UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

RENAN FONTELES ALBUQUERQUE

A NOVEL ADAPTIVE LEARNING VECTOR QUANTIZATION FOR TIME SERIES CLASSIFICATION

FORTALEZA

2018

RENAN FONTELES ALBUQUERQUE

A NOVEL ADAPTIVE LEARNING VECTOR QUANTIZATION FOR TIME SERIES CLASSIFICATION

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do título de Mestre em Engenharia Elétrica. Área de Concentração: Automação e Controle.

Orientador: Prof. Dr. Arthur Plínio de Souza Braga

Coorientador: Prof. Dr. Bismark Claure Torrico

FORTALEZA

2018

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

F762n Fonteles Albuquerque, Renan.
A Novel Adaptive Learning Vector Quantization for Time Series Classification / Renan Fonteles Albuquerque. – 2018.
146 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Tecnologia, Programa de Pós-Graduação em Engenharia Elétrica, Fortaleza, 2018.
Orientação: Prof. Dr. Arthur Plínio de Souza Braga.
Coorientação: Prof. Dr. Bismark Claure Torrico.

1. Reconhecimento de padrões. 2. Redes Neurais Artificiais. 3. Aprendizado Adaptativo. 4. Learning Vector Quantization. 5. Classificação de Séries Temporais. I. Título.

CDD 621.3

RENAN FONTELES ALBUQUERQUE

A NOVEL ADAPTIVE LEARNING VECTOR QUANTIZATION FOR TIME SERIES CLASSIFICATION

Master's dissertation presented to the Post Graduate Program in Electrical Engineering of the Federal University of Ceará as partial requirement for the degree of Master in Electrical Engineering. Concentration area: Control and Automation.

Approved in September 21st, 2018

EXAMINATION BOARD

Prof. Dr. Arthur Plínio de Souza Braga (Advisor)
Federal University of Ceará (UFC)

Prof. Dr. Bismark Claure Torrico (Co-advisor)
Federal University of Ceará (UFC)

Prof. Dr. Wilkley Bezerra Correia
Federal University of Ceará (UFC)

Prof. Dr. Ajalmar Rego da Rocha Neto
Federal Institute of Education, Science and Technology of Ceará (IFCE)

To my beloved family.

ACKNOWLEDGEMENTS

I would like to dedicate this thesis to my family, Cícero Raimundo Matos Albuquerque, Jacqueline Rios Fonteles Albuquerque and Camila Fonteles Albuquerque, for their support, encouragement and love.

I thank my supervisor, Prof. Dr. Arthur Plínio Souza Braga, for his guidance and partnership during the Master Course. I also thank my friends in the GPAR and PPGEE, for all conversations, discussions and leisure moments. I specially express my gratitude to Paulo Daving and Magno Prudêncio for all the support and friendship since the beginning of this journey. I thank in general all the engineers, researchers and professors who had the opportunity to evaluate and contribute with this work.

Finally, I would like to acknowledge the financial support of *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior* (CAPES) for my Master Scholarship.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

“The profound study of nature is the most fertile source of mathematical discoveries.”

(Joseph Fourier)

RESUMO

A Classificação de Séries Temporais é um problema de interesse em diversas áreas de pesquisa, contendo aplicações interessantes para o uso de técnicas de Aprendizado de Máquina. Dentre as soluções adotadas na literatura, os algoritmos baseados em Redes Neurais Artificiais (RNA) têm se destacado devido à sua capacidade de generalização. Nesta dissertação foi realizado um estudo sobre o desempenho das redes neurais no problema de classificação de séries temporais. É proposta uma nova abordagem adaptativa para a rede neural *Learning Vector Quantization* (LVQ) combinada com um método de agrupamento conhecido como *Self-Organizing Map* (SOM). O classificador proposto, denominado Adaptive-LVQ-SOM (ALVQ-SOM), permite a remoção e inclusão de protótipos com o objetivo de otimizar o desempenho de classificação da rede. Outras duas variações inspiradas no método ALVQ-SOM também são apresentadas: Driven-LVQ (dLVQ) e Driven-ALVQ-SOM (dALVQ). Para avaliar a eficácia do método proposto, um estudo comparativo foi conduzido entre os classificadores LVQ clássicos, o ALVQ-SOM e outros dois classificadores baseados em RNA: *Multi-Layer Perceptron* (MLP) e *Support Vector Machine* (SVM). Além disso, o algoritmo *K -Nearest Neighbours* (*k*-NN) foi inserido neste estudo pois este é considerado um algoritmo de referência na literatura de classificação de séries temporais. A metodologia adotada na avaliação dos algoritmos consiste na aplicação da técnica de validação cruzada 10-*Fold* na execução de simulações utilizando os diversos classificadores estudados, aplicados a conjuntos de dados distintos. Os resultados dos experimentos mostram que o método de LVQ adaptativo proposto (ALVQ-SOM) supera as versões clássicas do LVQ, apresentando desempenho de classificação superior na maioria dos cenários estudados.

Palavras-chave: Reconhecimento de padrões; Redes Neurais Artificiais, Aprendizado Adaptativo; Multi-Layer Perceptron; Support Vector Machine; Learning Vector Quantization;K-Nearest Neighbor; Classificação de Séries Temporais; Mineração de Séries Temporais

ABSTRACT

Time series classification is a problem of interest in several areas of research, containing interesting applications for the use of machine learning techniques. Among the solutions adopted in the literature, the algorithms based on Artificial Neural Network (ANN) have been outstanding due to their generalization capacity. In this dissertation, a study was conducted on the performance of neural networks in the problem of time series classification. A new adaptive variation of the Learning Vector Quantization (LVQ) neural network, combined with a clustering method known as Self-Organizing Map (SOM), has been proposed. The proposed classifier, called Adaptive-LVQ-SOM (ALVQ-SOM), allows the removal and inclusion of prototypes in order to optimize the classification performance of the network. Two other methods inspired by ALVQ-SOM are also presented: Driven-LVQ (dLVQ) and Driven-ALVQ-SOM (dALVQ). To evaluate the efficacy of the proposed method, a comparative study was conducted between the classical LVQ classifiers, ALVQ-SOM and two other ANN-based classifiers: Multi-Layer Perceptron (MLP) and Support Vector Machine (SVM). In addition, the algorithm K - Nearest Neighbors (k -NN) was inserted in this study, since this algorithm is considered a reference classifier in the literature of time series classification. The methodology adopted in the evaluation of the algorithms consists in the application of the cross-validation technique 10-fold in the execution of simulations using different classifiers, applied to distinct datasets. The results of the experiments show that the proposed adaptive LVQ (ALVQ-SOM) method outperforms the classical versions of LVQ, presenting superior classification performance in most of the studied scenarios.

Keywords: Pattern Recognition; Artificial Neural Networks; Adaptive learning; Multi-Layer Perceptron; Support Vector Machine; Learning Vector Quantization; K-Nearest Neighbor; Time Series Classification; Time Series Data Mining

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS AND ACRONYMS

Q_E	Quantization Error
k-NN	k-Nearest Neighbor
ALVQ	Adaptive LVQ
ALVQ-GV	Grbovic-Vucetic Adaptive LVQ
ALVQ-SOM	Adaptive LVQ with Self-Organizing Maps
ANN	Artificial Neural Network
CM	Confusion Matrix
CNN	Convolution Neural Networks
dALVQ	driven Adaptive LVQ with Self-Organizing Maps
dLVQ	Driven-LVQ
DNN	Deep Neural Network
DTW	Dynamic Time Warping
ED	Euclidean Distance
HAR	Human Activity Recognition
KM-SOM	Kohonen's Map
LVQ	Learning Vector Quantization
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
SOM	Self-Organizing Maps
SVM	Support Vector Machine
TS	Time Series
TSC	Time Series Classification
TSDM	Time Series Data Mining

LIST OF SYMBOLS

a_f	Adaptive factor
g_f	Growth factor
r_t	Removal threshold
B	Budget
α	Learning rate
α_0	Initial value of α
δ	Local gradient
$\mathbf{x}$	classifier's input vector
y	desired class of of the input vector $\mathbf{x}$
$\mathbf{w}$	Vector of neural weights
$\mathbf{W}$	Matrix of neural weights
$\mathbf{x}_{min}$	Vector of the lowest values for the attributes of an input vetor $\mathbf{x}$
$\mathbf{x}_{max}$	Vector of the highest values for the attributes of an input vetor $\mathbf{x}$
σ	Size of the neighborhood of the SOM neural network
σ_0	Initial value of σ

CONTENTS

1 INTRODUCTION

Time series are collections of sequential observations that describe the process of temporal evolution of a given variable. In general, these type of data is composed of observations from a measurable phenomenon that varies over time. Time series are present in several areas of knowledge, such as medicine, robotics, finances, and meteorology. Time series analysis is the field which studies how to extract meaningful information from time series in order to solve problems such as classification, regression, and prediction. Multiple papers in the literature have explored different strategies for analysing continuous temporal data extracted from a diversity of data sources. In general, these temporal data are collected from sensors which transforms the observed physical, chemical or biological phenomena into electrical measure (voltage or current) which can be represented in numerical values. Examples of time series data sources widely explored in the literature are: accelerometers (ANGUITA *et al.*, 2013), Electrocardiogram (ECG) (NOVIYANTO; ARYMURTHY, 2012; RAJESH; DHULI, 2017), Electroencephalogram (EEG) (HAJINOROOZI *et al.*, 2016; SORS *et al.*, 2018; JIAO *et al.*, 2018).

Time Series Classification (TSC) is a specific classification problem which involves finding a mapping function capable of assigning a time series data into a specific discrete class label. The data sequence property is essential for featuring the problem, as the pattern recognition depends on the samples sequence. A significantly number of time series classification algorithms have been explored in the literature recently (BAGNALL *et al.*, 2017).

For instance, Smith *et al.* (2017) have presented a study about the deployment of Artificial Neural Network (ANN) and statistics methods for classification of transmission signal in satellites communication. In Taqi *et al.* (2017) and Kumar *et al.* (2016) the authors have proposed classification models applied to EEG signals using Deep Neural Network (DNN), as in Chavan & Kolte (2017) they used Multi-Layer Perceptron (MLP). In Jen *et al.* (2008) is presented a classification method of ECG signals using linear feature analysis. An interesting area of research that has been widely discussed in recent years is the recognition of human activities from accelerometer data. Currently, Human Activity Recognition (HAR) has been shown to be a relevant topic in the field of time series classification (BAO; INTILLE, 2004; LESTER *et al.*, 2006; MANNINI; SABATINI, 2010; ATALLAH *et al.*, 2011; CASALE *et al.*, 2011; AYU *et al.*, 2016; BUENAVENTURA; TIGLAO, 2017).

As time series data can have complex characteristics, it is necessary to apply solutions that can handle the nonlinear operations. There are numerous classifiers that rely on different

approaches to create an appropriate classification model for a given problem. The state of the art in the area of pattern recognition has explored several algorithms for solving classification problems: Decisions Trees (MURTHY, 1998), Artificial Neural Networks (ZHANG, 2000), Bayesian Networks (JENSEN, 1996), Support Vector Machines (BURGES, 1998) and Instance-based learning classifiers (AHA, 1997; MANTARAS; ARMENGOL, 1998). A complete review of classification algorithms can be found in (KOTSIANTIS *et al.*, 2006). Among the pattern recognition methods, ANN-based algorithms become an interesting solution to address this problem due to its generalization ability (HAYKIN *et al.*, 2009).

Considering the efficacy of models based on neural networks in the solution of complex classification problems, this work seeks to carry out a comparative and evaluative study about such classifiers applied in the problem of classification of time series. This work explores in the literature multiple ANN-based solutions for comparing their performance in classifying time series. A benchmark using different data sets was built to analyze and compare several ANN strategies. Furthermore, a novel adaptive algorithm based on Learning Vector Quantization (LVQ) is proposed in order to improve the classification performance in time series problems.

1.1 Motivation

Applications in the area of time series analysis have been extensively explored in recent years due to the numerous problems involving temporal data. More specifically, in TSC problems, the construction of decision boundaries for separating data from different classes is a challenging task that usually requires machine learning and computational intelligence techniques. In order to build a classifier, an appropriate decision boundary must be defined. In general, classification performance (i.e. accuracy) is the criterion used for choosing the optimal model. Nevertheless, a diversity of criteria may be considered for selecting a classifier for a determined application (e.g., computational cost and implementation complexity). Based on Occam's razor, the simplest classifier would be considered the "best" model (BUNTINE, 1990). Usually, the simplest classifier is not the most successful in terms of accuracy performance, but it may present other characteristics which could be more desirable. Hence, it is often preferable to adopt simpler classification models that present a performance that suffice the application requirements. In summary, it is necessary to find a compromise between performance and simplicity of the designed model.

Time series are often high-dimensional data. Hence, depending on the volume of data available for training a classifier, the computational cost may increase significantly, turning complex models quite disadvantageous and sometimes unfeasible. Recent researches have shown frequent use of techniques based on Artificial Neural Networks (ANNs) such as Multi-Layer Perceptron (MLP) (HAYKIN *et al.*, 2009) , Deep Neural Network (DNN) (WANG *et al.*, 2018) and Convolution Neural Networks (CNN) (GU *et al.*, 2018) to solve TSC problems. Although ANN-based techniques have been employed successfully in a diversity of TSC problems, there are still issues to be addressed. For example, classifiers based on DNN and CNN are computationally expensive. Therefore, when designing a classifier, it is interesting to carry out model evaluations to determine if such techniques are indeed necessary to solve the problem, or if a simpler classifier would be more recommended.

In this context, an alternative simpler ANN-based intelligent classification model is the Learning Vector Quantization (LVQ) (KOHONEN, 1990). LVQ is a type of prototype-based model that present lower cost in comparison to networks such as MLP, DNN and CNN. Prototype-based algorithms are considered efficient models that present reduced computational cost compared to models whose training process is based on Backpropagation (i.e., MLP). In addition, LVQ is not inefficient in terms of memory consumption, since only a few prototypes are necessary for building a model.

The algorithms based on LVQ are rarely approached in the literature of TSC problems, and only few LVQ variations depend on adaptive strategies for solving complex classification problems. Motivated by the discrete number of studies in algorithms based on LVQ in the classification of temporal patterns, this work intends to contribute with the literature by proposing a new adaptive method based on LVQ, and comparing this method with other classifiers of different approaches. In this work, a comparative study is conducted on classification algorithms applied in time series data sets of different nature. ANN-based algorithms such as LVQ (classic and adaptive versions), MLP, Support Vector Machine (SVM) (CORTES; VAPNIK, 1995), and an instance-based naive method called k-NN (Nearest Neighbor) (NASIBOV; PEKER, 2011) are explored. In addition, a new approach of Adaptive-LVQ is proposed with the objective to improve the overall classification performance of the model. This master thesis aims to contribute with the state-of-the-art pattern recognition algorithms applied in time-series classification. Details about the general and specific objectives are discussed in the next section.

1.2 Objectives

The main goal of this dissertation is to propose and evaluate a novel adaptive LVQ-based variation in order to improve the classification performance in time series problems. As a general objective, this thesis aims to contribute to the state-of-the-art in time series classification by experimenting commonly used neural network techniques and assessing their effectiveness in classifying time series from different domains. In addition, several ANN-based techniques are compared to a benchmark algorithm called *k*-NN.

The specific objectives of this dissertation are:

1. To briefly review the state-of-art of neural network based algorithms applied in TSC problems.
2. To evaluate the ANN-based classifiers when applied to solve TSC problems.
3. To propose and evaluate a novel adaptive LVQ-based variation.
4. To carry out experiments using several datasets for comparing and evaluating multiple ANN-based techniques, the proposed adaptive-LVQ, and a naive instanced-based method for solving TSC problems.

1.3 Methodology

Initially, it is shown a bibliography review on time series fundamentals, with focus on the state-of-the-art of time series classification problem, covering the main articles of application of neural networks in TSC problem. From the bibliography review, several algorithms are selected for being experimented and studied. Within the selected algorithms, there are three ANN-based algorithms: MLP, LVQ and SVM. Moreover, the *k*-NN algorithm is included in this study due to its importance in time series classification. Furthermore, an adaptive LVQ-based algorithm is designed. For assessing the proposed method, and the other studied classifiers, several experiments are designed for comparing them, considering different aspects of performance. The main performance criterion is the classification accuracy, however, other criteria such as processing and memory cost are considered. Several computational experiments and simulations are performed using Matlab environment. These simulations are performed with different classification algorithms on multiple TSC datasets for comparing their classification performance.

A general performance experiment is conducted on seven datasets from UCR Repository (CHEN *et al.*, 2015). The results of the experiments are analyzed for each data set. Performance indices are calculated with the purpose to compare the classification performance between the experimented algorithms. The accuracy, precision and recall are chosen for better evaluate difference between classification strategies, as these metrics considers true-positive, true-negatives, false-positive and false-negative predictions. In addition, the classification performance distribution is analysed graphically by using box-plots, and the training convergence is assessed through learning curves.

1.4 List of Publications

The researches developed in this work originated the following publications:

- RENAN F. ALBUQUERQUE, ARTHUR P. S. BRAGA, BISMARK C. TORRICO, (2017). **Classificação de Dinâmicas em Sistemas Utilizandos Redes LVQ Adaptativas.** Anais da Conferência Brasileira de Dinâmica, Controle e Aplicações - DINCON 2017. São José do Rio Preto, SP.
- RENAN FONTELES A. , PAULO DAVING, ARTHUR P. S. BRAGA, (2018). **Adaptive Fuzzy Learning Vector Quantization (AFLVQ) for time series classification.** In 37th North-American Fuzzy Information Processing Society (NAFIPS) Annual Conference held jointly with the 5th Brazillian Congress of Fuzzy Systems.

1.5 Organization of the Thesis

The remaining of this thesis is organized as follows:

- In Chapter 2 the fundamentals on time-series are discussed, including time series definition and notation. Furthermore, the main applications of time series data mining are cited, with focus on classification problems. A brief discussion is held about the importance of time series data representation and similarity measures. These topics are considered the main challenges in classification of temporal data. The state-of-the-art of time series classification is reviewed, addressing important works in this area and presenting the most explored algorithms and applications. Finally, the main neural network based methods addressed in the literature are mentioned.
- Chapter 3 briefly introduces the classification methods covered in this thesis. The k-NN algorithm is presented as a general-purpose classifier for comparing with ANN approa-

ches (Section 3.1). In sequence, the ANN-based algorithms are described individually, covering concepts related to each neural network used in the simulations. The ANN-based algorithms Multi-Layer Perceptron (MLP) (Section 3.2), Self-Organizing Maps (SOM) (Section 3.3), LVQ (Section 3.4) and SVM (Section 3.5) are addressed.

- Chapter 4 introduces the proposed LVQ-based algorithms, including their main concepts, time complexity and implementation details. Furthermore, a framework is presented in order to permit the combination of the proposed adaptive modifications to any non-adaptive LVQ-based variation.
- Chapter 5 is focused on describing the performance evaluation metrics and the comparison methodology for evaluating the results among the classifiers studied. Concepts such as cross-validation , confusion matrix, accuracy, precision and recall are covered.
- In Chapter 6, the datasets and the experiments designed are detailed. At the end of this section, the experiment results are presented, analyzed and discussed.
- Finally, Chapter 7 concludes the master thesis with a final discussion about main concepts used in this work and present conclusions reached in previous chapters. In addition, the limitations of the proposed algorithms are analyzed, and possible future works and further research topics are suggested.

Every chapter start with an initial contextualization on the subject that will be discussed in the chapter. Similarly, at the end of the chapter there will be final considerations, which summarizes the subject discussed.

2 TIME SERIES CLASSIFICATION

Time is a variable which measures how long an action, process, condition or event exists or continues (MERRIAM-WEBSTER, 2004). Time is ubiquitous; thus diverse phenomena can be described in the temporal domain. Data sets which have a chronological sequence are called time series. Time Series (TS) is a category of data that is extensively explored in researches on several areas of knowledge (TAQI *et al.*, 2017; CHAVAN; KOLTE, 2017; KUMAR *et al.*, 2016; JEN *et al.*, 2008; MELGANI; BAZI, 2008; ROSA; MATTA, 2000).

A time series data set consists of sequences of numerical values obtained over repeated measurements of time. Typically, each measure is taken at equal time intervals (e.g., every minute, hour or day). Time series databases are popular in many applications such as stock market analysis, economic and sales forecasting, inventory studies, workload projections and process and quality control. They are also useful for studying natural phenomena (e.g., atmosphere, temperature, wind, earthquake), designing scientific and engineering experiments and developing medical treatments(HAN *et al.*, 2011).

In this chapter, the fundamentals of time series are introduced, covering the basic concepts and formal definitions. Moreover, a review of time series data mining is presented, by exploring mainly the state-of-the-art of time series classification (TSC), which is the main subject addressed in this work.

2.1 Fundamentals of Time Series

Time series consist on a set of observed values, ordered chronologically. A time series can also be defined as realization of a stochastic process (BROCKWELL; DAVIS, 2013).

A stochastic process is a collection of random elements $x = \{x_t, t \in T\}$ defined on a probability space, recorded according to the order they are obtained in time (DOUC *et al.*, 2014; CINLAR, 2013; BROCKWELL; DAVIS, 2013).

Therefore, a time series can be defined as a set of quantitative observations, organized in a chronological order where, generally, the time is assumed as a discrete variable (KIRCHGÄSSNER; WOLTERS, 2007). In this work, only discrete time series are considered. Hence, the index t, that represents the time, can only takes value in a set T composed by non-negative integers greater than zero. The set T can be defined as $T = \{1, 2, 3, 4..., n\}$, where n is the number of observations, usually known as the time series length.

2.1.1 Univariate and Multivariate Time Series

In time series analysis, the temporal data can be univariate or multivariate. In univariate time series, only one single observation is recorded sequentially over equal time periods. On the other hand, multivariate time series involves two or more input variables being recorded. In this thesis, dimensionality is considered as the quantity of observed variables in a specific instant.

Multivariate processes are described by several related time series simultaneously observed over time. In this work, multivariate time series and multidimensional time series are considered equivalent.

In high dimensional time series, the observed data (x_t) is an n-dimensional vector, where n represents the data dimensionality. Therefore, an observed data from a n-dimensional time series would be $\mathbf{x}_t = \{x_1, x_2, x_3, ..., x_n\}$

For instance, a univariate time series could be composed by measurements of temperature from a sensor during a certain amount of time. As an example of a multi-dimensional time series, a triaxial accelerometer measures three variables simultaneously (acceleration in x, y and z axis), providing a three-dimensional time series. In Figure 1, a univariate and a three-dimensional multivariate time series are presented. Note that these series were synthetically produced for exemplification.

Formally, univariate and multivariate time series can be defined by the following equations, respectively:

$$\mathcal{S}^{n,\ell} = \{x_t \in \mathbb{R}, n = 1\}_{t=1}^{\ell} \tag{2.1}$$

$$\mathcal{S}^{n,\ell} = \{\mathbf{x}_t \in \mathbb{R}^n, n \geq 2\}_{t=1}^{\ell} \tag{2.2}$$

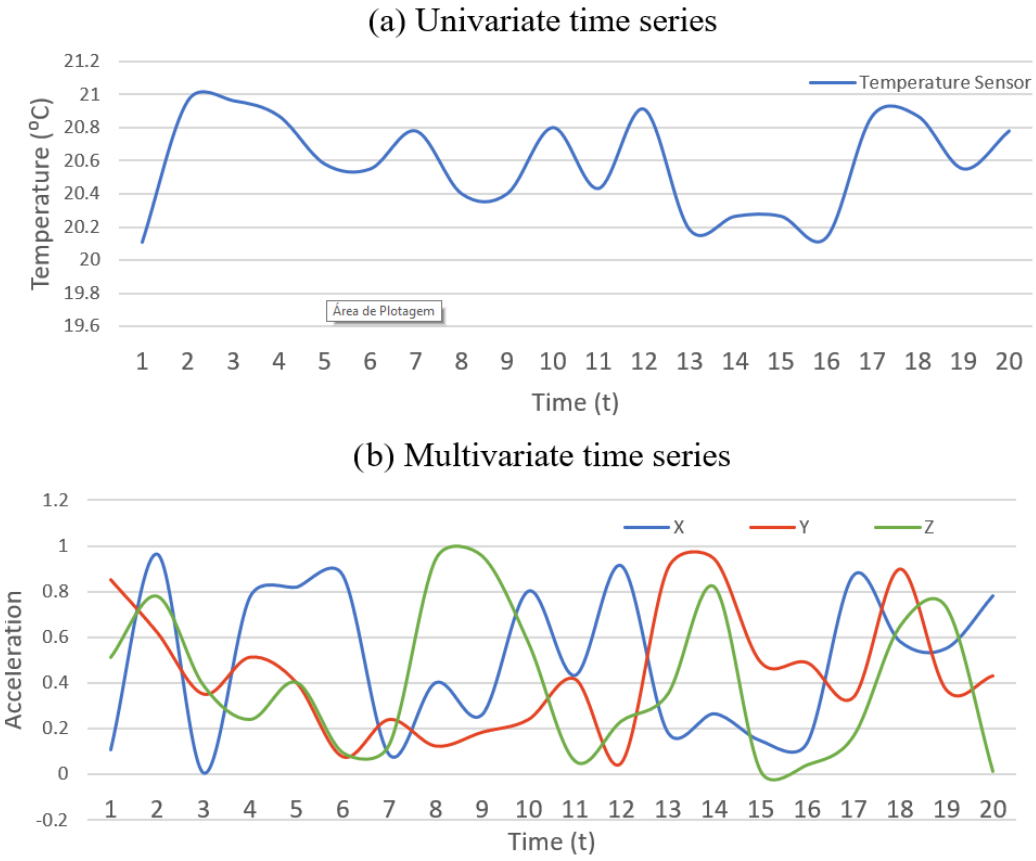
2.1.2 Time Series Definition

A time series is a finite sequence of observations $\mathcal{S} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, ..., \mathbf{x}_\ell\}$, which can be defined by:

$$\mathcal{S} = \{\boldsymbol{x_t}\}_{t=1}^{\ell} \tag{2.3}$$

where $\boldsymbol{x_t}$ is the observation in t that can be a one-dimensional or a multi-dimensional vector, and ℓ is the time series length, which represents the number of observations that compose $\mathcal{S}$.

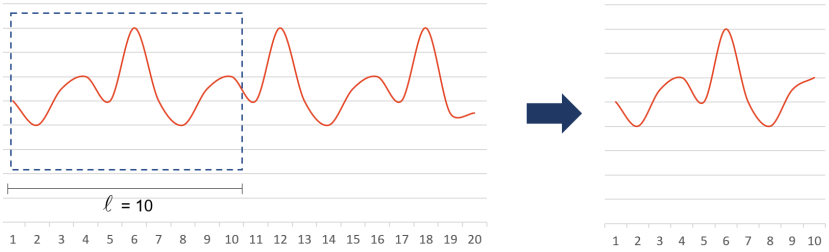
Figure 1 – Illustration of univariate and multivariate time series



Source: The author.

A time series subsequence (also called subseries) is a segment of a time series in a specific interval.A subsequence in an interval $[t_1 : t_2]$ can be defined as $\mathcal{S}[t_1 : t_2] = \{\mathcal{S}[t_1], \mathcal{S}[t_1 + 1], ..., \mathcal{S}[t_2]\}$. Figure 2 exhibits an example of a subsequence with length $\ell = 10$ and interval $[1 : 10]$, then $\mathcal{S}[1 : 10]$.

Figure 2 – Subsequence example for $\ell = 10$



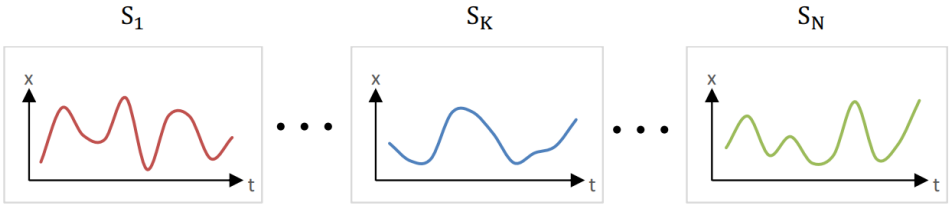
Source: The author.

A time series data set $\mathcal{D}$ consists of a set containing N time series samples $\{\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_N\}$, which can be generalized by the following equation:

$$\mathcal{D} = \{\mathcal{S}_i\}_{i=1}^{N} \tag{2.4}$$

Figure 3 illustrates a univariate time series data set with N samples. In the figure, each data sample is a time series and all time series have the same length (ℓ), for all $\mathcal{S} \in \mathcal{D}$.

Figure 3 – Time series data set with N samples



Source: The author.

2.2 Time Series Data Mining

Time Series Data Mining (TSDM) is the study of techniques, strategies and methods used to extract meaningful knowledge from data in time domain, known as time series (ESLING; AGON, 2012). Temporal data has numerous complex features which reveals several outstanding challenges in data mining techniques. Most challenge derives from the high dimensionality of the data and the difficulty of determining proper similarity measures for comparing different time series (FAKHRAZARI; VAKILZADIAN, 2017; BAGNALL *et al.*, 2017; ESLING; AGON, 2012; RALANAMAHATANA *et al.*, 2005). Among the challenges faced by time series mining algorithms, the problem of similarity measures and data representation are further detailed , due to their importance.

2.2.1 Similarity and Dissimilarity Measures

Similarity and dissimilarity measures are functions that defines the resemblance between two elements. In other words, they provide a way to quantify how similar, or dissimilar, two elements are from each other.

- **Similarity measure:** Functions that compares two objects and returns a value based on their degree of similarity. The closer the objects are to each other, the higher the resulting value of the function.
- **Dissimilarity measure:** Functions that compares two objects and returns a value based on their degree of dissimilarity. The closer the objects are to each other, the lower the resulting value of the function.

Considering $d(a,b)$ a dissimilarity measure between objects a and b, then $d(a,b)$ must satisfy the following restrictions:

- $d(a,b) \geq 0, \forall a,b$
- $d(a,a) = 0, \forall a$
- $d(a,b) = d(b,a), \forall a,b$

Similarity and Dissimilarity measures can be applied in data with different representations, such as vectors, matrices and objects. In this work, algorithms such as KNN, LVQ and SOM are based on dissimilarity measures. Throughout the dissertation, terms such as dissimilarity, proximity, resemblance and distance will be considered equivalent, since the smaller the distance between two elements, the more similar they are to each other.

Multiple data mining algorithms are based on comparing distances between time series. Numerous time series researchers have reviewed the main similarity measures used in the literature (BAGNALL *et al.*, 2017; FAKHRAZARI; VAKILZADIAN, 2017; LIAO, 2005; IGLESIAS; KASTNER, 2013; RALANAMAHATANA *et al.*, 2005). In the next subsections, two frequently used distance metrics are succinctly described: Euclidean Distance and Dynamic Time Warping (DTW).

2.2.1.1 Euclidean Distance (ED)

A typically used distance measure for time series is the Euclidean Distance (ED) (RALANAMAHATANA *et al.*, 2005). ED, also known as L_2-norm, compare the distance between each point from two time series. For two time series (S_1 and S_2) which have the same length ℓ, the ED can be computed by following equation.

$$d_{Euclidean}(\mathcal{S}_1, \mathcal{S}_2) = \|\mathcal{S}_1 - \mathcal{S}_2\|_2 = \sqrt{\sum_{t=1}^{\ell} (\mathcal{S}_1[t] - \mathcal{S}_2[t])^2} \tag{2.5}$$

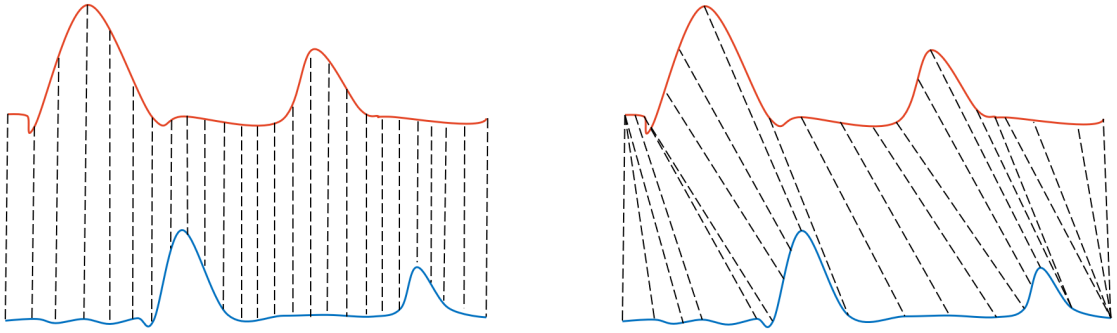
ED is a very simple similarity measure and it is widely used in the literature. However, such a measure is not appropriate for measuring the distance between unaligned time series. The alignment problem typically is solved by Dynamic Time Warping (DTW), introduced in the next subsection.

2.2.1.2 *Dynamic Time Warping (DTW)*

Dynamic Time Warping (DTW) is a distance-based similarity measure used for comparing unaligned two time series. DTW algorithm has been applied initially in speech recognition applications (MÜLLER, 2007; RABINER; JUANG, 1993) and it is strongly explored in time series data mining field.

The DTW approach consists in finding an optimal alignment between two time series, exploiting temporal distortions (warping) between them. In other words, DTW are specifically useful in comparing time series which have similar shapes but poor alignment. Figure 4 shows a comparison between ED and DTW.

Figure 4 – Euclidean distance and dynamic time warping measurements



(a) Euclidean Distance (b) Dynamic Time Warping

Source: The author.

Although they have similar patterns, the Euclidean distance or any other metric that calculates distance sample by sample, will not properly measure their proximity, as these time series are not aligned. In contrast, the DTW technique can find an optimal alignment between these two time series, resulting in a more accurate distance measurement.

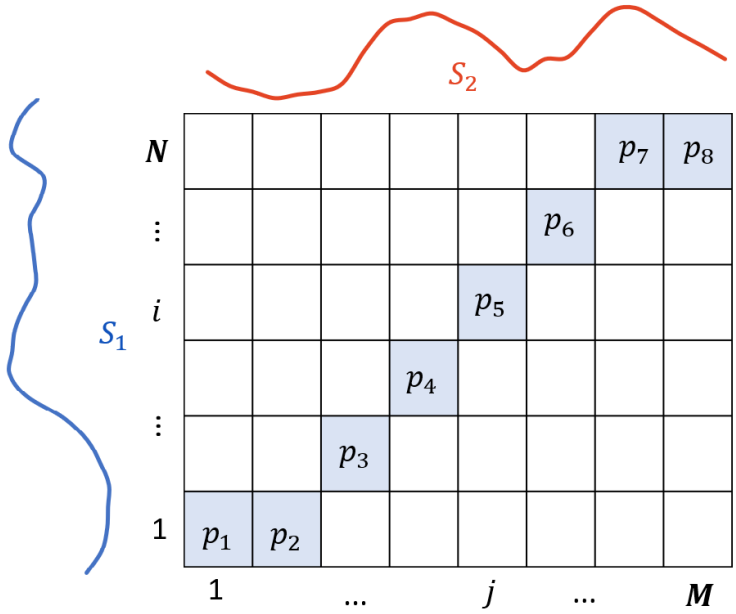
DTW algorithm is a typical optimization problem with constraints. For example, for two time series S_1 and S_2 whose lengths are respectively N and M, a N-by-M distance matrix can be constructed, where a (i, j) position in the matrix is given by $dist(S_1[i], S_2[j])$. In this matrix a path can be drawn (warping path), and it may be represented as $\mathcal{P} = p_1, p_2, ..., p_k, ..., p_K$, where p_k is a specific position in the matrix and K is the path length. Note that the warping path must follow the restriction where $max(N,M) \leq K < N + M - 1$.

For selecting a warping path, several constraints must be obeyed.

- **Boundary conditions:** The warping path must start with the point $p_1 = (1,1)$ and finish at point $p_K = (N,M)$
- **Continuity:** Consecutive adjacent path elements $p_k = (i_k, j_k)$ and $p_{k-1} = (i_{k-1}, j_{k-1})$, $i_k - i_{k-1}$ and $j_k - j_{k-1}$ should be less than or equal to 1.
- **Monotonicity:** The elements in the path must be monotonically spaced. Then, consider $p_k = (i_k, j_k)$, for any previous element path $p_t = (i_t, j_t)$ where $t < k$, $i_k - i_t$ and $j_k - j_t$ should be greater than or equal to 0.

An illustration of a distance matrix with a warp path highlighted in light blue is illustrated in Figure 5. The series S_1 and S_2 are samples extracted from ItalyPowerDemand dataset, from UCR Repository.

Figure 5 – Distance matrix with a warping path



Source: The author.

The path minimizing the warping cost is

$$DTW(S_1, S_2) = min\left\{\frac{\sqrt{\sum_{k=1}^{K} p_k}}{K}\right\} \tag{2.6}$$

A brief review about DTW can be found in Senin (2008). In particular, (BAGNALL *et al.*, 2017) present DTW algorithm improvements and variations found in the literature.

2.2.2 Data representation and dimensionality reduction

Data representation consist in methods used to represent information. In time series context, data is typically represented by a sequence of observations in a temporal domain. Originally, time series are high dimensional data (HAN *et al.*, 2011). Directly dealing with raw time series can be costly in processing and data storage terms. Therefore, data representation methods are used for reducing the original data dimensionality to improve the efficiency of algorithms in dealing with time series. Alternatively, time series can be represented in several different ways such as in spectral domain, wavelets and symbolic.

In high dimensionality data applications, such as in image processing and time series analysis, data sets can have much more variables than observations. Dealing with high dimensional data lead to a severe problem, usually referred to as the *curse of dimensionality*.

Working with massive amount of data can lead to serious problems in processing and storage requirements. For instance, a massive time series data could not fit into the main memory. Thus, multiple swapping requirements in main memory will be necessary in order to access all the data, resulting in a extremely slower algorithm. This problem is considered as the major bottleneck in data mining application, and can be solved by dimensionality reduction techniques (RALANAMAHATANA *et al.*, 2005).

In this work, dimensionality reduction and data representation is considered as equivalent concepts. Dimensionality reduction can be accomplished by two different approaches (SORZANO *et al.*, 2014):

- **Feature selection:** Consists in selecting the most relevant variables (or attributes) from the original input pattern. Usually, the selected features are those that mostly explain the data to be classified. In other words, the variables containing the most useful information are preserved, and the others discarded.
- **Feature extraction:** Consists in transforming the original input pattern variables (or attributes), generating a new set of variables. Typically, the new set of variables have a lower dimensionality (quantity of variables) than the original data set. Example of feature extraction techniques are Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA).

A survey on dimensionality reduction techniques can be found in Sorzano *et al.* (2014). Data representation techniques are described in details in (RALANAMAHATANA *et al.*, 2005).

This master's thesis deals with originally generated data (raw data), and no data representation technique is employed. Nevertheless, this topic is considered a worthwhile concept to be addressed.

2.2.3 Applications

In the literature, time series can be found in the most varied application domains, involving different types of problems. These applications can benefit from different data mining techniques. In the context of machine learning and time series data mining, the most commonly explored applications are: classification, clustering, forecasting, anomaly and motif detection (FAKHRAZARI; VAKILZADIAN, 2017). Several surveys and reviews have explored the main applications in time series data mining field. Valuable sources for data mining applications can be found in Fakhrazari & Vakilzadian (2017), Bagnall *et al.* (2017), Ralanamahatana *et al.* (2005). In this master's thesis, problems related to classification, clustering and anomaly detection are closely related to our work. Thus, these applications are described in more details.

2.2.3.1 Classification

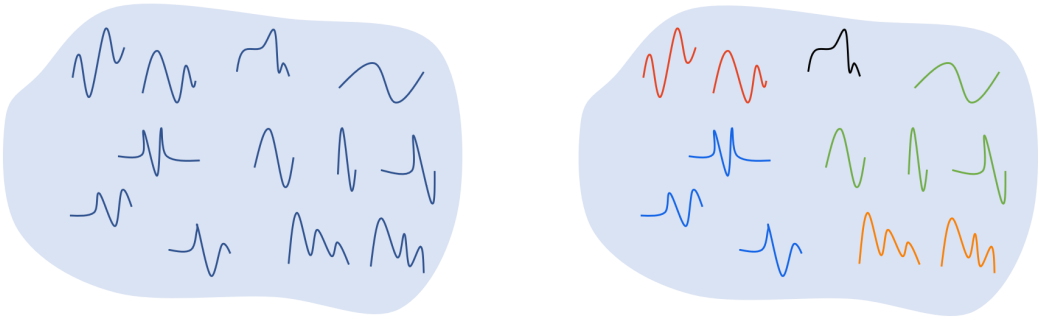
Classification of time series is a typical data mining task. In classification problems, the objective is to assign a specific predefined class or label to a time series. Time series classifiers are usually trained by supervised approaches, which consists of training a model by presenting the input pattern and the desired class. Figure 6 presents two types of time series data, unlabeled and labeled.

In Figure 6(a), several time series are illustrated, but none of them are assigned to a predefined class, this type of data are usually used in clustering problems, as will be discussed in Section 2.2.3.2. Figure 6(b) presents a different situation. All data sample are assigned to a specific class. In this example, multiple time series data are assigned to five different classes, where each class is represented by a color (red, blue, green, orange and black). Labeled data are commonly used in classification problems.

Each time series data sample is represented as N-dimensional vector, where N is the length of the data sample. Each observation can be one-dimensional or a multi-dimensional, depending on the time series (See section 2.1.1).

It can be seen in Figure 7 the representation of labeled and unlabeled data. Notice that the only difference between them is that labeled data has an additional value which specifies

Figure 6 – Unlabeled and Labeled time series data

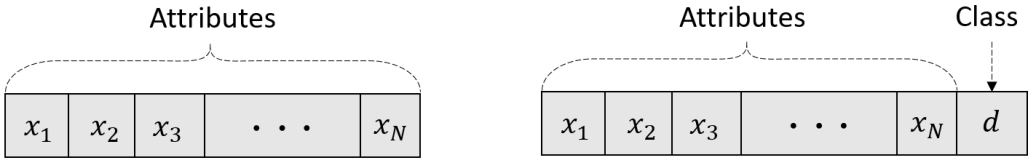


(a) Unlabeled time series data (b) Labeled time series data

Source: The author.

the class of a sample (d).

Figure 7 – Unlabeled and Labeled time series data representation



(a) Unlabeled data (b) Labeled data

Source: The author.

