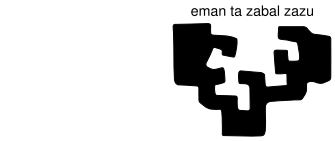
University of the Basque Country
UPV/EHU



Doctoral Thesis

Advances on Time Series Analysis using Elastic Measures of Similarity

---

Author:
Izaskun Oregi

Supervisors:
Prof. Dr. Javier Del Ser
Dr. Aritz Pérez

A Thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

in the

Department of Communications Engineering

July 8, 2020

Declaration of Authorship

I, Izaskun Oregi, declare that this thesis titled, "Advances on Time Series Analysis using Elastic Measures of Similarity" and the work presented in it are my own. I confirm that:

- This work was done entirely or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

To Xabi and Esti

UNIVERSITY OF THE BASQUE COUNTRY UPV/EHU

Abstract

Engineering School of Bilbao
Department of Communications Engineering

Doctoral Degree

Advances on Time Series Analysis using Elastic Measures of Similarity

by Izaskun Oregi

A sequence is a collection of data instances arranged in a structured manner. When this arrangement is held in the time domain, sequences are instead referred to as time series. As such, each observation in a time series represents an observation drawn from an underlying process, produced at a specific time instant. However, other type of data indexing structures, such as space- or threshold-based arrangements are possible. Data points that compose a time series are often correlated with each other. To account for this correlation in data mining tasks, time series are usually studied as a whole data object rather than as a collection of independent observations. In this context, techniques for time series analysis aim at analyzing this type of data structures by applying specific approaches developed to leverage intrinsic properties of the time series for a wide range of problems, such as classification, clustering and other tasks alike.

The development of monitoring and storage devices has made time series analysis proliferate in numerous application fields, including medicine, economics, manufacturing and telecommunications, among others. Over the years, the community has gathered efforts towards the development of new data-based techniques for time series analysis suited to address the problems and needs of such application fields. In the related literature, such techniques can be divided in three main groups: feature-, model- and distance-based methods. The first group (feature-based) transforms time series into a collection of features, which are then used by conventional learning algorithms to provide solutions to the task under consideration. In contrast, methods belonging to the second group (model-based) assume that each time series is drawn from a generative model, which is then harnessed to elicit knowledge from data. Finally, distance-based techniques operate directly on raw time series. To this end, these methods resort to specially defined measures of distance or similarity for comparing time series, without requiring any further processing. Among them, elastic similarity measures (e.g., dynamic time warping and edit distance) compute

the closeness between two sequences by finding the best alignment between them, disregarding differences in time, and thus focusing exclusively on shape differences.

This Thesis presents several contributions to the field of distance-based techniques for time series analysis, namely: i) a novel multi-dimensional elastic similarity learning method for time series classification; ii) an adaptation of elastic measures to streaming time series scenarios; and iii) the use of distance-based time series analysis to make machine learning methods for image classification robust against adversarial attacks. Throughout the Thesis, each contribution is framed within its related state of the art, explained in detail and empirically evaluated. The obtained results lead to new insights on the application of distance-based time series methods for the considered scenarios, and motivates research directions that highlight the vibrant momentum of this research area.

Acknowledgements

I owe a sincere debt of gratitude to my esteemed supervisors, Prof. Dr. Javier Del Ser and Dr. Aritz Pérez, for supporting me and having a lasting trust in me. Four years ago we launched a tin spaceship to an unknown place. During this journey, I have felt the vastness around making me losing control from time to time. However, in these moments you have always offered wise suggestions and guidance, redirecting the course of our spacecraft not only to save orbits but also to interesting unexplored places. I wish to extend my words of appreciation to Prof. Dr. José A. Lozano, for his continuous advice and for joining us in our journey. It has been an honor working with you. I hope this is just the first of many other research adventures.

I would like to thank TECNALIA for providing me with the necessary resources, facilities, and help for completing this work. Special thanks are due to Iñigo Arizaga, Joseba Laka, Isidoro Cirión, and Elena Urrutia for all their efforts to allow us to carry out our research lines. I would like to also thank all the people in the OPTIMA group (both in Derio and Miñano) for their guidance and full-time help through these years.

I take this opportunity to express my deep gratitude to all JRL members, especially to Ibai, Txus, Eneko, Aritz, Eric, and Esther. It has been a real pleasure sharing this time with you. Thank you for your friendship inside and outside the lab.

I am also grateful to Prof. Dr. Pierre-François Marteau for giving me the opportunity to visit the Institut de Recherche en Informatique et Systémes Aléatoires (IRISA) at the Université Bretagne Sud (UBS). I am thankful for your warm reception and collaboration. Special thanks to all *open space* mates, for stimulating debates, lunches, gym sessions, and their support during my stay at Vannes.

I wish to thank my family, especially my parents, Luis and Mariví, for supporting me, teaching me, and inculcating me the values that have brought me here. Thank you for your unconditional love and patience.

Finally, I would like to thank Esti and Xabi for being the funniest and most annoying people in the world, I love and hate you in equal measure. Thank you very much for giving me hope and being always supportive.

Contents

List of Figures

List of Tables

List of Abbreviations

Similarity Measures	
ED	**E**uclidean **D**istance
MD	**M**ahalanobis **D**istance
ESM	**E**lastic **S**imilarity **M**easures
OESM	**O**n-line **E**lastic **S**imilarity **M**easures
DTW	**D**ynamic **T**ime **W**arping
ODTW	**O**n-line **D**ynamic **T**ime **W**arping
EDIT	**EDIT** distance
EDR	**E**DIT **D**istance for **R**eal Sequences
ERP	**E**DIT distance with **R**eal **P**enalty
Time Series & Data Mining	
MTS	**M**ultivariate **T**ime **S**eries
TSA	**T**ime **S**eries **A**nalysis
TSC	**T**ime **S**eries **C**lassification
`K`**-NN**	`K` **N**earest **N**eighbors
NN	**N**earest **N**eighbors
SVM	**S**upport **V**ector **M**achine
DNN	**D**eep **N**eural **N**etwork
CNN	**C**onvolutional **N**eural **N**etwork
AML	**A**dversarial **M**achine **L**earning
Chapter 3	
DTW$_D$	**D**ependent **D**ynamic **T**ime **W**arping
DTW$_I$	**I**ndependent **D**ynamic **T**ime **W**arping
CDTW	**C**orrelation-based **D**ynamic **T**ime **W**arping
MDTW	**M**ahalanobis-based **D**ynamic **T**ime **W**arping
GA	**G**enetic **A**lgorithm
PSO	**P**article **S**warm **O**ptimization
EDA	**E**stimated **D**istribution **A**lgorithm
Chapter 4	
ACC	**Acc**uracy
PL	**P**redicted **L**abel
TL	**T**rue **L**abel
`short`	OESM with **short** range memory
`middle`	OESM with **middle** range memory
`large`	OESM with **large** range memory

`full`	OESM with **full** range memory

Chapter 5

PED	**P**attern **E**nd **D**etector
STSC	**S**reaming **T**ime **S**eries **C**lassification
AUC	**A**rea **U**nder the **C**urve
PED-NN	**PED** based **N**earest **N**eighbors
ODTW-NN	**ODTW** based **N**earest **N**eighbors
GS	**G**old-**S**tandard model

Chapter 6

BIM	**B**asic **I**terative **M**ethod
FGSM	**F**ast **G**radient **S**ign **M**ethod
PGD	**P**rojected **G**radient **D**escent
JSMA	**J**acobian **S**aliency **M**ap **A**ttack
C&W	**C**arlini and **W**agner attack
DkNN	**D**eep K-**N**earest **N**eighbors
OSVM	**O**ne-class **S**upport **V**ector **M**achine

Chapter 7

DTW-NN	**DTW** based **N**earest **N**eighbors
DTW-SNN	**DTW** based **S**moothed **N**earest **N**eighbors
DTW-BIO	**DTW** based **B**i-**o**bjective **O**ptimization
DTW-FG	**DTW** based **F**ast **G**radient
DTW-IFG	**DTW** based **I**terative **F**ast **G**radient

A good impression of myself
Not much to conceal
I'm saying nothing
But I'm saying nothing with feel

I simply am not here no way I...
Shut up be happy stop whining please

And because of who we are
We react in mock surprise
The curse of "there must be more"
So don't breathe here,
don't leave your bags

I simply am not here no way I...
Shut up be happy stop whining please

The dust in my soul
makes me feel the weight in my legs
My head in the clouds
and I'm zoning out
I'm watching TV
but I find it hard to stay conscious
I'm totally bored
but I can't switch off

Only apathy from the pills in me
It's all in me, all in you
Electricity
Its all in me, all in you
Only MTV and cod philosophy

We're lost in the mall, shuffling through the stores like zombies
What is the point?
What can money buy?
My hand's on a gun and I find the range, God tempt me
What did you say?
Think I'm passing out

Only apathy from the pills in me
It's all in me, all in you
Electricity
Its all in me, all in you
Only MTV and cod philosophy

Water so warm that day
I counted out the waves
As they broke into surf
I smiled into the sun

The water so warm that day
I was counting out the waves
And I followed their short life
As they broke into the shoreline
I could see you
But I couldn't hear you
You were holding your hat in the breeze
Turning away from me
In this moment you were stolen
There's black across the sun

Porcupine Tree. *Anesthetize.*
Fear of a Blank Planet, Roadrunner Records, 2007.

Part I

Introduction, Motivation and Objectives

Chapter 1

Learning from Time Series

A time series is a finite sequence of data observations indexed in order, where each observation represents the value of an event recorded in a structured way, e.g., at different time stamps, space positions, or other systematical procedure that implies ordering. Accordingly, a multi-dimensional time series (MTS) can be formally defined as:

$$\mathbf{X} = \begin{pmatrix} x_{11} & \dots & x_{1i} & \dots & x_{1n} \\ \vdots & & \vdots & & \vdots \\ x_{d1} & \dots & x_{di} & \dots & x_{dn} \\ \vdots & & \vdots & & \vdots \\ x_{D1} & \dots & x_{Di} & \dots & x_{Dn} \end{pmatrix}, \tag{1.1}$$

where each *observation* is represented by the D-dimensional array:

$$\mathbf{x}_i = (x_{1i}, \dots, x_{di}, \dots, x_{Di}) \text{ for } i = 1, \dots, n \tag{1.2}$$

and each *dimension* is denoted by the n-length sequence:

$$X^{(d)} = (x_{d1}, \dots, x_{di}, \dots, x_{dn}) \text{ for } d = 1, \dots, D. \tag{1.3}$$

When the phenomenon or process generating time series data produce an unique outcome (i.e., when $D = 1$), we deal with uni-dimensional time series. In this case, instead of Expression (1.1), a time series is given by:

$$X = (x_1, \dots, x_i, \dots, x_n)\,. \tag{1.4}$$

Time series are ubiquitous over a large number of domains. Indeed, processes underlying many real-world phenomena yield time series data. For instance, daily records of surface deformations or displacements (geophysics), signals of the electrical activity of the heart (medicine), hourly series of air pollutants (ecology), starlight curve observations (astronomy), gene expression sequences (microbiology), pixel histograms of binarized images, and other similar sources of information. In the view of the prevalence and relevance of time series data in these fields, a natural interest has developed within the research community around problems related to this kind of data. Moreover, the advent of technological advances on the

collection, storage, retrieval, and processing of large amounts of data has ignited even further the seek of efficient means for analyzing time series databases. As a consequence, the scientific community has intensified its research efforts towards the formulation of knowledge extraction problems defined on time series data, along with the design and validation of learning models capable of solving them efficiently.

Besides being a sequence of ordered data points, time series are characterized by an intrinsic property that makes them different from other types of data. Successive measurements in time series are often correlated with each other, which means that there is dependence among observations. Therefore, time series are frequently considered as a whole entity, not discrete collections of data points that can be analyzed independently from each other. This has prompted the design of specific techniques that exploit correlations among the constituent points of time series. The extraction of hidden information from temporal data has given rise to a vast research area widely known as time series analysis (TSA) [1], [2], which includes a manifold of challenging learning problems such as forecasting, classification, and clustering. Before delving into these core learning problems in TSA, we first pause at the definition of streaming time series, which play a fundamental role in the present Thesis.

1.1 Streaming Time Series

So far we have stated that time series emerge from processing the variables of limited data-producing events. However, in many applications time series arise from dynamic processes, where data instances are streamed at a high speed over time, continuously generating large volumes of samples. Such samples must be analyzed as fast as possible to comply with the memory and space limitations of the computing device under consideration. Illustrative examples of scenarios where stream data is produced include electricity supply series (energy) [3], human activity signals issued from wearable sensors (medicine) [4] or car flow sequences (traffic forecasting) [5]. In this context, we define a *streaming time series* as a concatenation of arriving data streams:

$$\mathbf{X}^{n+b} = \langle \mathbf{X}^{n}, \mathbf{X}^{n+1,n+b} \rangle \in \mathbb{R}^{D\times(n+b)}, \tag{1.5}$$

where $\langle\cdot\rangle$ denotes serial concatenation; $n+b$ is the total number of data points received heretofore; and:

$$\mathbf{X}^{n+1,n+b} = (\mathbf{x}_{n+1}, \ldots, \mathbf{x}_{n+b}) \in \mathbb{R}^{D\times b} \tag{1.6}$$

denotes the last received data *chunk* (also referred to as *batch* in the context of stream data mining).

In addition to being generated by a potentially endless process, streaming time series might also be non-stationary (i.e., the temporal structure of the time series can switch over time). Formally, a time series is said to be stationary when its statistical properties, such as mean and variance, do

not depend on the timestamp at which it is measured, i.e., do not change over time. Accordingly, stationary time series are those with neither trend nor seasonal behavior, which can be achieved through differentiation. In this Thesis, we assume that each stationary interval has an associated event that takes place for a limited time interval (e.g. a *jump* or a *squat* in a human activity signals). Accordingly, a streaming time series can be defined as a sequence of events over time, i.e.:

$$\mathbf{X}^n = \langle \mathbf{X}_1^{n_1}, \ldots, \mathbf{X}_\kappa^{n_\kappa} \rangle \in \mathbb{R}^{D \times n}, \tag{1.7}$$

where time series $X_\kappa^{n_\kappa} \in \mathbb{R}^{D \times n_\kappa}$ represents the κ-th event over the stream; and $n = \sum_{j=1}^{\kappa} n_j$ is the total number of observations in the stream so far. Event changes or transitions occur at points $n_j + 1$, in the limit between two consecutive events (for instance, between two consecutive jumps or a *jump* followed by a *squat*).

Based on the definitions as per (1.5) and (1.7), we can describe a streaming time series as a succession of events whose observations are received in small chunks over time. In adapting TSA procedures to such evolving data, the research community is currently proposing new models that might efficiently deal with the challenges arising from streaming data [6], [7]. In this context, related literature identifies two main strategies: passive and active – also known as blind and informed adaptation methods respectively [8]. Passive adaptation strategies update models to data continuously, without any explicit detection of change. To this end, they resort to different mechanisms for making the model forget knowledge acquired in the past, including sliding windows or diversity induction methods [9]. Active strategies, in contrast, apply change detection procedures to evaluate whether there has been a change as per the received data points [10]. When a change is detected, part of the knowledge captured by the model is reinitialized to learn from the upcoming stationary interval in an agile fashion.

1.2 Core Problems in Time Series Analysis

In this section we introduce some of the most relevant problems related to time series analysis, namely, time series forecasting [11], classification [12], clustering [13] and segmentation [14]. This will allow the reader to properly frame the contributions exposed in the remainder of the Thesis.

1.2.1 Time Series Forecasting

Time series forecasting is, arguably, among the most extensively studied paradigms in time series analysis. This task consists of mining the relationships among the entries of a time series to make predictions of future

unobserved values. Specifically, given $\mathbf{X}$ (input) and $Y = (y_1, \ldots, y_n)$ (target) time series, time series forecasting aims at learning a function:

$$\begin{aligned} f : \mathbb{R}^{D \times n} \times \mathbb{R}^n &\longrightarrow \mathbb{R}^\delta \\ (\mathbf{X}, Y) &\longrightarrow (y_{n+1}, ..., y_{n+\delta}) \end{aligned} \tag{1.8}$$

that uses historical data $(\mathbf{X}, Y)$ to estimate δ *future* values of the target time series Y, i.e. $(y_{n+1}, \ldots, y_{n+\delta})$, where $\delta \in \mathbb{N}$ is referred to as the *forecasting horizon* (i.e., the number of time steps estimated by f). This problem appears with special profusion in a diversity of application scenarios such as traffic control [11], energy management [15], [16] or finance [17], among others alike.

In this context, auto-regressive moving-average (ARMA) models are one of the most basic time series forecasting approaches investigated in the literature. By modeling the correlation among observations, these models are used to predict future values of stationary time series. For this purpose, ARMA models aim at fitting the coefficients of a linear equation, where the inputs correspond to past time series entries (auto-regression) and past prediction errors (moving-average) [18]. Although ARMA models, along with all its variants [19], have been applied in a wide range of domains, it has been often found that such models leave some complex relations unexplained. In order to overcome this issue, in the last decades supervised machine learning models for time series forecasting have progressively gained momentum in the literature [16].

When addressing forecasting problems with supervised learning methods, time series are transformed into a collection of input-output pairs (*examples*), that serve as a data showcase of the relationships between observations over the history of the time series. By using these examples as training data, supervised learning models learn a mapping between inputs (observation) and outputs (target). Beyond well-known supervised learning models utilized profusely for time series forecasting (e.g. Support Vector Regression, Decision Trees, or regression ensembles), in recent times forecasting via supervised learning has been dominated by Deep Neural Networks (DNN), which have shown a superior modeling capacity and predictive performance, especially when dealing with long-term recurrences. In this context, recent works have confirmed that Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNNs), either in isolation or in hybrid CNN-RNN architectures, are the most competitive models when handling many challenging real-world tasks, including energy market [20], solar radiation [16] or residential load forecasting [21].

1.2.2 Time Series Classification

Another relevant problem that frequently arises in the TSA field is the time series classification (TSC) task. Specifically, this problem consist of

learning a function f (a *classifier*):

$$\begin{aligned} f : \mathbb{R}^{D \times n} &\longrightarrow \{1, \ldots, L\} \\ \mathbf{X} &\longrightarrow l \end{aligned} \tag{1.9}$$

that predicts the category l of a test time series $\mathbf{X}$ based on a set of training series $\mathcal{D} = \{(\mathbf{X}_k, l_k)\}_{k=1}^{K}$ with K elements, where $\mathbf{X}_k \in \mathbb{R}^{D \times n}$ represents the k-th sample and $l_k \in \{1, \ldots, L\}$ its corresponding label.

One of the reasons for the importance of TSC lies in its capability to model real-world problems involving time series data. For instance, let us consider a scenario where heartbeats in ECG (electrocardiogram) data are categorized as *normal* or *abnormal*. In this context, a typical TSC problem would consist of diagnosing heart diseases by examining the relationships between labeled and new patient's ECG time series data. Precisely, TSC task is frequently applied to a wide range of fields, far beyond medicine [22]–[24], including science [25], [26] and language sign recognition [27].

To develop the rules that distinguish a class label from each other, one of the most frequently utilized classifier is the K-nearest neighbors (K-NN). These broadly known models work on the basis of a predetermined distance function. Then, when a K-NN classifier is queried for the label of a new time series, the distance to each sample in the training set is computed, so the label assigned to the new time series corresponds to the majority class of its K closest training examples in the training database. In the last decade, several works have proposed the use of K-NN classifier to deal with the TSC problem in a variety of application domains [25], [28], [29]. In [30], for example, the authors resort to K-NN for the classification of multi-dimensional time series. Similarly, Chen et al. [25] use K-NN classifiers to label the trajectories of moving objects, whereas Fard et al. [26] employ them to classify surgical tasks and gestures.

In addition to K-NN, other types of classifiers for time series can be found in the literature. Kampouraki et al. [31] evaluated the benefits of heart rate variability feature-based support vector machine (SVM) for ECG heartbeats signal classification. In [32], the authors extract statistical features from time series data to train a multi-layer neural network. Deng et al. [33] presented the so-called time series forest classifier, an efficient ensemble of decision trees classifiers that use entropy gain and a distance measure to evaluate the quality of leaves splits. In this case, the set of input samples consists of a collection of vectors composed by the mean, variance and slope measurements extracted from different intervals of each time series in the database. More recently, Pelletier et al. [34] have proposed a deep learning classifier to categorize satellite image time series. In this case, the authors employ convolutional layers to automatically learn temporal features that allows for the label identification. Yakhneko et al. [35] propose a Markov model that learns the relations among time series data points to classify biomolecular and text sequences.

1.2.3 Time Series Clustering

Time series clustering consists of building methods that automatically partition a time series database into groups (*clusters*), by minimizing the within-cluster and maximizing the between-cluster dissimilarity between time series. In fact, time series clustering and classification problems resemble each other. However, while classification procedures use the a priori knowledge of the class of each time series to train the model, clustering algorithms are built on unlabeled data. As a result, clustering procedures are usually trained for a different number of partitions, among which the set of clusters with the best performance with respect to a given evaluation metric is chosen.

When addressing this problem, partitional clustering algorithms are often used. Let $\mathcal{D} = \{\mathbf{X}_k\}_{k=1}^{K} \subseteq \mathbb{R}^{D \times n}$ be an unlabeled database of K time series. Partitional methods consist of splitting $\mathcal{D}$ into $\mathtt{K} < K$ non-overlapping clusters[1] $\boldsymbol{C} = \{C_1, \ldots, C_{\mathtt{K}}\}$ by minimizing the total within-cluster dissimilarity error $L_E(\cdot)$, given by:

$$L_E\left(C_1, \ldots, C_{\mathtt{K}}\right) = \sum_{\mathtt{k}=1}^{\mathtt{K}} \sum_{\mathbf{X} \in C_{\mathtt{k}}} D\left(\mathbf{X}, \mathbf{C}_{\mathtt{k}}\right), \tag{1.10}$$

where $D(\cdot, \cdot)$ denotes a measure of dissimilarity between sequences, and $\mathbf{C}_{\mathtt{k}}$ is a representative sequence of the $\mathtt{k}$-th cluster minimizing $\sum_{\mathbf{X} \in C_{\mathtt{k}}} D\left(\mathbf{X}, \mathbf{C}_{\mathtt{k}}\right)$. Two fundamental partitional algorithms are K-means [36] and K-medoids [37]. Specifically for the K-means approach, $D(\cdot, \cdot)$ is the squared Euclidean distance (ED), and $\mathbf{C}_{\mathtt{k}}$ the mean of the sequences within the $C_{\mathtt{k}}$ cluster. In contrast, for the $\mathtt{k}$-medoids $\mathbf{C}_{\mathtt{k}}$ represents the medoid time series among the elements in the $\mathtt{k}$-th cluster, and allows for more general formulations of the measure of dissimilarity $D(\cdot, \cdot)$ beyond the ED. In addition to these procedures, other basic time series clustering methods include hierarchical approaches, which group data into tree-like structures; density-based models, where clusters are defined as high density areas separated by subspaces of low density; or, grid-based algorithms, where clustering is performed based on a grid-structure division of the space [13], [38].

Clustering is especially useful to perform an exploratory analysis of unlabeled data. In many real-world domains, in practice, the provision of labeled data for model development can be extremely difficult or expensive to achieve. However, unlabeled data still possess information worth to be analyzed via clustering methods, as it can reveal their underlying structure. As a result, several works have been reported in this regard. For example, Räsänen et al. [39] apply K-means algorithm to group small customers according to their power. In this line, the authors in [40] utilized both K-means and K-medoids to group synthetically generated sequences, ECG signals, and electricity supply data. More recently, Rodrigues et al. propose an incremental clustering system for streaming time series [3]. Specifically, they presented a hierarchical clustering approach that

[1] We will utilize K to denote the number of clusters or the number of neighbors in K-NN classifiers, with the context indicating whether we refer to one or the other.

evolves with data based on a criterion to merge and split clusters using a correlation-based dissimilarity measure.

1.2.4 Anomaly Detection

Anomaly detection hinges on analyzing data with a view towards discriminating (or analyzing) *strange* observations called *outliers* or *anomalies* [41]. In general, such observations correspond to unusual observations of the underlying process, but they might also result from measurement errors or recording failures. In other words, an outlier is a sample that diverges from the overall pattern in a sample. Clearly, the presence of abnormal instances can negatively affect the performance of predictive models, as it can make the model focus excessively on such outliers and consequently, not generalize properly to new unseen data instances. In another vein, outliers may possess by themselves practical value, since they can be symptomatic of an event of interest, as it often happens in industrial prognosis [42]. For these reasons, the detection of anomalies is an essential step in many data mining applications.

Anomaly detection has also been a core problem in TSA from the very inception of the field. In particular, when dealing with time series data, anomalies can be categorized as point, subsequence, or time series outliers [43], [44]. Mathematically, we consider the dissimilarity function between sequences, $D(\cdot,\cdot)$, and let $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ denote a MTS with $\mathbf{x}_i \in \mathbb{R}^D$. Then, a data point $\mathbf{x}_i$ can be considered a *point outlier* if it deviates from its expected value $\overline{\mathbf{x}}_i$ by more than a threshold $\tau > 0$. That is, if:

$$D\left(\mathbf{x}_i, \overline{\mathbf{x}}_i\right) > \tau. \tag{1.11}$$

Similarly, a subsequence with ω points $\mathbf{X}^{a,a+\omega-1} \subseteq \mathbf{X}$ (with $1 \leq a \leq n+1-\omega$) can be regarded as an *subsequence outlier* if:

$$\sum_{i=0}^{\omega-1} D\left(\mathbf{x}_{a+i}, \overline{\mathbf{x}}_{a+i}\right) > \tau_{\text{sub}}, \tag{1.12}$$

where τ_{sub} denotes a threshold value. In words, a subsequence is declared to be a outlier if its points consistently diverge from their expected values, which can be thought to be representative of the usual behavior of the time series.

In this context, most point and subsequence outlier detection approaches consist of i) learning a time series prediction model that compute expected values of $\overline{\mathbf{x}}_i$; and ii) developing a strategy to seek the best value of the above thresholds τ or τ_{sub}. An illustrative example of such a procedure is presented in [45]. In this work, the authors develop a learning approach composed of a deep convolutional neural network time series predictor, where the threshold is fitted manually based on data.

An entire time series can also be considered an outlier: i) if, given a database, there is a sequence that is significantly dissimilar to other time series in the database, or ii) when a variable of a multi-dimensional time

series behaves oddly concerning other variables. To detect abnormal sequences in star light-curve databases, Rebbapragada et. al [46] present a K-means clustering procedure that yields a ranked list of outliers based on time series similarity to cluster centroids. In this line, the authors in [47] present a performance comparison of a variety of anomaly detection algorithms, including kernel-based (i.e. outliers are detected based on the similarity among sequences), window-based (subsequences are extracted from the time series, and anomalies are detected based on the score assigned to each subsequence), and Markovian techniques (probabilistic anomaly scores are issued from a Markovian model fitted to historical data).

1.2.5 Segmentation and Change Point Detection

As has been mentioned at the beginning of this section, time series are usually hard to process and analyze. Therefore, in TSA it is common practice to first apply segmentation[2] procedures to reduce the number of data points in the series. Specifically, the objective of time series segmentation is to construct an interval approximation of the time series that makes it more computationally manageable while preserving, at the same time, its essential structure.

The relevance of this problem in the TSA domain has increased over the years, for two main reasons: first, because it contributes to an efficient data management in terms of storage and evaluation; secondly, because it allows the identification of structural changes in time series, e.g., change points, stable periods, decreasing/increasing trend periods or primitive patterns that provides meaningful information of the observed process. Consequently, segmentation procedures have been utilized in many application domains including financial data evaluation [48]–[50], path prediction [51], genome sequence analysis [52] and gesture recognition [53], to mention a few.

In order to undertake segmentation tasks, methods based on the Piecewise Linear Approximation (PLA) algorithm have been broadly used. Namely, the goal of this type of segmentation algorithms is to generate an approximate representation of data using a (fixed) number of non-overlapping linear segments. PLA-based strategies are categorized into three main groups: sliding-window, bottom-up and top-down. Roughly, the idea behind sliding-window methods is to expand a segment, over the set of observations within a window of fixed length, until some threshold error (e.g., sum of residuals) is exceeded. Once the segment is fitted, the window slides and the approximation step is repeated using the next set of data points (not included in the previous step). On the other hand, bottom-up approaches start with a fine-grained approximation of segments along the time series. Such algorithms recursively merge adjacent segments until a stopping criterion is met, which can be a threshold error, the number of total segments, or any other condition alike. Conversely, top-down

[2]Depending on the context, time series segmentation problem can also be referred to as time series summarization or dimensionality reduction.

strategies start with the roughest approximation. In this case, the segmentation task is solved by splitting the segments until the stopping criterion is met. A detailed technical review along with an extensive empirical comparison of PLA-based segmentation algorithms can be found in [54].

From a different point of view, the authors in [49] proposed a segmentation algorithm that detects critical points in financial series to segment data according to a representative set stock-price patterns. In [52], a procedure for segmenting MTS is presented. More recently, the authors in [50] present a novel PLA-based segmentation approach, that uses an evolutionary optimization algorithm to minimize the error between the original and approximate series.

In the statistics literature, the segmentation problem is rather known as change point detection. In this context, the objective is to find change points, i.e., critical observations that represent abrupt changes or transitions in time series. Usually, the change point detection problem is formulated in terms of a hypothesis test, where the null hypothesis represents *no change*, whereas the alternative hypothesis is that *a change has occurred*. Different approaches have been proposed in the literature to address this problem. Recently, an exhaustive review of the most relevant change point detection algorithms have been summarized in [14]. Besides, the authors of this survey analyze the advantages and disadvantages of existing techniques and present open research challenges and future directions related to the change point detection problem.

1.3 TSA Models for Data Mining

In the previous sections, we have exposed the problems that lie at the core of TSA, along with a excerpt of works that address such problems from different technical perspectives. Depending indeed on the way such research works tackle TSA problems, the current literature classifies TSA learning procedures into three major categories: approaches based on extracted features, generative models built for the time series under analysis, and methods based on raw time series data [2], [12], [13], [55]. In what follows we provide a brief introduction to each of these categories, concluding with similarity-based approaches. The present thesis is focused on this latter family of learning algorithms for time series.

1.3.1 Feature-based Approaches

This class of models consists of learning algorithms built on feature vectors extracted from time series [10], [31], [32]. Given a TSA problem and a database, models of this type are constructed in two steps. First, each time series is transformed into a vector of features that preserve the essential information of the series. Secondly, a predictive model (e.g. a decision tree, a neural network, a K-medoids) is learned based on the extracted feature vector.

Two types of features can be found in the TSA literature: global and subsequence [12], [56], [57]. Global features refer to transformations

that capture the properties of whole time series into a single variable. An example of a global feature-based classifier is that in [32]. In this work, the authors propose a multi-layer perceptron classifier based on four-dimensional feature vectors that include statistical information of the time series (specifically, mean, variance, skewness, and kurtosis). Subsequence features are instead designed to capture local characteristics of the series. In this case, features are extracted from different intervals of the time series. In this context, Deng et al. [33] propose the so-called time series forest classifier, an efficient ensemble of decision trees classifiers whose input is drawn from a collection of feature vectors composed by the mean, variance and slope measurements extracted from different intervals of each time series in the database. Interestingly, feature-based approaches have been also proposed for streaming time series. Cavalcante et al. [10] have proposed Feature Extraction for Explicit Concept Drift Detection (FEDD), a concept drift detector method that identifies changes in time series data generator models. Based on the feature vector similarity given by Pearson correlation distance (or cosine distance), this method monitors the evolution of sequence features to test whether a concept change has occurred over the streaming time series.

1.3.2 Model-based Approaches

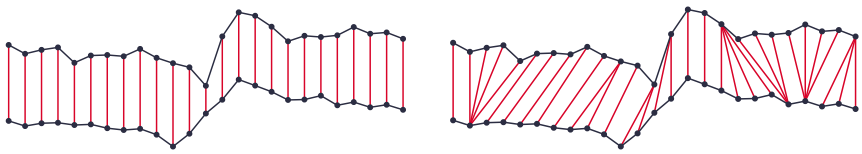
As opposed to feature-based approaches, model-based learning methods assume that each time series under analysis is drawn from a generative model (for instance, a Hidden Markov Model or an auto-regressive model). To this end, the first step when constructing a model-based learning approach is to choose the underlying generative model. Then, the parameters of the model or probability distribution(s) are learned by finding the values of such parameters that best fit them to the time series of the database.

Examples of model-based procedures include the aforementioned hidden Markov models, auto-regressive methods, and kernel models, among others. This family of approaches is exemplified by the work by Yakhneko et al in [35], where they resort a Markov model to classify biomolecular and text sequences. A time series clustering approach is presented in [58], in which a mixture of ARMA models is used to group time series in the database. Other model-based methods are outlined in [13], [14], [59].

1.3.3 Similarity-based Approaches

Finally, similarity-based models also referred to as shape-based or raw-data-based models [13], [55], [59], are procedures build on raw time series data. In this case, the similarity among time series is computed directly from the time series data, without relying on any other intermediate modeling or preprocessing stage. The core idea of these models lies in the definition of a measure of distance/similarity among time series which, in some contexts, is used to replace the inner similarity measures of conventional procedures such as K-NN, K-medoids or SVMs. That is, similarity-based methods exploit the measure of similarity among time series to adapt predictive models to temporal data.

Two main groups of similarity measures can be used when constructing similarity-based models for TSA: lock-step and elastic similarity measures [55] (see Figure 1.1). The family of lock-step similarities, including Euclidean or Manhattan distance, perform point-to-point comparisons. Generally, lock-step measures are easy to implement and run efficiently, but they cannot manage highly recurrent mismatches/gaps between time series, nor handle sequences with different lengths. On the contrary, elastic similarity measures (ESMs), which include, among others, dynamic time warping (DTW) [60], allow for more flexible comparisons. In doing so, ESMs shrink or stretch the time axis in order to find the best alignment between the time series under comparison, and obtain the smallest distance between them. As recurrently highlighted in the literature, numerous experiments have provided empirical evidence of the potential of similarity-based classifiers. However, these results have become more evident when the chosen measure of similarity belongs to the family of ESMs, especially when dealing with databases and tasks that intuitively require the shrinking/stretching properties of these measures (e.g. action recognition with gestural time series data, where the discriminability of the action being performed should not depend on the time or speed at which the gesture is performed). Therefore, in many similarity-based methods, ESMs have consolidated themselves as the similarity measures of reference.



(A) Lock-step. (B) Elastic similarity measure.

FIGURE 1.1: Observation alignment of lock-step and ESMs.

This type of algorithms have played a central role in a diverse collection of contexts and TSA scenarios [25], [26], [28]–[30]. Particularly, when addressing supervised problems (like classification) results drawn from a large number of experiments have shown that ESM based approaches (commonly under the DTW) are difficult to outperform [12], [61]. This Thesis is framed within this family of ESM-based distance models for TSA. Therefore, we complement this introductory chapter with an overview of the mathematical foundations of ESMs, which is provided in the next section.

1.4 Elastic Similarity Measures

The mathematical formulation of ESMs can be expressed as an optimization problem. As we have already mentioned, the essence of ESM resides in the ability to stretch or compress the time axis of time series at hand, so as to minimize the influence of local time shifts in the resulting measure of similarity. To this end, ESMs are based on a restricted number of time series alignments or paths, to each of which a weight is assigned.

These path weights represent the degree of dissimilarity of each proposed alignment. Hence, the optimization problem posed by ESMs seeks to find, among the all possible time series alignments, that of minimum weight.

1.4.1 Definition

For simplicity we consider two uni-dimensional time series $X = (x_1, \ldots, x_m)$ and $Y = (y_1, \ldots, y_n)$, where m and n represent the number of points in each sequence. Let a path p be the sequence of $(i, j) \in [1, m] \times [1, n]$ pairs that represent the alignment between observations $x_i \in X$ and $y_j \in Y$. On this basis, a path $p = \{(i_1, j_1), ..., (i_Q, j_Q)\}$ of length Q, is considered to be an allowed path if the following conditions are satisfied:

$$(i_1, j_1) = (1, 1), \tag{1.13}$$

$$(i_Q, j_Q) = (m, n); \text{ and} \tag{1.14}$$

$$(i_q - i_{q-1}, j_q - j_{q-1}) \in \{(1, 0), (1, 1), (0, 1)\} \text{ for } q = 2, ..., Q, \tag{1.15}$$

namely, if p matches every observation in time series X and Y from their beginning to their end, without skipping any point or going steps backward in time. A sketch of this process is illustrated in Figure 1.1. In order to select among the set of paths satisfying the above three conditions, a measure of dissimilarity or *cost function* $c(x_i, y_j)$ among x_i and y_j must be set. As we will later show, the selection of this dissimilarity sets the particular ESM under choice. On the basis of this cost function $c(x_i, y_j)$, we define the weight of an allowed warping path as:

$$w(p) = \sum_{(i,j) \in p} c(x_i, y_j), \tag{1.16}$$

i.e., as the aggregation of the costs of the (i, j) pairs that compose the path p. This last definition allows stating the definition of a generic ESM.

Definition 1 *Given a cost function $c(x_i, y_j)$, the ESM between time series X and Y is defined as:*

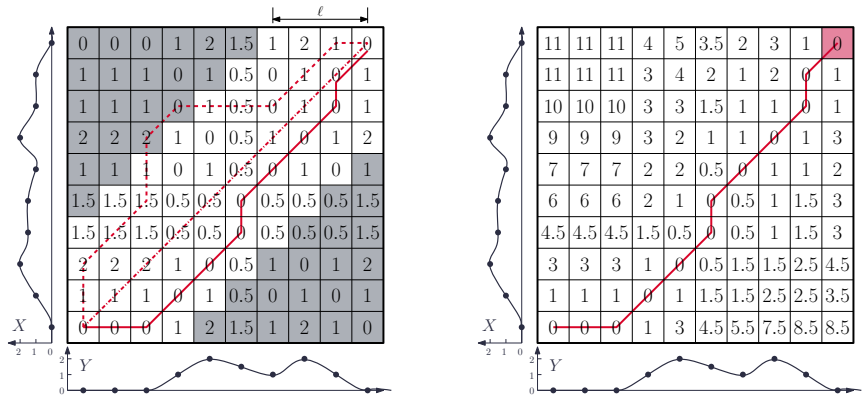
$$D(X, Y) = \min_{p \in \mathcal{P}} w(p), \tag{1.17}$$

where $\mathcal{P}$ is the set of allowed paths in the $[1, m] \times [1, n]$ lattice.

Figure 1.2a shows, for two simulated time series, a small set of 3 allowed paths indicated by the dashed, dotted and solid lines. In accordance with the above definition, these paths are formed by matched pairs that go from $(1, 1)$ to $(m, n) = (10, 10)$ concatenating $\uparrow$, $\nearrow$, $\rightarrow$ unitary steps as per (1.15). Besides, if we compute the weight of all $p \in \mathcal{P}$, note the optimal path corresponds to the solid path, for which $D(X, Y) = 0$.

For the sake of simplicity, we will hereafter use $D_{m,n}$ instead of $D(X, Y)$, to denote the ESM between $X = (x_1, \ldots, x_m)$ and $Y = (y_1, \ldots, y_n)$. Similarly, $c_{i,j}$ will be adopted to denote the cost function $c(x_i, y_j)$, and

$$p^* = \arg\min_{p \in \mathcal{P}} w(p) \tag{1.18}$$



(A) Alignment and Sakoe-Chiba band. (B) Recursive computation.

FIGURE 1.2: ESM measure between two time series X and Y with $m = n = 10$. The cost function corresponds to the ED, $c(x_i, y_j) = ||x_i - y_j||_2$. (a) Different examples of allowed paths over $[1, 10] \times [1, 10]$ lattice, where cell values show the cost function and grey areas the Sakoe-Chiba constraints. Under Sakoe-Chiba, the dashed path is forbidden, consequently it is excluded from the search space $\mathcal{P}$. (b) ESM computation using dynamic programming methods.

to denote the optimal warping path, i.e., the sequence of alignments that yields the minimum weight, hence $D_{m,n} = \sum_{(i,j)\in p^*} c_{i,j}$.

1.4.2 Computation

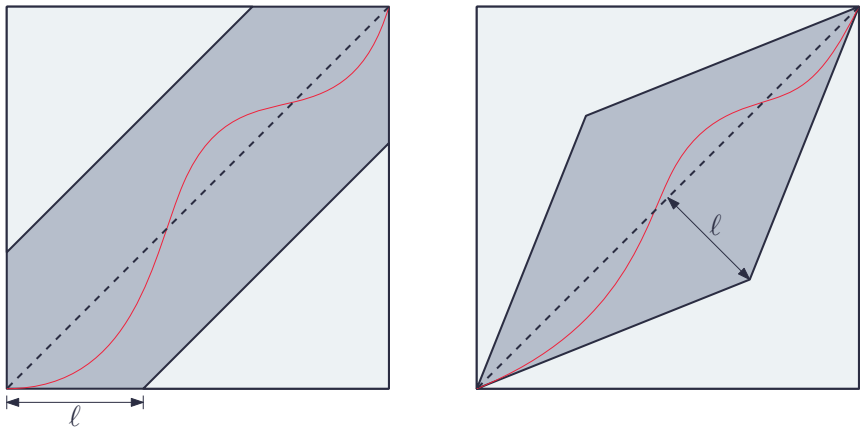
To solve the problem posed in (1.17) we can resort to a brute force approach, that is, we can compute all allowed paths to find among them that with the minimum weight. However, in terms of computational efficiency, this is not the best option to solve the problem above since the number of allowed paths grows exponentially with the size of the time series m and n. Instead, dynamic programming techniques are often utilized [62], by which the ESM is computed by the recursion:

$$D_{m,n} = c_{m,n} + \min\{D_{m-1,n}, D_{m-1,n-1}, D_{m,n-1}\}, \tag{1.19}$$

where $D_{0,0} = 0$ and $D_{0,j} = D_{i,0} = \infty$ for $i = 1, 2, ..., m$ and $j = 1, 2, ..., n$. This computation technique is shown in Figure 1.2b, where each cell (i, j) corresponds to $D_{i,j}$, the red cell corresponds to $D_{m,n}$ and the red line illustrates the optimal path. When using Expression (1.19), p^* can be obtained by going backwards across the matrix, with a computational complexity of $O(n \cdot m)$.

