Optimization in Semi-Supervised Classification of Multivariate Time Series

Pro Gradu
Timo Lintonen
2192796
Research Unit of Mathematical Sciences
University of Oulu
Fall 2018

Contents

Nomenclature

List of Abbreviations

kNN	k-nearest neighbor
Auslan	Australian sign language
CBD-GA	Class boundary detection using graphical analysis
CLeVer	descriptive Common principal component Loading-based Variable subset selection method
CPC	Common principal component
CPCA	Common principal component analysis
DTW	Dynamic time warping
ED	Euclidean distance
EM	Expectation maximization
i.i.d.	Independent and identically distributed
KKT	Karush-Kuhn-Tucker (condition)
LOF	Local outlier factor
MTS	Multivariate time series
OCC	One-class classification
PCA	Principal component analysis
PCA-SF	Principal component analysis similarity factor
PIP	Perceptually important point
PU	Positive-unlabelled (learning)
RBF	Radial basis function
RW	Ratanamahatana-Wanichsan (criterion)
S-C	Sakoe-Chiba (window)
SC	Stopping criterion

SSL	Semi-supervised learning
SVC2004	The first international signature verification contest
SVDD	Support vector data description
UTS	Univariate time series

Notations

^{T}	Transpose
$\#$	Cardinality of a set
$\boldsymbol{s}^{(k)}$	State
$\boldsymbol{u}^{(k)}$	Action
$\boldsymbol{W}^{(k)}$	Random disturbance
$\boldsymbol{X}$	Matrix, time series
$\boldsymbol{x}$	Vector
$\boldsymbol{X}^{(j)}$	One feature of a multivariate time series
λ_i	Lagrangian multiplier for an inequality constraint
$\mathbb{E}$	Expected value
$\mathbb{P}$	Probability function
$\mathcal{L}$	Lagrangian function
$\mathcal{X}$	Time series database
μ_j	Lagrangian multiplier for an equality constraint
μ_k	Decision rule
∇	Gradient
π	Policy
$\|\cdot\|$	Euclidean norm
$\widetilde{J}$	Forward dynamic programming algorithm cost function

$a^k_{\boldsymbol{s}^{(k)} \to \boldsymbol{s}^{(k+1)}}$	Transition cost
$B(\boldsymbol{x}, \epsilon)$	Ball with center $\boldsymbol{x}$ and radius ϵ
D, dom	Domain set and domain operator
$d(\cdot, \cdot)$	Distance function
d^*, p^*	Dual and primal problem optimal values
f, q	Objective functions for primal and dual problems
f_k	Function defining the system evolution
g_i	Function defining an inequality constraint
g_k	Cost functional
h_j	Function defining an equality constraint
J	Dynamic programming cost function
m	Number of dimensions
N	Number of time series
n	Number of observations, number of inequality constraints
N_U	Number of unlabelled instances
S	Set
t_i	Time stamp
$U_k(\boldsymbol{s}^{(k)})$	The set of admissible actions
W	Warping path
x	Scalar
aff	Affine hull
cl	Closure
int	Interior
relint	Relative interior
Std	Standard deviation

Introduction

People are increasingly interested in monitoring their own lives. Some are interested in their blood pressure, some about the amount and quality of sleep they get and some about the daily consumption of calories. To ease their lives, people use embedded devices that monitor their surroundings using sensors.

In the every day business, there are many things to monitor. Shops record surveillance videos to aid in the fight against robberies. Steel producers monitor the quality of the steel. Investment bankers are interested in the prices of the stocks in the market.

One thing in common in all of these processes is that they produce time series type of data. Time series do not satisfy the assumption of the data points being independent and identically distributed (i.i.d.). Most models created for non-temporal data are build on this particular assumption. The vast number of sources of time series motivates us to study models that are able to process time series type of data.

The continuously expanding masses of data have created a demand for machine learning. Some tasks that have been automated successfully are classification and clustering. In many domains, these tasks can be automated to a degree that the automation increases business value drastically.

In this thesis, I will study the semi-supervised classification of multivariate time series. The main argument is that in semi-supervised time series classification the use of temporal models is justified over the non-temporal models. In Section 1 I will carefully go through the mathematics behind two successful algorithms: support vector data description and dynamic time warping. Section 1 is mostly based on the existing literature in the books [7] and [6]. In Section 2, the special characteristics of the time series are discussed, especially the distance calculations on time series. The Section 3 is dedicated to one-class classification and positive-unlabelled learning. In Section 4 I will test the models empirically on three time series datasets. Sections 5 and 6 conclude this thesis on observation of the results and the possible future research.

In this thesis, I will present a novel proof to Theorem 2.9 in Section 2.3.3 and a new method for semi-supervised learning of time series called Peak evaluation using perceptually important points in Section 3.4.3. In Section 2.3.3, I will also prove Lemmas 2.7 and 2.8 that are used in the proof to Theorem 2.9.

1 Optimization

Mathematical optimization is a field that specializes in finding the optimal value of an objective function under some constraints. The optimum is the minimum value of a cost function or the maximum value of a utility function. The optimum was defined as the minimum or the maximum already in the ancient Greek around 500 BC when the philosophers were interested in finding optima in fields such as physics, astronomy and the quality of human life and government [32].

In machine learning optimization is in an important role. Many models are made flexible by introducing some hyper parameters. In some models these parameters may be chosen automatically to best fit the data at hand. This is achieved, for example, by minimizing a loss function, such as least squares or log-loss.

In this thesis, two algorithms are presented that rely on optimization. The support vector data description is heavily based on convex programming. The dynamic time warping algorithm uses dynamic programming.

1.1 Non-linear Programming

Let us first fix the notation in the following definition.

Definition 1.1. Let $f : X \to \mathbb{R}$ with $X \subset \mathbb{R}^m$ be a function to be minimized. Then a non-linear program is

$$\begin{aligned} \underset{\boldsymbol{x} \in X}{\text{minimize}} \quad & f(\boldsymbol{x}) \\ \text{subject to} \quad & g_i(\boldsymbol{x}) \leq 0, \quad i = 1, \ldots, n, \\ & h_j(\boldsymbol{x}) = 0, \quad j = 1, \ldots, p. \end{aligned} \tag{1.1}$$

The function f is an objective function. The functions g_i and h_j define inequality constraints and equality constraints, respectively. There is also the constrain $\boldsymbol{x} \in X$. This is called an *abstract constraint*. The constraints define a *feasible region*. That is the set of all the points satisfying all the constraints. Throughout this thesis, the objective function f and the inequality constraints defining functions $g_1, \ldots, g_n$ are assumed to be differentiable.

The non-linear program in Equation (1.1) is defined over some *domain*. The domain of the optimization problem is a set in which the objective function f and all the constraint functions g_i and h_i are defined. More formally, the domain D of the program in Equation (1.1) is

$$D = \text{dom } f \cap \left(\bigcap_{i=1}^{n} \text{dom } g_i \right) \cap \left(\bigcap_{j=1}^{m} \text{dom } h_j \right). \tag{1.2}$$

Historically, mathematical optimization has been divided into linear and non-linear programming. This thesis uses a modern, more meaningful division into convex and non-convex programming [50]. The next section is dedicated to convex programming.

1.1.1 Convex Programming

Convex programming holds many desirable properties. One of them is the fact that any locally optimal solution to a convex program is also global [50]. For example, the optimization of the k-means algorithm [23] minimizes a non-convex objective function that has multiple local minima. Standard procedure when running a k-means algorithm is to run it multiple times with different initial starting points. Even that, however, does not guarantee global optimum, and the result is usually a sub-optimal model.

A convex program is a special case of the non-linear program in Definition 1.1. The two elementary concepts of convex programming are a *convex set* and a *convex function*. These concepts are illustrated in Figure 1. A set S is convex if for any pair of points in the set S the convex combination of the points is in the set S. More formally,

Definition 1.2. The set S is convex if $t\boldsymbol{x}_1 + (1-t)\boldsymbol{x}_2 \in S$ for all $\boldsymbol{x}_1, \boldsymbol{x}_2 \in S$ and for all $t \in [0, 1]$.

There are two special cases to the convex sets. These are the empty set and the sets with only one element $\{\boldsymbol{x}\} \subset \mathbb{R}^m$ [50].

Definition 1.3. Let the set $X \subset \mathbb{R}^m$ be convex. A function $f : X \to \mathbb{R}$ is convex if

$$f(t\boldsymbol{x}_1 + (1-t)\boldsymbol{x}_2) \leq tf(\boldsymbol{x}_1) + (1-t)f(\boldsymbol{x}_2) \tag{1.3}$$

for all $\boldsymbol{x}_1, \boldsymbol{x}_1 \in X$ and for all $t \in [0, 1]$.

The link between a convex set and a convex function is that for any convex function g the set $S = \{\boldsymbol{x} \in \mathbb{R}^m \mid g(\boldsymbol{x}) \leq 0\}$ is convex. Let $\boldsymbol{x}_1, \boldsymbol{x}_2 \in S$. This means that $g(\boldsymbol{x}_1), g(\boldsymbol{x}_2) \leq 0$. Then

$$g(t\boldsymbol{x}_1 + (1-t)\boldsymbol{x}_2) \leq tg(\boldsymbol{x}_1) + (1-t)g(\boldsymbol{x}_2) \leq t * 0 + (1-t) * 0 = 0. \tag{1.4}$$

Thus, the point $t\boldsymbol{x}_1 + (1-t)\boldsymbol{x}_2 \in S$.

Remark 1.4. A function may also be *concave*. The function f in Definition 1.3 is concave if the inequality in Equation (1.3) holds with reversed direction such that $f(t\boldsymbol{x}_1 + (1-t)\boldsymbol{x}_2) \geq tf(\boldsymbol{x}_1) + (1-t)f(\boldsymbol{x}_2)$. It is easy to see from the inequality in Equation (1.3) that if a function f is convex, then the function $-f$ is concave. The following example introduces a very special convex function that is also concave.

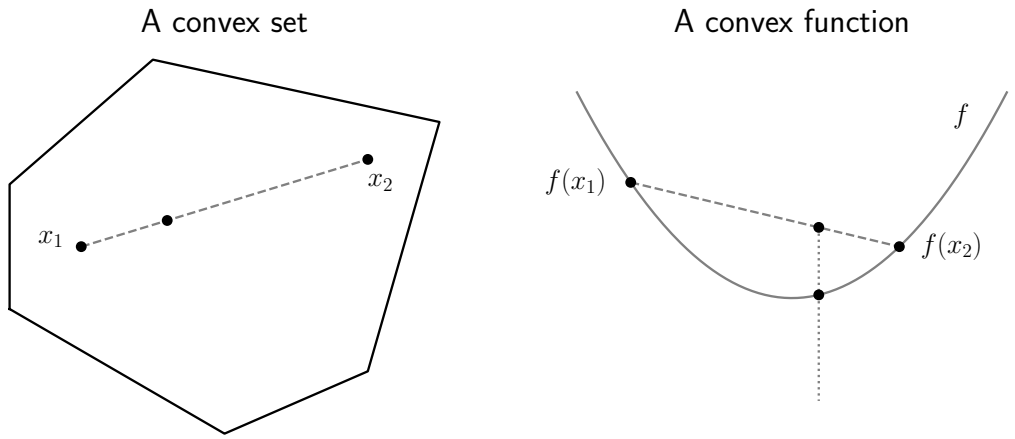

Figure 1: Left: A convex set. The convex combination of any two points in the set belongs to the set. Right: A convex function. The convex combination of any two function values overestimates the true value of the function.

Example 1.5. A linear function is convex. Let $\boldsymbol{c} \in \mathbb{R}^m$. The linear function $L : X \to \mathbb{R}$ is defined as $L(\boldsymbol{x}) = \boldsymbol{c}^T\boldsymbol{x}$. Now, the value of the function L in the point $t\boldsymbol{x}_1 + (1-t)\boldsymbol{x}_2$ is

$$\begin{aligned} L(t\boldsymbol{x}_1 + (1-t)\boldsymbol{x}_2) &= \boldsymbol{c}^T(t\boldsymbol{x}_1 + (1-t)\boldsymbol{x}_2) = t\boldsymbol{c}^T\boldsymbol{x}_1 + (1-t)\boldsymbol{c}^T\boldsymbol{x}_2 \\ &= tL(\boldsymbol{x}_1) + (1-t)L(\boldsymbol{x}_2). \end{aligned} \tag{1.5}$$

As stated in Remark 1.4, linear function is also concave. This follows from the fact that the equality $L(t\boldsymbol{x}_1 + (1-t)\boldsymbol{x}_2) = tL(\boldsymbol{x}_1) + (1-t)L(\boldsymbol{x}_2)$ satisfies both inequalities in Equation (1.3) and the one in Remark 1.4.

Adding a constant and multiplying with a positive real number preserves the direction of an inequality. This means that adding any constant to a convex function or multiplying a convex function with any positive real number results in a convex function. Also, a sum of two convex functions is a convex function as shown in the following example.

Example 1.6. A sum of any two convex functions is a convex function. Let the functions $g : X_1 \to \mathbb{R}$ and $h : X_2 \to \mathbb{R}$ be convex functions and let $f(\boldsymbol{x}) = g(\boldsymbol{x}) + h(\boldsymbol{x})$. Then $f(t\boldsymbol{x}_1 + (1-t)\boldsymbol{x}_2) = g(t\boldsymbol{x}_1 + (1-t)\boldsymbol{x}_2) + h(t\boldsymbol{x}_1 + (1-t)\boldsymbol{x}_2) = tg(\boldsymbol{x}_1) + (1-t)g(\boldsymbol{x}_2) + th(\boldsymbol{x}_1) + (1-t)h(\boldsymbol{x}_2) = tf(\boldsymbol{x}_1) + (1-t)f(\boldsymbol{x}_2)$.

Example 1.7. A quadratic function $f(\boldsymbol{x}) = (1/2)\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{A}\boldsymbol{x}$ is convex if the matrix $Q \in \mathbb{R}^{m*m}$ is positive semi-definite (PSD). A matrix is PSD if

$\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} \geq 0$ for any $\boldsymbol{x} \in \mathbb{R}^m$. Examples 1.5 and 1.6 imply that it is sufficient to show that the quadratic term is convex. Let $g(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x}$. Then

$$\begin{aligned} g(t\boldsymbol{x}_1 + (1-t)\boldsymbol{x}_2) =& (t\boldsymbol{x}_1 + (1-t)\boldsymbol{x}_2)^T\boldsymbol{Q}(t\boldsymbol{x}_1 + (1-t)\boldsymbol{x}_2) \\ =& t^2\boldsymbol{x}_1^T\boldsymbol{Q}\boldsymbol{x}_1 + (1-t)^2\boldsymbol{x}_2^T\boldsymbol{Q}\boldsymbol{x}_2 + 2t(1-t)\boldsymbol{x}_1^T\boldsymbol{Q}\boldsymbol{x}_2 \\ =& t[1-(1-t)]\boldsymbol{x}_1^T\boldsymbol{Q}\boldsymbol{x}_1 + (1-t)(1-t)\boldsymbol{x}_2^T\boldsymbol{Q}\boldsymbol{x}_2 \\ & + 2t(1-t)\boldsymbol{x}_1^T\boldsymbol{Q}\boldsymbol{x}_2 \\ =& t\boldsymbol{x}_1^T\boldsymbol{Q}\boldsymbol{x}_1 + (1-t)\boldsymbol{x}_2^T\boldsymbol{Q}\boldsymbol{x}_2 \\ & - t(1-t)\boldsymbol{x}_1^T\boldsymbol{Q}\boldsymbol{x}_1 - t(1-t)\boldsymbol{x}_2^T\boldsymbol{Q}\boldsymbol{x}_2 + 2t(1-t)\boldsymbol{x}_1^T\boldsymbol{Q}\boldsymbol{x}_2 \\ =& tg(\boldsymbol{x}_1) + (1-t)g(\boldsymbol{x}_2) - t(1-t)(\boldsymbol{x}_1 - \boldsymbol{x}_2)^T\boldsymbol{Q}(\boldsymbol{x}_1 - \boldsymbol{x}_2) \\ \leq& tg(\boldsymbol{x}_1) + (1-t)g(\boldsymbol{x}_2) \end{aligned} \tag{1.6}$$

since $(\boldsymbol{x}_1 - \boldsymbol{x}_2)^T\boldsymbol{Q}(\boldsymbol{x}_1 - \boldsymbol{x}_2) \geq 0$ for a PSD matrix $\boldsymbol{Q}$ and $t(1-t) \geq 0$ by definition. Thus, the function g is convex, which means that the function f is convex.

Lemma 1.8. *The intersection of two convex sets, such that $S = S_1 \cap S_2$, is a convex set.*

Proof. Let $\boldsymbol{x}_1, \boldsymbol{x}_2 \in S$. Then $\boldsymbol{x}_1, \boldsymbol{x}_2 \in S_1$ and $\boldsymbol{x}_1, \boldsymbol{x}_2 \in S_2$ by the definition of intersection. Let us assume that $\boldsymbol{x} = t\boldsymbol{x}_1 + (1-t)\boldsymbol{x}_2 \notin S$ for some $t \in (0,1)$. Then, by the definition of intersection, $\boldsymbol{x} \notin S_1$ or $\boldsymbol{x} \notin S_2$. This is a contradiction since the set S_1 and S_2 are convex sets. Thus, the set S is convex. □

Definition 1.9. The non-linear program in Equation (1.1) is convex if the objective function f is a convex function, and the feasible region is a convex set.

The inequality constraints $g_i(\boldsymbol{x}) \leq 0$ define convex sets if the functions g_i are convex. By Lemma 1.8, the intersection of these sets is a convex set. A set defined by an equality constraint $h_j(\boldsymbol{x}) = 0$ is a bit more restricted. The set $S_j = \{\boldsymbol{x} \in \mathbb{R}^m \mid h_j(\boldsymbol{x}) = 0\}$ is convex if $h_j(t\boldsymbol{x}_1 + (1-t)\boldsymbol{x}_2) = 0$ for all $\boldsymbol{x}_1, \boldsymbol{x}_2 \in S_j$ and $t \in [0,1]$. One family of functions that satisfy this property is *affine functions*. Affine function is a function $h_j(\boldsymbol{x}) = \boldsymbol{c}^T\boldsymbol{x} + b$ with $\boldsymbol{c} \in \mathbb{R}^m$ and $b \in \mathbb{R}$. Now

$$\begin{aligned} h_j(t\boldsymbol{x}_1 + (1-t)\boldsymbol{x}_2) &= \boldsymbol{c}^T(t\boldsymbol{x}_1 + (1-t)\boldsymbol{x}_2) + b \\ &= t\boldsymbol{c}^T\boldsymbol{x}_1 + tb + (1-t)\boldsymbol{c}^T\boldsymbol{x}_2 + (1-t)b \\ &= th_j(\boldsymbol{x}_1) + (1-t)h_j(\boldsymbol{x}_2) = 0. \end{aligned} \tag{1.7}$$

This property of affine functions leads to the following corollary:

Corollary 1.10. *The non-linear program in Equation (1.1) is convex if functions f and $g_1, \ldots, g_n$ are convex, and the function $h_1, \ldots, h_p$ are affine.*

1.1.2 Karush-Kuhn-Tucker conditions

Kuhn and Tucker [34] first published the necessary conditions for non-linear programming in 1951. Later these conditions became known as Karush-Kuhn-Tucker (KKT) conditions when it was revealed that Karush had stated these conditions in his master's thesis in 1938. In 1959 Slater [57] formulated the conditions under which the KKT conditions are also sufficient. The definitions and the proofs in this section are mainly from the book [7].

Let us consider the non-linear program in Equation (1.1). In this section, the non-linear program is not assumed to be convex since the KKT conditions can be derived to the non-convex programs, too. Let us first study *duality* and then carry on to the KKT conditions. The non-linear program in Equation (1.1) has a Lagrangian function:

Definition 1.11. The Lagrangian function to the non-linear program in Equation (1.1) is

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \sum_{i=1}^{n} \lambda_i g_i(\boldsymbol{x}) + \sum_{j=1}^{p} \mu_j h_j(\boldsymbol{x}). \tag{1.8}$$

The Lagrangian augments the objective function with the weighted constraint functions. These weights, λ_i and μ_j, are called *Lagrangian multipliers*. When considering duality, the non-linear program in Equation (1.1) is called a *primal problem*. The following definition defines a *Lagrangian dual problem* to the primal problem:

Definition 1.12. Lagrangian dual problem regarding the primal problem in Equation (1.1) is

$$\underset{\boldsymbol{\lambda} \succeq \mathbf{0}, \boldsymbol{\mu} \in \mathbb{R}^p}{\text{maximize}} \quad q(\boldsymbol{\lambda}, \boldsymbol{\mu}) = \inf_{\boldsymbol{x} \in X} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}). \tag{1.9}$$

In this thesis, the relation $\boldsymbol{\lambda} \succeq \mathbf{0}$ is used as a shorter notation for $\lambda_i \geq 0$ for all $i = 1, \ldots n$. Let the point $(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ be the point that maximizes the dual problem. Let d^* be the value of the objective function of the dual problem at that point. Then

$$d^* = q(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = \max_{\boldsymbol{\lambda} \succeq \mathbf{0}, \boldsymbol{\mu} \in \mathbb{R}^p} q(\boldsymbol{\lambda}, \boldsymbol{\mu}) = \max_{\boldsymbol{\lambda} \succeq \mathbf{0}, \boldsymbol{\mu} \in \mathbb{R}^p} \inf_{\boldsymbol{x} \in X} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}). \tag{1.10}$$

It is straightforward to see that the optimal value of the primal problem in Equation (1.1) has a similar form. The supremum of the Lagrangian subject to the point $(\boldsymbol{\lambda}, \boldsymbol{\mu})$ is

$$\sup_{\boldsymbol{\lambda} \succeq \mathbf{0}, \boldsymbol{\mu} \in \mathbb{R}^p} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \sup_{\boldsymbol{\lambda} \succeq \mathbf{0}, \boldsymbol{\mu} \in \mathbb{R}^p} \left[f(\boldsymbol{x}) + \sum_{i=1}^{n} \lambda_i g_i(\boldsymbol{x}) + \sum_{j=1}^{p} \mu_j h_j(\boldsymbol{x}) \right]. \quad (1.11)$$

The supremum in Equation (1.11) has a value $f(\boldsymbol{x})$ if $g_i(\boldsymbol{x}) \leq 0$ and $h_i(\boldsymbol{x}) = 0$ for all $i = 1, \ldots, n$ and $j = 1, \ldots, p$. If these conditions are not satisfied, the value is infinite. Let the point $\boldsymbol{x}^*$ be the point that optimizes the primal problem in Equation (1.1). Then the optimal value of the primal problem objective function is

$$p^* = f(\boldsymbol{x}^*) = \sup_{\boldsymbol{\lambda} \succeq \mathbf{0}, \boldsymbol{\mu} \in \mathbb{R}^p} \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \min_{\boldsymbol{x} \in X} \sup_{\boldsymbol{\lambda} \succeq \mathbf{0}, \boldsymbol{\mu} \in \mathbb{R}^p} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \quad (1.12)$$

because the point $\boldsymbol{x}^*$ must be in the feasible region. This means that $g_i(\boldsymbol{x}^*) \leq 0$ and $h_i(\boldsymbol{x}^*) = 0$ for all $i = 1, \ldots, n$ and $j = 1, \ldots, p$.

It is trivial that

$$\inf_{\boldsymbol{x} \in X} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \leq \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \leq \sup_{\boldsymbol{\lambda} \succeq \mathbf{0}, \boldsymbol{\mu} \in \mathbb{R}^p} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \quad (1.13)$$

for all $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{\lambda} \succeq \mathbf{0}$ and $\boldsymbol{\mu} \in \mathbb{R}^p$. Especially this relation holds for minimum and maximum values such that

$$d^* = \max_{\boldsymbol{\lambda} \succeq \mathbf{0}, \boldsymbol{\mu} \in \mathbb{R}^p} \inf_{\boldsymbol{x} \in X} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \leq \min_{\boldsymbol{x} \in X} \sup_{\boldsymbol{\lambda} \succeq \mathbf{0}, \boldsymbol{\mu} \in \mathbb{R}^p} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = p^*. \quad (1.14)$$

This property is called *weak duality*. Weak duality states that the optimal value of the dual problem in Equation (1.9) always lower bounds the optimal value of the primal problem in Equation (1.1). The difference $p^* - d^*$ is called the *duality gap*, and it is always non-negative. In the special case when the property in Equation (1.14) holds with equality the duality gap vanishes. This case is called *strong duality*. In case of strong duality, the value d^* is the best lower bound for the optimal value of the primal problem in Equation (1.1).

Let us assume that the strong duality holds. Then

$$\begin{aligned} f(\boldsymbol{x}^*) &= p^* = d^* = q(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \\ &= \inf_{\boldsymbol{x} \in X} \left(f(\boldsymbol{x}) + \sum_{i=1}^{n} \lambda_i^* g_i(\boldsymbol{x}) + \sum_{j=1}^{p} \mu_j^* h_j(\boldsymbol{x}) \right) \\ &\leq f(\boldsymbol{x}^*) + \sum_{i=1}^{n} \lambda_i^* g_i(\boldsymbol{x}^*) + \sum_{j=1}^{p} \mu_j^* h_j(\boldsymbol{x}^*) \leq f(\boldsymbol{x}^*) \end{aligned} \quad (1.15)$$

because $h_j(\boldsymbol{x}^*) = 0$ for all $j = 1, \ldots, p$ and $\lambda_i^* g_i(\boldsymbol{x}^*) \leq 0$ for all $i = 1, \ldots, n$. The latter inequality follows from the constraints $\lambda_i^* \geq 0$ and $g_i(\boldsymbol{x}^*) \leq 0$. The inequalities in Equation (1.15) can be replaced by equalities. This means that $\lambda_i^* g_i(\boldsymbol{x}^*) = 0$ for all $i = 1, \ldots, n$ because these terms must be non-negative. This property is called *complementary slackness*.

Let us consider the point $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \in \mathbb{R}^m \times \mathbb{R}^n \times \mathbb{R}^p$ where the point $\boldsymbol{x}^*$ minimizes the primal problem in Equation (1.1), and the point $(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ maximizes the dual problem in Equation (1.9). Let us also assume that the strong duality holds, such that $f(\boldsymbol{x}^*) = q(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$. Since the point $\boldsymbol{x}^*$ minimizes the Lagrangian $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ over the variable $\boldsymbol{x}$, the gradient must be zero at that point, such that $\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = 0$. Putting all this together, one has derived Karush-Kuhn-Tucker (KKT) conditions:

Theorem 1.13 (Karush-Kuhn-Tucker condition)**.** *Let the point $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ satisfy the strong duality. Then the point $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ also satisfies the following conditions:*

$$\nabla f(\boldsymbol{x}^*) + \sum_{i=1}^{n} \lambda_i^* \nabla g_i(\boldsymbol{x}^*) + \sum_{j=1}^{p} \mu_j^* \nabla h_j(\boldsymbol{x}^*) = 0 \tag{1.16}$$

$$g_i(\boldsymbol{x}^*) \leq 0 \text{ and } h_j(\boldsymbol{x}^*) = 0 \tag{1.17}$$

$$\lambda_i^* \geq 0 \tag{1.18}$$

$$\lambda_i^* g_i(\boldsymbol{x}^*) = 0 \tag{1.19}$$

for all $i = 1, \ldots, n$ and $j = 1, \ldots, m$.

Proof. The proof is in the derivation. □

1.1.3 Slater's condition

In the previous section, it was shown that the strong duality ensures that the KKT conditions hold. The strong duality, however, does not hold generally even for convex programs. This is shown in the following example.

Example 1.14. Let us consider a minimization problem

$$\begin{aligned} &\underset{\boldsymbol{x} \in \mathbb{R}}{\text{minimize}} && x \\ &\text{subject to} && x^2 \leq 0. \end{aligned} \tag{1.20}$$

This minimization problem is clearly a convex program. Its Lagrangian is $\mathcal{L}(x, \lambda) = x + \lambda x^2$. The objective function of the dual problem is the infimum of the Lagrangian, such that $q(\lambda) = \inf_{x \in \mathbb{R}}(x + \lambda x^2)$ with $\lambda \geq 0$. If $\lambda = 0$, then clearly $q(0) = -\infty$.

Let us assume that $\lambda > 0$. Now, $x+\lambda x^2$ is a parabola that opens upwards, and its minimum is at the root of its derivative. Let us set the derivative to zero such that $\mathrm{D}_x(x + \lambda x^2) = 1 + 2\lambda x = 0$. The derivative vanishes at $x = -1/2\lambda$ so the parabola reaches its minimum at that point. Using this information, the cost function of the dual problem is

$$q(\lambda) = \inf_{x \in \mathbb{R}}(x + \lambda x^2) = -\frac{1}{2\lambda} + \lambda\left(-\frac{1}{2\lambda}\right)^2 = -\frac{1}{4\lambda}. \tag{1.21}$$

The function q has no maximum when $\lambda \geq 0$. This means that the dual problem has no solution. For this reason, d^* is undefined, and $d^* \neq p^*$.

The problem of ensuring the strong duality, as demonstrated in Example 1.14, has lead to the study of *constraint qualifications*. There are various constraint qualifications that ensure strong duality. For convex programming, one such constraint qualification is *Slater's condition*. The *Slater point* and Slater's condition are defined later in this section in Definitions 1.17 and 1.18, respectively. These definitions use the definition of *relative interior*.

Definition 1.15. The relative interior of a set $S \subset \mathbb{R}^m$ is

$$\mathrm{relint}(S) = \{\boldsymbol{x} \in S \mid B(\boldsymbol{x}, \epsilon) \cap \mathrm{aff}(S) \subset S\} \tag{1.22}$$

for some $\epsilon > 0$.

In Equation (1.22) the set $B(\boldsymbol{x}, \epsilon) := \{\boldsymbol{x'} \in R^m \mid \|\boldsymbol{x'} - \boldsymbol{x}\| \leq \epsilon\}$ is a *ball* with a center $\boldsymbol{x}$ and radius ϵ. The operator $\|\cdot\|$ is the *Euclidean norm*.

Definition 1.16. Euclidean norm of a point $\boldsymbol{x} \in R^m$ is

$$\|\boldsymbol{x}\| := (\boldsymbol{x}^T\boldsymbol{x})^{1/2} = \sqrt{\sum_{i=1}^{m} {x_i}^2}. \tag{1.23}$$

Also, in Equation (1.22) the set $\mathrm{aff}(S) := \{\sum_{i=1}^{k} \alpha_i \boldsymbol{x}_i \in \mathbb{R}^m \mid \alpha_i \in \mathbb{R},\ \boldsymbol{x}_i \in S,\ \sum_{i=1}^{k} \alpha_i = 1\}$ is an *affine hull* of the set S for any $k = 1, 2, \ldots$.

Let us assume convexity from now on. Slater's condition is defined for a convex program.

Definition 1.17. Slater point is a point in the relative interior of the domain D that satisfies $g_i(\boldsymbol{x}) < 0$ for all $i = 1, \ldots, n$.

Definition 1.18 (Slater's Condition)**.** The convex program in Equation (1.1) satisfies Slater's condition if there is a Slater point in its feasible region.

Before proving that Slater's condition implies strong duality for convex program, let us prove two helpful theorems: Supporting Hyperplane Theorem and Separating Hyperplane Theorem.

Theorem 1.19 (Supporting Hyperplane Theorem)**.** *Let the set $S \subset \mathbb{R}^m$ be convex and non-empty. Let $\bar{\boldsymbol{x}} \notin int(S)$ be a point that is not in the interior of the set S. Then there is a hyperplane such that $\boldsymbol{a}^T\bar{\boldsymbol{x}} \leq b$ and $\boldsymbol{a}^T\boldsymbol{s} \geq b$ for all $\boldsymbol{s} \in S$.*

Proof. If the point $\bar{\boldsymbol{x}}$ is not in the interior of the set S, then it is either outside the closure of the set S or in the boundary of the set S. These two cases are proven separately. Let us first assume that $\bar{\boldsymbol{x}} \notin \mathrm{cl}(S)$.