A data set time series can be represented as a matrix. For instance, a labeled data set may be represented by the following matrix:

$$\mathbf{X} = \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,\ell} & d_1 \\ x_{2,1} & x_{2,2} & \cdots & x_{2,\ell} & d_2 \\ \vdots & \vdots & \ddots & \vdots & \\ & \vdots & & & \\ x_{N,1} & x_{N,2} & \cdots & x_{N,\ell} & d_N \end{pmatrix} \tag{2.7}$$

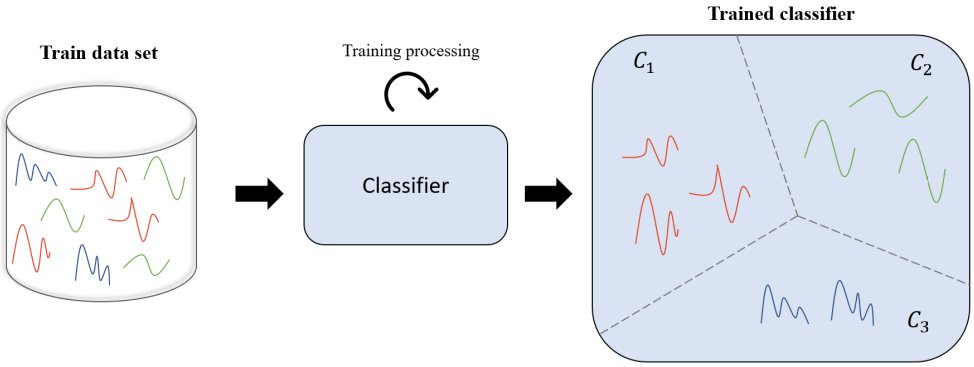
where $\mathbf{X}$ is the data set composed by N vectors of dimensionality $\ell + 1$. Note that the dimensionality of each vector is the time series length ℓ itself, followed by a desired label (or target). Therefore, a labeled sample $\mathbf{x}_i$ is given by:

$$\mathbf{x}_i = \begin{pmatrix} x_{i,1} & x_{i,2} & \cdots & x_{i,\ell} & d_i \end{pmatrix} \tag{2.8}$$

where d_i is a desired output for the i-th sample. In classification problem, $d_i \in \mathbb{N}$ as it represents desired class labels. Unlabeled datasets are represented by a matrix $\mathbf{X}$, without the last column of desired classes.

In learning-based classification algorithms, the classifier needs to be firstly trained, so it can learn how to distinguish different predefined data classes. The training process, also called learning, consists in presenting a set of time series samples (training set) that can be used by a classifier for adjusting properly a model for classifying any data from this domain. Figure 8 illustrates the training process. As illustrated in the figure, every sample on the training data set is presented to the classifier. Once the training is completed, a trained classifier is produced.

Figure 8 – Training process for classification



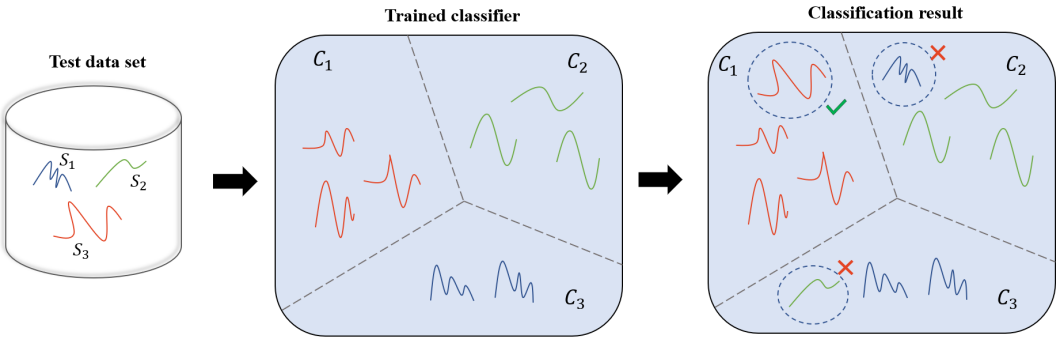
Source: The author.

Ideally, after training, the classifier should be able to classify correctly any data from the training set. However, it is not guaranteed that the model will be perfectly adjusted. On the other hand, even though a trained classifier can correctly classify all samples from training set, its robustness is not assured. A better way of evaluating the classifier's performance, is by verifying how well it performs classification in a set of samples which has not been presented to the classifier, during the training process. This set is usually referred as the test set.

Figure 9 describes a testing process, where a test set is presented to a trained classifier to classify its samples. In the following figure, the test set is composed by three time series (S_1,S_2 and S_3). After classification, only the time series S_1 was correctly classified, whereas the other two series were misclassified. Therefore, only one out of three was classified correctly, giving a poorly classification accuracy of 33.3%

The previous example show how to use a test set for determining the classifier generalization capability. Generalization determines how general a classifier is trained for a specific domain. In other words, a trained classifier has a good generalization if it is able to properly classify samples from test set. If a classifier presents high classification performance in training set and classify poorly test set samples, this classifier has not performed a proper generalization of the data.

Figure 9 – Testing process for a classifier



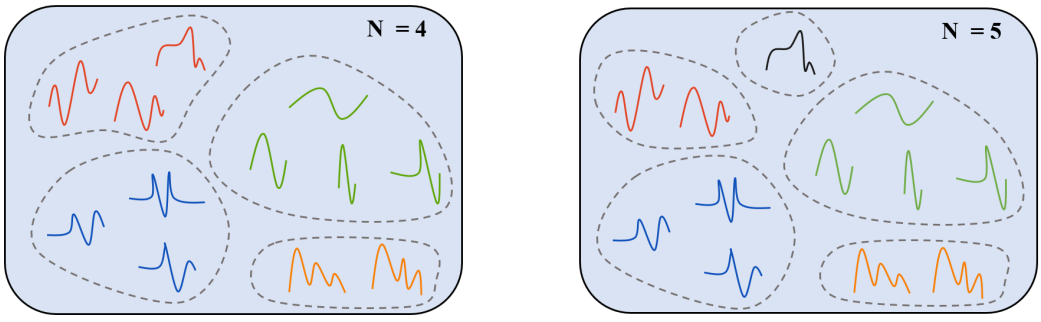
Source: The author.

2.2.3.2 *Clustering*

Clustering consists in grouping time series data into multiple clusters accordingly to a similarity measure (See section 2.2.1). Clustering algorithms are generally considered an unsupervised approach, where the data used for training a model are not assigned to any class, that is, the data is unlabeled (See Figure 6(a)).

Typically, clustering algorithms try to maximize variance between elements from different clusters while minimizing variance between elements in the same cluster. An important parameter for clustering algorithms is the quantity of clusters (N). Finding the optimal N is one of the challenges in clustering problem (ESLING; AGON, 2012). In Figure 10, two examples of clustering are illustrated, considering $N = 4$ and $N = 5$. It can be easily noticed that the parameter N directly affects in the clustering result. For further details about time series clustering techniques, a complete review can be referred in Kavitha & Punithavalli (2010) and Aghabozorgi *et al.* (2015).

Figure 10 – Clustering results for N = 4 and N = 5



(a) Clustering solution with 4 clusters

(b) Clustering solution with 5 clusters

Source: The author.

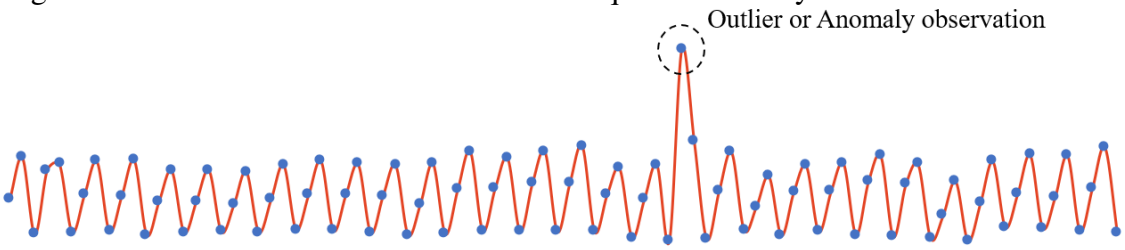
2.2.3.3 Anomaly Detection

Anomaly detection problems can be defined as finding abnormal or unexpected patterns in data (CHANDOLA *et al.*, 2009a). Anomaly, in time series, are generally infrequent outlier observations or subsequences that may appear in a time series. Example of anomaly behaviour are spikes, drops and level shifts. Anomaly detection problems can be described in several different ways. A common approach is to consider an anomaly detection problem as a binary classification problem, where the time series data can be classified as anomaly or not anomaly. This approach can also be generally divided into two categories:

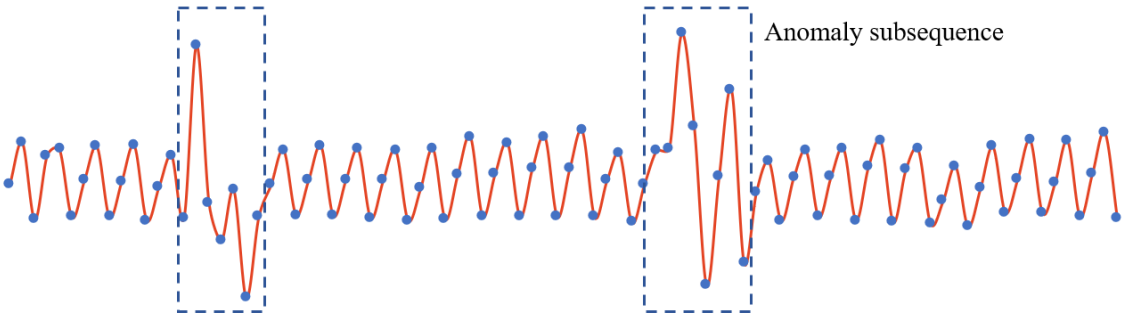
- **Observation Anomaly Detection:** It is a problem focused on finding anomaly observations or outliers in a time series.
- **Subsequence Anomaly Detection:** This problem consists in finding one or more anomaly subsequences in a time series.

In Figure 11, two groups of anomaly detection are shown. Figure 11(a) presents a single anomaly observation, which is typically called an outlier. In this example, an anomaly detection algorithm would classify all observations as not anomaly, except the outlier.

Figure 11 – Time series observation and subsequence anomaly detection



(a) Time Series Observation Anomaly Detection



(b) Time Series Subsequence Anomaly Detection

Source: The author.

Figure 11(b) illustrates an example of two subsequences anomaly in a time series. A systematic literature review in anomaly detection was conducted by Chandola *et al.* (2009a), whose fundamentals on anomaly detection, regarding its formal definition, categories, applications and challenges are addressed. In addition, studies about anomaly detection techniques in time series database are explored in Chandola *et al.* (2009b).

2.3 Time Series Classification Problem (TSC)

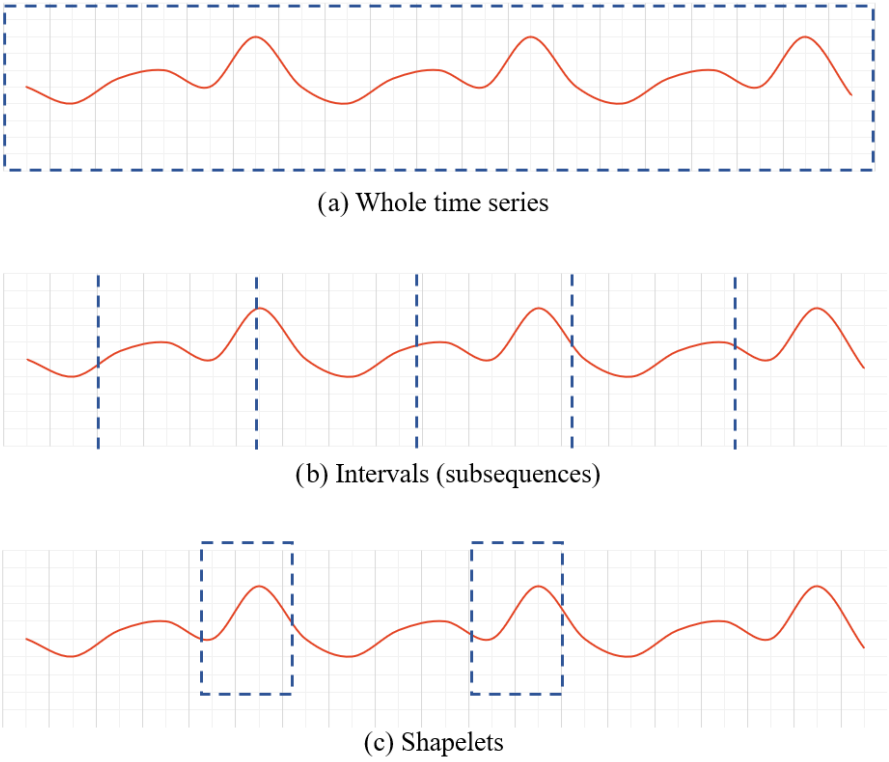
Time series classification (TSC) is a specific field in data mining which studies the main techniques for classifying multi-dimensional sequential data. In this context, classification models (generally called classifiers) are designed for learning how to distinguish specific classes from different time series. A classifier consists in a function that maps the space of possible inputs to a specific class.

In time series data mining, there are different algorithms with specific approaches for classifying time series. Bagnall *et al.* (2017) presented a taxonomy of TSC algorithms based on the type of discriminatory features explored by the technique. The TSC categories most closely related to this work are:

1. **Whole series:** Time series are compared as a vector or by a distance measure that uses all the data. Typically, whole series samples are composed by large number of observations. Algorithms based on this approach usually quantifies the distance between two series employing a similarity measure. Whole series similarity is appropriate when there may be discriminatory features over the whole series. These distance measures are usually employed with a Nearest Neighbour (NN) classifier (BAGNALL *et al.*, 2017).
2. **Intervals:** Instead of using the whole series, as cited in the previous approach, this technique select one or more intervals of the time series (subseries). This approach is also called phase dependent. This strategy can be very useful for dealing with time series that presents noise or outliers.
3. **Shapelets:** Shapelets is similar to the interval-based approach, however this method is focused on finding short patterns or subsequences (shapelets) that define a class. These pattern can appear anywhere in the series and its location is irrelevant. A class is distinguished by the presence or absence of one or more shapelets in the whole series.

Figure 12 exhibits the different approaches of classifying time series. As it can be noticed, Figure 12(a) presents an example of a whole time series. In this case, all observations from the time series is used for classification. In Figure 12(b) there is a different situation. In this case, the whole time series is segmented in several subsequences, where each subsequence will be classified separately.

Figure 12 – Time series classification approaches



(a) Whole time series

(b) Intervals (subsequences)

(c) Shapelets

Source: The author.

Finally, in Figure 12(c), the approach of classifying a time series by its shapelets is shown. In the figure, two shapelets are presented. A time series then can be distinguished by finding these highlighted forms.

Several other TSC approaches have been studied by Bagnall *et al.* (2017). For example, dictionary-based techniques deals with TSC by analyzing the frequency of recurring subsequence. Moreover, in model-based algorithms, each time series class is modeled, which can be compared by a similarity measure between models. Bagnall *et al.* (2017) also cited the possibility of combining multiple approaches in a single classifier.

Among the approaches discussed previously, only TSC problems involving whole time series and interval-based techniques will be covered. However, the algorithms used in the experiments can be easily adapted for working as with other approaches.

2.4 State-of-the-art

The key aspects for building an efficient time series classifier are mainly related to how to deal with temporal data. Therefore, great improvements in designing classifiers are not concentrated in classification strategies, but mainly in how to properly process the data to be classified. In general, time series are high dimensional data, noisy and usually present outliers (KLEIST, 2015). These particularities increase fairly the problem complexity. A large portion of the research in this area is based on designing techniques to circumvent the challenges presented by time series data. Specifically, many of the researches are based on tackling two major challenges in time series: data representation and similarity measure (GIUSTI, 2017; FAKHRAZARI; VAKILZADIAN, 2017; AGHABOZORGI *et al.*, 2015; ESLING; AGON, 2012; FU, 2011; RALANAMAHATANA *et al.*, 2005).

Choosing an appropriate data representation is essential for improving accuracy and efficiency in a data mining algorithms. Several of these methods also focus on reducing the data dimensionality, as it can contribute for improving algorithm processing performance and memory storage. In the literature, there are multiple time series data representation and dimensionality reduction techniques: Discrete Fourier Transform (DFT) (BLOOMFIELD, 2004), Discrete Wavelet Transformation (DWT) (CHAN; FU, 1999), Piecewise Aggregate Approximation (PAA) (KEOGH *et al.*, 2001) and Symbolic Aggregate approXimation (SAX) (LIN *et al.*, 2007).

Similarity measure or distance measure need to be carefully defined in classification algorithms. Some techniques depend heavily on a mechanism of measuring similarities between time series (See section 2.2.1). These techniques are generally called distance-based classification algorithms. A well-know distance-based algorithm is the Nearest Neighbor (NN), usually mentioned as k-NN, where k is the number of nearest neighbors. There are a significant number of distance measure method in the literature, such as Euclidean Distance (ED) (RALANAMAHATANA *et al.*, 2005), Dynamic Time Warping (DTW) (BERNDT; CLIFFORD, 1994) and Longest Common Subsequence (LCSS) (VLACHOS *et al.*, 2002). In Wang *et al.* (2013), an extensive experimental study was conducted for testing the effectiveness of different time series representations and similarity measures. Eight different time series representation method and nine similarity measures were tested in a variety of application domain (38 data sets).

Another important area in time series data mining is concerned in design and conduct experiments with machine learning techniques for evaluating the performance of existing and novel methods.

According to (XING *et al.*, 2010), techniques for sequential data classification can be categorized in three main groups: distance-based, model-based and feature-based algorithms.

Distance-based algorithms require a function to measure the similarity between sequences. In this approach, lazy learning methods, such as $k-$Nearest Neighbor classifier (k-NN), have been constantly explored in the literature. Such technique does not require a trained model for classification. A new instance is classified by comparing distance between the new element and the k-nearest neighbors on the training set. Several papers have employed $k-NN$ algorithm for solving TSC problems:

- **Nearest Neighbors (NN)** (DO *et al.*, 2017; PARVINNIA *et al.*, 2014; DUARTE *et al.*, 2014; LEE *et al.*, 2012; NASIBOV; PEKER, 2011);

Model-based algorithms assume that a sequence are generated by an underlying generative model M , with a probability distribution (XING *et al.*, 2010). A model M is firstly trained for tuning the model's parameters, then, the classification of a new instance is assigned to the class with the highest likelihood. Example of model-based algorithms are Naive Bayes (HILLS *et al.*, 2013) and Hidden Markov Model (HMM) (RONAO; CHO, 2014).

Feature-based algorithms are conventional classification methods designed for working with vector (XING *et al.*, 2010). A time series can be represented basically in two forms: a raw vector and a feature vector. A raw vector would consider a time series a high-dimensional vector as a input vector for a feature-based classification method. Alternatively, features can be extracted from the temporal data. Hence, a feature vector based on the time series characteristics can be built. Decision trees and neural networks are examples of feature-based algorithms.

From the last decade until recently different Neural Network based algorithms have been constantly employed in time series classification:

- **Multi-Layer Perceptron (MLP)** (DELGADO *et al.*, 2017; NAKANO; CHAKRABORTY, 2017; BHATIA; RANI, 2016; FEJFAR *et al.*, 2013; NOVIYANTO; ARYMURTHY, 2012; ORHAN *et al.*, 2011);
- **Learning Vector Quantization (LVQ)** (JAIN; SCHULTZ, 2018; MELIN *et al.*, 2014; BISWAL *et al.*, 2014; MOKBEL *et al.*, 2015; LIU *et al.*, 2015);
- **Support Vector Machine (SVM)** (RAJESH; DHULI, 2017; DOBROWOLSKI *et al.*, 2016; XU *et al.*, 2016; SHARMA; PACHORI, 2015; JEONG; JAYARAMAN, 2015; SU *et al.*, 2014; ŞTEFAN, 2012; ADNANE *et al.*, 2012; EKICI, 2012; ISCAN *et al.*, 2011; SALAHSHOOR *et al.*, 2010; ORSENIGO; VERCELLIS, 2010);

2.5 Summary

This chapter has presented basic concepts regarding time series data mining in pattern recognition problems. It was formally defined what are time series and resented some of the main data mining time series application research, highlighting the problems of classification, clustering and anomaly detection.

Several challenges in dealing with temporal data were discussed, as well as presented the main techniques employed in the literature for solving time series classification problems. Among them, focused on the neural network based algorithms (MLP, LVQ and SVM), and the naive k-nearest neighbor method, as they will be studied in this thesis.

In the Chapter 3, the k-NN method and the studied neural network algorithms are described in details.

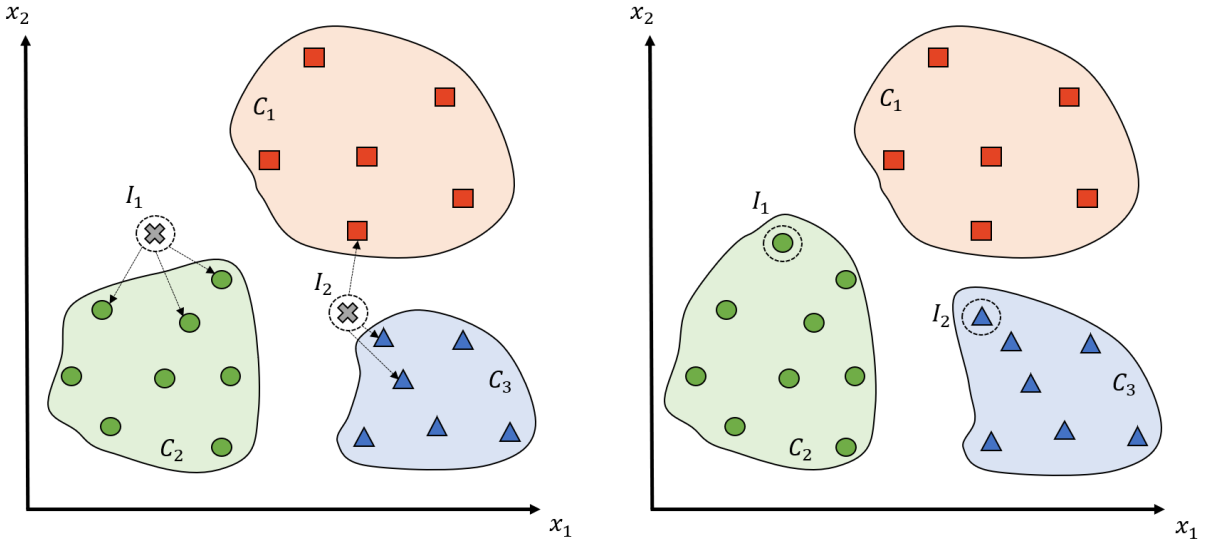
3 THEORETICAL BASIS

This chapter starts presenting a simple classification method called *k*-Nearest Neighbor (*k*-NN). In the literature, the *k*-NN classifier is considered a benchmark for time series classification. Furthermore, theoretical and mathematical concepts regarding the classification algorithms adopted in this work are discussed. The algorithms studied are Support Vector Machine (SVM) and three well-known neural network based classifiers: Multi-Layer Perceptron (MLP), Self-Organizing Map (SOM) and Learning Vector Quantization (LVQ).

3.1 *k*-Nearest Neighbor classifier

Nearest neighbor (NN) is an instance-based classification method, which strategy consists in finding in the training set the closest neighbors to a given new instance. The closest neighbor class will be used for classifying the new instance. In this approach, a parameter *k* is employed to define the number of closest prototypes that will be examined for deciding the class of the new instance. Typically, these classifiers are called *k*-NN. For instance, a 3-NN classifier consider three closest elements (from the training set) in order to assigning a class to the new element. Figure 13 exemplifies the classification using a 3-NN classifier.

Figure 13 – Example of *k*-nn classification



(a) Two unlabeled instances before 3-NN classification

(b) New instances after 3-NN classification

Source: The author.

NN-based classifiers depend on distance measures for comparing similarity between elements. Typically, it uses L_p norm distance measures, such as Euclidean Distance (See section 2.2.1.1). k-NN is one of the most common classification method in literature, and it is considered a benchmark method for time series classification (NASIBOV; PEKER, 2011; LEE *et al.*, 2012; DO *et al.*, 2017). For instance, 1-NN with DTW have demonstrated very effective on TSC problems (XI *et al.*, 2006).

In Figure 13(a), there are two new unlabeled instances, I_1 and I_2. The dashed arrows show the three nearest neighbors from each instance. In I_1 all nearest neighbors are from class C_1; therefore, I_1 will be assigned to this class. Similarly, in I_2, the majority of the nearest neighbors are from C_2, then this will be the class assigned for I_2. Figure 13(b) presents the classification result.

This example shows a simple strategy called majority vote, and this will be adopted in k-NN implementation. There are several other approaches for deciding how to assign the class based on the nearest neighbors. However, further details on this classifier will not be discussed as it is beyond the scope of this work.

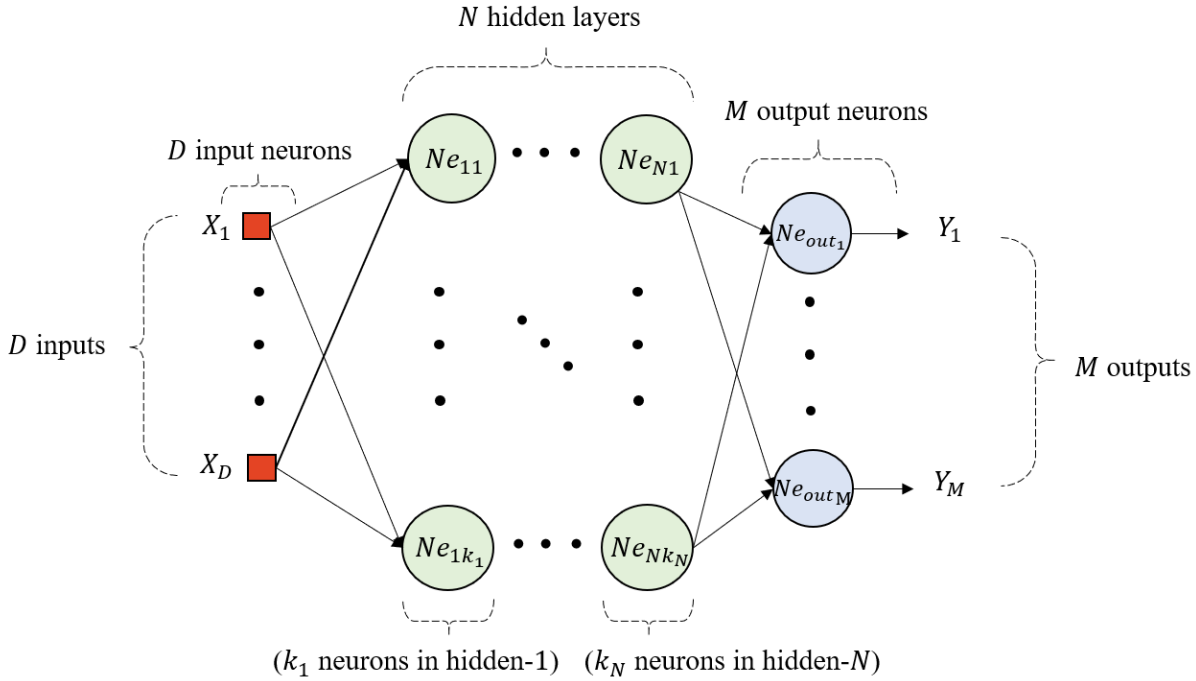
3.2 Multi-Layer Perceptron (MLP)

Multi-Layer Perceptron (MLP) is a widely used ANN-based technique, constantly applied to signal processing, adaptive control, pattern recognition, and so on. A typical MLP architecture is formed by a feedforward network composed by an input layer, with neurons for receiving the input patterns; a hidden layer, responsible for applying nonlinear mapping in the data; and the output layer, where the output is computed (See Figure 14). In this illustration, a general MLP is composed by D, N and M neurons for input, hidden and output layers, respectively. The number of neurons in each layer will depend on the problem in which the neural network is designed for.

The neurons presented in the hidden layer act as *feature detectors*. These features are detected during the learning process by performing nonlinear transformation on the input data into a new space known as *feature space* (HAYKIN *et al.*, 2009). Therefore, the complexity of a MLP-based neural network is generally defined by quantity of hidden neurons (N_h). Complex problems usually require great quantities of hidden neurons (N_h), since the greater N_h the higher is the networks non-linear mapping capability.

Nevertheless, the excess of N_h could make the network to specialize in the training dataset and lose its generalization capability, resulting in a problem called overfitting. On the other hand, a network with few N_h loses the capability to extract features from the dataset, presenting poor training, resulting in a problem known as underfitting. Regarding the neural network architecture design, there is no definitive method for determining the number of hidden neurons. Nonetheless, there are techniques developed to search optimal parameters and hyper-parameters for being applied in models. For example, Grid Search based algorithms are commonly used to find optimal parameters for classification learning models (JIMéNEZ *et al.*, 2009).

Figure 14 – Generic MLP architecture



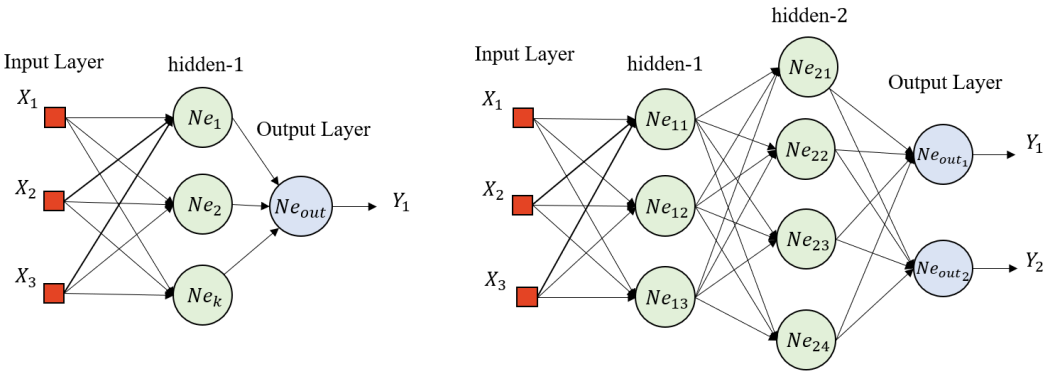
Source: The author.

In Figure 15, two examples of MLP architectures are presented. Note that the numbers separated by hyphens represent the quantity of neurons in each layer. For example, in Figure 15(b) the neural network consist of 3, 3, 4 and 2 neurons in input, first hidden, second hidden and output layers, respectively. These architectures are example of *shallow* MLP neural networks, as they are composed by few hidden layers. In contrast, MLP-based *deep* neural networks present high quantity of hidden layers.

3.2.1 MLP Training Process

MLP-ANN are trained by a supervised learning process, which adjust the synaptic weights of the network's neurons, so the resulted error from the output layer is the lowest possible.

Figure 15 – Examples of MLP architectures



(a) 3-3-1 Multi-Layer Percepetron (b) 3-3-4-2 Multi-Layer Perceptron

Source: The author.

Therefore, this problem can be interpreted as an optimization problem, since the objective is to minimize the cost function in each iteration.

In this optimization problem, the cost function is the sum of squared error (SSE):

$$SSE = \sum_{i=1}^{n} (d(n) - y(n))^2 = \sum_{i=1}^{n} (e(n))^2 \tag{3.1}$$

where N is the total of samples in a training set, $d(n)$ is the n-th desired output and $y(n)$ is the n-th predicted output. This optimization problem is solved by an algorithm called Backpropagation (BP) (HAYKIN *et al.*, 2009).

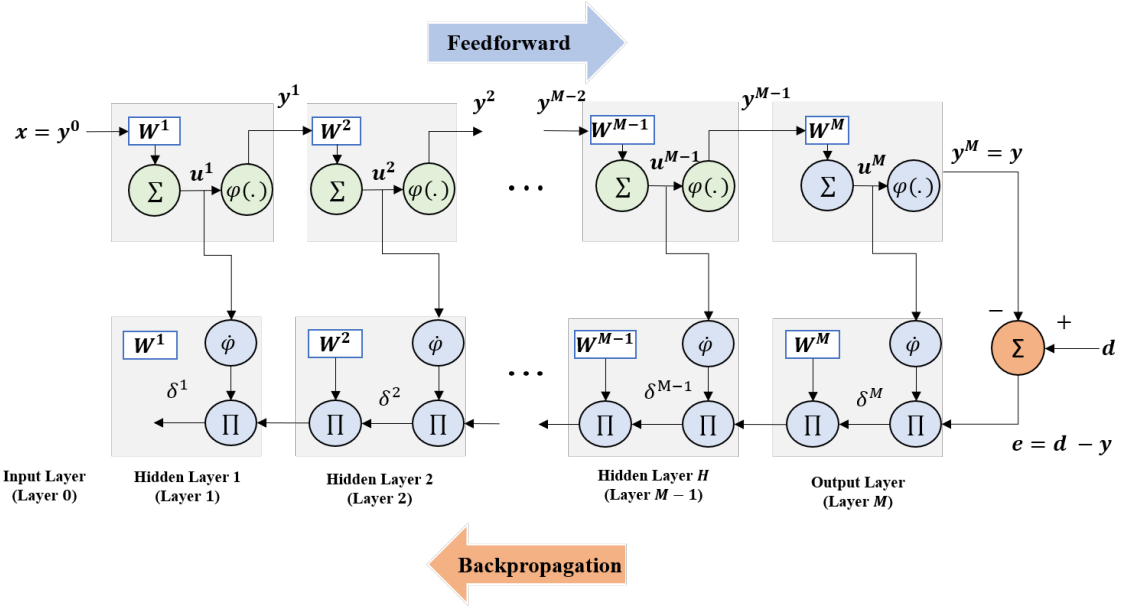
The backpropagation algorithm is a Least Mean Square (LMS) generalization which explores the gradient descend technique iteratively in order to minimize a cost function (HAYKIN *et al.*, 2009). In a MLP-ANN, the error calculated in the output layer is not explicitly related to hidden layer neurons, therefore, the hidden neurons influence in the output cannot be determined directly. Hence, the main contribution from BP algorithm in training a MLP is in providing a solution for adjusting hidden neurons' synaptic weights by propagating the error signal backwardly from the output layer to the input layer. BP algorithm can be divided in two main stages:

- **Feedforward:** The input pattern is presented to the neural network, and the signal is propagated forwardly from the input to the output layer. In this stage, the output of the ANN is calculated.
- **Backpropagation:** After resulting an output for the ANN, the error is calculated and it is propagate backwardly from the output layer to input layer. In this stage, the synaptic weights are adjusted.

3.2.1.1 *Feedforward*

In this stage, an instance $\mathbf{x}_i$ is presented to the network's input layer, and the signal is forwardly propagated to compute its respective output, y_i. Figure 16 illustrates the output $\mathbf{y}$ calculation in vector form.

Figure 16 – Feedforward propagation in vector form



Source: The author.

Considering n a time reference which represent the algorithm's iteration. For any layer m, the induced local field vector $\mathbf{u}^m$ is given by the linear combination between the previous layer output $\mathbf{y}^{m-1}$ and the matrix of weights $\mathbf{W}^m$ of the current layer, as described in Equation (3.2).

$$\mathbf{u}^m(n) = \mathbf{W}^m(n)(\mathbf{y}^{m-1}(n))^T \tag{3.2}$$

There is a matrix of weights for each hidden layer. A matrix of weights in layer m is given by:

$$\mathbf{W}^m = \begin{pmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,\ell} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,\ell} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k_m,1} & w_{n,2} & \cdots & w_{k_m,\ell} \end{pmatrix} \tag{3.3}$$

where k_m is the number of neurons in layer m and ℓ is the dimension of $\mathbf{y}^{m-1}$. The output signal $\mathbf{y}^m$ is calculated by applying an activation function to $\mathbf{u}^m$.

$$\mathbf{y}^m(n) = \varphi(\mathbf{u}^m(n)) \tag{3.4}$$

Note that Equation (3.2) and (3.4) are generalized for all layers, except the input layer. In input layer, the output is the pattern **x** itself:

$$\mathbf{y}^0(n) = \mathbf{x}(n) \tag{3.5}$$

3.2.1.2 Backpropagation

In backpropagation, the output result, estimated by the network in feedforward stage, is compared with the desired output, resulting in an error signal for this instance:

$$\mathbf{e}(n) = \mathbf{d}(n) - \mathbf{y}(n) \tag{3.6}$$

the calculated error from feedforward stage is propagated in backwards to the input layer, adjusting the weights from each layer accordingly to a learning rule. Firstly, the local gradient $\delta_j(n)$ is calculated for each output neuron $j = 1, ..., M$:

$$\delta_j^m(n) = e_j(n)\varphi_j'(u_j^m(n)). \tag{3.7}$$

As for calculate the local gradient for hidden neurons, the following equation is used:

$$\delta_j^m(n) = \varphi_j'(u_j^m(n)) \sum_k \delta_k^{m+1}(n) w_{kj}(n). \tag{3.8}$$

The learning rule used for adjusting the network's weights is given by the *Delta Rule*, defined by the following equation:

$$w_{kj}^m(n+1) = w_{kj}^m(n) - \alpha \delta_k^m y_j^{m-1} \tag{3.9}$$

The same equation can be written in matrix form:

$$\mathbf{W}^m(n+1) = \mathbf{W}^m(n) - \alpha \delta^m (\mathbf{y}^{m-1})^T \tag{3.10}$$

where $\mathbf{W}^m(n+1)$ and $\mathbf{W}^m(n)$ is the new and old matrix of weights, respectively. The learning rate (α) is defined by $0 < \alpha << 1$.

The backpropagation method is executed iteratively for each input sample applied to the network's input layer, until all the input dataset is presented. The training set can be presented to the neural network multiple times. A cycle where all training samples from training set are presented to the classification model is called epoch. The Mean Squared Error (MSE) is usually used as a performance measure for evaluating the backpropagation algorithm.

The training process should finish when a stop criteria is acquired. Generally, these criteria are limited by the number of epochs, or by predefined threshold error, or both. A drawback of the backpropagation algorithm lies in its slow convergence and costly processing. Furthermore, it is not guaranteed that this algorithm will achieve the global minimum error, as there is a risk the algorithm will be stuck into a local minimum, preventing the network from improving its model.

3.3 Self-Organizing Map (SOM)

Self-Organizing Map (SOM) is a neural network capable of organizing data into groups or clusters. This category of unsupervised learning algorithm has been introduced by Teuvo Kohonen (KOHONEN, 1997). SOM neural networks use only the data attribute for organizing clusters based on the similarity between samples.

The main objective of the SOM network consists in mapping a continuous high-dimensional space to a discrete space with reduced size. This mapping, or projection, is composed by N_w neurons (or prototypes) organized in a S-dimensional space, typically two-dimensional. Formally, for a continuous space $\mathcal{X} \subset \mathbb{R}^D$ and a discrete space $\mathcal{Y} \subset \mathbb{R}^S$, composed by N_w prototypes, a vector $\mathbf{x} \in \mathcal{X}$ will be represented by the network for a vector $y_{i^*} \in \mathcal{Y}$ by the mapping $i^*(x) : \mathcal{X} \to \mathcal{Y}$.

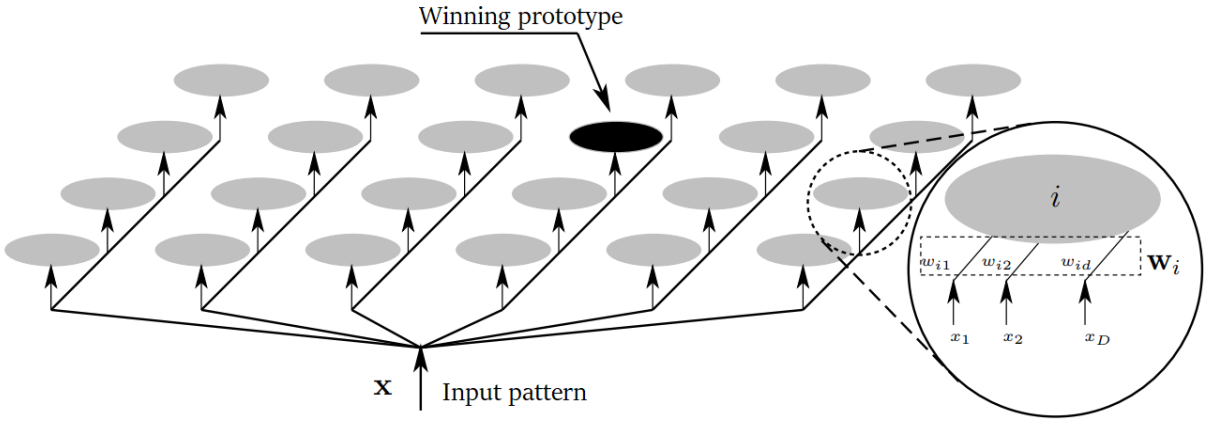
The SOM neural network is a competitive neural network that can be used in both clustering and vector quantization tasks. Clustering, as explained previously, consists in separating the data in groups based on a similarity criterion. Vector quantization is the task of replacing a set of N vectors by a set of N_w prototypes, in which $N_w << N$.

A structure of a SOM algorithm is generally divided in two layers: input layer and output layer. A sample from the dataset is presented to the input layer, which is connected to each neuron in output layer. All N_w neurons receive from the input layer a data sample $x(n) \in \mathbb{R}^D$, simultaneously. The attributes contained in $x(n)$ are weighted by the ith neuron's vector of weights $w_i(n) \in \mathbb{R}^D$. The architecture of a SOM network is shown in Figure 17.

3.3.1 SOM neural network training process

For training a SOM network, firstly the N_w neurons' weights are randomly initialized by small values. After proper initialization, the algorithm proceeds with three essential stages (HAYKIN *et al.*, 2009; MATTOS, 2011):

Figure 17 – Example of two-dimensional SOM network. The input and weight vectors are D-dimensional. The N_w neurons are uniformly arranged in a rectangular grid.



Source: Adapted from (MATTOS, 2011).

1. **Competition**: For a input pattern $\mathbf{x}(n)$, the SOM network calculates the nearest prototype (winning prototype) to $\mathbf{x}(n)$, based on a similarity measure:

$$i^*(n) = arg\ min \left\| \mathbf{x}(n) - \mathbf{w}_i(n) \right\|, \forall i \tag{3.11}$$

where $\|\cdot\|$ denotes the Euclidean Distance. As discussed in earlier sections, there are several other similarity measures.

2. **Cooperation**: The winner prototype's weights, as well as its neighbours' weights, are adjusted based on the following learning rule:

$$\mathbf{w}_i(n+1) = \mathbf{w}_i(n) + \alpha(n) h_{i,i^*(n)}(n)[\mathbf{x}(n) - \mathbf{w}_i(n)] \tag{3.12}$$

where $0 < \alpha(n) < 1$, correspond to the learning rate in iteration n. The funciton $h_{i,i^*(n)}$ is referred as neighbouring function. This function defines a neighbourhood around the winning prototype. Therefore, the prototypes that will be adjusted is the winning prototype and its neighbours. A typically used topological neighborhood is the *Gaussian* function:

$$h_{i,i^*(n)}(n) = \exp\left[-\frac{\left\| \mathbf{w}_i - \mathbf{w}_{i^*(n)} \right\|^2}{2\sigma^2(n)} \right] \tag{3.13}$$

where $\mathbf{w}_i$ and $\mathbf{w}_{i^*(n)}$ are, respectively, the coordinates of the i-th neuron and the winner neuron $i^*(n)$. The parameter $\sigma(n) > 0$ refers to the neighborhood width considered: the higher its value, the higher the number of neurons updated around the winning neuron.

To ensure the convergence of the SOM network weights to stable values during the training algorithm, it is necessary to reduce the parameter of neighboring and the learning step. Considering σ_0 and α_0 their initial values.

The neighborhood's size σ and the learning rate α can be reduced over time, by an exponential decay:

$$\sigma(n) = \sigma_0 \exp\left(-\frac{n}{\tau_1}\right) \qquad \alpha(n) = \alpha_0 \exp\left(-\frac{n}{\tau_2}\right) \tag{3.14}$$

where τ_1 and τ_2 are time constants to be defined by the designer.