ESM Complexity Reduction Strategies

Although the recurrence in (1.19) reduces the computational complexity of ESM to $O(m \cdot n)$, it is still unaffordable in many practical situations where time series are significantly large. In order to overcome this issue, several techniques have been proposed in the literature either to reduce the



(A) Sakoe-Chiba (B) Itakura parallelogram

FIGURE 1.3: Illustration of most frequently used ESM complexity reduction techniques.

number of ESM computations in learning methods [63]–[65], or to speed up the ESM computation itself by coarsening the resolution of the time series at hand [66], [67].

In this context, the Sakoe-Chiba [68] and the Itakura parallelogram [69] methods have been commonly applied in the field. To reduce the number of operations required in the ESM computation, these techniques limit the number of paths in $\mathcal{P}$ simply by adding some constraints to its definition. In particular, the paths considered by these strategies are composed of (i_q, j_q) pairs that must satisfy $|i_q - j_q| \leq g(i,j)\ \forall q = 1, ..., Q$, where $g : \mathbb{N}^2 \rightarrow \mathbb{N}$ is the constraint function:

- In the case of Itakura parallelogram (see Figure 1.3b), $g(i,j)$ is defined so that the lattice $[1,m] \times [1,n]$ is limited to a parallelogram with $g(1,1) = g(m,n) = 0$.
- In the case of Sakoe-Chiba (see Figure 1.3a), $g(i,j) = \ell$ for any (i,j) point in $[1,m] \times [1,n]$ lattice. Note that, in this way, $g(i,j)$ restricts the lattice to a band around the lattice diagonal, for which ℓ is referred to as the band width.

An illustrative example of the Sakoe-Chiba constraints is shown in Figure 1.2a for $\ell = 3$, where shadowed cells cover forbidden areas for the alignments. Note that the discarded paths are on average those of higher length and probably with higher weights. The application of the Sakoe-Chiba band or the Itakura parallelogram permits to compute the ESM in linear time concerning the length of the time series.

At this point, it is important to highlight that these constraints can also improve the performance of ESM-based classifiers. This fact was proven in [70], where the effect of the Sakoe-Chiba band was analyzed by measuring the accuracy of DTW-based NN classifiers over a benchmark of seven time series databases. By varying the value of the bandwidth ℓ, the authors

showed that narrow constraints improve in general the accuracy of the classifier.

1.4.3 ESM Set and Cost Functions

Although the ESMs family is a broad set of proposed measures of similarity, in this Thesis we focus on the most commonly used alternatives, namely, the dynamic time warping (DTW [60]), the edit distance (EDIT [71]), the edit distance for real sequences (EDR [25]) and the edit distance with real penalty (ERP [28]). The most remarkable differences among these ESMs reside in the data type they can handle, as well as in the definition of the cost function $c_{i,j} = c(x_i, y_j)$ (Table 1.1):

1. For the DTW, the cost function corresponds to the ED or the squared ED. Consequently, the DTW is utilized to calculate the similarity between two *numerical* time series, namely, with real-valued data points.

2. From a different perspective, the EDIT distance quantifies the similarity between two character sequences by counting the number of operations needed to convert one string into another. As shown in Table 1.1, the penalty for an operation is usually set to 1. Therefore, this ESM is suited for sequences of *characters* (symbols) rather than real-valued time series.

3. Given two numerical time series, EDR and ERP also count the number of operations needed to convert two sequences, where the equality holds whenever the absolute difference between the two time series falls within the threshold $\delta > 0$. Given the definitions in Table 1.1, we note that:

   - Just as in the EDIT distance, the cost of each EDR operation is 1.
   - The penalty for ERP is proportional to the ED.

TABLE 1.1: Different ESMs, their cost function $c_{i,j} = c(x_i, y_j)$ and data type.

Elastic Similarity Measure	**Cost Function**	**Data Type**
Dynamic Time Warping (DTW)	$c(x_i, y_j) = \|\|x_i - y_j\|\|_2$	Numerical
Edit Distance (EDIT)	$c(x_i, y_j) = \begin{cases} 0 & \text{if } x_i = y_j \\ 1 & \text{otherwise} \end{cases}$	Character
Edit Distance for Real sequences (EDR)	$c(x_i, y_j; \delta) = \begin{cases} 0 & \text{if } \|x_i - y_j\| \leq \delta \\ 1 & \text{otherwise} \end{cases}$	Numerical
Edit Distance with Real Penalty (ERP)	$c(x_i, x_j; \delta) = \begin{cases} 0 & \text{if } \|x_i - y_j\| \leq \delta \\ \|x_i - y_j\| & \text{otherwise} \end{cases}$	Numerical

We have hitherto elaborated on the context in which the current Thesis is framed. ESMs embody a notable branch of literature and activity in the field of time series analysis. However, there is still a long road ahead in what refers to the adaptation of these measures of similarity between time series to new scenarios and paradigms. There lies indeed the true contribution of the work presented in this Thesis, which is exposed in the following Chapter.

Chapter 2

Motivation and Objectives

Regardless of the outstanding results reported for distance-based models when using ESMs, in the last decade, several works have pointed out the need for novel similarity proposals that address new paradigms, problems, and needs related to time series. Some of these identified challenges aim at the analysis of multi-dimensional time series, learning from time series in stream environments, and the proposal of new TSA approaches that cope with emerging issues in data mining and machine learning [1], [2], [8]. This Thesis finds its motivation precisely in the adaptation of existing elastic similarity measures to these challenges, achieving new insights and novel technical approximations in the following research lines:

- **Measuring the similarity among multi-dimensional time series (MTS).** ESMs have been set as the choice when uni-dimensional time series are considered. However, this is not true for MST (namely, when the points of a time series have several dimensions), where the generalization of ESMs is all but obvious. A straightforward approach to account for the multidimensionality of MTS in the similarity definition could be to treat dimensions separately (independently) or jointly (dependently). However, disregarding whether the independent or the dependent approximation is taken, the resulting ESM is not necessarily suitable for practical applications where latent variables may result in some kind of relationship among MTS dimensions. This dichotomy has instigated the interest of the TSA community in the definitions of new ESMs capable of capturing and exploiting such inter-dimension dependencies, yet without a clear answer to date.

- **Adapting ESMs to stream learning settings.** In off-line (batch) settings, ESMs have turned out to be very effective when dealing with classification tasks due to their ability to handle time distortions and mitigate their effect on the resulting measure. In the streaming setting, where data increases continuously over time and not necessarily in a stationary manner, stream classification approaches are required to be fast, with limited memory consumption and capable of adapting to different stationary intervals. In this sense, the computational complexity of ESMs, and their lack of flexibility to accommodate different stationary intervals, make these similarity measures not compliant with the aforementioned requirements.

- **Evaluating the role of ESMs in the robustness of classification models against adversarial attacks.** It is widely acknowledged that TSA procedures, just like machine learning models in general, are built by assuming that data remain statistically stable over time. Indeed, this hypothesis helps models learn from data patterns, and ensures the successful predictive performance of the models (that is, models can *generalize* their output to new unseen data). This design principle, however, poses an inherent risk in terms of robustness. In the last few years, several works have warned that models built on classical procedures that assume stationary data can be easily misled by adding small perturbations to the input examples [72]. The study of this vulnerability and the derivation of safeguard strategies to make them robust against such attacks fall within what has been coined as Adversarial Machine Learning (AML). In this context, major efforts have been made lately to overcome this issue, especially in the field of deep learning and image classification, where such attacks may cause serious security issues [73]–[75]. However, the intersection between ESMs and AML is still uncharted as per the current state of the art, despite the susceptibility of EMS-based models to adversarial attacks, or the existence of prior work dealing with the extraction of sequence features from image data that could suggest potential synergies between both realms.

2.1 Main Contributions of the Thesis

The motivation and contributions of the present dissertation find their roots in the research lines exposed in the previous section. Specifically, the Thesis presents a number of novel procedures, techniques and tools that take a step further over the state of the art in TSA, not only in terms of their originality, novelty and performance concerning other algorithmic counterparts, but also by extending the applicability of ESMs towards two emerging topics in Machine Learning: stream learning and AML. These contributions are discussed in detail in Part II, including the necessary literature background to properly frame each contribution, and critically examining experimental results obtained to shed empirical evidence on their performance. A brief overview of the contents of each chapter in Part II is next provided.

2.1.1 Similarity Learning for MTS Classification

Chapter 3 aligns with the first research lines described at the beginning of this Chapter. It elaborates on the design of an ESM for comparing MTS series within a time series classification problem. As mentioned previously, related works in the field of TSA have widely agreed on ESMs as the measures of reference when classifying uni-dimensional time series. However, the possible statistical coupling between their different dimensions makes the generalization of these measures to the multivariate case not straightforward to formulate. In this context, we present a simple multi-dimensional DTW learning algorithm that finds the best weighted

combination between two different terms: (i) a dependent DTW, where the warping path is built on alignments of multi-dimensional data points (that is, MTS are treated as a whole); and (ii) an independent DTW, where a warping path is computed for each MTS dimension (i.e. MTS are regarded as a collection of independent uni-dimensional time series). In doing so, a benchmark of four heuristic wrappers is used to evolve the set of weighting coefficients towards maximizing a cross-validated measure of performance of NN classifiers using the weighted similarity measure. The performance of the proposed approach is validated over databases widely utilized in the literature related time series analysis. We conclude that the obtained performance gains can be enlarged by properly weighing the influence of each dimension in the alignments of the dependent DTW warping path.

2.1.2 On-line ESMs for Streaming Time Series

The main objective of Chapter 4 is to adapt ESMs to the requirements of stream settings. In many real-world applications, data are produced at high speed creating massive amounts of samples that must be analyzed as fast as possible, and then discarded, so as to deal with memory limitations. In addition to the (time and memory) computational complexity, the structure of arriving samples may change over time, as occurs in streaming time series when transitioning among successive events. Consequently, conventional procedures for off-line learning are not valid for the on-line analysis of evolving data. One possible way to overcome this issue is to accommodate ESMs to on-line scenarios. In this chapter we propose an adaptation of the DTW measure, which comprises i) a forgetting mechanism and ii) an incremental way of computing this ESM. The former makes the similarity consistent with passive adaptation strategies for streaming time series, by granting more importance in the computation of the similarity measure to recent observations. The latter reduces the computational complexity by avoiding unnecessary computations. To assess the performance of the proposed similarity measure, two different experiments are carried out. The first aims at showing the efficiency of the proposed adaptation in on-line scenarios. To this end, we calculate and compare the computation time for the elastic measures and their on-line adaptation. By analyzing the results drawn from a distance-based streaming machine learning model, the second experiment intends to show the effect of the forgetting mechanism on the resulting similarity value. The experimentation shows that the proposed ODTW comply with the computational requirements and flexibility constraints imposed by the complexity of streaming data.

2.1.3 Learning from ODTW Features

Chapter 5 addresses the problem of learning an OESM-based predictive model over a streaming time series. However, in this case, we resort to an active strategy when adapting the model to the non-stationary characteristics of the streaming time series. In off-line scenarios, where training data are fully available, a predictive model comes to operation after learning the value of its parameters from the stored data. On the contrary, in on-line

settings models are built and put into service continuously, while stream data are received. In most practical cases, data chunks received and used for model updating become eventually non-stationary, as happens in the transition between events in streaming time series. Furthermore, when stream data points have complex relationships as in time series, adaptation and prediction stages get even more involved due to the trade-off between the characterization and exploitation of this time correlation, and the adaptability to changes. This observation is even more relevant when such predictive models are based on the similarity between time series patterns since the characterization of the temporal structure of the streaming time series is essential for the performance of the algorithms.

Based on these premises, Chapter 5 presents an on-line model for detecting event completions in streaming time series based on ODTW measurements. The model, coined as pattern end detector (PED), splits the streaming time series into meaningful patterns, on which we can employ conventional (off-line) models to solve the classification task under consideration. Specifically, the problem addressed in this chapter consists of learning time series label transitions in streaming classification problems. Hence, when a streaming series is to be classified, the proposed PED model learns to detect changes over the stream based on features extracted from ODTW measurements between incoming samples. The result of this analysis is exploited for better adapting the ODTW-based classifier to upcoming stationary intervals. The performance of the proposed PED model is assessed and discussed over 10 benchmark streaming time series classification problems, drawing interesting insights on the detection accuracy of the PED model, as well as the effect of the detection latency on the performance of the on-line classifier.

2.1.4 Using TSA to Protect Deep Neural Networks

The goal of Chapter 6 is to evaluate the feasibility of using TSA procedures to defend image DNN classifiers from adversarial attacks. Given the unprecedented capacity to learn patterns from raw data, DNNs have become the *de facto* modeling choice to address complex machine learning tasks more specifically, in computer vision and speech recognition fields. However, recent works have empirically demonstrated that DNNs might undergo severe security issues when being fed with adversarial samples, namely, intelligently manipulated inputs crafted to confuse their output. To overcome this issue, a major effort has been made to find methods capable of making deep learning models robust to manipulated inputs. This work presents a new perspective for improving the robustness of DNNs in image classification. In computer vision scenarios, adversarial images are crafted by manipulating legitimate inputs so that the target classifier is eventually fooled, but the manipulation is not visually distinguishable by an external observer. The reason for the imperceptibility of the attack is that the human visual system fails to detect minor variations in color space, but excels at detecting anomalies in geometric shapes. We capitalize on this fact by extracting color gradient features from input images at

multiple sensitivity levels to detect possible manipulations. We resort to a deep neural classifier to predict the category of unseen images, whereas a discrimination model analyzes the extracted color gradient features with time series techniques to determine the legitimacy of input images. The performance of our method is assessed over experiments comprising state-of-the-art techniques for crafting adversarial attacks. Results corroborate the increased robustness of the classifier when using our discrimination module, yielding drastically reduced success rates of adversarial attacks that operate on imperceptible color modifications on the whole image, rather than on localized regions or around the existing shapes of the image.

2.1.5 Attacking ESM-based Classifiers

In line with the previous problem, Chapter 7 evaluates AML attacks over ESM-based time series classifiers. We have previously emphasized that AML studies the robustness of classification models when processing data samples that have been intelligently manipulated to produce a change in their output. Techniques aimed at crafting such adversarial samples exploit concrete vulnerabilities of the classifier at hand, by which perturbations can make a given data instance to be wrongly classified. In this context, the literature has so far gravitated on different AML strategies to modify data instances for diverse learning algorithms, in most cases for image classification. The work in this chapter builds upon this background literature to address AML for distance-based time series classifiers (e.g., nearest neighbors), in which adversarial samples are intelligently devised by taking into account the measure of similarity used to compare time series. In particular, we propose different attack strategies relying on guided perturbations of the input time series based on gradient information provided by a smoothed version of the distance-based model to be attacked. Furthermore, we formulate the AML sample crafting process as an optimization problem driven by the Pareto front defined over 1) a measure of distortion of the input sample concerning its original version; and 2) the probability of the crafted sample to confuse the model. In this case, this formulated problem is efficiently tackled by using multi-objective heuristic algorithms. Several experiments are discussed to assess whether the crafted adversarial time series succeeds when confusing the distance-based model under target.

2.2 Structure of the Thesis

This Thesis is divided into three main blocks. Part I consists of Chapters 1 and 2, and provides a brief preface and an overview of the fundamental elements of TSA. Although the reader is advised to first inspect Chapter 2 for a better understanding of the remainder of the Thesis, an expert in TSA problems with deep knowledge on ESMs can pass over this chapter and use it as a reference, only if needed. Part II, comprising Chapters 3 to 7, embodies the main scientific contributions of the Thesis. There is

no need for reading chapters within this part sequentially in order since they are arranged as per the research line under consideration. As such, Chapter 3 is independent of the others and deals with metric learning for time series classification. On the other hand, Chapters 4 and 5 address the adaptation of ESMs to stream settings. Part II consists of Chapters 6 and 7, which cover the use of ESMs in the context of AML. Finally, Part III, which only contains Chapter 8, summarizes the main contributions of the dissertation and outlines several research directions stemming from the findings presented throughout the Thesis. Figure 2.1 schematically depicts the different reading paths of the Thesis, depending on the background and specific interests of the reader.

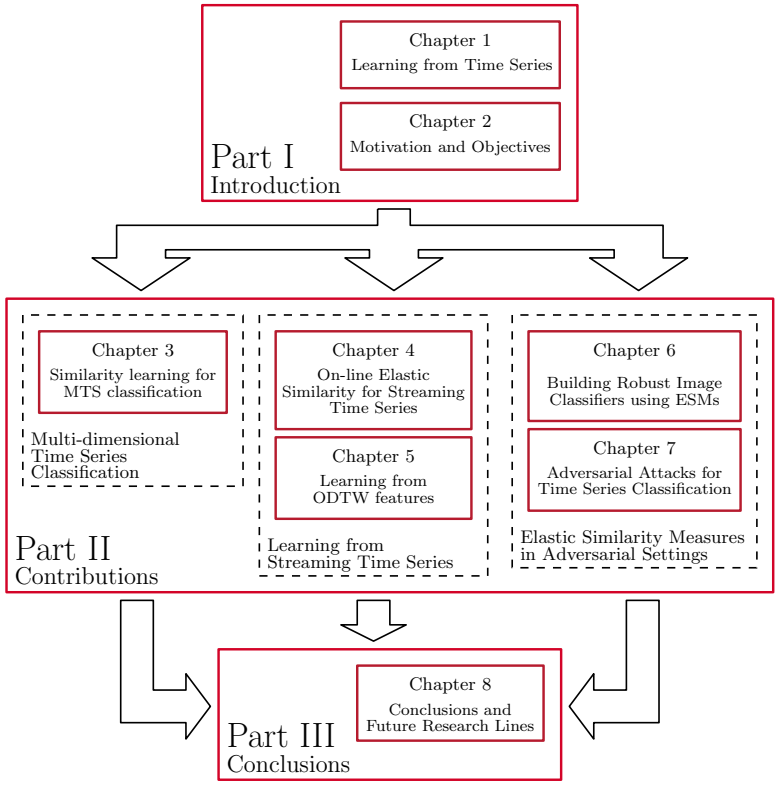


FIGURE 2.1: Diagram showing the structure of the Thesis.

2.3 Notation

Several notation conventions have been established in Chapter 1 to describe the formulation and experimentation shown in Chapters 3 to 7. However, for the sake of readability of the contents in place, the notation may not be consistent at certain points of these chapters. Nevertheless, at such points, the notation abuse will be clarified explicitly when unclear from the context and the common knowledge in the field.

Part II

Contributions

Chapter 3

Similarity Learning for Multi-Dimensional Time Series Classification

In previous chapters, we have emphasized on the outstanding performance of DTW-based TSA methods (see Section 1.3.3 and 1.4), as evinced by the upsurge of literature dealing with their application to different TSA problems. We have also stressed on the fact that DTW has surpassed several similarity and distance proposals to become the similarity measure of choice in many raw-data based TSC models for uni-dimensional time series data [12]. However, in the multi-dimensional case, such a superior performance is not that clear. In this regard, the relationship between the dimensions of the time series should be also considered. Thus, the generalization of the DTW similarity measure to the multi-dimensional case spans several challenges that remain insufficiently tackled to date. This chapter focuses on this research line by presenting an adaptation of the DTW measure for its use in multi-dimensional time series classification problems.

When measuring the similarity among MTS, we can consider each multi-dimensional sequence as a collection of uni-dimensional series thus, performing a DTW for each dimension of the problem independently. On the contrary, we can consider MTS as indivisible sequences. Thereby, DTW is computed at once so that the warping path is built on alignments among multi-dimension data points. We call these two approaches the *independent* and *dependent* DTW, respectively. On this basis, the multi-dimensional measure of similarity introduced in this chapter consists of a weighted combination between the independent and dependent DTW approaches. The optimization of these weights can be conceived as a sort of flexible feature weighting wrapper, by which the weights can be interpreted as the relative importance of the dimensions of the time series for the TSC at hand. We now delve into the fundamentals of this approach.

3.1 Introduction and Related Work

In Chapter 1 we have defined the DTW as a similarity measure that stretches and/or shrinks time series along time – prior to their distance computation – to accommodate local time shifts/warps and obtain the minimum distance between the series at hand. The recurrent equation to compute the DTW as per Expression (1.19) employs as pairwise distance (cost function) the standard Euclidean distance (see Table 1.1). To develop the multi-dimensional version of this measure, in this section we consider the squared Euclidean distance. Accordingly, the DTW between the uni-dimensional time series $X = (x_1, .., x_m)$ and $Y = (y_1, ..., y_n)$ is:

$$\mathrm{DTW}(X, Y) = \sqrt{\gamma_{m,n}}, \tag{3.1}$$

where $\gamma_{m,n}$ is the squared elastic similarity defined as:

$$\gamma_{m,n} = c(x_m, y_n) + \min\left\{\gamma_{m-1,n}, \gamma_{m-1,n-1}, \gamma_{m,n-1}\right\} \tag{3.2}$$

with $\gamma_{0,0} = 0$ and $\gamma_{i,0} = \gamma_{0,j} = \infty$ for $i = 1, \ldots, m$ and $j = 1, \ldots, n$, and $c(x_m, y_n) = (x_m - y_n)^2$ is the squared cost function.

When the points of a time series have several dimensions, we deal with multi-dimensional time series (MTS). In this context, let $\mathbf{X}$ denote a MTS with D dimensions, where each point is given by $\mathbf{x}_i = (x_{1i}, \ldots, x_{di}, \ldots, x_{Di})$, and each dimension by the m-length sequence $X^{(d)} = (x_{d1}, \ldots, x_{di}, \ldots, x_{dm})$ as per Equation (1.1). In this context, a straightforward approach to account for the multi-dimensionality in the above measure of similarity is to treat dimensions separately (independently) or jointly (dependently). The first approach assumes that all dimensions of the time series are independent, yielding a measure of warping similarity computed as the sum of the DTW for each dimension, i.e.:

$$\mathrm{DTW_I}(\mathbf{X}, \mathbf{Y}) = \frac{1}{D}\sum_{d=1}^{D} \mathrm{DTW}\left(X^{(d)}, Y^{(d)}\right). \tag{3.3}$$

However, this independence does not necessarily hold in practical applications. In fact, due to latent variables, it is common in multi-dimensional problems to present complex relations among the dimensions of the MTS. In consequence, the dependent DTW considers all dimensions straightaway using, as the inner cost function, the multi-dimensional version of the squared Euclidean distance. Then, the dependent measure of similarity $\mathrm{DTW_D}(\mathbf{X}, \mathbf{Y})$ is obtained by means of the recurrence in Expressions (3.1) and (3.2), where the cost function is given by:

$$c(\mathbf{x}_m, \mathbf{y}_n) = \sum_{d=1}^{D} (x_{dm} - y_{dn})^2. \tag{3.4}$$

As aforementioned in the introduction of this section, note the main difference between $\mathrm{DTW_I}$ and $\mathrm{DTW_D}$ lies in the number of warping paths. That is, while the former performs a warping path for each dimension of

the problem, the latter computes a single path that considers all dimensions at once.

Beyond this naive approach to account for the multi-dimensionality of time series in the DTW computation, the design of a similarity that properly captures and exploits the exiting relations between dimensions remains an active research area in the field of multi-dimensional time series classification. Bankó and Abonyi in [27] presented a classification algorithm that combines DTW and PCA-based segmentation in a hybrid scheme, coined as correlation based dynamic time warping (CBDTW). The classification algorithm segments MTS homogeneously by using, as the segmentation cost function, the Hotelling's T^2 statistic (i.e. the MTS distance to the origin in the PCA space), or the Q reconstruction error, which represent the information loss between the original data and its projection in the space of principal components. Once the segmentation is done, the DTW is computed. Similarly, [76] proposes a simple algorithm which selects the DTW measure – either independent or dependent – that scores best when predicting the labels, using k-NN, of the time series dataset under analysis. Interestingly for the scope of this chapter, the authors in [76] discovered that the threshold to select one distance or another depends on the training data used for its calculation, which ultimately unveils that practical databases feature a mixture of independence and dependence relationships between their dimensions that should be exploited in the definition of the distance between series. Recently, Mei et al. in [77] proposed a similarity measure called Mahalanobis distance based DTW (MDDTW), which combines Mahalanobis distance learning and DTW for classification tasks. In the proposed method, DTW_D is utilized to find the similarity between MTS. However, instead of using Expression (3.4) as the pairwise distance, they resorted to the generalized Mahalanobis distance given by:

$$c(\mathbf{x}_m, \mathbf{y}_n) = (\mathbf{x}_m - \mathbf{y}_n)\mathbf{S}(\mathbf{x}_m - \mathbf{y}_n)^T, \tag{3.5}$$

where $\mathbf{S} \in \mathbb{R}^{D \times D}$ is a symmetric positive semi-definite matrix. As concluded by the authors in [77], the model learning is computationally expensive due to the need for computing the DTW for different values of $\mathbf{S}$ during the learning process.

Our work is aligned with the above-noted need for computationally efficient multi-dimensional distance learning algorithms, and recent contributions dealing with parametric distance measures for MTS classification [30]. Specifically, we propose a weighted measure of similarity that combines both the independent and the dependent DTW approaches in its definition. Since the selection of a proper similarity is strongly biased by the database at hand, we resort to a heuristic wrapper driven by the cross-validated predictor score. Then, the optimization is just a similarity-based learning problem operating on the modified space spanned by the weighted combination of DTW similarities.

This technical approach can be framed within past contributions dealing with the use of wrapping heuristics for distance-based learners (e.g. [78]), where the distance measure along samples is optimized by weighting the value of their features rather than by tuning the metric itself.

To optimize the weights, four nature-inspired evolutionary meta-heuristics are used, simulated annealing, particle swarm optimization, genetic algorithms, and estimated distribution algorithms. The four alternatives are evaluated and compared by computer experiments over databases utilized in the literature. From the obtained results we will not only show the performance improvements achieved by every heuristic but also provide an intuitive insight on how further gains could be achieved.

The remainder of the chapter is organized as follows: Section 3.2 introduces the definition of the proposed DTW-based multi-dimensional similarity and the optimization procedure by means of a heuristic wrapper. Section 3.3 provides experimental results and finally, Section 3.4 concludes the chapter.

3.2 Proposed Similarity

We consider two multi-dimensional series $\mathbf{X}$ and $\mathbf{Y}$ in the D-dimensional space, each with n data points. Our proposal gravitates on reformulating the similarity as follows:

$$\begin{aligned} \mathrm{DTW}_{opt}(\mathbf{X},\mathbf{Y}) = &\sum_{d=1}^{D} \omega_d \mathrm{DTW}\left(X^{(d)}, Y^{(d)}\right) \\ &+ \left(1 - \sum_{d=1}^{D} \omega_d\right) \mathrm{DTW}_D(\mathbf{X},\mathbf{Y}), \end{aligned} \tag{3.6}$$

where $0 \leq \omega_d \leq 1/D$ and $0 \leq \sum_{d=1}^{D} \omega_d \leq 1$. Note from the definition that both DTW_D and DTW (for different dimensions) can be computed and storage in advance. Hence, in terms of computational cost, the optimization of the weights $\boldsymbol{\omega} = \{\omega_d\}_{d=1}^{D}$ is more efficient. The above definition allows for the flexibility required to tackle the MTS classification: depending on the values given to $\omega_1, \ldots, \omega_D$, the resulting similarity can range from the total dependence to a framework where all dimensions are independent of each other (i.e., from $\omega_d = 0\ \forall d \in \{1, \ldots, D\}$ to $\sum_{d=1}^{D} \omega_d = 1$).

3.2.1 Weights Optimization Procedure

As shown in Figure 3.1 (next page), values of $\boldsymbol{\omega}$ are optimized by means of a heuristic wrapper, where the score function is the accuracy obtained from a Nearest Neighbor (NN) classifier. Taking into account that NN is sensitive to changes in the training set and that it can suffer from overfitting, we follow [79] and compute the accuracy by using an estimator with low variance, namely, the m-repeated k-fold cross-validation ($m \times k$-cv), with $m = 10$ and $k = 2$. The $m \times k$-cv estimator consists of averaging m different performance estimates provided by a stratified k-fold cross-validation (k-cv).

Algorithm 3.1 (next page) sketches the procedure to measure the fitness of a set of $\boldsymbol{\omega}$ candidate values for any given training dataset. The k-cv error estimation procedure splits the training data into mutually exclusive

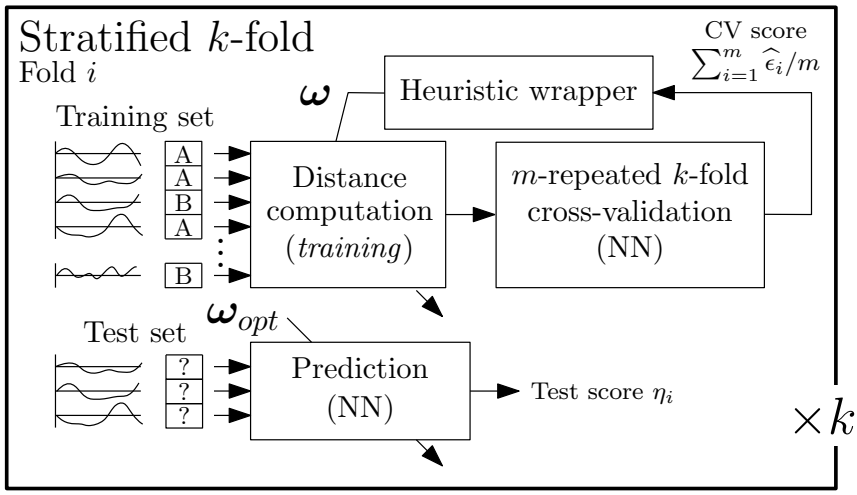


FIGURE 3.1: Proposed wrapper scheme for similarity optimization.

k-folds (line 3), from which training and validation sets are successively constructed (lines 5 and 6). Given a value of $\boldsymbol{\omega}$, the classifier is trained and evaluated using Expression (3.6) as the similarity among examples in a 1-NN model. The k-cv estimation results from averaging the performance scores achieved with each fold.

Algorithm 3.1 Computation of the fitness for the wrapper

1: **Require:** k, m, $\boldsymbol{\omega}$, training data
2: **for** $i \in \{1, \ldots, m\}$ **do**
3: ▷ Shuffle the training data
4: ▷ Sample k stratified folds from training data
5: **for** $j \in \{1, \ldots, k\}$ **do**
6: Set the j-th fold as the validation set
7: Set remaining folds as the training set
8: **for all** sample in the validation set **do**
9: Compute DTW_{opt} between the validation sample and every sample in the training set
10: Predict the label for the validation sample to be that of the training sample with $\min \mathrm{DTW}_{opt}$
11: **end for**
12: Compute ϵ_j score by comparing predicted and true labels
13: **end for**
14: compute k-cv as $\widehat{\epsilon}_i = \sum_{j=1}^{k} \epsilon_j/k$
15: **end for**
16: **return** fitness value given by $\sum_{i=1}^{m} \widehat{\epsilon}_i/m$

Regarding the heuristic wrapper, four are the different solvers utilized to seek the optimum value of weights $\boldsymbol{\omega} = \{\omega_d\}_{d=1}^{D}$: Simulated Annealing

(SA), Particle Swarm Optimization (PSO), Estimation of Distribution Algorithm (EDA) and Genetic Algorithm (GA). A brief description of each of these methods is next provided:

- SA [80] is a low-complexity optimization algorithm known to efficiently tackle problems with a small number of variables. The search process underlying this heuristic emulates the annealing technique in metallurgy, by which a material is heated and cooled in a controlled fashion to lead it to a state with minimum internal energy and hence, maximum hardness. This iterative search process is controlled mainly by 1) the method to permute the candidate solution at a given iteration; and 2) the temperature T of the material, which sets the probability that the mutated individual is accepted as the candidate solution.

- PSO [81] consists of a swarm of particles moving in the space of candidate solutions. Each individual in the swarm is characterized by its position over the search space (which in turn represents the solution $\boldsymbol{\omega}$ proposed by the particle), a velocity vector $\mathbf{v}$, and the memory of both its own best solution and the global best achieved by the entire swarm. The optimization procedure consists of spreading the information about good solutions through the swarm so that particles move over the space under a velocity vector biased by the positions of the aforementioned best solutions in the swarm.

- EDA [82] is a population-based optimization algorithm that guides the search by sampling promising solutions from learned generative probabilistic models. In EDAs new individuals are sampled from a probability distribution estimated from the previous generation of solutions and their associated fitness values. In this work, we assume for simplicity a canonical EDA, where optimization variables are assumed to be independent from each other in the probabilistic model.

- GA [83] are heuristic solvers inspired by observed processes in the genetic inheritance among generations of individuals. The main stages of the algorithm are selection, crossover, mutation, evaluation, and replacement. The search technique begins with a randomly generated initial population of individuals, from which a number of breeding solutions or parents are selected based on their fitness values. Then, in the crossover stage, two individuals are taken randomly from the selected population and combined to yield offspring solutions. Finally, each offspring solution undergoes small perturbations of its compounding variables under probability p_m. All offspring solutions are then evaluated, replacing the previous population. The procedure is repeated until a termination criterion is met (e.g. a fixed number of generations, or a marginal improvement of the best solution over successive generations).

3.3 Experimental Results and Discussion

The proposed approach is validated over different databases utilized in the literature related to time series classification. In particular, we will

use the articulatory word [84], cricket, and Auslan (sign language used by the Australian deaf community) [85] databases. It is important to remark that for all databases, time series have been normalized via Z-score so that each dimension has zero mean and unit standard deviations, i.e., $\forall d \in \{1, \ldots, D\}$ of a given MTS $\mathbf{X}$ of every database, we perform:

$$X_{\text{norm}}^{(d)} = \frac{X^{(d)} - \mathrm{E}\left[X^{(d)}\right]}{\sqrt{\mathrm{Var}\left[X^{(d)}\right]}}, \tag{3.7}$$

where $\mathrm{E}[\cdot]$ and $\mathrm{Var}[\cdot]$ denotes expectation and variance, respectively. A brief description of each database and the performed experiments is provided in what follows:

- The Auslan dataset (`AUSLAN`) comprises 95 different signs performed by 5 signers, yielding a total of 6648 time series, each with $d = 15$ dimensions [86]. From these dimensions x, y, z and *roll* attributes have been selected as predictors for a small subset of the overall database corresponding to sequences labeled with *all*, *answer*, *boy*, *buy*, *cold*, *come*, *crazy*, *different*, *exit* and *forget*. On the one hand, x, y and z variables record the up-down, right/left and forward/backward movements of the signers' hands, respectively. However, they should not be considered to compose an orthogonal basis, hence relations among variables are expected to emerge. On the other hand, the *roll* dimension tracks the palm rotation.

- The Cricket database (`CRICKET`) consists of a collection of 12 referee signals, each with 10 repetitions. The data contains observations of x, y and z axes motion measured with an accelerometer placed on both, left and right wrists of the referee. As in [76], we will use different dimension pairs to predict each signal.

- The Articulatory word dataset (`ARTI`) contains tongue, lips and head motion (using 12 sensors) of native English speakers performing 25 different words. Altogether, the dataset amounts up to 575 time series, each comprising the x, y and z position of each sensor. Out of the $D = 36$ available dimensions, we will use different combinations considering the sensors on the tongue tip (T1), the upper (UL), and the lower lip (LL), as done in [76].

Regarding the nature-inspired solvers, a summary of the specific parameter values utilized for each method is provided below. It must be noted that the same termination criterion is utilized for all algorithms in the benchmark, i.e. the algorithm is forced to stop when the fitness value of the best proposed solution does not improve for a maximum number of generations $\max_{\text{gen}}$.

- In the i-th iteration of the SA solver, the acceptance of a new solution $\boldsymbol{\omega}^{i+1} = \boldsymbol{\omega}^{i} + n$ – where n is a random variable given by a standard normal distribution, $\mathcal{N}(0, 1)$ – is ruled by $\boldsymbol{\omega}^{i+1}$, $\boldsymbol{\omega}^{i}$ and a temperature parameter

T_i that jointly define the solution acceptance probability. The temperature of the algorithm is enforced to go from value 1 to 0 along the iterations of the algorithm as $T_{i+1} = \xi T_i$ where $0 \leq \xi < 1$ is the cooling rate. In all simulations ξ and $\max_{\text{gen}}$ are set to 0.1 and 100, respectively.

- In PSO the parameters of the algorithm have been chosen so that the movement of each particle in the search space $\boldsymbol{\omega}$ is governed by $\boldsymbol{\omega}_{i+1} = \boldsymbol{\omega}_i + \boldsymbol{\alpha}_{\text{indv}} + \boldsymbol{\alpha}_{\text{global}} + \boldsymbol{\alpha}_{\text{neigh}}$, where $i \in \{1, \ldots, I\}$ denotes the iteration index and the vectors in the right side $\boldsymbol{\alpha}_{\text{indv}} = 0.5(\boldsymbol{\omega}_i - \boldsymbol{\omega}_{i-1})$, $\boldsymbol{\alpha}_{\text{global}} = 2.1 n_{\text{global}}(\boldsymbol{\omega}_{\text{global}} - \boldsymbol{\omega}_i)$ and $\boldsymbol{\alpha}_{\text{neigh}} = 2.1 n_{\text{neigh}}(\boldsymbol{\omega}_{\text{neigh}} - \boldsymbol{\omega}_i)$ correspond to the tendency to move towards the previous position, the influence to move towards the entire swarm best solution ($\boldsymbol{\omega}_{\text{global}}$), and the influence to move towards the neighbours best solution ($\boldsymbol{\omega}_{\text{neigh}}$), respe ctively. The maximum number of generations $\max_{\text{gen}}$ is set to 100, whereas n_{global} and n_{best} are realizations of a continuous random variable uniformly distributed in the range $[0, 1]$.

- In GA the population size is 40 individuals, from which the number of solutions in the selection step have been set to 20. In the crossover stage, the chosen operator is single-point crossover with probability $p_c = 0.9$. Finally the mutation of each offspring element $\omega_d \in \boldsymbol{\omega}$ is changed to $\omega_d + n$ with mutation probability $p_m = 0.1$ where $n \sim \mathcal{N}(0, 1)$. As in previous solvers the stopping criteria have been set to $\max_{\text{gen}} = 100$.

- In EDA the offspring values $\boldsymbol{\omega}^i$ of the i-th generation are drawn from a multidimensional Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}^i, \boldsymbol{\Sigma}^i)$ whose mean vector $\boldsymbol{\mu}^i$ and covariance matrix $\boldsymbol{\Sigma}^i$ are given by

$$\boldsymbol{\mu}^i = (\mu_1^i, \ldots, \mu_d^i, \ldots, \mu_D^i),\ \mu_d^i = \mathrm{E}[\boldsymbol{\omega}_d^{i-1}] \tag{3.8}$$

$$\boldsymbol{\Sigma}^i = \mathrm{diag}(\Sigma_1^i, \ldots, \Sigma_d^i, \ldots, \Sigma_D^i)\ :\ \Sigma_d = \mathrm{Var}[\boldsymbol{\omega}_d^{i-1}] \tag{3.9}$$

where $\boldsymbol{\omega}_d^{i-1} = \{\omega_d^{1,i-1}, \ldots, \omega_d^{100,i-1}\}$ corresponds to the vector collecting the values of the d-th variable along a 100-sized population of individuals drawn from $\mathcal{N}(\boldsymbol{\mu}^{i-1}, \boldsymbol{\Sigma}^{i-1})$ at generation $i - 1$.

3.3.1 Discussion on Predictive Score Results

Besides the computation of a performance estimate to evaluate the fitness of the $\boldsymbol{\omega}$ parameters proposed by the heuristic wrapper, the goodness of the optimized classifier when trained over DTW_{opt} should be measured over unseen test data. For this reason, we will use again stratified K-fold cross-validation to first split the entire database in training and test sets. Algorithm 3.2 illustrates, for K partitions, the procedure to compute the goodness measure. In essence, training and test sets are constructed using $K - 1$ folds for the former, and the remaining fold for the latter. For all possible training-test combinations, optimized $\boldsymbol{\omega}$ values are found by using one of the algorithms described previously (line 6). The average of the K computed scores for the train-test splits yields a measure of the expected performance of the proposed wrapper when facing new test samples.

TABLE 3.1: Average, first quartile and third quartile scores for the simulated databases and heuristic wrappers.

