Pós-Graduação em Ciência da Computação

Pedro Henrique Magalhães Braga

Semi-Supervised Self-Organizing Maps with Time-Varying Structures for Clustering and Classification

Recife
2019

Pedro Henrique Magalhães Braga

Semi-Supervised Self-Organizing Maps with Time-Varying Structures for Clustering and Classification

A M.Sc. Dissertation presented to the Center of Informatics of Federal University of Pernambuco in partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

Concentration Area: Computational Intelligence
Advisor: Hansenclever de França Bassani

Recife
2019

I dedicate this dissertation to all my family, especially to the memory of my father and grandfather, João Gonçalves Braga, that is always in my heart and mind. To friends and professors who gave me the necessary support to get here.

ABSTRACT

In recent years, the advances in technology have produced datasets of increasing size, not only regarding the number of samples but also the number of features. Unfortunately, despite these advances, creating a sufficiently large amount of properly labeled data with enough examples for each class is not an easy task. Organizing and labeling such data is challenging, expensive, and time-consuming. Also, it is usually done manually, and people can label with different formats and styles, incorporating noise and errors to the dataset. Hence, there is a growing interest in semi-supervised learning, since, in many learning tasks, there is a plentiful supply of unlabeled data, but insufficient labeled ones. Therefore, at the current stage of research, it is of great importance to put forward semi-supervised learning models aiming to combine both types of data, in order to benefit from the distinct information they can provide, to obtain better performances of both clustering and classification tasks, that would expand the range of machine learning applications. Moreover, it is also important to develop methods that are easy to parameterize in a way that become robust to the different characteristics of the data at hand. In this sense, the Self-Organizing Maps (SOM) can be considered as good options to address such objectives. It is a biologically inspired neural model that uses unsupervised and incremental learning to produce prototypes of the input data. However, such an unsupervised characteristic makes it unfeasible for SOM to execute Semi-Supervised Learning. In that way, this Dissertation presents some new proposals based on SOM to perform Semi-Supervised learning tasks for both clustering and classification. It is done by introducing to SOM the standard concepts of Learning Vector Quantization (LVQ), which can be seen as its supervised counterpart, to build hybrid approaches. Such proposals can dynamically switch between the two types of learning at training time, according to the availability of labels and automatically adjust themselves to the local variance observed in each data cluster. In the course of this work, the experimental results show that the proposed models can surpass the performance of other traditional methods not only in terms of classification but also regarding clustering quality. It also enhances the range of possible applications of a SOM and LVQ-based models by combining them with recent and promising techniques from Deep Learning to solve more complex problems commonly found in such field.

Keywords: Self-Organizing Maps. Semi-Supervised Learning. Clustering. Classification.

RESUMO

Nos últimos anos, os avanços na tecnologia tem produzido conjuntos de dados de tamanhos cada vez maiores, não apenas em relação ao número de amostras, mas também ao número de características. Infelizmente, apesar desses avanços, criar uma quantidade suficientemente grande de dados, adequadamente rotulados com amostras suficientes para cada classe, não é uma tarefa fácil. Organizar e rotular esses dados é desafiador, caro e demorado. Além disso, por ser geralmente feito de forma manual, pessoas podem rotular com diferentes formatos e estilos, incorporando ruído e erro aos dados. Assim, há um crescente interesse em aprendizagem semi-supervisionada, uma vez que, em muitas tarefas de aprendizagem, existe uma abundante quantidade de dados não rotulados, em contrapartida aos rotulados. Portanto, no atual estágio de pesquisa, é de grande importância desenvolver modelos de aprendizagem semi-supervisionada, com o intuito de combinar os dois tipos de dados, a fim de se beneficar das distintas informações que eles podem fornecer. Dessa forma, é possível obter melhores desempenhos para ambas as tarefas de agrupamento e classificação, o que pode expandir a gama de aplicações em aprendizagem de máquina. Ainda, desenvolver modelos que sejam fáceis de parametrizar de tal maneira que se tornem robustos às diferentes características dos dados disponíveis também é relevante. Nesse sentido, Mapas Auto-Organizáveis (SOM) podem ser considerados boas opções. O SOM é um modelo neural, biologicamente inspirado, que usa aprendizagem não-supervisionada e incremental para produzir protótipos dos dados de entrada. No entanto, sua característica não-supervisionada inviabiliza a realização de aprendizagem semi-supervisionada. Esta Dissertação apresenta algumas novas propostas de modelos baseados em SOM para realizar tarefas de aprendizagem semi-supervisionada tanto para agrupamento, como para classificação. Isso é feito introduzindo ao SOM conceitos da tradicional Quantização Ventorial (LVQ), que pode ser vista como sua versão supervisionada para construir abordagens híbridas. Tais propostas podem alternar dinamicamente entre duas formas de aprendizagem em tempo de treinamento, de acordo com a disponibilidade de rótulos, além de se ajustarem automaticamente às variâncias locais observadas em cada grupo de dados. No decorrer deste trabalho, os resultados experimentais mostram que os modelos propostos podem superar o desempenho de outros métodos tradicionais, não apenas em termos de classificção, mas também na qualidade de agrupamento. As propostas também aumentam a gama de possíveis aplicações de modelos baseados em SOM e LVQ, uma vez que os combinam com técnicas recentes e promissoras de aprendizagem profunda para resolver problemas mais complexos comumente encontrados em tal área.

Palavras-chave: Mapas Auto-Organizáveis. Aprendizagem Semi-Supervisionada. Agrupamento. Classificação.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

ALTSS-SOM	Adaptive Local Thresholds Semi-Supervised Self-Organizing Map
BMU	Best Matching Unit
cdf	Cumulative Distribution Function
CE	Clustering Error
CIFAR10	Canadian Institute For Advanced Research
ClaE	Classification Error
CNN	Convolutional Neural Network
CSOM	Convolutional Self-Organizing Map
DOC	Densitive-based Optimal projective Clustering
DSOM	Deep Self-Organizing map
DSSL	Deep Semi-Supervised Learning
DSSOM	Dimension Selective Self-Organizing Map
EM	Expectation Maximization
GLVQ	Generalized Learning Vector Quantization
GNG	Growing Neural Gas
GPU	Graphics Processing Units
GRLVQ	Generalized Relevance Learning Vector Quantization
GWR	Growing When Required
HMRF	Hidden Markov Random Fields
IEEE	Institute of Electrical and Electronics Engineers
IJCNN	International Joint Conference on Neural Networks
LARFDSSOM	Local Adaptive Receptive Field Dimension Selective Self-Organizing Map
LARFSOM	Local Adaptive Receptive Field Self-Organizing Map
LHS	Latin Hypercube Sampling
LP	Label Propagation
LS	Label Spreading
LVQ	Learning Vector Quantization
MLP	Multilayer Perceptron
NLP	Natural Language Processing
PCA	Principal Component Analysis
PROCLUS	PROjected CLUStering algorithm
ReLU	Rectified Linear Units
S^3VM	Semi-supervised Support Vector Machines

S3C	Spike-and-Slab Sparse Coding
SIFT	Scale Invariant Feature Transform
SOM	Self-Organizing Map
SSL	Semi-Supervised Learning
SS-SOM	Semi-Supervised Self-Organizing Map
SURF	Speeded-Up Robust Features
SVHN	Street View House Numbers
SVM	Support Vector Machine
t-SNE	t-Distributed Stochastic Neighbor Embedding
VIK	View-Invariant K-means
WCCI	World Congress on Computational Intelligence
WRN-28-2	Wide ResNet

LIST OF SYMBOLS

Δ	Gradient
Ω	Input space
Φ	Nodes indexes space
Ψ	Prototype space
β	Parameter that controls the rate of change of the moving average vector
$\boldsymbol{\delta}$	Moving average distance vector
$\boldsymbol{\lambda}$	The weight vector of GRLVQ
$\boldsymbol{\theta}$	Average sample vector of Batch SS-SOM
ε	A small number to avoid division by zero
η	Learning rate of SOM
$\hat{\boldsymbol{\delta}}$	Corrected moving average distance vector
$\mathbb{R}$	Real numbers set
ω	Relevance vector
ρ	Global relevance vector of DSSOM
σ	The width or radius of the topological neighborhood function of SOM
τ	Time-constant the controls the exponentual decay rate η
ξ	Weighted derivative distance

LIST OF ALGORITHMS

CONTENTS

1 INTRODUCTION

In a broad sense, the learning processes can be distinguished based on their fundamentally different types of tasks. In the first, called learning with a teacher, or supervised learning, involving only labeled data, the goal is to learn a function that maps an input to an output based on a set of labeled training examples. Each example consists of an input object and a corresponding desired (target) response. In the second, called unsupervised learning, involving only unlabeled data, there is no external teacher or critic to oversee the learning process. Instead, provision is made to find interesting structure in the data by learning from its statistical regularities to develop the ability to form internal representations for encoding its features. Finally, in the so-called reinforcement learning, the learning of an input-output mapping is performed through continued interaction with the environment in order to minimize some kind of cost function (Haykin, 2009; Chapelle *et al.*, 2009)

Moreover, over the last few years, the use of machine-learning technology has driven many aspects of modern society. Recent research on Artificial Neural Networks with supervised learning has shown significant advances. It is the most common form of machine learning, deep or not (LeCun *et al.*, 2015). Nowadays, it is not unusual to see on the news several practical applications, in diverse areas, such as Robotics (Levine *et al.*, 2016), Genomics (Araujo *et al.*, 2013), and Natural Language Processing (Zhou *et al.*, 2016). A key to the success of supervised learning, especially, deep supervised learning, is the availability of sufficiently large labeled training data. Unfortunately, despite these advances, creating a sufficiently large amount of properly labeled data with enough examples for each class (sometimes, in the order of thousands of patterns per class) is not an easy task. Organizing and labeling such data is a very complicated job. Labeling is expensive, time-consuming, and challenging. Also, it is usually done manually. Thus, people can label with different formats and styles, incorporating noise and errors to the dataset (Jindal *et al.*, 2016).

Because of that, the use of supervised learning methods became impractical in many applications such as in the medical field, where it is extremely difficult and expensive to obtain balanced labeled data. In other areas, such as robotics, the dynamic imposed makes it impossible to have real-time labels. Also, in certain problems, new categories of elements may frequently arise, making it infeasible to create a comprehensive previously labeled training dataset. On the other hand, unlabeled data usually can be easily obtained due to such advances that have produced datasets of increasing size, not only regarding the number of samples but also the number of features. In this sense, unsupervised learning can be applied to perform clustering tasks. However, clustering is a more difficult and challenging problem, and the nature of the data can make the clustering tasks even more difficult, so any kind of additional prior information in respect to the data can be useful to obtain a better performance.

Therefore, at the current stage of research, it is of great importance to put forward methods that can combine both types of data in order to benefit from the information they can provide, each of them in their way, that would expand the range of machine learning applications (Chapelle *et al.*, 2009). To do so, and also obtain performance improvements, Semi-Supervised Learning (SSL) is typically applied. It is a halfway between supervised and unsupervised learning and can be used to both clustering and classification tasks even with a lack of labeled data because unlabeled data has a large amount of discriminative information that can be fully explored by SSL algorithms (Chapelle *et al.*, 2009; Schwenker & Trentin, 2014). In addition, it is also worth pointing out that such interest for SSL is growing in the machine-learning (Xiaojin & Zoubin, 2002; Zhou *et al.*, 2004) alongside in the deep learning context (Goodfellow *et al.*, 2016; LeCun *et al.*, 2015), in which is also making use of SSL, such as in (Rasmus *et al.*, 2015; Liu *et al.*, 2015; Dozono *et al.*, 2016; Chen *et al.*, 2018; Hailat *et al.*, 2018).

Moreover, SSL can be further classified into semi-supervised classification and semi-supervised clustering (Schwenker & Trentin, 2014). Firstly, in semi-supervised classification, the training process tries to exploit additional information (often available as label classes) together with the unlabeled data to achieve a more accurate classification function. Secondly, in semi-supervised clustering, this prior information is used to obtain a better clustering performance (Basu *et al.*, 2002; Schwenker & Trentin, 2014). In this regard, semi-supervised approaches such as Label Propagation (LP) (Xiaojin & Zoubin, 2002) and Label Spreading (LS) (Zhou *et al.*, 2004) can be pinpointed. They operate on proximity graphs or connected structures to spread and propagate information about the class to nearby nodes according to a similarity matrix. This is based on the assumption that nearby entities should belong to the same class, in contrast to far away entities (Xiaojin & Zoubin, 2002; Herrmann & Ultsch, 2007). However, they have some drawbacks concerning the estimation of propagation radius, which is crucial for convergence.

Still, in the context of unsupervised learning, it is possible to highlight the prototype-based methods as a start point for introducing modifications to perform semi-supervised learning. They produce as a result prototypes that can properly represent the clusters identified, which are normally formed by similar samples that share general characteristics. K-Means (Basu *et al.*, 2002) and SOM (Kohonen, 1990; Bassani & Araujo, 2015) are the most basic examples of this approach. Therefore, semi-supervised K-means and SOM-based methods were very successful demonstrating their advantages over standard unsupervised approaches, being successfully applied for both semi-supervised clustering and classification tasks (Jain, 2010).

SOM is an unsupervised learning method, frequently applied for clustering. The Learning

Vector Quantization (LVQ) (Kohonen, 1995), on the other hand, is a supervised method, normally used for classification, that shares many similarities with SOM. They both were proposed by Teuvo Kohonen, and since then, various modifications have been proposed to improve their performances in more challenging problems, like those with thousands of features, commonly found in areas such as data mining (Kriegel *et al.*, 2009) and bioinformatics (Araujo *et al.*, 2013).

High-dimensional and more complex data pose different challenges for clustering and classification tasks. In particular, traditional similarity measures often applied in prototype-based methods may become meaningless due to the curse of dimensionality (Köppen, 2000), in which objects may appear approximately equidistant from each other, that is aggravated by the presence of irrelevant dimensions in the dataset. Hence, such problems require some adaptations and more sophisticated approaches. In this context, subspace clustering and projected clustering methods appear as common options. They aim at determining clusters in subspaces of the input dimensions. It involves not only the clustering itself but also the identification of the relevant subsets of the input dimensions for each cluster (Kriegel *et al.*, 2009). One way to achieve this is by applying local relevances to the input dimensions, which is the usual manner that both SOM and LVQ-based methods use to deal with such problems. It has been shown to provide significant performance improvements.

Recent SOM-based methods usually employ a threshold defining the minimum level of similarity for an input pattern to be considered associated with a cluster prototype. This threshold level is a parameter of the model which is shared by all prototypes (Bassani & Araujo, 2015; Bassani & Araújo, 2012). Thus, there are difficulties for the methods to define the regions that each prototype can represent independently. Those regions can be viewed as the local receptive field of the prototypes, and are commonly estimated using supervised approaches, as in Fischer *et al.* (2016). Such regions can also be associated with the idea of rejection options, early introduced by Chow (1970). It is related to the conditions of taking a classification or recognition decision for a particular point or a data region. Still, according to Chow (1970), because of uncertainties and noise inherent in any pattern recognition task, errors are generally unavoidable. Uncertainty has typically two sources: points being outliers or located in ambiguous regions (Vailaya & Jain, 2000). The option to reject is introduced to avoid an excessive misrecognition rate by converting it into rejection. This option can be viewed as the conditions for taking a classification or recognition decision for a particular point or a data region. For instance, an algorithm can perform reject-option decisions in such a way that any samples assigned to a class or cluster will only be accepted if some criterion is met. This is normally applied at classification time when there is considerable uncertainty associated. Thus, the input pattern does not affect the learning procedure. However, some approaches employ such decisions at training time aiming to learn such conditions more precisely (Fischer *et al.*, 2016).

In the case of SOM-based methods, those decisions can be related to the prototypes that are represented by the nodes in the map, which must accept or not other input patterns to be part of their representation pool. Still, such reject-option decisions define the first step towards an adaptation of the model complexity tailored to data regions with a high degree of uncertainty, e.g., introducing new prototypes which are capable of representing novel aspects of the data. (Fischer *et al.*, 2014). So far, the great part of models that use rejection options deal with just a single threshold shared by all prototypes, as well as most of them can handle only binary classification (Fischer *et al.*, 2016).

Moreover, the existing prototype-based methods are not suitable for working with data as complex as raw images. However, if a feature extraction step is carried out beforehand they can perform well. One way to do this is by applying standard extractors such as Scale Invariant Feature Transform (SIFT) (Lowe, 1999) and Speeded-Up Robust Features (SURF) (Bay *et al.*, 2006). Nevertheless, more recent approaches based on Deep Learning (LeCun *et al.*, 2015) and Transfer Learning (Yosinski *et al.*, 2014) techniques have presented promising results, such as in Oliver *et al.* (2018), and Medeiros *et al.* (2018).

Thus, not only the data itself can be used for clustering or classification tasks, but also useful characteristics or features that can be identified and extracted. Also, it is common to see many works using features extracted by neural networks models pre-trained on large datasets such as ImageNet (Deng *et al.*, 2009) to work around the computational cost necessary to train such amounts of data, whereby exploring the generalization capability of models to improve the performance and reduce the training cost and time. Finally, such strategies are often neglected in SSL field, but it is still a good option to take into consideration, including for subspace clustering.

Considering what has been set out, the objective of this Dissertation is to develop Semi-Supervised models based on the concepts of both SOM and LVQ in order to improve the results obtained with traditional SSL methods in the literature. From this point on, this Dissertation also has the following specific objectives:

1. Extend the application range of the models;

2. Improve not only the classification rate but also the clustering quality when there is no label available;

3. Develop a strategy to estimate local rejection options as a function of both local variance and the relevance of input dimensions to make pattern rejection decisions;

4. Reduce the parametric sensitivity and stabilize the performance of models to not degrade with changes in parameter values.

The main objective is first achieved with the proposal of Semi-Supervised Self-Organizing Map. Later on, to extend the application range of the models, the Batch SS-SOM is proposed. Finally, the last objectives are accomplished with the development of the last proposed model, Adaptive Local Thresholds Semi-Supervised Self-Organizing Map. The results obtained with SS-SOM were shown to have led to significant improvements in classification results for small amounts of labeled data in comparison with other semi-supervised models. Batch SS-SOM achieved surprisingly good results for the tasks it was designed for. Finally, ALTSS-SOM has shown to be very effective for clustering tasks solely likewise for classification, improving the results obtained with SS-SOM.

The rest of this dissertation is organized as follows: Chapter 2 presents essential concepts related to the areas where this work is inserted. Chapter 3 introduces related work in the literature that is the core behind the ideas developed to build the proposed models. Chapter 4 describes in detail the first model proposed as well as one of its extensions and preliminary results. Chapter 5 introduces a detailed explanation of the last proposed model together with some of its preliminary outcomes. Later on in Chapter 6, the experimental setup, methodology, obtained results, and comparisons will be discussed in order to validate the proposals. Finally, Chapter 7 concludes this dissertation by analyzing the obtained results and indicating future directions and practical applications for the proposed models.

2 THEORETICAL BACKGROUND

In the current chapter, some concepts will be discussed in order to establish the theoretical foundation related to the developed research. First, in Section 2.1, a SOM and how it works is explained. Its understanding is essential for the ideas proposed in the present work to be clear. Second, in Section 2.2, the LVQ is introduced. It is also of a great importance once its standard concepts are explored to build a hybrid approach by combining both SOM and LVQ. Third, Section 2.3 defines the concept of rejection option. Later on, Section 2.4 presents the problems of High-Dimensional Data, Subspace and Projected Clustering. Finally, Section 2.5 introduces and defines SSL, whereas Section 2.6 highlights forms of evaluating subspace and projected clustering problems.

2.1 SELF-ORGANIZING MAPS

SOM, proposed by Kohonen (1990), was intended as a viable alternative for more traditional neural network architectures. It is based on three essential processes: 1) Competition; 2) Cooperation; and 3) Adaptation, which leads to a *competitive learning*, where the neurons compete among themselves to be the most activated when an input pattern is presented. The competition results in a process that is called a *winner-takes-all* competition, which produces just one *winning neuron*. In a SOM, the neurons are placed at the vertices of a *lattice* or grid that is commonly one or two-dimensional. Therefore, SOM is characterized by the formation of a topographic map of the input patterns, in which the spatial locations of the neurons in the grid are indicative of intrinsic statistical features contained in the input patterns (Haykin, 2009).

SOM was initially created with the principal goal of transforming an incoming signal pattern of arbitrary dimension into a one or two-dimensional discrete map, by performing this transformation adaptively in a topologically ordered fashion. This results in a topology that maps data in a high-dimensional input space distribution into units in a low-dimensional map while preserving the similarities and the relations found between the data points in the input space. Therefore, it is possible to pinpoint two essential aspects of SOM: its capacity to create abstractions and its simplified way of exhibiting information. These aspects allow several applications in diverse areas, such as statistical pattern recognition, control of robot arms, sentence understanding, image compression, and much more (Kohonen, 1990). All of this converged to make SOM a model widely used for clustering tasks (Kohonen, 1990; Haykin, 2009).

An interesting fact about the development of SOM is its neurophysiological inspiration, in particular, in a distinct feature of the human brain: The brain is organized in many places in such a way that different sensory inputs are represented by topologically ordered computational maps (Haykin, 2009). It comes from both anatomical and physiological evidence of lateral interaction between cells: 1) in neural tissues, an activated neuron that triggers a pulse causes a short-range excitation of other neurons that ranges from 50 to 100 μm; 2) the propagation of the excitation to areas not related to the excitatory process is prevented by a penumbra of inhibitory action around excited area; and 3) a weaker excitatory action surrounds the inhibitory penumbra and ranges up to several centimeters of radius (Kohonen, 1982). Thus, certain parts of the brain organize themselves in a way that sensory inputs are represented by topologically ordered computational maps (Miikkulainen *et al.*, 2006). Particularly, sensory inputs such as tactile (Kaas *et al.*, 1983), visual (Hubel & Wiesel, 1962), and acoustic (Suga, 1990) are mapped onto different areas of the cerebral cortex in a topologically ordered manner. SOM captures the essential features of computational maps in the brain and yet remains computationally tractable. Hence, it is capable of performing data compression (i.e., by prototyping and dimensionality reduction of the inputs) (Haykin, 2009).

2.1.1 Basic Structure of a SOM

The basic structure of a SOM (Figure 1) consists of an input layer and an output layer. The input layer receives the synaptic inputs from the environment and propagates them to the output layer.

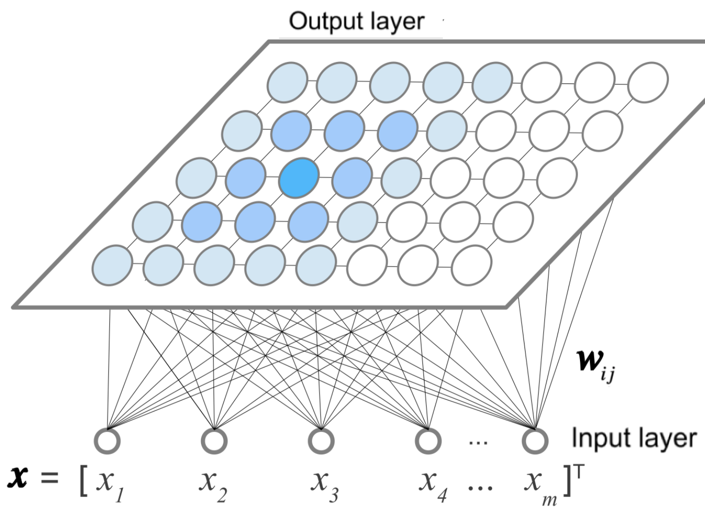

Output layer
$\boldsymbol{w}_{ij}$
Input layer
$\boldsymbol{x} = [x_1 \quad x_2 \quad x_3 \quad x_4 \ \dots \ x_m]^T$

Figure 1: The basic structure of a SOM. The units $\boldsymbol{x}$ are the input pattern. Each synaptic-weight vector $\boldsymbol{w}_{ij}$ represents a connection between the i-th node in the input layer and the j-th node in the output layer. In this configuration, each node in the output layer is directly connected with its neighbors (Adapted from Bassani (2014)).

Let m denote the dimension of the input space and $\boldsymbol{x} = [x_1, x_2, ..., x_m]^T$ the input pattern vector. The synaptic-weight vector of each neuron in the map has the same dimension as the input space. Let the synaptic weight vector of neuron j be denoted as $\boldsymbol{w}_j = [w_{j1}, w_{j2}, ..., w_{jm}]^T$, with $j = 1, 2, ..., l$, where l is the total number of neurons in the output map. The output layer computes the final map resulting from the self organization process and its topology is normally a two-dimensional *lattice*, where each node is connected with their immediate neighbors. The idea can be simply described as to store a large set of input vectors $\boldsymbol{x} \in \Omega$ by finding a smaller set of prototypes $\boldsymbol{w}_j \in \Psi$ so as to provide a good approximation, nearly to optimum, to the original input space Ω (Haykin, 2009; Kohonen, 1990).

2.1.2 Self-Organization in a SOM

There are three essential steps involved in the self-organization or learning process of a SOM: competition, adaptation, and cooperation. When an input pattern $\boldsymbol{x}$ is presented to the input layer, it is propagated to all nodes in the grid. Then the competition process begins among the nodes in order to choose the one that best matches the input pattern. In the SOM defined by Kohonen (1982, 1990), this is done by selecting the node with the minimum Euclidian distance to the input $\boldsymbol{x}$ as the winner node $i(\boldsymbol{x})$ (Equation 2.1).

$$i(\boldsymbol{x}) = \arg\min_j [D(\boldsymbol{x}, \boldsymbol{w}_j)], \quad j \in \Phi, \tag{2.1}$$

where Φ denotes all the nodes of the map and $D(\boldsymbol{x}, \boldsymbol{w}_j)$ is the Euclidian distance between $\boldsymbol{x}$ and the synaptic weight vector $\boldsymbol{w}_j$, as follows in Equation 2.2:

$$D(\boldsymbol{x},\boldsymbol{w}_j) = \sqrt{\sum_{i=1}^{m}(x_i - w_{ji})^2}, \tag{2.2}$$

where m is the number of dimensions. However, it is worth mentioning that minimizing the squared Euclidean distance is mathematically equivalent, but more efficient computationally.

In self-organizing process, synaptic-weight vectors $\boldsymbol{w}_j$ in the network are required to change in relation to the input vector $\boldsymbol{x}$. This defines the adaptation step. Moreover, it is crucial to the self-organization process that the synaptic-weight vectors are not affected independently of each other, but as topologically related subsets. It can be done by changing all the synaptic-weight vectors of a winner node and its local neighbors to make them closer to the input pattern, considering a more accurate approximation at every step. Different input signals in different steps affect different regions of the grid. Thus, after many steps of self-organization, the synaptic-weights in the grid will tend to acquire smoothly related values, equivalently to the input space (Kohonen, 1982, 1990).

Finally, Equation 2.3 defines the updated weight vector $\boldsymbol{w}_j(n+1)$ of a neuron as follows:

$$\boldsymbol{w}_j(n+1) = \boldsymbol{w}_j(n) + \eta(n)h_{j,i(\boldsymbol{x})}(n)(\boldsymbol{x} - \boldsymbol{w}_j(n)), \tag{2.3}$$

where $\eta(n)$ is the learning-rate decay function and $h_{j,i(\boldsymbol{x})}(n)$ is a topological neighborhood function that also decays throughout the time as $\eta(n)$.

For a good global ordering, Kohonen (1990) experimentally showed that the learning rate should be time-varying, starting at some high initial value η_0 and them shrink monotonically with time as per Equation 2.4.

$$\eta(n) = \eta_0 \exp\left(-\frac{n}{\tau_1}\right), \quad n = 0,1,2,\ldots, \tag{2.4}$$

where τ_1 is a time constant that controls the exponential decay rate as the time step n grows.

Still, it is possible to decompose the self-organization process into two phases: the self-organizing and the convergence phase. In the first, the topological ordering of the neurons is performed. In the latter, the fine-tuning of the feature map takes place and therefore provides an accurate statistical quantification of the input space (Haykin, 2009).

