


Figure 3.7: Average difference of the meta models' performance (in RMSE) between using $k' = 5$ and $k' = 10$ at $p = 5\%$ noise for each dataset

3.5 IQ-Reg: IQ Estimation for Regression

As mentioned, Individual Quality Estimation is not limited to the context of time-series classification. Here, using the framework of IQ Estimation, I develop the IQ-Reg approach for regression problems. Since IQ-Reg is similar to IQ-MAX and IQ-WV, therefore, the description below focuses on the differences.

Suppose we are given a primary regression model M and we aim at enhancing M with Individual Quality Estimation. The major steps of IQ-Reg are:

1. Split the labeled training data into two subsets $\mathcal{D}_A$ and $\mathcal{D}_B$.

2. Train the elementary model M on $\mathcal{D}_A$.

3. Let M predict the labels of $\mathcal{D}_B$. As opposed to the case of classification, for regression problems, these labels are continuous.

4. As the true labels of $\mathcal{D}_B$ are known, for each instance $x \in \mathcal{D}_B$, one can calculate the quality score of the prediction as the absolute error of the predicted labels:
$$q(x, M) = c(x) - f_M(x) \tag{3.9}$$
where $f_M(x)$ denotes the prediction of the primary model M and $c(x)$ is the true, in this case continuous, label of x.

5. Train a regression model M^* on $\mathcal{D}_B$ using the calculated errors as labels.

An example for the training procedure of IQ-Reg is shown in Figure 3.8.

When an unlabeled instance x' is processed, the primary model M predicts its label, while the meta model M^* estimates the error of this prediction. Then the final predicted label of x' is the label predicted by M corrected with the estimated error, i.e., the output of the meta-model, denoted as $f_{M^*}(x')$:

$$f_{IQReg}(x') = f_M(x') + f_{M^*}(x') \tag{3.10}$$

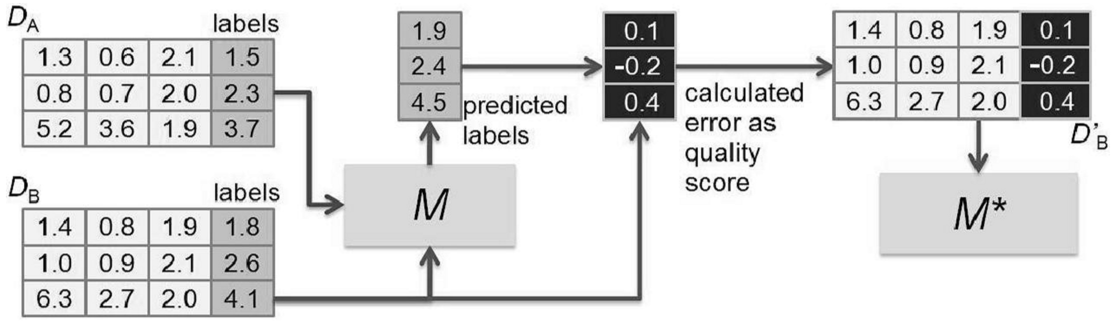


Figure 3.8: Training procedure of IQ-Reg

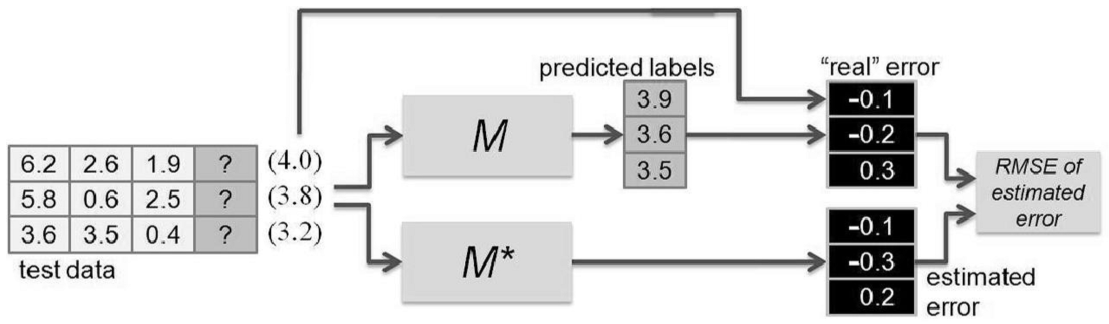


Figure 3.9: Procedure of the evaluation of IQ-Reg

3.5.1 Experiments with IQ-Reg

IQ-Reg is generic as it allows us to apply various regression models both at the primary level (as M) and at the meta-level (as M^*). Whenever a particular choice of M and M^* is made, one would like to evaluate how successfully M^* could predict the errors of M. As this evaluation procedure is non-trivial, I continue by outlining the applied evaluation protocol.

In order to evaluate IQ-Reg, one first trains both M and M^* on training data as described above. Then, using M^*, one can estimate the error for each instance of the disjoint test data $\mathcal{D}_{Test}$. Simultaneously, one can predict the labels of $\mathcal{D}_{Test}$ using M. Comparing the predicted labels to the ground-truth of the test data, one can calculate the true errors of the predicted labels. Finally, one can compare the estimated errors to the true errors. When doing so, I calculate RMSE (root mean squared error) between the vector of true errors and the vector of estimated errors. This process is shown in Figure 3.9.

In the first experiment, I systematically investigated various combinations of primary models M and meta models M^* for IQ-Reg. I used the following publicly available real-world datasets from the UCI repository [69]: i) Communities, ii) WineQuality (both red wines and white wines) and iii) Parkinson (both targets: motoric abilities and total abilities). As I observed very similar trends on all data

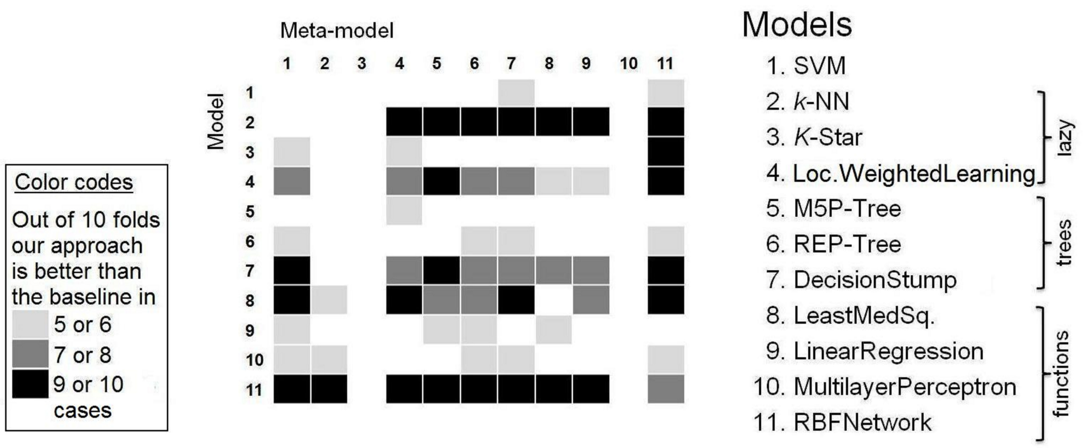


Figure 3.10: Results of the experimental evaluation: the performance of IQ-Reg with various combinations of models and meta-models (dataset: Communities). The same model types were used at the elementary and meta levels. For more details about the models see the WEKA software package [188].

sets, only results on the Communities data set are reported. As a simple baseline, I used the meta-model M^*_{bl} that estimates that the prediction of the primary model M is always perfectly accurate.

I performed 10-fold-cross-validation: in each round, out of the 9 training splits, 5 splits constituted D_A and the remaining 4 splits served as D_B.

Figure 3.10 summarizes the results: it shows for all the examined combinations of models and meta-models, in how many folds the meta model was better than the baseline. The applied models are listed in the right of the figure, I used the implementations from the WEKA software package[10] [188]. In the matrix, the horizontal dimension corresponds to the applied meta-model M^*, while the primary models are listed along the vertical dimension. For example, the 4th column position in the 2nd row corresponds to the combination where the primary model is k-NN and the meta model is LWL. The color of the cell shows how many times (in how many rounds of the 10-fold-cross-validation) the trained meta-model M^* was better than the baseline M^*_{bl}. Black, dark gray, and light gray cells mean that M^* is better than M^*_{bl} 9 or 10 times, 7 or 8 times, and 5 or 6 times respectively.

As a general observation, one can see that for almost all of the elementary models, there are meta-models that are capable to deliver relatively good error estimations, although the particular choice of the meta-model is important: some of the meta-models deliver very poor estimations, while others work well. As a further observation, one can see that RBF-Networks and SVMs seem to work generally well as meta-models.

10 http://www.cs.waikato.ac.nz/ml/weka/index.html

Chapter 4

Instance Selection

As described in Section 2.4.2, the efficiency of nearest-neighbor classification can be improved with several methods, one of the commonly applied approaches is *instance selection*. This approach reduces the size of the training set by selecting the most representative instances. Then, only the selected ones are used while classifying new instances. Instance selection is orthogonal to the other speed-up techniques described in Section 2.4.2, as it can be used together with them in order to achieve high efficiency.

According to the findings summarized in Section 2.4.3, the presence of hubs has to be taken into account for instance selection. *Score functions*, which allow for ranking the instances, constitute a core component of instance selection algorithms. In this section, I will explain that the suitability of such score functions is based on the hubness property that holds in most time series data sets.

As major contributions of this section, I analyze the complexity of the instance selection problem and show that my approach, INSIGHT, which is based on the aforementioned score functions, optimize the *coverage* of training data, in the sense that time series x covers an other time series x', if x' can be classified correctly using x.

I also propose the usage of graphs for better understanding the instance selection problem and for the design of new algorithms: the proposed notion of graph-coverage supports the analysis of the properties of my approach from the point of view of coverage maximization. For the above reasons, my approach is denoted as *Instance Selection based on Graph-coverage and Hubness for Time-series* or *INSIGHT*. INSIGHT generalizes FastAWARD [194], a state-of-the-art instance selection method for time-series classification (Section 2.4.2), by being able to use several formulae for scoring instances. Furthermore, instead of the "pessimistic" selection applied in FastAWARD, INSIGHT uses an "optimistic" strategy: while FastAWARD iteratively discards bad instances, beginning with the worst one, and the instances remaining at the end are considered as the selected ones, INSIGHT focuses on the identification of the good instances, and directly selects the good ones. Moreover, compared to FastAWARD, INSIGHT is simpler and computationally more efficient by avoiding iterative re-ranking of instances. I evaluate INSIGHT in experiments on the collection of time-series classification datasets described in Section 2.7 and I compare INSIGHT against FastAWARD. The results show that INSIGHT substantially outperforms FastAWARD both in terms of classification accuracy and execution time required for performing the selection of instances.

Most likely, the reason for this is that FastAWARD follows some decisions whose nature can be considered *ad-hoc* (such as the application of an iterative procedure or the tie-breaking criterion [194]). In particular, I provide insights into the fact that the iterative procedure of FastAWARD is not a well-formed decision, since its large computation time can be saved by ranking instances only once. Furthermore, I observed the warping window size to be less crucial, and therefore I simply use a fixed window size for INSIGHT (that outperforms FastAWARD using adaptive window size).

This Chapter is organized as follows. Section 4.1 discusses score-based instance selection in the light of the hubness phenomenon. In section 4.2, I analyze the complexity of the instance selection problem, and the properties of my approach. Section 4.3 presents the experimental results.

4.1 Score functions based on Hubness

INSIGHT performs instance selection by assigning a score to each instance and selecting instances with the highest scores (see Algorithm 5), therefore the ”intelligence” of INSIGHT is hidden in the applied score function. In this section, I explain the suitability of score functions in the light of the hubness property described in Section 2.4.3.

Good 1-occurrence Score

INSIGHT can use scores that take the good 1-occurrence of an instance x into account. Thus, a simple score function is the *good 1-occurrence score* $g_G(x)$:

$$g_G(x) = g_G^1(x) \tag{4.1}$$

Henceforth, when there is no ambiguity, the upper index 1 is omitted.

Relative Score

While x is being a good hub, at the same time it may appear as bad neighbor of several other instances. Thus, INSIGHT can also consider scores that take bad occurrences into account. This leads to scores that relate the good occurrence of an instance x to either its total occurrence or to its bad occurrence. For simplicity, I focus on the following *relative score*, however, other variations could be used too:

Algorithm 5 INSIGHT

```
Require: Time series dataset $\mathcal{D}$, Score Function $g$,
    Number of selected instances $N$
Ensure: Set of selected instances (time series) $\mathcal{D}'$
1: Calculate score function $g(x)$ for all $x \in \mathcal{D}$
2: Sort all the time series in $\mathcal{D}$ according to their scores $g(x)$
3: Select the top-ranked $N$ time series and return the set containing them
```

Relative score $g_R(x)$ of a time series x is the fraction of good 1-occurrences and total occurrences plus one (to avoid division by zero):

$$g_R(x) = \frac{g_G^1(x)}{g_N^1(x) + 1} \tag{4.2}$$

Xi's score

Interestingly, $g_G^k(x)$ and $g_B^k(x)$ allows us to interpret the ranking criterion used by Xi et al. in FastAWARD [194] as another form of score for relative hubness:

$$g_{Xi}(x) = g_G^1(x) - 2g_B^1(x) \tag{4.3}$$

4.2 Coverage and Instance Selection

Based on scoring functions, such as the ones described in the previous section, INSIGHT selects top-ranked instances (see Algorithm 5). However, while ranking the instances, it is also important to examine the interactions between them. For example, suppose that the first top-ranked instance allows correct 1-NN classification of almost the same instances as the second top-ranked instance. The contribution of the second top-ranked instance is, therefore, not important with respect to the overall classification. In this section, I describe the concept of *coverage graphs*, which helps to examine the aforementioned aspect of interactions between the selected instances. In Section 4.2.1, I examine the general relationship between coverage graphs and instance-based learning methods that can be used for classification, whereas in Section 4.2.2, I focus on the case of 1-NN time-series classification.

4.2.1 Coverage graphs for Instance-based Learning

I first define the notion of coverage graphs[1] , which in the sequel allows to cast the instance-selection problem as a graph-coverage problem:

1 Instead of the term *coverage graph*, one could alternatively use *similarity graph*.

Definition 8 (Coverage graph)**.** *A coverage graph $G_c = (V, E)$ is a directed graph, where each vertex $v \in V_{G_c}$ corresponds to a time series of the (labeled) training set. A directed edge from vertex v_x to vertex $v_{x'}$ denoted as $(v_x, v_{x'}) \in E_{G_c}$ states that instance x' contributes to the correct classification of instance x.*

I first examine coverage graphs for the general case of instance-based learning methods, which include the k-NN ($k \geq 1$) classifier and its generalizations, such as adaptive k-NN classification where the number of nearest neighbors k is chosen adaptively for each object to be classified [126], [187].[2] In this context, the *contribution* of an instance x' to the correct classification of an other instance x refers to the case when x' is among the nearest neighbors of x and they have the same label.

Based on the definition of the coverage graph, the coverage of a vertex and the coverage of a vertex-set are defined as follows:

Definition 9 (Coverage of a vertex and of vertex-set)**.** *A vertex v covers an other vertex v' if there is an edge from v' to v; $C(v)$ denotes[3] the set of all vertices covered by v: $C(v) = \{v' | v' \neq v \wedge (v', v) \in E_{G_c}\}$. Moreover, a set of vertices S_0 covers all the vertices that are covered by at least one vertex $v \in S_0$:*

$$C(S_0) = \bigcup_{\forall v \in S_0} C(v) \tag{4.4}$$

Let G_c denote the coverage graph constructed based on the (labeled) training data. Following the common assumption that the distribution of the test (unlabeled) data is similar to the distribution of the training data, the more vertices are covered in G_c, the better prediction is expected for new (unlabeled) data. Therefore, the objective of an instance-selection algorithm is to find the vertex-set S (i.e., select instances) that covers the entire set of vertices (i.e., the entire training data), or formally: $C(S) = V_{G_c}$. This, however, may not be always possible, such as when there exist vertices that are not covered by any other vertex. If a vertex v is not covered by any other vertex, this means that the out-degree of v is zero (there are no edges going from v to other vertices). Denote the set of such vertices with $V^0_{G_c}$. Then, an ideal instance selection algorithm should cover all *coverable* vertices, i.e., for the selected vertices S, an ideal instance selection algorithm should fulfill:

$$\bigcup_{\forall v \in S} C(v) = V_{G_c} \setminus V^0_{G_c} \tag{4.5}$$

In order to achieve the aforementioned objective, the trivial solution is to select all the instances of the training set, i.e., chose $S = V_{G_c}$. This, however is

2 Please notice that in the general case the resulting coverage graph has no regularity regarding both the in- and out-degrees of the vertices (e.g., in the case of k-NN classifier with adaptive k).

3 Instead of $C(v)$, one could alternatively use the notation $fanin(v)$.

not an effective instance selection algorithm, as the major aim of discarding less important instances is not achieved at all. Therefore, the natural requirement regarding the ideal instance selection algorithm is that it selects the *minimal* amount of those instances that together cover all coverable vertices. This way, the instance selection task is cast as a coverage problem:

Instance selection problem (ISP) — We are given a coverage graph $G_c = (V, E)$. We aim at finding a set of vertices $S \subseteq V_{G_c}$ so that: i) all the coverable vertices are covered (see Equation 4.5), and ii) the size of S is minimal among all those sets that cover all coverable vertices.

Next, I will show that this problem is NP-complete, because it is equivalent to the set-covering problem (SCP), which is NP-complete [47]. I proceed with recalling the set-covering problem:

Set-covering problem (SCP) — "An instance $(X, \mathcal{F})$ of the set-covering problem consists of a finite set X and a family $\mathcal{F}$ of subsets of X, such that every element of X belongs to at least one subset in $\mathcal{F}$. (...) We say that a subset $F \in \mathcal{F}$ covers its elements. The problem is to find a minimum-size subset $\mathcal{C} \subseteq \mathcal{F}$ whose members cover all of X"[47]. Formally: the task is to find $\mathcal{C} \subseteq \mathcal{F}$, so that $|\mathcal{C}|$ is minimal and $X = \bigcup\limits_{\forall F \in \mathcal{C}} F$.

Theorem 1. *ISP and SCP are equivalent.*

Proof. I show the equivalence in two steps. First, I show that ISP is a subproblem of SCP, i.e. for each instance of ISP a corresponding instance of SCP can be constructed (and the solution of the SCP-instances directly gives the solution of the ISP-instance). In the second step, I show that SCP is a subproblem of ISP. The both together imply equivalence.

For each ISP-instance one can construct a corresponding SCP-instance: $X := V_{G_c} \setminus V_{G_c}^0$ and $\mathcal{F} := \{C(v) | v \in V_{G_c}\}$ We say that vertex v is the *seed* of the set $C(v)$. The solution of SCP is a set $F \subseteq \mathcal{F}$. The set of seeds of the subsets in F constitute the solution of ISP:

$$S = \{v | C(v) \in F\} \tag{4.6}$$

While constructing an ISP-instance for an SCP-instance, we have to be careful, because the number of subsets in SCP can be greater than the size of X. Therefore, in the coverage graph G_c to be constructed, there are two types of vertices. Each first-type-vertex v_x corresponds to one element $x \in X$, and each second-type-vertex v_F correspond to a subset $F \in \mathcal{F}$. Edges go only from first-type-vertices to second-type-vertices, thus only first-type-vertices are coverable. There is an edge (v_x, v_F) from a first-type-vertex v_x to a second-type-vertex v_F if and only if the corresponding element $x \in X$ is included in the corresponding subset F, i.e. $x \in F$. When the ISP is solved, all the coverable vertices (first-type-vertices) are covered by a minimal set of vertices S. In this case, S obviously consists only of second-type-vertices. The solution of the SCP are the subsets corresponding to the vertices included in S:

$$\mathcal{C} = \{F | F \in \mathcal{F} \wedge v_F \in S\} \tag{4.7}$$

4.2.2 1-NN coverage graphs

In this section, I introduce the notion of 1-nearest neighbor (1-NN) coverage graphs which is motivated by the good performance of the 1-NN classifier for time-series classification. I show the optimality of INSIGHT for the case of 1-NN coverage graphs and how the NP-completeness of the general case is alleviated for this special case.

I begin with defining the specialization of the coverage graph based on the 1-NN relation:

Definition 10 (1-NN coverage graph)**.** *A 1-NN coverage graph, denoted by G_{1NN} is a coverage graph where $(v_x, v_{x'}) \in E_{G_{1NN}}$ if and only if time series x' is the first nearest neighbor of time series x and the class labels of x and x' are equal.*

This definition states that an edge points from each vertex v to the nearest neighbor of v if and only if this is a good nearest neighbor (i.e., their labels match). Thus, vertexes are *not* connected with their bad nearest neighbors.

From the practical point of view, in order to account for the fact that the size of selected instances is defined *a priori* (e.g., it is a user-defined parameter), a slightly different version of the Instance Selection Problem (ISP) is the following:

m-limited Instance Selection Problem (m-ISP) — If one wishes to select exactly m labeled time series from the training set, then, instead of selecting the minimal amount of time series that ensure total coverage, one can select those m time series that maximize the coverage. I call this variant *m-limited Instance Selection Problem* (m-ISP). The following proposition shows the relation between 1-NN coverage graphs and m-ISP:

Proposition 1. *In 1-NN coverage graphs, selecting those m vertices v_1, ..., v_m that have the largest covered sets $C(v_1)$, ..., $C(v_m)$ leads to the optimal solution of m-ISP.*

The validity of this proposition stems from the fact that, in 1-NN coverage graphs, the out-degree of all vertices is 1. This implies that each vertex is covered by *at most* one other vertex, i.e., the covered sets $C(v)$ are mutually disjoint for each $v \in V_{G_{1NN}}$.

Proposition 1 describes the optimality of INSIGHT, when the good 1-occurrence score (Equation 4.1) is used, since the size of the set $C(v_i)$ is the number of vertices having v_i as their first good nearest neighbor.

4.2.3 Coverage Graph for Relative Scores

It has to be noted that the described framework of coverage graphs can be extended to other scores too, such as the relative score of Equations 4.3. In such cases, one can additionally model bad neighbors and introduce weights on the edges of the graph.
Consider scores of the form

$$g_Y(x) = a\ g_G^1(x) - b\ g_B^1(x) \tag{4.8}$$

where a and b are user-defined real-number coefficients, e.g. for the score of Equation 4.3, $a = 1$ and $b = -2$. I extend the definition of the 1-NN coverage graph below:

Definition 11 (Weighted 1-NN coverage graph). *In a weighted 1-NN coverage graph, denoted by G^w_{1NN}, there is an edge $(v_x, v_{x'}) \in E_{G^w_{1NN}}$ if and only if time series x' is the first nearest neighbor of time series x. The weight of the edge $w(v_x, v_{x'})$ is*

$$w(v_x, v_{x'}) = \begin{cases} a & \textit{if the class labels of } x \textit{ and } x' \textit{ are equal} \\ b & \textit{otherwise} \end{cases} \tag{4.9}$$

For a weighted coverage graph I introduce the *coverage score* of a vertex v as the sum of weights of incoming edges to v:

$$C_{Score}(v) = \sum_{\forall v' \in C(v)} w(v', v) \tag{4.10}$$

In this framework, the selection of the highest-scored instances according to Equation 4.3 corresponds to selecting instances that have highest coverage score in case of $a = 1$ and $b = -2$.