Let a point $\hat{\boldsymbol{x}} \in \mathrm{cl}(S)$ be the projection of the point $\bar{\boldsymbol{x}}$ to the set $\mathrm{cl}(S)$. In other words, the inequality $\|\hat{\boldsymbol{x}} - \bar{\boldsymbol{x}}\| \leq \|\boldsymbol{x} - \bar{\boldsymbol{x}}\|$ holds for all $\boldsymbol{x} \in \mathrm{cl}(S)$. Let us define $\boldsymbol{a} = \hat{\boldsymbol{x}} - \bar{\boldsymbol{x}}$ and $b = \boldsymbol{a}^T\hat{\boldsymbol{x}}$. The vector $\boldsymbol{a}$ and the scalar b will define the supporting hyperplane.

This first case can be proven with a contradiction. Let us assume that the theorem does not hold for $\bar{\boldsymbol{x}} \notin \mathrm{cl}(S)$. This means that there is some $\boldsymbol{x} \in \mathrm{cl}(S)$ with $\boldsymbol{a}^T\boldsymbol{x} < b$. Because the set $\mathrm{cl}(S)$ is convex, there is a point $\boldsymbol{x}_t = t\boldsymbol{x} + (1-t)\hat{\boldsymbol{x}} \in \mathrm{cl}(S)$ with all $t \in [0, 1]$. It will be shown here that, with some t sufficiently close to zero, the point $\boldsymbol{x}_t$ is closer to the point $\bar{\boldsymbol{x}}$ than its projection $\hat{\boldsymbol{x}}$. In other words, it will turn out that $\|\boldsymbol{x}_t - \bar{\boldsymbol{x}}\| < \|\hat{\boldsymbol{x}} - \bar{\boldsymbol{x}}\|$ for some $t \in [0, 1]$.

The vector $\boldsymbol{x}_t - \bar{\boldsymbol{x}}$ can be written as

$$\boldsymbol{x}_t - \bar{\boldsymbol{x}} = t\boldsymbol{x} + (1-t)\hat{\boldsymbol{x}} - \bar{\boldsymbol{x}} = (\hat{\boldsymbol{x}} - \bar{\boldsymbol{x}}) + t(\boldsymbol{x} - \hat{\boldsymbol{x}}) = \boldsymbol{a} + t(\boldsymbol{x} - \hat{\boldsymbol{x}}). \quad (1.24)$$

The squared norm of the vector $\boldsymbol{x}_t - \bar{\boldsymbol{x}}$ is then

$$\begin{aligned} \|\boldsymbol{x}_t - \bar{\boldsymbol{x}}\|^2 &= \|\boldsymbol{a} + t(\boldsymbol{x} - \hat{\boldsymbol{x}})\|^2 \\ &= \boldsymbol{a}^T\boldsymbol{a} + t^2\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2 + 2t\boldsymbol{a}^T(\boldsymbol{x} - \hat{\boldsymbol{x}}) \\ &= \|\hat{\boldsymbol{x}} - \bar{\boldsymbol{x}}\|^2 + t\left(2(\boldsymbol{a}^T\boldsymbol{x} - \boldsymbol{a}^T\hat{\boldsymbol{x}}) + t\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|\right). \end{aligned} \quad (1.25)$$

Here, the term $\boldsymbol{a}^T\boldsymbol{x} - \boldsymbol{a}^T\hat{\boldsymbol{x}} = \boldsymbol{a}^T\boldsymbol{x} - b < 0$ by the assumption that the theorem does not hold for $\bar{\boldsymbol{x}} \notin \mathrm{cl}(S)$. Note that the term $\boldsymbol{a}^T\boldsymbol{x} - \boldsymbol{a}^T\hat{\boldsymbol{x}}$ does not depend on the variable t. When the multiplier t is small enough, the term $2(\boldsymbol{a}^T\boldsymbol{x} - \boldsymbol{a}^T\hat{\boldsymbol{x}}) + t\|\boldsymbol{x} - \hat{\boldsymbol{x}}\| < 0$. Then, from Equation (1.25), it can be seen that $\|\boldsymbol{x}_t - \bar{\boldsymbol{x}}\| < \|\hat{\boldsymbol{x}} - \bar{\boldsymbol{x}}\|$ for some $\boldsymbol{x}_t \in \mathrm{cl}(S)$. This inequality is a contradiction since $\hat{\boldsymbol{x}}$ was selected so that $\|\hat{\boldsymbol{x}} - \bar{\boldsymbol{x}}\| \leq \|\boldsymbol{x} - \bar{\boldsymbol{x}}\|$ for all $\boldsymbol{x} \in \mathrm{cl}(S)$. This contradiction proves that $\boldsymbol{a}^T\boldsymbol{x} \geq b$ for all $\boldsymbol{x} \in \mathrm{cl}(S)$.

Let us now prove the second case. Let us assume that $\bar{\boldsymbol{x}} \in \mathrm{cl}(S)$. Let the sequence $\{\boldsymbol{x}_k\}$ converge to the point $\bar{\boldsymbol{x}}$ so that $\boldsymbol{x}_k \notin \mathrm{cl}(S)$ for all $k = 1, 2, \ldots$.

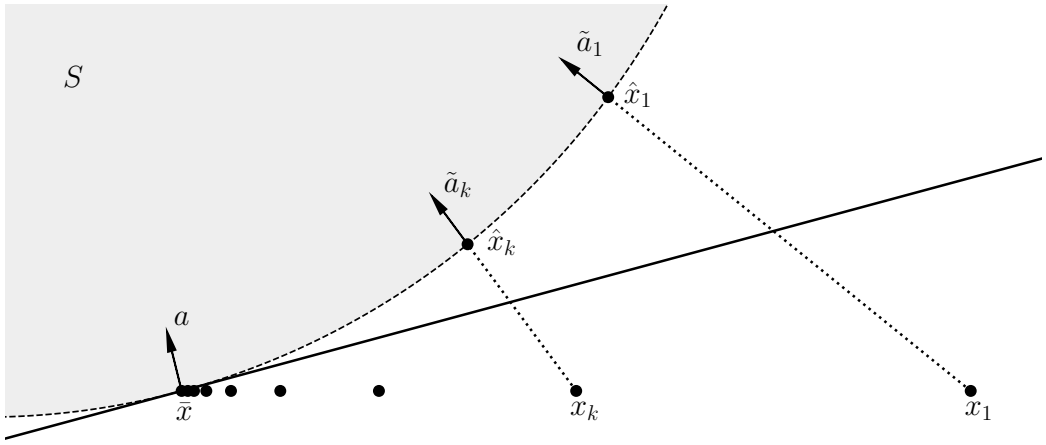

Figure 2: A visual proof of Supporting Hyperplane Theorem 1.19. The shaded area is a convex set. The normal vectors of the hyperplanes are the unit length vectors $\tilde{\boldsymbol{\alpha}}_k$ in the direction of the vectors $\hat{\boldsymbol{x}}_k - \boldsymbol{x}_k$. This sequence of normal vectors converges to the normal vector $\boldsymbol{a}$ of the hyperplane that supports the convex set.

This sequence exists because $\bar{\boldsymbol{x}} \notin \text{int}(S)$. Let the projections of these points to the set $\text{cl}(S)$ be $\hat{\boldsymbol{x}}_k$. From the previous part of this proof, it is known that there exists a hyperplane such that $\boldsymbol{a}_k^T \boldsymbol{x} \geq b_k$ for all $\boldsymbol{x} \in \text{cl}(C)$ with $\boldsymbol{a}_k = \hat{\boldsymbol{x}}_k - \boldsymbol{x}_k$ and $b_k = \boldsymbol{a}_k^T \boldsymbol{x}_k$ for all $k = 1, 2, \ldots$. Dividing by a positive real number does not change the direction of the inequality, so the inequality

$$\tilde{\boldsymbol{a}}_k^T \boldsymbol{x} := \left(\frac{\boldsymbol{a}_k}{\|\hat{\boldsymbol{x}}_k - \boldsymbol{x}_k\|} \right)^T \boldsymbol{x} \geq \frac{b_k}{\|\hat{\boldsymbol{x}}_k - \boldsymbol{x}_k\|} =: \tilde{b}_k \tag{1.26}$$

holds for all $\boldsymbol{x} \in \text{cl}(S)$. The vector $\tilde{\boldsymbol{a}}_k$ has a norm $\|\tilde{\boldsymbol{a}}_k\| = 1$. This is illustrated in Figure 2 where the arrows denote the vectors $\tilde{\boldsymbol{a}}_k$. Let us define $\boldsymbol{a} := \lim_{k \to \infty} \tilde{\boldsymbol{a}}_k$ and $b := \boldsymbol{a}^T \bar{\boldsymbol{x}}$. Now the supporting hyperplane satisfies $\boldsymbol{a}^T \boldsymbol{x} \geq b$ for all $\boldsymbol{x} \in \text{cl}(S)$. Finally, the proposition $\boldsymbol{a}^T \boldsymbol{s} \geq b$ holds for all $\boldsymbol{s} \in S$ because $S \subset \text{cl}(S)$. □

Theorem 1.20 (Separating Hyperplane Theorem)**.** *Let the sets S_1 and S_2 be convex, non-empty, disjoint sets, such that $S_1 \cap S_2 = \emptyset$. Then there is a hyperplane that separates them.*

Proof. Let us define a new set Y so that $Y = \{\boldsymbol{u} - \boldsymbol{v} \mid \boldsymbol{u} \in S_1,\ \boldsymbol{v} \in S_2\}$. Let us first show that this set is convex, and $\boldsymbol{0} \notin Y$. Then Supporting Hyperplane Theorem 1.19 can be applied to the point $\boldsymbol{0}$ and the set Y.

The set Y is convex because its elements are sums of elements from convex sets. Also, $\mathbf{0} \notin Y$. If it would be that $\mathbf{0} \in Y$, then $\mathbf{0} = \boldsymbol{u} - \boldsymbol{v}$ meaning that $\boldsymbol{u} = \boldsymbol{v}$. Then $\boldsymbol{u} \in S_1$ and $\boldsymbol{u} = \boldsymbol{v} \in S_2$ which is a contradiction since the sets S_1 and S_2 are disjoint.

From Theorem (1.19) it is known that there is a hyperplane such that $\boldsymbol{a}^T\mathbf{0} \leq b$ and $\boldsymbol{a}^T\boldsymbol{y} \geq b$ for all $\boldsymbol{y} \in Y$. This means that $\boldsymbol{a}^T\boldsymbol{y} \geq \boldsymbol{a}^T\mathbf{0} = 0$, which implies that $\boldsymbol{a}^T(\boldsymbol{u}-\boldsymbol{v}) \geq 0$. From this result, it can be seen that the inequality $\boldsymbol{a}^T\boldsymbol{u} \geq \boldsymbol{a}^T\boldsymbol{v}$ holds for all $\boldsymbol{u} \in S_1$ and $\boldsymbol{v} \in S_2$. Especially this is true for the infimum and the supremum, which means that $\inf_{\boldsymbol{u}\in S_1} \boldsymbol{a}^T\boldsymbol{u} \geq \sup_{\boldsymbol{v}\in S_2} \boldsymbol{a}^T\boldsymbol{v}$. This inequality proves the existence of the separating hyperplane. □

Since the non-linear program is assumed to be convex, let us assume that the equality constraints defining function h_j are affine. These constraints can be presented compactly as an equation $(h_1(\boldsymbol{x}), \ldots, h_p(\boldsymbol{x}))^T = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} = \mathbf{0}$ with $\boldsymbol{A} \in \mathbb{R}^{p\times m}$ and $\boldsymbol{b} \in \mathbb{R}^p$. For the following proof, let us assume that $\text{rank}(\boldsymbol{A}) = p$ so the rows of Jacobian regarding the equality constraints are linearly independent. If this condition is not satisfied, then either there are contradicting or unnecessary equality constraints [2].

Let us make another simplifying assumption that the set D has a non-empty interior. This means that $\text{int}(D) = \{\boldsymbol{x} \in \mathbb{R}^m \mid B(\boldsymbol{x}, \epsilon) \subset D\} \neq \emptyset$ for some $\epsilon > 0$. Then $\text{relint}(D) = \text{int}(D)$. This assumption simplifies the following proof at the cost of some generality. The proof will, however, remain general enough for the rest of this thesis.

Theorem 1.21. *Let relint(D) = int(D) and rank$(\boldsymbol{A}) = p$. Then for this convex program Slater's condition in Definition 1.18 implies strong duality.*

Proof. If the primal optimal solution is $p^* = -\infty$, then by weak duality the dual optimal solution is $d^* \leq p^* = -\infty$. This means that $d^* = p^*$, and the proposition is always true. Let us next assume that the primal optimal solution is finite.

Let us first define two disjoint, convex sets

$$\begin{cases} \mathcal{A} & = \{(\boldsymbol{u}, \boldsymbol{v}, t) \mid \boldsymbol{x} \in D,\ g_i(\boldsymbol{x}) \leq u_i,\ h_j(\boldsymbol{x}) = v_j,\ f(\boldsymbol{x}) \leq t, \\ & \qquad i = 1, \ldots n,\ j = 1, \ldots p\} \\ \mathcal{B} & = \{(\mathbf{0}, \mathbf{0}, s) \mid s < p^*\}. \end{cases} \tag{1.27}$$

Let us first show that the sets $\mathcal{A}$ and $\mathcal{B}$ can be separated by a hyperplane. This hyperplane will define a useful relation between Lagrangian and the optimal primal value p^*.

Clearly the set $\mathcal{B}$ is both convex and concave since it is a half line in $\mathbb{R}^{n+m+1}$. The set $\mathcal{A}$ is also convex because the functions f, g_i and h_j are

convex for a convex program. Let us study this proposition more carefully. Let us consider the point $\lambda(\boldsymbol{u}_1, \boldsymbol{v}_1, t_1) + (1-\lambda)(\boldsymbol{u}_2, \boldsymbol{v}_2, t_2) \in \mathbb{R}^{n+m+1}$ and points $(\boldsymbol{u}_1, \boldsymbol{v}_1, t_1), (\boldsymbol{u}_2, \boldsymbol{v}_2, t_2) \in \mathcal{A}$. There are a point $\boldsymbol{x}_1$ that satisfies the inequalities for $(\boldsymbol{u}_1, \boldsymbol{v}_1, t_1)$ and a point $\boldsymbol{x}_2$ that satisfies the inequalities for $(\boldsymbol{u}_2, \boldsymbol{v}_2, t_2)$. Now

$$f(\lambda \boldsymbol{x}_1 + (1-\lambda)\boldsymbol{x}_2) \leq \lambda f(\boldsymbol{x}_1) + (1-\lambda) f(\boldsymbol{x}_2) \leq \lambda t_1 + (1-\lambda) t_2 \qquad (1.28)$$

for any $\lambda \in [0, 1]$ because of the convexity of the function f. The functions g_i are convex and the functions h_j are affine and thus behave similarly to the function f. This means that the point $\lambda \boldsymbol{x}_1 + (1-\lambda)\boldsymbol{x}_2$ satisfies the inequalities for the point $\lambda(\boldsymbol{u}_1, \boldsymbol{v}_1, t_1) + (1-\lambda)(\boldsymbol{u}_2, \boldsymbol{v}_2, t_2)$ for all $\lambda \in [0, 1]$ and the set $\mathcal{A}$ is indeed convex.

The sets $\mathcal{A}$ and $\mathcal{B}$ do not intersect. If they did, there would be a point $(\boldsymbol{0}, \boldsymbol{0}, t) \in \mathbb{R}^{n+m+1}$ belonging in both sets $\mathcal{A}$ and $\mathcal{B}$. If the point $(\boldsymbol{0}, \boldsymbol{0}, t)$ belongs to the set $\mathcal{A}$, there exists a point $\boldsymbol{x} \in D$ that satisfies all the primal constraints $g_i(\boldsymbol{x}) \leq 0$ and $h_i(\boldsymbol{x}) = 0$. In addition, the objective function value at the point $\boldsymbol{x}$ has an upper bound $f(\boldsymbol{x}) \leq t$. If the point $(\boldsymbol{0}, \boldsymbol{0}, t)$ belongs to the set $\mathcal{B}$ too, the upper bound t satisfies the inequality $t < p^*$. This is a contradiction since p^* is the optimal primal solution. Thus, the sets $\mathcal{A}$ and $\mathcal{B}$ must be disjoint.

Now that the sets $\mathcal{A}$ and $\mathcal{B}$ are shown to be convex and disjoint, let us use Separating Hyperplane Theorem 1.20 on them. Separating Hyperplane Theorem 1.20 states that there exist a vector $(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\nu}}, \mu) \neq \boldsymbol{0}$ and a scalar $\alpha \in \mathbb{R}$ satisfying the following inequalities:

$$(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\nu}}, \mu)^T (\boldsymbol{u}, \boldsymbol{v}, t) \geq \alpha \qquad (1.29)$$

for all $(\boldsymbol{u}, \boldsymbol{v}, t) \in \mathcal{A}$ and

$$(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\nu}}, \mu)^T (\boldsymbol{0}, \boldsymbol{0}, s) \leq \alpha \qquad (1.30)$$

for all $(\boldsymbol{0}, \boldsymbol{0}, s) \in \mathcal{B}$.

Equation (1.29) implies that $\tilde{\boldsymbol{\lambda}} \succeq \boldsymbol{0}$ and $\mu \geq 0$. If any component in the vector $\tilde{\boldsymbol{\lambda}}$ or the scalar μ were negative, the corresponding component in $(\boldsymbol{u}, \boldsymbol{v}, t)$ could be arbitrarily large and still $(\boldsymbol{u}, \boldsymbol{v}, t) \in \mathcal{A}$. In other words, for any $\alpha \in \mathbb{R}$ there would be either u_i or t with both $(\boldsymbol{u}, \boldsymbol{v}, t) \in \mathcal{A}$ and $(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\nu}}, \mu)^T (\boldsymbol{u}, \boldsymbol{v}, t) < \alpha$. This would mean that the inner product $(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\nu}}, \mu)^T (\boldsymbol{u}, \boldsymbol{v}, t)$ would not be bounded from below. This contradicts Equation (1.29) and proves that $\tilde{\boldsymbol{\lambda}} \succeq \boldsymbol{0}$ and $\mu \geq 0$. Now there are two cases: either $\mu > 0$ or $\mu = 0$. Let us first prove that $\mu > 0$ implies strong duality.

Equation (1.30) implies that $\mu s \leq \alpha$. This is true for all $s < p^*$. Especially this relation holds for the supremum of the scalar s. From Equation (1.27), it can be seen that $\sup s = p^*$. This means that $\mu(\sup s) = \mu p^* \leq \alpha$.

For any $\boldsymbol{x} \in D$, there is a point $(g_1(\boldsymbol{x}), \ldots, g_n(\boldsymbol{x}), h_1(\boldsymbol{x}), \ldots, h_p(\boldsymbol{x}), f(\boldsymbol{x}))$ $\in \mathcal{A}$. From this and Equations (1.29) and (1.30), one can see that

$$\sum_{i=1}^{n} \tilde{\lambda}_i g_i(\boldsymbol{x}) + \sum_{j=1}^{p} \tilde{\nu}_j h_j(\boldsymbol{x}) + \mu f(\boldsymbol{x}) \geq \alpha \geq \mu p^* \tag{1.31}$$

for any $\boldsymbol{x} \in D$.

Let us divide Equation (1.31) by μ. This can be done since $\mu > 0$ by assumption. This results in the following equation

$$\sum_{i=1}^{n} \frac{\tilde{\lambda}_i}{\mu} g_i(\boldsymbol{x}) + \sum_{j=1}^{p} \frac{\tilde{\nu}_j}{\mu} h_j(\boldsymbol{x}) + f(\boldsymbol{x}) = \mathcal{L}(\boldsymbol{x}, \tilde{\boldsymbol{\lambda}}/\mu, \tilde{\boldsymbol{\nu}}/\mu) \geq p^* \tag{1.32}$$

which holds for all $\boldsymbol{x} \in D$. Taking the infimum over $\boldsymbol{x}$ of the Lagrangian in the inequality in Equation (1.32) results in inequality $\inf_{\boldsymbol{x} \in D} \mathcal{L}(\boldsymbol{x}, \tilde{\boldsymbol{\lambda}}/\mu, \tilde{\boldsymbol{\nu}}/\mu) = q(\tilde{\boldsymbol{\lambda}}/\mu, \tilde{\boldsymbol{\nu}}/\mu) \geq p^*$. By weak duality $q(\tilde{\boldsymbol{\lambda}}/\mu, \tilde{\boldsymbol{\nu}}/\mu) \leq p^*$. This means that $p^* = q(\tilde{\boldsymbol{\lambda}}/\mu, \tilde{\boldsymbol{\nu}}/\mu) = d^*$. In other words, strong duality holds if $\mu > 0$.

Let us now show that μ is positive when Slater's condition holds. Let us consider the second case where $\mu = 0$. This selection will lead to a contradiction. Now Equation (1.31) implies that

$$\sum_{i=1}^{n} \tilde{\lambda}_i g_i(\boldsymbol{x}) + \sum_{j=1}^{p} \tilde{\nu}_j h_j(\boldsymbol{x}) \geq 0 \tag{1.33}$$

for all $\boldsymbol{x} \in D$. Especially Equation (1.33) holds for the Slater point $\tilde{\boldsymbol{x}}$. In addition, the Slater point satisfies the equality constraints, so

$$\sum_{i=1}^{n} \tilde{\lambda}_i g_i(\tilde{\boldsymbol{x}}) \geq 0. \tag{1.34}$$

Since for the Slater point $g_i(\tilde{\boldsymbol{x}}) < 0$ for all $i = 1, \ldots n$, the left side of the inequality in Equation (1.34) can have non-negative value only if $\tilde{\boldsymbol{\lambda}} = \mathbf{0}$. Note that Separating Hyperplane Theorem (1.20) implies that $(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\nu}}, \mu) \neq \mathbf{0}$. Since $\mu = 0$ and $\tilde{\boldsymbol{\lambda}} = \mathbf{0}$, the only possibility left is that $\tilde{\boldsymbol{\nu}} \neq \mathbf{0}$.

Let us now show that the property $\tilde{\boldsymbol{\nu}} \neq \mathbf{0}$ cannot hold. Since $\mu = 0$ and $\tilde{\boldsymbol{\lambda}} = \mathbf{0}$, the inequality in Equation (1.33) implies that

$$\sum_{j=1}^{p} \tilde{\nu}_j h_j(\boldsymbol{x}) = \tilde{\boldsymbol{\nu}}^T (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \geq 0 \tag{1.35}$$

for all $\boldsymbol{x} \in D$ assuming that the function h_j are affine. The Slater point satisfies the equation

$$\sum_{j=1}^{p} \tilde{\nu}_j h_j(\tilde{\boldsymbol{x}}) = \tilde{\boldsymbol{\nu}}^T(\boldsymbol{A}\tilde{\boldsymbol{x}} - \boldsymbol{b}) = 0 \tag{1.36}$$

by definition. The Slater point is in the interior of the domain D, such that $\tilde{\boldsymbol{x}} \in \text{int}(D)$. This means that there exists at least one $\boldsymbol{x} \in D$ that satisfies the inequality $\tilde{\boldsymbol{\nu}}^T(\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}) < 0$ unless $\boldsymbol{A}^T\tilde{\boldsymbol{\nu}} = \boldsymbol{0}$. However, the equation $\boldsymbol{A}^T\tilde{\boldsymbol{\nu}} = \boldsymbol{0}$ cannot hold for $\text{rank}(\boldsymbol{A}) = p$ and $\tilde{\boldsymbol{\nu}} \neq \boldsymbol{0}$. From this it can be concluded that $\tilde{\boldsymbol{\nu}} = \boldsymbol{0}$. This means that the identity $(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\nu}}, \mu) \equiv \boldsymbol{0}$ holds, which contradicts Separating Hyperplane Theorem 1.20. This contradiction followed from the assumption $\mu = 0$. Since μ is non-negative, the only possibility is that $\mu > 0$, which was proven earlier in this proof. □

Let us summarize this section. Let the non-linear program in Equation (1.1) be convex. Let it also satisfy the Slater's condition in Definition 1.18. Then there exists at least one point $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ that satisfies strong duality. These points, which satisfy strong duality, also satisfy KKT conditions in Theorem 1.13. In addition, the variable $\boldsymbol{x}^*$ in any of those points is the global optimum to the primal problem in Equation (1.1).

1.2 Dynamic Programming

In the previous section, the minimization involved only one decision: the optimal selection of the point $\boldsymbol{x}^*$. This section is dedicated to the systems that involve multiple decision to be made over time. These systems are called dynamic. The main reference in this section is [6].

1.2.1 Discrete-time Dynamic System

Discrete-time dynamic system is a system that evolves with time. Discrete-time refers to that the system changes its state every once in a while. Let the state of the system at time k be $\boldsymbol{s}^{(k)} \in S_k$. A state change is observed at times $k = 0, 1, \ldots, K$. These time instants constitute a *planning horizon*. The value K is called the end of the planning horizon. These notations are used to formally define the discrete-time dynamic system.

Definition 1.22 (Discrete-time dynamic system)**.** Let the state $\boldsymbol{s}^{(0)} \in S_0$ be a fixed first state in a planning horizon $k = 0, 1, \ldots, K$. For the following time points, the system evolves according to a function $\boldsymbol{s}^{(k+1)} = f_k(\boldsymbol{s}^{(k)}, \boldsymbol{u}^{(k)}, \boldsymbol{W}^{(k)})$ where $\boldsymbol{u}^{(k)} \in U_k(\boldsymbol{s}^{(k)})$ is an action taken, and $\boldsymbol{W}^{(k)} \in D_k$ is a random disturbance at the time point k.

Let us assume that the random disturbances $\boldsymbol{W}^{(k)}$ are independent of each other. The domain $U_k(\boldsymbol{s}^{(k)})$ is a set of actions that are allowed at the current state. At each time point k, an action is chosen from the domain $U_k(\boldsymbol{s}^{(k)})$. The selection is done based on a rule:

Definition 1.23. Function μ_k is a decision rule. It makes a decision based on the knowledge of the current state, and it realizes as an action such that $\boldsymbol{u}^{(k)} = \mu_k(\boldsymbol{s}^{(k)})$.

Decision rules constitute a *policy*:

Definition 1.24. A policy $\pi = (\mu_0, \mu_1, \ldots, \mu_{K-1})$ is a sequence of decision rules selected over the planning horizon.

A policy is a sequence of functions. The policy determines which action to take at any time at any state. This means that the policy selected will generate an action depending on the current state. The goal of dynamic programming is to find the optimal policy. The optimal policy will determine the optimal actions in any state.

The value of the policy is measured by an objective function. In this section, the objective is to minimize a cost function. One characteristic of dynamic programming is that the cost function consists of multiple *cost functionals*. Let function g_k be a cost functional, a function of the current state, the action taken and a random disturbance. Its value is $g_k(\boldsymbol{s}^{(k)}, \boldsymbol{u}^{(k)}, \boldsymbol{W}^{(k)})$ for $k = 0, 1, \ldots, K-1$ and $g_K(\boldsymbol{s}^{(K)})$. Now, finding the optimal policy can be stated as a *dynamic program*:

Definition 1.25. Let the system in Definition 1.22 evolve according the functions f_k, and let the costs of the states, actions and random disturbances occur according the cost functionals g_k over a planning horizon $k = 0, 1, \ldots, K$. Then the dynamic program is

$$\begin{aligned} \underset{\boldsymbol{\pi} \in \boldsymbol{\Pi}}{\text{minimize}} \quad & J_{\boldsymbol{\pi}}(\boldsymbol{s}^{(0)}) = \mathbb{E}\left[g_K(\boldsymbol{s}^{(K)}) + \sum_{k=1}^{K-1} g_k(\boldsymbol{s}^{(k)}, \mu_k(\boldsymbol{s}^{(k)}), \boldsymbol{W}^{(k)}) \right] \\ \text{subject to} \quad & \boldsymbol{s}^{(k+1)} = f_k(\boldsymbol{s}^{(k)}, \mu_k(\boldsymbol{s}^{(k)}), \boldsymbol{W}^{(k)}) \quad , k = 0, \ldots K-1. \end{aligned} \tag{1.37}$$

The domain Π is a set of all *admissible policies*. For an admissible policy, all decisions yield an action that is allowed in the current state, such that $\mu_k(\boldsymbol{s}^{(k)}) \in U_k(\boldsymbol{s}^{(k)})$. The symbol $\mathbb{E}$ means expected value over all random vectors $\boldsymbol{W}^{(0)}, \boldsymbol{W}^{(1)}, \ldots, \boldsymbol{W}^{(K-1)}$.

Let us now define a *dynamic programming sub-problem* based on Definition 1.25:

Definition 1.26. Let the dynamic program be defined as in Definition 1.25. Its sub-problem over a planning horizon $l = k, \dots, K$ is

$$\begin{aligned} \underset{\boldsymbol{\pi}_k \in \boldsymbol{\Pi}_k}{\text{minimize}} \quad & J_{\boldsymbol{\pi}_k}(\boldsymbol{s}^{(k)}) = \mathbb{E}\left[g_K(\boldsymbol{s}^{(K)}) + \sum_{l=k}^{K-1} g_l(\boldsymbol{s}^{(l)}, \mu_l(\boldsymbol{s}^{(l)}), \boldsymbol{W}^{(l)}) \right] \\ \text{subject to} \quad & \boldsymbol{s}^{(l+1)} = f_l(\boldsymbol{s}^{(l)}, \mu_l(\boldsymbol{s}^{(l)}), \boldsymbol{W}^{(l)}) \quad , l = k, \dots K-1 \end{aligned} \tag{1.38}$$

In Equation (1.38), the policy $\pi_k = (\mu_k, \dots, \mu_{K-1})$ is a *truncated policy*. The existence of sub-problems is another characteristic for dynamic programming. The following section will show that a dynamic program may be solved by breaking it down into easier sub-problems and solving them in succession. This is possible because of the *principle of optimality* [5]. The following section is dedicated on the study of that principle.

1.2.2 Bellman's principle of optimality

In 1957, Bellman stated the often quoted principle of optimality:

> The Principle of Optimality. An optimal policy has the property that whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision. [5, p. 83]

This rather general definition is expressed more formally using Definitions 1.25 and 1.26 in the following statement: If the policy $\pi^* = (\mu_0^*, \mu_1^*, \dots, \mu_{K-1}^*)$ is an optimal policy for the dynamic programming problem in Equation (1.37), then the policy $\pi_k^* = (\mu_k^*, \dots, \mu_{K-1}^*)$ is an optimal policy for the dynamic programming sub-problem in Equation (1.38) [6]. This means that the problem in Equation (1.37) has an optimal substructure.

Let us use a shorter notation for the cost function of the sub-problem denoted as $J_k(\boldsymbol{s}^{(k)}) := J_{\pi_k}(\boldsymbol{s}^{(k)})$ from now on. Let us use a similar notation for the optimal solution to the sub-problem too, denoted as $J_k^*(\boldsymbol{s}^{(k)}) := J_{\pi_k^*}(\boldsymbol{s}^{(k)})$. The following theorem states the principle of optimality formally.

Theorem 1.27 (Bellman's principle of optimality)**.** *Let the policy $\pi^* = (\mu_0^*, \mu_1^*, \dots, \mu_{K-1}^*)$ minimize the problem in Equation (1.37). Then the truncated policy $\pi_k^* = (\mu_k^*, \mu_{k+1}^*, \dots, \mu_{K-1}^*)$ minimizes the sub-problem in Equation (1.38).*

Proof. This theorem is proven for finite planning horizon $k = 1, \dots, K$ and discrete random vectors $\boldsymbol{W}^{(0)}, \dots, \boldsymbol{W}^{(K-1)}$. The policy π^* is optimal, which

means that $J_{\pi^*}(\boldsymbol{s}^{(0)}) \leq J_{\pi'}(\boldsymbol{s}^{(0)})$ for all $\pi' \in \Pi$. Let us prove this theorem using a contradiction. Let us assume that the policy π_k^* is a sub-optimal policy for sub-problem in Equation (1.38). The sub-optimality of the policy π_k^* means that there exists some policy $\pi_k' = (\mu_k', \ldots, \mu_{K-1}') \in \Pi_k$ that satisfies the inequality

$$J_{\pi_k'}(\boldsymbol{s}^{(k)}) \leq J_{\pi_k^*}(\boldsymbol{s}^{(k)}) \tag{1.39}$$

for all possible states $\boldsymbol{s}^{(k)} \in S_k$. In addition, the strict inequality in Equation (1.39) holds for at least one state $\boldsymbol{s}^{(k)}$ that occurs with positive probability. This assumption will lead to a contradiction.