Algorithm 1 shows the process of training a SOM neural network.

Algorithm 1: SOM pseudo-code

Input:

N : Number of iterations
N_w : Number of prototypes
α_0 : Initial learning rate
σ_0 : Initial neighborhood
$\mathbf{X}$: Training data

Output:

W_{opt} : Optimal weights (All prototypes and their respective weights)

Algorithm:

1. Initialization

Initialize the prototypes' weights with random small values $\mathbf{W}_0 = rand(N_w, N_w)$
Initialize the learning rate $\alpha(1) = \alpha_0$
Initialize the neighborhood $\sigma(1) = \sigma_0$

2. Training

while *(Termination criteria not satisfied)* **do**
  for $n = 1$ ***to*** N **do**
    (2.1) Select a sample $\mathbf{x}(n)$ from the dataset
    (2.2) Find the nearest neuron (winner neuron) to sample $\mathbf{x}(n)$

$$i^*(n) = argmin \|\mathbf{x}(n) - \mathbf{w}_i(n)\|, \forall i$$

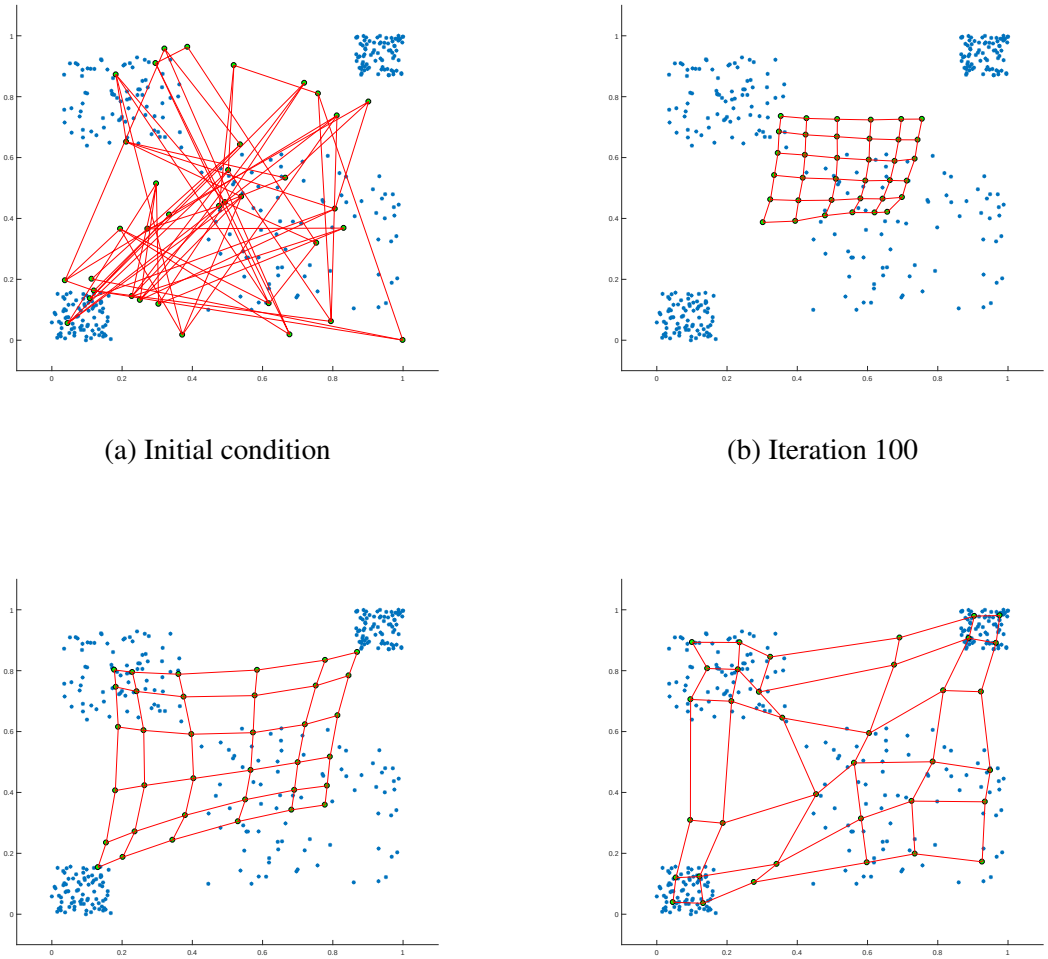
    (2.3) Adjust the weight of the winner neuron and its neighbours:

$$\mathbf{w}_i(n+1) = \mathbf{w}_i(n) + \alpha(n) h_{i,i^*(n)}(n)[\mathbf{x}(n) - \mathbf{w}_i(n)]$$

    (2.4) Update (decrease) the learning rate $\alpha(n)$
    (2.5) Update (decrease) the neighborhood $\sigma(n)$
  end
end

Figure 18 shows an example of a SOM network during training. This example consists of a two-dimensional dataset that must be mapped by the network. Notice that, although the amount of neurons is smaller than the number of samples, over time the SOM network is able to obtain a representation of the training data.

Figure 18 – Evolution of a Self-Organizing Map



(a) Initial condition

(b) Iteration 100

(c) Iteration 1000

(d) Iteration 4000

Source: The author.

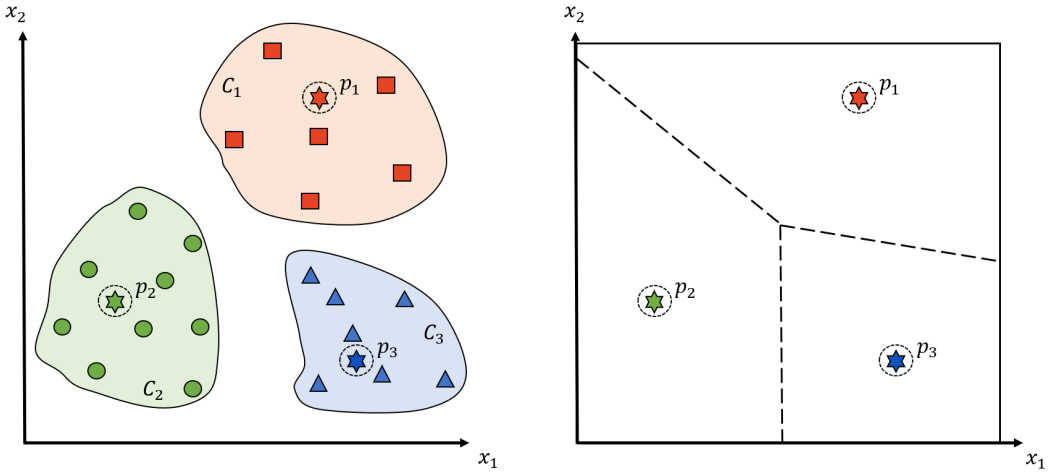
3.4 Learning Vector Quantization (LVQ)

Learning Vector Quantization (LVQ) is a prototype-based supervised classification algorithm which adopts a competitive learning strategy based on similarity measures (distance functions) and winner-takes-all approach. LVQ is a neural network based method proposed by Kohonen (1990).

Its architecture is composed by a layered feedforward network which has a competitive layer where the neurons compete among them based on a distance metric, or a similarity measure, between training instances and prototypes. This method aims to divide the data space into distinct regions and defining a vector prototype (or neuron) for each region. This process is also known as Vector Quantization.

Figure 19(a) illustrates a problem where there are three regions to be represented (or three classes). In this example, each region is represented by a unique prototype. The number of prototypes is not necessarily the same for each class. Depending on the problem, some classes may have more prototypes than others. In Figure 19(b) an illustration by voronoi diagram shows the prototypes and the regions represented by them.

Figure 19 – Prototype-based representations



(a) Prototype representation (b) Voronoi diagram

Source: The author.

3.4.1 Kohonen's LVQ1

The learning method in LVQ consists in using the input vector as guidance for organizing the prototypes in specific regions that defines a class. Firstly, a set of prototypes is initialized and for each prototype is assigned a class. Each class must be represented by at least one prototype. A class can have multiple prototypes, and one prototype only represents a unique class. Then, during the learning process, each instance from the training set is compared with all network's prototypes, using a similarity measure. LVQ-based algorithms are classified as competitive learning due to the selection of the closest prototype within the set of P prototypes:

$$i^*(n) = arg \min_{i=1}^{P} d(\mathbf{x}(n), \mathbf{w}_i) \tag{3.15}$$

where $i^*(n)$ is the index of the winner prototype (the closest prototype of an specific instance $\mathbf{x}(n)$). The distance is measured by a distance function. The Euclidean distance, or L_2-norm, is generally used to calculate this distance.

$$d(\mathbf{x}(n), \mathbf{w}_i) = \|\mathbf{x}(n) - \mathbf{w}_i\|_2 = \sqrt{\sum_{k=1}^{D} (x_k(n) - w_{ik})^2} \tag{3.16}$$

where D is the dimension of the instance $\mathbf{x}(n)$, which is the same for $\mathbf{w}_i$. If the class of an instance is equal to the class of the closest prototype (winner prototype), this prototype is moved towards the instance, otherwise it moves away.

Consider n as the iteration counter of the training algorithm. The learning rule for Kohonen's LVQ1 algorithm is given by:

$$\mathbf{w}_{i^*(n)}(n+1) = \begin{cases} \mathbf{w}_{i^*(n)}(n) + \alpha(n)[\mathbf{x}(n) - \mathbf{w}_{i^*(n)}(n)] & \text{if } C(\mathbf{w}_{i^*(n)}) = C(\mathbf{x}(n)); \\ \mathbf{w}_{i^*(n)}(n) - \alpha(n)[\mathbf{x}(n) - \mathbf{w}_{i^*(n)}(n)] & \text{if } C(\mathbf{w}_{i^*(n)}) \neq C(\mathbf{x}(n)). \end{cases} \tag{3.17}$$

For all prototypes $\mathbf{w}_i$ where $i \neq i^*(n)$, the prototypes remains the same.

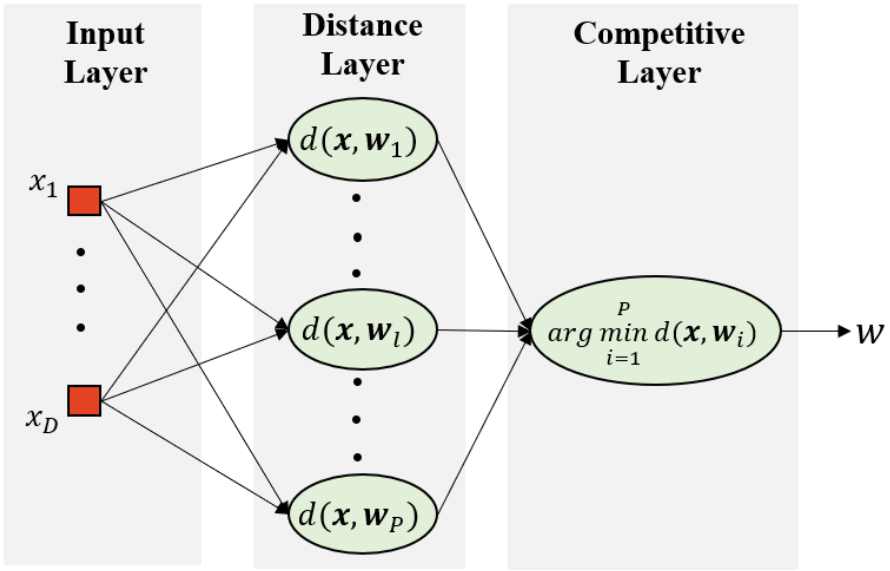
The parameter $\alpha(n)$ is the learning rate, and it should be kept at low values, preferably between 0 and 1. Furthermore, during training, $\alpha(n)$ should decrease monotonically. Among the several ways of decreasing $\alpha(n)$, a linearly decreasing learning rate was employed:

$$\alpha(n) = \alpha(0)(1 - \frac{n}{N}) \tag{3.18}$$

where $\alpha(0)$ is the initial learning rate and N is the maximum number of training iterations.

In Figure 20 a LVQ1 architecture is presented.

Figure 20 – LVQ1 architecture



Source: The author.

Note that the competitive layer consist in finding the closest prototype from an instance $\mathbf{x}$. Therefore, this represent a competitive layer for the Kohonen's LVQ1. As it will be seen in the next sections, the competitive layer will change accordingly to the LVQ strategy adopted. Algorithm 2 presents the pseudo-code for implementing a LVQ1 neural network.

Algorithm 2: LVQ1 pseudo-code

Input:

α_0 : Initial learning rate
$\mathbf{X}$: Training input samples
$\mathbf{Y}$: Desired output (classes)

Output:

W_{opt} : Optimal weights (All prototypes and their respective weights)

1. Initialization

(1.1) Initialize vector $\mathbf{w}_i$ with random values from training set
(1.2) Initialize the learning rate $\alpha(1) = \alpha_0$

2. Training

while *(Termination criteria not satisfied)* **do**
  for $n = 1$ ***to*** N **do**
    (2.1) Select an input pattern $\mathbf{x}(n)$ and its class $y(n)$ from training set
    (2.2) Find the closest prototype from pattern $\mathbf{x}(n)$ (Winner prototype)
      $i^*(n) = argmin \|\mathbf{x}(n) - \mathbf{w}_i(n)\|, \forall i$
    (2.3) Adjust the winner neuron's synaptic weights
    if $Class(\mathbf{w}_{i^*}(n)) = y(n)$ **then**
      $\mathbf{w}_{i^*(n)}(n+1) = \mathbf{w}_{i^*(n)}(n) + \alpha(n) * (\mathbf{x}(n) - \mathbf{w}_{i^*(n)}(n))$; `// Move prototype towards the pattern` $x(n)$
    else
      $\mathbf{w}_{i^*(n)}(n+1) = \mathbf{w}_{i^*(n)}(n) - \alpha(n) * (\mathbf{x}(n) - \mathbf{w}_{i^*(n)}(n))$; `// Move prototype away from the pattern` $\mathbf{x}(n)$
    end
    (2.4) Update learning rate
    $\alpha = \alpha_0 * (1 - \frac{n}{N})$
  end
end

3.4.2 Kohonen's LVQ2

Kohonen introduced in 1988 the LVQ2 algorithm, another variation similar to the original LVQ (KOHONEN *et al.*, 1988). In LVQ2 learning process, two prototypes $\mathbf{w}_{i_1^*}$ and $\mathbf{w}_{i_2^*}$ that respectively represent the first and second nearest prototypes to an instance $\mathbf{x}$, are adjusted simultaneously. The adjustment occurs only when the runner-up ($\mathbf{w}_{i_2^*}$) belongs to the correct class as the winner ($\mathbf{w}_{i_1^*}$) belongs to the incorrect class. Furthermore, these prototypes must fall into a zone defined around the mid plane between them. Let $d_{i_1^*}$ and $d_{i_2^*}$ be the distances of $\mathbf{x}$ to $\mathbf{w}_{i_1^*}$ and $\mathbf{w}_{i_2^*}$, respectively. An instance $\mathbf{x}$ will fall into a *window* of width w_s if Equation (3.19) is satisfied. It is recommended to adopt the width w_s between 0.2 and 0.3 (KOHONEN, 1990). The prototype LVQ2 learning rule is given by the Equation (3.20).

$$min\left(\frac{d_{i_1^*}}{d_{i_2^*}}, \frac{d_{i_2^*}}{d_{i_1^*}}\right) > s \text{ , where } s = \frac{1 - w_s}{1 + w_s} \tag{3.19}$$

$$\begin{aligned}\mathbf{w}_{i_1^*}(n+1) &= \mathbf{w}_{i_1^*}(n) - \alpha(n)[\mathbf{x}(n) - \mathbf{w}_{i_1^*}(n)] \\ \mathbf{w}_{i_2^*}(n+1) &= \mathbf{w}_{i_2^*}(n) + \alpha(n)[\mathbf{x}(n) - \mathbf{w}_{i_2^*}(n)]\end{aligned} \tag{3.20}$$

3.4.2.1 Kohonen's LVQ2.1

The variation of LVQ2 called LVQ2.1 is an improvement of the original LVQ2, which allows either $\mathbf{w}_{i_1^*}$ and $\mathbf{w}_{i_2^*}$ to be the closest prototype vectors to an instance $\mathbf{x}$ (KOHONEN, 1997). Therefore, considering an instance $\mathbf{x}$ into the *window*. The adjustment will occur if one of $\mathbf{w}_{i_1^*}$ and $\mathbf{w}_{i_2^*}$ belong to the correct class, as the other to the inccorect class. In constrast, LVQ2 requires $\mathbf{w}_{i_2^*}$ to belong to the same class as $\mathbf{x}$.

3.4.3 Kohonen's LVQ3

The LVQ3 algorithm is an improved version of LVQ2 variation which consist of introducing an additional rule component to ensure that $w_{i_1^*}$ continue approximating to the class $\mathbf{x}$, when $\mathbf{w}_{i_1^*}$ and $\mathbf{w}_{i_2^*}$ belong to the same class. The included component is:

$$\mathbf{w}_k(n+1) = \mathbf{w}_k(n) + \varepsilon\alpha(n)\left[\mathbf{x}(n) - \mathbf{w}_k(n)\right] \tag{3.21}$$

for $k \in \{i_1^*, i_2^*\}$, if $\mathbf{x}(n)$, $\mathbf{w}_{i_1^*}$ and $\mathbf{w}_{i_2^*}$ (See Equation (3.20)). According to Kohonen (1997), the recommended values for ε are between 0.1 and 0.5, relative to w_s = 0.2 or 0.3

3.4.4 Fuzzy-LVQ

Fuzzy-LVQ is a hybrid algorithm which combine the LVQ's learning rule with fuzzy logic. The following discussed FLVQ variation was proposed by Chung (CHUNG; LEE, 1993). This LVQ variation consists in optimizing a fuzzy objective function by minimizing the network output error, calculated by the difference of the class membership of the target and actual values, and minimizing the distances between training patterns and competing neurons. In their works, Chung & Lee (1993) define the following objective function:

$$Q^m(U,\mathcal{V}) = \sum_{j=1}^{N}\sum_{i=1}^{P}[(t_{ji})^m - (\mu_{ji})^m]d(\mathbf{x}_j, \mathbf{w}_i) \tag{3.22}$$

subject to the following constraints: $\sum_{i=1}^{c}\mu_{ji} = 1; \forall j$ and $\mu_{ji} \in [0,1]; \forall j,i$. The term $d(\mathbf{x}_j, \mathbf{w}_i)$ represents the distance between the i-th prototype and the jth instance (See Equation (3.22)).

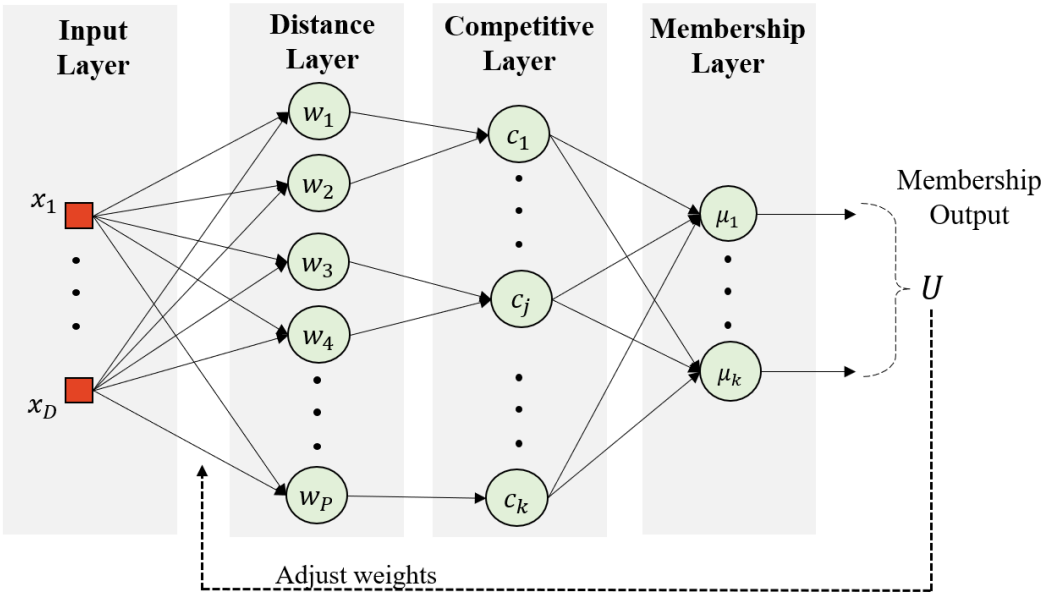
The fuzziness parameter m define weights for the membership functions for each prototype in a manner that the greater the value of m, the smoother is the learning process. The target class membership value of neuron i for input pattern j is represented by $t_{ji} \in \{0,1\}$. Hence, the FLVQ learning rule and the membership updating rule will be:

$$\mathbf{w}_i(n+1) = \mathbf{w}_i(n) + \alpha(n)[(t_{ji})^m - (\mu_{ji})^m][\mathbf{x}_j(n) - \mathbf{w}_i(n)]; \forall i \tag{3.23}$$

$$\mu_{ji} = \left[\sum_{\ell=1}^{P} \left(\frac{d(\mathbf{x}_j, \mathbf{w}_i)}{d(\mathbf{x}_j, \mathbf{w}_\ell)} \right)^{\frac{1}{m-1}} \right]^{-1} \tag{3.24}$$

In Figure 21 the FLVQ network is described.

Figure 21 – Fuzzy-LVQ Architecture



Source: The author.

In the figure, an input layer receives the instances from the dataset. The distance layer calculates the distance between each prototype to the presented instance. Then, in the competitive layer (called MIN layer by Chung & Lee (1993)), only the closest prototype from each class is chosen to undergo the fuzzy competition. Therefore, the membership computations and parametric vector modification will be applied at the maximum number of classes in the problem or k prototypes.

3.4.5 Quantization Error (Q_E)

In prototype-based algorithms, the prototypes can be considered quantization vectors, as they represent a specific region in the input data (PERES *et al.*, 2012).

For evaluating the vector quantization in a prototype-based algorithm, a Quantization Error (Q_E) can be used. This error metric is based on the average of the distances between prototypes and the instances of the data.

$$Q_E = \frac{1}{N}\sum_{j=1}^{N} \left\| \mathbf{x_j} - \mathbf{w}_{i^*} \right\|^2 \tag{3.25}$$

where N is the number of instances, $\mathbf{x}_j$ is the jth instance, and $\mathbf{w}_{i^*}$ is a prototype with the same class as $\mathbf{x}_j$. Given a set of prototypes $\mathcal{W} = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_P\}$, $\mathbf{w}_{i^*}$ represents the closest prototype to the instance x_j. The value w can be calculated by the Equation (3.15).

3.5 Support Vector Machine (SVM)

Support Vector Machine (SVM) is a statistic-based learning method proposed by Cortes & Vapnik (1995). SVM is a well-known machine learning technique, and has been widely explored among the researchers in machine learning field (ŞTEFAN, 2012; LIU *et al.*, 2013). The learning process of SVM classifiers aims to increase the generalization capability of the model by minimizing the empirical risk (*Empirical Risk Minimization*) and the structural risk (*Structural Risk Minimization*).

In SVM classifiers, the main objective is to construct an optimal hyperplane as a decision surface, which divides the patterns from training set in a manner which maximizes the margin between data from different classes(HAYKIN *et al.*, 2009). This chapter is devoted to introducing the fundamentals of SVM-based learning. Concepts such as optimal hyperplane, separation margin, hard and soft margin SVMs, and "kernel trick" will be addressed.

3.5.1 Fundamentals of SVM classification

Consider a labeled training set $\mathcal{T} = \{(\mathbf{x}_i, d_i)\}_{i=1}^{N}$ where $\mathbf{x}_i$ is the i-th input vector and d_i is the corresponding desired class. Assuming that $\mathcal{T}$ is composed by patterns from two classes: positive and negative, represented by the subsets $d_i = +1$ and $d_i = -1$, respectively. Assuming that the training set $\mathcal{T}$ is linearly separable, the decision surface may be presented in the form of a hyperplane:

$$\mathbf{w}^T\mathbf{x} + b = 0 \tag{3.26}$$

where $\mathbf{x}$ is an input vector, $\mathbf{w}$ is an adjustable weight vector (normal to the hyperplane), and b is a bias. Note that $\mathbf{x}, \mathbf{w}$ are vectors with the same dimension D, therefore $\mathbf{x}, \mathbf{w} \in \mathbb{R}^D$.

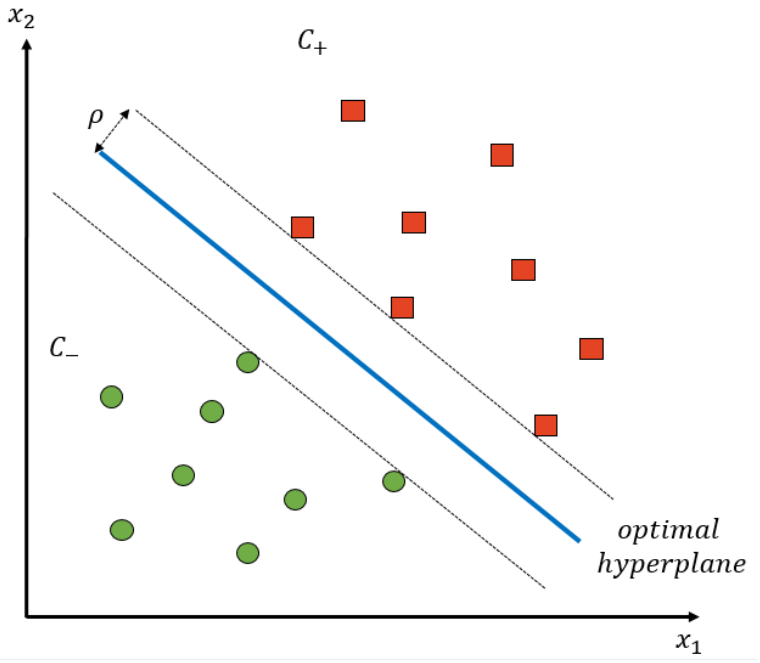
The hyperplane defined by $\mathbf{w}$ and b is subjected to the following constraints:

$$\begin{aligned} &\mathbf{w}^T\mathbf{x}_i + b \geq 0, \; then \;\; d_i = +1 \\ &\mathbf{w}^T\mathbf{x}_i + b < 0, \; then \;\; d_i = -1 \end{aligned} \tag{3.27}$$

In a linearly separable problem, there are infinite solutions available (infinite hyperplanes). Knowing that, what would be the optimal hyperplane to separate two classes? The optimal hyperplane would be the hyperplane positioned equidistantly from the classes. In other words, the optimal hyperplane will be the one that has the maximum distance in relation to the closest training samples of both classes. The separation between the hyperplane and the nearest training patterns is known as the *margin of separation*, denoted by ρ.

In SVM learning process, the main objective is to find the hyperplane that maximizes the separation margin ρ. This particular hyperplane is reffered to as the *optimal hyperplane* (HAYKIN *et al.*, 2009). Figure 22 illustrates the optimal hyperplane and the margin of separation.

Figure 22 – Optimal hyperplane for linearly separable patterns



Source: The author.

SVM-based algorithms aim to find an optimal weight vector $\mathbf{w}_o$ and bias b_o which define the *optimal hyperplane* as

$$\mathbf{w}_o^T\mathbf{x} + b_o = 0 \tag{3.28}$$

Once a SVM classifier is trained, a discriminant function can be obtained as follows

$$f(\mathbf{x}) = \mathbf{w}_o^T\mathbf{x} + b_o, \tag{3.29}$$

which provides an algebraic measure of the distance between $\mathbf{x}$ and the the optimal hyperplane.

$$\mathbf{x} = \mathbf{x}_p + r\frac{\mathbf{w}_o}{\|\mathbf{w}_o\|} \tag{3.30}$$

where $\mathbf{x}_p$ is the normal projection of $\mathbf{x}$ onto the optiomal hyperplane and r is the distance between $\mathbf{x}_p$ and $\mathbf{x}$. Notice that r is zero when $\mathbf{x}$ is on the hyperplane; r is positive if $\mathbf{x}$ is on the positive side of the optimal hyperplane and negative if $\mathbf{x}$ is on the negative side. By definition $f(\mathbf{x}_p) = 0$, as $g(\mathbf{x})$ measures distance between a point $\mathbf{x}$ to the hyperplane (HAYKIN *et al.*, 2009).

$$\begin{aligned}
f(\mathbf{x}) &= \mathbf{w}_o^T\mathbf{x} + b_o \\
&= \mathbf{w}_o^T\left[\mathbf{x}_p + r\frac{\mathbf{w}_o}{\|\mathbf{w}_o\|}\right] + b_o \\
&= \mathbf{w}_o^T\mathbf{x}_p + b_o + \mathbf{w}_o^T r\frac{\mathbf{w}_o}{\|\mathbf{w}_o\|} \\
&= 0 + r\frac{\mathbf{w}_o^T\mathbf{w}_o}{\|\mathbf{w}_o\|} \\
&= r\frac{\|\mathbf{w}_o\|^2}{\|\mathbf{w}_o\|} \\
&= r\|\mathbf{w}_o\|
\end{aligned}$$

Therefore, the distance r can be defined as

$$r = \frac{f(\mathbf{x})}{\|\mathbf{w}_o\|} \tag{3.31}$$

Note that the distance from the origin (when $\mathbf{x} = \mathbf{0}$) to the optimal hyperplane is given by

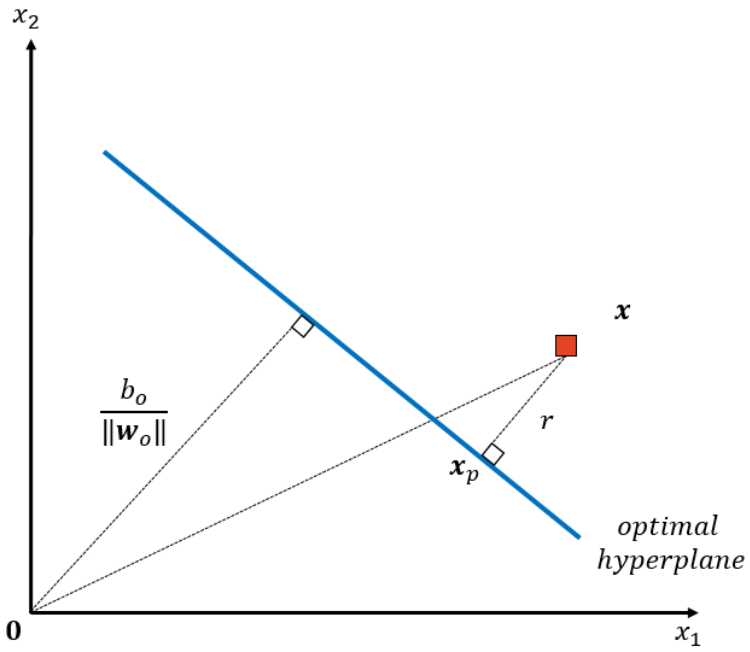
$$\begin{aligned}
r &= \frac{g(\mathbf{0})}{\|\mathbf{w}_o\|} = \frac{\mathbf{w}_o^T\mathbf{0} + b_o}{\|\mathbf{w}_o\|} \\
&= \frac{b_o}{\|\mathbf{w}_o\|}
\end{aligned}$$

Therefore, in this particular case, if $b_o > 0$, the origin is on the positive side of the optimal hyperplane; if $b_o < 0$, it is on the negative side; and if $b_o = 0$, the optimal plane passes through the origin. Figure 23 illustrates an geometric interpretation of the distance between a pattern $\mathbf{x}$ and the optimal hyperplane.

$$\rho = \frac{2}{\|\mathbf{w}_o\|} \tag{3.32}$$

Maximize the separation margin $\rho = \frac{2}{\|\mathbf{w}\|}$ is equivalent to minimize $\rho = \frac{1}{2}\left\|\mathbf{w}^2\right\|$.

Figure 23 – SVM hyperplane for classes C_1 and C_2



Source: The author.

In other words, the optimal hyperplane of separation can be described as a optimization problem where the goal is to minimize the loss function $J(\mathbf{w})$

$$J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} \tag{3.33}$$

subject to $d_i(\mathbf{w}^T\mathbf{x_i} + b) \geq 1,\ i = 1,2,...,N$ (HAYKIN *et al.*, 2009).

Note that the optimization problem defined in Equation (3.33) assumes the problem is linear separable. When the optimization problem is formulated based on the linearity assumption, the SVM classifier is known as SVM with *hard margin*.

3.5.2 Hard margin SVM classifier

The optimization problem defined in Equation (3.33) is known as primal problem. The loss function $J(\mathbf{w})$ is convex in $\mathbf{w}$, and the constraints are linear in relation to $\mathbf{w}$. Using *Lagrange* for solving this constrained problem will result in the following Lagrangian function

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{n} \alpha_i(d_i(\mathbf{w}^T\mathbf{x}_i + b) - 1), \tag{3.34}$$

The expanded form of the Lagrangian function is given by

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{n} \alpha_i d_i \mathbf{w}^T\mathbf{x}_i - b\sum_{i=1}^{n} \alpha_i d_i + \sum_{i=1}^{n} \alpha_i \tag{3.35}$$

where α_i represents the Lagrange multipliers, which are non-negative real numbers.

Deriving $L(\mathbf{w}, b, \alpha)$ in relation to $\mathbf{w}$ and b and equaling to zero, the following optimization conditions are obtained:

$$\frac{\partial L(\mathbf{w}, b, \alpha)}{\partial \mathbf{w}} = 0 \tag{3.36}$$

$$\frac{\partial L(\mathbf{w}, b, \alpha)}{\partial b} = 0 \tag{3.37}$$

resulting in

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i d_i \mathbf{x}_i \tag{3.38}$$

and

$$\sum_{i=1}^{n} \alpha_i d_i = 0 \tag{3.39}$$

respectively. The term $-b\sum_{i=1}^{n} \alpha_i d_i$ is equal 0 due to Equation (3.39). In this way, the Equation can be rewritten:

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{n} \alpha_i d_i \mathbf{w}^T \mathbf{x}_i + \sum_{i=1}^{n} \alpha_i \tag{3.40}$$

Moreover, the term $-\sum_{i=1}^{n} \alpha_i d_i \mathbf{w}^T \mathbf{x}_i$ can be simplified to $-\mathbf{w}^T\mathbf{w}$. Then, the Lagrangian equation will be

$$\begin{aligned} L(\mathbf{w}, b, \alpha) &= \frac{1}{2}\mathbf{w}^T\mathbf{w} - \mathbf{w}^T\mathbf{w} + \sum_{i=1}^{n} \alpha_i \\ &= -\frac{1}{2}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{n} \alpha_i \end{aligned} . \tag{3.41}$$

Substituting Equation (3.38) in Equation (3.41) the expression is reduced to

$$L(\alpha) = \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j d_i d_j \mathbf{x}_i^T \mathbf{x}_j \tag{3.42}$$

Note that Equation (3.42) depends only on Lagrangian multipliers. Hence, the optimization problem is defined in a dual formulation as follows

$$\begin{aligned} max\ L(\alpha) &= \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j d_i d_j \mathbf{x}_i^T \mathbf{x}_j \\ s.t \quad & \sum_{i=1}^{n} \alpha_i d_i = 0, \\ s.t \quad & \alpha_i \geq 0, i = 1, \ldots, n \end{aligned} \tag{3.43}$$

3.5.3 *Soft margin SVM classifier*

A hyperplane that separates correctly two classes rarely exists in real-world classification problems. Thus, it is necessary to adopt a specific formulation of the problem that allows some data samples to be classified incorrectly in order to preserve the generalization of the discriminant function and avoid overfitting. In other words, the discriminant function should not be excessively complex. In general, complex classification models are subject to overfitting as there is the model tends to fit the noise, mainly in problems containing discrepant samples (*outliers*). This formulation is called *soft margin* SVM, and it allows a relaxation in the constraints of the optimization problem by using threshold variables as described below:

$$d_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, i = 1, \ldots n, \tag{3.44}$$

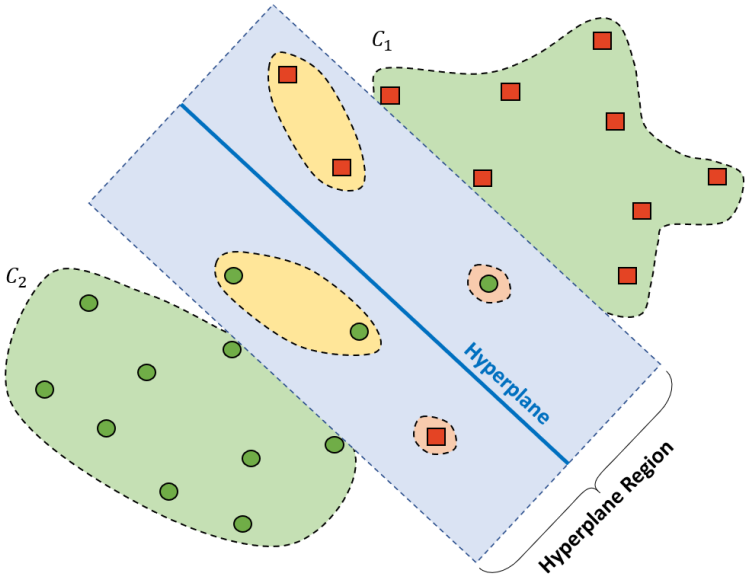
where $\xi_i \geq 0$. The variables ξ_i are known as *slack variables*.

Solving a non-linear separable problem with a linear model results in three main cases related to the classification of a pattern (or instance) $\mathbf{x}$:

1. Instance outside the hyperplane margin with correct labels (Green region)
2. Instance inside the hyperplane region with correct labels (Yellow region)
3. Instance incorrectly classified (Red region)

Figure 23 illustrates the three classification cases in non-separable problems.

Figure 24 – SVM hyperplane classification cases



Source: The author.

The three cases cited previously is closely related to the *slack variables* values.

Data samples in separable region (Green region) will have $\xi_i = 0$; for $0 < \xi_i \leq 1$, the instances $\mathbf{x}$ will be localized one the correct side, inside the hyperplane region (Yellow region). Finally, in case $\xi_i > 1$, the data sample will be incorrectly positioned inside the hyperplane region (Red region) (See Figure 23).

In order to solve the optimization problem using the *soft margin* formulation, the primal problem must be rewritten as follows

$$\begin{aligned} min\ J(\mathbf{w},\xi) &= \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}\xi_i \\ s.t \quad & y_i(\mathbf{w}^T\mathbf{x_i} + b) \geq 1 - \xi_i, \ \ i = 1,2,...,N \\ s.t \quad & \xi_i \geq 0 \ \ i = 1,2,...N \end{aligned} \tag{3.45}$$

where ξ_i are variables for permitting margin failure and the hyper-parameter C is a constant responsible to balance the trade-off between wide margin and small number of failures.

Similar to *hard margin* formulation, the solution will be defined by minimizing the Lagrangian function $L(\mathbf{w},b,\xi,\alpha,\beta$, with respect to $\mathbf{w},b$ and ξ, and maximized with respect to α and β. The Langrangian function formulated for soft margin SVM is given by

$$L(\mathbf{w},b,\xi,\alpha,\beta) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{n}\alpha_i(d_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 + \xi_i) - \sum_{i=1}^{n}\beta_i\xi_i, \tag{3.46}$$

The expanded form of the Langrangian function above is presented as

$$L(\mathbf{w},b,\xi,\alpha,\beta) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i d_i\mathbf{w}^T\mathbf{x}_i - b\sum_{i=1}^{n}\alpha_i d_i + \sum_{i=1}^{n}\alpha_i - \sum_{i=1}^{n}\alpha_i\xi_i - \sum_{i=1}^{n}\beta_i\xi_i, \tag{3.47}$$

As in the *hard margin* SVM, the solution of the problem requires that the cost-function be differentiated in relation to to $\mathbf{w},b$ and ξ and equals zero, in order to obtain the following optimization conditions:

$$\frac{\partial L(\mathbf{w},b,\xi,\alpha,\beta)}{\partial \mathbf{w}} = 0 \tag{3.48}$$

$$\frac{\partial L(\mathbf{w},b,\xi,\alpha,\beta)}{\partial b} = 0 \tag{3.49}$$

$$\frac{\partial L(\mathbf{w},b,\xi,\alpha,\beta)}{\partial \xi} = 0 \tag{3.50}$$

The above equations result respectively in the following constraints:

$$\mathbf{w} = \sum_{i=1}^{n}\alpha_i d_i\mathbf{x}_i \tag{3.51}$$

$$\sum_{i=1}^{n}\alpha_i d_i = 0 \tag{3.52}$$

$$C = \alpha_i + \beta_i \tag{3.53}$$

The simplification of Equation (3.46) from the above constraints results in the following Lagrangian equation

$$L(\mathbf{w},b,\xi,\alpha,\beta) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{n}\xi_i(\alpha_i+\beta_i) - \sum_{i=1}^{n}\alpha_i d_i \mathbf{w}^T\mathbf{x}_i - b\sum_{i=1}^{n}\alpha_i d_i + \sum_{i=1}^{n}\alpha_i - \sum_{i=1}^{n}\xi_i(\alpha_i+\beta_i) \tag{3.54}$$

It can be seen that the terms dependent on β_i of Equation (3.54) can be eliminated, resulting in

$$L(\mathbf{w},b,\xi,\alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{n}\alpha_i d_i \mathbf{w}^T\mathbf{x}_i - b\sum_{i=1}^{n}\alpha_i d_i + \sum_{i=1}^{n}\alpha_i \tag{3.55}$$

The term $-b\sum_{i=1}^{n}\alpha_i d_i$ is equal 0 due to Equation (3.52), and $-\sum_{i=1}^{n}\alpha_i d_i \mathbf{w}^T\mathbf{x}_i$ can be simplified to $-\mathbf{w}^T\mathbf{w}$. Then, the Lagrangian equation will be

$$\begin{aligned} L(\mathbf{w},b,\alpha) &= \frac{1}{2}\mathbf{w}^T\mathbf{w} - \mathbf{w}^T\mathbf{w} + \sum_{i=1}^{n}\alpha_i \\ &= -\frac{1}{2}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{n}\alpha_i \end{aligned} . \tag{3.56}$$

Substituting Equation (3.51) in Equation (3.56) the expression is reduced to

$$L(\alpha) = \sum_{i=1}^{n}\alpha_i - \sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j d_i d_j \mathbf{x}_i^T\mathbf{x}_j \tag{3.57}$$

Consequently, the Equation (3.57) depends only on Lagrangian multipliers. Hence, the optimization problem is defined in a dual formulation as follows

$$\begin{aligned} max\ L(\alpha) &= \sum_{i=1}^{n}\alpha_i - \sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j d_i d_j \mathbf{x}_i^T\mathbf{x}_j \\ s.t \quad & \sum_{i=1}^{n}\alpha_i d_i = 0, \\ s.t \quad & 0 \leq \alpha_i \leq C, i = 1,\ldots,n \end{aligned} \tag{3.58}$$

3.5.4 Kernel Function

The kernel function is essential for SVM-based algorithms. Non-linear transformation of the input to a higher dimensional feature space assists SVM algorithm in non-linear separable cases, since the data points are more easily classified in higher dimensions (HAYKIN *et al.*, 2009). Thus, in order to obtain a linearly separable problem, the input space $\mathbf{x} \in \mathbb{R}^d$ is mapped to a feature space $\Phi(\mathbf{x}) \in \mathbb{R}^q$, such that $q > d$.