4.3 Experiments

I experimentally examine the performance of INSIGHT with respect to effectiveness (i.e. classification accuracy), and efficiency (i.e. execution time required for instance selection). As baseline, FastAWARD [194] is used.[4]

I used 37 publicly available time series datasets from the collection described in Section 2.7.[5] I performed 10-fold-cross validation. INSIGHT uses $g_G(x)$ (Equation 4.1) as the default score function, however $g_R(x)$ (Equation 4.2) and $g_{Xi}(x)$

4 Note that I used my own implementation of FastAWARD because I could not find any other available implementation. Also note that the experimental protocol differs from the one used in [194], therefore, the results are not directly comparable. However, using my implementation of FastAWARD, I observed very similar behavior of FastAWARD compared to what was reported in [194].

5 For StarLightCurves the calculations took unacceptably long for FastAWARD, therefore this dataset is omitted.

Table 4.1: Accuracy ± standard deviation for INSIGHT and FastAWARD (underlined: winner)*

Dataset	FastAWARD	INS-$g_G(x)$	Dataset	FastAWARD	INS-$g_G(x)$
50words	0.526±0.041	0.642±0.046	Lighting7	0.447±0.126	0.510±0.082
Adiac	0.348±0.058	0.469±0.049	MALLAT	0.551±0.098	0.969±0.013
Beef	0.350±0.174	0.333±0.105	MedicalImg	0.642±0.033	0.693±0.049
Car	0.450±0.119	0.608±0.145	Motes	0.867±0.042	0.908±0.027
CBF	0.972±0.034	0.998±0.006	OliveOil	0.633±0.100	0.717±0.130
Chlorine	0.537±0.023	0.734±0.030	OSULeaf	0.419±0.053	0.538±0.057
CinC	0.406±0.089	0.966±0.014	Plane	0.876±0.155	0.981±0.032
Coffee	0.560±0.309	0.603±0.213	Sony	0.924±0.032	0.976±0.017
Diatom	0.972±0.026	0.966±0.058	SonyII	0.919±0.015	0.912±0.033
ECG200	0.755±0.113	0.835±0.090	SwedishLeaf	0.683±0.046	0.756±0.048
ECGFive	0.937±0.027	0.945±0.020	Symbols	0.957±0.018	0.966±0.016
FaceFour	0.714±0.141	0.894±0.128	Syntheticl	0.923±0.068	0.978±0.026
FacesUCR	0.892±0.019	0.934±0.021	Trace	0.780±0.117	0.895±0.072
FISH	0.591±0.082	0.666±0.085	TwoPatterns	0.407±0.027	0.987±0.007
GunPoint	0.800±0.124	0.935±0.059	TwoLead	0.978±0.013	0.989±0.012
Haptics	0.303±0.068	0.435±0.060	Wafer	0.921±0.012	0.991±0.002
Inline	0.197±0.056	0.434±0.077	WordsSyn	0.544±0.058	0.637±0.066
Italy	0.960±0.020	0.957±0.028	Yoga	0.550±0.017	0.877±0.021
Lighting2	0.694±0.134	0.670±0.096			

* The names of some datasets are abbreviated, for the full names see Table 2.2.

(Equation 4.3) are also being examined. The resulting combinations are denoted as INS-$g_G(x)$, INS-$g_R(x)$ and INS-$g_{Xi}(x)$, respectively.

The distance function for the 1-NN classifier is DTW (see Section 2.3.2). In contrast to FastAWARD, which determines the optimal warping window size w_{opt}^{DTW}, INSIGHT sets the warping-window size to a constant of 5%. (This selection is justified by the results presented in [143], [145], which show that relatively small warping window sizes lead to higher accuracy, see also Section 2.3.2.) In order to speed up the calculations, I used the LB_Keogh lower bounding technique [100] both for INSIGHT and for FastAWARD.

4.3.1 Results on Effectiveness

I first compare INSIGHT and FastAWARD in terms of classification accuracy that results when using the instances selected by these two methods. Table 4.1 presents the average accuracy and corresponding standard deviation for each data set, for the case when the number of selected instances is equal to 10% of the size of the training set (for INSIGHT, the INS-$g_G(x)$ variation is used). In the vast

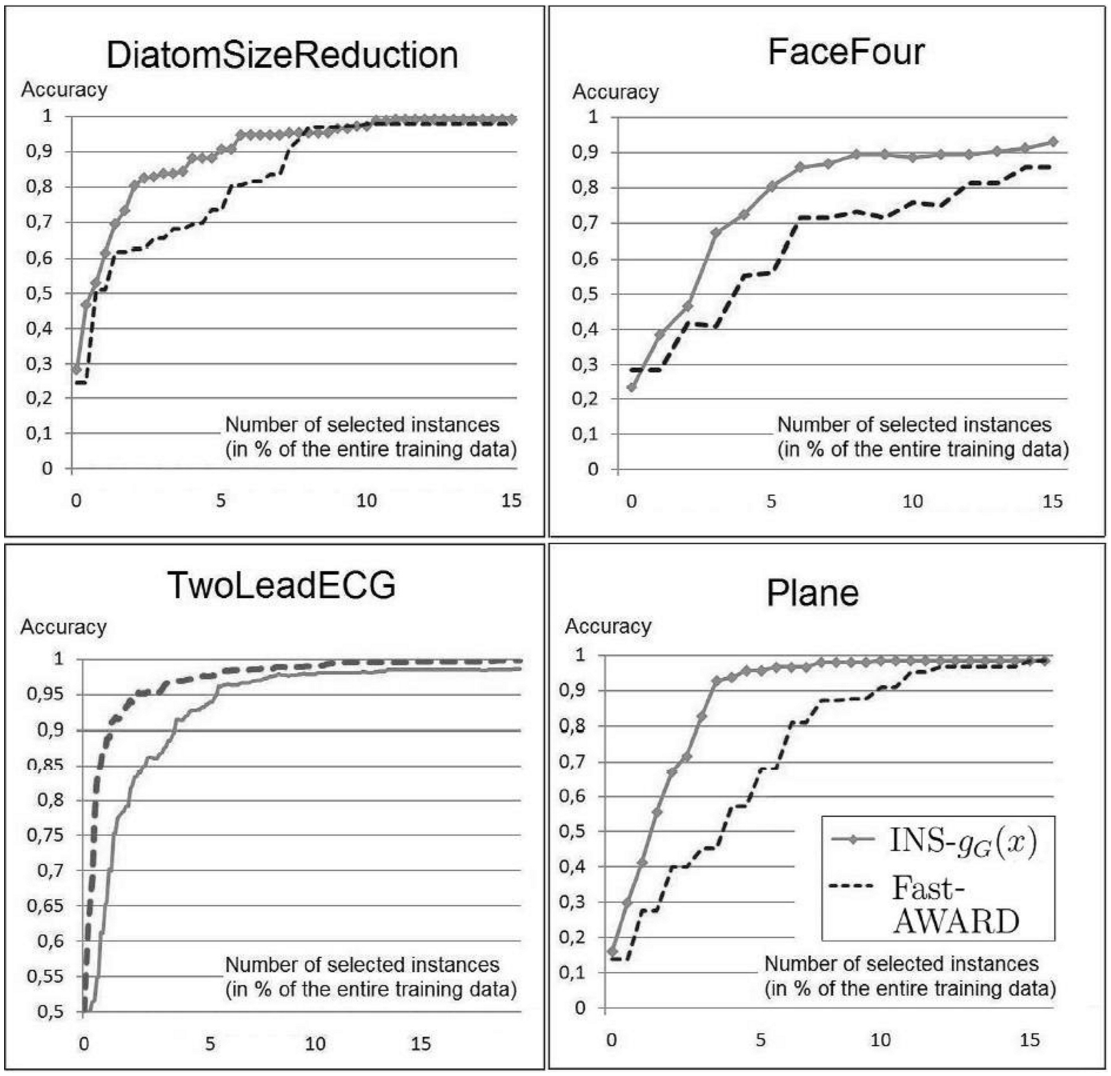


Figure 4.1: Accuracy as function of the number of selected instances (in % of the entire training data) for some datasets for FastAWARD and INSIGHT.

majority of cases, INSIGHT substantially outperforms FastAWARD. In the few remaining cases, their difference are remarkably small (in the order of the second or third decimal digit, which are not significant in terms of corresponding standard deviations).

I also compared INSIGHT and FastAWARD in terms of the resulting classification accuracy for varying number of selected instances. Figure 4.1 illustrates that INSIGHT compares favorably to FastAWARD. Analogous conclusions can be drawn for all other datasets for which INSIGHT outperforms FastAWARD.

Besides the comparison between INSIGHT and FastAWARD, what is also interesting is to examine their performance relative to using the entire training data (i.e., no instance selection is applied). Indicatively, for 17 datasets from Table 4.1

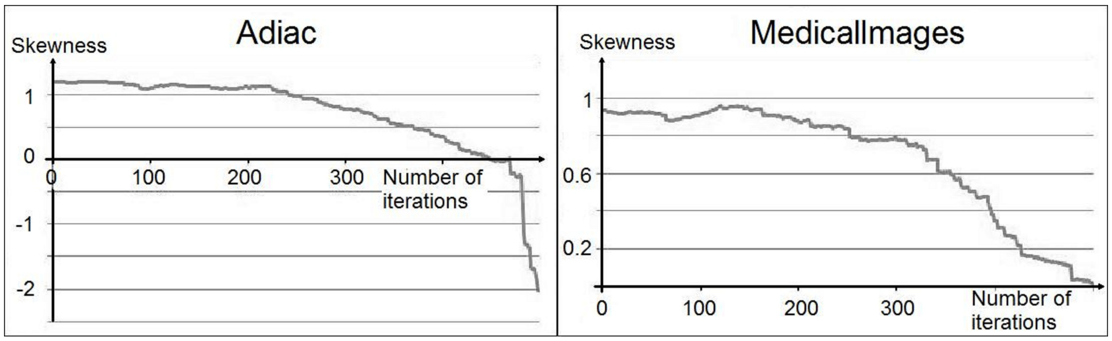


Figure 4.2: Skewness of the distribution of $g_G^1(x)$ as function of the number of iterations performed in FastAWARD. The skewness is shown on the vertical axis, while the number of performed iterations is shown on the horizontal axis. On the trend, the skewness decreases from iteration to iteration.

the accuracy resulting from INSIGHT[6] is worse by less than 0.05, compared to using the entire training data. For FastAWARD[7] this number is 4, which clearly shows that INSIGHT selects more representative instances of the training set than FastAWARD.

Next, I investigate the reasons for the presented difference between INSIGHT and FastAWARD. In Section 2.4.3, the skewness of good k-occurrence, $g_G^k(x)$, was identified as a crucial property for instance selection to work properly, since skewness renders good hubs to become representative instances. In the examination, I found that using the iterative procedure applied by FastAWARD, this skewness has a decreasing trend from iteration to iteration. Figure 4.2 exemplifies this by illustrating the skewness of $g_G^1(x)$ for two data sets as a function of iterations performed in FastAWARD. (In order to quantitatively measure skewness, the standardized third moment is used, see Equation 2.35.) The reduction in the skewness of $g_G^1(x)$ means that FastAWARD is not able to identify in the end representative instances, since there are no pronounced good hubs remaining. Note that FastAWARD iteratively drops *bad* instances, the instances *remaining at the end* are considered as the selected instances that are used for classification, therefore, the reduction of skewness is crucial.

In order to further understand that the reduced effectiveness of FastAWARD stems from its iterative procedure and not from its score function, $g_{Xi}(x)$ (Equation 4.3), I compare the accuracy of all variations of INSIGHT including INS-$g_{Xi}(x)$. Table 4.2 reports for how many datasets INS-$g_G(x)$, INS-$g_R(x)$ and INS-$g_{Xi}(x)$ win and loose compared to FastAWARD. Remarkably, INS-$g_{Xi}(x)$ clearly outperforms FastAWARD for the majority of cases, which verifies the previous statement. Moreover, the differences between the three variations are not large, indicating the robustness of INSIGHT with respect to the scoring function.

6 selection of 10 % of all the training instances with INS-$g_G(x)$

7 selection of 10 % of all the training instances

Table 4.2: Number of datasets where different versions of INSIGHT win/lose against FastAWARD

	INS-$g_G(x)$	INS-$g_R(x)$	INS-$g_{Xi}(x)$
Wins	32	33	33
Loses	5	4	4

Table 4.3: Execution times (in seconds, averaged over 10 folds) of instance selection using INSIGHT and FastAWARD

Dataset	FastAWARD	INS-$g_G(x)$	Dataset	FastAWARD	INS-$g_G(x)$
50words	94 464	203	Lighting7	5 511	8
Adiac	32 935	75	Mallat	4 562 881	19 041
Beef	1 273	3	MedicalImages	13 495	55
Car	11 420	18	Motes	17 937	55
CBF	37 370	67	OliveOil	3 233	5
Chlorine	16 920	1 974	OSULeaf	80 316	118
CinC	3 604 930	16 196	Plane	1 527	4
Coffee	499	1	Sony	4 608	11
Diatom	18 236	44	SonyII	10 349	23
ECG200	634	2	SwedishLeaf	37 323	89
ECGFiveDays	20 455	60	Symbols	165 875	514
FaceFour	4 029	6	SyntheticControl	3 017	8
FacesUCR	150 764	403	Trace	3 606	11
FISH	59 305	93	TwoPatterns	360 719	1 693
GunPoint	1 107	4	TwoLeadECG	12 946	45
Haptics	152 617	869	Wafer	923 915	4 485
InlineSkate	906 472	4 574	WordsSyn	101 643	203
Italy	1 855	6	Yoga	1 774 772	6 114
Lighting2	15 593	23			

* The names of some datasets are abbreviated, for the full names see Table 2.2.

4.3.2 Results on Efficiency

The computational complexity of INSIGHT depends on the calculation of the scores of the instances of the training set and on the selection of the top-ranked instances. Thus, for the examined score functions, the computational complexity is $\mathcal{O}(n^2)$, n being the number of training instances. The reason for this complexity is the calculation of the distance between each pair of training instances. For FastAWARD, its first step (leave-one-out nearest neighbor classification of the train instances) already requires $\mathcal{O}(n^2)$ execution time. However, FastAWARD performs additional computationally expensive steps, such as determining the best warping-window size and the iterative procedure for excluding instances. For this reason, INSIGHT is expected to require reduced execution time compared to FastAWARD. This is verified by the results presented in Table 4.3, which shows the execution time needed to perform instance selection with INSIGHT and FastAWARD. As expected, INSIGHT outperforms FastAWARD drastically.

Regarding the time for classifying new instances, please notice that both methods perform 1-NN using the same number of selected instances, therefore the classification times are equal.

Chapter 5

Fusion of Distance Measures

The success of DTW (see also Section 2.3.2) suggests that, in time-series classification, what really matters is the distance measure, i.e. when and why two time series are considered to be similar. By allowing for shiftings and elongations, DTW captures the global similarity of the shape of two time series very well. In general, however, many other characteristic properties might be crucial in a particular application, such as similar global or local behavior in the frequency domain that can be captured by the Fourier or Cosine-spectrum or the Wavelet Transform of the signal (see Section 2.3).

In this chapter, I examine this phenomenon in more detail. I consider time series distance measure that were described in Section 2.3 and discuss what kind of similarity (and dissimilarity respectively) they capture. As major contribution, I propose a framework that allows for the fusion of these different distance measures in a principled way. Within this framework, I develop a hybrid distance measure. I evaluate these findings in context of time-series classification on the large, publicly available collection of real-world datasets described in Section 2.7 and I show that the method achieves substantial, statistically significant, improvements in terms of classification accuracy.

5.1 Aspects of Similarity Captured by Various Time Series Distance Measures

The most important time series distance measures, such as Euclidean Distance over various representations, Dynamic Time Warping (DTW), Edit Distance on Real Sequences (EDR), Edit Distance with Real Penalty (ERP), Distance based on Longest Common Subsequences (LCSS) and DISSIM were reviewed in Section 2.3. This section gives a brief overview about which aspects of similarity are captured by each of these distance measures.

The Euclidean Distance over two (raw) time series x_1 and x_2, denoted as $d_{EU}(x_1, x_2)$ captures the *global similarity of two time series' shapes*. It does *not* *allow for elongations.* This is beneficial if any particular event (and the corresponding pattern) begins regularly at the (approximately) same position within the time series and the variance of the durations of the events is small.

In contrast, the intuition behind *Dynamic Time Warping* (DTW) is that we can not expect an event to happen (or a characteristic pattern to appear respectively) at *exactly* the same time position and its duration can also (slightly) vary. Therefore, DTW captures *global similarity of two time series' shapes* in a way that it allows for *shifting and elongations.*

The Euclidean Distance over the DFT-coefficients of two signals x_1 and x_2, denoted as $d^F_{EU}(x_1, x_2)$ in Section 2.3, captures the similarity in the *signal's periodic behavior.* This distance measure is beneficial, if different periodic behavior characterize the time series classes of the underlying application.

While DFT captures *global periodic* behavior, as described in Section 2.2.2, *wavelets* reflect both *local* and *global* character of a time series. Therefore, one can calculate the Euclidean Distance of the Wavelet-transform of two time series x_1 and x_2, denoted as $d^W_{EU}(x_1, x_2)$ (see also Section 2.3).

In order to be able to capture further aspects of similarity, I also use (a) DISSIM that is able to compute the similarity of two time series with different sampling rates as the integral between the both time series, (b) distance based on longest common subsequences (LCSS), (c) edit distance on real sequences (EDR), and (d) edit distance with real penalty (EPR).

5.2 Fusion of Distance Measures

In the recent work of Ding et al. [56], none of the examined distance measures could outperform DTW in general. However, in some specific tasks, one or the other distance measure worked better than DTW, which is likely to be explained by the fact that different aspects of similarity are relevant in different domains. In case of simple tasks, one of the distance measures may capture the relevant aspects of similarity entirely. This best distance measure can be found based on domain knowledge or by measuring e.g. the leave-one-out classification error on the train data for the candidate distance measures. In more complex cases, however, a single distance measure may not be sufficient alone. Thus, in order to achieve the desired accuracy, one needs to combine several distance measures. Such hybridization is often achieved in an *ad hoc* manner. In contrast, I develop a fusion schema for time series distance measures that allows to combine distance measures in a principled way.

In order to distinguish between the distance measures that we want to combine and the resulting distance measure, I refer to the former ones as *elementary distance measures* whereas to the later one as *fused distance measure.* My approach for fusion of distance measures[1] (see Figure 5.1) consists of the following steps:

1 Note that the elementary distance measures are not assumed to fulfill specific properties (such as triangular inequality).

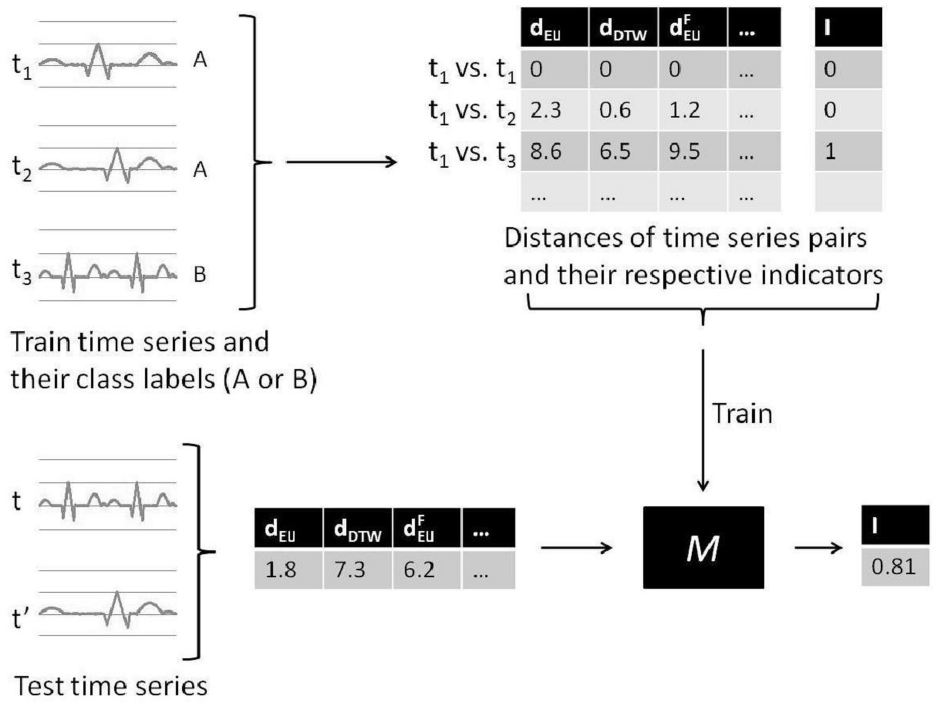

t1 A
t2 A
t3 B
Train time series and their class labels (A or B)
d_EU d_DTW d^F_EU ... I
t1 vs. t1 0 0 0 ... 0
t1 vs. t2 2.3 0.6 1.2 ... 0
t1 vs. t3 8.6 6.5 9.5 ... 1
Distances of time series pairs and their respective indicators
Train
t
t'
Test time series
d_EU d_DTW d^F_EU ...
1.8 7.3 6.2 ...
M
I
0.81

Figure 5.1: Example: fusion of time series distance measures. A regression model M is trained and its output is used as distance measure.

1. For all the *pairs* of time series in the train data, the distances are calculated using *all* the considered elementary distance measures.
2. In some of the above pairs, both time series belong to the same class, in others they belong to different classes. The indicator $\mathcal{I}(x_1, x_2)$ of a pair of time series (x_1, x_2) is defined as follows:

$$\mathcal{I}(x_1, x_2) = \begin{cases} 0 & \text{if } x_1 \text{ and } x_2 \text{ belong to the same class} \\ 1 & \text{otherwise} \end{cases} \tag{5.1}$$

3. A regression model $\mathcal{M}$ is trained: the distance values calculated in the first step are used as training data along with the corresponding indicators as labels.[2]
4. I propose to use the output of $\mathcal{M}$ as the fused distance measure. For a pair of time series (x, x'), where either or both of them can be unlabeled (test) time series, one can calculate the distance values using all the considered elementary distance measures. Then $\mathcal{M}$ is used to estimate (based on these distance values) the likelihood that x and x' belong to different classes. Finally, this estimation is used as the distance of x and x'.

2 I decided to use a regression model because it outputs continuous values which can be used as distances in a natural way.

Note that this approach is generic, as this framework allows the fusion of arbitrary distance measures using various regression models as $\mathcal{M}$. Furthermore, this fused distance measure can be used by various classification algorithms.

Also note that the above description is just the conceptual description of the approach. While implementing its first step (i.e. calculation of elementary distances), one would not separately calculate the distance of the pairs (x_1, x_2) and (x_2, x_1) if the current elementary distance measure is symmetric. Furthermore, one can pre-calculate and store the fused distances of pairs if the classification algorithm (which uses the fused distance measure) queries the distance of the same pair several times.

While fusing elementary distance measures according to the above description, all the pairs of time series are considered. Therefore, if the training data contains n time series, the elementary distances are required to be calculated $\mathcal{O}(n^2)$ times and the data used to train $\mathcal{M}$ contains $\mathcal{O}(n^2)$ records. In case of small data sets, this is not a problem. For large datasets, I propose to sample the pairs, and calculate the elementary distances only for the sample. In this case, a sufficiently large sample is used for training $\mathcal{M}$.

