UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE CIENCIAS MATEMÁTICAS

TESIS DOCTORAL

Desarrollo de sistemas expertos con programación funcional y metodología Big Data

(Expert System Development Using Functional Programming and Big Data Methodologies)

MEMORIA PARA OPTAR AL GRADO DE DOCTOR

PRESENTADA POR

Gabriel Antonio Valverde Castilla

DIRECTORES

Beatriz González Pérez
José Manuel Mira McWilliams

UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE CIENCIAS MATEMÁTICAS
Departamento de Estadística e Investigación Operativa

DOCTORADO EN
Ingeniería Matemática, Estadística e Investigación Operativa (IMEIO)

Tesis Doctoral.

Desarrollo de sistemas expertos con programación funcional y metodología Big Data

(Expert System Development Using Functional Programming and Big Data Methodologies)

MEMORIA PARA OPTAR AL GRADO DE DOCTOR
PRESENTADA POR:

Gabriel Antonio Valverde Castilla

Directores:

Beatriz José Manuel
González Pérez Mira McWilliams

Madrid Año 2022

A mis abuelas que se dejaron la vida trabajando.

Agradecimientos

"El éxito no es la clave de la felicidad. La felicidad es la clave del éxito. Si amas lo que haces, tendrás éxito". Esta cita de Albert Schweitzer resume la filosofía que ha guiado mi tesis. He invertido muchos años en descubrir que lo más importante era disfrutar del proceso y que la constancia se derivaría automáticamente de hacerlo. Por ello, quiero aprovechar estas palabras para agradecer a todas las personas que han formado parte de mi camino y que, de alguna manera, me han ayudado a encontrar la felicidad y el éxito en este logro.

Esta tesis no solo representa un trabajo de investigación, sino también un proceso personal de aprendizaje y madurez. En ella he aprendido no solo a tolerar, sino también a aceptar y disfrutar la incertidumbre, tanto en el ámbito analítico como en mi vida personal. Y todo esto ha sido posible gracias al apoyo incondicional de familiares, amigos y mentores.

Mi familia ha sido la piedra angular de mi vida y la principal fuerza detrás de mis logros. A mi padre, Gabriel, y a mi madre, Granada, les debo mi amor por el conocimiento y la educación. Gracias por inspirarme a perseguir mis metas y por su inquebrantable apoyo emocional. También a mi hermano, Álvaro, por su amistad, su confianza y por ser mi cómplice en todas mis aventuras.

No puedo dejar de agradecerte Amanda, mi guía incansable y mi apoyo constante, por tu amor, comprensión y por motivarme a ser una mejor versión de mí mismo. Gracias por enseñarme a valorar todo lo que tengo y a liberarme para disfrutar plenamente de la vida. Gracias también por tu infinita paciencia en los momentos en los que la tesis ha ocupado parte de nuestro tiempo.

Mis amigos y amigas han sido una fuente constante de apoyo y motivación en todo momento. En particular, quiero agradecer a Javier, Jorge, Carlos, David, quienes han compartido conmigo su entusiasmo por la investigación, habeís sido grandes amigos y compañeros en el camino del doctorado. Al trío BK: Manuel, Nacho, Elia por demostrarme que podía hacerlo. También a Ana, Rocío, Teresa y Ainara, quienes han estado siempre presentes y han sido una gran ayuda emocional en los momentos difíciles. Especialmente a ti Ainara, juntos hemos dado sentido a esta ciudad a cada pasito. A Carmen y

Bea, quienes me han dado refugio en Madrid y han sido una fuente constante de alegría y compañía. Gracias al núcleo extremeño en Madrid, Gloria, Mari Carmen y Jose, sin nuestros momentos de compartir acento y raíces tampoco habría sido posible. A los que siempre están, Miguel, Jesús y Zhen, todos mis amigos y amigas por su presencia en mi vida y por hacer de cada momento algo especial, incluso cuando mi mente tendía a pensar en tesis.

A mis directores de tesis. José Manuel Mira McWilliams, siempre comprometido con este proyecto, con infinita paciencia y cariño. Beatriz González, me has transmitido tu pasión por este oficio y por el estudio de la incertidumbre que ahora siempre tengo presente. A mi tutora, Begoña Vitoriano, me diste la primera oportunidad, aquel voto de confianza con la beca de colaboración, y no has dejado de apoyarme en este proceso. A los tres les agradezco su orientación y su ayuda constante en la elaboración de este trabajo. Sin su experiencia y dedicación, esta tesis no habría sido posible.

Quiero agradecer especialmente a aquellas personas que me dieron la oportunidad de iniciar mi carrera profesional y creyeron en mí. Gracias a Victoria, Miguel, Fede por confiar en mí y darme la oportunidad de trabajar a su lado. También quiero agradecer a Ricardo y Juanma, por su guía y apoyo en el inicio de mi camino académico y profesional. Sin ellos, todo esto no habría sido posible

Agradezco de manera especial a mis primeros alumnos Ana, Arturo y Elena quienes ahora son amigos y han sido testigos de cada uno de mis pasos. Gracias por confiar en mí y por ser parte de mi crecimiento personal y profesional. Vuestras experiencias y testimonios han sido una fuente de inspiración y motivación constante para seguir adelante.

Por último, pero no menos importante, quiero agradecer a los profesores de educación pública que me han enseñado y guiado en mi camino. Gracias a Encarna, por enseñarme a disfrutar de los infinitos mundos de la lectura. A Pepe Peñalver, por esa cara de felicidad que puso cuando le dije que estudiaría Matemáticas. A Maica, Cari, por enseñarme desde la filosofía a reflexionar sin trampas. A Agustín Nogales y MA Gómez-Villegas por su dedicación y su pasión por la enseñanza. A través de su ejemplo, he aprendido el valor de la educación y la importancia de luchar por mis sueños.

Gracias a todos los que han formado parte de esta travesía y han contribuido a mi crecimiento personal y profesional. Este logro no habría sido posible sin su apoyo y aliento.

And remember that you should not attack a complicated problem before you are able to solve a simplified version of it.

Teuvo Kohonen

Resumen

Desarrollo de sistemas expertos con programación funcional y metodología Big Data

Los sistemas expertos han ganado recientemente una gran popularidad en la industria y en el negocio debido a los avances en Big Data y Machine Learning. Estos sistemas están diseñados para realizar tareas de manera más eficiente que los humanos y abordar tareas cognitivas que los humanos no pueden. Esta tesis de doctorado investiga el uso de Big Data, programación funcional y teoría de categorías en el desarrollo de sistemas expertos, enfocándose específicamente en el mapa autoorganizativo (SOM) como una herramienta para la representación del conocimiento. El objetivo es crear un sistema experto basado en functores que pueda transformar un espacio de alta dimensión en un espacio de menor dimensión y fácil de interpretar, mientras preserva la topología del espacio métrico original.

El principal desafío encontrado durante esta investigación fue la representación incompleta del espacio probabilístico original al utilizar un SOM de una sola capa. Una revisión de la literatura reveló que se han propuesto algunas soluciones jerárquicas para abordar este problema, pero no existe una solución existente que utilice el ensamblado distribuido o el bootstrapping. Para abordar este desafío, se propuso un nuevo SOM de dos capas que utiliza el aprendizaje conjunto y se evalúa utilizando un método de validez alternativo. Se ha llevado a cabo un experimento computacional utilizando un modelo de mixtura de gaussiana estocástica para simular datos de baja frecuencia con varias estructuras y representaciones. También se discute el procedimiento de muestreo, la configuración de estratos, la estructura y los parámetros de los SOM y la inicialización de los nodos SOM.

En esta tesis de doctorado, la conservación de la topología, o la preservación de las relaciones entre los elementos de datos, se examina utilizando distancias basadas tanto en imágenes como en grafos, que fueron desarrolladas específicamente para este investigación. Estas distancias sirven para validar la estrategia propuesta a la hora de preservar las relaciones topológicas. Los resultados del experimento mostraron que el SOM de dos capas superó al SOM de una capa en la identificación de valores atípicos. Esto

puede permitir, que Two-layer SOM sea utilizado de manera adecuada como ese funtor sobre la categoría de el espacio probabilístico generación de datos.

Se muestra la principal contribución aportada por el autor de esta memoria. La más relevante, en cuanto a aportación metodológica, se extrae del trabajo presentado en las siguientes páginas.

Contribución : Valverde, G., McWilliams, J., González-Pérez, B. (2021): *One-Layer vs. Two-Layer SOM in the Context of Outlier Identification: A Simulation Study.* Applied Sciences. 11. 6241. 10.3390/app11146241. (JCR)

Abstract

Development of expert systems with functional programming and Big Data methodology

Expert systems have recently gained significant popularity in industry and business due to advances in big data and computational technology. These systems are designed to perform tasks more efficiently than humans and even tackle cognitive tasks that humans cannot. This PhD thesis investigates the use of big data, functional programming, and category theory in developing expert systems, specifically focusing on the self-organizing map (SOM) as a tool for knowledge representation. The goal is to create a functor-level expert system that can transform a high-dimensional space into a lower-dimensional, interpretable space while preserving the topology of the original metric space.

The main challenge encountered during this research was the incomplete representation of the original probabilistic space when using a single-layer SOM. A literature review revealed that some hierarchical solutions have been proposed to address this issue, but there is no existing solution using distributed ensembling or bootstrapping. To address this challenge, a new two-layer SOM was proposed that utilizes joint learning and was evaluated using an alternative validation method. A computational experiment is conducted using a stochastic Gaussian mixture model to simulate low-frequency strata with various structures and representations. The sampling procedure, strata configuration, structure and parameters of the SOMs, and SOM node initialization are also discussed.

In this PhD thesis, the conservation of the topology, or the preservation of the relationships between the data elements, is examined. Both distances based on images and graphs are utilized to achieve this goal, and these distances have been explicitly developed for the research presented in this thesis. These distances serve to validate the proposed strategy for preserving topological relationships. The experiment results showed that the Two-layer SOM outperformed the One-layer SOM in outlier identification. This could allow it to be used appropriately as that functor over the category of the probabilistic space of data generation.

The main paper provided by the author of this report is presented. The most relevant in terms of methodological contribution is extracted from the work presented in the following pages.

Contribution: Valverde, G., Mira McWilliams, J., González-Pérez, B. (2021): *One-Layer vs. Two-Layer SOM in the Context of Outlier Identification: A Simulation Study.* Applied Sciences. 11. 6241. 10.3390/app11146241. (JCR)

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Calabaza, yo te llevo en el corazón.

«Amanece que no es poco»,
Jose Luis Cuerda

Summary: This chapter details the motivations that have driven the author to develop this thesis. In Section 2.1 the general objectives pursued in this work are presented, and the specific ones concerning them. Section 2.2 presents the methodology followed to attain said objectives. Finally, in Section 2.3, a summary of the main scientific contributions of the author is presented, particularly those which have been the basis for this document.

Expert systems have become a motor for industry and business in the last few years. Russel [53] states that the computational revolution and big data management should lead to a new definition of an intelligent system. This author considers that an intelligent system capable of performing human tasks more efficiently could also carry out cognitive ones which are today unreachable for humans.

Other authors point out that this should be accompanied by an update, modification and revision of the use and development of some of the algorithms widely applied in Statistics [48], including some of the most theoretical postulates. Moreover, to assign a significant meaning to the massive *Big Data* sets, which are rapidly expanding in every domain of science and engineering, new ways of thinking and novel learning techniques are required to meet the different challenges..

During the last decade, automatic learning techniques have been widely adopted in fields where massive and complex data sets are generated, such as Medicine, Astronomy or Biology, given that the techniques provide possible solutions to extract the information within the data. However, when the

moment of applying the algorithm nears, the compilation of the different data sets becomes so complex that it is difficult to manage them using traditional learning methods, which were conceived from conventional data sets., not from Big data ones, for which they do not work correctly. For example, most of the traditional automatic learning algorithms are designed for data sets fully loaded onto computer memory, which is no longer feasible for Big data.

Therefore, the generation of significant advances from data analysis has been and is conditional on the modernization of analytical and statistical modelling tools, a task facing enormous challenges [48],[49], [12], [57].

In addition to these challenges, new ones emerge once the first goals in the commercial exploitation of the solutions, with a special focus on ethical, robustness and coherence issues of the solutions. It is not by chance that demand from industry for probabilistic, explanatory and causal systems is increasing with the high presence of predictive solutions in many fields. That is, solutions which deal with uncertainty as one of the main elements to model. Solutions not contemplating uncertainty present, in their practical application, certain restrictions: incoherent predictions, decisions outside the ethical framework or even biased by the data themselves. According to authors such as Pearl [46], some of these problems can be solved, including causality and prior expert knowledge when developing solutions. It is also noteworthy that other authors, such as Bishop [4], who have the opposite opinion, simply argue that it is impossible to require reasoning abilities from a computing machine.

The application of these techniques is increasingly reaching all fields of society. In the healthcare sector, the COVID-19 pandemic allowed researchers to implement real-time expert systems capable of keeping society up to date and supporting the decision-making of administrations [22]. Expert systems'great potential is shown in marketing, where the digital revolution has increased its presence [71]. Companies now own massive data sets on their customers, uses, and preferences. Its purpose is to identify customer profiles through which more wanted products are defined, the development of more customer-oriented campaigns and improve *life time value* [11, 26]. The large volume of data has brought along that no minority profile is negligible. For example, a small proportion of digital platform users can provide high cost effectiveness if it can lead a given product to its needs.