Hence, the duty of kernel functions is to take inner product of the mapped data points. Equation (3.59) defines the kernel function, which is commonly called the *kernel trick*, because it avoids the need to find the mapping function $\Phi(\cdot)$

$$K(\mathbf{x},\mathbf{y}) = \Phi(\mathbf{x})^T \Phi(\mathbf{y}). \tag{3.59}$$

In this sense, it is possible to redefine the Equations (3.43) and (3.58) referring to the classifier SVM with *hard margin* and with *soft margin*. Equations (3.60) and (3.61) present this redefinition, after applying the *kernel trick*, for *hard margin* and *soft margin*, respectively.

$$\begin{aligned} max\ L(\alpha) &= \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j d_i d_j K(\mathbf{x}_i,\mathbf{x}_j) \\ s.t \quad & \sum_{i=1}^{n} \alpha_i d_i = 0, \\ s.t \quad & \alpha_i \geq 0, i = 1,\ldots,n \end{aligned} \tag{3.60}$$

$$\begin{aligned} max\ L(\alpha) &= \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j d_i d_j K(\mathbf{x}_i,\mathbf{x}_j) \\ s.t \quad & \sum_{i=1}^{n} \alpha_i d_i = 0, \\ s.t \quad & 0 \leq \alpha_i \leq C, i = 1,\ldots,n \end{aligned} \tag{3.61}$$

Using the kernel function avoids the calculation of transforming the input data to a higher dimension. Kernel functions affect in classification performance. Hence, choosing the proper kernel function is a important task for training SVM-based algorithms and usually requires expert knowledge about the data. Some of the most common kernels used in the literature are presented in the table bellow. Gaussian kernel is also known as Radial Basis Function (RBF) kernel.

Table 1 – Examples of Kernel functions used in SVM algorithms (HAYKIN *et al.*, 2009)

Kernel	$K(\mathbf{x}_i,\mathbf{x}_j)$	**Observations**
Linear	$\mathbf{x}_i^T \cdot \mathbf{x}_j$	-
Polynomial	$(\mathbf{x}_i^T \cdot \mathbf{x}_j + 1)^p$	Parameter p represents the degree of the polynomial
Gaussian	$exp\big(-\frac{1}{2\sigma^2}\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2\big)$	Parameter σ^2 is defined by the user
Sigmoidal	$tanh(\beta_0 \mathbf{x}_i \cdot \mathbf{x}_j + \beta_1)$	Used only for several β_0 and β_1 values

3.5.5 Classifying approaches

As SVMs were initially designed to perform binary classification. Therefore, it is necessary to apply techniques for adapting SVM-based classifiers for solving Multiple-class classification problems. In the literature, two commonly techniques prevail: *one-against-all* and *one-against-one*. These methods are used to solve the multiple classification problem using several binary SVM-classifiers.

3.5.5.1 One-Against-All

This method consists on creating one model for each class. For example, in a problem consisted by a set of classes $\mathcal{C} = \{c_1, c_2, ...c_k\}$, k binary classifiers are built, and each model m_i is responsible for classifying a pattern $\mathbf{x}$ as positive or negative for the class c_i. A common problem in this approach is when more than one model classifies a pattern $\mathbf{x}$ as positive. In this case, only one model is correct and the others misclassified $\mathbf{x}$ (false positive). A solution for dealing with this problem is to compare the models among them for deciding which one is classifying correctly. One common used strategy is referred as *Win Max*, where the output selected is the class which have the greatest value calculated.

3.5.5.2 One-Against-One

In this classification approach, for a problem with c classes, $\frac{c(c-1)}{2}$ classifiers are built. Each classifier is trained using data of only two classes. Therefore, all possible pairwise classifiers are evaluated (WANG; XUE, 2014). Testing each classifier to an instance $\mathbf{x}$ would give one vote to the winning class. The class with more vote is assigned to $\mathbf{x}$.

3.6 Summary

In this chapter, a naive method known as k-NN, considered a benchmark algorithm for classifying time series,has been described. Furthermore, the fundamentals on Multi-Layer Perceptron (MLP), Self-Organizing Map (SOM), Learning Vector Quantization (LVQ) and Support Vector Machine (SVM) are discussed. Moreover, different variations of LVQ-based algorithms are presented.

In Chapter 4, the concept of Adaptive-LVQ is introduced. In addition, three proposed LVQ-based variations are detailed: dynamic learning rate LVQ (dLVQ) and adaptive clustering LVQ (ALVQ), as well as the combination of dLVQ and ALVQ (dALVQ).

Finally, a framework for applying these adaptive characteristics to any LVQ-based is proposed. Thus, the adaptive characteristics described in the proposed method can be combined with several LVQ-based variations, including LVQ1, LVQ2 and LVQ3 proposed by Kohonen (1997), and a Fuzzy-LVQ version, introduced by Chung & Lee (1993).

4 NOVEL APPROACHES FOR ADAPTIVE LVQ CLASSIFIERS

This chapter is devoted to present three novel approaches of Adaptive LVQ-based classifiers designed in this work:

- **ALVQ-SOM:** Consist in a hybridization of an Adaptive-Learning Vector Quantization with Self-Organizing Maps. In this approach, the inclusion of prototypes are based on using SOM algorithms applied on misclassified samples, during training.
- **dLVQ:** The driven-LVQ is an Adaptive-Learning Vector Quantization, which adaptation consist in identifying low representative prototypes and changing their learning rate for creating chance of convergence during training process. The idea behind the dLVQ is to boost prototypes of low representativity with the aim of converging to better positions
- **dALVQ-SOM:** It is a variation of ALVQ-SOM where the prototypes included with SOM algorithms also changes their learning rate for improving the convergence during training.

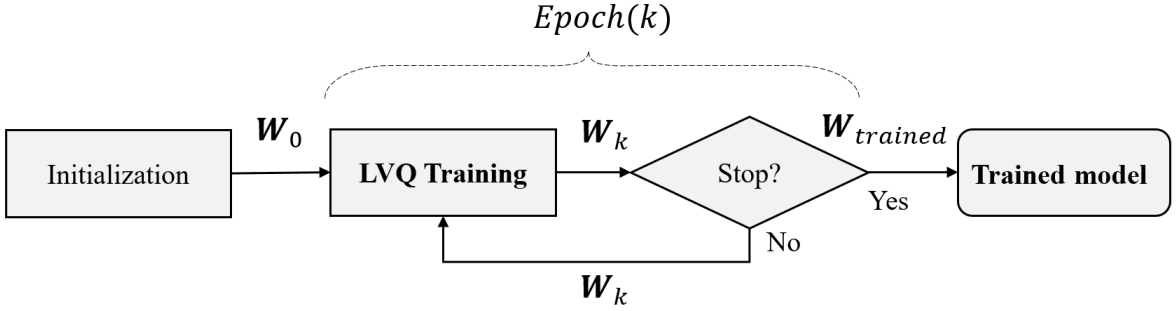
Initially, the basic concepts of adaptive-LVQ are introduced by describing the differences between classic algorithms and adaptive versions. Then, following the ideas introduced by Grbovic & Vucetic (2009), an adaptive LVQ-based algorithm called ALVQ-SOM is proposed, followed by two variations: dLVQ and dALVQ-SOM. Finally, the fundamentals of the proposed methods are discussed, by focusing mainly on their operation, described in flowcharts and pseudo-codes.

For describing the fundamentals on the methods, focusing mainly on its operation, illustrated in flowcharts and pseudo-codes.

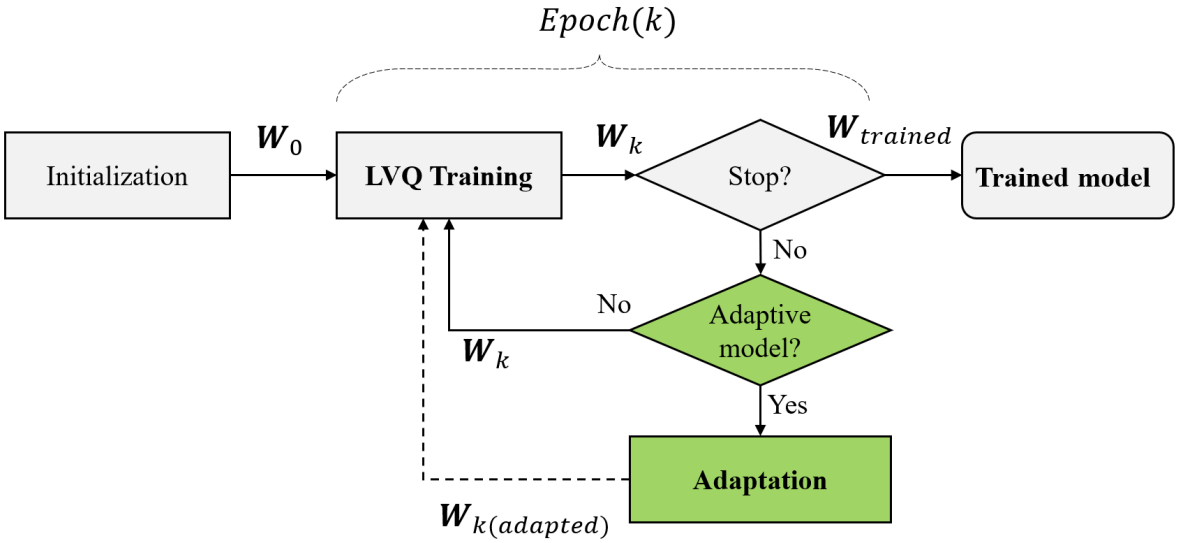
4.1 Adaptive LVQ (ALVQ)

Adaptive LVQ (ALVQ) neural networks is a category of LVQ-based algorithms that have the capacity of making adjustments in their architecture during training in order to improve the classification performance. In general, the adaptation consists of two basic operations: prototypes inclusion and prototype removal. Strategies of including and/or removing prototypes determines the Adaptive-LVQ performances. Moreover, in adaptive algorithms, it is necessary to adopt a decision rule for determining when an adaptation should occur. Adaptive models of classification with high adaptability can suffer from problems such as oscillations and non-convergence of learning. Figure 25 presents two flowcharts that describe the difference between traditional (non-adaptive) LVQ and adaptive LVQ.

Figure 25 – Classic and Adaptive LVQ training frameworks



(a) Classic LVQ training framework



(b) Adaptive-LVQ training framework

Source: The author.

A traditional LVQ algorithm, or non-adaptive, will always remain its structure. Note, in Figure 25(a), that there is no step for adaptation. Therefore, only the prototype weights are varying during training. In contrast, an adaptive LVQ may changes in terms of inclusion or removal of prototypes, allowing the ANN to improve its performance. In Figure 25(b), the adaptive steps are included and highlighted in green. First, an evaluation is performed on the neural network in order to verify the need of adaptation. It is important to realize that adaptation should only occur if needed. Once an adaptation is required, the adaptive method must include, remove or modify some prototypes based on a strategy that improves the performance of the neural network according to some criteria. In this work, the main criterion is the improvement in classification performance. An Adaptive-LVQ can be divided in two main stages:

- **Training:** During the training stage, the instances from the training dataset is presented to the neural network, and the prototypes are adjusted based on a learning rule from a LVQ-based variation.

- **Adaptation:** After completing an epoch, the resulted LVQ-network is evaluated in order to verify the need for adaptation. In this context, adaptation mean modify the LVQ structure by including, removing or modifying prototypes in order to improve the neural network's performance. If a specific criterion for adaptation is satisfied, an algorithm makes the necessary modifications accordingly to criterion that aim to improve the network performance).

Figure 25(b) presents a flowchart describing the process of training an Adaptive-LVQ model. First, the prototypes weights are initialized. In this work, the Kohonen's Self-Organizing Map is used for initializing the prototypes. Afterwards, the LVQ variation is executed during a whole epoch k. In the end of each epoch, the algorithm verifies the need for adaptation. Depending on this decision, the network is adapted ($\mathbf{W}_{k(adapted)}$) or not ($\mathbf{W}_k$). Then, the cycle restarts in the next epoch. After each epoch, a stop criterion is verified, once it is satisfied, the algorithm finishes, returning the trained model ($\mathbf{W}_{trained}$). An example of an Adaptive-LVQ was proposed by Grbovic & Vucetic (2009), and it will be discussed next section.

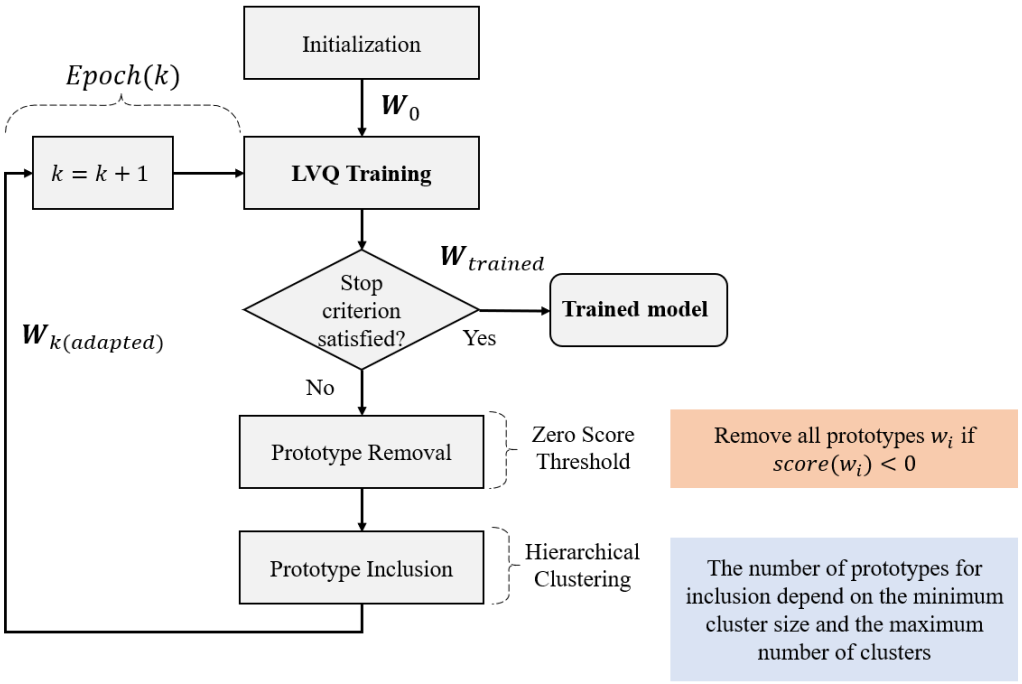
4.2 ALVQ-GV

The Adaptive-LVQ proposed by Grbovic & Vucetic (2009) has contribute as an inspiration in designing the algorithms proposed in this master thesis, and for simplicity, it will be referred as Grbovic-Vucetic Adaptive LVQ (ALVQ-GV). Hence, the ALVQ-GV will be briefly discussed. Figure 26 presents a flowchart describing the main step during ALVQ-GV operation.

In ALVQ-GV, after each epoch an adaptation occurs. Therefore, the authors have not applied any decision rule, regarding the need for adaptation. The process of adaptation in their work is based on firstly removing prototypes, by applying a threshold in a score calculated during training. Each prototype $\mathbf{w}_i$ has a score, and the prototypes which present negative scores at the end of an epoch are removed.

For inclusion, the new prototypes are generated by applying a Hierarchical Clustering method on the misclassified data. In ALVQ-GV, an variable called *Budget* (B) was used for determining the maximum number of prototypes supported by the neural network. Therefore, at the end of an epoch k, the number of prototypes available for inclusion is calculated by $P_{New} = B - P_k$, where P_k is the actual number of prototypes. If $P_{New} > 0$, P_{New} prototypes will be included. The variable B is useful to prevent too much growth of the network, also avoiding overfitting. In the next subsections, the inclusion and removal strategies are discussed in detail.

Figure 26 – Flowchart operation of the Adaptive Learning Vector Quantization proposed by Grbovic & Vucetic (2009) (ALVQ-GV)



Source: The author.

4.2.1 ALVQ-GV strategy for prototype inclusion

The ALVQ-GV uses hierarchy clustering applied to all misclassified samples in order to decide the new prototypes that will be included in the network. The number of prototypes to be included is controlled by two variables: *max cluster*, on the number of clusters, and *min cluster size*, on the cluster size (GRBOVIC; VUCETIC, 2009). The quantity of new prototypes is limited by a variable called *Budget* (*B*). This variable define the maximum prototype quantity supported by the network.

The main increase in time complexity in ALVQ-GV is due the performance of prototype inclusion function. As it uses a hierarchical clustering with average linkage, the asymptotic time complexity is given by:

$$\Theta\left(\sum_{i=1}^{N_C} n_i^2 \log n_i\right) \tag{4.1}$$

where n_i is the number of misclassified data samples from ith class, and N_C is the number of classes.

4.2.2 ALVQ-GV strategy for prototype removal

During removal, the ALVQ-GV algorithm calculate a score based on the following equation:

$$S_j^{ALVQ-GV} = A_j - B_j + C_j \tag{4.2}$$

where A_j counts how many times a prototype p_j correctly classified was not moved toward the sample; B_j counts how many times a prototype was moved far from for a misclassified sample and C_j counts how many times a prototype was moved toward the sample when correctly classified. A prototype p_j is removed when its $score_j$ is negative (GRBOVIC; VUCETIC, 2009).

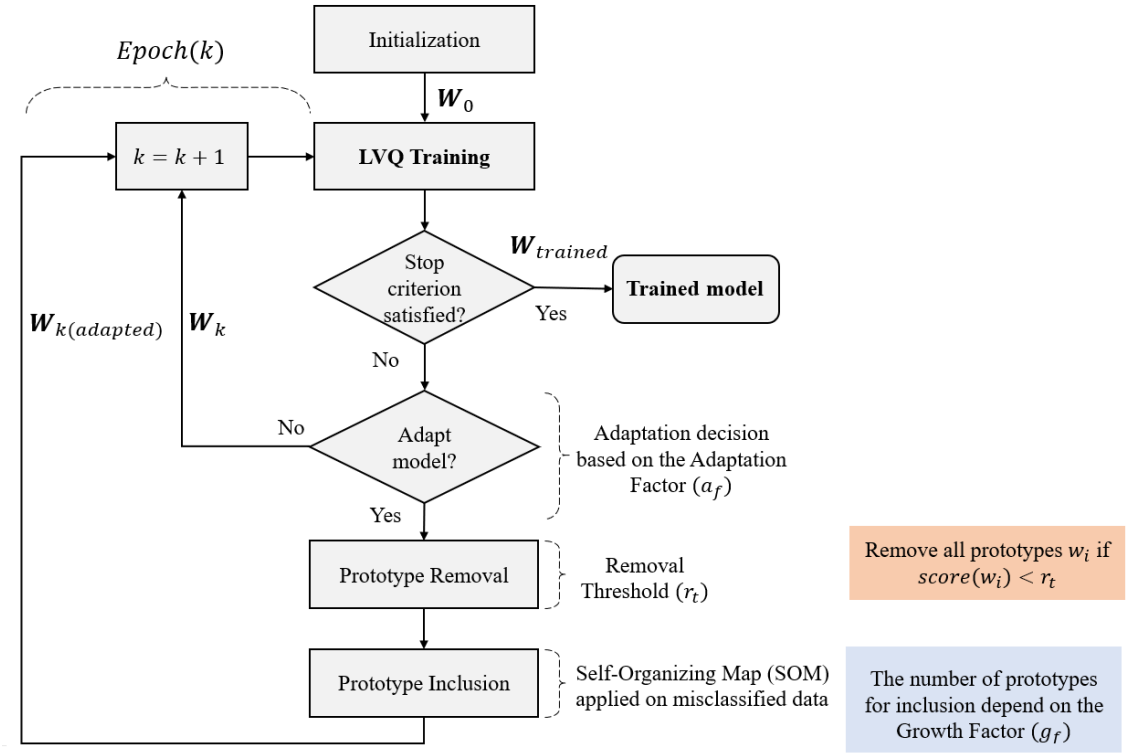
Regarding the time complexity, as the removal is based only on scores calculated over prototypes, its cost will mainly depend linearly on the number of prototypes P. The removal cost is based on $\Theta(N_S \cdot P \cdot N_E)$, for increments of A, B and C, and $\Theta(P)$, for verification for removing or not based on a threshold. As these operations are not time consuming, it can be concluded that removal does not affect significantly the computational performance.

4.3 Proposed Adaptive-LVQ (ALVQ-SOM)

Based on the algorithm ALVQ-GV, a novel variation has been designed with several modifications that may improve the overall classification performance. In general, the adaptive characteristics implies the ability of making changes in the network's structure by including or removing prototypes (codebooks or neurons). Our proposed algorithm, Adaptive LVQ with Self-Organizing Maps (ALVQ-SOM), consist of using Self-Organizing Maps (SOM) for generating new prototypes. The unsupervised method Kohonen's Map (KM-SOM) (KOHONEN, 1990) is proposed for inclusion of new prototypes in the neural network. As for removing, a similar strategy as the one adopted in ALVQ-GV is used, with several modifications. Figure 27 presents a flowchart describing the main steps during ALVQ-SOM operation.

In ALVQ-SOM, at the end of each epoch an adaptation may or not occurs. The decision will depend on a hyper-parameter called Adaptation Factor (a_f). The justification for including this hyper parameter is that frequent adaptation occurrences can result in an undesired oscillatory behavior of neural network training. The adaptation is composed by removal and inclusion of prototypes, respectively. The removal process in ALVQ-SOM consist in removing prototypes, at the end of each epoch, by applying a threshold in a score calculated during training. Any prototype $\mathbf{w}_i$ which has a score $score(\mathbf{w}_i) < r_t$ is removed.

Figure 27 – Flowchart describing the operation of Adaptive Learning Vector Quantization combined with Self-Organizing Map (ALVQ-SOM)



Source: The author.

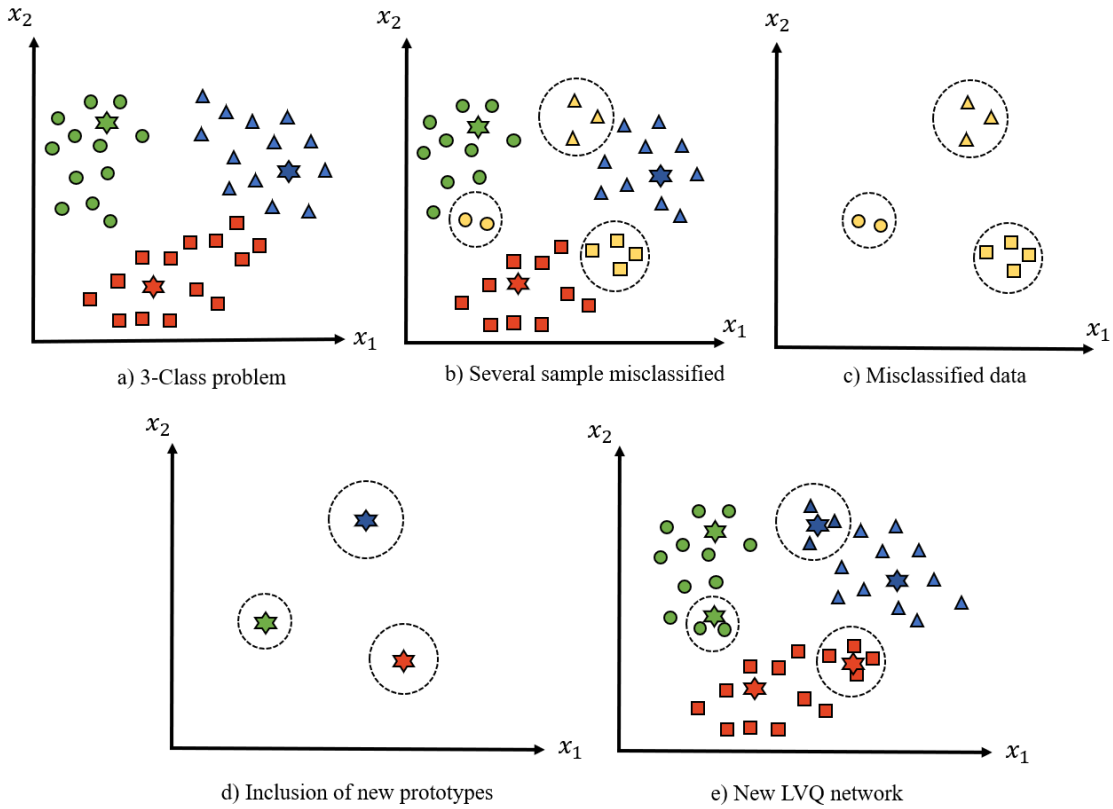
For inclusion, the new prototypes are generated by applying a Self-Organizing Map (SOM) method on the misclassified data. In ALVQ a variable called *Budget* (B) is employed for determining the maximum number of prototypes supported by the neural network, as used previously in ALVQ-GV. In the next subsections, the ALVQ-SOM is discussed in further details.

4.3.1 Proposed strategy for prototype inclusion

The strategy adopted for including new prototypes in a ALVQ-SOM neural network consists in using a SOM algorithm applied on misclassified samples. Note that any other SOM algorithm can be used. However, in this work only KM-SOM was employed. For simplicity, and in order to reinforce that any SOM algorithm can be used, the step of the proposed algorithm involving using KM-SOM will be described generally by SOM.

Figure 28, illustrates the process of including three prototypes, based on the misclassified data. The neural network growth is restricted by a variable which define the maximum number of prototypes supported by the neural network, known as Budget (B). Therefore, the number of prototypes will not overstep the pre-defined architecture size. Two approaches for including prototypes have been defined: *All classes at once* and *One class at a time*.

Figure 28 – Process of including three new prototypes to an adaptive LVQ



Source: The author.

Before getting into details of these approaches, a hyper-parameter named Growth Factor (g_f) must be introduced. This parameter is multiplicative factor that will be involved in the calculation of how many prototypes is reasonable to include to our LVQ network, in order to represent the misclassified points. This parameter will be used similarly in the two inclusion approaches. Note that the number of prototype will always depend on the number of misclassified points. Therefore, the better the classifier, the less prototypes will be included. The Kohonen's Map algorithm provides a solution for better representing poorly classified regions, with the cost of increasing the computational effort.

4.3.1.1 All classes at once (v1)

In this approach, all misclassified data is considered as a single data set D. Hence, the SOM algorithm is applied in all misclassified data, and then the class are assigned to each prototype, based on their distance to misclassified data. Consider N_{Miss} the total number of misclassified points. The number of new prototypes ($N_{P_{new}}$) to be included will be calculated by:

$$N_{P_{new}} = N_{Miss} \cdot g_f \tag{4.3}$$

Algorithm 3 presents the implementation steps for the process of inclusion using the *all classes at once* strategy.

Algorithm 3: ALVQ-SOM include prototype function (All classes at once approach)

Input: $LVQ_{old}, X_{Miss}, N_{classes}, g_f$; // LVQ network before adaptation, Misclassified samples, Number of classes, growing factor

Output: LVQ_{new} ; // LVQ network after adaptation

$N_{Miss} = size(X_{Miss})$;

$N_{P_{new}} = N_{Miss} \cdot g_f$; // Number of prototypes to be included

$p_{new} = SOM(X_{Miss}, N_{P_{new}})$; // List of new prototypes to be included

$p_{new} = findClasses(p_{new}, X_{Miss})$; // Find for each prototype a proper class

LVQ_{new} = IncludeNewPrototypes(p_{new}, LVQ_{old}); // Include new prototypes to the original LVQ network

Notice that in Algorithm 3 there is a function called *findClass (·)*. This function represents the procedure for finding appropriate class labels for the new prototypes to be added. This process is quite common when unsupervised algorithms are applied in classification problems. A typical strategy for labeling prototypes is based on the construction of clusters for each prototype. In each cluster, a verification is performed to determine the most frequent class in the cluster. The most frequent class in the cluster will be the one with the largest number of samples, and this will be the class assigned to the prototype.

4.3.1.2 One class at a time (v2)

In this approach, each class has a set of misclassified data. Hence, the SOM algorithm is applied in each data set D_i, separately. Consider $N_{Miss}^{(i)}$ the total number of misclassified points for i-th class. The number of new prototypes to be included of class i will be calculated by:

$$N_{P_{new}}^{(i)} = N_{Miss}^{(i)} \cdot g_f \tag{4.4}$$

Hence, the greater the presence of misclassified instances of a class i, the greater the number of new prototypes that will be included to represent this class. Algorithm 4 presents the implementation steps for including prototypes using *one class at a time* approach.

Algorithm 4: ALVQ-SOM include prototype function (One class at a time approach)

Input: $LVQ_{old}, X_{Miss}, N_{classes}, g_f$; // LVQ network before inclusion, Misclassified samples, Number of classes, growing factor

Output: LVQ_{new} ; // LVQ network after inclusion of prototypes

for $i = 1$ ***to*** $N_{classes}$ **do**

$N_{Miss}^{(i)} = size(X_{Miss}^{(i)})$;

$N_{P_{new}}^{(i)} = N_{Miss}^{(i)} \cdot g_f$; // Number of prototypes to be included

$p_{new}^{(i)} = SOM(X_{Miss}^{(i)}, N_{P_{new}}^{(i)})$; // List of new prototypes of class i to be included

LVQ_{new} = IncludeNewPrototypes(p_{new}, LVQ_{old}); // Include new prototypes to the original LVQ network

end

4.3.2 Proposed strategy for prototype removal

The prototype removal from the ALVQ-SOM is based on calculating a score for each prototype (Equation (4.5)), which can quantify how useful a prototype is for a LVQ network. A prototype p_j will be removed when its score ($S_j^{ALVQ-SOM}$) is lower than a removal threshold r_t. Low scores may indicate that a prototype is degrading the model or it is not contributing significantly to the classification performance. For example, prototypes that frequently classify incorrectly instances will present low scores, hence they must be removed.

$$S_j^{ALVQ-SOM} = A_j - B_j + C_j \tag{4.5}$$

where A_j counts how many times a prototype j classified correctly an instance and was not moved. B_j counts how many times the prototype j was moved away from a misclassified sample. C_j counts how many times the $j-th$ prototype was moved toward a correctly classified sample.

A_j and C_j quantifies the contribution made by the j-th prototype to the classification model performance. Hence, large values of A and C indicates that a certain prototype is highly significant, as it has correctly classified a sample in many iterations, during an epoch. Prototypes with low values of A and C may still be important, depending on the B_j value.

B_j quantities the degradation caused by a j-th prototype to the classification model's performance. Large values of B indicates that a certain prototype has frequently misclassified samples, as it was multiple times chosen as the winner, but it belonged to a different class of the samples. Algorithm 5 presents the pseudo-code for removing prototypes in ALVQ-SOM.

Note that the prototype removal process differs from the ALVQ-GV algorithm by the adoption of a threshold that can be used to tuning the tolerance in accepting low representative prototypes. In addition, it is important to notice that the term A is used only for the adaptive variables of ALVQ2-SOM and ALVQ3-SOM. The adaptive variation ALVQ1-SOM and AFLVQ-SOM only calculate the score based on terms A and B. The same goes for the ALVQ-GV variations.

Algorithm 5: ALVQ-SOM removal prototypes function

Input: LVQ_{old}, N_P, r_t ; `// LVQ network before removal, Number of prototypes, removal threshold`

Output: LVQ_{new} ; `// LVQ network after removal of prototypes`

for $j = 1$ ***to*** N_P **do**

  $S_j = A_j - B_j$; `// Score to evaluate the quality of a prototype`

  if $S_j < r_t$ **then**

    LVQ_{new} = removePrototype(j, LVQ_{old}) ; `// Remove prototype` j

  end

end

4.3.3 ALVQ-SOM implementation

The ALVQ-SOM constitutes a framework for transforming a classical LVQ learning algorithms into an adaptive version. Therefore, the proposed algorithm still require the learning rule of a classic LVQ algorithm. Algorithm 6 presents an example of pseudo-code which implements an ALVQ-SOM classifier using LVQ1 learning strategy.

4.3.4 ALVQ-SOM Hyper-parameters

In order to design a LVQ-based classifier, several parameters must be previously defined, depending on the LVQ variation. For any LVQ-based algorithm, parameter such as number of prototypes (P) and initial learning rate (α_0) are chosen by the designer. Similarly, ALVQ-SOM architecture requires tuning other three hyper-parameters : growth factor (g_f), adaptation factor (a_f) and removal threshold (r_t). In order to properly choose these parameters, it is important to understand how they can affect in training a LVQ neural network.

Algorithm 6: ALVQ1-SOM pseudo-code

Input:

α_0 : Initial learning rate
$\mathbf{X}$: Training input samples
$\mathbf{Y}$: Desired output (classes)

Output:

W_{opt} : Optimal weights (All prototypes and their respective weights)

1. Initialization

(1.1) Initialize vector $\mathbf{w}_i$ with random values from training set
(1.2) Initialize the learning rate $\alpha(1) = \alpha_0$
(1.3) Initialize the rewarding and penalty scores $A_i = 0$ and $B_i = 0, \forall i$

2. Training

while *(Termination criteria not satisfied)* **do**
  for $n = 1$ ***to*** N **do**
    (2.1) Select an input pattern $\mathbf{x}(n)$ and its class $\mathbf{y}(n)$ from training set $\mathbf{X}$
    (2.2) Find the closest prototype from pattern $\mathbf{x}(n)$ (Winner prototype)
      $i^*(n) = argmin \|\mathbf{x}(n) = \mathbf{w}_{i^*}(n)\|, \forall i$
    (2.3) Adjust the winner neuron's synaptic weights
    if $class(\mathbf{w}_{i^*}(n)) == \mathbf{y}(n)$ **then**
      $w_{i^*(n)}(n+1) = w_{i^*(n)}(n) + \alpha(n) * (x(n) - w_{i^*(n)}(n))$; `// Move prototype towards the pattern` $x(n)$
      $A_{i^*(n)} = A_{i^*(n)} + 1$; `// Increment the winner's rewarding score`
    else
      $w_{i^*(n)}(n+1) = w_{i^*(n)}(n) - \alpha(n) * (\mathbf{x}(n) - w_{i^*(n)}(n))$; `// Move prototype away from the pattern` $\mathbf{x}(n)$
      $B_{i^*(n)} = B_{i^*(n)} + 1$; `// Increment the winner's penalty score`
    end
    (2.4) Update learning rate
    $\alpha = \alpha_0 * (1 - \frac{n}{N})$
  end
  3. Adaptation
  (3.1) Verify the need for adaptation based on a_f
  if *adaptation is true* **then**
    (3.2) Remove prototypes which have score lower than r_t
    (3.3) Include prototypes based on g_f and in the total of misclassified samples
  end
end

4.3.4.1 Growth factor (g_f)

This parameter, combined with the number of misclassified samples (N_{Miss}), controls the number of prototypes $N_{P_{New}}$ included during adaptation. Depending on g_f value, a ALVQ-SOM may include excessively the number of prototypes, or may included none, as it still depends on N_{Miss}.

In order to avoid interpolation, $g_f \in [0,1]$. Therefore, that maximum number of prototype to be included, during adaptation, is the number of misclassified points itself, as the minimum number of new prototypes is zero. Low values of g_f represent a method that will tolerate more error during training. As large values of g_f tend to overfitting. Regarding time complexity, the larger g_f the greater the computational cost.

4.3.4.2 Removal threshold (r_t)

Removal threshold hyper-parameter controls when a prototype should be removed. For negative values of r_t, only poor prototypes are removed. The lower the r_t, the more flexible the algorithm will be. In order words, r_t may work as a tolerance threshold for deciding among poor prototypes which one of them may be acceptable to remain in the network.

For positive values ($r_t > 0$), the ALVQ-SOM tend to remove prototypes that contribute with the overall classification performance, even if slightly. The greater r_t the more strict will be the algorithm in selecting prototypes. In other words, for large positive values of r_t, only extremely representative prototypes will be selected during training. As for small positive values of r_t, fairly representative prototypes will be considered. It is important to highlight that high values of the removal threshold can cause underfitting, since the definition of unreachable values of r_t will result in the removal of many or even all prototypes during adaptation, causing oscillation and non-convergence of learning.

Note, that r_t can decide the number of prototypes that remain in the network. For example, consider a binary classification problem. Let a dataset $\mathcal{D}$ be composed by 50 samples for each class. If an ALVQ-SOM algorithm is configured with $r_t = 50$, the algorithm will try to find a prototype which present a score equals or greater than 50. This can only be achieved if the network have only 2 prototypes, one for each class. If the algorithm find these prototypes, it is guaranteed they will be positioned in a way that will maximize the training classification performance.

4.3.4.3 Adaptation factor (a_f)

Adaptation factor is a parameter which determines how often an adaptation should occur. In other words, a_f defines the intervals that an adaptation is taking place. For example, if $a_f = 10$ then every ten epochs, one adaptation occurs. Therefore, $a_f \in [1, N_{Epochs}]$, where N_{Epochs} is the number of epochs.

For $a_f = 1$, at the end of each epoch, an adaptation occurs. In contrast, for $a_f = N_{Epochs}$, there will be only an adaptation at the last epoch. In this case, it is recommended to ignore this adaptation, as there will be none epoch for letting the new prototypes to converge and to be trained properly. Therefore, for $a_f = N_{Epochs}$, the ALVQ-SOM is equivalent to a non-adaptive LVQ.

In summary, a_f is inversely proportional to the number of adaptations. Thus, the lower the a_f, the more adaptations will occur and these adaptations increase the exploration of new prototypes. However, frequent adaptations will prevent the prototype to be adjusted by the learning rule, which may cause the prototype to not converge to a proper position. As more adaptations will occur, the exploration of new prototypes will increase, raising the chance of finding a good prototypes. However, multiple adaptations will allow less iterations for the prototypes to converge. For example, if a set of new prototypes are included, they may not converge to satisfactory position during one epoch.

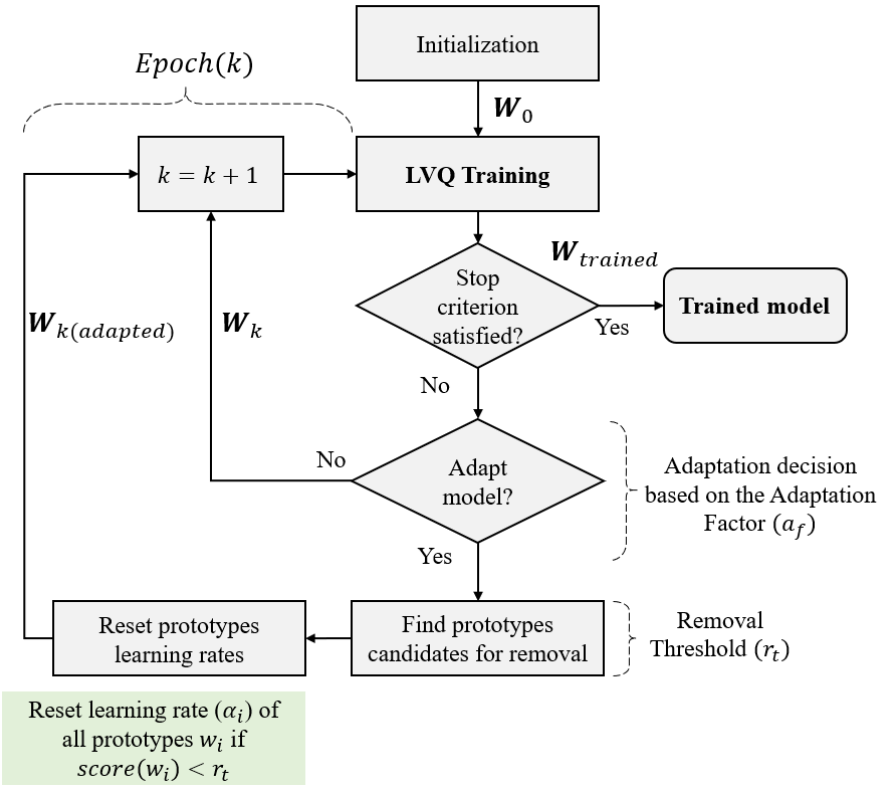
Considering the worst case scenario, where every epoch the new included prototypes will not converge. On the other hand, large values of a_f reduce the exploration, but allows the prototype to be adjusted by the learning rule. The problem of low values of a_f is that newly included prototypes need to be adjusted, and it may take more than one epoch.

4.4 Driven-LVQ (dLVQ)

Driven-LVQ (dLVQ) is a specific variation of Adaptive-LVQ-based method which has the capability of varying the learning rate of poorly trained prototypes. The dLVQ method uses the same strategy for qualifying the importance of a prototype as the algorithm ALVQ-SOM (See section 4.3.2). In ALVQ-SOM, when a prototype is qualified as poorly trained, it is removed. In dLVQ, instead of removing prototypes, new learning rates (higher than before) are assigned to low score prototypes, so they can be adjusted significantly and converge to a better location.

Figure 29 illustrates the flowchart framework for training a classifier based on dLVQ. Note that in this approach there is no prototype inclusion, and the removal step is replaced by a process of changing the learning rates of low quality prototypes.

Figure 29 – Driven-Learning Vector Quantization (dLVQ) flowchart



Source: The author.

In dLVQ, each adaptation creates a new generation. Prototypes that were modified in a specific generation will have the same learning rate. Therefore, each generation will have a specific learning rate.

$$\alpha = \begin{pmatrix} \alpha^{(0)} & \alpha^{(1)} & \cdots & \alpha^{(G)} \end{pmatrix} \tag{4.6}$$

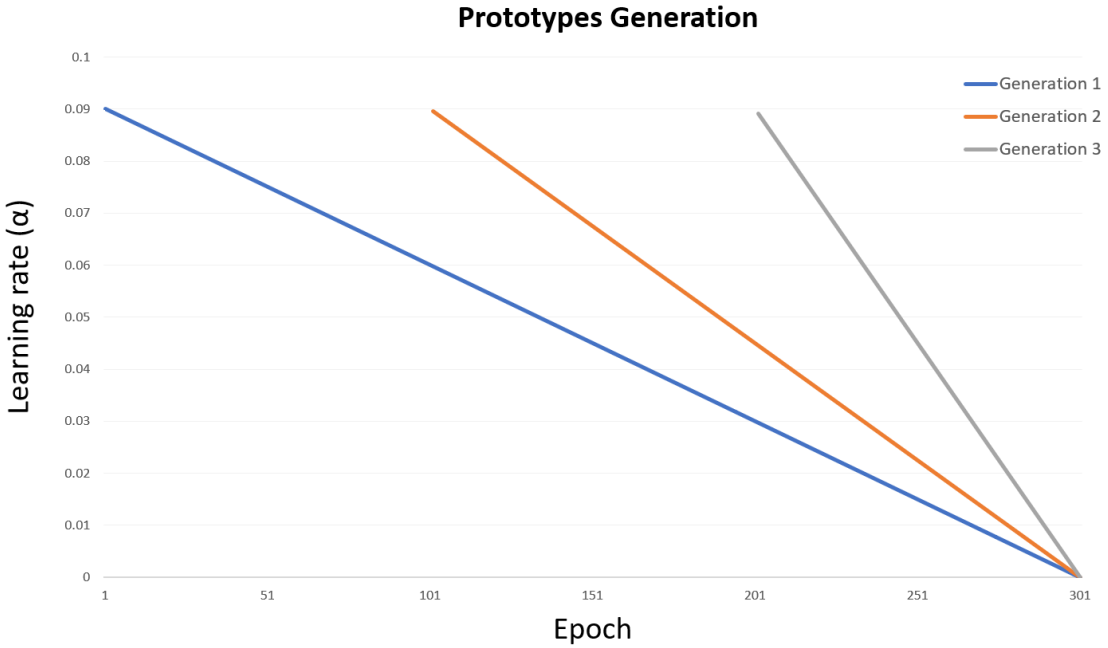
where G is the last generation created. Note that $\alpha^{(0)}$ is the initial learning rate of the network, and $\alpha^{(i)}$ is the learning rate of prototypes generated in i-th generation.