As mentioned before, in simple domains, one single distance measure might be sufficient to capture all the relevant aspects of similarity. In such cases, fusion of distance measures is not necessary and could introduce noise. In order to avoid it, I propose to select the best distance measure out of some *fused* distance measures (that use different regression models as $\mathcal{M}$) *and* all the *elementary* distance measures. In order to allow for this selection, one can judge the quality of each distance measure by its leave-one-out nearest neighbor classification error on the training data.

5.3 Experiments

Datasets. I examined 35 out of all the 38 datasets described in Section 2.7. I excluded 3 of them (Coffee, Beef, OliveOil) due to their small sizes (less than 100 time series).

Considered distance measures. I used all the elementary distance measures described in Section 5.1. I used two versions of DTW with warping window sizes constrained at 5% and 10% around the matrix diagonal (see Section 2.3.2 and [143], [144]).

Comparison protocol. As discussed in Section 2.4, in the time series domain, the nearest neighbor (NN) algorithm has been shown to be competitive and often even superior to many state-of-the-art classification algorithms. Therefore, I compared time series distance measures in context of 1-NN classification. I measured classification error as the misclassification ratio. I performed 10-fold cross validation. For each dataset, I tested whether or not the differences between the performance of my approach and its competitors are statistically significant. For this, I used t-test at significance level of 0.05.

Baselines. I used two state-of-the art time series classifiers as baselines: The first one is the 1-NN using DTW with warping window size constrained at 5%. This is denoted as DTW.

As the second baseline, I choose stacking with SVMs. For this second baseline, first, I performed leave-one-out classification of the training data with several 1-NN classifiers, each of them used a different elementary distance measure out of the examined ones. The outputs of these classifiers together with the true class labels served as training data for the SVM. At this step, each training instance corresponds to a training time series x: a training instance consists of the outputs of the different nearest neighbor classifiers for x and the true class label of x. After training the SVM, each test time series x' was classified with all the 1-NN classifiers. (As before, each of these classifiers used a different elementary distance measures out of the examined ones.) Then, these classification results were further processed by the SVM: at this step, a test instance of the SVM consists of the outputs of the classifiers for the test time series x', and the SVM was used to predict the class label of x'.

For the second baseline, I used the SVM-implementation from the WEKA machine learning library[3], which utilizes the one-against-one protocol for multi-class problems. I used a polynomial kernel, and determined proper values for the hyperparameters of the SVM (complexity constant C and the exponent of the polynomial kernel) using a hold-out subset of the training data. This second baseline is referred to as Stacking or short Stack in the proceeding tables.

Fusion of Distance Measures. I produced two fused distance measures[4]: for regression models $\mathcal{M}_1$ and $\mathcal{M}_2$, I used (i) linear regression and (ii) multilayer perceptron[5], respectively, from the Weka library. After training $\mathcal{M}_1$ and $\mathcal{M}_2$, the best-performing distance measure is selected out of the *fused* and *elementary* distance measures based on the leave-one-out nearest neighbor classification error on the train data. Finally, the selected distance measure is used in the 1-NN classifier when classifying test time series. This approach is denoted as FUSION.

Results. The results are shown in Table 5.2 and summarized in Table 5.1.

3 http://www.cs.waikato.ac.nz/ml/weka/

4 In order to save computational time, according to Section 5.2, for some large datasets I randomly sampled the pairs: I calculated similarities in case of FacesUCR and Mallat for 20 %, ChlorineConcentration and Yoga for 10%, TwoPatterns for 5%, Wafer for 1 %, and StarLightCurves for 0.5 % of all the pairs. In order to ensure fair comparison, I used the same sample of pairs both in my approach and for the baselines.

5 I used Weka's standard parameter-settings, i.e. learning rate: 0.3, momentum: 0.2, number of train epochs: 500. This model was used in my approach, FUSION. Tuning these parameters could further improve FUSION's performance.

Discussion. For many of the examined datasets, the classification task is simple: DTW's error rates are less than 10 %. In such cases, all methods worked well, and thus I could rarely observe statistically significant differences. In the last column of Table 5.2, I provide two symbols in form of $\pm/\pm$ where + denotes statistical significance and $-$ its absence against DTW and Stacking respectively. While FUSION significantly outperformed the baselines in 11 and 12 cases respectively, the baselines did not outperform FUSION significantly.

Note that in most of the cases we are not concerned with binary classification problems, however, the number of classes is more than two. In fact, this is one of the reasons why these datasets are challenging and this explains the relatively high error rates.

Table 5.1: Number of FUSION's wins/loses and ties against DTW and Stacking

	against	**DTW**	**against**	**Stacking**
	total	significant	total	significant
Wins	27	11	25	12
Ties	5	-	3	-
Loses	3	0	7	0

Table 5.2: Classification errors*

Dataset	DTW	Stack	FUSION	Dataset	DTW	Stack	FUSION
50words	0.194	0.801	0.187-/+	Mallat	0.027	0.026	0.021-/-
Adiac	0.346	0.603	0.316+/+	MedicalImg	0.191	0.399	0.190-/+
Car	0.250	0.250	0.166-/+	Motes	0.048	0.016	0.020+/-
CBF	0.000	0.001	0.000-/-	OSULeaf	0.272	0.217	0.136+/+
Chlorine	0.385	0.393	0.362+/+	Plane	0.000	0.000	0.000-/-
CinC	0.000	0.001	0.000-/-	Sony	0.019	0.006	0.014-/-
Diatom	0.003	0.006	0.003-/-	SonyII	0.017	0.011	0.007-/-
ECG200	0.125	0.090	0.070+/-	StarLight	0.170	0.111	0.115+/-
ECGFiveDays	0.008	0.002	0.005+/-	SwedishLeaf	0.155	0.276	0.101+/+
FaceFour	0.044	0.047	0.036-/-	Symbols	0.018	0.013	0.016-/-
FacesUCR	0.050	0.075	0.029+/+	Synthetic	0.005	0.007	0.003-/-
Fish	0.194	0.214	0.120+/+	Trace	0.000	0.000	0.000-/-
GunPoint	0.020	0.025	0.030-/-	TwoL.ECG	0.001	0.002	0.003-/-
Haptics	0.531	0.650	0.525-/+	TwoPatterns	0.057	0.001	0.000-/-
InlineSkate	0.443	0.463	0.434-/-	Wafer	0.030	0.027	0.027-/-
Italy	0.049	0.045	0.036+/-	WordsSyn	0.191	0.515	0.178-/+
Lighting2	0.083	0.147	0.115-/-	Yoga	0.145	0.127	0.131-/-
Lighting7	0.201	0.453	0.243-/+				

*The names of some datasets are abbreviated, for the full names see Table 2.2.
+ and - denote statistical significance and its absence against DTW/Stacking

Chapter 6

The GRAMOFON Ensemble Framework

As described in Section 2.5, the same recognition task can be solved by various models. Based on the outputs of these models, a meta-level method determines the class label of the instance (time series) to be classified.

In general, in order to achieve predictions of the required quality, the number of models used in such ensembles (also called committee of experts) may have to be large (several hundreds). Many models often deliver different predictions: due to the variety of models (such as SVMs [132], neural networks [83], decision trees [136], Bayesian models [151]) and the differences in the underlying principles and techniques, one expects diverse error characteristics for the distinct models. Ensembles assume that different models can compensate each other's errors and thus their right combination outperforms each individual model [55].

Next, I illustrate the aforementioned statement by a simple observation from the movie recommendation domain.[1] In the movie recommendation problem, users can describe their preferences in form of ratings: if a user likes a movie, she can rate it high, otherwise she can give a low rating for that movie. As the total number of movies is usually very high, most of the users only rate a small portion of all the movies (e.g. the ones they have previously watched). Based on the known ratings, the recognition models aim at capturing the taste of the users in order to be able to recommend new movies to the users. Therefore, for each user-movie pair, each model predicts a score that estimates how the user would rate that particular movie (see the tables p_1 and p_2 in the top left of Figure 6.1 on page 90 for an example of such predictions). One can average the predictions of several models for each user-movie pair (see $p\tilde{}_{1,2}$ in the top right of Figure 6.1 for an example for the average of p_1 and p_2). One can observe that the average of *all* the available models may outperform the *best* individual model. This is illustrated in Table 6.1, which presents results of simple ensembles of the

1 Although the following example focuses on movie recommendation, similar phenomena can be observed in many other domains too.

Table 6.1: Performance (RMSE[a]) improvement with respect to the best individual model using simple ensemble schemes on the AusDM-S dataset. (10 fold cross validation, averaged results, in each fold the best/worst model(s) were selected based on the performances on the train subset.)

Method	RMSE-improvement[c]
Average of *all* models' predictions[b]	2.40
Average of the *best* 10 models' predictions[b]	8.72
Average of the *worst* 10 models' predictions[b]	−20.84

[a] RMSE stands for Root Mean Squared Error, see Section 6.1 for its definition
[b] For each user-movie pair
[c] In contrast to Figure 6.1 on page 90, the ratings in the AusDM-S dataset are on the range between 1000 and 5000.

200 models contained in the AusDM-S dataset.[2] Furthermore, Table 6.1 shows that combining all the models may not be the best choice, because many of the models may share similar error characteristics, that can substantially depress the compensation effect: in particular, the average of the 10 individually best models' predictions outperforms the aforementioned average of the predictions of all the available models. In contrast, if one selects the 10 individually worst models, the average of their predictions perform much worse than the best model.

I argue that different models have a high potential to compensate each other's errors, but the right selection of the models is important, otherwise the aforementioned compensation effect may be depressed. How much the compensation effect is depressed, also depends on how robust the applied ensemble schema is against overfitting. For well-regularized ensemble methods, like stacking with linear regression or SVMs, the depression of compensation is typically much lower, but the phenomenon in general is similar: for example, training a multivariate linear regression as meta-model on *all predictions* of AusDM-S is likewise worse than training it only on the predictions of the *individually best 10 models* (RMSE-improvement: 8.58 vs. 9.42). However, the selection of the 10 individually best models may be far from perfect: the potential power of an ensemble may be much higher than the quality that is reached by combining the 10 individually best models. Thus, even in case of well-regularized models, the depression of compensation is an acute problem.

Therefore I aim at selecting the best, or at least a good enough subset of models for an ensemble. As the number of subsets is exponential in the number of models, exhaustive search is intractable. Therefore, conventional approaches perform systematic non-exhaustive search on the power set. My work is in line with this idea and, as an innovation, I let the pairwise error compensation power to drive this search. Instead of pairs, theoretically, one could use triplets, quadruples, etc.

2 The dataset is described in Section 6.4.1.

I decided for pairs because this is the simplest level where error compensation can be observed. Even though the number of triplets, quadruples, etc. is polynomial (just like in case of pairs), but in large, real-world sized applications, the degree of the polynomial often plays an important role. More importantly, pairs enable simple graph-based modeling and analysis of the problem[3], which has the benefit that one can import concepts and techniques from graph theory, like the minimal spanning tree, and such concepts can be applied to our problem.

As motivated before, in this chapter, I aim at exploiting the error compensation power by carefully selecting models for an ensemble. As major contribution, I propose GRAMOFON, the General Model-selection Framework based on Networks. GRAMOFON allows to describe a wide range of model selection strategies for ensembles varying from meta-filter to meta-wrapper methods.[4] Using the framework, I propose four ensemble methods: *Basic*, *BasicFast*, *RegOptMST* and *NetworkMST*. While the first performs a search that is exhaustive with respect to the *pairs* of models, in the latter three of these methods, I applied a principled scaling technique based on minimal spanning trees that guarantees to examine the contribution of each model. These strategies are evaluated in experiments on publicly available real-world data and they are compared to well-known ensemble methods such as boosting [159] and stacking [189]. The results indicate that my methods outperform the state-of-the-art.

This chapter is organized as follows. The context, in which GRAMOFON is developed as well as the notations used in the pseudocodes of this chapter are explained in Section 6.1. In section 6.2, I introduce GRAMOFON. Next, I deploy my four ensemble techniques in this framework (section 6.3). Finally, I present the results of my experiments.

6.1 Context and Notations

Previous chapters focused on classification and therefore the labels were discrete and nominal. In contrast, in the description of the GRAMOFON framework, it is assumed that the labels are *numeric* and *continuous*, such as preference for a given product, (expected) temperature at a given location, or (expected) time required for the recovery of a patient. In this context, the recognition model performs *regression*, i.e., predicts continuous labels instead of discrete ones, and therefore the quality of the model can be measured as the difference between the true and predicted label. I describe the conversion of classification problems into this setting in Section 6.5.

Compared to other chapters of this book, a further unique property of the settings used here is that the approach works exclusively at the meta-level: in

3 Triples, quadruples, etc. could be represented as hyper-edges too, however this would lead to computationally more expensive methods.

4 Filter (wrapper) methods score models without (with) involving the meta-model [7].

this chapter it is irrelevant how the aforementioned recognition models work. Instead, this section focuses on the meta-layer, i.e., on methods that use the labels predicted by the recognition models in order to produce a new label that is closer to the true label than the labels predicted by the recognition models. In order to avoid ambiguity, whenever necessary, the aforementioned recognition models are referred to as *elementary models*, while the method used at the meta-level for combining the predictions of the elementary models is referred to as *meta model.*

In this chapter the term *elementary model* (or simply *model*) is used in order to distinguish from a classifier. Formally, a model m is defined as a function that predicts a numerical value $m(x) \in \mathbb{R}$ for an instance x of a dataset $\mathcal{D}$:

$$m : \mathcal{D} \to \mathbb{R}, \quad x \mapsto m(x) \tag{6.1}$$

In order to distinguish from the classification problem, instead of $c(x)$, $y(x)$ is used in this section to denote the true, real-valued label (also called *target*); $\hat{y}(x)$ is an estimation for $y(x)$. In a procedural description (in the pseudocodes), similarly to a method call of an object-oriented programming language, I write

$$p = m.\text{predict}(\mathcal{D}) \tag{6.2}$$

that denotes *the column vector containing the predictions*[5] $\hat{y}(x)$ of the model m for all $x \in \mathcal{D}$. Similarly, the *column vector containing the values of the labels* for all $x \in \mathcal{D}$ is denoted as

$$\mathcal{D}.\text{labels} \tag{6.3}$$

Given a set M of models, I use the notation $\{p_i\}$, to denote the matrix consisting of column vectors $p_i = m_i.\text{predict}(\mathcal{D})$ for all $m_i \in M$. When training a vector classifier, such as linear regression or SVM using such a matrix, each *row* of the matrix is considered to be a training instance.

For a set M of elementary models $m_1, ..., m_N$, a *meta model* m^* is a function that estimates the label of an instance $x \in \mathcal{D}$ based on the predictions of the elementary models $m \in M$. The label predicted by the meta model is denoted as $m^*(x)$, $m^*(x) \in \mathbb{R}$.

$$m^* : \mathcal{D} \to \mathbb{R}, \quad x \mapsto m^*(x) = m^*(\ m_1(x), ..., m_N(x)\ ) \tag{6.4}$$

In this chapter, ensembles of the above type are considered. I focus on selecting a *subset* of all the available elementary models so that the performance of the meta model is optimized.

5 Throughout this section, it is assumed that the instances of sets, such as $\mathcal{D}$, are enumerated always in the same, deterministic order.

Given a data sample $\mathcal{D}_s$, one can use various error functions for measuring the quality of an elementary model m or a meta model m^*. In this chapter, I used the *root mean squared error* (RMSE), which is defined as follows:

$$\text{RMSE} = \sqrt{\frac{\sum_{\forall x \in \mathcal{D}_s} \left(m^{(*)}(x) - y(x)\right)^2}{|\mathcal{D}_s|}} \tag{6.5}$$

In the pseudocodes I use error functions as a function of the predicted and true class labels in form of

$$\text{RMSE}(p^{(*)}, \mathcal{D}_s.\text{labels}), \tag{6.6}$$

where $p^{(*)} = m^{(*)}.\text{predict}(\mathcal{D}_s)$.

6.2 General Model-Selection Framework

In this section I describe the General Model-Selection Framework based on Networks. First I define model-networks that constitute the core of the approach. Afterwards, the details of my approach are presented in Section 6.2.2.

6.2.1 Model-Networks

Given the prediction models $m_1, \ldots, m_N$, my goal is to find their best combination. As mentioned before, the key of our ensemble technique is the selection of models that compensate for each other's errors. Therefore, in my approach (Algorithm 7), a network is built first, this network is called *model-network*, and it is denoted as G in Algorithm 7 (line 5). Each node of the model-network corresponds to one of the models $m_1, \ldots, m_N$. The network is a complete graph (all nodes are connected). Connections are undirected and weighted, the weight of $\{m_j, m_k\}$ aims at reflecting both (i) the mutual *error compensation* power and (ii) the *quality* of the models m_j and m_k.

Formally, a connection weight function can be defined as follows:

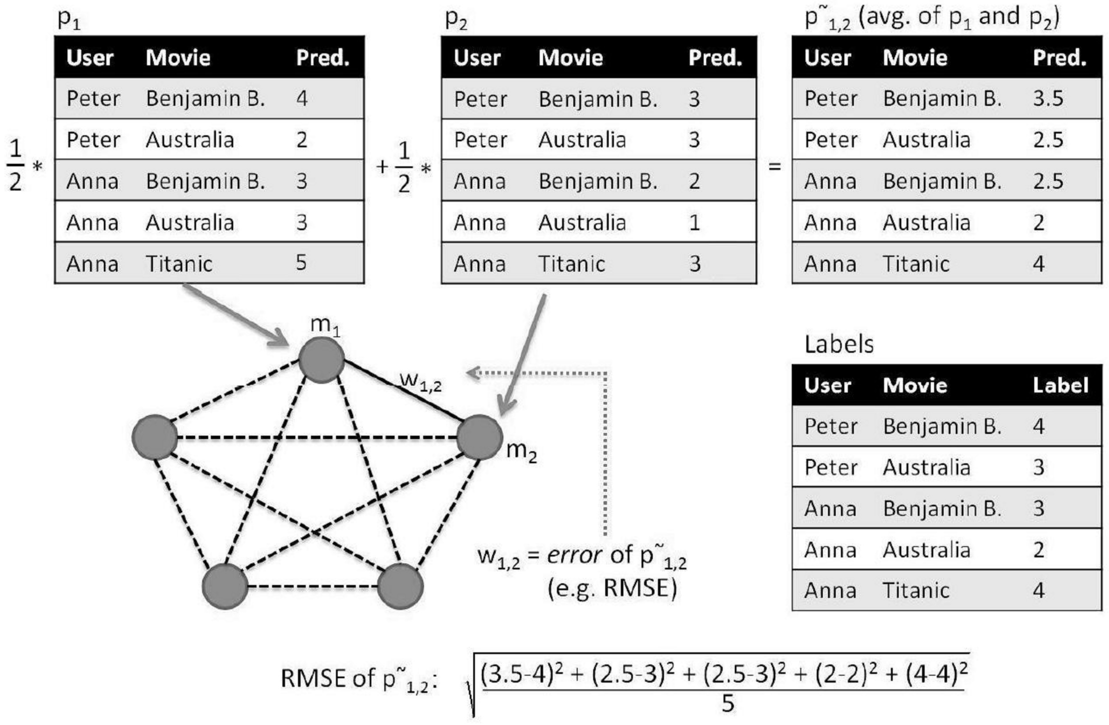
Definition 12 (Connection weight function)**.** *Given a network (or graph) G, a* connection weight function f_w *associates each edge e of G with a numerical value $w(e)$ that is called* weight *of e:*

$$f_w : E(G) \to \mathbb{R}, \quad e \mapsto w(e) \tag{6.7}$$

While calculating the weight of the edge between the nodes corresponding to the models m_j and m_k, Algorithm 6 averages for each data instance the *predictions* of both regression models m_j and m_k (lines 1...3). This results in a new prediction vector p. Then the error of p is returned (line 4), which is used as the weight of the connection $\{m_j, m_k\}$.

Algorithm 6 Connection Weight Function: connection_score

```
Require: Model $m_j$, Model $m_k$, Data sample $\mathcal{D}$, ErrorFunction $err$
Ensure: Connection weight of $\{m_j, m_k\}$
1: $p_1 = m_j$.predict($\mathcal{D}$)
2: $p_2 = m_k$.predict($\mathcal{D}$)
3: $\forall x \in \mathcal{D} : p[x] = (p_1[x] + p_2[x])/2$
       (Predictions are averaged for each instance $x$.)
4: return $err(p, \mathcal{D}$.labels)
```



p_1
$\frac{1}{2}*$
$+\frac{1}{2}*$
p_2
=
$p\tilde{}_{1,2}$ (avg. of p_1 and p_2)
m_1
m_2
$w_{1,2}$
$w_{1,2}$ = *error* of $p\tilde{}_{1,2}$ (e.g. RMSE)
Labels
RMSE of $p\tilde{}_{1,2}$: $\sqrt{\frac{(3.5-4)^2 + (2.5-3)^2 + (2.5-3)^2 + (2-2)^2 + (4-4)^2}{5}}$

p_1

User	Movie	Pred.
Peter	Benjamin B.	4
Peter	Australia	2
Anna	Benjamin B.	3
Anna	Australia	3
Anna	Titanic	5

p_2

User	Movie	Pred.
Peter	Benjamin B.	3
Peter	Australia	3
Anna	Benjamin B.	2
Anna	Australia	1
Anna	Titanic	3

$p\tilde{}_{1,2}$ (avg. of p_1 and p_2)

User	Movie	Pred.
Peter	Benjamin B.	3.5
Peter	Australia	2.5
Anna	Benjamin B.	2.5
Anna	Australia	2
Anna	Titanic	4

Labels

User	Movie	Label
Peter	Benjamin B.	4
Peter	Australia	3
Anna	Benjamin B.	3
Anna	Australia	2
Anna	Titanic	4

Figure 6.1: Calculation of the weights of connections. The weights of the connections represent mutual error compensation power, which is calculated as the error of the combined (averaged) predictions.

The calculation of the connection weights is illustrated in an example from the movie recommendation domain in Figure 6.1. The higher rating is predicted for a user-movie pair by an elementary model, the more probably (according to that elementary model) the particular user will like that movie. Here, I show how to calculate the weight of the connection between the prediction models m_1 and m_2. Tables in the left upper part of the figure show the predictions of m_1 and m_2 denoted by p_1 and p_2 respectively. These predictions are averaged in the right upper table denoted by $p\tilde{}_{1,2}$. Using the labeled training data (or a subset of that), one can approximate the error of $p\tilde{}_{1,2}$ which can be used as a connection weight.

6.2.2 Ensemble framework

The pseudocode of GRAMOFON, my General Model-Selection Framework based on Networks is shown in Algorithm 7. GRAMOFON works with various *error functions*, *subset score functions* and *meta model types*. In order to scale up the selection, one can specify a predicate called *examine* that drives the search by determining which connections should be examined and which ones should be excluded. As it will be shown in section 6.3, the specific choice of these parameters (error function, subset score function, meta model type and examine predicate) results in various ensemble methods having the common characteristic that they all exploit the error compensation effect. In the following paragraphs, I describe the steps of the algorithm in more detail.

While learning, the training data is divided into two disjoint subsets $\mathcal{D}_A$ and $\mathcal{D}_B$ of the same size (lines 3 and 4)[6]. This division of the training data is iteratively repeated in a round robin fashion (see the outer for-loop from line 2 to 19).