Let $p(\boldsymbol{w}^{(i)}, \boldsymbol{w}^{(j)})$ denote the joint probability. The joint probability is defined as $p(\boldsymbol{w}^{(i)}, \boldsymbol{w}^{(j)}) := \mathbb{P}(\boldsymbol{W}^{(i)} = \boldsymbol{w}^{(i)} \cap \boldsymbol{W}^{(j)} = \boldsymbol{w}^{(j)})$. Because the disturbances are independent of each other, the equality $p(\boldsymbol{w}^{(i)}, \boldsymbol{w}^{(j)}) = p(\boldsymbol{w}^{(i)})p(\boldsymbol{w}^{(j)})$ holds for all $i, j = 0, 1, \ldots, K-1$ where $i \neq j$ and for all $\boldsymbol{w}^{(i)} \in D_i, \boldsymbol{w}^{(j)} \in D_j$.

The state $\boldsymbol{s}^{(k+1)}$ is a random vector that is dependent on $\boldsymbol{W}^{(k)}$. This follows from the evolution of the system $\boldsymbol{s}^{(k+1)} = f_k(\boldsymbol{s}^{(k)}, \boldsymbol{u}^{(k)}, \boldsymbol{W}^{(k)})$. By recursion, this means that the value of a cost functional $g_k(\boldsymbol{s}^{(k)}, \boldsymbol{u}^{(k)}, \boldsymbol{W}^{(k)})$ depends on the previous random disturbances $\boldsymbol{W}^{(0)}, \ldots, \boldsymbol{W}^{(k)}$. On the other hand, the value of the cost functional $g_k(\boldsymbol{s}^{(k)}, \boldsymbol{u}^{(k)}, \boldsymbol{W}^{(k)})$ does not depend on any future disturbance $\boldsymbol{W}^{(l)}$ for $k < l$ as the random disturbances are independent of each other. This means that the expected value of the cost functional is $\mathbb{E}_{\boldsymbol{W}^{(l)}} g_k(\boldsymbol{s}^{(k)}, \boldsymbol{u}^{(k)}, \boldsymbol{W}^{(k)}) = g_k(\boldsymbol{s}^{(k)}, \boldsymbol{u}^{(k)}, \boldsymbol{W}^{(k)}) \sum_{\boldsymbol{w}^{(l)}} p(\boldsymbol{w}^{(l)}) = g_k(\boldsymbol{s}^{(k)}, \boldsymbol{u}^{(k)}, \boldsymbol{W}^{(k)})$ for all $k < l$.

The value of the dynamic programming cost function for the optimal policy π^* is

$$\begin{aligned} J_{\pi^*}(\boldsymbol{s}^{(0)}) &= \mathop{\mathbb{E}}_{\boldsymbol{W}^{(0)}, \ldots, \boldsymbol{W}^{(K-1)}} \left[g_K(\boldsymbol{s}^{(K)}) + \sum_{l=0}^{K-1} g_k(\boldsymbol{s}^{(l)}, \mu_l^*(\boldsymbol{s}^{(l)}), \boldsymbol{W}^{(l)}) \right] \\ &= \sum_{\boldsymbol{w}^{(0)}, \ldots, \boldsymbol{w}^{(K-1)}} p(\boldsymbol{w}^{(0)}), \ldots, p(\boldsymbol{w}^{(K-1)}) \Bigg[g_K(\boldsymbol{s}^{(K)}) \\ &\quad + \sum_{l=0}^{K-1} g_k(\boldsymbol{s}^{(l)}, \mu_l^*(\boldsymbol{s}^{(l)}), \boldsymbol{w}^{(l)}) \Bigg]. \end{aligned} \tag{1.40}$$

The second equality in Equation (1.40) follows from the independence of the random disturbances. Since the values of the cost functionals are independent of future disturbances, the right side of Equation (1.40) can be computed in

pieces such as

$$J_{\pi^*}(\boldsymbol{s}^{(0)}) = \sum_{\boldsymbol{w}^{(0)},\dots,\boldsymbol{w}^{(k-1)}} p(\boldsymbol{w}^{(0)}),\dots,p(\boldsymbol{w}^{(k-1)}) \Bigg[\sum_{l=0}^{k-1} g_l(\boldsymbol{s}^{(l)}, \mu_l^*(\boldsymbol{s}^{(l)}), \boldsymbol{w}^{(l)}) + \sum_{\boldsymbol{w}^{(k)},\dots,\boldsymbol{w}^{(K-1)}} p(\boldsymbol{w}^{(k)}),\dots,p(\boldsymbol{w}^{(K-1)}) \Bigg[g_K(\boldsymbol{s}^{(K)}) + \sum_{l=k}^{K-1} g_l(\boldsymbol{s}^{(l)}, \mu_l^*(\boldsymbol{s}^{(l)}), \boldsymbol{w}^{(l)}) \Bigg] \Bigg]. \tag{1.41}$$

Again, using the assumption of independent disturbances, Equation (1.41) can be rewritten using expected values. Then the nested expected value is nothing else than the optimal value of the sub-problem in Equation (1.38). Now the optimal value may be written using the sub-problem value such that

$$\begin{aligned} J_{\pi^*}(\boldsymbol{s}^{(0)}) &= \mathop{\mathbb{E}}_{\boldsymbol{W}^{(0)},\dots,\boldsymbol{W}^{(k-1)}} \Bigg[\sum_{l=0}^{k-1} g_l(\boldsymbol{s}^{(l)}, \mu_l^*(\boldsymbol{s}^{(l)}), \boldsymbol{W}^{(l)}) \\ &\quad + \mathop{\mathbb{E}}_{\boldsymbol{W}^{(k)},\dots,\boldsymbol{W}^{(K-1)}} \Bigg[g_K(\boldsymbol{s}^{(K)}) + \sum_{l=k}^{K-1} g_l(\boldsymbol{s}^{(l)}, \mu_l^*(\boldsymbol{s}^{(l)}), \boldsymbol{W}^{(l)}) \Bigg] \Bigg] \\ &= \mathop{\mathbb{E}}_{\boldsymbol{W}^{(0)},\dots,\boldsymbol{W}^{(k-1)}} \Bigg[\sum_{l=0}^{k-1} g_l(\boldsymbol{s}^{(l)}, \mu_l^*(\boldsymbol{s}^{(l)}), \boldsymbol{W}^{(l)}) + J_{\pi_k^*}(\boldsymbol{s}^{(k)}) \Bigg]. \end{aligned} \tag{1.42}$$

Using the linearity of the expected value, the following inequality holds:

$$\begin{aligned} &J_{\pi^*}(\boldsymbol{s}^{(0)}) \\ &\quad = \mathop{\mathbb{E}}_{\boldsymbol{W}^{(0)},\dots,\boldsymbol{W}^{(k-1)}} \Bigg[\sum_{l=0}^{k-1} g_l(\boldsymbol{s}^{(l)}, \mu_l^*(\boldsymbol{s}^{(l)}), \boldsymbol{W}^{(l)}) \Bigg] + \mathop{\mathbb{E}}_{\boldsymbol{W}^{(0)},\dots,\boldsymbol{W}^{(k-1)}} J_{\pi_k^*}(\boldsymbol{s}^{(k)}) \\ &\quad > \mathop{\mathbb{E}}_{\boldsymbol{W}^{(0)},\dots,\boldsymbol{W}^{(k-1)}} \Bigg[\sum_{l=0}^{k-1} g_l(\boldsymbol{s}^{(l)}, \mu_l^*(\boldsymbol{s}^{(l)}), \boldsymbol{W}^{(l)}) \Bigg] + \mathop{\mathbb{E}}_{\boldsymbol{W}^{(0)},\dots,\boldsymbol{W}^{(k-1)}} J_{\pi_k'}(\boldsymbol{s}^{(k)}). \end{aligned} \tag{1.43}$$

The last inequality is strict since there is at least one state $\boldsymbol{s}^{(k)}$ that corresponds to a probability $p(\boldsymbol{w}^{(0)},\dots,\boldsymbol{w}^{(k-1)}) > 0$ with $J_{\pi_k'}(\boldsymbol{s}^{(k)}) < J_{\pi_k^*}(\boldsymbol{s}^{(k)})$. This follows from the assumption of the policy π_k^* being sub-optimal. From Equation (1.43), it can be see that the policy

$$\pi^l = (\mu_0^*,\dots,\mu_{k-1}^*,\mu_k',\dots,\mu_{K-1}') \tag{1.44}$$

results in a lower over all cost function value $J_{\pi^l}(\boldsymbol{s}^{(0)}) < J_{\pi^*}(\boldsymbol{s}^{(0)})$. This is a contradiction, since the policy π^* is optimal policy. This proves Bellman's principle of optimality. □

Bellman's principle of optimality makes it possible to divide the dynamic program in Equation (1.37) into sub-problems. The reasoning behind this is that if the sub-problems are not solved optimally, then the solution to the dynamic program cannot be optimal. This strategy efficient because each sub-problem is a dynamic program. This means that the sub-problems may be divided further by using Bellman's principle of optimality. This recursion can then be used until solving the sub-problems becomes trivial. This idea is presented formally in Algorithm 1.

Algorithm 1 Backward Dynamic Programming Algorithm

1: $J_K(\boldsymbol{s}^{(K)}) = g_K(s^{(K)})$
2: **for all** $k = K-1, \ldots, 1, 0$ **do**
3: $J_k(\boldsymbol{s}^{(k)}) = \min_{\boldsymbol{u}^{(k)} \in U_k(\boldsymbol{s}^{(k)})} \mathbb{E}_{\boldsymbol{W}^{(k)}}[g_k(\boldsymbol{s}^{(k)}, \boldsymbol{u}^{(k)}, \boldsymbol{W}^{(k)}) + J_{k+1}(f_k(\boldsymbol{s}^{(k)}, \boldsymbol{u}^{(k)}, \boldsymbol{W}^{(k)}))]$
4: **end for**

Algorithm 1 starts by solving the trivial problems $J_K(\boldsymbol{s}^{(K)})$. These results are then used to solve the preceding sub-problems. The following theorem proves that Algorithm 1 yields an optimal solution, which is $J_k^*(\boldsymbol{s}^{(k)}) = J_k(\boldsymbol{s}^{(k)})$.

Theorem 1.28. *Let the random disturbances come from domains D_k that are finite or countable. Let $\mathbb{E}[g_k(\boldsymbol{s}^{(k)}, \mu_k(\boldsymbol{s}^{(k)}), \boldsymbol{w}^{(k)})] < \infty$ for all admissible policies $\pi_k = (\mu_k, \ldots, \mu_K)$ with decision rules $\mu_k(\boldsymbol{s}^{(k)}) \in U_k(\boldsymbol{s}^{(k)})$. Then the value $J_k(\boldsymbol{s}^{(k)})$ in Algorithm 1 is the optimal cost function value for the program in Equation 1.38. Let $\boldsymbol{u}^{(k)^*}$ be the solution to the kth minimization problem in Algorithm 1. If there exist optimal decisions satisfying identity $\boldsymbol{u}^{(k)^*} \equiv \mu_k^*(\boldsymbol{s}^{(k)})$, then $\pi^* = (\mu_0^*, \mu_1^*, \ldots, \mu_{K-1}^*)$ is the optimal policy to the program in Equation 1.37.*

Proof. Theorem 1.28 is proven by using induction. The case where $k = K$ is trivial since there is no action to take. It means that $J_{\pi_K}(\boldsymbol{s}^{(K)}) = \mathbb{E}[g_K(\boldsymbol{s}^{(K)})] = g_K(\boldsymbol{s}^{(K)})$ with the optimal policy being $\pi_K = \emptyset$. Let us now consider the cases where $k < K$.

Let us assume that the Algorithm 1 gives an optimal solution for some $k+1$. In other words, let $J_{k+1}(\boldsymbol{s}^{(k+1)}) = J_{k+1}^*(\boldsymbol{s}^{(k+1)})$. For the index k, the

optimal value $J_k^*(\boldsymbol{s}^{(k)})$ for the sub-problem in Equation (1.38) is

$$J_k^*(\boldsymbol{s}^{(k)}) = \min_{(\mu_k, \pi_{k+1}) \in \Pi_k} \mathbb{E}\left[g_K(\boldsymbol{s}^{(K)}) + \sum_{l=k}^{K-1} g_l(\boldsymbol{s}^{(l)}, \mu_l(\boldsymbol{s}^{(l)}), \boldsymbol{W}^{(l)}) \right] \tag{1.45}$$

by Definition 1.26. In Equation (1.45), a shorter notation is used for the policy $(\mu_k, \pi_{k+1}) := (\mu_k, \mu_{k+1}, \ldots, \mu_{K-1}) = \pi_k$. This shorter notation highlights the relation between the cost functions J_{π_k} and $J_{\pi_{k+1}}$ that are the cost functions of the successive sub-problems.

Let us use Bellman's principle of optimality to Equation (1.45): If the policy (μ_k^*, π_{k+1}^*) minimizes the cost function $J_{\pi_k}(\boldsymbol{s}^{(k)})$, then the policy π_{k+1}^* minimizes the cost function $J_{\pi_{k+1}}(\boldsymbol{s}^{(k)})$. This means that

$$\begin{aligned} J_k^*(\boldsymbol{s}^{(k)}) = \min_{\mu_k} \mathop{\mathbb{E}}_{\boldsymbol{W}^{(k)}} \Bigg[& g_k(\boldsymbol{s}^{(k)}, \mu_k(\boldsymbol{s}^{(k)}), \boldsymbol{W}^{(k)}) \\ & + \min_{\pi_{k+1} \in \Pi_{k+1}} \mathop{\mathbb{E}}_{\boldsymbol{W}^{(k+1)}, \ldots, \boldsymbol{W}^{(K-1)}} \Bigg[g_K(\boldsymbol{s}^{(K)}) \\ & + \sum_{l=k+1}^{K-1} g_l(\boldsymbol{s}^{(l)}, \mu_l(\boldsymbol{s}^{(l)}), \boldsymbol{W}^{(l)}) \Bigg] \Bigg] \\ = \min_{\mu_k} \mathop{\mathbb{E}}_{\boldsymbol{W}^{(k)}} & \left[g_k(\boldsymbol{s}^{(k)}, \mu_k(\boldsymbol{s}^{(k)}), \boldsymbol{W}^{(k)}) + J_{k+1}^*(\boldsymbol{s}^{(k+1)}) \right] \end{aligned} \tag{1.46}$$

where $\mu_k(\boldsymbol{s}^{(k)}) \in U_k(\boldsymbol{s}^{(k)})$. In Equation (1.46), the state at the time $k+1$ can be written as $\boldsymbol{s}^{(k+1)} = f_k(\boldsymbol{s}^{(k)}, \boldsymbol{u}^{(k)}, \boldsymbol{W}^{(k)})$. Then, for any state $\boldsymbol{s}^{(k)}$, the expression on the right side of Equation (1.46) is minimized by minimizing over the action $\boldsymbol{u}^{(k)}$. Now, by the induction assumption $J_{k+1}(\boldsymbol{s}^{(k+1)}) = J_{k+1}^*(\boldsymbol{s}^{(k+1)})$, the equation in Step 3 of Algorithm 1 holds since

$$\begin{aligned} J_k^*(\boldsymbol{s}^{(k)}) &= \min_{\boldsymbol{u}_k \in U_k(\boldsymbol{s}^{(k)})} \mathop{\mathbb{E}}_{\boldsymbol{W}^{(k)}} \left[g_k(\boldsymbol{s}^{(k)}, \boldsymbol{u}^{(k)}, \boldsymbol{W}^{(k)}) + J_{k+1}(f_k(\boldsymbol{s}^{(k)}, \boldsymbol{u}^{(k)}, \boldsymbol{W}^{(k)})) \right] \\ &= J_k(\boldsymbol{s}^{(k)}). \end{aligned} \tag{1.47}$$

Thus, by induction, Algorithm 1 gives the optimal solution at each index k such that $J_k(\boldsymbol{s}^{(k)}) = J_k^*(\boldsymbol{s}^{(k)})$. □

1.2.3 Deterministic discrete-time dynamic system

Deterministic discrete-time dynamic system is a special case of the system presented in Definition 1.22. In the absence of the disturbance, the discrete-time dynamic system is deterministic. This means that the next state is

determined fully by the current state and the action taken, such as $\boldsymbol{s}^{(k+1)} = f_k(\boldsymbol{s}^{(k)}, \boldsymbol{u}^{(k)})$. In this case, the cost functional is fixed on a fixed state and a fixed action. Instead of the cost functional, a transition cost is used. The transition cost is defined as $a^k_{\boldsymbol{s}^{(k)} \to \boldsymbol{s}^{(k+1)}} = g_k(\boldsymbol{s}^{(k)}, \boldsymbol{u}^{(k)})$ with $\boldsymbol{u}^{(k)} \in U_k(\boldsymbol{s}^{(k)})$ for all $k = 0, 1, \ldots, K-1$ and $a^K_{\boldsymbol{s}^{(K)} \to \boldsymbol{s}^{(K+1)}} = g_k(\boldsymbol{s}^{(K)})$.

To be precise, the transition cost is not well defined if two different actions on a same state result in a same successive state $\boldsymbol{s}^{(k+1)} = f_k(\boldsymbol{s}^{(k)}, \boldsymbol{u}^{(k)}) = f_k(\boldsymbol{s}^{(k)}, \boldsymbol{v}^{(k)})$ for some $\boldsymbol{u}^{(k)}, \boldsymbol{v}^{(k)} \in U_k(\boldsymbol{s}^{(k)})$ with $\boldsymbol{u}^{(k)} \neq \boldsymbol{v}^{(k)}$. In these cases, let us select $a^k_{\boldsymbol{s}^{(k)} \to \boldsymbol{s}^{(k+1)}} = \min\{g_k(\boldsymbol{s}^{(k)}, \boldsymbol{u}^{(k)}), g_k(\boldsymbol{s}^{(k)}, \boldsymbol{v}^{(k)})\}$ because the higher cost could never be in the optimal solution.

For completeness, let us set transition costs for illegal transitions and self-transitions, too. Let $a^k_{\boldsymbol{s}^{(k)} \to \boldsymbol{s}^{(k+1)}} = \infty$ for illegal transitions with the state change $\boldsymbol{s}^{(k+1)} = f_k(\boldsymbol{s}^{(k)}, \boldsymbol{u}^{(k)})$ where $\boldsymbol{u}^{(k)} \notin U_k(\boldsymbol{s}^{(k)})$. For self-transitions, $a^k_{\boldsymbol{s}^{(k)} \to \boldsymbol{s}^{(k+1)}} = 0$ with the state change $\boldsymbol{s}^{(k)} = \boldsymbol{s}^{(k+1)}$. Both definitions apply for all $k = 0, 1, \ldots, K-1$. Now, the complete definition for the transition cost is

$$a^k_{\boldsymbol{s}^{(k)} \to \boldsymbol{s}^{(k+1)}} = \begin{cases} 0 & \text{, if } \boldsymbol{s}^{(k)} = \boldsymbol{s}^{(k+1)} \\ g_k(\boldsymbol{s}^{(k)}, \boldsymbol{u}^{(k)}) & \text{, if } \boldsymbol{u}^{(k)} \in U_k(\boldsymbol{s}^{(k)}) \\ \infty & \text{, otherwise.} \end{cases} \tag{1.48}$$

Using the transition cost notation instead of the cost functionals, cost function J can be written as

$$\begin{aligned} J_\pi(\boldsymbol{s}^{(0)}) &= \mathop{\mathbb{E}}_{\boldsymbol{W}^{(0)}, \ldots, \boldsymbol{W}^{(K-1)}} [g_K(\boldsymbol{s}^{(K)}) + \sum_{k=0}^{K-1} g_k(\boldsymbol{s}^{(k)}, \mu_k(\boldsymbol{s}^{(k)}))] \\ &= g_K(\boldsymbol{s}^{(K)}) + \sum_{k=0}^{K-1} g_k(\boldsymbol{s}^{(k)}, \mu_k(\boldsymbol{s}^{(k)})) \\ &= a^K_{\boldsymbol{s}^{(K)} \to \boldsymbol{s}^{(K+1)}} + \sum_{k=0}^{K-1} a^k_{\boldsymbol{s}^{(k)} \to \boldsymbol{s}^{(k+1)}}. \end{aligned} \tag{1.49}$$

The expected value over the random disturbances in Equation (1.49) can also be removed because in a deterministic system, the cost functionals do not depend on any random disturbances.

Knowing the transition costs $a^k_{\boldsymbol{s}^{(k)} \to \boldsymbol{s}^{(k+1)}}$, finding the optimal policy is a minimization problem, or more specifically, a *shortest path problem*:

Definition 1.29. Let the transition costs be defined as in Equation (1.48). Then the shortest path problem is

$$\underset{(\boldsymbol{s}^{(0)},\ldots,\boldsymbol{s}^{(K)})}{\text{minimize}} \quad J_\pi(\boldsymbol{s}^{(0)}) = a^K_{\boldsymbol{s}^{(K)}\to\boldsymbol{s}^{(K+1)}} + \sum_{k=0}^{K-1} a^k_{\boldsymbol{s}^{(k)}\to\boldsymbol{s}^{(k+1)}}. \tag{1.50}$$

The shortest path problem in Definition 1.29 may be solved recursively using backward dynamic programming algorithm:

$$\begin{cases} J_K(\boldsymbol{s}^{(K)}) & = g_K(\boldsymbol{s}^{(K)}) = a^K_{\boldsymbol{s}^{(K)}\to\boldsymbol{s}^{(K+1)}} \\ J_k(\boldsymbol{s}^{(k)}) & = \min[g_k(\boldsymbol{s}^{(k)},\boldsymbol{u}^{(k)}) + J_{k+1}(\boldsymbol{s}^{(k+1)})] \\ & = \min[a^k_{\boldsymbol{s}^{(k)}\to\boldsymbol{s}^{(k+1)}} + J_{k+1}(\boldsymbol{s}^{(k+1)})] \end{cases} \tag{1.51}$$

with an optimal solution $J_0(\boldsymbol{s}^{(0)}) = \min[a^0_{\boldsymbol{s}^{(0)}\to\boldsymbol{s}^{(1)}} + J_1(\boldsymbol{s}^{(1)})]$.

Remark 1.30. Theorem 1.28 requires the cost functional values $g_k(\boldsymbol{x}^{(k)},\boldsymbol{u}^{(k)})$ to be finite. In this section, however, the transition costs are defined to be infinite for illegal transitions. This definition was chosen only to keep the notation more simple. For the same results, the transition costs do not need to be infinite for illegal transitions. A large enough value M would suffice. Let the constant M be remarkably greater than the sum of all the legal transitions, defined as $\sum_{k=0}^{K-1}\sum_{\boldsymbol{s}^{(k)}\in S_k}\sum_{\boldsymbol{u}^{(k)}\in U_k(\boldsymbol{s}^{(k)})} g_k(\boldsymbol{s}^{(k)},\boldsymbol{u}^{(k)}) \ll M < \infty$. Let the transition cost be $a^k_{\boldsymbol{s}^{(k)}\to\boldsymbol{s}^{(k+1)}} = M$ for illegal transitions where $\boldsymbol{s}^{(k+1)} = f_k(\boldsymbol{s}^{(k)},\boldsymbol{u}^{(k)})$ and $\boldsymbol{u}^{(k)} \notin U_k(\boldsymbol{s}^{(k)})$. Now the algorithm in Equation (1.51) would never pick an illegal transition if there is a legal transition available. Also, if $J_0(\boldsymbol{s}^{(0)}) \geq M$, it means that no admissible policy exists. With this remark in mind, the solution in Equation (1.51) is can be seen as optimal by Theorem 1.28.

Since there are no disturbances, and every state can be calculated from the actions taken, the shortest path problem may be reversed. In a reversed problem, the state $\boldsymbol{s}^{(0)}$ is the final state, and the fictitious state $\boldsymbol{s}^{(K+1)}$ is the first state of the system. Let us define transition costs as $a^{K-k}_{\boldsymbol{s}^{(k+1)}\to\boldsymbol{s}^{(k)}} := a^k_{\boldsymbol{s}^{(k)}\to\boldsymbol{s}^{(k+1)}}$. These are the costs for reversed transitions, and they are equal to the original, non-reversed costs. Let us also define states $\boldsymbol{t}^{(k)} := \boldsymbol{s}^{(K-k+1)}$, which means that $\boldsymbol{s}^{(l)} = \boldsymbol{t}^{(K-l+1)}$. The reversed shortest path problem is

$$\underset{(\boldsymbol{s}^{(1)},\ldots,\boldsymbol{s}^{(K+1)})}{\text{minimize}} \quad \widetilde{J}_\pi(\boldsymbol{s}^{(K+1)}) = a^K_{\boldsymbol{s}^{(1)}\to\boldsymbol{s}^{(0)}} + \sum_{k=1}^{K} a^{K-k}_{\boldsymbol{s}^{(k+1)}\to\boldsymbol{s}^{(k)}}, \tag{1.52}$$

which is equivalent to

$$\underset{(\boldsymbol{t}^{(0)},\ldots,\boldsymbol{t}^{(K)})}{\text{minimize}} \quad \widetilde{J}_\pi(\boldsymbol{t}^{(0)}) = a^K_{\boldsymbol{t}^{(K)}\to\boldsymbol{t}^{(K+1)}} + \sum_{k=0}^{K-1} a^k_{\boldsymbol{t}^{(k)}\to\boldsymbol{t}^{(k+1)}}. \tag{1.53}$$

The minimization problem in Equation (1.53) has been solved in Equation (1.51). When the replacement $\boldsymbol{t}^{(k)} = \boldsymbol{s}^{K-k+1}$ is used, the result is the forward dynamic programming algorithm.

$$\begin{cases} \widetilde{J}_K(\boldsymbol{s}^{(1)}) & = a^K_{\boldsymbol{s}^{(1)}\to\boldsymbol{s}^{(0)}} = a^0_{\boldsymbol{s}^{(0)}\to\boldsymbol{s}^{(1)}} \\ \widetilde{J}_k(\boldsymbol{s}^{(K-k+1)}) & = \min[a^k_{\boldsymbol{s}^{(K-k+1)}\to\boldsymbol{s}^{(K-k)}} + \widetilde{J}_{k+1}(\boldsymbol{s}^{(K-k)})] \\ & = \min[a^{K-k}_{\boldsymbol{s}^{(K-k)}\to\boldsymbol{s}^{(K-k+1)}} + \widetilde{J}_{k+1}(\boldsymbol{s}^{(K-k)})] \end{cases} \tag{1.54}$$

with an optimal solution $\widetilde{J}_0(\boldsymbol{s}^{(K+1)}) = \min[a^K_{\boldsymbol{s}^{(K)}\to\boldsymbol{s}^{(K+1)}} + J_1(\boldsymbol{s}^{(K)})]$. This algorithm is presented formally in Algorithm 2.

Algorithm 2 forward dynamic programming algorithm

1: $\widetilde{J}_K(\boldsymbol{s}^{(1)}) = a^0_{\boldsymbol{s}^{(0)}\to\boldsymbol{s}^{(1)}}$
2: **for all** $k = K-1,\ldots,1,0$ **do**
3: $\quad \widetilde{J}_k(\boldsymbol{s}^{(K-k+1)}) = \min_{\boldsymbol{s}^{(K-k)}}[a^{K-k}_{\boldsymbol{s}^{(K-k)}\to\boldsymbol{s}^{(K-k+1)}} + \widetilde{J}_{k+1}(\boldsymbol{s}^{(K-k)})]$
4: **end for**

2 Time Series

Time series are a sequence of observations made over time. Some examples of time series are market capitalization indices, such as S&P 500, electrocardiograph and audio files. Ding et al. [17] define time series in discrete time as a set of pairs of time stamps and observations.

Definition 2.1. Let $x_i \in \mathbb{R}$ be an observation that occurs at time t_i. Time series is a set of pairs $\boldsymbol{T} = \{(x_i, t_i)\}_{(i=1)}^n$ with $t_i < t_j$ for all $i < j$.

The time series $\boldsymbol{T}$ is a sample from some continuous signal. Usually, the sampling rate is assumed to be fixed [19, 24, 71], so that $\Delta t_i := t_i - t_{i-1}$ is constant for all i. In these cases the time stamp is usually irrelevant and may be dropped. With this assumption, the time series has the following representation:

Definition 2.2. Univariate time series $\boldsymbol{X} = (x_1,\ldots,x_n)$ is an ordered sequence of data points or a vector.

The series $\boldsymbol{X}$ defined this way is called a *univariate time series* (UTS). A data point x_i is usually called a feature or a dimension. In this thesis, however, it is called an observation or a dimension to avoid confusion when talking about feature subset selection in section 2.2.

There may be multiple simultaneous measurements for capturing different aspects of a single event. This kind of arrangement creates multiple time series that are logically connected by the event. These time series can be processed as one *multivariate time series* (MTS). MTS are defined using the definition of UTS in Definition 2.2:

Definition 2.3. Multivariate time series is a collection of univariate time series $\boldsymbol{X} = (\boldsymbol{X}^{(1)}, \ldots, \boldsymbol{X}^{(m)})$ with $\boldsymbol{X}^{(j)}$ being a univariate time series of length n for all $j = 1, \ldots, m$.

In this thesis, the UTS $\boldsymbol{X}^{(j)}$ in Definition 2.3 is called a *feature* of the MTS. The MTS in Definition 2.3 may also be presented as a matrix $\boldsymbol{X} \in \mathbb{R}^{n*m}$. It is also possible to present it as a sequence of multivariate observations $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ similarly to Definition 2.2.

Classification of time series data poses multiple challenges. First, time series may have different lengths. This means that the vectors to be compared may be of different dimension. Second, if MTSs consist of multiple features, classification may suffer from so-called *curse of dimensionality* [18]. Third, time series are collections of observations that are usually highly correlated to the previous and subsequent observations. This means that there are multiple ways to define distance between time series. Some distances represent the difference between time series well, while other distances may not.

The following sections will review some of the methods to tackle these challenges. Section 2.1 is about preprocessing of time series. Section 2.2 presents methods to reduce the number of features to tackle the curse of dimensionality. In section 2.3, some commonly used distance measures to compare the time series are discussed.

2.1 Preprocessing of time series data

There are two preprocessing steps that the time series generally require: resampling and Z-normalization. Resampling transforms the time series into desired length [48]. Z-normalization eliminates the scale and offset differences between the time series [4]. Both of these methods are designed for UTSs. They both, however, can be generalized to MTSs by performing them on each feature one at the time.

2.1.1 Natural cubic spline interpolation and resampling

Time series resampling consists of two steps: interpolation and the actual resampling. Interpolation step approximates the continuous signal behind the discrete time series data. In this thesis, the choice for the interpolation method is cubic spline. Cubic spline has a property called *smoothness*. Many signals in real world are smooth. Let us now define cubic spline:

Definition 2.4. Cubic spline is a piecewise smooth function $S : [t_1, t_n] \to \mathbb{R}$. Its value at the time stamp t is

$$S(t) = \begin{cases} C_1(t) & , \text{if } t_1 \leq t < t_2 \\ \dots & \\ C_i(t) & , \text{if } t_i \leq t < t_{i+1} \\ \dots & \\ C_{n-1}(t) & , \text{if } t_{n-1} \leq t \leq t_n \end{cases} \tag{2.1}$$

where functions $C_i(t) = d_i t^3 + c_i t^2 + b_i t + a_i$ are third degree polynomials with $d_i, c_i, b_i, a_i \in \mathbb{R}$ and $d_i \neq 0$ for all $i = 1, \dots, n-1$.