It is important to notice that each learning rate still decreases monotonically. However, they must decrease accordingly to each generation. Therefore, the linearly decreasing learning rate for the ith generation would be:

$$\alpha^{(i)}(n) = \alpha(0) \cdot (1 - \frac{n}{N}) \tag{4.7}$$

For example, in Figure 30 there are three different learning rates for different generations of prototypes. Note that at the blue line, the LVQ network training starts with an initial learning rate, which decreases linearly over the epochs. Then, at the epoch 100 an adaptation occurs and a new generation is created.

Figure 30 – Generation-based learning decay in Driven Adaptive Learning Vector Quantization (dALVQ)



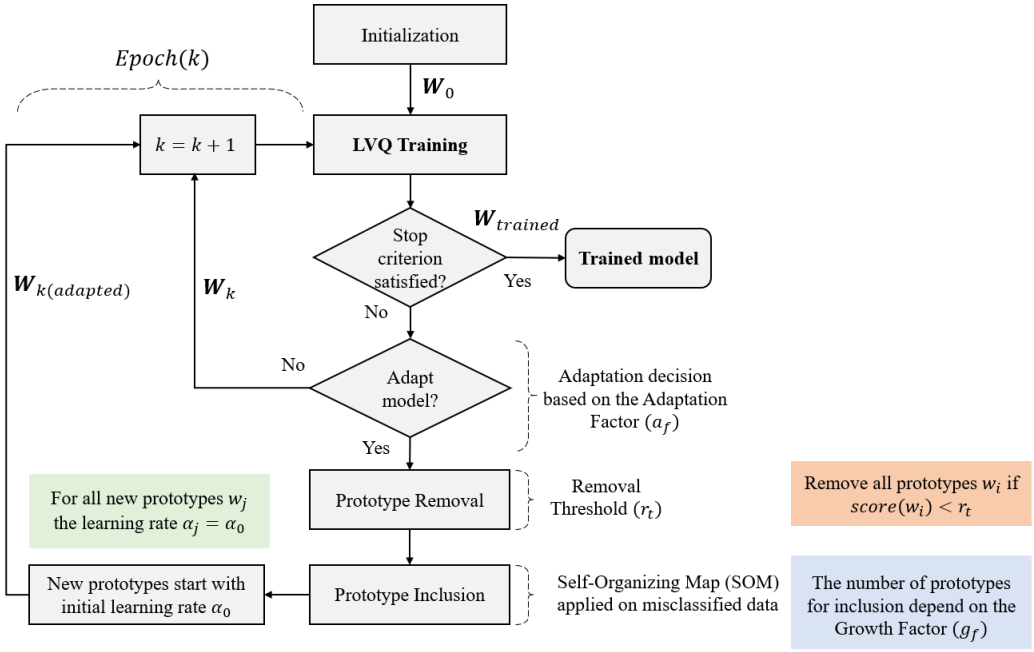
Source: The author.

Hence, prototypes are modified and new learning rates are assigned, referring to the orange line. Subsequently, new adaptations will occur and new generations will be created. In the example shown in Figure 30 two adaptations resulted in the inclusion of two new generations. In summary, there will be three learning rates: one for the initial prototypes of the network (represented by the blue color), and two for the prototypes included in the adaptations (represented by the colors orange and gray). The hyper-parameter Adaptation Factor (a_f) will be responsible for dictating the frequency of network adaptations.

4.5 Driven-Adaptive-LVQ (dALVQ)

The driven-learning rate can also be applied in an adaptive-LVQ. In driven Adaptive LVQ with Self-Organizing Maps (dALVQ), both characteristic of ALVQ and dLVQ are combined. Therefore, the dALVQ adaptation consists in removing poor representative prototypes, and including new prototypes, with generation-based learning rates. Figure 31 illustrates the flowchart framework for training a classifier based on dALVQ.

Figure 31 – Example of multiple learning rate per prototype generation



Source: The author.

4.6 Summary

In this chapter, fundamental concepts on Adaptive-LVQ were introduced. Moreover, two novel approaches for Adaptive-LVQ supervised learning were introduced: ALVQ-SOM and dLVQ. Besides that, a combination of the previous variations, called dALVQ, was also introduced. All presented methods can be combined with any LVQ-based variation, since it does not affects the algorithm's learning rule. Furthermore, the ALVQ framework for designing and training a classifier were presented.

Chapter 5 details the methodology for evaluating the novel proposed methods, as well as the other methods evaluated in this works.

5 EXPERIMENT METHODOLOGY

The main objectives of this work is to design, build and compare multiple neural network based classifiers, employed in different time series classification problems. In the context of time series classification, the experiments have been designed in order to achieve the following goals:

- Evaluate and compare the proposed LVQ-based method to other LVQ variations.
- Compare the efficiency of different learning approaches, among MLP, SVM and LVQ, in time series classification problems.
- Compare ANN-based algorithms to a benchmark naive method called k nearest neighbor.

In order to achieve those goals, it is necessary to define a proper testing methodology. In this chapter, the methodology applied in the experiments comparing different classifiers for time series classification is presented. Furthermore, the criteria used for evaluating performances and comparing different neural network models are detailed.

Finally, the methodology adopted for designing experiments is covered, including the main aspects considered for neural network analysis, such as number of neurons, training parameters, learning rate, pre-processing input data.

5.1 Performance Metrics

For evaluating a classifier, a set of metrics must be chosen properly, in order to asses the classifier quality. In this section, the general used performance metrics for evaluating classifiers are described.

5.1.1 Confusion matrix

A Confusion Matrix (CM) is a structure which provides general information about a supervised classifier performance, based on the class assignments to every sample of a specific dataset (MARTIN-DIAZ *et al.*, 2016). A CM is built by comparing desired, or actual, labels with predicted class labels. In Table 2, a two class problem confusion matrix is shown. Note that TP and TN stand for correctly labelled positive and negative samples. FN and FP symbolizes wrongly labelled negative and observations respectively. In a two-class classification problem, there are two classes: positive class and negative class. Each row from a CM represents the samples in the actual class, as each column represents the samples in predicted classes.

Table 2 – Confusion matrix for a two-class problem

Actual	**Prediction**		
	Positive (+)	**Negative (-)**	
Positive (+)	True Positive (TP)	False Negative (FN)	P_{actual}
Negative (-)	False Positive (FP)	True Negative (TN)	N_{actual}
	$P_{predicted}$	$N_{predicted}$	N_{Total}

The total number of actual positive and negative samples are denoted by P_{actual} and N_{actual}, respectively. Similarly, the total of sample which have been predicted to classes positive and negative are denoted by by $P_{predicted}$ and $N_{predicted}$. Finally, N_{Total} represents the total number of samples for this problem. In order to evaluate the classification performances of any method, it is interesting to evaluate measures such as accuracy, precision and recall. These measures can be extracted from a confusion matrix.

A common used score for assessing a classifier is the classification accuracy. Accuracy is the overall fraction of all correctly predicted samples among the whole data. This measure indicates the percentage of correct classifications relative to the total of samples. This measures can be calculated by the following equation:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} = \frac{TP+TN}{N_{Total}} \tag{5.1}$$

Consider $N_{Correct}$ the number of correct classifications. The accuracy equation may be simplified as follows:

$$Accuracy = \frac{N_{Correct}}{N_{Total}} \tag{5.2}$$

Another form of measuring the accuracy, is by classification error, which is given by $1 - Accuracy$. Even though accuracy is commonly used, it is an optimistic assumption, since it measures the overall classification performance. This may be a problem when dealing with imbalanced data, where each class have different number of samples. For evaluating the performance for each class, separately, it is necessary to use other metrics such as *precision* and *recall*. Precision can be defined as the fraction of positive results that are correctly identified among the positive predictions.

$$Precision = \frac{TP}{TP+FP} \tag{5.3}$$

Recall, also known as sensitivity, is a fraction of positive results that are correctly identified among the actual positive values.

$$Recall = \frac{TP}{TP+FN} \tag{5.4}$$

F-measure is a score which solves any contradiction between *precision* and *recall* (MARTIN-DIAZ *et al.*, 2016). For $\beta \in \mathbb{R}, \beta > 0$, the expression is:

$$F_\beta = \frac{(1+\beta)(precision \times recall)}{(\beta \times precision) + recall} \tag{5.5}$$

5.2 Cross-Validation

Cross-validation, or out-of-sample testing, are techniques used for evaluating models, by analyzing their performance on independent data sets. In a classification problem, a model uses the training set for learning patterns in the data, so it can generalize this information for predicting class of unknown data. The testing set is used for validating the model, as this set is composed by independent data, that is, data that have not been used for training the classifier.

5.2.1 Hold Out

The hold out cross-validation technique consist in randomly partitioning the data in two sets: training set and testing set. Usually, the testing set is smaller than the training set. The original Hold Out method involves only a single run. However, in order to evaluate statistically a model, this method will be applied several times for producing multiple independent results, which can used for extracting statistic measures such as mean and standard deviation.

For classification models, the commonly used evaluation measure is the *accuracy*. For instance, consider N_{Rounds} the number of rounds that a hold out cross validation will be applied on a classifier. Hence, N_{Rounds} classifiers will be built, from different training set, and each one will produce a hit rate result based on different testing sets. Then, the mean of all these accuracy results can be calculated as follows:

$$Mean(accuracy) = \frac{1}{N_{Rounds}} \sum_{i=1}^{N_{Rounds}} accuracy(i) \tag{5.6}$$

5.2.2 K-Fold

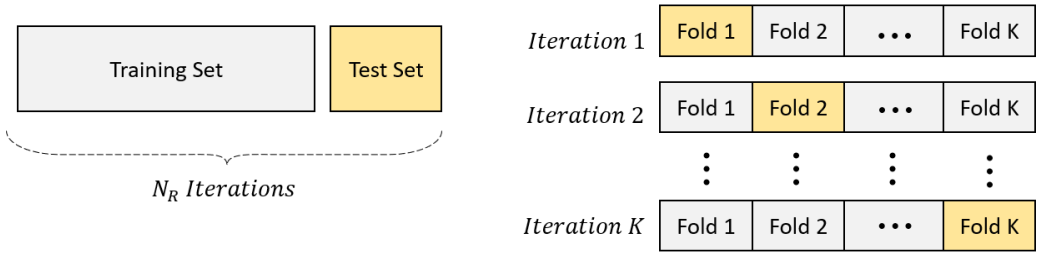
The *k-fold* cross-validation technique consists in dividing a dataset in K folds. For each iteration, a model will be trained using one fold as the testing set, and the remaining will form the training set. Therefore, *k-fold* method produces K models derived from K independent training/testing sets.

The *k-fold* may be more adequate for small datasets, as it can produce sets with higher variance. For calculating the mean in *k-fold* method, the average is calculated considering the accuracy measure of K models:

$$Mean(Accuracy) = \frac{1}{K}\sum_{i=1}^{K} accuracy(i) \tag{5.7}$$

Figure 32 illustrates how the Hold Out (Figure 32(a)) and K-Fold (Figure 32(b)) cross validation techniques separate the training and testing sets.

Figure 32 – Cross-validation techniques



a) Hold Out Cross-Validation

b) K-Fold Cross-Validation

Source: The author.

5.3 Experiment description

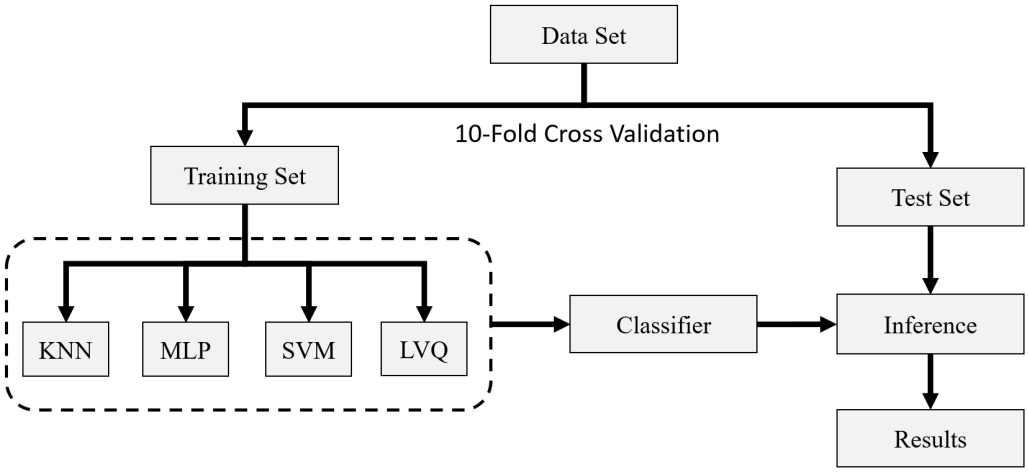
The experiments were designed for comparing multiple classifier models in different scenarios. A 10-Fold Cross-Validation technique was employed in order to increase the independence of the data for better evaluation of the proposed algorithm. From the execution, statistical measures were computed for summarizing the experiments. Mean and standard deviation have been used for comparing the accuracy among distinct classifiers. Boxplot diagrams were employed for analyzing the accuracy distribution for each LVQ-based algorithm. The experiment process is summarized in Figure 33.

Regarding data set pre-processing, the input attributes for all datasets used in our experiments were normalized between $[0,1]$. Considering a dataset with N samples, each sample n is normalized by the following equation:

$$x_j(n) = \frac{x_j(n) - x_j^{min}}{x_j^{max} - x_j^{min}}, n = 1, ..., N \tag{5.8}$$

where $x_j(n)$ is the j-th attribute of the n-th sample, as x_j^{max} and x_j^{min} are the maximum and minimum value of attribute j, respectively.

Figure 33 – Cross-validation techniques



Source: The author.

The selection of hyper-parameters of the models were performed in different ways, considering their influence on the classifier training. The hyper-parameters learning rate (α) and number of epochs (N_{Epochs}) for the LVQ and MLP models were defined empirically, from the convergence check in terms of classification error. The specific hyper-parameters of the LVQ2, LVQ3 and FLVQ algorithms were chosen based on the recommendations in the literature. The remaining hyper-parameters were selected using *Grid Search*. The cost function used in the search, for evaluating the classification models trained with different parameters, was the classification accuracy. In Table 3 the parameter for each classifier is summarized. Parameter optimization using Grid Search was applied only to parameters defined by closed intervals (See Table 3). Note that for LVQ-based algorithms, simulations were performed by varying the number of prototypes per class (N_{ppc}), by starting with $N_{ppc} = 1$ until $N_{ppc} = 10$. Thus, the number of prototypes in a LVQ network will be $N_{ppc} \times N_{Classes}$, where $N_{Classes}$ denotes the number of classes of the problem.

Regarding the ALVQ-SOM, adaptive factor (a_f) and removal threshold (r_t) were tuning in order to maintain the equivalence between the ALVQ-SOM algorithm and the ALVQ-GV. Therefore, the ALVQ parametric values are: $r_t = 0$ and $a_f = 1$. The Growth Factor (g_f) has been arbitrarily selected $g_f = 0.05$ to ensure small additions of prototypes. As for $a_f = 1$, every epoch will be followed by an adaptation. Selecting the sub-optimal hyper-parameters for LVQ-SOM is a difficult task, since they strongly depend on the dataset used. Poorly selection of LVQ-SOM hyper-parameters can lead to problems in convergence and classification performance (overfitting). Ideally, an optimization algorithm should be used for optimizing the LVQ-SOM hyper-parameters. However, this methodology is extremely time consuming when applied

Table 3 – Parameters used in experimenting different classification models

***k*-NN**	
$k \in [1,10]$	: Number of k- nearest neighbors
Euclidean distance	: Distance metric
LVQ	
$N_{ppc} \in [1,10]$	: Number of prototypes per class
$\alpha_0 = 0.09$	: Learning rate
$N_{Epochs} = 200$	: Number of epochs
Euclidean distance	: Distance metric
$w = 0.2$	: LVQ2 and LVQ3 parameter
$\varepsilon = 0.1$	: LVQ3 parameter
$m = 0.2$	: FLVQ parameter
MLP	
$N_e \in [1,15]$	: Number of hidden neurons
$\alpha = 0.2$	: Learning rate
$N_{Epochs} = 4000$	: Number of epochs
SVM	
$C \in [2^{-3}, 2^3]$	: Number of C
$\gamma \in [2^{-4}, 2^{-2}]$	: $\gamma = -\frac{1}{2\sigma^2}$
RBF Kernel	: Kernel

to multiple datasets. The k-NN method will be used as a reference for comparing the other algorithms. Eight datasets with distinct characteristics have been used in this comparative study.

The experiment is composed of several executions of the algorithms, using 10-Fold cross validations, repeated several executions (10 times). Multiple experiments were conducted with different number of prototypes per class (N_{ppc}) in order to find the best results for each classifier. The sub-optimal parameters for each classifier is presented in Appendix A. For each execution, the original data set is shuffled, in order to create independent sets in each execution. After all executions, statistics have been computed for summarizing the performance of the methods, resulting in a distribution of accuracy. The statistic-based measures used for expressing the result of accuracy was: mean and standard deviation.

During the following sections, ALVQ will be considered as an abbreviation of ALVQ-SOM. Moreover, two SOM-based methods of inclusion of prototypes ($v1$ and $v2$) are adopted (See subsection 4.3.1), and the hierarchy clustering used in the work of Grbovic & Vucetic (2009) will be referred as gv. For example, a dALVQ1$_{gv}$ is a driven-Adaptive Learning Vector Quantization algorithm, applied on LVQ1 variation, which adopt the hierarchical clustering for inclusion of prototypes, as proposed in ALVQ-GV.

5.4 Summary

In this chapter, several techniques and methods for evaluating the classification models were described. Common used performance metrics such as *accuracy*, *recall*, and *precision* were discussed. Furthermore, two cross-validation methodologies known as Hold Out and K-Fold were introduced, as they are interesting methods for evaluating classifiers on independent datasets.

Finally, a brief description on the experimentation methodology was presented, covering the hyper-parameters selections and normalization of input vectors. In the following chapter, the experimental procedures will be detailed, and the main results will be discussed.

6 EXPERIMENTS AND RESULTS

In this chapter, the experimental results obtained from computational simulations are presented and discussed. Several experiments have been conducted with different datasets for fairly analyzing the performance of studied algorithms. All simulations were developed and executed using Matlab. For SVM implementation, a toolbox called libsvm was used (CHANG; LIN, 2011).

For general performance evaluation of the proposed method, the classifiers with the best results have been selected in order to compare the classification performance, considering metrics such as accuracy, recall, precision and F1 score. This is the main experimental step of this work. In this experiment, the classical variations of the LVQ with the adaptive versions are compared, considering sub-optimal parameters of network configuration, found experimentally. The comparison takes place in two aspects: classification performance, memory cost and execution time. The main objective of the proposed algorithm is to improve classification performance. However, a cost analysis was performed with the objective of evaluating the feasibility of using this algorithm in practical engineering problems.

The processing cost was compared considering the execution time spent during the training. In the evaluation of memory cost, the final amount of prototypes generated by the network was verified. Finally, the accuracy was adopted as the main measure for evaluating classification performance.

Considering P_0 as the initial number of prototypes, the simulations were performed using $Budget = P_0$ (Section 4.3) in order to verify the classification performance when the number of prototypes is limited by initial architecture. In the next section, the presented results are summarized. More details on the results can be found in Appendix B.

Furthermore, a study on the influence of Adaptive Factor (a_f) in training a ALVQ algorithm was realized. In this experiment, the influence of the adaptation fact in the training an adaptive LVQ network is evaluated. For this, the experiments have been tested with the following adaptation values $a_f \in \{1, 2, 5, 10, 20, 30, 50, 70, 100\}$. In this experiment, the behavior of learning curves as well as the variation in computational cost were evaluated, since each adaptation has a computational cost involved.

6.1 General performance evaluation

For general performance evaluation, seven benchmark datasets for times series classification were selected from UCR Repository (CHEN *et al.*, 2015) for being used in simulations for comparative purposes. Each dataset present distinct characteristics, as the aim is to evaluate the performance of the proposed method and other different classification models over different circumstances. The datasets have been chosen by varying the number of input attributes (or time series' window length), number of classes and quantity of samples. Table 4 shows these datasets and their characteristics:

Table 4 – Description of UCR datasets

Dataset	**No. Classes**	**No. Samples**	**Input length**
Synthetic Control	6	600	60
Gun-Point	2	200	150
CBF	3	930	128
Trace	4	200	275
ECG200	2	200	96
Italy Power Demand	2	1096	24
Computers	2	500	720

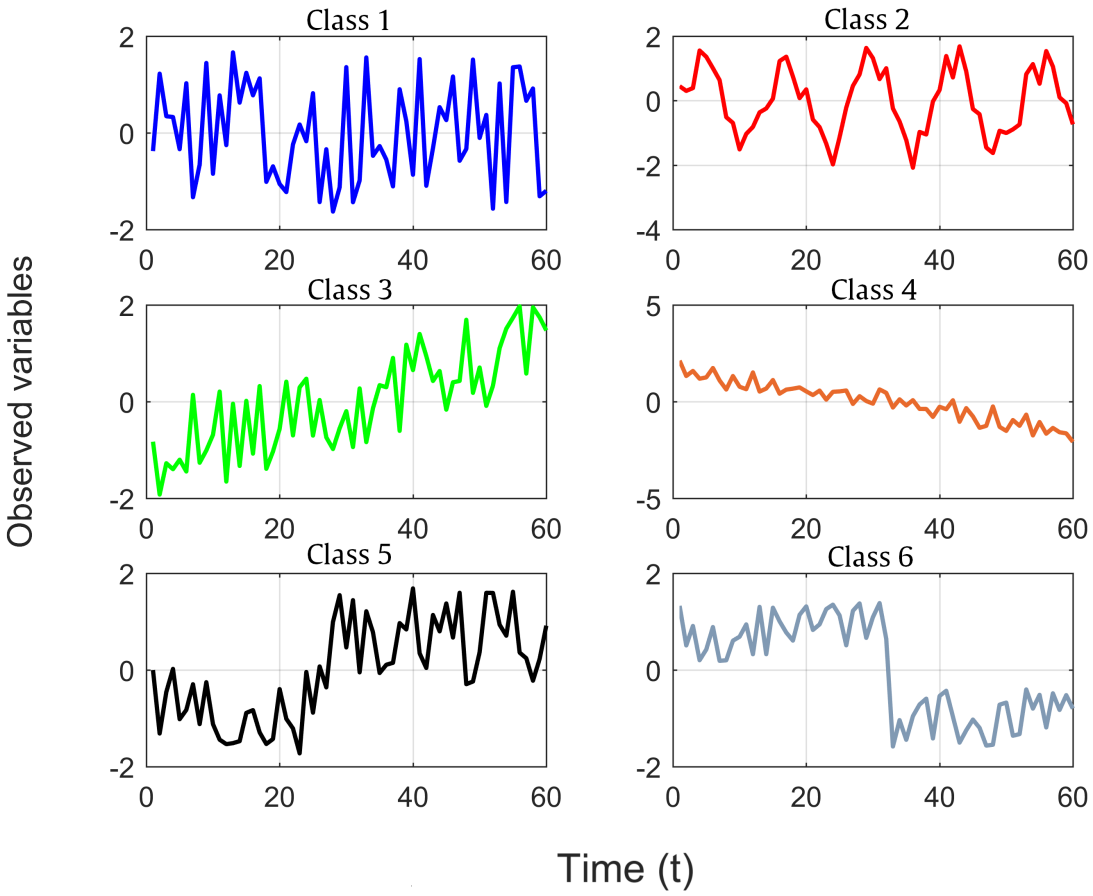
In the next subsections, each experiment, involving a single database, will be described and discussed. The general results of classification performance and accuracy distribution will be presented. As a general classification performance, accuracy, recall and precision are presented in terms of mean and standard deviation.

Furthermore, the accuracy distribution considering multiple experiments is presented using boxplots. Regarding the neural network training evolution over the epochs, learning curves are presented and interpreted. The results will be presented as follows: first the dataset is briefly described, then the obtained results are shown, including comparative tables between LVQ variations and among all classifiers. Moreover, tables containing quantity of prototypes and execution time are presented and discussed. Note that the execution time involves the process of training the model using training set, and testing the test set. Finally, boxplots of accuracy distribution for each variation of LVQ are shown and analyzed.

6.1.1 Synthetic Control Dataset

This dataset is composed by control charts synthetically generated by different processes, introduced in Alcock *et al.* (1999). There are six different classes of control charts: Normal (1), Cyclic (2), Increasing trend (3), Decreasing trend (4), Upward shift (5) and Downward shift(6) (BAGNALL *et al.*, 2018). Figure 34 illustrates the times series for each class of synthetic control dataset.

Figure 34 – Synthetic control time series



Source: The author.

Table 5 presents the comparison of the LVQ-based classification performance of the algorithms for each evaluated variation in the classification problem using the Synthetic Control dataset. In all cases, the adaptive LVQ version presented a higher average accuracy than the respective traditional LVQ models. The standard deviation of accuracy of the adaptive models LVQ1 and FLVQ were lower than the original models. However, the standard deviation of the accuracy of adaptive versions LVQ2 and LVQ3 is greater than their original versions.

Table 5 – Overall LVQ-based classification performance results (Synthetic Control).

Algorithms	**Accuracy**	**Precision**	**Recall**	F_1 **Score**
LVQ1	82.33 ± 7.79	74.52	82.82	78.36
dALVQ1$_{v2}$	$\mathbf{97.33 \pm 1.79}$	**97.57**	**97.31**	**97.44**
LVQ2	63.17 ± 11.15	45.55	61.14	51.79
dALVQ2$_{gv}$	83.67 ± 21.78	79.58	83.67	80.99
LVQ3	59.67 ± 14.61	42.24	59.46	48.96
ALVQ3$_{v2}$	92.17 ± 15.48	89.70	92.63	90.76
FLVQ	93.83 ± 06.26	94.24	92.95	93.59
dAFLVQ$_{gv}$	95.67 ± 01.96	95.93	95.92	95.92

Table 6 shows the time interval for training and testing, and the final network quantity of prototypes P_{c_i} for each class c_i. The largest reductions in the number of prototypes occur precisely in the adaptive versions of LVQ2 and LVQ3 - which possibly influenced the largest standard deviation.

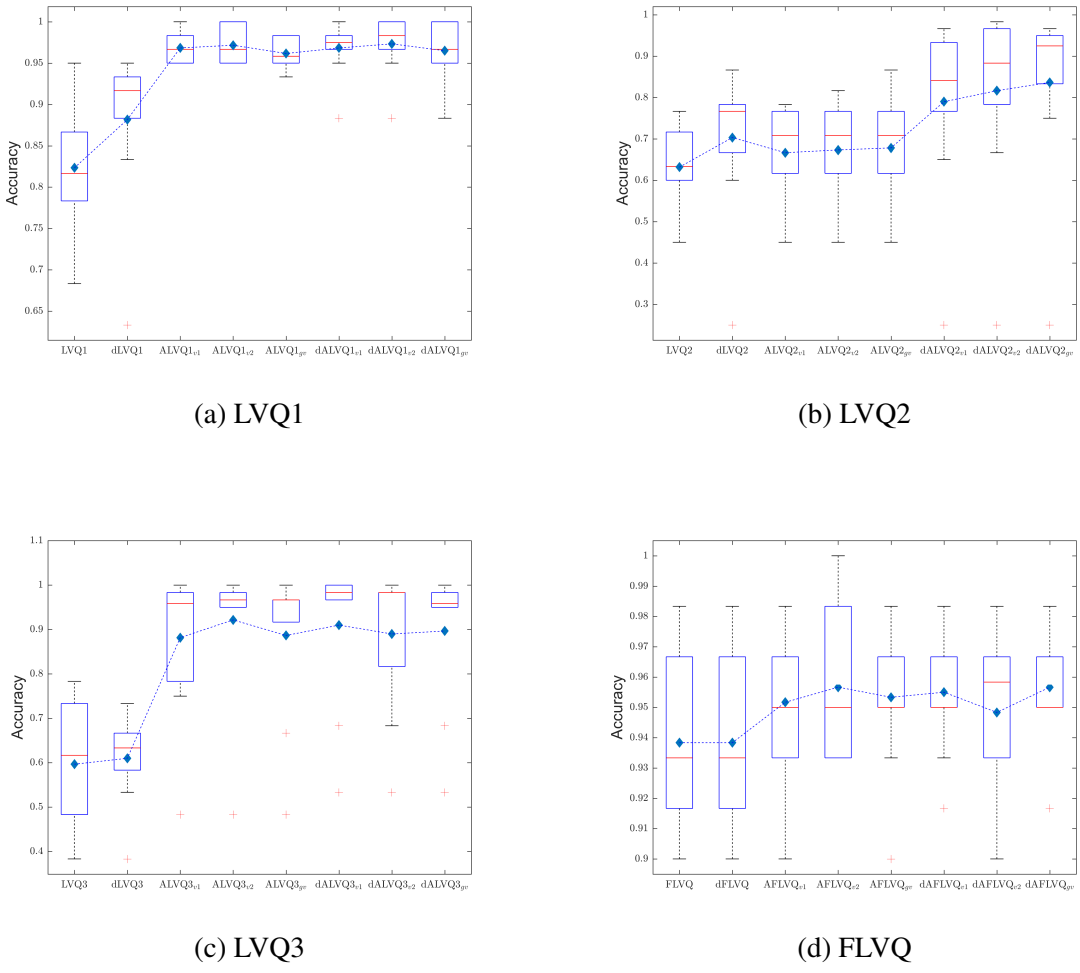
Table 6 – Overall cost regarding number of prototype and execution time of LVQ-based classification algorithms (Synthetic Control).

Algorithms	P_{c_1}	P_{c_2}	P_{c_3}	P_{c_4}	P_{c_5}	P_{c_6}	N_P	Execution time (s)
LVQ1	2	2	2	2	2	2	12	3.71
dALVQ1$_{v2}$	1	2	1	1	2	2	9	4.58
LVQ2	2	2	2	2	2	2	12	3.80
dALVQ2$_{gv}$	1	1	1	1	1	1	6	0.19
LVQ3	2	2	2	2	2	2	12	3.88
ALVQ3$_{v2}$	1	1	2	1	2	1	8	2.99
FLVQ	2	2	2	2	2	2	12	19.42
dAFLVQ$_{gv}$	4	2	2	1	1	1	11	21.98

Note that adaptive LVQ classifiers generally have higher execution time values. This is because adaptive LVQ training tends to be more time costly than static LVQ, due to their adaptation steps. However, there are cases where adaptive convergence is more efficient. For example, dALVQ2$_g v$ presented a significant lower execution time compared to classic LVQ2. This case is a typical example where the adaptation improves the classifier's training convergence.

Regarding statistical distribution of accuracy indices, Figure 35 presents the boxplots of accuracy for each cross-validation round.

Figure 35 – Accuracy distribution for LVQ-based classifiers (Synthetic Control)



(a) LVQ1 (b) LVQ2

(c) LVQ3 (d) FLVQ

Source: The author.

Table 7 shows the comparative accuracy performance between the best accuracy adaptive LVQ and three other classifiers: dALVQ1 presented higher performance than MLP and KNN, and performance close to SVM.

Table 7 – Overall classification performance results (Synthetic Control).

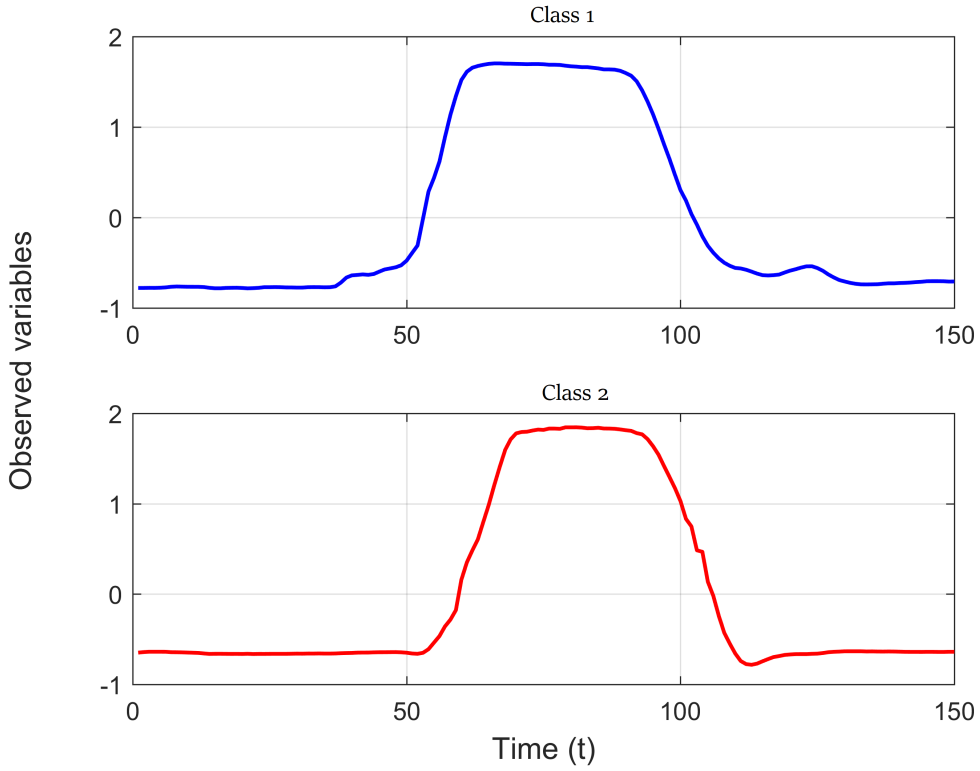
Algorithms	**Accuracy**	**Precision**	**Recall**	F_1 **Score**
dALVQ1$_{v2}$	97.33 ± 1.79	97.57	97.31	97.44
KNN	93.33 ± 3.04	93.49	93.24	93.35
SVM	$\mathbf{98.50 \pm 1.66}$	**98.39**	**98.70**	**98.54**
MLP	93.72 ± 02.86	92.54	93.02	92.77

6.1.2 Gun-Point Dataset

The Gun-Point dataset involves two actors (one female and other male) making a motion with their hand. The two classes are: Gun-Draw(1) and Point(2). For Gun-Draw the actors have their hands by their sides. They draw a replicate gun from a hip-mounted holster, point it at a target for approximately one second, then return the gun to the holster, and their hands to their sides. For Point the actors have their gun by their sides. They point with their index fingers to a target for approximately one second, and then return their hands to their sides.

The time series consist of recordings on the centroid X of the actor's right. (BAGNALL *et al.*, 2018). Figure 36 illustrates the times series for each class of Gun-Point dataset.

Figure 36 – Gun-Point time series



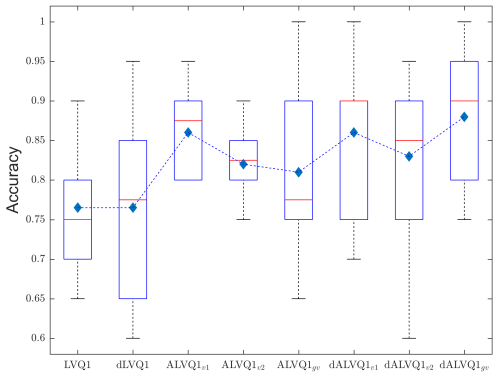
Source: The author.

Table 8 and Figure 37 show that in all cases there is an adaptive LVQ version that has a higher average accuracy than the respective traditional LVQ models. However, in the case of LVQ2, only one of the adaptive versions (Figure 37) surpassed the accuracy of the original model. The standard deviation of accuracy of the adaptive models LVQ2, LVQ3 and FLVQ were smaller than the original models. However, the standard deviation of accuracy of the LVQ1 adaptive version is close to the deviation of its original versions.

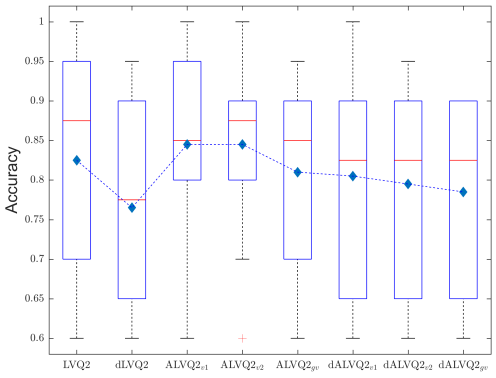
Table 8 – Overall LVQ-based classification performance results (Gun-Point).

Algorithms	**Accuracy**	**Precision**	**Recall**	**F_1 Score**
LVQ1	76.50 ± 07.47	78.63	76.50	77.54
dALVQ1$_{gv}$	88.00 ± 07.89	89.55	88.00	88.76
LVQ2	82.50 ± 14.58	88.36	82.50	85.14
ALVQ2$_{v2}$	84.50 ± 11.89	88.73	84.50	86.45
LVQ3	73.00 ± 10.59	82.11	73.00	77.16
dALVQ3$_{v1}$	$\mathbf{93.50 \pm 06.26}$	**93.97**	**93.50**	**93.73**
FLVQ	88.50 ± 07.09	89.04	88.50	88.77
dAFLVQ$_{v1}$	92.00 ± 05.37	92.49	92.00	92.25

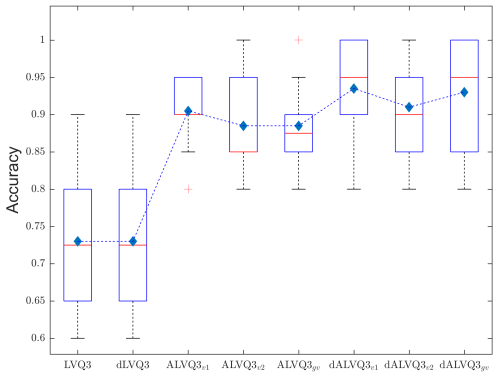
Figure 37 – Accuracy distribution for LVQ-based classifiers (Gun-Point)



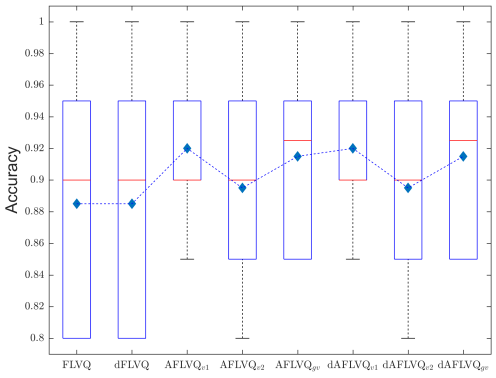
(a) LVQ1



(b) LVQ2



(c) LVQ3



(d) FLVQ

Source: The author.

Table 9 shows that the adaptive versions of LVQ1 and LVQ3 had little prototype reduction, reducing only 1 prototype in both cases. The LVQ2 variation has reduced 2 prototypes, and the accuracy performance increased slightly. As for FLVQ, the reduction was significant, considering that with 50% less prototypes the adaptive LVQ classifier presented higher accuracy then LVQ.

Table 9 – LVQ-based Overall cost (Gun-Point).

Algorithms	P_{c_1}	P_{c_2}	N_P	Execution time (s)
LVQ1	5	5	10	0.89
dALVQ1$_{gv}$	5	4	9	1.02
LVQ2	5	5	10	0.70
ALVQ2$_{v2}$	3	5	8	0.76
LVQ3	5	5	10	0.95
dALVQ3$_{v1}$	3	6	9	1.22
FLVQ	5	5	10	2.83
dAFLVQ$_{v1}$	2	3	5	2.62

It can be seen from the Table 9 that the execution time does not vary much between classic and adaptive versions of LVQ. This may be an indication that the classification problem in question is not of a high complexity. Therefore, it does not require the use of adaptive methods to ensure better data separation.

The table below (Table 10) shows the comparative performance between the best adaptive LVQ and three other classifiers: dALVQ3 presented the worst comparative performance, pointing out that the other classifiers were more efficient in establishing a boundary between the two class standards (dichotomy) of this dataset.

Table 10 – Overall classification performance results (Gun-Point).

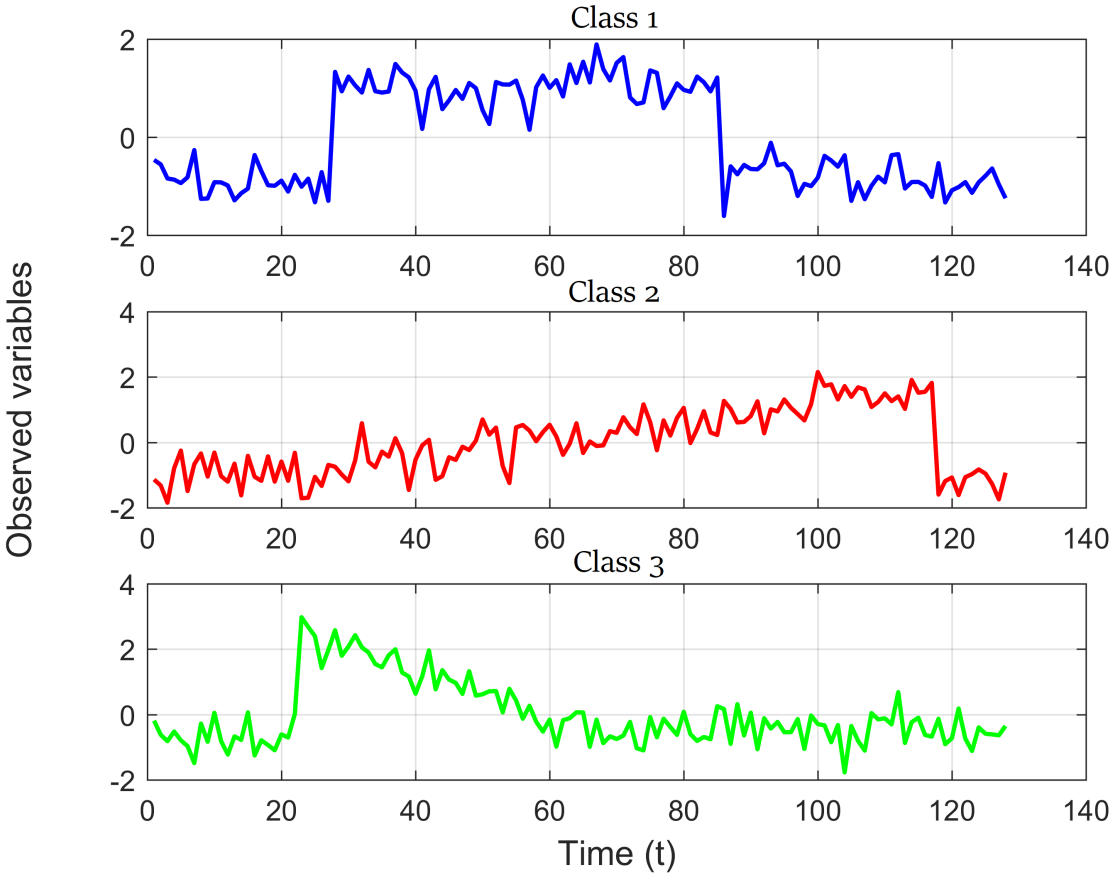
Algorithms	**Accuracy**	**Precision**	**Recall**	F_1 **Score**
dALVQ3$_{v1}$	93.50 ± 6.26	93.97	93.50	93.73
KNN	94.50 ± 3.69	95.09	93.80	94.43
SVM	98.50 ± 0.42	98.44	98.67	98.56
MLP	$\mathbf{100.00 \pm 0.00}$	**100.00**	**100.00**	**100.00**

An interesting result is that MLP-based classifier presented maximum accuracy of 100%. This may indicate that the problem has a nonlinear solution which was found by the MLP classifier in all training/test rounds. Therefore, in this case, the MLP model showed higher ability to transform temporal data into an proper feature space for data separation from this data set.