Furthermore, a neuron that is firing tends to excite neurons in its immediate neighborhood more than those farther away from it. This observation leads to the introduction of a topological neighborhood around the winning neuron j and makes it decay smoothly as the lateral distance increases (Haykin, 2009). In particular, such observation provides the idea of the cooperation step, where the winner node locates the center of a topological neighborhood of cooperating nodes that also must be adjusted. The function $h_{j,i}$ (Equation 2.5) denotes the topological neighborhood centered on the winning node i and encompassing a set of excited cooperating neurons j. It is directly related to the level of adaptation applied to each node in the neighborhood. This function is a unimodal function of the lateral distance. Therefore, it satisfies two distinct requirements: 1) it is symmetric about the maximum point, which is defined by $\left\|\mathbf{r}_j - \mathbf{r}_i\right\|^2 = 0$ meaning that it reaches the maximum for the winner (i.e., $j = i$); and 2) the amplitude of the neighborhood decreases monotonically with a Gaussian function with respect to increasing lateral distances (Kohonen, 1982, 1990; Haykin, 2009).

$$h_{j,i(\boldsymbol{x})}(n) = \exp\left(-\frac{\left\|\mathbf{r}_j - \mathbf{r}_i\right\|^2}{2\sigma^2(n)}\right), \quad j \in \Phi, \quad n = 0, 1, 2, \ldots, \tag{2.5}$$

where $\mathbf{r}_i$ and $\mathbf{r}_j$ defines the position of the winner i and its neighbor j in the grid, Φ denotes all the nodes of the map, and $\sigma(n)$ (Equation 2.6) represents the width or radius of the topological neighborhood function. The $\sigma(n)$ value measures the degree to which excited neurons in the vicinity participate in the learning process. It starts with a high value, σ_0, and decreases monotonically with time, corresponding to Equation 2.5 (Kohonen, 1982; Haykin, 2009).

$$\sigma(n) = \sigma_0 \exp\left(-\frac{n}{\tau_2}\right), \quad n = 0, 1, 2, \ldots, \tag{2.6}$$

where τ_2 is another time constant that controls the exponential decay rate as the time step n grows.

Thereby, the basic structure of a SOM is defined. It is important mentioning that input patterns with the same winner node are considered to belong to the same cluster and then can be represented by the synaptic-weight vector as the cluster prototype, once it summarizes the characteristics of the grouped inputs.

2.2 LEARNING VECTOR QUANTIZATION

The LVQ, also proposed by Kohonen (1995), is a family of algorithms for statistical pattern classification that uses prototypes (codebook vectors) to represent class regions. These regions are defined by hyperplanes between prototypes, resulting in Voronoi partitions (Nova & Estévez, 2014). While the basic SOM is unsupervised, the LVQ is characterized by supervised learning. Also, unlike in SOM, no neighborhoods around the so-called "winner" are defined during the learning process, whereby no spatial order of the codebook vectors is expected to ensure. Since LVQ was meant to be strictly for statistical classification and recognition, its only aim is to define class regions in the input space (Kohonen, 1995).

Basically, the LVQ classification scheme is based on the Best Matching Unit (BMU) (*winner-takes-all* strategy), as in SOM, with a Hebbian learning-based approach. To do so, let $\boldsymbol{X} = \{(\boldsymbol{x}_i, y_i), i \in 1, \ldots, N\}$ be the training set of N samples, where $\boldsymbol{x}_i$ is a m-dimensional vector in the feature space, and $y_i \in \{1, \ldots, C\}$ is its class label expressed by one of C possible classes. Then, LVQ iteratively tries to improve some initial set of P prototypes, which are characterized by $W = \left(\boldsymbol{w}_j, c_j\right), j \in 1, \ldots, P$, where $\boldsymbol{w}_j$ is m-dimensional as $\boldsymbol{x}_i$, and $c_j \in \{1, \ldots, C\}$ its class label, as y_i. The winner prototype is selected as the one with the minimum distance to the input vector, so the receptive field of $\boldsymbol{w}_j$ is defined by Equation 2.7 (Kohonen, 1995; Nova & Estévez, 2014).

$$R^j = \{\boldsymbol{x}_i \in \boldsymbol{X} = \arg\min_j [D(\boldsymbol{x}_i, \boldsymbol{w}_j)], \forall\, j \in 1, ..., P\},\ i \in 1, ..., N\}, \tag{2.7}$$

where $D(\boldsymbol{x}_i, \boldsymbol{w}_j)$ is a distance measure.

The learning process aims to determine the weight vectors in a way that the training data are mapped to their corresponding class label region. Because the standard LVQ has some drawbacks, various modifications of LVQ were proposed over the years. They ensure faster convergence, a better adaptation of the receptive fields, and an adaptation for complex data structures (Nova & Estévez, 2014).

2.3 REJECT OPTIONS

According to Chow (1970), because of uncertainties and noise inherent to any pattern recognition task, errors are generally unavoidable. Uncertainty has typically two reasons: points being outliers or located in ambiguous regions (Vailaya & Jain, 2000). The option to reject is introduced to avoid an excessive misrecognition rate by converting it into rejection. In this context, the concept of reject options is introduced. It is related to the conditions of taking a classification or recognition decision for a particular point or a data region. An early reject option defined by Chow (1970) says that if the costs for a misclassification versus a rejected data point are known, one can determine an optimal rejection threshold based on the probability of misclassification. A reject-option classifier is demonstrated in Figure 2. It can be seen that the class-conditional density $p(x|\omega_t)$ is thresholded in such a way that any object x assigned to ω_t will only be accepted if $p(x|\omega_t) > t_d$. It is typically applied at classification time when there is considerable uncertainty associated. Thus, the input pattern does not modify the models. However, some approaches use such rejection rule at training time aiming to learn such rules more precisely to improve the outcomes (Jiang & Mojon, 2003; Fischer *et al.*, 2016).

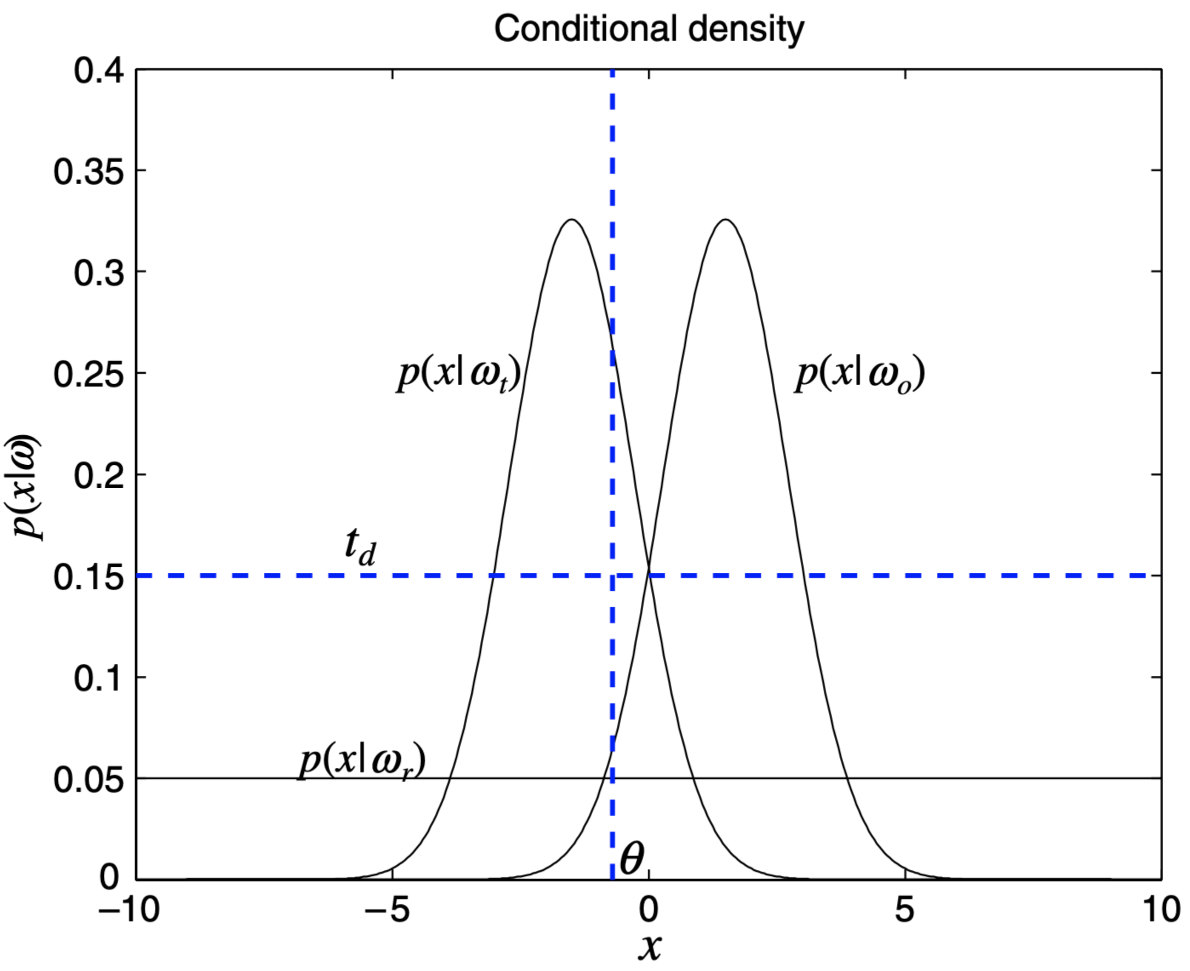


Figure 2: A synthetic example that illustrates the class-conditional densities for ω_t, ω_o, and ω_r, with a distance-based reject-option classifier. The classification boundary is specified by θ, and the rejection boundary by $t_d = 0.15$ (Landgrebe *et al.*, 2006).

Such rejection options define the first step towards an adaptation of the model complexity tailored to data regions with a high degree of uncertainty (Fischer *et al.*, 2014). The first alternative to think about is usually rejection based on deterministic certainty measures. Many of them are based on geometric quantities such as the distance to the decision border for classification tasks, or to the boundary of clusters and its subspaces for clustering tasks. So far, the great part of models that use rejection options deal with just a single threshold, as well as most of them can handle only binary classification, particularly with global thresholds and limits. However, extensions to more general settings like multi-label classification, multiple classes, multithresholding and local thresholding techniques have been considered (Fischer *et al.*, 2016; Jiang & Mojon, 2003). Further, there are also a few adaptive local thresholding techniques, such as in Chow & Kaneko (1972), Jiang & Mojon (2003) and Phansalkar *et al.* (2011).

In the literature, some state of the art strategies for rejection option are listed in Fischer *et al.* (2014). On considering both local and global rejection, with the latter being the most common form, they can be divided into three distinct categories Fischer *et al.* (2016): 1) probabilistic approaches; 2) turning deterministic measures into probabilities, and 3) deterministic approaches. In the SOM based context, these decisions are associated with the nodes in the map (i.e., when they must accept an input pattern to be part of its representation pool).

2.4 HIGH-DIMENSIONAL DATA, SUBSPACE AND PROJECTED CLUSTERING

Over the last decades, with the advances in technology, the modern capabilities of automatic data generation and acquisition have produced more challenging datasets with thousands of dimensions. The nature of such high-dimensional datasets requires a more sophisticated clustering approach. A significantly high number of dimensions implies the well-known curse of dimensionality (Köppen, 2000), where traditional similarity measures become meaningless. A common way to overcome such problems of high-dimensional data spaces where several features are correlated, or only some features are relevant is to perform a dimensionality reduction technique before performing any other task. It can be divided into feature extraction and feature selection. Feature extraction methods like Principal Component Analysis (PCA) are based on a transformation of the original features, i.e., it maps the original data space to a lower-dimensional one. On the other hand, feature selection methods aim at finding the best subset of the original features by discarding the ones that are not relevant, what is not always attainable because sometimes different features are important to distinct clusters (Pudil & Novovičová, 1998). Unfortunately, such techniques cannot be applied to clustering problems. In this sense, conventional clustering approaches to cope with such imposed problems are subspace clustering, projected clustering, pattern-based clustering, and correlation clustering (Kriegel *et al.*, 2009). The scope of the present work considers subspace and projected clustering.

When dealing with high-dimensional data, the presence of irrelevance features of correlations among subsets of features strongly influences the appearance of clusters in the original data space. Different subsets of features may be relevant for distinct clusters, and different correlations among the features may be relevant for different clusters. This phenomenon is called local feature relevance or local feature correlation (Kriegel *et al.*, 2009).

A very common premise to reduce the infinite search space of all possible subspace is to consider axis-parallel subspaces only. So, due to the exponential search space in certain cases, all algorithms that are limited to finding clusters in axis-parallel subspaces must only take into consideration different factors that usually affect the obtained results, for example, one can assume that a different set of features is relevant for each cluster, where the remaining features are irrelevant (Kriegel *et al.*, 2009). Notice that, in the literature, the terms projected clustering and subspaces clustering usually refer to the problem of finding axis-parallel clusters, but it is not always true (Vidal, 2011). It is also important to pinpoint that the problem of subspace clustering occurs even in low dimensions, but it is intensified as the number of dimensions increases. Figure 3 illustrates a subspace clustering problem. More precisely, Figure 3a displays a simulated dataset with three dimensions, in which there are 12 clusters. Note that, for each cluster, one of the three dimensions has the data points spreading along its whole domain. Thus such dimension is irrelevant for the clustering. Figure 3b is a 2D projection concerning only two dimensions. In this example, in the other two dimensions, the data points present a small variation around a central point, determining the clusters. In such dataset, none of the three dimensions can be removed without losing relevant information for 8 out of 12 clusters. Clusters like these are called subspace clusters (Vidal, 2011; Bassani & Araújo, 2012).

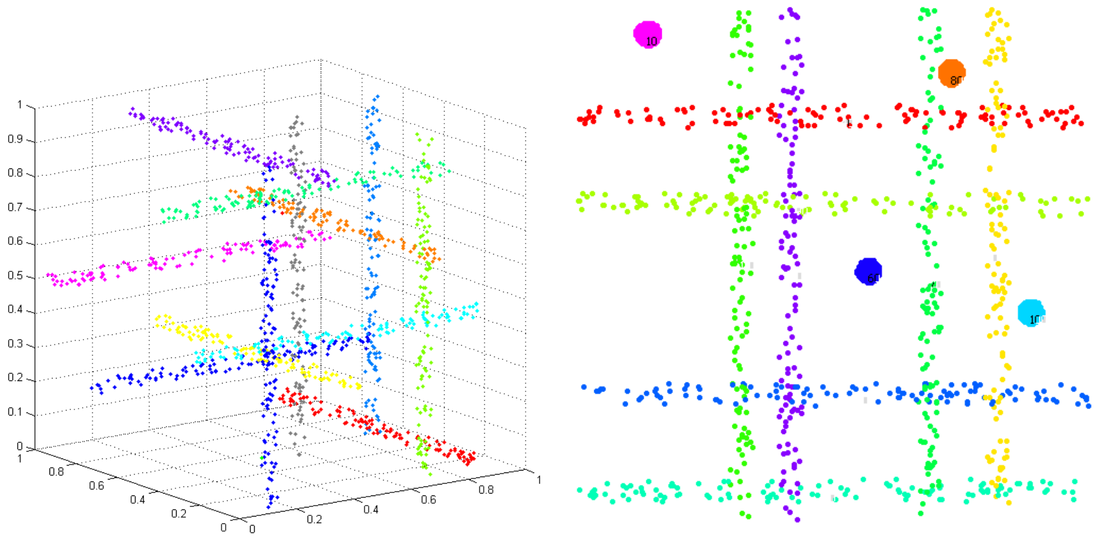

(a) Example of a 3D dataset (b) 2D Projection

Figure 3: Subspace Clustering Problems

According to Kriegel *et al.* (2009), the assumptions made when building such kind of models define its classification. First, there is the algorithmic approach employed to explore the exponential search space of all possible subspaces, which are categorized as algorithmic-oriented models. Depending on what they consider at the beginning, all the dimensions as relevant and then reduce it to match the data, or the contrary, they can be defined as Top-Down or Bottom-Up, respectively.

Second, there is the problem-oriented classification, which can be divided into four different classes of problem statements:

1. Projected Clustering Algorithms: A first class of models that aims to find a unique assignment of each pattern to exactly one subspace cluster. Generally, they try to find the projection to which the currently considered set of patterns is clustered best.

2. Soft Projected Clustering Algorithms: The second class of algorithms that is characterized by the assumption that the number of clusters, k, is known beforehand in a way that an objective function can be defined to derive the optimal set of k clusters. For such a class of algorithms, the subspaces are not assigned in a hard way. Different attributes may be uniquely weighted, but all of them contribute to the clustering.

3. Subspace Clustering Algorithms: A third class of algorithms that are dedicated in finding all clusters in all subspaces.

4. Hybrid Algorithms: The last class of algorithms that aims at finding something in between, usually clusters that may overlap, but not all clusters in all subspaces.

The research developed in the current work mainly focused on Subspace and Projected Clustering. Particularly, adaptive weighting distance functions will be adopted trying to cope with the challenges imposed by high-dimensional data. The basic idea, introduced in Kangas *et al.* (1990), Bassani & Araújo (2012) and considered here, is to apply a weighting factor for each input dimension. Even though it requires other mechanisms to be used combined, it is still a fundamental principle that is enforced in this work. Adaptive Weighting approaches and some models that use it will be discussed further in the next chapter.

2.5 SEMI-SUPERVISED LEARNING

In the past years, there has been a growing interest in a hybrid setting, referred to as Semi-Supervised Learning (SSL). SSL is a combination between supervised and unsupervised learning. In many learning tasks, there is a plentiful supply of unlabeled data, but insufficient labeled ones, since it can be expensive and hard to generate. The basic idea of SSL is to take advantage of both labeled and unlabeled data during the training, combining them to improve the performance of the models (Schwenker & Trentin, 2014; Jain, 2010; Chapelle *et al.*, 2009; Zhu, 2006).

Moreover, SSL can be further classified into semi-supervised classification and semi-supervised clustering (Schwenker & Trentin, 2014). Firstly, in the semi-supervised classification, the training set is given in two parts: $S = \{(\boldsymbol{x}_i, y_i) | \boldsymbol{x}_i \in \mathbb{R}^D, y_i \in Y, 1 \leq i \leq M\}$ and $U = \{\mathbf{u}_i \in \mathbb{R}^D | i = 1, \cdots, M\}$. Where S and U are the labeled and unlabeled data, respectively. At first hand, it is possible to consider a traditional supervised scenario using just S to build a classifier. However, the unsupervised estimation of the probability function $p(\boldsymbol{x})$ of the input set can take advantage of both S and U. Besides, classification tasks can reach a higher performance through the use of SSL as a combination of supervised and unsupervised learning (Schwenker & Trentin, 2014). Many semi-supervised classification algorithms have been developed in the past decades, and, according to Zhu (2006), we can structure them into the following categories: 1) Self-training; 2) SSL with generative models; 3) Semi-supervised Support Vector Machines (S^{3}VM), or Transductive SVM; 4) SSL with Graphs; and 5) SSL with Committees.

Secondly, in the semi-supervised clustering, the aim is to group the data in an unknown number of groups relying on some kind of similarity or distance measures in combination with objective functions. Clustering is a more difficult and challenging problem than classification, and the nature of the data can make the clustering tasks even more difficult, so any kind of additional prior information in respect to the data can be useful to obtain a better performance.

Therefore, the general idea behind semi-supervised clustering is to integrate some type of prior information in the process. For example, a subset of labeled data and further constraints on pairs of the patterns in form of *must-link* and *cannot-link* (Schwenker & Trentin, 2014; Zhu, 2006). Prototype-based models (e.g., k-means, and SOM), Hidden Markov Random Fields (HMRF), Expectation Maximization (EM), Label Propagation, and Label Spreading are examples that have been successful in this area (Schwenker & Trentin, 2014; Zhu, 2006; Basu *et al.*, 2002; Jain, 2010).

For instance, LP based methods operate on proximity graphs or connected structures to spread and propagate information about the class to nearby nodes according to a similarity matrix. It is based on the assumption that nearby entities should belong to the same class, in contrast to far away entities (Xiaojin & Zoubin, 2002; Herrmann & Ultsch, 2007). For LP purposes, each node is assigned to a label vector. A label vector $l_i \in [0,1]^k$ contains the probabilistic membership degrees of input samples to the available cluster. Here, the nodes propagate their label vectors to all adjacent nodes according to a defined distance W. Nodes belonging to a pre-classified input sample have fixed label vectors (Herrmann & Ultsch, 2007). Therefore, a similar alternative to LP is the so-called LS (Zhou *et al.*, 2004). It differs from LP due to modifications the way the similarity matrix is computed. LP uses the raw similarity matrix constructed from the data with no changes, whereas LS minimizes a loss function that has regularization properties allowing it to be often better regarding robustness to noise.

Some recent results, such as in Laine & Aila (2016) and Miyato *et al.* (2018), demonstrated that in certain cases, SSL presented a performance close to purely supervised learning, even when the great part of the labels are discarded. These results are commonly computed by taking an existent classification dataset but only using a small portion of it as labeled data whereas the rest are treated as unlabeled. However, there is not a well-established experimental methodology. Because of that, Oliver *et al.* (2018) tries to define it.

Moreover, it is also worth pointing out that the interest for SSL is growing in the machine learning alongside with the deep learning (LeCun *et al.*, 2015) context, as it is possible to see in Hailat *et al.* (2018), Chen *et al.* (2018), Laine & Aila (2016), Rasmus *et al.* (2015), Kingma *et al.* (2014), Zhou *et al.* (2004), Zhu *et al.* (2003), and Xiaojin & Zoubin (2002). Also, it is not unusual to see the term Deep Semi-Supervised Learning (DSSL) to express deep learning methods applicable to SSL. They range from approaches based on generative models (Kingma *et al.*, 2014) to transfer learning (Oliver *et al.*, 2018) and SOM-based models (Liu *et al.*, 2015; Dozono *et al.*, 2016).

2.6 EVALUATION OF SUBSPACE AND PROJECTED CLUSTERING

There are several ways of evaluating subspace and projected clustering in the literature. Section 2.6.1, Section 2.6.2 and Section 2.6.3 define some metrics, benchmark datasets and parameter sampling techniques that can be used to do such an evaluation.

2.6.1 Clustering Error

According to Patrikainen & Meila (2006) and Müller *et al.* (2009), there are numerous well-known existing metrics for comparing subspace and project clustering methods. However, all criteria for comparing clusterings are based on the so-called confusion matrix.

Let C be a clustering partitioning of the set of m data points into disjoint clusters $C_1, C_2, ..., C_K$ of sizes $m_1, m_2, ..., m_K$, where $\sum_{i=1}^{K} m_i = m$. The confusion matrix $M = \{m_{ij}\}$ is a $K \times K'$ matrix whose *ij*-th element is the number of points in the intersection of the two clusters, i.e., assuming that $C = \{C_1, C_2, ..., C_K\}$ and $C' = \{C'_1, C'_2, ..., C'_{K'}\}$ are two clusterings, the element m_{ij} of M is defined as $m_{ij} = |C_i \cap C'_j|$, and C' normally denotes the ground truth (the expected result), where K is the number of clusters found and K' is the number of cluster in the ground truth (Patrikainen & Meila, 2006).

Based on the previous definition, among several metrics, Clustering Error (CE) is a way of comparing clusterings. It is defined as the proportion of points which are clustered differently in C and C' after an optimal matching of clusters is found. Particularly, for ordinary clustering, it is the scaled sum of the nondiagonal elements of the confusion matrix, minimized over all possible permutations of rows and columns (Patrikainen & Meila, 2006). In this context, CE can be seen as a generalization of the Classification Error (ClaE), often used to evaluate regular clustering results. In fact, as the percentage of relevant dimensions increases, CE converges to the complement of the classification error (1 - ClaE) (Bassani & Araujo, 2015).

For subspace clustering, once a sample can be associated with more than one cluster, the rows and the columns of the confusion matrix M do not necessarily sum up to the number of clusters. Therefore, CE for subspace clustering is defined as in Equation 2.8.

$$CE(C, C') = \frac{|U| - D_{max}}{|U|}, \tag{2.8}$$

where D_{max} is the maximized sum of the diagonal elements of M and $|U|$ is the number of data matrix elements in the union of C and C'.

Therefore the CE measure maps each cluster found to at most one ground truth cluster and also each ground truth cluster to at most one cluster found. It takes into account not only the clusters produced but also the relevant dimensions found for each cluster, and it penalizes clustering results which split up a cluster in several smaller ones (concerning objects or dimensions). CE $\in [0, 1]$ interval, and the higher the value the better the clustering matching (Müller *et al.*, 2009).

2.6.2 Benchmark Datasets

First, the OpenSubspace framework (Müller *et al.*, 2009) provides seven real-world datasets that can be explored. These seven datasets (specified by Table 1) are adapted by Müller *et al.* (2009) from the UCI machine learning repository (Asuncion & Newman, 2007). However, the framework does not include information about their relevant dimensions.

Table 1: Specifications of the Real-World Datasets

Datasets	**Samples**	**Features**	**Classes**
Breast	198	2	33
Diabetes	768	2	8
Glass	214	6	9
Liver	345	2	6
Pendigits	7494	10	16
Shape	160	9	17
Vowel	990	11	10

Second, traditional image benchmark datasets, such as Canadian Institute For Advanced Research (CIFAR10), SVHN, MNIST, and FashionMNIST can be used. They are specified in Table 2.

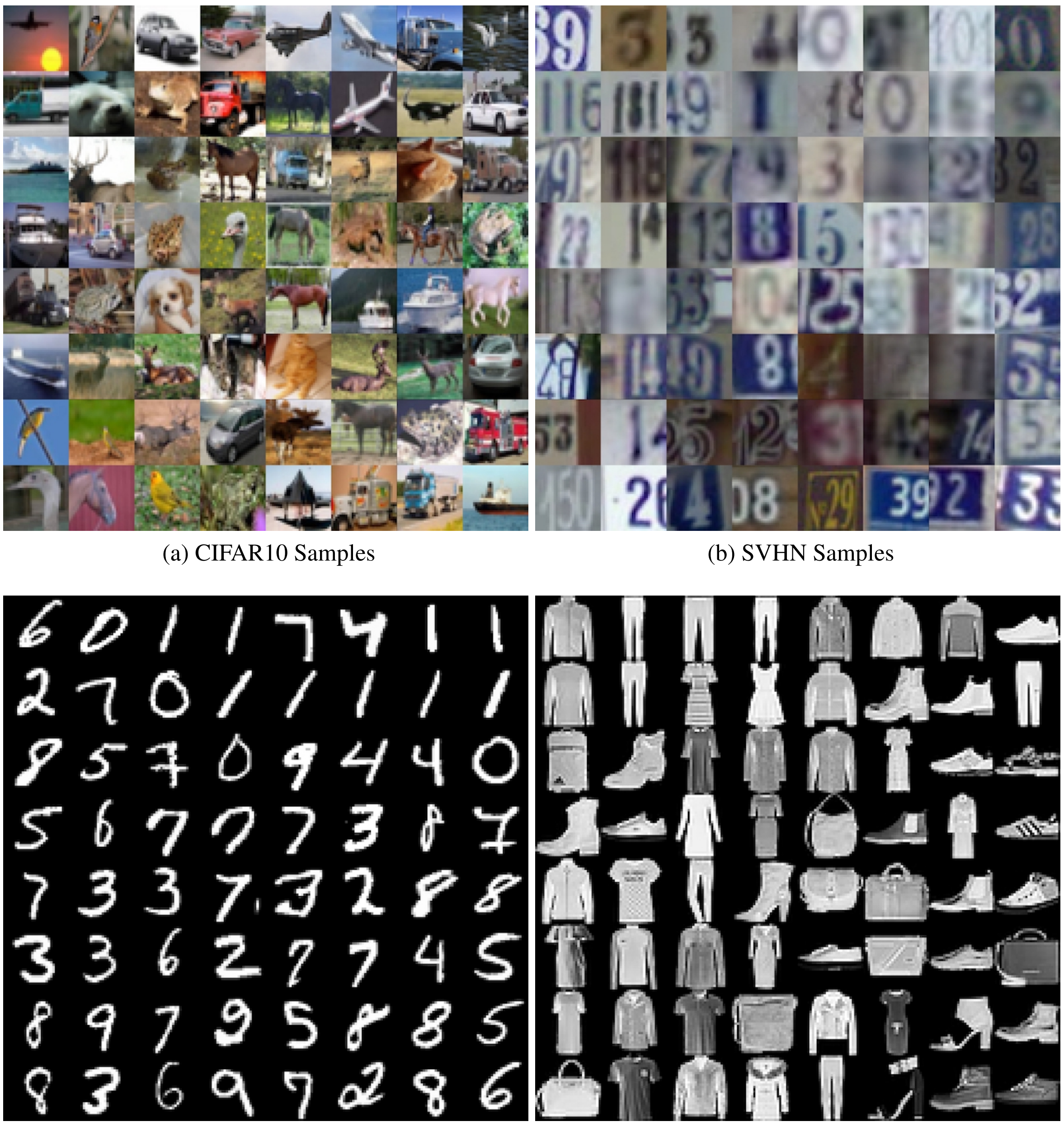
Table 2: Specifications of the Deep Learning Benchmark Image Datasets

Datasets	**Resolution**	**Channels**	**Classes**
CIFAR10	32 x 32	3	10
SVHN	32 x 32	3	10
MNIST	28 x 28	1	10
FashionMNIST	28 x 28	1	10

The CIFAR10 dataset consists of 60000 32x32 color images of 10 different classes, with 6000 images per class. There are 50000 training images and 10000 test images Krizhevsky & Hinton (2009). Figure 4a shows some of its samples. The SVHN is a real-world color image dataset of house numbers obtained from Google Street View images. It is mostly used for developing machine learning and object recognition algorithms with a minimal requirement for data processing and formatting. Also, SVHN has 10 classes, 73257 images for training, 26032 images for testing, and 531131 extra training data, all of them with a 32x32 resolution Netzer *et al.* (2011). Figure 4b illustrates some samples found in the training set.