In line 5, the model-network is build as described in Section 6.2.1. GRAMOFON iterates over the connections of the model-network (see the inner for-loop from line 9 to 18) in order to select models. While doing so, the connections are processed in the order of their weights, beginning with the connection corresponding to the best pair of models (see lines 8 and 9). In case of using RMSE as error function, smaller values indicate better predictions, therefore one processes the connections in *ascending* order with respect to their weights.

M_i denotes a set of models that are selected in the i-th iteration of the outer loop, and $score_{M_i}$ denotes the score of M_i. This score reflects how good the ensemble is based on the models in M_i. Formally speaking, the *subset score function* can be defined as a function that assigns a numerical value to the subset of models M_i. While iterating over the connections of the model-network, the algorithm tries to improve $score_{M_i}$ by adding models to M_i. If a model improves by at least ϵ, it will be added to M_i, otherwise not.

For each division of the training data (see the outer loop from line 2 to 19), the algorithm selects a set of models M_i. M_{final} denotes the set of such models that are contained at least n times among the selected models, i.e. improve at least n times by at least ϵ (n and ϵ are hyper-parameters). Finally, a meta model $\mathcal{M}$ of type *meta_model_type* is trained over the output of models of M_{final} using all training data instances (line 22). Then, $\mathcal{M}$ can be used for the prediction task (for unlabeled or test data).

Note that my framework operates fully at the meta level: the time series themselves, or, in case of vector data, the values of the attributes of instances are never accessed directly, only the prediction vectors delivered by the models are used. Also note that the hyper-parameters (ϵ and n) can be learned using a hold-out subset of the train data that is disjoint from $\mathcal{D}$.

6 This is a natural way to split because it allows effective learning of the selection since it balances well between fitting and avoiding of overfitting.

Algorithm 7
GRAMOFON: General Model-selection Framework based on Networks

```
Require: SubsetScoreFunction $f$, Predicate $examine$, ErrorFunction $err$,
    ModelType $meta\_model\_type$, Int $n$, Real $\epsilon$, set of all models $MSet$,
    labelled training data $\mathcal{D}$
Ensure: Ensemble of selected models
 1: data[ ] $splits$ = split $\mathcal{D}$ into 10 partitions
 2: for $i = 0; i < 10; i++$ do
 3:   data $\mathcal{D}_A \leftarrow splits[\ i\ ] \cup \ldots \cup splits[\ (i+4) \bmod 10\ ]$
 4:   data $\mathcal{D}_B \leftarrow splits[\ (i+5) \bmod 10\ ] \cup \ldots \cup splits[\ (i+9) \bmod 10\ ]$
 5:   $G \leftarrow$ build model-network using $\mathcal{D}_A$, calculate connection weights by Algo-
      rithm 6 for all connections (edges) $\{\ m_j, m_k\ \}$
 6:   $M_i \leftarrow$ empty set
      (In $M_i$ we collect the models selected in the $i$-th iteration.)
 7:   Let score$_{M_i}$ be the worst possible score
 8:   $E(G) \leftarrow$ sort the connections of $G$ according to their weights, begin with
      the best one
 9:   for all connections $\{m_j, m_k\}$ in $E(G)$, process connections according to
      the order do
10:     if $m_j \in M_i \wedge m_k \in M_i$ then proceed to the next connection
11:     if $examine(\{m_j, m_k\})$ then
12:       $M_i' \leftarrow M_i \cup \{m_j\} \cup \{m_k\}$
13:       $score_{M_i'} \leftarrow f(M_i', \mathcal{D}_A, \mathcal{D}_B, err, G)$
14:       if $score_{M_i'} + \epsilon < score_{M_i}$ (i.e., $score_{M_i'}$ is better than $score_{M_i}$ at least
          by $\epsilon$) then
15:         $M_i \leftarrow M_i'$,   $score_{M_i} \leftarrow score_{M_i'}$
16:       end if
17:     end if
18:   end for
19: end for
20: $M_{final} \leftarrow \{m \in MSet | m$ is included in at least $n$ sets among $M_0 \ldots M_9\}$
21: Let $p_i = m$.predict($\mathcal{D}$) for all $m \in M_{final}$
22: $\mathcal{M} \leftarrow$ train a model of type $meta\_model\_type$ over the prediction vectors of the
    models in $M_{final}$, i.e., use $\{p_i\}$ as training data and $\mathcal{D}$.labels as labels when
    training the model $\mathcal{M}$
23: return $\mathcal{M}$
```

6.3 Ensemble Techniques

As mentioned, the specific choice of the (i) error function *err*, (ii) subset score function f, (iii) *examine* predicate and (iv) *meta_model_type* lead to different ensemble techniques. In all of the proposed techniques, the error function calculates

```
Algorithm 8 Score Average Prediction: $f_{Avg}$
Require: Modelset $M$, Data samples $\mathcal{D}_A$ and $\mathcal{D}_B$, ErrorFunction $err$,
    Model-network $G$
1: for $\forall m_i \in M$ do $p_i = m_i$.predict($\mathcal{D}_B$),
2: $\forall x \in \mathcal{D}_B : p[x] = (\sum_i p_i[x])/M$.size
       (Predictions are averaged for each instance)
3: return $err(p, D_B$.labels)
```

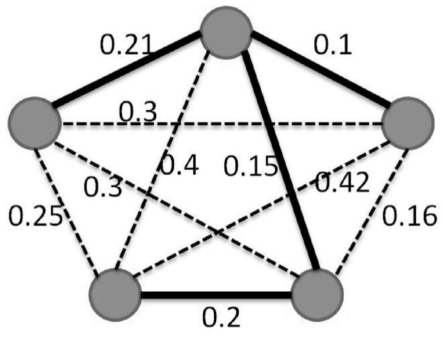


Figure 6.2: An example for a model-network with its minimal spanning tree (MST). Connections (edges) belonging to the MST are the bold ones. MST balances between two criteria: it contains good connections and all the nodes (models) are reachable through to connections of the MST.

root mean squared error (RMSE) (Eq. 6.5). As *meta_model_type* I propose to chose multivariate linear regression.

Next, I describe further characteristic settings of the ensemble techniques Basic, BasicFast, RegOptMST and NetworkMST.

6.3.1 Basic

When searching for the appropriate subset of models M_i, the *average of predictions* of models in M_i is calculated for each instance and M_i is scored based on that: I use f_{Avg} (Algorithm 8) as subset score function in the 13th line of Algorithm 7. In case of Basic, the *examine* predicate is constant true.

6.3.2 BasicFast

In order to save computation time, one can avoid the examination of all the connections of the model-network by examining only the most promising ones. Therefore, one first determines a minimal spanning tree (MST) of the model-network. Then, only the connections (edges) contained in the MST are examined. An example for a model-network with its MST can be seen in Figure 6.2.

In BasicFast the $examine_{\text{MST}}$ predicate is used, which is true for the connections (edges) contained in the minimal spanning tree of the model-network and false for the other connections. Similarly to the Basic technique, f_{Avg} (Algorithm 8) is chosen as subset score function.

Algorithm 9 Score Model Set using Linear Regression: f_{Reg}

```
Require: Modelset $M$, Data samples $\mathcal{D}_A$ and $\mathcal{D}_B$, ErrorFunction $err$,
    Model-network $G$
1: Let $p_i^A = m_i$.predict($\mathcal{D}_A$) for all $m_i \in M$
2: Train multivariate linear regression $\mathcal{L}$ using $\{p_i^A\}$ as training data and
   $\mathcal{D}_A$.labels as labels
3: $p = \mathcal{L}$.predict( $\{p_i^B\}$ ), where $p_i^B = m_i$.predict($\mathcal{D}_B$) for all $m_i \in M$
4: return $err(p, \mathcal{D}_B$.labels)
```

In an earlier version of these methods[7] [28], instead of $examine_{\text{MST}}$ I used the $examine_{\text{topN}}$ predicate that was true for the best N connections of the network. Connections (edges) selected by $examine_{\text{topN}}$, however, could form a (highly) connected subgraph of the model-network. Therefore there were no guarantees to contain all the nodes (models) among the examined ones. In contrast to $examine_{\text{topN}}$, in the case of $examine_{\text{MST}}$, the minimal spanning tree (MST) balances well between two criteria: it contains good connections (edges having small weights)[8] and it contains all the nodes of the model-network; thus every model has the chance to be selected. In order to empirically support these claims, when presenting the experiments in Section 6.4.1, I will report the number of examined models for both cases.

In addition, note that the MST can be found fast using e.g. Kruskal's simple greedy algorithm. [46]

6.3.3 RegOptMST

Similarly to BasicFast, in RegOptMST the $examine_{\text{MST}}$ predicate is used again. However, instead of f_{avg}, I use multivariate linear regression, i.e., the f_{Reg} function (Algorithm 9) in order to score the current model selection in the 13th step of Algorithm 7.

In Algorithm 9, a multivariate linear regression $\mathcal{L}$ is trained over the currently selected models' outputs for the data sample $\mathcal{D}_A$, then the algorithm calculates how well this linear regression fits to the data sample $\mathcal{D}_B$.

6.3.4 NetworkMST

This model selection technique operates exclusively on the model-network: f_{Net} serves as subset score function (Algorithm 10) and the $examine_{\text{MST}}$ predicate is used. Function f_{Net} calculates an average-like aggregation of the connection

7 called EarlyStop, RegOpt and GraphOpt

8 Remember that in the current case small weights denote good connections because RMSE is used as error function.

Algorithm 10 Score Model Set using the Model-Network: f_{Net}

```
Require: Modelset $M$, Data samples $\mathcal{D}_A$ and $\mathcal{D}_B$, ErrorFunction $err$,
    Model-network $G$
1: SumW $\leftarrow 0$
2: for $(\forall \{m_i, m_j\} | m_i, m_j \in M)$ do
3:     SumW $\leftarrow$ SumW $+ G$.connectionWeight($\{m_i, m_j\}$)
4: return $\frac{\text{SumW}}{(M.\text{size})^2 * \ln(M.\text{size})}$
```

weights, but gives priority to larger sets: the sum of the weights is divided by a number which, by increasing the number of connections, *grows faster* than the number of connections. This leads to lower scores for larger sets, which gives priority to larger sets: as RMSE is used as error measure when calculating the weights of the edges, smaller values correspond to better scores. If simply the average were calculated (without prioritizing large sets), the set M containing solely the nodes of the best connection (and no other nodes) would optimize the score function and therefore GRAMOFON would not be capable of finding any model set with size larger than 2.

6.3.5 Analysis

Basic examines $\mathcal{O}(N^2)$ connections (N is the number of models which is equal to the number of nodes in the model-network). In contrast, due to the $examine_{\text{MST}}$ predicate, BasicFast examines the most promising $\mathcal{O}(N)$ connections only. For this reason, BasicFast is expected to be substantially faster than Basic and to have approximately the same quality. One expects RegOptMST to be slower than BasicFast, because, from the computational point of view, training a linear regression is more expensive then calculating an average. In contrast to RegOptMST, which works in a meta-wrapper fashion, filter methods like NetworkMST are expected to be faster, because of the computationally inexpensive subset score function. Nevertheless, NetworkMST may produce worse results as only the information encoded in the model-network is taken into account.

Note that one can expect well-performing ensemble techniques, if the score function f and the *meta_model_type* are chosen in a way that there is a natural correspondence between them, like in the case of the above ensemble techniques.

Also note that Algorithm 8 and Algorithm 9 are conceptual descriptions of the score functions: in the implementation, the base models are not invoked as many times as the score function is called, but their prediction vectors are pre-computed and stored in an array.

Table 6.2: Main characteristics of the datasets associated to the RMSE task of the Ensembling Challenge at the Australian Data Mining Conference 2009.

	AusDM-S	AusDM-M	AusDM-L
Number of models	200	250	1151
Number of cases	15000	20000	50000

6.4 Experimental evaluation

I performed experiments on (i) datasets associated to the Australian Data Mining Conference 2009, (ii) datasets from the UCI repository and (iii) time series data. These experiments are described in the following subsections.

6.4.1 Experiments on the AusDM Datasets

Datasets. In order to allow for proper evaluation of the introduced techniques, I used datasets that were designed for the comparison of ensemble learning methods. In particular, I used the datasets of the *RMSE task* of the *Ensembling Challenge of the Australian Data Mining Conference*[9] 2009. In this challenge, three datasets were published, namely *Small*, *Medium*, and *Large* that are denoted in this section as AusDM-S, AusDM-M and AusDM-L respectively. These datasets contain the outputs of large collections of prediction models. All the models solve the same task, their outputs can therefore be combined in order to achieve better predictions. For this reason, these datasets are suited to evaluate different ensemble methods that combine the outputs of the models in various ways. The sizes of the datasets, i.e., number of models and cases, are summarized in Table 6.2.

The underlying prediction models were developed for movie rating prediction by different teams of the Netflix challenge. In this domain, the task is to predict for given user-movie pairs, how well the user will like that movie. The higher the rating, the more probably the user will like that movie (see also Figure 6.1). For more information on the Netflix prize see e.g. [13], [169], [170]. In the Netflix challenge, the task was to predict how users rate movies on a 1 to 5 scale (5=best, 1=worst). In the AusDM datasets, however, both the predicted ratings and the target were multiplied by 1000 and rounded to an integer value.

Experimental settings. I examined several baselines, namely the method proposed by Tsymbal [177], as well as stacking of different number of individually best models with linear regression and support vector machines (SVM).[10] This

9 http://www.tiberius.biz/ausdm09/

10 As described, in the examined data, only the outputs of the prediction models are present and therefore it allows for the comparison of learning methods that work exclusively on the meta-level. For this reason, I had to slightly modify Tsymbal's method in a way that it calculates the similarity based on the meta-level information.

Table 6.3: Performance (RMSE averaged over the 10 folds) of the baseline and my methods. (The numbers in parenthesis indicate in how many folds my method won against the baseline.)

Method	AusDM-S	AusDM-M	AusDM-L
SVM-Stacking best 20 models	871.97	872.09	876.66
Basic	869.86 (8)	868.34 (10)	871.88 (10)
BasicFast	869.96 (9)	868.48 (10)	872.04 (10)
RegOptMST	869.25 (8)	868.53 (10)	871.41 (10)
NetworkMST	868.63 (9)	866.86 (10)	870.77 (10)

selection of the individually best models is in accordance with [24]. In order to keep comparisons clear, I selected a single baseline, stacking of the individually best models with SVMs, because SVM is generally regarded as one of the best performing regression/classification methods.[11]

I used the WEKA-implementations[12] of SVM [92], [132], [161] (for the baseline) and Linear Regression (for RegOptMST and for the meta models). I performed 10-fold-crossvalidation.[13] Proper values for the hyperparameters of the SVM and my models (complexity constant C, exponent of the polynomial kernel e; and n, ϵ respectively) were determined using a hold-out subset of the training data.[14]

Results on Prediction Quality. The results regarding the performance of my methods are summarized in Table 6.3. The prediction quality is measured by the root mean squared error (RMSE) on the test data.

Similarly to [186] and [196], in parentheses I report the number of folds where my methods won against the baseline. Additionally, I also tested statistical significance with t-test at significance level of 0.05 and I found that my ensemble methods significantly outperformed the baseline in all of the cases.

Number of Examined Models. I compared the $examine_{\text{MST}}$ and $examine_{\text{topN}}$ predicates in terms of the number of examined models. These results are shown in Table 6.4. The results show that by simply selecting the best N connections, many models are *a priori* excluded from the search procedure while connections along the minimal spanning tree ensure that all of the models are examined.

11 In the reported results, I used stacking of the 20 individually best models because i) this number leads to very good performance for the baseline, and ii) ensures fair comparison of all examined methods by having approximately the same number of selected models.

12 http://www.cs.waikato.ac.nz/~ml/

13 This is not to be confused with the internal data splitting in Algorithm 7, which is performed in each each round of the 10-fold-crossvalidation on the current *training* data. In each round of the 10-fold-crossvalidation, Algorithm 7 is executed according to which this internal splitting of the current training data is iteratively repeated several times in a round robin fashion.

14 In order to simplify the reproducibility, I report the found SVM-hyperparameters: $e = 2^0 = 1$ and $C = 2^{-5}$ (AusDM-S), $C = 2^{-3}$ (AusDM-M), $C = 2^{-8}$ (AusDM-L).

Table 6.4: The number of examined models (averaged over 10 folds and rounded to integer values) in case of $examine_{\mathsf{MST}}$ and $examine_{\mathsf{topN}}$.

	AusDM-S	AusDM-M	AusDM-L
$examine_{\mathrm{topN}}$	52	59	211
$examine_{\mathrm{MST}}$	200	250	1151

Discussion. All the proposed techniques clearly outperform the baselines. As expected, compared to *Basic*, *BasicFast* lost almost nothing in terms of quality. Regarding *RegOptMST*, I observed that scoring the candidate model sets using linear regression (f_{Reg}) instead of averaging (f_{Avg}) could only slightly improve compared to *BasicFast* and *Basic* in case of 2 datasets. *NetworkMST* that works according to the filter schema, clearly outperforms the baselines, and it was slightly better than *Basic*. These observations are in accordance with the previously described expectations.

6.4.2 Experiments on UCI Datasets

As described in Section 6.2.2, my approach, GRAMOFON, works fully at the meta level. Therefore, only the outputs of the models (i.e. their predictions) are used, while neither the models themselves nor the data are accessed. Due to this reason, in the first experiments presented in the previous section I used the AusDM datasets because they are suitable for the evaluation of meta-learning methods as they contain the predictions of a large collection of various models.

Datasets. In order to be able to draw more generic conclusions, I performed experiments on datasets from the UCI Machine Learning repository [69] too. Out of the datasets associated with regression problems, I selected *Auto-Mpg* [137], *Communities-and-Crime* [147], *Housing* [137] and *Wine-Quality* [48].[15]

Elementary Prediction Models. In contrast to the AusDM datasets, the datasets from the UCI repository are not meta-level datasets, i.e., they do not contain the predictions of models, instead, they contain the elementary data. Therefore, for my approaches as well as for the stacking-based baselines, I had to construct elementary prediction models. In general, there are plenty of ways to do that: degrees of freedom include the *types* and *hyperparameters* of the chosen models. As kernel selection is one of the most crucial recent issues, and no single kernel seems to be an absolute winner [172], I decided to test my approach in the context of kernel selection. I trained SVMs with RBF kernels, normalized and unnormalized polynomial kernels. Similarly to the experiments on the AusDM data, I used the WEKA-implementations of these models. I varied γ

15 For Wine-Quality I used the dataset containing *white* wines.

Table 6.5: Performance of GRAMOFON (Basic) and its competitors: root mean squared error (RMSE) on test data averaged over 10 folds.

Method	Auto-Mpg	Communities and Crime	Housing	Wine-Quality
Stacking (RBF Network)	4.812	0.172	6.567	0.763
AdaboostRT	5.188	0.199	7.300	0.813
Basic	**2.671**	**0.136**	**3.574**	**0.697**

for the RBF kernels and the exponent for the polynomial kernels in the range $\{2^{-2}, 2^{-1}, ..., 2^{2}\}$.[16] Similarly, I set the complexity constant in the same range. This resulted in $3 \times 5 \times 5 = 75$ models.

Experimental Settings. I compared the method against two state-of-the-art ensemble approaches: (i) stacking with RBF Networks and (ii) boosting with AdaboostRT[17], a recent extension of Adaboost for regression problems [159].

I performed 10-fold-crossvalidation: in each round, 5 data splits served as train data for the elementary models, and 4 splits were used to train the meta model and 1 split was reserved as test data. In particular, I first trained the SVMs with various kernels on 5 splits (these SVMs were the elementary models in the experiment). Then the SVMs delivered predictions for the remaining 5 splits; out of which 4 splits served as training data for my approach and its stacking-based competitor, and I used the remaining 1 split as test data.

Results. Table 6.5 summarizes the results averaged over 10 folds. In order to keep the presentation simple, I only show results for the Basic approach as representative for the GRAMOFON-approaches. Note, however, that similarly to Basic, all the other approaches outperformed the competitors.[18] These results are in line with the observation that in challenging cases, bagging (that is a simple version of stacking) may outperform boosting [54].

Discussion. Regarding execution times, even for the largest out of these datasets, Communities and Crime, the ensemble construction with GRAMOFON (Basic)

16 For the normalized polynomial kernel the exponent of $2^0 = 1$ was not applicable, therefore I used $2^{0.5} = \sqrt{2} \approx 1.41$.

17 I tried AdaboostRT with various hyperparameters. In general, I found AdaboostRT to be robust in terms of hyperparameter settings, i.e., I observed only minor differences between the settings. I report results for the settings that generally performed best out of the examined ones. In order to simplify reproducibility, I report these hyperparameters of AdaboostRT: number of iterations $T = 100$, demarcating threshold $\phi = 0.5$, power coefficient $n = 1$. As elementary models I use decision stumps in AdaboostRT.

18 The differences between my approach and its competitors were always significant in terms of average and standard deviation: the differences were always larger than two-times standard deviation.

took only approximately half a minute. This was, however, five times longer than the execution time required for AdaboostRT. Despite this, I still argue that in terms of the trade-off between execution time and quality, it is worth using GRAMOFON, because the vast majority of execution time was spent not for constructing the ensemble, but for training the elementary models that took about 50 minutes. Compared to that, the ensemble construction was very fast, in case of GRAMOFON too.

6.5 GRAMOFON for Time-Series classification and regression

The described GRAMOFON framework works fully at the meta-level, i.e., exclusively with the outputs of elementary prediction models. Therefore, assuming that the elementary models work with time series, GRAMOFON can combine these elementary models in order to achieve more accurate predictions.

The previous description of the GRAMOFON framework focused on regression. However, the basic idea, i.e., paying special attention to the mutual error compensation of elementary prediction models, is not limited to regression models at all. In order to allow for classification, one can re-define the graph construction and specify the error function *err*, subset score function f, *examine* predicate and *meta_model_type*.

Furthermore, even without doing so, GRAMOFON can be used for time-series classification problems by converting the classification problem into a regression problem. In case of classification, labels describe discrete classes. When converting into a regression problem, instead of denoting a particular class, one can re-define labels so that they denote the *likelihood* (or probability) of a given class. Then, for the training data, the new labels of instances belonging to the given class will be 1, while other instances will have label 0. While predicting labels for new (test) instances, regression models generate continuous values for the likelihood of the selected class. Then, these values can be combined together with GRAMOFON that outputs a better estimation for the likelihood of the selected class. If one only aims at recognizing one single class (because e.g. only this class is relevant for the current application), one can set a decision threshold: whenever the likelihood generated by GRAMOFON is above that threshold, the instance is considered to be recognized as an instance of that class. If, however, all the classes are relevant, in a similar procedure, one can estimate the likelihood of each class separately and one can finally select the most probable class.

6.5.1 Experiments on Time Series Data

In order to demonstrate the applicability of GRAMOFON to time series data, I performed experiments on some of the datasets from the collection introduced in

Table 6.6: Performance of GRAMOFON (Basic) and its competitors: root mean squared error (RMSE) on test data averaged over 10 folds.