6.1.3 CBF Dataset

Cylinder-Bell-Funnel (CBF) is a simulated dataset introduced by Saito (2000). This problem consist in classify time series in cylinder, bell and funnel (BAGNALL *et al.*, 2018). Figure 38 illustrates the times series for each class of CBF dataset.

Figure 38 – CBF time series



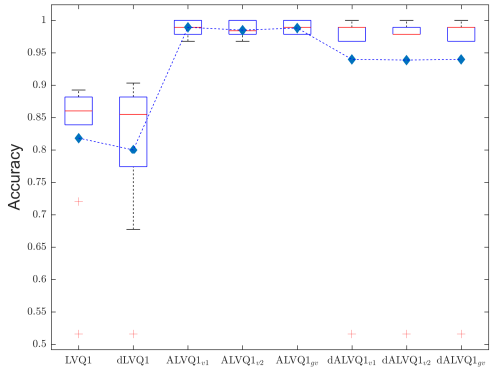
Source: The author.

Table 11 and Figure 39 show that in all cases there is an adaptive LVQ version that has a higher average accuracy than the respective traditional LVQ models. The standard deviation of accuracy of the adaptive models LVQ1 and FLVQ were smaller than the original models. However, the standard deviation of the accuracy of adaptive versions LVQ2 and LVQ3 is greater than their original versions. This significant difference in accuracy between the proposed Adaptive LVQ and the classical LVQ models is due to the fact that non-adaptive LVQ may not converge properly if there is a poor initialization of the network prototypes.

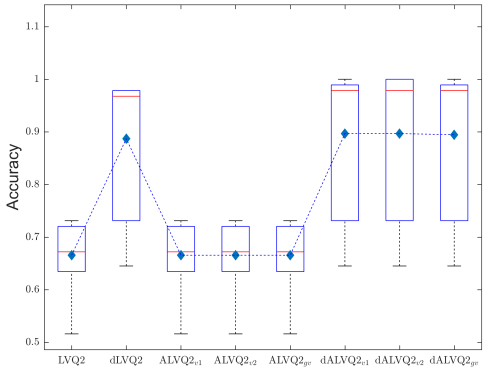
Table 11 – Overall LVQ-based classification performance results (CBF).

Algorithms	**Accuracy**	**Precision**	**Recall**	**F_1 Score**
LVQ1	81.83 ± 11.72	80.21	82.95	81.06
ALVQ1$_{v1}$	$\mathbf{98.92 \pm 1.13}$	**98.99**	**99.01**	**99.00**
LVQ2	66.56 ± 6.64	46.93	66.54	54.95
dALVQ2$_{v2}$	89.68 ± 14.58	83.17	89.13	85.65
LVQ3	66.56 ± 6.64	46.93	66.54	54.95
dALVQ3$_{v1}$	89.89 ± 14.70	83.47	89.23	85.85
FLVQ	88.49 ± 4.27	88.15	88.30	88.22
dAFLVQ$_{v1}$	98.60 ± 1.25	98.57	98.62	98.59

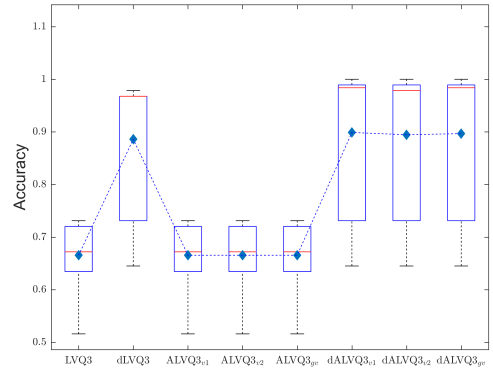
Figure 39 – Accuracy distribution for LVQ-based classifiers (CBF)



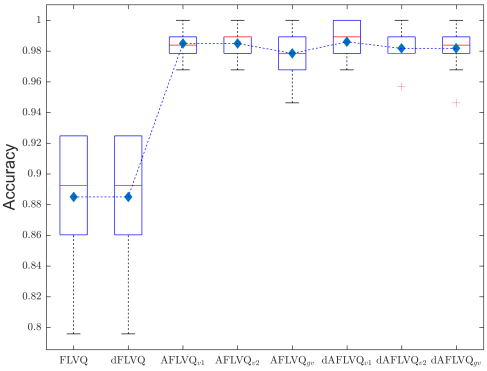
(a) LVQ1



(b) LVQ2



(c) LVQ3



(d) FLVQ

Source: The author.

Table 12 shows that there was no change in the number of prototypes of adaptive versions. The number of prototypes (3) in all LVQ models was equal to the number of classes (3). Note that with only one prototype per class, it is possible to define classification models with high classification performance. This shows that a linear solution can be used to solve this problem, considering that an LVQ network with only one prototype per class, with Euclidean distance measurement, is only able to perform a linear separation between the data.

Table 12 – Overall cost regarding number of prototype and execution time of LVQ-based classification algorithms (CBF).

Algorithms	P_{c_1}	P_{c_2}	P_{c_3}	N_P	Execution time (s)
LVQ1	1	1	1	3	5.90
ALVQ1$_{v1}$	1	1	1	3	8.44
LVQ2	1	1	1	3	0.14
dALVQ2$_{v2}$	1	1	1	3	0.26
LVQ3	1	1	1	3	0.13
dALVQ3$_{v1}$	1	1	1	3	0.27
FLVQ	1	1	1	3	18.66
dAFLVQ$_{v1}$	1	1	1	3	23.90

The table below (Table 16) shows the comparative performance between the best accuracy adaptive LVQ and three other classifiers: ALVQ1 presented the worst comparative performance (98.92), but very close to the performance of the other classifiers (99.46; 99.68 and 99.68).

Table 13 – Overall classification performance results (CBF).

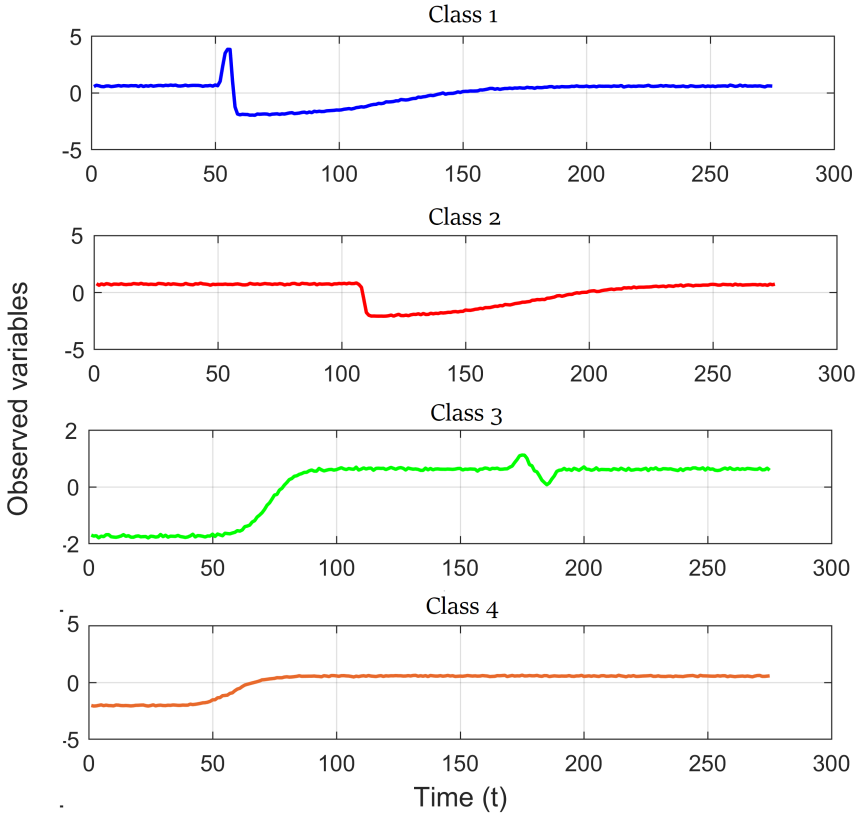
Algorithms	**Accuracy**	**Precision**	**Recall**	F_1 **Score**
ALVQ1$_{v1}$	98.92 ± 1.13	98.99	99.01	99.00
KNN	99.46 ± 0.26	99.47	99.46	99.47
SVM	$\mathbf{99.68 \pm 0.21}$	**99.67**	**99.68**	**99.67**
MLP	$\mathbf{99.68 \pm 0.21}$	**99.70**	**99.68**	**99.69**

From the general results of the classifiers, it is noteworthy that the problem tends to be solvable by linear or "slightly" nonlinear models, since all classifiers presented almost optimal results.

6.1.4 Trace Dataset

The Trace dataset is a subset of the Transient Classification Benchmark (Trace project). It is composed by a synthetic dataset designed to simulate instrumentation failures in a nuclear power plant, created by Davide Roverso. The dataset contains 200 instances of synthetic failures, divided in 4 classes, 50 samples per class (BAGNALL *et al.*, 2018). Figure 40 illustrates the times series for each class of synthetic control dataset.

Figure 40 – Trace time series



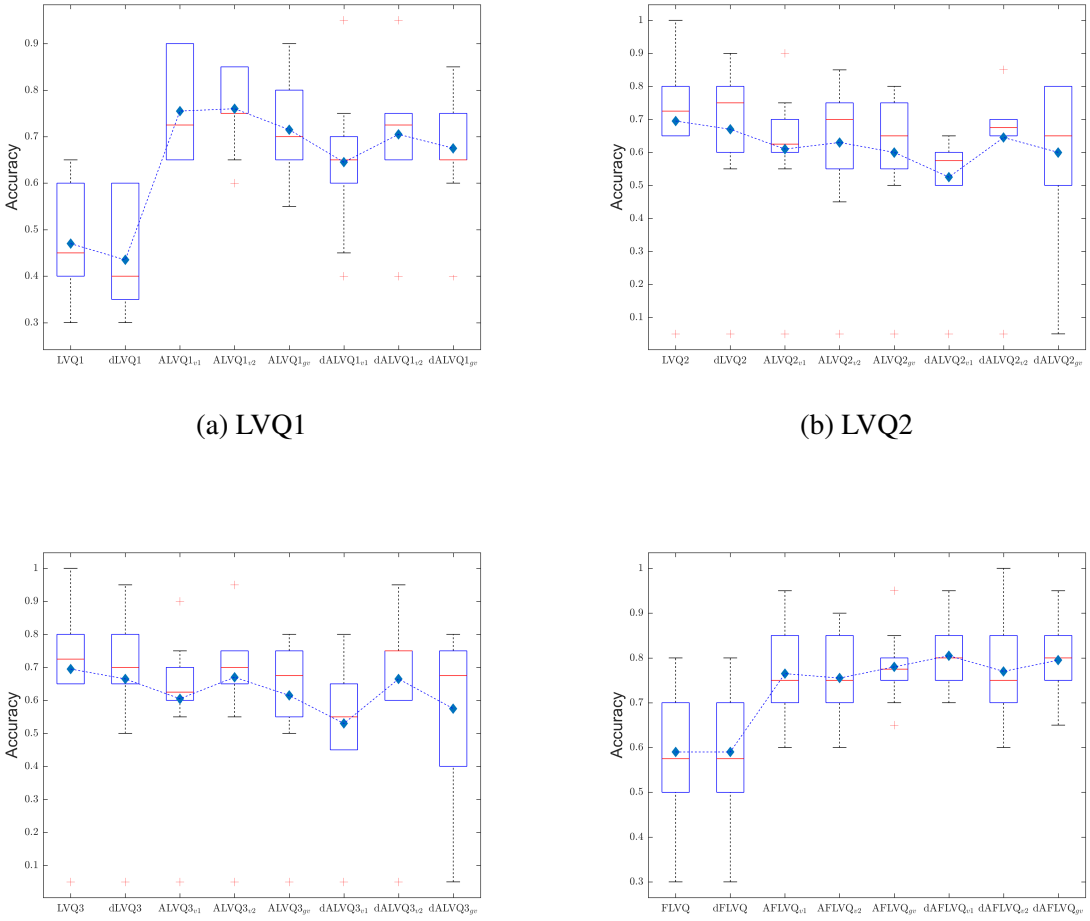
Source: The author.

Table 14 and Figure 41 show that in the LVQ1 and FLVQ cases the adaptive version presented a higher average accuracy than the respective traditional LVQ models. In cases LVQ2 and LVQ3, the adaptive versions performed very closely with slightly lower accuracy. The standard deviation of accuracy of the adaptive LVQ1 and FLVQ models was smaller than the original models. However the standard deviation of the accuracy of the adaptive versions LVQ2 and LVQ3 are close to the deviations of the original versions.

Table 14 – Overall LVQ-based classification performance results (Trace).

Algorithms	**Accuracy**	**Precision**	**Recall**	**F_1 Score**
LVQ1	47.00 ± 11.60	39.47	50.99	44.20
ALVQ1$_{v2}$	76.00 ± 08.43	77.60	77.05	77.19
LVQ2	69.50 ± 24.99	68.58	71.66	69.38
dLVQ2	67.00 ± 24.52	61.87	67.80	64.00
LVQ3	69.50 ± 24.99	68.58	71.66	69.38
ALVQ3$_{v2}$	67.00 ± 25.08	71.06	71.48	70.66
FLVQ	59.00 ± 15.06	58.14	58.27	58.17
dAFLVQ$_{v1}$	$\mathbf{80.50 \pm 7.98}$	**84.47**	**80.82**	**82.57**

Figure 41 – Accuracy distribution for LVQ-based classifiers (Trace)



(a) LVQ1

(b) LVQ2

(c) LVQ3

(d) FLVQ

Source: The author.

Table 36 shows that there was no change in the number of prototypes between the adaptive LVQ versions and their original versions. The number of prototypes (4) in all LVQ models was equal to the number of classes (4). Note that with only one prototype per class, it is possible to define classification models. However, the accuracy performance would probably increase if new prototypes were included properly. This shows that a non-linear solution would be more efficient to solve this problem, considering that an LVQ networks with multiple prototype per class are able to perform non-linear separations between the data.

Table 15 – LVQ-based overall cost (Trace).

Algorithms	P_{c_1}	P_{c_2}	P_{c_3}	P_{c_4}	N_P	Execution time (s)
LVQ1	1	1	1	1	4	1.52
ALVQ1$_{v2}$	1	1	1	1	4	2.57
LVQ2	1	1	1	1	4	0.77
dLVQ2	1	1	1	1	4	1.40
LVQ3	1	1	1	1	4	0.76
ALVQ3$_{v2}$	1	1	1	1	4	2.24
FLVQ	1	1	1	1	4	4.98
dAFLVQ$_{v1}$	1	1	1	1	4	6.40

The table below (Table 16) shows the comparative performance between the best adaptive LVQ and three other classifiers: dAFLVQv_1 presented the worst comparative performance.

Table 16 – Overall classification performance results (Trace).

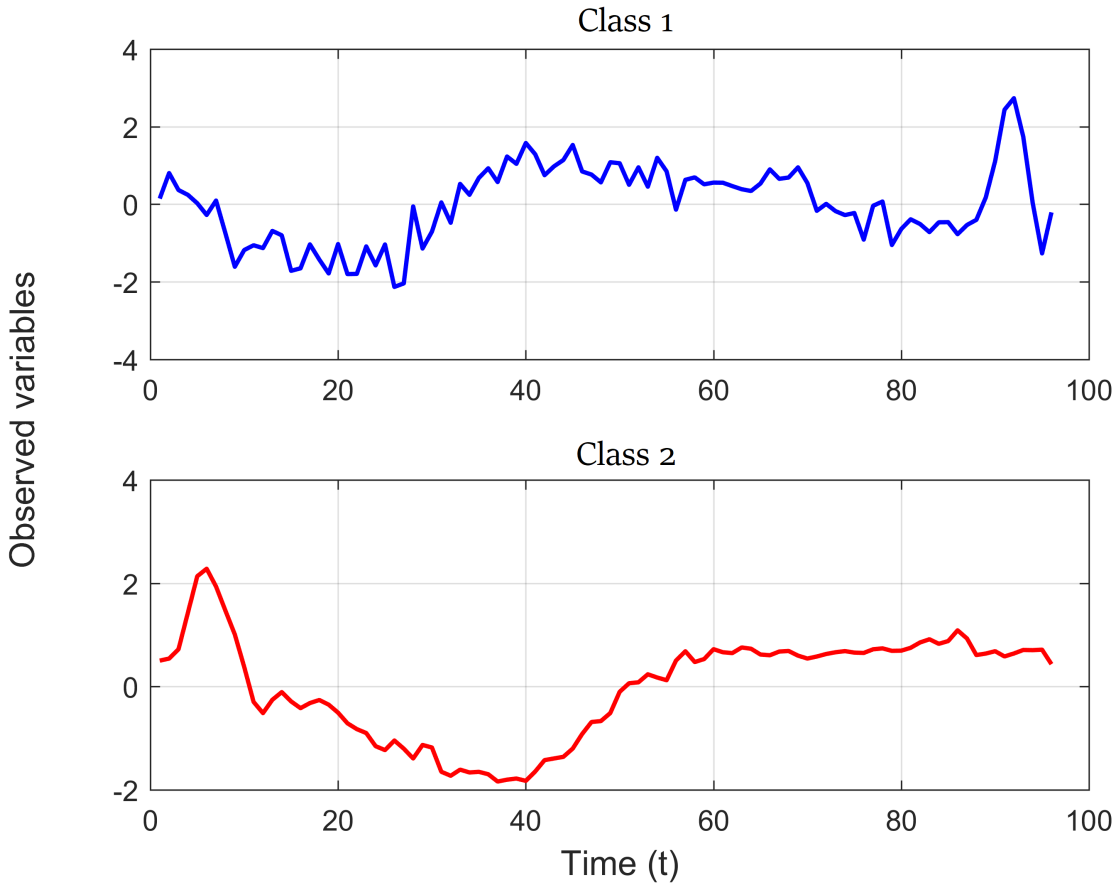
Algorithms	**Accuracy**	**Precision**	**Recall**	F_1 **Score**
dAFLVQ$_{v1}$	80.50 ± 7.98	84.47	80.82	82.57
KNN	87.50 ± 4.86	90.45	87.32	88.83
SVM	$\mathbf{95.00 \pm 2.27}$	**95.66**	**94.99**	**95.29**
MLP	85.50 ± 7.62	84.92	85.17	85.02

From the general results of the classifiers, it is noteworthy that the problem tends to be solvable by non-linear models. The higher results were presented by SVM, MLP, and KNN, which in this case are non-linear classifiers. However, the linear classifier constructed by one prototype per class in LVQ approach has not achieved high accuracy. In that case, the hyper-parameter *Budget* could be set to allow increasing the number of prototypes per class in order to improve the non-linearity of the model and increase its classification performance.

6.1.5 ECG200 Dataset

The ECG200 dataset was introduced by Olszewski (2001). Each series is composed by the electrical activity recorded during one heartbeat. The two classes are a normal heartbeat(1) and a Myocardial Infarction(2) (BAGNALL *et al.*, 2018). Figure 42 illustrates the times series for each class of ECG200 dataset.

Figure 42 – ECG200 time series



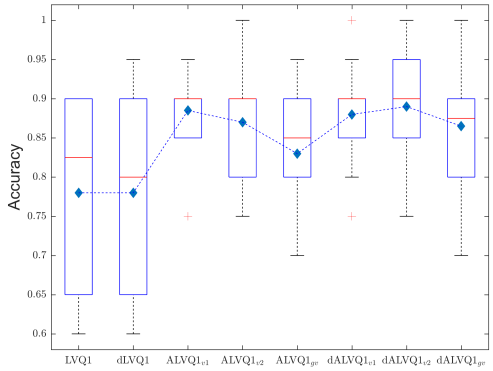
Source: The author.

Table 37 and Figure 43 show that in all cases the adaptive version presented a higher average accuracy than the respective traditional LVQ models. The standard deviation of accuracy of all adaptive models was smaller than the original models.

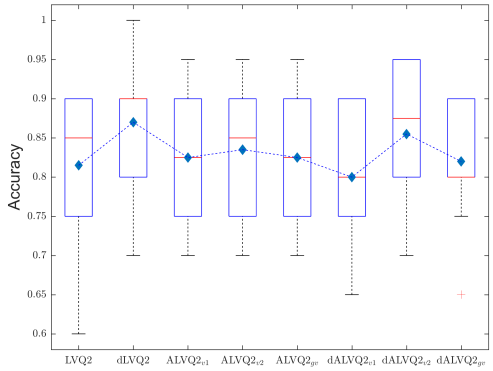
Table 17 – Overall LVQ-based classification performance results (ECG200).

Algorithms	**Accuracy**	**Precision**	**Recall**	**F_1 Score**
LVQ1	78.00 ± 11.83	76.35	74.47	75.24
dALVQ1$_{v2}$	89.00 ± 7.75	89.29	86.00	87.46
LVQ2	81.50 ± 10.29	79.32	79.32	79.29
dALVQ2$_{v2}$	88.50 ± 4.12	87.76	89.88	88.76
LVQ3	79.00 ± 11.25	83.53	72.57	77.04
dALVQ3$_{gv}$	87.00 ± 8.56	86.31	83.18	84.55
FLVQ	80.00 ± 10.33	79.24	79.82	79.52
AFLVQ$_{v2}$	$\mathbf{92.00 \pm 7.89}$	**91.66**	**92.04**	**91.84**

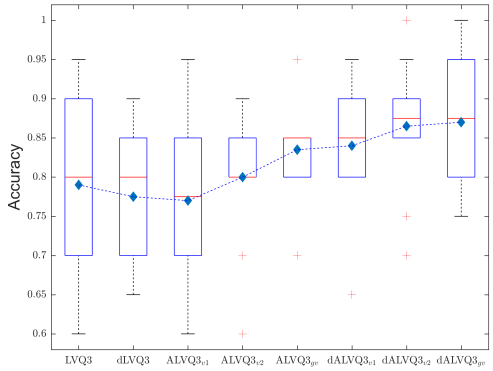
Figure 43 – Accuracy distribution for LVQ-based classifiers (ECG200)



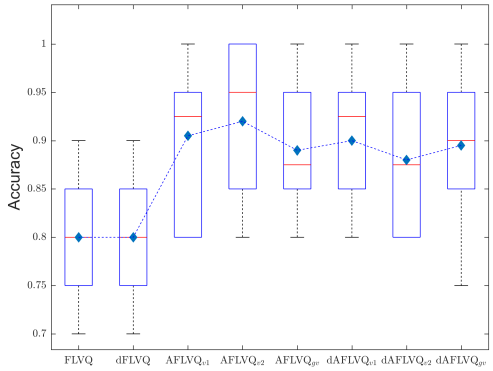
(a) LVQ1



(b) LVQ2



(c) LVQ3



(d) FLVQ

Source: The author.

Table 38 shows that there was a small reduction in the number of prototypes between the adaptive LVQ versions and their original versions (from 8 to 7 in all cases). The number of prototypes in all LVQ models (7 or 8) was higher than the number of dataset classes (2), which may have contributed to a better definition of the boundary between the two classes.

Table 18 – Overall cost regarding number of prototype and execution time of LVQ-based classification algorithms (ECG200).

Algorithms	P_{c_1}	P_{c_2}	N_P	Execution time (s)
LVQ1	4	4	8	1.53
dALVQ1$_{v2}$	2	5	7	2.57
LVQ2	4	4	8	1.20
dALVQ2$_{v2}$	1	6	7	2.20
LVQ3	4	4	8	1.58
dALVQ3$_{gv}$	5	2	7	2.20
FLVQ	4	4	8	4.59
AFLVQ$_{v2}$	2	5	7	5.57

The table below (Table 19) shows the comparative performance between the best adaptive LVQ and three other classifiers: AFLVQ comparative performance similar to the SVM classifier, inferior to the MLP model and superior to the KNN.

Table 19 – Overall classification performance results (ECG200).

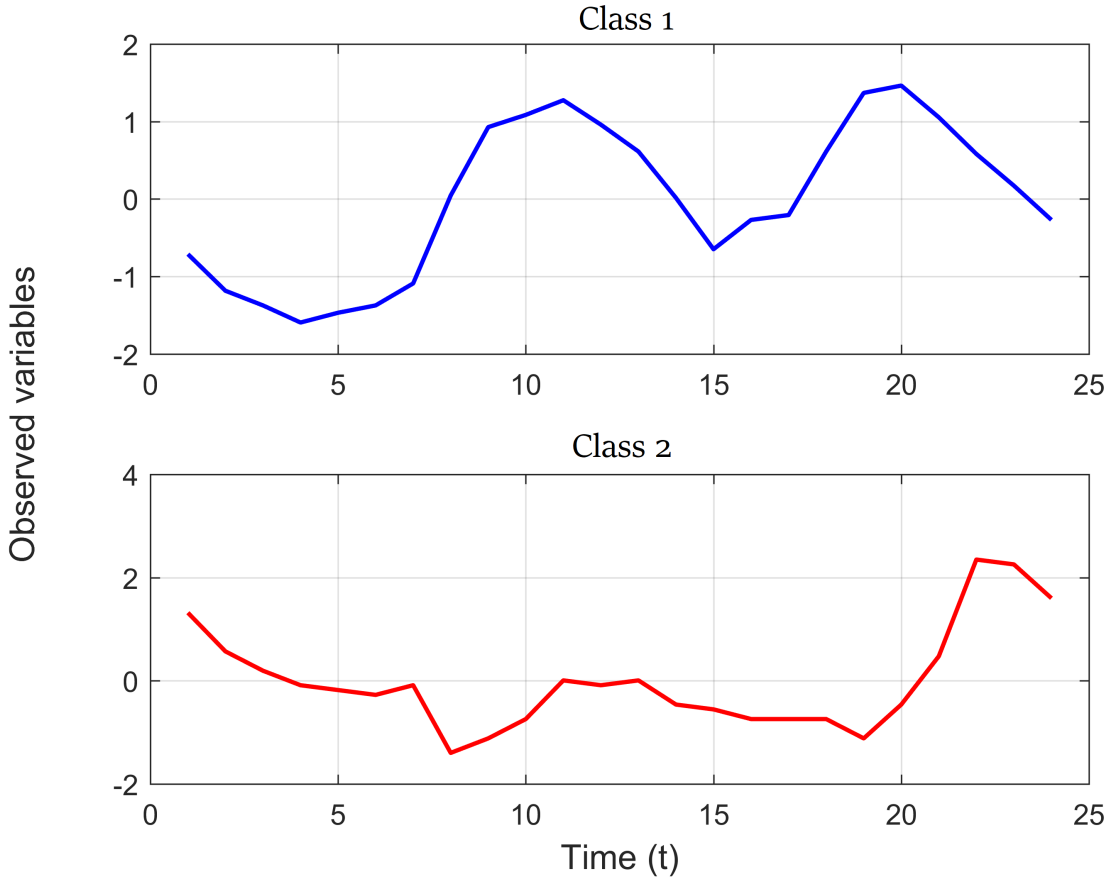
Algorithms	**Accuracy**	**Precision**	**Recall**	F_1 **Score**
AFLVQ$_{v2}$	92.00 ± 7.89	91.66	92.04	91.84
KNN	90.50 ± 6.85	90.26	88.12	89.13
SVM	92.50 ± 6.35	91.73	91.77	91.71
MLP	$\mathbf{95.00 \pm 4.77}$	**95.22**	**94.98**	**95.08**

From the general results of the classifiers, it can be noted that the problem tends to be solvable by non-linear models. In this cases, all classification models are non-linear and presented similar high accuracy. The classifier constructed by MLP approach presented the highest result, which suggests that this model presented greater capacity to transform the data into a non-linear space of greater data separability.

6.1.6 Italy Power Demand Dataset

The Italy Power Demand dataset is composed of data derived from twelve monthly electrical power demand time series from Italy, firstly introduced in Keogh *et al.* (2006). The classification problem consists in distinguishing days from the months from October to March (1) and from April to September (BAGNALL *et al.*, 2018). Figure 44 illustrates the times series for each class of Italy Power Demand dataset.

Figure 44 – Italy Power Demand time series



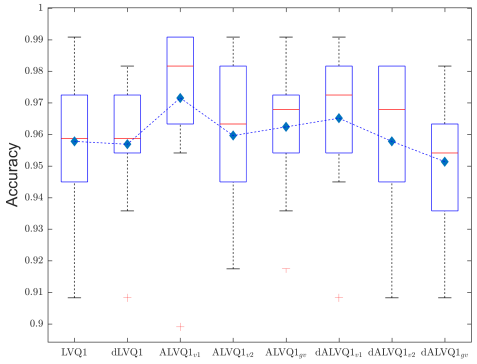
Source: The author.

Table 39 and Figure 45 show that in all cases the adaptive version had a higher average accuracy than the respective traditional LVQ models. The standard deviation of accuracy of the adaptive LVQ1, LVQ2 and LVQ3 models was smaller than the original models. The standard deviation of the adaptive FLVQ model was greater than the deviation from the original version.

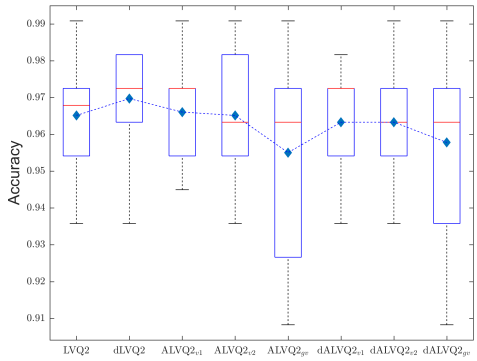
Table 20 – Overall LVQ-based classification performance results (Italy Power Demand).

Algorithms	**Accuracy**	**Precision**	**Recall**	**F_1 Score**
LVQ1	96.15 ± 2.11	96.14	96.14	96.14
ALVQ1$_{v2}$	96.61 ± 1.37	96.56	96.65	96.61
LVQ2	96.51 ± 1.49	96.46	96.56	96.51
dLVQ2	96.97 ± 1.37	96.96	97.01	96.98
LVQ3	58.44 ± 1.89	70.59	58.25	63.69
dALVQ3$_{v2}$	96.88 ± 1.38	96.84	96.89	96.87
FLVQ	96.88 ± 1.79	96.89	96.86	96.88
AFLVQ$_{v2}$	$\mathbf{97.16 \pm 2.85}$	**97.18**	**97.12**	**97.15**

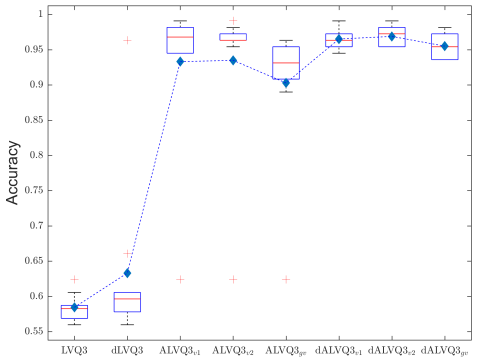
Figure 45 – Accuracy distribution for LVQ-based classifiers (Italy Power Demand)



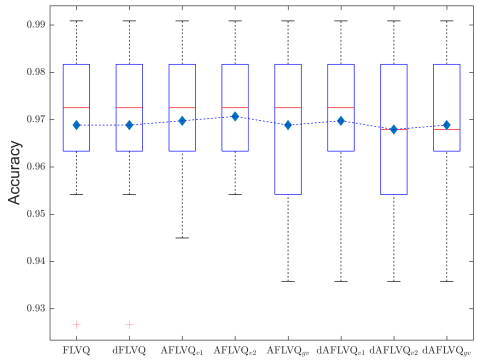
(a) LVQ1



(b) LVQ2



(c) LVQ3



(d) FLVQ

Source: The author.

Table 42 shows that there was little (from 18 to 17 in cases LVQ1 and LVQ3) or no reduction in the number of prototypes between adaptive LVQ versions and their original versions. The number of prototypes in all LVQ models (17 or 18) was higher than the number of dataset classes (2), which may have contributed to a better definition of the boundary between the two classes.

Table 21 – Overall cost regarding number of prototype and execution time of LVQ-based classification algorithms (Italy Power Demand).

Algorithms	P_{c_1}	P_{c_2}	N_P	Execution time (s)
LVQ1	9	9	18	9.55
ALVQ1$_{v2}$	8	9	17	13.54
LVQ2	9	9	18	9.71
dLVQ2	9	9	18	9.78
LVQ3	9	9	18	10.36
dALVQ3$_{v2}$	5	12	17	17.10
FLVQ	9	9	18	29.82
AFLVQ$_{v2}$	8	10	18	35.95

The table below (Table 22) shows the comparative performance between the best adaptive LVQ and three other classifiers: AFLVQ had a comparative performance similar to the KNN classifier, close to the SVM, and inferior to the MLP model.

Table 22 – Overall classification performance results (Italy Power Demand).

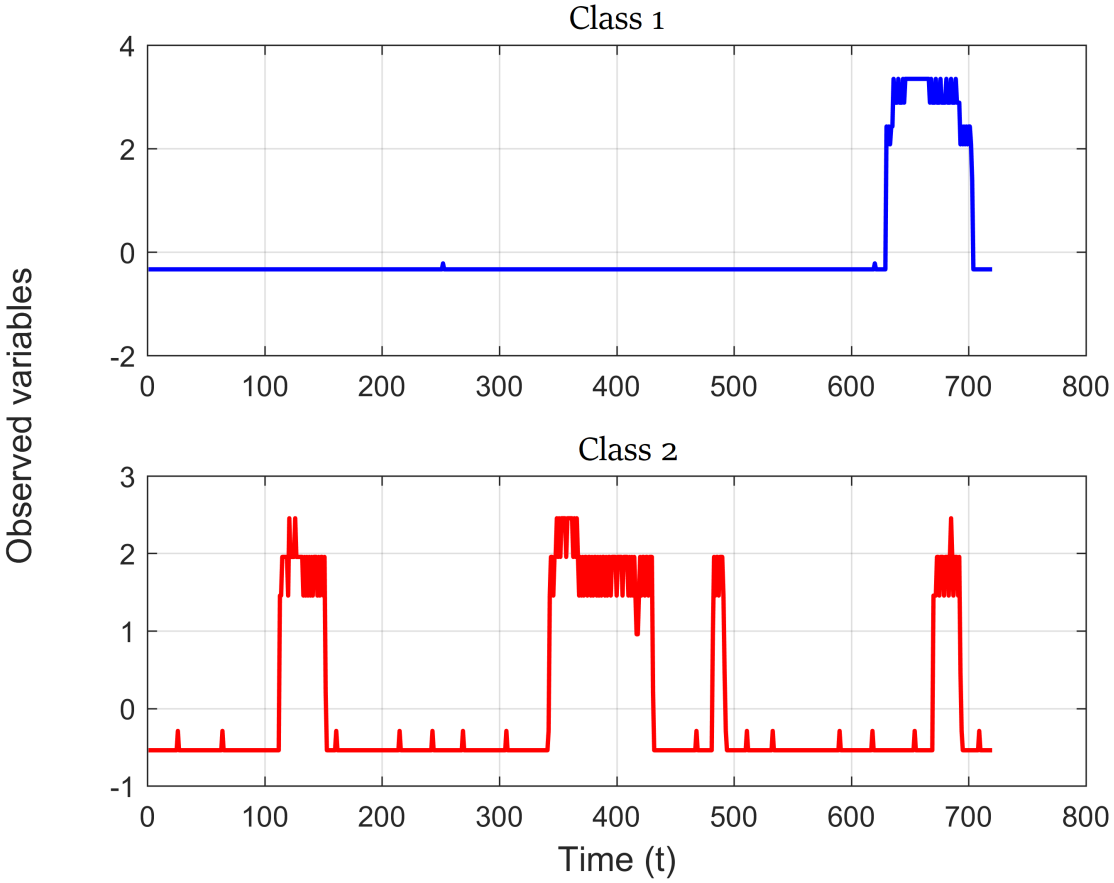
Algorithms	**Accuracy**	**Precision**	**Recall**	F_1 **Score**
AFLVQ$_{v2}$	97.16 ± 1.21	97.06	97.03	97.05
KNN	97.16 ± 2.00	97.19	97.14	97.17
SVM	97.52 ± 1.79	97.53	97.48	97.50
MLP	$\mathbf{98.35 \pm 1.28}$	**98.35**	**98.33**	**98.34**

The general results of the classifiers shows that the problem tends to be solvable by non-linear models. In this cases, all classification models are non-linear and presented high accuracy (close to 100%). The classifier constructed by MLP approach presented a slightly higher accuracy then the other classifiers. However, all classifiers have presented approximate classification performance.

6.1.7 Computers

Computers dataset is composed by recordings of behavioural data about how consumers use electricity within the home to help reduce the UK's carbon footprint, as part of government sponsored study called Powering the Nation. The data contains readings from 251 households, sampled in two-minute intervals over a month. Each series is length 720 (24 hours of readings taken every 2 minutes). The problem consist in identify two classes: Desktop (1) and Laptop (2) (BAGNALL *et al.*, 2018). Figure 46 illustrates the times series for each class of Computers dataset.

Figure 46 – Computers time series



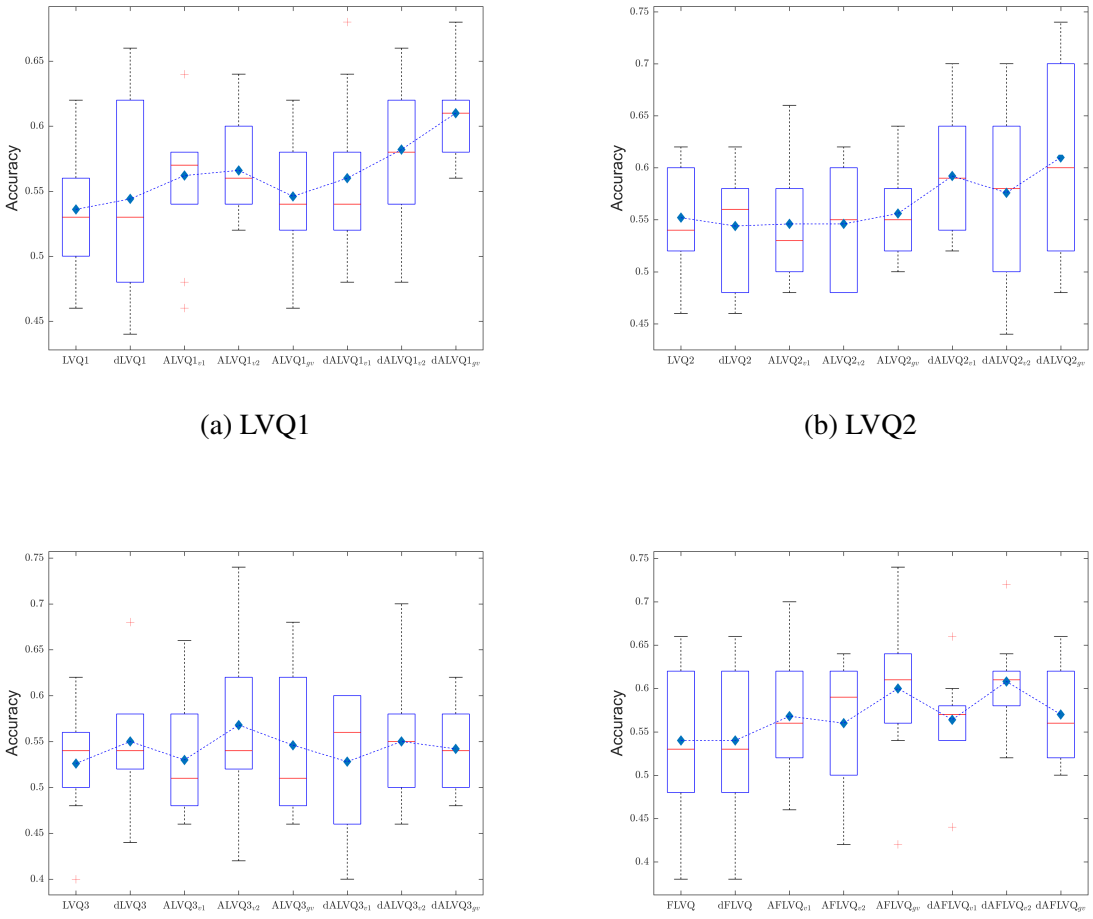
Source: The author.

Table 41 and Figure 47 show cases in which the adaptive version presented a higher average accuracy than the respective traditional LVQ models. The standard deviation of accuracy of the adaptive LVQ1 and FLVQ models was smaller than the original models. The standard deviation of the adaptive LVQ2 and LVQ3 models was greater than the deviations of the original versions.

Table 23 – Overall LVQ-based classification performance results (Computers).

Algorithms	**Accuracy**	**Precision**	**Recall**	**F_1 Score**
LVQ1	54.20 ± 4.94	55.00	55.00	55.00
dALVQ1$_{gv}$	$\mathbf{61.00 \pm 3.43}$	**61.62**	**61.49**	**61.56**
LVQ2	55.20 ± 5.18	55.37	55.30	55.33
dALVQ2$_{gv}$	61.00 ± 9.85	60.99	60.95	60.97
LVQ3	52.60 ± 6.04	51.92	52.67	51.95
ALVQ3$_{v2}$	56.80 ± 9.62	56.96	56.83	56.90
FLVQ	54.00 ± 8.74	54.54	54.39	54.47
dAFLVQ$_{v2}$	60.80 ± 5.27	61.24	61.07	61.15

Figure 47 – Accuracy distribution for LVQ-based classifiers (Computers)



(a) LVQ1

(b) LVQ2

(c) LVQ3

(d) FLVQ

Source: The author.

Table 42 shows that there was a small reduction (from 12 to 11) in the number of prototypes between the adaptive LVQ versions and their original versions.

Table 24 – Overall cost regarding number of prototype and execution time of LVQ-based classification algorithms (Computers).

Algorithms	P_{c_1}	P_{c_2}	N_P	Execution time (s)
LVQ1	6	6	12	19.14
dALVQ1$_{gv}$	8	3	11	39.23
LVQ2	6	6	12	17.60
dALVQ2$_{gv}$	7	4	11	35.67
LVQ3	6	6	12	24.44
ALVQ3$_{v2}$	3	8	11	44.73
FLVQ	6	6	12	34.71
dAFLVQ$_{v2}$	6	5	11	49.86

The table below (Table 25) shows the comparative performance between the best adaptive LVQ and three other classifiers: dAFLVQ had comparative performance similar to the KNN classifier, superior to SVM, and inferior to the MLP model. The number of prototypes in all LVQ models (11 or 12) was higher than the number of dataset classes (2), which may have contributed to a better definition of the boundary between the two classes.