The MNIST is a widely used deep learning database of handwritten digits. It has a training set of 60000 examples and a test set of 10000 examples. It is a combination of databases from digits of high school students and employees of the United States Census Bureau. The samples have a 28x28 greyscale resolution (LeCun, 1998). Figure 4c shows a grid of its samples. Fashion-MNIST is MNIST-like fashion product database. It shares the same image size and structure of training and testing sets of MNIST dataset (Xiao *et al.*, 2017). Figure 4d shows Fashion-MNIST samples.

Nowadays, the MNIST is considered an easy problem. The SVHN is much harder than MNIST because its images have lack of contrast, normalization and sometimes the digits are overlapped by others, or it has noisy features. Also, Fashion-MNIST is intended to serve as a direct drop-in replacement of the original MNIST dataset for benchmarking machine learning algorithms. All the images shown by Figure 4 are samples from each dataset described in Table 2, and were generated using t-Distributed Stochastic Neighbor Embedding (t-SNE), a dimensionality reduction technique that is particularly well suited for the visualization of high-dimensional datasets (Maaten & Hinton, 2008).



(a) CIFAR10 Samples

(b) SVHN Samples

(c) MNIST Samples

(d) FashionMNIST Samples

Figure 4: Samples from each dataset described in Table 2 generated using t-SNE (Maaten & Hinton, 2008).

2.6.3 Latin Hypercube Sampling - LHS

Usually, subspace clustering methods have several parameters and adjusting them is not an easy task. In this regard, a parameter sampling technique can be applied to find good parameters values and to understand how they affect the results. The Latin Hypercube Sampling (LHS) (McKay *et al.*, 1979; Helton *et al.*, 2005) is a parameter sampling technique that guarantees the full coverage of the range of each parameter. Let $\boldsymbol{x} = \{X_1, ..., X_k\}$ be the values of k input parameters, and S be the sample space of $\boldsymbol{x}$. Now, consider partitioning S into $\mathbf{L}$ disjoint intervals to sample parameter values aiming to represent all areas of the sample space S of $\boldsymbol{x}$. The LHS ensures that all portions of S are sampled and that each of the input parameter X_i has all portions if its distributions represented by input values. To do so, it divides the range of each parameter X_i into N intervals of equal probability $1/N$, resulting in a random selection of a single value from each interval. After that, each sampled component from X_i is matched at random with the other various X_i. LHS ensures that each component is represented in a fully stratified manner, no matter the importance that a component might have (McKay *et al.*, 1979; Helton *et al.*, 2005).

The next chapter will discuss some SOM and LVQ variations that are more suitable at solving the problems of high-dimensional data, subspace and projected clusters that were sketched here.

3 LVQ AND SOM-BASED MODELS

Section 2.1 and Section 2.2 of the previous chapter described the original SOM and LVQ, as proposed by Kohonen (1982, 1990) and Kohonen (1995), respectively. SOM is usually applied to unsupervised clustering tasks, whereas the LVQ is supervised and applied to classification. However, they both are not appropriate to deal with more challenging problems (or datasets), specifically when there is a great number of dimensions (Bassani & Araújo, 2012; Parsons *et al.*, 2004). Nowadays, datasets with thousands of dimensions are not rare and can be easily found in areas such as data mining (Kriegel *et al.*, 2009), bioinformatics (Araujo *et al.*, 2013), computer vision (Vidal, 2011), and more. Such datasets challenge not only the SOM and LVQ, but also other traditional clustering and classification algorithms that consider all the existent dimensions as relevant due to the presence of dimensions that present uncorrelated values for some samples, and therefore are not relevant for all clusters or classes. In high-dimensional data, these dimensions can misguide the algorithms by hiding clusters in noisy data (Bassani & Araújo, 2012). Recap Section 2.4 which stated that as a consequence of the curse of dimensionality, traditional distance metrics often used by traditional clustering and classification methods may become meaningless. The model proposed by Kangas *et al.* (1990) is an example of a SOM that uses a weighting distance metric, allowing it to start dealing with such mentioned problem. On the other hand, Hammer & Villmann (2002) also proposed a model to cope with the same problems, but in the context of LVQ.

Moreover, another important aspect to consider in the original SOM and in some of its variants is the fixed topology. It usually requires a deep understanding of the data, and may not adequately represent clusters that live in different subspaces (Bassani & Araújo, 2012; Bassani & Araujo, 2015). This issue has been addressed by SOM-based models that present a time-varying structure, as in Araujo & Rego (2013); Bassani & Araujo (2015). These maps learn the topology during the training process trying to determine an optimum arrangement at the end. This approach relies on an incremental and robust learning process, where not only the number of nodes but also the connections between them must be learned.

The SOM-based models proposed by Bassani & Araújo (2012) and Bassani & Araujo (2015) will be explained in Section 3.1 and Section 3.2, respectively. Also, the LVQ-based model introduced by Hammer & Villmann (2002) will be described in Section 3.3.

3.1 DIMENSION SELECTIVE SELF-ORGANIZING MAP - DSSOM

The Dimension Selective Self-Organizing Map (DSSOM), proposed by Bassani & Araújo (2012), is a fixed topology SOM with a dimension selective feature able to deal with subspace and projected clustering, that was based on (Kangas *et al.*, 1990). It is capable of adjusting the relevance of each dimension in the distance function differently for each node in the map. In DSSOM, instead of adjusting the scale of input patterns, as in Kangas *et al.* (1990), it uses the set of weighting factors (called relevance vectors by Bassani & Araújo (2012)) to allow that some dimensions do not interfere, or interfere less than others, in the grouping established by a given neuron. The adjustment of the relevance vector of each neuron is made during the training process adaptively. Another interesting feature of DSSOM is directly related with an important characteristic found in subspace clustering problems: an input sample can belong to more than one cluster since each cluster may take into account different subsets of the input dimensions. In the original SOM this is not allowed, however, DSSOM provides the necessary mechanisms to allow more than one winner for each input pattern (Bassani & Araújo, 2012).

Moreover, DSSOM considers the same adaptive weighted Euclidean distance (Equation 3.1) proposed by Kangas *et al.* (1990) to adjust the relevance of each dimension. However, the vector $\boldsymbol{\omega}_j$ does not indicate the scale adjustment but the relevance of each dimension for each node. Note that each element of $\boldsymbol{\omega}_j$ converges to a value between 0 and 1, which is inversely proportional to the variability observed in the respective component of the input patterns clustered in such a node (Bassani & Araújo, 2012).

$$\left[D_\omega(\boldsymbol{x},\boldsymbol{w}_j)\right]^2 = \sum_{i=1}^{m} \omega_{ji}^2 (x_i - w_{ji})^2. \tag{3.1}$$

Basically, in DSSOM, the winner of a competition is the one that presents the highest activation to the input pattern $\boldsymbol{x}$ according to the Equation 3.2.

$$s_1(\boldsymbol{x}) = \arg\max_j [ac(D_\omega(\boldsymbol{x},\boldsymbol{w}_j),\boldsymbol{\omega}_j)]. \tag{3.2}$$

The activation of a node, defined by Equation 3.3, is a function of a weighted distance to the input pattern, and the sum of the components of its relevance vector. Nodes that take into account more dimensions produce higher activations (Bassani & Araújo, 2012).

$$ac(D_\omega(\boldsymbol{x},\boldsymbol{w}_j),\boldsymbol{\omega}_j) = \frac{\sum_{i=1}^{m} \omega_{ji}}{\sum_{i=1}^{m} \omega_{ji} + D_\omega(\boldsymbol{x},\boldsymbol{w}_j) + \varepsilon}, \tag{3.3}$$

where ε is a small number to avoid division by zero.

The weights of the winner and its neighbors are updated as in the original SOM (Equation 2.3). However, in this step, the relevance vector is also updated considering another vector, $\boldsymbol{\delta}_j$, that estimates the average distance of input patterns clustered by node j (Bassani & Araújo, 2012).

$$\boldsymbol{\delta}_j(n+1) = (1-\beta)h_{j,i(\boldsymbol{x})}(n)\boldsymbol{\delta}_j(n) + \beta h_{j,i(\boldsymbol{x})}(n)|\boldsymbol{x} - \boldsymbol{w}_j|, \tag{3.4}$$

where $h_{j,i(\boldsymbol{x})}(n)$ is the topological neighborhood function of the traditional SOM (Equation 2.5) and $\beta \in]0,1[$.

After computing the update of the distance vector, each component of the relevance vector is updated according to Equation 3.5

$$\omega_{ji} = \begin{cases} 1 - (\delta_{ji}/\delta_{ji\max}) & \text{if } \delta_{ji\max} > 0 \\ 1 & \text{if } \delta_{ji\max} = 0, \end{cases} \tag{3.5}$$

where $\delta_{ji\max}$ is the highest component of the vector δ_j. It is also possible to impose a lower-bound limit ε_ω for each component to prevent from becoming zero and thus being totally ignored in the distance function. This scheme used by DSSOM has the advantage of adjusting only the parameter β, instead of the three parameters found in Kangas *et al.* (1990).

Still, the self-organization process of DSSOM has a global relevance vector, ρ, to allow more than one node to win a competition for a given input pattern, which is necessary to perform subspace clustering tasks. This vector penalizes the dimensions already considered by previous winners, and it is updated as per Equation 3.6 (Bassani & Araújo, 2012).

$$\rho_i = \rho_i\left(1 - \omega_{ki}\right), \quad i = 0,1,2....,m, \tag{3.6}$$

where ω_{ki} is i-th component of the relevance vector and k is the index of the winner node.

After updating ρ, while there is the chance of more possible winners existing (controlled by a specific criterion), the DSSOM continues looking for another winner by applying the global relevance vector, instead of the relevance vector of each node, in the related equations (Bassani & Araújo, 2012). Also, during clustering, if the activation produced by an input pattern is lower than a defined threshold, this pattern is considered as noise.

3.2 LOCAL ADAPTIVE RECEPTIVE FIELD DIMENSION SELECTIVE SELF-ORGANIZING MAP - LARFDSSOM

Fixed topology maps such as DSSOM are good tools for data visualization. However, in certain kinds of problems, there is a need to add nodes into the map as more data becomes available, improving incremental learning. In addition, modifying neighborhood relationships during training allows the map to fit best the topology presented in the data. Several models of time-varying structure have been proposed in the literature, such as Growing Neural Gas (GNG) (Kunze & Steffens, 1995), Growing When Required (GWR) (Marsland *et al.*, 2002) and Local Adaptive Receptive Field Self-Organizing Map (LARFSOM) (Araujo & Rego, 2013).

The model proposed by Bassani & Araujo (2015), called Local Adaptive Receptive Field Dimension Selective Self-Organizing Map (LARFDSSOM), is a SOM with time-varying structure based on DSSOM and LARFSOM. It takes advantage of the characteristics of both models. As in DSSOM, the nodes can apply different relevances to the input dimensions, and as in LARFSOM, the map only grows when new nodes are necessary, and the receptive field of the nodes is adapted during the self-organization process. The general operation of the map comprises three phases: 1) organization; 2) convergence; and 3) clustering.

In the organization phase, the nodes compete to form clusters of randomly chosen input patterns. The winner of a competition is the most active node according to a radial basis function, with the receptive field adjusted as a function of the local variance of the input patterns. The cooperation step is performed by adjusting the neighbors of the winner node. In LARFDSSOM, the neighborhood is formed by nodes that take into account a similar subset of the input dimensions (Bassani & Araujo, 2015). The competition and cooperations steps are repeated for a limited number of epochs. During this process, the nodes that do not win for a minimum number of patterns are removed from the map.

The convergence phase starts right after the organization. Here, the nodes are adjusted and removed when required, as in the first phase. However, there is no insertion of new nodes into the map. This phase finishes when a defined number of interactions is reached. When the convergence phase finishes, the map is ready to cluster input patterns. As in DSSOM, there is a feature of noise detection, where inputs that result in an activation lower than a threshold for a particular node are considered as noise, and the map does not cluster it. Also, LARFDSSOM can handle both subspace and project clustering.

Similarly to DSSOM, each node j in LARFDSSOM represents a cluster and is associated with three m-dimensional vectors, where m is the number of input dimensions: $\boldsymbol{c}_j = \{c_{ji}, i = 1, \cdots, m\}$ is the center vector that represents the prototype of the cluster j in the input space; $\boldsymbol{\omega}_j = \{\omega_{ji}, i = 1, \cdots, m\}$ is the relevance vector in which each component represents the estimated relevance, a weighting factor within [0, 1], that the node j applies for the i-th input dimension; and $\boldsymbol{\delta}_j = \{\delta_{ji}, i = 1, \cdots, m\}$ is the distance vector that stores a moving average of the observed distance between the input patterns $\boldsymbol{x}$ and the center vector $|\boldsymbol{x} - \boldsymbol{c}_j(n)|$. The $\boldsymbol{\delta}$ vector is used solely to compute the relevance vector, as in Bassani & Araújo (2012).

The winner node in LARFDSSOM, as said before, is the node that presents the highest activation value in response to the input pattern, according to Equation 3.7.

$$s_1(\boldsymbol{x}) = \arg\max_j [ac(D_\omega(\boldsymbol{x}, \boldsymbol{c}_j), \boldsymbol{\omega}_j)]. \tag{3.7}$$

The activation function $ac(D_\omega(\boldsymbol{x}, \boldsymbol{c}_j)$ (Equation 3.8) of a node is calculated as a radial basis function of the weighted distance $D_\omega(\boldsymbol{x}, \boldsymbol{c}_j)$ with the receptive field adjusted as a function of the norm of its relevance vector.

$$ac(D_\omega(\boldsymbol{x},\boldsymbol{c}_j),\boldsymbol{\omega_j}) = \frac{\sum_{i=1}^{m}\omega_{ji}}{\sum_{i=1}^{m}\omega_{ji} + D_\omega(\boldsymbol{x},\boldsymbol{c}_j) + \varepsilon}, \tag{3.8}$$

where ε is a small value to avoid division by zero and $D_\omega(\boldsymbol{x},\boldsymbol{c}_j)$ (Equation 3.9) is a weighted distance equivalent to the one used in DSSOM.

$$D_\omega(\boldsymbol{x},\boldsymbol{c}_j) = \sqrt{\sum_{i=1}^{m}\omega_{ji}(x_i - w_{ji})^2}. \tag{3.9}$$

Analogously to LARFSOM (Araujo & Rego, 2013), the LARFDSSOM has an activation threshold a_t that controls when a new node must be inserted into the map. If the activation computed for the winner is below a_t, a new node is created. Otherwise, the winner and its neighbors are updated by Equations 3.10, 3.11, and 3.12.

Firstly, LARFDSSOM considers two fixed learning rates: $e_b \in]0,1[$ and $e_n \in]0,e_b[$. With the former applied to winners and the latter to neighbors. The Equation 3.10 shows how the prototype vector $\boldsymbol{c}_j$ is adjusted, where e can be replaced by the appropriate constant learning rate (Bassani & Araujo, 2015).

$$\boldsymbol{c}_j(n+1) = \boldsymbol{c}_j(n) + e(\boldsymbol{x} - \boldsymbol{c}_j(n)), \tag{3.10}$$

Secondly, akin to DSSOM, LARFDSSOM computes the relevance vector through the moving average of the observed distance between the input pattern and the current center vector $\boldsymbol{c}_j$, as shown in Equation 3.11.

$$\boldsymbol{\delta}_j(n+1) = (1 - e\beta)\boldsymbol{\delta}_j(n) + e\beta(|\boldsymbol{x} - \boldsymbol{c}_j(n)|), \tag{3.11}$$

where $\beta \in]0,1[$ controls the rate of change of the moving average, e, as in Equation 3.10, can be replaced by the adequate learning rate, and $|\boldsymbol{x} - \boldsymbol{c}_j(n)|$ denotes the absolute value of the components, not the norm.

After updating the distance vector, the adjusting of the relevance vector is calculated according to an inverse logistic function defined by Equation 3.12.

$$\omega_{ji} = \begin{cases} \dfrac{1}{1 + \exp\left(\frac{\delta_{ji\text{mean}} - \delta_{ji}}{s(\delta_{ji\text{max}} - \delta_{ji\text{min}})}\right)} & \text{if } \delta_{ji\text{min}} \neq \delta_{ji\text{max}} \\ 1 & \text{otherwise,} \end{cases} \tag{3.12}$$

where $\delta_{ji\text{max}}$, $\delta_{ji\text{min}}$, $\delta_{ji\text{mean}}$ are the maximum, the minimum, and the mean of the components of the distance vector $\boldsymbol{\delta}_j$, respectively. The parameter $s > 0$ controls the slope of the logistic function (Bassani & Araujo, 2015).

Moreover, the neighborhood of LARFDSSOM can be better defined as per Equation

3.13, where two nodes are connected if they cluster patterns in similar subspaces (Bassani & Araujo, 2015).

$$\text{nodes } i \text{ and } j \text{ are } \begin{cases} \text{connected}, & \left\| \boldsymbol{\omega_i} - \boldsymbol{\omega_j} \right\| < minwd \\ \text{disconnected}, & \text{otherwise}, \end{cases} \tag{3.13}$$

where *minwd* is a connection threshold that defines the required level of similarity between the relevance vectors of two nodes for them to be connected. When $minwd < 0$, no connections are created, when $minwd > 1$, the map is fully connected, and when $minwd = 0.5$, only pairs of nodes with differences up to half the maximum are connected. Bassani & Araujo (2015) define its standard value as 0.5.

Furthermore, each node j in the map stores a variable $wins_j$ that accounts for the number of wins of this node since the last reset. A reset occurs after *age_wins* competitions. This is the moment when the nodes that present a number of wins below the limit $lp \times age\_wins$ are removed from the map, where *lp* is a parameter representing the lowest percentage of wins allowed for a node in the map. For instance, if lp is set to 0.01, a node must win at least 1% of the competitions; otherwise, it will be removed from the map in the next reset. After the removal, the number of wins of the remaining nodes is reset to zero.

In the clustering step of LARFDSSOM, if the activation value produced by a winner node for a particular input pattern is below the threshold a_t, this pattern is considered as an outlier. In summary, the LARFDSSOM improves some qualities of DSSOM by introducing three significant modifications: the time-varying structure, the dynamic of nodes removal and a neighborhood concerning only nodes that share similar subspaces. Also, it is still able to perform subspace and projected clustering tasks (Bassani & Araujo, 2015).

3.3 GENERALIZED RELEVANCE LEARNING VECTOR QUANTIZATION - GRLVQ

Generalized Relevance Learning Vector Quantization (GRLVQ), introduced by Hammer & Villmann (2002), is a pattern classification algorithm that generalizes its precursor, the Generalized Learning Vector Quantization (GLVQ) (Sato & Yamada, 1996), that is based on simple Hebbian learning and leads to worse and unstable results when applied to noisy real-world data. The GRLVQ incorporates to GLVQ an intuitive update rule to measure relevances of the input data dimensions and to allow efficient input pruning (Hammer & Villmann, 2002).

Moreover, the GRLVQ is proposed to be used in high-dimensional real-world datasets as the weighting factors allow the model to approximately determine the intrinsic data dimensionality, i.e., the relevances identify the dimensions irrelevant and/or noise commonly present in such datasets. At training time, the GRLVQ discriminates the influence of different components of the input, increasing or decreasing its relevance (Hammer & Villmann, 2002; De Araujo &

Guimaraes, 2016), so forth attenuating the curse of dimensionality (Köppen, 2000).

The training algorithm adapts the prototypes $\boldsymbol{w}_i$ in such a way that for each class $c \in \{1, ..., C\}$, the corresponding prototypes represent the class as accurately as possible. To do so, the difference of the samples belonging to the c-th class and the receptive field (the same as defined in Equation 2.7) of the corresponding prototypes should be minimized for each class. Thus, given $\boldsymbol{x}_i$ as a m-dimensional training vector, and y_i, its class label, denote by $\mu_\lambda(\boldsymbol{x}_i)$ some function which presents a negative value if the input vector is classified correctly, and a positive value, otherwise. Also, consider $f : \mathbb{R} \to \mathbb{R}$ as a monotonically increasing function. In considering this, the general framework scheme of GRLVQ consists on minimizing the cost function S via a stochastic gradient descent, as expressed in Equation 3.14 (Hammer & Villmann, 2002).

$$S = \sum_{i=1}^{m} f\left(\mu_\lambda(\boldsymbol{x}_i)\right), \tag{3.14}$$

where f is the sigmoidal function leading to a $sgd(\boldsymbol{x}_i) = (1 + exp(-\boldsymbol{x}_i))^{-1}$, and $\mu_\lambda(\boldsymbol{x}_i)$ is given by Equation 3.15 in which $d_\lambda^+(\boldsymbol{x}_i)$ and $d_\lambda^-(\boldsymbol{x}_i)$ are adaptive weighting distances to the next prototype labeled with y_i and the next prototype labeled with a label not equal to y_i, respectively. This yields a particular powerful and noise tolerant behavior since it combines adaptation near the optimum Bayesian borders whereas prohibiting possible divergences (Hammer & Villmann, 2002; De Araujo & Guimaraes, 2016).

$$\mu_\lambda(\boldsymbol{x}_i) = \frac{d_\lambda^+(\boldsymbol{x}_i) - d_\lambda^-(\boldsymbol{x}_i)}{d_\lambda^+(\boldsymbol{x}_i) + d_\lambda^-(\boldsymbol{x}_i)}. \tag{3.15}$$

The goal of adjusting the P prototypes to represent the data accurately, despite the closeness of data vectors belonging to different classes, is done by finding the nearest prototype $\boldsymbol{w}_{r^+}$ from the same class of the training vector, as well as the nearest prototype $\boldsymbol{w}_{r^-}$ from a different class. Then, in the update step, $\boldsymbol{w}_{r^+}$ is adjusted to get closer to the training vector $\boldsymbol{x}_i$, whilst $\boldsymbol{w}_{r^+}$ is pushed away, according to Equation 3.16 (Hammer & Villmann, 2002; Nova & Estévez, 2014).

$$\begin{aligned} \boldsymbol{w}_{r^+}(t+1) = \boldsymbol{w}_{r^+}(t) + 2 \times \Delta\boldsymbol{w}_{r^+}, &\quad \Delta\boldsymbol{w}_{r^+} = \varepsilon^+ \times f'(\mu_\lambda(\boldsymbol{x}_i)) \times \xi^- \times \frac{\partial d_\lambda^+(\boldsymbol{x}_i)}{\partial \boldsymbol{w}_i^+}, \\ \boldsymbol{w}_{r^-}(t+1) = \boldsymbol{w}_{r^-}(t) - 2 \times \Delta\boldsymbol{w}_{r^-}, &\quad \Delta\boldsymbol{w}_{r^-} = \varepsilon^- \times f'(\mu_\lambda(\boldsymbol{x}_i)) \times \xi^+ \times \frac{\partial d_\lambda^-(\boldsymbol{x}_i)}{\partial \boldsymbol{w}_i^-}, \end{aligned} \tag{3.16}$$

where e^+ and e^- are respectively the learning rates to the same and different class nearest prototypes, f' is the $sgd(\boldsymbol{x}_i)$ mentioned before, and ξ^- and ξ^+ are weighted derivative distances as per Equation 3.17.

$$\xi^- = \frac{2 \times d_\lambda^-(\boldsymbol{x}_i)}{\left(d_\lambda^+(\boldsymbol{x}_i) + d_\lambda^-(\boldsymbol{x}_i)\right)^2}, \quad \xi^+ = \frac{2 \times d_\lambda^+(\boldsymbol{x}_i)}{\left(d_\lambda^+(\boldsymbol{x}_i) + d_\lambda^-(\boldsymbol{x}_i)\right)^2}. \tag{3.17}$$

At this point, the GRLVQ introduces the weight vector λ that stores the relevance of each input dimension, a value that ranges from zero (irrelevant) to one (uttermost relevant), hence turning data preprocessing unnecessary. To avoid instabilities for the weighting factors, each component of $\boldsymbol{\lambda}$ is initialized by $\lambda_i = 1/m$, where m is the number of input dimensions, and a normalization to obtain $||\boldsymbol{\lambda}|| = 1$ is added. In each update step, the weight vector $\boldsymbol{\lambda}$ is also updated according to a learning procedure that follows the main update principles for prototype adjustments, i.e., it can be interpreted in a Hebbian way, as defined by Equation 3.18 (Hammer & Villmann, 2002).

$$\boldsymbol{\lambda}(t+1) = \boldsymbol{\lambda}(t) - \Delta\boldsymbol{\lambda}, \quad \Delta\boldsymbol{\lambda} = -\varepsilon^\lambda \times f'(\mu_\lambda(\boldsymbol{x}_i)) \times (\xi^+ \times d_\lambda^+(\boldsymbol{x}_i) - \xi^- \times d_\lambda^-(\boldsymbol{x}_i)), \tag{3.18}$$

where ε^λ is the learning rate for the relevance weights.

After each update, the weight vector $\boldsymbol{\lambda}$ is normalized as mentioned before, and all of the learning rates are linearly decreased according to Equation 3.19. Also, in order to keep $\boldsymbol{\lambda}$ with positive values, negative relevances are replaced by zero. This process is repeated until the model converges or some other stopping criteria is reached. Moreover, the computational cost of GRLVQ depends on the number of prototypes, P, input dimensions, m, and epochs, which is usually a fixed previously defined number (De Araujo & Guimaraes, 2016).

$$\varepsilon(t) = \frac{\varepsilon(0)}{1 + \tau \times (t - t_0)} \tag{3.19}$$

The characteristics and ideas of the models presented in this chapter arouse the interest in studying a hybrid environment where the advantages and concepts of each of them could be explored in combination, considering the supervised and unsupervised contexts, to perform Semi-Supervised Learning tasks. This results in two proposed models that will be discussed further in Chapter 4 and Chapter 5.

4 SEMI-SUPERVISED SELF-ORGANIZING MAP - SS-SOM

Semi-Supervised Self-Organizing Map is a semi-supervised hybrid LVQ-SOM, based on LARFDSSOM (Bassani & Araujo, 2015), with a time-varying structure (Araujo & Rego, 2013) and two different ways of learning. As in LARFDSSOM, it is possible for SS-SOM that the nodes consider different relevances for the input dimensions and adapt its receptive field during the self-organization process. More precisely, SS-SOM can learn in a supervised or unsupervised way. It switches between these two modes during the self-organization process according to the availability of the information about the class label for each input pattern. To achieve this, we modified the LARFDSSOM to include concepts from the standard LVQ (Kohonen, 1995) when the class label of some input pattern is given.

The rest of this Chapter is organized as follows: Section 4.1 presents the general operations of SS-SOM. Section 4.2 introduces the aspects in common for both operating modes of the model. Section 4.3 and Section 4.4 describe in details the unsupervised and supervised learning procedures, respectively. Later on in Section 4.5, the convergence phase is introduced. Moreover, Section 4.6 shows how the clustering and classification processes are conducted in SS-SOM. Section 4.7 presents Batch SS-SOM, an extension of the proposed model, SS-SOM. Finally, Section 4.8 summarizes the parameters of the model, and Section 4.9 brings some preliminary results and conclusions.

4.1 OPERATION OF SS-SOM

Likewise the LARFDSSOM, the operation of SS-SOM consists of three phases: 1) organization (Algorithm 1); 2) convergence (Algorithm 5); and 3) clustering or classification (Algorithm 6).