Dataset label	Variables	DTW_I		DTW_D		DTW_{opt}^{SA}		DTW_{opt}^{PSO}		DTW_{opt}^{GA}		DTW_{opt}^{EDAs}	
		Avg.	Q1 – Q3	Avg.	Q1 – Q3	Avg.	Q1 – Q3	Avg.	Q1 – Q3	Avg.	Q1 – Q3	Avg.	Q1 – Q3
AUSLAN-XYZR	$x, y, z, roll$	0.757	0.714 – 0.786	0.789	0.750 – 0.812	0.800	0.786 – 0.848	**0.814**	0.768 – 0.848	0.796	0.786 – 0.812	0.793	0.759 – 0.821
AUSLAN-XYZ	x, y, z	0.805	0.726 – 0.857	0.614	0.583 – 0.667	0.805	0.726 – 0.857	0.800	0.726 – 0.857	**0.814**	0.768 – 0.848	0.776	0.714 – 0.845
CRICKET-XRXL	x_{right}, x_{left}	0.946	0.917 – 0.958	0.921	0.917 – 0.927	**0.971**	0.958 – 1.000	**0.971**	0.958 – 1.000	0.971	0.958 – 1.000	**0.971**	0.958 – 1.000
CRICKET-XRYL	x_{right}, y_{left}	0.954	0.917 – 0.990	0.954	0.917 – 1.000	**0.975**	0.958 – 1.000	0.971	0.958 – 1.000	0.958	0.927 – 0.990	0.967	0.958 – 1.000
CRICKET-YRXL	y_{right}, x_{left}	0.988	1.000 – 1.000	0.983	1.000 – 1.000	**0.992**	1.000 – 1.000	**0.992**	1.000 – 1.000	**0.992**	1.000 – 1.000	0.983	1.000 – 1.000
CRICKET-YRYL	y_{right}, y_{left}	0.969	0.979 – 1.000	0.988	1.000 – 1.000	**0.992**	1.000 – 1.000	0.988	1.000 – 1.000	**0.992**	1.000 – 1.000	**0.992**	1.000 – 1.000
CRICKET-ZRZL	z_{right}, z_{left}	0.954	0.917 – 1.000	0.975	0.938 – 1.000	**0.979**	0.969 – 1.000	**0.979**	0.969 – 1.000	0.971	0.969 – 1.000	0.971	0.969 – 1.000
ARTI-UXTZ	UL_x, $T1_z$	0.839	0.830 – 0.860	0.863	0.845 – 0.880	0.891	0.883 – 0.907	**0.902**	0.900 – 0.920	0.894	0.880 – 0.920	0.894	0.880 – 0.917
ARTI-TZLZ	$T1_z$, LL_z	0.895	0.880 – 0.917	0.941	0.923 – 0.960	0.938	0.923 – 0.957	0.940	0.935 – 0.945	0.942	0.935 – 0.960	**0.944**	0.923 – 0.960
ARTI-TZLY	$T1_z$, LL_y	0.887	0.880 – 0.917	**0.933**	0.920 – 0.955	**0.933**	0.923 – 0.955	0.923	0.900 – 0.938	0.926	0.920 – 0.940	0.926	0.920 – 0.938
ARTI-LXTYTZ	LL_x, $T1_y$, $T1_z$	0.923	0.885 – 0.960	0.954	0.950 – 0.978	**0.966**	0.945 – 0.985	0.965	0.945 – 0.978	0.963	0.942 – 0.978	0.965	0.945 – 0.987
ARTI-TYTZ	$T1_y$, $T1_z$	0.892	0.883 – 0.917	**0.949**	0.925 – 0.960	0.944	0.923 – 0.960	0.945	0.925 – 0.960	0.943	0.925 – 0.960	0.945	0.925 – 0.960
ARTI-TYLZ	$T1_y$, LL_y	0.863	0.840 – 0.880	0.919	0.902 – 0.940	**0.924**	0.902 – 0.940	0.921	0.900 – 0.940	**0.924**	0.900 – 0.955	**0.924**	0.900 – 0.955
ARTI-TXTYUZ	$T1_x$, $T1_y$, UL_y	0.912	0.900 – 0.920	0.931	0.920 – 0.945	0.944	0.940 – 0.957	0.939	0.925 – 0.957	0.940	0.940 – 0.947	**0.947**	0.940 – 0.960
ARTI-TXTYTZ	$T1_x$, $T1_y$, $T1_z$	0.928	0.897 – 0.955	0.940	0.920 – 0.960	**0.945**	0.920 – 0.960	0.942	0.920 – 0.972	**0.945**	0.923 – 0.975	0.939	0.925 – 0.957

Algorithm 3.2 Computation of the test score

Require: K, dataset
Split dataset into K stratified folds
for $i \in \{1, \ldots, K\}$ **do**
    Set the i-th fold as the test set
    Set the remaining folds as the training set
    Optimize $\boldsymbol{\omega}$ using the training set, a heuristic solver and the fitness in Algorithm 3.1
    Predict test set labels with the classifier using $\text{DTW}_{opt}(\cdot, \cdot)$ with the optimized $\boldsymbol{\omega}$
    Compute performance score of this fold as η_i
end for
return K-cv test score as $\sum_{i=1}^{K} \eta_i / K$

Table 3.1 shows the average, first and third quartile of the estimated accuracy rates obtained by the independent DTW_I, dependent DTW_D and optimized DTW_{opt} models for each solver. The bold value in the table indicates the best average accuracy rate among all schemes in this benchmark. Several observations can be drawn from the results reported in this table. To begin with, all wrappers have similar performance, being SA slightly more accurate than the rest of the heuristic methods. Results for the `AUSLAN` dataset suggest that when compared to DTW_I and DTW_D, DTW_{opt} is more resilient to the selection of the variables, specially when tackled with the SA and PSO algorithms. `CRICKET` scores are complex to analyze because all models achieve high accuracy values over the simulated variable combinations. An exception is the `CRICKET-XRXL` case, for which the predictive score with the optimized similarity is higher than its dependent and independent counterparts for all utilized wrappers. Regarding the rest of datasets, lower performance gains are noted; we can conclude that in general, our model is at least as accurate as the best among DTW_I and DTW_D, regardless the method in use.

3.3.2 Discussion on the Optimized Weight Values

The average accuracy values shown in Table 3.1 provide, in terms of model fitness, the comparison of DTW_{opt} with DTW_I and DTW_D similarity-based models. However, such results do not render any insights on the statistical distribution of the obtained scores, nor do they shed any light on the values of their associated weights $\boldsymbol{\omega}$. A further analysis of the distribution of 10-fold cross-validation accuracy scores and the optimized values of $\{\omega_1, \ldots, \omega_D\}$ is done in Figures 3.2a through 3.2f in the form of box-plots. The subset of simulated cases included in these plots is a representative sample that best illustrates the casuistry that recurrently occurs in all performed experiments.

To begin with, Figures 3.2a and 3.2d illustrate the outcomes of the PSO solver where DTW_{opt} outperforms both DTW_D and DTW_I, with

non-zero values for all $\{\omega_1, \ldots, \omega_D\}$. This indicates that relationships existing between dimensions are important for classification. Moreover, to verify that the performance gaps for DTW_{opt} and those of DTW_D and DTW_I are statistically significant, we have performed a non-parametric Wilcoxon signed-rank test to check whether result samples come from distribution with different medians. Since the obtained p-value falls below 0.05 for both cases the hypothesis of statistical significance is confirmed. In particular, we get p-value = 0.007 when the test is performed with DTW_{opt} and DTW_D score samples and p-value = 0.01 with DTW_{opt} and DTW_I. Although we have mentioned before that SA is slightly more accurate, note that for this particular case, PSO is among the solvers with highest accuracy rates.

The discussion follows by Figures 3.2b and 3.2e, which exemplify, for the `ARTI-TZLY` database and SA solver, the case when DTW_{opt} and DTW_D render a similar predictive performance. As could be expected beforehand, the optimized (ω_1, ω_2) weights that gauge the contribution of each dimension in isolation to the optimized similarity in (3.6) are close to zero, in contrast to the dependent part contribution. By contrast, SA results for the `AUSLAN-XYZ` database (Figures 3.2c and 3.2f) unveil an identical performance of the independent and optimized similarity models, and a notably worse behavior of DTW_D. One would accordingly expect high values for the weights $\{\omega_1, \omega_2, \omega_3\}$ (close to $1/D = 1/3$), so that the independent part in (3.6) dominates over DTW_D. This does not hold in the plotted results, where the contribution of both independent and dependent parts are similar; even more, the dependent contribution is never negligible. The rationale behind this contradictory effect might lie on the tight coupling among variables imposed in the search for the warping path in DTW_D. As established by the proposed definition of DTW_{opt}, the optimization of the contribution of the dependent part to this combined similarity does not discriminate between dimensions. The variability of the weights in y and, more specifically, z dimensions (see Figure 3.2f) indicates that the DTW_D part is somehow compensated in order to reduce the lack of predictability of one of both variables equally weighted inside the dependent part. This last observation suggests that a generalization of the cost function in Equation (3.4) might allow for the optimization of each variable (dimension) of the dependent DTW.

3.4 Conclusions

In this chapter, we have defined a simple similarity measure for multidimensional time series classification that not only leverages the ability to accommodate time warps featured by the DTW, but also takes into account possible relations existing among dimensions. The proposed scheme is based on a heuristic wrapper that optimizes the values of the weights balancing the contribution of independent and dependent DTW. The optimization criterion is based on the maximization of the cross-validated prediction score of a distance-based classifier operating on the similarities iteratively refined by the heuristic. A benchmark of four meta-heuristic

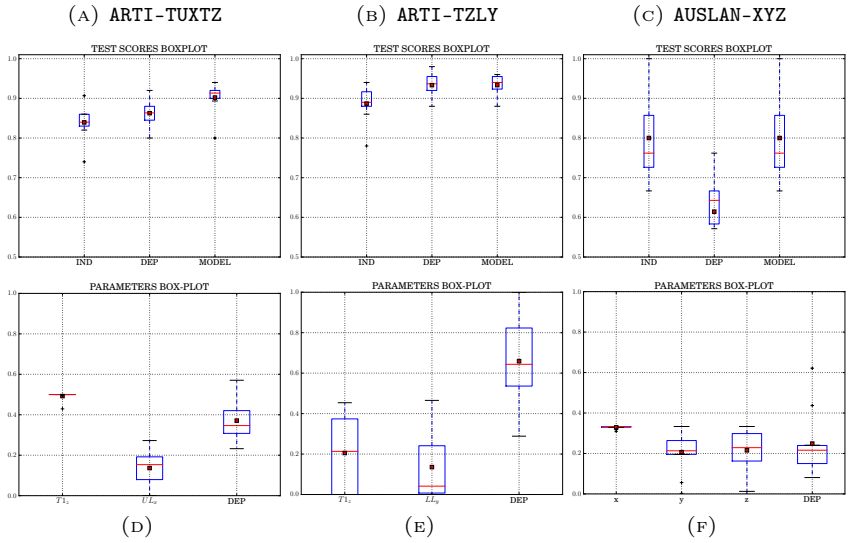


FIGURE 3.2: Box-plot corresponding to the test scores obtained via stratified 10-fold corresponding to the `ARTI-TUXTZ` (a), `ARTI-TZLY` (b) and `AUSLAN-XYZ` (c) datasets, along with box-plots showing the distribution of weights $\boldsymbol{\omega}$ obtained for each case: (d), (e) and (f), respectively. The red square denotes sample mean.

algorithms (SA, PSO, GA and EDA) has been used to learn the weights. When assessed the performance of these optimization algorithms over several databases from the literature, we have verified, on one hand, that our proposed model with a 1-NN classifier performs, in general, equal or better than the same learner with independent and dependent DTW similarity measures. On the other hand, we have seen that the performance gap between the heuristic solvers under consideration is narrow, yet SA seems to return, on average, slightly more accurate results.

Although the proposed measure has proven to be competitive, some open research paths can be followed to further improve its performance. We have seen that the adaptability of the independent part is higher than that of the dependent part due to its ability to treat each dimension separately. To ensure the variability of the dependent part in Expression (3.6), the definition of the inner distance in $\text{DTW}_{\boldsymbol{D}}$ should allow weighting differently each dimension in the warping path discovery process. In close agreement with the conclusions drawn in a similar study [77], this alternative formulation could additionally decrease the computation cost. Other error rate functions could be also considered – such as larger-margin nearest neighbor which is often utilized in distance learning tasks [87] – in order to guide the learning process of the weights to singularities of the classification problem (e.g, class imbalance).

Chapter 4

On-line Elastic Similarity Measures for Streaming Time Series

Considering the acclaimed performance of ESMs in conventional (i.e. off-line) distance-based TSA methods, this chapter examines how such measures can be adapted to streaming time series scenarios, where streams are composed of a sequence of events (see Section 1.1). As already anticipated in Chapter 1, the challenge in this learning scenario resides in the need for properly accounting for the correlation of the time series in an incremental fashion, along with the adaptation of the ESM to accommodate the transitions between successive events along the stream. Specifically, the chapter elaborates on the definition and validation of an on-line ESM that blends together 1) an efficient incremental computation of elastic similarity measures, and 2) a parametric forgetting mechanism that makes the similarity efficiently deal with transitions among events.

4.1 Introduction and Related Work

We have hitherto assumed that data are permanently stored in memory and thereby accessible to algorithms, on request, enabling procedures to pre-process data prior to model learning. However, in many real-world applications, this assumption does not necessarily hold. Instead, data are produced at high speed creating massive amounts of samples that must be analyzed as fast as possible, and then discarded, so as to deal with memory limitations. Moreover, arriving streams might not be stationary i.e., the underlying structure of the time series may change over time (see Section 1.1). As a consequence of these two issues, most of the algorithms proposed in the literature for (non-streaming) time series cannot be applied to analyze such data, mainly due to their lack of adaptation. In learning tasks defined over streaming time series, the need for adaptation is of utmost relevance when shifting from one event to another. The model designed to undertake the task must capture and exploit the time correlation of the time series under analysis and, at the same time, must selectively forget

the learned knowledge when data belonging to a new event (time series) arrives over the stream.

In this sense, the TSA community has recently focused its research on the development of new strategies that might handle the challenges arising from streaming data [6], [7]. For instance, Cavalcante et al. [10] have recently proposed FEDD, a concept drift detector that identifies changes in time series data generator models. Based on the feature vector similarity given by Pearson correlation distance (or cosine distance), this method monitors the evolution of sequence features in order to test whether a change has occurred in observed data. In [3] an incremental clustering system for time series data streams is presented. The so-called On-line Divisive-Agglomerative Clustering is a tree-like grouping technique that evolves with data based on a criterion to merge and split clusters using a correlation-based dissimilarity measure. In this sense, when measuring the similarity among data streams in on-line scenarios, L_p-norm and correlation-based distances are preferred to ESMs [88], [89]. Clearly, the choice of lock-step distances over ESMs relies on the fact that the former proposals provide an exact incremental expression, with minimal memory consumption and demanding a low computational effort.

Bearing this background in mind, a straightforward question that remains open to date is how to adapt ESMs to streaming time series, so that on-line distance-based models can capitalize on their benefits. This chapter addresses this task by proposing an On-line Elastic Similarity Measure (OESM), whose definition hinges on two essential characteristics: the first is a forgetting mechanism, which allows OESM to deal with different stationary intervals (events), whereas the second corresponds to a spotted property of elastic similarities that is exploited to compute the measure within a constant time and a fixed amount of consumed memory. To this end, we consider two different scenarios we coin as *on-line pattern* and *batch-pattern* scenarios. In the former, the OESM is adapted to measure the similarity between two streaming time series. In the latter, the similarity is computed between a streaming time series and a stored sequence. Two experiments are conducted to examine different aspects of the proposed OESMs. First, we test the computational efficiency of the OESM in terms of complexity. Then we study the effect of the forgetting mechanism by analyzing the results drawn by a 1-NN classifier over transitions between events.

The remainder of the chapter is organized as follows: Section 4.2 introduces the proposed OESM. Section 4.3 describes the experimentation and Section 4.4 discusses the obtained results. Finally, in Section 4.5 we summarize the main findings of the chapter, and outline future research lines.

4.2 On-line Elastic Similarity Measures

Analogously to the formulation of the offline ESM in Section **??**, the proposed OESM can be expressed as an optimization problem. That is, OESM consists of finding the path that aligns two time series with minimum cost,

from their beginning to their end, without skipping any point or going steps backward in time. However, as already emphasized in the introduction of this chapter, OESM must be computed incrementally to comply with the computational requirements of on-line settings, namely, fast execution and limited memory consumption. Based on dynamic programming principles similar to those in [90], the incremental computation proposed in this chapter avoids unnecessary computations thanks to a set of thoroughly selected and stored measurements, which we refer to as *frontier*. On the other hand, the adaptation of ESM to streaming time series is performed through a memory function. By under-weighting the contribution of past events, the memory function is set to forget the past and fit OESM to recent events. Moreover, this forgetting mechanism can be modified depending on the characteristics of the problem at hand. In this regard, two problem scenarios are considered:

- Batch pattern: in this first scenario, one of the time series under comparison (say, X^m) is stored in memory, whereas only the other sequence (Y^n) receives new examples over time, as in streaming classification or early classification problems [91].
- On-line pattern: in this second scenario, both time series X^m and Y^n receive new data points over time, as occurs in clustering tasks for streaming time series [3].

In what follows, the memory function and the incremental computation are introduced and described, with an emphasis on the required adaptations to treat on-line or batch pattern scenarios.

4.2.1 Definition of the Proposed OESM

To make OESM consistent with the evolution of the streaming time series, we propose a simple yet effective forgetting mechanism that mitigates, by means of a memory function, the contribution of earlier time series alignments in the ESM definition. Following the notation of the conventional ESM (Equation (1.16)), let X^m and Y^n denote two time series, and $p \in \mathcal{P}$ a path. In the on-line setting, the weight of a path p is redefined to be:

$$w_\rho(p) = \sum_{(i,j)\in p} \rho^{f(i,j)} c_{i,j}, \tag{4.1}$$

where $f : \mathbb{N}^2 \to \mathbb{R}$ is what we hereafter refer to as *memory function*, and $\rho \in (0, 1]$ the *memory parameter*. Note that the definition given above is just a generalization of Equation (1.16). Indeed, if $\rho = 1$, the weight of the path in Equation (4.1) is equivalent to that in Equation (1.16). Departing from these definitions, the proposed OESM is formulated in Definition 2. The reader should note that the adaptation of the ESM to on-line settings is produced just by replacing Equation (4.1) in (1.17).

Definition 2 *(On-line Elastic Similarity Measure) Let X^m and Y^n be two time series, and $w_\rho(p)$ the weight of a path p for a given memory parameter*

$\rho \in (0, 1]$. *The OESM between time series* X^m *and* Y^n *is defined as:*

$$D_\rho(X^m, Y^n) = \min_{p \in \mathcal{P}} w_\rho(p), \tag{4.2}$$

where $\mathcal{P}$ *is the set of allowed paths*

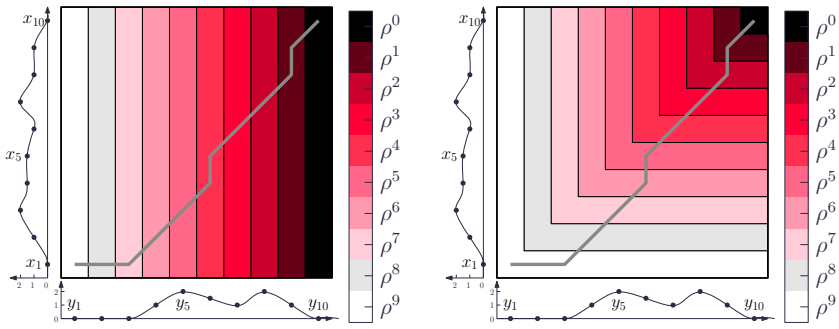
As in Chapter 1, we will use $D_{m,n}$ instead of $D_\rho(X^m, Y^n)$ to denote the similarity between X^m and Y^n streams. Next introduce the adaptation of the memory function to the particularities of the batch and on-line pattern scenarios.

Batch Pattern Scenario

Based on Definition 2, let us assume that Y^n is a streaming time series, whereas X^m is a sequence permanently stored in memory (representing e.g. a reference time series corresponding to a certain label to be predicted by a similarity-based classifier). In this scenario, the memory function in Expression (4.2) is given by:

$$f(i, j) = n - j. \tag{4.3}$$

An example of this particular memory function is shown in Figure 4.1a, where colored bands illustrate all (i, j) pairs in the lattice $[1, 10] \times [1, 10]$ sharing the same $f(i, j)$ value. Note that, by modulating the cost function with this memory function as per (4.3), the last alignments contribute more significantly to the weight of a given path than those that are far from the current sample (m, n) (Equation (4.1)). That is, the contribution of $c(i, j)\rho^{f(i,j)}$ to the resulting similarity value depends on the difference between n and j. Since in the batch pattern scenario X^m does not evolve over time, the proposed forgetting mechanism works with the evolution of Y^n (see Figure 4.1a), enabling OESM to forget its past.



(A) Batch pattern scenario (B) On-line pattern scenario

FIGURE 4.1: Representation of the memory function over the $[1, 10] \times [1, 10]$ lattice given two time series X^{10} and Y^{10}, for the batch (left) and on-line pattern scenarios (right). The color shapes represent the cells sharing the same values of $\rho^{f(i,j)}$. The gray dashed line depicts a warping path $p \in \mathcal{P}$.

On-line Pattern Scenario

In the on-line pattern scheme, the memory function corresponds to:

$$f(i,j) = \max\{m-i, n-j\}. \tag{4.4}$$

An example of the function is shown in Figure 4.1b for $m = n = 10$. Again, the colored bands in the plot denote all $(i,j) \in [1,m] \times [1,n]$ sharing the same $f(i,j)$ value. Moreover, the contribution of a (i,j) pair to the memory weight does not depend on the path, but on the proximity to the (m,n) point. In this case, the proposed memory function makes OESM forget past events of X^m and Y^n streams in accordance with their evolution rate. The reader should note that the proposed memory function corresponds to the Chebyshev (or L_∞-norm) distance.

Convergence of the Proposed OESM

Considering the definition of the memory function either in Equation (4.3) or in (4.4), we can assume that $\exists M > 0$ such that $c_{i,j} \leq M$, for any (i,j) in the $[1,m] \times [1,n]$ lattice. Hence, it is straightforward to see that Equation (4.1) is upper-bounded by a geometric series as:

$$w_\rho(p) \leq \sum_{i=1}^{\max\{m,n\}} \rho^{i-1} M. \tag{4.5}$$

It is known that when m and/or n tend to infinity, the geometric series converge if $|\rho| < 1$. Consequently, the similarity measure in Equation (4.2) would be upper bounded by:

$$D_{m,n} \leq \frac{M}{1-\rho}, \tag{4.6}$$

as long as $\rho < 1$. Specifically, since $c_{i,j} \leq 1$ for EDIT and EDR similarity measures, we can set $M = 1$. As a result, $D_{m,n} \leq (1-\rho)^{-1}$ holds for these particular measures.

4.2.2 Incremental Computation

We recall our arguments in Section 1.4.1, where we stated that ESM can be efficiently computed using dynamic programming methods. Likewise, we can solve the optimization problem in Equation (4.2). In this case, the recursive equation that provides the solution to OESM is given by:

$$D_{m,n} = c_{m,n} + \min \left\{ \begin{array}{l} \rho^{f(m-1,n)} D_{m-1,n}, \\ \rho^{f(m-1,n-1)} D_{m-1,n-1}, \\ \rho^{f(m,n-1)} D_{m,n-1} \end{array} \right\}, \tag{4.7}$$

where $D_{0,0} = 0$ and $D_{0,j} = D_{i,0} = \infty$ for $i = 1, 2, ..., m$ and $j = 1, 2, ..., n$. The computational complexity of the OESM is $O(m \cdot n)$, which results unfeasible for two time series that grow without bound. In order to alleviate the

complexity and adapt the computation to the on-line setting, we use the following observation, which stems directly from Equation (4.7):

Observation 1 *When computing the ESM, we compute not only the final result $D_{m,n}$, but also the intermediate results $D_{i,j}$ for all $i = 1,...,m-1$ and $j = 1,...,n-1$. In other words, the ESM between two time series X^m and Y^n requires the computation of the ESM between all subsequences $X^i = (x_1,...,x_i)$ and $Y^j = (y_1,...,y_j)$, for $i = 1,...,m-1$ and $j = 1,...,n-1$.*

In line with this observation, we define the *frontier* as:

$$\mathbf{F}^{m,n} = \{D_{i,n}\}_{i=1}^{m} \cup \{D_{m,j}\}_{j=1}^{n}, \tag{4.8}$$

which will be used to avoid unnecessary computations when new data samples arrive over the stream.

Turning our attention to streaming scenarios, we consider that streaming time series evolve from X^r to X^m and from Y^s to Y^n after receiving $X^{r+1,m} = (x_{r+1},...,x_m)$ and $Y^{s+1,n} = (y_{s+1},...,y_m)$ data samples, respectively. The goal of the incremental computation is to calculate $D_{m,n}$ by resorting to minimum computational resources. To this end, we assume that, before receiving $X^{r+1,m}$ and $Y^{s+1,n}$, we store in memory the frontier $\mathbf{F}^{r,s}$. By definition, $D_{m,n}$ requires the computation of the whole $m \times n$ similarity matrix, whose elements correspond to $D_{i,j}$ for all $i \in \{1,...,m\}$ and $j \in \{1,...,n\}$. However, in this particular situation we can apply Equation (4.7) recursively until $\mathbf{F}^{r,s}$ is reached, rather than propagating the recurrence back to $D_{0,0} = 0$.

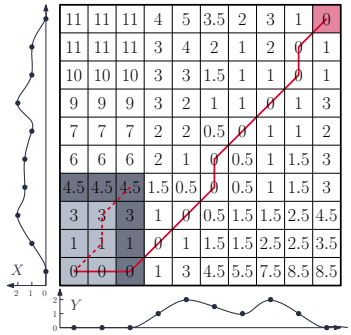


FIGURE 4.2: Incremental computation of the DTW between two generic time series, X^m and Y^n, with $m = n = 10$. The dark gray area depicts the frontier $\mathbf{F}^{4,3}$ over the 10×10 dimension measure matrix. The dashed lines illustrate the optimal path to $D_{4,3}$ and $D_{10,10}$.

An example of the incremental computation is depicted in Figure 4.2, where $D_{10,10}$ (red cell) is computed given $\mathbf{F}^{4,3}$ (dark gray cells). By storing the frontier, we avoid the computation of 4×3 intermediate results (i.e. the values shadowed in gray in the lower left corner of the matrix). In general, given the frontier $\mathbf{F}^{r,s}$, $r \times s$ computations are avoided. Hence, the computational complexity of the incremental computation is $O(m \cdot n - r \cdot s)$.

We next adapt the incremental computation described above for the batch and on-line pattern scenarios.

Batch Pattern Scenario

We first consider the batch pattern scenario illustrated in Figure 4.3, where the batch pattern X^m is fixed and stored permanently in memory, whereas the streaming sequence evolves from Y^s to Y^n. Since the length of the batch pattern is fixed, the given frontier corresponds to:

$$\mathbf{F}^{m,s} = \{D_{i,s}\}_{i=1}^{m}, \tag{4.9}$$

which is highlighted in blue in Figure 4.3.

The OESM computation for the batch pattern scenario consists of two main steps: (1) first $D_{m,n}$ is computed by using Equation (4.7) and $\mathbf{F}^{m,s}$; (2) then, $\mathbf{F}^{m,n}$ is stored as per Expression (4.9) so as to use it in future similarity measurements. This *compute-store* procedure is described as a pseudo-code in Algorithm 4.1, and graphically shown in Figure 4.3.

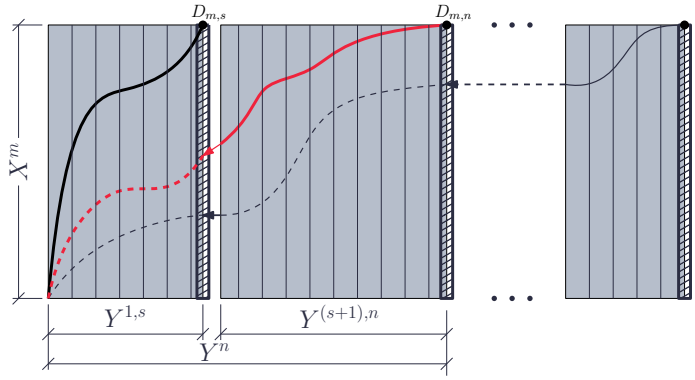


FIGURE 4.3: OESM computation procedure for a continuous flow of query data samples in the batch pattern scenario, where black and red curves illustrate the optimal paths to $D_{m,s}$ and $D_{m,n}$, respectively. Vertical solid lines represent the memory function, and the vertical shadowed gray areas denote the frontier.

Algorithm 4.1 OESM pseudo-code for the batch pattern scenario.

1: **Require:** $\mathbf{F}^{m,s}$, X^m , ρ.
2: **for** every newly arriving chunk $Y^{s,n}$ **do**
3: ▷ **Step 1**: *compute*
4: Compute $D_{m,n}$ based on $\mathbf{F}^{m,s}$ using Equation (4.7).
5: ▷ **Step 2**: *store*
6: Store frontier $\mathbf{F}^{m,n} = \{D_{i,n}\}_{i=1}^{m}$.
7: Set $s \leftarrow n$.
8: **end for**

At this point it is important to highlight that in the batch pattern scenario, the streaming time series is not required to be stored for incrementally computing the similarity measure. Consequently, the *memory* complexity of the batch pattern scenario is constant and equal to $O(m)$, thus meeting the requirements of the on-line learning algorithms. Regarding the running time, in this case the complexity is given by $O(m \cdot (n-s))$, where $n-s$ is the number of samples in the arriving chunk.

On-line Pattern Scenario

We proceed by considering the frontier as per Equation (4.8) and the recurrence in Expression (4.7). Since both X^m and Y^n grow in the on-line pattern scenario, they must be stored so as to compute $D_{m,n}$ incrementally. Specifically, we need past sequences $X^{1,r}$ and $Y^{1,s}$ to compute the $D_{i,j}$ elements in the measure matrix corresponding to $(i,j) \in [1,r] \times [s+1,n]$ and $(i,j) \in [r+1,m] \times [1,s]$ (i.e., $D_{i,j}$ entries highlighted in bold font in Figure 4.2). Consequently, the memory required to compute OESM in the on-line pattern scenario lacks an upper bound as the time series grow.

To overcome this issue, we embrace the use of Sakoe-Chiba constraints introduced in Section 1.4.2. Consequently, in the on-line pattern scenario the proposed OESM approach blends together the incremental computation procedure previously described in this section, and the Sakoe-Chiba strategy. Formally, let us consider the evolving scenario described in Figure 4.4, where ℓ is the Sakoe-Chiba band width, and the blue area corresponds to the stored frontier given by:

$$\mathbf{F}^{r,s} = \{D_{i,s}\}_{i=\max\{1,r-\ell\}}^{r} \cup \{D_{r,j}\}_{j=\max\{1,s-\ell\}}^{s}. \tag{4.10}$$

As in the previous example, the computation procedure for the on-line pattern scenario is exemplified in Figure 4.4, and described in Algorithm 4.2. Line 4 corresponds to the computation step, and lines 6 and 7 to the storage step. Accordingly, first $D_{m,n}$ is computed by applying the recurrence in (4.7), until the frontier is reached. Due to Sakoe-Chiba band constraints, we assume $D_{i,j} = \infty$ for those $i \in \{\max\{1, r-\ell\}, ..., m\}$ and $j \in \{\max\{1, s-\ell\}, ..., n\}$ not satisfying $|i-j| \leq \ell$. Then, according to the Definition in (4.10), the frontier $\mathbf{F}^{m,n}$ is stored. In addition, the last ℓ points of each time series must also be saved in this scenario, i.e., we would need to store $X^{\max\{1,n-\ell\},m}$ and $Y^{\max\{1,m-\ell\},n}$ subsequences for future OESM computations.

Once the length of the time series (m and n) is greater than ℓ, due to the imposed constraints the on-line pattern computation procedure leads to a limited space complexity of $O(\ell)$. Likewise, the constraints reduce the running time from quadratic to linear with chunk size, $O(\ell \cdot \max\{m-r, n-s\})$. According to Figure 4.4, the time series must grow similarly for OESM to be a consistent computation procedure. When one time series grows significantly faster than the other, eventually $|m-n| > \ell$. In this case, we store points of X^m and Y^n that might never be used in the computation of the similarity measure. Therefore, we should take into account the rate at which time series grow for a proper adaptation of the memory function

Algorithm 4.2 OESM pseudo-code for the on-line pattern scenario.

Require: $\mathbf{F}^{r,s}$, $X^{r-\ell,s}$, $Y^{s-\ell,s}$, ℓ, ρ.
for newly arriving chunk $Y^{s,n}$ and $X^{r,m}$ **do**
  ▷ **Step 1**: *compute*
  Compute $D_{m,n}$ from $\mathbf{F}^{r,s}$ using Equation (4.7).
  ▷ **Step 2**: *store*
  Store frontier $\mathbf{F}^{m,n}$ using Equation (4.10)
  Set $s \leftarrow n$ and $r \leftarrow m$.
end for

and the Sakoe-Chiba constraints. Hence, in the on-line pattern scenario, we assume that time series evolve at a speed such that arriving chunks satisfy $|m - n| \leq \ell$.

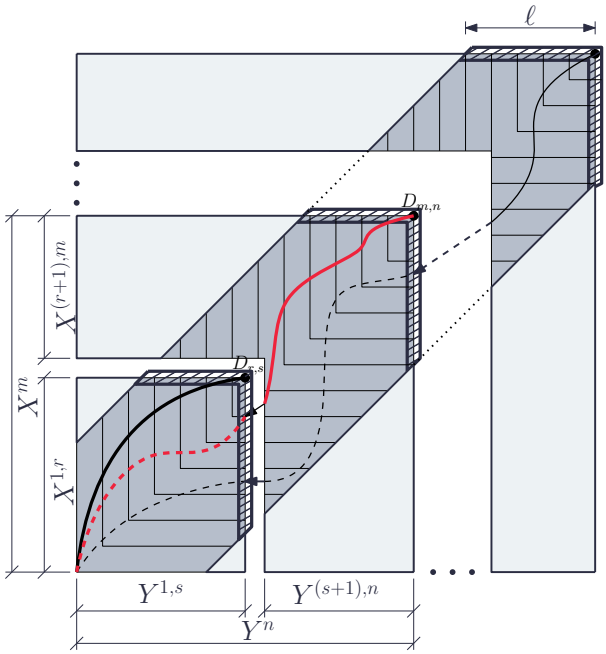


FIGURE 4.4: In the on-line pattern scenario, OESM computation procedure for a continuous flow of on-line pattern and query data samples. The black and red curves illustrate the optimal paths to $D_{m,s}$ and $D_{m,n}$.The shadowed dark gray areas correspond to the frontier. According to Equation (4.3), black solid lines over light gray areas show equidistant (i, j) points.

4.3 Experimental Setup

In this section we describe the experiments used to analyze, in terms of efficiency and the forgetting mechanism, the proposed OESM in both batch and on-line pattern scenarios. The Python 2.7 source code of OESM is

available at `http://bitbucket.org/izaskun_oregui/ESM`, and can be used to solve both pattern scenarios. The cost functions implemented in this repository correspond to DTW, EDR, ERP and EDIT distances.

4.3.1 Efficiency

In order to shed light on the efficiency of OESM, we compare its computation time in the batch and on-line pattern scenarios with the computation time required by the conventional ESM with the Sakoe-Chiba constraints. In this context, two sets of experiments are proposed. Considering a constant size of arriving chunks, the first set of experiments aims at analyzing the efficiency over time, i.e., we compare the running time as the length of the time series grows. In the second set of experiments, we consider arriving chunks of increasing size. The objective is to analyze the running time required to compute the similarity, assuming that the initial length of the time series is kept fixed.

To carry out these experiments, we compute the measure of similarity between two randomly generated streaming time series whose points are drawn from a uniform distribution with support $[0, 1]$. Furthermore, we establish the following experimental conditions:

- **Experiment 1:** we set the length of the arriving chunks to $b = 1$. Either in the on-line pattern scheme (OESM) or for the conventional ESM study, the streaming time series evolve from an initial size of 3 points to a final length of 70 points. In both cases the width of Sakoe-Chiba band has been set to 35. Regarding the batch pattern scheme (OESM), the length of the streaming time series ranges from 3 to 70, while the length of the batch time series is fixed at 35.

- **Experiment 2:** in this second experiment we analyze the running time of the OESMs as a function of the length of the arriving chunks. For both on-line or off-line settings, we compute the similarity measure for chunk size $b \in [1, 50]$ assuming that the length of the stored time series is $m = n = 35$. As in the previous experiment, the on-line pattern OESM and the conventional ESM implement Sakoe-Chiba constraints, where the band length ℓ is set equal to 35.

As we show in Section 1.4.3, the difference among ESM variants resides in the definition of the cost function. For this set of experiments, we assume that the time needed to compute the cost functions listed in Table 1.1 is similar. As a consequence, there is no need for replicating the experiments with all EMS variants. For these experiments, the considered cost function corresponds to the Euclidean distance, i.e., the results we show further correspond to those issued from the conventional DTW and its on-line adaptation.

4.3.2 Forgetting Mechanism

In this section we present the experimentation used to test the behavior of the proposed forgetting mechanism when dealing with non-stationary

streaming time series. The conducted experimentation aims to evaluate whether it allows OESMs to neglect the past and accommodate to incoming stationary intervals. We have decided to evaluate the forgetting mechanism by means of a supervised classification problem, which allows quantifying the behavior objectively, in terms of the accuracy. For this purpose we use a NN classifier, one of the simplest and most widely utilized distance-based approaches.

The experiments are carried out with specially designed non-stationary time series streams. As we subsequently explain in more detail, these streams are created by concatenating sequences contained in the Time Series Classification Archive [92] databases, which are assumed to represent different *events* of the streaming time series.

Generation of Non-stationary Streaming Time Series

The generation process uses time series contained in the UCR archive as a substrate from which to compose the streaming time series for our experimental benchmark. This repository collects real and synthetic databases, each consisting of multiple time series, each having one class label assigned from a set of possible class labels. In this sense, let $\mathcal{U} = \{(U_k, l_k)\}_{k=1}^{K}$ denote a UCR dataset, where $l_k \in \{1, ..., L\}$ represents the label of the $U_k = (u_1^k, ..., u_v^k)$ time series; L is the total number of class labels; and K is the total number of time series in the dataset. By drawing time series randomly from $\mathcal{U}$ and concatenating them, we construct a streaming time series following a predefined duration of *stationary* intervals. By *stationary* we mean that the label assigned to each event of a given interval remains the same during the duration of the interval. As a result, this duration establishes the position where label transitions occur. The steps involved in the creation of streaming time series can be summarized as follows:

1. First, P time series sharing the same class label are selected uniformly at random from the chosen dataset, which are then z-normalized and concatenated in order to model a stationary interval. In fact, data normalization has been widely acknowledged as a major open issue in stream data mining. Nevertheless, we assume that non-stationary streaming time series are composed of normalized events, so that the performance of the proposed adaptation becomes unbiased with respect to this aspect, and hence focuses strictly on the OESM adaptability. Parameter P denotes the length (periodicity) of the stationary interval, i.e., the number of time series of a given class concatenated before a label change is held.

2. Stationary intervals are assembled together (one after another), in such a way that two consecutive stationary intervals do not share the same class label. That is, stationary intervals are concatenated by forcing a transition at each union. We refer as T to the total number of transitions in a streaming time series.

An example showing the generation process of a streaming time series from a generic binary-class dataset $\mathcal{U}$ is depicted in Figure 4.5. In this example,

the parameters of the streaming time series are $T = 4$ transitions and $P = 2$ time series per stationary interval (periodicity).

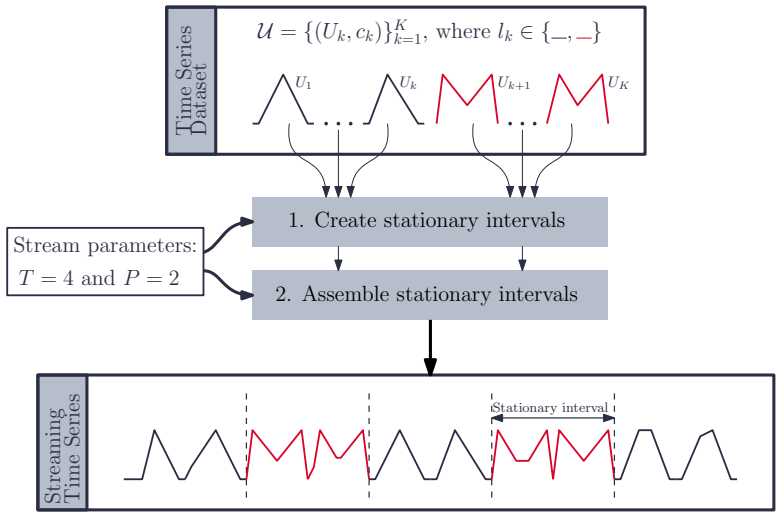


FIGURE 4.5: Stream time series generation process. The box at the top shows a binary-class database $\mathcal{U}$, where l_k (red or blue) denotes the ground truth label of time series U_k. Given the number of transitions $T = 4$ and the periodicity $P = 2$, the box at the bottom illustrates the resulting streaming time series, where the dashed vertical lines indicates the transition point.

TABLE 4.1: Main characteristics of the utilized UCR datasets.