Table 25 – Overall classification performance results (Computers).

Algorithms	**Accuracy**	**Precision**	**Recall**	F_1 **Score**
dALVQ1$_{gv}$	61.00 ± 3.43	61.62	61.49	61.56
KNN	60.60 ± 2.50	60.83	60.49	60.66
SVM	56.27 ± 4.63	56.27	57.89	57.06
MLP	$\mathbf{67.80 \pm 4.76}$	**67.80**	**68.61**	**68.20**

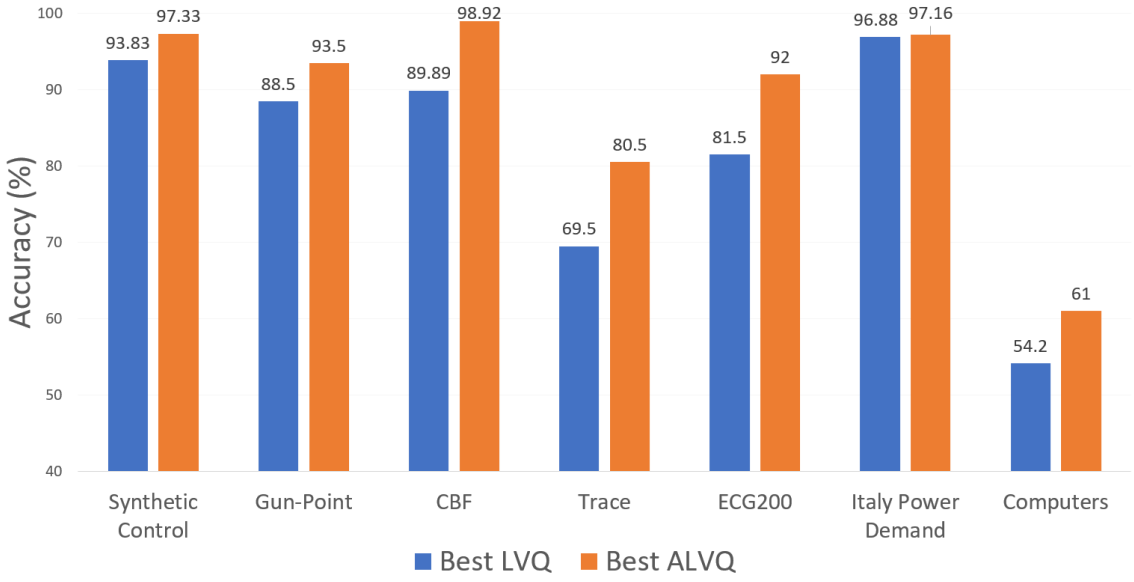
The general performance of the classifiers show that the classification models, in this case all nonlinear, were not able to classify the data and reach high accuracy indexes. This shows the disadvantage of using raw time series in model classification. In this case, it would be more interesting to apply feature extraction techniques to prepare the input vector for classification, so the classifiers would be able to better separate the data.

6.2 Discussion

The results have shown an enhanced improvement by employing the adaptive LVQ methods in comparison with the classic LVQs (LVQ1, LVQ2, LVQ3, and FLVQ). In all experiments, the best results in accuracy were attributed to one of the adaptive versions. Moreover, the results have shown that the precision and recall relationship calculated from the F1-score metric is balanced in the LVQ results, highlightning the method capacity in the detecting both false positive and false negative samples.

Figure 48 summarizes the comparison of the results obtained for each database, considering the best results obtained with the classic LVQ and the adaptive LVQ approach. As it can be seen, in the figure, the ALVQ has presented better results in all cases. However, in some scenarios, the ALVQ and LVQ presented very close results. Like for example in ItalyPowerDemand, whose difference in accuracy is only 0.28%.

Figure 48 – Comparison between the Adaptive-LVQ and classic LVQ approach



Source: The author.

Considering the accuracy distribution presented in the boxplots, it can be verified, in most graphs, a behavior where the first two boxes are practically equivalent, while others tend to increase in terms of accuracy. This shows that the dLVQ method presents results very similar to the classic versions, showing slight improvements. However, the other adaptive methods, based on ALVQ-SOM and ALVQ-GV, using strategies of removal and inclusion of prototypes, show very relevant improvements. Also, it is possible to notice that there are cases where there is overfitting using adaptive methods.

The results of the Trace dataset show that LVQ2 and LVQ3 variations outperformed their respective adaptive versions. This exemplifies an occurence of overfitting caused by the ALVQ (See Table 35).

By comparing each LVQ-based non-adaptive variation and the respective adaptive versions, it can be seen that the adaptive method present varieties in effectiveness, depending on the LVQ variation. For example, FLVQ variation applied on the dataset Synthetic Control has resulted similar results of *best* AFLVQ and FLVQ, varying only by 1.84% (See Figure 50). In contrast, the variation LVQ1 has presented a large diffence of 15%, comparing ALVQ1 and LVQ1 (See Figure 49). This variation is caused by the influence of the learning rule of each variation in classification performance.

Figure 49 – Comparison between the Adaptive-LVQ1 and classic LVQ1 approach

Source: The author.

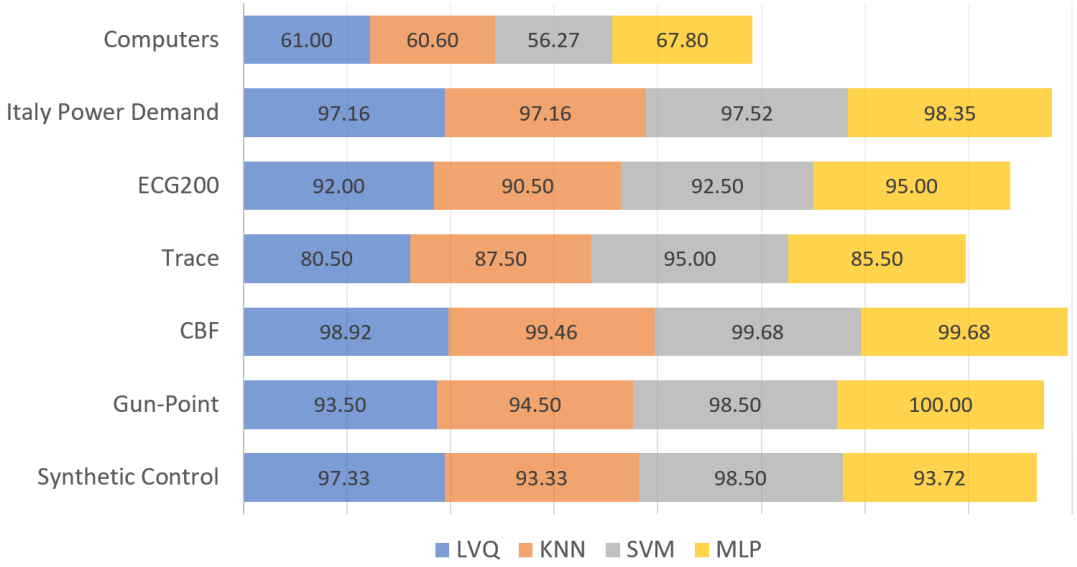
Figure 50 – Comparison between the Adaptive-FLVQ and classic FLVQ approach

Source: The author.

In general comparison among all classifiers, based on the results, it can be concluded that the LVQ methods has been more effective in comparison to the K-NN. However, LVQ-based classifiers seems to present lower accuracies than SVM and MLP, in most scenarios. However, the results show a certain equivalence between the outcomes obtained by the LVQ in comparison to the MLP and SVM, once in the majority dataset results their differences are not significantly high. The higher accuracy presented by MLP might be due to their capacity to perform non-linear transformations in input data, extracting abstract features that contribute to classification performance. Regarding SVM, its expressive results may be due to its capacity of dealing with high-dimensional input vectors.

As LVQ networks do not use any non-linear transformation strategy, it is expected that this method will show inferiority in some cases. In addition, LVQ-based classifiers depends on similarity measures, which may have a high degree of uncertainty in the context of time series, which is a important issue to be considered. Figure 51 presents a comparative bar graph of LVQ, MLP, SVM and KNN classifiers.

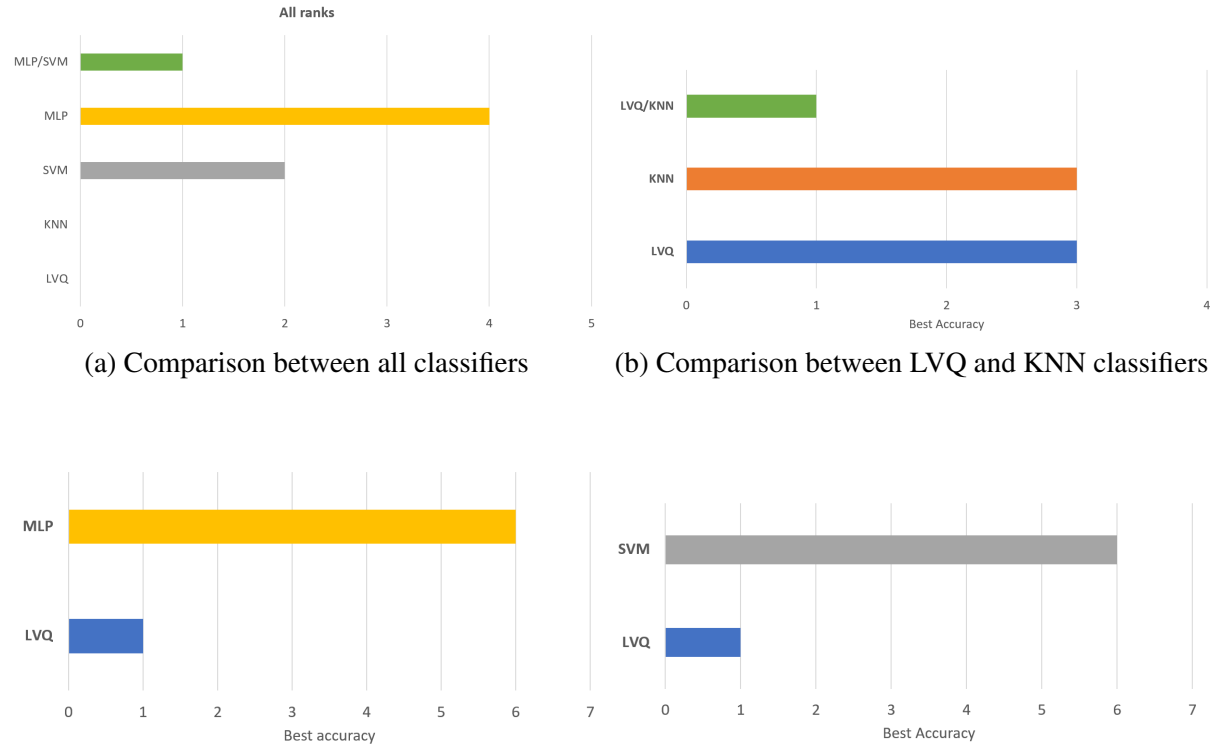
Figure 51 – Overall results of ALVQ, KNN, SVM and MLP classifiers for all 7 datasets



Source: The author.

In Figure 52, an illustration of comparative graphs comaring LVQ networks against all (a), LVQ against KNN (b), LVQ against MLP (c) and LVQ against SVM (d) is shown. These bar chars were built based on a rank comparision between classifiers, where it is verified how many times each classifier presented the best result. Regarding computational processing cost of non-adaptive and adaptive versions of LVQ, it was noticed that there is a trade-off between classification performance and execution time.

Figure 52 – Comparison between classifiers, considering a rank of best results



(a) Comparison between all classifiers

(b) Comparison between LVQ and KNN classifiers

(c) Comparison between MLP classifiers

(d) Comparison between SVM classifiers

Source: The author.

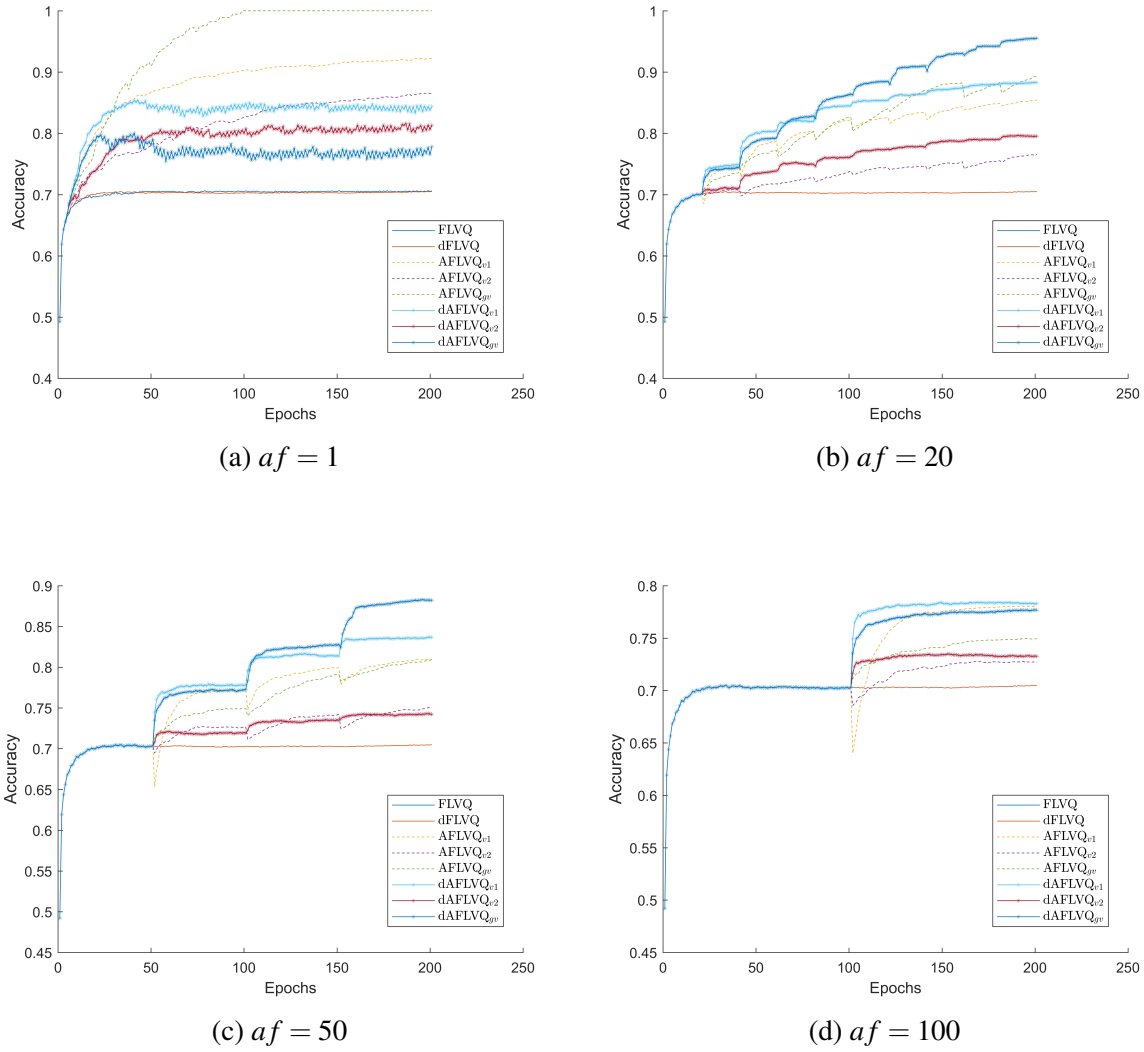
In general, the adaptive algorithms have a higher computational cost in terms of processing time. However, the improvement in classification performance compensates this cost. In addition, it has been noticed that, in some cases, the adaptive methods execute in less time than the classic ones. This depends on the stopping criterion chosen in the classifier training process. In the experiments carried out in this work, the training stop criterion was based on the number of epochs and a desired training accuracy. Therefore, the adaptive algorithms that obtained fast convergence, that is, achieve the desired accuracy, may fininsh the training in less time than a slow classic LVQ algorithms that do not converge. As an example, in Table 30 the dALVQ2$_{gv}$ has taken only 0.19 seconds for training, as the classic LVQ2 required 3.80 seconds.

Considering memory cost, reflected in the number of prototypes in the trained network, when $B > P_0$ there is more insertion of prototypes, therefore the memory cost increases. However, this may reflect in the improvement of the classification accuracy. In cases where $B = P_0$, the adaptive methods also outperformed non-adaptive algorithms, in spite of the limitation imposed by the hyper-parameter B. In fact, in some cases, besides having improved the classification performance, it was also verified the reduction of the quantity prototypes in the adaptive methods.

6.3 Adaptive factor (a_f) influence in training

For evaluating the influence of the hyperparamter Adaptive Factor (a_f) in learning, an experiment was conducted considering different values of a_f in order to analyze the learning curves convergences during adaptations. Figure 53 illustrates the evolution of the classification performance (train accuracy) over the epochs, considering multiple a_f values. The solid lines represent the algorithms which does not adapt by removing and/or including prototypes (LVQ and dLVQ). The dashed lines represent the adaptive algorithms without driven-learning strategy (ALVQ), and the line with stars are adaptive driven learning versions (dALVQ).

Figure 53 – Learning evolution for varying the hyperparameter af



(a) $af = 1$ (b) $af = 20$

(c) $af = 50$ (d) $af = 100$

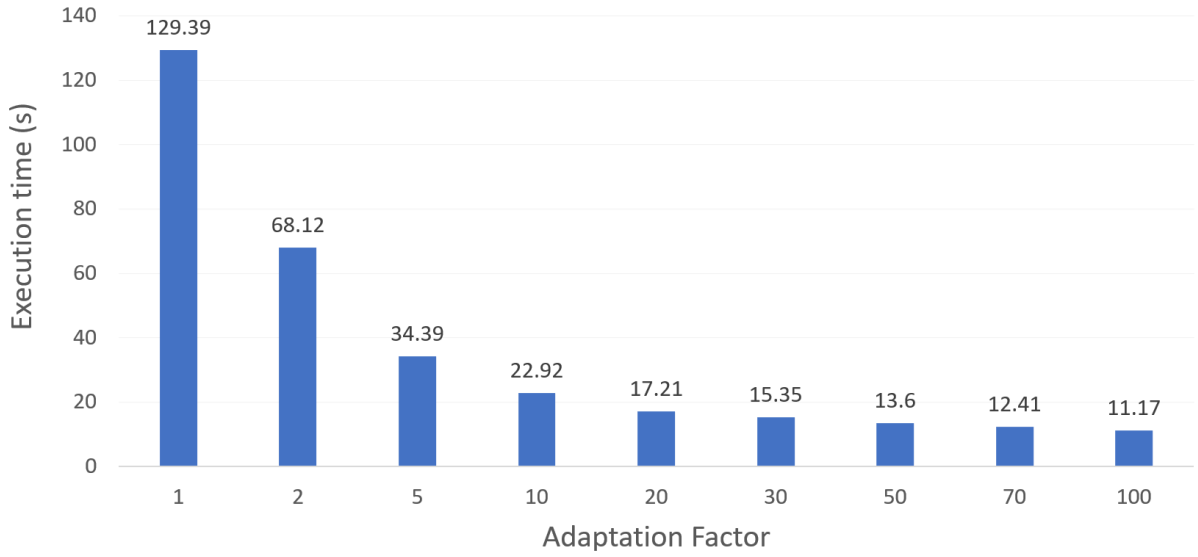
Source: The author.

In Figure 53(a), the learning curve does not clearly show the presence of adaptations. As $a_f = 1$, every epoch will be followed by an adaptation, producing an oscillatory learning.

The great advantage of using $af = 1$ lies in the velocity of convergence, which is quite superior compared to high values of a_f. However, the oscillatory behavior may cause a non-convergence of learning. The general behavior of the adaptive learning curve consists of radical growths at adaptation points. The lower the a_f, the greater will be the presence of "peaks"of learning. On the other hand, less frequent adaptations usually generate more expressive growths. Note that when the epoch is 100 the accuracy suddenly increases. This behaviour is caused by the process of adaptation.

Regarding processing cost (execution time), the obtained results have shown that increasing the number of adaptations during training increases the computational cost. As it can be seen in Figure 54, the lower the a_f the greater the training execution time. This increase in execution time is expected, since there is a computational cost involved in each adaptation. For low values of a_f, there will be more adaptations, so it will be more costly.

Figure 54 – Influence of Adaptive Factor (a_f) in execution time



Source: The author.

6.4 Summary

In this chapter, the results obtained in simulations using 7 datasets have been presented. Initially, the overall classification results were discussed, highlighting the best classifiers for each evaluated dataset. The performance regarding 10-Fold cross-validation tests were detailed in box-plots for analyzing the accuracy distribution among classifiers. Overall results have evidenced the efficiency of the adaptive-LVQ comparing to ordinary LVQ classifiers.

7 CONCLUSION AND FUTURE WORK

In this master thesis, the ALVQ-SOM have been proposed with the object to be applied in time series classification problems. This novel LVQ-based method is an alternative learning model for dealing with high-dimensional and complex datasets, where it is difficult to determine how many prototypes will be needed to build a classifier. ALVQ-SOM adopt a strategy for iterative verify the quality of prototypes. Therefore, it has the ability to preserve highly representative prototypes, while discards poorly adjusted prototypes.

The proposed ALVQ-SOM has outperformed in almost all scenarios, considering the LVQ variations tested. For several cases, certain LVQ variations presented poorly effective. In fact, the dLVQ variation has shown to be equivalent to classic version in most simulations. In contrast, the variations based on ALVQ-SOM and ALVQ-GV have shown significant improvements. For some datasets, the proposed adaptive LVQ method have surpassed more than 15% in test accuracy, which is a huge progress. Although ALVQ-SOM have outperform the classic LVQ variation, there are still some issues that must be addressed. For example, by analyzing the box-plots in the previous chapter, it can be seen that, in some cases, there is a trade-off between classification performance and accuracy variability. In other words, increasing the accuracy may lead to a increase in variability of the results.

Furthermore, the applicability of different neural network techniques to time series classification have been evaluated and compared. For this comparative study, the main focus was the classification accuracy. The experiment's results shown the artificial neural network are excellent alternative for dealing with time series classification problems. Among the ANN-based, the Multi-Layer Perceptron (MLP) have presented higher accuracy in almost all datasets. The *k*-NN benchmark classifier has also been evaluated. In our experiments, the ANN-based outperformed *k*-NN. In most cases the LVQ has shown general lower accuracy performance in comparing to MLP and SVM.

The underperform of the LVQ in comparison to the MLP and SVM methods can be explained by the importance of non-linear transformations of input pattern for time series classification. In addition, it is important to note that although the LVQ presented lower values than the MLP and SVM networks, in the majority of the results, their differences were small, around 2% difference, which makes this conclusion questionable.

Considering the computational effort of training, it can be concluded that there is an increased cost in processing when using the adaptive-LVQ versions. However, despite the inclusion of computational effort, improvements in classification performance are justifiable. In addition, considering the memory consumption, from the amount of prototypes resulting from the training, the proposed adaptive algorithm is an interesting choice to train LVQ-based methods with resource limitation, since even limiting the growth of network, the algorithm achieved expressive results.

Finally, this work confirms the viability of using ANN-based classifiers for time series classification problems. Furthermore, the experiment suggest that the proposed ALVQ-SOM have the potential of outperforming significantly the classic LVQ variations.

7.1 Future works

The proposed ALVQ-SOM models are quite flexible to allow various modifications for further increase the generalization accuracy. Some possible ideas for future works are listed below:

- To investigate alternative methods for inclusion of new prototypes. The classic Kohonen's SOM and hierarchical clustering have been tested. In this dissertation it was verified experimentally that the choice of such method directly reflects the quality of the obtained adaptive classifier. Therefore, an interesting direction for exploring the proposed ALVQ is to experiment the state-of-the-art time series clustering methods.
- To investigate alternative methods for evaluating prototype's quality. In this work, a score measure has been defined for evaluating prototypes in order to remove poorly representative neurons. This step in ALVQ-SOM is very important, as it is directly related to classification performance. By finding a more effective way of evaluating prototypes will improve the proposed algorithm's robustness.
- To deepen the studies on the hyper-parameters of the ALVQ-SOM network. Investigate alternative methods for selecting proper values of adaptive factor, growth factor and threshold removal.
- For further evaluate the classifiers studied in this work, it is intended to apply the algorithms in an real time application for evaluating it performances on online classification. For future researches, a suggestion is to adopt a problem called Human Activity Recognition (HAR) where human activities are classified based on data from motion sensors.

Furthermore, within machine learning area, a research field that is rapidly growing is the Deep Learning. Specifically in neural network approaches, several recent works employ Deep Learning network approaches (Deep Neural Networks) for solving TSC problems (WANG *et al.*, 2018; GU *et al.*, 2018; XIA *et al.*, 2018; KIM *et al.*, 2018; QI *et al.*, 2016). As discussed previously, time series are complex data. Generally, simpler classification methods require feature extraction techniques to prepare the input data in order to acquire a satisfactory performance. Working with temporal raw data usually requires more sophisticated algorithms. Therefore, Deep Learning-based algorithms are suitable for this goal. Hence, another objective for future works is in exploring deep learning approaches for complex time series classification problems.

REFERENCES

ADNANE, M.; JIANG, Z.; YAN, Z. Sleep–wake stages classification and sleep efficiency estimation using single-lead electrocardiogram. **Expert Systems with Applications**, Elsevier, v. 39, n. 1, p. 1401–1413, 2012.

AGHABOZORGI, S.; SHIRKHORSHIDI, A. S.; WAH, T. Y. Time-series clustering–a decade review. **Information Systems**, Elsevier, v. 53, p. 16–38, 2015.

AHA, D. W. **Lazy Learning**. Norwell, MA, USA: Kluwer Academic Publishers, 1997. ISBN 0-7923-4584-3.

ALCOCK, R. J.; MANOLOPOULOS, Y. *et al.* Time-series similarity queries employing a feature-based approach. In: **7th Hellenic conference on informatics**. Ioannina, Greece: [s.n.], 1999. p. 27–29.

ANGUITA, D.; GHIO, A.; ONETO, L.; PARRA, X.; REYES-ORTIZ, J. L. A public domain dataset for human activity recognition using smartphones. In: **ESANN**. [S.l.: s.n.], 2013.

ATALLAH, L.; LO, B.; KING, R.; YANG, G.-Z. Sensor positioning for activity recognition using wearable accelerometers. **IEEE transactions on biomedical circuits and systems**, IEEE, v. 5, n. 4, p. 320–329, 2011.

AYU, M. A.; ISMAIL, S. A.; MANTORO, T.; MATIN, A. F. A. Real-time activity recognition in mobile phones based on its accelerometer data. In: **2016 International Conference on Informatics and Computing (ICIC)**. [S.l.: s.n.], 2016. p. 292–297.

BAGNALL, A.; LINES, J.; BOSTROM, A.; LARGE, J.; KEOGH, E. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. **Data Mining and Knowledge Discovery**, Springer, v. 31, n. 3, p. 606–660, 2017.

BAGNALL, A.; LINES, J.; VICKERS, W.; KEOGH, E. **The UEA and UCR time series classification repository**. 2018.

BAO, L.; INTILLE, S. Activity recognition from user-annotated acceleration data. **Pervasive computing**, Springer, p. 1–17, 2004.

BERNDT, D. J.; CLIFFORD, J. Using dynamic time warping to find patterns in time series. In: SEATTLE, WA. **KDD workshop**. [S.l.], 1994. v. 10, n. 16, p. 359–370.

BHATIA, G.; RANI, S. Disease recognition and classification from movement patterns. In: **2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)**. [S.l.: s.n.], 2016. p. 3682–3687.

BISWAL, B.; BISWAL, M.; HASAN, S.; DASH, P. K. Nonstationary power signal time series data classification using lvq classifier. **Applied Soft Computing**, Elsevier, v. 18, p. 158–166, 2014.

BLOOMFIELD, P. **Fourier analysis of time series: an introduction**. [S.l.]: John Wiley & Sons, 2004.

BROCKWELL, P. J.; DAVIS, R. A. **Time series: theory and methods**. [S.l.]: Springer Science & Business Media, 2013.

BUENAVENTURA, C. V. S.; TIGLAO, N. M. C. Basic human activity recognition based on sensor fusion in smartphones. In: **2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)**. [S.l.: s.n.], 2017. p. 1182–1185.

BUNTINE, W. Myths and legends in learning classification rules. 1990.

BURGES, C. J. A tutorial on support vector machines for pattern recognition. **Data mining and knowledge discovery**, Springer, v. 2, n. 2, p. 121–167, 1998.

CASALE, P.; PUJOL, O.; RADEVA, P. Human activity recognition from accelerometer data using a wearable device. **Pattern Recognition and Image Analysis**, Springer, p. 289–296, 2011.

CHAN, K.-P.; FU, A. W.-C. Efficient time series matching by wavelets. In: **Proceedings 15th International Conference on Data Engineering (Cat. No.99CB36337)**. [S.l.: s.n.], 1999. p. 126–133. ISSN 1063-6382.

CHANDOLA, V.; BANERJEE, A.; KUMAR, V. Anomaly detection: A survey. **ACM computing surveys (CSUR)**, ACM, v. 41, n. 3, p. 15, 2009.

CHANDOLA, V.; CHEBOLI, D.; KUMAR, V. Detecting anomalies in a time series database. **Computer Science Department, University of Minnesota, Tech. Rep**, 2009.

CHANG, C.-C.; LIN, C.-J. Libsvm: a library for support vector machines. **ACM transactions on intelligent systems and technology (TIST)**, Acm, v. 2, n. 3, p. 27, 2011.

CHAVAN, A.; KOLTE, M. Eeg signals classification and diagnosis using wavelet transform and artificial neural network. In: **2017 International Conference on Nascent Technologies in Engineering (ICNTE)**. [S.l.: s.n.], 2017. p. 1–6.

CHEN, Y.; KEOGH, E.; HU, B.; BEGUM, N.; BAGNALL, A.; MUEEN, A.; BATISTA, G. **The UCR Time Series Classification Archive**. 2015. <www.cs.ucr.edu/~eamonn/time_series_data/ >.

CHUNG, F.-L.; LEE, T. Fuzzy learning vector quantization. In: **Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan)**. [S.l.: s.n.], 1993. v. 3, p. 2739–2743 vol.3.

CINLAR, E. **Introduction to stochastic processes**. [S.l.]: Courier Corporation, 2013.

CORTES, C.; VAPNIK, V. Support-vector networks. **Machine learning**, Springer, v. 20, n. 3, p. 273–297, 1995.

DELGADO, L.; HERNANDEZ, G.; ZAMORA, E.; SOSSA, H.; BARRETO, A.; RAMOS, F.; REYES, R. Classification of the estrous cycle through texture and shape features. In: **2017 IEEE Symposium Series on Computational Intelligence (SSCI)**. [S.l.: s.n.], 2017. p. 1–7.

DO, C.-T.; DOUZAL-CHOUAKRIA, A.; MARIÉ, S.; ROMBAUT, M.; VARASTEH, S. Multi-modal and multi-scale temporal metric learning for a robust time series nearest neighbors classification. **Information Sciences**, Elsevier, v. 418, p. 272–285, 2017.

DOBROWOLSKI, A.; SUCHOCKI, M.; TOMCZYKIEWICZ, K.; MAJDA-ZDANCEWICZ, E. Classification of auditory brainstem response using wavelet decomposition and svm network. **Biocybernetics and Biomedical Engineering**, Elsevier, v. 36, n. 2, p. 427–436, 2016.

DOUC, R.; MOULINES, E.; STOFFER, D. **Nonlinear time series: Theory, methods and applications with R examples**. [S.l.]: CRC Press, 2014.

DUARTE, F.; LOURENÇO, A.; ABRANTES, A. Classification of physical activities using a smartphone: evaluation study using multiple users. **Procedia Technology**, Elsevier, v. 17, p. 239–247, 2014.

EKICI, S. Support vector machines for classification and locating faults on transmission lines. **Applied Soft Computing**, Elsevier, v. 12, n. 6, p. 1650–1658, 2012.

ESLING, P.; AGON, C. Time-series data mining. **ACM Computing Surveys (CSUR)**, ACM, v. 45, n. 1, p. 12, 2012.

FAKHRAZARI, A.; VAKILZADIAN, H. A survey on time series data mining. In: **2017 IEEE International Conference on Electro Information Technology (EIT)**. [S.l.: s.n.], 2017. p. 476–481.

FEJFAR, J.; ŠT'ASTNỲ, J.; CEPL, M. *et al.* Time series classification using k-nearest neighbours, multilayer perceptron and learning vector quantization algorithms. **Acta Universitatis Agriculturae et Silviculturae Mendelianae Brunensis**, Mendel University Press, v. 60, n. 2, p. 69–72, 2013.

FU, T.-c. A review on time series data mining. **Engineering Applications of Artificial Intelligence**, Elsevier, v. 24, n. 1, p. 164–181, 2011.

GIUSTI, R. **Classicação de séries temporais utilizando diferentes representações de dados e ensembles**. Tese (Doutorado) — Universidade de São Paulo, 2017.

GRBOVIC, M.; VUCETIC, S. Learning vector quantization with adaptive prototype addition and removal. In: IEEE. **Neural Networks, 2009. IJCNN 2009. International Joint Conference on**. [S.l.], 2009. p. 994–1001.

GU, J.; WANG, Z.; KUEN, J.; MA, L.; SHAHROUDY, A.; SHUAI, B.; LIU, T.; WANG, X.; WANG, G.; CAI, J.; CHEN, T. Recent advances in convolutional neural networks. **Pattern Recognition**, v. 77, p. 354 – 377, 2018. ISSN 0031-3203. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0031320317304120>.

HAJINOROOZI, M.; MAO, Z.; JUNG, T.-P.; LIN, C.-T.; HUANG, Y. Eeg-based prediction of driver's cognitive performance by deep convolutional neural network. **Signal Processing: Image Communication**, Elsevier, v. 47, p. 549–555, 2016.

HAN, J.; PEI, J.; KAMBER, M. **Data mining: concepts and techniques**. [S.l.]: Elsevier, 2011.

HAYKIN, S. S.; HAYKIN, S. S.; HAYKIN, S. S.; HAYKIN, S. S. **Neural networks and learning machines**. [S.l.]: Pearson Upper Saddle River, NJ, USA:, 2009. v. 3.

HILLS, J.; LINES, J.; BARANAUSKAS, E.; MAPP, J.; BAGNALL, A. Classification of time series by shapelet transformation. **Data Mining and Knowledge Discovery**, 2013.

IGLESIAS, F.; KASTNER, W. Analysis of similarity measures in times series clustering for the discovery of building energy patterns. **Energies**, Multidisciplinary Digital Publishing Institute, v. 6, n. 2, p. 579–597, 2013.

ISCAN, Z.; DOKUR, Z.; DEMIRALP, T. Classification of electroencephalogram signals with combined time and frequency features. **Expert Systems with Applications**, Elsevier, v. 38, n. 8, p. 10499–10505, 2011.

JAIN, B. J.; SCHULTZ, D. Asymmetric learning vector quantization for efficient nearest neighbor classification in dynamic time warping spaces. **Pattern Recognition**, Elsevier, v. 76, p. 349–366, 2018.

JEN, K.; HWANG, Y. *et al.* Ecg feature extraction and classification using cepstrum and neural networks. **Journal of Medical and Biological Engineering**, WALTER H CHANG, v. 28, n. 1, p. 31, 2008.

JENSEN, F. V. **An introduction to Bayesian networks**. [S.l.]: UCL press London, 1996. v. 210.

JEONG, Y.-S.; JAYARAMAN, R. Support vector-based algorithms with weighted dynamic time warping kernel function for time series classification. **Knowledge-based systems**, Elsevier, v. 75, p. 184–191, 2015.

JIAO, Z.; GAO, X.; WANG, Y.; LI, J.; XU, H. Deep convolutional neural networks for mental load classification based on eeg data. **Pattern Recognition**, Elsevier, v. 76, p. 582–595, 2018.

JIMéNEZ Álvaro B.; LáZARO, J. L.; DORRONSORO, J. R. Finding optimal model parameters by deterministic and annealed focused grid search. **Neurocomputing**, v. 72, n. 13, p. 2824 – 2832, 2009. ISSN 0925-2312. Hybrid Learning Machines (HAIS 2007) / Recent Developments in Natural Computation (ICNC 2007). Disponível em: <http://www.sciencedirect.com/science/article/pii/S0925231209001222>.

KAVITHA, V.; PUNITHAVALLI, M. Clustering time series data stream-a literature survey. **arXiv preprint arXiv:1005.4270**, 2010.

KEOGH, E.; CHAKRABARTI, K.; PAZZANI, M.; MEHROTRA, S. Dimensionality reduction for fast similarity search in large time series databases. **Knowledge and information Systems**, Springer, v. 3, n. 3, p. 263–286, 2001.

KEOGH, E.; WEI, L.; XI, X.; LONARDI, S.; SHIEH, J.; SIROWY, S. Intelligent icons: Integrating lite-weight data mining and visualization into gui operating systems. In: **Sixth International Conference on Data Mining (ICDM'06)**. [S.l.: s.n.], 2006. p. 912–916. ISSN 1550-4786.

KIM, H. B.; LEE, W. W.; KIM, A.; LEE, H. J.; PARK, H. Y.; JEON, H. S.; KIM, S. K.; JEON, B. S.; PARK, K. S. Wrist sensor-based tremor severity quantification in parkinson's disease using convolutional neural network. **Computers in Biology and Medicine**, Elsevier, 2018.

KIRCHGÄSSNER, G.; WOLTERS, J. **Introduction to modern time series analysis**. [S.l.]: Springer Science & Business Media, 2007.

KLEIST, C. **Time series data mining methods**. Dissertação (Mestrado) — Humboldt-Universität zu Berlin, Wirtschaftswissenschaftliche Fakultät, 2015.

KOHONEN, T. The self-organizing map. **Proceedings of the IEEE**, IEEE, v. 78, n. 9, p. 1464–1480, 1990.

KOHONEN, T. **Self-Organizing Maps**. [S.l.]: Springer Berlin Heidelberg, 1997.

KOHONEN, T.; BARNA, G.; CHRISLEY, R. Statistical pattern recognition with neural networks: Benchmarking studies. In: **IEEE International Conference on Neural Networks**. [S.l.: s.n.], 1988. v. 1, p. 61–68.

KOTSIANTIS, S. B.; ZAHARAKIS, I. D.; PINTELAS, P. E. Machine learning: a review of classification and combining techniques. **Artificial Intelligence Review**, Kluwer Academic Publishers, v. 26, n. 3, p. 159–190, 2006.

KUMAR, S.; SHARMA, A.; MAMUN, K.; TSUNODA, T. A deep learning approach for motor imagery eeg signal classification. In: **2016 3rd Asia-Pacific World Congress on Computer Science and Engineering (APWC on CSE)**. [S.l.: s.n.], 2016. p. 34–39.

LEE, Y.-H.; WEI, C.-P.; CHENG, T.-H.; YANG, C.-T. Nearest-neighbor-based approach to time-series classification. **Decision Support Systems**, Elsevier, v. 53, n. 1, p. 207–217, 2012.

LESTER, J.; CHOUDHURY, T.; BORRIELLO, G. A practical approach to recognizing physical activities. **Pervasive Computing**, Springer, p. 1–16, 2006.

LIAO, T. W. Clustering of time series data—a survey. **Pattern recognition**, Elsevier, v. 38, n. 11, p. 1857–1874, 2005.

LIN, J.; KEOGH, E.; WEI, L.; LONARDI, S. Experiencing sax: a novel symbolic representation of time series. **Data Mining and knowledge discovery**, Springer, v. 15, n. 2, p. 107–144, 2007.

LIU, D.; QIAN, H.; DAI, G.; ZHANG, Z. An iterative svm approach to feature selection and classification in high-dimensional datasets. **Pattern Recognition**, Elsevier, v. 46, n. 9, p. 2531–2537, 2013.

LIU, X.; DU, H.; WANG, G.; ZHOU, S.; ZHANG, H. Automatic diagnosis of premature ventricular contraction based on lyapunov exponents and lvq neural network. **Computer methods and programs in biomedicine**, Elsevier, v. 122, n. 1, p. 47–55, 2015.

MANNINI, A.; SABATINI, A. M. Machine learning methods for classifying human physical activity from on-body accelerometers. **Sensors**, Molecular Diversity Preservation International, v. 10, n. 2, p. 1154–1175, 2010.

MANTARAS, R. L. D.; ARMENGOL, E. Machine learning from examples: Inductive and lazy methods. **Data & Knowledge Engineering**, Elsevier, v. 25, n. 1-2, p. 99–123, 1998.

MARTIN-DIAZ, I.; MORINIGO-SOTELO, D.; DUQUE-PEREZ, O.; ROMERO-TRONCOSO, R. D. J. Advances in classifier evaluation: Novel insights for an electric data-driven motor diagnosis. **IEEE Access**, IEEE, v. 4, p. 7028–7038, 2016.

MATTOS, C. L. C. **Comitês de Classificadores Baseados nas Redes SOM e Fuzzy ART com Sintonia de Parâmetros e Seleção de Atributos via Metaheurísticas Evolucionárias**. Tese (Doutorado) — Dissertação, Universidade Federal do Ceará, 2011.

MELGANI, F.; BAZI, Y. Classification of electrocardiogram signals with support vector machines and particle swarm optimization. **IEEE transactions on information technology in biomedicine**, IEEE, v. 12, n. 5, p. 667–677, 2008.

MELIN, P.; AMEZCUA, J.; VALDEZ, F.; CASTILLO, O. A new neural network model based on the lvq algorithm for multi-class classification of arrhythmias. **Information sciences**, Elsevier, v. 279, p. 483–497, 2014.

MERRIAM-WEBSTER. **Merriam-Webster's collegiate dictionary**. [S.l.]: Merriam-Webster, 2004.

MOKBEL, B.; PAASSEN, B.; SCHLEIF, F.-M.; HAMMER, B. Metric learning for sequences in relational lvq. **Neurocomputing**, Elsevier, v. 169, p. 306–322, 2015.

MÜLLER, M. **Information retrieval for music and motion**. [S.l.]: Springer, 2007. v. 2.

MURTHY, S. K. Automatic construction of decision trees from data: A multi-disciplinary survey. **Data mining and knowledge discovery**, Kluwer academic publishers, v. 2, n. 4, p. 345–389, 1998.

NAKANO, K.; CHAKRABORTY, B. Effect of dynamic feature for human activity recognition using smartphone sensors. In: **2017 IEEE 8th International Conference on Awareness Science and Technology (iCAST)**. [S.l.: s.n.], 2017. p. 539–543.

NASIBOV, E. N.; PEKER, S. Time series labeling algorithms based on the k-nearest neighbors' frequencies. **Expert Systems with Applications**, Elsevier, v. 38, n. 5, p. 5028–5035, 2011.

NOVIYANTO, A.; ARYMURTHY, A. M. Sleep stages classification based on temporal pattern recognition in neural network approach. In: **The 2012 International Joint Conference on Neural Networks (IJCNN)**. [S.l.: s.n.], 2012. p. 1–6. ISSN 2161-4407.

OLSZEWSKI, R. T. **Generalized feature extraction for structural pattern recognition in time-series data**. [S.l.], 2001.