In the organization phase, after the network initialization, the nodes start to compete to form clusters of randomly chosen input patterns. There are two different forms to decide who is the winner of a competition, which nodes need to be updated and when a new node needs to be inserted. If the class label of the input pattern is provided, the supervised mode (Section 4.4) is employed to adapt the nodes, otherwise, the unsupervised mode (Section 4.3) is used. The model can also be trivially modified to incorporate reinforcement learning, though this is left for future

Algorithm 1: SS-SOM

```
1  Initialize parameters $a_t$, $lp$, $\beta$, $age\_wins$, $e_b$, $e_n$, $s$, $minwd$, $epoch_{max}$, $e_w$, $N_{max}$;
2  Initialize the map with one node with $\boldsymbol{c}_j$ initialized at the first input pattern $\boldsymbol{x}_0$, $\boldsymbol{\omega_j} \leftarrow$
     $\mathbf{1}$, $\boldsymbol{\delta_j} \leftarrow \mathbf{0}$, wins$_j \leftarrow 0$ and class$_j \leftarrow$ noClass or class($\boldsymbol{x}_0$) if available;
3  Initialize the variable nwins $\leftarrow 1$;
4  for epoch $\leftarrow$ 0 to $epoch_{max}$ do
5      Choose a random input pattern $\boldsymbol{x}$;
6      Compute the activation of all nodes (Equation 3.8);
7      Find the winner $s_1$ with the highest activation ($a_s$) (Equation 3.7);
8      if $\boldsymbol{x}$ has a label then
9          Run the SupervisedMode($\boldsymbol{x}$, $s_1$) (Algorithm 4);
10     else
11         Run the UnsupervisedMode($\boldsymbol{x}$, $s_1$) (Algorithm 3);
12     if $nwins$ = $age\_wins$ then
13         Remove nodes with wins$_j < lp \times age\_wins$;
14         Update the connections of the remaining nodes (Equation 4.1);
15         Reset the number of wins of the remaining nodes:
16         wins$_j \leftarrow 0$;
17         $nwins \leftarrow 0$;
18     $nwins \leftarrow nwins + 1$;
19 Run the Convergence Phase (Algorithm 5);
```

work. As in LARFDSSOM, the cooperation is performed by adjusting the neighborhood around the winner node. However, the neighborhood of SS-SOM takes into account not only a similar subset of the input dimensions but also the class labels, connecting only nodes with the same class label or unlabeled nodes. As usual, the adaptation occurs by adjusting the weight vectors of the map. The equations will be presented following in this Chapter. Thereby, the competition, adaptation and cooperation steps are repeated for a limited number of epochs. In the meantime, with given a periodicity controlled by a parameter called *age_wins*, the nodes that do not win for a minimum number of patterns are removed from the map, as in LARFDSSOM.

The convergence phase starts after the organization phase. Here, the nodes are also updated and removed when necessary, similarly to the way conducted in the first phase. The difference is the fact that there is no insertion of new nodes. Moreover, this phase finishes the node removal cycle left by the organization phase and runs an additional cycle to ensure convergence. This phase in SS-SOM is pretty similar to the convergence phase of LARFDSSOM.

After finishing the convergence phase, the map can cluster and classify input patterns. Depending on the amount and distribution of labeled input patterns presented to the network during the training, after the convergence phase the map may have:

1. All the nodes labeled;

2. Some nodes labeled;

3. No nodes labeled.

For the first case, the clustering and classification are straightforward: each test pattern is associated with the label of the node with the highest activation. For the second case, if the node with the highest activation has no class, the SS-SOM continues looking for another node with a defined class label, and an activation above the threshold a_t. For the third and last case, we can identify the clusters of the input test patterns, but not their classes.

It is important to mention that in this work, only the task of projected clustering is investigated, i.e., each input pattern is assigned to a single cluster, though the proposed model supports subspace clustering as well.

The next sections describe all the processes involving both unsupervised and supervised operation modes, as well as how clustering and classification are conducted. After that, the parameters analysis and tuning are discussed.

4.2 ASPECTS IN COMMON FOR BOTH MODES

Indeed, to be consistent, each node in the map must have the same structure independently of the operating mode. Also, both supervised and unsupervised learning procedure share some aspects concerning the operations that are performed during the training time. All of those aspects will be discussed in the next Sections.

4.2.1 Structure of the Nodes

Exactly as in LARFDSSOM, each node j in SS-SOM represents a cluster and is associated with three m-dimensional vectors. They are: $\boldsymbol{c}_j = \{c_{ji}, i = 1, \cdots, m\}$, the center vector that represents the prototype of the cluster j in the input space; $\boldsymbol{\omega}_j = \{\omega_{ji}, i = 1, \cdots, m\}$, the relevance vector in which each component represents the estimated relevance, a weighting factor within [0, 1], that the node j applies for the i-th input dimension; and $\boldsymbol{\delta}_j = \{\delta_{ji}, i = 1, \cdots, m\}$, the distance vector that stores the moving average of the observed distance between the input patterns $\boldsymbol{x}$ and the center vector. Moreover, $\boldsymbol{\delta}$ is used exclusively to compute the relevance vector.

4.2.2 Activation of the Nodes

The activation of a node in SS-SOM is calculated as in LARFDSSOM, with a radial basis function of the weighted distance $D_\omega(\boldsymbol{x}, \boldsymbol{c}_j)$ (Equation 3.9) that has its receptive field adjusted as a function of its relevance vector. The activation grows as the distance decreases and as the relevances increases. Equation 3.8 shows the activation function.

4.2.3 Node Update

In SS-SOM, in order to update the vectors associated with the nodes (the winner, the neighbors or the winner of a wrong class), a fixed learning rate is used, depending on the undergoing procedure (Algorithm 4 or Algorithm 3). Algorithm 2 shows how the update occurs in SS-SOM. The nodes will be updated with the same equations ofLARFDSSOM given a learning rate.

Algorithm 2: Node Update of of SS-SOM

```
   Input: Node $s$, Learning Rate $lr$
1  Function UpdateNode($s$, $lr$):
2      Update the distance vector $\boldsymbol{\delta}_s$ according to $lr$ (Equation 3.11);
3      Update the relevance vector $\boldsymbol{\omega}_s$ (Equation 3.12);
4      Update the weight vector $\boldsymbol{c}_s$ according to $lr$ (Equation 3.10);
```

4.2.4 Node Removal

The parameter *age_wins* controls the periodicity of nodes removal. Whenever *age_wins* is reached, a reset occurs (lines 12-17 in Algorithm 1), it means that any nodes which do not win at least the minimum percentage of the competitions $lp \times age\_wins$ will be removed. To manage this, each node j in SS-SOM stores a variable $wins_j$ that represents the number of the node victories since the last reset. After each reset, the number of victories of the remaining nodes is set to zero.

4.2.5 Neighborhood Update

In SS-SOM, the neighborhood is formed by nodes with the same class or nearby unlabeled nodes that apply similar relevances for the input dimensions. So that, a connection between two nodes means they cluster patterns with the same class or at least in similar subspaces. Equation 4.1 considers these similarities between the relevances and classes of every pair of nodes to control this behavior.

$$\text{nodes } i \text{ and } j \text{ are} \begin{cases} \text{connected,} & \text{if (} class(i) = class(j) \text{ or} \\ & class(i) = noClass \text{ or} \\ & class(j) = noClass \text{)} \\ & \text{ and } \left\| \boldsymbol{\omega_i} - \boldsymbol{\omega_j} \right\| < minwd \\ \text{disconnected,} & \text{otherwise} \end{cases} \tag{4.1}$$

4.3 UNSUPERVISED MODE

Given an unlabeled input pattern, we look for a winner node disregarding their class labels. Therefore, as in LARFDSSOM (Equation 3.7), the winner of a competition is the node that is the most activated according to a radial basis function with the receptive field adjusted as a function of its relevance vector.

Similarly to LARFDSSOM, SS-SOM has an activation threshold a_t. According to this, if the activation of the winner is lower than a_t, a new node is inserted into the map at the position of the input pattern since the existing winner is not close enough. Otherwise, the winner and its neighbors are updated moving them closer to the input pattern (Section 4.2.3). Thereby, as in LARFDSSOM, two fixed learning rates are considered : 1) $e_b \in]0,1[$ for the winner node; and 2) $e_n \in]0,e_b[$ for its neighbors. Algorithm 3 presents this procedure.

Algorithm 3: Unsupervised Mode of SS-SOM

```
Input: Input pattern $\boldsymbol{x}$ and the first winner $s_1$;
1 if $a_{s_1} < a_t$ and $N < N_{max}$ then
2     Create a new node $j$ and set: $\boldsymbol{c}_j \leftarrow \boldsymbol{x}$, $\boldsymbol{\omega}_j \leftarrow \mathbf{1}$, $\boldsymbol{\delta}_j \leftarrow \mathbf{0}$, wins$_j \leftarrow 0$ and class$_j \leftarrow$ $noClass$;
3     Connect j to the other nodes as per Equation 4.1;
4 else
5     Update the winner node and its neighbors: UpdateNode($s_1$, $e_b$), UpdateNode($neighbors(s_1)$, $e_n$) (Algorithm 2);
6     Set wins$_{s_1} \leftarrow$ wins$_{s_1}$ + 1;
```

4.4 SUPERVISED MODE

In order to incorporate the supervised learning mode, each node in the map can be associated with a class label. Hence, when a labeled input pattern is given, we treat it differently. Algorithm 4 presents this procedure.

To improve the performance by exploring the information provided by the labeled patterns, SS-SOM takes the labels into account when looking for a winner, unlike the unsupervised mode that only considers the activation. If the most activated node s_1 has the same class of the input pattern or an undefined class (line 1 in Algorithm 4), a very similar procedure to the unsupervised mode (Section 4.3) is run (lines 2 to 9). The difference, in this case, is the fact that the class of s_1 is set as the same as the input pattern $\boldsymbol{x}$, as well as update its connections. Otherwise, if the winner node and the input pattern have different classes, SS-SOM continues trying to find another winner matching the following criteria (line 11): 1) has the same class of the input pattern or an undefined class, and 2) an activation higher than a_t. This procedure simulates an inhibition of the nodes with different classes.

Algorithm 4: Supervised Mode of SS-SOM

```
Input: Input pattern $\boldsymbol{x}$ and the first winner $s_1$;
1  if $class_{s_1}$ = $class(\boldsymbol{x})$ or $class_{s_1}$ = $noClass$ then
2      if $a_{s_1} < a_t$ and $N < N_{max}$ then
3          Create new node $j$ and set: $\boldsymbol{c}_j \leftarrow \boldsymbol{x}$, $\boldsymbol{\omega}_j \leftarrow \boldsymbol{1}$, $\boldsymbol{\delta}_j \leftarrow \boldsymbol{0}$, wins$_j \leftarrow 0$ and class$_j$
            $\leftarrow class(\boldsymbol{x})$;
4          Connect j to the other nodes as per Equation 4.1;
5      else if $a_{s_1} \geq a_t$ then
6          Update the winner node and its neighbors: UpdateNode($s_1$, $e_b$),
            UpdateNode($neighbors(s_1)$, $e_n$) (Algorithm 2);
7          Set class$_{s_1} \leftarrow class(\boldsymbol{x})$;
8          Update $s_1$ connections as per Equation 4.1;
9          Set wins$_{s_1} \leftarrow$ wins$_{s_1}$ + 1;
10 else
11     Try to find a new winner $s_2$ with $noClass$ or the same class of $\boldsymbol{x}$, and an
        activation $a_{s_2} \geq a_t$;
12     if $s_2$ exists then
13         Update the new winner node, its neighbors and the previous wrong winner:
            UpdateNode($s_2$, $e_b$), UpdateNode($neighbors(s_2)$, $e_n$) and UpdateNode($s_1$,
            $-e_w$) (Algorithm 2);
14         Set wins$_{s_2} \leftarrow$ wins$_{s_2}$ + 1;
15     else if $N < N_{max}$ then
16         Create new node $j$ and set: $\boldsymbol{c}_j \leftarrow \boldsymbol{x}$, $\boldsymbol{\omega}_j \leftarrow \boldsymbol{1}$, $\boldsymbol{\delta}_j \leftarrow \boldsymbol{0}$, wins$_j \leftarrow 0$ and class$_j$
            $\leftarrow class(\boldsymbol{x})$;
17         Connect $j$ to other nodes as per Equation 4.1;
```

If any node fulfills these conditions (line 12 in Algorithm 4), a new winner s_2 is said to be found. Then, s_2 and its neighbors will be updated as in the unsupervised mode (Section 4.3). However, the fact that s_1 was considered a "wrong" winner indicates the need to push it away from this input pattern. Therefore, similarly as in the LVQ, we push s_1 away from the input pattern with a fixed learning rate of $-e_w$. This procedure is presented in lines 13 and 14 of Algorithm 4. Otherwise, if there is no new winner and the maximum number of nodes in the map has not been reached, a new node is inserted into the map at the same position and with the same class of the input pattern $\boldsymbol{x}$ (lines 16 and 17 of Algorithm 4).

4.5 CONVERGENCE

The organization phase does not guarantee that the remaining nodes in the map are well positioned and connected as expected, according to the distribution, classes, and relations of the input data. Therefore, they may still need to be updated, for instance, to represent the input patterns previously clustered by removed nodes. In the convergence phase, the self-organization process continues, but the map is not allowed to create new nodes. It can be viewed as a reorganization process. Also, during the self-organization, the model may not reach the *age_wins* value, and hence the nodes may not achieve the lowest percentage of wins so that they also have to be removed. So, the reorganization finishes this cycle. After this process, the model runs another cycle of *age_wins* to ensure convergence with the remaining nodes. The Algorithm 5 describes in details how the convergence phase is carried on.

Algorithm 5: Convergence Phase of SS-SOM

```
1  $N_{max} \leftarrow N$;
2  while $nwins < age\_wins$ do
3      Choose a random input pattern $\boldsymbol{x}$;
4      Compute the activation of all nodes (Equation 3.8);
5      Find the winner $s_1$ with the highest activation ($a_s$) (Equation 3.7);
6      if $\boldsymbol{x}$ has a label then
7          Run the SupervisedMode($\boldsymbol{x}$, $s_1$) (Algorithm 4);
8      else
9          Run the UnsupervisedMode($\boldsymbol{x}$, $s_1$) (Algorithm 3);
10     if $nwins = age\_wins$ then
11         Remove nodes with $wins_j < lp \times age\_wins$;
12         Update the connections of the remaining nodes as per Equation 4.1;
13         Reset the number of wins of the remaining nodes:
14         $wins_j \leftarrow 0$;
15         $nwins \leftarrow 0$;
16     $nwins \leftarrow nwins + 1$;
17 nwins $\leftarrow 0$;
```

In the algorithm, N_{max} is set to the current number of nodes in the map (line 1). Then, the self-organization continues as usual. Finally, as said before, after *age_wins* is reached, a reset occurs in combination with a node removal and a reset of node wins. Lastly, the algorithm is rerun.

4.6 CLUSTERING AND CLASSIFICATION WITH SS-SOM

After the organization and convergence phases of SS-SOM, the center vectors $\boldsymbol{c}_j$ and the relevance vectors $\boldsymbol{\omega}_j$ of each node in the map can be used for clustering and classification of the test patterns. Algorithm 6 presents this procedure.

Each node in the map is associated with an index representing a cluster. The purpose of SS-SOM is to work only with projected clustering problems at first hand. So that, the test patterns are associated only to one cluster each. It means that we need to try to find the correct winner for each test pattern. To achieve this, as in Algorithm 4, we use the most active node as the winner, but if it has no defined class, SS-SOM continues trying to find another winner candidate with a defined class and an activation above a_t. If it exists, we use it to cluster and its label to classify the test pattern. Otherwise, the most active node is used only for clustering the input test pattern, and the model will not be able to classify it. All in all, if none of the nodes produces an activation equal to or above the threshold a_t for a particular pattern, it is considered as an outlier or noise.

Algorithm 6: Clustering and Classification with SS-SOM

```
1  foreach test input pattern x do
2      Present x to the map;
3      Compute the activation of all nodes (Equation 3.8);
4      Find the first winner $s_1$ with the highest activation ($a_s$) (Equation 3.7);
5      if $a_s \geq a_t$ then
6          $s_f \leftarrow s_1$;
7          if $class(s_f) = noClass$ then
8              Try to find a new winner $s_2$ with a defined class and an activation above
                $a_t$;
9              if $s_2$ exists then
10                 $s_f \leftarrow s_2$;
11         Assign x to the cluster of the winner node $s_f$;
12         Set the predicted class of x to be the same class of the winner node $s_f$ (it
            could be defined or undefined);
13     else
14         Assign x to the outliers set;
```

4.7 BATCH SS-SOM

According to LeCun *et al.* (2015), it is expected that unsupervised and semi-supervised learning become far more important in the longer term. The successes of purely supervised learning are overshadowing them. So that, at the current stage of research, it would be of great importance to make possible an approach capable of making use of the advances that came with the rising of Deep Learning and also stimulate the development of semi-supervised approaches in a deep learning context.

To make it possible for SS-SOM to take advantage of the parallelism of a Graphics Processing Units (GPU) in its training, the Batch SS-SOM was developed. It was implemented in the PyTorch framework, and introduces essential modifications to the original SS-SOM model. Such changes allowed SS-SOM to use mini-batch training and to be more integrated with other Deep Learning approaches, that commonly use the same framework and structure.

First, when a mini-batch is given to the model, it is separated into two different mini-batches: 1) the unsupervised mini-batch; and 2) the supervised mini-batch, as shown by the first two columns of Figure 5. For the first case, the learning process continues straightforwardly to the unsupervised mini-batch learning procedure that will be discussed further in this current section. On the other hand, the latter case results in three distinct situations, illustrated by the last column of Figure 5, that must be handled differently after finding the winner node for each sample contained in the supervised mini-batch (as discussed in Section 4.4):

1. **Figure 5-A:** A node with a not defined class is the winner for a labeled sample;
2. **Figure 5-B:** A node with a defined class is the winner for samples of the same class;
3. **Figure 5-C:** A node with a defined class is the winner for samples of different classes, including or not its class

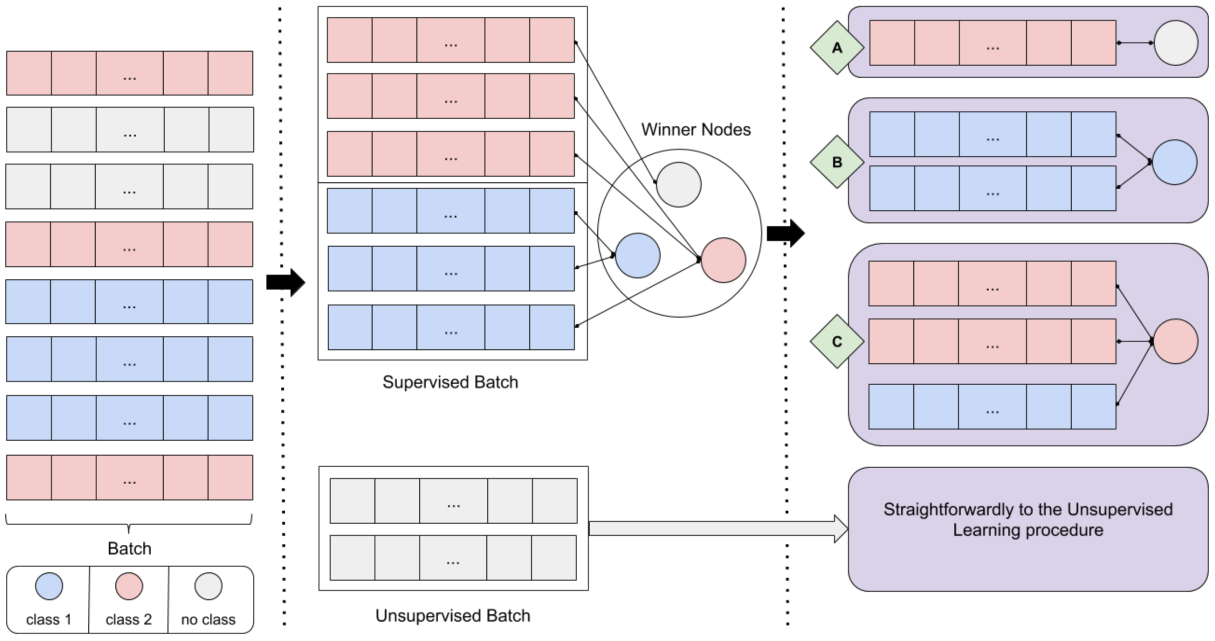


Figure 5: The basic operation performed by Batch SS-SOM when a mini-batch is given, and its resulting cases.

Figure 6 shows how each of these mentioned situations are handled. First, in the Figure 6-A workflow, the actions are to set the node class to be the same as the input pattern and then update it towards such input just as in Section 4.4. Second, in Figure 6-B, it is first necessary to compute the average vector considering all the samples that are under this situation. To do so, the average vector $\boldsymbol{\theta}$ is calculated according to Equation 4.2. Here, recap the unsupervised batch from Figure 5, that is meant to be sent straightforwardly to the unsupervised learning procedure. It is done by sending the unlabeled average vector $\boldsymbol{\theta}$. So in the end, what is sent is a one-dimensional vector.

$$\boldsymbol{\theta} = \frac{1}{S}\sum_{k=1}^{S} \boldsymbol{x}_k, \tag{4.2}$$

where S is the number of samples $\boldsymbol{x}$ from case B. The resulting vector $\boldsymbol{\theta}$ is then sent to the usual supervised update procedure of SS-SOM, where the target label is the common class of the samples (Section 4.4).

Third, the case ilustrated in Figure 6-C is handled as follows: for all the classes contained in this subset of samples, every different class duplicates the original winner node j by preserving the centroid vector $\boldsymbol{c}_j$, the distance vector $\boldsymbol{\delta}_j$ and the relevance vector $\boldsymbol{\omega}_j$, but setting the class of the new duplicated node to be the same as the current treated class, as well as setting its number of victories to zero. After that, for each class l found in the current subset, a vector $\boldsymbol{\theta}_l$ is calculated and the respective duplicated node is updated using both $\boldsymbol{\theta}_l$ and l, suchlike in Figure 6-B, whereas the original winner node is updated using its corresponding $\boldsymbol{\theta}$ and class.

Still, notice that when this situation occurs for an unlabeled winner node j, the first calculated $\boldsymbol{\theta}$ vector and its related class are used to update j as in Figure 6-A. Finally, all the operations executed in the Batch SS-SOM are performed in parallel on the GPU, which optimizes the computational cost and allows the model to be applied to more complex tasks and datasets.

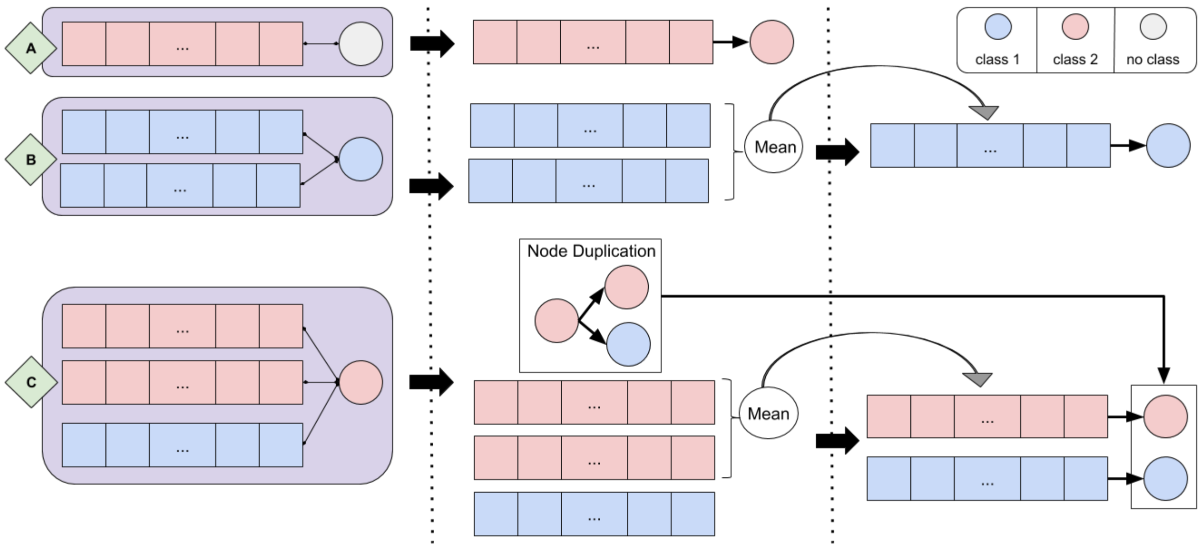


Figure 6: How to handle each distinct situation from the Batch SS-SOM operation when a mini-batch is given.

4.8 A SUMMARY OF THE PARAMETERS

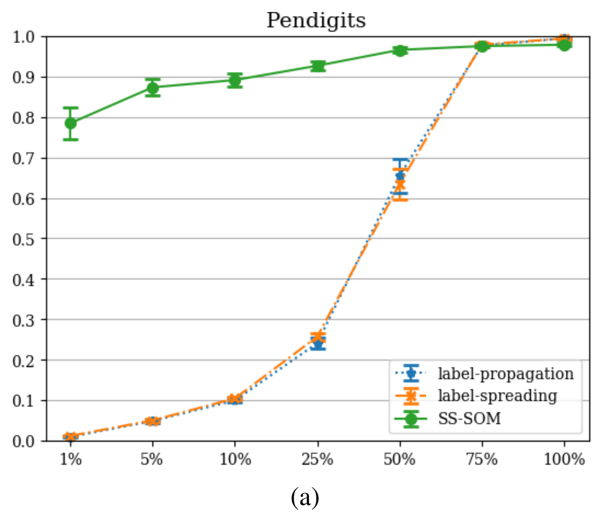
SS-SOM inherits all parameters from LARFDSSOM and includes a new parameter called e_w. This parameter provides a specific learning rate for the update of wrong winners, as described in Section 4.4. It means that SS-SOM has 11 parameters to be set up. Despite this being a high number of parameters, a sensitivity analysis shown in Bassani & Araujo (2015) revealed that only a_t and lp present a high impact on the results of LARFDSSOM, and the SS-SOM kept this characteristic. Therefore, we can keep the other parameters values fixed inside defined ranges, given their marginal influences, including the number of epochs. The parameter a_t is crucial since it defines the receptive field of the nodes. During the training, it affects the number of nodes inserted in the map. During the clustering and classifying procedure, it can be seen as an outlier threshold that controls a noise filtering mechanism. The parameter lp defines the minimum percentage of input patterns that a node has to cluster for not being removed from the map in a certain epoch. This parameter is dataset dependent and has a substantial impact on the results. The other parameters will be discussed further in the next sections.

4.9 PRELIMINARY EXPERIMENTAL RESULTS AND CONCLUSION

This chapter presented the SS-SOM, an approach for classification and clustering with SSL. This section presents the preliminary results of SS-SOM in order to establish a chronological order of the research, once another model was developed to address some aspects of SS-SOM aiming to enhance its capabilities. More details of the experiments done with SS-SOM will be given in Chapter 6. For now, the preliminary experiments compare the classification performance of SS-SOM against the following semi-supervised models: Label Spreading (Zhou *et al.*, 2004) and Label Propagation (Xiaojin & Zoubin, 2002). They were chosen due to their popularity and connection to several other important algorithms, such as Markov random walks algorithms (Szummer & Jaakkola, 2002) and mean field approximation (Peterson, 1987; Jordan *et al.*, 1999).

Two real-world datasets (see Section 2.6.2) and a 3-times 3-fold cross-validation were used, while training and testing the model 500 times for each fold with different parameter values sampled from the parameter ranges presented in Table 3 and Table 4, according to a LHS (McKay *et al.*, 1979; Helton *et al.*, 2005). Also, for studying the effects of different levels of supervision, the models were trained with the following percentages of label availability: 1%, 5%, 10%, 25%, 50%, 75% and 100%. The best accuracy achieved by each method in each fold was recorded for each dataset varying the supervision percentage.

The results presented in Figure 7 illustrate the robustness of SS-SOM, even in situations in which only a small number of labeled data is available. Its performance is superior to other models except when 100% of the labels are given, but it still achieves comparable results.



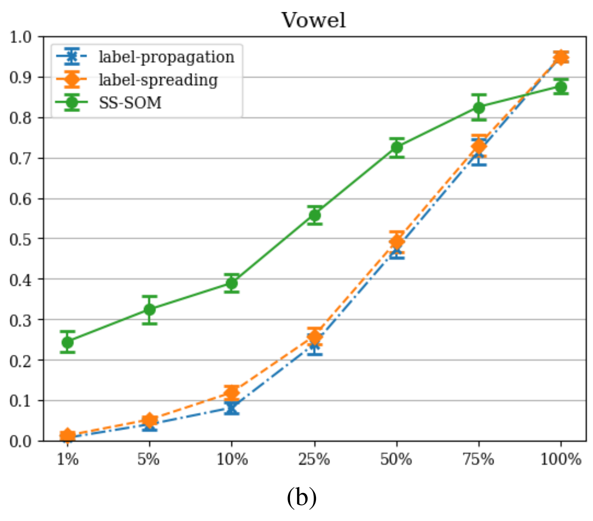


Figure 7: Best mean accuracy and standard deviation as function of the percentage of supervision on (a) Pendigits and (b) Vowel datasets

As said in Section 4.8, SS-SOM has 11 parameters, but only two of them present a high impact on the results. In Figure 8, the a_t was chosen to illustrate this behavior. It is clear that it is a crucial parameter. On the other hand, Figure 9 shows the results as a function of the e_b parameter, in which the results suffer marginal influences in comparison with the behavior of a_t. The blue dots in the figures indicates the accuracy obtained with SS-SOM as a function of the given parameter value, whereas the red lines are the linear fits to the data, which is a common and effective way of studying the sensitivity of parameters, as Iman & Helton (1988) demonstrates. It is possible to identify trends in parameter values as a function of the obtained metric values using this technique, as it is observed with the parameter a_t. Note that such a strong dependency of parameter presents difficulties for the applicability of the model, since it loses generalization power, and ends up becoming dependent on the data which it was tuned for.