In this context, smooth means that the second derivative S'' is continuous on the interval $(t_1, t_n) \subset \mathbb{R}$. The cubic spline in Equation (2.1) consists of $n-1$ third degree polynomials $C_1, \dots, C_{n-1}$. This means that there are $4(n-1)$ coefficients to solve.

Cubic spline interpolates the time series $\boldsymbol{X} = \{(x_i, t_i)\}_{i=1}^n$ if

$$C_i(t_i) = x_i \text{ and } C_i(t_{i+1}) = x_{i+1} \tag{2.2}$$

for all polynomials $C_i : \mathbb{R} \to \mathbb{R}$ with $i = 1, \dots, n-1$. Cubic spline must also be smooth. This means that the first and the second derivatives must be continuous. The smoothness condition is ensured by constraints

$$C_i'(t_{i+1}) = C_{i+1}'(t_{i+1}) \tag{2.3}$$

$$C_i''(t_{i+1}) = C_{i+1}''(t_{i+1}) \tag{2.4}$$

for all $i = 1 \dots, n-2$. In total, there are $4(n-1)-2$ constraints in Equations (2.2), (2.3) and (2.4). A good choice for the two missing constraints is

$$C_1''(t_1) = C_{n-1}''(t_n) = 0 \tag{2.5}$$

if the time series to be interpolated is not known to be periodic. With this constraint in Equation (2.5) the spline is *natural cubic spline*.

The next step is the actual resampling. The time series $\boldsymbol{X}$ can be resampled into any length n' by using the natural cubic spline interpolation. Let $t'_1 = t_1$, $t'_{n'} = t_n$ and $t'_j \in (t_1, t_n)$ with $t'_j \neq t'_k$ for all $j, k = 2, 3, \ldots n' - 1$. These are the new time stamps for the resampled time series. The resampled time series is then

$$\boldsymbol{X}' = \{(S(t'_j), t'_j)\}_{j=1}^{n'}. \tag{2.6}$$

2.1.2 Z-normalization

In most cases, each time series must be Z-normalized so that comparing them would be meaningful [28]. The Z-normalization can handle the scale invariance and offset [54]. For example, in the Australian sign language dataset [27], some people might have wider range of movement when signing the same sign. The Z-normalization is defined as

$$\boldsymbol{z_i} = \frac{\boldsymbol{x}_i - \mu}{\mathrm{Std}(\boldsymbol{X})} \tag{2.7}$$

for all $i = 1, \ldots, n$. In Equation (2.7), μ denotes the mean of the time series $\boldsymbol{X}$ and $\mathrm{Std}(\boldsymbol{X})$ is its standard deviation.

2.2 Feature Subset Selection of Time Series

In this thesis, a MTS is represented as a matrix $\boldsymbol{X} \in \mathbb{R}^{n*m}$ with n being the length of the time series, and m is the number of features. The features are the individual UTS that constitute the MTS instance in Definition 2.3. Feature subset selection is one important step in time series classification pipeline. Feature subset selection reduces the number of features by selecting the best features from all the m features of the MTS.

Another way to compress the time series is to use dimensionality reduction. This means that the number of observations is reduced in each feature from the total of n dimensions. This is usually done to reduce the size of the time series on a drive and to speed up the computations, such as similarity searches, in large databases [65]. These optimizations are out of the scope of this thesis, so the dimensionality reduction techniques will be omitted.

MTS data suffers from the curse of dimensionality[1]just the same as other forms of data-analysis [18]. As the m increases, the MTS tend to be more equidistant. This hurts classification. The feature subset selection is a less researched topic than the traditional dimension reduction for i.i.d data [22]. CLeVer is one of the few feature subset selection techniques in the literature. It is an unsupervised technique, which means that the data do not need to be annotated beforehand. It is used in this thesis because of this

particular property. Unsupervised learning and its relation to supervised and semi-supervised learning is discussed more in detail in Section 3. Let us first take a look at principal component analysis method since CLeVer builds on that method.

2.2.1 Common Principal Component Analysis

Principal component analysis (PCA) is a widely used method for dimensionality reduction and feature selection used in many disciplines [36]. It can be used as a feature selection method in an unsupervised setting. Let $\boldsymbol{X} \in \mathbb{R}^{n*m}$ be a matrix representing MTS. Its principal components are the eigenvectors $\boldsymbol{p}_i$ of its covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{m*m}$ [62]. The covariance between two features is

$$\mathrm{Cov}(\boldsymbol{X}^{(i)}, \boldsymbol{X}^{(j)}) = s_{ij} = \frac{1}{n-1} \sum_{k=1}^{n} (\boldsymbol{X}_k^{(i)} - \mu_i)(\boldsymbol{X}_k^{(j)} - \mu_j) \tag{2.8}$$

where μ_l is the mean of the feature $\boldsymbol{X}^{(l)}$. The covariance matrix is the matrix $\mathrm{Cov}(\boldsymbol{X}) := \boldsymbol{\Sigma} = [s_{ij}]_{i,j=1}^{m}$. The reduced dimensionality representation of matrix $\boldsymbol{X}$ is calculated as $\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{P}_k$ where $\boldsymbol{P}_k = (p_1, \ldots, p_k)^T \in \mathbb{R}^{m*k}$ is a matrix of k dominant eigenvectors of the covariance matrix $\boldsymbol{\Sigma}$, and $k < m$ [62]. In other words, the matrix $\boldsymbol{Y}$ is a projection of the matrix $\boldsymbol{X}$ to the subspace spanned by the k eigenvectors with the largest eigenvalues.

The scales of the features affect the covariance matrix $\boldsymbol{\Sigma}$. To get meaningful results, the features must be of a similar scale. One way to achieve this is to use correlations instead of covariances. In this thesis, Z-normalization is used for similar results. The Z-normalization is a natural choice here since it is a usual step in the time series classification pipeline.

Common principal component analysis (CPCA) is a generalization of PCA that can be used to analyse a MTS database. There are two steps to obtaining CPCA for the MTS database. First, every MTS is described by the t most important principal components. These are the eigenvectors that correspond to the largest eigenvalues. The first common principal component (CPC) is then obtained by bisecting the angles between the first principal components of each MTS. The other common principal components are obtained in a similar fashion. [69]

The CPCA algorithm is presented in Algorithm 3. In Algorithm 3, the matrix $\boldsymbol{T}(t)$ is a truncation matrix. Let us define the truncation matrix as

[1] This term is a little misleading in this context. The curse of dimensionality is caused mainly by the number of features rather than the dimensionality of a time series. This is because the observations close to each other in a time series are highly correlated.

$\boldsymbol{T}(t) = [\boldsymbol{I}_t\ \boldsymbol{0}_{t,m-t}] \in \mathbb{R}^{t*m}$. The truncation matrix selects the first t rows of matrix $\boldsymbol{U} \in \mathbb{R}^{m*m}$ according to the equation $\boldsymbol{U}_{tr} = \boldsymbol{T}(t)\boldsymbol{U}$.

Algorithm 3 Common Principal Component Analysis

Require: MTS database $\mathcal{X}$ with N time series, number of latent variables t
Ensure: Common principal component matrix $\boldsymbol{C}$

$\boldsymbol{H} = \boldsymbol{0} \in \mathbb{R}^{m*m}$
$\boldsymbol{C} = \boldsymbol{0} \in \mathbb{R}^{t*m}$
for all $\boldsymbol{X} \in \mathcal{X}$ **do**
    $\boldsymbol{Z} \leftarrow Z$-normalize $\boldsymbol{X}$
    Calculate covariance matrix $\boldsymbol{\Sigma} = Cov(\boldsymbol{Z})$
    $\boldsymbol{U}, \boldsymbol{S}, \boldsymbol{U}^T \leftarrow SVD(\boldsymbol{\Sigma})$
    Truncate matrix $\boldsymbol{U}$ according $\boldsymbol{U}_{tr} = \boldsymbol{T}(t)\boldsymbol{U}$
    $\boldsymbol{H} \leftarrow \boldsymbol{H} + \boldsymbol{U}_{tr}^T \boldsymbol{U}_{tr}$
end for
$\boldsymbol{V}, \boldsymbol{S}, \boldsymbol{V}^T \leftarrow SVD(\boldsymbol{H})$
Truncate matrix $\boldsymbol{V}$ according $\boldsymbol{V}_{tr} = \boldsymbol{T}(t)\boldsymbol{V}$
$\boldsymbol{C} \leftarrow \boldsymbol{V}_{tr}$

CPCA can be used to transform the original features into latent features. The CPCs are stored in the matrix $\boldsymbol{C}$ in Algorithm 3. The transformed time series are computed similarly to PCA such that $\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{C}^T$. The matrix $\boldsymbol{Y} \in \mathbb{R}^{n*t}$ is the transformed time series, and its features are linear combinations of the features of the original time series $\boldsymbol{X}$.

The CPCA finds the latent features of the original time series. In ideal case, CPCA compresses the relevant information from the MTS into just a few features. This, however, may not always be desirable since it makes the results harder to interpret. In this thesis, the original, untransformed features are kept for easier interpretation. The following method, CLeVer, selects the most relevant features from the original features.

2.2.2 CLeVer-family

CLeVer is one of the few feature subset selection methods designed specially for multivariate time series [22]. It is based on CPCA. The common principal components of CPCA can be used for feature ranking and clustering [69]. The CLeVer-family feature subset selection selects the features from the original features without transforming them. This makes results easier to interpret.

The CLeVer-family does not fix the number of common principal components. Instead it requires the eigenvectors of the covariance matrices to explain at least δ percent of the variance of each MTS. The number of CPCs

t_{max} is determined at the run-time. The value t_{max} is defined as the minimum number of eigenvectors that explain at least δ percent of the variance of each MTS. Algorithm 3 will be augmented so that it finds the value t_{max}. This augmented CPCA algorithm is presented in Algorithm 4.

Algorithm 4 Common Principal Component Analysis for CLeVer

Require: MTS database $\mathcal{X}$ with N time series, threshold δ
Ensure: Common principal component matrix $\boldsymbol{C}$

$\boldsymbol{H} = \boldsymbol{0} \in \mathbb{R}^{m*m}$
$t_{max} = 0$
for all $\boldsymbol{X} \in \mathcal{X}$ **do**
    $\boldsymbol{Z} \leftarrow Z$-normalize $\boldsymbol{X}$
    Calculate covariance matrix $\boldsymbol{\Sigma} = Cov(\boldsymbol{Z})$
    $\boldsymbol{U}, \boldsymbol{S}, \boldsymbol{U}^T \leftarrow SVD(\boldsymbol{\Sigma})$
    Save $\boldsymbol{U}$ in $\mathcal{U}$
    $t = \arg\min_{t=1,\ldots,m} \sum_{i=0}^{t} \boldsymbol{S}_{ii}$ such that $\sum_{i=0}^{t} \boldsymbol{S}_{ii} \geq \delta * \sum_{i=0}^{m} \boldsymbol{S}_{ii}$
    if $t > t_{max}$ **then**
        $t_{max} \leftarrow t$
    end if
end for
for all $\boldsymbol{U} \in \mathcal{U}$ **do**
    Truncate matrix $\boldsymbol{U}$ according $\boldsymbol{U}_{tr} = \boldsymbol{T}(t_{max})\boldsymbol{U}$
    $\boldsymbol{H} \leftarrow \boldsymbol{H} + \boldsymbol{U}_{tr}^T \boldsymbol{U}_{tr}$
end for
$\boldsymbol{V}, \boldsymbol{S}, \boldsymbol{V}^T \leftarrow SVD(\boldsymbol{H})$
Truncate matrix $\boldsymbol{V}$ according $\boldsymbol{V}_{tr} = \boldsymbol{T}(t_{max})\boldsymbol{V}$
$\boldsymbol{C} \leftarrow \boldsymbol{V}_{tr}$

The CLeVer-family consist of three feature subset selection methods: CLeVer-Rank, CLeVer-Cluster and CLeVer-Hybrid. All three CLeVer algorithms first calculate the CPC matrix $\boldsymbol{C}$ in Algorithm 4. The elements in the matrix $\boldsymbol{C} \in \mathbb{R}^{t_{max}*m}$ have a useful interpretation. The element $\boldsymbol{C}_{ij}$ is the contribution of the feature j to the CPC i. The CLeVer algorithms use this interpretation.

CLeVer-Rank is the most simple of the CLeVer algorithms, and it will be presented first. In CLeVer-Rank, the features are ranked based on the matrix $\boldsymbol{C}$. CLeVer-Rank assigns each feature a score s using the interpretation of the elements in the matrix $\boldsymbol{C}$. The score of the feature j is the Euclidean norm of the column j in the matrix $\boldsymbol{C}$, defined as $s_j = (\sum_{i=1}^{t_{max}} \boldsymbol{C}_{ij}^2)^{1/2}$. CLeVer-Rank then simply chooses the k highest scoring features.

CLeVer-Cluster and CLeVer-Hybrid both cluster the features based on the

columns in the matrix $\boldsymbol{C}$ in k clusters using k-means. The hypothesis is that the features in the same cluster are highly correlated since they yield similar patterns to the matrix $\boldsymbol{C}$. In CLeVer-Cluster and CLeVer-Hybrid, one feature is selected from each cluster. CLeVer-Cluster selects the feature closest to the centroid of the cluster since it can be seen as the most correlated feature to the other features in the cluster. CLeVer-Hybrid scores each feature within the cluster similarly to CLeVer-Rank method and then selects the highest scoring feature from each cluster. To recap, both CLeVer-Cluster and CLeVer-Hybrid select one feature from each cluster but the difference between them is that CLeVer-Cluster selects the feature closest to the centroid, while CLeVer-Hybrid selects the highest scoring feature within the cluster.

2.3 Distance and Similarity Measures

This section is a review of some distances commonly used for time series. A *distance* is a function $d : \mathbb{R}^m \times \mathbb{R}^m \to [0, \infty)$ that measures the dissimilarity between two objects $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^m$. For the function d to be a distance, it must satisfy three conditions:

Definition 2.5. Let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^m$. Function $d : \mathbb{R}^m \times \mathbb{R}^m \to [0, \infty)$ is a distance, if

1. $d(\boldsymbol{x}, \boldsymbol{y}) \geq 0$
2. $d(\boldsymbol{x}, \boldsymbol{y}) = 0 \Leftrightarrow \boldsymbol{x} = \boldsymbol{y}$
3. $d(\boldsymbol{x}, \boldsymbol{y}) = d(\boldsymbol{y}, \boldsymbol{x})$.

In this thesis, there are two other dissimilarity measures that are not distances. The first one is the *pseudo-distance*. It is a function d that satisfies the conditions 1 and 3 in Definition 2.5 and $d(\boldsymbol{x}, \boldsymbol{x}) = 0$. The second one is the *asymmetric dissimilarity measure* that is even more general. The asymmetric dissimilarity measure requires non-negativity according to the condition 1 in Definition 2.5 and $d(\boldsymbol{x}, \boldsymbol{x}) = 0$. On the other hand, there are also more specific distance measures than the distance in Definition 2.5. One especially useful specification is the *metric*:

Definition 2.6. Metric is a distance that satisfies the triangle inequality $d(\boldsymbol{x}, \boldsymbol{y}) \leq d(\boldsymbol{x}, \boldsymbol{z}) + d(\boldsymbol{z}, \boldsymbol{y})$ for all $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{R}^m$.

Many methods, especially those that index time series, assume triangle inequality [38]. This means that if the distance measure used is not a metric, there may occur unexpected behavior and unreliable results with many much researched methods.

There are numerous different distances designed to measure the distance of time series. One charasteristic to time series distances is the property called *elasticity* [38]. Elasticity is the ability of a distance measure to locally "stretch" time series. Non-elastic measures can only compare observations with equal time stamps or indices. Elastic measures are able to compare observations with almost equal time stamps or indices depending on the data. This procedure of comparing nearby time stamps or indices is called *time shifting* [38].

Marteau [38] defines three categories for time series distance measures:

1. non-elastic metric measures,
2. elastic measures that are not metric,
3. metric elastic measures.

Non-elastic metric measures are metrics that do not allow time shifting. Elastic measures that are not metrics allow time shifting but do not satisfy the triangle inequality in Definition 2.6. Metric elastic measures are metrics that allow time shifting.

The following sections will present three distance measures for time series. First is the Euclidean distance (ED), second is the Dynamic Time Warping (DTW) distance and third is a similarity factor based on Principal Component Analysis (PCA-SF). Of these distance measures, ED and PCA-SF are non-elastic metrics, and DTW is a non-metric elastic measure. Elastic metrics are not considered in this thesis since they depend greatly on hyperparameters that are hard to optimize in semi-supervised setting.

ED is selected because of its popularity and easy implementation. DTW is selected because of its strong mathematical foundation and the fact that it's value has been proven empirically in numerous studies [3]. PCA-SF is selected because of the noise reducing properties of PCA.

2.3.1 Euclidean Distance

Euclidean distance is the simplest distance measure for time series of equal length. It is defined between two equal length UTSs. The Euclidean distance of two time series $\boldsymbol{X}$ and $\boldsymbol{Y}$ of length n is $\mathrm{ED}(\boldsymbol{X}, \boldsymbol{Y}) = (\sum_{i=1}^{n}(x_i - y_i)^2)^{1/2}$.

The Euclidean distance can be generalized to two equal length MTSs. Let $\boldsymbol{x}_i, \boldsymbol{y}_i \in \mathbb{R}^m$ be observations at time stamp i on time series $\boldsymbol{X}$ and $\boldsymbol{Y}$, respectively. Let $d(\boldsymbol{x}_i, \boldsymbol{y}_i)$ be a distance between those observations. Then the Euclidean distance between the two time series $\boldsymbol{X}$ and $\boldsymbol{Y}$ is

$$\mathrm{ED}(\boldsymbol{X}, \boldsymbol{Y}) = \left(\sum_{i=1}^{n} d(\boldsymbol{x}_i, \boldsymbol{y}_i)^2 \right)^{1/2}. \tag{2.9}$$

MTSs commonly vary in length [24]. However, Euclidean distance cannot handle time series of unequal length. To overcome this shortcoming time series may be reinterpolated [48]. In this thesis, a natural cubic spline in Definition 2.4 is used to approximate the continuous process behind the discrete sampling. A discrete time series of desired length is then received by sampling the approximation with a fixed rate.

2.3.2 Dynamic Time Warping

Dynamic time warping (DTW) is a much-researched algorithm that measures the distance between two time series. DTW computes the distance in a time-normalized manner, and it is based on dynamic programming [52]. The time-normalized distance is optimal in the sense that DTW aligns the two time series according to their peaks and valleys. The following definition of DTW is based on [52] and [29].

Suppose sequences $\boldsymbol{X} = (x_i)_{i=1}^n$ and $\boldsymbol{Y} = (y_j)_{j=1}^m$ are time series of equal or unequal length. Let $d_{(i,j)} = d(x_i, y_j)$ be some distance measure between the observations x_i and y_j. Let a sequence $W = (w_k)_{k=1}^K$ be a *warping path*. Its elements are the pairs $w_k = (i, j)$ with some $i = 1, \ldots, n$ and $j = 1, \ldots, m$. A valid warping path W fulfils three constraints: monotonic, continuity and boundary conditions. Monotonicity means that for any successive warping path element $w_k = (i, j)$ and $w_{k-1} = (i', j')$, indices increase monotonically meaning that $i' \leq i$ and $j' \leq j$. In addition, the path must advance on every path element such that $w_k \neq w_{k-1}$. The continuity condition prevents data from disappearing. Continuity of the warping path is defined as $i - i' \leq 1$ and $j - j' \leq 1$. Finally, both time series are evaluated from the start to the end with the boundary condition, which is $w_1 = (1, 1)$ and $w_K = (n, m)$. Now, dynamic time warping distance is defined over all valid warping paths with the distances $d_{(i,j)}$ as

$$\mathrm{DTW}(\boldsymbol{X}, \boldsymbol{Y}) := \min_W \sqrt{\sum_{w \in W} {d_w}^2}. \tag{2.10}$$

From the monotonic and continuity conditions, it is easy to see that every element w_k in a valid warping path has one of the three possible relations to its predecessor $w_{k-1} = (i', j')$: either $w_k = (i' + 1, j')$, $w_k = (i', j' + 1)$ or $w_k = (i' + 1, j' + 1)$.

The count of the valid warping paths increases exponentially with respect to the lengths of the time series. The minimization in DTW measure is possible to solve in feasible time using dynamic programming. The cumulative

squared distance is defined recursively as

$$c^2(i,j) = {d_{(i,j)}}^2 + \min\{c^2(i-1,j), c^2(i,j-1), c^2(i-1,j-1)\} \tag{2.11}$$

with $c^2(1,1) = {d_{(1,1)}}^2$. The undefined cumulative squared distances are handled as if $c^2(0,j) = c^2(i,0) = \infty$. Then the DTW measure is

$$\mathrm{DTW}(\boldsymbol{X}, \boldsymbol{Y}) = \sqrt{c^2(n,m)} \tag{2.12}$$

as defined with the cumulative squared distance. It will be proven that this is an optimal solution to the minimization problem in Equation (2.10) in the following section in Theorem 2.9.

Multiple distance measures can be used between observations $\boldsymbol{x}_i$ and $\boldsymbol{y}_j$ for multivariate time series. A common measure is Euclidean distance [66] but other l_p-norm based distances can also be used [25].

It is useful to apply a global constraint, such as Sakoe-Chiba band [52], to the optimal warping path computation in Equation (2.12) [48]. The Sakoe-Chiba band constrains the distance of indices that may be compared. The index pair (i,j) is inside of the Sakoe-Chiba band if $|i-j| \leq r$ with r being a positive integer. The cumulative squared distance is undefined outside of the Sakoe-Chiba band. These values are handled as if $c^2(i,j) = \infty$ for indices $|i-j| > r$. This way, the parameter r defines a window that prevents excessive warping for the time series.

The Sakoe-Chiba band generally improves both speed of the DTW and classification accuracy as showed in [48]. An old rule-of-thumb for Sakoe-Chiba band window size r is 10% of the length of the time series, but the optimal size depends on the data and is usually less than that [48]. The window size of $r = 0$ reverts back to the Euclidean distance. This is easy to see since $|i-j| = 0$ implies that $i = j$. For equal length time series using window size $r = 0$, the cumulative distance is $c^2(i,j) = {d_{(i,j)}}^2 + \min\{c^2(i-1,j-1)\}$. The two other options for cumulative squared distance are undefined for $r = 0$ since $|i-1-j| = |i-j-1| = 1$. Then, for $r = 0$, the distance is $\mathrm{DTW}(\boldsymbol{X}, \boldsymbol{Y}) = \sqrt{c^2(n,n)} = \sqrt{{d_{(n,n)}}^2 + \cdots + {d_{(1,1)}}^2} = \mathrm{ED}(\boldsymbol{X}, \boldsymbol{Y})$. This means that Euclidean distance upper bounds DTW. The difference between Euclidean distance and DTW is illustrated in Figure 3.

It is possible to measure the distance of two time series of different length using DTW. However, if the time series are not of equal length, the DTW measure must be normalized based on the lengths of the time series. The normalization is executed as $\mathrm{DTW}(\boldsymbol{X}, \boldsymbol{Y}) = \sqrt{c^2(n,m)/N}$ with some scalar N. The choice of the normalization factor N, however, is not straightforward. Possible choices for the normalization factor are, for example, the length of

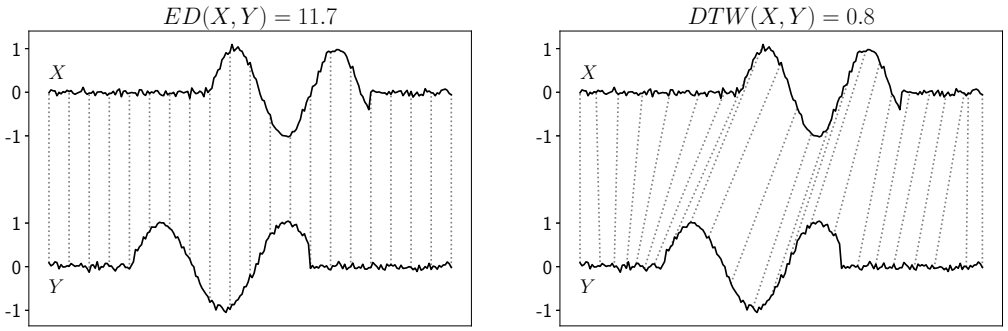

Figure 3: The difference between Euclidean distance and DTW. DTW aligns the peaks and the valleys of the time series that are out of phase. Euclidean distance does not align the time series.

the shorter time series, the length of the longer time series and the length of the optimal path. Thus, even if it is possible to measure distances of time series of different lengths, it is recommended to first resample the time series to some fixed length. Both normalization and resampling yield similar classification results. [48]

One major drawback for DTW is that it is not a metric. DTW does not satisfy the triangle inequality in Definition 2.6. It is easy prove that DTW is not a metric with a counter example. Let $\boldsymbol{X} = (1, 0, 0, 0)$, $\boldsymbol{Y} = (0, 1, 0, 0)$ and $\boldsymbol{Z} = (0, 0, 1, 0)$. The distances between the time series are $\text{DTW}(\boldsymbol{X}, \boldsymbol{Y}) = 1$, $\text{DTW}(\boldsymbol{Y}, \boldsymbol{Z}) = 0$ and $\text{DTW}(\boldsymbol{X}, \boldsymbol{Z}) = \sqrt{2}$. The calculations of these distances is presented in Table 1. The triangle inequality does not hold between the time series $\boldsymbol{X}$, $\boldsymbol{Y}$ and $\boldsymbol{Z}$ since $\text{DTW}(\boldsymbol{X}, \boldsymbol{Y}) + \text{DTW}(\boldsymbol{Y}, \boldsymbol{Z}) = 1 < \sqrt{2} = \text{DTW}(\boldsymbol{X}, \boldsymbol{Z})$. This counter example proves that DTW does not satisfy the triangle inequality.

It is also noted here that DTW is not an actual distance either by Definition 2.5. Clearly, $\boldsymbol{Y} \neq \boldsymbol{Z}$ but $\text{DTW}(\boldsymbol{Y}, \boldsymbol{Z}) = 0$. However, DTW satisfies a condition $\text{DTW}(\boldsymbol{X}, \boldsymbol{X}) = 0$ so DTW is a *pseudodistance*.

2.3.3 Dynamic Time Warping as Dynamic Programming problem

There is a strong mathematical background in dynamic programming behind the DTW algorithm. The DTW algorithm was derived from this background by Sakoe and Chiba [52]. In this section I will derive the algorithm from the theory of dynamic programming presented in Section 1.2 using similar reasoning.

Many domains produce time series that are locally out of phase. Matching these time series time stamp by time stamp may yield low similarities for the

	$d(\boldsymbol{X}_i, \boldsymbol{Y}_j)^2$			
X_4	0	1	0	**0**
X_3	0	1	**0**	0
X_2	0	1	**0**	0
X_1	**1**	**0**	1	1
	Y_1	Y_2	Y_3	Y_4

	$d(\boldsymbol{Y}_i, \boldsymbol{Z}_j)^2$			
Y_4	0	0	1	**0**
Y_3	0	0	1	**0**
Y_2	1	1	**0**	1
Y_1	**0**	**0**	1	0
	Z_1	Z_2	Z_3	Z_4

	$d(\boldsymbol{X}_i, \boldsymbol{Z}_j)^2$			
X_4	0	0	1	**0**
X_3	0	0	**1**	0
X_2	0	**0**	1	0
X_1	**1**	1	0	1
	Z_1	Z_2	Z_3	Z_4

Table 1: The DTW calculations between the time series $\boldsymbol{X}$, $\boldsymbol{Y}$ and $\boldsymbol{Z}$ according the definition in Equation (2.10). The matrices represent the squared distances between the observations in the time series. The optimal warping paths are marked with bold numbers. DTW is the square root of the sum of the elements in the optimal warping path. This means that $\text{DTW}(\boldsymbol{X}, \boldsymbol{Y}) = 1$, $\text{DTW}(\boldsymbol{Y}, \boldsymbol{Z}) = 0$ and $\text{DTW}(\boldsymbol{X}, \boldsymbol{Z}) = \sqrt{2}$.

time series even if they are produced by similar events. In these cases, for a more meaningful measure of dissimilarity, the measure must be local scaling invariant [4]. Local scaling invariance can be achieved by many-to-one and one-to-many time stamp matching.

Let the state $\boldsymbol{s}^{(k)} = (i, j)$ be an index pairing where $i = 1, \ldots, n$ is an index of time series $\boldsymbol{X}$ and $j = 1, \ldots, m$ is an index of time series $\boldsymbol{Y}$. Given a sequence of states $(\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(K)})$, a possibly asymmetric dissimilarity measure can be defined as

$$d_a(\boldsymbol{X}, \boldsymbol{Y}) = \sqrt{g(\boldsymbol{s}^{(K)}) + \sum_{k=1}^{K-1} g(\boldsymbol{s}^{(k)})}. \tag{2.13}$$

The function g is the squared distance between the observations at the time stamps i and j, and its value is $g(\boldsymbol{s}^{(k)}) = g(i, j) = d(\boldsymbol{X}_i, \boldsymbol{Y}_j)^2$.

Considering the time series $\boldsymbol{X}$ and $\boldsymbol{Y}$, there are three possibilities to match the time stamps of the time series $\boldsymbol{X}$ to the time stamps of the time series $\boldsymbol{Y}$: one-to-one, one-to-many and many-to-one. These options produce sequence prototypes $\{(i, j), (i + 1, j + 1)\}$, $\{(i, j), (i, j + 1)\}$ and $\{(i, j), (i + 1, j)\}$, respectively, assuming that the time cannot flow backwards. Each sequence of states must be composed of these prototypes. To ensure that every time stamp in both time series are matched, let us require that $\boldsymbol{s}^{(1)} = (1, 1)$ and $\boldsymbol{s}^{(K)} = (n, m)$.

A deterministic discrete time dynamic system can be defined using the sequence prototypes so that

$$\boldsymbol{s}^{(k+1)} = \boldsymbol{s}^{(k)} + \boldsymbol{u}^{(k)} \tag{2.14}$$

where the action is $\boldsymbol{u}^{(k)} \in \{(0,1),(1,1),(1,0)\}$. To ensure that the dissimilarity measure is well-defined, the action $\boldsymbol{u}^{(k)}$ must result in an admissible state $\boldsymbol{s}^{k+1} = (i,j)$ where $i = 1,\ldots,n$ and $j = 1,\ldots,m$. Let $\boldsymbol{u}^{(k)} \in U(\boldsymbol{s}^{(k)})$ and

$$U(i,j) = \begin{cases} \{(0,1)\} & ,\text{if } i = n \\ \{(1,0)\} & ,\text{if } j = m \\ \{(0,1),(1,1),(1,0)\} & ,\text{otherwise} \end{cases} \tag{2.15}$$

for all $k = 1,\ldots,K-1$.

Let us now define DTW similarly to Equation (2.10). DTW is a dissimilarity measure, or a distance function, so Equation (2.13) will be used to define it. Let the cost function be the squared dissimilarity measure in Equation (2.13), so $J_{\boldsymbol{\pi}}(\boldsymbol{s}^{(1)}) = d_a(\boldsymbol{X},\boldsymbol{Y})^2 = g(\boldsymbol{s}^{(K)}) + \sum_{k=1}^{K-1} g(\boldsymbol{s}^{(k)})$. DTW is constrained by the system in Equation (2.14) and by the endpoint constraints $\boldsymbol{s}^{(1)} = (1,1)$ and $\boldsymbol{s}^{(K)} = (n,m)$. The cost function and the constraints constitute in the following dynamic program as defined in Definition 1.25:

$$\begin{aligned} \underset{\boldsymbol{\pi} \in \boldsymbol{\Pi}}{\text{minimize}} \quad & J_{\boldsymbol{\pi}}(\boldsymbol{s}^{(1)}) = g(\boldsymbol{s}^{(K)}) + \sum_{k=1}^{K-1} g(\boldsymbol{s}^{(k)}) \\ \text{subject to} \quad & \boldsymbol{s}^{(k+1)} = \boldsymbol{s}^{(k)} + \mu_k(\boldsymbol{s}^{(k)}), \quad k = 1,\ldots,K-1, \\ & \boldsymbol{s}^{(1)} = (1,1), \\ & \boldsymbol{s}^{(K)} = (n,m). \end{aligned} \tag{2.16}$$

Using the program in Equation (2.16), DTW is defined as

$$\text{DTW}(\boldsymbol{X},\boldsymbol{Y}) = \sqrt{J_{\pi^*}(1,1)}. \tag{2.17}$$

The value $J_{\pi^*}(1,1)$ is the optimal solution to a minimization problem in Equation (2.16).