In this context, our contribution to Self Organizing Maps (SOM) [32] allows for developing an expert system capable of designing the best marketing campaign in a given business environment. In this kind of problem, a reduction of customer space is frequently implemented by grouping them through unsupervised Machine Learning (k-means, TSNE, UMAP, SOM,..) [18],[71].

The Self Organizing Map, proposed and developed in the 1980s by Teuvo Kohonen [32] is an unsupervised neural network, and as such, it has the

additional feature of being both a clustering and projection/dimensionality reduction technique. Besides defining a categorical variable which accounts for patterns, SOM has the main purpose of generating a representation in low dimension, typically two, so that the original topological structure does not disappear. This dual functionality is of great value for the problem posed since it allows to design of a reduced state space on which to apply commercial operations without losing reference to prior states and the situation of similar profiles; in other words, it maintains the neighbourhood structure.

The algorithm aims to achieve vector representation on n dimensions onto a bidimensional centroid grid, which we shall call nodes, thus intending to maintain topology in the sense that points close in the original space remain so in the map and likewise with those which are apart. Thus requires a number of arguments, such as the grid size, distance to be used and the number of iterations. A range of alternative versions of the original SOM exists, which establish different methodologies to reduce or mitigate some defects that can appear when training the algorithm. Some include strategies to select the initial points properly, increase granularity and detail in some of the grids or to, work with different distances to build probabilistic nodes or increase computing capacity.

However, any SOM, as well as any other unsupervised learning technique, also presents a limitation due to the large data volume. Given that it is a dimension reduction algorithm, SOM tends to generalise, and we should bear in mind that aggregations required to represent each of the neurons with a single descriptive statistic produce the possibility of finding several modes in the same neighbourhood.

This causes that in situations with unbalanced cases, there exist subsamples which are diluted in the rest. This constitutes an example of one of the objectives of the thesis, to show how in specific cases, the classical way of applying some of the algorithms to new Big Data situations may not be adequate, and it becomes necessary to redesign alternative schemes to meet these new challenges, to develop them and implement them for use in different cases.

1.1. Objectives

In this work, we focus on the usefulness of expert systems as a tool for democratizing statistical solutions for Big Data modeling. These systems allow for the extraction of information from data and improve decision making while maintaining rigor and scientific methodology. It has become clear that traditional analytical systems must be incorporated into this new paradigm, which may require modifications to certain algorithms to fit a new axiomatic. For example, the central theorem in simple random sampling with replacement is no longer valid in many real-world Big Data cases that typically

exhibit multimodal distributions.

Our main contribution is the development of a new variation of the parallelized Self-Organizing Map algorithm that addresses the issue of strong imbalances in probability distributions within a population group. This problem often arises when working with multivariate Gaussian mixtures, where some low-frequency but important groups may not be represented, resulting in a loss of the topological structure, which is a key characteristic of the SOM.

This problem, while not unique to our algorithm, has not been extensively studied in the literature because it is not evident in non-Big Data databases. We have introduced ensemble learning techniques that allow for bias towards certain groups based on their relevance rather than just their frequency, thus ensuring that they are preserved in the final map while maintaining the original topology of the data.

The concept of topology maintenance is controversial, and this thesis aims to address this challenge by constructing a distance metric that defines it as we understand it that points close together in the original space are also close together in the map. We explore this concept of proximity in the map from different perspectives, such as nodes of a graph defined by the map structure or using the map as an image and measuring distance between images.

We are also working on several real-world applications of our developed Two-layer SOM solution, comparing its performance to other techniques. This work has led to the creation of expert systems related to this thesis, which use Two-layer SOM as a knowledge base or as the foundation of the inferential engine. The use of functional programming based on category theory allows for flexibility in different environments where a common map structure is used despite the diversity of the original spaces.

The general objectives of this thesis, as well as the specific objectives related to each of them, are outlined below.

Objetive 1: Perform implementations with parallelized processing on distributed data.

1. Compare the performance results on different configurations and optimization systems from the theoretical point of view.

2. Generate graphic procedures that allow the visualization of the results.

Objetive 2: Evolution of the algorithm to a multilayer SOM structure that would allow to identify hierarchical topological structures in the original space and reduce the effect of underrepresentation of low frequency areas. Sokolovska [58].

1. Design of metrics for comparison of the structure of the distribution

of points in strata with respect to the original population Yin [77] and Van Hulle [28].

2. Topology maintenance evaluation metric through neighbourhood ordering and generation of graph structures.

3. Study of the effects of various forms of stratification on the result.

4. Methodological and experimental justification of parallelization [74]

1.2. Methodology

The following methodology has been followed to achieve the objectives described:

- The initial objectives of this thesis **Objective 1** was to study and deepen the understanding of Big Data tools, database systems, and the configuration of Big Data architectures for data collection and the construction of expert systems [36, 22]. To achieve this, the tutor of this thesis has been trained and specialized in using tools such as Spark, Mongodb, Kafka, Scala, and python as programming languages.

- Simultaneously, work, in the same line of deepening or maturing, has also been done on the development of tools for the automation of a digital marketing campaign manager. To do this, it is proposed to characterize customers and design an automatic decision-making tool based on reinforcement learning.

- The aim is to create an expert system in the form of a functor, capable of transferring the original space to a low-dimensional, interpretable space that represents the original metric space with its topology. Among the various algorithms considered, the Self Organizing Map (SOM) is one of the best alternatives.

- However, it may have the problem that some groups of elements from the original space are not represented in the final view. After conducting a literature review, it was found that some hierarchical solutions have been proposed for this problem, but there is no existing solution using distributed ensembling or bootstraping. This leads to the definition of **Objective 2**.

- To address this issue, different intuitive strategies are investigated to simulate this situation that could occur in a real dataset. Finally, an example is defined using Gaussian mixture to characterize the problem, as it has the advantage of being able to be defined in different dimensions and with different combinations of elements that are prone to not

being identified. This is a known type of problem in the state of the art.

- The experiment is then designed, defining the types of examples that are easier to solve and those that are more similar to the complex real-world situation. Once defined, alternative solutions are validated, leading to the Two-layers solution, which needs to be compared and validated using a measure of the similarity of the original and reconstructed metric spaces.

- To compare the results, different distance measurement techniques are applied to SOM, to images and graphs. This allows for the identification of the measure that best satisfies the guiding criterion and validates the solution obtained in different scenarios created with the sampling system. This process is the main contribution of this thesis. All this process is reflected in the main contribution of this thesis [69].

1.3. Contributions

In this section the main contribution provided by the author of this report is presented. The most relevant in terms of methodological contribution is extracted from the work presented in the following pages.

Contribution 1: Valverde, G., McWilliams, J., González-Pérez, B. (2021): *One-Layer vs. Two-Layer SOM in the Context of Outlier Identification: A Simulation Study.* Applied Sciences. 11. 6241. 10.3390/app11146241. (JCR)

1.4. Dissertation structure

The structure of this PhD thesis includes an introduction, literature review, background, methodology, results, discussion, conclusion, and references. The introduction (Chapter 2 and 1) sets the context and objectives of the research, while the literature review examines the existing research on expert systems and Self-Organizing Maps (SOM) and their integration. The background chapter (Chapter 3) provides an overview of the concepts and techniques relevant to the research, including expert systems (Section 3.1), category theory and functional programming (Section 3.2), big data (Section 3.3), and distances (Section 3.4).

The Self-Organizing Map (SOM) chapter (Chapter 4) covers the introduction and operation of SOM and its algorithm and mathematical properties (Section 4.3). It also reviews different alternatives to SOM and discusses the original sequential SOM in detail. The chapter also covers the quality of

SOM, including topology-based and density-based measures (Section 4.6). Finally, the chapter concludes with a discussion of the future of SOM and its potential applications.

The one-layer vs. two-layer SOM (Chapter 5) introduces the principal contributions. The use of Two-layer SOM in the context of outlier identification. It presents a computational experiment using a stochastic Gaussian mixture model for simulations (Section 5.2). The conservation of topology, or the preservation of the relationships between the elements in the data, is analyzed using image-based and graph-based distances, as well as a toy example (Section 5.3). The interpretation of the results and a concluding discussion are also presented (Section 5.4).

The Conclusions and Future Work chapter (Chapter 5.5 and 5.5) summarizes the main findings of the research and discusses the implications of the results. It also identifies further research areas and suggests future work directions.

Chapter 2

Introducción

Calabaza, yo te llevo en el corazón.

«Amanece que no es poco»,
Jose Luis Cuerda

Resumen:

Sirva este capítulo introductorio para detallar las motivaciones que han llevado al autor al desarrollo de esta memoria. En la Sección 2.1 se exponen los objetivos generales perseguidos en la elaboración de este trabajo, así como los distintos objetivos específicos relativos a los mismos. En la Sección 2.2 se presenta la metodología seguida para alcanzar los objetivos enunciados. Finalmente, en la Sección 2.3 se hace una reseña de las principales contribuciones científicas aportadas por el autor, particularmente aquellas que han servido como punto de partida para la escritura de esta memoria.

Los sistemas expertos se han convertido en el motor de la industria y el negocio en los últimos años. Russel [53] plantea que la revolución computacional y de gestión de datos masivos debe llevar a una nueva definición de sistema inteligente. Considera que un sistema inteligente que es capaz de realizar tareas humanas de manera más eficiente, además podría realizar tareas cognitivas que hoy en día son inabordables por un humano.

Otros autores mencionan que esto debe ir acompañado de una actualización, modificación y revisión del uso y desarrollo de algunos de los algoritmos ampliamente utilizados en el campo de la estadística [48]. También de algunos de sus postulados más teóricos. Además, para dar sentido significativo, al conjunto de datos masivos *Big Data* que se ha estado expandiendo rápidamente en todos los dominios de la ciencia y la ingeniería, también se requiere de nuevas formas de pensar y técnicas de aprendizaje novedosas para abordar los diversos desafíos.

Durante la última década, las técnicas de aprendizaje automático se han adoptado ampliamente en campos donde se generan conjuntos de datos masivos y complejos, como la medicina, la astronomía, la biología, ya que estas técnicas brindan posibles soluciones para extraer la información latente en los datos. Sin embargo, a medida que se acerca el momento de aplicar el algoritmo, la recopilación de conjuntos de datos es tan compleja que es difícil manejarlos utilizando métodos de aprendizaje tradicionales, ya que estos fueron concebidos a partir de conjuntos de datos convencionales. No fueron diseñados y no funcionan bien con grandes volúmenes de datos. Por ejemplo, la mayoría de los algoritmos tradicionales de aprendizaje automático están diseñados para datos que se cargarían por completo en la memoria, lo que ya no es válido en el contexto de Big Data.

Por lo tanto, la generación de avances significativos a partir de los datos ha estado y está supeditada a una modernización de los sistemas analíticos y de modelado estadístico, tarea que presenta enormes desafíos [48],[49], [12], [57].

A estos desafíos debemos añadir los nuevos retos que surgen una vez alcanzadas las primeras cotas en la explotación comercial de las soluciones, con las cuestiones éticas, la robustez y la coherencia de las soluciones como principal foco. No es casualidad que, con el alto grado de implantación actual de soluciones predictivas en multitud de ámbitos, sea mayor la demanda que la industria hace de sistemas probabilísticos, explicativos y causales; es decir, soluciones que traten la incertidumbre como uno de los elementos fundamentales a modelar. Las soluciones que no contemplan dicha incertidumbre presentan en su aplicación práctica ciertas limitaciones: predicciones incoherentes, decisiones fuera del marco ético e incluso sesgadas por los propios datos. Según autores como Pearl [46] algunos de estos problemas pueden resolverse incluyendo causalidad, información a priori experta a la hora de construir dicha solución. Cabe remarcar, que también podemos encontrar otros autores como [4] que opinan en dirección opuesta, argumentando simplemente que no es posible exigir a una máquina de computo razonar.

La aplicación de estas técnicas alcanza cada vez más todos los campos de la sociedad. En el ámbito médico, la pandemia del COVID-19 dio a los investigadores la oportunidad de implementar sistemas expertos en tiempo real, aistemas capaces de mantener informada a la sociedad y servir como soporte a la toma de decisiones gubernamentales [22]. El gran potencial de los sistemas expertos se viene demostrando en el ámbito del marketing, donde la revolución digital no ha hecho más que incrementarlo [71]. Las compañías disponen de cantidades masivas de datos en tiempo real sobre sus clientes, sus usos y sus gustos. Su objetivo es identificar perfiles de los clientes con los que definir productos más deseados, construir campañas de captación más personalizadas, y mejorar el *life time value* [11, 26]. El gran volumen de datos ha hecho que ningún perfil minoritario sea despreciable. Por ejemplo,

una pequeña proporción de usuarios de plataformas digitales puede significar una gran rentabilidad si se consigue orientar un determinado producto a sus necesidades.

En este contexto, nuestra contribución sobre *Self-Organizing Map* (SOM) [32] permite el desarrollo de un sistema experto capaz de determinar la mejor campaña de marketing a aplicar en un determinado entorno comercial. En este tipo de problemas es usual reducir el espacio de clientes agrupándolos mediante técnicas no supervisadas de Machine Learning (kmedias, TSNE, UMAP, SOM,...) [18],[71].

El Mapa Autoorganizativo (Self-Organizing Map), propuesto y desarrollado en los 80 por Teuvo Kohonen [32] es un tipo de red neuronal; cuyo aprendizaje es no supervisado. Como algoritmo no supervisado tiene otra peculiaridad y es que es tanto un algoritmo de clustering como un algoritmo de proyección o reducción dimensional. Además de definir una variable categórica que explique los patrones; SOM tiene como objetivo principal generar una representación de dimensión menor, generalmente dos, de tal manera que la estructura topológica del espacio original no desaparezca. Esta bifuncionalidad es de gran valor en el problema planteado, ya que permite definir un espacio de estados reducido sobre el que aplicar acciones comerciales sin perder de referencia estados previos y la situación de perfiles similares, mantiene el vecindario.