Moreover, SS-SOM also has a low sample efficiency (Wang *et al.*, 2016). Sample efficiency denotes the amount of experience that an algorithm needs to learn. The aim of having a sample efficient method is to reduce the number of simulation steps (i.e., samples of the model or environment). Thus, sample efficient methods are methods which learn faster by only using a small number of samples or the important ones. Since the framework of SS-SOM ignores several samples in cases where the map is full, and the winning node is not close enough from an input pattern, the sample efficiency is reduced because a larger number of samples became necessary for the model to learn. The same behavior is found in LARFDSSOM. Those are the reasons behind the development of the next proposed model that will be shown in Chapter 5.

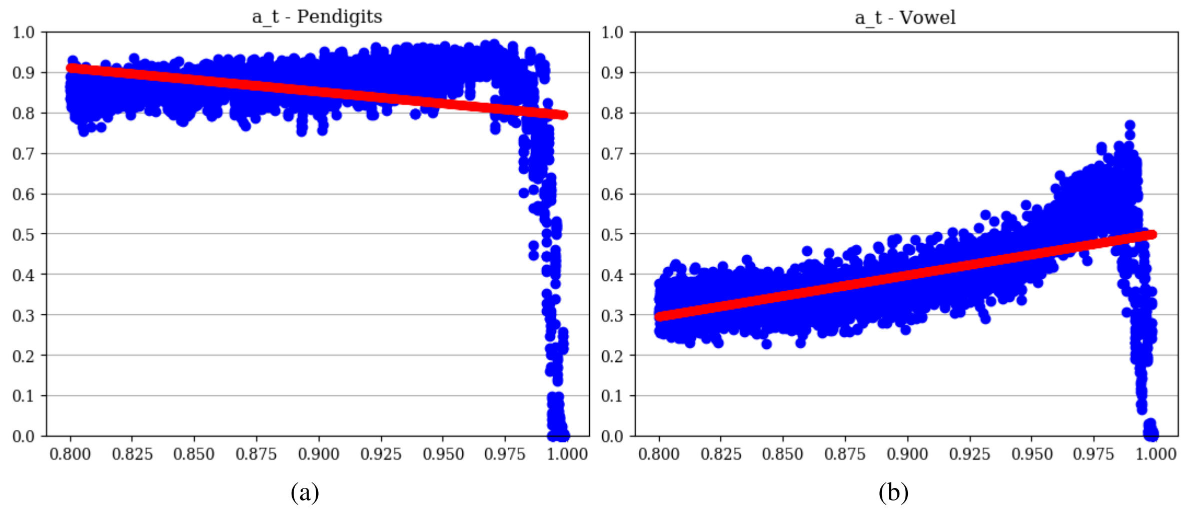
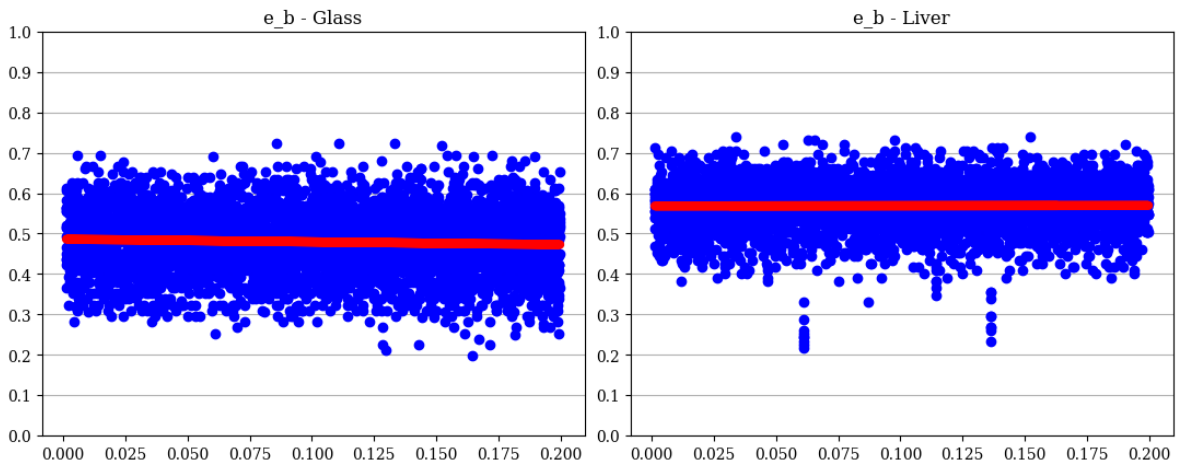


Figure 8: Preliminar sensitivity analysis with a scatter plot of the Accuracy obtained with SS-SOM as a function of parameter a_t on (a) Pendigits and (b) Vowel datasets



(a) (b)

Figure 9: Preliminar sensitivity analysis with a scatter plot of the Accuracy obtained with SS-SOM as a function of parameter e_b on (a) Glass and (b) Liver datasets

5 ADAPTIVE LOCAL THRESHOLDS SEMI-SUPERVISED SELF-ORGANIZING MAP - ALTSS-SOM

Adaptive Local Thresholds Semi-Supervised Self-Organizing Map is a SOM with Adaptive Local Thresholds (Jiang & Mojon, 2003) based on SS-SOM. Hence, being based on SS-SOM, ALTSS-SOM can also learn in a supervised or unsupervised way depending on the availability of labels, and maintains the general characteristics of its predecessors. However, it introduces new behaviors on both sides, supervised and unsupervised, to allow a better usage, and consequently a better understanding of the data statistics. By doing this, ALTSS-SOM aims at overcoming the problems presented by SS-SOM, such as the high sensitivity to the parameters, and the low sample efficiency. Additionally, with the proper changes, ALTSS-SOM targets achieving better results for both classification and clustering tasks.

Therefore, the parameterized activation threshold (a_t) used in both previous methods is replaced by an adaptive thresholding technique that takes into account the local variance to provide the model the ability to learn the receptive field of each node. The objective is to estimate optimal local regions in the space with respect to the distribuition of the input patterns $\boldsymbol{x}$ for each node in the map. To do so, inspired by the Adam algorithm, a method for efficient stochastic optimization that only requires first-order gradients with little memory requirement (Kingma & Ba, 2014), ALTSS-SOM updates exponential moving averages of the distances between each input pattern $\boldsymbol{x}$ and the centroid of the nodes for each dimension (the vector $\boldsymbol{\delta}_j$ in the algorithms). In SS-SOM and LARFDSSOM, this estimate was done by using not only β but also the learning rate e in equation (Equation 3.11). However, ALTSS-SOM modified this approach to use solely the parameter $\beta \in [0,1)$ for controling the exponential decay rate of the moving averages.

The moving averages themselves are estimates of the first moment (the mean) of the distances between the input patterns and the centroids of the nodes. Because of that, such means can be used as estimates of the uncentered variance of the nodes in each dimension. However, these moving averages are initialized as vectors of zeros, leading to moment estimates that are biased towards zero, especially during the initial steps, and when the decay rate is small (close to 1) (Kingma & Ba, 2014). Still, according to Kingma & Ba (2014), this initialization bias can be counteracted, resulting in a bias-corrected estimate $\hat{\boldsymbol{\delta}}_j$. During the learning process, this bias-corrected estimate $\hat{\boldsymbol{\delta}}_j$, together with the relevance vector $\boldsymbol{\omega}_j$ can be used as reject options (Chow, 1970), determining whether or not an input pattern is in the receptive field of a winner node.

The rest of this Chapter is organized as follows: Section 5.1 presents the operations of ALTSS-SOM. Section 5.2 introduces the structure of the nodes in the model. Section 5.3 briefly discusses the way the competition step is carried. Section 5.4 and Section 5.5 present how the bias-corrected moving averages are estimated, and how the local thresholds are computed, respectively. Section 5.6 describes the process of updating the nodes in the map. Moreover, Section 5.7 and Section 5.8 show in details both unsupervised and supervised learning procedures. Later on, Section 5.9 explains the node removal step in ALTSS-SOM, whereas Section 5.10, the neighborhood. Finally, Section 5.11 summarizes the parameters of the model, and Section 4.9 brings some preliminary results and conclusions.

5.1 OPERATION OF ALTSS-SOM

Similarly to SS-SOM and LARFDSSOM, the overall operation of the ALTSS-SOM comprises three phases: 1) organization (Algorithm 7); 2) convergence; and 3) clustering or classification.

Algorithm 7: ALTSS-SOM

```
1  Initialize parameters $lp$, $\beta$, $age\_wins$, $e_b$, $e_n$, $s$, $minwd$, $epoch_{max}$, $N_{max}$;
2  Initialize the map with one node with $\boldsymbol{c}_j$ initialized at the first input pattern $\boldsymbol{x}_0$, $\boldsymbol{\omega}_j \leftarrow$
     $\mathbf{1}$, $\boldsymbol{\delta_j} \leftarrow \mathbf{0}$, $\hat{\boldsymbol{\delta}}_j \leftarrow \mathbf{0}$, $\mathrm{t}_j \leftarrow 0$, $\mathrm{wins}_j \leftarrow 0$ and $\mathrm{class}_j \leftarrow noClass$ or $class(\boldsymbol{x}_0)$ if
     available;
3  Initialize the variable nwins $\leftarrow 1$;
4  for $epoch \leftarrow 0$ to $epoch_{max}$ do
5      Choose a random input pattern $\boldsymbol{x}$;
6      Compute the activation of all nodes (Equation 3.8);
7      Find the winner $s_1$ with the highest activation (Equation 3.7);
8      if $\boldsymbol{x}$ has a label then
9          Run the SupervisedMode($\boldsymbol{x}$, $s_1$) (Algorithm 11);
10     else
11         Run the UnsupervisedMode($\boldsymbol{x}$, $s_1$) (Algorithm 10);
12     if $nwins = age\_wins$ then
13         Remove nodes with $\mathrm{wins}_j < lp \times age\_wins$;
14         Update the connections of the remaining nodes (Equation 4.1);
15         Reset the number of wins of the remaining nodes:
16         $\mathrm{wins}_j \leftarrow 0$;
17         $nwins \leftarrow 0$;
18     $nwins \leftarrow nwins + 1$;
19 Run the Convergence Phase;
```

In the organization phase, the network is initialized, and the nodes start to compete to form clusters of randomly chosen input patterns. The first node of the map is created at the same position of the first input pattern. As in SS-SOM, there are two distinct ways to define the winner of a competition, to decide when a new node must be inserted and when the nodes need to be updated. However, in ALTSS-SOM, before a node is updated, it is necessary to decide if it will affect the whole node structure or just the weighted averages and the relevance vectors. If the input pattern class label is provided, it will be done in the supervised mode (Section 5.8), otherwise, in the unsupervised mode (Section 4.3).

The neighborhood of ALTSS-SOM is defined as the same as in SS-SOM. Nonetheless, it defines the nodes that will be adjusted together with the winner, thus outlining the cooperation step. The competition and cooperation steps are repeated for a limited number of epochs, and during this process, some nodes are removed periodically, conforming to a defined removal rule that a node must win at least for a minimum number of patterns to continue in the map, as in SS-SOM.

The convergence phase starts right after the organization process. In this phase, the nodes are also updated and removed when required, like in the way conducted in the first phase but with a slight difference: there is no insertion of new nodes. Finally, when the convergence phase finishes, the map clusters and classifies input patterns. At this stage, as in the SS-SOM, there are three possible scenarios: 1) all of the nodes have a defined representing class; 2) a mixed scenario, with some nodes labeled and other not; and 3) none of the nodes labeled. The first scenario will allow both classification and clustering tasks to be executed straightforwardly. The second will add one more step to the process because if the most activated node does not have a defined class, the algorithm continues trying to find a next highly activated node with a defined class. The last scenario only provides the ability to cluster.

5.2 STRUCTURE OF THE NODES

In ALTSS-SOM, each node j in the map represents a cluster and is associated with four m-dimensional vectors, where m is the number of input dimensions: The first three vectors, $\boldsymbol{c}_j$, $\boldsymbol{\omega}_j$, and $\boldsymbol{\delta}_j$, are the same as defined in Section 4.2.1. Note, however, that $\boldsymbol{\delta}$ in SS-SOM and ALTSS-SOM can be seen as the biased first moment estimate. Because of that, ALTSS-SOM introduces a fourth vector, $\hat{\boldsymbol{\delta}}_j = \{\hat{\delta}_{ji}, i = 1, \cdots, m\}$, which is the bias-corrected first moment estimate that the algorithm computes to counteract the bias towards zero of $\boldsymbol{\delta}_j$, specifically at the initial steps. The $\hat{\boldsymbol{\delta}}_j$ vector is used to compute the relevance vector $\boldsymbol{\omega}_j$, and both of them are used to approximate the variance of each node, taking into account how significant each dimension is. Such variance is used to define local reject options during the learning process every time that a new input pattern is presented to the map.

5.3 COMPETITION

ALTSS-SOM tries to choose the winner of a competition as the most activated node given an input pattern $\boldsymbol{x}$, except in certain cases that will be discussed in Section 5.8, when the label is available. In ALTSS-SOM, likewise in SS-SOM, the most activated node $s(\boldsymbol{x})$ is defined as per Equation 3.7, and the activation function is calculated according to a radial basis function with the receptive field adjusted as a function of its relevance vector $\boldsymbol{\omega}_j$, as shown in Equation 3.8.

5.4 ESTIMATING BIAS-CORRECTED MOVING AVERAGES

In ALTSS-SOM, the procedure that updates the distance vectors, as well as the relevance vectors, is shown in the Algorithm 8.

Algorithm 8: Update Relevances

```
  Input : Input pattern $\boldsymbol{x}$, Node $s$
1 Function UpdateRelevances($\boldsymbol{x}$, $s$):
2     Set $t_s \leftarrow t_s + 1$;
3     Update the distance vector $\boldsymbol{\delta}_s$ (Equation 5.1);
4     Update the corrected distance vector $\hat{\boldsymbol{\delta}}_s$ (Equation 5.2);
5     Update the relevance vector $\boldsymbol{\omega}_s$ (Equation 5.3);
```

The distance vectors are initialized as a vector of zeros and updated through a moving average of the observed distance between the input pattern and the current center vector $\boldsymbol{c}_j$, as per Equation 5.1:

$$\boldsymbol{\delta}_j(n+1) = \beta\boldsymbol{\delta}_j(n) + (1-\beta)(|\boldsymbol{x} - \boldsymbol{c}_j(n)|), \tag{5.1}$$

where $\beta \in]0,1]$ is the parameter that controls the rate of change of the moving average (i.e., the exponential decay rate), and $|\boldsymbol{x} - \boldsymbol{c}_j(n)|$ denotes the absolute value applied to the elements of the vectors.

In order to correct the bias towards zero of $\boldsymbol{\delta}_j$ at the initial timesteps, caused by initializing the moving averages with zeros, as in the Adam algorithm (Kingma & Ba, 2014), ALTSS-SOM divides it by the term $\left(1-\beta^{t_j}\right)$, where t_j indicates the current timestep of each node j. In sum, the bias-corrected moving averages vectors are updated at every node timestep according to the Equation 5.2:

$$\hat{\boldsymbol{\delta}}_j(n+1) = \frac{\boldsymbol{\delta}_j(n)}{1-\beta^{t_j}} \tag{5.2}$$

To obtain accurate information about the relevance of each dimension for a given node, an update of the relevance vectors must follow every moving averages update. It is calculated by an inverse logistic function of the bias-corrected estimated distances $\hat{\boldsymbol{\delta}}_{ji}$, as follows in Equation 5.3.

$$\omega_{ji} = \begin{cases} \dfrac{1}{1 + \exp\left(\frac{\hat{\delta}_{ji\text{mean}} - \hat{\delta}_{ji}}{s(\hat{\delta}_{ji\text{max}} - \hat{\delta}_{ji\text{min}})}\right)} & \text{if } \hat{\delta}_{ji\text{min}} \neq \hat{\delta}_{ji\text{max}} \\ 1 & \text{otherwise,} \end{cases} \tag{5.3}$$

where $\hat{\delta}_{ji\text{max}}$, $\hat{\delta}_{ji\text{min}}$ and $\hat{\delta}_{ji\text{mean}}$ are respectively the maximum, the minimum and, the mean of the components of the bias-corrected moving average vector $\hat{\boldsymbol{\delta}}_j$, and the the parameter $s > 0$ controls the slope of the logistic function (Bassani & Araujo, 2015). This function is pretty similar to Equation 3.12, however, instead of using $\boldsymbol{\delta}_j$, the ALTSS-SOM replaces it by $\hat{\boldsymbol{\delta}}_j$ in order to get a more accurate and unbiased moving average value.

5.5 LOCAL THRESHOLDS

The distance vectors $\hat{\boldsymbol{\delta}}$ represent the corrected moving average of the observed distances between the input patterns $\boldsymbol{x}$ and center vectors $\boldsymbol{c}$ for each node j in the map. As a result, they can be considered as the variances of the nodes, as stated before.

In addition, the $\boldsymbol{\omega}$ vectors express how each of the dimensions is important for each node, which indicates the subspaces of the input dimensions of a given dataset. This information corroborates with the definition of a local threshold, together with the estimated variance in the form of the $\hat{\boldsymbol{\delta}}$ vectors.

Combining them, it is possible to define a local region around each node center $\boldsymbol{c}_j$ to act like a reject option for some input patterns. If only the variances were used, some unimportant dimensions with a low variance could misguide the process when a similar input pattern $\boldsymbol{x}$ is outside the acceptance region of a node j, but only in dimensions that are not relevant to it. Therefore, a flexible variance is defined to act as a local threshold and rejection option to mitigate such problems:

$$\text{Var}(\hat{\boldsymbol{\delta}}_j, \boldsymbol{\omega}_j) = \frac{\hat{\boldsymbol{\delta}}_j}{\boldsymbol{\omega}_j} \tag{5.4}$$

When a dimension has a high relevance to the node, it will not impact its variance value. However, when a dimension has a small relevance, ALTSS-SOM will relax the constraints to allow a better definition of subspaces. Therefore, the general acceptance rule is defined by Equation 5.5, where the idea is to approximate to an optimal rule.

$$A(\boldsymbol{x}, \boldsymbol{c}_j, \boldsymbol{v}_j) = \begin{cases} True, & \boldsymbol{x}_i \in \left]\boldsymbol{c}_{ji} \pm \boldsymbol{v}_{ji}\right[, \\ & \forall\, \mathbf{c}_{ji} \in \boldsymbol{c}_j,\, \boldsymbol{x}_i \in \boldsymbol{x}, \text{ and } \boldsymbol{v}_{ji} \in \boldsymbol{v}_j \\ False, & \text{otherwise,} \end{cases} \tag{5.5}$$

where $\boldsymbol{x}$, $\boldsymbol{c}_j$ are respectively the input pattern, and the center vector, and $\boldsymbol{v}_j = \text{Var}(\hat{\boldsymbol{\delta}}_j, \boldsymbol{\omega}_j)$ is the relaxed variance vector as per Equation 5.4.

5.6 NODE UPDATE

As in LARFDSSOM, ALTSS-SOM updates the winner node and its neighbors using two distinct learning rates, e_b, and e_n, respectively. The relevances and weighted moving averages are updated as shown in Section 5.4, and the centroid vector $\boldsymbol{c}_j$, given a learning rate *lr*, is updated through Equation 3.10. The Algorithm 9 shows how the whole update occurs.

Algorithm 9: Node Update of of ALTSS-SOM

```
  Input : Input pattern $\boldsymbol{x}$, Node $s$, Learning Rate $lr$
1 Function UpdateNode($s$, $lr$):
2     UpdateRelevances($\boldsymbol{x}$, s) (Algorithm 8);
3     Update the weight vector $\boldsymbol{c}_s$ (Equation 3.10);
```

5.7 UNSUPERVISED MODE

Given an unlabeled input pattern, the most activated node is considered as the winner, disregarding its class labels. In this sense, ALTSS-SOM verifies if the condition expressed by the Equation 5.5 is satisfied. If so, the winner and its neighbors are updated towards the input pattern. Otherwise, a new node is inserted into the map at the input pattern position. However, since s_1 is the original winner, it will improve its knowledge about the region where it is located by updating its moving averages and relevances, but not its center. This mechanism provides the nodes the ability to learn about the region they are inserted in. An additional case is handled when the map has reached the maximum number of nodes. In this case, aiming at not losing the information that the input pattern can provide, as in previous models, and improving sample efficiency, ALTSS-SOM updates the moving average and the relevance vectors of the winning node. Algorithm 10 illustrates this procedure.

5.8 SUPERVISED MODE

Algorithm 11 shows how this supervised procedure is conducted. If the most activated node s_1 has the same class of the input pattern or a not defined class, a very similar approach to the unsupervised mode is applied. The difference is directly related to the fact that is necessary to set s_1 class as the same class of the given input pattern $\boldsymbol{x}$, as well as to update its connections. Otherwise, the ALTSS-SOM tries to find a new winner with the same class of the input pattern $\boldsymbol{x}$ or a not yet defined class.

If some new node takes the place of s_1 as a new winner s_2, the acceptance criteria expressed by the Equation 5.5 is verified. If so, and the map is not full, the new winner and its neighbors are updated. Otherwise, only the moving averages and relevance vector of s_2 are

Algorithm 10: Unsupervised Mode of ALTSS-SOM

```
   Input : Input pattern $\boldsymbol{x}$ and the first winner $s_1$;
1  if $A(\boldsymbol{x}, \boldsymbol{c}_{s_1}, Var(\hat{\boldsymbol{\delta}}_{s_1}, \boldsymbol{\omega}_{s_1}))$ is True and $N < N_{max}$
2  then                                                        ▷ See Equation 5.5
3      Update the winner node and its neighbors: UpdateNode($s_1$, $e_b$),
         UpdateNode($neighbors$($s_1$), $e_n$) (Algorithm 9);
4      Set wins$_{s_1} \leftarrow$ wins$_{s_1}$ + 1;
5  else if $A(\boldsymbol{x}, \boldsymbol{c}_{s_1}, Var(\hat{\boldsymbol{\delta}}_{s_1}, \boldsymbol{\omega}_{s_1}))$ is False then     ▷ See Equation 5.5
6      Create a new node $j$ and set: $\boldsymbol{c}_j \leftarrow \boldsymbol{x}$, $\boldsymbol{\omega}_j \leftarrow \mathbf{1}$, $\boldsymbol{\delta}_j \leftarrow \mathbf{0}$, $\hat{\boldsymbol{\delta}}_j \leftarrow \mathbf{0}$, $t_j \leftarrow 0$, class$_j$
         $\leftarrow$ $noClass$ and wins$_j \leftarrow lp \times nwins$;
7      Connect j to the other nodes as per Equation 4.1;
8      Update the relevances vector of $s_1$: UpdateRelevances($\boldsymbol{x}$, $s_1$) (Algorithm 8);
9  else
10     Update the relevances vector of $s_1$: UpdateRelevances($\boldsymbol{x}$, $s_1$) (Algorithm 8);
```

updated in order to give the same chance received by s_1 to improve its knowledge about the surrounding area.

If there are no new nodes to replace s_1 as a new winner, and the map is not full, the s_1 node is duplicated, preserving the moving averages vectors, the centroid vector as well as the relevance vector. However, the class of this new duplicated node is set to the same as the input pattern. The other parameters are set as usual. If none of the above conditions are fulfilled, the ALTSS-SOM solely updates the moving averages and relevance vector of the first defined winner s_1.

5.9 NODE REMOVAL

In ALTSS-SOM, as in SS-SOM and LARFDSSOM, each node j stores a variable $wins_j$ that accounts for the number of nodes victories since the last reset. Whenever $nwins$ reaches the age_wins value, a reset occurs. It implies to the moment when nodes that did not win at least the minimum percentage of the competition $lp \times age\_wins$ are removed from the map. After a reset, the number of victories of the remaining nodes is reset to zero. Finally, to avoid the removal of recently created nodes, when a new node is inserted, its number of wins is set to $lp \times nwins$, where $nwins$ indicates the number of competitions that have occurred since the last reset.

5.10 NEIGHBORHOOD UPDATE

As in SS-SOM, the neighborhood ALTSS-SOM is formed by nodes with the same class label or a not defined class label that presents a similar relevance in the input dimensions. It means that two nodes are connected if they have similar classes and subspaces. Equation 4.1 considers these criteria for every pair of nodes.

Algorithm 11: Supervised Mode of ALTSS-SOM

```
Input : Input pattern $\boldsymbol{x}$ and the first winner $s_1$;
if $class_{s_1} = class(\boldsymbol{x})$ or $class_{s_1} = noClass$ then
    if $A(\boldsymbol{x}, \boldsymbol{c}_{s_1}, Var(\hat{\boldsymbol{\delta}}_{s_1}, \boldsymbol{\omega}_{s_1}))$ is False and $N < N_{max}$
    then                                                        ▷ See Equation 5.5
        Create new node $j$ and set: $\boldsymbol{c}_j \leftarrow \boldsymbol{x}$, $\boldsymbol{\omega}_j \leftarrow \mathbf{1}$, $\boldsymbol{\delta}_j \leftarrow \mathbf{0}$, $\hat{\boldsymbol{\delta}}_j \leftarrow \mathbf{0}$, $t_j \leftarrow 0$, class$_j$
          $\leftarrow class(\boldsymbol{x})$ and wins$_j \leftarrow lp \times nwins$;
        Connect j to the other nodes as per Equation 4.1;
        Update the relevances vector of $s_1$: UpdateRelevances($\boldsymbol{x}$, $s_1$) (Algorithm 8);
    else if $A(\boldsymbol{x}, \boldsymbol{c}_{s_1}, Var(\hat{\boldsymbol{\delta}}_{s_1}, \boldsymbol{\omega}_{s_1}))$ is True then          ▷ See Equation 5.5
        Update the winner node and its neighbors: UpdateNode($s_1$, $e_b$),
          UpdateNode($neighbors(s_1)$, $e_n$) (Algorithm 9);
        Set class$_{s_1} \leftarrow class(\boldsymbol{x})$;
        Update $s_1$ connections as per Equation 3.13;
        Set wins$_{s_1} \leftarrow$ wins$_{s_1} + 1$;
    else
        Update the relevances vector of $s_1$: UpdateRelevances($\boldsymbol{x}$, $s_1$)
else
    Try to find a new winner $s_2$ as the next node with highest activation and $noClass$
      or the same class of $\boldsymbol{x}$;
    if $s_2$ exists then
        if $A(\boldsymbol{x}, \boldsymbol{c}_{s_2}, Var(\hat{\boldsymbol{\delta}}_{s_2}, \boldsymbol{\omega}_{s_2}))$ is True and $N < N_{max}$
        then                                                    ▷ See Equation 5.5
            Update the new winner node and its neighbors: UpdateNode($s_2$, $e_b$) and
              UpdateNode($neighbors(s_2)$, $e_n$) (Algorithm 9);
        else
            Update the relevances vector of $s_2$: UpdateRelevances($\boldsymbol{x}$, $s_2$)
        Set wins$_{s_2} \leftarrow$ wins$_{s_2} + 1$;
    else if $N < N_{max}$ then
        Create new node $j$ by duplicating $s_1$ and set: $\boldsymbol{c}_j \leftarrow \boldsymbol{c}_{s_1}$, $\boldsymbol{\omega}_j \leftarrow \boldsymbol{\omega}_{s_1}$, $\boldsymbol{\delta}_j \leftarrow \boldsymbol{\delta}_{s_1}$,
          $\hat{\boldsymbol{\delta}}_j \leftarrow \hat{\boldsymbol{\delta}}_{s_1}$, $t_j \leftarrow 0$, class$_j \leftarrow class(\boldsymbol{x})$ and wins$_j \leftarrow lp \times nwins$;
        Connect $j$ to other nodes as per Equation 4.1;
    else
        Update the relevances vector of $s_1$: UpdateRelevances($\boldsymbol{x}$, $s_1$)
```

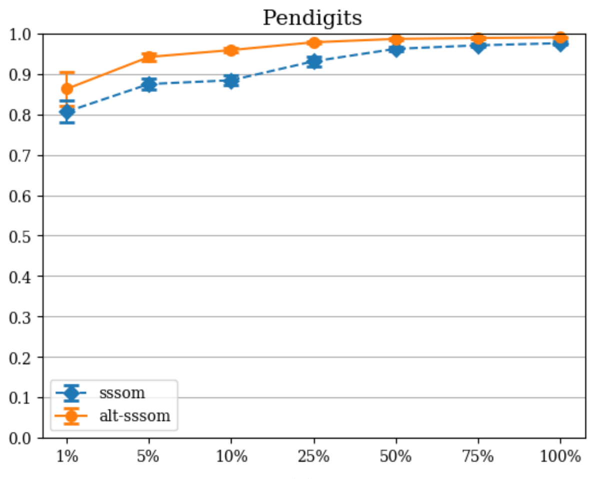
5.11 A SUMMARY OF THE PARAMETERS

ALTSS-SOM removes two parameters from its predecessor, SS-SOM. First, the parameter a_t, that has a great impact on the results as shown by Bassani & Araujo (2015) and Section 4.8. It was replaced by the adaptive local threshold technique introduced by ALTSS-SOM (Section 5.5) that can define and learn the space region that a node can represent during the training. Second, the parameter e_w was also removed due to its irrelevance in the learning process after removing a_t, i.e., ALTSS-SOM has nine parameters to be set up. It was concluded from sensitivity analysis using scatter plots of the obtained metric results as function of the parameters values, and linear fits to the data, as conducted in Section 4.9. More precisely, such analysis revealed that there is no parameter with a high impact on the results anymore. This method seeks to establish a good level of self-adjustment, in a way that we can keep the parameters values fixed inside predefined ranges.