Database ($\mathcal{U}$)	Time series length (v)	Number of classes (C)	Sakoe-Chiba band (ℓ)
`Gun Point`	150	2	0
`Two Lead ECG`	82	2	4
`Wafer`	152	2	4
`Synthetic Control`	60	6	4
`Two Patterns`	128	4	5
`Plane`	144	7	9
`CBF`	128	3	14
`Faces UCR`	131	14	16
`Symbols`	398	6	32

For each batch and on-line pattern scenario and UCR dataset in Table 4.1, $N_{\text{CP}} = 50$ different on-line classification problems have been created. For each dataset $\mathcal{U}$, these classification problems consist of a query stream and a set of reference (streaming) time series defined as:

- **Batch pattern scenario:** the query stream of the batch pattern scenario is a streaming time series with $T = 14$ transitions and a periodicity of $P = 1$. The batch pattern set (i.e., the reference time series set) is just a subset of $\mathcal{U}$, where all class labels are equally represented, each with 10 randomly selected time series.

- **On-line pattern scenario:** the query stream of the on-line pattern scenario consists of $T = 9$ transitions with a periodicity of $P = 3$. In this case, the on-line pattern set (i.e., the reference stream set) consists of $10 \cdot C$ stationary streams (10 per class label), each created by concatenating 30 time series with equal labels (that is, with $T = 0$ and $P = 30$).

For the sake of fairness, the reference and query streams do not share any original time series, i.e., in the classification problem design, the random sampling is made without replacement.

Gold-standard Model

For comparison purposes we consider, as gold-standard, a classifier based on the ESM that uses the query stream information (total number of transitions T and the periodicity P) to reset the measure of similarity and therefore, to adapt to the incoming stationary interval. By doing so, the gold-standard classifier can be regarded as a perfect model, as i) it forgets old stationary intervals immediately; and ii) it adapts to new intervals by taking into account all query stream points information.

Memory Parameter

The selection of an appropriate value for the memory parameter is not trivial, as it roughly depends on the problem at hand. In our case we will set the memory parameter considering the length v of the time series belonging to every UCR dataset $\mathcal{U}$ (second column in Table 4.1). The considered memory parameters are given by:

$$\rho \in \{0.0001^{1/v}, 0.1^{1/v}, 0.5^{1/v}, 1^{1/v}\}. \tag{4.11}$$

In words, by using these values the contribution of the time series point v steps prior to the sample at hand is weighted by $\{0.0001, 0.1, 0.5, 1\}$. We will hereafter refer to the selected values as short ($\rho = 0.0001^{1/v}$), middle ($\rho = 0.1^{1/v}$), large ($\rho = 0.5^{1/v}$), and full ($\rho = 1$) range memory. Since the OESM is completely equivalent to the ESM when $\rho = 1$, full range memory OESM and ESM will be used indistinctly.

4.4 Results and Discussion

In this section we present and comment on the experimental results.

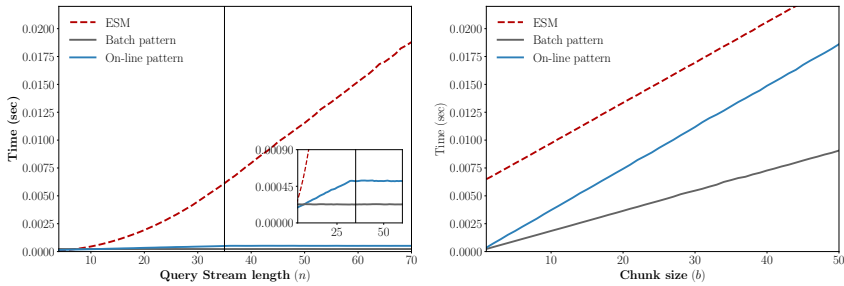
4.4.1 Computational Efficiency

Results for the running time are shown in Figure 4.6. The plot on the left illustrates the results for Experiment 1, whereas the plot on the right shows the results for Experiment 2 (see Section 4.3.1). After 100 independent runs of each experiment, the plots show the average computational time (in seconds) required by each scenario using a Python 2.7 naive code executed on a single i7 core at 3.10 GHz: the conventional ESM with Sakoe-Chiba

band (red dashed), the on-line pattern (blue) and the batch pattern (gray). In Figure 4.6a, the vertical solid line indicates the point where the Sakoe-Chiba band is exceeded.

- **Experiment 1:** as can be observed in the plot, the running time required by the conventional ESM (i.e., the ESM computed as per Equation (1.19)) grows in a quadratic manner when the query stream length n is lower than or equal to the Sakoe-Chiba band width ℓ, and linearly once $n > \ell$ (see red dashed curve). Regarding the running time of the on-line pattern scenario, it grows linearly when $n \leq \ell$. However, once $n > \ell$, the complexity remains constant (blue solid line). For the batch pattern scenario, the computational time is constant for any length of the time series (gray solid line). Observe that once n exceeds the Sakoe-Chiba band, the running time required by the on-line pattern is twice the running time for the batch pattern scenario (gray versus blue solid lines). This is just the consequence of imposing $\ell = m = 35$. When a new sample arrives, the number of elements to be computed in the distance matrix is equal to the number of elements in the frontier, namely, 35 in the batch pattern scheme and 69 in the on-line pattern.

- **Experiment 2:** due to the Sakoe-Chiba band constraints, the running times of both the conventional ESM and the on-line pattern grow linearly. Moreover, the curves are nearly parallel (compare red dashed and blue solid lines). The required time also increases linearly for pattern scenario (gray solid line). The selected time series length and the Sakoe-Chiba band width make the computational complexity $O(b)$ for on-line and batch pattern scenarios. However, we note that the slopes are quite different (blue and gray solid lines).



(A) Experiment 1 (B) Experiment 2

FIGURE 4.6: Average running times of conventional ESM and the proposed OESM under different parameters. The number of independent runs is 100.

4.4.2 Reaction Capacity and Predictive Performance

Given the $N_{CP} = 50$ streaming time series built from each UCR dataset in Table 4.1, and the values for the memory parameter ρ in Equation (4.11), we now discuss, for both batch and on-line pattern scenarios, the results

delivered by the OESM-based and gold-standard classifiers. To this end, we report on the average accuracy and the average ranking defined in the following section.

Evaluation Method

In order to numerically quantify the accuracy of each classifier either in the batch or in the on-line pattern scenario, we consider that a classifier correctly predicts the class label of an arriving query point if the predicted label coincides with the label of the stationary interval it belongs to. Let $\mathcal{U}_\kappa$ denote the κ-th generated streaming time series built from the UCR database $\mathcal{U}$. For the i-th arriving query stream point (recall that the size of the arriving chunk is $b = 1$), the predictive accuracy is computed as:

$$\mathrm{ACC}_{\mathcal{U}_\kappa}(i;\rho) = \begin{cases} 1 & \text{if} \quad \mathrm{PL}(i) = \mathrm{TL}(i), \\ 0 & \text{otherwise} \end{cases} \tag{4.12}$$

where ρ is the memory parameter, and $\mathrm{PL}(i)$ and $\mathrm{TL}(i)$ are, for the arriving point i, the predicted label and true label, respectively. Note that, $\mathrm{TL}(i)$ is the same for all i belonging to the same stationary interval (event).

As already known, the accuracy of the 1-NN classifier strongly depends on the selected reference and query streams. To account for this statistical variability, the average accuracy $\overline{\mathrm{ACC}}_{\mathcal{U}}(i;\rho)$ is computed over the N_{CP} streaming time series built on $\mathcal{U}$.

Given $\mathcal{U}$ and a set of memory parameters, suppose we compute the average accuracy for each ρ in the set. In order to extract an overall view of the influence of the memory parameter ρ across all classification problems, we have analyzed the experimental results right after a label transition (i.e., at the *beginning* of a stationary interval), in the *middle* or at the *end* of the stationary interval. To do so, we apply the following process to each part (*beginning*, *middle*, *end*):

1. For each i in the corresponding part, and given memory parameter ρ, we compute the ranking produced by the average accuracy.

2. We compute the average ranking for the average accuracies obtained in the previous step. In this context, we will use $\bar{\mathrm{r}}_1(\mathcal{U},\rho)$, $\bar{\mathrm{r}}_2(\mathcal{U},\rho)$ and $\bar{\mathrm{r}}_3(\mathcal{U},\rho)$ to denote the average ranking in the *beginning*, *middle* and *end* parts, respectively.

For the sake of brevity, we only show a representative sample of the experiment results in this section. The results issued by the remaining experiments are available in `http://bitbucket.org/izaskun_oregui/ESM`.

Results: Batch Pattern Scenario

The outcomes for the batch pattern scenario are shown in Figure 4.7. Considering all values of the memory parameter ρ and the classification problems built on `Faces UCR` dataset, Figure 4.7a illustrates the average accuracy, $\overline{\mathrm{ACC}}_{\mathcal{U}}(i;\rho)$, over the last 2 simulated stationary intervals. As

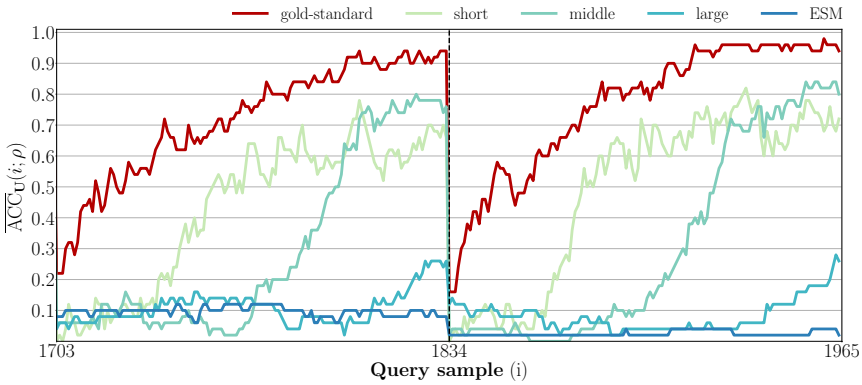
can be observed in this plot, the results drawn from the gold-standard classifier are the best throughout the intervals (red curve). Moreover, scores of this model show a continuous progress that starts from an initial value of 0.2 to 0.95 in both intervals. On the contrary, the results issued by the ESM classifier do not improve over time. In fact, the obtained scores are those of random guessing. Likewise, the results from the long-range memory OESM-based classifier are those of random guessing (expect at the end of the stationary intervals where results slightly improve). As opposed to the gold-standard model, where the measure of similarity is reset immediately after the transition between events, long-range and ESM models are highly influenced by past alignments. As a result, these latter classifiers are unable to adapt to such transitions. A balance between the steady progress of the gold-standard model and the impasse of the long-range and ESM models is observed for short- and middle-range memory models. In these cases, the smaller the memory parameter value is, the sooner the accuracy increases due to its capacity to forget. However, in certain cases this involves a penalty in score once we go ahead on the stationary interval. Indeed, this noted behavior is consistent with the stability-plasticity dilemma formulated in the context of incremental learning over non-stationary streams. This dilemma states that there is a trade-off between the learning capability of a predictive model and its flexibility and speed to capture evolving concepts along the stream [93]. Finally, we note that the curves tend to be noisier as ρ decreases. This is in part due to the influence of the high difference between past and present alignments in the OESM.

In order to elucidate whether the observed evolution of $\overline{\text{ACC}}_{\mathcal{U}}(i;\rho)$ can be extrapolated to all datasets $\mathcal{U}$ in the benchmark, Figure 4.7b shows the empirical average ranking distribution across all $\mathcal{U}$ in Table 4.1. Three plots are included to focus on the (*beginning*,*middle*,*end*) parts of the stationary intervals. In close accordance with what was expected in Section 4.3.2, the gold-standard results to be the best positioned model in all partitions (red violin plot). For the OESM-based models, the ranking varies as we progress on the stationary interval as follows:

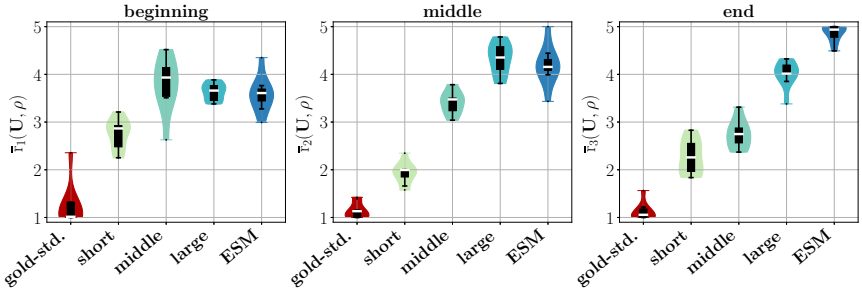
- ***Beginning*:** immediately after every stream transition, scores from the OESM-based classifiers are those from random guessing. In this case the short-range memory model is slightly better than the rest of the OESM-based classifiers (middle-range, long-range and ESM). This is directly related to the fact that short-range memory model reacts sooner than the rest.

- ***Middle*:** a considerable improvement is observed for the short-range memory model and, to a lesser extend, for the middle-range memory classifier. Again, this behavior is in line with the aforementioned idea: the smaller ρ is, the sooner the accuracy score recovers after the event change.

- ***End*:** note from Figure 4.7a that while the average accuracy issued from the short-range memory model is stabilized, the middle-range memory model accuracy continues to improve. Consequently, the middle-range



(A) Accuracy score



(B) Average ranking distribution

FIGURE 4.7: Performance of the NN classifier for the batch pattern scenario and different values of ρ: **(a)** Accuracy score over the last 2 simulated stationary intervals of the classification problems (streaming time series) built on the `Faces UCR` dataset; **(b)** Average ranking distribution from the last 5 simulated stationary intervals considering all databases in Table 4.1. The violin plots represent for each partition (*beginning*,*middle*,*end*), the average ranking distribution. The horizontal white line denotes the median from all considered datasets.

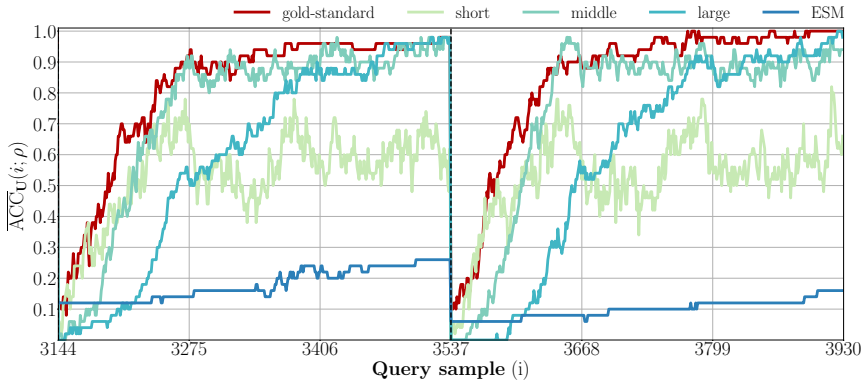
memory model is, in terms of average ranking, the model improving most in this partition.

The average ranking distribution for the long-range and ESM classifiers are steadily worse over different partitions. This is an expected result since the scores drawn from these models do not improve over the interval.

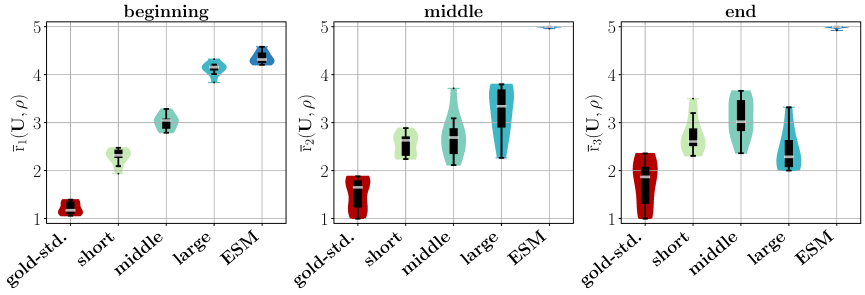
Results: On-line Pattern Scenario

The results for the on-line pattern scenario are summarized in Figure 4.8. Given the streaming time series built on `Faces UCR` database, Figure 4.8a illustrates the average accuracy issued from the gold-standard and OESM-based classifiers. As observed in this plot, the accuracy score is very similar to the behavior described for the batch pattern scenario, where models with smaller values of the memory parameter ρ react sooner to transitions than those with larger values. Moreover, the penalty in the scores of

most forgetful models is also more evident than in the previous scenario, as elicited by results issued by the short-range memory model. Again, since the ESM model is fully influenced by the past events, the results drawn from this model are still the same as those from random guessing. However, the long-range memory model is able to accommodate stream transitions. This is due to the periodicity increase, i.e., in contrast to the batch pattern scenario where $P = 1$, in the on-line pattern scenario $P = 3$, hence, the long-range memory model *has more time* to forget past events and to adapt to the prevailing stationary interval.



(A) Accuracy score



(B) Average ranking distribution

FIGURE 4.8: Predictive performance of the 1-NN classifier for proposed ρ values in the on-line pattern scenario: **(a)** Given the classification problem built `Faces UCR` database, average accuracy over the last 2 simulated stationary intervals; **(b)** Considering all datasets in Table 4.1 and the last 5 simulated stationary intervals, the violin plots in the figure represent, for each part (*beginning*, *middle*, *end*), the average ranking distribution, where the horizontal white line denotes the median.

Similarly to the discussion on the previous scenario, the results considering all the datasets in Table 4.1 are summarized in Figure 4.8b. In this case, the analysis of each partition is outlined below:

- ***Beginning*:** as in the batch pattern scenario, the gold-standard model is the best at the beginning of the stationary interval. In contrast, the OESM models (short-range, middle-range, long-range and ESM models)

do not perform similarly any longer. In this case, the average rank gets worse as the memory parameter value increases. As the length of the partition increases, OESM models have more time to react and, consequently, to adapt to streams transitions, being those models with small ρ values the quickest to react.

- ***Middle*:** again, gold-standard is the best performing model. Regarding the OESM-based classifiers, middle- and long-range models are those improving the most. This occurs because (i) their flexibility to adapt to changes is slower, and (ii) the score penalty is lower for those models with higher ρ values. Finally, the ESM is the worst model for all the considered datasets.

- ***End*:** in this case, the average ranking median is 2 for the conventional gold-standard model, which means that, in some datasets, results drawn from the OESM can be as accurate as that of the conventional model. As mentioned previously, the long-range memory model is able to accommodate stationary interval with lower penalty (in the score) than models with smaller ρ values. As a result, the improvement of the long-range memory model is noteworthy. The ESM, once again, is the model scoring worst given its lack of adaptability to changes between events.

4.5 Conclusions

This chapter has elaborated on the On-line Elastic Similarity Measure (OESM), namely, an adaptation of conventional ESMs suited to measure the similarity between time series in two different streaming scenarios: on-line pattern and batch pattern. In either case, the proposed OESM computes the similarity incrementally, and incorporates a novel forgetting mechanism to adapt to changes in the streaming time series. We have shown that the proposed adaptation is conceptually simple, that it can be computed using limited computational resources, and that it is able to neglect the past so as to focus on newly arriving points over time. Regarding this last issue, the results drawn from NN classifiers have shown that the behavior of the proposed forgetting mechanisms is in line with the stability-plasticity dilemma identified for incremental learning models over non-stationary data streams [93].

For these reasons, the proposed OESM spans a manifold of research lines in the pattern recognition field. In particular, it accommodates a direct adaptation of distance-based methodologies to the on-line setting. In other words, OESM can be adopted by any learning algorithm for evolving scenarios hinging on distances of similarity between time series, such as nearest-neighbor classifiers, kernel-based classifiers, partitional or hierarchical clustering. Moreover, the proposed formulation can accommodate other possible inner cost functions $c(x_i, y_j)$, besides those in DTW, EDR, EDIT and others alike.

With the aim of giving insights into the influence of the memory parameter ρ, the parameter value setting has been made based on the UCR

time series length. As evidenced by the discussed results, different ρ values yield different behaviors in the prediction performance of the model. There stems one of the research lines planned for the future, specifically, the development of a strategy to select the best value of ρ given the characteristics of a given problem. Furthermore, the proposed OESM can be utilized for other supervised and unsupervised classification problems – such as early classification or clustering – where streaming time series are involved. Finally, normalization techniques suited for on-line environments will be investigated, so that the proposed OESM can be applied to streaming contexts without assuming any prior normalization.

Chapter 5

Learning from ODTW Features

In Chapter 4 we demonstrated that the proposed OESM, i.e., the adaptation of ESM to the on-line setting, has desirable properties for measuring the similarity among streaming time series. First, we showed that OESM can be updated in constant time with respect to the size of arriving data chunks. For this purpose, we computed the run time of the OESM under different conditions. Then, we provided evidence that the proposed OESM accommodates the streaming time series dynamics by virtue of its forgetting mechanism. To this end, we developed several OESM based NN classifiers to evaluate the adaptation of the OEMS objectively in terms of accuracy. Based on the results issued from this latter set of experiments, we empirically showed that OESM-NN classifiers are more reactive to structural changes of the streaming time series when the memory of the utilized OESM is small. On the contrary, since the OESM incorporates more information about its past as the memory parameter increases, we demonstrated that OESM-based classifiers tend to draw more accurate results at the cost of more time needed to adapt to new events.

This chapter builds upon these previous findings by developing an active adaptation strategy to improve both the accuracy and adaptability to changes in streaming time series classifiers. To this end, we capitalize on Observation 1 introduced in the previous chapter, which states that ESM computation requires the estimation of the similarity among all subsequences of the time series at hand. Specifically, we leverage the set of intermediate OESM computations – termed *streaming frames* – to develop an on-line pattern end detector (PED) that identifies the boundary between consecutive events in streaming time series. Then, we exploit the decisions made by the proposed PED to better adapt predictive algorithms (in this case, NN classifiers) to upcoming streaming time series events.

5.1 Introduction and Related Work

When analyzing dynamic scenarios that evolve over time, it is of utmost importance that predictive models adjust properly to changes in the

streams over time. Hence, in recent years there have been numerous proposals aimed at designing learning algorithms capable of performing efficiently in such evolving scenarios [6], [7]. Two main groups of algorithms can be distinguished depending on the mechanism utilized for adaptation [8]. On the one hand, passive (or blind) strategies are based on forgetting mechanisms such as the under-weighting procedure of the OESM, or sliding window methods to accommodate predictive models to received data [93]. On the contrary, active (or informed) adaptation strategies adjust predictive models to upcoming events based on ad-hoc methods devised to analyze upcoming data, towards determining whether the underlying process has changed [3], [9], [10].

On this basis, this chapter tackles the problem of developing an active adaptation strategy for on-line classifiers. Similarly to the change point detection problem, which aims at finding abrupt changes in time series, the goal of our devised method is to detect pattern realizations (*events*) in streaming time series, in order to leverage conventional off-line classifiers and adapt them to upcoming events. For this purpose, we endow classifiers with a procedure that tests incoming streams to detect such transitions among events. In this context, we use our proposed PED model. This procedure consists of a deep Convolutional Neural Network (CNN) whose trainable parameters are learned from streaming frames, i.e., from images that are made up of ODTW measurements between reference patterns and arriving observations. We evaluate the performance of the proposed PED over streaming time series classification problems defined over 10 benchmark problems. Specifically, we use NN classifiers for this task and compare its performance to that of the ODTW-NN and the gold-standard models introduced in Chapter 4.

The remainder of the chapter is organized as follows: Section 5.2 formulates the streaming classification problem handled in this chapter. Section 5.3 defines the PED model, whereas Section 5.4 presents and discusses the results drawn from the conducted experimentation. Finally, Section 5.5 summarizes the main contributions of the chapter and outlines future research directions departing from this work.

5.2 Streaming Time Series Classification

To develop the PED model, we assume the batch-pattern scenario introduced in Section 4.3.2. This scenario undertakes the problem of classifying every arriving data chunk that comprises the streaming time series based on a set of stored patterns. Formally, given the database:

$$\mathcal{B} = \{(X_k, l_k)\}_{k=1}^{K}, \tag{5.1}$$

where $l_k \in \{1, \ldots, L\}$ is the category of the reference pattern (event) $X_k = (x_1^k, \ldots, x_i^k, \ldots, x_m^k)$, the streaming time series classification (STSC) problem consists of classifying the streaming time series defined as the sequence of events:

$$\langle Y_1, \ldots Y_q, \ldots, Y_\beta \rangle, \tag{5.2}$$

where sub-sequence $Y_\beta \in \mathbb{R}^{n_\beta}$ represents the β-th event which, in turn, has an associated label in $\{1, \ldots, L\}$. Following the notation in Chapter 1, a streaming time series can be also defined as:

$$Y^n = \left(y_1, \ldots, y_j, \ldots, y_n\right), \tag{5.3}$$

where n is the total number of data points received so far, i.e., $n = \sum_{q=1}^{\beta} n_q$; and y_j represents an observation which fall within one of the events in Equation (5.2). Therefore, each $y_j \in Y^n$ has an associated class label among $\{1, \ldots, L\}$, which is to be predicted by a classifier. At this point, it is important to recall that Chapter 4 elaborated on an incremental ESM (namely, OESM) that enabled similarity-based classifiers to leverage the elastic properties of such measures when addressing learning problems defined on streaming time series.

As stated in the introduction, we herein address the STSC problem by developing an active strategy that analyzes the OESM measure to adapt distance-based classifiers to transitions between events of the streaming time series. To this end, the proposed PED method examines measurements of the ODTW, namely, features extracted from the measure matrix:

$$\mathbf{M} = \begin{pmatrix} D_{1,1} & D_{1,2} & \ldots & D_{1,n} \\ D_{2,1} & D_{2,2} & \ldots & D_{2,n} \\ \vdots & \vdots & & \vdots \\ D_{m,1} & D_{m,2} & \ldots & D_{m,n} \end{pmatrix} = \left(D_{i,j}\right) \in \mathbb{R}^{m \times n}, \tag{5.4}$$

where each entry is given by the ODTW similarity:

$$D_{m,n} = ||x_m - y_n||_2 + \min\left\{\rho D_{m-1,n-1}, D_{m-1,n}, \rho D_{m,n-1}\right\} \tag{5.5}$$

computed between Y^n and one of the reference patterns in $\mathcal{B}$ for the batch-pattern scenario. It is worth to emphasize that the measure matrix in (5.4) results from Observation 1 introduced in the previous chapters, which states that the computation of the OESM through the recurrence (4.7) needs intermediate results among all X^i $(i = 1, \ldots, n)$ and Y^j $(j = 1, \ldots, m)$ sub-sequences.

Throughout Chapter 4 we displayed the measure matrix several times to introduce different concepts related to the OESM. Specifically, we used it to represent the forgetting mechanism (Figure 4.1), depict the frontier (Figure 4.2), as well as to illustrate the OESM computation procedure (Figures 4.3 and 4.4). As a matter of fact, the measure matrix contains useful information that can be analyzed to discover knowledge from data [90]. This is indeed the approach of this chapter: to take advantage of the information collected in this matrix over time to detect transitions between streaming events.

In order to provide evidence on the rich structure of $\mathbf{M}$, Figure 5.1 shows, for different memory parameter values, examples of the ODTW measure matrix between a generic reference pattern (black sequence on the left) and an evolving streaming series composed by 7 events (black sequence on the top). The $\mathbf{M}$ matrix corresponding to the gold-standard

model is depicted in the first row of this figure. As in the previous chapter, in this case we assume that the gold-standard represents a scenario with access to inner information of the streaming time series, namely, to event changes. Hence, in the gold-standard scenario, we assume that the similarity (in this case the DTW) is automatically tailored to the dynamics of the streaming series by initializing it at every transition between consecutive events. The resulting matrix $\mathbf{M}$ shows a periodic behavior, where smaller similarities are located near the optimal path (red curve), and where $D_{i,j} \in \mathbf{M}$ increases as we move away from it. Within the boundary between consecutive events, both the measure matrix and the optimal path undergo an abrupt change resulting from the DTW initialization. Indeed, it is worth noting that such a change illustrates coherent alignments between the reference series and each of the patterns in the streaming series.

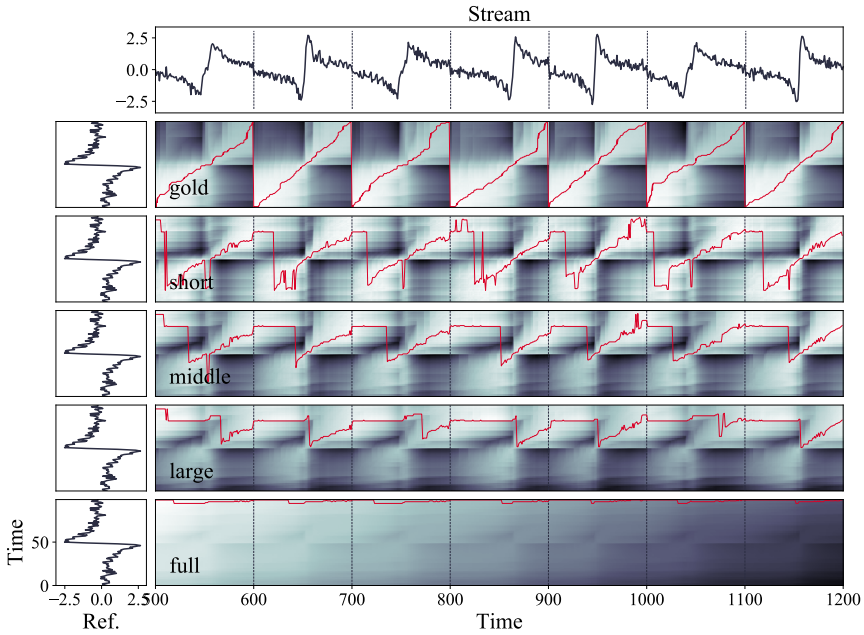


FIGURE 5.1: ODTW measure matrix for gold-standard, short, middle, large and full range memory ODTW scenarios. In each plot, dark colors represent high $D_{i,j}$ values, whereas light colors correspond to small distance measurements. Vertical dashed lines represent the boundaries (or transitions) between consecutive patterns (stationary intervals). In the gold-standard scenario, the red curve illustrates the optimal path p^*, computed separately within each event. In the remaining figures, the red curve depicts $p_{\min}$ (i.e. the position of the minimum ODTW measurement at each time stamp), which can be interpreted as the optimal path of the ODTW when configured with the corresponding memory parameter (short, middle, range, full).

Similarly to the set of experiments conducted in Chapter 4, plots from row 2 to 5 in Figure 5.1 display measure matrix $\mathbf{M}$ for short ($\rho = 0.0001^{1/\nu}$ with $\nu = 100$), middle ($\rho = 0.01^{1/\nu}$), large ($\rho = 0.1^{1/\nu}$), and full ($\rho = 1.00$,

which is equivalent to compute DTW) memory ranges. In contrast to the gold-standard framework, in these cases stream transitions are unknown, so the ODTW cannot be initialized. However, the initialization can be overridden by making ODTW focus on recent observations, i.e., by using a small memory range through the choice of the ρ value. By doing so, $\mathbf{M}$ is more reactive to local changes, thus providing more detailed insights about the configuration of the streaming time series.

Indeed, if we inspect closely the image maps of the measure matrix, we can see that the structure of $\mathbf{M}$ degrades with respect to the gold-standard (first row) more severely as ρ increases. Accordingly, $p_{\min}$ (red curve depicting the position of the minimum ODTW measurement over time) features an increasing delay with ρ (compare red curves around pattern limits), meaning that the ODTW computed with large/full range memory need more time to accommodate newly arriving events. For instance, in the full-range memory example (a scenario in which ODTW does not forget anything from the past because $\rho = 1$), $p_{\min}$ consists of *matching* the last data points in the reference pattern with every point in the stream. Similarly, in the large-range memory example, $p_{\min}$ shows that ODTW requires at least half of the interval to forget the past and adapt to new stream events (note that $p_{\min}$ is almost a diagonal in the second half of every event). In contrast, when using short- (and, to some extend, middle-) memory ranges, the adaptation of the ODTW is achieved in the first half of the period, thus showing that the reference series aligns with the stream events properly.

Going back to the computation/storage procedure introduced in Section 4.2.2, we showed that the OESM can be efficiently updated by recording a small set of intermediate OESM measurements, i.e., by repeatedly saving selected elements of the measure matrix over time. Hence, when new data points arrive, we do not compute the whole measure matrix, but the sub-matrix corresponding to the new measurements. Bearing this observation in mind, let us assume that we received the following chunk:

$$Y^{n+1-\varpi,n} = (y_{n+1-\varpi}, \ldots, y_n)\,, \tag{5.6}$$

which contains ϖ observations. Then, in the batch-pattern scenario, the similarity is updated by computing the sub-matrix:

$$\mathbf{M}^{n+1-\varpi,n} = \begin{pmatrix} D_{1,n+1-\varpi} & \cdots & D_{1,n} \\ D_{2,n+1-\varpi} & \cdots & D_{2,n} \\ \vdots & \ddots & \vdots \\ D_{m,n+1-\varpi} & \cdots & D_{m,n} \end{pmatrix}, \tag{5.7}$$

which we referred to as the streaming frame. In words, the measure matrix (5.4) can be built in an on-line fashion, concatenating $\mathbf{M}^{n+1-\varpi,n}$ whenever new data points arrive, just as frames in a video stream.

Our active adaptation strategy embraces the above observations by relying on streaming frames rather than on single values of the similarity $D^{m,n}$. We analyze $\mathbf{M}^{n+1-\varpi,n}$ to determine whether the pattern associated

with an event has finished at the n-th timestamp. That is, the PED model developed in this chapter and explained in the next section capitalizes on the underlying structure of the measure matrix (as in Figure 5.1) as reflected by the streaming frames evolving over time.

For a better understanding, in what follows we assume that the ODTW – and thus, the measure matrix – is computed by using short-to-middle memory parameter values. Consequently, when clear from the context, we will avoid any explicit references to the memory parameter. Furthermore, we will use $\mathbf{M}^n$, instead of $mathbf{M}^{n+1-\varpi,n}$, to denote the streaming frame computed for the last ϖ observed points $Y^{n+1-\varpi,n}$.

5.3 Pattern End Detection Model

This section describes the necessary steps to build the PED model which, as stated previously, comprises a binary CNN classifier learning from the ODTW streaming frames. Just as in conventional CNN architectures [94], the classifier utilized in this work is composed by two main blocks: i) a set of convolutional layers, which extract hierarchical features from raw data (in our case, ODTW streaming frames); and ii) a series of fully-connected neural layers, which map the extracted features to a binary label indicating whether a transition among events is occurring at the timestamp corresponding to the input streaming frame.

Let us consider the STSC problem comprising the reference pattern database $\mathcal{B}$ and the streaming time series Y^n defined in Section 5.2. In addition, we assume that Y^n is recorded and labeled beforehand, so we know the limits occurring in the stream. In this context, the PED learning procedure follows in order these steps:

1. **Computation of streaming frames:** we define a $(m \times n)$ image composed of K channels as:

$$\overline{\mathbf{M}} = [\mathbf{M}_1, \ldots, \mathbf{M}_k, \ldots, \mathbf{M}_K], \tag{5.8}$$

where $\mathbf{M}_k$ is the measure matrix resulting from the computation of the ODTW between Y^n and $X_k \in \mathcal{B}$ for a given value of the memory parameter ρ. Streaming frames are obtained by applying a fixed-size sliding window. Thus, moving a window of ϖ time units across the measure matrix (sample by sample along the stream time axis), we extract a series of $n - \varpi$ streaming frames given by:

$$\left\{\overline{\mathbf{M}}^{\varpi}, \ldots, \overline{\mathbf{M}}^{i}, \ldots, \overline{\mathbf{M}}^{n}\right\}, \tag{5.9}$$

where $\overline{\mathbf{M}}^i$ represents the i-th streaming frame, which collects ODTW similarity measurements between the reference patterns in $\mathcal{B}$ and the $Y^{i+1-\varpi,i}$ sub-sequence of the streaming time series. The black arrow and shaded squares plotted in Figure 5.2 illustrate the direction of the sliding window and some of the resulting streaming frames, respectively.

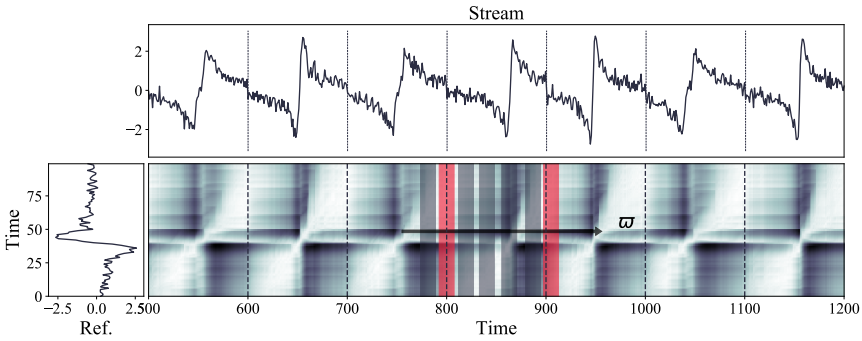


FIGURE 5.2: Illustration of the ODTW streaming frame extraction and labeling processes. As we move a window of ϖ time units along the measure matrix (indicated by the black arrow), we extract streaming frames. Some of these frames are depicted as red (*event change*) and gray (*no event change*) squares.

2. **Construction of the labeled streaming frame database $\mathcal{M}$:** the next step is to generate the streaming frame database, i.e. the assignment of labels to $\overline{\mathbf{M}}^{i+1-\varpi,i}$ samples for $i = \varpi, \ldots, n$. To this end, we split streaming frames into $\{0, 1\}$ classes, where frames falling within the 0 (1) category represent *no event change* (correspondingly, *event change*) in the stream. Accordingly, we generate the database of streaming frames as:

$$\mathcal{M} = \left\{ \left(\overline{\mathbf{M}}^{i+1-\varpi,i}, l_i \right) \right\}_{i=\varpi}^{n}, \tag{5.10}$$

where the category l_i of the i-th streaming frame is set to 0 (i.e., *no change*) if all observations in $Y^{i+1-\varpi,\varpi}$ belong to the same event; and 1 (i.e., *event change*) otherwise. In Figure 5.2, gray squares show frames categorized as 0, whereas frames labeled belonging to 1 category are depicted in red. Since the goal of PED is to detect limits between consecutive events, it is desirable that ϖ is small so that the procedure can detect event changes close to the instant where they truly occur.

Given a value of ϖ, it is important to note that the sliding window method can yield a highly imbalanced database $\mathcal{M}$. In particular, if we choose small window sizes, the ratio of streaming frames belonging to the *no event change* class is significantly higher than that of frames labeled as *event change*. It is widely acknowledged that unless properly counteracted, learning from imbalanced databases can lead to highly-biased models that do not generalize properly [95]. Hence, we apply a random under-sampling technique to decrease the ratio of *no event change* frames to achieve a balanced $\mathcal{M}$.

3. **Training of the binary CNN classifier:** we train the CNN classifier based on generated ODTW streaming frame database $\mathcal{M}$ (using conventional gradient-based backpropagation algorithms), yielding the proposed PED model that can detect transitions among events from new streaming frames.

5.4 Experimental Study

In this section, we empirically analyze the performance of the PED model and incidentally, the value of streaming frames as an information source. In particular, we evaluate:

1. The accuracy of the PED model when detecting event changes.

2. The contribution and efficiency of the PED when integrated with TSA procedures. That is, we analyze whether predictions issued by the PED model can be leveraged when addressing on-line TSA problems, such as the STSC task described in Section 5.2.

The first goal is undertaken by analyzing the precision and prediction delay of the PED model. On the other hand, in order to quantify the efficiency of our procedure for on-line TSA problems we devise a PED-based active adaptation strategy that adjusts conventional classifiers (in particular, DTW-NN classifiers) as per the evolution of the streaming time series under analysis. Then, the efficiency of PED is evaluated by comparing the classification accuracy of the DTW-NN model endowed with PED with the gold-standard (i.e., event changes known a priori) and the passive ODTW-based NN classifier proposed in Section 4.3. As has been already mentioned, the gold-standard model can be regarded as the holistic model for our PED-based classifier, as it uses query stream information to initialize the DTW and tailor the NN to new events. Hence, the more the accuracy issued from the DTW-NN model with PED approaches that of the gold-standard, the more efficient PED can be thought to be when adapting the DTW-NN classifier to event transitions. The motivation for using ODTW-NN in our experiments stems from the interest in comparing the performance of passive and active adaptation strategies in STSC problems.

A public repository has been made available at `https://git.code.tecnalia.com/Izaskun.oregui/2019_active-stream-classifier.git`, which includes data and Python code needed for reproducing all the experiments conducted in this chapter.

5.4.1 Description of the STSC Problems

Our STSC problems are built on 10 benchmark time series databases extracted from the `UCR Archive` [92] and the time series generation procedures provided in the supplementary material of [96]. The main features of these furnished databases are summarized in Table 5.1. From left to right, columns denote the name, length of the patterns, the total number of classes L, and the number of elements in each database. Among them, we have identified those synthetically generated sets with a symbol "$\diamondsuit$" which, as stated before, have been produced by following the procedure described in [96]. The parameters for the synthetic sequences have been set as 5 for the noise level, 10 for the shift level (except for `TWO PATTERNS` and `TWO PATTERNS MOD`, where the shift level is 5), and 10 for the warp

level. Furthermore, none of the generated databases include outlier patterns nor observations (check supplementary material in [96] for detailed information of each parameter). The rest of the databases in Table 5.1 correspond to those downloaded directly from the `UCR Archive`.

TABLE 5.1: Main features of benchmark databases.