El objetivo del algoritmo es conseguir representar un conjunto de vectores de dimensión n en un mallado bidimensional de centroides, que llamaremos nodos, buscando que la topología, en el sentido de que los vecindarios, se mantenga, y que puntos que se encontraban cercanos en el espacio original lo estén en éste y que los separados también lo estén. Para ello necesita una serie de argumentos como tamaño del mallado, distancia a utilizar y número de iteraciones. Existe una variedad de versiones alternativas al SOM original que establecen distintas metodologías para reducir o mitigar algunos defectos que se pueden dar al entrenar este algoritmo. Algunos de ellos incluyen estrategias para seleccionar bien los puntos iniciales, aumentar la granularidad y el detalle en algunos de los mallados o trabajar con distintas distancias, construir nodos probabilísticos o mejorar la capacidad de computo.

Sin embargo, cualquiera SOM, al igual que cualquier otra técnica no supervisada, también presenta una limitación debido al gran volumen de datos con el que trabajamos y perdía alguno de los conjuntos de clientes más rentables e interesantes por ser poco frecuentes. Al tratarse de un algoritmo de reducción dimensional, SOM, tiende a generalizar, debemos tener en cuenta que las agregaciones que sirven para representar cada una de las neuronas con un único estadístico descriptivo, lo que elimina la posibilidad de encontrar varias modas en un mismo vecindario. Esto provoca que en situaciones de desbalanceo de casos entre una u otra haya submuestras que queden difuminadas entre el resto.

Este es un ejemplo de uno de los objetivos a tratar en la tesis, mostrar como en casos concretos la forma clásica de aplicar algunos de los algoritmos a las nuevas situaciones Big Data puede no ser adecuada y es necesario replantear alternativas que puedan responder ante estos nuevos desafíos, desarrollarlas y ponerlas en funcionamiento en diversos casos de uso.

2.1. Objetivos

En este marco, se incide en la utilidad de los sistemas expertos como herramienta para la democratización de las soluciones estadísticas al modelado de Big Data, a la hora de extraer la información subyacente a los datos y mejorar la toma de decisiones sin descuidar el rigor y la metodología científica. Se ha puesto de manifiesto la necesidad de incorporar los sistemas analíticos tradicionales a este nuevo paradigma. En este sentido son necesarias modificaciones de algunos de los algoritmos para ajustarlos a una nueva axiomática. Un ejemplo puede ser el teorema central en el muestreo aleatorio simple con reemplazamiento, que deja de tener validez en casos reales Big Data que suelen presentar distribuciones multimodales [23, 39].

Las aportaciones principales de este trabajo van en esta línea, se ha trabajado concretamente en una nueva variación del algoritmo Self-Organazing Map paralelizada que solventa el problema detectado al ser aplicado a un conjunto poblacional con fuerte desbalanceo en las distintas distribuciones de probabilidad que lo componen.

Este problema aparece al trabajar con mixturas de gaussianas multivariante, donde algunos grupos de baja frecuencia y gran interés no tienen representante, es decir desaparecen como grupo en la proyección definida por el mapa. Esto rompe la estructura topológica, característica fundamental del SOM.

El problema, que no es exclusivo de este algoritmo, no ha sido muy estudiado en la literatura por no ser tan evidente en bases de datos no Big Data. Se han introducido técnicas de ensemble learning que han permitido sesgar la influencia de ciertos grupos teniendo en cuenta su relevancia no sólo su frecuencia, de modo que acaban apareciendo en el mapa final como grupo diferenciado conservándose la topología original de los datos.

El concepto de mantenimiento de topología es controvertido, en esta tesis se aborda el reto de construir una distancia que la defina tal y como la entendemos, que dos puntos cercanos en el espacio original se sitúen próximos en el mapa. Profundizamos en este concepto de proximidad en el mapa desde distintos puntos de vista, ya sea como nodos de un grafo definido por la estructura mapa o utilizando el mapa como si de una imagen se tratara y midiendo distancia entre imágenes.

Finalmente, se está trabajando en varias aplicaciones reales de la solución desarrollada, Two-layer SOM, comprobando sus virtudes frente a otras

técnicas. Todo este trabajo ha dado lugar a la construcción de sistemas expertos, relacionados con esta tesis, que se apoyan en Two-layer SOM como base de conocimiento o como base del motor inferencial. La aplicación de la programación funcional, basada en la teoría de categorías, aporta flexibilidad de uso en distintos entornos en los que trabajamos con una estructura de mapa común aunque los espacios originales sean diversos.

A continuación se concretan los objetivos generales a desarrollar en esta memoria, así como los objetivos específicos relativos a cada uno de ellos, atendiendo al orden de aparición de los mismos en las siguientes páginas.

Objetivo 1: Realizar implementaciones con procesamiento paralelizado sobre datos distribuidos.

1. Comparar los resultados de rendimiento sobre diferentes configuraciones y sistemas de optimización desde el punto de vista teórico.
2. Generar procedimientos gráficos que permitan la visualización de los resultados.

Objetivo 2: Evolución del algoritmo a una estructura SOM multicapa que permitiera identificar estructuras topológicas jerárquicas en el espacio original y disminuir el efecto de infrarrepresentación de zonas de baja frecuencia. Sokolovska [58].

1. Diseño de métricas de comparación de estructura de distribución de puntos en estratos con respecto a la población original Yin [77] and Van Hulle [28].
2. Métrica de evaluación de mantenimiento de topología mediante ordenamiento de vecindarios, generación de estructuras de grafos.
3. Estudio de efectos de diversas formas de estratificación en el resultado.
4. Justificación metodológica y experimental de la paralelización [74]

2.2. Metodología

Para alcanzar los objetivos descritos se ha seguido la siguiente metodología.

- Para cubrir los objetivos iniciales **Objetivo 1** ha habido un proceso de maduración, en el aspecto de Big Data y programación funcional que se reflejan en la memoria, a partir de distintos proyectos relacionados con el análisis de datos en el ámbito médico [36, 22]. Donde se pudo profundizar en el estudio de herramientas de Big Data, sistemas de bases de datos y configuración de arquitecturas Big Data para la

recolección de datos y las construcción de sistemas expertos, con herramientas como Spark, Mongodb, Kafka. Scala, python como lenguajes de programación, en las que el autor de esta tesis se ha formado y especializado.

- Simultáneamente, en la misma línea de profundicación o maduración, se estaba trabajando en desarrollo de herramientas para la automatización de un gestor de campañas de marketing digital, para lo que se propuso caracterizar a los clientes y diseñar una herramienta automática de toma de decisiones basada en aprendizaje por refuerzo. Para la caracterización buscamos definir un sistema experto, una solución en forma de funtor, capaz de trasladar el espacio original a un espacio de baja dimensión, interpretable y que represente el espacio métrico original con su tolopogía. Entre diversos algortimos encontramos Self Organizing Map [32].

- Una vez elegido como alternativa, se encuentra que puede tener el problema de que algunos grupos de elementos del espacio original que no estén representados en la visión final. Después de un análisis bibliográfico vemos que un tema tratado con algunas soluciones jerárquicas pero no parece que haya una solución trabajada con ensembling distribuido o bootstraping. Se define **Objetivo 2**.

- Fueron investigadas distintas estrategias intuitivas en las que simular esta situación que se ha podido encontrar en el conjunto de datos real. Finalmente se ha definido un ejemplo que caracterizar el problema utilizando mixtura de gausianas, que tiene la ventaja de poderse definir en distintas dimensiones y con distintas combinaciones de elementos propensos a no ser identificados y en el estado del arte es un tipo de problema conocido. Se define el diseño del experimento, seleccionando los tipos de ejemplos que pueden ser más sencillos de solucionar y aquellos que se asemejan más a la situación compleja real. Una vez definidas se realiza el experimento con el que ir validando distintas soluciones alternativas que concluyen con la solución Two-layers, que para ser comparada y validada necesita una medida de la similitud de los espacios métricos, original y reconstruido.

  Para poder comparar los resultados hacemos un análisis de distintas técnicas de medición de distancias aplicadas a SOM, a imágenes y grafos, a partir de ellas se construye la medida que identifica mejor el criterio que nos guía y se valida la solución obtenida en distintos escenarios construidos con nuestros sistema de muestreo. Todo este proceso se ve reflejado en la principal contribución de esta tesis [69].

2.3. Contribuciones

En esta sección se muestra la principal contribución aportada por el autor de esta memoria. La más relevante en cuanto a aportación metodológica se extrae del trabajo presentado en las siguientes páginas.

Contribución 1: Valverde, G., McWilliams, J., González-Pérez, B. (2021): *One-Layer vs. Two-Layer SOM in the Context of Outlier Identification: A Simulation Study.* Applied Sciences. 11. 6241. 10.3390/app11146241. (JCR)

2.4. Estructura del documento

La estructura de esta tesis doctoral incluye una introducción, revisión de la literatura, antecedentes, metodología, resultados, conclusión, trabajo futuro y referencias. La introducción (Capítulo 2 y 1) establece el contexto y los objetivos de la investigación, mientras que la revisión de la literatura examina la investigación existente sobre sistemas expertos y mapas autoorganizativos (SOM) y su integración. El capítulo de antecedentes (Capítulo 3) proporciona una visión general de los conceptos y técnicas relevantes para la investigación, incluyendo sistemas expertos (Sección 3.1), teoría de categorías y programación funcional (Sección 3.2), Big Data (Sección 3.3) y distancias (Sección 3.4).

El capítulo del mapa autoorganizativo (SOM) (Capítulo 4) cubre la introducción y el funcionamiento de SOM y sus propiedades algorítmicas y matemáticas (Sección 4.3). También revisa diferentes alternativas a SOM y discute en detalle el SOM secuencial. El capítulo también aborda la calidad de SOM, incluyendo medidas basadas en topología y densidad (Sección 4.6). Finalmente, el capítulo concluye con una discusión sobre el futuro de SOM y su relación con sistemas expertos, teoría de categoría y Big Data..

El capítulo de SOM One-layer vs. Two-layer (Capítulo 5) introduce las contribuciones principales. El uso del SOM de dos capas en el contexto de la identificación de valores atípicos. Presenta un experimento computacional utilizando un modelo de mezcla gaussiana estocástica para simulaciones (Sección 5.2). Se analiza la conservación de topología, o la preservación de las relaciones entre los elementos en los datos, utilizando distancias basadas en imágenes y gráficos, así como un ejemplo sencillo (Sección 5.3). También se presenta la interpretación de los resultados y una discusión (Sección 5.4).

El capítulo de conclusiones y trabajo futuro (Capítulo 5.5 y5.5) resume los principales hallazgos de la investigación y discute las implicaciones de los resultados. También identifica áreas de investigación adicionales y sugiere direcciones de trabajo futuro.

Chapter 3

Background

Summary: This chapter sets out the concepts that justify the content of this thesis. Consequently, we distinguish three subjects that refer to the title of the thesis and the last one that has been necessary to build the validation system. In the first part, Section 3.1 defines and describes an expert system, listing each of its versions and highlighting its transformation work as an intermediate tool between users and computer or learning systems. Section 3.2 introduces category theory as a mathematical basis for functional programming and its usefulness in combination with neural networks. We will show how functional programming is a fundamental part of the diffusion in the use of neural networks, facilitating the necessary syntax to build them through software. Thirdly, in Section 3.3 this paradigm is presented as a tool to implement and execute a variety of algorithms but also as a generator of new problems or perspectives that lead to modifications or improvements on the proposed algorithms. Finally, in Section 3.4, we recall some classic concepts of measure theory, specifically the definitions of metric and distance that have been essential to validate some of the proposed solutions.

Resumen: Este capítulo establece los conceptos que justifican el contenido de esta tesis. En consecuencia, distinguimos tres temas que se refieren al título de la tesis y el último que ha sido necesario para construir el sistema de validación. En la primera parte, la Sección 3.1 define y describe un sistema experto, enumerando cada una de sus versiones y destacando su trabajo de transformación como una herramienta intermedia entre los usuarios y la computadora o los sistemas de aprendizaje. La Sección 3.2 introduce la teoría de categorías como base matemática para la programación funcional y su utilidad en combinación con las redes neuronales. Mostraremos cómo la programación funcional es parte fundamental de la difusión en el uso de redes neuronales, facilitando la sintaxis necesaria para construirlas a través del

software. En tercer lugar, en la Sección 3.3, este paradigma se presenta como una herramienta para implementar y ejecutar una variedad de algoritmos, pero también como generador de nuevos problemas o perspectivas que conducen a modificaciones o mejoras en los algoritmos propuestos. Finalmente, en la Secció 3.4, recordamos algunos conceptos clásicos de teoría de medidas, específicamente las definiciones de métrico y distancia que han sido esenciales para validar algunas de las soluciones propuestas.

3.1. Expert System

In recent times, it was thought that issues such as theorem proof, speech and pattern recognition, certain games (such as chess or checkers), and complex systems of deterministic or stochastic nature had to be addressed by human beings since they require certain abilities that are only found in humans (such as the ability to think, perceive, remember, learn, see, smell, etc.). However, the work carried out in the last three decades by researchers from various fields has shown that machines can formulate and solve many of these problems.