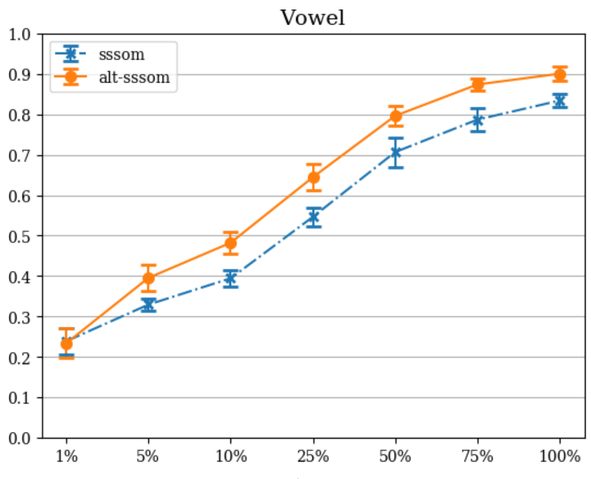
5.12 PRELIMINARY EXPERIMENTAL RESULTS AND CONCLUSIONS

This section presents the preliminary results of ALTSS-SOM in order to continue the chronological order of the research. The further details of the experiments done with ALTSS-SOM will be given in Chapter 6. For the current stage of reading, the preliminary experiments compare the classification performance of ALTSS-SOM against SS-SOM with the same experimental setup and methodology used in Section 4.9.

The results exhibited in Figure 10 illustrate the improvements that this adaptive local threshold approach can provide. Its performance is superior to SS-SOM in all levels of supervision.



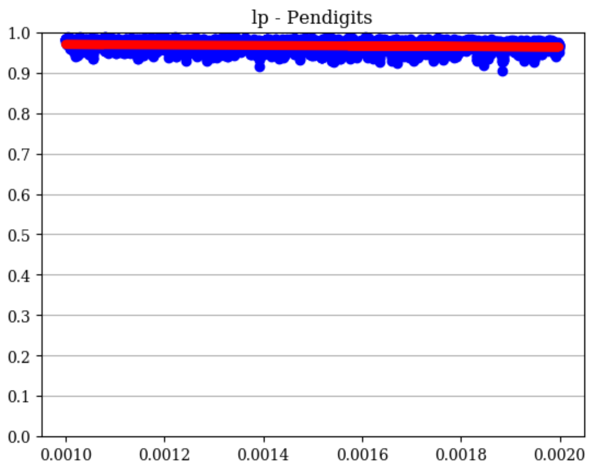
(a)



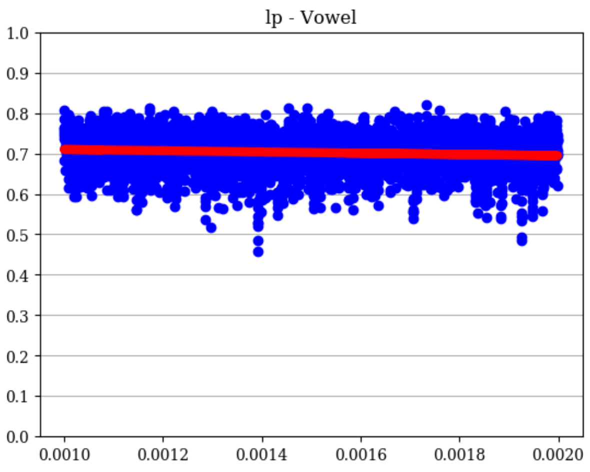
(b)

Figure 10: Best mean accuracy and standard deviation as function of the percentage of supervision on (a) Pendigits and (b) Vowel datasets

As said in Section 5.11, the ALTSS-SOM has nine parameters, but none of them has a high impact on the results anymore. In Figure 11, the lp was chosen to illustrate this behavior, once it was a parameter of significant impact in previous versions. It is clear that the dependency on the parameters is softened due to how ALTSS-SOM cope with the data information during the training. All the other parameters have the same behavior, playing the same role in the model.



(a)



(b)

Figure 11: Preliminar sensitive analysis with a scatter plot of the Accuracy obtained with ALTSS-SOM as a function of parameter lp on (a) Pendigits and (b) Vowel datasets

This chapter presented the ALTSS-SOM, an approach for classification and clustering with SSL that introduces the concept of adaptive local thresholds to define the local receptive field of the nodes. Next, in Chapter 6, all the experiments of an extensive evaluation with several datasets will be shown in order to draw the conclusions and future work of this dissertation.

6 MODELS VALIDATION

6.1 METRICS

The accuracy metric was used to evaluate the proposed models with respect to the others regarding classification rate. It is the ratio of the number of correct predictions to the total number of input samples. On the other hand, for the experiments involving the clustering task solely, the CE metric takes place. It was previously defined in Section 2.6.1.

6.2 DATASETS

All, the datasets used in the experiments were specified in Section 2.6.2. First, the performance of the proposed models was evaluated in the seven real-world datasets provided by the OpenSubspace framework (Müller *et al.*, 2009). However, the framework does not include information about their relevant dimensions. Therefore, all of the dimensions were considered as relevant when calculating the CE metric in the clustering experiments. The datasets were also used to evaluate the model on considering the accuracy metric.

Second, in order to extend the application range of the proposed models by evaluating their capabilities of classifying and clustering images, traditional image benchmark datasets, such as CIFAR10, MNIST, FashionMNIST, and SVHN were used. However, to make it feasible, it was necessary to perform a feature extraction step followed by a transfer learning technique. It was done using a pre-trained network on ImageNet (Deng *et al.*, 2009), and then evaluating SS-SOM similarly to the experiments conducted in Medeiros *et al.* (2018). Those results can provide a better understanding of their capabilities. Moreover, Batch SS-SOM was studied using a different approach to extract features.

6.3 PARAMETERS TUNING

Usually, subspace clustering methods have several parameters and adjusting them is not an easy task. So, aiming to address this issue, each method was run 500 times with different parameter values sampled within a previously established interval, according to a LHS (McKay *et al.*, 1979; Helton *et al.*, 2005), which was described in Section 2.6.3. Moreover, to define the best ranges to be used, sensitive analysis techniques were used, as in Iman & Helton (1988) and Saltelli *et al.* (2000). They both show that given the probabilistic basis of LHS, it can provide direct estimates for the Cumulative Distribution Function (cdf) and variance of models. It is done by pairing the obtained results measure and the parameter values sampled from the LHS distribution used to obtain each of the results. Also, it is possible to use this in combination with a linear regression model to get intuitions about the trends of the obtained results as a function of the parameter value used (Iman & Helton, 1988; Saltelli *et al.*, 2000). Such an analysis can be done for each parameter in order to draw a better understanding of their impact and influence on the models.

It is important mentioning that SOM-based methods are stochastic and/or depend on initial conditions. Despite they may not achieve their best results in a single run, for all of the experiments performed with real-world datasets (Table 1), the methods were executed only once for each parameter setting in order to keep the same number of experiments. The parameter ranges for the proposed and comparable methods will be given the next sections. Precisely for the proposed methods, the ranges used are suitable for datasets scaled in the [0, 1] interval. For the other methods used in CE comparison, the parameter ranges were the same as those used in Bassani & Araujo (2015).

6.4 STATISTICAL TESTS

In the following Sections and Chapters, there are mentions to statistical differences between models. When this occurs, the statistical test used was the Wilcoxon test (Wilcoxon, 1945).

The Wilcoxon test is a non-parametric statistical test used to compare two related samples or measurements. It can be seen as an alternative to the t-test when the population cannot be assumed to be normally distributed. As this was not trivial to determine, the Wilcoxon was chosen with a level of significance of 5%. Another reason for the choice of the Wilcoxon test is the dependency of the measurement samples, once they are originated from the same datasets for each further evaluated model, showing then a condition of relation.

6.5 EXPERIMENTAL SETUP

In order to evaluate the performance of the proposed models in different contexts and applications, the experiments were organized as follows:

1. Section 6.6 evaluates the performance of SS-SOM in conditions of different percentages of labeled data.

2. Section 6.7 illustrates the sensitivity of SS-SOM to its parameters and how it can impact the results.
3. Section 6.8 demonstrates an extended capability of SS-SOM by applying the model to perform classification tasks using traditional image classification datasets and transfer learning techniques.
4. Section 6.9 analyzes a case study concerning the performance and behavior of the proposed Batch SS-SOM on traditional image classification datasets.
5. Section 6.10 pictures the performance of ALTSS-SOM in the same conditions as those of Section 6.6.
6. Section 6.11 discusses the results obtained by ALTSS-SOM for clustering tasks solely, without using any labels in comparison with other clustering methods.
7. Section 6.12 provides the sensitivity analysis of ALTSS-SOM in order to draw some conclusions about the improvements made since the first proposed version.
8. Section 6.13 shows the outcomes obtained from both SS-SOM and ALTSS-SOM in a completely supervised scenario.

All of the aforementioned experiments will give the conditions to draw the proper conclusions about the conducted research.

6.6 SS-SOM: CLASSIFICATION ACCURACY WITH DIFFERENT PERCENTAGES OF LABELED DATA

In order to evaluate the classification capabilities of SS-SOM, it is compared with the following semi-supervised methods: Label Propagation (Xiaojin & Zoubin, 2002) and Label Spreading (Zhou *et al.*, 2004). They were described in Section 2.5, and chosen due to their popularity and connection to several other important algorithms, such as Markov random walks (Szummer & Jaakkola, 2002) and mean field approximation (Peterson, 1987; Jordan *et al.*, 1999). In this sense, for studying the effects of the different levels of supervision, i. e., the percentage of labeled data, all of them were trained, using the seven real-world datasets from the OpenSubspace framework (Müller *et al.*, 2009), with the following percentages: 1%, 5%, 10%, 25%, 50%, 75% and 100%. Also, the maximum number of nodes for SS-SOM (N_{max}) was set to be the size of the training set, because, in the worst case, it is expected that the model creates one cluster for each training sample.

So, for all of the algorithms, on each dataset, a 3-times 3-fold cross-validation was used. Each method was trained and tested 500 times for each fold with different parameter values sampled from the parameter ranges presented in Table 3 and Table 4, according to a LHS (Helton *et al.*, 2005), while the best accuracy achieved by each method in each fold was recorded for each dataset. It comprises a total of 752,000 experiments. After that, the mean and the standard deviation of the best results for each dataset were calculated separately. A projected clustering problem was considered, where each sample should be assigned to a single cluster, and SS-SOM was set to operate in such mode. For classification purposes, if available, we use the node class as the predicted class. Otherwise, it is straightforwardly considered as an error. The next section presents the obtained results and their analysis.

Table 3: Parameter Ranges for SS-SOM

Parameters	**min**	**max**
Activation threshold (a_t)	0.80	0.999
Lowest cluster percentage (lp)	0.001	0.01
Relevance rate (β)	0.001	0.5
Max competitions (age_wins)	$1 \times S^*$	$100 \times S^*$
Winner learning rate (e_b)	0.001	0.2
Wrong winner learning rate (e_w)	$0.01 \times e_b$	$1 \times e_b$
Neighbors learning rate (e_n)	$0.002 \times e_b$	$1 \times e_b$
Relevance smoothness (s)	0.01	0.1
Connection threshold ($minwd$)	0	0.5
Number of epochs ($epochs$)	1	100

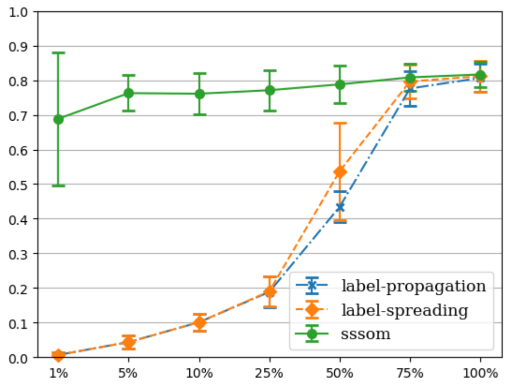
* S is the number of input patterns in the dataset.

Table 4: Parameter Ranges for Label Spreading and Label Propagation

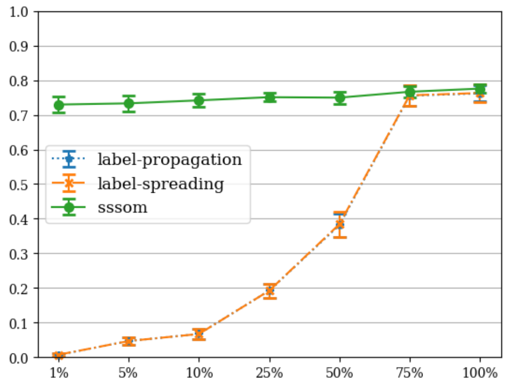
Parameters	**min**	**max**
Kernel Function[1]	1	2
γ (for RBF Kernel)	10	30
Number of Neighbors (for KNN Kernel)	1	100
α^*	0	1
Number of epochs	20	100

[1]1: RBF and 2: KNN. * α is only used for label spreading.

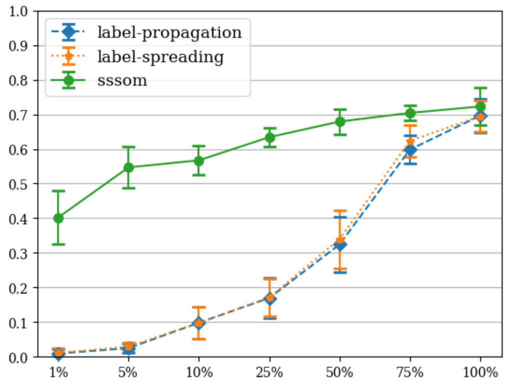
Figure 12 shows the obtained results as a function of the percentage of labeled data. In all datasets, the performance of the SS-SOM is superior to the other semi-supervised methods concerning the supervision rate between 1% up to 75%, whereas with the highest percentage (100%) the difference is smaller, but it still outperforms them or obtain comparable results. These results show the robustness of SS-SOM in situations when only a small number of labeled data is available.



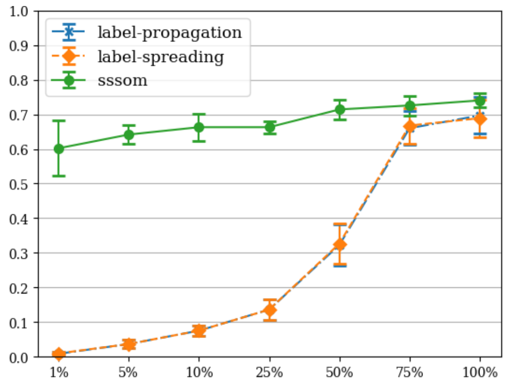
(a) Breast



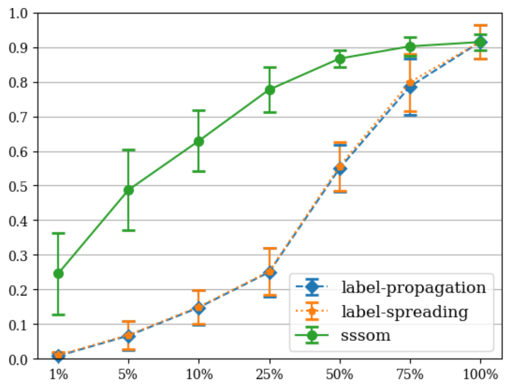
(b) Diabetes



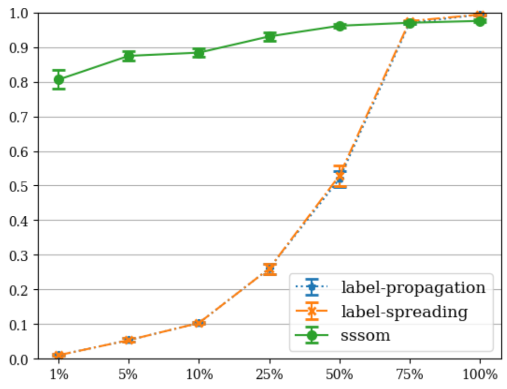
(c) Glass



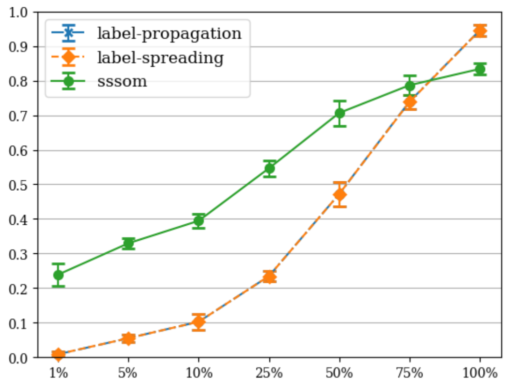
(d) Liver



(e) Shape



(f) Pendigits



(g) Vowel

Figure 12: Best mean accuracy and standard deviation as function of the percentage of supervision on (a) Breast, (b) Diabetes, (c) Glass, (d) Liver, (e) Shape, (f) Pendigits, and (g) Vowel datasets for SS-SOM, Label Spreading and Label Propagation.

6.7 SS-SOM: SENSITIVITY ANALYSIS

Bassani & Araujo (2015) showed examples of scatter plots produced when conducting the sensitivity analysis of LARFDSSOM clustering the seven real-world datasets from OpenSubspace framework. The scatter plots indicated that the parameter a_t is primarily responsible for the variation observed in the results, followed by l_p. To get a deeper understanding about such analysis, we replicated the sensitivity analysis for SS-SOM. Note that SS-SOM works exactly as LARFDSSOM when there are no labels available. So that, it is expected from SS-SOM to reproduce the same behavior of LARFDSSOM in such analysis.

The charts displayed in Figure 13 were obtained from SS-SOM. It is clear, as for Bassani & Araujo (2015), that a_t plays a role of significant impact on the results.

To highlight this behavior, the parameter a_t was isolated for a thorough analysis with seven datasets. The results can be seen in Figure 14. For instance, in Figure 14f, a slight change in a_t, e.g., from 0.96 to 0.98, results in a drastic fall of Accuracy. It is reasonable to repeat such analysis taking into consideration the experiments with 50% of labeled data, since this will allow us assess the average results of both supervised and unsupervised learning procedures, once 50% is a halfway between both usages at training time.

The effect of lp is much less significant then the impact of a_t. Therefore, it can be better observed with a_t assuming a fixed value. In this regard, the behavior remained the same as for LARFDSSOM (Bassani & Araujo, 2015). With such analysis at hand, it is clear the necessity to build a methods less sensitive to the change of parameters values.

6.8 SS-SOM: IMAGE CLASSIFICATION PERFORMANCE

SS-SOM was put to the test for image classification benchmarks trying to demonstrate an extended capability and application range. More precisely, a pre-trained DenseNet-161 (Huang *et al.*, 2017) network on ImageNet (Deng *et al.*, 2009) was used to perform a transfer learning, extracting the features of each dataset shown in Table 2 to feed the SS-SOM. DenseNet was chosen due to its performance results for object recognition tasks on CIFAR10, CIFAR100, SVHN and ImageNet. DenseNets are currently the state-of-the-art on most of them (Huang *et al.*, 2017). Figure 15 shows their architecture. Moreover, the suffix "161" denotes the configuration of the convolutional layers in each Dense block, that can be seen in Figure 15.

Additionally, the image resolution on ImageNet normally varies depending on the application, but on average it is 469x387 pixels, and normally a pre-processing step is applied to rescales them to 256x256. However, the experiments conducted in this work considered a resolution of 224x224 to fit the input size of the DenseNet-161.

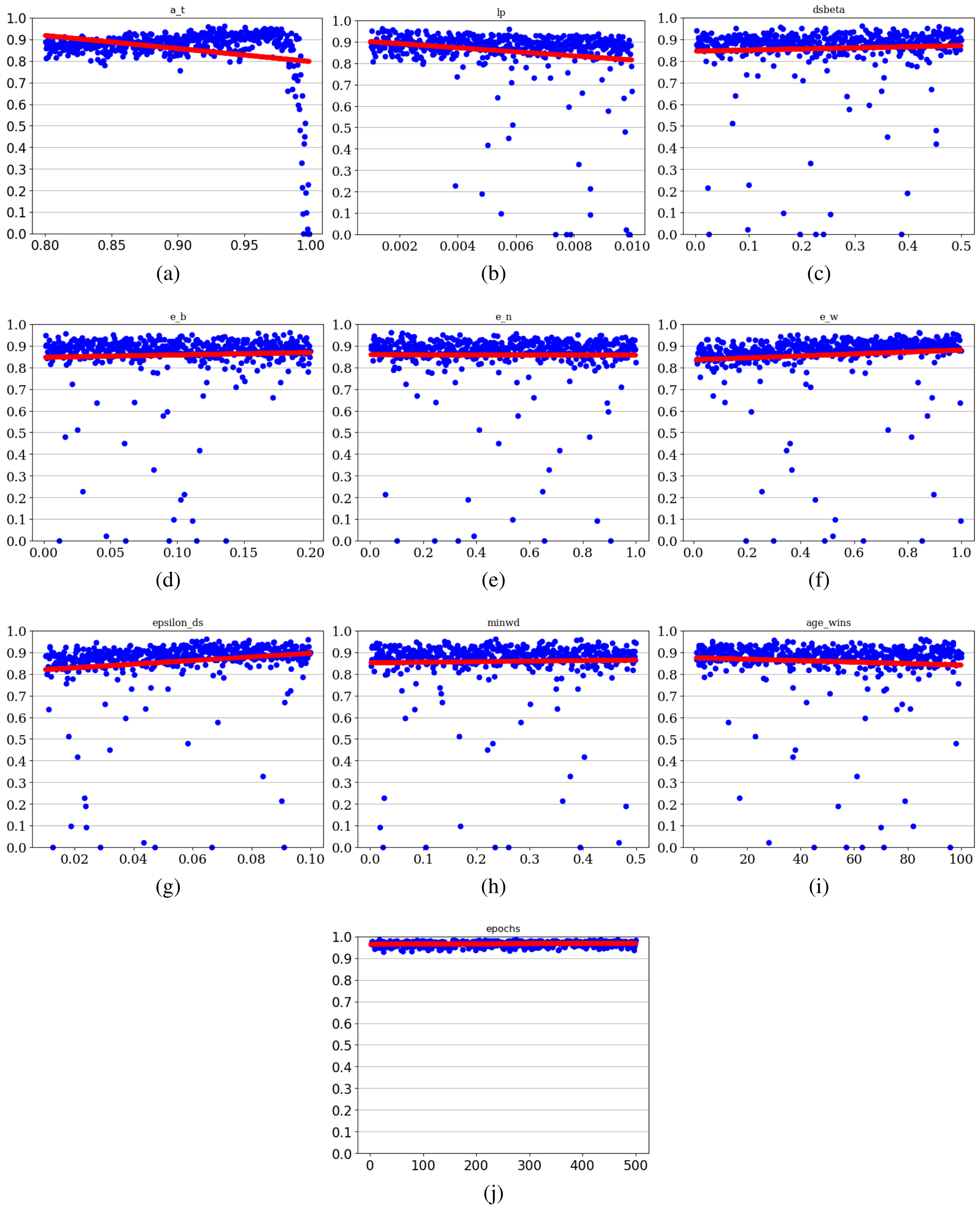


Figure 13: Sensitivity analysis with a scatter plot of the Accuracy obtained with SS-SOM as a function of its parameters, (a) a_t, (b) lp, (c) dsbeta, (d) e_b, (e) e_n, (f) e_w, (g) epsilon_ds, (h) minwd, (i) age_wins, and (j) epochs, for Pendigits real-world dataset using 50% of the available labels, and 1 fold randomly chosen from the 3-times 3-fold cross validation scheme. The red lines are the linear fits to the data, which is a common and effective way of studying the sensitivity of parameters, as Iman & Helton (1988) demonstrates.

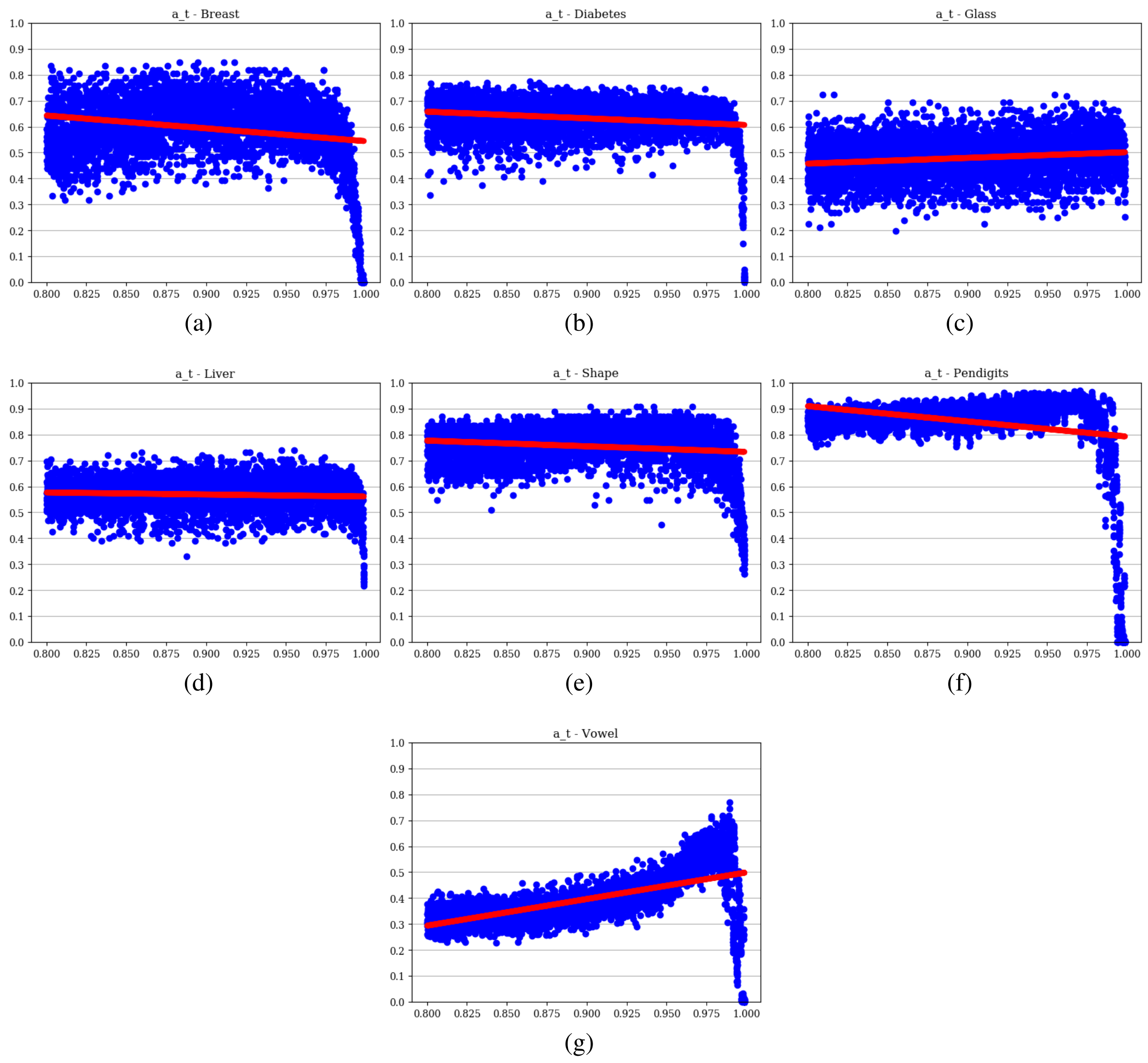


Figure 14: Sensitivity analysis with a scatter plot of the Accuracy obtained with SS-SOM as a function of parameter a_t on (a) Breast, (b) Diabetes, (c) Glass, (d) Liver, (e) Shape, (f) Pendigits, and (g) Vowel real-world datasets using 50% of the available labels. The red lines are the linear fits to the data.