The minimization problem in Equation (2.16) is equivalent to a shortest path problem

$$\begin{aligned} \underset{(\boldsymbol{s}^{(1)},\ldots,\boldsymbol{s}^{(K)})}{\text{minimize}} \quad & J_{\pi}(\boldsymbol{s}^{(1)}) = a_{\boldsymbol{s}^{(0)} \to \boldsymbol{s}^{(1)}} + \sum_{k=1}^{K-1} a_{\boldsymbol{s}^{(k)} \to \boldsymbol{s}^{(k+1)}} \\ \text{subject to} \quad & \boldsymbol{s}^{(0)} = (0,0), \\ & \boldsymbol{s}^{(K)} = (n,m), \end{aligned} \tag{2.18}$$

with $\boldsymbol{s}^{(k)} = (i,j)$, $i = 0,1,\ldots,n$, $j = 0,1,\ldots,m$. The transition costs

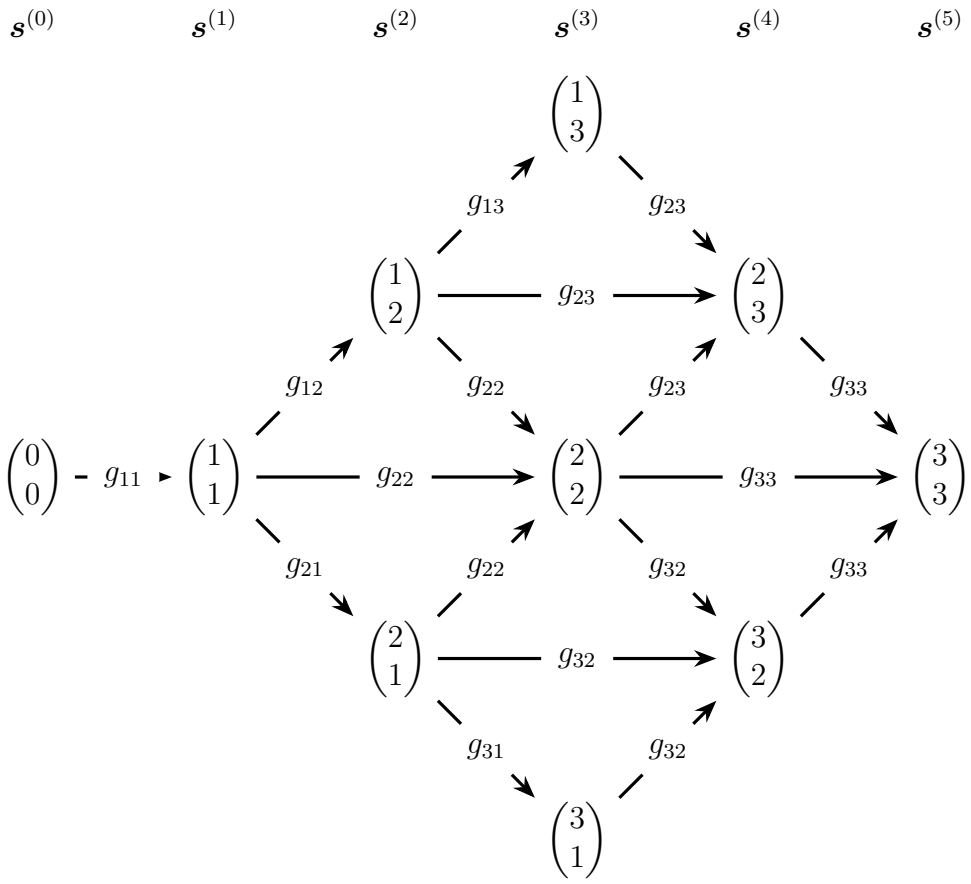

Figure 4: DTW algorithm as a shortest path problem. The shorter notation g_{ij} means in this figure the cost functional $g_{ij} := g(i,j)$. The states with legal transitions form a weighted directed acyclic graph.

$a_{\boldsymbol{s}^{(k)} \to \boldsymbol{s}^{(k+1)}}$ in Equation (1.48) are defined as

$$a_{(i,j)\to(i',j')} = \begin{cases} 0 & \text{, if } (i',j') = (i,j) \\ g(i',j') & \text{, if } (i',j') \in \{(i,j+1),(i+1,j+1),(i+1,j)\} \\ & \text{ and } i' = 1,\ldots,n \text{ and } j' = 1,\ldots,m \\ \infty & \text{, otherwise} \end{cases} \tag{2.19}$$

for all $k = 0, 1, \ldots, K-1$. The state $\boldsymbol{s}^{(0)} = (0,0)$ is a fictitious state, whose only purpose is to provide the initial cost $g(1,1)$ to the cost function $J(\boldsymbol{s}^{(K)})$ in Equation (2.18). The shortest path problem in Equation (2.18) is illustrated in Figure 4 as a weighted directed acyclic graph.

The length of the planning horizon can be fixed to K such that $K =$

$n+m-1$. That is the length of the longest admissible path that does not contain self-transitions. Such a policy exists because of the monotonicity constraint. The monotonicity constraint requires both indices i and j to be increasing and either index i or index j to be strictly increasing at any point. Clearly, the longest admissible path does not contain diagonal transitions $(i,j) \to (i+1, j+1)$ where not forced. This follows from the fact that the diagonal transition $(i,j) \to (i+1, j+1)$ may be replaced with two successive transitions $(i,j) \to (i+1, j) \to (i+1, j+1)$. One example of the longest admissible path is the path

$$\begin{aligned}\pi = \Big(&a_{(0,0)\to(1,1)}, a_{(1,1)\to(2,2)}, \\ &\ldots, a_{(1,m-1)\to(1,m)}, a_{(1,m)\to(2,m)}, \ldots, a_{(n-1,m)\to(n,m)}\Big).\end{aligned} \tag{2.20}$$

The path in Equation (2.20) has the length $K = n+m-1$. Adding anything except self-transitions in the longest admissible path would render the path inadmissible.

Let us solve the shortest path problem in Equation (2.18) using the forward dynamic programming algorithm in Algorithm 2. The recursive solution is then

$$\begin{cases} \widetilde{J}_K(\boldsymbol{s}^{(1)}) & = a_{(0,0)\to\boldsymbol{s}^{(1)}} \\ \widetilde{J}_k(\boldsymbol{s}^{(K-k+1)}) & = \min_{\boldsymbol{s}^{(K-k)}}[a_{\boldsymbol{s}^{(K-k)}\to\boldsymbol{s}^{(K-k+1)}} + \widetilde{J}_{k+1}(\boldsymbol{s}^{(K-k)})] \end{cases} \tag{2.21}$$

with an optimal solution $J_{\pi^*}(1,1) = \widetilde{J}_0(n,m)$.

Let us define the set of possible states as

$$S := \{1, \ldots, n\} \times \{1, \ldots, m\}. \tag{2.22}$$

Now each state $\boldsymbol{s}^{(k)} \in S$ for all $k = 1, \ldots, K$. The only state outside the set of the possible states is the fictitious initial state $\boldsymbol{s}^{(0)}$. Let us fix the current state $\boldsymbol{s}^{(K-k+1)} = (i,j)$. If the current is $(i,j) = (1,1)$, then the value in Step 3 in Algorithm 2 is $\widetilde{J}_k(1,1) = \min_{\boldsymbol{s}^{(K-k)}}[a_{\boldsymbol{s}^{(K-k)}\to(1,1)} + \widetilde{J}_{k+1}(\boldsymbol{s}^{(K-k)})]$. Now, the only transition with finite transition cost is the self-transition $(1,1) \to (1,1)$, so $\widetilde{J}_k(1,1) = \min\{0 + \widetilde{J}_{k+1}(1,1)\} = \widetilde{J}_{k+1}(1,1)$ for all $k = 0, 1, \ldots, K-1$. This follows from the definition of the transition cost in Equation (2.19): the state $(1,1)$ can only be reached from the states $(0,0)$ and $(1,1)$, while $\boldsymbol{s}^{(K-k)} \neq (0,0)$ for all $k = 0, 1, \ldots, K-1$. This means that, by recursion,

$$\widetilde{J}_k(1,1) = \widetilde{J}_K(1,1) = g(1,1) \tag{2.23}$$

for all $k = 0, 1, \ldots, K$.

I will now prove a lemma that lets us implicitly calculate the transition costs simplifying the notation.

Lemma 2.7. *Let $\widetilde{J}_k(i,j)$ be a sub-result as stated in the solution in Equation (2.21). Then $\widetilde{J}_k(i,j) = g(i,j) + \min \widetilde{J}_{k+1}(\{(i,j-1),(i-1,j-1),(i-1,j)\} \cap S)$ for any $(i,j) \in S \setminus \{(1,1)\}$.*

Proof. Let us first simplify Step 3 in Algorithm 2 for the shortest path problem in Equation (2.18). Let us fix the current state $\boldsymbol{s}^{(K-k+1)} = (i,j)$, and let $(i,j) \in S \setminus \{(1,1)\}$. Now, by Equation (2.19), the transition cost is finite only if the previous state $\boldsymbol{s}^{(K-k)}$ is in the set

$$\boldsymbol{s}^{(K-k)} \in \{(i,j),(i,j-1),(i-1,j-1),(i-1,j)\} \cap S. \tag{2.24}$$

In this case, since the self-transition has the cost 0, the value in Step 3 in Algorithm 2 is

$$\begin{aligned} \widetilde{J}_k(i,j) &= \min_{\boldsymbol{s}^{(K-k)} \in S} [a_{\boldsymbol{s}^{(K-k)} \to (i,j)} + \widetilde{J}_{k+1}(\boldsymbol{s}^{(K-k)})] \\ &= \min \Big[\{0 + \widetilde{J}_{k+1}(i,j)\} \\ &\quad \cup g(i,j) + \widetilde{J}_{k+1}(\{(i,j-1),(i-1,j-1),(i-1,j)\} \cap S) \Big] \end{aligned} \tag{2.25}$$

for all $k = 0, 1, \ldots, K-1$ by Equation (2.19).

Let us take a closer look at the notation used in Equation (2.25). With the sum of a term and a set I mean the image of a subset such as

$$\begin{aligned} &g(i,j) + \widetilde{J}_{k+1}(\{(i,j-1),(i-1,j-1),(i-1,j)\} \cap S) \\ &:= \big\{ g(i,j) + \widetilde{J}_{k+1}(i',j') \mid (i',j') \in \{(i,j-1),(i-1,j-1),(i-1,j)\} \cap S \big\}. \end{aligned} \tag{2.26}$$

The union in Equation (2.25) means a union of the set $\{\widetilde{J}_{k+1}(i,j)\}$ of single element and the image in Equation (2.26). Note that the set $\{(i,j-1),(i-1,j-1),(i-1,j)\} \cap S$ is non-empty when $(i,j) \in S \setminus \{(1,1)\}$.

The Equation (2.25) is already almost the same as the one in Lemma 2.7. The value $0 + \widetilde{J}_{k+1}(i,j)$ in Equation (2.25) is the cumulative cost for self-transition. This means that all that is left to prove is that, for any $(i,j) \in S \setminus \{(1,1)\}$, the self-transition is not the only optimal decision (if optimal at all). I will prove this using induction.

If $k = K-1$, then $\widetilde{J}_K(i,j) = \infty$ for all $(i,j) \in S \setminus \{(1,1)\}$. This can be seen from Equations (2.19) and (2.21): the only legal transition from the state $(0,0)$ is the transition $(0,0) \to (1,1)$. Now, by Equation (2.25), equation $\widetilde{J}_{K-1}(i,j) = g(i,j) + \min \widetilde{J}_K(\{(i,j-1),(i-1,j-1),(i-1,j)\} \cap S)$ holds since $\widetilde{J}_K(i,j) = \infty$. This means that the proposition in Lemma (2.7) holds for $k = K-1$.

Let us now assume that the proposition

$$\widetilde{J}_k(i,j) = g(i,j) + \min \widetilde{J}_{k+1}(\{(i,j-1),(i-1,j-1),(i-1,j)\} \cap S) \quad (2.27)$$

holds for some index k and for all $(i,j) \in S \setminus \{(1,1)\}$. I will show that then it holds for the index $k-1$, too. By Equation (2.25), the index $k-1$ satisfies equation

$$\begin{aligned} \widetilde{J}_{k-1}(i,j) = \min \Big[& \{\widetilde{J}_k(i,j)\} \\ & \cup\, g(i,j) + \widetilde{J}_k(\{(i,j-1),(i-1,j-1),(i-1,j)\} \cap S) \Big]. \end{aligned} \quad (2.28)$$

The induction assumption in Equation (2.27) can be used to the value $\widetilde{J}_k(i,j)$. Then Equation (2.28) can be written as

$$\begin{aligned} \widetilde{J}_{k-1}(i,j) = \min \Big[& \\ & \{g(i,j) + \min \widetilde{J}_{k+1}(\{(i,j-1),(i-1,j-1),(i-1,j)\} \cap S)\} \\ & \cup\, g(i,j) + \widetilde{J}_k(\{(i,j-1),(i-1,j-1),(i-1,j)\} \cap S) \Big] \\ = g(i,j) + \min \Big[& \widetilde{J}_{k+1}(\{(i,j-1),(i-1,j-1),(i-1,j)\} \cap S) \\ & \cup\, \widetilde{J}_k(\{(i,j-1),(i-1,j-1),(i-1,j)\} \cap S) \Big]. \end{aligned} \quad (2.29)$$

In the second equality in Equation (2.29), an observation regarding the real numbers is used such as $\min\{a, \min\{b,c\}\} = \min\{a,b,c\}$ where $a,b,c \in \mathbb{R} \cup \{\infty\}$.

It is easy to see that $\widetilde{J}_{k+1}(i',j') \geq \widetilde{J}_k(i',j')$ when $(i',j') \in S$. This follows from the fact that $\widetilde{J}_k(i',j') = \min\{\widetilde{J}_{k+1}(i',j')\} \cup g(i',j') + \widetilde{J}_{k+1}(\{(i',j'-1),(i'-1,j'-1),(i'-1,j')\} \cap S)$ when $(i',j') \in S \setminus \{(1,1)\}$, and $\widetilde{J}_k(1,1) = \widetilde{J}_{k+1}(1,1)$ when $(i',j') = (1,1)$. In other words, the value $\widetilde{J}_k(i',j')$ is the minimum of a set that includes the value $\widetilde{J}_{k+1}(i',j')$. Thus, the induction assumption in Equation (2.27) holds for the index $k-1$, and, by induction,

$$\widetilde{J}_k(i,j) = g(i,j) + \min \widetilde{J}_{k+1}(\{(i,j-1),(i-1,j-1),(i-1,j)\} \cap S) \quad (2.30)$$

for all $k = 0,1,\ldots,K-1$ and $(i,j) \in S \setminus \{(1,1)\}$. □

By the result in Lemma 2.7 the recursive solution in Equation (2.21) can

be rewritten as

$$\begin{cases} \widetilde{J}_K(1,1) & = \widetilde{J}_k(1,1) = g(1,1) \\ \widetilde{J}_K(i,j) & = \infty \\ \widetilde{J}_k(i,j) & = g(i,j) + \min \widetilde{J}_{k+1}(\{(i,j-1),(i-1,j-1),(i-1,j)\} \cap S) \end{cases} \tag{2.31}$$

for all $(i,j) \in S \setminus \{(1,1)\}$. The optimal solution is $J_{\pi^*}(1,1) = \widetilde{J}_0(n,m)$. The following lemma and its proof will further simplify the solution in Equation (2.31). In addition, the following lemma will give us the algorithm that is identical to the DTW algorithm presented in Equation (2.12).

Lemma 2.8. *The index k in the solution in Equation (2.31) is not necessary.*

Proof. I will prove this lemma with the definition of sub-problems in Definition 1.26. Let us consider the following sub-problem to the shortest path problem in Equation (2.18):

$$\begin{aligned} \underset{(\boldsymbol{s}^{(1)},\ldots,\boldsymbol{s}^{(L)})}{\text{minimize}} \quad & J_{\pi_L}(\boldsymbol{s}^{(1)}) = a'_{\boldsymbol{s}^{(0)} \to \boldsymbol{s}^{(1)}} + \sum_{l=1}^{L-1} a'_{\boldsymbol{s}^{(l)} \to \boldsymbol{s}^{(l+1)}} \\ \text{subject to} \quad & \boldsymbol{s}^{(0)} = (0,0), \\ & \boldsymbol{s}^{(L)} = (n',m'), \end{aligned} \tag{2.32}$$

where $L = m' + n' - 1 < K$ and $n' \leq n$, $m' \leq m$. The transition costs in Equation (2.32) are defined as

$$a'_{(i,j)\to(i',j')} = \begin{cases} 0 & \text{, if } (i',j') = (i,j) \\ g(i',j') & \text{, if } (i',j') \in \{(i,j+1),(i+1,j+1),(i+1,j)\} \\ & \text{ and } i' = 1,\ldots,n' \text{ and } j' = 1,\ldots,m' \\ \infty & \text{, otherwise.} \end{cases} \tag{2.33}$$

Let $S' = \{1,\ldots,n'\} \times \{1,\ldots,m'\}$. Now the minimization problem in Equation (2.32) is a shortest path problem, and it is similar to the one in Equation (2.18). As a shortest path problem, it has a solution similar to the one in Equation (2.31):

$$\begin{cases} \widetilde{J}_L(1,1) & = \widetilde{J}_l(1,1) = g(1,1) \\ \widetilde{J}_L(i,j) & = \infty \\ \widetilde{J}_l(i,j) & = g(i,j) + \min \widetilde{J}_{l+1}(\{(i,j-1),(i-1,j-1),(i-1,j)\} \cap S'). \end{cases} \tag{2.34}$$

The optimal solution is $J_{\pi_L^*}(1,1) = \widetilde{J}_0(n',m')$. Note, however, that this sub-problem is useful according to Bellman's principle of optimality in Theorem 1.27 only if the state (n',m') is present in the optimal path π^* for the original problem in Equation 2.18. For the purpose of this proof, this can be assume without the loss of generality.

Let us synchronize the indices with the original problem in Equation (2.18) by adding a fixed integer $K-L$ to the index. This does not change the solution in Equation (2.34) in any way. Then the solution, whose indices are synchronized with the problem in Equation (2.18), is

$$\begin{cases} \widetilde{J}_K(1,1) & = \widetilde{J}_{K-L+l}(1,1) = g(1,1) \\ \widetilde{J}_K(i,j) & = \infty \\ \widetilde{J}_{K-L+l}(i,j) & = g(i,j) \\ & \quad + \min \widetilde{J}_{K-L+l+1}(\{(i,j-1),(i-1,j-1),(i-1,j)\} \cap S') \end{cases} \tag{2.35}$$

Now, the optimal solution is $J_{\pi_L^*}(1,1) = \widetilde{J}_{K-L}(n',m')$, and it is equivalent to the solution in Equation (2.34). Every admissible path for the problem in Equation (2.32) has length L including the optimal solution. To keep the admissible paths admissible, only self-transitions may be added. Since the self-transitions have no effect on the cost function value, the equations $\widetilde{J}_{K-L}(n',m') = \widetilde{J}_{K-L-1}(n',m') = \cdots = \widetilde{J}_0(n',m')$ hold. Note that $L = n' + m' - 1$ by definition in Equation (2.32). In addition, the state $(n',m') \in S' \subset S$, so it is a possible state in the shortest path problem in Equation (2.18). This means that, by using the sub-problems in Equation (2.32) and Bellman's principle of optimality in Theorem 1.27, the following rule has been derived:

$$\text{if } k \leq K - i - j + 1, \text{ then } \widetilde{J}_k(i,j) = \widetilde{J}_0(i,j). \tag{2.36}$$

The rule in Equation 2.36 applies for all $k = 0, 1, \ldots, K$ and $(i,j) \in S$. Note that it is already shown that $\widetilde{J}_k(1,1) = \widetilde{J}_0(1,1)$ in Equation (2.23).

DTW has been defined as $\mathrm{DTW}(\boldsymbol{X},\boldsymbol{Y}) = \sqrt{J_{\pi^*}(1,1)}$ in Equation (2.17). Due to the endpoint constraints, the only interesting cost function value is the value at the state (n,m) with the index $k = 0$. It has an optimal value $\widetilde{J}_0(n,m)$ by Equation (2.31). For $\widetilde{J}_0(n,m)$ the rule applies because $K-m-n+1 = K-K \geq 0$. This implies the trivial result $\widetilde{J}_0(n,m) = \widetilde{J}_0(n,m)$ by Equation (2.36).

Let the rule in Equation (2.36) now apply for some index k with some index pair $(i,j) \in S$. This means that $k \leq K - i - j + 1$, implying that $\widetilde{J}_k(i,j) = \widetilde{J}_0(i,j)$. By adding 1 to the inequality, the following inequalities

hold:

$$\begin{aligned} k+1 &\leq K-i-j+2 \\ &= K-i-(j-1)+1 = K-(i-1)-j+1 \\ &\leq K-(i-1)-(j-1)+1. \end{aligned} \tag{2.37}$$

The inequality in Equation (2.37) implies that

$$\begin{aligned} &\widetilde{J}_{k+1}(\{(i,j-1),(i-1,j-1),(i-1,j)\} \cap S) \\ &\quad = \widetilde{J}_0(\{(i,j-1),(i-1,j-1),(i-1,j)\} \cap S). \end{aligned} \tag{2.38}$$

To summarize, the rule in Equation (2.36) applies to the index $k = 0$ with the state (n, m). By the inequality in Equation (2.37), the rule then applies to the index $k = 1$ with the previous states $(n, m-1)$, $(n-1, m-1)$ and $(n-1, m)$. By recursive use of the inequality in Equation (2.37), the value $J_{\pi^*}(1,1) = \widetilde{J}_0(n,m)$ can be solved using the following algorithm:

$$\begin{cases} \widetilde{J}_0(1,1) & = g(1,1) \\ \widetilde{J}_0(i,j) & = g(i,j) + \min \widetilde{J}_0(\{(i,j-1),(i-1,j-1),(i-1,j)\} \cap S). \end{cases} \tag{2.39}$$

By definition in Equation (2.17), DTW distance between the time series $\boldsymbol{X}$ and $\boldsymbol{Y}$ is $\mathrm{DTW}(\boldsymbol{X}, \boldsymbol{Y}) = \sqrt{J_{\pi^*}(1,1)}$. This value is equal to the value in Equation (2.12). □

By Lemma 2.8, the definition of the cost function is not necessary for solving the DTW distance. To highlight this, the cost function notations $\widetilde{J}_0$ in Equation (2.39) can be replaced with some arbitrary function notation c^2 for equivalent results. Of course, then the function g in Equation (2.39) must be defined such that

$$g(i,j) = \begin{cases} d(\boldsymbol{X}_i, \boldsymbol{Y}_j)^2 & \text{, if } (i,j) \in S \\ \infty & \text{, otherwise.} \end{cases} \tag{2.40}$$

This result is summarized in the following theorem.

Theorem 2.9. *The DTW measure between time series $\boldsymbol{X}$ and $\boldsymbol{Y}$ defined in Equations (2.16) and (2.17) may be solved recursively as*

$$\begin{cases} c^2(1,1) = d(\boldsymbol{X}_1, \boldsymbol{Y}_1)^2 \\ c^2(i,j) = d(\boldsymbol{X}_i, \boldsymbol{Y}_j)^2 + \min\{c^2(i,j-1), c^2(i-1,j-1), c^2(i-1,j)\} \\ c^2(i,0) = c^2(0,j) = \infty \end{cases} \tag{2.41}$$

with

$$DTW(\boldsymbol{X}, \boldsymbol{Y}) = \sqrt{c^2(n, m)}, \tag{2.42}$$

where $i = 1, \ldots, n$ *and* $j = 1, \ldots, m$.

Proof. Solving the dynamic program in Equation (2.16) is equivalent to solving the shortest path problem in Equation (2.18). By Lemmas 2.7 and 2.8, the shortest path problem in Equation (2.18) has a recursive solution presented in Equation (2.39). By defining $c^2(i, 0) = c^2(0, j) = \infty$, the minimization in Equation (2.39) is equivalent to

$$\begin{aligned} &\min c^2(\{(i, j-1), (i-1, j-1), (i-1, j)\} \cap S) \\ &\quad = \min\{c^2(i, j-1), c^2(i-1, j-1), c^2(i-1, j)\}. \end{aligned} \tag{2.43}$$

Since the function g is defined as $g(i, j) = d(\boldsymbol{X}_i, \boldsymbol{Y}_j)^2$, the optimal cost function value to the dynamic program in Equation (2.16) is $c^2(n, m)$ as obtained from Equation (2.41). By the definition of DTW in Equation (2.17), DTW measure can be calculated using Equation (2.42). □

Here I have derived the recursive algorithm for DTW. I omitted the Sakoe-Chiba window from the proofs for simplicity. In practice, it is easy to add the Sakoe-Chiba window constraint to the solution. Let the window length be $r \geq 1$. The window constraint is added to the transition cost in Equation (2.19) such that

$$a_{(i,j)\to(i',j')} = \begin{cases} 0 & \text{, if } (i', j') = (i, j) \\ g(i', j') & \text{, if } (i', j') \in \{(i, j+1), (i+1, j+1), (i+1, j)\} \\ & \quad \text{and } i' = 1, \ldots, n \text{ and } j' = 1, \ldots, m \\ & \quad \text{and } |i' - j'| \leq r \\ \infty & \text{, otherwise.} \end{cases} \tag{2.44}$$

The set of possible states inside the Sakoe-Chiba window is the set $S_{S-C} = \{(i, j) \in S \mid |i - j| \leq r\}$. This set is in line with the transition costs in Equation (2.44).

2.3.4 Principal Component Analysis Similarity Factor

Principal Component Analysis Similarity Factor (PCA-SF) differs from Euclidean distance and DTW since it is a similarity measure. In addition, PCA-SF was not originally proposed to measure the similarity of MTSs but the similarity of datasets [33]. It was later proposed for MTSs by [56] since MTSs are a type of dataset.

Krzanowski [33] proposed principal component analysis similarity factor as a measure of similarity between datasets. Similarity is calculated from the angles between the principal components. The number of principal components is set so that they explain at least 95% of variance in each data set. The modified version of PCA-SF takes into account the amount of variance explained by the principal components. This modified PCA-SF, which will simply be called PCA-SF, is defined as [26, 55]

$$S_{PCA}^{\lambda} := \frac{\sum_{i=1}^{k} \sum_{j=1}^{k} (\lambda_i^{(1)} \lambda_j^{(2)}) \cos^2 \theta_{ij}}{\sum_{i=1}^{k} \lambda_i^{(1)} \lambda_i^{(2)}}. \tag{2.45}$$

In Equation (2.45), the parameter k is the number of principal components, and parameters $\lambda_i^{(1)}$ and $\lambda_i^{(2)}$ are the ith eigenvalues of the datasets one and two, respectively. In addition, in Equation (2.45), the parameter θ_{ij} is the angle between ith principal component of the dataset one and jth principal component of the dataset two, respectively. This similarity factor is suitable for measuring the similarity between multivariate time series of equal or unequal length [56].

3 Semi-Supervised Learning

Learning paradigms are usually classified into supervised, semi-supervised and unsupervised learning based on the availability of feedback or labels. In supervised learning, each observation has a label. In unsupervised learning, there are no labels at all. In semi-supervised learning (SSL) there are some observations with labels and a lot of data without labels.

This section reviews some selected one-class classification (OCC) and positive-unlabelled (PU) learning methods. OCC is learning from positive data only with no known negative examples, while PU is learning from unlabelled data from both classes with only a few known positive examples. Technically, OCC methods are not semi-supervised learning. OCC methods are so unique that they are a category on their own. They are, however discussed in this section since they are not supervised or unsupervised methods either. In this section, both OCC and PU methods will be applied to novelty detection. The novelty detection is defined in the following section.

3.1 Novelty Detection and One-Class Classification

Novelty detection is a branch of pattern recognition. It refers to finding novel patterns from the data. Ntalampiras, Potamitis and Fakotakis [42] define a

novelty as data that differ from the training examples significantly. Novelty detection is closely related to anomaly detection, which aims at finding unexpected or anomalous behavior of patterns in data [9]. In some cases, anomalies are thought to be synonymous to novelties [45]. In this thesis, an anomaly is an abnormal event in the training data, such as an artefact or a malfunction of a recording device, whereas novelty is an event that is not present in the training data, such as a forged signature.

One-class classification is a binary classification task in which the positive class is well represented in training data, but the negative class is sampled poorly or not at all. Khan and Madden [31] list three settings in OCC according the availability of training data:

- Data from the positive class only
- Data from the positive class and few examples from the negative class
- Data from the positive class and data without class labels.

This thesis assumes the third data availability assumption. The goal is to find all the positive time series based on just one annotated example while rejecting negative time series.

Before defining OCC, let us define a binary classifier. A binary classifier is a function $g : \mathbb{R}^m \to C$, where $C = \{-1, +1\}$. Score function $f : \mathbb{R}^m \to \mathbb{R}$ assigns a classifier score to a data point. Decision rule is a function $b : \mathbb{R} \to \{-1, +1\}$ that links the classifier score to the classifier such that $b \circ f = g$. For example, for a Bayesian classifier the score function could be $f(\boldsymbol{x}) = p(C = +1|\boldsymbol{x})$, and the decision rule could be

$$b(z) = \begin{cases} +1 & \text{, if } z \geq \frac{1}{2} \\ -1 & \text{, otherwise.} \end{cases} \tag{3.1}$$

Tax [59] defines a one-class classifier in a similar way. A score function is either a distance $d(\boldsymbol{x})$ or a resemblance $p(\boldsymbol{x})$ of the query point $\boldsymbol{x}$ to the positive class +1. The one-class classifier is a function $g(\boldsymbol{x}) = I(d(\boldsymbol{x}) < \theta_d)$ or $g(\boldsymbol{x}) = I(p(\boldsymbol{x}) > \theta_p)$, where $I(\cdot)$ is an indicator function. In this definition, the decision rule is $b(z) = I(f(\boldsymbol{x}) > \theta)$ with a threshold θ. Usually the threshold θ is optimized after the optimization of the score function to get the best trade-off between the two types of errors. [59] The two type of errors are false positives and false negatives. The concept of the two type of errors is defined more precisely in Section 4.

OCC falls between supervised and unsupervised learning tasks [64]. It is not semi-supervised learning in the sense that it does not make use of the

unlabelled data. It is not supervised learning either since there is no data from the negative class. OCC is close to unsupervised learning, and many assumptions and methods of unsupervised learning apply to OCC as such.

3.1.1 Support Vector Data Description

Tax and Duin [60] proposed a support vector data description (SVDD) to solve the task of one-class classification. The method is inspired by the success of the support vector machine. This method finds the minimum volume sphere that encloses most of the data points of the positive class. Support vector data description does not make assumptions of the underlying distribution of the positive class This is helpful since it is difficult or even impossible to sample the whole domain of the positive class [13]. The following mathematical formulation of support vector data description is based on [60].