The broad discipline known as artificial intelligence (AI) deals with these problems initially considered impossible, intractable, and difficult to formulate using computers. Feigenbaum, a pioneer in AI research, defines AI as follows [3]:

"Artificial Intelligence is the branch of science that deals with the design of intelligent computing systems, that is, systems that exhibit the characteristics we associate with intelligence in human behaviour, such as language comprehension, learning, reasoning, problem-solving, etc."

Currently, the field of AI encompasses several sub-areas, such as expert systems, automatic theorem proving, automatic game playing, speech and pattern recognition, speech processing, natural language, artificial vision, robotics, neural networks, etc.

This section focuses on expert systems. Although expert systems are one of the research areas within AI, most, if not all, of the other areas have a component of expert systems as part of them.

We start with some definitions of expert systems in Section 3.1.1. Section 3.1.2 provides examples illustrating the motivations for expert systems in various fields of application. These examples highlight the importance and broad applicability of expert systems in practice. Section 3.1.2 lists some of the reasons for using expert systems. The main types of expert systems are presented in Section 3.1.3. Section 3.1.4 discusses and analyzes the structure of expert systems and their main components. The different steps required for

the design, development, and implementation of expert systems are discussed in Section 3.1.5.

3.1.1. What is an expert system?

Expert systems are computer-based systems that aim to replicate the **decision-making** abilities of human experts in a specific domain. These systems often use a combination of machine learning algorithms, knowledge representation, and reasoning techniques to provide intelligent solutions to complex problems.

Many definitions of expert systems can be found in the existing literature. For example, [60] defines expert systems as "machines that think and reason as an expert would do it in a certain speciality or field". This definition emphasizes the ability of expert systems to process and manipulate data in a way that is intelligible and meaningful to answer questions, even when they are not fully specified.

Other definitions have emerged due to the rapid development of technology. For example, [7] and [20] define expert systems as computer systems (hardware and software) that simulate human experts in a given area of expertise. These definitions highlight the ability of expert systems to learn, reason, and communicate with humans and other systems and make appropriate decisions and provide explanations for their actions.

Expert systems have been applied to various fields, including economics, industry, medicine, engineering, and transportation. These systems can provide valuable assistance to human experts by processing and analyzing large amounts of data, making inferences and decisions, and providing explanations for their actions.

3.1.2. Why expert systems? Illustrative examples

Expert systems have many applications in different fields. Some examples of expert systems include:

1. Bank transactions: Expert systems can be used to automate banking transactions, such as depositing or withdrawing money from an account, using automated teller machines (ATMs).

2. Planning problems: Expert systems can be used to solve complex scheduling problems, such as organizing and allocating classrooms for final exams in a large university, to optimize certain objectives.

3. Medical diagnosis: Expert systems can be used to collect, organize, store, and retrieve medical information, such as patient records, and to diagnose diseases based on a set of symptoms.

4. Secret agents: Expert systems can be used to solve puzzles and riddles, such as the location of secret agents in different countries.

5. Financial analysis: Expert systems can be used to analyze and forecast financial markets using statistical and mathematical models. For example, an expert system could use a linear regression model to predict a company's stock price, given its historical data and market trends.

6. Engineering design: Expert systems can be used to assist engineers in the design of complex systems, such as bridges, buildings, and aircraft. For example, an expert system could use a finite element method to simulate the stress and deformation of a structure and optimize its design parameters.

7. Scientific discovery: Expert systems can be used to assist scientists in the discovery of new knowledge using data mining and machine learning techniques. For example, an expert system could use a clustering algorithm to identify patterns and trends in a dataset and suggest hypotheses and predictions.

Expert systems can provide valuable assistance to human experts in a wide range of fields by processing and analyzing large amounts of data, making inferences and decisions, and providing explanations for their actions.

The development or acquisition of an expert system often comes with a significant upfront cost, but the ongoing maintenance and use of the system are relatively inexpensive. The benefits of expert systems, including cost savings, time efficiency, and improved accuracy, are substantial and can be quickly realized. However, a feasibility study and cost-benefit analysis should be conducted before pursuing the implementation of an expert system.

There are several reasons why expert systems are valuable tools:

- Expert systems allow individuals with limited expertise to tackle complex problems that would otherwise require a specialist's knowledge. This is particularly useful in situations where experts are scarce or inaccessible. Additionally, expert systems enable broader access to knowledge by allowing more people to use the system.

- Expert systems can integrate the expertise of multiple human experts, resulting in more reliable solutions. This is because an expert system that combines the insights of several experts can provide a more comprehensive perspective than a single human expert.

- Expert systems can provide answers and solve problems much faster than a human expert, making them valuable in time-critical situations.

- Some problems may be too complex for a human expert to solve, or the solutions provided by human experts may not be reliable. In these cases, expert systems can provide reliable and timely answers through their ability to process and approximate complex operations quickly.
- Expert systems can be used to perform tedious, dull, or uncomfortable tasks that would be difficult for humans to undertake. In some cases, expert systems may be the only feasible option, such as controlling an aeroplane or space capsule.
- Expert systems can provide significant cost savings. The use of expert systems is particularly advisable in the following scenarios:
  - When knowledge is hard to come by or is based on rules that can only be acquired through experience.
  - When the continual improvement of knowledge is crucial and/or when the problem is subject to changing rules or regulations.
  - When human experts are expensive or difficult to locate.
  - When the knowledge of the users on the topic is limited.

3.1.3. Types

Expert systems can be used to tackle two types of problems: essentially deterministic problems and essentially stochastic problems. For example on Section 3.1.2, although Examples 1 (banking transactions) and 2 (traffic control) may contain some elements of uncertainty, they are essentially deterministic problems. In the field of medicine (see Example 4), the relationships between symptoms and diseases are known only with a certain degree of certainty (the presence of a set of symptoms does not always imply the presence of a disease). These types of problems may also include some deterministic elements, but they are fundamentally stochastic problems.

Consequently, expert systems can be classified into two main types based on the nature of the problems they are designed to solve: deterministic and stochastic.

Deterministic problems can be formulated using rules that relate to various well-defined objects. Expert systems that deal with deterministic problems are known as rule-based systems, because they draw conclusions based on a set of rules using a logical reasoning mechanism.

It is necessary to introduce some means of dealing with uncertain situations. For example, some expert systems use the same structure as rule-based systems, but present a measure associated with the uncertainty of the rules and their premises. In this case, some propagation formulas can calculate the uncertainty associated with the conclusions. Over the past decades, some measures of uncertainty have been proposed. Examples of these measures

are the certainty factors, used in shells for generating expert systems such as the MYCIN expert system [5]; fuzzy logic [80]; and evidence theory [55].

Another intuitive and rigorous measure of uncertainty is probability, in which the joint distribution of a set of variables is used to describe the dependencies between them, and conclusions are drawn using well-known formulas from probability theory. This is the case of the PROSPECTOR expert system [17] which uses Bayes'theorem for mineral exploration.

Expert systems that use probability as a measure of uncertainty are known as probabilistic expert systems and the reasoning strategy is called probabilistic reasoning.

Probabilistic expert systems are artificial intelligence systems that use probability as a measure of uncertainty to make decisions and solve problems. These systems rely on a computer program that encodes the expert knowledge and uses probabilistic reasoning, also known as probabilistic inference, to calculate the likelihood of different events and their dependencies. Probabilistic expert systems are particularly useful when the knowledge and relationships between different variables are uncertain or subject to change, such as in medical diagnosis or financial prediction.

The use of probabilistic expert systems has increased significantly in recent decades thanks to the introduction of probabilistic network models, such as Markov and Bayesian networks. These models use a graphical representation of the relationships between the variables to define the joint probability distribution and propagate uncertainty efficiently, allowing for more accurate and reliable conclusions. Examples of software tools for developing probabilistic expert systems include HUGIN, PyMC3, Pyro, bnlearn, dbnR and DoWhy. Some books that provide a general introduction to different measures of uncertainty and expert systems include [5], [45], [20], [29].

3.1.4. Components

An expert system is a computer program that encodes the knowledge and expertise of human experts in a specific domain and uses logical or probabilistic reasoning to make decisions and provide solutions to complex problems. Expert systems consist of several components that work together to perform this task, including a human component, a knowledge base, a knowledge acquisition subsystem, a consistency check subsystem, an inference engine, and a user interface.

- The human component of an expert system consists of one or more subject-matter experts and knowledge engineers who collaborate to provide the knowledge and expertise required by the system.
- The knowledge base is a structured and organized representation of the expert knowledge that is encoded in the system. The knowledge

acquisition subsystem controls the flow of new knowledge from the human experts to the knowledge base, and ensures that the knowledge is relevant, accurate, and consistent.

- The consistency check subsystem verifies the coherence and consistency of the knowledge base, and prevents inconsistent or contradictory knowledge from entering the system.

- The inference engine is the core component of an expert system, as it is responsible for applying the knowledge to the data and making conclusions. Depending on the type of expert system, the inference engine can use deterministic or probabilistic reasoning to draw conclusions and provide solutions to the given problem.

  The user interface is the component that allows the user to interact with the expert system, providing input data, requesting information, and receiving the results of the system's reasoning and decision-making.

- The knowledge acquisition subsystem is an essential component of expert systems. When the initial knowledge is limited and the inference engine cannot draw conclusions, the knowledge acquisition subsystem is used to obtain the necessary information and continue the inference process. In some cases, the user may provide this information through the user interface, which must also include consistency checks to ensure the reliability of the information provided.

- The explanation subsystem provides explanations to the user for the conclusions drawn or actions taken by the expert system, while the learning subsystem allows the expert system to improve its knowledge base through the discovery of new information or the estimation of parameters. These components work together to enable expert systems to provide valuable advice to users.

Expert systems are unique in their ability to gain experience and improve over time through the use of available data. This data can be collected by both experts and non-experts, and is utilized by the knowledge acquisition subsystem and the learning subsystem. The various components of expert systems (Fig. 3.1) allow them to perform a wide range of tasks, including but not limited to acquiring and verifying knowledge, storing knowledge, requesting new knowledge, learning from the knowledge base and available data, performing inference and reasoning, explaining conclusions and actions, and communicating with human experts and other expert systems. These capabilities make expert systems valuable tools for many different applications.

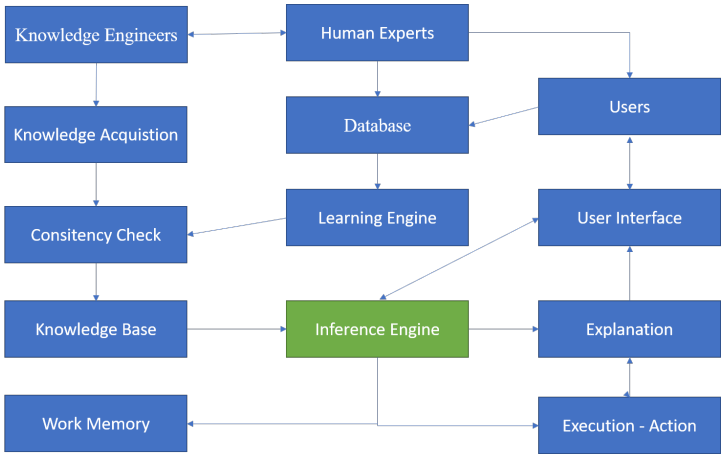


Figure 3.1: Components of a expert system

3.1.5. Development

Weiss [75] suggest the following stages for the design and implementation of an expert system Fig. 3.2:

- Formulation of the problem. The first stage in any project is typically the definition of the problem to be solved. Since the main goal of an expert system is to answer questions and solve problems, this stage is perhaps the most important in the development of an expert system. If the system is poorly defined, it is expected to provide incorrect answers.
- Finding human experts who can solve the problem. In some cases, however, databases can play the role of the human expert.
- Design of an expert system. This stage includes the design of structures to store knowledge, the inference engine, the explanation subsystem, the user interface, etc.
- Selection of the development shell, tool, or programming language. It must be decided whether to build a custom expert system or to use a shell, a tool, or a programming language. If there is a shell that satisfies all the requirements of the design, it should be the choice, not only for financial reasons but also for reliability reasons. Shells and commercial tools are subject to quality controls, which other programs are not.
- Development and testing of a prototype. If the prototype does not pass the required tests, the previous stages (with the appropriate modifications) must be repeated until a satisfactory prototype is obtained.
- Refinement and generalization. In this stage, flaws are corrected and new possibilities not included in the initial design are added.

- Maintenance and updating. In this stage, the user poses problems or defects of the prototype, corrects errors, updates the product with new advances, etc.