Method	Motes	Lighting2	ItalyPowerDemand	GunPoint
Average[a]	0.161	0.301	0.174	0.059
Best[b]	0.133	0.439	0.236	0.063
Basic	**0.131**	**0.267**	**0.173**	**0.035**

[a] Average of the predictions of the 12 examined elementary models
[b] Best elementary model selected in each round of the 10 fold cross validation

Section 2.7. I selected some of the small datasets associated with binary classification tasks and changed the semantics of the labels so that the labels denote the likelihood of one of the classes.[19] Therefore, all the true labels were either 0 or 1.

In the experiment, I used 12 nearest neighbor classifiers as elementary models. For all these classifiers, k, the number of nearest neighbors taken into account was set to 1. The variety of the classifiers were ensured by using different distance measures for each classifier.[20]

After redefining the semantics of the class labels, I split the data into two disjoint subsets of equal size. I used the first one as training data for the elementary models, then I predicted labels by the elementary models for the second split. More concretely, I used the first split in the nearest neighbor classifiers in order to predict labels of the second split.

For the time series of the second split, both the outputs of various prediction models (nearest neighbor classifiers with different distance measures) and the true class labels were present, and I constructed a *meta-level data set* that contains the predictions of the various elementary models together with the true class labels. Then I used this meta-level data set in an experiment similar to the one described in Section 6.4.1.

The results[21] (RMSE averaged over the ten rounds of the 10 fold cross validation on the constructed meta-level data sets) are summarized in Table 6.6. As a representative of the GRAMOFON techniques, only Basic is shown.

19 This kind of redefinition of the semantics of the labels could be implemented simply: in case of the selected class, the labels were changed to 1, while in case of the other class, the labels were changed to 0.

20 I used the same distance measures as in Section 5.3.

21 Note that in Section 5.3, *classification error* is reported, while here RMSE is used as quality measure. This is in accordance with the redefined semantics of the labels and the previous sections of this Chapter.

Chapter 7

Motifs for Time-Series Classification

In this Chapter, I focus on enhancing recognition systems by taking recurrent patterns in time series data into account. These recurrent patterns are called motifs. Motifs serve as basis for the extraction of additional features. I use these additional features (together with the ones already present form other sources) in conventional classifiers, in particular: Bayesian Networks and SVMs. As the major contribution of this Chapter, In the experiments I show that motifs may enhance recognition systems.

7.1 Generalized Semi-Contiguous Motifs

As described in Section 2.4.4, in case of motif-based classification of time series, in the first phase, one searches for motifs in the time series. In this section, I describe my motif discovery approaches in detail. Throughout this description, I assume that the time series are *converted into a sequence of discrete symbols* (e.g. using SAX, see Section 2.2.3). By *motif discovery* I mean finding which subsequences occur frequently in the converted time series.

7.1.1 Definitions

Given a database of converted time series $\mathcal{D}_{SAX}$, a set of symbols Σ and a taxonomic relation T_Σ over Σ, the maximal number of allowed gaps n_m, the maximal allowed length of each gap d_m, and a minimum support threshold $s \in \mathbb{Z}$, I define matching and motif as follows.

Definition 13 (Matching of two symbols)**.** *A* symbol $z \in \Sigma$ matches *an other symbol $z' \in \Sigma$ if either $z = z'$ or z' is a descendant of z according to T_Σ, z is called the* matching symbol*, z' is called the* matched symbol*. (See Figure 2.12 for an example.)*

Definition 14 (Matching of sequences and time series)**.** *A* sequence of symbols x_0 semi-contiguously matches *a converted times series $x \in \mathcal{D}_{SAX}$, if all symbols of x_0 match at least one symbol in x so that (i) the order of matched symbols are the same as the order of matching symbols, and (ii) the matched symbols in x build a basically contiguous subsequence, but maximal n_m gaps are allowed provided that the length of each gap is not larger than d_m.*

Definition 15 (Generalized semi-contiguous motif)**.** *A sequence of symbols x_0 is called* generalized semi-contiguous motif*, if it matches at least s time series in $\mathcal{D}_{SAX}$. The number of matched time series is called* support *of x_0.*

7.1.2 Anti-monotonicity Constraints

Checking the support of all possible sequences in order to decide whether they are motifs or not is computationally infeasible due to the large number of possible sequences. Thus, the search space needs to be pruned in order to reduce the number of sequences to be examined and an efficient implementation for support counting is needed. For this reason, I adapt constraints in [3], [165] and [166] for semi-contiguous generalized motifs, and, in the next sections, I extend optimization ideas in [19] and [20] for generalized semi-contiguous motifs.

The basic intuition behind the constraints on generalized semi-contiguous motifs is to use of the anti-monotonous property of the support function: (i) if a sequence x_0 contains another sequence x'_0, the support of x_0 is less than or equal to the support of x'_0, (ii) if a sequence x_0 is less general than some other sequence x'_0, the support of x_0 is less than or equal to the support of x'_0.

These conditions hold as the number of time series matching x_0 cannot be higher than the number of time series matching x'_0, as every time series x matching x_0 also matches x'_0. Next, I describe these constraints formally. Both constraints are consequences of the definition of support and matching. Let x_0 be a sequence over Σ (Σ contains elementary symbols and taxonomic wild-cards, therefore, besides elementary symbols, the sequence may include taxonomic wild-cards too):

$$x_0 = (w_1, w_2, \ldots, w_{k-1}, w_k), \text{ each } w_i \in \Sigma,\ 0 < i \le k \tag{7.1}$$

Constraint 1 (Sequential Anti-monotonicity)**.** Let x'_0 be a subsequence of x_0:

$$x'_0 = (w_j, w_{j+1}, w_{j+2}, \ldots w_{j+l}),\ 0 < j \le j + l \le k \tag{7.2}$$

In this case: $support(x_0) \le support(x'_0)$.

Constraint 2 (Taxonomic Anti-monotonicity)**.** Denote the transitive closure of the taxonomic relation T_Σ with T^*_Σ, i.e., $(w, w') \in T^*_\Sigma$ means that w is a (not necessarily direct) descendant of w' in T_Σ. Assume that x'_0 is a more general sequence than x_0:

$$x'_0 = (w'_1, w'_2, \ldots, w'_{k-1}, w'_k) \quad \text{with } \forall i : (w_i, w'_i) \in T^*_\Sigma, 0 < i \le k \tag{7.3}$$

In this case: $support(x_0) \le support(x'_0)$.

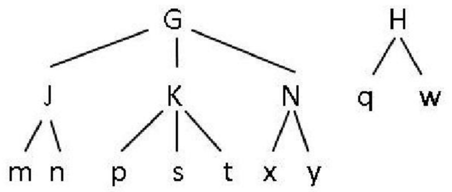
These constraints suggest to check the shorter and more general sequences first, whether they are motifs or not. For example, if we are given the taxonomy depicted in the left of Figure 7.1, pattern (G, H) would be checked before (G, w) and (G, H, H).

7.1.3 Extensions of Apriori

The first algorithm I propose for finding generalized semi-contiguous motifs is an extended version of the algorithm Apriori [3]. The basic Apriori algorithm iterates over three phases:

1. *Candidate generation* – Some sequences are selected so that their support is checked in order to decide whether these sequences are motifs or not. The selected sequences are called *candidates*. In the first iteration, all possible sequences of length one are candidates. In other iterations, candidates are generated based on the motifs found in the previous iteration. The algorithm terminates if no more candidates can be generated.
2. *Support counting* – The support of each candidate is determined.
3. *Filter infrequent candidates* – The candidates with support lower than the given threshold s are deleted. The other ones are motifs.

a)



b)

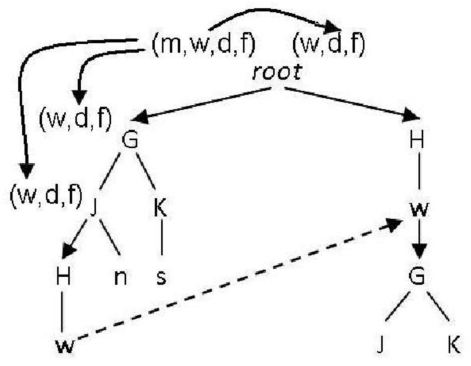


Figure 7.1: a) An example taxonomy of the symbols G, H, J, K, N, m, n, p, s, t, x, y, q and w. This taxonomy has two roots: G and H. b) An example illustrating the data-structure used for counting the support of candidates and storing motifs in the extended Apriori. Straight arrows denote sequential children, lines denote taxonomic children. One of the cross-pointers is depicted with dotted arrow. Curved arrows show the recursion steps according to the applied recursive search schema.

The computational cost of Apriori highly depends on the applied data structure. Tries (or prefix trees) have been shown to be efficient for storing candidates and the found motifs, see e.g. [19]. In the trie I used, there are *taxonomic* and *sequential* edges, a path from the root to a node encodes a sequence. A simplified example of the data structure is shown in the right of Figure 7.1, the two different types of edges are shown with straight lines and straight arrows respectively. Consider, for example, the path $(root, G, J, H, w)$. There are two different types of edges in this path: sequential edges are $(root,G)$ and (J,H), while taxonomic edges are (G,J) and (H, w). Each sequential edge denotes a new symbol of the sequence. The taxonomic edges in the path specialize the symbols. In this path, G was specialized to J, and H was specialized to w, therefore the path encodes the sequence (J, w).

There are also cross-pointers in the data structure that point from a sequence $(w_1, w_2, \ldots, w_k)$ to the sequence $(w_2, \ldots, w_k)$. In order to keep the example simple, only one cross-pointer is depicted by the dashed arrow.[1] These "cross-pointers" allow quick candidate generation.

Candidate Generation

In this section I describe candidate generation that is performed in all iterations of Apriori.

Let c_r denote the count of roots in the taxonomy T_Σ. At the beginning of the first iteration, there are c_r candidates, and each one corresponds to one of the roots of the taxonomy. These are the most general and shortest sequences (they consist of one item).

In each iteration, after counting the support of the current candidates, the frequent patterns (motifs) are determined, and the candidates for the next iteration are generated based on the motifs already found. There are two possible ways to generate candidates from the motifs that are already known to be frequent.

Specification — As application of Constraint 2, given that $x_0 = (w_1, \ldots, w_{k-1}, w_k)$ is a motif, and therefore frequent, its *specification* is $x'_0 = (w_1, \ldots, w_{k-1}, w'_k)$, where the last symbol w_k of x_0 is replaced by a more specific one, w'_k, i.e., $(w'_k, w_k) \in T^*_\Sigma$. According to Constraint 1, the specification of x_0 can only be frequent if $x_i = (w_2, \ldots, w_{k-1}, w'_k)$ is also frequent. Due to the cross-pointers, this can be checked quickly. If x_i is frequent, x'_0 will be a candidate and the number of time series matched by x'_0 will be determined in the support counting phase of Apriori.

1 Note that symbols in the sequences may denote symbols from any level of the taxonomy. As shown, in the trie, several edges may correspond to one single symbol: in the example shown in the right of Figure 7.1, the symbol w is represented by two edges: $(*, H)$ and (H, w), where $*$ is J in the left branch of the trie and *root* in the right branch of the trie.

Sequential Extension — As application of Constraint 1, a motif x_0 of length 1 may sequentially be extended by any other motif x_{ii} of length 1: x_0 and x_{ii} are concatenated to the new candidate x''. For longer motifs: knowing that the sequence $x_0 = (w_1, w_2, \ldots, w_{k-1}, w_k)$ is a motif, its sequential extension $x''_0 = (w_1, w_2, \ldots, w_k, w_{k+1})$, can only be a motif if $x_{ii} = (w_2, \ldots, w_{k-1}, w_k, w_{k+1})$ is frequent, i.e. x_{ii} is a motif. Therefore, if x_{ii} is frequent, x''_0 will be a candidate and the number of time series matched by x''_0 will be determined in the *support counting* phase of Apriori.

When generating candidates, in each iteration of Apriori, *all possible* specifications and sequential extensions are applied in order to generate candidates for the next iteration. Due to this policy, it is always known in advance, if the sequences p_i and p_{ii} are motifs: the support of p_i and p_{ii} (whether they are frequent or not), is determined before this information is necessary for the generation of candidates.

The new candidates are stored in the trie. As both candidates and motifs (sequences that will be checked for frequency and the ones that are already found to be frequent) are stored in the trie, one needs to distinguish between motifs and candidates, which is possible –, for example by assigning a binary indicator to each node of the trie (or by using counters in the nodes, see the subsequent paragraphs).

When extending the trie in order to contain the new candidates, the cross-pointers have to be updated according to their semantics as well.

Support Counting

Similarly to candidate generation, support counting is performed in each iteration of Apriori. In this section, I describe how I implemented this step.

For each node, the trie contains a counter. The counter of a node shows how many converted time series are matched by the sequence corresponding to that node of the trie. When extending the trie with new candidates, their counters are set to zero. Then, the converted dataset of time series is processed sequentially, one sequence at a time. For each time series, the counters of the matched candidates are incremented.

Matched candidates can be found efficiently using a doubly recursive search scheme. This is shown in the right of Figure 7.1 and in the pseudocode of Algorithm 11. In the right of Figure 7.1, counting is first invoked for the *root*-node with the converted time series (m, w, d, f). Then, counting is invoked for the nodes G, J and *root* with the tail of the previous sequence, i.e. (w, d, f). This is shown by curved arrows. The details of the procedure are explained below.

When processing the time series dataset, for each time series x, the function called support_count0 is invoked for the root of the trie in line 3 of Algorithm 11. At this point, the current node is the root of the trie and the second argument of support_count0 is the entire time series x. In the function support_count0, the first symbol of x is checked whether it is matched by any of the candidates. Then, the function support_count0 is recursively invoked with the tail of the current time series[2] for

1. that sequential child of the current node which matches the first symbol of the time series (if there is such a child),
2. all *those* taxonomic descendants of the aforementioned sequential child that match the first symbol of the time series (if there are such descendants), and
3. the current node.

Note that this step is a generalization of the corresponding step in [19]. The first two types of recursion allow for taxonomic wild-cards, while the third one allows for gaps.

During this traversal of the trie, the counters of the candidates are incremented so that at the end, i.e., when all the time series of the dataset are processed, the counters show how many time series are matched by each candidate.

In contrast to [19], in case of generalized semi-contiguous motifs, some additional administration is necessary: during the doubly recursive search, one has to take into account (and therefore possibly not invoke some of the third type recursion steps because of) (i) the number of "gaps" while matching the (d_m, n_m) semi-contiguous candidate to the input sequence, and (ii) the length of the current "gap", if there is currently a "gap" in the matched time series.[3]

While processing a single time series, it is possible to arrive several times at the same node, as a candidate may be matched by several segments of the time series. Therefore, we also have to take care to not increment the counter of a node twice while processing a time series.

In order to increase the efficiency, I propose to use the adapted version of the pruning technique described in [20], i.e., while traversing the trie according to the described schema, I propose not to visit those nodes that do not lead to any of the candidates, i.e. through which none of the candidates can be reached.

7.1.4 Extensions of Eclat

The second algorithm I present for finding generalized sequential semi-contiguous motifs is based on the Eclat algorithm [79]. In contrast to Apriori that examines sequences for being motifs in the order of their length and specificity, and therefore Apriori performs search in a bread-first-search manner beginning with the

2 With *tail* of a time series $x = (w_1, w_2, ..., w_k)$ the time series that is identical to x but does not contain its first element is meant: $\text{tail}(x) = (w_2, ..., w_k)$.

3 In my implementation, this additional administration is done by additional arguments of the support_count0 function that are set each time when the function is (recursively) invoked.

Algorithm 11
Support Counting for Generalized Sequential Semi-contiguous Motifs

```
Require: SAX-Converted Time Series Dataset $\mathcal{D}$
function support_count(SAX-Converted Time Series Dataset $\mathcal{D}$)
    for each time series $x \in \mathcal{D}$ do
        call support_count0(trie_of_candidates.root, $x$)
        if none of the counters were incremented because of $x$
                (That is: $x$ did not support any of the candidates)
        then ignore $x$ in the subsequent iterations of apriori
    end for
end

function support_count0(TrieNode node, TimeSeries $x$)
    Symbol first_symbol = getFirstSymbol($x$)
    TimeSeries tail_sequence = tailTimeSeries($x$)        (See also Footnote 2)
    TrieNode n1 = sequential child node of node which matches first_symbol
    if exists n1 and exists at least one candidate that is reachable over n1
    then
        $N = \{ n1 \} \cup$ set of such taxonomic descendants of n1
              that are matched by first_symbol
        for each $n0 \in N$
            if n0 is candidate and n0 has not been supported by $x$ before
            then incrementSupport(n0)
            call support_count0(n0, tail_sequence)
        end for
    end if
    if maximal gap length and maximal gap count not exceeded
        then call support_count0(node,tail_sequence)
end
```

shortest and most general sequences, Eclat organizes the search in a depth-first-search manner, i.e., some long and specific sequences are examined *sort* after the beginning of the search process.

Eclat assumes that each time series $x \in \mathcal{D}$ has a unique integer identifier, called *time series identifier* or *TID*.[4] The basic idea is to construct for each symbol $w \in \Sigma$ a so called *TID-list*, a list of TIDs of those time series that contain w. A *necessary* (but not sufficient) condition for a generalized semi-contiguous sequence $x_0 = (w_0, w_1, ..., w_k)$ to match the time series x is that all the symbols w_0, w_1, ..., w_k are contained in x. Therefore, if one wants to check the sequence $x_0 =$

4 The algorithm originates from the frequent itemset mining community where records of the database are often called *transactions*, and therefore the abbreviation *TID* originally stands for *transaction identifier*.

$(w_0, w_1, ..., w_k)$ for being frequent, one can first intersect the TID-lists of w_1, w_2, ..., w_k. If the length of the intersection is smaller than the predefined minimum support threshold s, one can be sure that x_0 is *not* frequent. If, on the other hand, the length of the intersection of the TID-list is larger than or equal to s, the intersection of the TID-list defines a *superset* of the set of time series that are matched by x_0. Then, a simple solution for determining the true support of x_0 is checking all the time series of the intersected TID-lists whether they are really matched by x_0. Algorithm 12 is based on this idea.

In Algorithm 12, additionally to the TID-list of a symbol w, the *TID-list of a sequence* x_0 is also used, which is the list of TIDs of those time series that match x_0. Keeping the anti-monotonicity constraints in mind, beginning from the shortest and most generic ones, sequences are grown and specialized (generic wildcards are replaced by more specific symbols), as long as the resulting sequences are frequent. When growing a sequence, the sequence is always extended by one of the roots of the taxonomy (these are the most generic symbols) in lines 8 through 17 of Algorithm 12. When specializing the sequence, the last symbol is replaced by a more generic one in lines 18 through 29 of Algorithm 12.

Algorithm 12 finds all the generalized semi-contiguous motifs due to the application of the above-described steps of growing and specialization of sequences. While doing so, it accesses the original time series dataset first when the TID-lists of all the symbols are created at the beginning (line 2). Afterwards, only those time series of the dataset are accessed that *potentially* match the currently examined sequence. For datasets of modest size that fit into the main memory (which supports random access to time series) this is often sufficient. Therefore, we could use this algorithm in joint work with Sebastian Blohm [16].

Speeding-up Algorithm 12

In order to speed up Algorithm 12 for large time series datasets, one can *avoid the accesses to the time series database in lines 11...16 and 22...27.* In order to do so, we need to modify the TID-lists. Up to now, TID-lists of a sequence x_0 (in some cases the sequence x_0 contained only one symbol) were a list of identifiers of time series matching x_0. One time series identifier was contained at most once in the list, even if the x_0 matched a time series at several positions.

In the modified version, each entry of the TID-list of a sequence x_0 corresponds to one *matching*, and therefore the same TID can be contained several times, if the corresponding time series is matched several times by x_0. Further, the entries of the TID-list are not single numbers, but triplets. Besides the time series identifier, such a triplet (*TID*,*pos*,*gaps*) contains additional information about the matching: the position, denoted as *pos*, where the given matching *ends* in the time series, and the number of *gaps* in the given matching.

Algorithm 12

Extended Eclat for Generalized Sequential Semi-contiguous Motifs

We used this algorithm in joint work with Sebastian Blohm [16].

This is a simplified conceptual description, for efficient implementation, in particular avoiding some of the accesses to the dataset, please see also Section *Speeding-up Algorithm 12*.

Require: SAX-Converted Time Series Dataset $\mathcal{D}$, minimum support threshold s

Ensure: Set of generalized sequential semi-contiguous motifs denoted as *motifs*

```
 1: function eclat(SAX-Converted Time Series Dataset $\mathcal{D}$,
    minimum support threshold $s$)
 2:   TID_Lists $TL$ = initTIDLists()
          (TID-Lists for $\forall w \in \Sigma$, including wild-cards)
 3:   call eclat0(empty prefix,list of all transaction IDs)
 4:   return motifs
 5: end

 6: function eclat0(TimeSeriesPrefix $p$, TID_List_of_Prefix $tid$)
 7:   $motifs$.add($p$)
 8:   for $\forall w \in \mathcal{D} \mid w$ is a root do
 9:     $p1$ = concatenation($p$,$w$)
10:     $tid1$ = intersect($tid$, $TL$.getTIDList($w$))
11:     if $tid1$.size() $\geq s$ then
12:       for $\forall$ $id \in tid1$ do
13:         if $p1$ does not match $\mathcal{D}$.getInputSequence($id$) then $tid1$.delete($id$)
14:       end for
15:       if $tid1$.size() $\geq s$ then call eclat0($p1$,$tid1$)
16:     end if
17:   end for
18:   if length($p$) $> 0$ then
19:     for $\forall w \in \mathcal{D} \mid w$ is direct descendant of lastItem($p$) do
20:       $p1$ = replaceLast($p$,$w$)
21:       $tid1$ = intersect($tid$, $TL$.getTIDList($w$))
22:       if $tid1$.size() $\geq s$ then
23:         for $\forall$ $id \in tid1$ do
24:           if $p1$ does not match $\mathcal{D}$.getInputSequence($id$) then $tid1$.delete($id$)
25:         end for
26:         if $tid1$.size() $\geq s$ then call eclat0($p1$,$tid1$)
27:       end if
28:     end for
29:   end if
30: end
```

Denote

$$TL(x_0) = \Big((tid_1,pos_1,gaps_1), ..., (tid_k,pos_k,gaps_k)\Big)$$

and

$$TL(w) = \Big((tid'_1,pos'_1,gaps'_1), ..., (tid'_m,pos'_m,gaps'_m)\Big)$$

the TID-list of a sequence x_0 and a symbol w respectively. The intersection of these two lists is calculated as follows. For all tid_i of $TL(x_0)$, one searches for those tid'_j of $TL(w)$ that are equal to tid_i. Suppose, $(tid'_j,pos'_j,gaps'_j)$ is one of the entries of $TL(w)$ corresponding to the entry $(tid_i,pos_i,gaps_i)$ of $TL(x_0)$, i.e., $tid_i{=}tid'_j$. In this case:

- If $pos'_j = pos_i + 1$, i.e., w directly follows x_0 in the current time series[5], then $(tid_i,pos_i + 1,gaps_i)$ is inserted into the TID-list of the intersection.
- If $0 < pos'_j - pos_i \leq d_m + 1$, i.e., w follows x_0 in the current time series, but there is a gap between them and the length of the gap does not exceed the maximal allowed gap length, **and** $gap_i < n_m$, i.e, the number of gaps in the matching of x_0 is less than the maximal allowed number of gaps, then $(tid_i,pos'_j,gaps_i + 1)$ is inserted into the TID-list of the intersection.