Let $\boldsymbol{a} \in \mathbb{R}^m$ be the center of a hypersphere and R be its radius. The volume of a hypersphere in an m-dimensional space depends only on the radius of the hypersphere. Given dataset $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)^T \subset \mathbb{R}^m$, it is possible to find the minimum volume hypersphere that bounds the dataset by solving the following minimization problem:

$$\begin{aligned} \underset{R \in \mathbb{R}, \boldsymbol{a} \in \mathbb{R}^m}{\text{minimize}} \quad & R^2 \\ \text{subject to} \quad & \|\boldsymbol{x}_i - \boldsymbol{a}\|^2 \leq R^2, \quad i = 1, \ldots, n. \end{aligned} \tag{3.2}$$

The goal of SVDD is to describe the data. If the dataset $\boldsymbol{X}$ includes outliers, the description found by using Equation (3.2) may not be good. The outliers "stretch" the hypersphere to include the low density region between the normal data and outliers. To take the outliers in the training data into account, Tax and Duin [60] propose to relax the constraints in Equation (3.2). This can be done by introducing non-negative variables $\boldsymbol{\xi}$ similarly to soft margin support vector machine in [15].

Let $\xi_i \geq 0$ for all $i = 1, \ldots, n$. The *soft constraints* for the program in Equation (3.2) may be written as

$$\|\boldsymbol{x}_i - \boldsymbol{a}\|^2 \leq R^2 + \xi_i \tag{3.3}$$

for all $i = 1, \ldots, n$. The soft constraints mean here that the solutions outside of the feasible region of the program in Equation (3.2) are not rejected but penalized. The penalty is proportional to the error made by using the soft constraints. This error is measured by a sum $\sum_{i=1}^{n} \xi_i$. The penalized objective function is then

$$R^2 + C \sum_{i=1}^{n} \xi_i. \tag{3.4}$$

The parameter $C > 0$ in Equation (3.5) is a control parameter that is used to control the balance between the volume of the hypersphere and the classification error.

Note that the program in Equation (3.2) and its soft margin counterpart in Equations (3.3) and (3.4) are non-convex. The inequality constraint function $g_i(R, \boldsymbol{a}, \boldsymbol{\xi}) = \boldsymbol{x}_i^T \boldsymbol{x}_i + \boldsymbol{a}^T \boldsymbol{a} - 2\boldsymbol{x}_i^T \boldsymbol{a} - R^2 - \xi_i$ in Equation (3.3) is convex with respect to $\boldsymbol{a}$ and linear with respect to ξ_i but concave with respect to R [11]. Chang, Lee and Lin [11] proposed to replace the squared radius R^2 in Equations (3.6) and (3.5) with $\bar{R} = R^2$. With this replacement, the objective function is

$$f(\bar{R}, \boldsymbol{a}, \boldsymbol{\xi}) = \bar{R} + C \sum_{i=1}^{n} \xi_i \tag{3.5}$$

with constraints

$$\|\boldsymbol{x}_i - \boldsymbol{a}\|^2 \leq \bar{R} + \xi_i. \tag{3.6}$$

Of course, then a condition $\bar{R} \geq 0$ is required since the radius of a hypersphere must be non-negative.

Chang et al. [11] further argued that the dual problem would be infeasible with selection $C < 1/n$. In addition, a selection of $C > 1$ will always lead hard constraint program in Equation (3.2) [11]. For this reason, the penalty parameter is required to satisfy the limits $1/n < C \leq 1$ through out this thesis. This requirement for the parameter C has another pleasant consequence: Chang et al. [11] proved that the constraint $\bar{R} \geq 0$ is not necessary when $1/n < C \leq 1$. The following remark will discuss the case where $\bar{R} < 0$.

Remark 3.1. Let $\bar{R} = 0$. Equation (3.6) implies that then $\xi_i \geq \|\boldsymbol{x}_i - \boldsymbol{a}\|^2$ for all $i = 1, \ldots, n$. Since Euclidean norm is always non-negative, the variables can be selected such as $\xi_i^* = \|\boldsymbol{x}_i - \boldsymbol{a}\|^2 \geq 0$. The selection $\bar{R} = 0$ and $\boldsymbol{\xi} = \boldsymbol{\xi}^* := (\xi_1^*, \ldots, \xi_n^*)$ satisfies the soft constraints in Equation (3.6). With this selection, the cost function in Equation (3.5) has a value

$$f(0, \boldsymbol{a}, \boldsymbol{\xi}^*) = C \sum_{i=1}^{n} \|\boldsymbol{x}_i - \boldsymbol{a}\|^2 \tag{3.7}$$

for any $\boldsymbol{a} \in \mathbb{R}^m$.

Let $C > 1/n$. This inequality is equivalent to the selection $C = 1/n + \epsilon/n$ with any $\epsilon > 0$. Now, because of the soft constraints in Equation (3.6), each variable ξ_i has a lower bound $\xi_i \geq -\bar{R} + \|\boldsymbol{x}_i - \boldsymbol{a}\|^2$ for all $i = 1, \ldots, n$ and any $\bar{R} \in \mathbb{R}$. From this, a lower bound to the weighted error term in Equation (3.5) can be derived such that:

$$C \sum_{i=1}^{n} \xi_i \geq \frac{1+\epsilon}{n} \sum_{i=1}^{n} -\bar{R} + \|\boldsymbol{x}_i - \boldsymbol{a}\|^2 = -(1+\epsilon)\bar{R} + \frac{1+\epsilon}{n} \sum_{i=1}^{n} \|\boldsymbol{x}_i - \boldsymbol{a}\|^2. \tag{3.8}$$

The lower bound in Equation (3.8) can be used to derive a lower bound to the cost function in Equation (3.5):

$$f(\bar{R}, \boldsymbol{a}, \boldsymbol{\xi}) = \bar{R} + C \sum_{i=1}^{n} \xi_i \geq -\epsilon \bar{R} + C \sum_{i=1}^{n} \|\boldsymbol{x}_i - \boldsymbol{a}\|^2. \tag{3.9}$$

The lower bound in Equation (3.9) holds for any $(\bar{R}, \boldsymbol{a}, \boldsymbol{\xi}) \in \mathbb{R}^{1+m+n}$ that satisfies the inequalities in Equation (3.6). If $\bar{R} < 0$ then Equation (3.7) and the lower bound in Equation (3.9) imply that $f(\bar{R}, \boldsymbol{a}, \boldsymbol{\xi}) > f(0, \boldsymbol{a}, \boldsymbol{\xi}^*)$. This is true for any $(\bar{R}, \boldsymbol{a}, \boldsymbol{\xi}) \in \mathbb{R}^{1+m+n}$ that satisfies the inequalities in Equation (3.6). This means that the constraint $\bar{R} \geq 0$ is not necessary when $C > 1/n$.

Using this modification and the error variable $\boldsymbol{\xi}$, the minimum volume problem is equivalent to the following minimization problem:

$$\begin{array}{cc} \underset{\bar{R} \in \mathbb{R}, \boldsymbol{a} \in \mathbb{R}^m, \boldsymbol{\xi} \succeq 0}{\text{minimize}} & \bar{R} + C \sum_{i=1}^{n} \xi_i \\ \text{subject to} & \|\boldsymbol{x}_i - \boldsymbol{a}\|^2 \leq \bar{R} + \xi_i, \quad i = 1, \ldots, n \end{array} \tag{3.10}$$

with $1/n < C \leq 1$.

Remark 3.2. In the program in Equation (3.10), there is an abstract constraint $(\bar{R}, \boldsymbol{a}, \boldsymbol{\xi}) \in \mathbb{R} \times \mathbb{R}^m \times \mathbb{R}^n_+ = X$. The notation $\mathbb{R}_+$ means non-negative real numbers. This abstract constraint must be taken into account throughout the analysis of the program in Equation (3.10).

Remark 3.3. With the abstract constraint $(\bar{R}, \boldsymbol{a}, \boldsymbol{\xi}) \in X$, the domain D of the program in Equation (3.10) is now $D = X$. However, the affine hull of the domain is $\text{aff}(D) = \mathbb{R}^{1+m+n}$. Then, the set in Equation (1.22) is $B((\bar{R}, \boldsymbol{a}, \boldsymbol{\xi}), \epsilon) \cap \mathbb{R}^{1+m+n} = B((\bar{R}, \boldsymbol{a}, \boldsymbol{\xi}), \epsilon)$. This means that the relative interior in Definition 1.15 is $\text{relint}(D) = \text{int}(D)$. Now Theorem 1.21 can be used to analyse the program in Equation (3.10).

Let us consider the program in Equation (3.10) with $1/n < C \leq 1$. The objective function is linear since it is a sum of two linear functions $f_{\bar{R}}(\bar{R}) = \bar{R}$ and $f_{\boldsymbol{\xi}}(\boldsymbol{\xi}) = C \sum_{i=1}^{n} \xi_i$. The functions in the inequality constraints $g_i(\bar{R}, \boldsymbol{a}, \boldsymbol{\xi}) = \boldsymbol{x}_i^T \boldsymbol{x}_i + \boldsymbol{a}^T \boldsymbol{a} - 2\boldsymbol{x}_i^T \boldsymbol{a} - \bar{R} - \xi_i \leq 0$ are convex functions for all $i = 1, \ldots, n$. This is true since functions $g_{\bar{R}}(\bar{R}) = -\bar{R}$ and $g_{\boldsymbol{\xi},i}(\boldsymbol{\xi}) = -\xi_i$ are linear, functions $g_{\boldsymbol{a},i}(\boldsymbol{a}) = \boldsymbol{a}^T \boldsymbol{a} - 2\boldsymbol{x}_i^T \boldsymbol{a}$ are quadratic and terms $\boldsymbol{x}_i^T \boldsymbol{x}_i$ are constants for all $i = 1, \ldots, n$. In addition, the set X in the abstract constraint is convex. Thus, by Corollary 1.10, the program in Equation (3.10) is a convex program.

Clearly, the feasible region of the programming task in Equation (3.10) is non-empty. Since the radius $\bar{R}$ is not bounded from above, there is always a hypersphere that encloses the data with any center $\boldsymbol{a}$. For example, let $\bar{R} = \sum_{j=1}^{n} \|\boldsymbol{x}_j\|^2$, $\boldsymbol{a} = \mathbf{0} \in \mathbb{R}^m$ and $\boldsymbol{\xi} = \boldsymbol{\epsilon} = (\epsilon, \ldots, \epsilon)^T \in \mathbb{R}^n$ with $\epsilon > 0$. The point $(\sum_{j=1}^{n} \|\boldsymbol{x}_j\|^2, \mathbf{0}, \boldsymbol{\epsilon})$ is in the interior of the domain D with any dataset $\boldsymbol{X} = (\boldsymbol{x}_j)_{j=1}^{n}$. In addition, the inequalities $g_i(\bar{R}, \boldsymbol{a}, \boldsymbol{\xi}) = \|\boldsymbol{x}_i\|^2 - \sum_{j=1}^{n} \|\boldsymbol{x}_j\|^2 - \epsilon < 0$ hold for all $i = 1, \ldots, n$. Thus, the point $(\sum_{j=1}^{n} \|\boldsymbol{x}_j\|^2, \mathbf{0}, \boldsymbol{\epsilon}) \in D$ is an interior point to the feasible region, too. This means that the program in Equation (3.10) satisfies Slater's condition in Definition 1.18. Since the program in Equation (3.10) is also convex, the KKT conditions in Theorem 1.13 are necessary and sufficient for the global optimum.

Let us derive the Lagrangian for the program in Equation (3.10) with $1/n < C \leq 1$ as in Definition 1.11. Here, the abstract constraints $\boldsymbol{\xi} \succeq \mathbf{0}$ must be taken into account. The constraint $\boldsymbol{\xi} \succeq \mathbf{0}$ may be ensured in the Lagrangian with an additional term $\sum_{i=1}^{n} \gamma_i(-\xi_i)$, where $\boldsymbol{\gamma} \succeq \mathbf{0}$, as if it were any ordinary inequality constraint. Now, the Lagrangian of the program in Equation (3.10) is

$$\begin{aligned}\mathcal{L}(\bar{R}, \boldsymbol{a}, \boldsymbol{\xi}, \boldsymbol{\lambda}, \boldsymbol{\gamma}) = \bar{R} + C \sum_{i=1}^{n} \xi_i + \sum_{i=1}^{n} \lambda_i (\boldsymbol{x}_i^T \boldsymbol{x}_i + \boldsymbol{a}^T \boldsymbol{a} - 2\boldsymbol{x}_i^T \boldsymbol{a} - \bar{R} - \xi_i) \\ + \sum_{i=1}^{n} \gamma_i (-\xi_i).\end{aligned} \tag{3.11}$$

Let the point $(R^*, \boldsymbol{a}^*, \boldsymbol{\xi}^*)$ be the optimal solution for the program in Equation (3.10). Since the point $(R^*, \boldsymbol{a}^*, \boldsymbol{\xi}^*)$ is optimal, it satisfies KKT conditions in Theorem 1.13 together with some point $(\boldsymbol{\lambda}^*, \boldsymbol{\gamma}^*)$. The point $(\boldsymbol{\lambda}^*, \boldsymbol{\gamma}^*)$ is the optimal solution to the dual problem of the primal problem in Equation (3.10) where $1/n < C \leq 1$. The point $(\boldsymbol{\lambda}^*, \boldsymbol{\gamma}^*) \succeq \mathbf{0}$ exists because the program in Equation (3.10) is convex and it satisfies Slater's condition in Definition 1.18. The partial derivatives of the Lagrangian in Equation (3.11) with respect to $\bar{R}$, $\boldsymbol{a}$ and $\boldsymbol{\xi}$ are

$$\frac{\partial}{\partial \bar{R}} \mathcal{L}(\bar{R}, \boldsymbol{a}, \boldsymbol{\xi}, \boldsymbol{\lambda}, \boldsymbol{\gamma}) = 1 - \sum_{i=1}^{n} \lambda_i \tag{3.12}$$

$$\frac{\partial}{\partial \boldsymbol{a}} \mathcal{L}(\bar{R}, \boldsymbol{a}, \boldsymbol{\xi}, \boldsymbol{\lambda}, \boldsymbol{\gamma}) = 2 \sum_{i=1}^{n} \lambda_i \boldsymbol{a} - 2 \sum_{i=1}^{n} \lambda_i \boldsymbol{x}_i \tag{3.13}$$

$$\frac{\partial}{\partial \xi_i} \mathcal{L}(\bar{R}, \boldsymbol{a}, \boldsymbol{\xi}, \boldsymbol{\lambda}, \boldsymbol{\gamma}) = C - \lambda_i - \gamma_i \quad \text{for all } i = 1, \ldots, n. \tag{3.14}$$

By the stationary condition in Equation (1.16), all the partial derivatives in Equations (3.12), (3.13) and (3.14) equal to zero at the optimal point $(R^*, \boldsymbol{a}^*, \boldsymbol{\xi}^*, \boldsymbol{\lambda}^*, \boldsymbol{\gamma}^*)$. Equation (3.12) implies that $\sum_{i=1}^n \lambda_i^* = 1$. This is applied to the optimal center. Now, by Equation (3.13), the optimal center is the point $\boldsymbol{a}^* = \sum_{i=1}^n \lambda_i^* \boldsymbol{x}_i / \sum_{i=1}^n \lambda_i^* = \sum_{i=1}^n \lambda_i^* \boldsymbol{x}_i$. This means that the optimal center for the hypersphere is a convex combination of the data points. To summarize, the stationary condition in Equation (1.16) states for the Lagrangian in Equation (3.14) that

$$\boldsymbol{a}^* = \sum_{i=1}^n \lambda_i^* \boldsymbol{x}_i \quad \text{with } \sum_{i=1}^n \lambda_i^* = 1 \text{ and} \tag{3.15a}$$

$$C - \lambda_i^* - \gamma_i^* = 0 \quad \text{for all } i = 1, \ldots, n. \tag{3.15b}$$

In addition, the dual feasibility in Equation (1.18) ensures that $\lambda_i^*, \gamma_i^* \geq 0$ for all $i = 1, \ldots, n$. The stationary condition in Equation (3.15b) implies that $C - \lambda_i^* \geq 0$ since γ_i^* is non-negative. Finally, this means that the inequality $0 \leq \lambda_i^* \leq C$ holds. This inequality together with Equation (3.15a) will be the constraints for the dual problem as in Equation (1.9).

The value of the Lagrangian function in Equation (3.11) at the optimum is

$$\begin{aligned}
\mathcal{L}(R^*, \boldsymbol{a}^*, \boldsymbol{\xi}^*, \boldsymbol{\lambda}^*, \boldsymbol{\gamma}^*) &= R^* + C \sum_{i=1}^n \xi_i^* + \sum_{i=1}^n \lambda_i^* \boldsymbol{x}_i^T \boldsymbol{x}_i + \boldsymbol{a}^{*T} \boldsymbol{a}^* \sum_{i=1}^n \lambda_i^* \\
&\quad - 2 \sum_{i=1}^n \lambda_i^* \boldsymbol{x}_i^T \boldsymbol{a}^* - R^* \sum_{i=1}^n \lambda_i^* - \sum_{i=1}^n \lambda_i^* \xi_i^* - \sum_{i=1}^n \gamma_i^* \xi_i^* \\
&= R^* - R^* \sum_{i=1}^n \lambda_i^* + \sum_{i=1}^n (C - \lambda_i^* - \gamma_i^*) \xi_i^* \\
&\quad + \sum_{i=1}^n \lambda_i^* \boldsymbol{x}_i^T \boldsymbol{x}_i + \boldsymbol{a}^{*T} \boldsymbol{a}^* \sum_{i=1}^n \lambda_i^* - 2 \sum_{i=1}^n \lambda_i^* \boldsymbol{x}_i^T \boldsymbol{a}^*.
\end{aligned} \tag{3.16}$$

By Equation (3.15a), the equation $\sum_{i=1}^n \lambda_i^* = 1$ holds. Further, by Equation (3.15b), also the equations $C - \lambda_i^* - \gamma_i^* = 0$ hold for all $i = 1, \ldots, n$. This simplifies the Lagrangian in Equation (3.16) into

$$\mathcal{L}(R^*, \boldsymbol{a}^*, \boldsymbol{\xi}^*, \boldsymbol{\lambda}^*, \boldsymbol{\gamma}^*) = \sum_{i=1}^n \lambda_i^* \boldsymbol{x}_i^T \boldsymbol{x}_i + \boldsymbol{a}^{*T} \boldsymbol{a}^* - 2 \sum_{i=1}^n \lambda_i^* \boldsymbol{x}_i^T \boldsymbol{a}^*. \tag{3.17}$$

Again, by Equation (3.15a), the optimal center $\boldsymbol{a}^*$ is known to be a convex combination of the data points with weights λ_i. This form of the optimal

center lets us simplify the Lagrangian in Equation (3.17) further, such that

$$\begin{aligned}
\mathcal{L}(R^*, \boldsymbol{a}^*, \boldsymbol{\xi}^*, \boldsymbol{\lambda}^*, \boldsymbol{\gamma}^*) &= \sum_{i=1}^{n} \lambda_i^* \boldsymbol{x}_i^T \boldsymbol{x}_i + \left(\sum_{i=1}^{n} \lambda_i^* \boldsymbol{x}_i \right)^T \left(\sum_{j=1}^{n} \lambda_j^* \boldsymbol{x}_j \right) \\
&\quad - 2 \sum_{i=1}^{n} \lambda_i^* \boldsymbol{x}_i^T \left(\sum_{j=1}^{n} \lambda_j^* \boldsymbol{x}_j \right) \\
&= \sum_{i=1}^{n} \lambda_i^* \boldsymbol{x}_i^T \boldsymbol{x}_i + \sum_{i=1}^{n} \sum_{j=1}^{n} (\lambda_i^* \boldsymbol{x}_i)^T (\lambda_j^* \boldsymbol{x}_j) \\
&\quad - 2 \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i^* \boldsymbol{x}_i^T (\lambda_j^* \boldsymbol{x}_j) \\
&= \sum_{i=1}^{n} \lambda_i^* \boldsymbol{x}_i^T \boldsymbol{x}_i - \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i^* \lambda_j^* \boldsymbol{x}_i^T \boldsymbol{x}_j.
\end{aligned} \tag{3.18}$$

This is the objective function for the dual problem of the primal problem in Equation (3.10).

To find the optimal value $\boldsymbol{\lambda}^*$, the dual problem in Equation (1.9) may be solved. The dual problem for the primal in Equation (3.10) is

$$\begin{aligned}
\underset{\boldsymbol{\lambda} \in \mathbb{R}^n}{\text{maximize}} \quad & \sum_{i=1}^{n} \lambda_i \boldsymbol{x}_i^T \boldsymbol{x}_i - \sum_{i,j=1}^{n} \lambda_i \lambda_j \boldsymbol{x}_i^T \boldsymbol{x}_j \\
\text{subject to} \quad & \sum_{i=1}^{n} \lambda_i = 1, \\
& 0 \leq \lambda_i \leq C, \quad i = 1, \ldots, n.
\end{aligned} \tag{3.19}$$

This problem may be solved efficiently by, for example, using sequential minimal optimization (SMO) algorithm [46]. The high level idea in SMO is to find two Lagrange multipliers λ_i and λ_j that violate the KKT conditions in Theorem 1.13. The two multipliers are then optimized as a one-dimensional constrained quadratic optimization using the dual constraints $\lambda_j = 1 - \lambda_i - \sum_{k \neq i,j} \lambda_k$ and $0 \leq \lambda_i, \lambda_j \leq C$.

To simplify the notation, from now on, let $(\bar{R}, \boldsymbol{a}, \boldsymbol{\xi}, \boldsymbol{\lambda}, \boldsymbol{\gamma})$ be the point that satisfies KKT conditions in Theorem 1.13. The classification of a point $\boldsymbol{z} \in \mathbb{R}^m$ depends on its distance to the center $\boldsymbol{a}$. Let $d_{\boldsymbol{a}}(\boldsymbol{z})^2 := \|\boldsymbol{z} - \boldsymbol{a}\|^2$ be the distance between $\boldsymbol{z}$ and the center. This distance can be computed using the following equation:

$$d_{\boldsymbol{a}}(\boldsymbol{z})^2 = \boldsymbol{z}^T \boldsymbol{z} - 2 \sum_{i=1}^{n} \lambda_i \boldsymbol{z}^T \boldsymbol{x}_i + \sum_{i,j=1}^{n} \lambda_i \lambda_j \boldsymbol{x}_i^T \boldsymbol{x}_j. \tag{3.20}$$

Equation (3.20) follows from Definition 1.16 where $\|\boldsymbol{z}-\boldsymbol{a}\|^2 = (\boldsymbol{z}-\boldsymbol{a})^T(\boldsymbol{z}-\boldsymbol{a})$ and the property that the optimal center is a convex combination of data points $\boldsymbol{a} = \sum_{i=1}^n \lambda_i \boldsymbol{x}_i$. The point $\boldsymbol{z}$ is in the positive class, if $d_{\boldsymbol{a}}(\boldsymbol{z})^2 \leq \bar{R}$ and in the negative class otherwise. The squared radius $\bar{R}$ can be solved with the following equation:

$$\bar{R} = \boldsymbol{x}_k^T \boldsymbol{x}_k - 2\sum_{i=1}^n \lambda_i \boldsymbol{x}_k^T \boldsymbol{x}_i + \sum_{i,j=1}^n \lambda_i \lambda_j \boldsymbol{x_i}^T \boldsymbol{x}_j \tag{3.21}$$

where $\boldsymbol{x}_k$ is any point at the boundary. A boundary point $\boldsymbol{x}_k$ is any point corresponding to the index that satisfies the inequality $0 < \lambda_k < C$. This follows from the complementary slackness in Equation (1.19) that states that $\lambda_i(\|\boldsymbol{x}_i - \boldsymbol{a}\|^2 - \bar{R} - \xi_i) = 0$ and $\gamma_i(-\xi_i) = 0$. Conditions $\lambda_i > 0$ and $\gamma_i > 0$ hold for such an index i that $C - \lambda_i - \gamma_i = 0$. That is the index i with $0 < \lambda_i < C$. Then $\|\boldsymbol{x}_i - \boldsymbol{a}\|^2 - \bar{R} - \xi_i = 0$ and $\xi_i = 0$, which means that $\|\boldsymbol{x}_i - \boldsymbol{a}\|^2 = d_{\boldsymbol{a}}(\boldsymbol{x_i})^2 = \bar{R}$.

Tax and Duin [59] proposed that the support vector data description can be made more flexible with *kernels*.

Definition 3.4. Kernel is a function $K : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$. Its value is defined as $K(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^T \phi(\boldsymbol{x}')$ with some $\phi : \mathbb{R}^m \to \mathbb{R}^M$.

The function $\phi : \mathbb{R}^m \to \mathbb{R}^M$ in Definition 3.4 is a feature mapping. It maps the data points from the input space into a higher dimensional feature space where $m < M$. Using the feature mapping ϕ to the data in the primal problem in Equation (3.10) transforms the primal problem into a new primal problem

$$\begin{aligned} \underset{\bar{R} \in \mathbb{R}, \boldsymbol{a} \in \mathbb{R}^M, \boldsymbol{\xi} \succeq \boldsymbol{0}}{\text{minimize}} \quad & \bar{R} + C\sum_{i=1}^n \xi_i \\ \text{subject to} \quad & \|\phi(\boldsymbol{x}_i) - \boldsymbol{a}\|^2 \leq \bar{R} + \xi_i, \quad i = 1, \ldots, n, \\ & \xi_i \geq 0, \qquad\quad i = 1, \ldots, n \end{aligned} \tag{3.22}$$

with $1/n < C \leq 1$. The dual is derived in a similar fashion as in Equation (3.19). Then the dual is

$$\begin{aligned} \underset{\boldsymbol{\lambda} \in \mathbb{R}^n}{\text{maximize}} \quad & \sum_{i=1}^n \lambda_i K(\boldsymbol{x}_i, \boldsymbol{x}_i) - \sum_{i,j=1}^n \lambda_i \lambda_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ \text{subject to} \quad & \sum_{i=1}^n \lambda_i = 1, \\ & 0 \leq \lambda_i \leq C, \quad i = 1, \ldots, n. \end{aligned} \tag{3.23}$$

The distance to the center is computed similarly to Equation (3.20), such that

$$d_{\boldsymbol{a}}^K(\boldsymbol{z})^2 = K(\boldsymbol{z},\boldsymbol{z}) - 2\sum_{i=1}^{n}\lambda_i K(\boldsymbol{z},\boldsymbol{x}_i) + \sum_{i,j=1}^{n}\lambda_i\lambda_j K(\boldsymbol{x}_i,\boldsymbol{x}_j). \tag{3.24}$$

A typical choice for a kernel in Equation (3.23) is the radial basis function (RBF) kernel $K(\boldsymbol{x},\boldsymbol{x}') = \exp(-\|\boldsymbol{x}-\boldsymbol{x}'\|^2/\sigma^2)$ [59, 60]. In the RBF kernel, the parameter $\sigma > 0$ is a bandwidth parameter that must be selected optimally. The dot product in the program in Equation (3.19) can also be thought as a kernel with a feature mapping $\phi(\boldsymbol{x}) = \boldsymbol{x}$. In fact, the dot product in Equation (3.19) is called a linear kernel in the Support Vector Machine literature.

There has been much research on the unsupervised selection of the bandwidth parameter for the RBF kernel. Aggarwal [1] suggested a selection of bandwidth σ to be of the same magnitude as the pairwise distances in the dataset. One possible choice for the bandwidth is $\sigma = \sigma_A := \text{median}_{i<j}(\|\boldsymbol{x}_i - \boldsymbol{x}_j\|)$ for large datasets of $n > 1000$ and $\sigma \in [2,3] * \sigma_A$ for smaller datasets with $n \leq 1000$ [1]. This division to large and small is due the fact that smaller dataset are more prone to induce overfitting. The subscript A refers to Aggarwal's method of choosing the bandwidth parameter. This method works well when the distribution of distances between data points is unimodal. That means that the domain of the positive class does not have a clear cluster structure. If the domain is clustered, there are better options for the unsupervised selection of the bandwidth parameter [13].

3.1.2 Local Outlier Factor

The Local Outlier Factor (LOF) is another one-class classifer. It is a density based outlier detection algorithm. It classifies data points into inliers and outliers based on the point's *local density* relative to the neighboring points' local densities. To estimate the local densities, it uses the distances from the point to its nearest neighbors. LOF was first defined in [8].

Before defining LOF, a few other definitions are needed. Let $\boldsymbol{X} \subset D$ be a dataset in some domain D. Let k be any positive integer. Let us now define *k-distance*:

Definition 3.5. The k-distance of an object $\boldsymbol{p} \in D$ related to the dataset $\boldsymbol{X}$ is

$$d_{k\text{NN}}(\boldsymbol{p}) := d(\boldsymbol{p},\boldsymbol{x}_{(k)}). \tag{3.25}$$

The object $\boldsymbol{x}_{(k)}$ is any object in the dataset $\boldsymbol{X}$ that satisfies both inequalities

$$\#\{\boldsymbol{x}_i \in \boldsymbol{X} \setminus \{\boldsymbol{p}\} \mid d(\boldsymbol{p},\boldsymbol{x}_i) \leq d(\boldsymbol{p},\boldsymbol{x}_{(k)})\} \geq k \tag{3.26}$$

and

$$\#\{\boldsymbol{x}_i \in \boldsymbol{X} \setminus \{\boldsymbol{p}\} \mid d(\boldsymbol{p}, \boldsymbol{x}_i) < d(\boldsymbol{p}, \boldsymbol{x}_{(k)})\} < k. \tag{3.27}$$

Remark 3.6. The cardinality of the set in Equation (3.26) may indeed be larger than k. This happens almost never if the data follows any continuous distribution. It is, however, possible on an ordinal scale.

In other words, k-distance is the distance to any object that is the kth closest to the query object in the dataset. With Definition 3.5, a *k-neighborhood* can be defined. The *k-distance neighborhood* or the k-neighborhood is the set of k closest objects in the dataset to the query object $\boldsymbol{p}$. Formally, the definition is:

Definition 3.7. The k-neighborhood of the object $\boldsymbol{p}$ in the dataset $\boldsymbol{X}$ is

$$kNN(\boldsymbol{p}) := \{\boldsymbol{x}_i \in \boldsymbol{X} \setminus \{\boldsymbol{p}\} \mid d(\boldsymbol{p}, \boldsymbol{x}_i) \leq d_{kNN}(\boldsymbol{p})\}. \tag{3.28}$$

From Definition 3.7, a rather unintuitive result can be seen that the size of the k-neighborhood is not necessary k. Indeed, the size of the k-neighborhood may be larger than k as shown in Remark 3.6. This situation occurs when there are multiple points in the dataset that are exactly k-distance away from the query.

Intuitively, local density is a measure of how tightly packed the objects are in some locality. This intuition leads to an idea that density is inversely proportional to the distances to the nearest neighbors. To reduce the fluctuation, *reachability distance* is used instead of the actual distance.

Definition 3.8. The reachability distance is

$$d_{reach}(\boldsymbol{x}, \boldsymbol{x}_i) := \max\{d_{k\mathrm{NN}}(\boldsymbol{x}_i), d(\boldsymbol{x}, \boldsymbol{x}_i)\} \tag{3.29}$$

with $\boldsymbol{x} \in D$ and $\boldsymbol{x}_i \in \boldsymbol{X}$.

Reachability distance is called a distance in the literature even though it is not symmetric. The local density can be estimated by using reachability distance. *Local reachability density* is the inverse of the average reachability distance from the object to its nearest neighbors, formally defined as

$$lrd(\boldsymbol{x}) := 1/\frac{\sum_{\boldsymbol{x}_i \in kNN(\boldsymbol{x})} d_{reach}(\boldsymbol{x}, \boldsymbol{x}_i)}{\#kNN(\boldsymbol{x})}. \tag{3.30}$$

LOF-score of an object $\boldsymbol{x}$ is the average of ratios of local reachability density of $\boldsymbol{x}$ to local reachability density of its nearest neighbors.