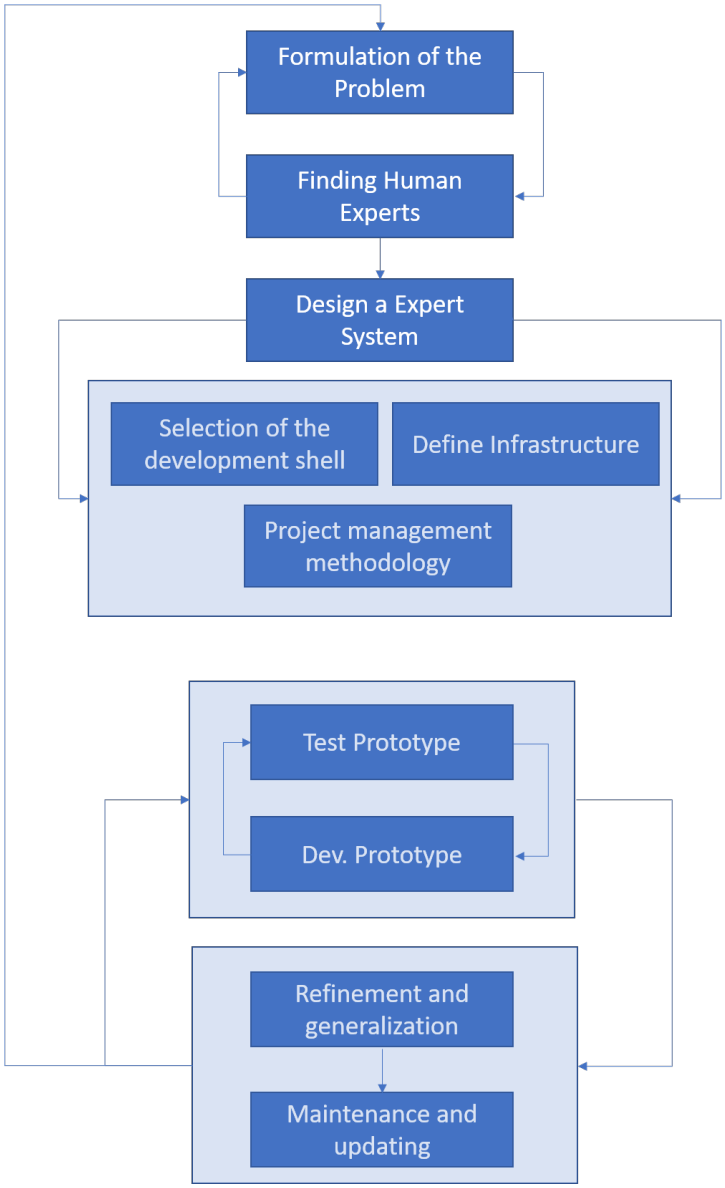


Figure 3.2: Development steps of a expert system

All these stages (Fig. 3.2) influence the quality of the resulting expert system, which must always be evaluated based on the contributions of users.

In this section, a brief overview of the scope and domain of some areas of AI other than expert systems is provided. It should be noted that this is not

an exhaustive list of all areas of AI and that AI is a rapidly developing field, with new branches emerging to address new challenges in this constantly growing science.

In addition to expert systems, there are many other areas within AI that are worth exploring. Some of these areas include:

Machine learning: This field focuses on algorithms that can learn from and make predictions based on data. This can involve developing systems that can improve their performance over time through experience.

Natural language processing: This area focuses on developing algorithms that can understand, interpret, and generate human language. This can include tasks like language translation, text summarization, and dialogue generation.

Robotics: This area focuses on developing intelligent robots that can perform tasks in the real world. This can involve developing algorithms for planning, perception, and control, as well as designing the hardware and software for robot systems.

Computer vision: This area focuses on developing algorithms that can understand and interpret visual data. This can include tasks like image recognition, object detection, and scene understanding.

Neural networks: This area focuses on developing algorithms that are inspired by the structure and function of the human brain. This can involve training networks of interconnected nodes, called neurons, to learn from data and make predictions.

3.1.6. Uncertainty

An expert system based on rules is a type of AI program that uses a knowledge base of pre-defined rules to make decisions or provide recommendations. This type of system relies on the expertise of human experts to define the rules that the system will use to make decisions.

For example, let's say we are building an expert system to help diagnose medical conditions. The knowledge base for this system would be made up of a set of rules defined by doctors and medical professionals. These rules might include things like:

- If a patient has a fever and a cough, he/she may have the flu.
- If a patient has chest pain and difficulty breathing, he/she may be experiencing a heart attack.
- If a patient has swelling in the joints and a rash, he/she may have arthritis.

The expert system would use these rules to make recommendations to doctors, who can then use the information to diagnose patients and prescribe appropriate treatments.

An expert system based on rules is a type of artificial intelligence system that uses a set of rules to make decisions and solve problems. These rules are typically derived from the knowledge and expertise of human experts in a particular domain, such as medical diagnosis or financial analysis. For example, a medical expert system might use a set of rules to analyze symptoms and make a diagnosis based on the most likely condition. In this way, the expert system acts as a "virtual expert" by applying the same reasoning and decision-making processes as a human expert would.

In contrast, an expert system based on probability uses probabilistic models and algorithms to make decisions and solve problems. These systems often use data-driven approaches, such as machine learning and statistical analysis, to build models that can make predictions or recommendations based on the data they have been trained on. For example, a financial expert system might use historical stock market data to build a predictive model that can help investors make investment decisions.

One key difference between expert systems based on rules and those based on probability is the type of knowledge they use. Rule-based systems rely on explicit, pre-defined rules that are derived from human expertise, while probabilistic systems learn from data and can adapt to new situations. Another difference is the way these systems make decisions and solve problems. Rule-based systems use a top-down, logical approach to problem-solving, while probabilistic systems use a bottom-up, data-driven approach. Both approaches have their strengths and weaknesses, and which one is more suitable for a given problem depends on the specific context and requirements.

An expert system based on rules may use a set of pre-defined rules to reach a conclusion or make a decision. For example, an expert system for diagnosing medical conditions may use a set of rules to evaluate a patient's symptoms and determine a potential diagnosis. The rules may be defined by medical experts and may include statements such as "if a patient has a fever and a cough, they may have the flu" or "if a patient has chest pain and difficulty in breathing, they may have pneumonia".

On the other hand, an expert system based on probability may use statistical techniques to evaluate the likelihood of certain outcomes or conclusions. For example, an expert system for financial forecasting may use historical data and probability distributions to predict the future performance of a stock or portfolio. The system may use various algorithms to evaluate the probability of different outcomes, such as the likelihood that a stock will increase or decrease in value over a certain period of time.

For the same model, we could have an expert system based on rules that uses a set of pre-defined rules to diagnose medical conditions, and another expert system based on probability that uses statistical techniques to forecast the future performance of stocks. In the first system, the rules may be defined by medical experts and may include statements such as "if a patient has a

fever and a cough, they may have the flu" or "if a patient has chest pain and difficulty breathing, they may have pneumonia". In the second system, the system may use historical data and probability distributions to predict the future performance of a stock or portfolio, using algorithms to evaluate the likelihood of different outcomes.

3.2. Category theory and functional programming

In its essence, category theory is the study of composition. It is concerned with how objects and morphisms can be combined and related to each other logically and meaningfully. This kind of structured approach is frequently found in many branches of mathematics, and it has also been shown to have strong connections with logic and type theory through the use of Cartesian closed categories. In functional programming, some design concepts such as monads can be traced back to category theory. To gain a deeper understanding of these topics, one must first become familiar with the basic constructs that can be defined within a category or between categories. This can be achieved by exploring category theory and its connections to other areas of mathematics using the functional programming language Scala.

Category theory is a powerful mathematical tool that provides a common language for mathematicians from different fields to communicate and understand each other's work. By abstracting away the specific details of individual functions and focusing on their composition, category theory allows for a powerful and flexible way of thinking about mathematical concepts. In addition to its applications within mathematics, category theory has also found use in computer science. This thesis will explore the various concepts and applications of category theory using functional programming, specifically the Scala language. This will provide a foundation for further study and research in this area [61, 41].

3.2.1. Categories

Category theory is a fundamental mathematical framework that allows for the study of mathematical structures and their relationships through the use of categories and functors. At its core, a category Def: 3.2.1 is simply a collection of objects and arrows (morphisms) between those objects, with a defined composition of arrows. Some of the most interesting categories in mathematics are those that are related to specific mathematical structures, such as Top (the category of topological spaces and continuous functions) and Grp (the category of groups and homomorphisms). In these cases, category theory provides a unifying framework for different mathematical theories, allowing for the transfer of results and ideas between different branches of mathematics. Despite its simplicity, category theory has far-reaching conse-

quences and applications, making it an essential tool for modern mathematics. In this thesis, we will explore the various concepts and applications of category theory in more depth, providing a foundation for further study and research in this area.

Definition 3.2.1 (**Category**)**.** A category $\mathcal{C}$ is:

1. A collection of objects $\mathcal{O}(\mathcal{C})$.
2. For each pair of objects A, B in $\mathcal{O}(\mathcal{C})$, a collection of maps $C(A, B)$ of A to B. A morphism f on $C(A, B)$ will be denoted $f : A \to B$.
3. To each pair of maps $f : A \to B$, $g : B \to C$, a map of $g \circ f : A \to C$ called the composition morphism of f and g.

such that:

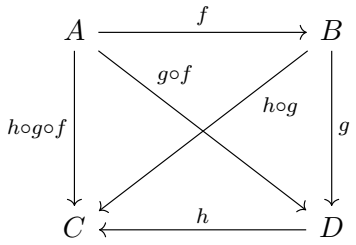
1. For every object $A \in O(C)$, there exists a map $idA \in C(A, A)$ that we call the identity morphism.
2. For every morphism $f : A \to B$: $idB \circ f = f \circ idA = f$
3. Associativity of the composition: For every triplet of morphisms $f : A \to B$, $g : B \to C$, $h : C \to D$:

$$(h \circ g) \circ f = h \circ (g \circ f) \tag{3.1}$$

In category theory, it is common to use the notation $\mathrm{Hom}(A, B)$ to refer to the collection of morphisms between A and B, which we have previously referred to as $C(A, B)$.

We often use diagrams to visually express certain properties. With the necessary tools we can give a precise definition of a diagram, for the moment it will suffice to define it as a graph formed by a finite number of objects connected to each other by morphisms of the same category.

The concept of commutativity of a diagram is the usual one from other fields. As an example, property (3.1) can be expressed by saying that the diagram 3.2 commutes.



(3.2)

Given a morphism $f : A \to B$, we will call the domain of f to be $\operatorname{dom} f = A$. Similarly, the codomain of f will be $\operatorname{cod} f = B$. In addition, we will refer to the collection of morphisms in a category as $M(C)$.

There is no shortage of examples of categories, even if we limit ourselves to a specific field of mathematics. Some illustrative examples are the following:

Example 2.1. The category Set formed by sets as objects and functions between sets as morphisms. The composition of morphisms corresponds, as expected, to the composition of functions.

Example 2.2. The category Grp is formed by groups as objects and group homomorphisms as morphisms. The composition of morphisms corresponds to the usual operation of composing group homomorphism.

Example 2.3. The category Top is formed by topological spaces as objects and continuous functions as morphisms. The composition of morphisms corresponds to the usual operation of composing continuous functions.

Example 2.4. The category Vect_k is formed by vector spaces over a fixed field k as objects and linear maps as morphisms. The composition of morphisms corresponds to the usual operation of composing linear maps.

Note. The morphisms need not be functions. In fact, the definition of a category only requires that there be a set of objects and a set of morphisms between each pair of objects, with a defined composition of morphisms. This allows for a wide range of possible examples of categories, some of which do not involve functions at all.

For example, consider the category Grp of groups, where the objects are groups and the morphisms are group homomorphisms. In this case, the morphisms are not functions, but rather maps between groups that preserve the group structure. Similarly, in the category Top of topological spaces, the morphisms are continuous functions, which are not necessarily functions in the usual sense.

Overall, the flexibility of the definition of a category allows for a wide range of examples and applications, many of which involve morphisms that are not functions.

Example 2.5. The category Ring is formed by rings as objects and ring homomorphisms as morphisms. The composition of morphisms corresponds to the usual operation of composing ring homomorphisms.

Example 2.6. The category Ab is formed by abelian groups as objects and group homomorphisms that preserve the abelian structure as morphisms.

The composition of morphisms corresponds to the usual operation of composing group homomorphisms

Example 2.8. The category FinSet is formed by finite sets as objects and functions between finite sets as morphisms. The composition of morphisms corresponds to the usual operation of composing functions. For example, diagram 3.3.

$$id_A \circlearrowright A \xrightarrow{\quad f \quad} B \circlearrowright id_B \tag{3.3}$$

Note. We will often use finite categories when discussing diagrams in later sections. On the other hand, the categories we will often work with will be very large, and in these cases, standard set theory can become unwieldy. For example, in standard set theory, the notion of the set of all sets leads to inconsistencies, as shown by Cantor's theorem. Therefore, we will often use the terms "collection" or "family" instead of "set" in general. This will allow us to avoid the pitfalls of working with very large sets in our discussions of category theory.

Example 2.9. The category Cat is formed by small categories as objects and functors between small categories as morphisms. The composition of morphisms corresponds to the usual operation of composing functors.

Example 2.10. The category PoSet is formed by partially ordered sets as objects and order-preserving functions as morphisms, for example: $f : a \to b$ **iif** $a \leq b$. The composition of morphisms corresponds to the usual operation of composing functions.

In some cases, it will be useful to consider a collection of objects or morphisms as a set. To do this, we introduce the following definitions:

Definition 3.2.2. A category $\mathcal{C}$ is said to be small if the sets of objects and morphisms in $\mathcal{C}$ are both sets. For instance, every finite category is small, but *Set* is not small.

Definition 3.2.3. . A category $\mathcal{C}$ is said to be locally small if, for every pair of objects A and B in $\mathcal{C}$, the set of morphisms from A to B in $\mathcal{C}$ is a set. All the examples we have given are locally small. We will need to introduce additional concepts to provide examples of categories that are not locally small. We will typically work with local small categories.

We can also construct categories from other categories. The most important example of this is the opposite category:

Definition 3.2.4. Given a category $\mathcal{C}$, the opposite category or dual $\mathcal{C}^{op}$ is the category with the same objects as $\mathcal{C}$, but where a morphism $f : B \to A$ in $\mathcal{C}^{op}$ is a morphism $f : A \to B$ in $\mathcal{C}$. Clearly, $(\mathcal{C}^{op})^{op} = \mathcal{C}$. We will frequently encounter related concepts by taking the dual of a category. For instance, the terminal object of a category (which we will discuss later) is just the initial object of its opposite category.