Database ($\mathcal{U}$)	Pattern length (ν)	Num. of classes (L)	Num. patterns (K)
CBF◊	100	3	2223
TWO PATTERNS◊	100	4	2444
TWO PATTERNS MOD◊	100	2	2082
SYNTHETIC CONTROL◊	100	6	2946
RATIONAL◊	100	4	2444
FACES UCR	131	14	2205
GUN POINT	150	2	200
PLANE	144	7	210
TWO LEAD ECG	81	2	1162
WAFER	152	2	7164

Analogously to our experimental design in Chapter 4, for each database in Table 5.1 we develop a STSC problem which, we recall, consists of categorizing streaming time series based on a set of stored reference patterns (see Section 5.2). For this purpose, we split each database into three non-oversampling sets, namely, the reference pattern set, streaming time series for PED model training, and query streams respectively. Let $\mathcal{U} = \{(U_k, l_k)\}_{k=1}^{K}$ be a benchmark database with K examples, where $U_k = (u_1, \ldots, u_\nu)$ represents the k-th sequence and $l_k \in \{1, \ldots, L\}$ its class label. Given this notation, the STSC problem is given by:

- The set of reference patterns $\mathcal{B} \subset \mathcal{U}$. In this case, we assume that the database is composed of one reference pattern per class, hence $|\mathcal{B}| = L$.
- By drawing randomly and concatenating time series, we generate $\mathcal{D}_{\text{train}}$, namely, the set of training series used to fit the PED model. More precisely, training streaming series are given by:

$$\mathcal{D}_{\text{train}} = \left\{Y_k^{n_{\text{train}}}\right\}_{k=1}^{K_{\text{train}}}, \tag{5.11}$$

  where K_{train} represents the number of samples in the training set $\mathcal{D}_{\text{train}}$. Each stream $Y_k^{n_{train}}$ is composed by $\beta_{\text{train}} = 10 \cdot L$ events (i.e., 10 events per class), yielding a total of $n_{\text{train}} = 10 \cdot L \cdot \nu$ data points.
- Finally, the classification performance is assessed over a set of test streaming series, $\mathcal{D}_{\text{test}}$. Again, concatenating randomly selected reference series from a subset of $\mathcal{U}$, we generate $K_{\text{test}} = 12$ streams, each composed of $\beta_{\text{test}} = 20 \cdot L$ events (i.e., 20 events per class), and $n_{\text{test}} = 20 \cdot L \cdot \nu$ data points. Accordingly, the set of test streaming time series is defined as:

$$\mathcal{D}_{\text{test}} = \left\{Y_k^{n_{\text{test}}}\right\}_{k=1}^{K_{\text{test}}}, \tag{5.12}$$

  where $Y_k^{n_{\text{test}}}$ denotes the k-th test streaming time series.

Except for the transition and periodicity parameters, which are not considered in this chapter, the streaming time series generation process used

in this chapter is similar to that displayed in Figure 4.5. Table 5.2 shows a summarized description of the generated STSC problems.

TABLE 5.2: Summarized description of developed STSC problems.

Database ($\mathcal{U}$)	$\lvert\mathcal{B}\rvert$	$\mathcal{D}_{\text{train}}$		$\mathcal{D}_{\text{test}}$	
		K_{train}	β_{train}	K_{test}	β_{test}
CBF◊	3	50	30	12	60
TWO PATTERNS◊	4	37	40	12	80
TWO PATTERNS MOD◊	2	80	20	12	40
SYN. CONTROL◊	6	25	60	12	120
RATIONAL◊	4	37	40	12	80
FACES UCR	14	13	140	12	240
GUN POINT	2	80	20	12	40
PLANE	7	25	70	12	140
TWO LEAD ECG	2	80	20	12	40
WAFER	2	80	20	12	40

5.4.2 Predictive Performance of the PED model

We first analyze the accuracy of the PED model when detecting event changes in the streaming time series. To this end, we fit parameters of the CNN to each of the STSC databases. Then, we examine the PED model accuracy as per the precision and the distribution of the detection delay. In particular, the precision returns the proportion of correctly predicted changes among all predicted changes, i.e.:

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}, \tag{5.13}$$

thus giving reliable statistics for analyzing the predictive accuracy of the model. On the other hand, the delay measures the distance (in time steps) between the real event transition and its estimation by the PED model. We use the distribution of this difference to evaluate the eventual lag in the decisions issued by our developed active strategy.

Learning the PED Model

Given one of the classification problems listed in Table 5.2, the memory parameter ρ, and the size of the sliding window ϖ, we construct a PED model by following the steps described in Section 5.3. That is, we first compute the ODTW measure matrix for each streaming time series in $\mathcal{D}_{\text{train}}$. Subsequently, we apply the sliding window to extract streaming frames and label them to yield $\mathcal{M}$. Finally, we learn a binary CNN classifier from this generated database of supervised streaming frames. To this end, we split $\mathcal{M}$ into two non-overlapping subsets: training ($\mathcal{M}_{\text{train}}$) and validation ($\mathcal{M}_{\text{val}}$). The first set comprises 80% of the samples in $\mathcal{M}$, and is used to train the CNN. The second set $\mathcal{M}_{\text{val}}$ gathers the remaining 20%, and is utilized for validating the resulting model.

For each STSC problem, Tables 5.3a to 5.3d display the architecture of the CNN classifier and details on the memory and streaming frames'

TABLE 5.3: DNN training parameters and architecture for benchmark STSC problem. (A) Main characteristics of the DNN training databases. DNN architecture for (B) `CBF` (100, 100), `TWO PATTERNS` (50, 100), `TWO PATTERNS MOD` (50, 100), `SYNTHETIC CONTROL` (25, 200), `RATIONAL` (75, 100), `TWO LEAD ECG` (50, 100) and `GUN POINT` (50,100). Likewise, (C) displays DNN architecture for `PLANE` (20, 100) and `FACES UCR` (10, 500). Finally, in (D) the structure of DNN for `WAFER` (25, 100) database is reported. Utilized epochs and batch sizes are specified in brackets next to the name of the databases.

Database $\mathcal{U}$	**Memory** ρ		$\mathcal{M}_{\text{train}}$		$\mathcal{M}_{\text{val}}$	
	Range	Value	N. frames	N. streams	N. frames	N. streams
`CBF`◊	`middle` $0.1^{1/\nu}$	0.9772	18640	40	4660	10
`RATIONAL`◊	`middle` $0.1^{1/\nu}$	0.9772	18780	30	4382	7
`TP`◊	`middle` $0.1^{1/\nu}$	0.9772	18780	30	4382	7
`TP MOD`◊	`middle` $0.1^{1/\nu}$	0.9772	18360	60	6120	20
`TWO LEAD ECG`	`middle-short` $0.01^{1/\nu}$	0.9719	18360	60	6120	20
`SYN. CONTROL`◊	`middle` $0.1^{1/\nu}$	0.9772	18920	20	4730	5
`PLANE`	`middle-short` $0.01^{1/\nu}$	0.9685	22120	20	5530	5
`FACES UCR`	`middle` $0.1^{1/\nu}$	0.9826	22260	10	6678	3
`GUN POINT`	`middle` $0.1^{1/\nu}$	0.9848	18360	10	6120	10
`WAFER`	`middle` $0.1^{1/\nu}$	0.9850	18360	10	6120	10

(A)

Layer Type	**Description**
2D Convolutional	16 filters (3×3) stride (1, 1), *same*, ReLu
2D Convolutional	32 filters (1×1) stride (1, 1), *valid*, ReLu
2D Max Pooling	Pool size (2×2), *valid*
2D Convolutional	64 filters (1×1) stride (1, 1), *valid*, ReLu
2D Max Pooling	Pool size (2×2), *valid*
Linear	20 units
Dropout	0.3 rate
Sigmoid	2 units

(B)

Layer Type	**Description**
2D Convolutional	16 filters (3×3) stride (1, 1), *same*, ReLu
2D Convolutional	1 filters (1×1) stride (1, 1), *valid*, ReLu
2D Convolutional	32 filters (3×3) stride (1, 1), *same*, ReLu
2D Max Pooling	Pool size (2×2), *valid*
2D Convolutional	64 filters (3×3) stride (1, 1), *same*, ReLu
2D Convolutional	64 filters (3×3) stride (1, 1), *valid*, ReLu
2D Max Pooling	Pool size (2×2), *valid*
Linear	20 units
Dropout	0.3 rate
Sigmoid	2 units

(C)

Layer Type	**Description**
2D Convolutional	16 filters (3×3) stride (1, 1), *same*, ReLu
2D Convolutional	32 filters (1×1) stride (1, 1), *valid*, ReLu
2D Max Pooling	32 filters (3×3) Pool size (2×2), *valid*
2D Convolutional	64 filters (3×3) stride (1, 1), *valid*, ReLu
2D Convolutional	128 filters (1×1) stride (1, 1), *same*, ReLu
2D Max Pooling	Pool size (2×2), *valid*
Linear	100 units
Dropout	0.3 rate
Linear	20 units
Sigmoid	2 units

(D)

databases considered for its training. As can be observed in these tables, all classifiers consist of a series of stacked convolutional layers and a final fully-connected layer mapping the output of the last convolutional output to the binary target to be predicted. When backpropagating the gradients to learn the parameters of the model, we opt for binary cross-entropy loss and an Adam optimizer with a learning rate equal to 0.001. Regarding the memory of the ODTW, we choose the value of ρ that performs best for the PED model. To this end, we have considered three different CNN classifiers, each trained on frames of a particular ρ, and we have selected the parameter that produces the best validation AUC. In this design phase we have considered short ($\rho = 0.0001^{1/\nu}$), short-middle ($\rho = 0.01^{1/\nu}$) and middle ($\rho = 0.1^{1/\nu}$) range memories. These proposed values meet our claims made at the end of Section 5.2, where we stated that the ODTW should be computed with short-to-middle range memories. As aforementioned at the end of Section 5.3, the size of the sliding window is desirable to be small to enclose predictions of the PED close to the true limits between

consecutive events. Thus, we use a fixed value of $\varpi = 7$ time units for all the problems in Table 5.2.

Performance Results of the PED Model

The upper part of each plot in Figures 5.5 and 5.6 (in subsequent pages) illustrates the outcomes of the PED model for the conducted experiments. Specifically, each black curve represents a streaming time series (last five events) of the $\mathcal{D}_{\text{test}}$ set, vertical dashed lines depict limits between consecutive events, and red dots and crosses indicate true and false change PED predictions, respectively. In this regard, we assume that the detector performs correctly if the predicted event changes *overlap* the true limits of the test streaming time series. That is, given a test stream $Y_k^{n_{\text{test}}} \in \mathcal{D}_{\text{test}}$, we consider that PED correctly predicts event changes if the streaming frame corresponding to the i-th arriving observation, namely, $\overline{\mathbf{M}}^{i+1-\varpi,i}$, is labeled as *change* and a real event change takes places between $i+1-\varpi$ and i time instants.

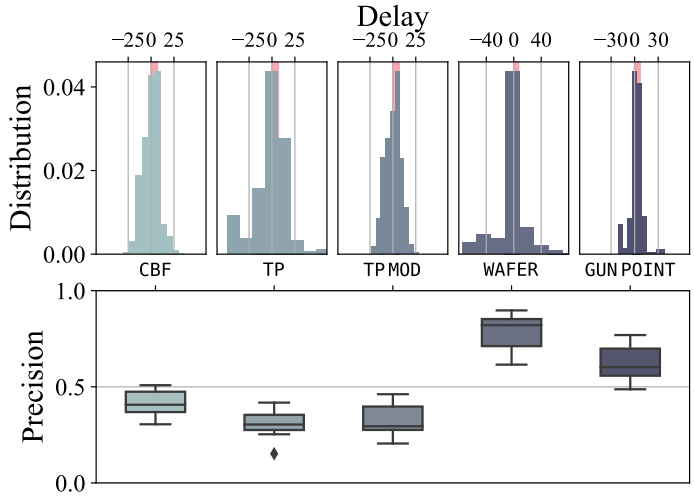


FIGURE 5.3: PED model performance results for `CBF`, `TWO PATTERNS (TP)`, `TWO PATTERNS MOD (TP MOD)`, `WAFER` and `GUN POINT` STSC problems. On top of the figure, histograms depict the distribution of the detection delay, where negative (positive) delays indicate the detector has predicted changes before (after) the real occurrence. In the lower part, boxplots illustrate the PED model precision.

On this basis, we evaluate the accuracy of the PED by computing, for each database $\mathcal{U}$ in Table 5.2 and $Y_k^{n_{\text{test}}} \in \mathcal{D}_{\text{test}}$, the delay and precision. Results corresponding to `CBF`, `TWO PATTERNS`, `TWO PATTERNS MOD`, `WAFER` and `GUN POINT` databases are shown in Figure 5.3, whereas outcomes for `FACES UCR`, `RATIONAL`, `SYNTHETIC CONTROL`, `TWO LEAD ECG` and `PLANE` databases are depicted in Figure 5.4. In both cases, histograms on the top of this figure represent the distribution of the obtained delays, where red bands in the background lie between 0 and ϖ, and represent

the range where true event changes occur. Accordingly, the more delay measurements fall into this band, the more accurate the PED is.

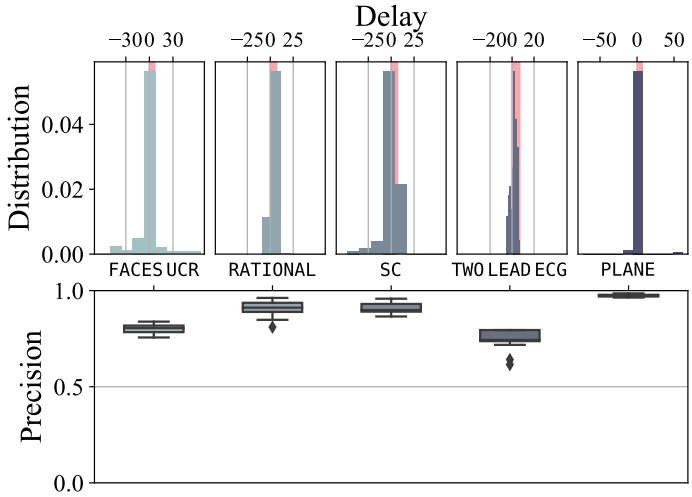


FIGURE 5.4: PED model performance results for `FACES UCR`, `RATIONAL`, `SYNTHETIC CONTROL (SC)`, `TWO LEAD ECG` and `PLANE` classification problems. Similarly to Figure 5.3, histograms illustrate the distribution of the delay, and boxplots the PED model precision.

This being said, we note that the PED trained for databases corresponding to Figure 5.4 show delay distributions with lower variance than those in Figure 5.3. However, it is important to note that, given the shape of the patterns within this last group (see e.g. `CBF`), the limits established for these examples become unclear for the ODTW/DTW, due to their elasticity property. Hence, despite the predictor appears to be quite imprecise, such an inaccuracy does not alter the main structure of the patters, as will be later supported by the STSC performance results (see discussion at the end of Section 5.4.3).

Regarding the precision of the PED model, boxplots depicted at the bottom of Figures 5.3 and 5.4 show the distribution of this score for each of the databases. According to Expression (5.13), the precision lies in $[0, 1]$, where 0 indicates failure and 1 perfect performance. In line with the delay results, PED models for databases in Figure 5.4 are more precise than models in Figure 5.3.

A summarized overview of these results is given in Table 5.4, which lists the average precision and delay for each database. We can observe that except for `CBF`, `TWO PATTERNS`, `TWO PATTERNS MOD` and `GUN POINT`, the reported results for the considered benchmark are encouraging in terms of precision. As argued previously, the limits between consecutive events in `CBF`, `TWO PATTERNS`, `TWO PATTERNS MOD` and `GUN POINT` databases are diffuse for the ODTW/DTW similarity, as per a visual inspection of the events in Figures 5.5 and 5.6. Consequently, streaming frames lose discriminating power, leading to a degradation of the precision and an increase of the delay of the PED model for these databases.

TABLE 5.4: Summary of the performance statistics of the PED model over the STSC problems under consideration.

Database $\mathcal{U}$	Precision	Delay	Classification Accuracy		
			PED-NN	GS	ODTW-NN
CBF◊	0.41 ± 0.06	−0.06 ± 8.73	0.76 ± 0.24	0.88 ± 0.17	0.63 ± 0.42
TWO PATTERNS◊	0.31 ± 0.07	−1.85 ± 18.31	0.47 ± 0.26	0.58 ± 0.30	0.40 ± 0.37
TWO PATTERNS MOD◊	0.32 ± 0.08	0.14 ± 8.73	0.80 ± 0.23	0.85 ± 0.21	0.64 ± 0.43
SYNTHETIC CONTROL◊	0.91 ± 0.03	0.98 ± 8.10	0.82 ± 0.17	0.88 ± 0.11	0.30 ± 0.15
RATIONAL◊	0.90 ± 0.04	3.72 ± 4.07	0.53 ± 0.28	0.57 ± 0.28	0.46 ± 0.24
FACES UCR	0.80 ± 0.02	0.43 ± 13.34	0.39 ± 0.20	0.45 ± 0.20	0.20 ± 0.21
GUN POINT	0.62 ± 0.09	1.55 ± 8.24	0.74 ± 0.14	0.78 ± 0.12	0.53 ± 0.15
PLANE	0.97 ± 0.01	4.08 ± 5.77	0.73 ± 0.34	0.87 ± 0.20	0.47 ± 0.40
TWO LEAD ECG	0.74 ± 0.06	1.38 ± 2.81	0.69 ± 0.15	0.72 ± 0.12	0.68 ± 0.18
WAFER	0.79 ± 0.09	−1.17 ± 24.89	0.55 ± 0.13	0.58 ± 0.14	0.51 ± 0.13

5.4.3 Efficiency of PED for STSC problems

In this subsection we evaluate the efficiency of the PED model when coupled with a DTW-NN classifier to solve STSC problems. To this end, we use the accuracy introduced in Equation (4.12). For a given database $\mathcal{U}$ and $Y_k^{n_{\text{test}}} \in \mathcal{D}_{\text{test}}$, the accuracy of the i-th arriving data point is given by:

$$\text{ACC}_k(i) = \begin{cases} 1 & \text{if} \quad \text{PL}(i) = \text{TL}(i), \\ 0 & \text{otherwise}, \end{cases} \tag{5.14}$$

where $\text{PL}(i)$ and $\text{TL}(i)$ represent the predicted label and the true label for time instant i, respectively.

Gold-standard and ODTW-NN Classifiers

As stated at the beginning of this section, we use two alternative NN-based classifiers to compare the performance of our PED based DTW-NN classifier. These classifiers correspond to the gold-standard (GS) and the ODTW-based nearest neighbor (ODTW-NN) procedure. Based on the canonical DTW, the GS approach consists of a holistic NN-based classifier that exploits perfectly a priori knowledge of the event changes. When a change occurs, the measure of similarity is reset right at the time where a new event begins, so that the model accommodates the upcoming event. On the other hand, the ODTW-NN method utilizes the forgetting mechanism of the ODTW measurement to passively adapt the classifier to the event changes of the streaming time series (see Chapter 4 for a detailed description).

GS can be thought to be the *perfect* model because it essentially reduces to an off-line DTW-NN model performed separately for each event of the test stream. However, in most realistic streaming scenarios it is difficult to know when an event transition occurs. Therefore, to ensure a more realistic comparison we use the ODTW-NN classifier. In this case we use `middle` ($0.1^{1/\nu}$) or `middle-short` ($0.01^{1/\nu}$) range memory values depending on the classification problem at hand (see Table 5.2 for a list of the specific values in use). That is, we utilize the same memory parameter values for both the PED model and the ODTW-NN approach for two reasons: i) to evaluate whether the proposed PED-based adaptation is less reactive than

the passive adaptation strategy included in the ODTW-NN classifier; and ii) to compare both PED-based DTW-NN and ODTW-NN classifiers in terms of accuracy.

PED-based DTW-NN Classifier

Our active adaptation strategy consists of exploiting the event change detection flag issued by PED to trigger the adaptation of DTW-NN classifiers to upcoming events. That is, we use the output of PED when analyzing new streaming frames over time to i) let a DTW-NN classifier predict the label of the received stream sample (if PED model outputs *no change*); or ii) adapt the overall model to the new event by resetting the measure of similarity (if the PED issued *event change*). It is important to note the main difference between this classifier and the GS procedure: while GS is unrealistically aware of the event changes occurring in the stream, our approach depends on the predictions output by PED. Thereby, GS is expected to perform better than the PED-based DTW-NN approach, as it does not undergo any false positives, nor does it suffer from any delay in the detection of the event shift. Consequently, GS serves as an upper bound of the performance that the PED-based DTW-NN model could achieve (hereafter referred to as PED-NN). The closer the outcome of PED-NN is to that of GS, the more accurate the PED model can be considered to be.

STSC Performance Results

Results corresponding to `CBF`, `RATIONAL` and `TWO LEAD ECG` STSC problems are summarized at the bottom of Figure 5.5 and 5.6, where each curve show the performance of PED-NN (green), GS (black) and ODTW-NN (gray) classifiers. More precisely, such plots display the accuracy over time averaged over the $K_{\text{test}} = 12$ streaming time series in $\mathcal{D}_{\text{test}}$:

$$\overline{\text{ACC}}(i) = \frac{1}{K_{\text{test}}} \sum_{k=1}^{K_{\text{test}}} \text{ACC}_k(i), \tag{5.15}$$

where $\text{ACC}_k(i)$ represents the accuracy of the i-th arriving point of the k-th streaming time series as per Equation (5.14).

We focus on the results obtained for `CBF` STSC problem (Figure 5.5a). For this problem, we observe that both the PED-NN and the GS classifiers show similar average accuracy results, specially in the middle of every event. On the contrary, at the beginning/end of the event the predictions issued by the PED-NN model degrade, getting closer to those corresponding to the ODTW-NN approach. Indeed, if we pay attention to the outputs of the PED model (red dots), it is straightforward to notice that such predictions have an ample variability, which implies that the similarity is reset in the PED-NN model sooner/later than expected. As a result, the accuracy decreases around the true limits of the event. Results issued for the `RATIONAL` STSC problem (Figure 5.6b) exhibit a similar behavior for both PED-NN and GS classifiers. Regarding the ODTW-NN, the evolution of $\overline{\text{ACC}}$ stabilizes to levels below the accuracy of PED-NN and GS as we

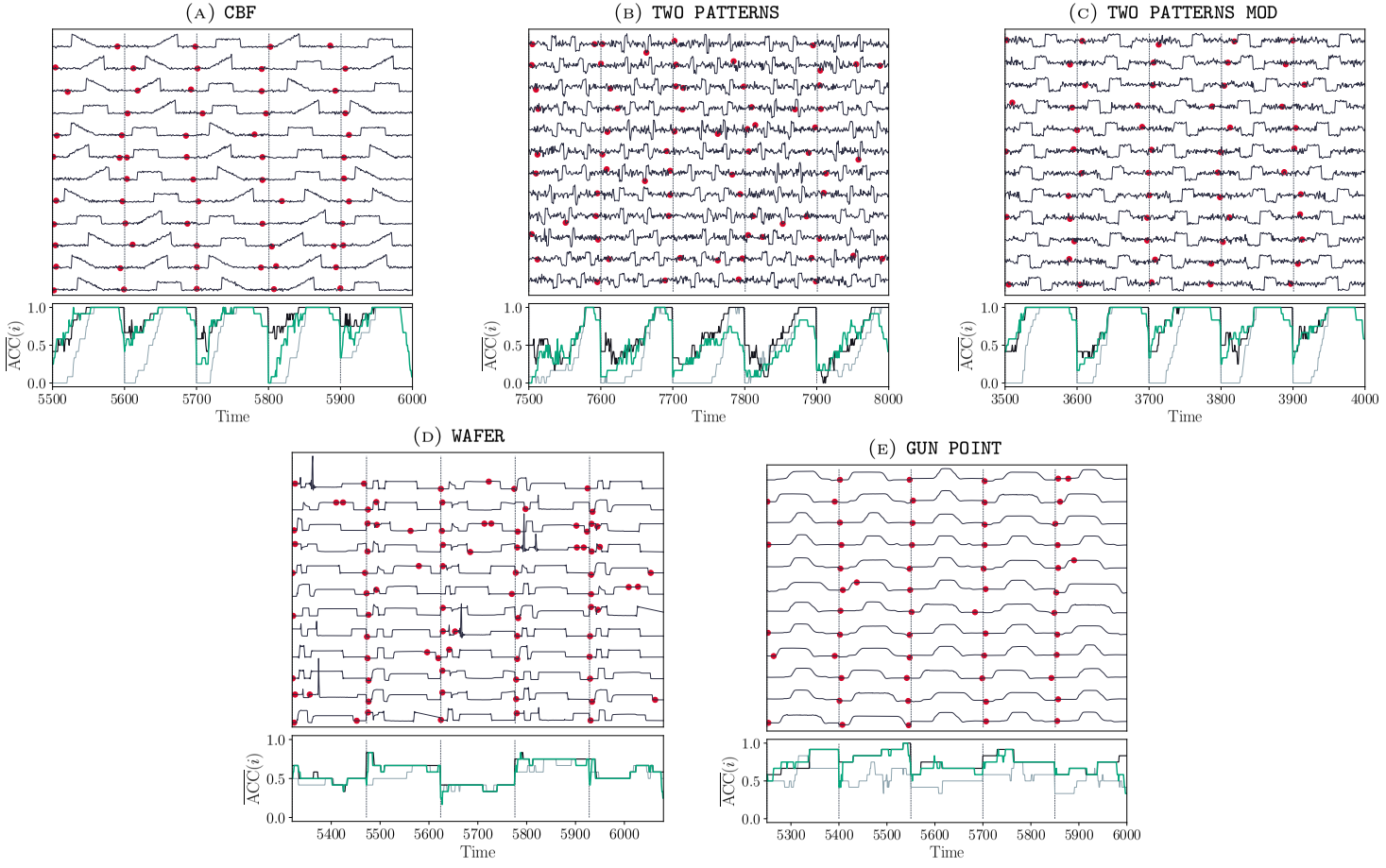


FIGURE 5.5: Results of the PED model in terms of detection performance (top) and average accuracy of the PED-NN model over STSC problems (bottom) for (a) `CBF`, (b) `TWO PATTERNS`, (c) `TWO PATTERNS MOD`, (d) `WAFER` and (e) `GUN POINT` problems. On the top, black curves depict the last 5 events of each streaming time series in $\mathcal{D}_{\text{test}}$, black vertical lines depict event changes and red dots PED model change predictions. On the bottom, gray, black and green curves illustrate average accuracy scores $\overline{\text{ACC}}$ along the last 5 events of test streams for GS, ODTW-NN and PED-NN classifiers, respectively.

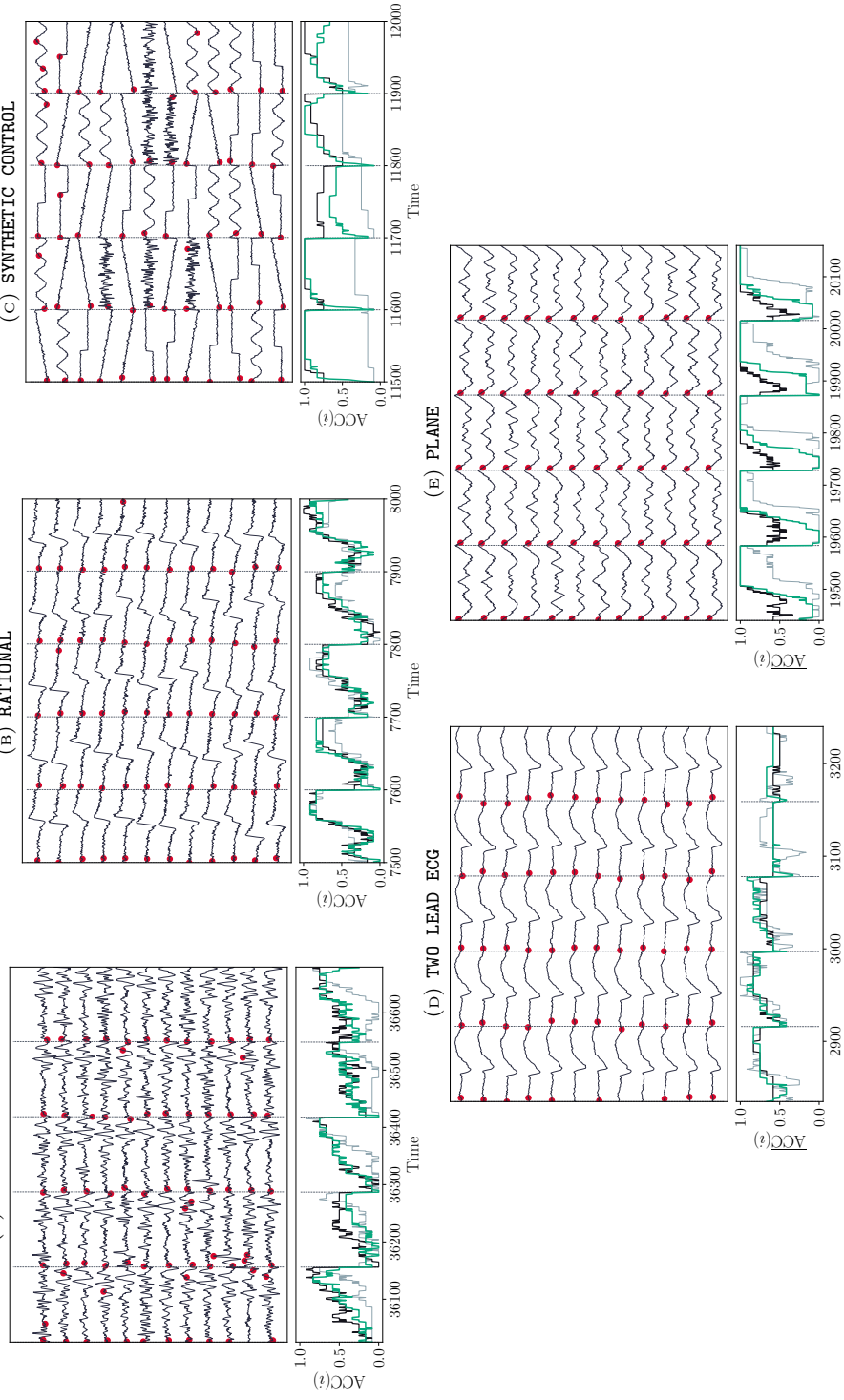


FIGURE 5.6: PED model *change* predictions (top) and average accuracy of the PED-NN model (bottom) for (a) FACES UCR, (b) RATIONAL, (c) SYNTHETIC CONTROL, (d) TWO LEAD ECG and (e) PLANE STSC problems.

move forward along the event. These conclusions also hold for the `TWO LEAD ECG` database (Figure 5.6d). In contrast to the previous example, in this case the ODTW-NN classification accuracy evolves along with GS and PED-NN classifiers.

For the sake of of brevity, we have only described in detail the progression of $\overline{\text{ACC}}$ for three representative experiments. The detailed results for the remaining databases are depicted in Figures 5.5 and 5.6. According to the division made in the previous section, we have grouped in Figure 5.5 the databases with diffuse event changes (e.g. `CBF`). On the other hand, results in Figure 5.6 correspond to databases with well-delimited event changes, such as `RATIONAL` and `TWO LEAD ECG` problems.

Averaged results obtained from all the STSC experiments are summarized in Table 5.4. For each utilized classifier (PED-NN, GS and ODTW-NN), the last three columns list the $\overline{\text{ACC}}$ results averaged over the whole simulated period, i.e., for $i = 1, \ldots, n_{\text{test}}$. As can be concluded from these results, both the PED-NN classifier and the GS model render similar performance levels in almost all the conducted experiments, which unveils that predictions of the PED model do not affect severely the structure of the streaming time series. Moreover, if we compare the results of PED-NN and ODTW-NN classifiers, we can see that the former leads to better results. Hence, the PED-NN classifier proposed in this chapter not only efficiently accommodates the NN classifier to upcoming events without disturbing any essential relationship among consecutive data points but also yields more accurate results as it incorporates DTW (which uses the information of the full event) instead of ODTW (which forgets information of the past as it moves forward along the event).

On a closing note, these results are conclusive in regard to the convenience of active adaptation strategies for streaming time series classification with respect to passive adaptation methods like the ODTW proposed in the previous chapter. Furthermore, they emphasize the rich information about the stream dynamics contained in the measure matrix, spanning several research lines departing from this statement.

5.5 Conclusions and Future Research

In this chapter, we have proposed to analyze streaming frames (i.e., partial similarity measurements among all sub-sequences of streaming time series) towards identifying the transition between consecutive events. Our motivation for this purpose is to adapt DTW-NN models by leveraging the detection of such transitions. To this end, we have proposed PED, a CNN classifier that categorizes streaming frames over time into *event change* or *no event change* classes. When a change is detected, a DTW-NN can adapt to the upcoming event by resetting the computation of the DTW on which it relies to predict the class of the arriving stream. A benchmark comprising 10 streaming time series classification databases has been designed to assess the performance of the proposed model and to compare it to that of i) the passive adaptation mechanism proposed in Chapter 4 (memory

mechanism), and ii) a gold-standard DTW-NN scheme that assumes perfect a priori knowledge of the instants at which event transitions occur to reset the DTW computation. The reported results have demonstrated that the accuracy of the proposed PED-based DTW-NN procedure and the gold-standard classifier behave similarly, which underlines the value of streaming frames as an information source of value for problems defined over streaming time series.

Several research directions are planned to investigate further uses of the information contained in streaming frames, including clustering and early classification in streaming time series. Moreover, alternative deep neural network architectures can be also studied to decrease the variability of predicted event changes. Finally, given the potential showcased by streaming frames when detecting changes between events, further efforts will be invested towards examining whether streaming frames can also discriminate among labels in time series classification.

Chapter 6

Using TSA Tools to build Robust Image Classifiers

In Chapters 4 and 5 we have emphasized that, in the off-line setting, predictive models assume stationary data. This assumption means that both training and test samples are governed by the same distribution. This assumption, however, can pose a serious security threat for predictive models. Actually, several recent works have shown that learning algorithms can be easily misled when fed with intelligently manipulated data. Consequently, an individual can intentionally alter the performance of a model by slightly manipulating, either in the training or in the test stage, the latent distribution of the data, i.e., breaking away the stationary assumption. The field of Adversarial Machine Learning (AML) addresses this issue, deriving new attack strategies and countermeasures devised to make predictive models robust against adversarial inputs.

This chapter frames its contribution within the AML research area, by proposing a DTW-based defense strategy to improve the robustness of Deep Neural Network (DNN) image classifiers against adversarial attacks. DNN classifiers have been recently proven to be specially prone to adversarial attacks. In order to increase their robustness we propose a DTW-based one-class classifier, which analyzes the reliability of input images by processing the sequences generate from the color gradient of input images at different levels of sensitivity. As will be later shown, the capability of a DNN model to detect a diversity adversarial attacks is in general improved when the model is endowed with the proposed defense strategy.

6.1 Introduction and Related Work

In recent years, DNNs have shown a superior performance in challenging classification problems that had remained open for decades. These models have been shown to outperform traditional feature-based learning methods, particularly in image classification and speech recognition, where DNNs, by virtue of their capacity to learn features from unprocessed data, have achieved unrivaled levels of accuracy. As a result, DNNs have recently become the modeling reference in a plethora of application domains such

as recommender systems [97], driving assistants [98], industrial prognosis [42], [99] or medical diagnosis [100], among others [101].

One of the principal reasons for the renowned success of DNNs hinges on their internal architecture: a layer-wise composition of functions capable of transforming the input data into a representation characterized by a higher abstraction level [102], [103]. This layered architecture, together with new forms of neural computation (such as convolutional filters and recurrent neurons endowed with sequence learning capabilities), have been proven to be superior at discovering non-linear input-output relationships from complex data, while requiring very little data preprocessing. As opposed to conventional machine learning models, DNNs can be learned directly from raw data, without any need for designing and applying feature extraction techniques that can be computationally demanding, or even unfeasible in problems with highly-dimensional data.

Notwithstanding their superior performance, DNNs also undergo several limitations and caveats. On one hand, DNNs are computationally expensive to train, as they require large amounts of data due to their large parametric space. On the other hand, interpreting the reasons why an algorithm has arrived to a particular decision still remains unclear. Therefore, DNNs are frequently regarded as black-box models with an empirical – rather than theoretical – understanding of their inner behavior [104]. The lack of interpretability [105] and high number of parameters bring on one last limitation of DNNs: their vulnerability to adversarial attacks. The first example of this weakness was presented in [72], where it was revealed that DNNs draw counter-intuitive results when provided with subtly yet intelligently manipulated inputs, coined as *adversarial examples*. The implications of this noted issue can be better understood when exemplified in a practical scenario. For instance, if we consider an autonomous vehicle integrating a DNN to detect obstacles and pedestrians from images taken from the street, the observations and discussions held in [72] make it possible that an attacker could mislead the DNN – and, consequently, endanger the safety of pedestrians – by making small perturbations in the analyzed images, which can be as easy as sticking an apparently inoffensive printed film onto a traffic signal [106].

Clearly, the existence of adversarial examples poses a severe threat for DNN-based applications. For this reason, the machine learning community has recently taken a closer look at the intersection between DNN and computer security, thereby spawning a flurry of research related to the AML paradigm [73], [75]. AML techniques are widely described as a two-player game: a *defender* aims to build a model capable of solving a classification task, whereas an *attacker* attempts to devise a strategy to degrade the performance of the model by subtly modifying the data examples to be predicted. In this context, a number of different attacking alternatives can be made available to the attacker, depending on the final objective of the attack [107]. In this work, we focus on *evasion* attacks, namely, attacks where the adversary perturbs test inputs in order to make the model produce a label error. Such attacks are commonly of two types: *targeted* or *untargeted*. Specifically, when considering *targeted* attacks, the

objective is to mislead the predictive model by producing a label error towards a specific label of interest, e.g., *authorized*. When dealing with *untargeted* attacks, the goal is to produce an indiscriminate error in the model of the defender, no matter which erroneous label is predicted. A second aspect to consider when setting the rules of an adversarial game is the knowledge the attacker possesses about the system to be attacked prior to the adversarial sample crafting process. The attacker might have *full*, *partial* or *no knowledge* of the architecture of the model under target, its parameter values, and/or the training data from where it was learned. In the literature, such scenarios are referred to as *white-box*, *gray-box* or *black-box* attacks.

As previously mentioned, in computer vision scenarios attackers have some constraints when manipulating legitimate images. Indeed, crafted adversarial examples must be similar to the original ones, so an external observer can visualize and recognize the original object without noticing any modification in the image. In this context, theoretical studies dealing with human visual perception of objects have shown that most neurons conforming the primary human visual cortex are sensitive to color variations, and are functionally devoted to the extraction of edges from a given scene by reacting strongly to chromatic contrast borders (see [108], [109] and references therein). In other words, the human visual cortex is known to be more sensitive to shape changes than color variations. This weakness is what adversarial players leverage by modifying pixel values in the image, so the crafted adversarial sample achieves its goal efficiently by fooling the DNN, without any apparent modification to the human eye.

This chapter builds on this noted weakness to present a novel method to make DNNs for image classification more robust against adversarial examples. The proposed approach consists of a one-class distance feature classifier that determines the legitimacy of input images by checking feature vectors built on measures of the DTW (see Section **??**). More precisely, it hinges on the similarity among sequences that summarize the color content extracted from input images at different levels of sensitivity. As explained in the remainder of the chapter, two steps are needed to build this type of classifiers. First, a similarity matrix is generated by computing the similarity among sequences build on images color differences. More precisely, these sequences are extracted by from each image in the classification database. The second consists of learning a distance feature classifier using each row (or column) of the similarity matrix as an input feature vector for the model. With the idea of exploiting the potential of ESMs in such classifiers, the authors in [110] presented a SVM classifier for time series classification whose kernel was built by means of the feature vectors extracted from the computation of the DTW between time series. In particular, two different DTW-based kernels were considered in this work, namely, the Gaussian Dynamic Time Warping $K_{\mathrm{GDTW}}(Z,Y) = \exp\left[-\gamma DTW\left(Z,Y\right)\right]$, with $\gamma > 0$; and the Negated Dynamic Time Warping given by:

$$K_{\mathrm{NDTW}}(Z,Y) = -DTW\left(Z,Y\right), \tag{6.1}$$

which will be used to compute the kernel for the SVM model of the herein proposed adversarial defense method. Results presented by the authors in [110] showed that the SVM built in this way can be as efficient as other benchmark sequence classifiers.

Several experiments are devised and set up to quantitatively assess the performance of the proposed method. We will first examine the behavior of the discriminator when deployed as a defensive countermeasure to several state-of-the-art attack strategies, for which performance statistics will be computed for both legitimate and adversarial inputs. We compare the obtained results with those drawn by eight different defense methods. The obtained results will show that the proposed strategy reduces the success rate of those attacks that attempt to confuse the classifier by slightly manipulating pixels over the entire original image. To the best of our knowledge, there is no prior work in the literature using tools from time series analysis for developing adversarial defense strategies.

The chapter is organized as follows: Section 6.2 places a short background on AML. Section 6.3 describes the proposed defense strategy in detail. Section 6.4 presents the experimental setup designed to shed light on the performance of the proposed approach, followed by a thorough discussion on the obtained results. Finally, Section 6.6 summarizes the main contributions of the chapter, and outlines future research lines motivated by our reported findings.