Definition 3.9. The LOF-score of a point $\boldsymbol{x}$ relative to a dataset $\boldsymbol{X}$ is

$$LOF(\boldsymbol{x}) := \frac{1}{\#kNN(\boldsymbol{x})} \sum_{\boldsymbol{x}_i \in kNN(\boldsymbol{x})} \frac{lrd(\boldsymbol{x}_i)}{lrd(\boldsymbol{x})} \tag{3.31}$$

where each $\boldsymbol{x}_i \in \boldsymbol{X}$.

LOF scores each object based on its local density. The object is classified as an outlier if its score exceeds some threshold value, such that $LOF(\boldsymbol{x}) > \theta$. Unlike with SVDD, the threshold for LOF-scores is not determined through the optimization procedure. One possible way of selecting the parameter θ is to first estimate the number of outliers n_{out} in the data set $\boldsymbol{X}$ and then set the threshold θ to the LOF-score of the n_{out}th highest scoring object. The fraction of estimated outliers in the data, namely n_{out}/n, is called *contamination*. This approach, of course, requires some prior knowledge of the data. The other parameter in LOF algorithm is the number of neighbors k.

3.2 The Use of One-Class Classification in the Semi-Supervised Classification of Time Series

One-class classifiers are able to construct a normal operating domain. They are binary classifiers that classify the objects in the normal operating domain into the positive class and the objects outside it into the negative class. Provided that there is data from the normal operation, a one-class classifier can be trained to find the normal operating domain. In this thesis, one dataset at a time is selected to represent the normal operation. In Section 4.3.1, the normal operation is signing a specific sign language sign, and in Section 4.3.2, it is a signature from a legitimate person. The hypothesis in this thesis is that the processes that result in multiple observations outside the normal operating domain can be classified as abnormal. Zhao, Wang and Xiao [70] used SVDD in a similar way to detect faults in chillers.

Another way to use OCC involves using the datasets as time series. Each time series is viewed as an instance. The normal operating domain can be defined inside the collection of time series based on the distances between the time series. The abnormal processes are then the time series that are far away from the example time series. This approach, however, requires multiple examples of normal processes, which, by the assumption, are not available.

A second problem with this case rises from the fact that DTW is not a metric as defined in Definition 2.6. Kernel methods, such as SVDD, assume that the kernel is positive semi definite. Lei and Sun [35] proved that DTW creates kernels that are not positive semi definite. This is likely due to DTW

not being a metric. Thus, using DTW to construct a kernel may weaken the performance of SVDD. Lei and Sun proved this worsened performance experimentally in [35].

This thesis uses the first approach: one dataset is used to compute the normal operation domain, and the unlabelled datasets are then compared against this learned domain. Let the dataset $\boldsymbol{X}$ be the example time series. Let us train the selected one-class model using $\boldsymbol{X}$ as the positive dataset. Then, for every other time series $\boldsymbol{Y}$, each time instant $\boldsymbol{y}_i$ is tested against the trained model. The discriminating factor is how many of those time instants are inside the normal operating domain learned by the selected OCC model. It is assumed that the time series are more likely normal processes when most of the time instants are within this normal operating domain. For each time series $\boldsymbol{Y}$, a score x is recorded that is the fraction of time instants within the normal operating domain.

The scores x are separated into normal and abnormal using a mixture model of two normal distributions. Let us assume that the score x follows a normal distribution both for normal operation and defected operation. Let $x \sim N(\mu_1, \sigma_1)$ for normal operation and $x \sim N(\mu_2, \sigma_2)$ for abnormal operation. Then the distribution of scores x is the mixture distribution

$$\begin{aligned} f(x) &= \mathbb{P}(C = +1) f(x|C = +1) + \mathbb{P}(C = -1) f(x|C = -1) \\ &= \frac{\tau}{\sqrt{2\pi}\sigma_1} \exp\left(-\frac{1}{2\sigma_1^2}(x - \mu_1)^2\right) + \frac{1-\tau}{\sqrt{2\pi}\sigma_2} \exp\left(-\frac{1}{2\sigma_2^2}(x - \mu_2)^2\right) \\ &= \tau f_1(x) + (1 - \tau) f_2(x). \end{aligned} \tag{3.32}$$

In Equation (3.32), the parameter τ is the prior probability of time series $\boldsymbol{Y}$ being a normal process. In other words, $\tau = \mathbb{P}(C = +1)$. The parameters τ, μ_1, μ_2, σ_1 and σ_2 may be estimated using Expectation-Maximization (EM) algorithm [16]. Using Bayes' theorem, an estimate for the posterior probability can be defined as

$$\hat{\mathbb{P}}(C = +1|x) = \frac{\hat{\tau}\hat{f}_1(x)}{\hat{f}(x)} \tag{3.33}$$

with $\hat{f}_i = (\sqrt{2\pi}\sigma_i)^{-1} \exp[-(x - \hat{\mu}_i)^2/(2\hat{\sigma}_i^2)]$ for $i = 1, 2$ and $\hat{f}(x) = \hat{\tau}\hat{f}_1(x) + (1-\hat{\tau})\hat{f}_2(x)$. Now the time series $\boldsymbol{Y}$ is classified as normal operation if $\hat{\mathbb{P}}(C = +1|x) > 1/2$ and abnormal otherwise.

3.3 Positive-Unlabelled Learning

Without negative class examples or perfectly representable data from the entire positive class domain, novelty detection methods' capabilities limit to finding novel patterns. The model cannot separate the desirable behavior from the undesirable but rather the normal behavior from the previously unseen [61]. The model in the setting of this thesis needs a method for telling whether the previously unseen behavior is normal or abnormal. There are two approaches to this. The first one is active learning. In active learning, the model chooses its most uncertain predictions. The model then proceeds to ask for labels on those uncertain predictions from an oracle, which usually is a human expert [53]. The other method is PU learning. PU learning is a special case of SSL. In PU learning, the training set consists of some positive examples and a set of unlabelled examples from both positive and negative classes. In this thesis, PU learning approach is chosen since active learning requires occasionally guidance from the user, and that is against the idea of automation.

PU learning has drawn attention since in many domains it is hard to acquire labelled data [20, 43, 51]. Chapelle, Schölkopf and Zien [12] note that the SSL can improve classification if the data satisfy a low density separation assumption. This assumption holds if the boundary between the classes lies within the low density region. The PU models presented in this thesis rely on this assumption.

Wei and Keogh [67] argue that unlabelled data can aid in the classification problem. This encourages us to consider PU-learning as an alternative to OCC. With PU-learning, it is possible to classify the data with only one positive example [14], while one-class classification still needs a representative sample of positive data [45]. This is illustrated in Figure 5. One PU framework, self-training, has proved to be especially useful in SSL of time series. Self-training will be studied in the following section.

3.4 Self-training

One method that is capable of learning from under sampled positive and unlabelled data is self-training [49]. This means that self-learning is a form of PU learning. The self-training methods add observations from the unlabelled data into the positive class one by one until a stopping criterion is met. The model is retrained between each addition with the current labels. Wei and Keogh [67] proposed this kind of method for SSL of time series data. They proposed that the stopping criterion can be based on the minimum distance between classified instances. The original method proposed by Wei and Keogh

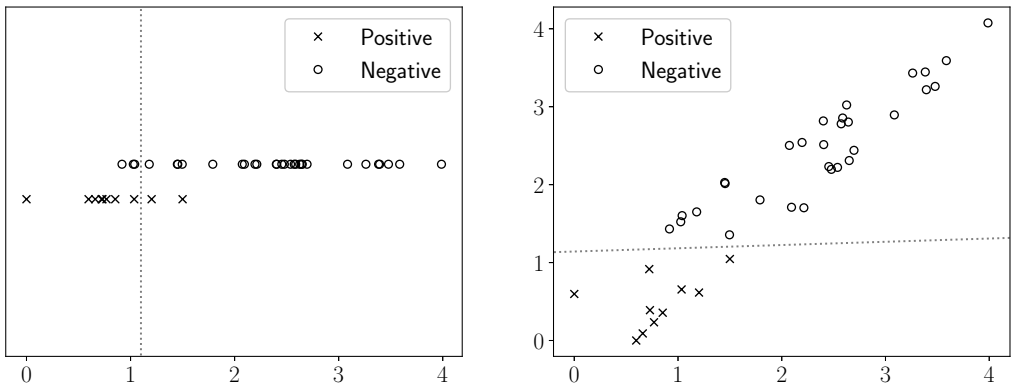

Figure 5: An example of the PU-learning. Left: The distance from the first positive example. The classes cannot be separated linearly based on the distance to the positive example. Right: The y-axis shows the distance from the next closest example. This data is acquired from the first step of self-training. This data is linearly separable.

works as an early stopping criterion, when the negative class much denser than the positive class [40]. However, if the assumption of denser negative class does not apply or if early stopping is not desirable, better stopping criteria are needed.

Self-training is an iterative method in which the model is retrained in each iteration with its most confident predictions [63]. In this thesis, a 1NN-DTW classifier will be used in the iterative training. With 1NN-DTW, the most confident prediction will be the time series that is the closest or the most similar to any time series in the positive class. This intuition is used in [40, 49, 14, 21]. This idea is illustrated in Figure 6.

To my knowledge, there are four stopping criteria for time series self-training in the literature:

- WK: Wei-Keogh criterion [67]
- RW: Ratanamahatana-Wanichsan criterion [49],
- CBD-GA: Class Boundary Detection using Graphical Analysis [21],
- LCLC: Learning from Common Local Clusters [40, 41].

This thesis will propose a new stopping criterion called Peak Evaluation using Perceptually Important Points (PIP) in Section 3.4.3. All of these criteria are based on *MinDist*-series. The *MinDist*-series are defined later in this

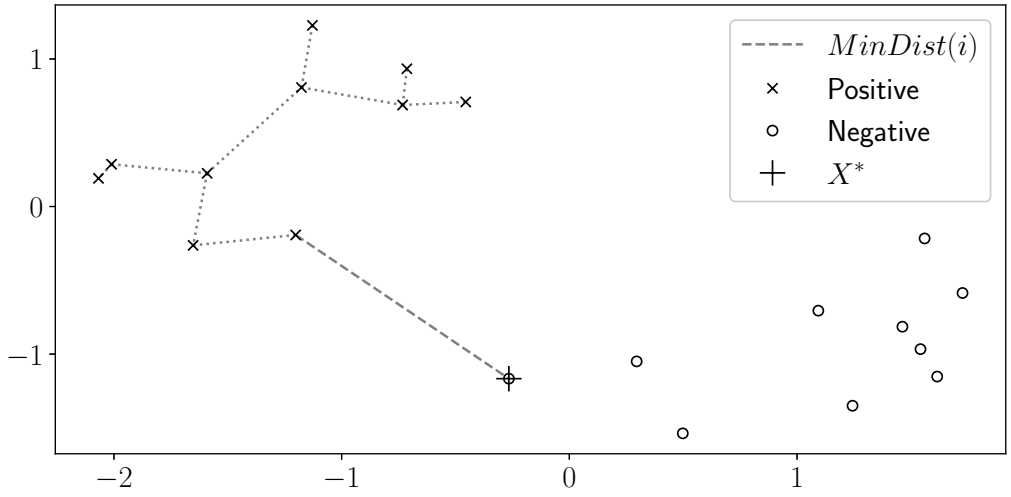

Figure 6: An example of Algorithm 5. The algorithm advances over the class boundary at the step i. This results in the longest distance in the $MinDist$ tree.

section. LCLC differs from the others as it first clusters the time series. LCLC has been constructed only to use Euclidean distance on UTSs. This is too restrictive for the use cases in this thesis, so LCLC will no longer be considered. In addition, WK is considered to be rather inaccurate [49, 21] and likely to result in early stopping [40]. WK will not be considered further in this thesis because of this reported low performance.

Ratanamahatana and Wanichsan [49] define the algorithm for self-training that uses 1NN-DTW. This algorithm is presented in Algorithm 5. Gonzáles et al. [21] call this algorithm 1NN propagation. The Algorithm 5 assumes a 1NN-DTW-classifier. For a more general algorithm, Steps 2 and 8 must be replaced. In Step 2 the time series $\boldsymbol{X}^*$ is selected as the most confident candidate to the positive class from the unlabelled set for the chosen classifier.

In [49, 21, 40, 41], the information recorded in Step 3 is the $MinDist$-series. The $MinDist$-series is defined for the i:th time series $\boldsymbol{X}^*_{(i)}$ moved from U to P as $MinDist(i) = \min_{Y \in P} \mathrm{DTW}(\boldsymbol{X}^*_{(i)}, Y)$. An example of a $MinDist$-series is presented in Figure 7. In other words, the $MinDist$-series records the minimum distance between a time series in the positive set and a time series in the unlabelled set. This distance can be interpreted as the minimum distance between the positive set and the unlabelled set [40]. This hints that the $MinDist$-series might hold information on the optimal stopping criterion.

Algorithm 5 1NN-DTW self-training

Require: Positive set P, unlabelled set U, stopping criterion
Ensure: Binary 1NN-DTW-classifier

1: **while** $U \neq \emptyset$ **do**
2: Select $\boldsymbol{X}^* = \arg\min_{\boldsymbol{X} \in U}\{\mathrm{DTW}(\boldsymbol{X}, \boldsymbol{Y}) | \boldsymbol{Y} \in P\}$
3: Record the information the stopping criterion uses
4: $P \leftarrow P \cup \{\boldsymbol{X}^*\}$
5: $U \leftarrow U \setminus \{\boldsymbol{X}^*\}$
6: **end while**
7: Divide P into positive set P and negative set N based on information from Step 3 using stopping criterion
8: Train a 1NN-classifier with labelled sets P and N

3.4.1 RW-criterion

The RW-criterion was first introduced in [49]. It is based on finding the maximum of stopping criterion confidence (SCC). Let $MinDist$ be defined as in Section 3.4 and let $N_U = \#U$ be the number of time series initially in the unlabelled set. Ratanamahatana and Wanichsan [49] define the SCC as

$$SCC(i) = \frac{|MinDist(i) - MinDist(i-1)|}{\mathrm{Std}(MinDist(1), \ldots, MinDist(i))} * \frac{N_U - (i-1)}{N_U}. \tag{3.34}$$

In other words, SSC is the detrended $MinDist$-series scaled by the standard deviation so far. In addition, SSC is penalized linearly as the number of time series added to the set P increases. The optimal stopping criterion is the iteration that results in the maximum SCC. The positive labels are then assigned to the time series up to the index $i - 2$.

3.4.2 CBD-GA-criterion

The CBD-GA-criterion, introduced in [21], is a self-training method based on the graphical analysis of the $MinDist$-series. Graphical analysis is used to segment the $MinDist$-series into intervals. Each interval starts with an ascending curve, continues with a descending curve and ends in a stable curve. These intervals hold information about the behavior of the data in the low-density region that Algorithm 5 is modelling. The following definition is based on the work in [21].

The rules to segment the $MinDist$-series are presented here. The ascending curve is a segment in which $MinDist(i) \leq MinDist(i+1)$. Similarly, the descending curve is a segment in which $MinDist(i) \geq MinDist(i+1)$. The stable curve is defined based on the preceding descending curve. The stable

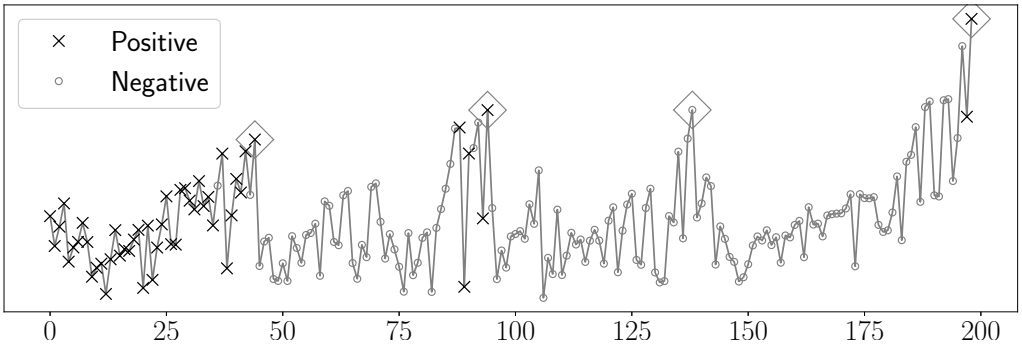

Figure 7: An example of the $MinDist$ series. On x-axis is the order number of a move of a time series from the unlabelled set to the positive set. On y-axis is the distance from the moved time series to the nearest time series in the positive set. There is a clear local maximum in the $MinDist$ series at the class boundary at the index 44. There are four peaks that are marked with diamonds. The four peaks suggest four clusters in the structure of the data. There are some outlying examples from the target class at the indices around 90 and 200.

curve is a segment in which $MinDist(e) - \beta * hd(s,e) \leq MinDist(i) \leq MinDist(e) + \beta * hd(s,e)$. In this definition of the stable curve, the index s marks the starting index of the preceding descending curve, and the index e marks the ending index of the preceding descending curve. The function $hd(s,e) = MinDist(s) - MinDist(e)$ is the height of the preceding descending curve and $\beta \in (0,1]$ is a parameter that controls the definition of stability. Gonzáles et al. [21] proposed a parameter value $\beta = 0.3$. They rationalize this selection with empirical experiments.

The CBD-GA algorithm segments the $MinDist$-series into intervals I using these definitions of curves. The first interval starts with the first ascending curve. When the ascending curve ends, the algorithm starts to explore the descending curve beginning from the index the ascending curve ends. When the descending curve ends, the algorithm explores, how many of the successive indices fit the definition of the stable curve. The first interval ends to the final index of this stable curve. Then the algorithm proceeds to find the following intervals in a similar manner.

Gonzáles et al. [21] provide interpretation for each curve and a connection to the class boundary. When in the ascending curve, the self-training algorithm searches time series in sparse region near the class boundary. When in the descending curve, the self-training algorithm has passed the class boundary and moves towards a denser region. When in the stable curve, the self-

training algorithm learns the new class in its densest region. This means that the optimal stopping criterion is found at the index that marks the end of the ascending curve and the start of the descending curve.

To find the optimal interval, the intervals are scored. Gonzáles et al. [21] define five scoring functions for the intervals. Let $ha(I)$ be the height of the ascending curve of the interval, $hd(I)$ the height of the descending curve of the interval, and $ws(I)$ the length of the stable curve of the interval. In addition, let us define the following normalized measures: $hdn(I) := hd(I)/\max MinDist(i)$ and $wsn(I) := ws(I)/(N_U - 1)$. Let I_s be the index of the end of the ascending curve and start of the descending curve on the interval. The linear weight for the index i is defined as $LW(i) := (N_U - i + 1)/N_U$. Now, the five scoring functions are defined in the following definition:

Definition 3.10. The scoring functions for CBD-GA are

- $SC_1(I) = hd(I) * LW(I_s)$,
- $SC_2(I) = ha(I) * LW(I_s)$,
- $SC_3(I) = ws(I) * LW(I_s)$,
- $SC_4(I) = \max\{hd(I), ha(I)\} * LW(I_s)$,
- $SC_5(I) = \max\{hdn(I), wsn(I)\} * LW(I_s)$.

The positive labels are given to indices up to the index $I_s - 1$ of the highest scoring interval.

3.4.3 Peak Evaluation Using Perceptually Important Points

This section presents a novel stopping criterion for 1NN-DTW in Algorithm 5. The algorithm is called Peak evaluation using perceptually important points. It was inspired by the perceptual skeleton presented in [39].

Extracting important points is a dimensionality reduction technique that puts an emphasis on preserving the local minima and maxima of the time series [47]. The goal is to extract the first significant local maximum from the *MinDist*-series. This first significant local maximum is the most likely index in the *MinDist*-series at which the 1NN-DTW Algorithm 5 moves over the class boundary. This intuition follows from the assumption that the class boundary lies within the low-density region, where the distances between observations are greater compared to the high-density regions. These greater distances cause peaks in the *MinDist*-series.

Mojsilović [39] defined perceptually important points (PIP) for MTSs. The definition is inspired by the way how human mind perceives the series. The human mind constructs a simplified representation of the series by perceiving and remembering the most important turning points. First, the time series $\boldsymbol{X} \in \mathbb{R}^{n*m}$ is appended by the time stamps. The appended time series is the series $\boldsymbol{Z} := (i, \boldsymbol{x}_i)_{i=1}^{n} \in \mathbb{R}^{n*(1+m)}$ where $\boldsymbol{x}_i$ is an observation in the time series $\boldsymbol{X}$ at the time stamp or the index i. The first two PIPs are the first and the last appended observations $(1, \boldsymbol{x}_1)$ and $(n, \boldsymbol{x}_n)$. These two PIPs define a line L in the appended space $\mathbb{R}^{1+m}$ such that $L = \{z \in \mathbb{R}^{1+m} \mid z = t(1, \boldsymbol{x}_1) + (1-t)(n, \boldsymbol{x}_n),\ t \in \mathbb{R}\}$. Each appended observation is then projected to the line L. The appended observation $(i, \boldsymbol{x}_i)$ with the maximal distance from itself to its projection is selected as the next PIP. The algorithm continues by defining lines between each successive PIP. Each following PIP is selected as the appended observation that has the maximal distance to the its projection on the closest line L. [39]

The definition of PIPs used in this thesis is based on the definition in [39]. In this definition, linear interpolation is used instead of the lines L that are used in [39]. The linear interpolation works just like the cubic spline interpolation in Equation 2.1, but the cubic functions C_i are replaced by linear functions $L_i(x) = b_i x + a_i$. Also, linear interpolation only uses the constraints in Equation 2.2. This is because there are only $2(n-1)$ coefficient b_i and a_i to solve in the linear interpolation.

First, the first and the last observations $\boldsymbol{x}_1$ and $\boldsymbol{x}_n$, respectively, are selected as first two PIPs. Then the set of PIPs are interpolated into time series of length n using linear interpolation. This interpolation is done one feature at the time for MTSs. The next PIP is the time instant with the greatest distance to its interpolated counterpart. The interpolation and the PIP selection steps are repeated until the interpolation is a good enough of an approximation to the original time series. This process is presented formally in Algorithm 6.

Algorithm 6 will eventually end. After n picks of PIPs, the interpolated time series $\boldsymbol{Z}$ will be the same as the original time series $\boldsymbol{X} = \boldsymbol{Z}$. Then the approximation error is $d(\boldsymbol{Z}, \boldsymbol{X}) = 0$. However, the stopping condition at Step 8 is not well defined. This thesis proposes to record the approximation error for each interpolation loop such that

$$e_l = d(\boldsymbol{Z}_l, \boldsymbol{X}). \tag{3.35}$$

In Equation (3.35), the series $\boldsymbol{Z}_l$ is the interpolation at the loop l as defined in Step 7 in Algorithm 6. This procedure will generate error series $\boldsymbol{e} = (e_1, \ldots, e_{n-2})$ that is usually mostly descending. The error series decreases

Algorithm 6 Find perceptually important points

Require: Time series $\boldsymbol{X} \in \mathbb{R}^{n*m}$

Ensure: Perceptually important points P, indices I

1: $P \leftarrow \{\boldsymbol{x}_1, \boldsymbol{x}_n\}$
2: $I \leftarrow \{1, n\}$
3: $l \leftarrow 1$
4: **for all** $j = 1, \ldots, m$ **do**
5: $\quad \boldsymbol{Z}_l^{(j)} \leftarrow$ linear interpolation $f(D)$ for $f : [1, n] \rightarrow \mathbb{R}$ using pairs $(i, f(i))$ with $i \in I$ and $f(i) = x_i^{(j)}$ and set $D = \{1, \ldots, n\}$
6: **end for**
7: $\boldsymbol{Z}_l \leftarrow (\boldsymbol{Z}_l^{(1)}, \ldots, \boldsymbol{Z}_l^{(m)}) \in \mathbb{R}^{n*m}$
8: **if** $\boldsymbol{Z}_l$ is not a good enough of an approximation to $\boldsymbol{X}$ **then**
9: $\quad$ **for all** $i = 1, \ldots, n$ **do** $d_i \leftarrow d(\boldsymbol{x}_i, \boldsymbol{z}_i)$
10: $\quad$ **end for**
11: $\quad i^* \leftarrow \arg\max\{d_i\}_{i=1}^n$
12: $\quad$ Append P with $\boldsymbol{x}_{i^*}$
13: $\quad$ Append I with i^*
14: $\quad l \leftarrow l + 1$
15: $\quad$ Go to Step 4
16: **end if**

rapidly at the beginning and slower towards the end. This is because the first approximations are coarse while the later approximations are finer due to the maximization Step 11 in Algorithm 6. The approximation $\boldsymbol{Z}_l$ will be good enough at the index l where the "elbow" [30] of the error series can be found. The elbow is a major turning point in the error series. It is the point, where the previous errors decline quickly and the following errors decline slower. This elbow-method has been traditionally used to find the number of clusters in a dataset [30].

In this thesis, the elbow is defined as the point that deviates the most from the straight line. First, the error series is scaled to the range $[0, 1]$ so that it may compared to the line between the points $(1, 1)$ and $(n - 2, 0)$. The value of this line at the index l is $(n-2-l)/(n-3)$. This line represents an error series without any elbow.

The errors in the scaled error series will be compared to the values on that line. Since the scaled error series decrease more rapidly at the beginning, the elbow will reside below this line. Comparing of the error series to the straight line can be seen as a linear penalty. With this in mind, comparing

is equivalent to adding a negative, linear penalty

$$-\frac{n-2-l}{n-3} = -1 + \frac{l-1}{n-3} \tag{3.36}$$

to each scaled element of the error series with an index $l = 1, \ldots, n-2$. Mathematically, finding the elbow in the error series is now equivalent to minimization

$$l^* = \underset{l=1,\ldots,n-2}{\arg\min} \left(\frac{e_l}{e_{max}} + \frac{l-1}{n-3} \right). \tag{3.37}$$

In Equation (3.37), the element e_l is the error defined in Equation (3.35), and the element $e_{max} := \max\{e_1, \ldots e_{n-2}\}$ is the largest error in the error series. The optimal approximation is the approximation $\boldsymbol{Z}_{l^*}$. In other words, the optimal approximation has $l^* + 2$ PIPs.

The high-level idea in finding the PIPs is to simplify the *MinDist*-series just the right amount. The goal is to get rid of minor fluctuations but, at the same time, preserve the significant changes. This is also the motivation for the novel stopping condition presented in this thesis for Algorithm 6: if the following PIPs will reduce the approximation error a lot, all the significant changes has not yet been found. Also, if the following PIPs do not improve the approximations enough, the PIPs are starting to capture the minor fluctuations.

When approximating the *MinDist*-series using PIPs, the first local maximum in the interpolated series $\boldsymbol{Z}_{l^*}$ is the point of interest. Of course, if the first local maximum is the first element $\boldsymbol{z}_1 = \boldsymbol{x}_1$, the second local maximum will be considered. This local maximum in the interpolation represents the first significant peak in the *MinDist*-series. The following local maxima are the following significant peaks. Each peak suggests that Algorithm 5 travels through a low density region from a cluster to another. I will argue that the first significant peak found this way is the optimal stopping criterion for 1NN-DTW Algorithm 5. Therefore, the first cluster is classified as positive and all the following clusters negative. This stopping criterion, along with RW- and CBD-GA-criteria, are presented graphically in Figure 8.

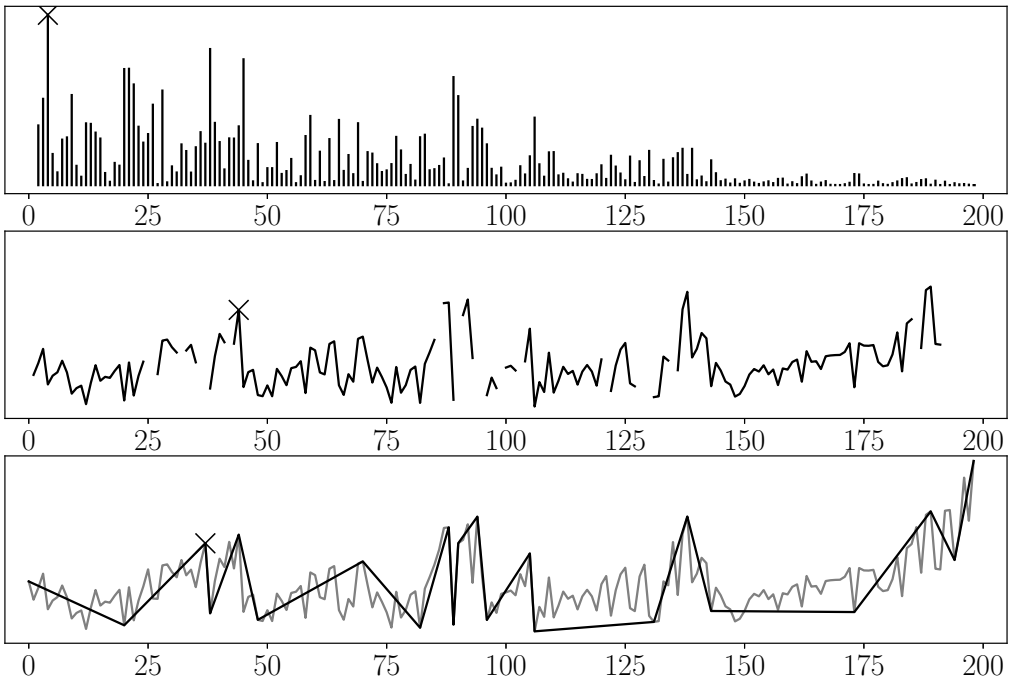

Figure 8: The three stopping criteria used in this thesis on the *MinDits*-series presented in Figure 7. The optimal stopping point of each criterion is marked with a cross. Top: RW-criterion. The data are the stopping confidence criteria defined in Equation (3.34). Middle: CBD-GA-criterion. The figure shows the segments formed by the CBD-GA-algorithm. Bottom: Peak evaluation using perceptually important points. On the foreground in black is the approximation $\boldsymbol{Z}$ defined in Algorithm 6 in Step 7. On the background in gray is the original *MinDist*-series of Figure 7 for reference.

4 Experiments

In this section, the methods presented in this thesis are compared using three different datasets.

4.1 Unsupervised Model Evaluation

Evaluation of one-class classifier is not straightforward. The distribution estimation of the negative class is very hard if not impossible in many cases. This is a direct effect of the lack of negative examples. Without the distribution estimate of the negative class, performance evaluation is based on assumptions [64]. One such assumption is that the distribution of the negative class is uniform over some domain $D \subset \mathbb{R}^m$. In higher dimensions, however, this assumption becomes highly inefficient [59]. The evaluation in the PU setting is not any easier. Without annotated data there is no ground truth to compare the results against.

For these reasons, the models will be evaluated in a supervised setting. This way it is possible to get accurate indicators of the performance of the selected models. The models, however, do not benefit from the use of the annotated data, so this approach may still be called semi-supervised.

The datasets will not be divided into training and testing datasets like in [49, 14]. The reason is that there are not many instances in each class. Besides, the performance of the self-training Step 7 in Algorithm 5 is considered more crucial in this thesis. This evaluation setting was also used in [21].

4.2 Design of the Testing Software

The experiments were executed on computer with a custom made testing software. The software was designed to implement the data processing and the models presented in this thesis. The implementation handles the testing pipeline that consists of data reading, preprocessing, feature subset selection, distance calculations and classification. The testing pipeline for one-class methods SVDD and LOF differ from the one presented here such that they handle the distance calculations implicitly.

The implementation of the testing software is based on a modular, layered architecture. The advantage of using layers is that each layer may be changed without affecting the other layers. There are three layers: data reading, preprocessing and analysing. Each module completes one task inside one layer. The user may add existing modules and write new modules depending on the need. I implemented a module for reading from file in data reading layer.