Definition 3.2.5. Let A and B be objects in a category $\mathcal{C}$. A morphism $f : B \to A$ in $\mathcal{C}$ is called a monomorphism (or monic) if, for every pair of morphisms $g : A \to B$ and $h : A \to B$ such that $f \circ g = f \circ h$, we have $g = h$.

Proposition 3.2.6. *In Set, a map is a monomorphism if and only if it is injective.*

Proof. *Suppose $f : B \to C$ is a monomorphism. Let b, $b_0 \in B$ be such that $f(b) = f(b_0)$. Let $A = b$ and define $g : A \to B$ as $g(b) = b_0$. Then $f(id_A(b)) = f(g(b))$ since b is the only element of A, we have: $f \circ id_A = f \circ g$ Hence, since f is a monomorphism, we have: $id_A = g$ Therefore, $b_0 = g(b) = id_A(b) = b$.*

Conversely, suppose $f : B \to C$ is an injective morphism. Let g, $h : A \to B$ be morphisms such that $f \circ g = f \circ h$. Let $a \in A$ be arbitrary. Since $f(g(a)) = f(h(a))$ and f is injective, we have $g(a) = h(a)$. That is, we have $g = h$.

Definition 3.2.7. A morphism $f : A \to B$ is called an epimorphism (or epic) if, for any pair of morphisms $g : B \to C$ and $h : B \to C$, we have $g \circ f = h \circ f$ implies $g = h$.

Proposition 3.2.8. *In Set, a map is an epimorphism if and only if it is surjective.*

Proof. *Suppose $f : A \to B$ is an epimorphism. Suppose f were not surjective. Then there exists $b \in B$, which is not the image of any element of A. Let $g, h : B \to 1, 2$ such that $g(x) = h(x) = 1$ for all $x \in B \backslash b$, $g(b) = 1$ and $h(b) = 2$. Then we have $g \circ f = h \circ f$. Since f is an epimorphism, this implies that $g = h$, but this is a contradiction.*

Let $f : A \to B$ be a surjective map. Let $g, h : B \to C$ such that $g \circ f = h \circ f$. For all $b \in B$, there exists $a \in A$ such that $f(a) = b$, then: $g(b) = g(f(a)) = h(f(a)) = h(b)$, so $g = h$.

Definition 3.2.9. A map $f : A \to B$ is isomorphism if there exists $g : B \to A$ such that $g \circ f = id_A$ and $f \circ g = id_B$.

Two objects A and B will be said to be isomorphic if there is an isomorphism between them.

Clearly, in *Set*, a monic and an epicmorphism is isomorphic. However, all the isomorphisms are not of this form. For example, in Set, the map

that sends all the odd natural numbers to zero and the rest to itself is an isomorphism (and therefore an epic and monic morphism), but it is not a monic epic morphism.

The following result will be fundamental in the sequel.

Proposition 3.2.10. *Let $\mathcal{C}$ be a category, and let A and B be objects in $\mathcal{C}$. If $f : A \to B$ is an isomorphism, then it is both a monomorphism and an epimorphism.*

Proof. *Let $g : B \to A$ be such that $g \circ f = id_A$ and $f \circ g = id_B$. Let h_1, $h_2 : A \to C$ be such that $f \circ h_1 = f \circ h_2$. Then:*
$g \circ f \circ h_1 = g \circ f \circ h_2$
Since $g \circ f = id_A$, we have: $h_1 = h_2$
Hence, f is a monomorphism.
Let g_1, $g_2 : C \to B$ be such that $g_1 \circ f = g_2 \circ f$. Then: $g_1 \circ f \circ g = g_2 \circ f \circ g$
Since $f \circ g = id_B$, we have: $g_1 = g_2$
Hence, f is an epimorphism.
It follows that if $f : A \to B$ is an isomorphism in a category C, then it is both a monomorphism and an epimorphism in C. However, the converse is not necessarily true: monic and epic morphisms may exist that are not isomorphisms.

We will often make use of the following result.

Proposition 3.2.11. *Let C be a category and let $f : A \to B$ and $g : B \to C$ be morphisms in C. If f is a monomorphism and g is an epimorphism, then $g \circ f$ is a monomorphism.*

Proof. *Let h_1, $h_2 : X \to A$ be such that $(g \circ f) \circ h_1 = (g \circ f) \circ h_2$. Then:*
$g \circ f \circ h_1 = g \circ f \circ h_2$
Since f is a monomorphism, we have:
$h_1 = h_2$
Therefore, $g \circ f$ is a monomorphism.

The following result is also frequently used.

Proposition 3.2.12. *Let $\mathcal{C}$ be a category and let $f : A \to B$ and $g : B \to C$ be morphisms in $\mathcal{C}$. If f is an epimorphism and g is a monomorphism, then $g \circ f$ is an epimorphism.*

Proof. *Let h_1, $h_2 : C \to X$ be such that $h_1 \circ (g \circ f) = h_2 \circ (g \circ f)$. Then:*
$h_1 \circ g \circ f = h_2 \circ g \circ f$
Since g is a monomorphism, we have:
$h_1 \circ g = h_2 \circ g$
Since f is an epimorphism, we have:
$h_1 = h_2$
Therefore, $g \circ f$ is an epimorphism.

3.2.2. Functors

It seems that many concepts, such as Set and Top, can be organized into categories. This raises the question: can we create a category of categories, where the objects are themselves categories? In order to do this, we must define morphisms between categories that preserve their internal structure. While the lack of sets of sets also prevents the existence of a category of categories, this idea leads to the concept of a functor.

Definition 3.2.13. A functor F between a pair of categories $\mathcal{C}$ and $\mathcal{D}$ is a pair of functions: F_O and F_M, where $F_O : \mathcal{O}(\mathcal{C}) \to \mathcal{O}(\mathcal{D})$ and $F_M : \mathcal{M}(\mathcal{C}) \to \mathcal{M}(\mathcal{D}))$, such that:

A functor must satisfy the following properties:

- It preserves the domain and codomain of morphisms: for any morphism $f : A \to B$, we have $F_M(f) : F_O(A) \to F_O(B)$.
- It preserves the identity morphism: $F_M(id_A) = id_{F_O(A)}$.
- It preserves the composition of morphisms: for any morphisms $f : A \to B$ and $g : B \to C$, we have $F_M(g \circ f) = F_M(g) \circ F_M(f)$.
- We will write $F : C \to D$ and often use F to refer to either F_M or F_O depending on the context.

As functors can be thought of as functions between categories, we can define composition of functors.

Definition 3.2.14. Given two functors $F : C \to D$ and $G : D \to E$, we can compose them to form a new functor $G \circ F : C \to E$. This functor has the following properties:

$(G \circ F)_O = G_O \circ F_O$ $(G \circ F)_M = G_M \circ F_M$

This defines the composition of two functors F and G.

3.2.3. Monoids and Monads

In algebra, a monoid $(M, \cdot)$ is a set M together with a binary operation $\cdot : M \times M \to M$ that is associative and has a unit element in M. An alternative way of viewing a monoid is as a category with a single object. If we call A the only object of this category, we can identify the elements of M with the morphisms $f : A \to A$ and the composition of morphisms with the binary operation. The identity element is identified with the identity map.

For example, the monoid $(\mathbb{N}, +)$ corresponds to the category:

In turn, for any locally small category $\mathcal{C}$, every object $A \in \mathcal{C}$ induces a monoid $\mathcal{C}(A, A)$.

3.2.3.1. Monoidal category

One way to define monoids in the context of category theory is through the concept of an object monoid in a category.

Definition 3.2.15 (Monoidal Category)**.** A monoidal category is a category $\mathcal{C}$ equipped with a bifunctor $\otimes : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ and a unit object $I \in \mathcal{C}$, along with natural isomorphism:

- The associate: $\alpha_{A,B,C} : A \otimes (B \otimes C) \to (A \otimes B) \otimes C$
- The left joiner: $\lambda_A : I \otimes A \to A$
- The right joiner: $\rho_A : A \otimes I \to A$

such that the following diagrams commute:

$$\begin{array}{ccccc} A \otimes (B \otimes (C \otimes D)) & \xrightarrow{\alpha} & (A \otimes B) \otimes (C \otimes D) & \xrightarrow[\alpha]{} & ((A \otimes B) \otimes C) \otimes D) \\ \downarrow{\scriptstyle id \otimes \alpha} & & & & \uparrow{\scriptstyle \alpha \otimes id} \\ A \otimes ((B \otimes C) \otimes D) & & \xrightarrow{\qquad\qquad\alpha\qquad\qquad} & & (A \otimes (B \otimes C)) \otimes D) \end{array}$$

$$\begin{array}{ccc} A \otimes (I \otimes C) & \xrightarrow{\qquad\alpha\qquad} & (A \otimes I) \otimes C \\ & {\scriptstyle id \otimes \lambda} \searrow \quad \swarrow {\scriptstyle \rho \otimes id} & \\ & A \otimes C & \end{array}$$

and finally: $\lambda I = \rho I$

For instance, the category of sets with the cartesian product forms a monoidal category, where the tensor product of two sets A and B is simply their cartesian product A x B. In a monoidal category, there exists a natural isomorphism between $A \otimes B$ and $B \otimes A$, which means that the order in which the sets are tensored does not affect the result. A monoidal category with this property is called a symmetric monoidal category.

Examples of symmetric monoidal categories include the category of sets with the cartesian product and the category of abelian groups with the tensor product. In the latter case, the tensor product of two groups A and B is the Z-module tensor product $A \otimes_Z B$. This is an abelian group with a bilinear product $\otimes$ that satisfies certain properties.

Note that The category of sets and the category of abelian groups are interesting examples to consider in the context of monoidal categories because they are both commonly used examples in the study of category theory. The category of sets, in particular, is often used as a starting point for introducing many of the basic concepts of category theory, such as the notion of a category, a functor, and a natural transformation. This makes it a useful example for illustrating the fundamental ideas of monoidal categories and how they relate to other concepts in category theory.

The category of abelian groups, on the other hand, is interesting because it is an example of a symmetric monoidal category that is not equivalent to the category of sets. This allows for a more in-depth exploration of the properties of symmetric monoidal categories and how they differ from other types of monoidal categories. Additionally, the tensor product of abelian groups is a well-studied and important concept in abstract algebra, making it a useful example for demonstrating the relevance and applications of monoidal categories in other areas of mathematics.

3.2.3.2. Enriched category

Let us remember that a category C is said to be locally small if for any two objects $A, B \in C$, the set of morphisms $C(A, B) \in sets$. An enriched category is a generalization of this concept, where the category of sets is replaced with any other symmetric monoidal category V. Informally, a category C is V-enriched if for any two objects $A, B \in C$, the set of morphisms $C(A, B) \in V$. For this definition to be valid, there must also be ceSrtain compatibility conditions with the composition in the category.

For example, the category of sets with the cartesian product forms a symmetric monoidal category V that could be used to enrich another category. In this case, a category C would be $V - enriched$ if for any two objects $A, B \in C$, the set of morphisms $C(A, B)$ is a set with a cartesian product operation. Similarly, the category of abelian groups with the tensor product forms another symmetric monoidal category V that could be used to enrich a category.

Definition 3.2.16. An enriched category, denoted by C, consists of the following:

- A collection of objects.
- For every pair of objects A, B in C, an object called a hom-object, denoted by $C(A, B)$, which belongs to a set V.
- For each A in C, a map called the identity map, denoted by $Id_A : I \rightarrow C(A, A)$, on V.

- For all A, B, C in C, a map denoted by $\circ : C(B,C) \otimes C(A,B) \to C(A,C)$ on V, such that the following diagrams commute:

$$\begin{array}{ccc} \mathcal{C}(C,D) \otimes \mathcal{C}(B,C) \otimes \mathcal{C}(A,B) & \xrightarrow{id \otimes \circ} & \mathcal{C}(C,D) \otimes \mathcal{C}(A,C) \\ \downarrow{\scriptstyle \circ \otimes id} & & \downarrow{\scriptstyle \circ} \\ \mathcal{C}(B,D) \otimes \mathcal{C}(A,B) & \xrightarrow{\quad\circ\quad} & \mathcal{C}(A,D) \end{array}$$

$$\begin{array}{ccc} \mathcal{C}(A,B) \otimes I & \xrightarrow{id \otimes id_A} & \mathcal{C}(A,B) \otimes \mathcal{C}(A,A) \\ & \searrow^{\cong} & \downarrow{\scriptstyle \circ} \\ & & \mathcal{C}(A,B) \end{array}$$

$$\begin{array}{ccc} \mathcal{C}(B,B) \otimes \mathcal{C}(A,B) & \xleftarrow{id_B \otimes id} & I \otimes \mathcal{C}(A,B)) \\ \downarrow{\scriptstyle \circ} & \swarrow_{\cong} & \\ \mathcal{C}(A,B) & & \end{array}$$

As an example, let us consider the category of small categories, denoted by Cat, equipped with a Cat-category structure, that is, a 2-category.