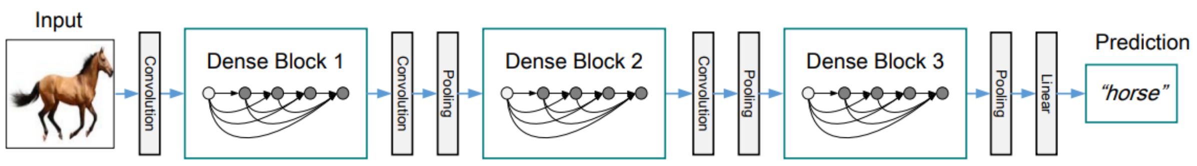


Figure 15: DenseNet architecture (Huang *et al.*, 2017)

Particularly, the performance on the features extracted from CIFAR10 dataset with DenseNet-161 (Huang *et al.*, 2017) pre-trained model using the PyTorch framework (Paszke *et al.*, 2017) was measured. A sampling of 4000 balanced labeled data from the CIFAR10 was made to avoid oversampling or undersampling of any class when trying to evaluate the result using a low amount of labeled data. After that, the SS-SOM was trained using the same ranges as those defined in Table 3. However, only ten parameters set were sampled, and the one that presented the best results on average was chosen to perform the experiments and compare further with other models, not only with 4000 balanced labeled data but also with all labels.

In this context, the SS-SOM is compared with with S3C (Hui, 2013) and VIK (Goodfellow *et al.*, 2012). The S3C is an effective feature discovery algorithm for both supervised and semi-supervised learning with small amounts of labeled data based on inferences. The VIK is a modified simple K-means dictionary learning. It extends the concept of spatial pooling by drawing a strategy of directly modeling complex invariances of object features. Also, despite being remarkable in the literature, they both have an experimental methodology in which it was possible to compare without the need for implementation and rerunning. They just tune the model to its parameters, and after finding the best configuration, the model is run only once.

Table 5 shows the obtained results. The SS-SOM presented the best results. It is worth mentioning that because of computational costs, it was not possible to reproduce more recent techniques of the area, such as Wide ResNet (WRN-28-2) (Oliver *et al.*, 2018), that claims to be the state-of-the-art for SSL. Despite the fact they are harder to beat, this set of experiments clarified that SS-SOM can perform well for such complex problems.

Table 5: Classification rate obtained by SS-SOM, S3C and VIK using 4000 and all labeled data.

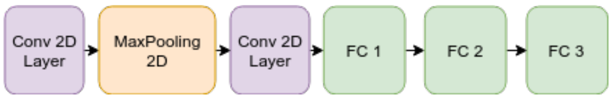
Accuracy	**4000**	**All**
SS-SOM	**0.72**	**0.85**
Spike-and-Slab Sparse Coding	0.68	-
View-Invariant K-means	0.71	0.82

6.9 BATCH SS-SOM: A CASE STUDY

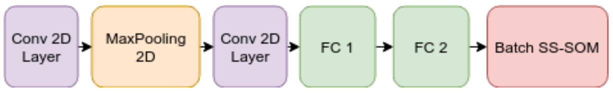
This case study aims at analyzing if the proposed Batch SS-SOM also performs well in image classification benchmark datasets, since SS-SOM presented good results concerning the classification rate for CIFAR10 allied with feature extraction and a transfer learning technique. The following strategy was developed to do so. First, since MNIST and Fashion-MNIST are composed of grayscale images, a new Convolutional Neural Network (CNN) model (Goodfellow *et al.*, 2016) was intended to be trained from scratch, for feature extraction. Specifically, the features before the classification layer were used as the input for Batch SS-SOM. Second, several supervision rates, i.e., the percentage of labeled data, were defined, as in Section 6.6, for studying the effects of such variability of available labels in the outcome results on three image datasets: MNIST, Fashion-MNIST, and SVHN. It is worth mentioning that the sampling was not balanced.

Different architectures were used in order to achieve better results for each dataset in particular. Figure 16, Figure 17, and Figure 18 show the pipeline used to extract the features of

SVHN, MNIST, and FashionMNIST, respectively. For MNIST and FashionMNIST, the extracted features fed Batch SS-SOM model with 32 input dimensions, whereas for SVHN, it was 84 input dimensions. The chosen numbers of dimensions were arbitrarily defined at first hand. However, the obtained results were satisfactory for the sake of discussion, since they were close to other benchmark results as Liang & Hu (2015), Netzer *et al.* (2011), and Xiao *et al.* (2017). Therefore, it is necessary to carry out more detailed experiments to figure out and understand the behavior that such numbers of dimensions have in the performance of Batch SS-SOM. Finally, some recent well-established techniques such as Batch Normalization (Ioffe & Szegedy, 2015) and Dropout (Srivastava *et al.*, 2014) were used while building the architectures. Moreover, the most currently used activation function, Rectified Linear Units (ReLU) (Nair & Hinton, 2010), was also employed.

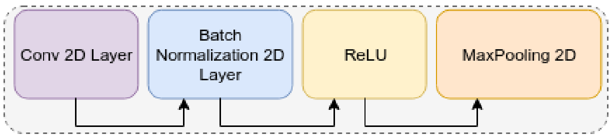


(a) Custom SVHN CNN Model.

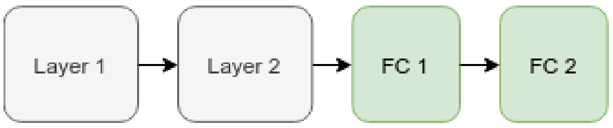


(b) Batch SS-SOM with features extract from layer before classifier.

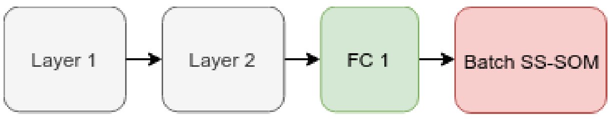
Figure 16: SVHN Training Pipeline



(a) Custom layer used on MNIST CNN Model.



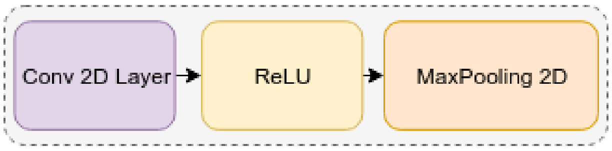
(b) MNIST CNN Model.



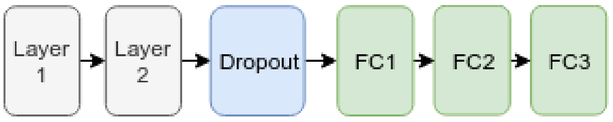
(c) Batch SS-SOM with features extract from layer before classifier.

Figure 17: MNIST Training Pipeline

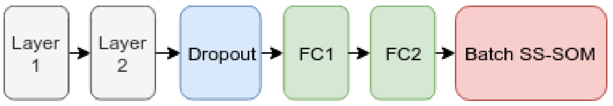
The results presented in Figure 19 show that the Batch SS-SOM performs well even with few labels and still improves as the number of labeled samples grows. Note, however, that such gains are not so significant, especially after a certain point that can be defined around



(a) Custom layer used on FashionMNIST CNN Model.



(b) FashionMNIST CNN Model.



(c) Batch SS-SOM with features extract from layer before classifier.

Figure 18: FashionMNIST Training Pipeline

5% of labeled data, where the performance stabilizes. This behavior was observed in all datasets, showing that the Batch SS-SOM is a good approach for training in image classification benchmarks. Notice that it is a challenge for a great variety of models. Such performance obtained by the Batch SS-SOM defines a promising path through the use and application of SOM-based methods for more complex and new challenges imposed by the advances of technology. The ranges for Batch SS-SOM were defined as the same as those used in Table 3 for the original SS-SOM.

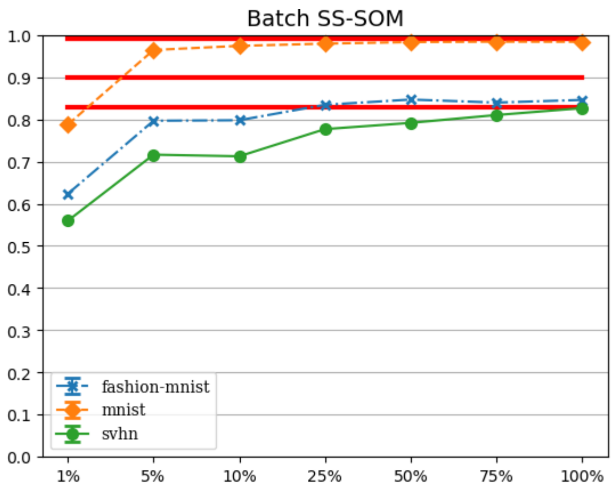


Figure 19: The Accuracy results obtained with Batch SS-SOM for FashionMNIST, SVHN and MNIST dataset as function of the percentage of labeled data. The red lines indicate the results that were targeted for MNIST, SVHN and FashionMNIST, respectively, from top to bottom (Liang & Hu, 2015; Netzer *et al.*, 2011; Xiao *et al.*, 2017).

6.10 ALTSS-SOM: CLASSIFICATION ACCURACY WITH DIFFERENT PERCENTAGESOF LABELED DATA

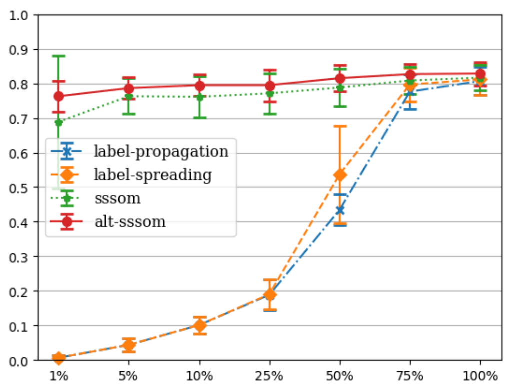
In order to evaluate the classification rate of ALTSS-SOM, the experiments conducted in Section 6.6 were replicated, adding the ALTSS-SOM in the comparison. The ranges used for ALTSS-SOM are defined in Table 6, whereas the ranges of the other methods were the same as those used in Section 6.6. The maximum number of nodes for ALTSS-SOM was set to be $N_{max} = 200$.

Table 6: Parameter Ranges for ALTSS-SOM

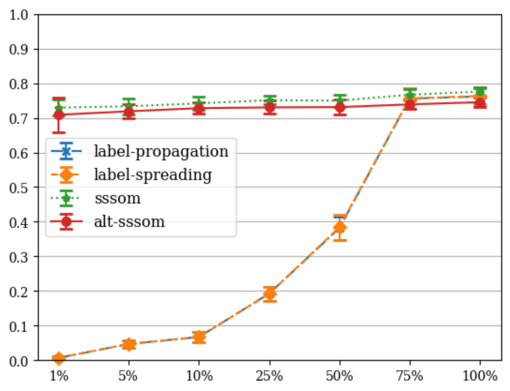
Parameters	**min**	**max**
Lowest cluster percentage (lp)	0.001	0.002
Relevance rate (β)	0.90	0.95
Max competitions (age_wins)	$1 \times S^*$	$100 \times S^*$
Winner learning rate (e_b)	0.001	0.2
Neighbors learning rate (e_n)	$0.002 \times e_b$	$1 \times e_b$
Relevance smoothness (s)	0.01	0.2
Connection threshold ($minwd$)	0	0.5
Number of epochs ($epochs$)	1	100

* S is the number of input patterns in the dataset.

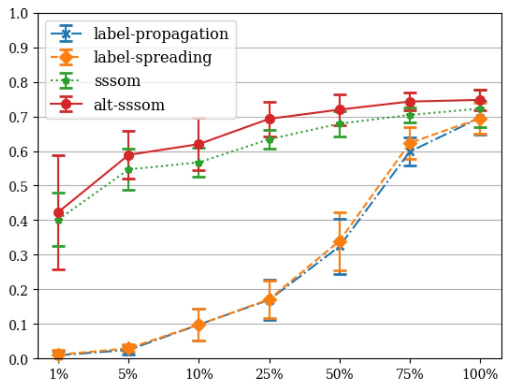
Figure 20 shows the results of ALTSS-SOM in comparison with SS-SOM, Label Propagation and Label Spreading in the real-world datasets. The results are shown as a function of the percentage of labels that were used. Overall, ALTSS-SOM improved the performance of SS-SOM, except for the Diabetes dataset (Figure 20b) where the results obtained were slightly lower, but yet comparable. The flexibility provided by the estimation of the receptive field of nodes, and the improved sample efficiency allowed such results. Still, the standard deviation for all datasets in all supervision levels was also minimized, which indicates another positive aspect of the ALTSS-SOM: it is more robust to variations on both datasets and parameters. While the other semi-supervised learning methods surpassed SS-SOM in Pendigits and Vowel datasets, ALTSS-SOM achieved a consistent improvement by outperforming the results of both LP and LS, in all datasets, except for Vowel at 100%.



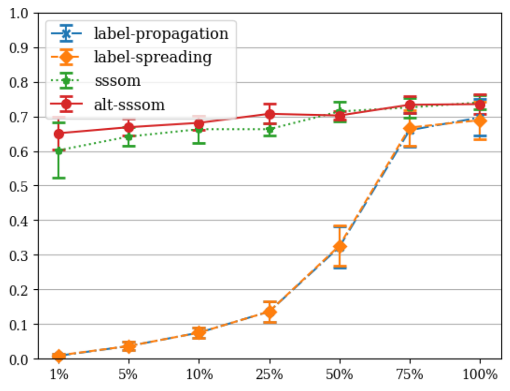
(a) Breast



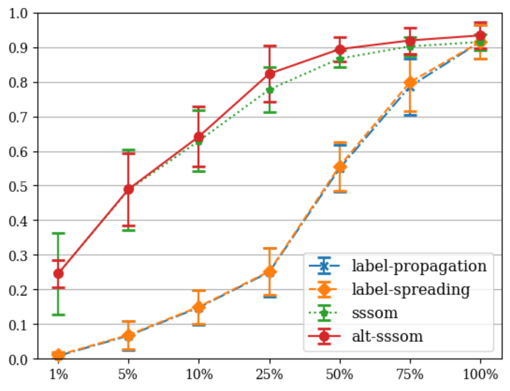
(b) Diabetes



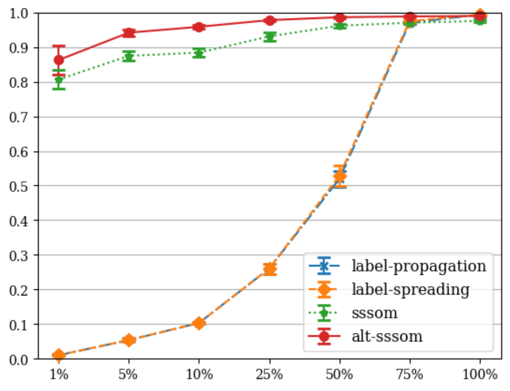
(c) Glass



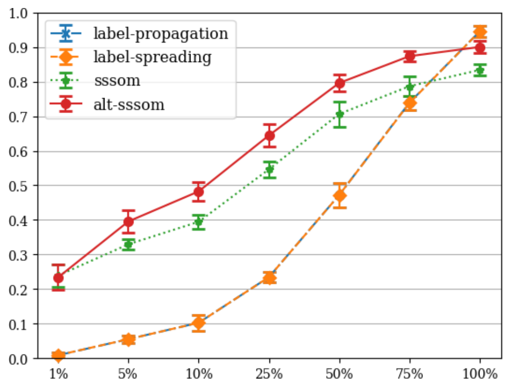
(d) Liver



(e) Shape



(f) Pendigits



(g) Vowel

Figure 20: Best mean accuracy and standard deviation as function of the percentage of supervision on (a) Breast, (b) Diabetes, (c) Glass, (d) Liver, (e) Shape, (f) Pendigits, and (g) Vowel datasets for ALTSS-SOM, SS-SOM, Label Spreading and Label Propagation

6.11 ALTSS-SOM: CLUSTERING PERFORMANCE

Aiming to assess the performance of ALTSS-SOM in a purely unsupervised clustering task due to the changes made in the original framework that it was inspired, it was compared with Densitive-based Optimal projective Clustering (DOC), PROjected CLUStering algorithm (PROCLUS) and LARFDSSOM/SS-SOM. We refer the LARFDSSOM and SS-SOM together due to their equivalence for clustering tasks solely, as mentioned in Chapter 4. The first two methods are originally from the data mining area. This choice of comparison was defined in accordance to the analysis provided by Bassani & Araujo (2015), where LARFDSSOM presented the best results overall, and DOC and PROCLUS appeared as the two best options on average concerning subspace approaches in a distinction of data mining applications. The parameters used for ALTSS-SOM to execute such clustering tasks are slightly different from the previous classification tasks. They are pointed in Table 7, whereas the ranges of the other methods were the same as those used in Bassani & Araujo (2015).

Table 7: Parameter Ranges for ALTSS-SOM on Clustering Tasks

Parameters	**min**	**max**
Lowest cluster percentage (lp)	0.001	0.01
Relevance rate (β)	0.001	0.5
Max competitions (age_wins)	$1 \times S^*$	$100 \times S^*$
Winner learning rate (e_b)	0.001	0.2
Neighbors learning rate (e_n)	$0.002 \times e_b$	$1 \times e_b$
Relevance smoothness (s)	0.01	0.1
Connection threshold ($minwd$)	0	0.5
Number of epochs ($epochs$)	1	100

* S is the number of input patterns in the dataset.

For a better understanding of the comparison, DOC (Procopiuc *et al.*, 2002) is a cell-based method that searches for sets of grid cells containing more than a certain number of objects by using a Monte Carlo based approach that computes, with high probability, a good approximation of an optimal projective cluster. PROCLUS (Aggarwal *et al.*, 1999) is a clustering-oriented algorithm that aims to find clusters in small projected subspaces. It is done by optimizing an objective function of the entire set of clusters, such as the number of clusters, average dimensionality, or other statistical properties.

Table 8 shows the results of the CE obtained with the methods. It shows that no method achieved the best result for all real-world datasets. ALTSS-SOM presented the best result for 6 of the seven datasets, though it achieved the same results of LARFDSSOM for Breast and Glass datasets. Also, ALTSS-SOM was the second best for Diabetes, what shows a similar behavior when compared with the results obtained in the Section 6.10 which took into account the classification rate. On considering a general comparison, ALTSS-SOM present the best results on average. It is worth mentioning the similarity between the results of ALTSS-SOM and LARFDSSOM can be attributed to the fact that there were no labeled noise samples in the datasets, to the unknown information about the irrelevant dimensions, and to the intrinsic characteristics inherited by ALTSS-SOM from the LARFDSSOM. Also, once DOC does not have a direct way to control the number of clusters, it presented difficulties to find out the correct value.

Moreover, PROCLUS presented good results when the parameter controlling the number of clusters was defined close to the correct value. The good results obtained by LARFDSSOM is directly related to a good choice of the parameters a_t and lp, which significantly impact the results. ALTSS-SOM achieves good results without the needing of very precise definition of the parameter values, what is more deeply evaluated in the next section.

Table 8: CE Results for Real-World Datasets

CE	Breast	Diabetes	Glass	Liver	Pendigits	Shape	Vowel	Avg	STD
DOC	**0.763** (1)	0.654	0.439	0.580	0.566	0.419	0.142	0.509	0.201
PROCLUS	0.702 (2)	0.647 (2)	0.528 (2)	0.565	0.615	0.706	0.253	0.574	0.156
LARFDSSOM/SS-SOM	**0.763** (1)	**0.727** (1)	**0.575** (1)	0.580 (2)	0.737 (2)	0.719 (2)	0.317 (2)	0.631	0.158
ALTSS-SOM	**0.763** (1)	0.697 (2)	**0.575** (1)	**0.603** (1)	**0.741** (1)	**0.738** (1)	**0.319** (1)	**0.633**	0.156

6.12 ALTSS-SOM: SENSITIVITY ANALYSIS

In previous methods, the most two critical parameters were the a_t and lp, while a_t played a role of great importance due to its high impact on the results with just a small change on its values, i.e., a_t impacted the results exponentially. Because of that, a sensitivity analysis was also performed for ALTSS-SOM in order to elucidate the improvements over the previous versions. Figure 21 shows the results. Notice that there are no significant trends to parameter values, and they do not damage the performance with slight changes in its values. This same behavior was observed for all the other datasets used in this set of experiments.

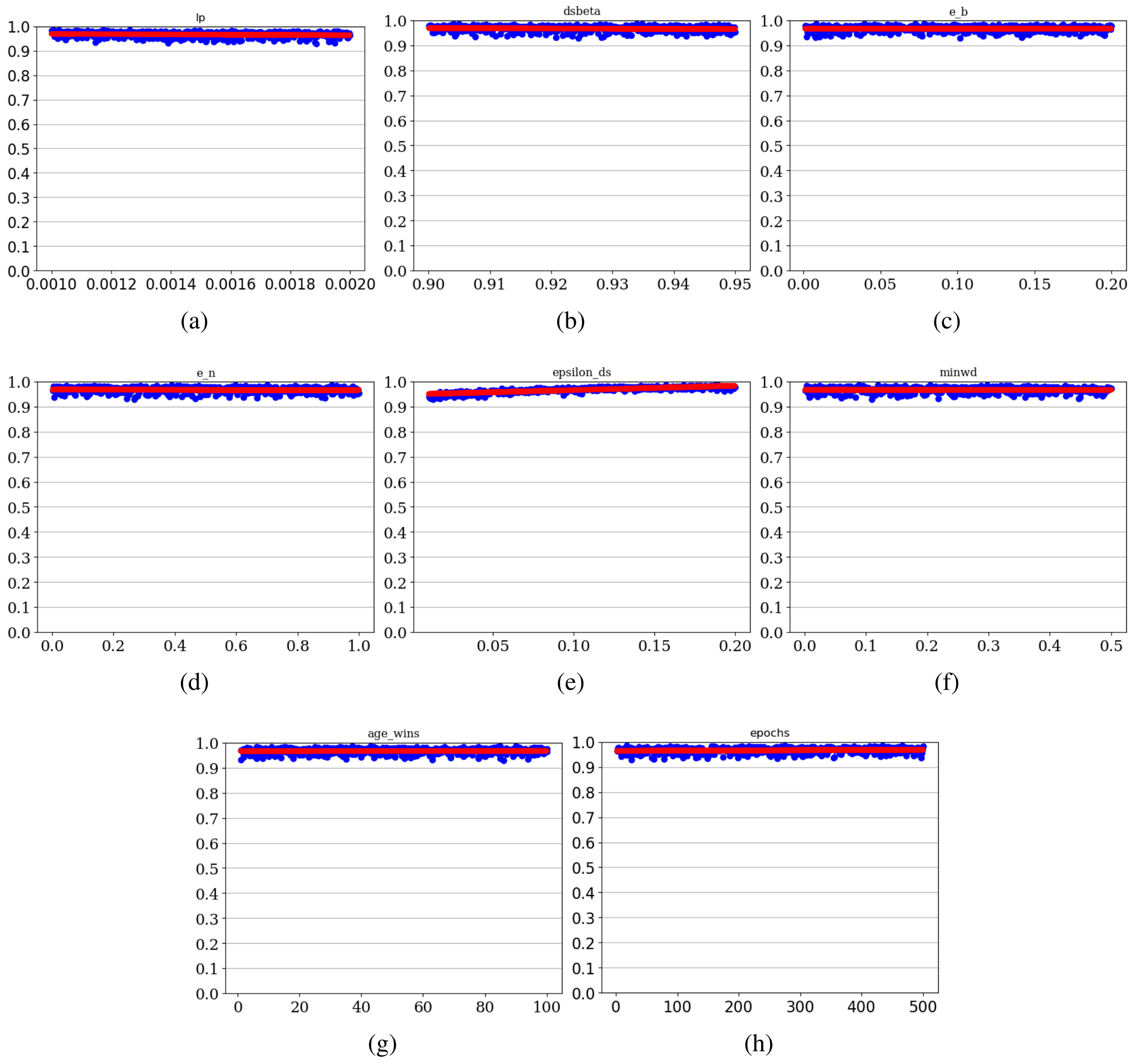
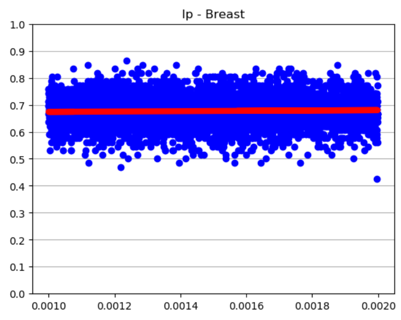


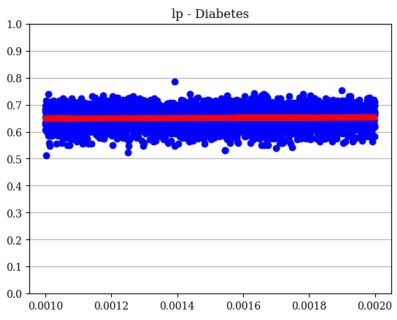
Figure 21: Sensitivity analysis with a scatter plot of the Accuracy obtained with ALTSS-SOM as a function of its parameters, (a) lp, (b) dsbeta, (c) e_b, (d) e_n, (e) epsilon_ds, (f) minwd, (g) age_wins, and (h) epochs, for Pendigits real-world dataset using 50% of the available labels, and 1 fold randomly chosen from the 3-times 3-fold cross validation scheme. The red lines are the linear fits to the data.

Despite of that, the parameters lp and e_b were chosen due to its semantical importance to carry out a more detailed analysis aiming at showing that with ALTSS-SOM the same range of parameters work well for a variety of datasets and that such range does not exist for the previous methods (LARFDSSOM and SS-SOM). Figure 22 shows the scatter plots of ALTSS-SOM accuracy as a function of the parameter *lp* for the datasets trained with 50% of labels to illustrate a scenario where both forms of learning impact the outcome. Note that for all datasets, *lp* did not show a significant impact on the results. It is also worth mentioning that the plots in Figure 22 are the combination of each parameter value for each cross-validation set. Here in ALTSS-SOM, the *lp* is the most important parameter because it defines more clearly the behavior of the algorithm, however, the results show that its impact is not very significant inside the specified ranges (i.e.

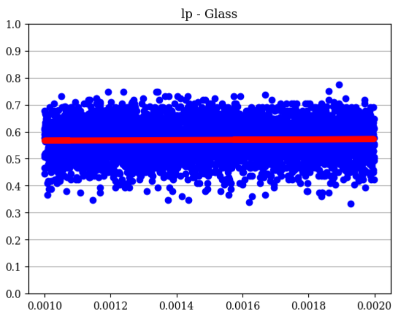
below 0.002).



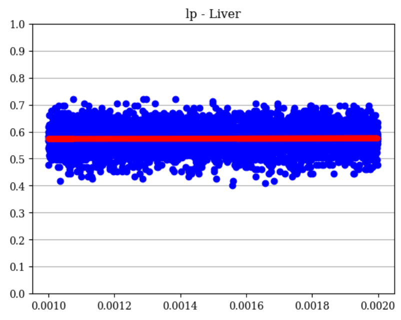
(a)



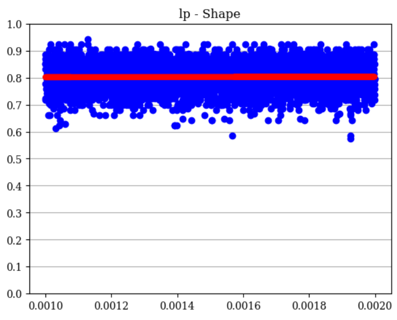
(b)



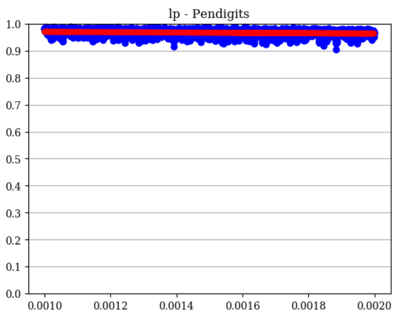
(c)



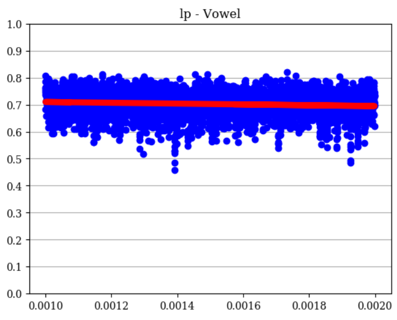
(d)



(e)



(f)



(g)

Figure 22: Scatter plots of the Accuracy obtained with ALTSS-SOM as a function of its parameter *lp* on (a) Breast, (b) Diabetes, (c) Glass, (d) Liver, (e) Shape, (f) Pendigits, and (g) Vowel datasets trained with 50% of labels. The red lines are the linear fits to the data.

Figure 23 shows the scatter plots of the parameter e_b. This time we chose the datasets Breast, Vowel and Pendigits to illustrate the impact of parameters. We first started with a wide range from 0.001 to 0.4 (Figure 23a to Figure 23c), however, the linear fit was mostly horizontal, not indicating a trend, again. We then shrank the range to 0.001 to 0.2 and reran the experiments. The results were as expected, keeping stable as shown by Figure 23d to Figure 23f. These experiments clarify how the model is robust to parameter changes. The parameters *lp* and e_b were taken for study due to their semantic importance, since other parameters presented similar behavior, with none of them acting a role as a_t and *lp* in the before-mentioned versions.