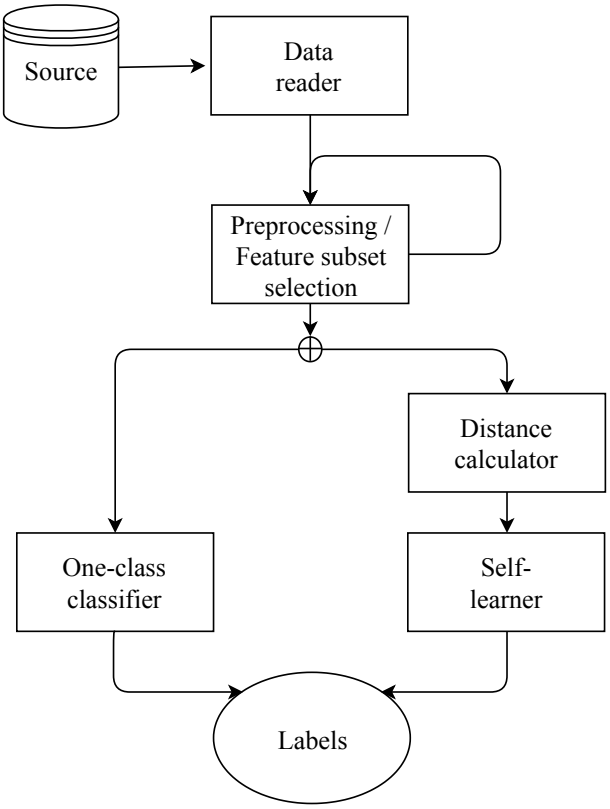

Figure 9: The deployment of the testing software.

In preprocessing layer, I implemented Z-normalization in Equation (2.7), cubic spline resampling in Equation(2.6) and CLeVer-Hybrid in Section 2.2.2. In analysing layer, I implemented SVDD in Equation (3.23) and LOF in Equation (3.31) using a one-class interface, ED in Equation (2.9), DTW in Equation (2.10) and PCA-SF in Equation (2.45) using a distance measure interface and RW in Equation (3.34), CBD-GA in Definition 3.10 and PIP in Equation (3.37) using a self-training interface.

Now I will explain the deployment of the testing software. A data reader is always needed. The preprocessing and feature subset selection modules are optional but they can always be linked together. This follows from a property of preprocessing and feature subset selection modules: both input and output a time series database. One-class methods operate directly on a time series database. Distance calculators calculate the distance matrix using the time series database and self-learners analyse those distance matrices. This means that a self-learner must always be paired with some distance calculator. The deployment process is presented in Figure 9.

The testing software was coded using Python 3.5. Python was chosen because of its ease of prototyping and its extensive machine learning libraries. For SVDD I used implementation in LIBSVM-3.22 with LIBSVM Tools [10], for LOF Scikit-Learn-0.19 [44] and for DTW TSLearn-0.1.18 [58].

The data reader reads the time series from the disk and loads them into memory as a 3-dimensional NumPy-array. The first dimension is the number of time series, the second is the length of the time series and the third is the number of features per time series. Preprocessors process the NumPy-array in place. One-class methods process the time series as such and output predicted labels for each time series. The self-learners use distance matrices $\boldsymbol{D} = [d(\boldsymbol{X}_i, \boldsymbol{X}_j)]_{i,j=1}^{N}$ provided by the distance calculators. The self-learners also output the predicted labels.

4.3 Evaluation

In binary classification, there are two types of errors. Type 1 error or false positive (FP) is an instance from the negative class falsely classified as positive. Type 2 error or false negative (FN) is the opposite: an instance from the positive class falsely classified as negative. These two errors may be compactly presented in a confusion matrix in Table 2.

Table 2: Confusion matrix

		Observed	
		Positive	Negative
Predicted	Positive	#TP	#FP
	Negative	#FN	#TN

This thesis uses evaluation metrics precision and recall and their harmonic mean also known as F_1-score.

$$\text{Precision} = \frac{\#\text{TP}}{\#\text{TP} + \#\text{FP}} \tag{4.1a}$$

$$\text{Recall} = \frac{\#\text{TP}}{\#\text{TP} + \#\text{FN}} \tag{4.1b}$$

$$F_1\text{-score} = \frac{2}{(\text{Precision})^{-1} + (\text{Recall})^{-1}} \tag{4.1c}$$

In the tables of this section, the following abbreviations will be used: RW for Ratanama-Wanichsan-criterion, GA-l for class boundary detection using

graphical analysis criteria and PIP for peak evaluation using perceptually important points. The number l in abbreviation GA-l refers to the scoring function in Definition 3.10. Likewise, the abbreviation DTW-k% refers to dynamic time warping with a window length of k percent of the length of the time series. Also, CLeVer means CLeVer-hybrid.

4.3.1 Australian Sign Language

The first dataset is the Australian sign language (Auslan) sign data [27]. The data are hand movements of nine native Auslan signers. All signers signed 95 different Auslan signs, and three takes were recorded from each signer. The hand movements were recorded using positional sensors on both hands. Both hands were recorded on six degrees of freedom. In addition, every finger was tracked on how bent they are. In summary, Auslan data set consists of 2565 time series of varying length from 95 classes with 22 features. The 22 features come from the 22 sensors.

In each class, one example is taken as the initial positive example. The positive example is the sign signed by the ninth signer on the first take. This example was chosen because its nearest neighbor is most frequently in the same class across all classes and every distance used.

Z-normalization is used on each time series. The Z-normalization was used, because the absolute positions are irrelevant. The absolute position is affected for example by the size difference between the signers.

One class at a time will represent the positive class. All the rest 94 classes construct the negative class. This setting is highly unbalanced. As a result the simulated expected F_1-score for this setting is 2.1% with a random classifier.

To experiment the performance of feature selection, 12 features were selected based on prior knowledge of the data. It is typical for the Auslan that many signs are signed using only the dominant hand. It is also known from the description of the data set that the readings from the fingers may be inaccurate [27]. Using these bits of information I picked 12 features: x-, y- and z-positions with pitch, yaw and roll recordings on the dominant hand, thumb, index and middle finger recordings on the dominant hand, and x-, y- and z-positions on non-dominant hand. CLeVer-Hybrid was used twice to first select 4 features and then again to select 12 features. First time CleVer-hybrid picked the y- and z-positions of the non-dominant hand, and the thumb and the index finger readings also from the non-dominant hand. Second time CleVer-hybrid picked all the features from the non-dominant hand, and the x-position of the dominant hand.

The movements that define the meaning of the sign can be quite complex. To model these complex movements, appropriate values for the hyperparam-

eters must be used in the models. This means a low value for σ in SVDD and a small neighborhood size in LOF. Let us also make the assumption that the datasets do not include outliers. This is a decent assumption, since one dataset represent one sign. The maximum penalty will be used for SVDD, and the minimum contamination for LOF. The average performance of the OCC approaches are presented in Table 3.

Table 3: Auslan as one-class classification average F_1-scores

	SVDD	LOF	SVDD*	LOF*
CLeVer-4	2.9	2.7	**9.6**	7.9
CLeVer-12	3.1	3.8	**15.2**	11.3
Hand-picked 12	4.2	5.6	**20.4**	17.6
No FS	4.3	5.7	16.9	**17.3**

For SVDD, I used parameters $\sigma = \sigma_A$ and $C = 1$. For LOF, I used a contamination of 0% and the neighborhood size 3. The neighborhood size 3 was selected because it is the lowest neighborhood size that allows us to use reachability distance [8].

Both approaches, SVDD and LOF, used normal distribution assumption on the scores. The scores were classified based on the EM-algorithm using Equation (3.33). The results for SVDD and LOF in Table 3 are rather disappointing since both classifiers are hardly beating the random classifier. This may be because of the normal distribution assumption. For this reason I tried a linear classifier on LOF and SVDD scores that maximize the F_1-scores using the annotated data. This is of course cheating because using annotated data changes this approach from semi-supervised to supervised. These results are presented in Table 3 as LOF* and SVDD*.

The performance of the OCC models is low. This may be because the number of observations in the dataset is quite low. With only 58 observations on average in each datasets, it is really unlikely to get a good representation of the normal operating domain.

Before applying the time series distance measures, the time series are resampled into the average length of 58 time points. The average length is used so that the lengths of the time series change as little as possible on average. For distance measures, ED, DTW and PCA-SF will be used. The window parameter for DTW is selected as 10% by the rule of thumb [49]. Without prior knowledge, the rule of thumb is the only usable criterion. To get a better understanding of the effect of the window parameter, let us also use two other values 3% and 6%. The average performances of the time series

distance measures combined with the PU methods is presented in Table 4. The results with feature selection methods are presented in Table 5.

Table 4: Auslan average F_1-scores without feature selection

	RW	GA-1	GA-2	GA-3	GA-4	GA-5	PIP
ED	24.7	22.3	17.4	10.7	22.3	22.3	**28.7**
DTW-3%	28.8	17.8	18.8	12.2	18.7	17.8	**41.0**
DTW-6%	27.5	17.2	12.1	13.1	17.2	17.2	**41.6**
DTW-10%	27.9	17.0	11.8	12.6	17.0	17.0	**41.5**
PCA-SF	4.1	6.0	4.2	5.2	6.0	6.0	**6.9**

Table 5: Auslan average F_1-scores with DTW-6%

	RW	GA-1	GA-2	GA-3	GA-4	GA-5	PIP
CLeVer-4	**8.4**	5.4	5.3	4.2	5.4	5.4	7.3
CLeVer-12	17.0	11.5	8.4	11.0	12.2	11.5	**23.2**
Hand-picked 12	32.9	28.3	22.7	14.4	29.5	28.3	**39.7**

The CLeVer-hybrid did not perform very well with this data set. It picked features from the non-dominant hand, and that decreased the signal to noise ratio. The feature subset selection with hand-picked features improved the classification with all methods except for the PIP. This indicates that feature selection may be useful but CLeVer may not be the best method.

4.3.2 Signature verification contest

The second data set is the First International Signature Verification Contest (SVC2004) [68]. There were 40 users that gave 20 samples of their signatures on-line on a pen-based input device. In addition, there are 20 skilled forgeries of each users signature. The device recorded the x- and y-positions of the pen and whether the pen was pressed against the device or lifted. For each user, there are 40 time series with three features. In addition, there are two classes, legitimate and forged, both having 20 time series for each user. Here, the theoretical expected F_1-score for random classifier is 50%.

The pen down feature was dropped since it is binary: either down or lifted. No other feature selection was used as there were only two features left. In addition, Z-normalization was used to get the relative positions of the pen. Otherwise the size of handwriting and starting position on the device would affect the classification.

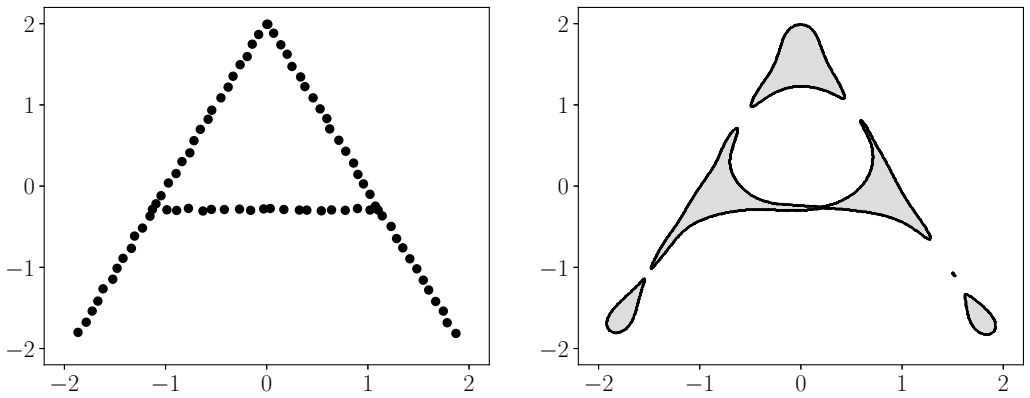

Figure 10: A carefully drawn letter 'A' and its normal operating domain learned by SVDD using RBF-kernel with $\sigma = 0.5$ and $C = 1$.

The parameter for OCC methods are selected similarly to Section 4.3.1. For SVDD a bandwidth parameter $\sigma = \sigma_A$ was used without a multiplier for small dataset. This is because of the nature of signatures as they are usually rather complex. A small value of bandwidth parameter usually results in over-fitting but here it is used it to capture the complex shape of a signature. An example of the letter 'A' analysed with SVDD is presented in Figure 10. For LOF a contamination of 0% and neighborhood size 3 are used. The average F_1-score for SVDD is 62.0% and the average F_1-score for LOF is 57.5%.

The cubic spline resampling was used to resample the time series into average lengths. The parameters for distance ED, DTW and PCA-SF and PU methods RW, CBD-GA and peak evaluation are selected exactly as in Section 4.3.1. The average results for time series distances and PU methods are presented in Table 6. The best combination is PIP with DTW-6% achiev-

Table 6: Signature verification average F_1-scores

	RW	GA-1	GA-2	GA-3	GA-4	GA-5	PIP
ED	26.3	53.4	53.7	50.6	52.2	53.4	**65.0**
DTW-3%	17.7	75.6	69.9	67.4	73.6	**76.0**	75.7
DTW-6%	19.0	71.3	73.3	66.4	71.3	71.9	**84.4**
DTW-10%	16.4	69.9	75.9	66.9	71.7	70.6	**83.4**
PCA-SF	12.8	47.6	47.6	47.6	47.6	47.6	**64.9**

ing the average F_1-score of 84.4%. The RW method performed very badly. This might be, because the classes 'legitimate' and 'forged' are balanced and

there are only twenty examples in each class. The SCC in Equation (3.34), which the RW method uses, applies linear weight to the stopping criterion stressing the earliest observations. However, it is known that there must be many examples to estimate standard deviation accurately. These properties together weaken the classification greatly. With this dataset, even the one-class approaches LOF and SVDD clearly beat the RW method.

4.3.3 Rocks

In the Rocks dataset, minerals are explored using active hyperspectral detection. In the detection scheme, the target is illuminated using near infrared laser, and the spectrum of the reflection is measured. Each spectrum is then analysed as a UTS. [37]

The data are a stream of measured spectra. The data consist of measurements from six different minerals and some background when switching from a mineral to the other. Averaging is used to reduce noise. The stream is segmented into two second intervals. The spectra in each segment is then averaged into one spectrum. Some segments had measurements from both mineral and background. These segments are included in the data to enforce the unsupervised setting and they are labelled as background. The time series are also normalized so the background lighting would not affect the classification.

On average, there are 33 examples in each of the six classes. The background class is the seventh class, but it will not be analysed. Its purpose is to always be in the negative class. In total, there are 299 data points, some of which are background. In this setting, each measurement is the positive example in turn, and the performances are averaged over the classes. Z-normalization is used with this dataset too since, for example, background lighting, distance to the material and humidity may affect the measurements.

By a visual examination of the data, different classes behave a little differently. Classes 2 and 6 form quite clear clusters separated from other classes. All rocks except for the rock 3 forms rather dense classes. The background class is really noisy occasionally generating instances similar to the examples in the rock classes.

The results for Euclidean distance are presented in Table 7. The easiest classes to recognise are the classes 2 and 6. They satisfy the assumption of low density separation. Also, all the methods except CBD-GA-3 benefit from the larger variance of the class 3.

The PIP method does not work very well. This is because of the really noisy background class. The PIP points try to model the noise in the background too accurately and the method does not notice the subtle change

Table 7: Rocks ED average F_1-scores

	RW	GA-1	GA-2	GA-3	GA-4	GA-5	PIP
1	12.2	**31.7**	30.6	11.7	30.2	**31.7**	31.0
2	73.0	27.7	30.0	**98.5**	30.0	27.7	31.0
3	58.5	**75.1**	75.0	25.8	75.0	**75.1**	61.9
4	27.4	27.9	28.7	**45.5**	28.3	27.9	29.9
5	11.3	23.4	26.5	15.4	24.9	23.4	**27.5**
6	**94.6**	29.9	32.3	79.6	32.3	29.9	33.3

between the rock classes. The RW method works well. It is the only method that can really adapt to that background noise because of the running standard deviation in the stopping criterion confidence terms. The *MinDist*-series has the tendency to push the outlying observations, in this case the background measurements, to the end of the *MinDist*-series. This means that RW method penalizes the background twice: first by linear weight, then by compensating for larger variance.

The performance of the DTW-3% setting is presented in Table 8. The

Table 8: Rocks DTW-3% average F_1-scores

	RW	GA-1	GA-2	GA-3	GA-4	GA-5	PIP
1	11.4	**33.6**	30.5	25.3	**33.6**	**33.6**	**33.6**
2	55.2	29.5	25.3	**60.0**	29.5	29.5	29.5
3	58.8	74.8	**76.5**	74.9	74.8	74.8	68.4
4	23.2	27.5	26.0	**47.2**	27.5	27.5	41.9
5	8.7	**26.6**	24.6	19.1	**26.6**	**26.6**	24.9
6	**75.7**	31.7	27.3	63.7	31.7	31.7	32.2

DTW lets the wavelengths mix. This clearly lowers the performance in the classes 2 and 6 that were easy to recognise with Euclidean distance. However, mixing of the wavelengths improves the F_1-scores in the cases that had low scores when using Euclidean distance. This indicates that the use of DTW only adds noise to the rocks dataset. Similar results can be seen with the other window lengths of the Sakoe-Chiba band. Those results are presented in Tables 9 and 10.

Table 9: Rocks DTW-6% average F_1-scores

	RW	GA-1	GA-2	GA-3	GA-4	GA-5	PIP
1	23.7	37.1	30.3	**40.8**	27.8	37.1	38.6
2	55.2	30.6	25.1	**60.0**	25.1	30.6	46.6
3	65.1	75.1	76.5	30.4	**75.7**	75.1	68.1
4	31.0	50.1	25.8	**51.4**	25.2	50.1	44.6
5	13.3	22.1	24.4	22.9	23.0	22.1	**25.5**
6	**73.8**	32.9	28.1	63.7	28.1	32.9	47.4

Table 10: Rocks DTW-10% average F_1-scores

	RW	GA-1	GA-2	GA-3	GA-4	GA-5	PIP
1	15.6	37.2	30.7	**39.6**	37.2	37.2	35.3
2	55.2	27.5	**60.0**	**60.0**	27.5	27.5	35.4
3	72.9	74.8	**76.5**	29.8	74.8	74.8	68.4
4	23.2	48.3	24.8	**51.1**	48.3	48.3	41.1
5	10.6	23.8	22.2	24.5	23.8	23.8	**25.7**
6	**72.9**	29.6	45.6	63.7	30.6	29.6	34.4

5 Conclusion

In this thesis, I studied mathematical optimization in machine learning with two major applications: support vector data description and dynamic time warping. The support vector data description is inspired by the success of support vector machines. It is an anomaly detection method that has been successfully applied to digit recognition and pattern denoising among other domains [11]. The dynamic time warping is a distance measure for time series that can overlook the local invariance in the time series. This property is invaluable when comparing data in domains such as spoken word or handwritten documents since the speech rate and handwriting differ from person to person.

Both of these methods rest on well-researched mathematical foundation. In this thesis, I derived these two algorithms showing their mathematical basis and proving the underlying theorems. This work proves that the algorithms are correct and the results they give are accurate. In addition, this thesis presented a novel self-training method for time series called peak evaluation using perceptually important points.

I applied both of these models, among some other models, to the positive-unlabelled scenario. The positive-unlabelled learning is designed to enhance

the classification in the cases where the annotated data is hard to come by. In many real life situations, labelling the data may be expensive, prone to errors or downright impossible.

In Section 4, it was shown that temporal models using time series distance measures and PU-learning have an upper hand compared to non-temporal OCC models in semi-supervised time series classification. The peak evaluation method showed great performance compared to other self-training methods in the literature on two datasets. The lower performance on the third dataset was explained. I showed that multivariate time series feature subset selection can improve classification accuracy in semi-supervise learning setting even though the unsupervised method CLeVer did not excel.

6 Future Work

There seems to be no reliable unsupervised method for multivariate time series feature subset selection in the literature. The results in Table 5 show that feature subset selection could improve classification accuracy.

The field of time series distance measures have not seen major improvements since the dynamic time warping was discovered in the 1970. The time warp edit distance [38] sounds promising since it allows elasticity, similarly to dynamic time warping, but is also metric, similarly to Euclidean distance.

Self-training of time series is heavily dependent on the $MinDist$-series. As is evident from Figure 7, dividing $MinDist$-series into positive and negative classes depends on how well the $MinDist$-series and the metric used can represent these classes. Some other methods such as amending self-learners [63] could possibly increase the accuracy.

References

[1] Charu C. Aggarwal and Saket Sathe. *Which Outlier Detection Algorithm Should I Use?*, pages 207–274. Springer International Publishing, Cham, 2017.

[2] Andreas Antoniou and Wu-Sheng Lu. *Practical Optimization: Algorithms and Engineering Applications.* Springer Publishing Company, Incorporated, 1st edition, 2007.

[3] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. The great time series classification bake off: A review and experimental evaluation of recent algorithmic advances. *Data Min. Knowl. Discov.*, 31(3):606–660, May 2017.

[4] Gustavo E. A. P. A. Batista, Eamonn J. Keogh, Oben Moses Tataw, and Vinícius M. A. de Souza. Cid: an efficient complexity-invariant distance for time series. *Data Mining and Knowledge Discovery*, 28(3):634–669, May 2014.

[5] Richard Bellman. *Dynamic Programming.* Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

[6] Dimitri P Bertsekas. *Dynamic programming and optimal control.* Belmont, Mass. : Athena Scientific, 3rd ed edition, 2005. Previous edition: 2000.

[7] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* Cambridge University Press, New York, NY, USA, 2004.

[8] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: Identifying density-based local outliers. *SIGMOD Rec.*, 29(2):93–104, May 2000.

[9] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, July 2009.

[10] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[11] Wei-Cheng Chang. A revisit to support vector data description (svdd). 2013.

[12] Olivier Chapelle, Bernhard Schlkopf, and Alexander Zien. *Semi-Supervised Learning*. The MIT Press, 1st edition, 2010.

[13] A. Chaudhuri, D. Kakde, C. Sadek, L. Gonzalez, and S. Kong. The mean and median criteria for kernel bandwidth selection for support vector data description. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 842–849, Nov 2017.

[14] Yanping Chen, Bing Hu, Eamonn J. Keogh, and Gustavo E. A. P. A. Batista. Dtw-d: time series semi-supervised learning from a single example. In Rayid Ghani, Ted E. Senator, Paul Bradley, Rajesh Parekh, and Jingrui He, editors, *KDD*, New York, NY, USA, 2013. ACM.

[15] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, Sep 1995.

[16] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[17] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn Keogh. Querying and mining of time series data: Experimental comparison of representations and distance measures. *Proc. VLDB Endow.*, 1(2):1542–1552, August 2008.

[18] Yifei Ding and Eamonn Keogh. Query suggestion to allow intuitive interactive search in multidimensional time series. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management*, SSDBM '17, pages 18:1–18:11, New York, NY, USA, 2017. ACM.

[19] Philippe Esling and Carlos Agon. Time-series data mining. *ACM Comput. Surv.*, 45(1):12:1–12:34, December 2012.

[20] E. Ferretti, M. L. Errecalde, M. Anderka, and B. Stein. On the use of reliable-negatives selection strategies in the pu learning approach for quality flaws prediction in wikipedia. In *2014 25th International Workshop on Database and Expert Systems Applications*, pages 211–215, Sept 2014.

[21] Mabel González, Christoph Bergmeir, Isaac Triguero, Yanet Rodríguez, and José M Benítez. On the stopping criteria for k-nearest neighbor in positive unlabeled time series classification problems. *Information Sciences*, 328:42 – 59, 2016.

[22] Min Han and Xiaoxin Liu. Feature selection techniques with class separability for multivariate time series. *Neurocomputing*, 110:29 – 34, 2013.

[23] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[24] Bing Hu, Yanping Chen, and Eamonn Keogh. *Time Series Classification under More Realistic Assumptions*, pages 578–586. Society for Industrial and Applied Mathematics, 2013.

[25] Young-Seon Jeong, Myong K. Jeong, and Olufemi A. Omitaomu. Weighted dynamic time warping for time series classification. *Pattern Recognition*, 44(9):2231 – 2240, 2011. Computer Analysis of Images and Patterns.

[26] Michael C. Johannesmeyer, Ashish Singhal, and Dale E. Seborg. Pattern matching in historical data. *AIChE Journal*, 48(9):2022–2038, 2002.

[27] Mohammed Waleed Kadous. *Temporal Classification: Extending the Classification Paradigm to Multivariate Time Series*. PhD thesis, New South Wales, Australia, Australia, 2002. AAI0806481.

[28] Eamonn Keogh and Shruti Kasetty. On the need for time series data mining benchmarks: A survey and empirical demonstration. *Data Mining and Knowledge Discovery*, 7(4):349–371, Oct 2003.

[29] Eamonn Keogh and Chotirat Ann Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and Information Systems*, 7(3):358–386, Mar 2005.

[30] David J. Ketchen and Christopher L. Shook. The application of cluster analysis in strategic management research: An analysis and critique. *Strategic Management Journal*, 17(6):441–458, 1996.

[31] Shehroz S. Khan and Michael G. Madden. A survey of recent trends in one class classification. In Lorcan Coyle and Jill Freyne, editors, *Artificial Intelligence and Cognitive Science*, pages 188–197, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[32] Serkan Kiranyaz, Moncef Gabbouj, and Turker Ince. *Multidimensional Particle Swarm Optimization for Machine Learning and Pattern Recognition*. Adaptation, Learning, and Optimization. Springer, 2014. Contribution: organisation=sgn,FACT1=1
Portfolio EDEND: 2013-07-29.

[33] W. J. Krzanowski. Between-groups comparison of principal components. *Journal of the American Statistical Association*, 74(367):703–707, 1979.

[34] H. W. Kuhn and A. W. Tucker. Nonlinear programming. In *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492, Berkeley, Calif., 1951. University of California Press.

[35] H. Lei and B. Sun. A study on the dynamic time warping in kernel machines. In *2007 Third International IEEE Conference on Signal-Image Technologies and Internet-Based System*, pages 839–845, Dec 2007.

[36] Stephan Liwicki, Georgios Tzimiropoulos, Stefanos Zafeiriou, and Maja Pantic. Euler principal component analysis. *International Journal of Computer Vision*, 101(3):498–518, Feb 2013.

[37] Albert Manninen, Teemu Kääriäinen, Tomi Parviainen, Scott Buchter, Miika Heiliö, and Toni Laurila. Long distance active hyperspectral sensing using high-power near-infrared supercontinuum light source. *Opt. Express*, 22(6):7172–7177, Mar 2014.

[38] P. F. Marteau. Time warp edit distance with stiffness adjustment for time series matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(2):306–318, Feb 2009.

[39] A. Mojsilović. Perceptual indexing of multivariate time series. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*, volume 2, pages II–533–II–536, April 2007.

[40] Minh Nhut Nguyen, Xiao-Li Li, and See-Kiong Ng. Positive unlabeled learning for time series classification. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, IJCAI'11, pages 1421–1426. AAAI Press, 2011.

[41] Minh Nhut Nguyen, Xiao-Li Li, and See-Kiong Ng. Ensemble based positive unlabeled learning for time series classification. In Sang-goo Lee, Zhiyong Peng, Xiaofang Zhou, Yang-Sae Moon, Rainer Unland, and Jaesoo Yoo, editors, *Database Systems for Advanced Applications*, pages 243–257, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[42] S. Ntalampiras, I. Potamitis, and N. Fakotakis. Probabilistic novelty detection for acoustic surveillance under real-world conditions. *IEEE Transactions on Multimedia*, 13(4):713–719, Aug 2011.

[43] Pathima Nusrath Hameed, Karin Verspoor, Snezana Kusljic, and Saman Halgamuge. Positive-unlabeled learning for inferring drug interactions based on heterogeneous attributes. *BMC Bioinformatics*, 18:1 – 15, 2017.

[44] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[45] Marco A.F. Pimentel, David A. Clifton, Lei Clifton, and Lionel Tarassenko. A review of novelty detection. *Signal Processing*, 99:215 – 249, 2014.

[46] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, April 1998.

[47] Kevin B. Pratt and Eugene Fink. Search for patterns in compressed time series. *International Journal of Image and Graphics*, 02(01):89–106, 2002.

[48] Chotirat Ann Ratanamahatana and Eamonn Keogh. Everything you know about dynamic time warping is wrong. In *Third Workshop on Mining Temporal and Sequential Data*. Citeseer, 2004.

[49] Chotirat Ann Ratanamahatana and Dechawut Wanichsan. *Stopping Criterion Selection for Efficient Semi-supervised Time Series Classification*, pages 1–14. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[50] R. Tyrrell Rockafellar. Lagrange multipliers and optimality. *SIAM Rev.*, 35(2):183–238, June 1993.

[51] J. K. Rout, A. Dalmia, K. K. R. Choo, S. Bakshi, and S. K. Jena. Revisiting semi-supervised learning for online deceptive review detection. *IEEE Access*, 5:1319–1327, 2017.

[52] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49, Feb 1978.

[53] Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[54] Mohammad Shokoohi-Yekta, Bing Hu, Hongxia Jin, Jun Wang, and Eamonn Keogh. Generalizing dtw to the multi-dimensional case requires an adaptive approach. *Data Mining and Knowledge Discovery*, 31(1):1–31, Jan 2017.

[55] A. Singhal and D. E. Seborg. Pattern matching in historical batch data using pca. *IEEE Control Systems*, 22(5):53–63, Oct 2002.

[56] Ashish Singhal and Dale E. Seborg. clustering multivariate time-series data. *Journal of Chemometrics*, 19(8):427–438, 2006.

[57] Morton Slater. Lagrange multipliers revisited. Cowles Foundation Discussion Papers 80, Cowles Foundation for Research in Economics, Yale University, 1959.

[58] Romain Tavenard. tslearn: A machine learning toolkit dedicated to time-series data, 2017. `https://github.com/rtavenar/tslearn`.

[59] David Martinus Johannes Tax. *One-class classification: Concept learning in the absence of counter-examples*. PhD thesis, Technische Universiteit Delft, 2001.

[60] David M.J. Tax and Robert P.W. Duin. Support vector data description. *Machine Learning*, 54(1):45–66, Jan 2004.

[61] B. B. Thompson, R. J. Marks, J. J. Choi, M. A. El-Sharkawi, Ming-Yuh Huang, and C. Bunje. Implicit learning in autoencoder novelty assessment. In *Neural Networks, 2002. IJCNN '02. Proceedings of the 2002 International Joint Conference on*, volume 3, pages 2878–2883, 2002.

[62] Michael E. Tipping and Christopher M. Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(3):611–622, 1 1999.

[63] Isaac Triguero, Salvador García, and Francisco Herrera. Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study. *Knowledge and Information Systems*, 42(2):245–284, Feb 2015.

[64] Santiago D. Villalba and Pádraig Cunningham. An evaluation of dimension reduction techniques for one-class classification. *Artificial Intelligence Review*, 27(4):273–294, Apr 2007.

[65] Qiang Wang and Vasileios Megalooikonomou. A dimensionality reduction technique for efficient time series similarity analysis. *Information systems*, 33(1):112–132, 2008.

[66] T. Warren Liao. Clustering of time series data-a survey. *Pattern Recogn.*, 38(11):1857–1874, November 2005.

[67] Li Wei and Eamonn Keogh. Semi-supervised time series classification. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 748–753, New York, NY, USA, 2006. ACM.

[68] Dit-Yan Yeung, Hong Chang, Yimin Xiong, Susan George, Ramanujan Kashi, Takashi Matsumoto, and Gerhard Rigoll. Svc2004: First international signature verification competition. In David Zhang and Anil K. Jain, editors, *Biometric Authentication*, pages 16–22, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[69] H. Yoon, K. Yang, and C. Shahabi. Feature subset selection and feature ranking for multivariate time series. *IEEE Transactions on Knowledge and Data Engineering*, 17(9):1186–1198, Sept 2005.

[70] Yang Zhao, Shengwei Wang, and Fu Xiao. Pattern recognition-based chillers fault detection method using support vector data description (svdd). *Applied Energy*, 112:1041 – 1048, 2013.

[71] Seyedjamal Zolhavarieh, Saeed Aghabozorgi, and Ying Wah Teh. A Review of Subsequence Time Series Clustering. *The Scientific World Journal*, 2014:19, 2014.