ORHAN, U.; HEKIM, M.; OZER, M. Eeg signals classification using the k-means clustering and a multilayer perceptron neural network model. **Expert Systems with Applications**, Elsevier, v. 38, n. 10, p. 13475–13481, 2011.

ORSENIGO, C.; VERCELLIS, C. Combining discrete svm and fixed cardinality warping distances for multivariate time series classification. **Pattern Recognition**, Elsevier, v. 43, n. 11, p. 3787–3794, 2010.

PARVINNIA, E.; SABETI, M.; JAHROMI, M. Z.; BOOSTANI, R. Classification of eeg signals using adaptive weighted distance nearest neighbor algorithm. **Journal of King Saud University-Computer and Information Sciences**, Elsevier, v. 26, n. 1, p. 1–6, 2014.

PERES, S. M.; ROCHA, T.; BISCARO, H. H.; MADEO, R. C. B.; BOSCARIOLI, C. Tutorial sobre fuzzy-c-means e fuzzy learning vector quantization: Abordagens híbridas para tarefas de agrupamento e classificação. **Revista de Informática Teórica e Aplicada**, v. 19, n. 1, p. 120–163, 2012.

QI, Z.; WANG, B.; TIAN, Y.; ZHANG, P. When ensemble learning meets deep learning: a new deep support vector machine for classification. **Knowledge-Based Systems**, Elsevier, v. 107, p. 54–60, 2016.

RABINER, L. R.; JUANG, B.-H. **Fundamentals of speech recognition**. [S.l.]: PTR Prentice Hall Englewood Cliffs, 1993. v. 14.

RAJESH, K. N.; DHULI, R. Classification of ecg heartbeats using nonlinear decomposition methods and support vector machine. **Computers in biology and medicine**, Elsevier, v. 87, p. 271–284, 2017.

RALANAMAHATANA, C. A.; LIN, J.; GUNOPULOS, D.; KEOGH, E.; VLACHOS, M.; DAS, G. Mining time series data. In: **Data mining and knowledge discovery handbook**. [S.l.]: Springer, 2005. p. 1069–1103.

RONAO, C. A.; CHO, S. Human activity recognition using smartphone sensors with two-stage continuous hidden markov models. In: **2014 10th International Conference on Natural Computation (ICNC)**. [S.l.: s.n.], 2014. p. 681–686. ISSN 2157-9563.

ROSA, I. C. R. da; MATTA, C. E. da. Classificação de sinais eletrocardiográficos usando redes neurais artificiais. 2000.

SAITO, N. Local feature extraction and its applications using a library of bases. In: **Topics in Analysis and Its Applications: Selected Theses**. [S.l.]: World Scientific, 2000. p. 269–451.

SALAHSHOOR, K.; KORDESTANI, M.; KHOSHRO, M. S. Fault detection and diagnosis of an industrial steam turbine using fusion of svm (support vector machine) and anfis (adaptive neuro-fuzzy inference system) classifiers. **Energy**, Elsevier, v. 35, n. 12, p. 5472–5482, 2010.

SENIN, P. Dynamic time warping algorithm review. **Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA**, Citeseer, v. 855, p. 1–23, 2008.

SHARMA, R.; PACHORI, R. B. Classification of epileptic seizures in eeg signals based on phase space representation of intrinsic mode functions. **Expert Systems with Applications**, Elsevier, v. 42, n. 3, p. 1106–1117, 2015.

SMITH, A.; EVANS, M.; DOWNEY, J. Modulation classification of satellite communication signals using cumulants and neural networks. In: IEEE. **Cognitive Communications for Aerospace Applications Workshop (CCAA), 2017**. [S.l.], 2017. p. 1–8.

SORS, A.; BONNET, S.; MIREK, S.; VERCUEIL, L.; PAYEN, J.-F. A convolutional neural network for sleep stage scoring from raw single-channel eeg. **Biomedical Signal Processing and Control**, Elsevier, v. 42, p. 107–114, 2018.

SORZANO, C. O. S.; VARGAS, J.; MONTANO, A. P. A survey of dimensionality reduction techniques. **arXiv preprint arXiv:1403.2877**, 2014.

SU, W.-T.; PING, X.-O.; TSENG, Y.-J.; LAI, F. Multiple time series data processing for classification with period merging algorithm. **Procedia Computer Science**, Elsevier, v. 37, p. 301–308, 2014.

TAQI, A. M.; AL-AZZO, F.; MARIOFANNA, M.; AL-SAADI, J. M. Classification and discrimination of focal and non-focal eeg signals based on deep neural network. In: **2017 International Conference on Current Research in Computer Science and Information Technology (ICCIT)**. [S.l.: s.n.], 2017. p. 86–92.

VLACHOS, M.; KOLLIOS, G.; GUNOPULOS, D. Discovering similar multidimensional trajectories. In: **Proceedings 18th International Conference on Data Engineering**. [S.l.: s.n.], 2002. p. 673–684. ISSN 1063-6382.

WANG, J.; CHEN, Y.; HAO, S.; PENG, X.; HU, L. Deep learning for sensor-based activity recognition: A survey. **Pattern Recognition Letters**, p. –, 2018. ISSN 0167-8655. Disponível em: <https://www.sciencedirect.com/science/article/pii/S016786551830045X>.

WANG, X.; MUEEN, A.; DING, H.; TRAJCEVSKI, G.; SCHEUERMANN, P.; KEOGH, E. Experimental comparison of representation methods and distance measures for time series data. **Data Mining and Knowledge Discovery**, Springer, v. 26, n. 2, p. 275–309, 2013.

WANG, Z.; XUE, X. Multi-class support vector machine. In: **Support Vector Machines Applications**. [S.l.]: Springer, 2014. p. 23–48.

XI, X.; KEOGH, E.; SHELTON, C.; WEI, L.; RATANAMAHATANA, C. A. Fast time series classification using numerosity reduction. In: ACM. **Proceedings of the 23rd international conference on Machine learning**. [S.l.], 2006. p. 1033–1040.

XIA, Y.; WULAN, N.; WANG, K.; ZHANG, H. Detecting atrial fibrillation by deep convolutional neural networks. **Computers in biology and medicine**, Elsevier, v. 93, p. 84–92, 2018.

XING, Z.; PEI, J.; KEOGH, E. A brief survey on sequence classification. **ACM Sigkdd Explorations Newsletter**, ACM, v. 12, n. 1, p. 40–48, 2010.

XU, J.; TANG, L.; LI, T. System situation ticket identification using svms ensemble. **Expert Systems with Applications**, Elsevier, v. 60, p. 130–140, 2016.

ZHANG, G. P. Neural networks for classification: a survey. **IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)**, IEEE, v. 30, n. 4, p. 451–462, 2000.

ŞTEFAN, R.-M. A comparison of data classification methods. **Procedia Economics and Finance**, v. 3, p. 420 – 425, 2012. ISSN 2212-5671. International Conference Emerging Markets Queries in Finance and Business, Petru Maior University of Tîrgu-Mures, ROMANIA, October 24th - 27th, 2012. Disponível em: <http://www.sciencedirect.com/science/article/pii/S2212567112001748>.

APPENDIX A – SUB-OPTIMAL PARAMETERS FOR *K*-NN, MLP AND SVM

This appendix contain the tables composing all the experiments performed with the LVQ-based algorithms..

Table 26 – Sub-optimal parameters for *k*-NN classifier for each dataset

Datasets	k^*	*Accuracy*
Synthetic Control	2	93.33 ± 03.04
Gun-Point	1	94.50 ± 03.69
CBF	9	99.68 ± 00.26
Trace	1	87.50 ± 04.86
ECG200	3	90.50 ± 06.85
Italy Power Demand	5	97.16 ± 02.00
Computers	9	60.60 ± 2.50

Table 27 – Sub-optimal parameters for SVM classifier for each dataset

Datasets	C^*	γ^*	*Accuracy*
Synthetic Control	2^{-1}	2^{-4}	98.50 ± 00.66
Gun-Point	2^{2}	2^{-4}	98.50 ± 00.41
CBF	2^{3}	2^{-4}	99.67 ± 00.52
Trace	2^{3}	2^{-3}	95.00 ± 02.27
ECG200	2^{1}	2^{-3}	92.50 ± 05.40
Italy Power Demand	2^{-1}	2^{-3}	97.52 ± 01.78
Computers	2^{-1}	2^{1}	56.27 ± 04.63

Table 28 – Sub-optimal parameters for MLP classifier for each dataset

Datasets	N_e^*	*Accuracy*
Synthetic Control	5	93.72 ± 02.86
Gun-Point	14	100.00 ± 00.00
CBF	9	99.67 ± 00.21
Trace	15	85.50 ± 07.62
ECG200	3	97.00 ± 01.50
Italy Power Demand	14	96.79 ± 1.55
Computers	10	67.80 ± 4.75

APPENDIX B – OVERALL RESULT TABLES ($B = P_0$)

Table 29 – Overall LVQ-based classification performance results (Synthetic Control).

Algorithms	Accuracy	Precision	Recall	F_1 Score
LVQ1	82.33 ± 7.79	74.52	82.82	78.36
dLVQ1	88.17 ± 9.41	89.02	89.55	89.24
ALVQ1$_{v1}$	96.83 ± 1.66	97.27	96.79	97.02
ALVQ1$_{v2}$	97.17 ± 2.23	97.72	97.27	97.49
ALVQ1$_{gv}$	96.17 ± 2.09	96.54	96.34	96.44
dALVQ1$_{v1}$	96.83 ± 3.37	97.05	97.07	97.06
dALVQ1$_{v2}$	97.33 ± 3.53	97.61	97.58	97.59
dALVQ1$_{gv}$	96.50 ± 3.64	96.57	96.52	96.54
LVQ2	63.17 ± 11.15	45.55	61.14	51.79
dLVQ2	70.33 ± 17.70	58.20	69.59	62.88
ALVQ2$_{v1}$	66.67 ± 12.67	50.53	63.99	55.95
ALVQ2$_{v2}$	67.33 ± 13.41	50.78	64.51	56.33
ALVQ2$_{gv}$	67.83 ± 14.08	52.42	65.20	57.52
dALVQ2$_{v1}$	79.00 ± 21.26	71.61	79.13	74.58
dALVQ2$_{v2}$	81.67 ± 22.40	75.69	82.31	78.22
dALVQ2$_{gv}$	83.67 ± 21.78	79.58	83.67	80.99
LVQ3	59.67 ± 14.61	42.24	59.46	48.96
dLVQ3	61.00 ± 9.72	46.36	62.48	53.18
ALVQ3$_{v1}$	88.17 ± 16.54	83.24	87.99	85.09
ALVQ3$_{v2}$	92.17 ± 15.48	89.70	92.63	90.76
ALVQ3$_{gv}$	88.67 ± 17.12	84.66	88.77	86.16
dALVQ3$_{v1}$	91.00 ± 16.33	87.89	91.57	89.43
dALVQ3$_{v2}$	89.00 ± 16.14	85.18	89.42	86.96
dALVQ3$_{gv}$	89.67 ± 15.69	86.74	90.19	88.16
FLVQ	93.83 ± 2.84	94.11	93.71	93.91
dFLVQ	93.83 ± 2.84	94.11	93.71	93.91
AFLVQ$_{v1}$	95.17 ± 2.54	95.33	95.10	95.21
AFLVQ$_{v2}$	95.67 ± 2.51	95.83	95.47	95.64
AFLVQ$_{gv}$	95.33 ± 2.46	95.62	95.45	95.54
dAFLVQ$_{v1}$	95.50 ± 2.09	95.78	95.69	95.73
dAFLVQ$_{v2}$	94.83 ± 2.88	94.89	94.84	94.86
dAFLVQ$_{gv}$	95.67 ± 1.96	95.93	95.92	95.92

Table 30 – Overall cost regarding number of prototype and execution time of LVQ-based classification algorithms (Synthetic Control).

Algorithms	P_{c_1}	P_{c_2}	P_{c_3}	P_{c_4}	P_{c_5}	P_{c_6}	N_P	Execution time (s)
LVQ1	2	2	2	2	2	2	12	3.71
dLVQ1	2	2	2	2	2	2	12	3.75
ALVQ1$_{v1}$	1	2	1	1	1	3	9	3.59
ALVQ1$_{v2}$	1	2	1	1	1	2	8	3.34
ALVQ1$_{gv}$	2	2	2	2	1	1	10	5.60
dALVQ1$_{v1}$	1	2	1	1	1	4	10	5.67
dALVQ1$_{v2}$	1	2	1	1	2	2	9	4.58
dALVQ1$_{gv}$	2	4	1	1	1	1	10	4.69
LVQ2	2	2	2	2	2	2	12	3.80
dLVQ2	2	2	2	2	2	2	12	3.79
ALVQ2$_{v1}$	1	1	1	2	1	1	7	2.80
ALVQ2$_{v2}$	1	1	1	1	1	1	6	2.66
ALVQ2$_{gv}$	1	1	2	2	1	1	8	2.06
dALVQ2$_{v1}$	1	1	1	2	1	1	7	1.05
dALVQ2$_{v2}$	1	1	1	1	1	1	6	0.55
dALVQ2$_{gv}$	1	1	1	1	1	1	6	0.19
LVQ3	2	2	2	2	2	2	12	3.88
dLVQ3	2	2	2	2	2	2	12	3.89
ALVQ3$_{v1}$	1	1	3	1	2	1	9	3.26
ALVQ3$_{v2}$	1	1	2	1	2	1	8	2.99
ALVQ3$_{gv}$	1	1	2	1	2	1	8	3.17
dALVQ3$_{v1}$	1	1	3	1	1	2	9	2.74
dALVQ3$_{v2}$	1	1	3	1	1	1	8	3.57
dALVQ3$_{gv}$	3	1	2	1	1	1	9	3.64
FLVQ	2	2	2	2	2	2	12	19.42
dFLVQ	2	2	2	2	2	2	12	19.41
AFLVQ$_{v1}$	2	1	1	3	1	1	9	21.17
AFLVQ$_{v2}$	3	1	2	1	1	1	9	19.12
AFLVQ$_{gv}$	4	2	2	1	1	1	11	22.27
dAFLVQ$_{v1}$	2	1	1	3	1	1	9	19.43
dAFLVQ$_{v2}$	2	1	2	2	1	1	9	23.87
dAFLVQ$_{gv}$	4	2	2	1	1	1	11	21.98

Table 31 – Overall LVQ-based classification performance results (Gun-Point).

Algorithms	**Accuracy**	**Precision**	**Recall**	F_1 **Score**
LVQ1	76.50 ± 7.47	78.63	76.50	77.54
dLVQ1	76.50 ± 11.56	79.47	76.50	77.93
ALVQ1$_{v1}$	86.00 ± 5.68	87.85	86.00	86.90
ALVQ1$_{v2}$	82.00 ± 4.83	83.46	82.00	82.72
ALVQ1$_{gv}$	81.00 ± 11.25	83.36	81.00	82.14
dALVQ1$_{v1}$	86.00 ± 9.37	87.08	86.00	86.53
dALVQ1$_{v2}$	83.00 ± 11.35	84.30	83.00	83.64
dALVQ1$_{gv}$	88.00 ± 7.89	89.55	88.00	88.76
LVQ2	82.50 ± 14.58	88.36	82.50	85.14
dLVQ2	76.50 ± 13.34	83.96	76.50	79.88
ALVQ2$_{v1}$	84.50 ± 12.79	89.14	84.50	86.64
ALVQ2$_{v2}$	84.50 ± 11.89	88.73	84.50	86.45
ALVQ2$_{gv}$	81.00 ± 13.29	87.37	81.00	83.89
dALVQ2$_{v1}$	80.50 ± 13.83	86.18	80.50	83.11
dALVQ2$_{v2}$	79.50 ± 12.57	85.18	79.50	82.11
dALVQ2$_{gv}$	78.50 ± 11.56	84.48	78.50	81.26
LVQ3	73.00 ± 10.59	82.11	73.00	77.16
dLVQ3	73.00 ± 10.06	82.06	73.00	77.15
ALVQ3$_{v1}$	90.50 ± 4.97	91.22	90.50	90.86
ALVQ3$_{v2}$	88.50 ± 6.26	89.51	88.50	89.00
ALVQ3$_{gv}$	88.50 ± 5.80	88.98	88.50	88.74
dALVQ3$_{v1}$	93.50 ± 6.26	93.97	93.50	93.73
dALVQ3$_{v2}$	91.00 ± 6.58	91.45	91.00	91.23
dALVQ3$_{gv}$	93.00 ± 7.53	93.33	93.00	93.16
FLVQ	88.50 ± 7.09	89.04	88.50	88.77
dFLVQ	88.50 ± 7.09	89.04	88.50	88.77
AFLVQ$_{v1}$	92.00 ± 5.37	92.33	92.00	92.16
AFLVQ$_{v2}$	89.50 ± 5.99	89.99	89.50	89.74
AFLVQ$_{gv}$	91.50 ± 6.26	92.26	91.50	91.87
dAFLVQ$_{v1}$	92.00 ± 5.37	92.49	92.00	92.25
dAFLVQ$_{v2}$	89.50 ± 5.99	89.99	89.50	89.74
dAFLVQ$_{gv}$	91.50 ± 5.30	92.27	91.50	91.88

Table 32 – Overall cost regarding number of prototype and execution time of LVQ-based classification algorithms (Gun-Point).

Algorithms	P_{c_1}	P_{c_2}	N_P	Execution time (s)
LVQ1	5	5	10	0.89
dLVQ1	5	5	10	0.88
ALVQ1$_{v1}$	3	6	9	1.42
ALVQ1$_{v2}$	2	4	6	6.77
ALVQ1$_{gv}$	5	4	9	1.09
dALVQ1$_{v1}$	3	5	8	1.53
dALVQ1$_{v2}$	2	4	6	6.55
dALVQ1$_{gv}$	5	4	9	1.02
LVQ2	5	5	10	0.70
dLVQ2	5	5	10	0.73
ALVQ2$_{v1}$	3	5	8	0.73
ALVQ2$_{v2}$	3	5	8	0.76
ALVQ2$_{gv}$	4	4	8	0.94
dALVQ2$_{v1}$	4	5	9	0.71
dALVQ2$_{v2}$	4	4	8	0.79
dALVQ2$_{gv}$	4	4	8	0.82
LVQ3	5	5	10	0.95
dLVQ3	5	5	10	0.94
ALVQ3$_{v1}$	3	5	8	0.86
ALVQ3$_{v2}$	3	5	8	3.67
ALVQ3$_{gv}$	4	5	9	1.09
dALVQ3$_{v1}$	3	6	9	1.22
dALVQ3$_{v2}$	3	4	7	6.32
dALVQ3$_{gv}$	4	6	10	0.89
FLVQ	5	5	10	2.83
dFLVQ	5	5	10	2.85
AFLVQ$_{v1}$	3	4	7	2.73
AFLVQ$_{v2}$	2	3	5	4.12
AFLVQ$_{gv}$	5	3	8	2.72
dAFLVQ$_{v1}$	2	3	5	2.62
dAFLVQ$_{v2}$	2	3	5	4.26
dAFLVQ$_{gv}$	5	3	8	2.73

Table 33 – Overall LVQ-based classification performance results (CBF).

Algorithms	**Accuracy**	**Precision**	**Recall**	F_1 **Score**
LVQ1	81.83 ± 11.72	80.21	82.95	81.06
dLVQ1	80.00 ± 12.19	78.68	81.22	79.52
ALVQ1$_{v1}$	98.92 ± 1.13	98.99	99.01	99.00
ALVQ1$_{v2}$	98.49 ± 1.26	98.50	98.57	98.53
ALVQ1$_{gv}$	98.82 ± 0.94	98.74	98.83	98.78
dALVQ1$_{v1}$	93.98 ± 14.93	92.65	95.42	93.61
dALVQ1$_{v2}$	93.87 ± 14.87	92.39	95.35	93.45
dALVQ1$_{gv}$	93.98 ± 14.93	92.54	95.49	93.60
LVQ2	66.56 ± 6.64	46.93	66.54	54.95
dLVQ2	88.71 ± 13.88	82.34	88.01	84.67
ALVQ2$_{v1}$	66.56 ± 6.64	46.93	66.54	54.95
ALVQ2$_{v2}$	66.56 ± 6.64	46.93	66.54	54.95
ALVQ2$_{gv}$	66.56 ± 6.64	46.93	66.54	54.95
dALVQ2$_{v1}$	89.68 ± 14.55	83.23	89.02	85.62
dALVQ2$_{v2}$	89.68 ± 14.58	83.17	89.13	85.65
dALVQ2$_{gv}$	89.46 ± 14.42	82.92	88.91	85.41
LVQ3	66.56 ± 6.64	46.93	66.54	54.95
dLVQ3	88.60 ± 13.81	82.20	87.92	84.56
ALVQ3$_{v1}$	66.56 ± 6.64	46.93	66.54	54.95
ALVQ3$_{v2}$	66.56 ± 6.64	46.93	66.54	54.95
ALVQ3$_{gv}$	66.56 ± 6.64	46.93	66.54	54.95
dALVQ3$_{v1}$	89.89 ± 14.70	83.47	89.23	85.85
dALVQ3$_{v2}$	89.46 ± 14.42	83.04	88.84	85.43
dALVQ3$_{gv}$	89.68 ± 14.56	83.22	89.01	85.61
FLVQ	88.49 ± 4.27	88.15	88.30	88.22
dFLVQ	88.49 ± 4.27	88.15	88.30	88.22
AFLVQ$_{v1}$	98.49 ± 1.04	98.49	98.49	98.49
AFLVQ$_{v2}$	98.49 ± 1.16	98.42	98.58	98.50
AFLVQ$_{gv}$	97.85 ± 1.76	97.85	97.78	97.81
dAFLVQ$_{v1}$	98.60 ± 1.25	98.57	98.62	98.59
dAFLVQ$_{v2}$	98.17 ± 1.14	98.19	98.08	98.13
dAFLVQ$_{gv}$	98.17 ± 1.61	98.16	98.07	98.11

Table 34 – Overall cost regarding number of prototype and execution time of LVQ-based classification algorithms (CBF).

Algorithms	P_{c_1}	P_{c_2}	P_{c_3}	N_P	Execution time (s)
LVQ1	1	1	1	3	5.90
dLVQ1	1	1	1	3	5.96
ALVQ1$_{v1}$	1	1	1	3	8.44
ALVQ1$_{v2}$	1	1	1	3	8.41
ALVQ1$_{gv}$	1	1	1	3	8.32
dALVQ1$_{v1}$	1	1	1	3	6.53
dALVQ1$_{v2}$	1	1	1	3	6.54
dALVQ1$_{gv}$	1	1	1	3	6.54
LVQ2	1	1	1	3	0.14
dLVQ2	1	1	1	3	0.17
ALVQ2$_{v1}$	1	1	1	3	0.82
ALVQ2$_{v2}$	1	1	1	3	0.82
ALVQ2$_{gv}$	1	1	1	3	0.85
dALVQ2$_{v1}$	1	1	1	3	0.26
dALVQ2$_{v2}$	1	1	1	3	0.26
dALVQ2$_{gv}$	1	1	1	3	0.27
LVQ3	1	1	1	3	0.13
dLVQ3	1	1	1	3	0.17
ALVQ3$_{v1}$	1	1	1	3	0.84
ALVQ3$_{v2}$	1	1	1	3	0.92
ALVQ3$_{gv}$	1	1	1	3	0.85
dALVQ3$_{v1}$	1	1	1	3	0.27
dALVQ3$_{v2}$	1	1	1	3	0.27
dALVQ3$_{gv}$	1	1	1	3	0.26
FLVQ	1	1	1	3	18.66
dFLVQ	1	1	1	3	18.66
AFLVQ$_{v1}$	1	1	1	3	23.92
AFLVQ$_{v2}$	1	1	1	3	23.49
AFLVQ$_{gv}$	1	1	1	3	23.41
dAFLVQ$_{v1}$	1	1	1	3	23.90
dAFLVQ$_{v2}$	1	1	1	3	23.46
dAFLVQ$_{gv}$	1	1	1	3	23.10

Table 35 – Overall LVQ-based classification performance results (Trace).

Algorithms	**Accuracy**	**Precision**	**Recall**	F_1 **Score**
LVQ1	47.00 ± 11.60	39.47	50.99	44.20
dLVQ1	43.50 ± 12.26	35.29	50.57	41.20
ALVQ1$_{v1}$	75.50 ± 10.66	75.60	74.37	74.91
ALVQ1$_{v2}$	76.00 ± 08.43	77.60	77.05	77.19
ALVQ1$_{gv}$	71.50 ± 10.29	67.58	69.25	68.23
dALVQ1$_{v1}$	64.50 ± 15.17	62.16	63.10	61.92
dALVQ1$_{v2}$	70.50 ± 13.63	65.85	69.98	67.10
dALVQ1$_{gv}$	67.50 ± 12.53	68.23	68.30	67.53
LVQ2	69.50 ± 24.99	68.58	71.66	69.38
dLVQ2	67.00 ± 24.52	61.87	67.80	64.00
ALVQ2$_{v1}$	61.00 ± 22.09	60.69	64.57	61.76
ALVQ2$_{v2}$	63.00 ± 23.83	64.50	63.42	63.32
ALVQ2$_{gv}$	60.00 ± 21.60	60.61	61.96	60.64
dALVQ2$_{v1}$	52.50 ± 17.52	53.45	55.31	53.64
dALVQ2$_{v2}$	64.50 ± 22.29	67.73	68.20	67.21
dALVQ2$_{gv}$	60.00 ± 23.21	58.71	63.14	60.13
LVQ3	69.50 ± 24.99	68.58	71.66	69.38
dLVQ3	66.50 ± 25.06	63.85	68.48	65.35
ALVQ3$_{v1}$	60.50 ± 21.92	59.33	63.66	60.67
ALVQ3$_{v2}$	67.00 ± 25.08	71.06	71.48	70.66
ALVQ3$_{gv}$	61.50 ± 22.37	62.04	62.99	61.86
dALVQ3$_{v1}$	53.00 ± 20.30	52.97	58.12	54.58
dALVQ3$_{v2}$	66.50 ± 23.93	66.05	66.43	65.62
dALVQ3$_{gv}$	57.50 ± 23.83	56.36	60.15	57.60
FLVQ	59.00 ± 15.06	58.14	58.27	58.17
dFLVQ	59.00 ± 15.06	58.14	58.27	58.17
AFLVQ$_{v1}$	76.50 ± 10.01	78.84	76.10	77.39
AFLVQ$_{v2}$	75.50 ± 10.12	76.06	73.61	74.75
AFLVQ$_{gv}$	78.00 ± 8.23	80.75	78.87	79.76
dAFLVQ$_{v1}$	80.50 ± 7.98	84.47	80.82	82.57
dAFLVQ$_{v2}$	77.00 ± 12.52	79.87	77.88	78.83
dAFLVQ$_{gv}$	79.50 ± 9.26	81.49	79.66	80.51

Table 36 – Overall cost regarding number of prototype and execution time of LVQ-based classification algorithms (Trace).

Algorithms	P_{c_1}	P_{c_2}	P_{c_3}	P_{c_4}	N_P	Execution time (s)
LVQ1	1	1	1	1	4	1.52
dLVQ1	1	1	1	1	4	1.52
ALVQ1$_{v1}$	1	1	1	1	4	2.53
ALVQ1$_{v2}$	1	1	1	1	4	2.57
ALVQ1$_{gv}$	1	1	1	1	4	2.52
dALVQ1$_{v1}$	1	1	1	1	4	2.67
dALVQ1$_{v2}$	1	1	1	1	4	2.62
dALVQ1$_{gv}$	1	1	1	1	4	2.65
LVQ2	1	1	1	1	4	0.77
dLVQ2	1	1	1	1	4	1.40
ALVQ2$_{v1}$	1	1	1	1	4	2.27
ALVQ2$_{v2}$	1	1	1	1	4	2.22
ALVQ2$_{gv}$	1	1	1	1	4	2.23
dALVQ2$_{v1}$	1	1	1	1	4	1.95
dALVQ2$_{v2}$	1	1	1	1	4	1.98
dALVQ2$_{gv}$	1	1	1	1	4	1.96
LVQ3	1	1	1	1	4	0.76
dLVQ3	1	1	1	1	4	1.43
ALVQ3$_{v1}$	1	1	1	1	4	2.24
ALVQ3$_{v2}$	1	1	1	1	4	2.24
ALVQ3$_{gv}$	1	1	1	1	4	2.25
dALVQ3$_{v1}$	1	1	1	1	4	1.96
dALVQ3$_{v2}$	1	1	1	1	4	1.96
dALVQ3$_{gv}$	1	1	1	1	4	1.97
FLVQ	1	1	1	1	4	4.98
dFLVQ	1	1	1	1	4	5.01
AFLVQ$_{v1}$	1	1	1	1	4	6.11
AFLVQ$_{v2}$	1	1	1	1	4	6.11
AFLVQ$_{gv}$	1	1	1	1	4	6.12
dAFLVQ$_{v1}$	1	1	1	1	4	6.40
dAFLVQ$_{v2}$	1	1	1	1	4	6.16
dAFLVQ$_{gv}$	1	1	1	1	4	6.18

Table 37 – Overall LVQ-based classification performance results (ECG200).

Algorithms	**Accuracy**	**Precision**	**Recall**	F_1 **Score**
LVQ1	78.00 ± 11.83	76.35	74.47	75.24
dLVQ1	78.00 ± 12.52	76.28	74.38	75.03
ALVQ1$_{v1}$	88.50 ± 5.80	88.22	87.67	87.81
ALVQ1$_{v2}$	87.00 ± 7.89	87.03	83.31	85.05
ALVQ1$_{gv}$	83.00 ± 8.23	86.11	76.25	80.54
dALVQ1$_{v1}$	88.00 ± 7.15	87.95	87.51	87.51
dALVQ1$_{v2}$	89.00 ± 7.75	89.29	86.00	87.46
dALVQ1$_{gv}$	86.50 ± 9.73	88.50	81.39	84.62
LVQ2	81.50 ± 10.29	79.32	79.32	79.29
dLVQ2	85.00 ± 9.13	84.62	83.89	84.17
ALVQ2$_{v1}$	78.00 ± 4.83	78.48	79.03	78.66
ALVQ2$_{v2}$	83.50 ± 8.18	83.11	80.86	81.86
ALVQ2$_{gv}$	82.00 ± 8.88	82.35	77.75	79.83
dALVQ2$_{v1}$	80.50 ± 9.56	80.28	77.82	78.90
dALVQ2$_{v2}$	88.50 ± 4.12	87.76	89.88	88.76
dALVQ2$_{gv}$	81.00 ± 9.94	81.71	76.51	78.86
LVQ3	79.00 ± 11.25	78.03	72.57	75.07
dLVQ3	77.50 ± 8.58	76.12	71.10	73.28
ALVQ3$_{v1}$	77.00 ± 11.11	76.85	79.46	78.06
ALVQ3$_{v2}$	80.00 ± 9.13	78.76	81.14	79.87
ALVQ3$_{gv}$	83.50 ± 6.26	81.64	84.08	82.79
dALVQ3$_{v1}$	84.00 ± 8.43	83.65	87.05	85.28
dALVQ3$_{v2}$	86.50 ± 8.83	85.76	87.60	86.62
dALVQ3$_{gv}$	87.00 ± 8.56	86.31	83.18	84.55
FLVQ	80.00 ± 10.33	79.24	79.82	79.52
dFLVQ	80.00 ± 10.33	79.24	79.82	79.52
AFLVQ$_{v1}$	90.00 ± 5.77	90.45	87.47	88.88
AFLVQ$_{v2}$	92.00 ± 7.89	91.66	92.04	91.84
AFLVQ$_{gv}$	87.50 ± 4.25	87.51	84.98	86.13
dAFLVQ$_{v1}$	87.00 ± 4.83	86.70	84.05	85.28
dAFLVQ$_{v2}$	88.50 ± 4.74	89.39	85.70	87.42
dAFLVQ$_{gv}$	90.50 ± 4.97	91.09	88.92	89.91

Table 38 – Overall cost regarding number of prototype and execution time of LVQ-based classification algorithms (ECG200).

Algorithms	P_{c_1}	P_{c_2}	N_P	Execution time (s)
LVQ1	4	4	8	1.53
dLVQ1	4	4	8	1.54
ALVQ1$_{v1}$	2	6	8	2.78
ALVQ1$_{v2}$	2	5	7	2.63
ALVQ1$_{gv}$	5	2	7	1.97
dALVQ1$_{v1}$	2	5	7	2.89
dALVQ1$_{v2}$	2	5	7	2.57
dALVQ1$_{gv}$	5	2	7	2.19
LVQ2	4	4	8	1.20
dLVQ2	4	4	8	1.38
ALVQ2$_{v1}$	1	5	6	2.64
ALVQ2$_{v2}$	2	4	6	2.72
ALVQ2$_{gv}$	3	4	7	1.87
dALVQ2$_{v1}$	2	5	7	2.67
dALVQ2$_{v2}$	1	6	7	2.20
dALVQ2$_{gv}$	3	3	6	1.63
LVQ3	4	4	8	1.58
dLVQ3	4	4	8	1.58
ALVQ3$_{v1}$	3	3	6	2.93
ALVQ3$_{v2}$	3	3	6	2.88
ALVQ3$_{gv}$	3	3	6	2.37
dALVQ3$_{v1}$	4	3	7	3.01
dALVQ3$_{v2}$	4	3	7	2.69
dALVQ3$_{gv}$	5	2	7	2.20
FLVQ	4	4	8	4.59
dFLVQ	4	4	8	4.54
AFLVQ$_{v1}$	2	6	8	5.60
AFLVQ$_{v2}$	2	5	7	5.57
AFLVQ$_{gv}$	6	1	7	5.20
dAFLVQ$_{v1}$	2	6	8	5.65
dAFLVQ$_{v2}$	3	5	8	5.45
dAFLVQ$_{gv}$	6	1	7	5.20

Table 39 – Overall LVQ-based classification performance results (Italy Power Demand).

Algorithms	**Accuracy**	**Precision**	**Recall**	F_1 **Score**
LVQ1	96.15 ± 2.11	96.14	96.14	96.14
dLVQ1	95.50 ± 2.05	95.60	95.43	95.52
ALVQ1$_{v1}$	95.32 ± 1.53	95.52	95.28	95.40
ALVQ1$_{v2}$	96.61 ± 1.37	96.56	96.65	96.61
ALVQ1$_{gv}$	95.96 ± 1.31	96.03	96.03	96.03
dALVQ1$_{v1}$	95.50 ± 2.05	95.60	95.43	95.52
dALVQ1$_{v2}$	95.50 ± 2.05	95.60	95.43	95.52
dALVQ1$_{gv}$	95.50 ± 2.05	95.60	95.43	95.52
LVQ2	96.51 ± 1.49	96.46	96.56	96.51
dLVQ2	96.97 ± 1.37	96.96	97.01	96.98
ALVQ2$_{v1}$	96.61 ± 1.44	96.66	96.52	96.59
ALVQ2$_{v2}$	96.51 ± 1.88	96.47	96.54	96.51
ALVQ2$_{gv}$	95.50 ± 2.78	95.64	95.54	95.59
dALVQ2$_{v1}$	96.33 ± 1.50	96.32	96.32	96.32
dALVQ2$_{v2}$	96.33 ± 1.78	96.34	96.34	96.34
dALVQ2$_{gv}$	95.78 ± 2.49	95.87	95.72	95.79
LVQ3	58.44 ± 1.89	70.59	58.25	63.69
dLVQ3	63.30 ± 11.91	74.37	63.91	68.60
ALVQ3$_{v1}$	93.30 ± 10.98	95.13	92.49	93.56
ALVQ3$_{v2}$	93.49 ± 10.98	95.34	92.64	93.74
ALVQ3$_{gv}$	90.28 ± 10.05	92.52	89.66	90.84
dALVQ3$_{v1}$	96.51 ± 1.28	96.55	96.50	96.52
dALVQ3$_{v2}$	96.88 ± 1.38	96.84	96.89	96.87
dALVQ3$_{gv}$	95.50 ± 1.70	95.58	95.57	95.57
FLVQ	96.88 ± 1.79	96.89	96.86	96.88
dFLVQ	96.88 ± 1.79	96.89	96.86	96.88
AFLVQ$_{v1}$	96.97 ± 1.37	96.94	96.96	96.95
AFLVQ$_{v2}$	97.16 ± 2.85	97.18	97.12	97.15
AFLVQ$_{gv}$	96.88 ± 1.69	96.83	96.90	96.86
dAFLVQ$_{v1}$	96.97 ± 1.68	96.94	96.96	96.95
dAFLVQ$_{v2}$	96.79 ± 1.74	96.78	96.76	96.77
dAFLVQ$_{gv}$	96.88 ± 1.63	96.84	96.89	96.86

Table 40 – Overall cost regarding number of prototype and execution time of LVQ-based classification algorithms (Italy Power Demand).

Algorithms	P_{c_1}	P_{c_2}	N_P	Execution time (s)
LVQ1	9	9	18	9.55
dLVQ1	9	9	18	9.54
ALVQ1$_{v1}$	13	4	17	13.72
ALVQ1$_{v2}$	8	9	17	13.54
ALVQ1$_{gv}$	12	6	18	12.09
dALVQ1$_{v1}$	9	9	18	11.01
dALVQ1$_{v2}$	9	9	18	11.26
dALVQ1$_{gv}$	9	9	18	10.99
LVQ2	9	9	18	9.71
dLVQ2	9	9	18	9.78
ALVQ2$_{v1}$	8	7	15	56.90
ALVQ2$_{v2}$	6	9	15	11.39
ALVQ2$_{gv}$	9	6	15	12.05
dALVQ2$_{v1}$	8	8	16	14.35
dALVQ2$_{v2}$	8	8	16	12.34
dALVQ2$_{gv}$	10	5	15	10.58
LVQ3	9	9	18	10.36
dLVQ3	9	9	18	10.71
ALVQ3$_{v1}$	7	9	16	36.43
ALVQ3$_{v2}$	4	12	16	16.06
ALVQ3$_{gv}$	12	3	15	12.80
dALVQ3$_{v1}$	3	10	13	28.05
dALVQ3$_{v2}$	5	12	17	17.10
dALVQ3$_{gv}$	13	4	17	13.71
FLVQ	9	9	18	29.82
dFLVQ	9	9	18	29.78
AFLVQ$_{v1}$	7	10	17	36.02
AFLVQ$_{v2}$	8	10	18	35.95
AFLVQ$_{gv}$	12	6	18	35.41
dAFLVQ$_{v1}$	8	9	17	35.28
dAFLVQ$_{v2}$	6	7	13	31.41
dAFLVQ$_{gv}$	11	6	17	35.95

Table 41 – Overall LVQ-based classification performance results (Computers).

Algorithms	**Accuracy**	**Precision**	**Recall**	F_1 **Score**
LVQ1	53.60 ± 5.06	53.49	53.44	53.46
dLVQ1	54.40 ± 7.88	55.06	55.17	55.11
ALVQ1$_{v1}$	56.20 ± 5.85	55.74	55.79	55.76
ALVQ1$_{v2}$	56.60 ± 3.89	56.53	56.41	56.47
ALVQ1$_{gv}$	54.60 ± 4.53	54.77	54.73	54.75
dALVQ1$_{v1}$	56.00 ± 6.18	56.21	56.27	56.24
dALVQ1$_{v2}$	58.20 ± 5.29	58.11	58.08	58.09
dALVQ1$_{gv}$	61.00 ± 3.43	61.62	61.49	61.56
LVQ2	55.20 ± 5.18	55.37	55.30	55.33
dLVQ2	54.40 ± 5.95	54.86	54.78	54.82
ALVQ2$_{v1}$	54.60 ± 5.50	54.46	54.41	54.44
ALVQ2$_{v2}$	54.60 ± 5.58	54.99	54.95	54.97
ALVQ2$_{gv}$	55.60 ± 4.88	56.00	55.88	55.94
dALVQ2$_{v1}$	59.20 ± 6.05	59.89	59.85	59.87
dALVQ2$_{v2}$	57.60 ± 8.21	57.58	57.49	57.54
dALVQ2$_{gv}$	61.00 ± 9.85	60.99	60.95	60.97
LVQ3	52.60 ± 6.04	51.92	52.67	51.95
dLVQ3	55.00 ± 6.06	54.30	54.65	54.16
ALVQ3$_{v1}$	53.00 ± 6.75	52.93	53.04	52.98
ALVQ3$_{v2}$	56.80 ± 9.62	56.96	56.83	56.90
ALVQ3$_{gv}$	54.60 ± 8.75	54.82	54.77	54.80
dALVQ3$_{v1}$	52.80 ± 7.55	52.80	52.81	52.81
dALVQ3$_{v2}$	55.00 ± 7.20	55.13	55.20	55.16
dALVQ3$_{gv}$	54.20 ± 4.37	53.78	53.63	53.70
FLVQ	54.00 ± 8.74	54.54	54.39	54.47
dFLVQ	54.00 ± 8.74	54.54	54.39	54.47
AFLVQ$_{v1}$	56.80 ± 6.68	57.35	56.97	57.16
AFLVQ$_{v2}$	56.00 ± 7.94	56.60	56.70	56.65
AFLVQ$_{gv}$	60.00 ± 8.49	60.22	60.37	60.29
dAFLVQ$_{v1}$	56.40 ± 5.56	56.72	56.73	56.73
dAFLVQ$_{v2}$	60.80 ± 5.27	61.24	61.07	61.15
dAFLVQ$_{gv}$	57.00 ± 5.75	57.07	57.05	57.06

Table 42 – Overall cost regarding number of prototype and execution time of LVQ-based classification algorithms (Computers).

Algorithms	P_{c_1}	P_{c_2}	N_P	Execution time (s)
LVQ1	6	6	12	19.14
dLVQ1	6	6	12	19.21
ALVQ1$_{v1}$	6	5	11	40.11
ALVQ1$_{v2}$	6	5	11	40.04
ALVQ1$_{gv}$	6	5	11	35.64
dALVQ1$_{v1}$	7	4	11	44.45
dALVQ1$_{v2}$	7	4	11	45.60
dALVQ1$_{gv}$	8	3	11	39.23
LVQ2	6	6	12	17.60
dLVQ2	6	6	12	18.75
ALVQ2$_{v1}$	5	6	11	21.92
ALVQ2$_{v2}$	5	6	11	21.22
ALVQ2$_{gv}$	6	5	11	21.25
dALVQ2$_{v1}$	5	6	11	48.31
dALVQ2$_{v2}$	6	5	11	47.16
dALVQ2$_{gv}$	7	4	11	35.67
LVQ3	6	6	12	24.44
dLVQ3	6	6	12	23.15
ALVQ3$_{v1}$	4	7	11	50.13
ALVQ3$_{v2}$	3	8	11	44.73
ALVQ3$_{gv}$	4	7	11	36.62
dALVQ3$_{v1}$	5	6	11	51.38
dALVQ3$_{v2}$	4	7	11	46.58
dALVQ3$_{gv}$	5	6	11	40.96
FLVQ	6	6	12	34.71
dFLVQ	6	6	12	34.05
AFLVQ$_{v1}$	9	3	12	51.57
AFLVQ$_{v2}$	6	5	11	50.81
AFLVQ$_{gv}$	9	3	12	47.69
dAFLVQ$_{v1}$	9	3	12	54.35
dAFLVQ$_{v2}$	6	5	11	49.86
dAFLVQ$_{gv}$	9	3	12	49.53