6.2 Adversarial Machine Learning

Let us assume a defender who is asked to solve a classification task based on a dataset of legitimate (i.e. non-adversarial) examples $\mathcal{D} = \{(\mathbf{x}_k, l_k)\}_{k=1}^{K}$, where $\mathbf{x}_k \in \mathcal{X}$ represents the k-th training example and $l_k \in \{1, \ldots, L\}$ its corresponding class label. In order to accomplish this goal, we assume the defender resorts to a DNN classifier that learns a composition of functions:

$$f(\mathbf{x}; \Theta) = f^{(M)}(\boldsymbol{\theta}_M; f^{(M-1)}(\boldsymbol{\theta}_{M-1}; \ldots f^{(1)}(\boldsymbol{\theta}_1; \mathbf{x}))), \tag{6.2}$$

where the parametric function $f^{(m)}(\boldsymbol{\theta_m}; \cdot)$ $(m \in \{1, \ldots, M\})$ represents the m-th layer of the DNN; and $\Theta = \{\boldsymbol{\theta}_m\}_{m=1}^{M}$ the set of weights to be learned. To this end, the DNN model is provided with $\mathcal{D}$ and adjusts the weights in Θ by reducing a cost (loss) function $C\left(f(\mathbf{x}_k; \Theta), l_k\right)$ [94]. When clear from the context, we will hereafter use $f(\mathbf{x})$ instead of $f(\mathbf{x}; \Theta)$ to denote a given DNN, $C_f\left(\mathbf{x}_k, l_k\right)$ instead of $C\left(f(\mathbf{x}_k; \Theta), l_k\right)$ to express its loss function; and $f_l(\mathbf{x})$ to express the result drawn by the DNN for example $\mathbf{x}$ and label l.

Given a DNN classifier $f(\cdot)$ trained on the legitimate dataset $\mathcal{D}$, and a clean example $\mathbf{x}$, the AML paradigm aims at producing a perturbed example $\tilde{\mathbf{x}} = \mathbf{x} + \boldsymbol{\delta}_{\text{adv}}$ so that $f(\mathbf{x}) \neq f(\tilde{\mathbf{x}})$, i.e., $\tilde{\mathbf{x}}$ confuses the model. We must not lose sight of the fact that adversarial samples must not only fool the target DNN, but also any possible external observer, e.g., attacks should not be visually noticed when $\mathbf{x}$ represents an image. In this case, the added perturbation $\boldsymbol{\delta}_{\text{adv}}$ is usually constrained, so it is undetectable to the human eye. A common strategy in this regard is to impose $||\boldsymbol{\delta}_{\text{adv}}||_p \leq \varepsilon$,

where $||\cdot||_p$ denotes the L_p-norm and $\varepsilon \in \mathbb{R}^+$ is the maximum allowed perturbation. It is important to note that as ε decreases, so does the difference between the legitimate image $\mathbf{x}$ and the crafted adversarial example $\tilde{\mathbf{x}}$, which ultimately reduces the probability of $\tilde{\mathbf{x}}$ being detected by an external observer. Based on these observations, the AML problem can be formally stated as follows:

$$\begin{aligned} \boldsymbol{\delta}_{\text{adv}} = \arg\min_{\boldsymbol{\delta}\in\mathcal{X}} \quad & \Psi(\mathbf{x}, \mathbf{x}+\boldsymbol{\delta}), \\ \text{subject to:} \quad & ||\boldsymbol{\delta}||_p \leq \varepsilon, \\ & f(\mathbf{x}+\boldsymbol{\delta}) = l', \end{aligned} \tag{6.3}$$

where $\Psi(\mathbf{x}, \mathbf{x}')$ is a measure of dissimilarity between sample $\mathbf{x}$ and $\mathbf{x}'$, and l' denotes the class label that $f(\cdot)$ must predict for the perturbed example. In targeted problems, where the attacker is willing to cause specific label errors in the model, l' refers to the target label, i.e., $l' \in \{1, \ldots, l-1, l+1, \ldots, L\}$, where l is the class label of the legitimate image $\mathbf{x}$, i.e., $l = f(\mathbf{x})$. For untargeted problems, l' can be any label different from the original one, that is, $l' \neq l$.

6.2.1 Attack Strategies

In general, solving the problem in (6.3) is not a straightforward task. As a result, many adversarial example generation techniques have been proposed in the literature as means to solve this problem [111], [112]. This chapter focuses on gradient-based adversarial crafting approaches, which use gradient-based optimization algorithms to search solutions for the AML problem [73]. These attack strategies include the Fast Gradient Sign Method (FGSM [113]), the Basic Iterative Method (BIM [114]), the Projected Gradient Descent (PGD [115]), the Jacobian Saliency Map Attack (JSMA [111]) and the Carlini and Wagner attack (C&W [116]). These methods were originally designed to generate adversarial samples against DNNs, and have lately become reference attack strategies in the AML literature for two main reasons: their computational efficiency, and their accuracy when attacking image classifiers [73], [74], [117]–[119]. This section briefly introduces each of these methods.

When analyzing the stability of DNN classifiers, Szegedy et al. first realized that slight, non-random image variations were able to change the prediction of DNN classifiers [72]. Indeed, by adapting the limited-memory version of the Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) optimization algorithm to solve the constrained problem in (6.3), the authors demonstrated that they were able to craft a large number of examples against different benchmark image classifiers. In a similar way, the attack presented in [116] resorted to an alternative formulation of the aforementioned optimization problem and the Adam optimizer to find accurate adversarial samples. Specifically, the authors proposed the following alternative

problem to develop the C&W attack:

$$\begin{aligned} \boldsymbol{\delta}_{\text{adv}} = \arg\min_{\boldsymbol{\delta}\in\mathcal{X}} \quad & ||\boldsymbol{\delta}||_p + c \cdot h(\mathbf{x}+\boldsymbol{\delta}), \\ \text{subject to:} \quad & \mathbf{x}+\boldsymbol{\delta} \in [0,1]^{|\mathcal{X}|}, \end{aligned} \tag{6.4}$$

where $h(\cdot)$ was defined such that $f(\mathbf{x}+\boldsymbol{\delta}) = l' \Leftrightarrow h(\mathbf{x}+\boldsymbol{\delta}) \leq 0$. Besides, in order to meet the constraint in (6.4), the authors introduced $\boldsymbol{w}$ as:

$$\delta_i = \frac{1}{2}\left[\tanh(w_i+1)\right] - x_i \;\forall i \in \{1,\ldots,|\mathcal{X}|\}, \tag{6.5}$$

which ensures that $x_i + \delta_i \in [0,1]$, since $|\tanh(w_i)| \leq 1$. Hence, combining (6.4) and (6.5) yields the following non-constrained optimization problem:

$$\boldsymbol{w}_{\text{adv}} = \arg\min_{\boldsymbol{w}\in\mathbb{R}^{|\mathcal{X}|}} \left\| \frac{1}{2}\left[\tanh(\boldsymbol{w}+1)\right] \right\|_p + c \cdot h\left(\frac{1}{2}\left[\tanh(\boldsymbol{w}+1)\right]\right), \tag{6.6}$$

where:

$$h(\cdot) = \max\left\{\max\left[f_l(\cdot) \mid l \neq l'\right] - f_{l'}(\cdot), -\kappa\right\}, \tag{6.7}$$

and $\kappa \geq 0$ in (6.7) represents the confidence under which the generated sample misleads the classifier. In this chapter we will use $p = 2$ (i.e. L_2-norm) when crafting adversarial samples from this attack strategy.

The C&W attack is accurate in crafting adversarial samples at the cost of a high computational complexity. Due to its relative simplicity and lower processing demand, the FGSM attack [113] has become one of the most frequently utilized adversarial crafting techniques in AML related contributions. In contrast to the problem formulated above, where $\boldsymbol{\delta}_{\text{adv}}$ is computed by minimizing a thoughtfully constructed objective function, FGSM calculates the perturbations by using the gradient of the loss function to minimize it towards the target label l'. That is:

$$\boldsymbol{\delta}_{\text{adv}} = \texttt{sign}\left[\nabla_{\mathbf{x}} C_f(\mathbf{x}, l')\right], \tag{6.8}$$

where $\texttt{sign}[\cdot]$ denotes the sign function, $C_f(\cdot,\cdot)$ the loss of the DNN classsifier, and $\nabla_{\mathbf{x}}$ the gradient operator with respect to variable $\mathbf{x}$. In this context, the adversarial example is given by:

$$\tilde{\mathbf{x}} = \mathbf{x} + \varepsilon \cdot \texttt{sign}\left[\nabla_{\mathbf{x}} C_f(\mathbf{x}, l')\right]. \tag{6.9}$$

In a similar line of reasoning, an iterative version of the FGSM approach coined as BIM was proposed in [114]. This attack strategy produces adversarial samples by the following recurrence:

$$\widetilde{\mathbf{x}}^{(t+1)} = \texttt{Clip}_{\mathbf{x},\varepsilon}\left[\tilde{\mathbf{x}}^{(t)} + \alpha \cdot \boldsymbol{\delta}_{\text{adv}}^{(t)}\right], \tag{6.10}$$

where $t \in \{1,\ldots,T_{\max}\}$ denotes the iteration number, α is a predefined constant, $\boldsymbol{\delta}_{\text{adv}}^{(t)} = \texttt{sign}[\nabla_{\mathbf{x}} C_f(\tilde{\mathbf{x}}^{(t)}, l')]$ is the perturbation made to the original sample along iterations, and $\texttt{Clip}_{\mathbf{x},\varepsilon}[\cdot]$ is the so-called clipping operator

of the input sample defined for $x_i \in \mathbf{x}$ and $\tilde{x}_i \in \tilde{\mathbf{x}}$, as:

$$\texttt{Clip}_{\mathbf{x},\varepsilon}[\tilde{\mathbf{x}}] = (\min\{1, x_i + \varepsilon, \max\{0, x_i - \varepsilon, \tilde{x}_i\}\})_{i=1}^{|\mathcal{X}|}, \tag{6.11}$$

where $|\mathcal{X}|$ denotes the number of elements of $\mathbf{x}$. This clipping operator ensures that, in the L_∞ space, the resulting adversarial sample $\tilde{\mathbf{x}}$ falls within the ε-neighborhood of the legitimate sample $\mathbf{x}$. Likewise, the PGD attack proposed in [115] takes recursive steps in the direction of the greatest loss by forcing the perturbed sample to be projected on a L_∞-ball of ε radius at each step. However, in contrast to BIM, the PGD starts by adding a random perturbation around the ε-neighborhood of the input sample.

The authors in [111] proposed JSMA, a crafting technique that exploits the gradients of $f(\cdot)$, the function learned during the training phase, to mislead DNN classifiers. Specifically, JSMA computes the Jacobian matrix of $f(\cdot)$ in order to estimate the adversarial saliency map, which is used to identify the pixels that imprint the most significant changes on the DNN model. Hence, small perturbations are inserted into such pixels, producing adversarial samples that might mislead the model. The proposed strategy comprises three main steps that are repeated recursively until a stopping criteria is met. In the first step, the Jacobian matrix is computed for the input sample $\mathbf{x}$ as:

$$J_f(\mathbf{x}) = (f_{il'}) \in \mathbb{R}^{|\mathcal{X}| \times L}, \text{ where } f_{il'} = \frac{\partial f_{l'}(\mathbf{x})}{\partial x_i}, \tag{6.12}$$

and $f_{l'}(\mathbf{x})$ denotes the value of $\mathbf{x}$ to be assigned to the target class label l' (i.e., the value output by the DNN when $f(\mathbf{x}) = l'$). The second step corresponds to the construction of the adversarial saliency map $S(\mathbf{x}, l') = \{S_i(\mathbf{x}, l')\}_{i=1}^{|\mathcal{X}|}$, whose i-th component is given by:

$$S_i(\mathbf{x}, l') = \begin{cases} 0 & \text{if } J_{il'} < 0 \text{ or } \sum\limits_{j \neq l'} J_{ij} > 0, \\ J_{il'} \left| \sum\limits_{j \neq l'} J_{ij} \right| & \text{otherwise.} \end{cases} \tag{6.13}$$

The final step of the JSMA adversarial crafting procedure corresponds to the modification of $\mathbf{x}$, which is made by increasing its identified critical features by means of a distortion parameter α. Based on the above definition, it can be observed that the probability of the target class l' increases (and thereby, the likelihood of the remaining class labels decreases) when features with larger saliency values $S_i(\mathbf{x}, l')$ are increased. Hence, after identifying the feature $i_{\max}$ with maximal saliency value, i.e., $i_{\max} = \arg\max_i S_i(\mathbf{x}, l')$, it is perturbed via distortion parameter α. This process is repeated until i) the modified input fools the classifier; ii) all pixels have been modified; or iii) the dissimilarity between the original and the modified sample exceeds ε.

6.2.2 Defense Strategies

After the seminal work of Szegedy et al. in [72], several defense mechanisms have been proposed ever since with the aim of making DNNs resilient to adversarial attacks. This section overviews the most renowned techniques used for this purpose, by considering whether the defense is held during the training phase of the model under attack [73] or, instead, the technique at hand relies on detecting whether a test sample has been modified to confuse the target model:

- *Model training*: in this first category two canonical defense strategies stand out when increasing the robustness of DNN to adversarial examples, namely, adversarial training [113] and defensive distillation [120]. These techniques aim to build robust DNN by using modified samples during the training stage. Specifically, the adversarial training procedure proposed in [113] improves the robustness of DNNs by introducing adversarial samples during the training stage, whereas defensive distillation aims to smooth the decision surface of the model. To this end, the defensive distillation strategy trains two identical DNNs, the initial network $f(\cdot)$ and the distilled model $f_d(\cdot)$; the latter exploits knowledge extracted from $f(\cdot)$ (in particular, class probability vectors) to make the model resilient to adversarial samples. Instead of learning the initial network $f(\cdot)$ to smooth the targets of the input images, the Label Smoothing technique [121] directly modifies the annotation of the training set.

- *Adversarial detection*: an alternative defense approach is to analyze the input test examples before their classification. By using auxiliary models that identify divergences between training and prediction data, adversarial inputs can be detected and filtered out before being predicted, thereby making the classifier predict exclusively *clean* examples. This is indeed the design rationale of Deep K-Nearest Neighbors (DkNN), a hybrid model proposed in [74] that combines a DNN and a K-NN) classifier. The DNN is used as the base classifier: during its training stage, paths made by training samples through the internal components (i.e., layers and neural connections) of the network are traced and saved. During test time, paths covered by test inputs are compared with those saved at the training stage by a K-NN classifier. Predictions are declared legitimate only for those test samples whose labels predicted by the DNN are supported by predictions of their K nearest training samples. Other detection proposals contributed lately include the Feature Squeezing and the Spatial Smoothing defense strategies [119]. As stated by their authors, the objective of Feature Squeezing is to limit the capacity of the attacker by transforming input images into a lower-color-depth version of the original samples. On the other hand, Spatial Smoothing utilizes blurring techniques to reduce the differences among individual pixels. In order to detect adversarial samples, these techniques compare the prediction of the original and transformed samples to determine whether the image under test has been manipulated. Following this idea of preprocessing input images, the Thermometer Encoding approach [122] applies

a non-linear and non-differentiable function to transform the input image before processing it through the DNN classifier. Likewise, the JPEG Compression method [123] employs the compression of input images in order to mitigate/erase the variations introduce by the adversarial players. Another scheme for adversarial detection is the Total Variation Minimization defense strategy [117], which rebuilds the original image with minimum variation from a set of randomly selected pixels.

Similarly to the strategies in the last of the above categories, the approach proposed in this chapter can be regarded as an adversarial detection strategy that learns an auxiliary model capable of discriminating clean and adversarial samples. In contrast to the model presented by Papernot et al. [74], our approach uses an independent[1] discriminator built on a set of specifically extracted features that encode color discontinuities from input images. Detecting on the basis of these extracted features overcomes the limited ability of the human visual cortex when detecting minor variations in color space. Specifically, we extract a set of sequences from each image in the database through the evaluation of color changes at different levels of sensitivity. Then, we transform each sequence into a feature vector by computing the distances to other series in the database. A one-class classifier is then fit based on the obtained vectors of features. It is important to note that the proposed discrimination model does not depend on the attacker's strategy, nor does it rely on any assumption about the defender's classification model. That is, our model is focused on attacks that rely on slight pixel variations over the whole image, thus unseen samples are uniquely assessed over their color change distribution regardless of the classifier in use.

6.3 Proposed Adversarial Defense Method

In previous sections we have stressed on the fact that the human eye system detects shape anomalies more efficiently than color variations [108], [109]. This is, in fact, the reason why in image classification attackers tend to slightly manipulate the color space of clean images to maximize the probability of misleading the target model. By ensuring small modifications in the color space, the chances of a successful attack are high while making the adversarial image apparently invariant for an external observer (see FGSM and BIM in Section 6.2.1). In order to discriminate such subtle perturbations, we propose a two-step preprocessing approach capable of discerning whether the test sample is adversarial or legitimate, followed by the prediction of the category by the target model exclusively for those inputs declared as legitimate.

The proposed approach is described in detail in Algorithms 6.1 (training) and 6.2 (prediction). It consists of a *discriminator* model which decides whether an image is legitimate or adversarial before the classifier under attack actually predicts its category. This independence between

[1] In this context, we use the term *independent* to emphasize that the discriminator is built without any consideration of the model to be defended.

the discriminator and the classifier implies that i) our approach can be applied to enhance the robustness of classifiers of any kind (not only DNN); and ii) the design of the discriminator is not restricted to any specific adversarial crafting strategy.

The discriminator is designed as a one-class classifier learned from a set of legitimate images $\mathcal{D}$, which permits the identification of manipulated samples disregarding the attacker's strategy. Without any loss of generality, the discriminator is chosen to be a One-class Support Vector Machine (OSVM [124]), whose kernel function (i.e., the similarity function that allows comparing any pair of examples) is based on the DTW measure of similarity between time series. Given its ability to handle local distortions and mitigate their effect on the computed similarity, this elastic measure of similarity has been widely utilized in different problems with time series, such as time series classification or clustering [12]. As mentioned in Section 6.1, SVM-based models with DTW kernels have been proven to perform efficiently in several studies [110].

Algorithm 6.1 Proposed method: *training*

Require: Dataset $\mathcal{D} = \{(\mathbf{x}_k, l_k) : k = 1 \ldots, K\}$ composed by 3-channel legitimate images $\mathbf{x}_k = [\mathbf{x}_k^R, \mathbf{x}_k^G, \mathbf{x}_k^B]$, edge detection algorithm $\texttt{Edge}_\tau(\mathbf{x})$ with sensitivity parameter τ, and value range for the sensitivity parameter $\{\tau_1, \ldots, \tau_m, \ldots, \tau_M\}$.

for $c \in \{R, G, B\}$ **do**
  ▷ Let $\mathbf{K} = \left(K_{u,v}\right) \in \mathbb{R}^{N \times N}$ be the kernel matrix.
  for $u \in \{1, \ldots, K\}$ **do**
    for $v \in \{1, \ldots, K\}$ **do**
      ▷ Define the edge count sequences:
      $\mathbf{e}_u^c = \{e_{u,m}^c\}_{m=1}^M$ for channel image $\mathbf{x}_u^c$
      $\mathbf{e}_v^c = \{e_{v,m}^c\}_{m=1}^M$ for channel image $\mathbf{x}_v^c$
      for $m \in \{1, \ldots, M\}$ **do**
        ▷ Process channel image with edge detector:
        $\mathbf{x}_{u,\tau_m}^{c,edge} = \texttt{Edge}_{\tau_m}(\mathbf{x}_u^c)$
        $\mathbf{x}_{v,\tau_m}^{c,edge} = \texttt{Edge}_{\tau_m}(\mathbf{x}_v^c)$
        ▷ Set edge count for sensitivity τ_m and channel c:
        $e_{u,m}^c \leftarrow$ number of edge pixels in $\mathbf{x}_{u,\tau_m}^{c,edge}$
        $e_{v,m}^c \leftarrow$ number of edge pixels in $\mathbf{x}_{v,\tau_m}^{c,edge}$
      end for
      $K_{u,v} = -DTW(\mathbf{e}_u^c, \mathbf{e}_v^c)$ (negated DTW, Expression (6.1))
    end for
  end for
  Train a one-class classifier $f_{oc}^c(\mathbf{e}^c)$ for edge count sequences based on the kernel matrix
end for
return Trained discriminators (one-class classifiers) $f_{oc}^R(\cdot)$, $f_{oc}^G(\cdot)$, $f_{oc}^B(\cdot)$

In this work we deal with images, which we represent by their constituent RGB channel images $\mathbf{x} = [\mathbf{x}^R, \mathbf{x}^G, \mathbf{x}^B]$. To detect whether an image $\mathbf{x}$ is legitimate, the discriminator resorts to a gradient-based edge

detection algorithm to extract a binary image per every RGB channel (lines **10** and **11**). This edge detection phase is carried out for several sensitivity levels τ of the edge detection algorithm (e.g., the so-called *threshold* of a Canny detector [125]), so that the resulting set of binary images reflects the spatial distribution of the detected edges per every color channel. We aggregate the edge information contained in these images by summing up all their pixels (lines **13** and **14**), resulting in a sequence of *edge counts* per every channel, each denoting the number of edge pixels in the image detected at different sensitivity levels.

The rationale behind this procedure is that these sequences, when obtained from an adversarial sample, yield a higher number of detected edge pixels than the original image from where it was furnished. Adversarial attacking mechanisms modify the pixels of the image disregarding the effect of the attack in the distribution of edges detected at different sensitivities. As we will later verify experimentally, adversarial attacks produce more pixels to be detected as edges beyond a given level of sensitivity of the detector, thereby yielding a sequence shape that can be efficiently discriminated from the distribution of these sequences characterized over legitimate images. In other words, the boundary of legitimate samples circumscribed by the one-class classifier is determined by the variation of their color content.

Algorithm 6.2 Proposed method: *prediction*

Require: Image $\mathbf{x} = [\mathbf{x}^R, \mathbf{x}^G, \mathbf{x}^B]$ to be predicted, discriminators $f_{oc}^R(\mathbf{x}^R)$, $f_{oc}^G(\mathbf{x}^G)$ and $f_{oc}^R(\mathbf{x}^B)$ trained as per Algorithm 6.1, classifier $f(\mathbf{x};\Theta)$.

1: **for** $c \in \{R, G, B\}$ **do**
2: | Let $\mathbf{e}^c = \{e_m^c\}_{m=1}^M$ be the edge count sequence of channel image $\mathbf{x}^c$
3: | **for** $m \in \{1, \ldots, M\}$ **do**
4: | | Process channel image with edge detector: $\mathbf{x}_{\tau_m}^{c,edge} = \texttt{Edge}_{\tau_m}(\mathbf{x}^c)$
5: | | Set e_m^c to the number of edge pixels in $\mathbf{x}_{\tau_m}^{c,edge}$
6: | **end for**
7: | Compute $f_{oc}^c(\mathbf{e}^c)$
8: **end for**
9: **if** $f_{oc}^R(\mathbf{e}^R) = f_{oc}^G(\mathbf{e}^G) = f_{oc}^B(\mathbf{e}^B) = \texttt{legitimate}$ **then**
10: **return** $\hat{l}$ the label predicted by $f(\mathbf{x};\Theta)$ classifier
11: **else**
12: **return** $\hat{l} = \texttt{adv}$ ($\mathbf{x}$ is not legitimate)
13: **end if**

Once a one-class classifier has been trained for every channel $c \in \{R, G, B\}$ (line **19** in Algorithm 6.1), the proposed method is ready to declare whether a newly arriving test sample $\mathbf{x}'$ is adversarial ($l' = \texttt{adv}$) or, instead, must be regarded as legitimate and, thus, subsequently assigned a predicted label $l' \in \{1, \ldots, L\}$ by the trained classifier. This process is summarized in Algorithm 6.2: each channel of the image to be predicted is first decomposed in its three channel images (line **1**), processed through the edge detector over the sensitivity value range (lines **2** to **6**), and then processed through the three discriminators $f_{oc}^R(\cdot)$, $f_{oc}^G(\cdot)$ and $f_{oc}^B(\cdot)$ (line

7). The outputs of the three discriminators must be aggregated to deliver a consensus on the legitimacy of the input image. For this purpose, we will declare an input image as adversarial if the discriminator of at least one channel predicts that the channel at hand has been altered with respect to its training samples (lines **9** to **12**). Other aggregation functions can be used instead for this same purpose, e.g., a majority voting between the outputs of the discriminators. In all cases, the aggregated output of the three one-class classifiers indicates if the image at hand is legitimate and can be safely predicted by the trained classifier.

6.4 Experimental Setup

In this section we describe the experiments used to evaluate the performance of the proposed method. The overall objective of our experimentation is to assess the robustness of our method to adversarial samples. To this end, we focus our analysis on the effect of the discriminator module in the overall performance of the proposed approach. Specifically, we attack a DNN classifier with adversarial samples crafted by means of the canonical methods introduced in Section 2, assuming in all cases a white-box scenario (i.e., the attacker is entirely aware of the structure and trained parameter values of the classifier to be attacked). Once adversarial examples have been furnished, we compute the attack detection rate of our proposed discriminator, as well as the success rate of the attack when aiming to confuse the target model $f(\mathbf{x};\boldsymbol{\Theta})$. We enrich the experimentation by considering two different adversarial scenarios depending on the final objective of the attacker (targeted or untargeted evasion):

1. *Targeted scenario*, where the goal is to force specific classification errors when predicting test samples towards selected labels of interest.

2. *Untargeted scenario* where the goal is to decrease its confidence by causing indiscriminate classification errors for test samples.

Experiments in this chapter consider three image classification datasets widely used in the related literature and made publicly available for the community:

- The `MNIST` [126] dataset, consisting of $K = |\mathcal{D}| = 70,000$ instances that are split into two subsets, the training set with $K_{\text{train}} = 60000$ samples and the test set with $K_{\text{train}} = 10000$ samples). Samples are gray-scale 28×28 images of handwritten digits annotated with the digit itself, i.e., $L = 10$.

- The Street View House Numbers (`SVHN` [127]) dataset, whose inputs $\mathbf{x}_k$ represent 32×32 RGB images collected by Google Street View, depicting cropped house numbers, with $K = 99,289$ samples labeled with $L = 10$ classes (in this case, $K_{\text{train}} = 73257$ and $K_{\text{test}} = 26032$).

- The German Traffic Sign Recognition Benchmark (`GTSRB` [128]), which comprises $50,000$ images of $L = 43$ different traffic signs captured at

varying image sizes. In this last case, we have considered images with a minimum size of 40×40 pixels, resulting in $K = 4000$ RGB images. We have upscaled or downscaled their resolution to make them all of equal size: 128×128 pixels.

In order to build the classifier, the discriminator and the adversarial samples, each database has been partitioned into three mutually exclusive subsets, namely, train, test and crafting subsets (see Table 6.1). Train and test subsets contain samples from the training set of the source database, which are used to respectively train and evaluate the performance of the DNN and OSVM models. The crafting subset is forged from the test set of the source dataset to generate adversarials. All subsets except for the latter have been drawn by following a stratified random sampling technique. To create the crafting subset, we have decreased the sample rate of `MNIST` and `SVHN` labels to 25 images per class (5 in the case of the `GTSRB` dataset), so as to reduce the time required to conduct the experimentation.

TABLE 6.1: Description and subset splits of the considered datasets.

Dataset $\mathcal{D}$		Train	Test	Crafting
Name	**# labels (L)**			
`MNIST`	10	57000	3000	250
`SVHN`	10	65930	7327	250
`GTSRB`	43	19171	1725	215

We now describe the learning procedures utilized to build the target classifier that the attacker aims to confuse, as well as the proposed method to discriminate adversarial samples. Given that we focus on image classification, the target classifier is chosen to be a DNN model. The architecture of the target DNN classifier is summarized in Table 6.2. As shown in this table, the network is made up of 3 convolutional layers, and a fully-connected layer to allow for multi-class classification. Parameters of the DNN are learned through backpropagation using Adam optimizer with learning rate equal to 0.001, batches of 200 training samples and 13 epochs. For the `GTSRB` dataset, an additional dense layer is placed in-between before the output SoftMax layer, and the number of epochs is increased to 15. It is important to note that these configurations replicate the ones used recently in [74], where the DkNN defense method against adversarial attacks for image classification mentioned in Section 6.2.2 was presented.

Regarding the discrimination module, the OSVM classifier uses the negated time warping kernel function $K_{\mathrm{NDTW}}(\mathbf{e}, \mathbf{e}')$ defined in Expression (6.1). In this case, $\mathbf{e}$ and $\mathbf{e}'$ represent the edge count sequences of two different channel images $\mathbf{x}^c$ and $\mathbf{x}'^c$ respectively. Thus, each element $e_m \in \mathbf{e}$ represents the edge count for a value of the sensitivity parameter $\tau_m \in [\tau_1, \ldots, \tau_M]$ (lines **3** to **14** in Algorithm 6.1). In our experiments, a Canny edge detection algorithm is adopted as $\mathtt{Edge}_{\tau_m}(\mathbf{x}^c)$ (lines **10** and **11** in Algorithm 6.1) to extract such edge information, using $M = 256$ sensitivity values $\{\tau_1, \ldots, \tau_M\} = \{0, \ldots, 255\}$ that represent the *high-level threshold* of the algorithm (the *low-level* threshold is always set to 0). In

TABLE 6.2: DNN architecture for the target model.

Layer Type	Description	Configuration
2D Convolutional	64 filters (8×8) stride $(2,2)$, *same*, ReLu	Categorical crossentropy loss Adam optimizer Learning rate 0.001 Batch size 200 13 epochs (`MNIST`, `SVHN`) 15 epochs (`GTSRB`)
2D Convolutional	128 filters (6×6) stride $(2,2)$, *valid*, ReLu	
2D Convolutional	128 filters (5×5), stride $(1,1)$, *valid*, ReLu	
Linear*	200 units*	
Linear	L units (L: number of classes)	

*: only for the `GTSRB` dataset.

this way, by changing τ_m, one obtains edge maps at different levels of sensitivity: the higher τ_m is, the more restrictive the Canny detector will be to declare a pixel as an edge. The OSVM variant in [124] is further controlled by a parameter $\nu \in (0, 1]$ that denotes the fraction of training samples allowed to be outside the estimated region. The value of this parameter is selected by searching over a fine-grained grid of possible ν values, from which we choose the one leading to better inlier characterization statistics (i.e., maximum fraction of legitimate samples detected as such by the discriminator) measured over a validation holdout from the training set.

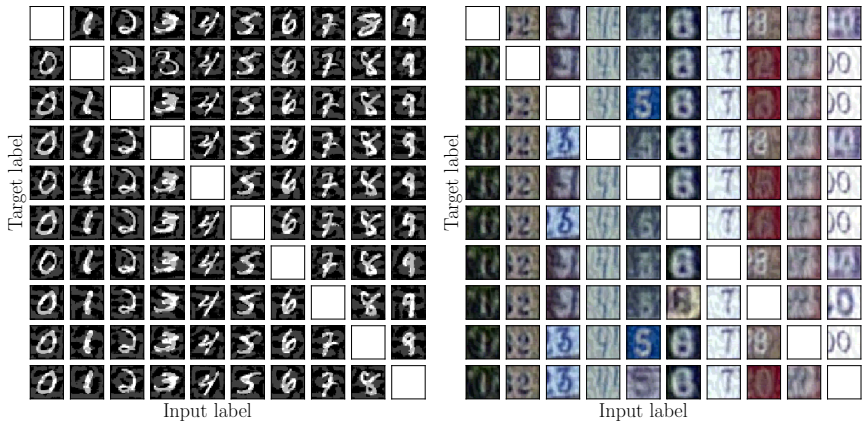
TABLE 6.3: Detection statistics of the discriminator and classification accuracy of the DNN model for the considered datasets.

Dataset $\mathcal{D}$	**DNN** (accuracy)	**OSVM** (% positive inliers)	**OSVM+DNN**
`MNIST`	98.1%	100.0% ($\nu = 0.01$)	98.1%
`SVHN`	87.4%	81.0% ($\nu = 0.12$)	77.7%
`GTSRB`	91.9%	89.3% ($\nu = 0.01$)	83.6%

Once the discrimination model is trained as per Algorithm 6.1, the evaluation of the performance of the overall image classification model is provided by testing the model over an unseen 10% set of samples selected at random from $\mathcal{D}$, and not used for training the OSVM and the DNN models. Table 6.3 reports the performance of the DNN classifier (accuracy), the OSVM discriminator (detection rate), and their combination OSVM+DNN. In the latter case, we quantify the ratio of test images that were declared to be legitimate by the discriminator, and whose label was correctly predicted by the classifier. It can be observed in this table that the inclusion of the discriminator degrades the accuracy of the overall scheme. However, it provides the attacked classifier with a significantly enhanced robustness against adversarial attacks, as the results discussed in the following section clearly show.

6.5 Results and Discussion

In this section we evaluate the results of the proposed defense approach when being attacked with slight-color variation based adversarial crafting techniques described in Section 6.1. Furthermore, we consider targeted and untargeted attacks, yielding four adversarial sample sets that have been built, for all datasets, by selecting a number of legitimate images per class label drawn from $\mathcal{D}$. As shown in Table 6.1, 25 samples per label are selected for the `MNIST` and `SVHN` datasets (in total, 250 samples per dataset), whereas for `GTSRB` this number is reduced down to 5 samples per label (respectively, 215 samples) due to the increased computational burden of processing a larger number of classes. These selected samples are exclusively used for furnishing the attacks, hence they are not considered for training the DNN classifier nor fed to the OSVM discriminator.



(A) `MNIST` (B) `SVHN`

FIGURE 6.1: Example of adversarial examples furnished by BIM that manage to confuse the DNN towards targeted misclassified labels. For the `MNIST` dataset, parameters have been set to $T_{\max} = 100$, $\alpha = 0.01$ and $\varepsilon = 0.01$, whereas for the `SVHN` dataset, $T_{\max} = 20$, $\alpha = 0.005$ and $\varepsilon = 0.05$.

An illustrative subset of the adversarial examples crafted by BIM that succeed at confusing the DNN classifier is illustrated in Figures 6.1.a (for the `MNIST` dataset) and 6.1.b (for the `SVHN` dataset). In these plots, adversarial images are arranged in an $L \times L$ matrix such that the example in the (l', l) position represents a legitimate image with an original label l, manipulated by BIM to make the classifier predict its class as $l' \neq l$. For instance, the image at position $(5, 0)$ in Figure 6.1.a corresponds to a legitimate image of a 0 digit that has been modified to be classified as a 5 by the DNN model. It is straightforward to note that in the `MNIST` dataset, the perturbation made by the adversarial attack gives rise to gray areas over the black background of the image that maintain the shape information of the digit in the image. In contrast, the richer RGB color space of the `SVHN` database (Figure 6.1.b) makes it more difficult to visually discern the attack inserted by BIM in the image. This observation is at the heart

of the design of our proposed defense method (to rely on the detection of edges in the channels of the image at different resolutions), for which we next provide further empirical intuition.

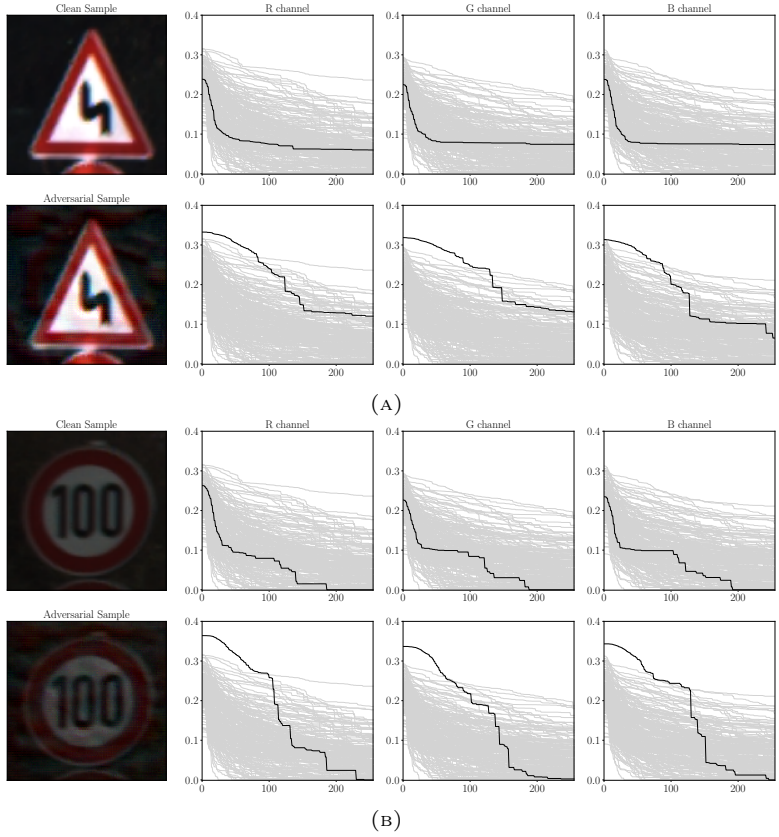


FIGURE 6.2: Sequences extracted from two clean images of the SVHN database and those of their adversarial versions produced by BIM with $T_{\max} = 20$, $\alpha = 0.005$ and $\varepsilon = 0.10$. In both cases, adversarial samples are crafted to force the DNN to classify the original images (double curve and 100 kmph speed limit traffic signs) as a 20 km/h speed limit sign. Gray lines in R, G and B channel plots represent the sequences $\mathbf{e}_n^c = \{e_{n,m}^c\}_{m=1}^M$ used to train the OSVM, whereas the black line corresponds to the sequence of the traffic sign image.

We now focus on Figure 6.2a, where we visualize the output edge count sequences $\mathbf{e}_n^c = \{e_{n,m}^c\}_{m=1}^M$ for the three RGB channels of an original image retrieved from the SVHN database, and that of its adversarial version. As previously indicated, edges have been extracted for different threshold values $\{\tau_m\}_{m=1}^{256}$ of a Canny detector. These plots also depict, shaded in gray, the total of sequences corresponding to the training set of this database, which serves as a reference of the distribution of sequences for legitimate images. It is important to observe that even if the adversarial image closely resembles the original image, modifications made by BIM on the original sample imprint a shape change of the edge count sequence in all three

channels. It is indeed this change that the discriminator aims to detect. Furthermore, the elasticity provided by the utilized DTW-based kernel allows accommodating the relative variability of sequences corresponding to legitimate images over the x-axis, roughly in a similar fashion to what occurs with this similarity measure over time series with non-linear warps over the time domain. The second example in Figure 6.2b also supports this observation by again demonstrating that the edge count sequence of the adversarial attack has a different shape of the attack when compared to that of its legitimate (clean) version. In both depicted examples, the OSVM discriminator, once trained with the sequences in gray, is able to detect the presence of an adversarial attack in the image.

Once the discriminator is trained with the edge count sequences of legitimate examples, the DNN classifier can use it as a preprocessor to decide whether a given image $\mathbf{x}$ is adversarial (predicted label $\hat{l} = \texttt{adv}$) or legitimate. In the latter, the predicted label for the image is given by the output of the DNN classifier, i.e., $\hat{l} = f(\mathbf{x}; \Theta)$. To quantitatively assess the effectiveness of the combined OSVM+DNN approach, we compute the ratio of adversarial images – out of the 250 (`MNIST`, `SVHN`) or 215 samples (`GTSRB`) selected from $\mathcal{D}$ for crafting attacks – that succeed at confusing the attacked model. We compute this *attack success* rate aggregated over all classes of each database and for both targeted and untargeted scenarios, meaning that, for the targeted scenario, only attacks that produce the specifically desired misclassification error are considered a success. Furthermore, results are reported for the undefended DNN classifier, the OSVM discriminator, and the (OSVM+DNN) combination. In the case of the discriminator, the attack success is defined as the ratio of adversarial images that are not detected as such by the trained OSVM model. Finally, we also consider different ε values (in accordance with those used in [74] for attacking these datasets), so that we can also assess how the intensity of attack impacts on its detectability by the proposed discriminator.

The obtained success rates are summarized in Table 6.4. As expected, for both FGSM and BIM attacks, the success rate over the DNN always increases with ε, due to the fact that this parameter defines the amplitude of the perturbation of the attack as per Equations (6.9), (6.10) and (6.11). On the contrary, the OSVM discriminator reports lower success rates of the attack as ε increases, since attacks are easier to detect despite being more effective in achieving its goal to confuse the DNN classifier. Adversarial samples produced by large ε values are more likely to mislead the DNN, but easier to detect for a human observer. Conversely, subtle attacks may not be visually detectable by the observer, but might fail to confuse the classifier due to the low intensity of the adversarial attack. When combining the proposed discriminator and the DNN classifier, near-zero success rates are obtained by the attacker against the combined OSVM+DNN approach for both targeted and untargeted scenarios. In particular, the success rates when attacking the OSVM+DNN scheme for the `MNIST` and `GTSRB` databases are close to zero for all considered cases, i.e., no adversarial attack is able to confuse the DNN classifier, even for ε values that yielded very high attack success rates for the undefended

DNN classifier. For instance, in the GTSRB dataset more than 98% of the crafted adversarial images by BIM with $\varepsilon = 0.10$ confuse the DNN model towards untargeted labels; when including our proposed discriminator, all such attacks are detected, lowering the success rate of the attacker down to 0.00%. When turning the focus to the SVHN database, similarly good detection scores are observed for the OSVM+DNN approach, yet attack success rates are not as low as the other two databases considered in our benchmark. The reason being that images in this database are less defined and noisier than those in the other two databases, thereby making it more difficult to properly characterize the space of sequences spanned by legitimate images.

TABLE 6.4: Success rate (%) of the attacker against DNN / OSVM / OSVM+DNN. In each cell, the results show the success of adversarial examples when crafted to confuse the DNN model by using some of the techniques described in Section 6.2.1.

$\mathcal{D}$	Technique	Parameters	Targeted	Untargeted
MNIST	FGSM	$\varepsilon = 0.10$	7.23 / 0.00 / 0.00	34.94 / 0.00 / 0.00
		$\varepsilon = 0.25$	49.13 / 0.00 / 0.00	91.97 / 0.00 / 0.00
		$\varepsilon = 0.30$	53.95 / 0.00 / 0.00	94.78 / 0.00 / 0.00
	BIM $T_{\max} = 100$ $\alpha = 0.01$	$\varepsilon = 0.10$	10.84 / 0.00 / 0.00	44.98 / 0.00 / 0.00
		$\varepsilon = 0.25$	94.87 / 0.00 / 0.00	100.00 / 0.00 / 0.00
		$\varepsilon = 0.30$	99.78 / 0.00 / 0.00	100.00 / 0.00 / 0.00
SVHN	FGSM	$\varepsilon = 0.02$	32.83 / 89.18 / 28.08	73.42 / 86.94 / 63.06
		$\varepsilon = 0.05$	60.46 / 13.71 / 5.46	95.50 / 13.96 / 11.71
		$\varepsilon = 0.07$	63.81 / 2.05 / 1.00	97.30 / 2.70 / 2.25
	BIM $T_{\max} = 20$ $\alpha = 0.005$	$\varepsilon = 0.02$	51.20 / 90.24 / 45.20	79.73 / 91.44 / 72.07
		$\varepsilon = 0.05$	93.59 / 24.57 / 20.42	98.65 / 22.52 / 21.62
		$\varepsilon = 0.07$	98.55 / 6.71 / 6.01	100.00 / 4.50 / 4.50
GTSRB	FGSM	$\varepsilon = 0.05$	55.84 / 3.61 / 0.91	84.58 / 3.48 / 2.99
		$\varepsilon = 0.10$	76.36 / 0.13 / 0.07	94.53 / 0.00 / 0.00
		$\varepsilon = 0.15$	81.20 / 0.00 / 0.00	96.52 / 0.00 / 0.00
	BIM $T_{\max} = 20$ $\alpha = 0.005$	$\varepsilon = 0.05$	72.59 / 2.12 / 0.72	95.02 / 2.99 / 2.99
		$\varepsilon = 0.10$	91.44 / 0.00 / 0.00	98.51 / 0.00 / 0.00
		$\varepsilon = 0.15$	91.44 / 0.00 / 0.00	98.51 / 0.00 / 0.00

Once the performance of the proposed method has been analyzed with respect to the applied color variations (i.e., based on the maximum allowed perturbation ε of FGSM and BIM attacks), we now proceed by discussing on a benchmark where we compare the performance of the proposed discriminator for untargeted attacks with respect to the benchmark techniques introduced in Subsection 6.2.2. The experimentation next discussed uses the IBM Adversarial Robustness Toolbox [129] for the Label Smoothing, JPEG Compression, Feature Squeezing, Spatial Smoothing, Total Variance Minimization and Thermometer Encoding defenses. Regarding DkNN, we have utilized the implementation of this defense strategy provided in the Cleverhans library [130]. Furthermore, this set of experiments also considers an extended range of adversarial attacks. Apart

from the FGSM and BIM methods used in the first set of experiments, PGD, JSMA and C&W attacks have been also included in the benchmark.

Results for this benchmark are summarized in Table 6.5. For each dataset-attack-defense combination *defender accuracies* are reported, which denotes the ratio of adversarial samples that do not succeed in confusing the DNN model. In our case, we have considered that the attack fails when 1) an adversarial sample is detected as such by the discriminator; or 2) when an adversarial image is declared as legitimate by the discriminator, but does not yield a different label predicted by the DNN model than its original *clean* version. A grid search has been performed to tune the parameters for the attack and defense strategies. For every attack and dataset, the chosen parameters are those maximizing the efficiency of the attack over the unprotected DNN model. Parameters for the defense strategies are set to those maximizing its adversarial detection accuracy.

TABLE 6.5: Defender accuracy (%) against untargeted adversarial images when using attack and defense strategies from the related literature, and the proposed OSVM+DNN approach.

$\mathcal{D}$	Attack	DNN (unprotected)	Feature Squeezing	Spatial Smoothing	JPEG Compression	Label Smoothing	Thermometer Encoding	Total Variation	DkNN	OSVM+DNN (proposed)
MNIST	FGSM	9.03	9.03	14.85	18.87	22.49	31.72	23.27	40.96	**100.00**
	BIM	0.00	0.00	6.42	5.62	10.44	17.67	17.67	27.31	**100.00**
	PGD	0.80	0.80	16.06	3.61	4.01	26.90	23.29	23.69	**100.00**
	JSMA	0.00	55.82	50.20	68.27	76.70	87.95	69.47	**92.77**	43.37
	C&W	0.00	75.90	76.30	81.12	93.57	96.38	84.73	95.98	**98.79**
SVHN	FGSM	4.50	8.55	24.32	21.17	26.12	28.37	43.69	30.63	**88.29**
	BIM	1.35	3.60	15.31	17.56	14.41	19.36	40.09	24.32	**78.38**
	PGD	4.50	4.50	12.61	8.11	11.26	18.01	4.95	4.50	**36.48**
	JSMA	0.01	44.14	68.01	64.84	73.42	78.33	54.50	**85.13**	2.25
	C&W	0.00	44.14	60.81	74.77	86.48	75.22	57.65	**90.54**	2.70
GTSRB	FGSM	5.47	6.46	10.94	36.31	36.81	22.52	48.25	54.22	**100.00**
	BIM	1.49	2.48	4.47	18.01	40.09	13.06	45.04	38.85	**100.00**
	PGD	5.47	5.47	6.46	5.97	5.97	13.43	6.96	29.85	**82.08**
	JSMA	0.00	37.31	74.12	69.65	71.14	78.60	78.10	**82.08**	2.48
	C&W	0.04	65.17	77.61	79.10	82.58	83.58	79.60	**86.06**	7.46

This being said, Table 6.5 illustrates that DkNN and OSVM+DNN approaches yield a notably superior defender accuracy in comparison to other benchmark models. More precisely, for the three databases under consideration, the reported scores elucidate that our model clearly dominates the benchmark when detecting FGSM, BIM and PGD attacks. However, it fails at detecting adversarial samples crafted with the JSMA and C&W attack methods. The reasons for this performance degradation can be discerned if we visually inspect adversarial samples crafted by each of the considered attack strategies. Therefore, we shift the focus of the discussion towards Figure 6.3, where the effect of the considered attack strategies is exemplified for a 0 digit of the `MNIST` database. Interestingly, we observe that the proposed OSVM+DNN model has more accurate results when the color perturbations spread out along the whole surface of the

image (FGSM, BIM and PGD). In contrast, the DkNN appear to be more efficient in detecting adversarial samples in which color variations concentrate around already existing edges or modify the shape/forms of the input image (JSMA and C&W). Such perturbations do not produce significant differences between edge count sequences for different sensibilities, thereby lowering the capability of our OSVM+DNN approach to discern between legitimate and adversarial images.

(A) FGSM (B) BIM (C) PGD (D) JSMA (E) C&W

FIGURE 6.3: Adversarial `MNIST` examples crafted by different attack strategies.

These findings motivate future directions around improving the detection performance of the proposed approach for adversarial attacks of this nature. We will elaborate further on these research lines in Section 6.6. However, before concluding the discussion it is important to gauge these comparative results jointly with the universality of our approach with respect to the image classifier and attack technique in use. DkNN is specifically designed for DNN image classifiers, as it essentially operates by inspecting the internal activations of the DNN layers to decide whether an image to be predicted is legitimate or adversarial. This fact underscores one of the main advantages of the scheme presented in this chapter: the proposed discriminator exclusively observes information contained in the image, without any consideration of the image classification model in use.

6.6 Conclusions and Future Work

This chapter proposes a novel method to make image classification models robust against adversarial samples crafted by an attacker to confuse their predicted labels. Unlike other techniques reported in the literature for this purpose, our design builds upon the proven capacity of the human cortex system to better discriminate shapes than slight color variations. This fact suggests that adversarial attacks could be better detected if color variations imprinted by such attacks are decomposed in an alternative space, emphasizing changes in the distribution of edges over each color channel of the image under analysis. We have elaborated on this research hypothesis to build a one-class classifier that discriminates adversarial and legitimate images by analyzing them over a sequence of edge counts of the color channels of the image, under different levels of edge detection sensitivity. This approach allows translating color variations in the image to shape changes of the edge count sequences extracted from the image. Furthermore, by concentrating on the effects of the attack on the color distribution of the image, our defense method is independent of the specific

image classifier under attack and the technique used to craft the adversarial image.

In particular, in order to build the discriminator, a one-class SVM classifier is learned from a kernel based on the DTW for the edge count sequences. By doing so, the classifier focuses the discrimination between legitimate and adversarial samples strictly on shape differences in their edge count sequences. This particular kernel choice permits the false positives to be minimized (i.e., legitimate images identified as an adversarial by the discriminator) that could result from varying color intensities and other image artifacts impacting on the distribution of edges over the image.

Experimental results obtained for three different image classification dabases have shed light on the increased robustness against adversarial endowed by our proposed approach. We have found that the OSVM discriminator built on edge-based information from images is capable of discriminating most adversarial samples crafted from FGSM, BIM and PGD. The color perturbations imprinted by these attack strategies cover the whole surface of the image. Regarding the detection of adversarials generated by JSMA and C&W, our model draws poor performance results, compared with state-of-the-art strategies in `SVHN` and `GTSRB`, due to the different nature of the perturbations made by these attacks. The color changes derived from such perturbations concentrate on localized regions rather than over the whole image, lessening their impact on the edge count sequences on which our scheme is based.

Several research avenues stem from the findings reported in this chapter. To begin with, the proposed procedure can be improved by considering other color-content based feature vectors. In particular, we propose to develop of new features that i) can reveal the modifications applied around original abrupt color discontinuities and/or ii) can be sensitive to large color variations applied in small rate of image pixels. Moreover, we plan to investigate how to extrapolate our insights gained on the impact of adversarial attacks on the characteristics of the images to deal with other kind of attacks not contemplated in this study (e.g., one-pixel attacks [131]). To this end, a great deal of focus will be placed on understanding the reasons why such adversarial attacks eventually succeed in confusing DNN models, analyzing image features that could possibly correlate with the adversarial attack. Efforts will be also devoted towards considering the more restrictive scenario of steganography [132], [133], where in addition to misleading a classifier, the adversarial attack must also embed payload information that must be extracted after classification.

Chapter 7

Adversarial Sample Crafting for Time Series Classification

In Chapter 6 we have seen that a distance-based defense strategy can make DNNs classifiers for image data robust against adversarial attacks. However, in many practical scenarios it is the case that the data itself is composed by labeled sequences, wherein the problem is to decide which label to assign to a new given test sequence. For instance, the problem of time series classification posed in Section 1.2.2 refers to this problem when sequences are time series. When this is the case, the use of ESM measures (in particular, DTW) with NN classifiers has been shown to yield a superior performance in problems where the correspondence between time series and labels require a given degree of independence of the similarity with respect to non-linear transformations (warps) in the time domain.

Unfortunately, the outstanding performance of DTW-NN classifiers over such problems comes along with similar concerns with their susceptibility to adversarial attacks than those noted for DNN classifiers. Furthermore, it is unclear whether attack strategies originally developed for DNN classifiers can be extrapolated to DTW-NN classifiers. The contribution of this chapter contributes precisely to this research gap by proposing novel attack strategies against DTW-NN classifiers. Through the forthcoming sections we will reformulate the AML problem from Expression (6.3) for this particular class of classification models. Once the AML problem for DTW-based NN classifiers has been posed, we will propose three different attack strategies, giving an empirically informed insight on their success rate and transferability among different data splits.

7.1 Introduction and Related Work

As mentioned above, when dealing with sequences ESMs allow for non-linear warps over time, so that the computation of the similarity of two time series becomes independent of local time shifts. This property allows classifiers relying on ESMs to perform better in applications where time warps and lags between time series do not imply that their class labels

differ from each other. However, as occurs with DNN classifiers, the good predictive performance of DTW-based NN do not exempt them from being sensitive to adversarial attacks, aimed at changing their prediction. For the sake of self-completeness of this chapter, we briefly revisit recently proposed techniques for adversarial sample crafting from a general perspective. We refer to Chapter 6, where Section 6.2.1 provides a comprehensive introduction to the AML problem, along with a detailed introduction to more benchmark attacks and defense strategies.

This being said, we recall the assumptions and notation from Section 6.2.1 to consider a training set $\mathcal{D} = \{(\mathbf{x}_k, l_k)\}_{k=1}^{K}$, where $\mathbf{x}_k \in \mathcal{X}$ is the k-th training example, and $l_k \in \{1, \ldots, L\}$ its corresponding class label. As mentioned in Chapter 1, a classification problem consists of learning a function $f : \mathcal{X} \rightarrow \{1, \ldots, L\}$ from $\mathcal{D}$ that predicts the class label $l \in \{1, \ldots, L\}$ of any unseen input example $\mathbf{x} \in \mathcal{X}$. In this context, the goal of an adversarial attack is to generate a perturbed version of a legitimate example $\mathbf{x}$ that is classified wrongly by classifier f. The perturbed sample is produced as $\widetilde{\mathbf{x}} = \mathbf{x} + \boldsymbol{\delta}_{\text{adv}}$, where the perturbation vector $\boldsymbol{\delta}_{\text{adv}}$ yields from:

$$\boldsymbol{\delta}_{\text{adv}} = \arg\min_{\boldsymbol{\delta}} \Psi(\mathbf{x}, \mathbf{x} + \boldsymbol{\delta}), \text{ subject to } f(\mathbf{x}) \neq f(\mathbf{x} + \boldsymbol{\delta}), \tag{7.1}$$

where $\boldsymbol{\delta}$ is a vector with the same dimension than $\mathbf{x}$, and $\Psi(\mathbf{x}, \mathbf{x}')$ is a measure of dissimilarity between sequences $\mathbf{x}$ and $\mathbf{x}'$. A common measure of dissimilarity used when $\mathbf{x}$ and $\mathbf{x}'$ represent sequence data is the L_p-norm distance $d_p(\mathbf{x}, \mathbf{x}')$. Since solving (7.1) is not trivial, Szegedy et al. [72] proposed to reformulate the above problem for L_2-norm distance, and to solve it by means of a L-BFGS solver, yielding:

$$\boldsymbol{\delta}_{\text{adv}} = \arg\min_{\boldsymbol{\delta}} \; \varepsilon \cdot d_2(\mathbf{x}, \mathbf{x} + \boldsymbol{\delta}) + C_f(\mathbf{x} + \boldsymbol{\delta}, l), \tag{7.2}$$

where $l = f(\mathbf{x})$ (i.e. the true label of $\mathbf{x}$), $C_f(\mathbf{x}, y)$ is the cost (loss) function used to learn f from $\mathcal{D}$ (e.g. Mean Square Error in a linear regressor), and $\varepsilon > 0$ is a constant set to ensure that $d_2(\mathbf{x}, \widetilde{\mathbf{x}}) = ||\widetilde{\mathbf{x}} - \mathbf{x}||_2 \leq \varepsilon$. Line search strategy was proposed to infer the optimal value of ε.

Instead of solving (7.2) to optimality, Goodfellow et al. [113] proposed an alternative technique to craft adversarial samples, *Fast Gradient Sign Method* (FGSM), which was already explained in Section 6.2.1. Given a value of ε fixed beforehand, the distortion is given by:

$$\boldsymbol{\delta}_{\text{adv}} = \varepsilon \cdot \texttt{sign}\left[\nabla_{\mathbf{x}} C_f(\mathbf{x}, l_{\otimes})\right], \tag{7.3}$$

where $\texttt{sign}[\cdot]$ denotes the sign function, $l_{\otimes} \neq f(\mathbf{x})$ is the target class to which the adversarial sample $\widetilde{x}$ is to be misclassified, and $\nabla_{\mathbf{x}}$ is the gradient with respect to $\mathbf{x}$. In this approach the distortion is upper bounded under the L_∞-norm, that is, $||\widetilde{\mathbf{x}} - \mathbf{x}||_\infty \leq \varepsilon$. It is important to observe that the application of FGSM only ensures that a solution to the problem in (6.3) is provided if a proper selection of ε is made. In this same line of work, in [134] a fast gradient strategy was proposed for $p = 2$ (i.e., to ensure the

distortion meets $||\widetilde{\mathbf{x}} - \mathbf{x}||_2 \leq \varepsilon$). In this case, the distortion is:

$$\boldsymbol{\delta}_{\text{adv}} = \varepsilon \cdot \frac{\nabla_{\mathbf{x}} C_f(\mathbf{x}, l_{\circledast})}{||\nabla_{\mathbf{x}} C_f(\mathbf{x}, l_{\circledast})||_2}. \tag{7.4}$$

More recently, the work in [114] proposed an iterative version of the FGSM technique, which computes the adversarial sample recurrently as:

$$\widetilde{\mathbf{x}}^{(t+1)} = \widetilde{\mathbf{x}}^{(t)} + \boldsymbol{\delta}_{\text{adv}}(\widetilde{\mathbf{x}}^{(t)}), \tag{7.5}$$

where $t \in \{1, \ldots, T_{\max}\}$, $\widetilde{\mathbf{x}}^{(1)} = \mathbf{x}$, and:

$$\boldsymbol{\delta}_{\text{adv}}(\widetilde{\mathbf{x}}^{(t)}) = \alpha \cdot \texttt{sign}\left[\nabla_{\mathbf{x}} C_f(\widetilde{\mathbf{x}}^{(t)}, y_{\circledast})\right]. \tag{7.6}$$

In order to ensure that adversarial samples satisfy that $||\widetilde{\mathbf{x}} - \mathbf{x}||_\infty$ is bounded by ε, α is set to $\varepsilon / T_{\max}$. Similarly, for the distortion to meet $||\widetilde{\mathbf{x}} - \mathbf{x}||_2 \leq \varepsilon$, it must be computed as:

$$\boldsymbol{\delta}_{\text{adv}}(\widetilde{\mathbf{x}}^{(t)}) = \alpha \frac{\nabla_{\mathbf{x}} C_f(\widetilde{\mathbf{x}}^{(t)}, y_{\circledast})}{||\nabla_{\mathbf{x}} C_f(\widetilde{\mathbf{x}}^{(t)}, y_{\circledast})||_2}, \tag{7.7}$$

which also depends on ε and $T_{\max}$ to achieve a feasible solution to the problem in (7.1), similarly to what was previously noted for FGSM.

Unfortunately, the properties of ESMs make it not straightforward to extrapolate the above adversarial sample crafting strategies to ESM based models. There emerges the twofold contribution of this chapter: (i) to extend the formulation of an adversarial machine learning problem to DTW-based time series classifiers, and (ii) to propose novel strategies to produce adversarial time series capable of confusing DTW based NN models. For this specific similarity measure, we derive three approaches to construct such adversarial samples: the first two hinge on the adaptation of the gradient-based attack strategies reported previously for DNN models and image data, for which we develop an expression for the gradient over the distance space spanned by the DTW. The last strategy aims at circumventing the susceptibility of gradient-based strategies to its parametric configuration (i.e. gradient step size) by formulating the sample crafting process as a bi-objective optimization problem. This problem is subsequently tackled by using multi-objective heuristics. Computer experiments are performed with databases from the UCR repository [92] to validate the effectiveness of our proposed strategies when producing adversarial time series, as well as the capability of the produced samples to confuse models constructed on databases different than the one utilized to build the attack.

The remainder of this chapter is structured as follows: Section 7.2 and subsections therein describe the proposed strategies to construct adversarial time series for DTW based classifiers. Section 7.3 presents and discusses the obtained results from the performed computer experiments. Finally, Section 7.4 ends the chapter by drawing concluding remarks and identifying future research directions.

7.2 Attacking DTW-based NN Classifiers

Our main design goal is to generate adversarial samples capable of misleading DTW based NN classifiers. To this end we adapt the adversarial sample crafting techniques reviewed in Section 7.1 to the DTW. At this point it is important to highlight that these methods were originally aimed at confusing DNNs designed for image classification. In such models, $C_f(\mathbf{x}, y)$ is selected to be, in most cases, a cross-entropy loss function. Given its differentiability, the application of gradient-based techniques is straightforward. However, NN is a lazy classifier (it requires no training phase), and has neither an associated model f nor a loss function $C_f(\mathbf{x}, y)$.

Following the workaround used by Papernot et al. in [112], we resort to a smoothed, differentiable approximation of the NN classifier. In this manner we can proceed forward with the reformulation of the adversarial sample crafting problem in Equation (7.1) with the DTW similarity measure. That is, let us assume that the class of a time series U^n must be predicted under the DTW similarity measure. For this purpose, the NN classifier uses a training set of the form $\mathcal{U} = \{(U_k, l_k)\}_{k=1}^{K}$, where $U_k = (u_1^k, ..., u_n^k)$ represents a training (reference) time series, and $l_k \in \{1, ..., L\}$ its associated class label. We consider the square Euclidean distance as the cost function for the DTW computation, i.e. the similarity between $X \in \mathbb{R}^m$ and $Y \in \mathbb{R}^n$ is given by $\mathrm{DTW}(X, Y) = \sqrt{\gamma_{m,n}}$, where $\gamma_{m,n} = (x_m - y_n)^2 + \min\left\{\gamma_{m-1,n}, \gamma_{m-1,n-1}, \gamma_{m,n-1}\right\}$.

When considering a NN classifier, instead of computing the minimum distance among the query and every training time series, we can define a *conditional probability* for a given class $l \in \{1, ..., L\}$ as:

$$\mathrm{P}(l|U) = \frac{1}{K} \sum_{k=1}^{K} \Gamma_{l_k,l} \cdot \sigma(U, U_k), \tag{7.8}$$

where $\Gamma_{l_k,l}$ is the Kronecker delta, and $\sigma(U, U_k)$ the soft-max function:

$$\sigma(U, U_k) = \frac{\exp\left[-\mathrm{DTW}(U, U_k)^2\right]}{\sum_{r=1}^{K} \exp\left[-\mathrm{DTW}(U, U_r)^2\right]}. \tag{7.9}$$

Given the conditional probability definition, the classifier predicts a label for U as $\arg\max_l \mathrm{P}(l|U)$, i.e., as the class maximizing the output conditional probability of the smoothed kNN as per (7.8). Thereby, the DTW based smoothed NN algorithm (DTW-SNN) produces an adversarial sample $\widetilde{U}$ for a time series U whose true label is l as:

$$\widetilde{U} = U + \Omega_{\mathrm{adv}}, \tag{7.10}$$

where in this case, Ω_{adv} is a time series that represents the perturbation imprinted to U that can be computed by solving an optimization problem

similar to (7.1):

$$\Omega_{\text{adv}} = \underset{\Omega \in \mathbb{R}^n}{\text{minimize}} \quad \text{DTW}(U, U+\Omega) \tag{7.11a}$$

$$\text{subject to}$$

$$\text{P}(l|U+\Omega) < 1/L\,, \tag{7.11b}$$

where the constraint states that there is at least one class in $\{1,\ldots,l-1,l+1,\ldots,L\}$ predicted for $U+\Omega$ with higher probability than l itself.

Once we have defined a smoothed version of a NN classifier, we now derive three different adversarial sample crafting techniques to attack this particular kind of classifiers under the DTW measure. Such techniques are inspired partly by those reviewed in Section 7.1, and are described next:

- The first technique, coined as DTW based Bi-objective Optimization (DTW-BIO), aims at jointly considering the probability of the perturbed time series to successfully confuse the target model, as well as the amount of distortion imprinted to the original time series. Given a time series database $\mathcal{U}$ and the DTW-SNN model defined as per Equation (7.8), let Ω_{adv} denote the noise added to a clean sample $U \in \mathcal{U}$ whose true label is l. Expression (7.11) postulates that the optimal Ω_{adv}, when applied to the original sample, is capable of misleading the classifier while minimally corrupting the clean sample. When solving this problem, it is important to note that there is no control on the value of $\text{P}(l|\widetilde{U})$. That is, we ensure that $\text{P}(l|\widetilde{U}) < 1/L$, but the problem formulated in (6.3) does not allow the attack to be designed for arbitrarily low values of $\text{P}(l|\widetilde{U})$. Therefore, we reformulate this problem as an bi-objective optimization problem that seeks a set of Pareto-optimal Ω_U balancing between the similarity between U and $\widetilde{U}$ and the class probability $\text{P}(l|\widetilde{U})$:

$$\Omega_{\text{adv}} = \arg\min_{\Omega} \left\{\text{DTW}\left(U, U+\Omega\right), \text{P}(l|U+\Omega)\right\}, \tag{7.12}$$

  where, since $\text{DTW}\left(U, U+\Omega\right)$ denotes a measure of dissimilarity, the lower it is, the more similar U and $U+\Omega$ will be. In order to efficiently tackle this bi-objective problem, we resort to a canonical version of the NSGA-II meta-heuristic solver [135].
- The second technique (DTW-FG) is a DTW based extension of the FSGM of Goodfellow et al [113]. We limit the dissimilarity between samples as $\text{DTW}(U, \widetilde{U}) \leq \varepsilon$, where ε is a positive constant. We derive a DTW-based gradient descent technique to compute the perturbation Ω_{adv} by moving the clean time series U along the direction of the negative gradient of $\text{P}(l|U)$:

$$\Omega_{\text{adv}} = -\varepsilon \frac{\nabla \text{P}(l|U)}{||\nabla \text{P}(l|U)||_2}, \tag{7.13}$$

  where $\nabla \text{P}(l|U)$ is the gradient of the class probability, and $||\cdot||_2$ is the L_2-norm. Section 7.2.1 provides further details on this gradient computation.

- The third technique (DTW-IFG) adapts the iterative version of the fast gradient method of Kurakin et al. [114] to the DTW setting. We consider an initial condition of the form $\widetilde{U}^{(0)} = U$. In this case, the adversarial sample would be computed by following the recurrence:

$$\widetilde{U}^{(t)} = \widetilde{U}^{(t-1)} + \Omega_{\text{adv}}(\widetilde{U}^{(t-1)}) \text{ for } t \in \{1, \dots, T_{\max}\}, \tag{7.14}$$

where:

$$\Omega_{\text{adv}}(\widetilde{U}^{(t-1)}) = -\alpha \frac{\nabla \text{P}(l|\widetilde{U}^{(t-1)})}{||\nabla \text{P}(l|\widetilde{U}^{(t-1)})||_2}. \tag{7.15}$$

and $\alpha = \varepsilon / T_{\max}$ to ensure $\text{DTW}(U, \widetilde{U}^{(t)}) \leq \varepsilon$. In this case, gradients are also given as per Section 7.2.1.

7.2.1 $\text{P}(l|U)$ Gradient Computation

In what follows we formally derive the gradient for the DTW setting, which is needed by the proposed DTW-FG and DTW-IFG adversarial crafting strategies as per (7.13) and (7.15). That is to compute, for $d \in \{1, ..., n\}$:

$$\nabla \text{P}(l|U) = \frac{\partial \text{P}(l|U)}{\partial u_d}, \tag{7.16}$$

where $U = (u_1, \dots, u_d, \dots, u_n)$ is a time series with n data points. To this end, we consider the DTW-SNN model introduced in Section 7.2. The partial derivative with respect the input variable u_d is given by:

$$\frac{\partial \text{P}(l|U)}{\partial u_d} = \frac{1}{K} \left[\sum_{k=1}^{K} \Gamma_{l_l, l} \frac{\partial \sigma(U, U_k)}{\partial u_d} \right]. \tag{7.17}$$

For simplicity we rewrite the soft-max function as $\sigma(U, U_k) = H_1 / H_2$, where $H_1 = \exp\left[-\text{DTW}(U, U_k)^2\right]$ and $H_2 = \sum_{r=1}^{K} \exp\left[-\text{DTW}(U, U_r)^2\right]$. Therefore, Equation (7.17) can be rewritten as:

$$\frac{\partial \text{P}(l|U)}{\partial u_d} = \frac{1}{K} \left[\sum_{k=1}^{K} \delta_{l_k, l} \frac{\frac{\partial H_1}{\partial u_d} H_2 - H_1 \frac{\partial H_2}{\partial u_d}}{H_2^2} \right]. \tag{7.18}$$

To compute the partial derivatives of H_1 and H_2, we need to differentiate $\text{DTW}(U, U_k)$. To this end, let p_k^* be the optimal alignment between U_k and U. In other words, let p_k^* be the alignment satisfying:

$$\text{DTW}(U, U_k) = \sqrt{\sum_{(i,j) \in p_k^*} (u_i - u_j^k)^2}, \tag{7.19}$$

where u_i and u_j^k are the i-th and j-th observations of U and U_k time series, respectively (see Equations (1.16) and (1.17)). Therefore, the derivatives

of H_1 and H_2 are given by:

$$\frac{\partial H_1}{\partial u_d} = -2 \sum_{(i,j)\in p_k^*} \Gamma_{i,d}(u_d - u_j^k) \exp\left[-\mathrm{DTW}(U, U_k)^2\right], \tag{7.20}$$

$$\frac{\partial H_2}{\partial u_d} = -2 \sum_{r=1}^{K} \left\{ \sum_{(i,j)\in p_r^*} \Gamma_{i,d}(u_d - u_j^r) \exp\left[-\mathrm{DTW}(U, U_r)^2\right] \right\}, \tag{7.21}$$

where we recall that $\Gamma_{i,d}$ denotes the Kronecker delta.

7.3 Experiments

We now discuss the results obtained from a set of experiments aimed at assessing the vulnerability of DTW-NN classifiers against adversarial samples generated by the techniques presented in Section 7.2. In this set of experiments we have used the `Synthetic Control` time series classification database from the UCR archive [92]. This database is divided in a training and test subsets, both with 300 labelled time series (50 per class label). Given the training set, the challenge associated to this database is to classify every time series in the test set in one of $L = 6$ classes. Two are the reasons for selecting this dataset: 1) the length of the time series (60 observations), which allows for DTW computations within reasonable times; and ii) the performance of DTW-based NN classifiers which, for this case, report accuracy scores of 0.993.

The discussion will focus on the so-called *intra-technique transferability* [112]. Specifically, we will evaluate the capacity of an adversarial sample crafted by the proposed methods to mislead classification models belonging to the same family (namely, NN classifiers), yet trained with different datasets. Consider a set of adversarial samples $\widetilde{\mathcal{U}}$ built on a classifier f; in other words, if adversarial samples in $\widetilde{\mathcal{U}}$ have been designed properly, f should misclassify all of them. The intra-technique transferability is therefore defined as:

$$R_{\mathrm{intra}}\left(\widetilde{\mathcal{U}}, f, f'\right) = \frac{\left|\left\{\widetilde{U} \in \widetilde{\mathcal{U}} : \ f'(\widetilde{U}) \neq f'(U)\right\}\right|}{|\widetilde{\mathcal{U}}|}, \tag{7.22}$$

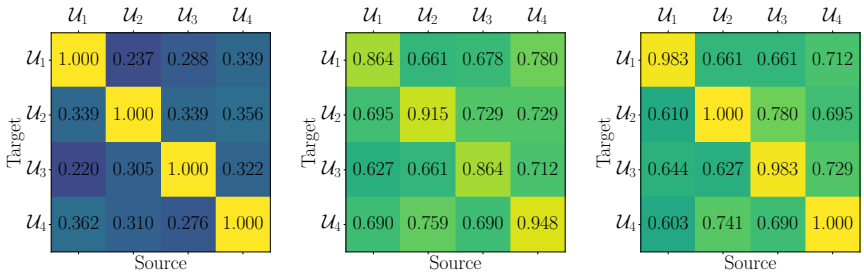
where $|\cdot|$ denotes set cardinality, f is the classifier for which adversarial samples $\widetilde{\mathcal{U}}$ were designed, and f' represents a classifier of the same family of f but trained with a different database. In our case, f and f' are both DTW-based NN classifiers with different training sets.

To evaluate the intra-technique transferability across more than one model under attack, we have split the subset of 300 test time series in 5 disjoint, stratified splits. Four of these sets are used as training sets for the DTW-NN models $\{\mathcal{U}_i\}_{i=1}^4$, while the remaining one $\mathcal{U}_5$ is used to craft adversarial samples. Consequently, the set of time series that are modified to confuse models trained on $\mathcal{U}_i$ do not participate at all in the training process of such models. This being said, $R_{\mathrm{intra}}\left(\widetilde{\mathcal{U}}, f, f'\right)$ will be

quantified for all 16 combinations of the considered training sets, e.g. given a training database $\mathcal{U}_i$ for the DTW-NN model, the analysis will verify whether samples crafted to confuse this trained model (*source* model) are misclassified by another DTW-NN model (*target* model) trained with $\mathcal{U}_{i'}$, where $i, i' \in \{1, \ldots, 4\}$.

7.3.1 Results and Discussion

We begin by discussing the results in Figures 7.1a to 7.1c, where the intra-technique transferability as per (7.22) is shown for all training set combinations between the source and target classifiers and for the proposed DTW-BIO, DTW-FG and DTW-IFG techniques. For the DTW-BIO approach, the NSGA-II solver has been configured with a population size of 100 individuals, tournament selection, SBX crossover and polynomial mutation with respective distribution parameters equal to 15 and 20, and a maximum of 10^4 number of fitness evaluations. Given the set of Pareto-optimal solutions for each $U \in \mathcal{U}_5$, the adversarial sample was produced by adding the perturbation whose $\mathrm{P}(l|\widetilde{U})$ was closer to (yet smaller than) $1/L = 1/6$. To generate gradient-based adversarial samples, ε has been set to 4.0, which has been tuned off-line as per off-line trial experiments. For the DTW-IFG case, the maximum number of iterations has been set to $T_{\max} = 25$.



(A) DTW-BIO (B) DTW-FG (C) DTW-IFG

FIGURE 7.1: Intra-technique transferability for adversarial samples drawn by a) DTW-BIO; b) DTW-FG; and c) DTW-IFG. Results must be interpreted as e.g. *as shown in entry* $(1, 3)$ *of DTW-BIO, 28.8% of the adversarial samples crafted from source model trained with dataset* $\mathcal{U}_3$ *are misclassified by target model trained with dataset* $\mathcal{U}_1$. The average accuracy computed over 4 DTW based NN classifiers fed with training sets $\mathcal{U}_1$, $\mathcal{U}_2$, $\mathcal{U}_3$ or $\mathcal{U}_4$ is 0.979 measured over $\mathcal{U}_5$ as their test set.

A first inspection of these plots reveals that as expected, the crafted adversarial samples perform best when both source and target models share the same training set (diagonal entries in the matrices). This noted fact is in accordance with similar conclusions extracted by Papernot et al in [112], where NN models (under Euclidean distance) were found to be, to a certain extent, robust to intra-technique transferability. However, it is clearly shown that gradient-based strategies produce more transferable adversarial samples than the DTW-BIO technique, due to the fact that

this latter approach overfits the crafted perturbation to the training set of the source model. This calls for further research aimed at determining whether overfitting is due to the small sizes of the training datasets (60 time series in total, 10 per class), as well as to the disjoint split by which they were built.

We now turn the focus on Figure 7.2, where we show, for the three proposed schemes, the achieved trade-off between the distortion of a corrupted sequence with respect its clean version, and the class probability $P(l|\widetilde{U})$ of the produced adversarial time series. To trace the evolution of DTW-FG as a function of the distortion, the technique has been run for $\varepsilon \in \{1.0, 2.0, 3.0, 4.0\}$. For the DTW-IFG, the maximum distortion has been set to be $\varepsilon = 4.0$ and the maximum number of iterations to $T_{\max} = 25$. The curve depicts the similarity and class probability values for the partial adversarial time series yielded by the recurrence in Expression (7.15). In this second plot we can also observe that DTW-BIO attains a more diverse set of adversarial samples than the other techniques, while requiring less parameter tuning. The downside is a higher computational complexity which, depending on the application scenario, can make this approach less convenient than its gradient-based counterparts.

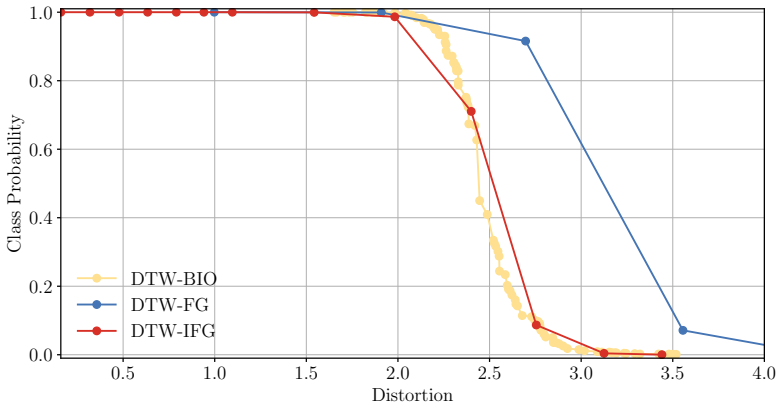


FIGURE 7.2: Class probability $P(l|\widetilde{U})$ of the proposed adversarial sample crafting schemes as a function of the distortion for an arbitrarily chosen time series whose true label is $l = 2$: DTW-BIO (yellow), DTW-FG (blue) and DTW-IFG (red).

Finally, Figure 7.3 compares visually the original sequence and its corresponding adversarial time series. From top to bottom, the first figure depicts the original sequence (black) over the time series in $\mathcal{U}$ sharing its class label $l = 1$ (gray). The second figure illustrates the adversarial sample crafted by DTW-BIO (red solid line), as well as the reference time series of the predicted class (in this case the gray time series belong to class $l = 3$). Likewise, the third and fourth figures illustrate the adversarial samples generated by DTW-FG and DTW-IFG, respectively. As opposed to the DTW-BIO example, it is interesting to observe that gradient-based methods produce an adversarial sample that is misclassified by the target model as label 6. In all cases, the degradation of the adversarial example

is almost negligible with respect to its clean version, which underscores the conclusion that robust models against subtle adversarial attacks are also needed for time series classifiers relying on elastic similarity measures.

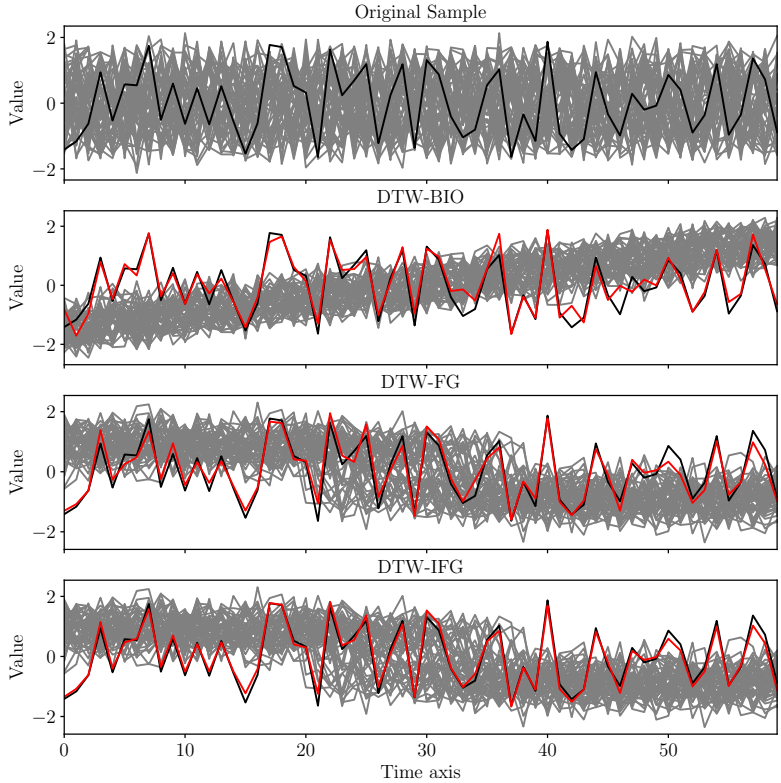


FIGURE 7.3: Comparison between an arbitrary original sequence (black) and its adversarial samples (red) crafted by every proposed technique. The class label predicted for the original and adversarial sequences is represented by the gray set of time series depicted in the background of each subfigure: from top to bottom, class label $l = 1$ (which is the class of the original time series), $l = 3$ (DTW-BIO) and $l = 6$ (DTW-FG and DTW-IFG).

7.4 Conclusions

This chapter has presented three adversarial time series crafting techniques aimed at misleading outstanding DTW-based NN classifiers for time series. Specifically, the proposed portfolio of techniques hinge on an optimization problem that aims at discovering the smallest perturbation that, when added to a legitimate time series, forces DTW-based classifiers to issue erroneous predictions for the perturbed time series. To solve this problem, two gradient-based techniques (DTW-FG and DTW-IFG) and a bi-objective optimization problem (DTW-BIO) have been used.

The performance of these techniques when attacking DTW based NN classifiers has been empirically assessed by inspecting the intra-technique

transferability across 4 classifiers fed with different training data. We have utilized the `Synthetic Control` public time series dataset from the UCR archive. Conclusions drawn from the results obtained with these experiments are listed below:

- Regardless the adversarial sample crafting technique, the observed intra-technique transferability evinces that crafted adversarial time series are capable of misleading DTW-based NN classifiers trained with different training sets than the one used for the crafting process.
- Adversarial time series produced by DTW-FG and DTW-IFG are, in general, more transferable than those furnished by the BIO-DTW technique. However, BIO-DTW is more effective when attacking target classifiers whose training set is the same than that of the source model used for generating the adversarial samples.
- The difference between generated samples and their original version is almost negligible for the majority of time series considered in the experiments, disregarding the crafting technique.

Several future research lines are rooted on the findings reported in this chapter. To begin with, each adversarial sample technique should be evaluated in a benchmark encompassing a higher number of time series classification problems. Since DTW permits to compute the similarity among time series of different length, this study can be extended to the case where the time series in source and target training sets have different number of observations. We will also analyze how the transferability of adversarial samples produced by DTW-BIO evolves with the degree of overlapping between the training sets of source and target classification models. Finally, we plan to analyze the so-called *cross-technique* transferability, which aims at verifying the capability of adversarial samples to confuse different target classification models when their learning algorithms differ from that used in the source classification model (using distance based models such as Gaussian Mixtures or SVMs).

Part III

Concluding Remarks

Chapter 8

Concluding Remarks

8.1 Conclusions

Time series have traditionally played a central role in a plethora of real-world machine learning problems formulated in scenarios hinging on this kind of data. This noted prevalence of time series data across different domains and disciplines has occurred *on a par* with notable improvements in data collection, storage and retrieval. This technological ecosystem has sharply ignited the need for efficient modeling approaches capable of dealing with complex learning paradigms defined on time series data. In this vein, this Thesis has placed the spotlight on novel approaches that rely on measures of elastic similarity between time series (ESMs), with a view towards the formulation and efficient solving of two problems in the time series analysis field: streaming time series classification, and adversarial machine learning. The novel contributions achieved in the Thesis are next summarized:

Development of a novel elastic similarity measure for multi-dimensional time series classification tasks.

First, Chapter 3 has proposed a framework to learn a similarity measure for MTS to undertake classification tasks. The challenge resides in learning a multi-dimensional DTW formulation that considers time series of the same class closer than series among different categories. To this end, we have proposed DTW_{opt}, a multi-dimensional measure of similarity that captures the relations between the different dimensions of the time series, and efficiently exploits them for classification. The proposed framework has been modeled through two extreme approximations: the independent, $\text{DTW}_{\boldsymbol{I}}$, and the dependent, $\text{DTW}_{\boldsymbol{I}}$, DTW formulations. In the former approximation, the similarity results from averaging the DTW computed for each dimension of the problem. In this context, each dimension is treated independently from others, thus performing a warping path (i.e, an alignment) for each dimension of the problem. By doing so, MTS are considered as a collection of unrelated uni-dimensional sequences. In the dependent formulation in contrast, all dimensions are aligned jointly using the same warping path thereby, considering MTS as a whole entity with deep relationships among dimensions.

On this basis, the proposed similarity learning problem consists of finding the best combination between the independent and dependent DTW formulations that minimizes the error of a NN classifier. Specifically, the proposed similarity measure, DTW_{opt}, consists of a heuristic wrapper that maximizes the cross-validation score of a DTW_{opt}-NN classifier by weighting the contribution of each dimension of the independent formulation and the dependent measurement. To solve the underlying optimization problem efficiently, four heuristic solvers have been considered, namely, Simulated Annealing (SA), Particle Swarm Optimization (PSO), Genetic Algorithm (GA) and Estimation of Distribution Algorithm (EDA). Results have been reported on the classification accuracy resulting from the evaluation of each framework over several reference time series classification databases. The obtained results have shown that DTW_{opt} based NN classifier performs equal or better than DTW_I- and DTW_D-based counterparts. No significant differences were found among heuristic solvers, although the DTW_{opt} optimized via SA issued the most accurate results in the discussed benchmark. Given the degrees of freedom of the formal definitions of DTW_I and DTW_D, we observed that our proposal tends to adapt the independent part better than its dependent counterpart.

Formulation of an on-line elastic similarity measure (OESM) that accommodates canonical elastic similarity measures to the characteristics of streaming time series.

Chapter 4 has stressed on the computational issues of learning problems formulated over streaming time series, namely, the speed, volume and changing structure of streaming time series data. Hence, on-line learning algorithms are often required to accommodate the dynamics of the streaming data efficiently (i.e., with minimal resource usage). The OESM proposed in this chapter has been precisely conceived to comply with these two requirements: the efficient update of the measure and its adaptation to upcoming events. In addition, we consider two different streaming time series scenarios: (i) the batch pattern scenario, where the similarity is measured between a stored sequence and the streaming time series; and (ii) the on-line pattern scenario, where the similarity is computed between two streaming series that evolve over time.

In both streaming scenarios, OESM comprises a forgetting mechanism and an incremental computation procedure. On one hand, the forgetting mechanism accommodates the similarity to evolving streaming time series by under-weighting the contribution of old observations. To this end, a memory parameter $\rho \in (0, 1]$ has been defined to tune this weighting mechanism. On the other hand, the incremental computation updates the OESM by means of the so-called *frontier*, i.e., intelligently stored intermediate similarity measurements that are employed thereafter to avoid redundant computations on newly arriving observations.

Two sets of experiments have been conducted to evaluate the performance of the formulated adaptation. In the first set, we have assessed the efficiency of the OESM in terms of computational complexity. For this purpose, running times of OESM and ESM have been reported for

different streaming scenarios. The obtained results have evinced that the use of the frontier set a limit in the usage of computational resources, thereby making the OESM a suitable approach in this regard for streaming scenarios. In the second set of experiments, we have evaluated the performance of the proposed forgetting mechanism. In doing so, we have examined the performance of NN models relying on the proposed OESM over streaming time series classification problems, and for different values of the memory parameter ρ. Results from this second experimental study conclude that the forgetting mechanism behaves in accordance with the stability-plasticity dilemma, which states that on-line learning procedures can retain old knowledge to grasp new information over the stream (stability), or discard old knowledge to adapt to the prevailing stream data (plasticity). The challenge in this matter is to properly account for these two properties in the algorithmic design of the learning algorithm, and adapt it according to the characteristics of the stream at hand. Obtained results conform to this established intuition, and open up new research directions towards the configuration of the memory parameter of the proposed OESM.

Application of elastic similarity frames to develop an active adaptation strategy for streaming time series classification in batch pattern scenarios.

The contribution of Chapter 5 must be also appraised within the context of streaming time series classification, and constitutes an alternative method to the need for adaptability that ESM impose when used within streaming contexts. Specifically, instead of facing this from a passive forgetting mechanism as in Chapter 4, here we have proposed an active scheme to identify transitions between periods of the streaming time series composed by events with different labels. Specifically, the proposed approach is founded on the analysis of ODTW streaming frames, i.e., on arrays of partial similarity measurements among the reference patterns and incoming stream data points. By doing so, the procedure detects boundaries between streaming time series class labels that can be leveraged to actively adapt DTW-NN classifiers to such changes. To this end, we have developed the Pattern End Detector (PED) model, a deep Convolutional Neural Network disposed to categorize streaming frames into *change* or *not change* classes. When a change is detected, the PED model intervenes the DTW-NN by forcing the classifier to reset the DTW similarity, thereupon accommodating the NN classifier to the upcoming class. We evaluate the proposed framework on a benchmark of time series classification databases. In this context, the reported PED model delay and false alarm results have provided empirical insights on the utility of the proposed method when combined with DTW-based NN classifiers. In fact, classifiers endowed with the proposed PED model and a gold-standard detector (i.e., a DTW-based NN classifier that initializes the similarity when required, assuming perfect a priori knowledge of the points in time at which label changes occur) behave similarly in terms of detection accuracy and

predictive performance, which ultimately buttress the suitability of elastic similarity frames to detect label changes.

Development of attack and defense adversarial machine learning strategies based on the DTW measure of similarity.

Chapters 6 and 7 have shifted the focus of the Thesis towards the domain of Adversarial Machine Learning (AML), targeting the sensitivity of machine learning models against intelligently modified input samples devised to confuse their output. To begin with, Chapter 6 has presented a novel adversarial defense strategy that protect Deep Neural Networks (DNN) for image classification against such adversarial attacks. Specifically, we have developed an adversarial detection mechanism that analyzes the color gradients of input images to detect minor modifications made by adversarial attacks. To this end, a one-class SVM (OSVM) classifier is learned that distinguishes legitimate inputs from adversarial samples by analyzing feature vectors of the DTW. That is, by evaluating the similarity among edge count sequences – which encode the color variations of an image under different sensitivity levels of the Canny edge detector procedure – of the legitimate training images and the input example. To evaluate the performance of the proposed OSVM discriminator, we have analyzed the detection rate of the proposed strategy when attacking three different DNN image classifiers via a benchmark of image adversarial sample crafting techniques. Obtained results have rendered accurate detection rates when dealing with attacks that operate over the whole image space (i.e., when attacks make minor color variations, without damaging geometric shapes). By contrast, the proposed method fails to detect attacks that imprint changes over small regions of the image, or around the existing shapes of the image.

Second, Chapter 7 has elaborated on three novel adversarial time series crafting strategies against DTW-NN models aimed to classify sequences, for which traditional attack methods from the AML literature cannot be extrapolated easily. In two of the developed strategies, we have adapted state-of-the-art gradient-based crafting techniques to the DTW-NN classifier. In the third strategy, we have generated adversarial samples by formulating it as a bi-objective optimization problem, where solutions are governed by a Pareto front between the DTW distortion and the probability of the attack to mislead the model under target. For each proposed attack technique, we have discussed on the intra-technique transferability, an evaluation measurement that quantifies the capability of a strategy to generate samples that mislead classifiers of the same family (NN in this case), albeit trained on different data splits of the database at hand. Results stemming from this experimentation study substantiate that an important number of crafted adversarial samples can mislead DTW-NN classifiers. However, adversarial samples generated from the gradient-based crafting strategies have been found to be, in general, more versatile (in terms of transferability) than those drawn from the bi-objective optimization strategy.

8.2 List of Publications

The research work conducted while pursuing this doctoral degree has given rise to several contributions in journals and conferences of the machine learning area, which are listed below:

- **Journal publications:**
  - Izaskun Oregi, Aritz Pérez, Javier Del Ser, Jose A. Lozano, "On-line elastic similarity measures for time series", Pattern Recognition, Vol. 88, pp. 506-517, 2019 (**JCR** 5.898, 14/134 ARTIFICIAL INTELLIGENCE, **Q1**).
  - Izaskun Oregi, Javier Del Ser, Aritz Pérez, Jose A. Lozano, "Robust image classification against adversarial attacks using elastic similarity measures between edge count sequences", Neural Networks, Vol. 128, pp. 61-72, 2020 (**JCR** 5.785, 16/134 ARTIFICIAL INTELLIGENCE, **Q1**).
  - Izaskun Oregi, Aritz Pérez, Javier Del Ser, Jose A. Lozano, "Learning label transitions from dynamic time warping features for streaming time series classification", under review, 2020.
- **Conference publications:**
  - Izaskun Oregi, Javier Del Ser, Aritz Pérez, Jose A. Lozano, "Nature-inspired approaches for distance metric learning in multivariate time series classification", IEEE Congress on Evolutionary Computation (CEC), pp. 1992-1998, 2017.
  - Izaskun Oregi, Aritz Pérez, Javier Del Ser, Jose A. Lozano, "On-line dynamic time warping for streaming time series", Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD), pp. 591-605, 2017.
  - Izaskun Oregi, Javier Del Ser, Aritz Pérez, Jose A. Lozano, "Adversarial sample crafting for time series classification with elastic similarity measures", International Symposium on Intelligent and Distributed Computing, pp. 26-39, 2018.

8.2.1 Other Publications

The candidate has also collaborated actively in other assorted research lines leading to the following publications:

- **Journal publications**
  - Esther Villar-Rodríguez, Javier Del Ser, Izaskun Oregi, Miren Nekane Bilbao, Sergio Gil-Lopez, "Detection of non-technical losses in smart meter data based on load curve profiling and time series analysis", Energy, Vol. 137, pp. 118-128, 2017 (**JCR** 5.537, 4/59 THERMODYNAMICS, **Q1**).

- Antonio Benítez-Hidalgo, Antonio J. Nebro, Jose M. Garcia-Nieto, Izaskun Oregi, Javier Del Ser, "jMetalPy: a python framework for multi-objective optimization with metaheuristics", Swarm and Evolutionary Computation, Vol. 51, 100598, 2019 (**JCR** 6.330, 11/134 ARTIFICIAL INTELLIGENCE, **Q1**).
- Jesús L. Lobo, Izaskun Oregi, Albert Bifet, Javier Del Ser, "Exploiting the stimuli encoding scheme of evolving spiking neural networks for stream learning", Neural Networks, Vol. 123, pp. 118-133, 2020 (**JCR** 5.785, 16/134 ARTIFICIAL INTELLIGENCE, **Q1**).
- Jesús L. Lobo, Igor Ballesteros, Izaskun Oregi, Javier Del Ser, Sancho Salcedo-Sanz, "Stream learning in energy IoT systems: a case study in combined cycle power plants", Energies, Vol. 13, N. 3, 740, 2020 (**JCR** 2.707, 56/103 ENERGY & FUELS, **Q3**).

- **Conference publications:**
  - Ibai Laña, Javier Del Ser, Manuel Vélez, Izaskun Oregi, "Joint feature selection and parameter tuning for short-term traffic flow forecasting based on heuristically optimized multi-layer neural networks", Advances in Intelligent Systems and Computing book series (AISC, volume 514), pp. 91–100, 2017.
  - Aritz D. Martínez, Eneko Osaba, Izaskun Oregi, Iztok Fister, Javier Del Ser, "Hybridizing differential evolution and novelty search for multimodal optimization problems", ACM Genetic and Evolutionary Computation Conference Companion, pp. 1980-1989, 2019.
  - Ibai Laña, Esther Villar-Rodríguez, Urtats Etxegarai, Izaskun Oregi, Javier Del Ser, "A question of trust: statistical characterization of long-term traffic estimations for their improved actionability", IEEE Intelligent Transportation Systems Conference, pp. 1922-1928, 2019.
  - Javier Del Ser, Ibai Laña, Eric L. Manibardo, Izaskun Oregi, Eneko Osaba, Jesus L. Lobo, Miren Nekane Bilbao, Eleni I. Vlahogianni, "Deep echo state networks for short-term traffic forecasting: performance comparison and statistical assessment", IEEE Intelligent Transportation Systems Conference, 2020 (under review).

8.2.2 Short research visits.

- 11 June 2018 - 24 June 2018. Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga, Spain. Supervisor: Prof. Dr. José Francisco Aldana Montes.
- 15 September 2019 - 15 December 2019. Institut de Recherche en Informatique et Système Aléatoires (IRISA), Université Bretagne Sud, France. Supervisor: Prof. Dr. Pierre-François Marteau.

8.3 Future Research Lines

As exposed throughout the chapters in Part II, the findings reported in this Thesis take a step beyond the state of the art in the application of

elastic similarity measures to different time series learning problems. The conclusions held in such chapters have also stimulated several alternative research directions to be pursued in the future, as well as new questions and hypotheses worth to be explored in depth. We here summarize the most noteworthy issues derived from this Thesis, referring to the concluding sections of Chapters 3 to 7 for further details on their rationale and envisaged approach:

- In order to further enhance the performance of the DTW_{opt} similarity proposed in Chapter 3, we plan to provide a greater flexibility to DTW_{D}. As pursued in [77], an alternative dependent DTW could be formulated to weight differently the contribution of each dimension to the warping path. By doing so, we expect that the dependent part of DTW_{opt} captures more accurately the relationships among the dimensions of the time series. In this vein, it would be also interesting to study other error rate functions that are commonly used in distance learning tasks, e.g. the larger-margin nearest neighbor formulation [87].

- Regarding the on-line measure of similarity OESM defined in Chapter 4, we plan to use it to adapt other procedures for time series analysis to streaming time series, including kernel-based classifiers, `K`-medoids or hierarchical clustering algorithms. Additionally, we will also devote efforts towards normalization techniques that might not alter the shape of the streaming time series under analysis, so that OESM-based models can be applied without any prior assumption on the normalization of the stream.

- In relation to the memory parameter ρ governing the capability of the OESM to retain knowledge captured over the streaming time series, we plan to develop a strategy that selects most appropriate value according to the characteristics of a given problem e.g., adjusting ρ to the expected average length of stationary intervals. We also propose to increase the list of cost functions: apart from DTW, EDR, EDIT and other functions alike. In particular, we suggest the use of linear correlation-based costs, where the measure of similarity considers the mean and the variance (or co-variance) of the time series.

- Several research lines are aligned with the concept of elastic similarity frames coined in Chapter 5. Specifically, we propose to exploit this data structure to develop further active adaptation procedures for addressing anomaly detection or classification problems over streaming time series. One of the main advantages of elastic similarity frames is that they can be normalized on stream settings, thus overriding any need for normalizing the streaming time series at hand. In particular, regarding supervised learning tasks (e.g. anomaly detection or classification), we are determined to build a concept drift detection procedure based on streaming frames. That is, we plan to develop an algorithm in which non-stationarities in the correspondence between events (time series) and labels will be detected by processing underlying structures of streaming frames computed over the streaming time series. Then, in

the eventuality of a detected drift, we plan to properly accommodate the learned model to such concept changes.

- With respect to the active adaptation strategy developed in Chapter 5, we will move towards the design of early classification models for streaming time series. Furthermore, explainable Artificial Intelligence (XAI) techniques will be investigated to understand what the PED model observes in the streaming time series (and eventually, the early classification model), so as to extract further insights and use this augmented information to further improve its performance or develop new detection/adaptation strategies.

- When it comes to the AML defense technique presented in Chapter 6, we foresee that the proposed OSVM detector can be enhanced by using novel features that are more sensitive to modifications near the edges of the original image, and/or circumscribed to small pixel regions of the image. A closer look will be also taken at encryption techniques such as steganography, which shares interesting commonalities with AML.

- Finally, a natural step following the experimental study on adversarial attacks for DTW-based classifiers discussed in Chapter 7 is to analyze the so-called *cross-technique transferability*. That is, the capability of adversarial samples crafted on the DTW-NN classifier to confuse different target classification models (e.g. SVMs or Gaussian mixtures). By doing so, we can characterize the performance of the developed methods as black-box attack strategies, and identify potential weaknesses of defense strategies reported in the AML field.

Bibliography

[1] T.-C. Fu, "A review on time series data mining", *Engineering Applications of Artificial Intelligence*, vol. 24, no. 1, pp. 164–181, 2011.

[2] P. Esling and C. Agon, "Time-series data mining", *ACM Computing Surveys*, vol. 45, no. 1, p. 12, 2012.

[3] P. P. Rodrigues, J. Gama, and J. Pedroso, "Hierarchical clustering of time-series data streams", *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, no. 5, pp. 615–627, 2008.

[4] J.-L. Reyes-Ortiz, L. Oneto, A. Samà, X. Parra, and D. Anguita, "Transition-aware human activity recognition using smartphones", *Neurocomputing*, vol. 171, pp. 754–767, 2016.

[5] E. L. Manibardo, I. Laña, J. L. Lobo, and J. Del Ser, "New perspectives on the use of online learning for congestion level prediction over traffic data", *arXiv:2003.14304*, 2020.

[6] J. Gama, *Knowledge discovery from data streams*. CRC Press, 2010.

[7] G. Krempl, I. Žliobaite, D. Brzeziński, E. Hüllermeier, M. Last, V. Lemaire, T. Noack, A. Shaker, S. Sievi, M. Spiliopoulou, *et al.*, "Open challenges for data stream mining research", *ACM SIGKDD Explorations Newsletter*, vol. 16, no. 1, pp. 1–10, 2014.

[8] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation", *ACM computing surveys (CSUR)*, vol. 46, no. 4, pp. 1–37, 2014.

[9] L. L. Minku, A. P. White, and X. Yao, "The impact of diversity on online ensemble learning in the presence of concept drift", *IEEE Transactions on knowledge and Data Engineering*, vol. 22, no. 5, pp. 730–742, 2009.

[10] R. C. Cavalcante, L. L. Minku, and A. L. Oliveira, "Fedd: Feature extraction for explicit concept drift detection in time series", in *Neural Networks (IJCNN), 2016 International Joint Conference on*, IEEE, 2016, pp. 740–747.

[11] I. Lana, J. Del Ser, M. Velez, and E. I. Vlahogianni, "Road traffic forecasting: Recent advances and new challenges", *IEEE Intelligent Transportation Systems Magazine*, vol. 10, no. 2, pp. 93–109, 2018.

[12] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, "The great time series classification bake off: A review and experimental evaluation of recent algorithmic advances", *Data Mining and Knowledge Discovery*, vol. 31, no. 3, pp. 606–660, 2017.

[13] T. W. Liao, "Clustering of time series data—a survey", *Pattern recognition*, vol. 38, no. 11, pp. 1857–1874, 2005.

[14] S. Aminikhanghahi and D. J. Cook, "A survey of methods for time series change point detection", *Knowledge and information systems*, vol. 51, no. 2, pp. 339–367, 2017.

[15] C. Deb, F. Zhang, J. Yang, S. E. Lee, and K. W. Shah, "A review on time series forecasting techniques for building energy consumption", *Renewable and Sustainable Energy Reviews*, vol. 74, pp. 902–924, 2017.

[16] C. Voyant, G. Notton, S. Kalogirou, M.-L. Nivet, C. Paoli, F. Motte, and A. Fouilloy, "Machine learning methods for solar radiation forecasting: A review", *Renewable Energy*, vol. 105, pp. 569–582, 2017.

[17] A. Kazem, E. Sharifi, F. K. Hussain, M. Saberi, and O. K. Hussain, "Support vector regression with chaos-based firefly algorithm for stock market price forecasting", *Applied soft computing*, vol. 13, no. 2, pp. 947–958, 2013.

[18] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[19] J. G. D. Gooijer and R. J. Hyndman, "25 years of time series forecasting", *International Journal of Forecasting*, vol. 22, no. 3, pp. 443 –473, 2006.

[20] J. Wang and J. Wang, "Forecasting energy market indices with recurrent neural networks: Case study of crude oil price fluctuations", *Energy*, vol. 102, pp. 365–374, 2016.

[21] W. Kong, Z. Y. Dong, Y. Jia, D. J. Hill, Y. Xu, and Y. Zhang, "Short-term residential load forecasting based on lstm recurrent neural network", *IEEE Transactions on Smart Grid*, vol. 10, no. 1, pp. 841–851, 2017.

[22] W. A. Chaovalitwongse, Y. Fan, and R. C. Sachdeo, "On the time series k-nearest neighbor classification of abnormal brain activity", *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 37, no. 6, pp. 1005–1016, 2007.

[23] F. Lotte, M. Congedo, A. Lécuyer, F. Lamarche, and B. Arnaldi, "A review of classification algorithms for eeg-based brain–computer interfaces", *Journal of neural engineering*, vol. 4, no. 2, R1, 2007.

[24] S.-N. Yu and Y.-H. Chen, "Electrocardiogram beat classification based on wavelet transformation and probabilistic neural network", *Pattern Recognition Letters*, vol. 28, no. 10, pp. 1142–1150, 2007.

[25] L. Chen, M. T. Özsu, and V. Oria, "Robust and fast similarity search for moving object trajectories", in *ACM SIGMOD International Conference on Management of Data*, ACM, 2005, pp. 491–502.

[26] M. J. Fard, A. K. Pandya, R. B. Chinnam, M. D. Klein, and R. D. Ellis, "Distance-based time series classification approach for task recognition with application in surgical robot autonomy", *The International Journal of Medical Robotics and Computer Assisted Surgery*, vol. 13, no. 3, 2017.

[27] Z. Bankó and J. Abonyi, "Correlation based dynamic time warping of multivariate time series", *Expert Systems with Applications*, vol. 39, no. 17, pp. 12814 –12 823, 2012.

[28] L. Chen and R. Ng, "On the marriage of lp-norms and edit distance", in *30th International Conference on Very Large Data Bases*, VLDB Endowment, vol. 30, 2004, pp. 792–803.

[29] Y.-S. Jeong, M. K. Jeong, and O. A. Omitaomu, "Weighted dynamic time warping for time series classification", *Pattern Recognition*, vol. 44, no. 9, pp. 2231–2240, 2011.

[30] T. Górecki and M. Łuczak, "Multivariate time series classification with parametric derivative dynamic time warping", *Expert Systems with Applications*, vol. 42, no. 5, pp. 2305–2312, 2015.

[31] A. Kampouraki, G. Manis, and C. Nikou, "Heartbeat time series classification with support vector machines", *IEEE Transactions on Information Technology in Biomedicine*, vol. 13, no. 4, pp. 512–518, 2008.

[32] A. Nanopoulos, R. Alcock, and Y. Manolopoulos, "Feature-based classification of time-series data", *International Journal of Computer Research*, vol. 10, no. 3, pp. 49–61, 2001.

[33] H. Deng, G. Runger, E. Tuv, and M. Vladimir, "A time series forest for classification and feature extraction", *Information Sciences*, vol. 239, pp. 142–153, 2013.

[34] C. Pelletier, G. I. Webb, and F. Petitjean, "Temporal convolutional neural network for the classification of satellite image time series", *Remote Sensing*, vol. 11, no. 5, p. 523, 2019.

[35] O. Yakhnenko, A. Silvescu, and V. Honavar, "Discriminatively trained markov model for sequence classification", in *Fifth IEEE International Conference on Data Mining (ICDM'05)*, IEEE, 2005, 8–pp.

[36] A. K. Jain, "Data clustering: 50 years beyond k-means", *Pattern recognition letters*, vol. 31, no. 8, pp. 651–666, 2010.

[37] L. Kaufman and P. J. Rousseeuw, *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 2009, vol. 344.

[38] S. Aghabozorgi, A. S. Shirkhorshidi, and T. Y. Wah, "Time-series clustering–a decade review", *Information Systems*, vol. 53, pp. 16–38, 2015.

[39] T. Räsänen and M. Kolehmainen, "Feature-based clustering for electricity use time series data", in *International conference on adaptive and natural computing algorithms*, Springer, 2009, pp. 401–412.

[40] A. Bagnall and G. Janacek, "Clustering time series with clipped data", *Machine Learning*, vol. 58, no. 2-3, pp. 151–178, 2005.

[41] A. Carreño, I. Inza, and J. A. Lozano, "Analyzing rare event, anomaly, novelty and outlier detection terms under the supervised classification framework", *Artificial Intelligence Review*, pp. 1–20, 2019.

[42] A. Diez-Olivan, J. Del Ser, D. Galar, and B. Sierra, "Data fusion and machine learning for industrial prognosis: Trends and perspectives towards industry 4.0", *Information Fusion*, vol. 50, pp. 92–111, 2019.

[43] M. Gupta, J. Gao, C. C. Aggarwal, and J. Han, "Outlier detection for temporal data: A survey", *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 9, pp. 2250–2267, 2013.

[44] A. Blázquez-García, A. Conde, U. Mori, and J. A. Lozano, "A review on outlier/anomaly detection in time series data", *arXiv:2002.04236*, 2020.

[45] M. Munir, S. A. Siddiqui, A. Dengel, and S. Ahmed, "Deepant: A deep learning approach for unsupervised anomaly detection in time series", *IEEE Access*, vol. 7, pp. 1991–2005, 2018.

[46] U. Rebbapragada, P. Protopapas, C. E. Brodley, and C. Alcock, "Finding anomalous periodic time series", *Machine learning*, vol. 74, no. 3, pp. 281–313, 2009.

[47] V. Chandola, V. Mithal, and V. Kumar, "Comparative evaluation of anomaly detection techniques for sequence data", in *2008 Eighth IEEE international conference on data mining*, IEEE, 2008, pp. 743–748.

[48] G. Das, K.-I. Lin, H. Mannila, G. Renganathan, and P. Smyth, "Rule discovery from time series.", in *KDD*, vol. 98, 1998, pp. 16–22.

[49] F.-L. Chung, T.-C. Fu, V. Ng, and R. W. Luk, "An evolutionary approach to pattern-based time series segmentation", *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 5, pp. 471–489, 2004.

[50] A. M. Durán-Rosal, P. A. Gutierrez, S. Salcedo-Sanz, and C. Hervás-Martínez, "A statistically-driven coral reef optimization algorithm for optimal size reduction of time series", *Applied Soft Computing*, vol. 63, pp. 139–153, 2018.

[51] M. Goldhammer, S. Köhler, K. Doll, and B. Sick, "Camera based pedestrian path prediction by means of polynomial least-squares approximation and multilayer perceptron neural networks", in *2015 SAI Intelligent Systems Conference (IntelliSys)*, IEEE, 2015, pp. 390–399.

[52] E. Bingham, A. Gionis, N. Haiminen, H. Hiisilä, H. Mannila, and E. Terzi, "Segmentation and dimensionality reduction", in *SIAM International Conference on Data Mining*, SIAM, 2006, pp. 372–383.

[53] O. Amft, H. Junker, and G. Troster, "Detection of eating and drinking arm gestures using inertial body-worn sensors", in *Ninth IEEE International Symposium on Wearable Computers (ISWC'05)*, IEEE, 2005, pp. 160–163.

[54] E. Keogh, S. Chu, D. Hart, and M. Pazzani, "Segmenting time series: A survey and novel approach", in *Data mining in time series databases*, World Scientific, 2004, pp. 1–21.

[55] A. Abanda, U. Mori, and J. A. Lozano, "A review on distance based time series classification", *Data Mining and Knowledge Discovery*, vol. 33, no. 2, pp. 378–412, 2019.

[56] B. D. Fulcher, "Feature-based time-series analysis", in *Feature Engineering for Machine Learning and Data Analytics*, CRC Press, 2018, pp. 87–116.

[57] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: A review", *Data Mining and Knowledge Discovery*, vol. 33, no. 4, pp. 917–963, 2019.

[58] Y. Xiong and D.-Y. Yeung, "Mixtures of ARMA models for model-based time series clustering", in *IEEE International Conference on Data Mining*, IEEE, 2002, pp. 717–720.

[59] Z. Xing, J. Pei, and E. Keogh, "A brief survey on sequence classification", *ACM Sigkdd Explorations Newsletter*, vol. 12, no. 1, pp. 40–48, 2010.

[60] D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series", in *Workshop on Knowledge Discovery in Databases*, Seattle, WA, 1994, pp. 359–370.

[61] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh, "Querying and mining of time series data: Experimental comparison of representations and distance measures", *Proceedings of the Very Large Data Bases Endowment*, vol. 1, no. 2, pp. 1542–1552, 2008.

[62] R. E. Bellman and S. E. Dreyfus, *Applied dynamic programming*. Princeton University Press, 2015.

[63] E. Keogh, "Exact indexing of dynamic time warping", in *28th International Conference on Very Large Data Bases*, VLDB Endowment, 2002, pp. 406–417.

[64] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, "A symbolic representation of time series, with implications for streaming algorithms", in *8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, ACM, 2003, pp. 2–11.

[65] Y. Sakurai, M. Yoshikawa, and C. Faloutsos, "Ftw: Fast similarity search under the time warping distance", in *24th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, ACM, 2005, pp. 326–337.

[66] E. J. Keogh and M. J. Pazzani, "Scaling up dynamic time warping for datamining applications", in *6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 2000, pp. 285–289.

[67] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space", *Intelligent Data Analysis*, vol. 11, no. 5, pp. 561–580, 2007.

[68] H Sakoe and S Chiba, "Dynamic programming algorithm optimization for spoken word recognition", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 26, no. 1, pp. 43–49, 1978.

[69] F. Itakura, "Minimum prediction residual principle applied to speech recognition", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 23, no. 1, pp. 67–72, 1975.

[70] C. A. Ratanamahatana, J. Lin, D. Gunopulos, E. Keogh, M. Vlachos, and G. Das, "Mining time series data", in *Data mining and knowledge discovery handbook*, Springer, 2005, pp. 1069–1103.

[71] A. Marzal and E. Vidal, "Computation of normalized edit distance and applications", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 9, pp. 926–932, 1993.

[72] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks", *arXiv:1312.6199*, 2013.

[73] I. Goodfellow, P. McDaniel, and N. Papernot, "Making machine learning robust against adversarial inputs", *Communications of the ACM*, vol. 61, no. 7, pp. 56–66, 2018.

[74] N. Papernot and P. McDaniel, "Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning", *arXiv:1803.04765*, 2018.

[75] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning", *Pattern Recognition*, vol. 84, pp. 317–331, 2018.

[76] M. Shokoohi-Yekta, B. Hu, H. Jin, J. Wang, and E. Keogh, "Generalizing dtw to the multi-dimensional case requires an adaptive approach", *Data mining and knowledge discovery*, vol. 31, no. 1, pp. 1–31, 2017.

[77] J. Mei, M. Liu, Y.-F. Wang, and H. Gao, "Learning a mahalanobis distance-based dynamic time warping measure for multivariate time series classification", *IEEE transactions on Cybernetics*, vol. 46, no. 6, pp. 1363–1374, 2015.

[78] M. R. Peterson, T. E. Doom, and M. L. Raymer, "Ga-facilitated knn classifier optimization with varying similarity measures", in *2005 IEEE Congress on Evolutionary Computation*, IEEE, vol. 3, 2005, pp. 2514–2521.

[79] J. D. RodríGuez, A. Pérez, and J. A. Lozano, "A general framework for the statistical analysis of the sources of variance for classification error estimators", *Pattern recognition*, vol. 46, no. 3, pp. 855–864, 2013.

[80] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing", *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[81] J. Kennedy, "Particle swarm optimization", *Encyclopedia of machine learning*, pp. 760–766, 2010.

[82] P. Larrañaga and J. A. Lozano, *Estimation of distribution algorithms: A new tool for evolutionary computation*. Springer Science & Business Media, 2001, vol. 2.

[83] D. Whitley, "A genetic algorithm tutorial", *Statistics and computing*, vol. 4, no. 2, pp. 65–85, 1994.

[84] J. Wang, A. Balasubramanian, L. Mojica de la Vega, J. R. Green, A. Samal, and B. Prabhakaran, "Word recognition from continuous articulatory movement time-series data using symbolic representations", 2013.

[85] D. Dua and C. Graff, *UCI machine learning repository*, 2017. [Online]. Available: `http://archive.ics.uci.edu/ml`.

[86] M. W. Kadous *et al.*, *Temporal classification: Extending the classification paradigm to multivariate time series*. University of New South Wales Kensington, 2002.

[87] K. Q. Weinberger and L. K. Saul, "Distance metric learning for large margin nearest neighbor classification", *Journal of Machine Learning Research*, vol. 10, no. Feb, pp. 207–244, 2009.

[88] J. Beringer and E. Hüllermeier, "Online clustering of parallel data streams", *Data & Knowledge Engineering*, vol. 58, no. 2, pp. 180–204, 2006.

[89] M.-Y. Yeh, B.-R. Dai, and M.-S. Chen, "Clustering over multiple evolving streams by events and correlations", *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 10, 2007.

[90] F. Zhou, F. De la Torre, and J. K. Hodgins, "Hierarchical aligned cluster analysis for temporal clustering of human motion", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 3, pp. 582–596, 2013.

[91] U. Mori, A. Mendiburu, S. Dasgupta, and J. A. Lozano, "Early classification of time series by simultaneously optimizing the accuracy and earliness", *IEEE Transactions on Neural Networks and Learning Systems*, 2017.

[92] H. A. Dau, E. Keogh, K. Kamgar, C.-C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, Yanping, B. Hu, N. Begum, A. Bagnall, A. Mueen, and G. Batista, *The ucr time series classification archive*, `https://www.cs.ucr.edu/~eamonn/time_series_data_2018/`, 2018.

[93] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar, "Learning in non-stationary environments: A survey", *IEEE Computational Intelligence Magazine*, vol. 10, no. 4, pp. 12–25, 2015.

[94] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT Press Cambridge, 2016, vol. 1.

[95] B. Krawczyk, "Learning from imbalanced data: Open challenges and future directions", *Progress in Artificial Intelligence*, vol. 5, no. 4, pp. 221–232, 2016.

[96] U. Mori, A. Mendiburu, and J. A. Lozano, "Similarity measure selection for clustering time series databases", *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 1, pp. 181–195, 2015.

[97] H. Wang, N. Wang, and D.-Y. Yeung, "Collaborative deep learning for recommender systems", in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 2015, pp. 1235–1244.

[98] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks", in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2013, pp. 6645–6649.

[99] J. Wang, Y. Ma, L. Zhang, R. X. Gao, and D. Wu, "Deep learning for smart manufacturing: Methods and applications", *Journal of Manufacturing Systems*, vol. 48, pp. 144–156, 2018.

[100] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. Van Der Laak, B. Van Ginneken, and C. I. Sánchez, "A survey on deep learning in medical image analysis", *Medical Image Analysis*, vol. 42, pp. 60–88, 2017.

[101] L. Deng, D. Yu, *et al.*, "Deep learning: Methods and applications", *Foundations and Trends® in Signal Processing*, vol. 7, no. 3–4, pp. 197–387, 2014.

[102] J. Schmidhuber, "Deep learning in neural networks: An overview", *Neural Networks*, vol. 61, pp. 85–117, 2015.

[103] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning", *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[104] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation", *PloS one*, vol. 10, no. 7, e0130140, 2015.

[105] A. B. Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. García, S. Gil-López, D. Molina, R. Benjamins, and F. Herrera, "Explainable Artificial Intelligence (XAI): concepts, taxonomies, opportunities and challenges toward Responsible AI", *Information Fusion*, vol. 58, pp. 82–115, 2020.

[106] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust physical-world attacks on deep learning visual classification", in *IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1625–1634.

[107] B. Biggio, G. Fumera, and F. Roli, "Security evaluation of pattern classifiers under attack", *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 4, pp. 984–996, 2014.

[108] K. J. Seymour, M. A. Williams, and A. N. Rich, "The representation of color across the human visual cortex: Distinguishing chromatic signals contributing to object form versus surface color", *Cerebral Cortex*, vol. 26, no. 5, pp. 1997–2005, 2015.

[109] R. Shapley and M. J. Hawken, "Color in the cortex: Single-and double-opponent cells", *Vision Research*, vol. 51, no. 7, pp. 701–717, 2011.

[110] S. Gudmundsson, T. P. Runarsson, and S. Sigurdsson, "Support vector machines and dynamic time warping for time series", in *IEEE International Joint Conference on Neural Networks*, IEEE, 2008, pp. 2772–2776.

[111] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings", in *IEEE European Symposium on Security and Privacy (EuroS&P)*, IEEE, 2016, pp. 372–387.

[112] N. Papernot and P. McDaniel, "Transferability in machine learning: From phenomena to black-box attacks using adversarial samples", *arXiv:1605.07277*, 2016.

[113] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples", *arXiv:1412.6572*, 2014.

[114] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world", *arXiv:1607.02533*, 2016.

[115] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks", *arXiv:1706.06083*, 2017.

[116] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks", in *2017 IEEE Symposium on Security and Privacy*, IEEE, 2017, pp. 39–57.

[117] C. Guo, M. Rana, M. Cisse, and L. Van Der Maaten, "Countering adversarial images using input transformations", *arXiv:1711.00117*, 2017.

[118] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning", *arXiv:1602.02697*, 2016.

[119] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks", *arXiv:1704.01155*, 2017.

[120] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks", in *IEEE Symposium on Security and Privacy*, IEEE, 2016, pp. 582–597.

[121] D. Warde-Farley and I. Goodfellow, "11 adversarial perturbations of deep neural networks", *Perturbations, Optimization, and Statistics*, vol. 311, 2016.

[122] J. Buckman, A. Roy, C. Raffel, and I. Goodfellow, "Thermometer encoding: One hot way to resist adversarial examples", 2018.

[123] G. K. Dziugaite, Z. Ghahramani, and D. M. Roy, "A study of the effect of jpg compression on adversarial images", *arXiv:1608.00853*, 2016.

[124] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, and J. C. Platt, "Support vector method for novelty detection", in *Advances in Neural Information Processing Systems*, 2000, pp. 582–588.

[125] J. Canny, "A computational approach to edge detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 6, pp. 679–698, 1986.

[126] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition", *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[127] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning", in *Deep Learning and Unsupervised Feature Learning Workshop, NIPS*, 2011.

[128] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition", *Neural networks*, vol. 32, pp. 323–332, 2012.

[129] M.-I. Nicolae, M. Sinn, M. N. Tran, B. Buesser, A. Rawat, M. Wistuba, V. Zantedeschi, N. Baracaldo, B. Chen, H. Ludwig, I. Molloy, and B. Edwards, "Adversarial robustness toolbox v1.0.1", *arXiv:1807.01069*, 2018.

[130] N. Papernot, F. Faghri, N. Carlini, I. Goodfellow, R. Feinman, A. Kurakin, C. Xie, Y. Sharma, T. Brown, A. Roy, A. Matyasko, V. Behzadan, K. Hambardzumyan, Z. Zhang, Y.-L. Juang, Z. Li, R. Sheatsley, A. Garg, J. Uesato, W. Gierke, Y. Dong, D. Berthelot, P. Hendricks, J. Rauber, and R. Long, "Technical report on the cleverhans v2.1.0 adversarial examples library", *arXiv:1610.00768*, 2018.

[131] J. Su, D. V. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks", *IEEE Transactions on Evolutionary Computation*, vol. 23(5), pp. 828–841, 2019.

[132] A. Cheddad, J. Condell, K. Curran, and P. Mc Kevitt, "Digital image steganography: Survey and analysis of current methods", *Signal processing*, vol. 90, no. 3, pp. 727–752, 2010.

[133] I. J. Kadhim, P. Premaratne, P. J. Vial, and B. Halloran, "Comprehensive survey of image steganography: Techniques, evaluations, and trends in future research", *Neurocomputing*, vol. 335, pp. 299–326, 2019.

[134] T. Miyato, S.-i. Maeda, M. Koyama, and S. Ishii, "Virtual adversarial training: A regularization method for supervised and semi-supervised learning", *arXiv:1704.03976*, 2017.

[135] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II", *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.