Formally: $(tid_k,pos_k,gaps_k) \in TL(x_0) \cap TL(w) \Leftrightarrow$

$$\Bigg(\Big((tid_k,pos_k - 1,gaps_k) \in TL(x_0) \wedge (tid_k,pos_k,0) \in TL(w)\Big) \bigvee$$
$$\Big((tid_k,pos_k - d_g,gaps_k - 1) \in TL(x_0) \wedge (tid_k,pos_k,0) \in TL(w)\Big)\Bigg)$$
$$\text{where } 1 < d_g \leq d_m \quad \text{and} \quad gaps_k \leq n_m.$$

When calculating the intersections of TID-lists according to the above-described procedure, the accesses to the time series dataset in lines 11...16 and 22...27 of Algorithm 12 become unnecessary, because the intersection now contains exactly the matchings of the new sequence (generated from x_0 and w). Now, the support of this new sequence equals to the number of *distinct* TIDs in the TID-list of the intersection (instead of the *length* of the TID-list).

7.2 Experiments

Dataset. The data used in the experiments was collected at the Fresenius Clinics. It contains recordings of dialysis sessions for 725 patients. The patients consult the doctor regularly for treatment, some data (like blood pressure, body temperature...) is recorded every time, which leads to a sequence of observations. There are about 40 different time series per patient. Some pieces of master data of the

5 the one having time series identifier: tid_i

patients (like age, sex, body mass index[6],...) are also available. There are two groups (classes) of patients: "normal" (53%) and "risky" (47%). This dataset was previously used in [105], [106], the Reader is referred to [105] for a more detailed description.

As usual in time series motif detection [105], [106], [127], as preprocessing step, I converted time series into a sequence of discrete symbols using Symbolic Aggregate Approximation (Section 2.2.3). The number of different symbols used in the experiments was $m_{SAX} = 7$, I aggregated elements of the time series over non-overlapping frames of length 4, i.e., $l_{SAX} = 4$.

Experimental Settings. In the experiments I examined the impact of motifs in the common scenario when both master data of the patients and some time series data are available. I discovered motifs on different time series separately, i.e. separately on the time series of blood pressure, body temperature, etc. Minimum support threshold was set to 0.05. For simplicity, I used contiguous motifs without wild-cards.[7] In line with [106], I select the best predictive motifs for each class: I select motifs that predict the "normal" class with a probability of 80% and the motifs that predict the "risky" class with a probability of 75%. Furthermore, I only select motifs that are statistically significant for a class (χ^2 test, significance level: $\alpha = 0.05$) and limit the total number of Apriori-iterations to 20 in order to get short and therefore *local* motifs.

I performed 10-fold-cross-validation (see Section 2.6). In each round of the 10-fold-cross-validation, motifs are derived using the current training set, i.e., the above described procedure of motif discovery and selection is performed on the current training set.[8] Then, the selected motifs are used to derive features in order to allow for the usage of a vector classifier. In Section 2.4.4 binary features were derived for each motif, these binary features indicate whether or not the motif is included in a time series. As features, in some cases (these cases are listed later) I used the total number of predictive motifs (TNPM) for each class, i.e., I used two features, one of them is the number of motifs that are characteristic for the class of *risky* patients while the other one is the number of motifs that are characteristic for the class of *normal* patients.

As vector classifier, I used the WEKA-implementations of SVMs (with RBF kernel) and Bayesian Networks.[9] The hyper-parameters of the SVMs (complexity

6 Body mass index (or BMI) is calculated based on the weight and height of the patient and it indicates whether the patient has overweight or underweight.

7 Note that my experiments reported in [36] indicate that both gaps and taxonomic wild-chards (that are allowed in more complex semi-contiguous motif types) can further improve accuracy compared to the case of using simple motifs. In order to keep the presentation simple, in this section only the experiments with contiguous motifs are reported.

8 Note that this experimental protocol differs from the one used in [106], thus our results are not directly comparable.

9 The choice of Bayesian Networks is also justified by the interpretability of the resulting model which is an important requirement in the medical domain [148].

Table 7.1: Classification accuracy (in %) with and without motifs

	Baseline (without motifs)	With motifs TNPM features	With motifs TNPM features + indicators
SVM	72,40	72,12	75,61
SVM (logistic)	72,38	73,35	76,43
Bayesian Network	73,84	74,76	74,76

constant and exponent) are learned by grid search in the range $2^{-10}, 2^{-9}, \dots 2^3$ and $2^{-10}, 2^{-9}, \dots 2^{11}$ using a hold-out subset of the training data.

As baselines, I used vector classifiers (SVMs and Bayesian Network respectively) without features derived from motifs. As features, the baselines used both basic aggregates and the master data of patients. The baselines were compared to vector classifiers that additionally use motif-features, i.e., features derived from motifs. Regarding the additional motif-features, I examined two cases: (i) only TNPM-features were used, and (ii) both TNPM-features and binary indicators were used.

Results. All the examined models (including baselines and motif-based models) worked much better than the trivial classifier that always outputs the majority class' label. The results (Table 7.1) show that motifs may enhance classification accuracy, compared to the case when only master data is also available.

Part III

Conclusions and Outlook

Chapter 8

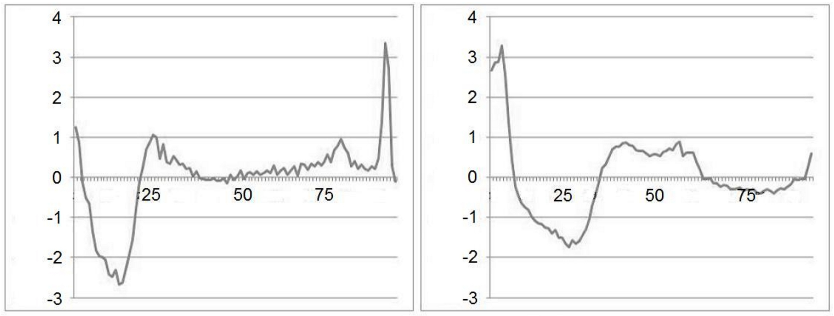
Outlook: Some Related Applications

In the previous sections, I proposed concepts, techniques and algorithms for time-series classification. The core ideas of them, however, can often be adapted for other domains too, therefore, together with my collaborators, we used some of the proposed techniques in various applications. In this section, these applications are surveyed without complete descriptions. References are given to the sources containing further information.

8.1 Classification of Electrocardiograph Signals

As described in Section 2.7.1, many recognition tasks related to electrocardiograph (ECG) signals can be formulated as time-series classification problems. Fast and accurate classification of ECG signals is substantial in many cases: e.g. in order to call the emergency service in advance (if the patient is monitored continuously at the intensive care department), or in order to allow for the thorough analysis of long ECG signals (an ECG signal recorded over several days contains several hundred-thousands heart beats). Therefore, in my joint work with Julia Koller [34], we applied INSIGHT (Chapter 4) for recognition problems related to ECG signals. In order to keep the presentation simple and compact, the results on time series datasets containing ECG signals were already reported in the tables and figures of Chapter 4.

Figure 8.1 provides more details by giving some examples for the classification of some ECG signals of the ECG200 dataset. Note that in most cases, both algorithms classified the signals correctly, the examples in Figure 8.1 aim at illustrating the differences. Also note that all variants of my approach (INSIGHT with different score functions) agreed on the classification of these signals.



True label	2 (abnormal)		2 (abnormal)	
Classification results		Correct classification?		Correct classification?
FastAWARD	1 (normal)	NO	1 (normal)	NO
INSIGHT	2 (abnormal)	YES	2 (abnormal)	YES

Figure 8.1: Some signals from the ECG200 dataset, their true class labels and the class labels output by the nearest neighbor classifier with $k = 1$ after selecting instances with FastAWARD and my approach, INSIGHT.

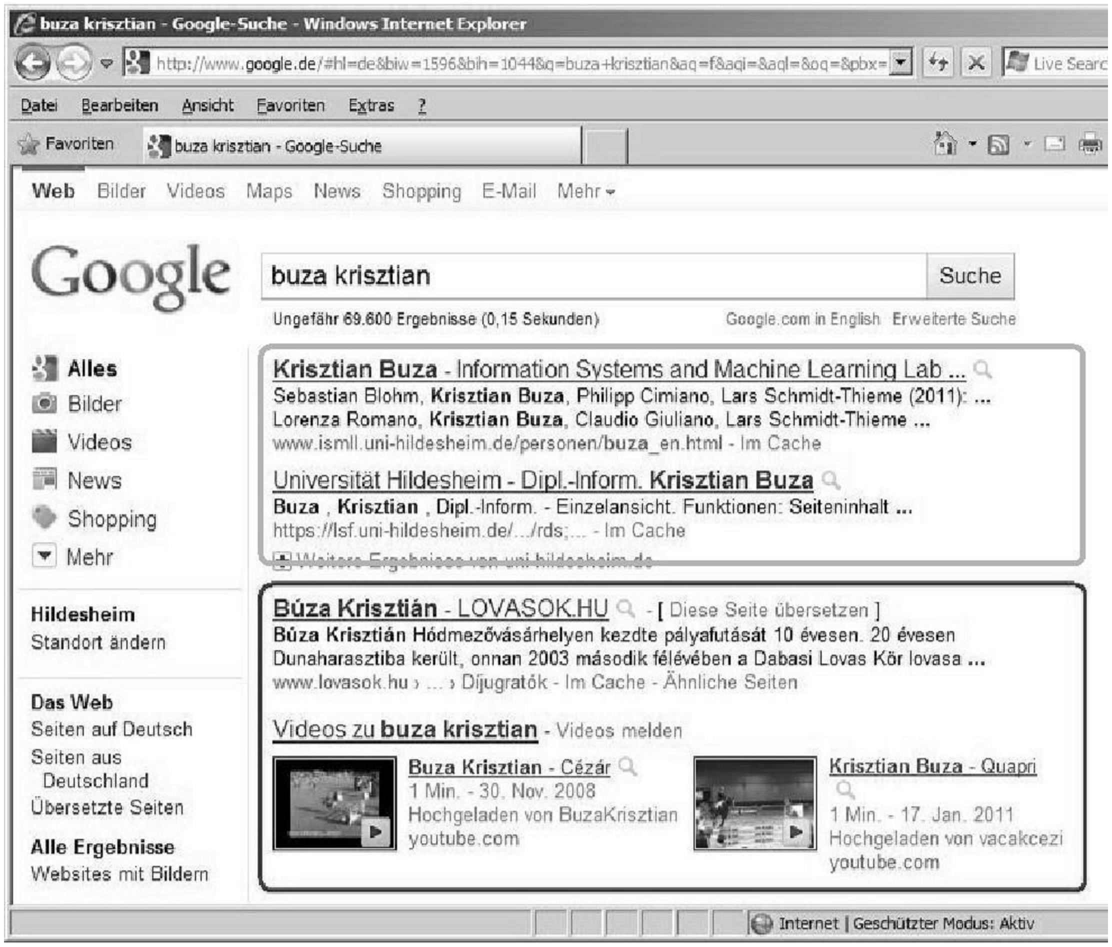
8.2 Extraction of Semantic Relations

A large portion of the information of websites is presented in form of natural language texts. Representing this information in a more uniform structure may be beneficial both for human users and machines by allowing e.g. faster and more accurate search and reasoning. In collaboration with Philipp Cimiano and Sebastian Blohm we were concerned with the task of *relation extraction*, which can be introduced by the means of an example as follows: consider the relation *bornInYear* between persons and the year they were born in, based on textual information (such as texts of web pages), we aim at finding tuples for this relation, i.e., pairs of person names and years they were born in.

Using generic sequential semi-continuous patterns, especially Algorithm 12 in Chapter 7, we developed a method for relation extraction that works for a wide variety of semantic relations such as *currencyOf*, *locatedIn*, *headquarteredIn*[1] and the aforementioned *bornInYear*. Using frequent pattern mining techniques, we recognized the textual patterns that characterize the occurrence of the relevant pieces of information, and used these textual patterns in order to extract the tuples for the given relations.

We compared generic sequential semi-continuous patterns with other, simpler pattern types such as continuous patterns without gaps. Our experiments showed that generic sequential semi-continuous patterns were able to improve both precision and recall, see [16] for the details.

1 Out of these relations, *currencyOf* refers to the connection between a state and its currency, *locatedIn* associates a city or town with the country where it is located, while *headquarteredIn* gives for a company the city of its central office.


buza krisztian - Google-Suche - Windows Internet Explorer
http://www.google.de/#hl=de&biw=1596&bih=1044&q=buza+krisztian&aq=f&aqi=&aql=&oq=&pbx=
Live Search
Datei Bearbeiten Ansicht Favoriten Extras ?
Favoriten
buza krisztian - Google-Suche
Web Bilder Videos Maps News Shopping E-Mail Mehr
Google
Suche
Google.com in English Erweiterte Suche
Alles
Bilder
Videos
News
Shopping
Mehr
Hildesheim
Standort ändern
Das Web
Seiten auf Deutsch
Seiten aus Deutschland
Übersetzte Seiten
Alle Ergebnisse
Websites mit Bildern
Internet | Geschützter Modus: Aktiv

buza krisztian

Ungefähr 69.600 Ergebnisse (0,15 Sekunden)

Krisztian Buza - Information Systems and Machine Learning Lab ...
Sebastian Blohm, Krisztian Buza, Philipp Cimiano, Lars Schmidt-Thieme (2011): ...
Lorenza Romano, Krisztian Buza, Claudio Giuliano, Lars Schmidt-Thieme ...
www.ismll.uni-hildesheim.de/personen/buza_en.html - Im Cache

Universität Hildesheim - Dipl.-Inform. Krisztian Buza
Buza , Krisztian , Dipl.-Inform. - Einzelansicht. Funktionen: Seiteninhalt ...
https://lsf.uni-hildesheim.de/.../rds;... - Im Cache

Búza Krisztián - LOVASOK.HU - [Diese Seite übersetzen]
Búza Krisztián Hódmezővásárhelyen kezdte pályafutását 10 évesen. 20 évesen Dunaharasztiba került, onnan 2003 második félévben a Dabasi Lovas Kör lovasa ...
www.lovasok.hu › ... › Díjugratók - Im Cache - Ähnliche Seiten

Videos zu buza krisztian - Videos melden

Buza Krisztian - Cézár
1 Min. - 30. Nov. 2008
Hochgeladen von BuzaKrisztian
youtube.com

Krisztian Buza - Quapri
1 Min. - 17. Jan. 2011
Hochgeladen von vacakcezi
youtube.com

Figure 8.2: Searching for a person name may result in web pages belonging to different persons because several person may have the same name.

8.3 Web People Search

Suppose we are searching for information about a person on the web. Common web search engines, such as Google or Yahoo, support keyword-based search, i.e., we can type the name of the person as keyword and the search engine returns a list of pages containing that name. The list is usually ranked by relevance: the most important web pages are shown first to the user.

Assuming that a person is uniquely identified by her or his name, the above technique faithfully solves the task of finding information about a person. In reality, however, often several persons have the same name. For example, if you are searching for Krisztian Buza in Google, you will find a person riding horses, who is surely different from me (see Figure 8.2). In such cases, presenting the search results in a more structured way, in particular, clustering the results according to persons, may be beneficial. These benefits are amplified in situations when one famous person *dominates* the web: due to the ranking, all the web pages shown at the beginning of the list may refer to the dominant person that could be different from the one that we are really searching for.

This problem was addressed in the Web People Search (WEPS) Workshop series.[2] [6] Together with my collaborators, Lorenza Romano and Claudio Guiliano, we participated at the 2nd WEPS Workshop and presented a possible solution for the above-described task of web page clustering according to persons. Our solution is similar to the technique for fusion of time series distance measures presented in Chapter 5: in the WEPS Workshop, we used a pairwise approach and trained a regression model in order to recognize whether or not two web pages refer to the same person. For a more details the Reader is referred to [152].

8.4 Clustering Images According to Events

Many web sites, e.g. Facebook, Flickr or Last.fm[3] allow their users to upload and browse images. These images are often associated with events, such as concerts, marriages or birthdays. We are concerned with *concrete events* having a well-defined location and time, such as the marriage of a *given* person on a *particular* day at a *well-defined* location.

An interesting task is to find which pictures belong to the same event, i.e., clustering the images according to events. Together with Philipp Cimiano, Timo Reuter and Lucas Drumond we worked on this problem. We enhanced the method of Becker et al. [12] by using a model that is similar to the one introduced in Chapter 5. We utilized the pairwise approach: we considered pairs of images and constructed a machine learned distance measure. This construction was similar to the training of the regression model $\mathcal{M}$ in Chapter 5. Then, we used this distance measure in the single linkage clustering algorithm. A detailed description of our approach can be found in [150].

Methods similar to the ones we used to cluster web pages according to persons and images according to events, i.e., methods based on machine learned distance measures, have recently been applied in various other domains, e.g. the construction of a gold standard for a medical image retrieval task [66].

8.5 Ontology induction

The so called *social tagging systems* or *folksonomies*, that are web pages, where users can label resources (pieces of music, video films, news articles, scientific publications, etc.) with keywords they favor, have become popular in the last decade.

2 http://nlp.uned.es/weps/index.php

3 http://www.facebook.com/ , http://www.flickr.com/ , http://www.lastfm.de/

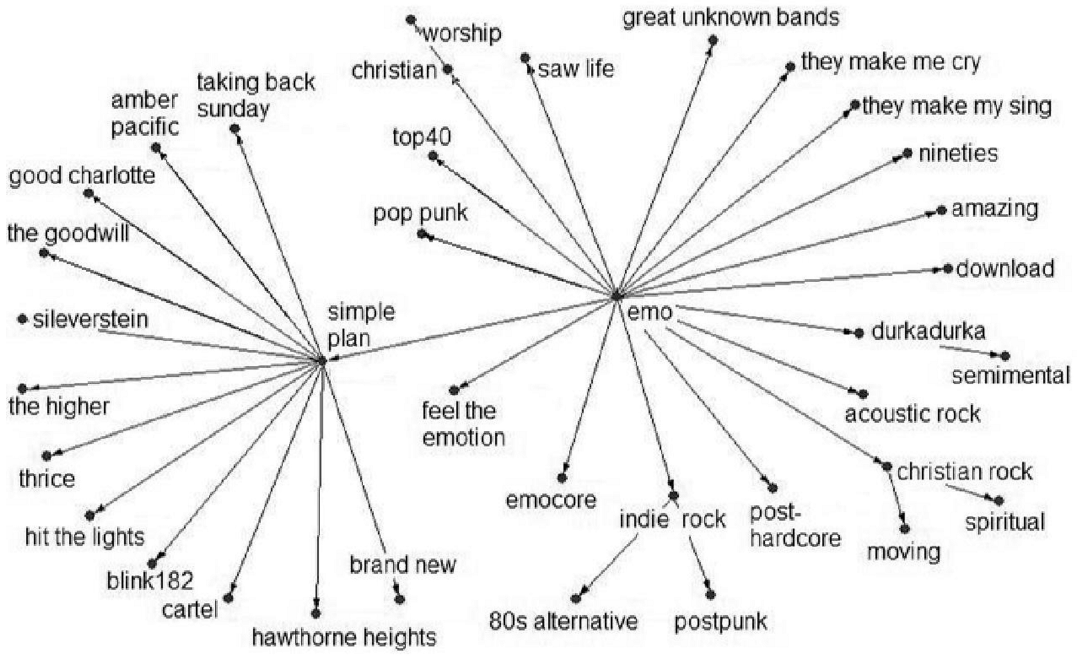


Figure 8.3: An excerpt of the induced ontology

Folksonomies may substantially contribute to more efficient search and information retrieval on the web because they provide a decentralized and shared environment where ordinary users play the role of voluntary annotators. But despite the compelling idea of folksonomies, its uncontrolled nature can bring problems, such as: synonymy, homonymy, and polysemy, which lowers the efficiency of content indexing and searching. Another problem is that folksonomy users are heterogeneous and thus have different levels of knowledge about a domain, what can lead to very personal tag assignments, thus lowering the potential for knowledge sharing. In this sense, folksonomies need some sort of expert intervention in order to make the shared vocabulary converge to a well agreed and therefore meaningful knowledge representation.

Together with Leandro Marinho we worked on the problem of inducing an ontology that integrates users' keywords into a domain expert taxonomy. We introduced a process of several stages, one of the core components of this process was build on *frequent itemset mining techniques*, that were similar to the ones used in Chapter 7. The details of our approach are described in [8], an excerpt of the induced ontology can be seen in Figure 8.3.

8.6 Analysis of Aeroplane Engine Vibration

An interesting recognition task regarding time series refers to the analysis of the vibration of aeroplane engines.

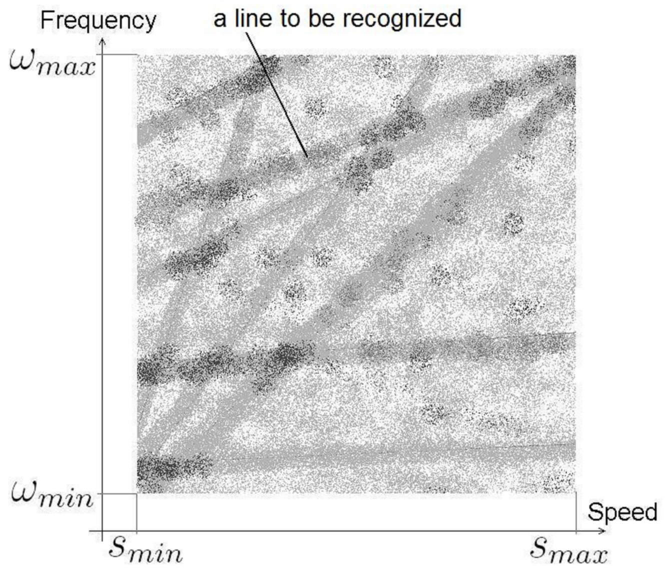


Figure 8.4: An illustrative example for a Campbell diagram

Vibration is the response of a system to an internal or external stimulus causing it to oscillate. In the aerospace domain, components of a jet engine are exposed to vibrations caused by unsteady forces, i.e., relative motions of rotating and non-rotating parts. Resonance can damage the system or its components, see e.g. [64], therefore, the analysis of vibration data is a fundamental issue in order to detect dangerous vibrations and resolve the problem of possible damage by redesign.

During engine tests, sensors are placed on the components in order to record vibration data. The vibration depends on the speed at which the engine currently operates. Therefore, this measurement is performed at many different speeds in order to obtain the vibration profile while the engine accelerates or decelerates. The recordings are, therefore, time series.

In order to allow for the recognition of the relevant information, instead of the raw representation of the data, a special representation, called *Campbell diagram* is used. For each considered speed, it is examined how intensively different frequencies are expressed in the vibration signal. The horizontal and vertical axes of a Campbell diagram correspond the speed and frequency respectively, while the color of a pixel encode the intensity of the vibration at the given speed and frequency. As shown in the example in Figure 8.4, Campell diagrams are often noisy. The data of the diagram in Figure 8.4 is fictive but imitates the characteristics of real Campbell diagrams we worked with.

Regions of intensive vibration are not located at random, but they can be described by parametric geometric curves, by lines in the most simple case. These lines are relevant for detecting dangerous vibrations. Therefore, together with my collaborators, Christine Preisach and Andre Busche, we aimed at identifying lines in Campbell diagrams using *Hough transformation* [59]. I refer to [35] for a more detailed description of our work.