Definition 3.2.17. In a monoid category $(C, \otimes, I)$, a monoid is an object M equipped with two maps $\mu : M \otimes M \to M$ and $\eta : I \to M$ that satisfy the following two properties:

The associative property: for all $m_1, m_2, m_3 \in M$, the following equation holds:

$$\begin{array}{ccc} ((M \otimes M) \otimes M) & \xrightarrow{\quad\alpha\quad} & (M \otimes (M \otimes M)) \\ \downarrow{\scriptstyle \mu \otimes id} & & \downarrow{\scriptstyle id \otimes \mu} \\ (M \otimes M) & & (M \otimes M) \\ & \searrow^{\mu} \quad \swarrow_{\mu} & \\ & M & \end{array}$$

The unit property: for all $m \in M$, the following equations hold:

$$\begin{array}{ccccc} (I \otimes M) & \xrightarrow{\eta \otimes id} & (M \otimes M) & \xleftarrow{id \otimes \eta} & (M \otimes I) \\ & \searrow^{\rho} & \downarrow{\scriptstyle \mu} & \swarrow_{\rho} & \\ & & M & & \end{array}$$

Some examples of monoids include:

Monoids in the algebraic sense, which are objects in the monoid category $(\mathrm{Set}, \times, 1)$.

Rings, which are monoids in the monoidal category of abelian groups $(\mathrm{Ab}, \otimes, \mathbb{Z})$ described in Example 5.1.

k-algebras, which are monoids on the monoid category of k-vector spaces $(\mathrm{Vect}_k, \otimes_k, k)$.

One more example of particular importance to us is monads.

3.2.3.3. Monad

Given a category $\mathcal{C}$, we consider the category $\mathcal{C}^{\mathcal{C}}$ where the objects are endofunctors and morphisms are given by the composition between functors.

Proposition 3.2.18. *The category $\mathcal{C}^{\mathcal{C}}$ forms a monoidal category with the composition as tensor product and the identity functor* Id *as unit object.*

Proof. *This is a consequence of $\mathcal{C}$ being a 2-category. Therefore, we can define monoids on $\mathcal{C}^{\mathcal{C}}$. Such monoids are called monads.*

Therefore, we can define monoids on $\mathcal{C}^{\mathcal{C}}$. These monoids are called monads. Let's try to give a more explicit definition, specializing the definition of monoid taking into account that the composition operation $\circ$ is associative and the identity functor Id satisfies the identity properties:

- $\circ$ is associative
- Identity functor satisfies:

$$F \circ \mathrm{Id} = F$$
$$\mathrm{Id} \circ F = F$$

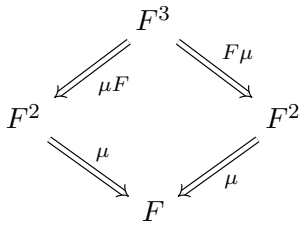
- The bifunctor $\circ$ acting on natural transformations is the horizontal composition $\star$.
- For the natural identity transformation $\iota : F \Rightarrow F$:

$$\mu \star \iota = \mu F$$
$$\iota \star \mu = F\mu$$

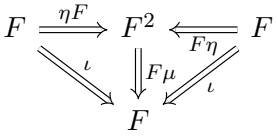
- We will write F^n to refer to $F \circ \cdots \circ F$ (n times).

Definition 3.2.19. A monad is an endofunctor F of a category $\mathcal{C}$ with two natural transformations $\mu : F^2 \Rightarrow F$ and $\eta : \mathrm{Id} \Rightarrow F$ that satisfy the following conditions:

- The associative property:



- Unit property:



Example Let $\mathcal{C}$ be the category of sets and let $T : \mathcal{C} \to \mathcal{C}$ be the functor that takes a set X to the set $TX = X \times 0, 1$. The natural transformations $\mu : T^2 \to T$ and $\eta : I \to T$ are defined as follows:

$$\mu(X) = (x, (0, b)), (x, (1, b)) \mid x \in X, b \in 0, 1$$
$$\eta(X) = X \times 1$$

It is easy to check that the triple (T, μ, η) satisfies the monad laws, so (T, μ, η) is a monad.

3.2.4. Scala and Spark

Functional programming [13] is a programming style that originated with IPL [8] and Lisp [59]. In recent years, it has gained more practical interest and has been adopted by popular languages such as C++ and Java. Some languages, like Scala [1], Haskell and Standard ML, were designed with functional programming in mind and are inspired by mathematics. Key features of functional programming include pure functions, a strong type system, polymorphic functions, algebraic data types, and lazy evaluation.

- Pure functions: These are functions that do not have any side effects and always return the same output for a given input.

- Strong type system: Functional programming languages often have a strict and static type system, which can help prevent runtime errors and to detect mistakes in the code.

[1] www.scalalang.org

- Polymorphic functions: These are functions that can operate on multiple types, allowing for greater flexibility and reusability. As a consequence of the previous point, it is possible to write generic functions in functional programming languages like Haskell and Scala. For instance, in Haskell, the type declaration $f :: forall\ a.to- > to$ describes f as a function that can be applied to all types and returns the same output type. In Scala, this could be expressed as follows, Fig. 3.4.
- Functional programming languages also allow one to impose constraints on polymorphic types through mechanisms like type classes in Scala, which are similar to interfaces in object-oriented languages like Java. Additionally, it is possible to impose constraints on the polymorphic return type of a function, as shown in the following example, Fig. 3.3.
- Algebraic data types: These are data types that can be defined in terms of other data types, allowing one to create complex and "expressive types".
- Lazy evaluation: This is a technique where expressions are only evaluated when they are needed rather than being immediately assessed. This can improve performance and make working with large or infinite data sets easier.

```
1 def read[A](str: String)(implicit ev: Read[A]): A = {
2     // Function body
3 }
```

Figure 3.3: Polymorphic function

In Fig. 3.3, the implicit ev: Read parameter is used to impose the constraint that the type A must be a member of the Read class. This allows the read function to interpret the input string as a value of type A.

```
1 def f[A](arg: A): A = {
2     // function body
3 }
```

Figure 3.4: Example of Scala function

Scala is a programming language that combines the best features of both functional and object-oriented programming. It is a statically-typed language, which means that the type of a variable must be specified at compile time, rather than at runtime.

Spark is a popular open-source distributed computing platform that is built on top of the Scala programming language. It provides a unified API for working with large datasets, and can be used to process data in a distributed and parallel manner.

One of the key advantages of Scala and Spark over other functional programming languages is their ability to scale to large datasets and distributed environments easily. Scala's static type system and functional programming features make it well-suited for working with distributed data, and Spark's API makes it easy to parallelize and distribute data processing tasks.

Another advantage of Scala and Spark is their support for data science and machine learning. Scala provides a rich set of libraries and tools for working with data, and Spark's in-memory computing capabilities make it well-suited for building and training machine-learning models on large datasets.

Overall, Scala and Spark are powerful tools for working with large datasets and building distributed and scalable applications. Their combination of functional programming features, strong type system and support for data science and machine learning make them an attractive choice for many developers and data scientists.

3.2.4.1. Introduction to Scala

We will introduce the minimum amount of Scala necessary to understand the code given here. We will not have a complete picture of the language, but enough to be able to follow the code. If you are already familiar with Scala, you can skip this section.

Scala allows us to declare a variable and its type, Fig. 3.5.

```
1 val x: Int = 8
```

Figure 3.5: Example of variable definition in Scala

It obviously implements basic arithmetic, Fig. 3.6.

```
1 val y = (x + 4) * 5
2 // This is a comment
3 // y = 60
```

Figure 3.6: Example of arithmetic

Functions are defined in a similar way, Fig. 3.7.

```
def double(x: Int): Int = 2 * x
```

Figure 3.7: Example of function definition

Functions with multiple parameters are defined as higher-order functions, Fig. 3.8.

```
def multiply(x: Int, y: Int): Int = x * y
```

Figure 3.8: Higher-order function

We can understand that multiply takes two integers and multiplies them, but we can also interpret that multiply is applied to x and returns a function with type $Int \rightarrow Int$, which we can then apply to the other argument. This idea leads us to the concept of partially applied functions, Fig. 3.9.

```
val double = multiply(2, _)
```

Figure 3.9: Curryling

We can define polymorphic functions, as explained in the previous section, Fig. 3.10.

```
def max[A](x: A, y: A)(implicit ord: Ordering[A]): A =
  if (ord.lt(x, y)) x else y
```

Figure 3.10: Polymorphic function

Here we also see the usual if-then-else conditional structure. We will often work at the level of functions, so it will be useful to look at the composition operator, Fig. 3.11.

```
def compose[A,B,C](f: B => C, g: A => B): A => C =
  x =>
```

Figure 3.11: Compose function

Scala also supports algebraic data types, which are defined with an expression of the form:

```
sealed trait Dato[T1, ..., Tn]
case class Constructor1[A1, ..., Am](a11: A1, ..., a1m: Am)
    extends Dato[T1, ..., Tn]
...
case class ConstructorM[AM1, ..., AMq](aM1: AM1, ..., aMq: AMq)
    extends Dato[T1, ..., Tn]
```

Figure 3.12: Algebraic data types

Here, $T_1, ..., T_n$ are arguments of the *typeDato*, which has different constructors $Constructor_1, ..., Constructor_M$, each with its corresponding parameters.

To better understand this, let's see an example:

```
sealed trait Maybe[A]
case class Nothing[A]() extends Maybe[A]
case class Just[A](a: A) extends Maybe[A]
```

Figure 3.13: Nothing, just and maybe fundamental type in Scala.

The data type Maybe[A] has two constructors: Nothing (with no parameters) and Just[A], with parameter A. Recall that in functional programming, constructors carry all the information about the data type, so we can identify any object of type Maybe[A] with one of its constructors, Fig. 3.13.

This leads us to the interpretation that an object of type Maybe[A] is either *Nothing* or Just[A]. But let us be careful here, *Nothing* and Just[A] are not objects, but constructors, so we cannot use them directly. We can use them to create objects of the corresponding type, Fig. 3.14.

```
val nothing: Maybe[Int] = Nothing[Int]()
val just: Maybe[Int] = Just[Int](8)
```

Figure 3.14: Using constructors

To access the value of a Just, we need to use pattern matching, Fig. 3.15

```
def getValue(m: Maybe[Int]): Int = m match {
  case Just(x) => x
  case _ => 0
}
```

Figure 3.15: Pattern matching

Here we see that case $Just(x)$ matches any object of type Maybe[Int]

that was constructed with the Just constructor, and binds the value of the parameter A of Just[A] to the variable x. The pattern case matches any other value and returns 0.

In the next section, we will see how to use these concepts to define categorical structures in Scala. We will follow the definitions and notation introduced in the previous section, but we will use Scala code to define the categorical structures.

First, we define the Category trait/class, which represents a category, Fig. 3.16.

```
1 trait Category[Ob, Mor] {
2   def id(a: Ob): Mor
3   def compose(f: Mor, g: Mor): Mor
4 }
```

Figure 3.16: The category trait

Here, Ob and Mor are the types of objects and morphisms in the category, respectively. The Category trait defines two methods: id, which returns the identity morphism for a given object, and compose, which composes two morphisms.

We can now define a category by implementing the Category 3.16 trait, Fig. 3.17

```
1 object Set extends Category[Int, (Int, Int) => Int] {
2   def id(a: Int): (Int, Int) => Int = (x, y) => x
3   def compose(f: (Int, Int) => Int, g: (Int, Int) => Int): (Int
      , Int) => Int =
4     (x, y) => f(g(x, y), y)
5 }
```

Figure 3.17: Category example

Here we define the Set category, where the objects are integers and the morphisms are functions of type $(Int, Int) \to Int$. The id function returns a function that takes two integers and returns the first one, which represents the identity morphism in this category. The compose function composes two morphisms by applying the first one to the result of the second one and the second argument.

We can now use the Set category to define morphisms and perform operations on them, Fig. 3.18.

```
val f: (Int, Int) => Int = (x, y) => x + y
val g: (Int, Int) => Int = (x, y) => x * y

val h = Set.compose(f, g)
// h: (Int, Int) => Int = <function2>

val i = Set.id(3)
// i: (Int, Int) => Int = <function2>
```

Figure 3.18: Composition in Scala

Here we define two morphisms f and g, and then we compose them by using the compose method of the *Set* category. We also use the id method to get the identity morphism for the object 3.

3.2.4.2. Functors in Scala

In the context of type-safe programming, Scala is a statically-typed language, which means that it checks the types of variables and expressions at compiling time, rather than at runtime. This allows the compiler to catch type errors before the program is executed, making the code safer and more predictable.

As a statistically-typed language, Scala can work with a wide range of types, including primitive types (such as integers, floating-point numbers, and booleans), user-defined types (such as classes and objects), and composite types (such as arrays, tuples, and collections). It also has support for parametric polymorphism, which allows types to be defined as generic parameters that can be instantiated with different types at different points in the program.

Overall, the specific types that Scala can use will depend on the context in which the code is being written and the requirements of the program. However, as a general rule, Scala is capable of working with a wide range of types, which makes it a powerful and flexible language for type-safe programming.

Scala implements functors through its typeclasses. For example, Fig. 3.19.

```
trait Functor[F[_]] {
  def map[A, B](fa: F[A])(f: A => B): F[B]
}
```

Figure 3.19: Functor

Here, F acts as both F and FO. If you're uncertain of what map does in this context, it can be helpful to add some type parameters in the appropriate place, Fig. 3.20.

```
def map[A, B, F[_]](fa: F[A])(f: A => B): F[B]
```

Figure 3.20: Functor typed

This shows that map takes any function of type A =>B and transforms it into a function of type F[A] =>F[B]. This is precisely what we previously called FM. With this in mind, let's adapt the laws of functors to Scala, Fig. 3.21

```
def map[A, B, F[_]](fa: F[A])(id: A => A): F[A] = fa
def map[A, B, C, F[_]](fa: F[A])(f: A => B, g: B => C): F[C] =
    map(map(fa)(f))(g)
```

Figure 3.21: Functor typed

3.2.4.3. Monoid in Scala

In Scala, you can work with monoids by using the Monoid trait from the cats library. This trait defines several methods that a monoid must implement, such as combine for combining two values of the monoid and empty for the neutral element of the monoid.