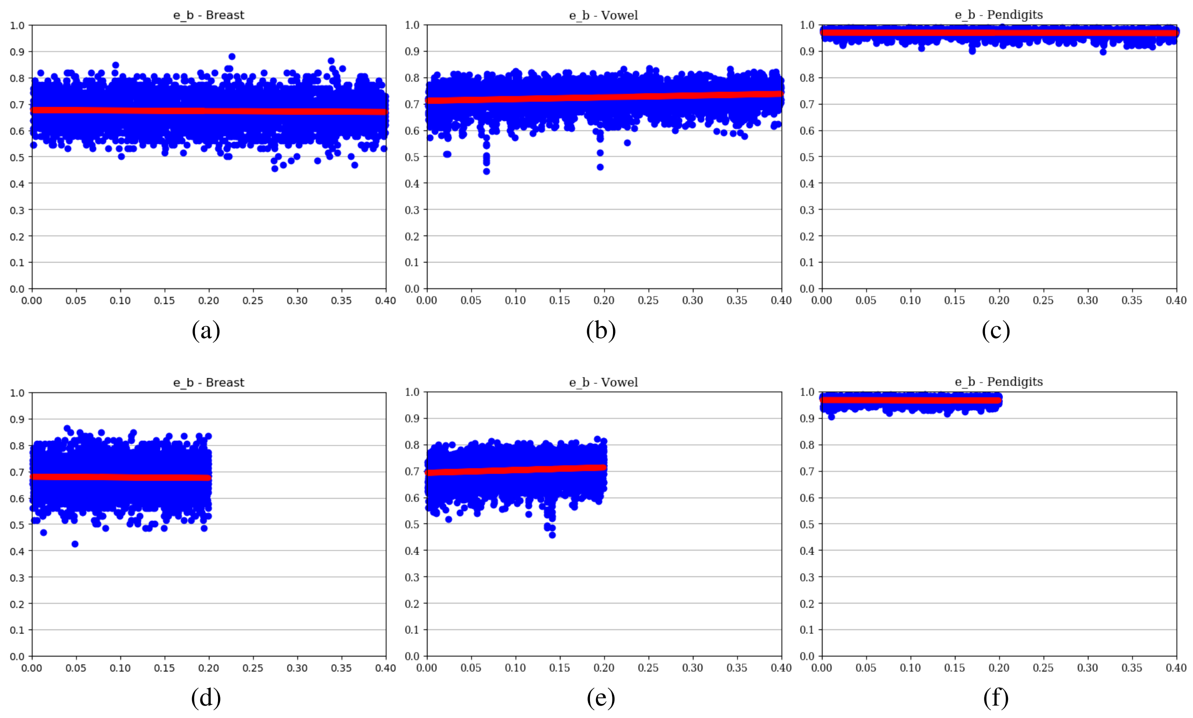


Figure 23: Scatter plots of the Accuracy obtained with ALTSS-SOM as a function of its parameter e_b on (a) Breast, (b) Diabetes, (c) Glass, (d) Liver, (e) Shape, (f) Pendigits, and (g) Vowel datasets trained with 50% of labels to illustrate how the parameter ranges were defined. The red lines are the linear fits to the data.

6.13 SS-SOM AND ALTSS-SOM: FULLY SUPERVISED PERFORMANCES

To draw a better understanding of the behavior of both SS-SOM and ALTSS-SOM, they were valuated in the scenario of a regular supervision learning task. It is not expected that they go well due to the fact that they were not built for such a scenario.

Still, their capabilities are pushed in order to do a comparison with traditional supervised methods such as Multilayer Perceptron (MLP) (Haykin, 2009), Support Vector Machine (SVM) (Cortes & Vapnik, 1995), and Generalized Relevance Learning Vector Quantization (Hammer & Villmann, 2002). The ranges used for those methods are given from Table 9 to Table 11. SS-SOM, ALTSS-SOM, LP, and LS were ran with the ranges defined in Table 3, Table 6, and Table 4 from the previous sections.

Table 9: Parameter Ranges for MLP

Parameters	**min**	**max**
Number of neurons in each layer	1	100
Number of hidden layers	1	3
Learning rate	0.001	0.1
Momentum	0.85	0.95
Epochs	100	200
Optimizer[1]	1	3
Activation function[2]	1	3
Learning Decay[3]	1	3

[1] 1: lbfgs; 2: sgd; 3: adam; [2] 1: logistic; 2: tanh; 3: relu;
[3] 1: constant; 2: invscaling; 3: adaptative.

Table 10: Parameter Ranges for GRLVQ

Parameters	**min**	**max**
Number of nodes	10	30
Positive learning rate	0.4	0.5
Negative learning rate	0.01	0.05
Weights learning rate	0.15	0.2
Learning Decay	0.000001	0.00002
Number of epochs	5000	10000

Table 11: Parameter Ranges for SVM

Parameters	**min**	**max**
C	0.1	10
Kernel Function[1]	1	4
Degree of polynomial kernel function	3	5
Gamma of kernel functions 2, 3 and 4	0.1	1
Independent term in kernel functions 2 and 3	0.01	1

[1] 1: linear, 2: poly, 3: rbf and 4: sigmoid.

Table 12 shows the results of SS-SOM, ALTSS-SOM and other semi-supervised methods using 100% of the labeled data, allowing a comparison with supervised methods such as GRLVQ, MLP, and SVM. Notice that when there are more than one global best for a particular dataset, it means that the results have no statistical differences.

Table 12: Accuracy Results for Real-World Datasets with 100% of the labeled data

Accuracy	Breast	Diabetes	Glass	Liver	Pendigits	Shape	Vowel	Avg
SS-SOM	0.816 (0.036)	**0.776 (0.012)**	0.723 (0.053)	**0.740 (0.020)**	0.975 (0.004)	0.915 (0.023)	0.834 (0.017)	0.826
ALTSS-SOM	**0.828 (0.034)**	0.745 (0.014)	**0.748 (0.029)**	0.735 (0.029)	0.990 (0.002)	**0.934 (0.038)**	0.900 (0.017)	**0.840**
LP	0.806 (0.041)	0.762 (0.022)	0.696 (0.049)	0.697 (0.053)	**0.994 (0.001)**	0.915 (0.050)	**0.945 (0.017)**	0.831
LS	0.811 (0.044)	0.763 (0.025)	0.695 (0.045)	0.689 (0.054)	**0.994 (0.001)**	0.915 (0.050)	**0.945 (0.017)**	0.830
MLP	**0.848 (0.031)**	**0.794 (0.017)**	**0.741 (0.023)**	**0.767 (0.034)**	0.993 (0.002)	**0.934 (0.037)**	0.883 (0.026)	**0.851**
SVM	0.832 (0.029)	0.787 (0.021)	0.721 (0.048)	0.746 (0.035)	**0.997 (0.001)**	0.932 (0.046)	**0.904 (0.015)**	0.846
GRLVQ	0.830 (0.029)	0.769 (0.026)	0.682 (0.035)	0.694 (0.034)	0.915 (0.006)	0.825 (0.044)	0.514 (0.024)	0.747

In bold, the best results for each dataset on each category: semi-supervised and supervised methods. The underlined results indicate the global best (more than one global best means that there is no statistical difference between the underlined results).

The proposed models show a comparable performance with the other semi-supervised methods, where the most significant difference is for Vowel. Also, SS-SOM appears as the best overall among the semi-supervised methods (the first four in the table), for the Diabetes and Liver datasets. Still, concerning only semi-supervised models, ALTSS-SOM presented the best results for Breast, Glass and Shape, while Label Propagation and Label Spreading are the best for Pendigits and Vowel. Among the supervised models (the last three in the table), MLP is undoubtedly the best.

On considering all methods at 100% of supervision, the MLP is the best on average in three of seven datasets. However, in two of them (Diabetes and Liver), there are no statistical differences between SS-SOM and MLP, as well as between ALTSS-SOM and MLP for the Breast datasets. The ALTSS-SOM is the best on average for Glass. However, there is still no difference for the second best, the MLP. Moreover, ALTSS-SOM and MLP are tied on average for the Shape dataset. Moreover, LP and LS are the best ones for Vowel, and in the end, there is no difference at all between the obtained results by all models on Pendigits, except the GRLVQ.

The two proposed methods showed results better than or at least close to the best found in comparison with others supervised and semi-supervised methods, even with it not being the primary objective of this work.

6.14 SUMMARIZING THE CONDUCTED EXPERIMENTS

By the understanding that the experiments may have become too dense and that the reader may have lost the tracking about the obtained results, a summary was built in Table 13 to show the big picture of what was accomplished.

Table 13: Experimental Summary

Experiment	**Result**
SS-SOM Semi-Supervised Capabilities (Section 6.6)	Showed to be the best on average in comparison with traditional approaches.
SS-SOM Sensitivity Analysis (Section 6.7)	Demonstrated a high dependency and sensibility to the parameter a_t.
SS-SOM with Transfer Learning (Section 6.8)	Surprisingly good results were obtained. It opens a new promising path of SOM-based models applications.
Batch SS-SOM Results (Section 6.9)	It showed to be good, however, needs to be better explored and enhanced with newer techniques from the Deep Learning area.
ALTSS-SOM Semi-Supervised Capabilities (Section 6.10)	It improved the results of SS-SOM. Its robustness was shown to have led to significant gains regarding the previous model.
ALTSS-SOM Clustering Quality (Section 6.11)	The changes made in the framework provided the chance for the ALTSS-SOM to improve or at least maintain the clustering quality in comparison with other benchmarks.
ALTSS-SOM Sensitivity Analysis (Section 6.12)	ALTSS-SOM reduced the dependency and sensibility to the parameters drastically. It showed to became steady even though with significant changes in the parameters values.
SS-SOM and ALTSS-SOM Fully Supervised (Section 6.13)	Both methods presented good results, including being better in some cases, despite the fact they were not built for that.

7 FINAL CONSIDERATIONS

In this chapter the final analysis of the proposed models, the main contributions to the science, the published papers, the limitations found in the models, and finally, future works and possible applications are presented.

7.1 ANALYSIS OF THE PROPOSED MODELS

Chapter 1 and Chapter 2 introduced several problems and challenges that are emerging as technology advances. More precisely, the lack of labels in the midst of the great volume of information that has been produced every day is still a problem for a significant number of machine learning models. This Dissertation was intended to take another step attaining to build more sophisticated solutions for such a problem by trying to accomplish the objectives pointed in Chapter 1.

The main objective was to develop Semi-Supervised models based on the concepts of both SOM and LVQ to improve the results obtained with traditional SSL methods in the literature. It was achieved by proposing and validating SS-SOM and ALTSS-SOM. They both carry elements of supervision inspired by LVQ that were introduced to a SOM-based framework. The behavior of SS-SOM was shown to have led to significant improvements in classification results for small amounts of labeled data, establishing its position as a good option when dealing with such problems. It showed its robustness under this condition, being better than other semi-supervised models, achieving impressive results even with only 1% of labeled data, in comparison with other SSL methdos. ALTSS-SOM was the second approach for semi-supervised self-organizing maps applied to cluster and classification tasks. The behavior of ALTSS-SOM showed advances in comparison with SS-SOM. It consolidates a position of a good choice in situations where only a small portion of labels are available. The experiments conducted in Section 6.6 and Section 6.10 endorse such conclusion.

The specific objectives concern four distinct points. The first was related to extending the application range of the proposed models. To do so, the SS-SOM was tested with extracted features by performing a transfer learning, that allowed a SSL prototype-based method such as SS-SOM, that is not suitable for working with data as complex as raw images or audio, to perform well and extend its application range, even in such conditions that it was not built to deal with, as can be seen in Section 6.8. It is also worth mentioning that because of computational costs, we were not able to reproduce more recent techniques that are harder to beat. Furthermore, Batch SS-SOM, a novel approach that allows a mini-batch training procedure for the traditional shallow architecture of a SOM, was proposed. The results were surprisingly good. It is true that a great range of techniques that may benefit the performance of Batch SS-SOM exist. However, the first version established a good baseline for future development and the experiments conducted in Section 6.9 defines it as a promising path to follow.

The second one intended to improve not only the classification rate of the models but also the clustering quality when there is no label available. It was fulfilled with ALTSS-SOM, that showed improved results not only regarding classification rate but also in clustering aspects.

The third relies on the development of a strategy able to estimate local rejection options as a function of both local variance and the relevance of input dimensions to make pattern rejection decisions. It is the central point behind the ideas of ALTSS-SOM, that was explained in Chapter 5.

The last aimed at reducing the dependency and the variability of models to some parameters. Such behavior was reached with ALTSS-SOM, as demonstrated in Section 6.7 and Section 6.12. This parametric robustness can be considered as one of the most important contributions of this current work.

Moreover, it is important mentioning some additional considerations. Both SS-SOM and ALTSS-SOM were put on trial in a scenario of full supervision by comparing their results with state-of-the-art methods developed specifically for supervised learning. Even though not being built for that, both models presented good results, being better than or at least close to the best methods. Additionally, ALTSS-SOM was able to reduce in two the number of parameters whereby improving the performance in both contexts of classification and clustering metrics.

Finally, the usage of a relaxed estimated variance allowed the method to explore the local information of the data clusters better. Also, the modifications proposed in ALTSS-SOM provided the ability to improve its sample efficiency by not merely discarding data in certain cases but kept digging into its characteristics in order to establish a better understanding of their statistics. This can be summarized in the idea of developing models that can exploit to the utmost the information carried in the data, either for generating prototypes or for adjusting their receptive fields, but also being able to recognize and disregard outliers when necessary,

7.2 CONTRIBUTIONS TO SCIENCE

The research carried out in the development of this Dissertation resulted in a set of contributions to the correlated areas in the science:

The SS-SOM and ALTSS-SOM models were shown to be promising for classification tasks even when only a few amounts of data is available. Also, they both can switch between different forms of learning, being able to perform semi-supervised classification and clustering, but also each of these tasks in an isolated form when submitted to such conditions.

In particular, the results obtained with SS-SOM not only showed its capabilities for traditional classification tasks but also to more complex applications with the use of transfer learning techniques in more challenging data. Moreover, its extension for GPU and mini-batch training, Batch SS-SOM, may open a good path to be explored. It also provides more integrability of the model with other Deep Learning approaches, that commonly use the same framework and structure, allowing the usage of the model as an intermediate layer in next versions, for example.

Lastly, ALTSS-SOM showed to be very effective for clustering tasks solely likewise for classification. Therefore, this set of propositions is the most important contribution of this work.

7.2.1 Published Papers

During the development of this dissertation, the article related to the model SS-SOM, was published in the 2018 International Joint Conference on Neural Networks (IJCNN), at the 2018 Institute of Electrical and Electronics Engineers (IEEE) World Congress on Computational Intelligence (WCCI).

- P. H. Braga and H. F. Bassani, "A semi-supervised self-organizing map for clustering and classification," in 2018 International Joint Conference on Neural Networks (IJCNN). IEEE, 2018, pp. 1–8.

Also, another paper related to the model ALTSS-SOM was accepted for publication in the 2019 IJCNN.

- P. H. Braga and H. F. Bassani, "A Semi-Supervised Self-Organizing Map with Adaptive Local Thresholds," in 2019 International Joint Conference on Neural Networks (IJCNN). IEEE, 2019.

7.3 LIMITATIONS OF THE MODELS

Despite the efforts made in proposing the models, some points have not yet been adequately addressed, and need to be taken into consideration in the future.

First, the experiments conducted on this Dissertation concerned the same labels for both training and testing (for classification tasks). Therefore, it may be necessary to extend the studies to evaluate how the models behave in scenarios where more generalization power is needed, such as in Robotics, where new categories of elements may frequently arise.

Second, the ALTSS-SOM was not stressed as much to verify its full subspace clustering capabilities, once the experiments with simulated data were not conducted on time. For example, it was not verified if ALTSS-SOM is truly robust to noise. However, it showed to be good in a great variety of applications.

Finally, even though with some optimizations performed to provide a more efficient computation of the operations, it is necessary to analyze different forms to allow SOM-based models to make use of the great computational power and parallelism provided by GPU devices. It must be considered to work around the sequential operations necessary for SOM-based models to work, such as the nodes update for each input pattern when training in batch, which make some applications unfeasible.

7.4 FUTURE WORK

Because this work comprises several proposals, a large amount of possibilities arises. However, it is possible to briefly compile each of the following points to be explored in a close future:

- Develop an adequate stop criterion to reduce the training time of all proposed models.
- Use the unsupervised error to build a model with more than one layer.
- Define a hierarchical approach to provide better exploitation of the data statistics and to increase the abstraction level autonomously.
- The connection between the nodes can take into account the proximity and not only its subspaces and classes.
- Adjust the models for online learning. To achieve this, it is necessary to change the node removal to consider cases of not observing data in some clusters for a given time
- With a small change, SS-SOM, ALTSS-SOM and consequently the Batch SS-SOM can incorporate reinforcement learning, thus being capable of switching between three different learning approaches, to exploit several forms of information available.
- Explore in more detail the transfer learning techniques with the developed models, especially the ALTSS-SOM that was not taken into account in such application.
- Optimize the Batch SS-SOM with more recent deep learning techniques to achieve better results.
- Study the computational complexity of the models and propose proper optimizations.

REFERENCES

Aggarwal, C. C., Wolf, J. L., Yu, P. S., Procopiuc, C., & Park, J. S. (1999). Fast algorithms for projected clustering. In *ACM SIGMoD Record*, 28(2):61–72.

Araujo, A. F. & Rego, R. L. (2013). Self-organizing maps with a time-varying structure. *ACM Computing Surveys*, 46(1):7.

Araujo, F. R., Bassani, H. F., & Araujo, A. F. (2013). Learning vector quantization with local adaptive weighting for relevance determination in genome-wide association studies. In *Proceedings of the 2013 International Joint Conference on Neural Networks (IJCNN)*, 1–8.

Asuncion, A. & Newman, D. (2007). Uci machine learning repository.

Bassani, H. F. (2014). *Modelos neurais modulares para aquisição de linguagem natural*. PhD thesis, Universidade Federal de Pernambuco.

Bassani, H. F. & Araújo, A. F. (2012). Dimension selective self-organizing maps for clustering high dimensional data. In *Proceedings of the 2012 International Joint Conference on Neural Networks (IJCNN)*, 1–8.

Bassani, H. F. & Araujo, A. F. R. (2015). Dimension Selective Self-Organizing Maps With Time-Varying Structure for Subspace and Projected Clustering. *IEEE Transactions on Neural Networks and Learning Systems*, 26(3):458–471.

Basu, S., Banerjee, A., & Mooney, R. (2002). Semi-supervised clustering by seeding. In *Proceedings of 19th International Conference on Machine Learning*.

Bay, H., Tuytelaars, T., & Van Gool, L. (2006). Surf: Speeded up robust features. In *European Conference on Computer Vision*, 404–417.

Chapelle, O., Scholkopf, B., & Zien, A. (2009). Semi-supervised learning. *IEEE Transactions on Neural Networks*, 20(3):542–542.

Chen, L., Yu, S., & Yang, M. (2018). Semi-supervised convolutional neural networks with label propagation for image classification. In *2018 24th International Conference on Pattern Recognition (ICPR)*, 1319–1324.

Chow, C. (1970). On optimum recognition error and reject tradeoff. *IEEE Transactions on Information Theory*, 16(1):41–46.

Chow, C. & Kaneko, T. (1972). Automatic boundary detection of the left ventricle from cineangiograms. *Computers and Biomedical Research*, 5(4):388–410.

Cortes, C. & Vapnik, V. (1995). Support vector machine. *Machine Learning*, 20(3):273–297.

De Araujo, F. R. B. & Guimaraes, K. S. (2016). Inference of high-order epistatic interactions using generalized relevance learning vector quantization with parametric adjustment. In *Proceedings of the 2016 International Conference on Tools with Artificial Intelligence (ICTAI)*, 648–654.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 Computer Vision and Pattern Recognition (CVPR)*, 248–255.

Dozono, H., Niina, G., & Araki, S. (2016). Convolutional self organizing map. In *Proceedings of the 2016 International Conference on Computational Science and Computational Intelligence (CSCI)*, 767–771.

Fischer, L., Hammer, B., & Wersing, H. (2014). Rejection strategies for learning vector quantization. In *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*.

Fischer, L., Hammer, B., & Wersing, H. (2016). Optimal local rejection for classifiers. *Neurocomputing*, 214:445–457.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Goodfellow, I., Courville, A., & Bengio, Y. (2012). Large-scale feature learning with spike-and-slab sparse coding. *arXiv preprint arXiv:1206.6407*.

Hailat, Z., Komarichev, A., & Chen, X. (2018). Deep semi-supervised learning. In *2018 24th International Conference on Pattern Recognition (ICPR)*, 2154–2159.

Hammer, B. & Villmann, T. (2002). Generalized relevance learning vector quantization. *Neural Networks*, 15(8-9):1059–1068.

Haykin, S. (2009). *Neural Networks and Learning Machines*. Prentice-Hall, third edition.

Helton, J. C., Davis, F., & Johnson, J. D. (2005). A comparison of uncertainty and sensitivity analysis results obtained with random and latin hypercube sampling. *Reliability Engineering & System Safety*, 89(3):305–330.

Herrmann, L. & Ultsch, A. (2007). Label propagation for semi-supervised learning in self-organizing maps. *Proceedings of the 6th WSOM*.

Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2261–2269.

Hubel, D. H. & Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1):106–154.

Hui, K. Y. (2013). Direct modeling of complex invariances for visual object features. In *International Conference on Machine Learning*, 352–360.

Iman, R. L. & Helton, J. C. (1988). An investigation of uncertainty and sensitivity analysis techniques for computer models. *Risk Analysis*, 8(1):71–90.

Ioffe, S. & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint:1502.03167*.

Jain, A. K. (2010). Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651–666.

Jiang, X. & Mojon, D. (2003). Adaptive local thresholding by verification-based multithreshold probing with application to vessel detection in retinal images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(1):131–137.

Jindal, I., Nokleby, M., & Chen, X. (2016). Learning deep networks from noisy labels with dropout regularization. In *Proceedings of the 16th IEEE International Conference on Data Mining (ICDM)*, 967–972.

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., & Saul, L. K. (1999). An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233.

Kaas, J. H., Merzenich, M. M., & Killackey, H. P. (1983). The reorganization of somatosensory cortex following peripheral nerve damage in adult and developing mammals. *Annual Review of Neuroscience*, 6(1):325–356.

Kangas, J. A., Kohonen, T. K., & Laaksonen, J. T. (1990). Variants of self-organizing maps. *IEEE Transactions on Neural Networks*, 1(1):93–99.

Kingma, D. P. & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P., Mohamed, S., Rezende, D. J., & Welling, M. (2014). Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, 3581–3589.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69.

Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480.

Kohonen, T. (1995). Learning vector quantization. In *Self-Organizing Maps*, 175–189. Springer.

Köppen, M. (2000). The curse of dimensionality. In *5th Online World Conference on Soft Computing in Industrial Applications*, 4–8.

Kriegel, H.-P., Kröger, P., & Zimek, A. (2009). Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 3(1):1.

Krizhevsky, A. & Hinton, G. (2009). Learning multiple layers of features from tiny images. Technical report, Citeseer.

Kunze, M. & Steffens, J. (1995). Growing cell structure and neural gas. In *Proceedings of the 4th AIHEP Workshop, Pisa (World Scientific)*.

Laine, S. & Aila, T. (2016). Temporal ensembling for semi-supervised learning. *arXiv preprint arXiv:1610.02242*.

Landgrebe, T. C., Tax, D. M., Paclík, P., & Duin, R. P. (2006). The interaction between classification and reject performance for distance-based reject-option classifiers. *Pattern Recognition Letters*, 27(8):908–917.

LeCun, Y. (1998). The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436.

Levine, S., Pastor, P., Krizhevsky, A., & Quillen, D. (2016). Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *CoRR*.

Liang, M. & Hu, X. (2015). Recurrent convolutional neural network for object recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 3367–3375.

Liu, N., Wang, J., & Gong, Y. (2015). Deep self-organizing map for visual classification. In *Proceedings of the 2015 International Joint Conference on Neural Networks (IJCNN)*, 1–6.

Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Proceedings of the 7th IEEE International Conference on Computer vision*, 2:1150–1157.

Maaten, L. v. d. & Hinton, G. (2008). Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605.

Marsland, S., Shapiro, J., & Nehmzow, U. (2002). A self-organising network that grows when required. *Neural Networks*, 15(8-9):1041–1058.

McKay, M. D., Beckman, R. J., & Conover, W. J. (1979). Comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245.

Medeiros, H. R., de Oliveira, F. D., Bassani, H. F., & Araujo, A. F. (2018). Dynamic topology and relevance learning SOM-based algorithm for image clustering tasks. *Computer Vision and Image Understanding*.

Miikkulainen, R., Bednar, J. A., Choe, Y., & Sirosh, J. (2006). *Computational maps in the visual cortex*. Springer Science & Business Media.

Miyato, T., Maeda, S.-i., Ishii, S., & Koyama, M. (2018). Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Müller, E., Günnemann, S., Assent, I., & Seidl, T. (2009). Evaluating clustering in subspace projections of high dimensional data. *Proceedings of the VLDB Endowment*, 2(1):1270–1281.

Nair, V. & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 807–814.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., & Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011(2):5.

Nova, D. & Estévez, P. A. (2014). A review of learning vector quantization classifiers. *Neural Computing and Applications*, 25(3-4):511–524.

Oliver, A., Odena, A., Raffel, C., Cubuk, E. D., & Goodfellow, I. J. (2018). Realistic evaluation of deep semi-supervised learning algorithms. *arXiv preprint arXiv:1804.09170*.

Parsons, L., Haque, E., & Liu, H. (2004). Subspace clustering for high dimensional data: a review. *ACM SIGKDD Explorations Newsletter*, 6(1):90–105.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., & Lerer, A. (2017). Automatic differentiation in pytorch. In *NIPS-W*.

Patrikainen, A. & Meila, M. (2006). Comparing subspace clusterings. *IEEE Transactions on Knowledge and Data Engineering*, 18(7):902–916.

Peterson, C. (1987). A mean field theory learning algorithm for neural networks. *Complex Systems*, 1:995–1019.

Phansalkar, N., More, S., Sabale, A., & Joshi, M. (2011). Adaptive local thresholding for detection of nuclei in diversity stained cytology images. In *Proceedings of the 2011 International Conference on Communications and Signal Processing (ICCSP)*, 218–220.

Procopiuc, C. M., Jones, M., Agarwal, P. K., & Murali, T. (2002). A monte carlo algorithm for fast projective clustering. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, 418–427.

Pudil, P. & Novovičová, J. (1998). Novel methods for feature subset selection with respect to problem knowledge. In *Feature Extraction, Construction and Selection*, 101–116. Springer.

Rasmus, A., Berglund, M., Honkala, M., Valpola, H., & Raiko, T. (2015). Semi-supervised learning with ladder networks. In *Advances in Neural Information Processing Systems*, 3546–3554.

Saltelli, A., Chan, K., Scott, E. M., *et al.* (2000). *Sensitivity analysis*, volume 1. Wiley New York.

Sato, A. & Yamada, K. (1996). Generalized learning vector quantization. In *Advances in neural information processing systems*, 423–429.

Schwenker, F. & Trentin, E. (2014). Pattern classification and clustering: A review of partially supervised learning approaches. *Pattern Recognition Letters*, 37:4–14.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Suga, N. (1990). Biosonar and neural computation in bats. *Scientific American*, 262(6):60–71.

Szummer, M. & Jaakkola, T. (2002). Partially labeled classification with markov random walks. In *Advances in Neural Information Processing Systems*, 945–952.

Vailaya, A. & Jain, A. (2000). Reject option for VQ-based Bayesian classification. In *Proceedings of ICPR*, 2048.

Vidal, R. (2011). Subspace clustering. *IEEE Signal Processing Magazine*, 28(2):52–68.

Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., & de Freitas, N. (2016). Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83.

Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

Xiaojin, Z. & Zoubin, G. (2002). Learning from labeled and unlabeled data with label propagation. Technical report, Carnegie Mellon University.

Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems*, 3320–3328.

Zhou, D., Bousquet, O., Lal, T. N., Weston, J., & Schölkopf, B. (2004). Learning with local and global consistency. In *Advances in Neural Information Processing Systems*, 321–328.

Zhou, J., Cao, Y., Wang, X., Li, P., & Xu, W. (2016). Deep recurrent models with fast-forward connections for neural machine translation. *CoRR*.

Zhu, X. (2006). Semi-supervised learning literature survey. Technical report, Department of Computer Science, University of Wisconsin-Madison.

Zhu, X., Ghahramani, Z., & Lafferty, J. D. (2003). Semi-supervised learning using Gaussian fields and harmonic functions. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 912–919.