Chapter 9

Conclusions and future work

The proceeding sections point out some pieces of the possible future work and conclude this book.

9.1 Extensions of the proposed techniques

The proposed techniques — IQ estimation, instance selection, fusion of similarity measures, the GRAMOFON ensemble framework and pattern mining — are not limited to the application domains presented in the experiments, they can be extended for other classification problems and algorithms.

In particular, one of the generic features of the proposed IQ estimation mechanism (Chapter 3) is that, in context of the k-NN classifier, IQ estimation can be employed for learning other parameters than k, such as parameters of the distance measure. More importantly, IQ estimation is not limited for the problem of k-NN classification of time series and regression problems over vector data (Section 3.5), since it can be used in combination with other classification algorithms and data types, whenever the complexity of the data requires an individualized approach. Therefore, future work involves the examination of IQ estimation in more general contexts, such as classification of structured data, texts, images or videos, and classification of multivariate and/or unevenly sampled time series.

Out of the above-mentioned domains and problems, in cases when the data is complex and therefore calculations are computationally expensive, e.g. recognition problems related to images and videos, *instance selection* is of major relevance. As future work, INSIGHT (Chapter 4) can be extended for these cases. Another promising direction is to combine instance selection with IQ estimation in order to achieve *fast* and *accurate* classification.

Fusion of distance measures (Chapter 5) could also be combined with IQ estimations: one could, for example, construct several fused distance measures and select the one to be used *individually*, i.e., dependent of the current time series.

The method presented for fusion of distance measures (Chapter 5) is not limited to time-series classification. As future work, one can examine fusion of distance measures in other contexts (e.g. more complex, structured data) or use a fused distance measure for *clustering* of time series. As the presented fusion approach works on data instance pairs, for large datasets, it may be beneficial to develop sampling strategies with special focus on the possibly imbalanced nature of the pair indicators. It would also be worth to examine in more depth, whether *all* the distance measures are worth to be fused or one should rather select a subset of them, because many of them could do better and faster than all [205].

As future work, one could use the GRAMOFON framework (Chapter 6) for the above-mentioned selection of a subset of distance measures as well as for deploying ensemble methods other than the presented ones.

Regarding pattern-based classification, one of the most challenging questions refers to the selection of patterns. When searching frequent patterns in time series, usually, a large number of patterns can be found. Many of them, however, are irrelevant for classification and/or their occurrences strongly correlate (as they are very similar patterns). The selection of a representative set of patterns – which can be used for classification – is a challenging task. While I focused on frequent patterns being correlated with one of the classes, in other applications, salient or unusual patterns (outliers) may be relevant. [190]

9.2 Some Open Questions

The presence of hubs, the phenomenon I described in Section 2.4.3 and exploited for speeding-up time-series classification in Chapter 4, was previously observed in scale-free networks [9]. In order to make use of hubs for recognition problems (such as time-series classification), one has to take the class labels into account. Based on the labels, Radovanovic et al. [140], [141], [142] defined *good* and *bad* hubs and made classification more accurate, while Tomasev et al. [174] explored hub-based clustering in case of conventional vector data. In Chapter 4, I used hubness in order to speed up time-series classification by instance selection. Research questions that directly arise are:

1. Besides the ones already explored, are there other, better ways of exploiting hubness for classification or clustering of time series?

2. Can hubness be exploited for other time-series classification problems, e.g. classification of multivariate or unevenly sampled time series or early classification?

The theory of complex networks has been studied intensively in the last decade, see e.g. [9], [50], and, aside from the presence of hubs, various other properties of natural and artificially created real-world networks have been identified, such as (i) the *small-world* property which states that two objects are likely to be connected via short paths, (ii) the *clique or clustering* property, according to which the neighbors of an object are likely to be directly linked, and (iii) properties describing dynamic behavior or evolution of complex networks. These observations suggest the research question whether other concepts of complex network theory can constitute foundations for new time-series classification problems:

3. Can characteristic properties of complex networks (beyond the presence of hubs) motivate new time-series classification methods either for conventional time-series classification or for its variants, such as the ones listed in the previous question?

Distributions similar to the ones in complex networks have been found to be characteristic for many phenomena of nature and social behavior [10], [53], [179]. Especially the presence of *bursts* [10] seems to be quite generic as it characterizes the dynamics of natural and artificial processes. Often, time series are generated by exactly these processes (in a time series, the value of a feature of such a process is recorded in consecutive moments of time). Therefore, one question is how bursts affect time series:

4. How do bursts influence the resulting time series? Do they generate characteristic patterns that could be used to identify classes, or do burst generate noise that should be eliminated in order to allow for more accurate classification?

Optimization of resource and energy consumption, reduction of waste-production and pollution are some of the most important challenges of the upcoming century. Based on graph theory, we are able to find e.g. the most environmental friendly route (the one with minimal fuel consumption and CO_2-emission) between to cities. Besides slowly changing structures (such as the road network) many processes are characterized by some kind of a temporal behavior and many *parameters* of static structures could change quickly (e.g. a the capacity of road could be limited due to weather conditions, traffic jam or an unforeseen accident). Therefore, time series theory in combination with graph theory, taking into account the recently observed properties of realistic networks, such as the ones mentioned above, as well as theoretical results, e.g. [91], could potentially contribute to the alleviation of the aforementioned problems.

The applications envisioned above require substantial extensions of the conventional time-series classification framework. At the same time, the currently existing technologies have not been fully exploited yet: reducing the braking distance of cars through the analysis of the drivers' brain waves (electroencephalography signals or EEG) is just one of the many emerging applications [82]. Furthermore, considerable effort has been devoted to the generation of natural language summaries from time series [164], and the usage of temporal features for various tasks such as web spam filtering [63].

9.3 Conclusions

In this book, I focused on the time-series classification problem that is the common theoretical background of many real-life recognition tasks. I successfully addressed two basic challenges of the problem: accuracy and execution time. Out of the proposed techniques, individual quality (IQ) estimation, fusion of distance measures, the GRAMOFON framework and motif-based classification aimed at making time-series classification more accurate while the proposed instance selection algorithm can substantially reduce recognition time. The proposed techniques are orthogonal to each other, i.e., in real recognition systems, they can be applied together in combination in order to achieve fast and accurate recognition.

In Chapter 3, I proposed a mechanism for individual quality (IQ) estimation. This considers a set of models (e.g. k-NN classifiers with different k values) and uses meta-level regression models that estimate the quality of each such model. In the framework of IQ estimation, I proposed three approaches: IQ-MAX, IQ-WV and IQ-Reg. My first proposed approach, IQ-MAX, selects separately for each time series the classifier with the maximum estimated quality. My second approach, IQ-WV combines the results of the primary-level classifiers according to the weighted voting schema, for which I used the estimated qualities as weights. I developed IQ-Reg for regression problem over vector data. All these approaches allow for adapting to characteristics that vary among the different regions in a dataset and therefore make the classification more accurate while adding only a small overhead in execution time.

In Chapter 4, I examined the problem of instance selection for speeding-up time-series classification. I introduced a principled framework for instance selection based on coverage graphs and the recently observed phenomenon of hubness. I proposed INSIGHT, a novel instance selection method for time series.

In Chapter 5, I focused on distance measures for time-series classification. I discussed which aspects of similarity they capture. As in complex applications several of these similarity aspects may be relevant simultaneously, I developed a generic framework which allowed fusion of various similarity measures in a principled way.

In Chapter 6, I proposed GRAMOFON, the General Model-selection Framework based on Networks. GRAMOFON supports stacking-based ensembles with appropriate model selection in case if large number of models are present. While doing so, special attention was paid to the selection of those models that compensate each other's errors.

In Chapter 7, I defined a novel motif type, which I named generalized semi-continuous motifs. I proposed efficient algorithms to discover them, and showed that motifs may improve the accuracy of time-series classification.

Besides time-series classification that includes various domains on its own, the proposed techniques (or their variants) can contribute to a wide variety of further applications. The interested Reader is referred to [8], [150] and [152].

Besides extensions of the proposed techniques, directions of future work could potentially include emerging domains such as integration of the presented results with complex network theory or reduction of braking distance of cars through the analysis of the drivers brain waves.

Bibliography

[1] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *Foundations of Data Organization and Algorithms*, volume 730 of *Lecture Notes in Computer Science (LNCS)*, pages 69–84. Springer, 1993.

[2] R. Agrawal, K.-I. Lin, H.S. Sawhney, and K. Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *Proceedings of 21th International Conference on Very Large Data Bases (VLDB)*, pages 490–501, 1995.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)*, pages 478–499. Morgan Kaufmann, 1994.

[4] D.W. Aha, D. Kibler, and M.K. Albert. Instance-Based Learning Algorithms. *Machine Learning*, 6(1):37–66, 1991.

[5] S. Arlot and A. Celisse. A Survey of Cross-Validation Procedures for Model Selection. *Statistics Surveys*, 4:40–79, 2010.

[6] J. Artiles, J. Gonzalo, and S. Sekine. The Semeval-2007 WEPS Evaluation: Establishing a Benchmark for the Web People Search Task. In *Proceedings of the 4th International Workshop on Semantic Evaluations (Semeval)*, 2007.

[7] M. Bacauskiene, A. Verikas, A. Gelzinis, and D. Valincius. A feature selection technique for generation of classification committees and its application to categorization of laryngeal images. *Pattern Recognition*, 42:645–654, 2009.

[8] L. Balby Marinho, K. Buza, and L. Schmidt-Thieme. Folksonomy-based collabulary learning. In *The Semantic Web - ISWC 2008*, volume 5318 of *Lecture Notes in Computer Science (LNCS)*, pages 261–276. Springer, Berlin/Heidelberg, 2008.

[9] A.L. Barabási. *Linked: How Everything Is Connected to Everything Else and What It Means for Business, Science, and Everyday Life*. Plume, 2003.

[10] A.L. Barabási. *Bursts: The Hidden Pattern Behind Everything We Do*. EP Dutton, 2010.

[11] E. Bauer and R. Kohavi. An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants. *Machine Learning*, 36(1):105–139, 1999.

[12] H. Becker, M. Naaman, and L. Gravano. Learning Similarity Metrics for Event Identification in Social Media. In *Proceedings of the 3rd ACM International Conference on Web Search and Data Mining*, pages 291–300. ACM, 2010.

[13] J. Bennett and S. Lanning. The Netflix Prize. In *Proceedings of KDD Cup and Workshop*, 2007.

[14] D. Berndt and J. Clifford. Using Dynamic Time Warping to Find Patterns in Time Series. In *AAAI-94 Workshop on Knowledge Discovery in Databases*, pages 229–248, 1994.

[15] S. Blohm. *Large-Scale Pattern-Based Information Extraction from the World Wide Web*. PhD thesis, Karlsruhe Institute of Technology (KIT), 2010.

[16] S. Blohm, K. Buza, P. Cimiano, and L. Schmidt-Thieme. Relation Extraction for the Semantic Web with Taxonomic Sequential Patterns. In Vijayan Sugumaran and Jon Atle Gulla, editors, *Applied Semantic Web Technologies*. Auerbach Publishers Inc., 2011.

[17] F. Bodon. A fast apriori implementation. In *Proceedings of the 1st IEEE ICDM Workshop on Frequent Item Set Mining Implementations (FIMI)*, volume 90 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2003.

[18] F. Bodon. A trie-based apriori implementation for mining frequent item sequences. In *Proceedings of the 1st International Workshop on Open Source Data Mining: Frequent Pattern Mining Implementations*, pages 56–65. ACM, 2005.

[19] C. Borgelt. Efficient implementations of apriori and eclat. In *Proceedings of the 1st IEEE ICDM Workshop on Frequent Item Set Mining Implementations (FIMI)*, volume 90 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2003.

[20] C. Borgelt. Recursion pruning for the apriori algorithm. In *Proceedings of the 2nd IEEE ICDM Workshop on Frequent Item Set Mining Implementations (FIMI)*, volume 126 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2004.

[21] G. Bortolan and J.L. Willems. Diagnostic ECG Classification Based on Neural Networks. *Journal of Electrocardiology*, 26:75, 1993.

[22] M.F. Botsch. *Machine Learning Techniques for Time Series Classification.* Cuvillier, 2009.

[23] H. Brighton and C. Mellish. Advances in Instance Selection for Instance-Based Learning Algorithms. *Data Mining and Knowledge Discovery*, 6(2):153–172, 2002.

[24] R. Bryll, R. Gutierrez-Osuna, and F. Quek. Attribute bagging: Improving accuracy of classifier ensembles by using random feature subsets. *Pattern Recognition*, 36(6):1291–1302, 2003.

[25] J. Buhler and M. Tompa. Finding Motifs Using Random Projections. *Journal of Computational Biology*, 9(2):225–242, 2002.

[26] D. I. Buza. Mozgásminta felismerése számítógép segítségével. In *Ifjú Kutatók Nemzetközi Konferenciája (magyarországi forduló)*, 2008.

[27] K. Buza, A. Nanopoulos, T. Horváth, and L. Schmidt-Thieme. GRAMOFON: General Model-Selection Framework Based on Networks. *Neurocomputing*, 2011.

[28] K. Buza, A. Nanopoulos, and L. Schmidt-Thieme. Graph-Based Model-Selection Framework for Large Ensembles. In *Proceedings of the 5th International Conference on Hybrid Artificial Intelligence Systems*, volume 6076 of *Lecture Notes in Computer Science/Lecture Notes in Artificial Intelligence (LNCS/LNAI)*, pages 557–564, Berlin/Heidelberg, 2010. Springer.

[29] K. Buza, A. Nanopoulos, and L. Schmidt-Thieme. Time-Series Classification Based on Individualised Error Prediction (Best Paper Award). In *Int'l. Conf. on Computational Science and Engineering (CSE).* IEEE, 2010.

[30] K. Buza, A. Nanopoulos, and L. Schmidt-Thieme. Fusion of Similarity Measures for Time Series Classification. In *Proceedings of the 6th International Conference on Hybrid Artificial Intelligence Systems*, volume 6679 of *Lecture Notes in Computer Science/Lecture Notes in Artificial Intelligence (LNCS/LNAI)*, pages 253–261, Berlin/Heidelberg, 2011. Springer.

[31] K. Buza, A. Nanopoulos, and L. Schmidt-Thieme. INSIGHT: Efficient and Effective Instance Selection for Time-Series Classification. In *15th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, volume 6635 of *Lecture Notes in Computer Science/Lecture Notes in Artificial Intelligence (LNCS/LNAI)*, pages 149–160. Springer, 2011.

[32] K. Buza, A. Nanopoulos, and L. Schmidt-Thieme. IQ Estimation for Accurate Time-Series Classification. In *Symposium Series on Computational Intelligence, Symposium on Computational Intelligence and Data Mining (CIDM).* IEEE, 2011.

[33] K. Buza, A. Nanopoulos, and L. Schmidt-Thieme. Individualized Error Estimation for Classification and Regression Models. In *34nd Annual Conference of the Gesellschaft für Klassifikation (GfKl 2010).* Springer, to appear.

[34] K. Buza, A. Nanopoulos, L. Schmidt-Thieme, and J. Koller. Fast Classification of Electrocardiograph Signals via Instance Selection. In *First IEEE Conference on Healthcare Informatics, Imaging, and Systems Biology (HISB)*, 2011.

[35] K. Buza, C. Preisach, A. Busche, L. Schmidt-Thieme, W.H. Leong, and M. Walters. Eigenmode identification in campbell diagrams. In *International Workshop on Machine Learning for Aerospace*, 2009.

[36] K. Buza and L. Schmidt-Thieme. Motif-Based Classification of Time Series with Bayesian Networks and SVMs. In *32nd Annual Conference of the Gesellschaft für Klassifikation, Studies in Classification, Data Analysis, and Knowledge Organization, Advances in Data Analysis, Data Handling and Business Intelligence*, pages 105–114. Springer, 2010.

[37] J. Caiado. *Classification and clustering of time series.* LAP Lambert Academic Publishing, 2010.

[38] K. Chakrabarti, E. Keogh, S. Mehrotra, and M. Pazzani. Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases. *Transactions on Database Systems (TODS)*, 27(2):188–228, 2002.

[39] K.P. Chan and W.C. Fu. Efficient Time Series Matching by Wavelets. In *Proceedings of the 15th International Conference on Data Engineering (ICDE)*, pages 126–133. IEEE Computer Society, 1999.

[40] L. Chen and R. Ng. On the Marriage of lp-Norms and Edit Distance. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB)*, pages 792–803, 2004.

[41] L. Chen, M.T. Özsu, and V. Oria. Robust and Fast Similarity Search for Moving Object Trajectories. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 491–502. ACM, 2005.

[42] Y. Chen, M.A. Nascimento, B.C. Ooi, and A.K.H. Tung. Spade: On Shape-Based Pattern Detection in Streaming Time Series. In *23rd International Conference on Data Engineering*, pages 786–795. IEEE, 2007.

[43] D.-Y. Chiu, Y.-H. Wu, and A.L.P. Chen. An efficient algorithm for mining frequent sequences by a new strategy without support counting. page 375. IEEE Computer Society, 2004.

[44] P. Christen. Automatic Record Linkage Using Seeded Nearest Neighbor and Support Vector Machine Classification. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 151–159. ACM, 2008.

[45] F. Cismondi, A.S. Fialho, S.M. Vieira, J.M.C. Sousa, S.R. Reti, M.D. Howell, and S.N. Finkelstein. Computational Intelligence Methods for Processing Misaligned, Unevenly Sampled Time Series Containing Missing Data. In *Symposium Series on Computational Intelligence, Symposium on Computational Intelligence and Data Mining (CIDM)*. IEEE, 2011.

[46] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2001.

[47] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2011.

[48] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547–553, 2009.

[49] B. Csatári and Z. Prekopcsák. Class-Based Attribute Weighting for Time Series Classification. In *Proceedings of the 14th International Student Conference on Electrical Engineering*, 2010.

[50] P. Csermely. *Weak links: Stabilizers of Complex Systems from Proteins to Social Networks*. Springer, 2006.

[51] I. Daubechies. *Ten Lectures on Wavelets*. Society for industrial and applied mathematics, 2004.

[52] L. Devroye, L. Györfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition*. Springer Verlag, 1996.

[53] Z. Dezso, E. Almaas, A. Lukacs, B. Racz, I. Szakadat, and A.L. Barabasi. The Dynamics of Information Access in the Online Media. *Bulletin of the American Physical Society*, 2005.

[54] T.G. Dietterich. An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization. *Machine Learning*, 40(2):139–157, 2000.

[55] T.G. Dietterich. Ensemble methods in machine learning. In *Multiple Classifier Systems*, volume 1857 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin / Heidelberg, 2000.

[56] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh. Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures. *Proceedings of the VLDB Endowment*, 1(2):1542–1552, 2008.

[57] C. Domeniconi and D. Gunopulos. Adaptive Nearest Neighbor Classification Using Support Vector Machines. *Neural Information Processing Systems (NIPS)*, 2001.

[58] C. Domeniconi, J. Peng, and D. Gunopulos. Locally Adaptive Metric Nearest-Neighbor Classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2002.

[59] R.O. Duda and P.E. Hart. Use of the Hough Transformation to Detect Lines and Curves in Pictures. *Communications of the ACM*, 15(1):11–15, 1972.

[60] N. Duffy and D. Helmbold. Boosting Methods for Regression. *Machine Learning*, 47(2):153–200, 2002.

[61] S. Dzeroski, I. Slavkov, V. Gjorgjioski, and J. Struyf. Analysis of Time series Data with Predictive Clustering Trees. In *Proceedings of the 5th International Workshop on Knowledge Discovery in Inductive Databases*, pages 47–58, 2006.

[62] D.R. Eads, D. Hill, S. Davis, S.J. Perkins, J. Ma, R.B. Porter, and J.P. Theiler. Genetic Algorithms and Support Vector Machines for Time Series Classification. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 4787, pages 74–85, 2002.

[63] M. Erdélyi and A.A. Benczúr. Temporal analysis for web spam detection: An overview. In *1st International Temporal Web Analytics Workshop (TWAW) in conjunction with the 20th International World Wide Web Conference in Hyderabad, India*, 2011.

[64] D.J. Ewins. *Modal Testing: Theory, Practice and Application.* Research Studies Press, Ltd., 2000.

[65] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast Subsequence Matching in Time-Series Databases. *ACM SIGMOD Record*, 23(2):419–429, 1994.

[66] J.S. Faruque, C.F. Beaulieu D.L. Rubin, and G. Tye S. Napel R.M. Summers J. Rosenberg, A. Kamaya. A Scalable Reference Standard of Visual Similarity for a Content-Based Image Retrieval System. In *First IEEE Conference on Healthcare Informatics, Imaging, and Systems Biology (HISB)*, 2011.

[67] P. Ferreira and P. Azevedo. Protein Sequence Classification Through Relevant Sequence Mining and Bayes Classifiers. *Progress in Artificial Intelligence*, pages 236–247, 2005.

[68] P.G. Ferreira, P.J. Azevedo, C.G. Silva, and R.M.M. Brito. Mining Approximate Motifs in Time Series. In *Proceedings of the 9th International Conference on Discovery Science*, 2006.

[69] A. Frank and A. Asuncion. UCI machine learning repository, http://archive.ics.uci.edu/ml , 2010.

[70] E. Frentzos, K. Gratsias, and Y. Theodoridis. Index-Based Most Similar Trajectory Search. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE)*, pages 816–825. IEEE, 2007.

[71] M.E. Futschik and B. Carlisle. Noise-Robust Soft Clustering of Gene Expression Time-Course Data. *Journal of Bioinformatics and Computational Biology*, 3(4):965–988, 2005.

[72] D. Garrett, D.A. Peterson, C.W. Anderson, and M.H. Thaut. Comparison of Linear, Nonlinear, and Feature Selection Methods for EEG Signal Classification. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 11(2):141–144, 2003.

[73] W. Gaul and L. Schmidt-Thieme. Mining Generalized Association Rules for Sequential and Path Data. In *Proceedings of the International Conference on Data Mining (ICDM)*, pages 593–596. IEEE, 2001.

[74] P. Geurts. Pattern Extraction for Time Series Classification. In *Principles of Data Mining and Knowledge Discovery*, pages 115–127. Springer, 2001.

[75] M.P. Griffin and J.R. Moorman. Toward the Early Diagnosis of Neonatal Sepsis and Sepsis-like Illness Using Novel Heart Rate Analysis. *Pediatrics*, 107(1):97, 2001.

[76] M. Grochowski and N. Jankowski. Comparison of Instance Selection Algorithms II. Results and Comments. volume 3070 of *Lecture Notes in Computer Science/Lecture Notes in Artificial Intelligence (LNCS/LNAI)*, pages 580–585, Berlin/Heidelberg, 2004. Springer-Verlag.

[77] C. Gruber, M. Coduro, and B. Sick. Signature Verification with Dynamic RBF Networks and Time Series Motifs. In *10th International Workshop on Frontiers in Handwriting Recognition*, 2006.

[78] D. Gunopulos and G. Das. Time Series Similarity Measures and Time Series Indexing. *SIGMOD Record*, 30(2):624, 2001.

[79] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 1–12. ACM, 2000.

[80] T. Hastie and R. Tibshirani. Discriminant Adaptive Nearest Neighbor Classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(6), 1996.