Here is an example of how to use the Monoid trait to work with a monoid in Scala, Fig. 3.22

```
import cats.Monoid

// Define a monoid for strings that concatenates strings and
    has the empty string as the neutral element
implicit val stringMonoid: Monoid[String] = new Monoid[String]
    {
  def combine(x: String, y: String): String = x + y
  def empty: String = ""
}

// Use the monoid to combine two strings
val x = "Hello"
val y = "World"
val z = Monoid[String].combine(x, y)
// z = "HelloWorld"

// Use the monoid to combine a list of strings
val strings = List("foo", "bar", "baz")
val combined = Monoid[String].combineAll(strings)
// combined = "foobarbaz"
```

Figure 3.22: Monoid

In this example, we define an implicit Monoid instance for strings that concatenates strings using the + operator and has the empty string as the neutral element. We then use this instance to combine two strings and a list of strings using the *combine* and *combineAll* methods from the Monoid trait.

3.2.4.4. Functional Programming in Machine Learning

Functional programming is a programming paradigm that is based on the evaluation of mathematical functions. It is a declarative programming style that focuses on what the program should do, rather than how it should do it. In functional programming, functions are first-class citizens, which means that they can be passed as arguments to other functions and returned as results from functions.

Spark is a popular open-source distributed computing framework that is widely used for data processing and analytics. It is written in the Scala programming language, which is a functional programming language that runs on the Java Virtual Machine (JVM). Spark provides a high-level API for functional programming in Scala, which allows developers to write distributed programs in a concise and expressive way.

Functional programming has several advantages that make it particularly well-suited for use in Artificial Intelligence (AI) applications.

1. First, functional programming languages are typically declarative rather than imperative, which means that they focus on what the desired outcome should be rather than on how to achieve it. This is particularly useful in AI, where the goal is often to find the optimal solution to a problem rather than to specify the exact steps to take to solve it.

2. Second, functional programming languages are generally more concise and easier to read than imperative languages, which makes it easier to express complex algorithms and logical rules. This can be especially useful in AI, where many algorithms are based on complex mathematical equations or logical rules that can be difficult to express in other programming languages.

3. Third, functional programming languages are often more expressive than imperative languages, which means that they allow developers to express complex algorithms and logical rules more concisely and more clearly. This can be especially useful in AI, where many algorithms are based on complex mathematical equations or logical rules that can be difficult to express in other programming languages.

4. Fourth, functional programming languages are often more modular and composable than imperative languages, which means that they can

be easily combined with other languages or libraries to create more complex AI applications. This can be especially useful in AI, where many algorithms require the use of multiple languages or libraries to achieve their desired outcome.

In summary, functional programming is advantageous in AI because it allows developers to express complex algorithms and logical rules more concisely, more clearly, and more modularly than they could be using imperative languages. This can make it easier to develop, maintain, and debug AI applications and can also make it easier to combine AI algorithms with other languages or libraries to create more complex AI applications.

In the context of machine learning, functional programming with Spark can be used to build and train machine learning models in a scalable and efficient manner. Spark provides a set of machine-learning libraries that support various algorithms and techniques for classification, regression, clustering, and recommendation. These libraries can be used in combination with the functional programming features of Scala to develop machine learning pipelines that can process large datasets in parallel on a cluster of machines.

Introduction to Spark

Spark is a popular open-source distributed computing platform that is built on top of the Scala programming language. It provides a unified API for working with large datasets, and can be used to process data in a distributed and parallel manner.

To use Spark in Scala, one first needs to add the Spark dependencies to your project. Here is an example of how to do this using the sbt build tool, Fig. 3.23.

```
1 libraryDependencies += "org.apache.spark" %% "spark-core" % "
    2.4.5"
```

Figure 3.23: Load spark library

Once the dependencies are added, you can create a SparkContext and start working with Spark. Here is an example of how to create a SparkContext and read in a dataset:

```
1 import org.apache.spark.SparkContext
2
3 val sc = new SparkContext("local[*]", "My Spark App")
4
5 val data = sc.textFile("/path/to/data.csv")
```

Figure 3.24: Load data

Once one has a SparkContext and a dataset, he can use the various Spark APIs to transform and analyze the data. For example, one can use the map and filter functions to transform the data, or use the reduce function to perform aggregations.

Overall, Spark provides a powerful and easy-to-use API for working with large datasets in Scala. Its support for distributed and parallel processing allows you to quickly and efficiently analyze and process your data.

A brief example of how you could use Spark to predict churn for a telecom company's clients. For example, to train a logistic regression model on a dataset using the Spark MLlib library, we can use the following code, by steps: read data, process data, train the model and validate.

First, you would need to create a SparkContext and read in the dataset containing information about the telecom company's clients Fig. 3.24.

Next, you would need to clean and prepare the data for modeling. This might involve tasks such as removing missing values, transforming categorical variables into numerical ones, and scaling the data. Here is an example of how you could do this using the Spark API, Fig. 3.25.

```
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.feature.StandardScaler

// Remove rows with missing values
val cleanedData = data.na.drop()

// Transform categorical variables into numerical ones
val transformedData = cleanedData.map(row => {
  // Create a new Row object with the transformed values
  Row(
    row.getAs[String]("client_id").hashCode.toDouble,
    row.getAs[String]("plan").hashCode.toDouble,
    row.getAs[String]("service_type").hashCode.toDouble,
    row.getAs[Double]("monthly_charges"),
    row.getAs[Double]("num_calls"),
    row.getAs[Double]("num_texts"),
    row.getAs[Double]("data_usage")
  )
})

// Scale the data using StandardScaler
val scaler = new StandardScaler()
  .setInputCol("features")
  .setOutputCol("scaled_features")

val Array(train, test) = transformedData.randomSplit(Array(0.7,
    0.3))

scaler.fit(train)
val trainScaled = scaler.transform(train)
val testScaled = scaler.transform(test)
```

Figure 3.25: Data processing in spark

In this code, the VectorAssembler have been used as transformer to combine the individual features of the dataset into a single feature vector, randomSplit have been used to split the data in train and test and StandardScaler have been using to scale data.

```
import org.apache.spark.ml.classification.LogisticRegression

val model = new LogisticRegression().setMaxIter(100).fit(train)

set val prediction = model.transform(test)
val accuracy = prediction.filter("prediction == label").count()
    / test.count()
```

Figure 3.26: Modeling in spark

Finally, the LogisticRegression class is used to train the model. The trans-

form method evaluate the model on the testing set, and compute the accuracy of the predictions.

In the context of category theory, a pipeline is a sequence of computations that transform input data into output data. A monad is a mathematical structure that represents a sequence of computations, so it can be used to model a pipeline.

For example, consider a pipeline that applies multiple machine learning algorithms to a dataset. The input data is transformed by each algorithm in the pipeline, and the output data is passed on to the next algorithm in the pipeline. We can define a monad for this pipeline as follows, Fig. 3.27.

```
1
2 type M = Dataset unit Î·(x: Dataset) = x bind Î¼(x: Dataset, f:
      Dataset => Dataset) = f(applyAlgorithms(x))
```

Figure 3.27: Pipeline as a Monad

Dataset is the type of the input and output data, *applyAlgorithms* is the function that applies the machine learning algorithms to the input data and produces the output data, and f is a function that takes the output data and produces the final result.

This allows to chain together multiple computations in a sequence, where the output of one computation is used as the input of the next computation. This makes it easier to analyze the properties and behavior of the pipeline in a general and abstract setting. For example, in the case of a a pipeline in Scala that applies data cleaning, feature engineering, normalization, feature selection, dimensional reduction, model training with cross-validation, and prediction over the objective variable.

```
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.feature.{StringIndexer, VectorAssembler, StandardScaler, ChiSqSelector, PCA}
import org.apache.spark.ml.classification.LogisticRegression
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
import org.apache.spark.ml.tuning.{ParamGridBuilder, CrossValidator}

// Define the input and output columns
val inputCols = Array("col1", "col2", ...)
val outputCol = "target"

// Create the data cleaning, feature engineering, and normalization stages
val dataCleaning = new DataCleaning().setInputCols(inputCols).setOutputCol("cleaned")

val featureEngineering = new FeatureEngineering().setInputCol("cleaned").setOutputCol("engineered") val normalization = new StandardScaler().setInputCol("engineered").setOutputCol("normalized")

// Create the feature selection and dimensional reduction stages

val featureSelection = new ChiSqSelector().setNumTopFeatures(10).setFeaturesCol("normalized") .setLabelCol(outputCol).setOutputCol("selected")

val dimensionalReduction = new PCA().setK(5).setInputCol("selected").setOutputCol("reduced")

// Create the StringIndexer for the categorical output variable
val indexer = new StringIndexer() .setInputCol(outputCol) .setOutputCol("indexed")

// Create the classification algorithm val lr = new LogisticRegression() .setFeaturesCol("reduced") .setLabelCol("indexed") .setPredictionCol("prediction")

// Create the CrossValidator and Evaluator

val cv = new CrossValidator() .setEstimator(lr) .setEvaluator(new // Fit the pipeline to the training data val model = pipeline.fit(train) // Make predictions on the test data val predictions = model.transform(test)
MulticlassClassificationEvaluator()) .setEstimatorParamMaps(new ParamGridBuilder().addGrid(lr.regParam, Array(0.1, 0.01, 0.001)) .build()) .setNumFolds(5)

// Create the Pipeline val pipeline = new Pipeline() .setStages(Array(dataCleaning, featureEngineering, normalization, featureSelection, dimensionalReduction, indexer, cv))

// Fit the pipeline to the training data
val model = pipeline.fit(train)

// Make predictions on the test data val predictions = model.transform(test)
```

Figure 3.28: Data Pipeline as a Monad

Pipeline as a Monad: This pipeline applies the specified data cleaning, feature engineering, normalization, feature selection, dimensional reduction, and classification algorithms to the input data, and outputs the predicted labels for the test set.

Functional programming with Spark allows us to express these operations in a concise and composable way, and to leverage the power of distributed computing to train machine learning models on large datasets.

Functional programming is a powerful paradigm for building machine learning pipelines, because it allows developers to express complex operations in a declarative and composable way. This makes it easier to write, debug, and maintain machine learning programs, and enables the development of reusable and modular code that can be easily extended and adapted to different scenarios.

In addition, functional programming provides strong guarantees of correctness, because it relies on immutable data structures and pure functions. This means that the behavior of a functional program can be predicted and verified based on its inputs and outputs, without considering the order in which the operations are executed or the internal state of the program. This can help to avoid common errors and bugs in machine learning programs, and to ensure that the results of the program are consistent and reproducible.

3.2.4.5. Category theory in Machine Learning

Category theory is an important mathematical discipline that provides a general framework for studying the structure and properties of mathematical objects and their transformations. In the context of machine learning, category theory can be used to formalize and analyze the relationships between different machine learning algorithms and their underlying mathematical structures.

One important application of category theory in machine learning is in the study of semi-supervised learning, which is a technique for training machine learning models on datasets with incomplete or noisy labels. Category theory provides a formal framework for understanding the mechanisms of semi-supervised learning 3.3.5 [4], and for analyzing the performance of different algorithms in this context.

Another important application of category theory in machine learning is in the study of self-organizing maps, which are neural network models that are used for clustering and visualization of high-dimensional data. Category theory provides a mathematical framework for understanding the structure and behavior of self-organizing maps.

For example, consider a semi-supervised learning scenario in which we want to train a classifier on a dataset that contains both labeled and unlabeled examples. Using category theory, we can model this scenario as a functor

from a category of labeled examples to a category of classifiers, where the functor maps each labeled example to the classifier trained on that example.

This functor can be composed with other functors that represent different operations on the labeled examples, such as data augmentation, regularization, or feature selection. The resulting composite functor represents the overall semi-supervised learning pipeline, and its properties and behavior can be analyzed using the tools of category theory. As we can see in Fig. 3.25.

Similarly, in the context of self-organizing maps, category theory can be used to model the relationships between different neural network architectures and learning algorithms. For example, we can define a category of self-organizing maps where the objects are the different architectures, and the morphisms are the different learning algorithms.

This category can be equipped with additional structure, such as a metric or a topology, that captures the properties of the self-organizing maps and the behavior of the learning algorithms. This allows us to compare different self-organizing map models and learning algorithms, and to study their performance and properties in a rigorous and general way.

Overall, category theory provides a powerful mathematical framework for understanding and analyzing the structure and behavior of machine learning algorithms, and for designing and implementing efficient and effective machine learning pipelines.

3.3. Big Data

3.3.1. What is Big Data?

The term *Big Data* refers to the management and analysis of large and complex data sets that exceed the capabilities of traditional databases and data management tools. These data sets are often unstructured, coming from a variety of sources such as web-based resources, videos, and text documents, and may be stored in different repositories, both within and outside an organization. The challenge of Big Data lies not only in the sheer volume of data, but also in the difficulty of extracting value from it due to its poorly structured format.

To address these challenges, Big Data technologies have been developed to enable organizations to process and analyze large volumes of data in real time. These technologies include distributed storage systems, parallel processing frameworks, and advanced algorithms for data mining and machine learning. As the volume, variety and collection velocity of data continue to increase, the importance of Big Data technologies will continue to grow, providing organizations with the tools and insights they need to make better decisions and drive innovation.

3.3.2. