UNIVERSITY OF PADOVA

MASTER'S DEGREE IN TELECOMMUNICATION ENGINEERING
DEPARTMENT OF INFORMATION ENGINEERING

Unsupervised clustering of IoT signals through feature extraction and self organizing maps

Student:
Ibrahim YANKINE

Supervisor:
Prof. Michele ROSSI

Co-Supervisor:
Dr. Davide ZORDAN

Aprile 2017

Anno Accademico 2016-2017

UNIVERSITY OF PADOVA
Department of Information Engineering

Master's degree in Telecommunication Engineering

Unsupervised clustering of IoT signals through feature extraction and self organizing maps

by Ibrahim YANKINE

Abstract

The rapid growth of the Internet of Things (IoT) in different scenarios has led to the acquisition of large-scale IoT data. The time-critical extraction of meaningful information from such data is very important for a large range of applications such as environmental monitoring or vehicular traffic management, to name a few. The objective of this thesis is to build a clustering model to inspect the structural properties of a dataset composed of different IoT signal types and to classify these through unsupervised clustering algorithms. To this end, a feature-based representation of the signals is computed, obtaining a high dimensional feature space. Different feature selection algorithms from the literature are then used to obtain reduced feature spaces, so as to decrease the computational cost and the memory demand, while retaining most of the precision of classifiers built on top of the full feature set. Thus, the IoT signals are clustered using Self-Organizing Maps (SOM), which explore and assemble knowledge from the multi-class dataset, while also providing a convenient visualization tool. The performance of the proposed SOM-based approach is attained for different feature selection algorithms, achieving good classification accuracies.

Contents

Abstract **iii**

List of Figures **vii**

List of Tables **xi**

1 INTRODUCTION **1**
1.1 The Internet of Things 1
1.2 Background and motivation 2

2 DATA ANALYSIS **5**
2.1 Time Series 5
2.2 Database 7
2.3 Features extraction 8
2.3.1 Normalization 10
2.3.2 Notation and Matrix representation 11
2.4 Features selection 11
2.4.1 Feature selection types 13
2.4.1.1 Filter methods 13
2.4.1.2 Wrapper methods 14
2.4.1.3 Comparison of the two methods 14
2.4.2 Feature selection algorithms 15
2.4.2.1 Greedy forward feature selection (GFS) . 15
2.4.2.2 Mutual Information (MI) 15
2.4.2.3 Relief-F 16
2.4.2.4 Unsupervised Discriminative Feature Selection (UDFS) 17

3 GENERAL OVERVIEW **19**
3.1 Artificial neural networks 19
3.1.1 Structure of an artificial neuron 20
3.1.2 Network architectures 23
3.2 The learning paradigms 25
3.2.1 Self-Organized learning 28
3.3 Vector quantization: VQ 33
3.4 Clustering and Classification 35
3.4.1 Clustering 35
3.4.2 Different types of clustering 37
3.4.3 Clustering algorithms 38

3.4.3.1 k-means . . . 39
3.4.3.2 Hierarchical clustering algorithms . . . 39
3.4.3.3 DBSCAN . . . 41
3.4.3.4 Neural Network . . . 41
3.4.4 Classification . . . 41
3.4.4.1 k-Nearest Neighbors (k-nn) . . . 42
3.4.4.2 SOM-based classifier . . . 43
3.4.5 Conclusion . . . 43

4 SELF-ORGANIZING MAPS 45
4.1 The SOM algorithm . . . 46
4.1.1 Competition . . . 47
4.1.2 Cooperation . . . 48
4.1.3 Adaptation . . . 49
4.2 Summary of the SOM algorithm . . . 51
4.3 Properties of the SOM . . . 52

5 CLUSTERING OF THE SOM 57
5.1 Data subdivision . . . 57
5.2 Performance measures . . . 59
5.3 SOM Classifier . . . 60
5.3.1 Setup . . . 60
5.3.2 Visualization . . . 61
5.4 Two-step clustering . . . 67

6 EXPERIMENTAL RESULTS 71
6.1 Clustering case: 20 classes . . . 72
6.2 Clustering case: 12 classes . . . 82

7 CONCLUSION 89

Bibliography 91

List of Figures

2.1 Instance-based classification involves measuring the distance between pairs of time series represented as an ordered set of measurements in the time domain. The upper portion displays two time series, while in the lower portion, shaded part illustrates the distance between the two signals. 6
2.2 An alternative approach, involves representing time series using a set of features that summarize their properties. A reduced set of features is produced by features selection. A classifier can use the reduced feature-based representation of the time series to classify new time series. 7
2.3 Matrix representation of the database. 11
2.4 Database representation, operations are the features while Time series are the signals. All data are normalized between [0,1] and the values are colored according to the colorbar on the right side. 12

3.1 Biological neuron: two tipically connected neurons. . . . 21
3.2 Structure of an artificial neuron. 22
3.3 Typically used activation functions: (a) Threshold function. (b) Sigmoid function for varying slope parameter α. 23
3.4 Feedforward network: represented with a single layer of neurons. 25
3.5 Feedforward network: a multi-layer network represented with one hidden layer and one output layer of neurons. . 26
3.6 Linear input-output relation for a neuron k in the network. 32
3.7 Different types of clusters. 38

4.1 Two-dimentional SOM, Kohenen model 46
4.2 Different shapes of SOM lattice 51
4.3 Relationship between feature map Φ and synaptic-weight vector $\mathbf{w}_{i(\mathbf{x})}$ of winning neuron i. 53

5.1 Model building flowchart. 58
5.2 General scheme of the simulation model. The core of the classifier is represented by the clustering process by SOM followed by the autolabel procedure. 62

5.3 The process of the synaptic-weights update of SOM starting from an initial casual state to an ordered topology for various epochs in a 10x10 neurons map size. 64
5.4 The process of the synaptic-weights update of SOM starting from an initial casual state to an ordered topology for various epochs in a 10x10 neurons map size. 65
5.5 After synaptic-weights final update, the auto labeling result of the SOM classifier for two different epochs . . . 66
5.6 Comparison of SOM (a) and hierarachical clustering (b) result for 20 classes dataset for a 10x10 map size 68

6.1 Average precision *vs* number of features for different feature selection algorithms. Comparison for 20 classes with $L = 5 \times 5$. 73
6.2 Average quantization error *vs* number of features for different features selection algorithms. Comparison for 20 classes with $L = 5 \times 5$. 74
6.3 Average topographic error *vs* number of features for different features selection algorithms. Comparison for 20 classes with $L = 5 \times 5$. 75
6.4 Average precision vs number of features for different features selection algorithms. Comparison for 20 classes with $L = 10 \times 10$. 76
6.5 Average quantization error *vs* number of features for different features selection algorithms. Comparison for 20 classes with $L = 10 \times 10$. 77
6.6 Average topographic error *vs* number of features for different features selection algorithms. Comparison for 20 classes with $L = 10 \times 10$. 78
6.7 Average precision *vs* number of epochs for different features selection algorithms. Comparison for 20 classes with $L = 10 \times 10$ and 75 features. 79
6.8 Average quantization error *vs* number of epochs for different features selection algorithms. Comparison for 20 classes with $L = 10 \times 10$ and 75 features. 80
6.9 Average topographic error *vs* number of epochs for different features selection algorithms. Comparison for 20 classes with $L = 10 \times 10$ and 75 features. 80
6.10 The classifier label information when considering 20 classes with $L = 10 \times 10$ and 75 features for the MI selection method 81
6.11 Average precision vs number of features for different features selection algorithms. Comparison for 12 classes with $L = 10 \times 10$. 83

6.12 Average quantization error *vs* number of features for different features selection algorithms. Comparison for 12 classes with $L = 10 \times 10$. 84
6.13 Average topographic error *vs* number of features for different features selection algorithms. Comparison for 12 classes with $L = 10 \times 10$. 84
6.14 Average precision error *vs* number of features for different features selection algorithms. Comparison for 12 classes with $L = 15 \times 15$. 86
6.15 Average precision error *vs* number of features for different features selection algorithms. Comparison for 12 classes with $L = 20 \times 20$. 86
6.16 Average quantization error *vs* number of features for different features selection algorithms. Comparison for 12 classes with $L = 20 \times 20$. 87
6.17 Average topographic error *vs* number of features for different features selection algorithms. Comparison for 12 classes with $L = 20 \times 20$. 87
6.18 The classifier label information for 12 classes with $L = 10 \times 10$ and 75 features using MI 88

List of Tables

2.1 Data signals types in the dataset, the labels and the number of signals in each class. 9

5.1 Data signals types in the new dataset with the labels and the number of signals in each class. 69

To my family, to Sonia.

Chapter 1

INTRODUCTION

1.1 The Internet of Things

We live in the post-PC era. The number of people using the Internet is growing rapidly. In many places, different types of objects equipped with sensors can be found. Smartphones, PCs, vehicles, houses and handheld devices are interconnected in such a way that our everyday-life and environment adapt in a more interactive and informative manner. Consequently, the environment that surrounds us is becoming increasingly smarter. This leads to the concept of "smart environment", which was defined by the forefather of Ubiquitous Computing, Mark Weiser [1], as:

> "*A physical world that is richly and invisibly interwoven with sensors, actuators, displays, and computational elements, embedded seamlessly in the everyday objects of our lives, and connected through a continuous network*".

This is the underlying essence of the Internet of Things (IoT) paradigm, a term that was first conceived by Kevin Asthon [2]. To take full advantage of the available Internet technology, there is a need to deploy large-scale, platform-independent, wireless sensor network (WSN) infrastructures that include data management and processing, actuation and analytics. Recent advances in technologies such as micro-electro-mechanical systems, wireless communications, and digital electronics have resulted in the development of small devices having the ability to sense, compute, and communicate wirelessly over short distances [3][4]. These small, low cost and low powered sensors spatially distributed in every environment, will bring the connectivity to even the smallest objects installed in any kind of context, at an affordable cost. Smart devices that sense ambient temperature and control heating and cooling systems, sensors that manage digital features in vehicles, pacemakers for heart control and location devices that monitor the whereabouts of many types of equipment will produce a multitude of data that will spread all over to the Internet, and empower a new wave of technological developments through cloud computing, smartphones and new software applications. However, we underline that such large

amount of data will often be highly unstructured, ambiguous, and inadequate for direct use in decision making tasks. Nevertheless, some structure, although hidden, is often present in the raw data. This structure is also often useful to the development of context or data-driven applications that learn from past experience. Thus, mining useful information from unstructured and raw data appears to be a necessary step for the successful accomplishment of tasks, such as planning, decision making and strategy building.

The use of Machine Learning techniques has been rapidly expanding to many application areas. The aim is to implement efficient algorithms that are capable of extracting valuable information from IoT data streams. This is why, one of the most interesting topics in data analysis, is how to optimally use this extracted information to improve aspects in the everyday-life of individuals or in the businesses run by companies.

A typical challenge, however, is represented by the limited power capability of IoT sensors, which are often battery powered, and to the consequent need of extending their battery life. Therefore, special attention has to be paid to the design of algorithms that are computationally efficient. At the design level, particular care must be taken for the energy-optimization. New, compact devices with extremely low-power circuitry and energy efficient architectures and protocol suites, as well as energy storage sources coupled with energy transmission and harvesting methods, capable of computing data reduction based on the correlation of sensed readings can efficiently reduce the amount of required transmissions and thus improve the conservation of energy. As a matter of fact, efficient algorithms for data compression and clustering are needed in order to make such a design effective.

1.2 Background and motivation

he ever growing quantity of data that is collected everyday calls for efficient methods that are able to jointly tackle the problems of data analysis and storage. This thesis takes inspiration from this and investigates the underlying information in a given dataset composed of temporal signals collected from diverse fields, such as medicine, environmental monitoring, demography and biology.

To reach our goal, the Matlab HCTSA framework has been used to extract the feature representation of the signals in the dataset.

The work in this thesis relates to the unsupervised classification of signals in a given IoT dataset. Our two main contributions are: first, we compare different state-of-art feature selection algorithms taken from the literature and use them to obtain a reduced set of features. Second, we implement a clustering model combining three techniques, in particular:

- a *Self-Organizing Map (SOM)* for its capability to combine vector quantization and projection, and for being an excellent visualization tool (often as a 2-dimensional lattice) of the clustered feature space.
- a *hierarchical clustering* used on top of the SOM to get a better resolution and understanding of the clustering attained by the SOM.
- a *classifier* that transfers the unsupervised Self-Organizing Map method into a structured and supervised classification setting able to extract performance evaluations on unseen data once the SOM is stable (i.e., after a training process).

This thesis is organized as follows. In Chapter 2, we illustrate the dataset of the different IoT signals that are taken into consideration, and describe the different techniques used to extract meaningful features from them. In Chapter 3, we provide a general literature review about the algorithms used in this thesis, specifically analyzing them in their relation to clustering. In Chapter 4, a detailed explanation of the self-organizing map is presented. In Chapter 5, we describe the methodologies implemented and the set up for the classification and visualization of Self-Organizing Map and hierarchical clustering. In Chapter 6, numerical results of our experiments are discussed. Finally, in Chapter 7 we draw the conclusions and provide some possible future research directions.

Chapter 2

DATA ANALYSIS

2.1 Time Series

Time series, measurements of a quantity taken over time, are measured and analyzed across the scientific disciplines such as bio-signals in healthcare, gas analysis in chemical industries, rates of inflation in economics and atmospheric air temperature in climate science. In this thesis, signals gathered from different IoT scenario are considered. The enormous amount of data produced everyday has attracted extensive research interests. Researchers are working on finding efficient models solution and algorithms to tackle the problem of data analysis. The extraction of useful information from a number of time series is a fundamental passage to deal with the ever-growing need to boost data storage by using new compression mechanism and efficient transmission of data. Many instruments have been used from various summary statistics, to time-series models [5][6]. Time-series clustering and classification has conventionally been addressed by defining a distance metric approach that involves the comparison of the sequential values of time series. There are essentially two main challenges in time-series classification: the selection of an appropriate representation of the time series, and the selection of a suitable measure of dissimilarity or proximity.

The most straightforward representation of a time series is based on its time-domain form, then distances between time series relate to differences between the time-ordered measurements themselves. A traditional approach to the classification of time series problem used in data mining community, focus on classifying short time series which encode meaningful patterns. This approach is referred to as *instance-based* classification [7], where new time series are classified by matching them to similar instances of time series with a known classification (Fig. 2.1). The above method has the advantage of being easy when the modeling of a specific test instance is required but has the drawback of having high computational load when the task is performed.

An alternative approach, called *feature-based* classification, which consists in representing time series in a more robust and efficient way,

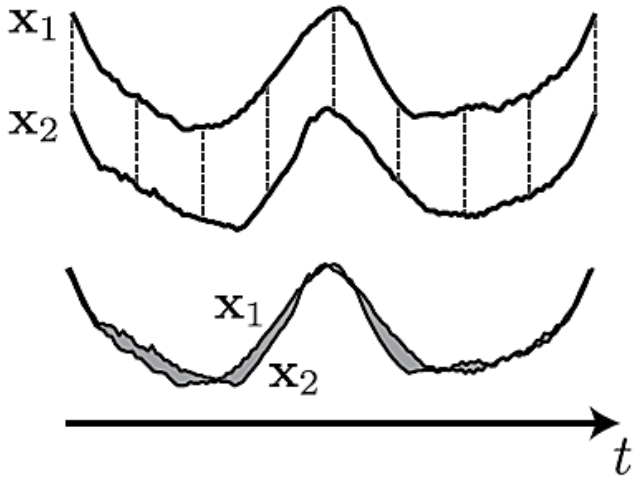


FIGURE 2.1: Instance-based classification involves measuring the distance between pairs of time series represented as an ordered set of measurements in the time domain. The upper portion displays two time series, while in the lower portion, shaded part illustrates the distance between the two signals.

uses an extracted set of attributes, or features that summarize time series properties and thereby transform the temporal problem to a static one [7]. This approach is more suited for time series corresponding to streams of data rather than the short pattern-like time series typically studied in temporal data mining. A peculiarity of this alternative method is the possibility to represent long sequences of time samples into a d-dimensional vector, where d is the number of features extracted from the input data. An important advantage of feature-based method is that clustering and classification algorithms can be used directly on the features, rather than the temporal representation of the times series. Moreover feature selection algorithms can reduce the initial set of features (extracting the most relevant ones) and potentially give applications the ability to improve their performance and reduce their complexity and computational time. The feature-based representation scheme is graphically described in Fig. 2.2.

In the next section we present the full dataset used in the experiment, section 2.3 introduces a Matlab framework used to automate the process of features extraction. In section 2.4 some features selection algorithms that are used in this work are briefly defined.

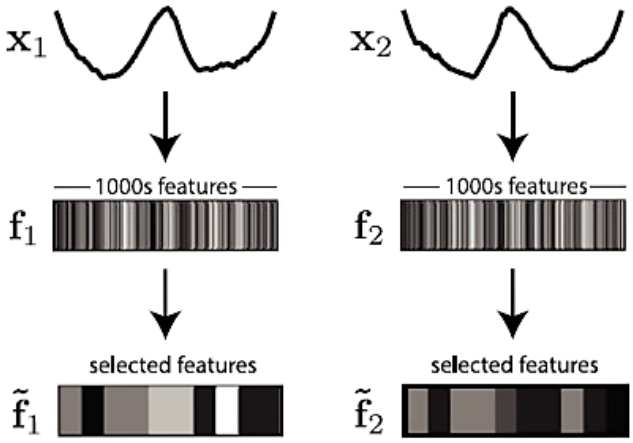


FIGURE 2.2: An alternative approach, involves representing time series using a set of features that summarize their properties. A reduced set of features is produced by features selection. A classifier can use the reduced feature-based representation of the time series to classify new time series.

2.2 Database

In this thesis, for the clustering problem, several types of signals taken from different systems and measured in many different ways are analyzed, to infer their intrinsic structure. The data are categorized and stored from three major domain: *biomedical domain*, in which data are signals collected from medical devices, *environmental domain* in which data are signals collected from outdoor/indoor devices and *structural domain*, in which data are signals collected by devices deployed to verify buildings stability. A further division of the data into different groups is made, based on the signal type. So each domain is structured as follow:

- *biomedical database* is subdivided into: Diastolic Arterial Blood Pressure (ABPdias), Accelerometer Lateral direction (AccelLateral), Accelerometer Sagittal direction (AccelSagittal), Accelerometer Vertical direction (AccelVertical), Breathing waveform (Breathing), Electrocardiogram (ECG), Heart Rate (HR), Intra Cranial Pressure (ICP), Photoplethysmogram (PPG), Pulse rate (Pulse), RR interval (RR).

- *environmental database* organized as: Humidity rate (Humidity), Solar irradiance (Solar), Superficial Temperature (SurfTemp), Temperature (Temp), Wind Direction(WindDir) ,WindSpeed (WindSpeed).

- *structural database* is subdivided into: Alarm rate (Alarm), Strain rate (Strain), Raw Reading measurement (RawReading).

The strings in the parenthesis are quick references to each respective group. In Table 2.1 we list the different signals used in our experiments, including the number of instances in each group and a numeric label used in our algorithm for groups identification.

The different signals in the dataset we are dealing with have some important issues. They come with different number of samples, in multiple cases data are missing and others have discontinuities, noisy and degraded area caused by failures in the system of acquisition. To deal with this problems prior to any elaboration, every signal has passed through a pre-processing stage in which degraded and discontinuities areas have been removed to reduce the level of corruption in the feature representation. From the initial database of signals, a new dataset has been created and it is composed by the same number of time series for each subcategory. To be more precise, 100 time series with a length of 500 samples from each subcategory has been selected. The subset of 500 consecutive samples, as well as the starting point from which this 500 consecutive samples has been extracted, has been chosen randomly from randomly chosen time series. In this way a most general and unbiased representation of the signals is chosen such that the results can be as much reliable and accurate as possible. It is worth noting that the number of samples should be chosen wisely: a low value could make the feature computation unreliable (e.g. statistical features like correlation, mean and variance need a sufficient number of samples) however a high value can raise the level of noise and outliers as well as the computational complexity. Finally the created dataset is composed by 2000 time series, build by taking 100 time series from each of the 20 groups forming the initial dataset. Note that noisy signals and outliers are still include in this final dataset, this poses a fair challenging environment for the unsupervised clustering algorithm.

2.3 Features extraction

Feature extraction is the part of the thesis in which we actually derived the features from our dataset outlined in section 2.2. To extract the feature-based representation of each time series in the database we used the Highly Comparative Time Series Analysis (HTCSA) MATLAB framework, an assembled library of time-series analysis operations [7]. Each operation is an algorithm that summarizes a time series with a single real number. The framework contains a set of over

Name	class/label	instances(#)
ABPDias	1	100
AccelLateral	2	100
AccelSagittal	3	100
AccelVertical	4	100
Alarm	5	94
Breathing	6	100
ECG	7	100
HR	8	99
Humidity	9	99
ICP	10	100
PPG	11	100
Pulse	12	96
RR	13	100
RawReading	14	97
Solar	15	100
Strain	16	97
SurfTemp	17	100
Temp	18	98
WindDir	19	100
WindSpeed	20	99

TABLE 2.1: Data signals types in the dataset, the labels and the number of signals in each class.

9000 different operations that define various time-series properties, including basic statistics of the distribution (e.g., location, spread, measures of Gaussianity, properties of outliers), linear correlations (e.g., autocorrelations, Fourier power spectrum measures), information theoretic and entropy measures (e.g., auto-mutual information, Approximate Entropy, Lempel-Ziv complexity), methods from nonlinear time-series analysis (e.g., correlation dimension, Lyapunov exponent estimates), and model fits (e.g., goodness of fit and parameter values from autoregressive moving average (ARMA) and state space models) [7]. All of these analysis methods are encoded algorithmically as operations. Each operation, ρ, is an algorithm that takes a time series $x = (x_1, x_2, \cdots, x_d)$, as input, and outputs a single real number, i.e., $\rho : \mathbb{R}^d \longrightarrow \mathbb{R}$. After applying all the different operations to each time series, an initial set of features is derived. From this initial set a cleansing operation is made to remove some "special values", resulting in a final set of features which is smaller than the initial one. "special values" output operations are for example infinities, imaginary numbers, NaNs, or values that may not be appropriate to be applied to a given time series, e.g., when a time series is too short, or when a positive-only distribution is being fit to data that is not positive.

2.3.1 Normalization

A challenging environment where several types of time series are mixed together, like in our dataset, produces many time-series operations with different distributions of outputs. Choosing the right transformation that allows them to be compared meaningfully is of crucial importance. In particular, when calculating distances between feature vectors, the range of outputs of all operations should be similar so that all operations are weighted equally. For this reasons the features have to be normalized to avoid difficulties that stem from their measurement in different units. There are a number of possible transformations that can be applied to different sets of operation outputs, e.g., *z-score*, linear rescaling to the [0,1] interval and nonlinear rescaling to the [0,1] interval. The MATLAB framework, that we adopted, produce distributions of operation outputs that are often multi-modal, and frequently contain significant outliers. To handle this issue it utilizes an outlier-robust sigmoidal transform, that is less sensitive to outliers than the other transformation [8]. Formally we have

$$\hat{\mathbf{f}} = \left\{1 + \exp\left[-\frac{\mathbf{f} - \mathrm{median}(\mathbf{f})}{1.35 \times \mathrm{iqr}(\mathbf{f})}\right]\right\}^{-1}, \tag{2.1}$$

where $\hat{\mathbf{f}}$ represents the normalized output of a given operation across all time series, $\mathbf{f}$ is the raw output, $\mathrm{median}(\mathbf{f})$ is the sample median of $\mathbf{f}$, and $\mathrm{iqr}(\mathbf{f})$ is its interquartile range. After the application of Eq.(2.1),

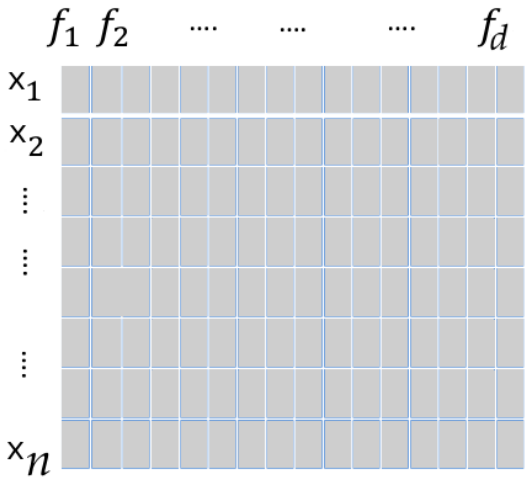


FIGURE 2.3: Matrix representation of the database.

the results are linearly rescaled to the unit interval so that every operation has the same range of normalized outputs (from 0 to 1).

2.3.2 Notation and Matrix representation

All available operations of the HCTSA toolbox have been applied to the 2000 time series in the dataset and the resulting output is composed of 1979 feature vectors each one with a dimension of 4957 features. The final dataset is smaller compare to the initial one because some time series and operations have been removed due to their "special values". Each operation is then normalized using Eq. 2.1 and the entire dataset can be represented as an $n \times d$ matrix $\mathcal{F}$ defined as;

$$\mathcal{F} = (\hat{\mathbf{f}}_1, \hat{\mathbf{f}}_2, ..., \hat{\mathbf{f}}_n) \in \mathbb{R}^{n \times d}, \tag{2.2}$$

where n is the number of feature vectors and d is the dimensionality of the features space. Let $\mathbf{X} = \{x_1, x_2, ..., x_n\}$ be the set of feature vectors in the database and $\mathbf{f} = \{f_1, f_2, ..., f_d\}$ be the set of features, such that $\mathcal{F}$ is the conventional matrix representation where elements of $\mathbf{X}$ represent rows and elements of $\mathbf{f}$ represent columns (see Fig. 2.3).
In Fig. 2.4 a graphical representation of the dataset is shown, where each row represent a time series, each column represent a feature and the color represent the normalized values.

2.4 Features selection

Time series classification in feature-based representation is characterized by data instance which are typically described by a large number

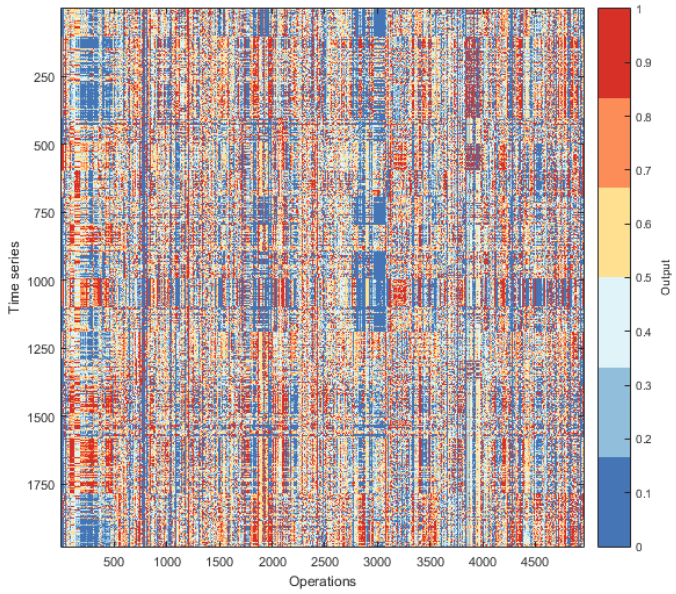


FIGURE 2.4: Database representation, operations are the features while Time series are the signals. All data are normalized between [0,1] and the values are colored according to the colorbar on the right side.

of features. Most of these features are irrelevant or redundant, which negatively affects the efficiency and the effectiveness of the learning algorithm. The selection of relevant features is a crucial task which can be used to allow a better understanding of the data or improve the performance of the learning process. Given a particular dataset with high dimensionality, it is not possible to select feature manually. Moreover, features selected for a given application usually don't fit well as best features for other applications. Furthermore, for many applications, the mechanisms underlying the data are not well understood, making it difficult to develop a well-motivated set of features for clustering/classification task.

Our objective is to find an automatic and reliable way to select features that best represent the structure of our database. The process of feature selection is completely data-driven and does not require any knowledge of the dynamical mechanisms underlying the time series or how they were measured [7]. As outlined in section 2.3.2 the framework we used produce 4957 different features for each time series in the database. That is an enormous amount of features to be processed directly and leads to what is commonly referred to as the *curse of dimensionality*. This weakens the reliability of the trained analysis systems because of the over fitting that may occur during the training process of the data. This thesis objective is to cluster different IoT signals with different characteristic, the statistical property of these signals can have an enormous impact on the success of the clustering

algorithm. If the properties of each signal fails to express the statistical regularity exploited by the learning algorithm, then learning will fail. Since we are not able to use all extracted features, different feature selection algorithms are used to build a subset of relevant features that best represent the statistic of the signals in order to reduce the complexity and the computational time. It is important to note that because they use different approaches, these algorithm produce different results, which consequently produce different performances when applied to the classification process. Each of these feature selection algorithm is well defined and has the potential to be a fully automatic and computationally tractable process.

The benefits we obtained from applying these feature selection algorithms for clustering and classification include a reduction in the amount of data needed to achieve learning, improved predictive accuracy, learning of knowledge that is more compact and easily understood, and reduced execution time.

In Chaper 7, a comparison is performed among these different approaches in order to assess which one gives the best performance for the classification purpose.

2.4.1 Feature selection types

Existing feature selection methods typically fall into two broad categories: *filter method* and *wrapper method*. Within both categories, algorithms can be further differentiated between supervised and unsupervised feature selection. Supervised feature selection scheme is defined when the dataset has known label while unsupervised feature selection scheme exploit the local structure of data distribution to selectively find the optimal features.

2.4.1.1 Filter methods

In the filter approach the feature selection method is independent of the classification algorithm to be applied to the selected features and assess the relevance of features by looking only at the intrinsic properties of the data. In most cases a feature relevance rank is built, and low-ranking features are removed. The subset of features left after feature removal is presented as input to the classification algorithm. Filter approach has the advantage to execute fast, to be computationally simple and to easily scale to high-dimensional datasets. Moreover, the filter approach is independent of the classification algorithm so feature selection needs to be performed only once, and then different classifiers can be evaluated. The principals drawbacks of filter methods are that they ignore the interaction with the classification algorithm. Moreover most of the proposed approaches are univariate which means that each

feature is considered separately, thereby ignoring feature correlations. Lastly, they do not handle noise (i.e., corrupted or incomplete data), which may lead to worse classification performance when compared to other types of feature selection algorithms.

2.4.1.2 Wrapper methods

The optimal feature subset should depend on the specific biases and heuristics of the classification algorithm. Based on this assumption, in the wrapper method the feature selection method uses the result of the classification algorithm to determine how good a given feature subset is. In this setup, a search procedure in the space of possible feature subsets is defined, and various subsets of features are generated and evaluated. Thus the quality of the best feature subset is directly measured by the performance of the classification algorithm applied. The advantages of wrapper method include the interaction between feature subset search and model selection, and the ability to take into account feature correlations. The principals drawbacks of this approach is the high risk of overfitting and high computational demand. Besides, this wrapper approach tends to be much slower than the filter approach, as the classification algorithm is applied to each feature subset considered by the search.

2.4.1.3 Comparison of the two methods

Wrappers often give better results than filters because of the retroactive error correcting information form the classification algorithm. However, the evaluation performed by the classification algorithm on each and every possible set of features makes wrappers less general than filters because the feature selection process is tightly coupled with a learning algorithm and must be repeated when switching from one learning algorithm to another. Filters on the other hand usually execute many times faster than wrappers, and therefore stand a much better chance of scaling to databases with a large number of features than wrappers do. In general, filters do not require to be re-executed when changing the learning algorithm and can sometimes provide the same benefits for learning as wrappers do. A filter can provide pre-selection for a wrapper to achieve a better performance for a particular learning algorithm. In such a way wrapper process will likely result in a shorter and faster search.

2.4.2 Feature selection algorithms

In the following subsections different feature selection algorithms used for the task of clustering and classification are outlined. Recall the notation used in section 2.2, let the usual $\mathbf{f} = \{f_1, f_2, \cdots, f_d\}$, be the initial feature set of our dataset, $\tilde{\mathbf{f}} = \{\tilde{f}_1, \tilde{f}_2, \cdots, \tilde{f}_m\} \subseteq \mathbf{f}$, with $m \leq d$, the reduced feature set and C the n-dimensional vector representing the classes (or groups) of the signals (see table 2.1). f_i is the i^{th} individual feature of $\mathbf{f}$ and $f_i(j)$ is the j^{th} component of f_i. Likewise, $C(j)$ is the j^{th} class component in vector C.

2.4.2.1 Greedy forward feature selection (GFS)

The greedy forward feature selection algorithm is a wrapper and supervised selection method used by the HCTSA framework and works as follows [8]:

- (i) Given the initial feature set $\mathbf{f}$, a classifier computes the classification scores of all individual features, f_i, and the feature with the highest classification score is selected as the first feature in the reduced set, denoted as $\tilde{f}_1$.
- (ii) The classification scores of all features in combination with $\tilde{f}_1$ are calculated and the feature that, in combination with $\tilde{f}_1$, produces the highest classification score is chosen next as $\tilde{f}_2$ and added to the reduced set.
- (iii) The procedure is repeated, choosing the operation that provides the greatest improvement in classification score at each iteration until a termination criterion is reached, yielding the reduced set of m features: $\tilde{\mathbf{f}}$. For iterations at which multiple features produce equally good classification scores, one of them is selected at random.

The algorithm terminates at the point at which the improvement in the classification score of the training set upon adding an additional feature drops below a certain threshold, or when the training set misclassification rate drops to 0 (after which no further improvement is possible). Another stopping criterion is to decide the size of the reduce set by terminating the algorithm at a prefixed number of features.

2.4.2.2 Mutual Information (MI)

This section introduces the principles of information theory by focusing on entropy and mutual information and explains the reasons of their use in feature selection. Mutual information (MI) based feature selection is a filter method which is relatively simple and efficient to

use, for such reason it is vastly utilized. It evaluates the "information content" of each individual feature with respect to the data classes and selects a subset of relevant features. Let consider the entropy (a measure of a random variable uncertainty and a measure of the average amount of information required to describe it) $H(C)$ of the class vector. If the probabilities for the different classes are $P(c)$; $c = 1, 2, ..., n$; then the initial uncertainty of the class vector is;

$$H(C) = -\sum_{c=1}^{n} P(c) \log_2 P(c), \tag{2.3}$$

while the average uncertainty knowing each feature vector of the set **f** is the conditional entropy;

$$H(C|\mathbf{f}) = -\sum_{k=1}^{d} P(f_i) \left(\sum_{c=1}^{n} P(c|f_i) \log_2 P(c|f_i) \right), \tag{2.4}$$

where $P(c|f_i)$ is the conditional probability for class c given the feature vector f_i. Formally mutual information is defined as;

$$I(C; \mathbf{f}) = H(C) - H(C|\mathbf{f}), \tag{2.5}$$

and it is symmetric with respect to C and **f** and, sometime it is reduced to the following expression:

$$I(C; \mathbf{f}) = I(\mathbf{f}; C) = \sum_{c,i} P(c, f_i) \log_2 \frac{P(c, f_i)}{P(c)P(f_i)}. \tag{2.6}$$

Therefore MI measures the amount by which the knowledge provided by the feature decreases the uncertainty about the class. The MI algorithm is described by the following procedure:

- (i) Given the initial feature set **f**, an empty set $\tilde{\mathbf{f}}$, for each feature $f \in \mathbf{f}$ compute $I(C; f)$ (the MI between the feature and the signal class).
- (ii) Find the feature $\tilde{f}$ that maximizes $I(C; f)$, add it to the reduced set $\tilde{\mathbf{f}}$ and consider **f** without the selected f for the next search.
- (iii) Repeat the procedure until $|\tilde{\mathbf{f}}| = m \leq d$.

2.4.2.3 Relief-F

RELIEF [9] and its multi-class extension Relief-F [10] select features by the use of pattern based learning to assign a relevance weight to each feature in order to separate pattern from different classes. A feature

receive a strong weight if it differentiates between patterns from different classes and has the same value for patterns of the same class. Features are ranked by weight and those that exceed a user-specified threshold are selected to form the final subset. Assume that a pattern x is randomly selected from n data samples. Then the score of the f^{th} feature W_f is defined by Relief as,

$$W_f = W_f - \frac{diff(f, x, nH)^2}{n} + \frac{diff(f, x, nM)^2}{n}, \tag{2.7}$$

where nH is the nearest-hit which is the closest same-class pattern and nM is the nearest-miss which is the closest different-class pattern. The function *diff* calculates the difference between two patterns for a given feature. When features are discrete the difference is defined as either 1 (the values are different) or 0 (the values are the same), while for continuous features the difference is the actual difference normalized to the interval [0, 1]. Dividing by n guarantees that all weights W_f are in the interval [-1, 1]. Basic RELIEF algorithm does not handle incomplete data and performs only on two class estimation problems. Relief-F is an enhancement to RELIEF that enables it to cope with multi-class, noisy and incomplete domains. Since the estimate W_f of the feature f is an approximation of difference of probabilities, to increase their reliability Relief-F searches for k nearest hits/misses instead of only one nearest hit/miss and averages the contribution of all k nearest hits/misses. And instead of randomly select one nearest-miss nM from different class, in order to deal with the multi-class issue, Relief-F selects one nearest-miss $M(c)$ for each different class in class vector C and averages their contribution for updating the estimate W_f. The average is weighted with the prior probability of each class $P(c)$ leading to the final updating function [10]:

$$\begin{aligned} W_f &= W_f - \frac{\sum_{j=1}^{k} diff(f, x, nH_j)}{n \times k} + \\ &+ \frac{\sum_{c \neq class(x)} \left[\frac{P(c)}{1 - P(class(x))} \sum_{j=1}^{k} diff(f, x, M_j(c)) \right]}{n \times k}. \end{aligned} \tag{2.8}$$

2.4.2.4 Unsupervised Discriminative Feature Selection (UDFS)

UDFS [11] is an unsupervised feature selection algorithm which analyzes features jointly and simultaneously utilizing discriminative information and local structure of data distribution. The objective of the algorithm is to select a subset with size $m \leq d$ which contains the most representative features. As a result, the data patterns represented by

the selected features can well preserve the discriminative and geometrical structure as the data represented by the original d-dimensional features. Since label information is unavailable for unsupervised feature selection, a discriminative clustering method, which is designed for seeking the most linearly separated clusters through a multi-output regularized linear regression model is used to detect the cluster structure and so predicting the clusters label.

In order to select the most discriminative features so that the separability between these clusters is maximized, for each pattern x_i, a local set $\mathcal{N}_k(x_i)$ is constructed and comprise x_i and its k nearest neighbors $x_{i_1}, ..., x_{i_k}$. Denote $\mathbf{X}_i = [x_i, x_{i_1}, ..., x_{i_k}]$ as a local data matrix. From [11] it is assumed that there is a linear classifier $W \in \mathcal{R}^{d\times c}$ which classifies each data pattern to a predicted output class and a local discriminative score for each pattern is defined. The induced regression problem, which is defined through the UDFS objective function, can be formulated as follows:

$$\min_{W^TW=I} Tr(W^TMW) + \gamma\|W\|_{2,1}, \tag{2.9}$$

where for an arbitrary matrix $\mathbf{A} \in \mathbf{R}^{r\times p}$, the $l_{2,1}$-norm is defined as;

$$\|\mathbf{A}\|_{2,1} = \sum_{i=1}^{r} \sqrt{\sum_{j=1}^{p} \mathbf{A}_{ij}^2}. \tag{2.10}$$

The regularization term $\|W\|_{2,1}$ controls the capacity of W and also ensures that W is sparse in rows, making it particularly suitable for feature selection. The orthogonal constraint, $W^TW = I$, is imposed to avoid arbitrary scaling and avoid the trivial solution of all zeros. M is the term that includes all the dependencies relative to the data local structure. If w_i denotes the i^{th} row of W, i.e., $W = [w^1, ..., w^d]^T$, the objective function shown in Eq. (2.9) can be also written as

$$\min_{W^TW=I} Tr(W^TMW) + \gamma \sum_{i=1}^{d} \|w^i\|_2. \tag{2.11}$$

From Eq. (2.11) it can be seen that many rows of the optimal W shrink to zeros. Consequently, for a data vector $\mathbf{x}_i$, $\mathbf{x}_i^{'} = W^T\mathbf{x}_i$ is a new representation of $\mathbf{x}_i$ using only a small set of selected features. Alternatively, each feature $f_i|_{i=1}^{d}$ can be ranked according to $\|w^i\|_2$ in descending order and top ranked features can be selected.

Chapter 3

GENERAL OVERVIEW

3.1 Artificial neural networks

Artificial neural networks (ANNs), are an attempt to mimic the structure of a biological system, the human brain, in a computer environment. It therefore represents one of the most promising computational tools in the artificial intelligence research area. For this reason some aspects of the way in which the brain perform information processing will be reviewed in the next paragraph.

Human brain can be described as a biological neural network, a highly complex, nonlinear and massively parallel information processing system with an interconnected web of processing elements transmitting elaborate patterns of electrical signals. The processing elements are a type of cells that does not regenerate, unlike cells in the rest of the body. That is why these cells are assumed to be the ones that allow humans and others mammals to think, remember or recall previous experiences in everyday living. These cells, called neurons, are estimated to be approximatively 10 billion in the human brain and form 60 trillion connections [12]. With the capability to organize theses cells, the brain can perform certain computations (e.g., pattern recognition, perception, and motor control) many times faster than the fastest digital computer in existence today. Consider, the human vision, for example, the brain routinely accomplishes perceptual recognition tasks (e.g., recognizing a familiar face embedded in an unfamiliar scene) in approximatively 100-200 ms, whereas tasks of much lesser complexity take a great deal longer on a powerful computer. If we compare computer and the brain, it can be noted that, theoretically, the latter should be slower than the former: in fact, computer chip comprises elements with a switching time of nanoseconds while the brain contains neurons, with a switching time of only milliseconds. However, the brain makes up for the relatively slow rate of operation by having a truly staggering number of neurons with massive interconnections between them. The key feature of the brain, which makes it an enormously efficient structure, is represented by its plasticity [13] [14], i.e., the ability to adapt the neural connections (by creating new connections and

modifying the existing ones) to the surrounding environment and then supply the information needed to interact with it.

The fundamental information-processing unit of a biological neural networks is the neuron. A typical structure of a generic brain neuron is shown in Figure 3.1 and consists of four elements, dendrites, synapses, cell body (or soma), and axon. Axons are the transmission lines, and dendrites are the receptive zones, both constitute two types of cell filaments that extrude from the soma; an axon, which comes out of the soma through the axon hillock, has a smoother surface, fewer branches, and greater length, whereas a dendrite (so called because of its resemblance to a tree) has an irregular surface and more branches (that get thinner the more they are far from the soma). Incoming signal from a neuron is transferred to another neuron by a special connection called the synapse. Such connection can usually be found at the dendrites of a neuron, sometimes also directly at the soma. This process of communication between two neurons (the presynaptic neuron and the postsynaptic neuron), is called neurotransmission or synaptic transmission [15] and typically involves electro-chemical signal from the axon of the presynaptic neuron to the dendrites (or soma) of the postsynaptic neuron. The structure of the cell body of every neuron is enclosed by a plasma membrane which is semipermeable to certain electrically charged ions. The membranes of the cells exhibit different degrees of permeability for each one of these ions. The permeability is determined by the number and size of pores in the membrane, the so-called ionic channels. Channels are principally permeable to sodium, potassium or calcium ions. The specific permeability of the membrane leads to different distributions of these ions in the interior and the exterior of the cells. This produce a voltage difference across the membrane, called the membrane potential. A typical neuron's membrane potential, in resting state, is about -70 mV. To maintain this potential an ionic pump guarantees that the concentration of ions does not change with time. A neural signal in the presynaptic neuron is initiated when its membrane potential reach a threshold potential of about -55 mV. The signal is an electric impulse called an action potential and travels rapidly along the cell's axon reaching its terminals, at the synapses. There, neurotransmitters produced by a chemical process, bind to the receptors of the postsynaptic neuron affecting its membrane potential, as a result, the signal is transmitted to the postsynaptic neuron.

3.1.1 Structure of an artificial neuron

The ANN, which is usually implemented by using electronic components or simulated in software on a digital computer, is an adaptive

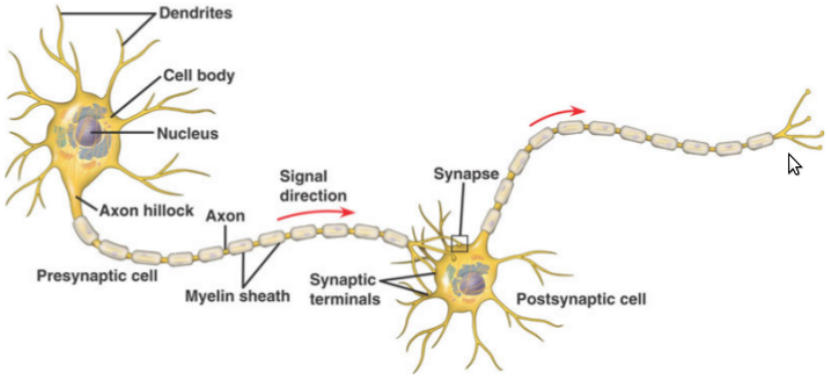


FIGURE 3.1: Biological neuron: two tipically connected neurons.

machine designed to model the way in which the brain works. However, the purpose of artificial neurons and, consequently, neural networks, is not to completely resemble biological neurons or the nervous system. Rather, ANNs are created in an attempt to understand the mechanism that enables humans to solve problems that traditional computing cannot. With the help of the artificial equivalent of the brain neurons, ANNs are used to solve a variety of problems (e.g., pattern recognition, classification [16], prediction [17], optimization, associative memory and control [18]). ANNs consist of a large number of neurons linked together by a large number of weighted connections, called *synaptic weights* that encode the network's experiential knowledge from the surrounding environment. This is done through a learning (or training) process defined as learning algorithm and consist in tuning these inter-neurons connections in an orderly fashion, according to the data given as input to the network [12]. In 1943 McCulloch, a neuroscientist, and Pitts, a logician [19] proposed a mathematical model for the artificial neuron, which is presented in Figure 3.2. Three basic elements of the neuron model can be identified:

- A set of synapses, each of which is characterized by a weight or strength. More precisely, a signal x_j at the input of synapse j connected to neuron k is multiplied by the synaptic weight w_{kj}. The synaptic weight of an artificial neuron may lie in a range that includes negative as well as positive values.
- An adder (or linear combiner) for summing the input signals, weighted by the respective synaptic weights of the neuron.
- An activation function limiting the amplitude of the output of a neuron. This function limits the permissible amplitude range of the output signal to some finite value.

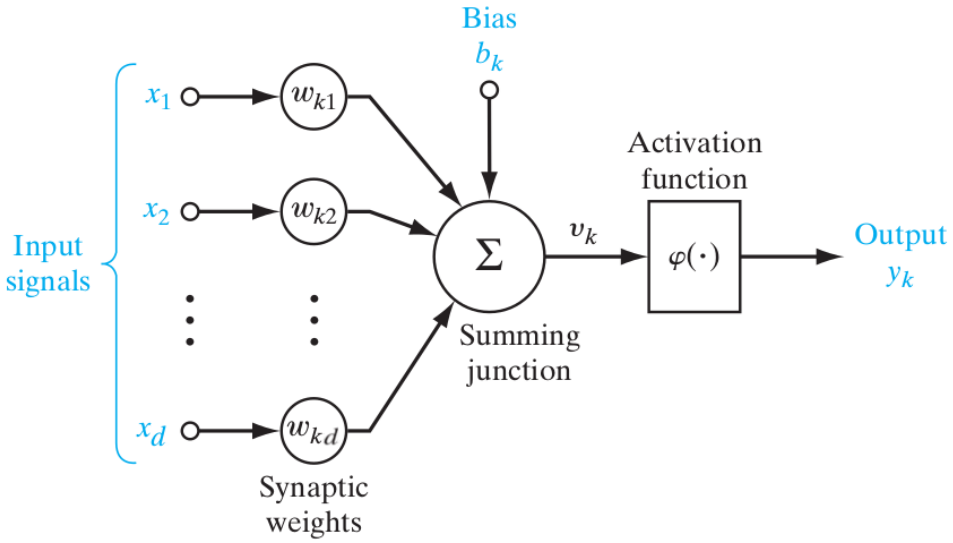


FIGURE 3.2: Structure of an artificial neuron.

The artificial neuron presented in Figure 3.2 contains an externally applied bias b_j which has the effect of increasing or lowering the net input of the activation function, depending on whether it is positive or negative, respectively. Mathematically a neuron k can be described by its output function y_k [12]:

$$y_k = \phi\left(\sum_{j=1}^{d} w_{kj}x_j + b_j\right), \tag{3.1}$$

where $x_j, j = 1, 2, .., d$ are the input signals; w_{kj} are the synaptic weights of neuron k; b_j is the bias; $\phi(\cdot)$ is the activation function; y_k is the output signal of the neuron. The activation function defines the output of a neuron k in terms of $v_k = u_j + b_j = \sum_{j=1}^{d} w_{kj}x_j + b_j$ which is referred to as the induced local field. Then, Eq. (3.1) maybe reformulated as $y_k = \phi(v_k)$. In Figure 2.4 two basic types of activation functions are presented, namely the Threshold function, Sigmoid function.

- The *threshold function* (or *Heaviside function*), given by:

$$\phi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0. \end{cases}$$

- The *sigmoid function*, whose graph is 'S'-shaped and which is by

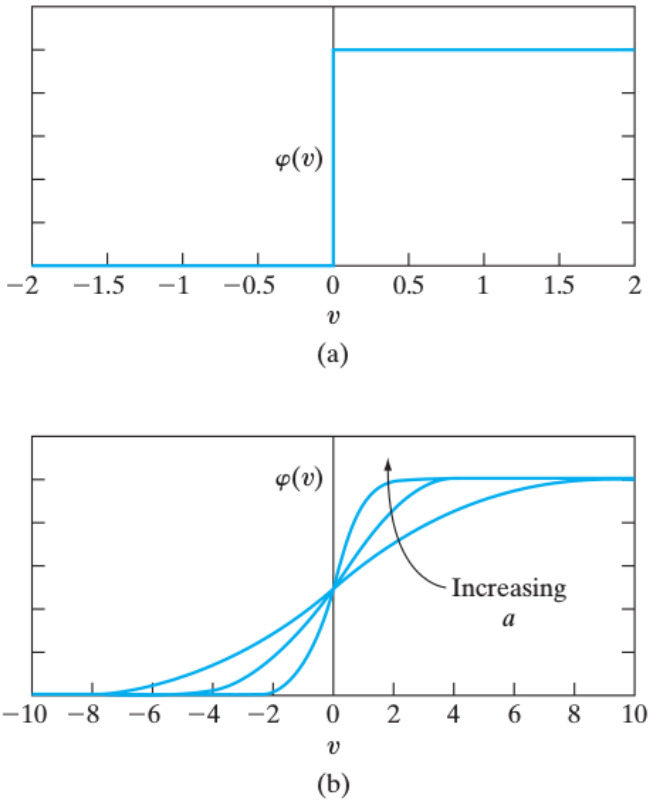


FIGURE 3.3: Typically used activation functions: (a) Threshold function. (b) Sigmoid function for varying slope parameter α.

far the most common form of activation function used in the construction of neural networks. An example of the sigmoid function is the *logistic function*, defined by

$$\phi(v) = \frac{1}{1 + e^{-\alpha v}}, \tag{3.2}$$

where α is the *slope parameter*. It has desired asymptotic properties and is a strictly increasing function that shows smoothness. In the limit, as the slope parameter approaches infinity, the sigmoid function becomes simply a threshold function. While a threshold function assumes the value of 0 or 1, a sigmoid function assumes a continuous range of values from 0 to 1. Note also that the sigmoid function is differentiable, whereas the threshold function is not.

3.1.2 Network architectures

The manner in which the neurons are structured, i.e., the *network architecture*, is intimately linked with the learning algorithm used to train

the network. We may therefore speak of learning algorithms used in the design of neural networks as being *structured*. In general, the network architectures are *layered* and may be identified in two major categories: Feed-forward networks and Feedback networks.

- In **Feed-forward** network there is only a one-directional flow of the signal, from input to output. There is no feedback, which means a layer is not affected by the components outputs of the same layer or by the components outputs of successive layers. This method associates inputs with outputs in a straightforward manner. Two types of feed-forward networks can be distinguished: single-layer feed-forward networks and multi-layer feed-forward networks.

  – The *single-layer* feed-forward network is the simplest form of layered network. This network has an input layer of source nodes that projects to an output layer of computational neurons, but not vice versa. In Figure 3.4 the case of four nodes in both the input and output layers is illustrated. It is called a single-layer network, because there is only one output layer of computation nodes, i.e., neurons.. The input layer is not taken into account because it does not perform any computation.

  – The *multi-layer* feed-forward network, shown in Figure 3.5, distinguishes itself by the presence of one or more *hidden layers*, whose computation nodes are correspondingly called *hidden neurons* or *hidden units*; the term "hidden" refers to the fact that this part of the neural network is not directly seen from either the input or output of the network. The task of the hidden units is to ensure the connection between the external input and the network output in some useful manner. The addition of hidden layers enables the network to extract higher-order statistics from its input. In a rather loose sense, the network acquires a *global* perspective despite its local connectivity, due to the extra set of synaptic connections and the extra dimension of neural interactions [14]. The source nodes supply the network with the input vectors of information, or activation patterns, which are the input signals for the second layer of neurons, or the first hidden layer. The output signals of the second layer are fed to the third layer and so on for the rest of the network. The neurons in each layer of the network have as their inputs the output signals of the preceding layer only. The set of output signals of the neurons in the final layer, output layer of the network represents the overall answer of the network to the

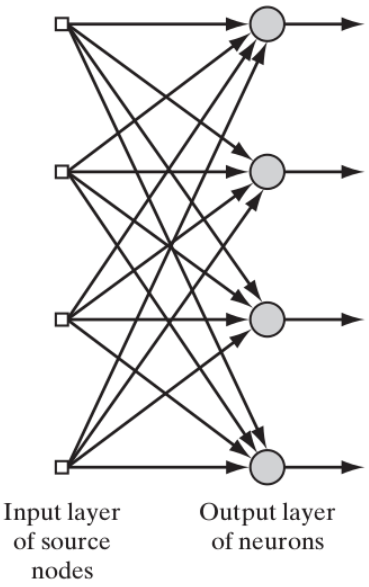


FIGURE 3.4: Feedforward network: represented with a single layer of neurons.

activation pattern introduced to the nodes in the first, input layer.

- In **Feedback Network** signals can travel in both directions by introducing loops in the network. It distinguishes itself from a feed-forward neural network in that it has at least one feedback loop that can be viewed as a single layer of neurons composing the network, where each neuron feeds its output signal back to the inputs of all other inputs. Feedback networks are dynamic systems that can be very powerful and complicated. They operate by continuously changing they state until they reach an equilibrium point. The equilibrium point is maintain until the input changes and a new equilibrium needs to be found. Feedback architectures are also referred to as *interactive* or *recurrent* networks, although the latter term is often used to denote feedback connections in single-layer structures.

3.2 The learning paradigms

Given a task, neural network require a learning algorithm to operate. Learning in neural networks can be describe by using the formulation made by Mendel and McClaren in 1970 [11].

Learning is a process by which the free parameters of a neural network are adapted through a continuing process of stimulation by the environment

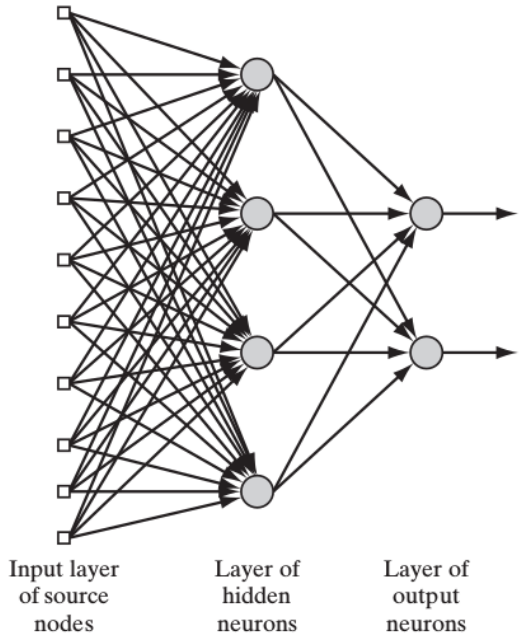


FIGURE 3.5: Feedforward network: a multi-layer network represented with one hidden layer and one output layer of neurons.

in which the network is embedded. The type of learning is determined by the manner in which the parameter changes take place.

During the occurrence of the learning process, in neural network, the updating of the network structure and synaptic weights is performed such that the network achieves the specified goals of the application of interest. The values of the synaptic weights are usually obtained by learning from an available set of input signals often defined as *training set*. The training set is composed of *patterns* or *examples* provided by the real world and used as inputs to the neural network. Such patterns can be *labeled*, in which case, each pattern is represented as an *input signal* paired with a corresponding *desired response* (i.e., target output) or *unlabeled*, where patterns are input signals with *unknown* target output. The performance of the network is smoothed with time by the iterative updates of the weights and heavily depends on the specific architecture and also on the learning process. For the design of an ANN the choice of these latter two constitute a crucial task.
In a broad sense, learning processes in neural networks, can be distinguished in two modes:

- **Supervised learning (or learning with a teacher)** is a process used to infer knowledge in a training set, which is composed

by a pair corresponding to an input pattern and a specific desired response. The goal is to build a *general model* (i.e., the network structure) during training phase that will be able to identify novel patterns presented at its input. Conceptually, the learning process adjusts the network parameters under the combined influence of the examples and the error signal, which is defined as the difference between the desired response and the actual response of the network. This adjustment is carried out iteratively in a step-by-step fashion with the aim of minimizing the error signal in accordance with an appropriate statistical criterion. The training goes on until the network reaches stability, where changes in the synaptic weights are no further noticed or significant. Supervised learning is sometime called *learning with a teacher* because the teacher or supervisor is aware of the environment in which the process is taking place. Thus, the teacher provides assistance and help to the learning process to improve the minimization of the error signal. Supervised learning is particularly used in the field of classification to deal with categorical instances and in regression for continuous instance-based prediction.

- **Unsupervised learning (or learning without a teacher)** sometimes *self-organized learning* contrary to the supervised learning performs on training set with unlabeled examples. Therefore, no prior statistics of the data jointly with their category labels are known. Hence, no teacher is there to oversee the learning process. The goal is to collect the examples into categories based only on their observable features, such that each category contains objects that share some important properties. In some cases, provision is made for a *task-independent measure* of the quality of representation that the network is required to learn, and the free parameters of the network are optimized with respect to that measure. Once the network has become tuned to the statistical regularities of the input data, it develops the ability to form internal representations for encoding features of the input data and thereby create new categories automatically.

This thesis focus on unsupervised clustering and classification of IoT signals through the use of self-organizing maps (SOMs), which is a particular case of *artificial neural network*. In Chapter 4, a detail review of the SOM architecture and the corresponding learning algorithms is presented. The principles behind self-organized learning on which SOM is built, is outlined in the following section.

3.2.1 Self-Organized learning

An important goal of unsupervised learning is to learn from its environment (i.e., unlabeled inputs) and, through training, to improve its performance in such a way that the underlying structure of the data is unveiled and well represented. The principles on which it is based are described in the following [12]:

Principle 1. Self-amplification

This first principle of self-organization states the following:
Modifications in the synaptic weights of a neuron tend to self-amplify in accordance with Hebb's postulate of learning, which is made possible by synaptic plasticity.

At the local level, for a single neuron, the modifications of the synaptic weights must be based on available pre-synaptic and post-synaptic signals for *self-amplification* to occur. The requirements of self-amplification and locality specify a feedback mechanism, by means of which a strong synapse leads to the coincidence of pre-synaptic and post-synaptic signals. In turn, the synapse is increased in strength by such a coincidence. The mechanism described here is the very essence of Hebbian learning.
The oldest and most famous of all learning rules is certainly the *Hebb's postulate of learning*; it is named in honor of the neuropsychologist Hebb. Hebb's book *The Organization of Behavior* (1949) asserts the following (p. 62):

When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic changes take place in one or both cells such that A's efficiency as one of the cells firing B is increased.

Thus the modification of the weights of a cell A, will gain major efficiency through a feedback mechanism from influencing a near neuron B. This explains the plasticity of neurons in a self-organized network. The statement on Hebb's postulate of learning is made in a neurobiological context. These requirements can be expanded and rephrased in a two-part rule according to [20] [21].

1. If two neurons on either side of a synapse are activated simultaneously (i.e.,synchronously), then the strength of that synapse is selectively increased.

2. If two neurons on either side of a synapse are activated asynchronously, then that synapse is selectively weakened or eliminated.

Such a synapse is called a *Hebbian synapse* (The original Hebb's rule did not containpart 2). An Hebbian synapse can be seen as

a synapse that uses a *time-dependent, highly local, and strongly interactive mechanism to increase synaptic efficiency as a function of the correlation between the presynaptic and postsynaptic activities*. From this definition, given by [22], the following four key mechanisms (properties) that characterize Hebbian learning can be deduced:

1. *Time-dependent mechanism.* This mechanism refers to the fact that the modifications in a Hebbian synapse depend on the exact time of occurrence of the presynaptic and postsynaptic signals.

2. *Local mechanism.* By its very nature, a synapse is the transmission site where information-bearing signals (representing ongoing activity in the presynaptic and postsynaptic units) are in *spatiotemporal* contiguity. This locally available information is used by a Hebbian synapse to produce a local synaptic modification that is input specific.

3 *Interactive mechanism.* The occurrence of a change in a Hebbian synapse depends on signals on both sides of the synapse. That is, the Hebbian form of learning depends on "true interaction" between presynaptic and postsynaptic signals in the sense that we cannot make a prediction from either one of these two activities by itself. Note also that this dependence or interaction may be deterministic or statistical in nature.

4 *Conjunctional or correlational mechanism.* One interpretation of Hebb's postulate of learning is that the condition for a change in synaptic efficiency is the conjunction of presynaptic and postsynaptic signals. Thus, according to this interpretation, the co-occurrence of presynaptic and postsynaptic signals (within a short interval of time) is sufficient to produce the synaptic modification. It is for this reason that a Hebbian synapse is sometimes referred to as a *conjunctional synapse*. For another interpretation of Hebb's postulate of learning, we may think of the interactive mechanism characterizing a Hebbian synapse in statistical terms. In particular, the correlation over time between presynaptic and postsynaptic signals is viewed as being responsible for a synaptic change. Accordingly, a Hebbian synapse is also referred to as a *correlational synapse*. Correlation is indeed the basis of learning [23].

To formulate Hebbian learning in mathematical terms, consider a synaptic weight w_{kj} of neuron k with presynaptic and postsynaptic signals denoted by x_j and y_k, respectively. The adjustment applied to the

synaptic weight w_{kj} at time-step n is expressed in the general form as

$$\Delta w_{kj}(n) = w_{kj}(n+1) - w_{kj}(n) = f(y_k(n), x_j(n)), \tag{3.3}$$

where $f(\cdot)$ is a function of both postsynaptic and presynaptic signals. The signals $x_j(n)$ and $y_k(n)$ are often treated as dimensionless. The function f in Eq. (3.3) admits many forms, a special case and the simplest form of Hebbian learning is described by

$$\Delta w_{kj}(n) = \eta y_k(n) x_j(n), \tag{3.4}$$

where η is a positive constant that determines the *rate of learning* and is referred to as the *learning-rate*. Eq. (3.4) clearly emphasizes the correlational nature of a Hebbian synapse which refers exclusively to excitatory synapses, and has the unfortunate property that it can only increase synaptic weights by the repeated application of the input signal (presynaptic activity) x_j, thus washing out the distinctive performance of different neurons in a network, as the connections drive into saturation. At that point, no new information will be stored in the synapse, and selectivity is lost. However, when the Hebbian rule is augmented by a stabilization rule, (e.g. keeping constant the total strength of synapses upon a given neuron), it tends to "sharpen" a neuron's predisposition, causing its firing to become better and better correlated with a cluster of stimulus patterns. The stabilization rule is taken care of in the second principle.

Principle 2. Competition

This second principle of self-organization states the following:

The limitation of available resources, in one form or another, leads to competition among the synapses of a single neuron or an assembly of neurons, with the result that the most vigorously growing (i.e., fittest) synapses or neurons, respectively, are selected at the expense of the others.

This second principle is made possible by synaptic plasticity (i.e., adjustability of a synaptic weight). For a given single neuron to stabilize, for example, there must be competition among its synapses for limited resources (e.g., energy) in such a way that the increase in strength of some synapses in the neuron is compensated for by a decrease in strength of others. Accordingly, only the "successful" synapses can grow in strength, while the less successful synapses tend to weaken and may eventually disappear altogether. One way to introduce competition among the synapses of a neuron is to incorporate some form of *normalization* in the learning rule for the adaptation of the synaptic weights. The basic form of η leads to unlimited growth of the synaptic weights,

which is unacceptable on physical grounds, so the effect of normalization is essential for stabilization. By re-writing Eq. (3.4) as

$$w_{kj}(n+1) = w_{kj}(n) + \eta y_k(n) x_j(n), \tag{3.5}$$

normalization is applied as follow [24]:

$$w_{kj}(n+1) = \frac{w_{kj} + \eta y_k(n) x_j(n)}{\left(\sum_{j}^{d} (w_{kj} + \eta y_k(n) x_j(n))^2 \right)^{1/2}}, \tag{3.6}$$

where the summation in the denominator extends over the complete set of synapses associated with the neuron k and d is the cardinality of the synapses. If parameter η is small and a linear model of neuron k as depicted in Figure 3.6 is used then the denominator of Eq. (3.6) can be expanded as a power series in η, demonstrating that its final form is [12]

$$w_{kj}(n+1) = w_{kj}(n) + \eta y_k(n)(x_j(n) - y_k(n) w_{kj}(n)), \tag{3.7}$$

where the term $y_k(n)x_j(n)$ represents the usual Hebbian modifications to synaptic weight w_{kj} and therefore accounts for the self-amplification effect dictated by Principle 1 of self-organization. The negative part $-y_k(n)w_{kj}(n)$ is responsible for stabilization in accordance with Principle 2, which requires competition among the synapses of the neurons. It is related to a *forgetting*, or *leakage* term, that is frequently used in learning rules, but with the difference that it becomes more pronounced with a stronger post-synaptic signal $y_k(n)$.
At the network level, a competitive process may prevail by proceeding as follows [25]:

- To begin with, the neurons in the network are all the same, except for some randomly distributed synaptic weights; thus, the neurons respond differently to a given set of input patterns.

- A specific limit is imposed on the "strength" (e.g., the sum of synaptic weights) of each neuron in the network.

- The neurons compete with each other in accordance with a prescribed rule for the right to respond to a given subset of inputs; consequently, only one output neuron, or one neuron per group, is active at a time. The neuron that wins the competition is called a *winner-takes-all neuron* (or *winning* neuron or *best-matching unit (BMU)*).

In *competitive-learning process* the output neurons of a neural network compete among themselves for being the one to be *active*

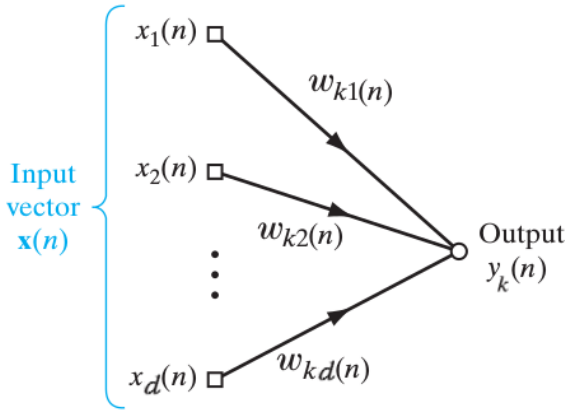


FIGURE 3.6: Linear input-output relation for a neuron k in the network.

(*fired*). Thus, whereas in a neural network based on Hebbian learning several output neurons may be active simultaneously, in the case of competitive learning, only a single output neuron, or one output neuron per group, is active at any time. It is this feature that makes competitive learning highly suited to discover those statistically salient features that may be used to classify a set of input patterns.
Accordingly, the individual neurons of the network learn to specialize on sets of similar patterns, and thereby become *feature detectors*.

Principle 3. Cooperation

This third principle of self-organization states the following:

Modifications in synaptic weights at the neural level and in neurons at the network level tend to cooperate with each other.

The cooperation may arise because of synaptic plasticity or because of simultaneous stimulation of presynaptic neurons brought on by the existence of the right conditions in the external environment. Consider first the case of a single neuron: a single synapse on its own cannot efficiently produce favorable events. Rather, there has to be cooperation among the neuron's synapses, making it possible to carry coincident signals strong enough to activate that neuron. At the network level, cooperation may take place through *lateral interaction* among a group of *excited* neurons. In particular, a neuron that is firing tends to excite the neurons

in its immediate neighborhood more so than those farther away from it. Over the course of time, we typically find that a cooperative system evolves through a sequence of small changes from one configuration to another, until an equilibrium condition is established. It is also important to note that in a self-organizing system that involves both competition and cooperation, *competition always precedes cooperation*.

Principle 4. Structural Information

This fourth, and last principle of self-organization states the following:

The underlying order and structure that exist in an input signal represent redundant information, which is acquired by a self-organizing system in the form of knowledge.

Structural information contained in the input data is therefore a prerequisite to self-organized learning. It is also noteworthy that whereas self-amplification, competition, and cooperation are processes that are carried out within a neuron or a neural network, structural information, or redundancy, is an inherent characteristic of the input signal. Consider, for example, a voice or video signal. When such a signal is sampled at a high rate, the resulting sampled signal is correspondingly found to exhibit a higher degree of *correlation* between adjacent samples. The meaning of this high correlation is that, on average, the signal does not change rapidly from one sample to the next, which, in turn, means that the signal contains *structured*, or *redundant*, information. In other words, correlation is synonymous with structure and redundancy. To appreciate the importance of structure, suppose that all the redundant information contained in a signal is completely removed. What we are then left with is a completely non redundant signal that is unpredictable and may therefore be indistinguishable from noise. Given this kind of an input, no self-organizing or unsupervised-learning system can function.

3.3 Vector quantization: VQ

Vector quantization (VQ) is a lossy data/pattern compression method based on the concept of block coding, derived from the fundamental result of Shannon's rate-distortion theory. The VQ design problem can be stated as follows. Given a vector source with its statistical properties known, given a *distortion measure*, and given the number of *codevectors*, find a **codebook** and a **partition** which result in the smallest average distortion.

Assume a training data sequence *input pattern* consisting of N input vectors:

$$\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}, \tag{3.8}$$

with *probability density function* $p_{\mathbf{x}}(\mathbf{x})$ and N is assumed to be sufficiently large so that all the statistical properties of the input are captured by the training data sequence. We assume that the input vectors are d-dimensional, i.e.,

$$\mathbf{x}_i = (x_{i,1}, x_{i,2}, ..., x_{i,d}), \quad i = 1, 2, ..., N. \tag{3.9}$$

Let L be the number of codevectors and let

$$\mathcal{C} = \{\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_L\}, \tag{3.10}$$

represents the codebook. Each codevector is d-dimensional, i.e.,

$$\mathbf{y}_l = (y_{l,1}, y_{l,2}, ..., y_{l,d}), \quad l = 1, 2, ..., L. \tag{3.11}$$

Let $\mathcal{S}_l$ be the decision region associated with codevector $\mathbf{y}_l$ and let

$$\mathcal{P} = \{\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_L\}, \tag{3.12}$$

denote the partition of the space. If the input vector $\mathbf{x}_i$ is in the decision region $\mathcal{S}_l$, then its approximation (denoted by *quantization rule* $q(\mathbf{x}_i)$) is $\mathbf{y}_l$:

$$q(\mathbf{x}_i) = \mathbf{y}_l, \quad \text{if} \quad \mathbf{x}_i \in \mathcal{S}_l. \tag{3.13}$$

Assuming a *squared-error distortion measure*, the average distortion is given by:

$$d(\mathbf{x}_i, \mathbf{y}_l) = \|\mathbf{x}_i - \mathbf{y}_l\| = \sum_{k=1}^{d} (x_{ik} - y_{lk})^2, \tag{3.14}$$

where $\|.\|$ denote the *Euclidean distance*. A quality measure can be inferred by the *mean squared error* (MSE) or by the *root mean squared error* (RMSE):

$$\begin{aligned} \text{MSE} &= E[d(\mathbf{x}, \mathbf{y}_l)] = \sum_{l=1}^{L} \int_{\mathcal{S}_l} \|\mathbf{u} - \mathbf{y}_l\|^2 f_{\mathbf{x}}(\mathbf{u}) d\mathbf{u}, \\ \text{RMSE} &= \sqrt{E[d(\mathbf{x}, \mathbf{y}_l)]} = \sqrt{\sum_{l=1}^{L} \int_{\mathcal{S}_l} \|\mathbf{u} - \mathbf{y}_l\|^2 f_{\mathbf{x}}(\mathbf{u}) d\mathbf{u}}. \end{aligned}$$

The design of an optimal VQ can be succinctly defined as follow: given $\mathcal{X}$ and L, determine $\mathcal{C}$ and $\mathcal{P}$ such that the average distortion is minimized. For $\mathcal{C}$ and $\mathcal{P}$ to be solution of the above minimization problem, they must satisfy the following two optimality criteria.

1. **Nearest Neighbor Condition (NNC)**:

$$\mathcal{S}_l = \{\mathbf{x} : d(\mathbf{x}, \mathbf{y}_l) \leq d(\mathbf{x}, \mathbf{y}_j) \quad \forall l \neq j\}. \tag{3.15}$$

   This condition implies that the optimal decision region $\mathcal{S}_l$ should consists of all vectors that are closer to $\mathbf{y}_l$ than to any of the other codevectors. Hence, eq. (3.13) can equivalently be written as, $\mathcal{Q}(\mathbf{x}) = \text{argmin}_{y_l}(\mathbf{x}_i - \mathbf{y}_l)$, i.e., $\mathbf{y}_l$ satisfying the equation is the best approximated codevector to the input vector $\mathbf{x}_i$.

2. **Centroid Condition (CC)**:

$$\mathbf{y}_l = \frac{\sum_{\mathbf{x}_i \in \mathcal{S}_l} \mathbf{x}_i p_{\mathbf{x}}(\mathbf{x}_i)}{\sum_{\mathbf{x}_i \in \mathcal{S}_l} p_{\mathbf{x}}(\mathbf{x}_i)}, \quad l = 1, 2, ..., L, \tag{3.16}$$

   This condition says that the codevector $\mathbf{y}_l$ or *centroid* should be the average of all those training vectors that are in decision region $\mathcal{S}_l$. At least, one training vector should be present into each decision region and this is accounted by the probability mass function, $p_{\mathbf{x}}(\mathbf{x}_i)$.

The classical vector quantization technique to achieve such a mapping i.e., that satisfies the above two conditions is the LBG algorithm, introduced in [26].

3.4 Clustering and Classification

In this section, we will review some of the literature on data clustering and data classification.

3.4.1 Clustering

Clustering is the most popular form of automatic unsupervised data analysis. It can be viewed as a type of *vector quantization* process or *summarization* in which the detailed data within a data set are abstracted and compressed to a smaller set of class descriptions, one for each class, that summarize the characteristics of the data in each data subset defined as *cluster*. So clustering is a descriptive data analysis task, that aims at finding the intrinsic structure in collection of objects by grouping them into clusters (homogeneous region), based on the values of their features, such that those within each cluster are more closely related to one another than objects assigned to different clusters.

Clustering is an unsupervised learning task. Therefore, given a data set of unlabelled data a clustering algorithm tries to group them to more meaningful clusters. As a second step assign a label will be assigned to each cluster, providing a means for generalizing over the

data objects and their features, in contrast to supervised learning (i.e., *classification*), where for the data set a label or target is already given to the patterns (training set). Clustering is very useful especially for large and high dimensional datasets as it provides a simplification of the underlying data distribution, and it also helps to uncover hidden structure and knowledge. A drawback in clustering, is that, the idea of approximating a group of similar data inputs using few clusters descriptors has the consequence of losing fine details (the same issue is found in general data compression methods). Since there is no universal definition of clustering, there is no universal measure with which to compare clustering algorithms. Many of these algorithms have been extensively studied in a wide variety of disciplines including signals processing, information retrieval [27], biology, statistics, pattern recognition, machine learning and data mining [28][29][30].

In cluster analysis an important parameter to be be defined is a measure of similarity (or dissimilarity) between the individual objects being clustered. One of the most popular method to measure the similarity between two vectors, suppose d-dimensional space, is the *Euclidean distance*:

$$d(\mathbf{x}_1, \mathbf{x}_2) = \|\mathbf{x}_1 - \mathbf{x}_2\| = \sqrt{\sum_{r=1}^{d} (x_{1r} - x_{2r})^2}, \tag{3.17}$$

where $\|\cdot\|$ represent the norm.
The clustering criterion then is expressed by a cost function or some other type of transition rule that assesses the quality of a given grouping.

Goal of clustering

The goal of clustering is to inspect the underlying data distribution structure with the purposes of improving research for understanding hidden knowledge. In the case of well-separated clusters and few dimensional data set (e.g., up to 3-dimensions), an eye inspection using visualization techniques to better understand the data distribution can suffice, sometimes. However, when dimensionality start to increase, examining the structure of data become quite difficult. Thereby, robust clustering algorithms operating in the full-dimensional data space, with the ability to produce useful output that can be easily inspect by users, are needed. Clustering is not an easy task and has to deal with a variety of problems, some of which are:

- *Handling High Dimensionality*. Often, complex real-world concepts are accompanied by a large number of features. This force an estimator (e.g., a classifier) to deal with a high number of features to train with and therefor, to be able generalize afterward.

Within these features, often, many are either redundant or irre levant and their use usually impacts on the complexity and the memory demand in implementation.

- *Heterogeneity of clusters*. Distance based clustering algorithms tend to find spherical clusters with similar size and density. Clustering algorithms which are able to detect clusters of arbitrary shape, size, density, and data coverage, would help in gaining a deeper insight into the different correlations between the features which, in turn, can greatly facilitate the decision making processes.

- *Interpretability of the Results*. High dimensional spaces are cumbersome even for the most advanced visualization techniques. Many clustering algorithms can produce different results. It is essential to have cluster descriptors that can be easily assimilated by the final user.

3.4.2 Different types of clustering

Amongst clustering algorithms there is a distinction based on how they regroup the objects. In the following, various types of clusterings are distinguished:

- *Hierarchical versus Partitional*: partitional clustering is a simple division of a data set into non-overlapping clusters such that each object of the dataset is assigned to exactly one cluster. Instead in hierarchical clustering a nested regrouping is perform, and subclusters are created. This gives hierarchical clustering a tree-base structure where each node on the tree is the union of its children. The leaves of the tree are often singleton, i.e., clusters of individual data objects.

- *Exclusive versus Overlapping versus Fuzzy*: exclusive clustering assigns each object to a single cluster. Which means that a data object belongs to exactly one cluster. In overlapping (or nonexclusive) clustering, the general case, an object may be assigned to multiple clusters. For instance, a grouping of people by age and sex is exclusive whereas a grouping by disease category is nonexclusive since a person can have several diseases at the same time. Finally, in fuzzy clustering, each object belongs to every cluster with a membership probability value (membership weight) between 0 (does not belong) to 1 (absolutely belong).

- *Complete versus Partial*: partial clustering only cluster certain portion of the dataset, the objects of interest and discard the rest. By contrast, a complete clustering assigns every object to a cluster, regardless of importance and interest.

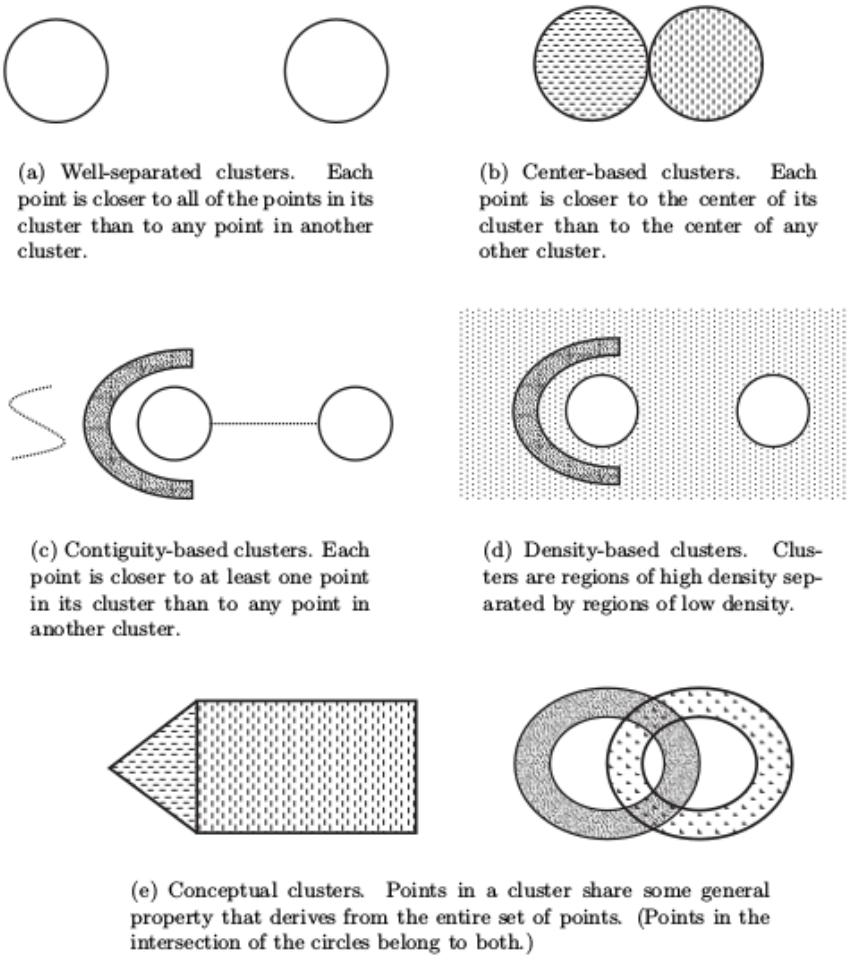


FIGURE 3.7: Different types of clusters.

Figure 3.7 represents different types of clusters in which the majority are exclusive clusters except for the case in (e) where the second cluster (right) is overlapping.

3.4.3 Clustering algorithms

This section briefly describes a selection of popular clustering methods which are categorized based on their cluster model, some listed above. As already stated, there is no objectively "correct" clustering algorithm. Often, for a particular problem, it needs to be chosen experimentally, unless there is a mathematical reason to prefer one cluster type over another. The following overview will only list the most prominent examples of clustering algorithms.

3.4.3.1 k-means

k-means clustering is an unsupervised clustering algorithm, based on the Vector Quantization (Section 3.3) method. k-means aim is to assign n inputs to a pre-defined number of clusters k, so that each input belongs to the cluster with the nearest *mean* (or centroid), serving as the prototype of the cluster. Typically, k initial seeds are randomly selected and iteratively re-organised by assigning each object to its closest centroid. Then, new means are calculated to be the centroids of the objects in the new clusters. This process goes on until no further changes take place, i.e., when the assignments no longer change. The optimizing criterion in the clustering process is the sum-of-squared error between the objects in the clusters and their respective centroids. The k-means algorithm is sensitive to the choice of the initial k value, so at the initialization k should be a varying parameter to be optimize. k-means generally works well on datasets with isotropic cluster shape, since it tends to create compact clusters.

3.4.3.2 Hierarchical clustering algorithms

Hierarchical algorithm divides a data set into a sequence of nested partitions to obtain a number k of clusters, the hierarchy is cut at the relevant depth. Depending on whether the clustering is performed top-down, i.e. from a single cluster (root) to the maximum number of clusters (singletons), or bottom-up, i.e. from the maximum number of clusters to a single cluster, two types of hierarchical algorithm can be distinguished:
Agglomerative hierarchical clustering starts with every single object in a single cluster. Then it repeats merging the closest pair of clusters according to some similarity criteria until all of the data are in one cluster. There are some disadvantages for agglomerative hierarchical clustering, such as data pattern that have been incorrectly grouped at an early stage cannot be reallocated and different similarity measures for measuring the similarity between clusters may lead to different results.
Divisive hierarchical clustering starts with all objects in one cluster and repeats splitting large clusters into smaller pieces. It has the same drawbacks as agglomerative hierarchical clustering. Another disadvantage of divisive clustering is that it is computationally more problematic than agglomerative clustering, because it needs to consider all possible divisions into subsets. In combination with SOM, the agglomerative clustering is used to assess the performance of our clustering approach.

Agglomerative hierarchical clustering

Agglomerative clustering algorithm [31][32] generates clusters by a sequence of merge operations in a bottom-up fashion. Agglomeration

process starts by initializing each input pattern as its own cluster. Then the two closest clusters based on a certain criterion, often measuring the proximity between clusters, are merged at each step and the process is repeated until the desired number of clusters has been obtained. There are different possible merging rules on which hierarchical algorithm works and based on their implementation different clustering solutions can be produced. Usually in hierarchical methods a tree diagram called, *dendrogram*, which represents the nested grouping of objects and similarity levels is used to illustrate the result of the clustering process. The clusters are obtained by cutting the dendrogram at the desired similarity level. In the general cases where merging rules are considered, agglomerative hierarchical clustering method can be divided in the following algorithms: *single linkage*, *complete linkage*, *unweighted pair-group method using averages* and *Ward's Linkage*. Each of which is described below.

Let define $\mathcal{A}$ and $\mathcal{B}$ as two clusters, with cardinality $|\mathcal{A}|$ and $|\mathcal{B}|$ respectively, and $d(\cdot,\cdot)$ the euclidean distance metric defined by Eq. (3.17)

- *Single Linkage* method (also known as the *connectedness*, the *minimum* method or the *nearest neighbor* method) is one of the simplest hierarchical clustering methods. It considers the distance between two clusters to be equal to the shortest distance from any member of one cluster to any member of the other cluster. The distance is defined by the two most similar objects, mathematically:
$$D(\mathcal{A},\mathcal{B}) = \min_{\mathbf{x}\in\mathcal{A},\mathbf{y}\in\mathcal{B}} d(\mathbf{x},\mathbf{y}), \tag{3.18}$$

- *Complete Linkage* method (also called the *maximum* method or the *furthest neighbor* method) Unlike the single-link method, uses the furthest neighbor distance to measure the dissimilarity between two clusters as:
$$D(\mathcal{A},\mathcal{B}) = \max_{\mathbf{x}\in\mathcal{A},\mathbf{y}\in\mathcal{B}} d(\mathbf{x},\mathbf{y}), \tag{3.19}$$

- *Average Linkage* method is also referred as *UPGMA*, which stands for "unweighted pair group method using arithmetic averages". In the group average method, the distance between two clusters is defined as the average of the distances between all possible pairs of data points that are made up of one data point from each cluster.
$$D_{avg}(\mathcal{A},\mathcal{B}) = \frac{1}{|\mathcal{A}|\cdot|\mathcal{B}|}\sum_{\mathbf{x}\in\mathcal{A}}\sum_{\mathbf{y}\in\mathcal{B}} d(\mathbf{x},\mathbf{y}), \tag{3.20}$$

- *Ward's Linkage* Ward's method is also known as the *minimum variance method*, implements a merging rule consisting in the mini

mization of the total within-cluster variance. At each step it finds the pair of clusters that leads to the minimum increase in the total within-cluster variance after merging. This increase is a weighted squared distance between *centroids* i.e., cluster centers, based on the Euclidean distance.

$$\Delta(\mathcal{A}, \mathcal{B}) = \frac{|\mathcal{A}| \cdot |\mathcal{B}|}{|\mathcal{A}| + |\mathcal{B}|} d(\mu_{\mathcal{A}}, \mu_{\mathcal{B}})^2, \tag{3.21}$$

where $\mu_{\mathcal{A}}$ and $\mu_{\mathcal{B}}$ are the centroids of the two clusters respectively and Δ is the merging cost.

3.4.3.3 DBSCAN

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) search for inputs objects whose neighborhoods are within a pre-assign ϵ radius that contains at least a minimum number of other inputs equal to *MinPts*. A set of core objects with overlapping neighborhoods define the skeleton of the cluster. Objects lying inside the neighborhood of core objects without being cores themselves are considered to be the boundaries of the clusters, while the remaining are labeled as noises or outliers. DBSCAN can discover arbitrary-shaped clusters, is insensitive to outliers and order of the data input. DBSCAN fails to cluster dataset with high-dimensional spaces consisting in large differences in densities and is very sensitive to the input parameters ϵ and *MinPts*, which must be specified by the user.

3.4.3.4 Neural Network

Advanced methods such as neural networks, especially those falling in the ANNs category (see Section 3.1), e.g. represented by Kohonen self-organizing maps (SOMs), have been used extensively for both clustering and classification, and have established that neural networks are a promising alternative to various conventional clustering methods. SOM provide a way to represent multidimensional data in a smaller dimensional space, usually with just one or two dimensions. The process of reducing the dimensionality of vectors is similar to the Vector Quantization. In addition, the Self-organizing map creates a network able to store information into neurons, such that any topological relationship within the input data set is maintained.

3.4.4 Classification

Classification is the problem of identifying to which among a set of clusters a new object (or instance) belongs, on the basis of a dataset containing objects whose cluster membership, i.e., *class-label* is known.

To acquire the knowledge that allows the discrimination of different clusters, a *classifier* goes through the learning process. During the learning process, in the classification task, the set of examples being treated is divided into two mutually exclusive and exhaustive sets, called the training set and the test set. The classification process is correspondingly divided into two phases: training, when a classification model is built from the training set and testing, when the model is evaluated on the test set. In the training phase the algorithm has access to the values of both examples and the desired response of the training set, and it uses that information to build a classification model. This model represents classification knowledge, essentially, a relationship between examples and classes that allows the prediction of the class of any unseen example given as input to the previous built model. For testing, the examples in the test set are unseen data. In the testing phase, first a prediction of the class for each example is made, then the algorithm is allowed to see the actual class. By confronting the predicted class and the actual class a performance curve can be drawn and analyzed. One of the major goals of a classification algorithm is to maximize the predictive accuracy obtained by the classification model when classifying examples in the test set, unseen during training. In the following the two classifiers used in this work are explained.

3.4.4.1 *k*-Nearest Neighbors (*k*-nn)

The k-nn is an instance-based method used to classify an object based on its distance to each member of the training set. The training examples are mapped into a multi-dimensional feature space, which is partitioned into regions by the class labels of the training samples. A point in the space is assigned to the most frequent class label among the k-nearest training samples. During the training phase, the algorithm only stores the examples and their class labels. Hence, the k-nn is considered a supervised method. In the actual classification phase, a reference dataset is compared with an unknown data. The distance from the unknown data to the k-nearest neighbors determines its class assignment by either averaging the class numbers of the k-nearest refe rence points or by obtaining a majority vote from them.

Let x be an example of a d-dimensional feature space, i.e., an object to be classified and let, a set of n examples contained in the training set with known classes. The k-nearest neighbors of x among the n known examples should be selected first, according to a defined distance, usually the euclidean distance (see 3.17). Once the k-nearest neighbors of x have been selected, the class of x is determined by majority vote, i.e., the most represented class among k is selected.

$$class(x) = \underset{i \in \mathcal{S}}{\operatorname{argmax}}(n_i), \tag{3.22}$$

where $\mathcal{S}$ is the set representing the regions defined by the class labels and n_i is the score achieved in the region with class i. The choice of k is very important. In general, larger values of k reduce the effect of noise on the classification, but make boundaries between classes less distinct. A good k can be selected by parameter optimization using, for example, cross-validation. The particular case of $k = 1$ is called the nearest-neighbor algorithm, because a test example is simply assigned to the same class as the nearest example from the training set.

3.4.4.2 SOM-based classifier

As outlined in sub-section 3.4.3.4 the SOM neural network simultaneously performs a topology-preserving projection from the input data space onto a regular two-dimensional grid. Here, we list some of the reasons of SOM being used as a classifier:

- Weights representing the solution are found by iterative training.
- SOM has a simple structure for physical implementation and interpretation.
- SOM can easily map large and complex distributions.
- The generalization property of the SOM produces appropriate results for the input vectors that are not present in the training set

3.4.5 Conclusion

In this thesis work SOM is used as a clustering tool and as classifier for its powerful visualization and dimensionality reduction capability, for the topologically ordering property to adequately cluster and classify different type of signals. An extensive description of its theoretical aspects is given Chapter 4.

Chapter 4

SELF-ORGANIZING MAPS

Self-organizing maps (SOM) were introduced by Kohonen [33] [34], and have become a very popular tool used for visualization of high dimensional data spaces, clustering/vector quantization (VQ) and at the same time spatial ordering preserving of the input data space reflected by the ordering of the codebook vectors (cluster centroids). The basic idea of SOM is to map the data patterns onto an n-dimensional grid of neurons or units. That grid forms what is known as the output space, as opposed to the input space that is the original space where the data patterns are, as depicted in Figure 4.1. The neurons in the output layer are generally arranged in a one or two-dimensional lattice although most of the implementations of SOM use a rectangular grid of neurons. When even distances between the neurons in the output space are needed, hexagonal grids are sometimes used. Higher dimensional grids can be used, but are not common since it is not possible to easily visualize the output space. During competitive learning process the neurons become selectively tuned to various input patterns by competing among themselves. The synaptic weights of the neurons that win the competitions are modified according to an adaptation rule. If the SOM has been trained successfully, patterns that are close in the input space will be mapped to neurons that are close (or the same) in the output space. Thus, SOM is “topology preserving" in the sense that neighborhoods are preserved through the mapping process. This mapping correspond to a particular domain or intrinsic statistical feature of the input data, without any prior knowledge on the input distribution. That’s why the name *self-organizing map*.

Generally, SOM has the same disadvantage of *vector quantization*, since no matter how much we train the network, any given input pattern and the neuron it is mapped to will not be equal. Hence there is some difference between a pattern and its codebook vector representation. This difference is referred as to *quantization error* (Qe), and it is used as a measure of how well the SOM performs. Another meaningful parameter is the *topological error* (Te), which measure how good is SOM in clustering similar patterns into near regions of the map by calculating the distance between the winning neuron and second winning neuron (i.e., the second nearest neuron to an input data). In

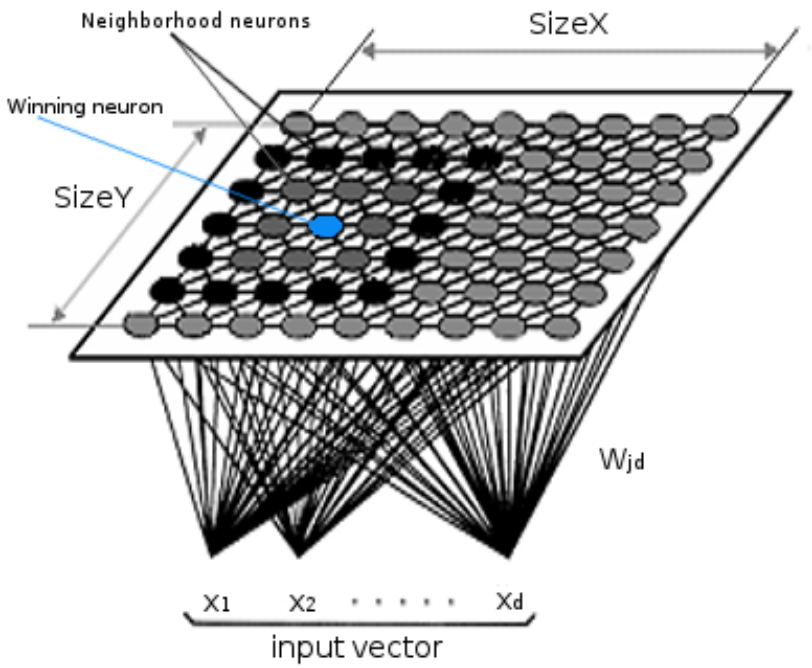


FIGURE 4.1: Two-dimentional SOM, Kohenen model

cluster analysis SOMs are particularly useful for visualization because they can be used to explore the groupings and the relations within high-dimensional data by projecting the data onto a two-dimensional image that clearly indicates regions of homogeneity. The formation of the SOM follows a specific procedure starting with the initialization of the synaptic weights in the network, normalized between 0 and 1, to overcome the fact that certain variables may overwhelm others in the learning process. Weights are chosen by randomly picking small values from a number generator. Accordingly the feature map is not organized a priori. Once the initialization is finished, the definition of the SOM follows three important processes; competition, cooperation and synaptic adaptation [12].

4.1 The SOM algorithm

The tree essential processes that define SOM are summarize as follow:

1. *Competition*. For each input pattern, the neurons in the network compute their respective values of a discriminant function.This

discriminant function provides the basis for competition among the neurons. The neuron with the largest value of discriminant function is declared winner of the competition.

2. *Cooperation*. The winning neuron determines the spatial location of a topological neighborhood of excited neurons, thereby providing the basis for cooperation among such neighboring neurons.

3. *Synaptic Adaptation*. This last mechanism enables the excited neurons to increase their individual values of the discriminant function in relation to the input pattern through suitable adjustments applied to their synaptic weights. The adjustments made are such that the response of the winning neuron to the subsequent application of a similar input pattern is enhanced.

The two first processes are in accordance with two of the four principles of self-organization described in section 3.2.1. While a modified form of the Hebbian learning is instead used in the adaptive process to account for the principle of self-amplification. As explained in section 3.2.1, the presence of redundancy in the input data, though not mentioned explicitly in describing the SOM algorithm, is essential for learning, since it provides knowledge about the underlying structure of the input activation patterns. Descriptions of the processes of competition, cooperation, and synaptic adaptation are detailed in what follows.

4.1.1 Competition

Let $\mathbf{x} = (x_1, x_2, ..., x_d)$ be an input pattern (vector) selected at random from the input space, where d represent the dimension of the input space. Let the synaptic-weight of neuron j in the output layer be written as

$$\mathbf{w}_j = [w_{j1}, w_{j2}, \cdots, w_{jd}]^T, \qquad j = 1, 2, \cdots, L, \tag{4.1}$$

where L is the total number of neurons in the output layer and let n be the discrete-time coordinate. The best match of the input pattern $\mathbf{x}$ with the synaptic-weight vectors $\mathbf{w}_j$, can be found by calculating the euclidean distance between them, the neuron j that minimize this distance is selected as the winning neuron or equivalently *best-matching unit (BMU)*. If we identify this neuron with the index $i(\mathbf{x})$, we may then determine $i(\mathbf{x})$ by applying the following condition, which sums up the essence of the competition process among the neurons.

$$i(\mathbf{x}) = \underset{1 \leq j \leq L}{\operatorname{argmin}} \|\mathbf{x}(n) - \mathbf{w}_j(n)\|. \tag{4.2}$$

4.1.2 Cooperation

In the cooperative process, a topological neighborhood is defined so that the winning neuron locates the center of a topological neighborhood of cooperating neurons. In particular, the winning neuron will excite the neurons in its immediate neighborhood more than those farther away from it. Let $h_{j,i}$ denote the topological neighborhood centered on winning neuron i and $d_{j,i}$ denote the lateral distance between the winning neuron i and the excited neuron j. The topological neighborhood $h_{j,i}$ can be a unimodal function of the lateral distance $d_{j,i}$ satisfying the following two distinct requirements[12]:

1. $h_{j,i}$ is symmetric about the maximum point defined by $d_{j,i} = 0$; in other words, it attains its maximum value at the winning neuron i for which the distance $d_{j,i}$ is zero.

2. The amplitude of $h_{j,i}$ decreases monotonically with increasing lateral distance $d_{j,i}$, and decays to zero as $d_{j,i} \to \infty$.

For example a translation invariant function for $h_{j,i}$ that satisfies these requirements is the Gaussian function

$$h_{j,i} = exp\left(-\frac{d_{j,i}^2}{2\sigma^2}\right), \tag{4.3}$$

where parameter σ is a parameter that measures the degree to which excited neurons in the neighborhood of the winning neuron participate in the learning process. Another unique feature of the SOM algorithm is that the size of the topological neighborhood is permitted to shrink with time. This requirement is satisfied by making the width σ of the topological neighborhood function $h_{j,i}$ decrease with time. A popular choice for the dependence of σ on discrete time n is the *exponential decay* described by

$$\sigma(n) = \sigma_0 \exp\left(-\frac{n}{\tau_1}\right), \qquad n = 1, 2, ..., \tag{4.4}$$

where σ_0 is the value of σ at the initiation of the SOM algorithm and τ_1. is a time constant to be chosen by the designer. Correspondingly, $h_{j,i}$ assumes a time-varying form of its own,

$$h_{j,i}(n) = \exp\left(-\frac{d_{j,i}^2}{2\sigma^2(n)}\right), \qquad n = 1, 2, ..., \tag{4.5}$$

where $\sigma(n)$ is defined by Eq.(4.4).

4.1.3 Adaptation

In this last process, for the network to be self-organizing, the synaptic-weight vector $\mathbf{w}_j$ of neuron j must change in relation to the input pattern $\mathbf{x}$. In Hebb's postulate of learning, stated in section 3.2.1, a synaptic weight is increased with a simultaneous occurrence of presynaptic and postsynaptic activities. However, such behavior of the Hebbian hypothesis in its basic form is not suitable for the unsupervised learning being considered here, since changes in connectivities occur in one direction only, finally driving all the synaptic-weights into saturation. To overcome this problem, the Hebbian hypothesis is modified by including a *forgetting term* $g(y_j)\mathbf{w}_j$, where $\mathbf{w}_j$ is the usual synaptic-weight vector of neuron j and $g(y_j)$ is some positive scalar function of the response y_j. The constant term in the Taylor series expansion of $g(y_j)$ must be zero, as a requirement. Given such a function, the change to the weight vector of neuron j in the lattice may be expressed as:

$$\Delta\mathbf{w}_j = \eta y_j \mathbf{x} - g(y_j)\mathbf{w}_j, \tag{4.6}$$

where η is the *learning-rate parameter* of the algorithm. The requirement on $g(y_j)$, may be satisfy by a linear function, such as;

$$g(y_j) = \eta y_j,$$

and for the winning neuron $i(\mathbf{x})$, Eq. (4.6) can be simplified by setting the response

$$y_j = h_{j,i(\mathbf{x})}.$$

Finally, given the input pattern $\mathbf{x}(n)$ and the synaptic-weight vector $\mathbf{w}_j(n)$ of neuron j at time n, we define the updated weight vector $\mathbf{w}_j(n+1)$ at time $n+1$ as:

$$\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \eta(n) h_{j,i}(n) \left(\mathbf{x}(n) - \mathbf{w}_j(n)\right), \tag{4.7}$$

which is applied to all the neurons in the lattice that lie inside the topological neighborhood of the winning neuron i [34][35][36]. Equation (4.6) has the effect of moving the synaptic-weight vector $\mathbf{w}_i$ of the winning neuron i toward the input vector $\mathbf{x}$. Upon repeated presentations of each input vector also the synaptic-weights vectors of the neurons in the winner topological neighbors adjust to resemble (in a lesser extent) the input vector. The converged weight vectors approximate the input probability distribution function, and can be viewed as prototypes representing the input data. The learning-rate parameter $\eta(n)$, according to the principle of stochastic approximation should start at some initial value η_0 and then decrease gradually with increasing time n. This requirement can be satisfied by the following expression:

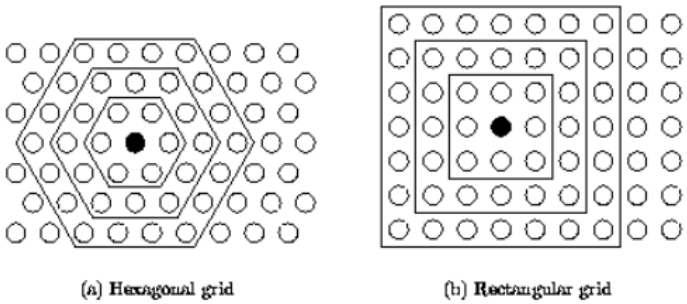
$$\eta(n) = \eta_0 \exp\left(-\frac{n}{\tau_2}\right), \quad n = 1, 2, ..., \tag{4.8}$$

where τ_2 is time constant of the SOM algorithm.
The adaptation of the synaptic weights in the network, can be decomposed in accordance with Eq. (4.6), into two phases: an ordering or self-organizing phase, followed by a convergence phase. These two phases of the adaptive process are described below:

1. The learning-rate parameter $\eta(n)$ used to update the synaptic-weight vector $\mathbf{w}_j(n)$ should be time-varying. In particular, during the first 1000 iterations or so, of the SOM algorithm, $\eta(n)$ should begin with a value close to 0.1; thereafter, $\eta(n)$ should decrease gradually, but staying above 0.01. The exact form of variation of $\eta(n)$ with n is not critical; linear, exponential, or inversely proportional to n function shapes can be used. It is however during this initial phase of the algorithm that the topological ordering of the weight vectors $\mathbf{w}_j(n)$ takes place. This phase of the learning process is called the *ordering phase*. The remaining iterations of the algorithm are needed principally for the fine tuning of the computational map; this second phase of the learning process is called the *convergence phase*. For good statistical accuracy, $\eta(n)$ should be maintained during the convergence phase at a small value (on the order of 0.01 or less) for a fairly long period of time, which is typically thousands of iterations. So, to be concise a good choice for the described parameters is:

$$\begin{aligned} \eta_0 &= 0.1, \\ \sigma_0 &= \text{the radius of the lattice}, \\ \tau_1 &= \frac{1000}{\log \sigma_0}, \\ \tau_2 &= 1000. \end{aligned}$$

2. For topological ordering of the weight vectors $\mathbf{w}_j(n)$, to take place, careful consideration has to be given to the neighborhood function $h_{j,i}$. Generally, the function $h_{j,i}$ can be of a variety of shapes of region around the winning neuron, the most used shapes are rectangular or hexagonal as shown in Fig. 4.2. In any case, $h_{j,i}$ usually begins such that it includes all neurons in the lattice and then gradually shrinks over time. To be specific, during the initial phase of 1000 iterations or so, when topological ordering on the synaptic-weight vectors takes place, the radius of $h_{j,i}$ is permitted to shrink linearly with time n to a small value of neighboring neurons. Eventually during the convergence phase of the algorithm, $h_{j,i}$ should contain only the nearest neighbors of winning neuron i, which may eventually be 1 or 0 neighboring neurons.



(a) Hexagonal grid (b) Rectangular grid

FIGURE 4.2: Different shapes of SOM lattice

4.2 Summary of the SOM algorithm

Three basic steps are involved in the application of the SOM algorithm after the *initialization* phase: *sampling*, *similarity matching*, and *synaptic weights update*. Next these steps are described:

ALGORITHM 1

Require: $\mathcal{X}$: the data set; d: the dimension of the input pattern; L: the SOM lattice dimension; the number of iterations; η_0, σ_0, τ_1, τ_2: parameters;

1. **Initialization**. For each neuron j in the lattice initialize the synaptic-weight vectors $\mathbf{w}_j(0)$ with small random values.

2. **repeat**

3. **Sampling**. Draw a pattern $\mathbf{x}$ from $\mathcal{X}$ at random;

4. **Similarity matching**. Find the best-matching (winning) neuron $i(\mathbf{x})$ at time-step n by using the minimum-distance criterion:

$$i(\mathbf{x}) = \underset{1 \leq j \leq L}{\operatorname{argmin}} \|\mathbf{x}(n) - \mathbf{w}_j(n)\|. \tag{4.9}$$

5. **Updating**. Adjust the synaptic-weight vectors of all excited neurons by using the update formula

$$\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \eta(n) h_{j,i(\mathbf{x})}(n) \left(\mathbf{x}(n) - \mathbf{w}_j(n)\right), \tag{4.10}$$

where $\eta(n)$ is the learning-rate parameter and $h_{j,i(\mathbf{x})}$ is the neighborhood function centered around the winning neuron $i(\mathbf{x})$; both $\eta(n)$ and $h_{j,i(\mathbf{x})}$ are varied dynamically during learning for best results.

6. **Until**. Continue with step 2 until no noticeable changes in the synaptic-weight vectors are observed or the maximum number of iterations is reached.

4.3 Properties of the SOM

Once the SOM algorithm has converged, the result is a topological representation of the input data space, in the sense that adjacent neurons in the lattice will tend to have similar synaptic-weight vectors and will correspond to a particular class or feature that displays the important statistical characteristics of the input data.

To begin with, let $\mathcal{X}$ denote a *spatially continuous input space*, the topology of which is defined by the relationship of its elements, i.e., the vectors $\mathbf{x} \in \mathcal{X}$. Let $\mathcal{A}$ denote a *spatially discrete output space*, the topology of which is endowed by arranging a set of neurons as the computation nodes of a lattice. Let Φ denote a nonlinear transformation called a **feature map**, which maps the input space $\mathcal{X}$ onto the output space $\mathcal{A}$:

$$\Phi : \mathcal{X} \longrightarrow \mathcal{A}, \tag{4.11}$$

Given an input vector $\mathbf{x}$, the SOM algorithm proceeds by first identifying a best-matching, or winning neuron, $i(\mathbf{x})$ in the output space $\mathcal{A}$, in accordance with the feature map. The synaptic-weight vector $\mathbf{w}_i$ of neuron $i(\mathbf{x})$ may then be viewed as a *pointer* for that neuron into the input space $\mathcal{X}$. These two operations are depicted in Fig. 4.3. The self-organizing feature mapping Φ has some important properties, as described here:

Property 1. Input Space Approximation
The feature map Φ, represented by the set of synaptic weight vectors $\{w_j | j = 1, 2, ..., L\}$ in the output space $\mathcal{A}$, provides a good approximation to the input space $\mathcal{X}$.

The basic aim of the SOM algorithm is to store a large set of input vectors $\mathbf{x} \in \mathcal{X}$ by finding a smaller set of prototypes $\mathbf{w}_j \in \mathcal{A}$, so as to provide a "good" approximation to the original input space $\mathcal{X}$. The theoretical basis of the idea is the vector quantization theory, and the motivation for which this property is interesting is data dimensionality reduction or data compression (see section 3.3).

Property 2. Topological Ordering
The feature map Φ computed by the SOM algorithm is topologically ordered in the sense that the spatial location of a neuron in the lattice corresponds to a particular domain or feature of input patterns.

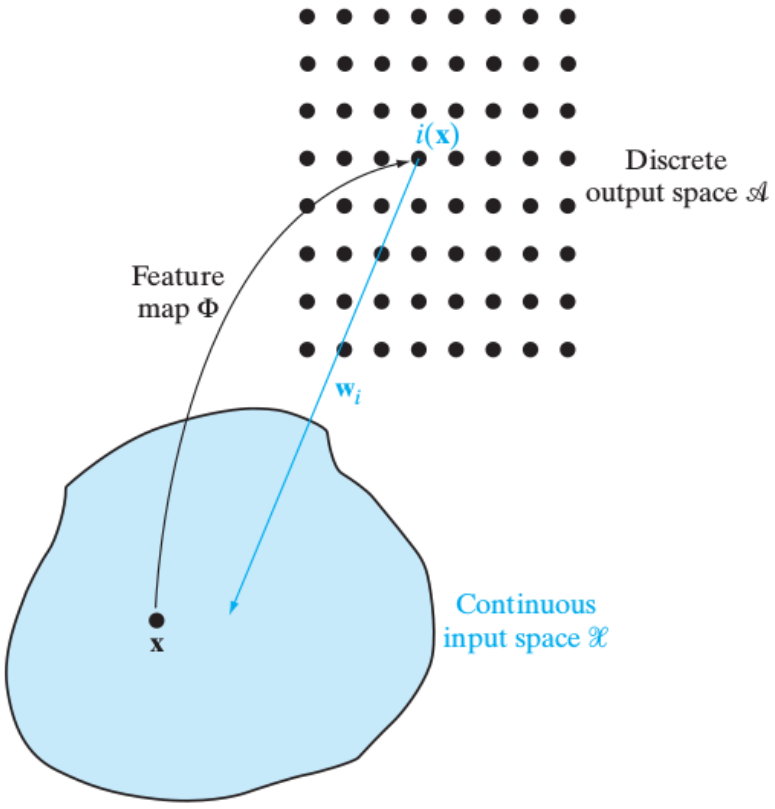


FIGURE 4.3: Relationship between feature map Φ and synaptic-weight vector $\mathbf{w}_{i(\mathbf{x})}$ of winning neuron i.

The topological ordering property is a direct consequence of the weight update equation that forces the weight vector $\mathbf{w}_{i(\mathbf{x})}$ of the winning neuron $i(\mathbf{x})$ to move toward the input vector $\mathbf{x}$. The crucial factor is that the weight updates also move the weight vectors $\mathbf{w}_j$ of the closest neighboring neurons j along with the winning neuron $i(\mathbf{x})$. Together, these weight changes cause the whole output space to become appropriately ordered. We can visualize the feature map Φ as an *elastic* or *virtual net* with a grid like topology. Each output node can be represented in the input space at coordinates given by their weights. Then if the neighboring nodes in output space have their corresponding points in input space connected together, the resulting image of the output grid reveals directly the topological ordering at each stage of the network training

Property 3. Density Matching
The feature map Φ reflects variations in the statistics of the input distribution: regions in the input space from which the sample training vectors $\mathbf{x}$ are drawn with high probability of occurrence are mapped onto larger domains of the output space, and therefore with better resolution than regions of input space from which training vectors are drawn with low probability.

Let $p_{\mathbf{x}}(\mathbf{x})$ denote the multidimensional pdf of the random input vector $\mathbf{X}$, a sample realization of which is denoted by $\mathbf{x}$ and let $m(\mathbf{x})$ denote the map *magnification factor*, defined as the number of neurons represented by a small volume $d\mathbf{x}$ in the input space $\mathcal{X}$. For the SOM algorithm to *match the input density* exactly, the following *proportionality relationship* is required [37]:

$$m(\mathbf{x}) \propto p_{\mathbf{x}}(\mathbf{x}) \tag{4.12}$$

This property implies that if a particular region of the input space contains frequently occurring stimuli, it will be represented by a larger area in the feature map than a region of the input space where the stimuli occur less frequently. In general literatures the analysis of Eq. (4.12) have been made in the case of one-dimensional grids. In such context we have that the magnification factor $m(\mathbf{x})$ is *not* proportional to $p_{\mathbf{x}}(\mathbf{x})$. In the SOM algorithm the proportionality relationship for the one-dimensional case becomes [38]:

$$m(\mathbf{x}) \propto p_{\mathbf{x}}^{2/3}(\mathbf{x}) \tag{4.13}$$

Then for the case of one-dimensional and likewise in higher- dimensional grid, SOM fails to achieve the proportionality relationship of Equation 4.12. Hence, SOM algorithm is an approximation which tends to over-represent regions of low input density and to under-represent regions of high input density.

Property 4. Feature Selection
Given data from an input space with a non-linear distribution, the self organizing map is able to select a set of best features for approximating the underlying distribution.

This property is a natural culmination of properties 1 through 3. *Principal Component Analysis* (PCA) is able to compute a dimensionality reduction of the input by taking advantage of the variance in the training data. It does this by computing the eigenvector associated with the largest eigenvalue of the correlation matrix. So PCA is fine when there's a linear input-output relation. Instead if the data forms a curved line or surface, linear PCA won't work, but a SOM will overcome this approximation problem by virtue of its topological ordering property. The SOM provides a discrete approximation finding the so-called principal curves or principal surfaces, and may therefore be viewed as a non-linear feature selector.

Chapter 5

CLUSTERING OF THE SOM

The main goal of this thesis is to provide an unsupervised clustering procedure to cluster several real-word signals related to the IoT scenario. To do it we have employed several schemes including a feature extraction tool, different feature selection algorithms, a SOM classifier, and an agglomerative hierarchical clustering algorithm (AHC) for the purpose of efficiently grouping different classes of signals. The clustering is carried out using a two-step approach. The first part consist in extracting qualitative visual information describing the data structure property from the SOM. Secondly, quantitative information are extracted by clustering the SOM using the AHC with a tunable parameter. Using this approach results in a reduced computational load of the algorithm, especially when applied to large dataset, moreover, different preprocessing strategies can be selected in a limited amount of time. This chapter describes the fundamental steps followed to build a simulation procedure to obtain the performance that allow us to analyze our algorithm. The procedure of our approach is summarize in Figure 5.1.

5.1 Data subdivision

In this experiment we used the dataset described in Section 2.2. After the feature extraction, all data have been normalized to values between 0 and 1. To realize our model we divided the dataset into a train set and test set. Training component is used to power the SOM and create a codebook, which is then used to evaluate the generalization of the model, by presenting the test set. The subdivision ratio is important mainly because if we use too many signals in the training phase to produce the codebook, there is a risk of over-fitting, this means that we will not have the capacity to generalize. And testing on non-general model leads to results that can not be considered valid. Whereas, if we use more data in the test phase, our model will have little information to learn from, and this leads to a poor classification result. On the basis of what is expressed we chose to split the dataset using 70% for the training set and the remaining 30% for the test set. It is to be noted that this procedure was done on all groups in the dataset. In particular

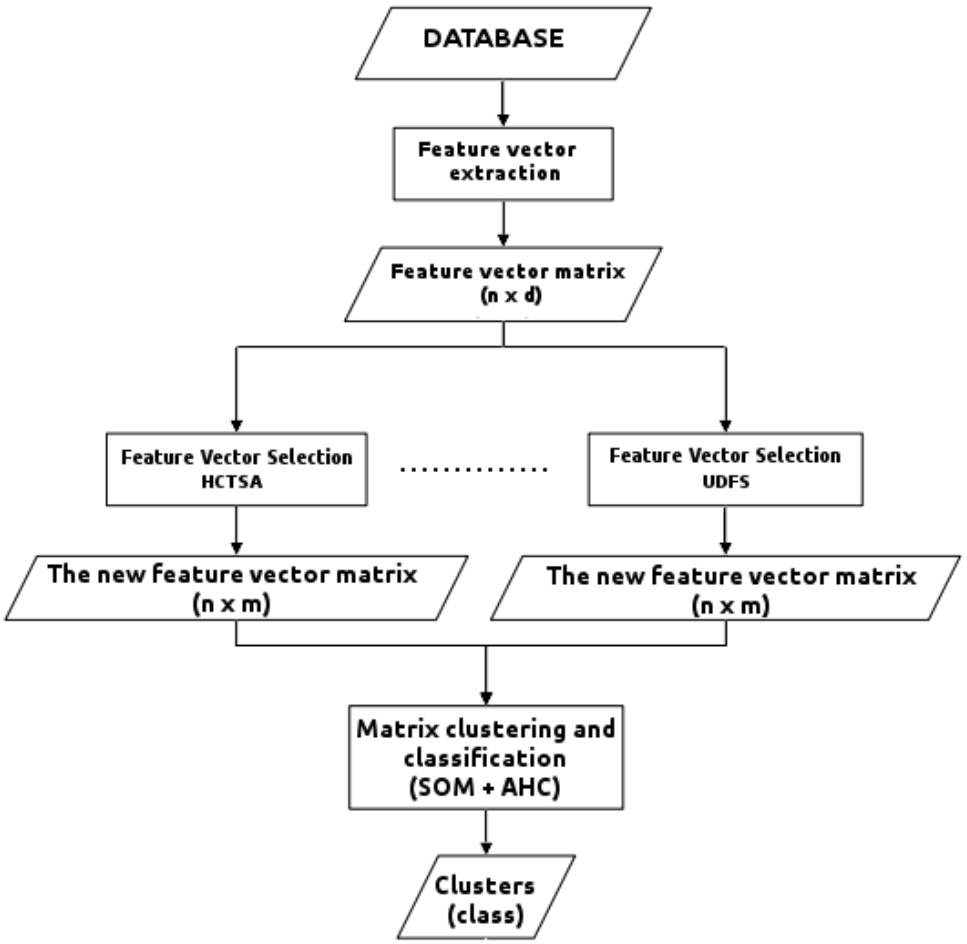


FIGURE 5.1: Model building flowchart.

each individual group was partitioned with a 70/30 ratio to obtain a balanced proportion of the signals in each of the 20 groups.

5.2 Performance measures

The following map quality measures are used.

- *Quantization error (Qe)* This error measure the average distance between each data vector and its BMU, it is a measure of the map resolution. Thus, the optimal map is expected to yield the smallest average quantization error.

$$Qe = \frac{1}{N}\sum_{i=1}^{N}\|\mathbf{x}_i - \mathbf{w}_{i(\mathbf{x})}\|, \tag{5.1}$$

  where N is the number of input pattern, $\mathbf{x}_i$ is the input pattern and $\mathbf{w}_{i(\mathbf{x})}$ is the winning neuron weight vector.

- *Topographic error (Te)* This error measure the proportion of data vectors for which first and second BMU are not adjacent units. Therefor the lower the topographic error is, the better the Self-Organizing Map preserves the topology.

$$Te = \frac{1}{N}\sum_{i=1}^{N}\delta(\mathbf{x}_i), \tag{5.2}$$

  where $\delta(\mathbf{x}_i)$ is 1 if the BMU and the second BMU of $\mathbf{x}_i$ are not adjacent. Otherwise it is 0.

In the following some performance measures used for the classification process are presented.

- *Confusion Matrix*

		Predicted value	
		Positive	Negative
Real value	Positive	*tp*	*fn*
	Negative	*fp*	*tn*

  Where tp, fp, tn and fn represent respectively, the numbers of true positive, false positive, true negative and false negative quantity, which are fully described in [39]. The above confusion matrix represent a binary class case. In our case it has been extended to all classes in the dataset. We have then extracted the following quantitative average performance measures over all the classes.

– *Precision* (P), *Recall* (R) and *F-score* (F1). They are defined as follows:

$$\begin{aligned} P &= \frac{tp}{fp+tp}, \\ R &= \frac{tp}{fn+tp}, \\ F1 &= \frac{2PR}{P+R}. \end{aligned}$$

5.3 SOM Classifier

The capacity of SOM to represent the data structure is very important. A proper analysis of the clusters and evaluation of the performance depends on that. To use SOM as a classifier we exploit the projection property for which the input data that usually belong to higher dimensional space is forced into many codevectors organized in a two dimensional structure.
There are two fundamental issues: the first is the non deterministic nature the clustering and the second is the relationship between the clusters. As an unsupervised learning method the clustering leads in general to different results for different simulations. The relationship between clusters instead can be seen in the planar surface by checking the distances between the codevectors. However, this relationship is in general difficult to deduce exactly, since the size of the codevectors is much bigger than the planar surface size. Despite this, we still have an insight about the classification regions. In order to have a better representation of these regions an *autolabel* mechanism is used. And it consists in the assignation of class memberships to each neuron in the map after completion of the training phase, that is, when the synaptic-weights of the neurons reach stability. The construction of the map goes through three different steps: the initialization of the map, the training of the map and finally the autolabel process.

5.3.1 Setup

The first step corresponds to the definition of the initial parameters of the SOM. This operation can be performed automatically by using some default initialization function derived from the knowledge of the training set properties. For example a rule of thumb to select the size of the SOM is to use the formula $5\sqrt{n}$, where n defines the number of pattern in the training set. Otherwise the definition can be done manually by the user. The important parameters defined in section 4.1.3, such as neighborhood radius, iteration (epochs) and neighborhood function, are chosen. Generally, the task of defining optimal values for specific classification parameters is achieved based on simulations. For now,

for the setup of the classifier, we consider a square map of dimension $L = 10 \times 10$, initial learning rate $\eta_0 = 0.9$, initial radius $\sigma_0 = 5$, a Gaussian shaped neighborhood function and an iteration number of 1000 epochs. As for the data pattern, we decided to choose at first few features from the data matrix without considering any feature selection algorithm. Note that for now we do this to test the functionality of the classifier. When performance is to be taken, some parameters will be chosen based on performed simulation of the algorithm.
The next step correspond to the training of the map and it is done by following the algorithm 1 of section 4.2. More precisely, for each epoch all data pattern in the training set are randomly chosen and feed as input to the SOM, which is trained accordingly. The process is repeated and at the end of the convergence phase a codebook is defined.
In the third and last part the autolabeling mechanism takes place after the clustering process finished. In practice, once the codebook is build, all patterns (with known class membership) are feed again to the map and each neuron (codevector) stores the *number* of data pattern for which it is the winning unit. These numbers are stored into class counter vectors owned by all neurons. After all training patterns have been presented to the SOM, by majority of the votes for a particular class on each node a class membership is assigned to that node.
As an example, suppose a neuron is fired prevalently by signals of class membership 10, after the completion of the counting, by a majority vote that neuron will be assigned to class 10. It should be noted that the autolabel procedure is the only supervised part of this approach since the creation of the codebook, and thus the learning process, occurs in a non-supervised context. It is possible by the same nature of SOM, that some neurons do not have a label assignment since they are never excited directly. These neurons as we shall see, usually represent the natural boundary created between the clusters of the map. Once labels are given to all the neurons, which happens after the training phase, the classifier is ready to be used in the testing phase for classification of unseen data pattern. Fig. 5.2 illustrates the classifier model.

5.3.2 Visualization

A visualization analysis of the input data projected into the codebook can be done already after the above setup experiment. In fact from Fig. 5.3 and 5.4 we can see how the synaptic-weights are changing from a casual initial state to a more ordered situation. This behavior confirm property 1 and 2 of SOMs (see section 4.3). From a more careful look of the map some grouping can be noticed, similar colors in the map distinguish similar pattern, so color distribution defines various clusters. This is an unsupervised clustering algorithm, and therefore we have no direct information on the exact number of clusters, nor their

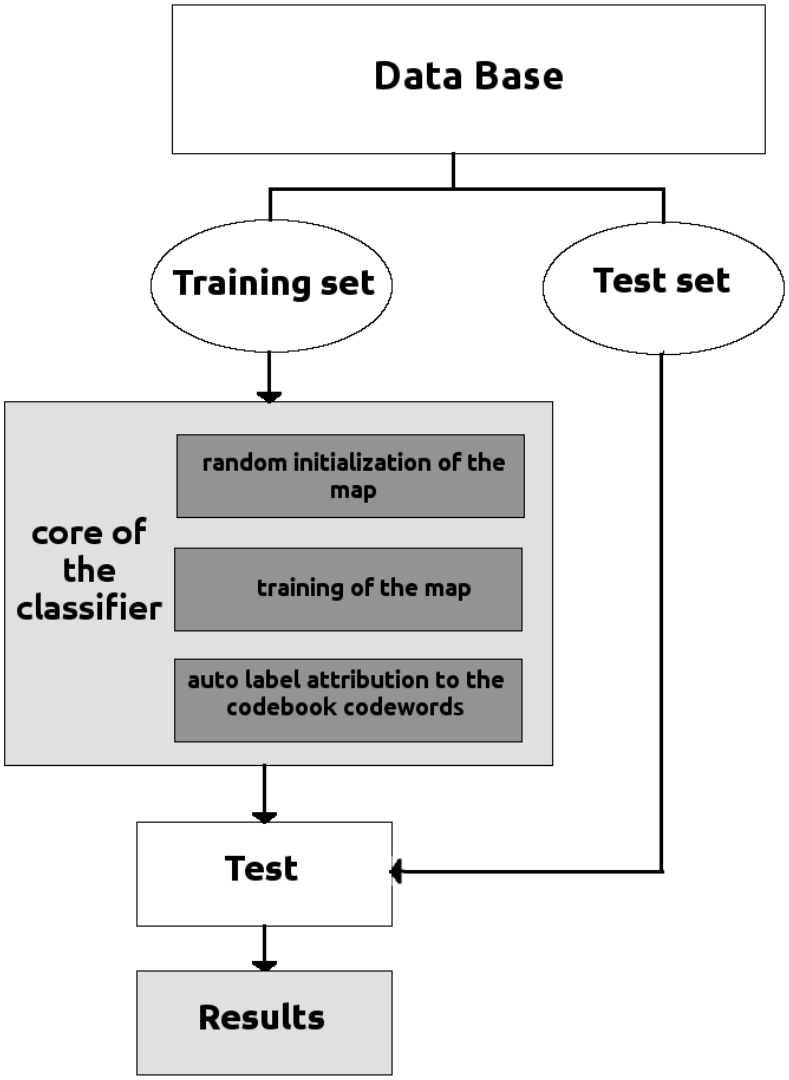
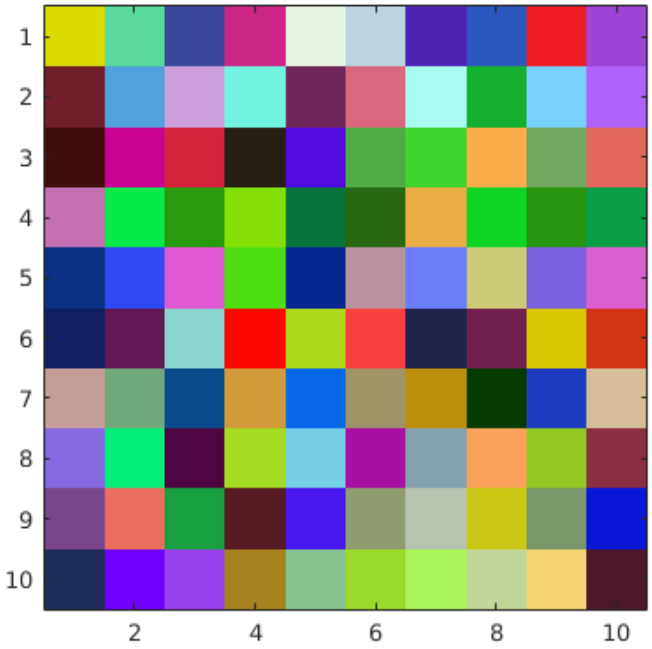
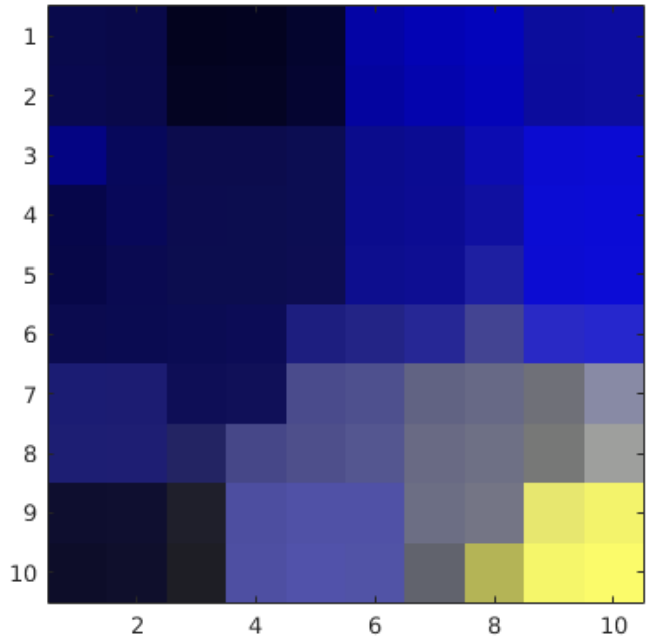


FIGURE 5.2: General scheme of the simulation model. The core of the classifier is represented by the clustering process by SOM followed by the autolabel procedure.

distribution in the map. Knowing a priori the structure of our dataset we can only make preliminary supposition from a first visual inspection. The ability to pre-analyze the data distribution in a visual way is one of the strengths of the self-organizing map. In this case the colors distribution under represent the clusters in the dataset. In fact it can be noted that the colors are mainly of three or four types, so it is difficult to see 20 clusters, representing 20 classes as we expected.
When the autolabel procedure is used to extract label information we can immediately perceive the arrangement of the clusters and the various classes, as shown in Fig. 5.5a. We also can see outliers, which are those neurons indicated with zero, because they do not belong to any class of signals. These neurons are important and vary with different epoch, especially for small maps and at lower epochs their numbers can be very substantial and this cause a degrading behavior in the capability of the model to generalize. As we will show this plays an important role in the performance of the classifier. A particular to be noted is that by looking at Fig. 5.5a for 200 epochs some classes are not even recognized, this gives the idea that in this stage, the map is very unstable and need more iteration. In fact when we reach 1000 epochs in Fig. 5.5b a distinct situation appears, even though not all groups are tightly cluster, the map is able to discriminate at least all the signals types of our dataset. Another remark is that the way color representation is used failed to fully grasp the entire grouping situation, this is mainly due to the fact that we are trying make a visualization of high dimensional space codevectors using a 3-d dimensional space, which is clearly inadequate. But for a pre-analysis step, in combination with the classifier labels information we were able to make some assumption that we needed to continue in this experiment. One assumption is that from various simulations we noticed that a special grouping scheme do not change, in fact as depicted in Fig. 5.5b the structure of the subregion including class 2, 3 and 4 remain the same. That is to say they are always cluster together, in different combinations but always together. An inspection of a bigger map describes the same situation for other groups. It is clear that there is a grouping structure in the dataset, in which some signals have near or the same statistical property. This confirms some knowledge we had about the dataset, in fact, some signals are derived from a transformation of raw data signals that are those actually acquired by the sensor (e.g. Strain and Alarm are derived from RawReading) while other signals have very similar statistics (e.g. ABPDias, Pulse, HR and AccelLateral, AccelSagittal, AccelVertical). Under these intuitions we were encourage to integrate these information in the classifier to assess the results of various simulations that are discussed in Chapter 6.

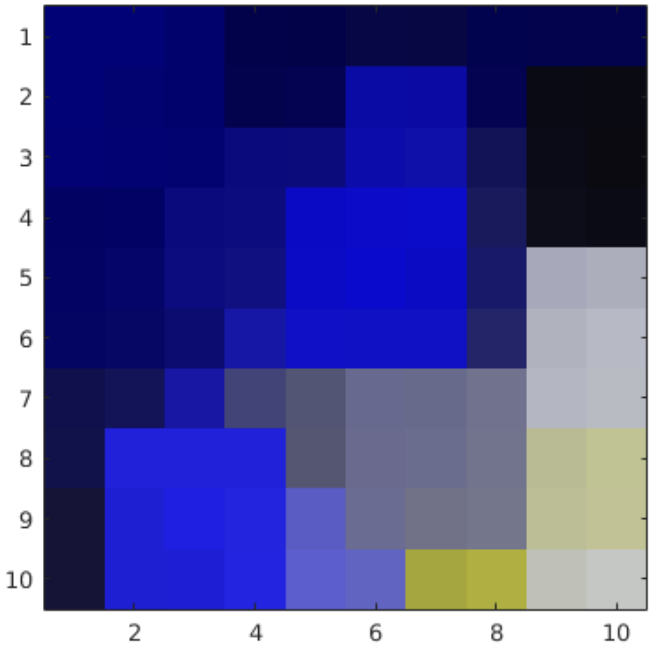


(A) initial state

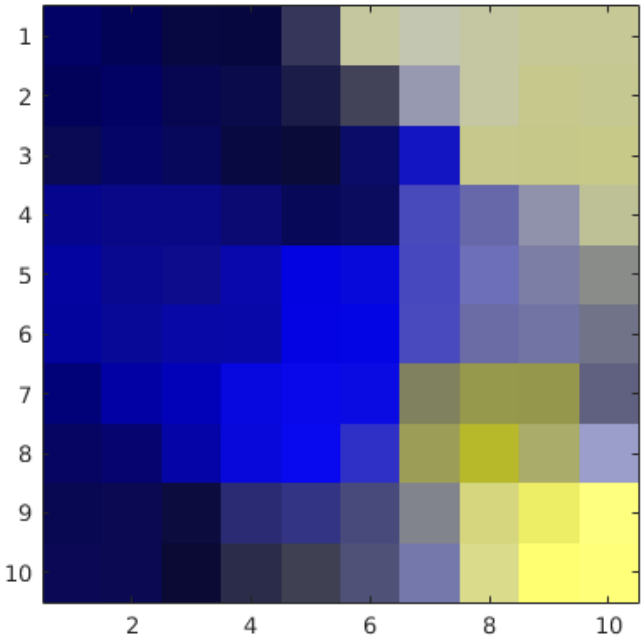


(B) 200 epochs

FIGURE 5.3: The process of the synaptic-weights update of SOM starting from an initial casual state to an ordered topology for various epochs in a 10x10 neurons map size.

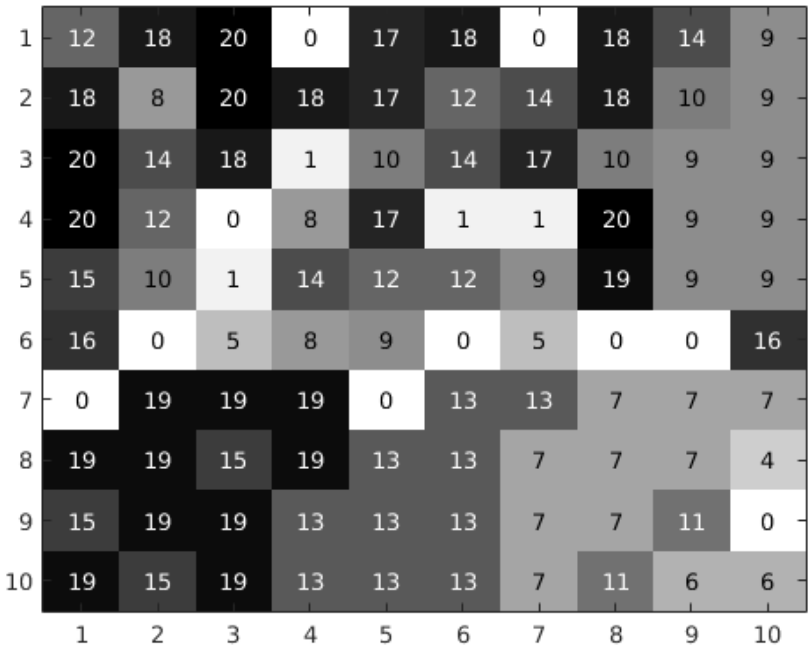


(A) 400 epochs

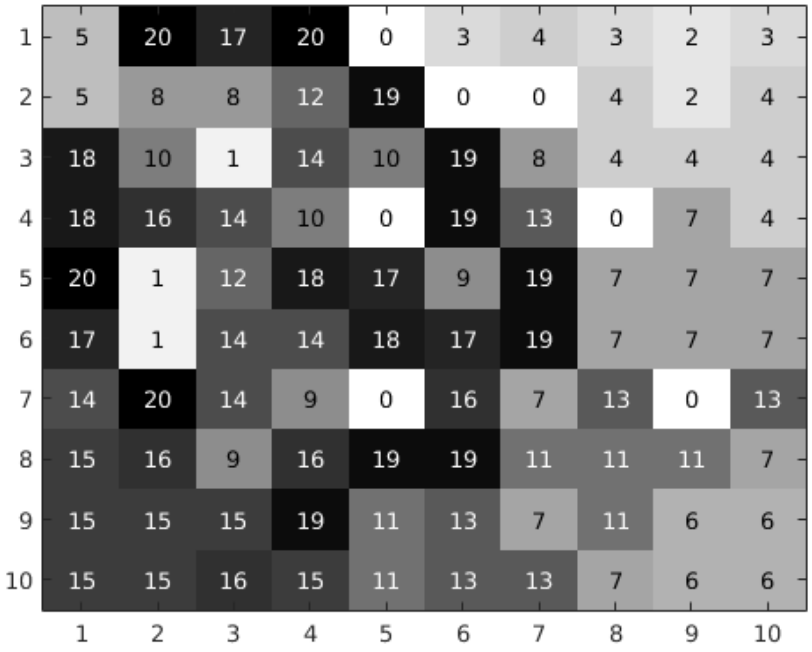


(B) 1000 epochs

FIGURE 5.4: The process of the synaptic-weights update of SOM starting from an initial casual state to an ordered topology for various epochs in a 10x10 neurons map size.



(A) 200 epochs



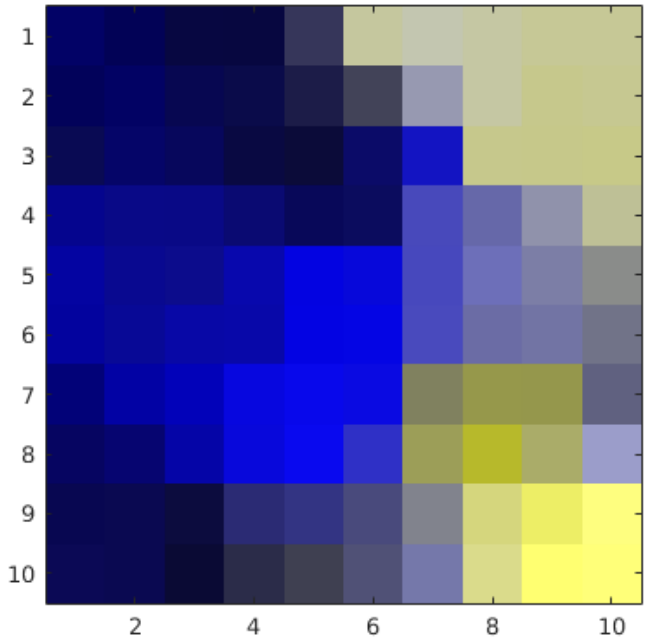
(B) 1000 epochs

FIGURE 5.5: After synaptic-weights final update, the auto labeling result of the SOM classifier for two different epochs

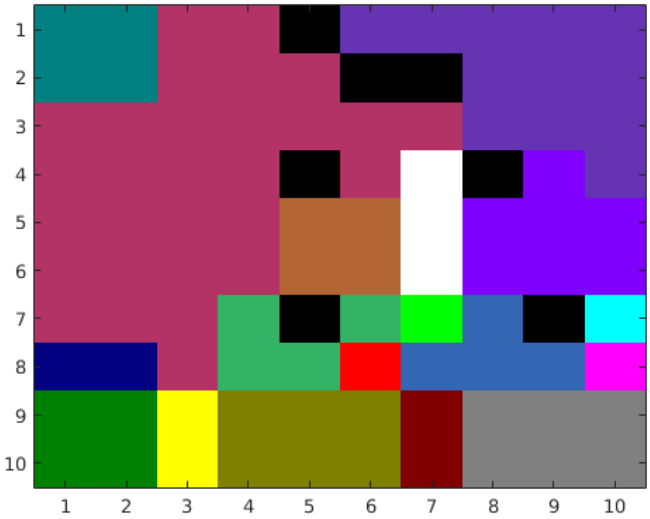
5.4 Two-step clustering

The classifier is used to evaluate the performance but also to optimize certain parameters. Remember that apart for the labels assignment, this is an unsupervised learning regime so the actual number of clusters is unknown. Whereas, the SOM visualization power for such a high dimensional data space is approximative. This brings us to the last setting stage of our model, which consist in using the agglomerative clustering algorithm (AHC) discussed in section 3.4 to transform the map from a partitive to an hierarchical structure. After trying all the merging rules described there, we ultimately choose the *average linkage* method, which is the one with he best grouping result. In general, the hierarchical algorithm has a cost that depends on the number of the data input and scales at worst-case as $O(N^2 \log N)$ [40]. In this case, AHC is convenient, since the vector quantization of the SOM is exploited to cluster on codebook. Which is much more faster than to cluster directly the signals in the dataset.

Thank to the hierarchical structure, a parameter ϵ tied to some property of the data structure is set. Such ϵ therefore depends on the training set statistic and is used to decide the inspection level of the hierarchy in order to extract information not present in the SOM partitive clusters. The parameter ϵ is tunable and it is used to indicate the level at which the dendrogram should be cut. In particular ϵ is tied to the correlation between the clusters produced by the hierarchical algorithm and indicates the cutting level that gives the best performance on the classifier. At that level the number of clusters can be extracted and a comparison of the SOM and the hierarchical output can be exploited to check for similarities. Fig. 5.6b for example shows the result of the hierarchical clustering of the SOM shown in fig 5.6a by stetting ϵ in correspondence of 20 classes. A visual inspection with the labeled map of Fig. 5.5b shows that the two structures are alike only if certain groups are considered. In particular, the group formed by classes (2,3,4), class (6), class (15) and class (7) can be clearly distinguished, while the other classes are still randomly distributed in a unique block. This behavior is caused by the fact that the map is trained on overlapping signals class types with the first few features from the raw data matrix in Fig. 2.4. Due to this overlapping nature the performance remains very poor at this level. It is to noted that while the map visualization provide some insight into the structure of the dataset, it does not necessarily explain why a particular dataset is easily separable or not. In fact, when the classes are projected into clusters of distinguishable areas, a separation with a high accuracy is possible for a classifier, but if the classes are highly overlapping after the projection, it does not mean the dataset is not separable. Rather, it indicates that classification for such dataset is difficult. This is not a proof but an indication.



(A) 1000 epochs



(B) 1000 epochs

FIGURE 5.6: Comparison of SOM (a) and hierarachical clustering (b) result for 20 classes dataset for a 10x10 map size

Name	Label	instances(#)
ABPDias	1	395
Accels	2	300
RawReading	3	288
Temp	4	198
Breathing	5	100
ECG	6	100
Humidity	7	99
PPG	8	100
RR	9	100
Solar	10	100
WindDir	11	100
WindSpeed	12	99

TABLE 5.1: Data signals types in the new dataset with the labels and the number of signals in each class.

Through various simulations and comparison of the SOM output with the hierarchical clustering result, we came to the conclusion that the number of clusters present in the dataset we had to deal with could be actually less than 20. It also confirms the observations outlined at the end of section 5.3.2 and it is a mean to also confirms the overlapping behavior of the different clusters, at least when 20 classes are considered. Some analysis of the results from our simulations showed that the number of clusters that achieve good performance and an acceptable signals types differentiability was around 12. Considering all previous observations and simulations results, we decided to build a new dataset. For this new dataset signals derived from transformation made on the raw signals retrieved by the sensors were grouped together, as well as signals with very similar statistics. In such a way the new dataset results in a reduced numbers of classes, hence, in a reduced number of clusters. In Tab. 5.1 a summary of the new dataset is presented.

In the next chapter results of the simulations to assess the performance of the model are given. In particular the comparison amongst the different feature selection algorithms for the case of 20 and 12 classes are shown. Evaluations of the quality of the different maps in terms of quantization error and topographic error are derived as well as the visual results (labels maps) of the classifier when it is derived from the best performer amongst the feature selection algorithms.

Chapter 6

EXPERIMENTAL RESULTS

In this chapter various test results performed using the model discussed in the previous chapter are shown. In particular the feature selection algorithms defined in section 2.4 are tested and compared to assess how the reduction of features affects the clustering and classification performance of the model and which one amongst them achieves the best results. As highlighted in the course of this thesis, dimensionality reduction has a strong impact on the computation time as well as on the storage requirements of the learning process. Hence, the selection of the reduced set of features is important, and should not change the particular structure of the data on which it is applied.

To obtain the performance each results is derived by iterating the learning process for a number of epochs. At different epochs intervals, the SOM is stopped, and a codebook is created. A classifier is then automatically built upon this codebook using the training set, as described in section 5.3. Once the classifier is ready, the testing set is used to retrieve classification performance results by using the class labels information to calculate a *confusion matrix* from which other quantitative average performance is derived (see section 5.2). Meanwhile, after the completion of the training process some information regarding the map behavior are retrieved as well. More precisely, quality measures regarding the quantization error (Qe) and the topographic error (Te), defined in section 5.2, are saved and used to comment the classification process. The above procedure is done on all the feature selection algorithms. In particular, in an iterative manner, a subset of features is selected according to each feature selection algorithm, and fed as training samples to the map. The number of features to be used at each step is preset by the user. In this case we have decided to increase the number of features in an interval from 40 to 200. More features could be used for the training but that goes with an expense in the computation time and the memory needed to store the features. In fact, computation time impacts considerably in the experiments, apart from the time each training and testing process take, for each feature selection algorithm and each feature number the program is executed 100 times, in order to assess the average performance.

In the following sections the testing is done in two part. The first part of the testing considers the input dataset as it is initially given, thus by taking into account 20 different classes. Then, the second part consists on testing the modified dataset with 12 classes (see Tab. 5.1), derived from the consideration we have made in section 5.3.2 and section 5.4. Finally the simulations results of each part are given. We will go by inspecting the results for different map sizes and for different training epochs. The initials default parameters used to train the SOM are: the initial learning rate $\eta_0 = 0.9$, the initial radius $\sigma_0 = \sqrt{L}/2$, number of epochs 1000 and the neighborhood function with a Gaussian shape. Prior to each training the map weights are initiated with small random numbers and the initial state of a map is equal for all the feature selection algorithms.

6.1 Clustering case: 20 classes

In this section results of the model is given for maps trained on the dataset of 20 classes. For the moment we will concentrate on the map with dimension $L = 5 \times 5$, which corresponds to 25 neurons. Fig. 6.1 displays the *average precision* curves for different feature selection algorithms with respect to the number of the selected features. It can be seen that the best performing algorithms in terms of average precision are given by GFS and MI. They have very similar results and perform greatly better with respect to the others. They initially present a growing behavior for increasing number of features until they reach a certain peak value, then in a second time especially for higher number of features they instead show a decreasing tendency. More particularly at low features number (i.e., 40 features) the UDFS algorithm present the worst performance followed by noSEL. It is to be noted that in noSEL no feature selection algorithm is applied, the features are consecutively chosen and fed to the map. It is interesting to see that while the general trend of the curves is decreasing with the increasing number of features, the Relief-F and the UDFS continuously increase or maintain a steady-state behavior. In this scheme noSEL continuously decreases with the number of features. This behavior however was expected, since without an adequate selection of relevant features it is difficult to efficiently represent the structure of the dataset. An important consideration which can be derived is that the overall classification performance are very poor. In fact, the best algorithm merely reaches a peak 46% average precision at 60 features. One explanation for this is that the map size is to small to represent the dataset structure efficiently, thus different classes end up been fused together. Another issue is represent by the fact that in this dataset many signals belong to overlapped clusters, which cause the classifier to create misplaced classes

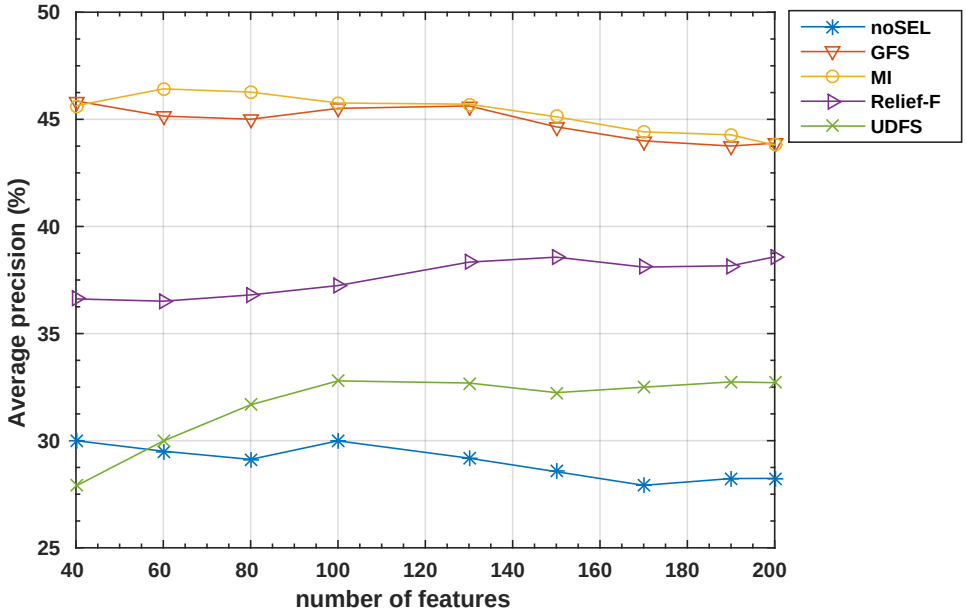


FIGURE 6.1: Average precision *vs* number of features for different feature selection algorithms. Comparison for 20 classes with $L = 5 \times 5$.

representation. More precisely, when small map sizes are considered what happens is that during the training phase, the synaptic-weight vector associated with some neurons (and to the neurons in their vicinity) will take longer to stabilize because they undergo more frequent and (especially) dissimilar updates. This leads to a not well defined distribution of the data space in the map, which imply a loss in terms of generalization ability and causes the misclassification of the signals in the testing set, which clearly results in a considerable deterioration of the average performance.

In Fig. 6.2 and in Fig. 6.3 the quality measures of the trained SOM are shown for different feature selection algorithms. Fig. 6.2 displays the evolution of the average quantization error as a function of the number of features considered. This quantity represent the quality of the vector quantization technique provided by the SOM. It can be seen that the average quantization error decreases with the increasing number of features. This is not surprising since the model provided by the SOM to represent the input data space distribution should become more and more accurate as the number of relevant features used for training (and thus for shaping the model) increases. In fact, a major number of discriminant features forces the neurons to be more specialized. This creates a better representation of the input data space, thus allowing a closer distance between each pattern and its BMU.

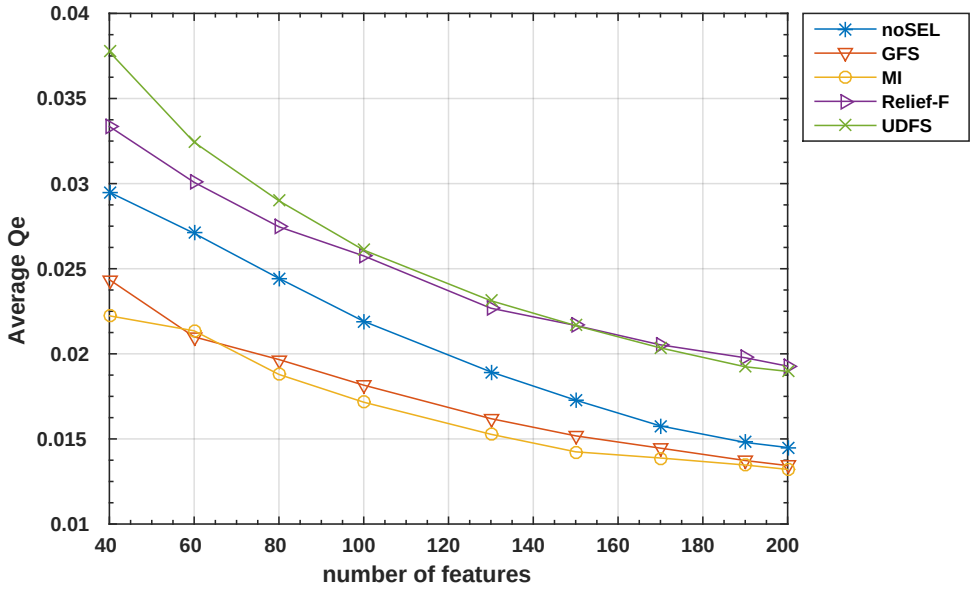


FIGURE 6.2: Average quantization error *vs* number of features for different features selection algorithms. Comparison for 20 classes with $L = 5 \times 5$.

In Fig. 6.3 the evolution of the average topographic error is represented against the selected number of features. It is interesting to see how this quality measure increases with the number of features. The meaning of this observation is that, in general, increasing the number of features will result in a more disordered map state. An explanation for this is given by the fact that for small map sizes many different signals are forced to be distributed in to small regions. Hence, many second BMUs end up being in the vicinity of their respective BMUs. This is particularly evident for the lowest features number. Since the lower the topographic error is, the better the SOM preserves the topology and the better the clusters are formed. This results indicate, as in the case of Qe in Fig. 6.2, that the relevant features selected by the algorithms MI and GFS are proving to be the best in preserving also the topology of the maps, meaning better distribution of data space. These two quality measures are signs of good performing maps and thus of the production of good classifiers which performance ultimately is reflected on the classification process.

Unfortunately for this map size, the quality measures are not very reliable, since due to the fact that when the number of neurons is small, a given neuron will be the best matching unit for a greater number of different signals in the training set, which leads to an under represented data distribution. It is also unfortunate that the only non-supervised features selector, UDFS, that would permit a certain degree

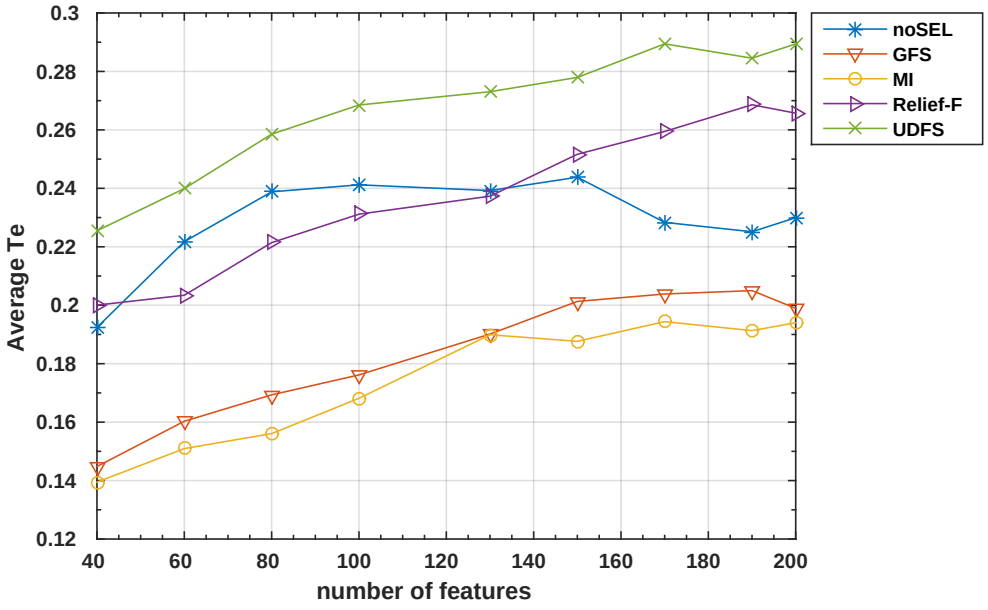


FIGURE 6.3: Average topographic error *vs* number of features for different features selection algorithms. Comparison for 20 classes with $L = 5 \times 5$.

of automation, still has the worst quality curve together with Relief-F, as they keep increasing rapidly with the number of features considered and especially for high features numbers, above 150 features where the others tend to get better or at least remain constants.

An interesting comparison between the wrappers and the filters methods used for this dataset can be outlined. Overall the algorithms with supervised feature selection greatly outperform the non-supervised ones. Surprisingly, the filter MI and the wrapper GFS have comparable performance in all the experiments. While noSEL has the worst results in the classification process and it represents a particular situation, the UDFS is the worst amongst all feature selection methods. The filter Relief-F performs in the middle although it is designed to work on multi-class scenario. We suspect that, the inferior result of Relief-F especially with respect to the other filter MI is a consequence of the overlapped nature of the classes in the dataset, for which the algorithm has difficulty to deal with. The particular situation of noSEL is represented by the fact that although presenting a good quality on the trained maps with respect to Relief-F and UDFS, it is not able to produce better results in the classification process.

Until now a map of size $L = 5 \times 5$ have been considered and we have discovered that its size is not sufficient to accurately represent the data in the dataset, producing not very reliable results. In fact,

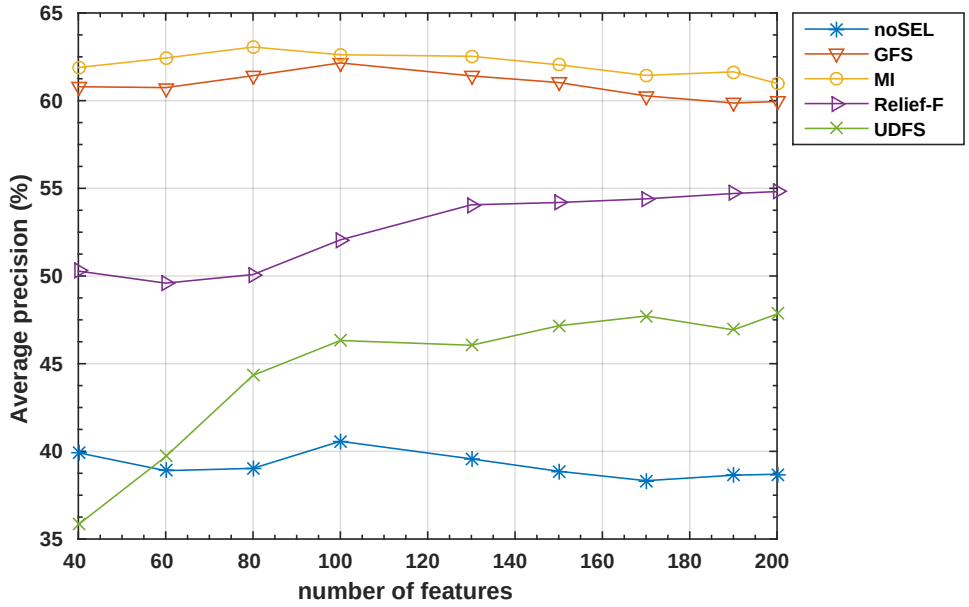


FIGURE 6.4: Average precision vs number of features for different features selection algorithms. Comparison for 20 classes with $L = 10 \times 10$.

the average precision is very low and the average quality measure Qe of this map achieves results which also indicate its bad vectors quantization performance. For such reason the size of the map has been increased to $L = 10 \times 10$ which corresponds to 100 neurons. In Fig. 6.4 the average precision for this case is shown, what can be immediately said is that the overall evolution of the curves does not change much. They are very similar to the case in Fig. 6.1 with some noticeable improvements for all the algorithms. Their average precision have increased at least by the 15%. It is a confirmation of the theory for which a larger number of neurons provide the SOM the ability to build a better representation of the data feature space. In this situation the classifier is able to perform much better. Others slights changes can be commented, for example the filter method has increase its distance with respect to the wrapper GFS, and distances between the three last algorithms are increasing as well. It is to be noted that among all the results, noSEL has increased the less, reaching an improvement of only 10%. Relief-F and UDFS are the only algorithms with increasing behavior for higher number of features, this means they still have to reach their maximum value which correspond to features number beyond 200. Unfortunately this also corresponds to higher memory consumption and greater computation time.

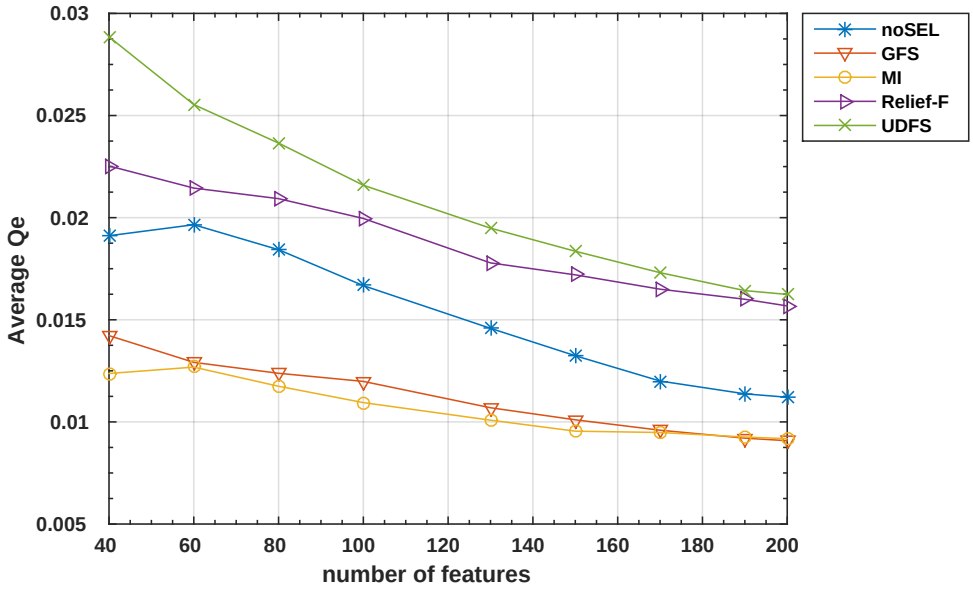


FIGURE 6.5: Average quantization error *vs* number of features for different features selection algorithms. Comparison for 20 classes with $L = 10 \times 10$.

As in the previous map size the quality measures were used for the analysis of the performance of the classification process. For this map size the same approach is used. In Fig. 6.5 and in Fig. 6.6 the average Qe and the average Te are represented respectively. They display the improvements achieved by a major number of neurons in the map. As Fig. 6.5 shows, when the number of neurons in a map increases, the average quantization error decreases accordingly. The curves have become flatter, but their overall evolution did not change much, except for Relief-F which become slightly better. What has changed instead is that the average Te has become worst especially in the lowest features number range. In fact by looking at Fig. 6.6 it can be seen that the evolution of the curves are flatter, with the average Te for MI passing from 0.14 to 0.18. Hopefully, this does not translate into worst performance. The result comes from the fact that for this map the neighbors of the winning neuron are more spread around it and the topological preservation is more stable. Moreover, for larger SOM the neurons are specialized in training signals (by updating the synaptic-weights) from the same class membership. Hence, the creation of the clusters becomes more efficient and each cluster is composed mainly by similar signals.

In fig. 6.7 the evolution of the average precision with respect to the number of epochs is shown. This results are obtained using 75 features on the training set from each feature selection algorithm. In this case

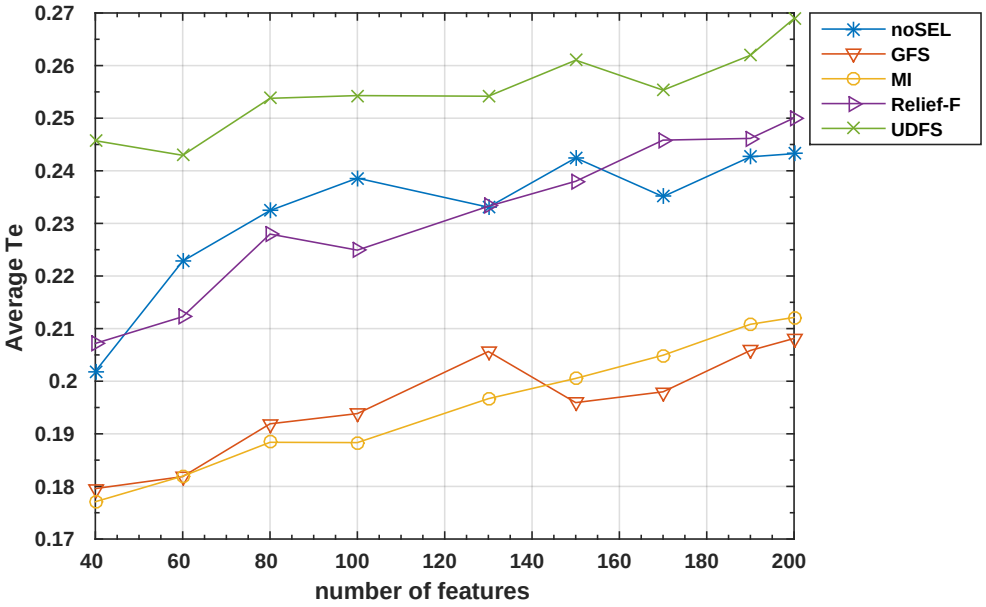


FIGURE 6.6: Average topographic error *vs* number of features for different features selection algorithms. Comparison for 20 classes with $L = 10 \times 10$.

we have extended the number of epochs form 1000 to 2000 to inspect the learning process behavior at higher training length. It can be noted that there exist two different phases of the training which are clearly distinguishable and correspond to the phases discussed in Chapter 4 at section 4.1.3. The first phase is the ordering phase which goes from 100 to 1000 epochs, here the map undergoes heavy changes in the synaptic weights due to the large learning rate, causing the curves to have the increasing behavior. The second phase is the convergence phase, in this case what happens is that the synaptic weights of the winning neurons are dominated by fine adjustments, since the learning rate assumes very small values. Therefore, the curves have a steady-state behavior which can be seen in the interval from 1000 to 2000 epochs. As it is depicted, in the first phase the different feature selection algorithms show different increasing behavior. GFS and MI increase more rapidly than the others confirming to have better stabilization and representations of the data distribution. This is also confirmed by the quality measures. By looking Fig. 6.8 it can be seen that the quantization error for these two algorithms have fastest decreasing behaviors toward the smallest error values and in Fig. 6.9 for the topographic error in which case they take the lowest errors.

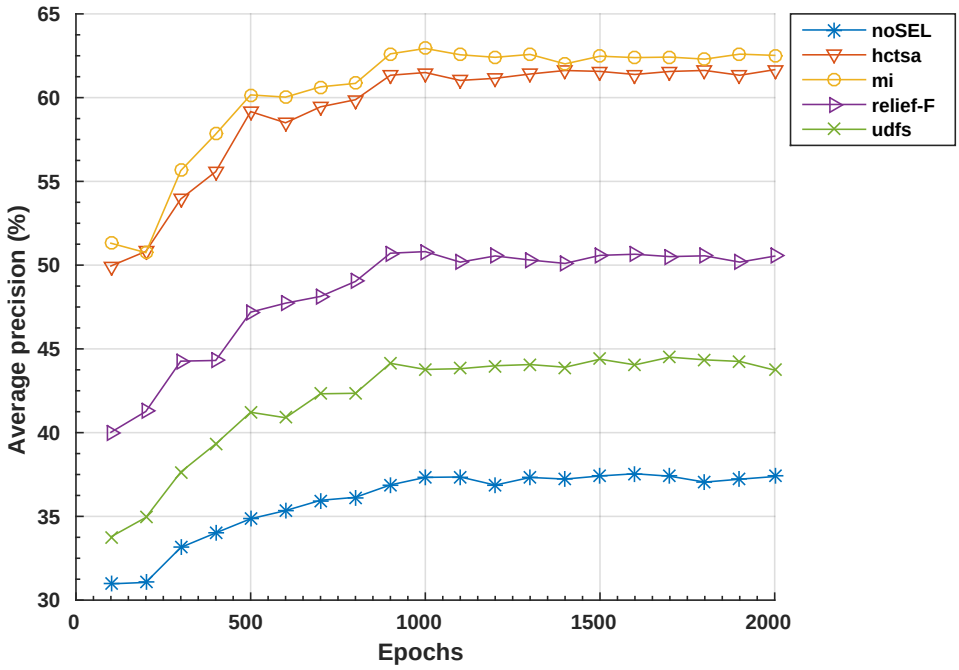


FIGURE 6.7: Average precision *vs* number of epochs for different features selection algorithms. Comparison for 20 classes with $L = 10 \times 10$ and 75 features.

An interesting aspect to recall is that of the peculiar situation for which noSEL achieves better maps quality but produces bad average precision is still valid also for this map size. This may suggests that, for this dataset, in the learning process the features selected from noSEL are able to produce good maps with respect to UDFS and Relief-F but fails to generalize the model afterward. That is why it performs so poorly in the test set evaluation. Any way if only the quality measures are taken into consideration then UDFS is the algorithm with the poorest performance in producing maps with good qualities. This may be explained by the fact that feature selection in non-supervised scenario is very complicated and the performance strongly depend on the relations between features. In this context it is very important to deal with the presence of corrupted or incomplete data which play the fundamental role in degrading the chance of a correct learning process. Clearly the features selected by UDFS are not suitable to express the data structure correctly even though the algorithm achieves better average precision than noSEL.

In Fig. 6.10 we have plotted the information of the labels produce by the classifier trained on the MI algorithm using 75 features from which we have obtained the average precision results of Fig. 6.7. What can be immediately noted is that the clusters representation is not well

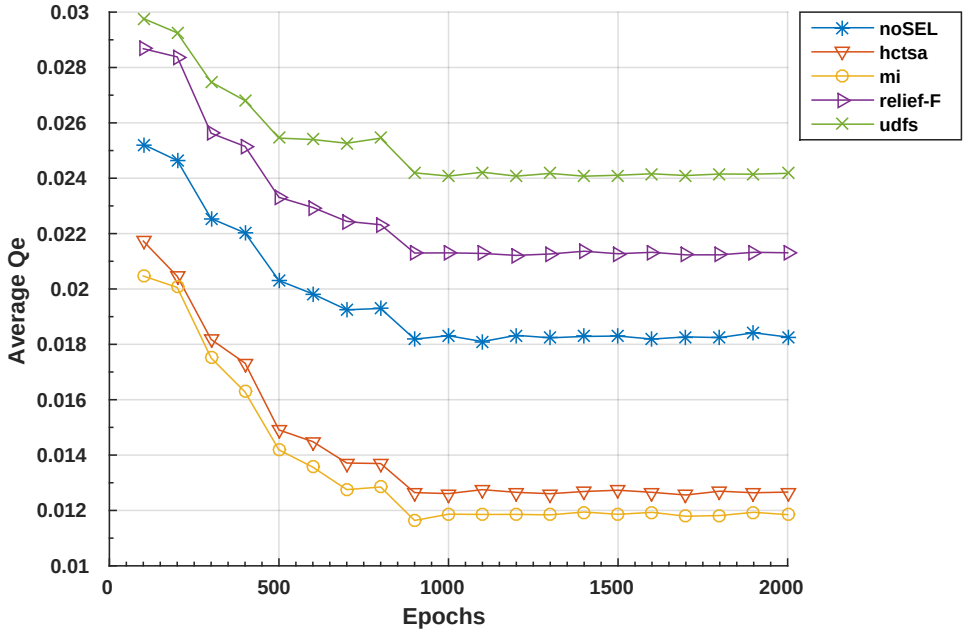


FIGURE 6.8: Average quantization error *vs* number of epochs for different features selection algorithms. Comparison for 20 classes with $L = 10 \times 10$ and 75 features.

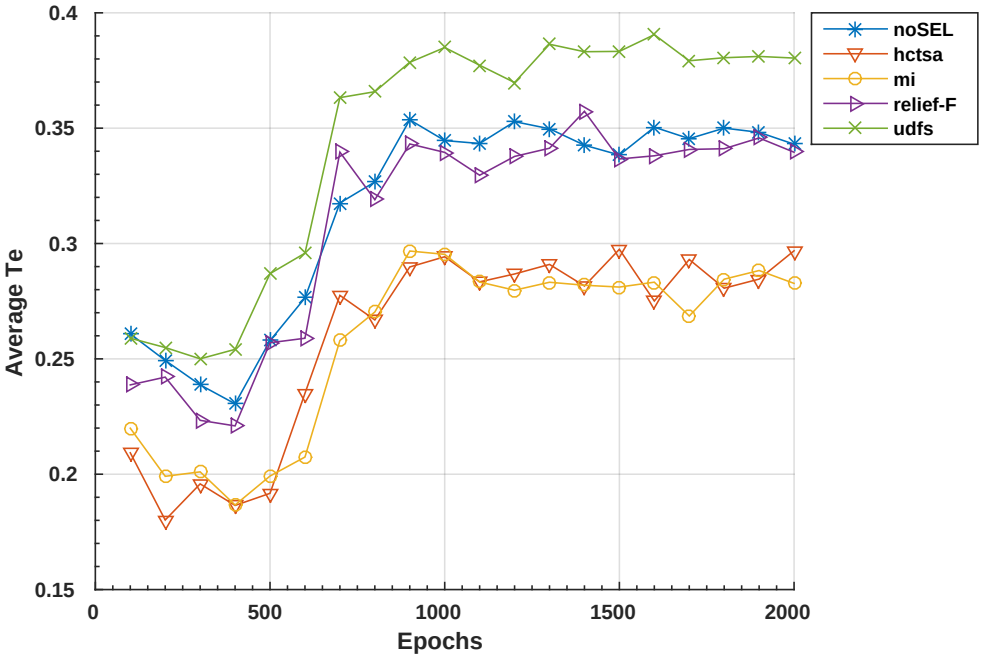


FIGURE 6.9: Average topographic error *vs* number of epochs for different features selection algorithms. Comparison for 20 classes with $L = 10 \times 10$ and 75 features.

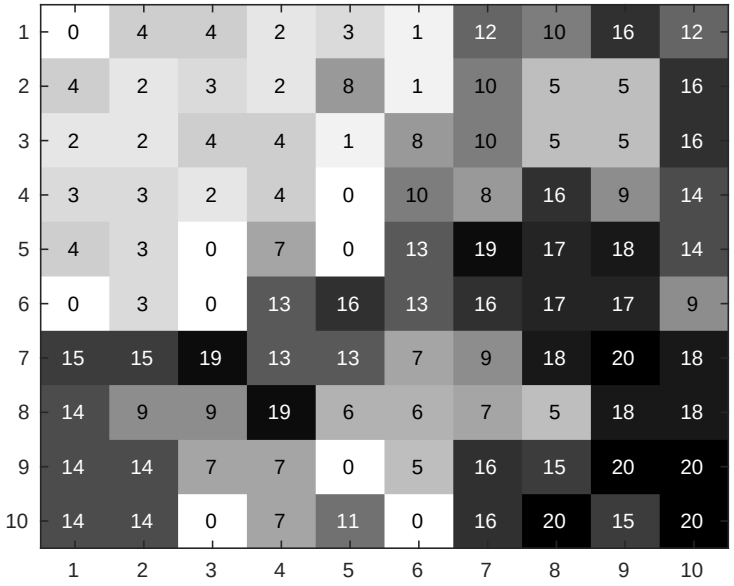


FIGURE 6.10: The classifier label information when considering 20 classes with $L = 10 \times 10$ and 75 features for the MI selection method

defined over the map. Clearly the overlapping nature of certain clusters, like (2), (3) and (4) is distinguishable and represent the strong correlations amongst many signals of different classes in the dataset. It can also be seen that some clusters are subdivided between various regions of the map, for example class (7) and class (19) while others, like class (11) are simply under-represented.

By using this visual information we have provided the considerations discussed in the previous chapter and had sufficient means to assess the structure of the dataset and to elaborate the experiments onto the reduced number of clusters for which results are presented in the next section.

Going by the fact that we have discovered, from previous results, that small maps size are not capable to fully define the structure of the dataset. In the following section we have decided to omit the testing on the $L = 5 \times 5$ map and to concentrate our effort on maps with higher dimensions.

6.2 Clustering case: 12 classes

This section presents the simulation results obtained using the new constructed dataset defined in Tab. 5.1. In this case signals that have similar or the same statistic were combined together, resulting in a reduced number of classes, hence a reduced number of clusters. The simulations were made as in the previous case starting from a map initialized to small random real values. The initialization of each map is equal for all feature selection algorithms. Parameters of the SOM are, the initial learning rate $\eta_0 = 0.9$, the initial radius $\sigma_0 = \sqrt{L}/2$ and the number of epochs 1000.

Fig. 6.11 shows the simulation results on a map of size $L = 100$ neurons, with a $L = 10 \times 10$ grid representation. The classification performance of the model for the new dataset are higher compared to the previous ones, which signify that the classifier performs better on clusters which are well defined and separable. In fact the average precision reaches 83% with the MI algorithm, much higher compare to the 20 classes case for the same map. GFS and MI algorithms continue to confirm to be the best algorithms with MI performing slightly better. An interesting result is given by Relief-F which now performs similarly to the first two algorithms, especially in the range of features where they reach their peaks (i.e., 60-80 features). The reason behind this improvement can be explained by the fact that Relief-F is an algorithm conceived to work on multi-class scenarios and performs well on dataset with good classes separation. Therefore, this result proves that the new dataset represents the structure of the signals better than the previous one. In this case the different clusters are statistically well represented, thus the classifier works on a more stable map where the clusters are more compact, vary less and are easily identifiable. It can also be noted that with this dataset the unsupervised method (UDFS) has the worst performance at fewer features but get better right after 60 features. It outperforms noSEL and continues to grow without reaching its peak value. This is normal since UDFS uses discriminative information and local structure of data distribution to select the relevant features. This means that UDFS need many more features to fully grasp the structure of the dataset. But as already said, the use of a large number features comes with the expense in memory demand and high computation load. In general noSEL remains the worst in terms of average precision, especially when considering high features number. A result like this was expected since one of the objective of this thesis (see section 2.4) was to demonstrate that feature selection algorithms have many advantages with respect to using raw data features in clustering and classification process. It may be said that even if it is not among the best performer in this list of algorithm, noSEL in this situation reaches a performance that is not so bad.

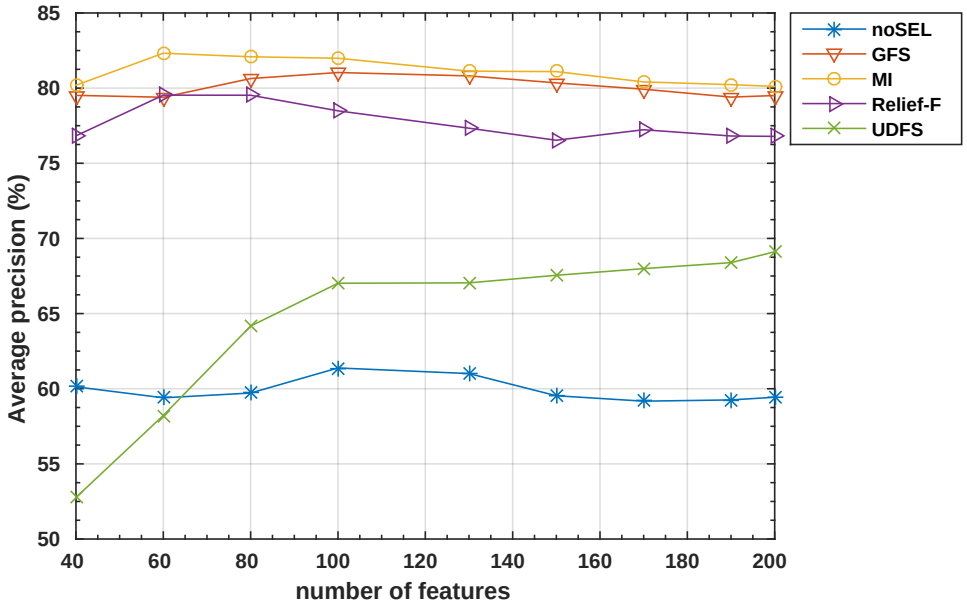


FIGURE 6.11: Average precision vs number of features for different features selection algorithms. Comparison for 12 classes with $L = 10 \times 10$.

As in the previous situations, where the quality measures were used to analyze and to evaluate the performance, also in this case they have been used to derive considerations for the classification performance. In Fig. 6.12 we have plotted the average quantization error against the selected features. The most obvious change is the drastic improvement of the Relief-F with respect to the previous 20 classes case. It is to be remembered that in that case it was inferior even to noSEL. This improvement also explains why Relief-F has performed so much better in terms of average precision, in fact it can be seen that Relief-F has committed much less quantization error. In this case, it even reaches a level comparable to GFS and MI and proves to be a good competitor to them.

Overall for the others algorithms Qe does not change much with respect to the previous case when the same map size is considered. For what concerns the topographic error, Fig. 6.13 shows that the evolution of the curves remains the same for the majority of the algorithms. The only exception is represent by Relief-F which has decreased considerably reaching the level of GFS and MI in a confirmation of its improvement also for this quality measure. It is to be noted that both in Fig. 6.12 and in Fig. 6.13 Relief-F has performed better than noSEL and that is a different result compared to what happened in the previous 20 classes case.

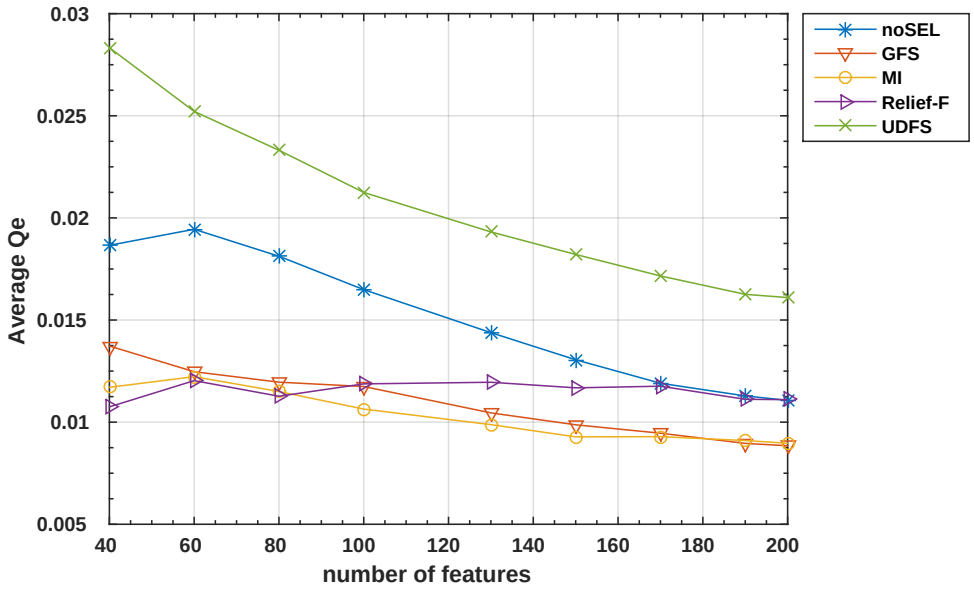


FIGURE 6.12: Average quantization error *vs* number of features for different features selection algorithms. Comparison for 12 classes with $L = 10 \times 10$.

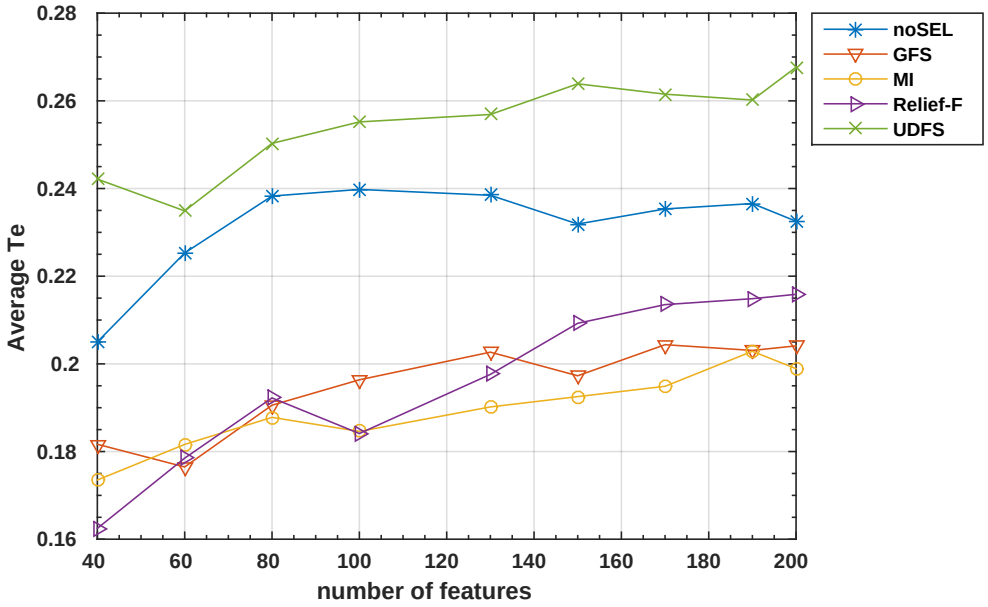


FIGURE 6.13: Average topographic error *vs* number of features for different features selection algorithms. Comparison for 12 classes with $L = 10 \times 10$.

In Fig. 6.14 and in Fig. 6.15 the maps size have been increased to 15×15 which corresponds to 225 neurons and to 20×20 which corresponds to 400 neurons respectively. This has been done to study the performance at increasing map size. In general, for what regard the average precision, what can be noted is that similar behaviors are obtained as in the situation where the map size was increased from 25 neurons to 100 neurons. That is to say that a bigger map leads to the overall increment of the average precision for all the feature selection algorithms. This scenario goes on until the map reaches a limit of data distribution representation and starts to shows not noticeable effects in the performance. A situation like this happens for example if we go from the map of size 15×15 to the map of size 20×20. In fact, in this situation the average precision shows no noticeable increments. Moreover, the curves evolution are similar to the ones in Fig. 6.11. However, there are noticeable changes in the quality measures which improvements continue to grow when the map size is increased. This is illustrate in Fig. 6.16 for the quantization error and in Fig. 6.17 for the topographic error, both are the result of the map trained on 400 neurons. It can clearly be noted for Qe that all the curves have decreased to smaller values, which means that they have increase the map quality. Especially for the three algorithms GFS, MI and Relief-F which are now below the value 0.01. While for the Te they stay in the interval value of 0.18-0.22. An other important change to be noted is the flatness of the curves which indicate that the qualities measures for this lattice remain comparable for any of the features number considered. These last quality measures confirm the capacity of larger SOM to better represent data features space into the lattice. However, there are limits to the improvements with respect to the growing map size. What we noticed is that for some bigger maps the SOM starts to overrepresent the data distribution and as a consequence there is a major presence of outliers neurons (i.e., neurons with zero label) which as we know reduce the capacity of the model to generalize in the testing phase.

Now as a post analysis considerations, taking into account the memory demand and the computational time for some maps conditions, we can conclude that the most appropriate algorithms are able to distinguish and classify the signals at relatively small number of features, around 60-80 features, giving appreciable average precision of the order of 86% for a map size of 15×15. Which is certainly a great advantage considering the size of the full set of 4957 features for each signals in the dataset. In this experiment GFS, MI and Relief-F have proved to be the most appropriate algorithms. But Relief-F and MI are filters method and have demonstrated to be faster feature selection algorithms than GFS (a wrapper) and would be the most appropriate choice in terms of rapidity, hence computational costs.

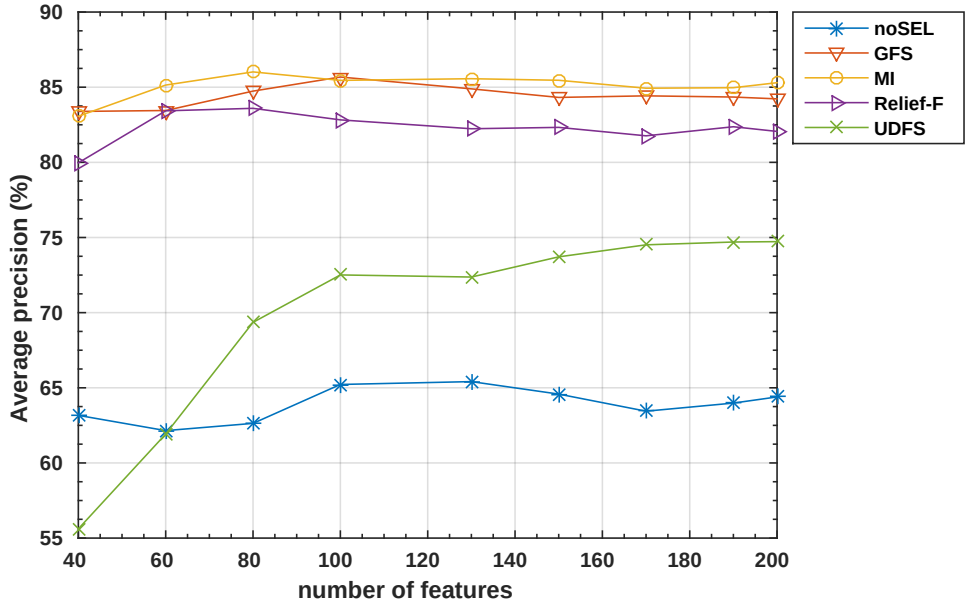


FIGURE 6.14: Average precision error *vs* number of features for different features selection algorithms. Comparison for 12 classes with $L = 15 \times 15$.

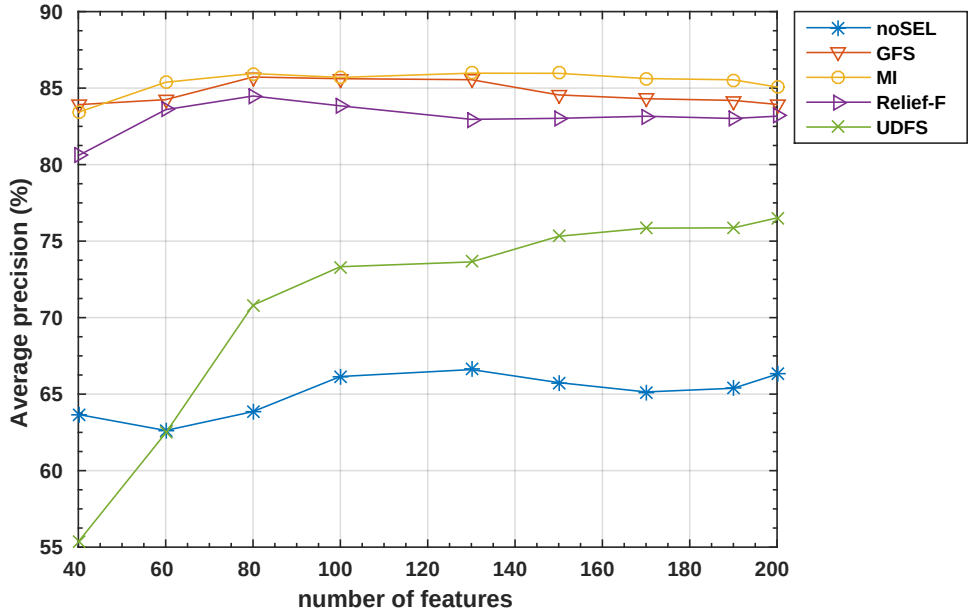


FIGURE 6.15: Average precision error *vs* number of features for different features selection algorithms. Comparison for 12 classes with $L = 20 \times 20$.

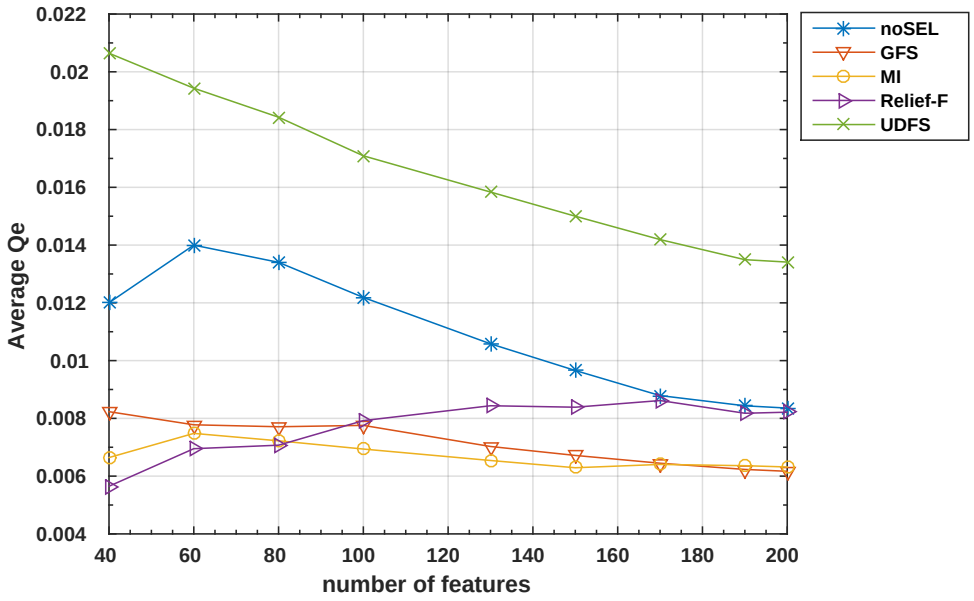


FIGURE 6.16: Average quantization error *vs* number of features for different features selection algorithms. Comparison for 12 classes with $L = 20 \times 20$.

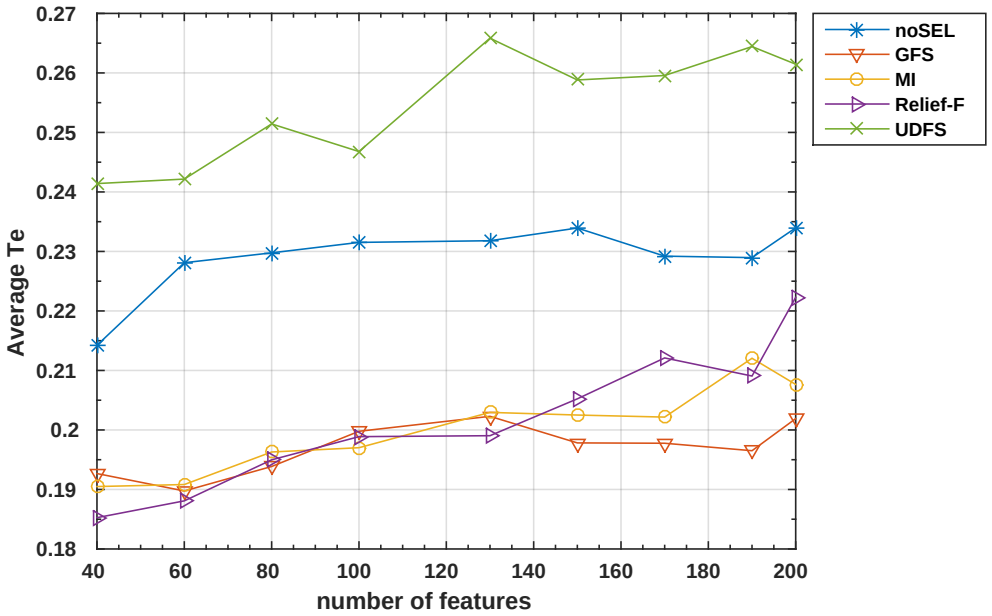


FIGURE 6.17: Average topographic error *vs* number of features for different features selection algorithms. Comparison for 12 classes with $L = 20 \times 20$.

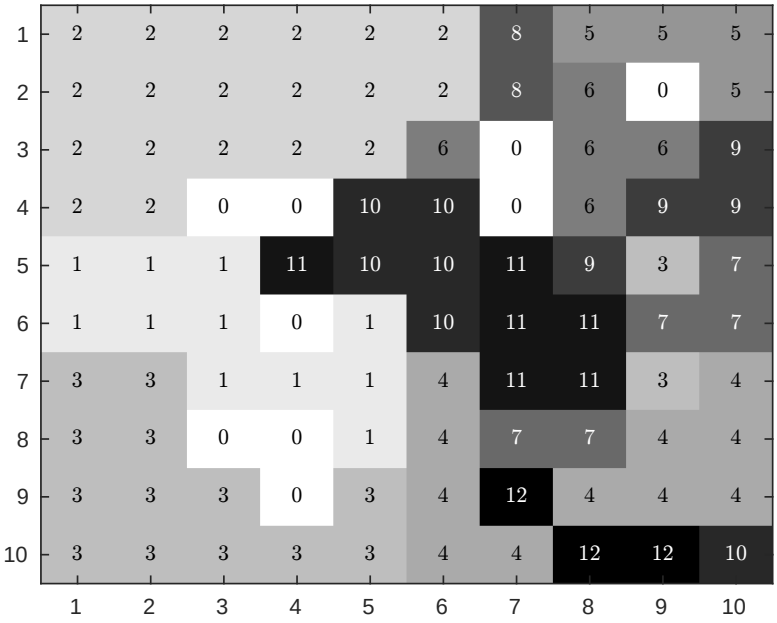


FIGURE 6.18: The classifier label information for 12 classes with $L = 10 \times 10$ and 75 features using MI

In Fig. 6.18 we have plotted the map with the information of the labels produced by the classifier using the MI feature selection algorithm of Fig 6.11 trained with 75 features, as done in Fig. 6.10 for the 20 classes case. In this case the clusters are better separated and have a representation that is well defined, even though there are still some clusters which are subdivided (in a lesser extent with respect to that of Fig. 6.10), like cluster (10) and cluster (7). As expected, the density representation of the map reflects the new dataset and it can clearly be seen that the distribution of the input data space is biased toward the bigger clusters which contain the major numbers of signals.

Chapter 7

CONCLUSION

In this thesis the design of an unsupervised clustering model able to execute the process of grouping IoT signals in clusters and classify them according to their statistics properties have been proposed. The analysis was performed on a dataset composed of different signals coming from different IoTs scenario. A method of features extraction provided by the Matlab HCTSA framework was employed and different feature selection algorithms taken from the literature have been tested for the sake of demonstrating their ability to bring advantages to the clustering and classification process.

The proposed model is obtained using the Kohonen Self-Organizing Feature map, belonging to the family of the neural network architectures featuring continuous learning and adaptation capabilities, on which an original internal classifier has been constructed. Due to the high number of classes, i.e., clusters, composed of many overlapping signals present in the first dataset, a very poor classification performance was obtained, independently of the feature selection algorithm. However, by exploiting the powerful visual analysis provide by SOM and the hierarchical algorithm, a more realistic class representation dataset was built by grouping the most correlated signals. With the new dataset all the performance have considerably improved reaching high average precision.

The encouraging results were very helpful for the choice of the suitable feature selection algorithm, used for the speeding up of the clustering and classification process of massive datasets. During the learning process various initialization parameters and maps size has been studied. We have noticed some disadvantages and some positive characteristics in the construction and the use of the SOM as a clustering tool. More precisely, it should be noted that whenever we increase the number of features, or increase the maps size, the use the SOM becomes very demanding on computing resources. The visualization capacity combine with the labels information provided by the classifier helped to distinguish whether different clusters were easily separable, or were overlapping, giving us the ability to analyze clusters in the dataset sometime without having a priori informations. When the SOM is combined with the suitable feature selection algorithm it

prove to be very efficient in helping for the achievement of higher performance with the benefit of reduced computational cost and memory use, hence, for the reduction of the energy consumption. For example, in a wireless network, the management of the energy cost of nodes is a fundamental issue to deal with. Clustering with the purpose of extracting reduced by meaningful set of information in order to reduce retransmission is one way to resolve the issue.

Some possible future research may be to extend the analysis to more datasets; to employ others feature selection algorithms, especially the unsupervised methods; to further investigate the procedure on the features extraction; to enrich the set of performance indexes to better evaluate the classification result; to extend the optimization of SOM by considering more parameters and other optimization techniques, such as nested SOMs structure, in order to achieve better clusters separability, and finally to study the impact of different classification algorithms.

Bibliography

[1] M. Weiser and R. Gold. “The origins of ubiquitous computing research at PARC”. In: *IBM Systems Journal* (1999).

[2] K. Ashton. “That "Internet of Things" thing”. In: *RFiD Journal* (2009).

[3] J. Gubbi et al. “Internet of things (IoT): A vision, architectural elements, and future directions”. In: *Future Generation Computer Systems* 29 (2013), pp. 1645–1660.

[4] L. Atzori, A. Iera, and G. Morabito. “The internet of things: A survey”. In: *Computer Networks* 54 (2010), pp. 2787–2805.

[5] X. Wang et al. “Experimental comparison of representation methods and distance measures for time series data”. In: *Data Mining Knowl. Discovery* 26 (2013), pp. 275–309.

[6] T. W. Liao. “Clustering of time series data—a survey”. In: *Pattern Recognition* 38 (2005), pp. 1857–1874.

[7] B. D. Fulcher, M. A. Little, and N. S. Jones. “Highly comparative time-series analysis: the empirical structure of time series and their methods”. In: *J. Roy. Soc. Interface* 10 (2013), p. 83.

[8] B. D. Fulcher and N. S. Jones. “Highly comparative feature-based time-series classification”. In: *IEEE Transactions on Knowledge and Data Engineering* 26 (2014), p. 3026.

[9] K. Kira and L. Rendell. “A practical approach to feature selection”. In: *In Proceedings of the ninth international workshop on Machine learning* (1992).

[10] M. Robnik Šikonja and I. Kononenko. “Theoretical and empirical analysis of relieff and rrelieff”. In: *Machine learning* 53 (2003), pp. 23–69.

[11] Y. Yang et al. “L2,1-norm Regularized Discriminative Feature Selection for Unsupervised Learning”. In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence* 2 (2011), pp. 1589–1594.

[12] S. Haykin. “Neural Networks and Learning Machines”. In: *Prentice Hall* 30 (2009).

[13] J. Eggermont. “The Correlative Brain: Theory and Experiment in Neural Interaction, ser. Studies of brain function”. In: *Springer-Verlag* (1990).

[14] P. S. Churchland and T. J. Sejnowski. “The computational brain”. In: *The MIT Press* (1992).

[15] R. Stufflebeam. ““Neurons, synapses, action potentials, and neurotransmission,”” in: *Consortium on Cognitive Science Instruction* (2008).

[16] C.M. Bishop. “"Neural Networks for Pattern Recognition"”. In: *Oxford University Press* (1995).

[17] A.K. Palit and D. Popovic. “Computational Intelligence in Time Series Fore- casting: Theory and Engineering Applications”. In: *Springer, 1st edition* (2005).

[18] G. A. Barreto and A. F. R. Araújo. “Identification and control of dynamical systems using the self-organizing map”. In: *IEEE Transactions on Neural Networks* 5 (2004), pp. 1244–1259.

[19] W. McCulloch and W. Pitts. ““A logical calculus of the ideas immanent in nervous activity,”” in: *The bulletin of mathematical biophysics* 5 (1943), pp. 115–133.

[20] G. S. Stent. “A physiological mechanism for hebb’s postulate of learning”. In: *Proceedings of the National Academy of Sciences* 70 (1973), pp. 997–1001.

[21] J.P. Changeux and A. Danchin. “Selective stabilisation of developing synapses as a mechanism for the specification of neuronal networks”. In: *Neurobiology of Learning and Memory* 2 (1990), p. 229.

[22] T. H. Brown, E. W. Kairiss, and C. L. Keenan. “Hebbian synapses: biophysical mechanisms and algorithms”. In: *Annual review of neuroscience* 13 (1990), pp. 475–511.

[23] Z. Chen et al. “Correlative learning: a basis for brain and adaptive systems”. In: *John Wiley & Sons* 49 (2008).

[24] E. Oja. “Simplified neuron model as a principal component analyzer”. In: *Journal of mathematical biology* 15 (1982), pp. 267–273.

[25] D. E. Rumelhart and D. Zipser. “Feature discovery by competitive learning”. In: *Cognitive science* 9 (1985), pp. 75–112.

[26] Y. Linde, A. Buzo, and R.M Gray. “An Algorithm for Vector Quantizer Design”. In: *IEEE Transactions on Communications* 28 (1980).

[27] J. S. Deogun S. K. Bhatia. “Conceptual clustering in information retrieval”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 3 (1998).

[28] M. H. Dunham. “Data Mining: Introductory and Advanced Topics”. In: *Prentice Hall* (2003).

[29] J. Han, M. Kamber, and A. K. Tung. “Spatial Clustering Methods in Data Mining: A Survey”. In: *Geographic Data Mining and Knowledge Discover* (2001).

[30] A. K. Jain and R. C. Dubes. "Algorithms for Clustering Data". In: *Prentice-Hall* (1988).

[31] Gan. G et al. "Data Clustering: Theory, Algorithms, and Applications". In: *Series on Statistics and Applied Probability* (2007).

[32] R. O. Duda and P. E. Hart. "Pattern Classification and Scene Analysis". In: *John Wiley& Sons* (1973).

[33] T. Kohonen. "The self-organizing map". In: *Proceedings of the IEEE* 78 (1990), pp. 1464–1480.

[34] T. Kohonen. "Self-organized formation of topologically correct feature maps". In: *Biol. Cybern* 43 (1982), pp. 59–69.

[35] H. Ritter, T. Martinetz, and K. Schulten. "Neural Computation and Self-Organizing Maps:An Introduction". In: *Addison-Wesley* (1992).

[36] T. Kohonen. "Self–Organizing Maps (2 ed.)" In: *Springer, Heidelberg* 30 (1997).

[37] S. Amari. "Topographic organization of nerve fields". In: *Bulletin of Mathematical Biology* 42 (1980), pp. 333–364.

[38] H. Ritter. "Asymptotic level density for a class of vector quantization processes". In: *IEEE Transactions on Neural Networks* 2 (1991), pp. 173–175.

[39] Wikipedia. In: (). URL: `https://en.wikipedia.org/wiki/Precision_and_recall`.

[40] F. Murtagh. "Complexities of hierarchic clustering algorithms: State of the art". In: *Computational Statistics Quarterly* 2 (1984), pp. 101–113.