[81] T. Hastie, R. Tibshirani, and J.H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 5th Chapter*. Springer Verlag, 2009.

[82] S. Haufe, M.S. Treder, M.F. Gugler, M. Sagebaum, G. Curio, and B. Blankertz. EEG potentials predict upcoming emergency brakings during simulated driving. *Journal of Neural Engineering*, 8:056001, 2011.

[83] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1994.

[84] J. Hipp, A. Myka, R. Wirth, and U. Güntzer. A New Algorithm for Faster Mining of Generalized Association Rules. In *Proceedings of the 2nd European Symposium on Principles of Data Mining and Knowledge Discovery*, pages 74–82. Springer-Verlag, 1998.

[85] T.K. Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.

[86] F. Itakura. Minimum Prediction Residual Principle Applied to Speech Recognition. *Transactions on Acoustics, Speech and Signal Processing*, 23(1):67–72, 1975.

[87] M. Jahrer, A. Töscher, and R. Legenstein. Combining Predictions for Accurate Recommender Systems. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and Data Mining*, pages 693–702. ACM, 2010.

[88] A.K. Jain, R.C. Dubes, and C.C. Chen. Bootstrap Techniques for Error Estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (5):628–633, 2009.

[89] N. Jankowski and M. Grochowski. Comparison of Instance Selection Algorithms I. Algorithms Survey, 2004.

[90] K.L. Jensen, M.P. Styczynski, I. Rigoutsos, and G.N. Stephanopoulos. A Generic Motif Discovery Algorithm for Sequential Data. *Bioinformatics*, 22(1):21, 2005.

[91] Z. Kása. Hamiltonian cycles in de bruijn graphs. In *7th Joint Conference on Mathematics and Computer Science*, 2008.

[92] S.S. Keerthi, S.K. Shevade, C. Bhattacharyya, and K.R.K. Murthy. Improvements to platt's smo algorithm for svm classifier design. *Neural Computation*, 13(3):637–649, 2001.

[93] A. Kehagias and V. Petridis. Predictive Modular Neural Networks for Time Series Classification. *Neural Networks*, 10(1):31–49, 1997.

[94] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases. *ACM SIGMOD Record*, 30(2):151–162, 2001.

[95] E. Keogh and S. Kasetty. On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration. *Data Mining and Knowledge Discovery*, 7(4):349–371, 2003.

[96] E. Keogh, J. Lin, A. Fu, and H. Van Herle. Finding the Unusual Medical Time Series: Algorithms and applications. *Transactions on Information Technology in Biomedicine, Special Post-conference Issue "Mining Biomedical Data/CBMS2005"*, 2005.

[97] E. Keogh and M. Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In *Proceedings of the 4th International Conference of Knowledge Discovery and Data Mining*, pages 239–241. AAAI Press, 1998.

[98] E. Keogh and C.A. Ratanamahatana. Exact Indexing of Dynamic Time Warping. *Knowledge and Information Systems*, 7(3):358–386, 2005.

[99] E. Keogh, C. Shelton, and F. Moerchen. Workshop and challenge on time series classification, 2007.

[100] E. Keogh, L. Wei, X. Xi, S.H. Lee, and M. Vlachos. LB_Keogh Supports Exact Indexing of Shapes Under Rotation Invariance with Arbitrary Representations and Distance Measures. In *Proceedings of the 32nd International Conference on Very Large Data bases (VLDB)*, pages 882–893. VLDB Endowment, 2006.

[101] E.J. Keogh and M.J. Pazzani. Scaling up Dynamic Time Warping for Datamining Applications. In *6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 285–289. ACM, 2000.

[102] E.J. Keogh and M.J. Pazzani. Derivative Dynamic Time Warping. In *Proceedings of the 1st SIAM International Conference on Data Mining (SDM)*. SIAM, 2001.

[103] S. Kim and P. Smyth. Segmental hidden markov models with random effects for waveform modeling. *The Journal of Machine Learning Research*, 7:945–969, 2006.

[104] S.W. Kim, S. Park, and W.W. Chu. An Index-Based Approach for Similarity Search Supporting Time Warping in Large Sequence Databases. In *International Conference on Computer Communications and Networks (ICCCN)*, page 0607. IEEE Computer Society, 2001.

[105] T. Knorr. Identifying patients at risk: Mining dialysis treatment data. In *2nd German Japanese Symposium on Classification*, Berlin, 2006.

[106] T. Knorr. Motif discovery in multivariate time series and application to hemodialysis treatment data, 2006.

[107] F. Korn, H.V. Jagadish, and C. Faloutsos. Efficiently Supporting Ad Hoc Queries in Large Datasets of Time Sequences. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 289–300. ACM, 1997.

[108] V. Kunik, Z. Solan, S. Edelman, E. Ruppin, and D. Horn. Motif Extraction and Protein Classification. In *Proceedings of the Computational Systems Bioinformatics Conference (CSB)*, pages 80–85. IEEE Computer Society, 2005.

[109] V.I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. In *Soviet Physics Doklady*, volume 10, pages 707–710, 1966.

[110] G.-Z. Li and T.-Y. Liu. Feature selection for bagging of support vector machines. In *Pacific Rim International Conferences on Artificial Intelligence (PRICAI)*, volume 4099 of *LNCS*, pages 271–277. Springer, 2006.

[111] J. Lin, E. Keogh, S. Lonardi, and B. Chiu. A Symbolic Representation of Time Series, with Implications for Streaming Algorithms. In *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*. ACM, 2003.

[112] J. Lin, E. Keogh, L. Wei, and S. Lonardi. Experiencing SAX: a Novel Symbolic Representation of Time Series. *Data Mining and Knowledge Discovery*, 15(2):107–144, 2007.

[113] H. Liu and H. Motoda. On issues of Instance Selection. *Data Mining and Knowledge Discovery*, 6(2):115–130, 2002.

[114] I.L. MacDonald and W. Zucchini. *Hidden Markov and Other Models for Discrete-Valued Time Series.* Chapman & Hall London, 1997.

[115] S.G. Mallat. *A Wavelet Tour of Signal Processing.* Academic Press, 1999.

[116] S. Marcel and J.R. Millan. Person Authentication using Brainwaves (EEG) and Maximum a Posteriori Model Adaptation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29:743–752, 2007.

[117] D.D. Margineantu and T.G. Dierrerich. Pruning adaptive boosting. In *Proceedings of the International Conference on Machine Learning (ICML).*

[118] R. Martens and L. Claesen. On-line Signature Verification by Dynamic Time-Warping. In *Proceedings of the 13th International Conference on Pattern Recognition*, volume 3, pages 38–42. IEEE, 1996.

[119] F. Melgani and Y. Bazi. Classification of electrocardiogram signals with support vector machines and particle swarm optimization. *IEEE Transactions on Information Technology in Biomedicine*, 12(5):667–677, 2008.

[120] A.M. Molinaro, R. Simon, and R.M. Pfeiffer. Prediction Error Estimation: a Comparison of Resampling Methods. *Bioinformatics*, 21(15):3301, 2005.

[121] F. Mörchen. Time series feature extraction for data mining using dwt and dft. Technical report, 2003.

[122] M.D. Morse and J.M. Patel. An Efficient and Accurate Method for Evaluating Time Series Similarity. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 569–580. ACM, 2007.

[123] A. Nanopoulos, R. Alcock, and Y. Manolopoulos. Feature-based classification of time-series data. In *Information Processing and Technology*, pages 49–61. Nova Science Publishers, Inc., 2001.

[124] R. Niels. *Dynamic Time Warping: an Intuitive Way of Handwriting Recognition?* Master Thesis, Radboud University Nijmegen, The Netherlands, 2004.

[125] R.T. Olszewski. *Generalized Feature Extraction for Structural Pattern Recognition in Time-Series data.* PhD thesis, School of Computer Science, Carnegie Mellon University, 2001.

[126] S. Ougiaroglou, A. Nanopoulos, A. Papadopoulos, Y. Manolopoulos, and T. Welzer-Druzovec. Adaptive k-Nearest-Neighbor Classification Using a Dynamic Number of Nearest Neighbors. In *Advances in Databases and Information Systems*, pages 66–82. Springer, 2007.

[127] P. Patel, E. Keogh, J. Lin, and S. Lonardi. Mining Motifs in Massive Time Series Databases. In *Proceedings of the International Conference on Data Mining (ICDM)*, pages 370–377. IEEE, 2002.

[128] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.C. Hsu. Mining Sequential Patterns by Pattern-Growth: The Prefixspan Approach. *Transactions on Knowledge and Data Engineering*, pages 1424–1440, 2004.

[129] Y. Peng. A novel ensemble machine learning for robust microarray data classification. *Computers in Biology and Medicine*, 36(6):553–573, 2006.

[130] V. Petridis and A. Kehagias. *Predictive modular neural networks: applications to time series*, volume 466 of *The Springer International Series in Engineering and Computer Science*. Springer Netherlands, 1998.

[131] R. Plamondon and S.N. Srihari. Online and Off-line Handwriting Recognition: a Comprehensive Survey. *Pattern Analysis and Machine Intelligence*, 22(1):63–84, 2002.

[132] J. Platt. Fast training of support vector machines using sequential minimal optimization. *Advances in Kernel Methods - Support Vector Learning*, 1998.

[133] I. Pramudiono and M. Kitsuregawa. FP-tax: Tree Structure Based Generalized Association Rule Mining. In *Proceedings of the 9th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 60–63. ACM, 2004.

[134] C. Preisach. *Relational Semi-Supervised Classification Using Multiple Relations*. PhD thesis, University of Hildesheim, Germany, 2010.

[135] C. Preisach and L. Schmidt-Thieme. Ensembles of relational classifiers. *Knowl. Inf. Syst.*, 14:249–272, 2008.

[136] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986. 10.1023/A:1022643204877.

[137] J.R. Quinlan. Combining instance-based and model-based learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, page 236, 1993.

[138] J.R. Quinlan. Bagging, Boosting, and C4.5. In *Proceedings of the National Conference on Artificial Intelligence*, pages 725–730, 1996.

[139] L. Rabiner and B. Juang. An Introduction to Hidden Markov Models. *ASSP Magazine*, 3(1):4–16, 1986.

[140] M. Radovanović, A. Nanopoulos, and M. Ivanović. Nearest Neighbors in High-Dimensional Data: The Emergence and Influence of Hubs. In *Proceedings of the 26rd International Conference on Machine Learning (ICML)*, pages 865–872. ACM, 2009.

[141] M. Radovanović, A. Nanopoulos, and M. Ivanović. Hubs in Space: Popular Nearest Neighbors in High-Dimensional Data. *The Journal of Machine Learning Research (JMLR)*, 11:2487–2531, 2010.

[142] M. Radovanović, A. Nanopoulos, and M. Ivanović. Time-Series Classification in Many Intrinsic Dimensions. In *Proceedings of the 10th SIAM International Conference on Data Mining (SDM)*, pages 677–688, 2010.

[143] C.A. Ratanamahatana and E. Keogh. Everything You Know about Dynamic Time Warping is Wrong. In *SIGKDD International Workshop on Mining Temporal and Sequential Data*, 2004.

[144] C.A. Ratanamahatana and E. Keogh. Making Time-Series Classification More Accurate using Learned Constraints. In *SIAM International Conference on Data Mining*, pages 11–22, 2004.

[145] C.A. Ratanamahatana and E. Keogh. Three Myths about Dynamic Time Warping Data Mining. In *Proceedings of SIAM International Conference on Data Mining (SDM)*. SIAM, 2005.

[146] T.M. Rath and R. Manmatha. Word Image Matching using Dynamic Time Warping. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages II–521. IEEE, 2003.

[147] M.A. Redmond and A. Baveja. A data-driven software tool for enabling cooperative information sharing among police departments. *European Journal of Operational Research*, 141:660–678, 2002.

[148] B. Reiz and L. Csató. Bayesian network classifier for medical data analysis. *International Journal of Computers Communications & Control*, 4(1):65–72, 2009.

[149] S. Rendle and L. Schmidt-Thieme. Object Identification with Constraints. In *Proceedings of the 6th International Conference on Data Mining (ICDM)*, pages 1026–1031. IEEE Computer Society, 2006.

[150] T. Reuter, P. Cimiano, L. Drumond, K. Buza, and L. Schmidt-Thieme. Scalable event-based clustering of social media via record linkage techniques. In *Fifth International AAAI Conference on Weblogs and Social Media*, 2011.

[151] I. Rish. An Empirical Study of the Naive Bayes classifier. In *17th International Joint Conference on Artificial Intelligence (IJCAI), Workshop on Empirical Methods in Artificial Intelligence*, 2001.

[152] L. Romano, K. Buza, C. Giuliano, and L. Schmidt-Thieme. XMedia: Web People Search by Clustering with Machinely Learned Similarity Measures. In *2nd Web People Search Evaluation Workshop (WePS-2), 18th World Wide Web Conference (WWW)*, 2009.

[153] H. Sakoe and S. Chiba. Dynamic Programming Algorithm Optimization for Spoken Word Recognition. *Acoustics, Speech and Signal Processing*, 26(1):43–49, 1978.

[154] Y. Sakurai, M. Yoshikawa, and C. Faloutsos. FTW: Fast Similarity Search under the Time Warping Distance. In *Proceedings of the 24th SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 326–337. ACM, 2005.

[155] S. Salvador and P. Chan. Toward Accurate Dynamic Time Warping in Linear Time and Space. *Intelligent Data Analysis*, 11(5):561–580, 2007.

[156] L. Schmidt-Thieme. *Assoziationsregel-Algorithmen für Daten mit komplexer Struktur*. Peter Lang Verlag, 2003.

[157] B. Schuller, S. Reiter, R. Muller, M. Al-Hames, M. Lang, and G. Rigoll. Speaker Independent Speech Emotion Recognition by Ensemble Classification, 2005.

[158] J. Shieh and E. Keogh. iSAX: Indexing and Mining Terabyte Sized Time Series. In *Proceeding of the 14th International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 623–631. ACM, 2008.

[159] D.L. Shrestha and D.P. Solomatine. Experiments with AdaBoost.RT, an Improved Boosting Scheme for Regression. *Neural Computation*, 18(7):1678–1710, 2006.

[160] J. Sill, G. Takacs, L. Mackey, and D. Lin. Feature-Weighted Linear Stacking. *arXiv (http://arxiv.org/abs/0911.0460)*, 911.

[161] A.J. Smola and B. Scholkopf. A tutorial on support vector regression, 1998.

[162] S. Sonnenburg, G. Ratsch, C. Schafer, and B. Scholkopf. Large Scale Multiple Kernel Learning. *The Journal of Machine Learning Research*, 7:1531–1565, 2006.

[163] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the 5th International Conference on Extending Database Technology (EDBT)*, pages 3–17, 1996.

[164] Somayajulu G. Sripada, Ehud Reiter, Jim Hunter, and Jin Yu. Generating english summaries of time series data using the gricean maxims. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pages 187–196, New York, NY, USA, 2003. ACM.

[165] K. Sriphaew and T. Theeramunkong. A New Method for Finding Generalized Frequent Itemsets in Generalized Association Rule Mining. In *Proceedings of the 7th International Symposium on Computers and Communications (ISCC)*, pages 1040–1045. IEEE Computer Society, 2002.

[166] K. Sriphaew and T. Theeramunkong. Fast Algorithms for Mining Generalized Frequent Patterns of Generalized Association Rules. *Transactions on Information and Systems*, E87–D(3):761–770, 2004.

[167] Z. Syed and C.-C. Chia. Computationally generated cardiac biomarkers: Heart rate patterns to predict death following coronary attacks. In *SIAM International Conference on Data Mining*. 2011.

[168] P. Sykacek and S. Roberts. Bayesian Time Series Classification. *Advances in Neural Information Processing Systems*, 2:937–944, 2002.

[169] G. Takács, I. Pilászy, B. Németh, and D. Tikk. Matrix Factorization and Neighbor Based Algorithms for the Netflix Prize Problem. In *Proceedings of the 2008 ACM Conference on Recommender systems*, pages 267–274. ACM, 2008.

[170] G. Takács, I. Pilászy, B. Németh, and D. Tikk. Scalable Collaborative Filtering Approaches for Large Recommender Systems. *The Journal of Machine Learning Research (JMLR)*, 10:623–656, 2009.

[171] A.C. Tan and D. Gilbert. Ensemble machine learning on gene expression data for cancer classification, 2003.

[172] D. Tikk, P. Thomas, P. Palaga, J. Hakenberg, and U. Leser. A Comprehensive Benchmark of Kernel Methods to Extract Protein-Protein Interactions from Literature. *PLoS Computational Biology*, 6(7):2–8, 2010.

[173] K.M. Ting and I.H. Witten. Stacked Generalization: When Does it Work? In *15th Intertnational Joint Conference on Artifical Intelligence-Vol. 2*, pages 866–871. Morgan Kaufmann, 1997.

[174] N. Tomasev and M. Ivanovic M. Radovanovic, D. Mladenic. The Role of Hubness in Clustering High-Dimensional Data. In *15th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, volume 6634 of *Lecture Notes in Computer Science/Lecture Notes in Artificial Intelligence (LNCS/LNAI)*, pages 183–195. Springer, 2011.

[175] G. Tsoumakas, L. Angelis, and I.P. Vlahavas. Selective fusion of heterogeneous classifiers. *Intelligent Data Analysis*, 9:511–525, 2005.

[176] G. Tsoumakas, I. Katakis, and I.P. Vlahavas. Effective voting of heterogeneous classifiers. In *Proceedings of the 11th European Conference on Machine Learning (ECML)*, volume 3201, pages 465–476, 2004.

[177] A. Tsymbal and D.W. Patterson S. Puuronen. Ensemble feature selection with simple bayesian classification. *Information Fusion*, 4:87–100, 2003.

[178] K. Tuda, G. Rätsch, S. Mika, and K.R. Müller. Learning to Predict the Leave-One-Out Error of Kernel Based Classifiers. volume 2130 of *LNCS*, pages 331–338. Springer Verlag, 2001.

[179] A. Vazquez, B. Rácz, A. Lukács, and A.L. Barabási. Impact of Non-Poissonian Activity Patterns on Spreading Processes. *Physical review letters*, 98(15):158702, 2007.

[180] E. Vidal, F. Casacuberta, J.M. Benedi, M.J. Lloret, and H. Rulot. On the Verification of Triangle Inequality by Dynamic Time-Warping Dissimilarity Measures. *Speech Communication*, 7(1):67–79, 1988.

[181] M. Vlachos, D. Gunopoulos, and G. Kollios. Discovering Similar Multidimensional Trajectories. In *Proceedings of the International Conference on Data Engineering (ICDE)*, page 0673. IEEE Computer Society, 2002.

[182] M. Vlachos, D. Gunopulos, and G. Das. Rotation Invariant Distance Measures for Trajectories. In *Proceedings of the 10th International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 707–712.

[183] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. Keogh. Indexing Multi-Dimensional Time-Series with Support for Multiple Distance Measures. In *Proceedings of the 9th International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 216–225. ACM, 2003.

[184] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. Keogh. Indexing Multidimensional Time-Series. *The VLDB Journal*, 15(1):1–20, 2006.

[185] X. Wang, L. Ye, E. Keogh, and C. Shelton. Annotating Historical Archives of Images. In *Proceedings of the 8th ACM/IEEE-CS Joint Conference on Digital Libraries*, pages 341–350. ACM, 2008.

[186] G.I. Webb, J.R. Boughton, and Z. Wang. Not so naive bayes: Aggregating one-dependence estimators. *Machine Learning*, 58(1):5–24, 2005.

[187] D. Wettschereck and T.G. Dietterich. Locally Adaptive Nearest Neighbor Algorithms. *Advances in Neural Information Processing Systems*, pages 184–184, 1994.

[188] I.H. Witten and E. Frank. Data Mining: Practical Machine Learning Tools and Techniques. 2011.

[189] D.H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

[190] J. Woodbridge, M. Lan, M. Sarrafzadeh, and A. Bui. Salient Segmentation of Medical Time Series. In *First IEEE Conference on Healthcare Informatics, Imaging, and Systems Biology (HISB)*, 2011.

[191] M. Wozniak. A Hybrid Decision Tree Training Method Using Data Streams. *Knowledge and Information Systems*, pages 1–13, 2010.

[192] M. Wozniak and M. Zmyslony. Designing Fusers on the Basis of Discriminants – Evolutionary and Neural Methods of Training. *Hybrid Artificial Intelligence Systems (HAIS)*, pages 590–597, 2010.

[193] Y.L. Wu, D. Agrawal, and A. El Abbadi. A Comparison of DFT and DWT Based Similarity Search in Time-Series Databases. In *Proceedings of the 9th International Conference on Information and Knowledge Management*, pages 488–495. ACM, 2000.

[194] X. Xi, E. Keogh, C. Shelton, L. Wei, and C.A. Ratanamahatana. Fast Time Series Classification Using Numerosity Reduction. In *Proceedings of the 23rd International Conference on Machine Learning (ICML)*, pages 1033–1040. ACM, 2006.

[195] Z. Xing, J. Pei, P.S. Yu, and K. Wang. Extracting Interpretable Features for Early Classification of Time Series. In *SIAM International Conference on Data Mining (SDM)*, 2011.

[196] Y. Yang, G.I. Webb, J. Cerquides, K.B. Korb, J. Boughton, and K.M. Ting. To select or to weigh: A comparative study of linear combination schemes for superparent-one-dependence estimators. *IEEE Transactions on Knowledge and Data Engineering*, 19:1652–1665, 2007.

[197] D. Yankov, E. Keogh, J. Medina, B. Chiu, and V. Zordan. Detecting Time Series Motifs under Uniform Scaling. In *Proceedings of the 13th International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 844–853. ACM, 2007.

[198] L. Ye and E. Keogh. Time series shapelets: a novel technique that allows accurate, interpretable and fast classification. *Data Mining and Knowledge Discovery*, 22:149–182, 2011. 10.1007/s10618-010-0179-5.

[199] B.K. Yi and C. Faloutsos. Fast Time Sequence Indexing for Arbitrary Lp Norms. In *Proceedings of the 26th International Conference on Very Large Data Bases*, pages 385–394, 2000.

[200] B.K. Yi, H.V. Jagadish, and C. Faloutsos. Efficient Retrieval of Similar Time Sequences under Time Warping. In *Proceedings of the 14th International Conference on Data Engineering (ICDE)*, pages 201–208. IEEE, 1998.

[201] G.P. Zhang and V.L. Berardi. Time Series Forecasting with Neural Network Ensembles: an Application for Exchange Rate Prediction. *Journal of the Operational Research Society*, 52(6):652–664, 2001.

[202] Z.-H. Zhou, Y. Jiang, Y.-B. Yang, and S.-F. Chen. Lung cancer cell identification based on artificial neural network ensembles. *Artificial Intelligence in Medicine*, 24(1):25–36, 2002.

[203] Z.-H. Zhou and W. Tang. Selective ensemble of decision trees. In *Rough Sets, Fuzzy Sets, Data Mining and Granular Computing*, pages 476–483. Springer, 2003.

[204] Z.-H. Zhou, J. Wu, and W. Tang. Ensembling neural networks: Many could be better than all. *Artificial Intelligence*, 137(1–2):239–263, 2002.

[205] Z.H. Zhou, J. Wu, and W. Tang. Ensembling Neural Networks: Many Could Be Better Than All. *Artificial Intelligence*, 137(1-2):239–263, 2002.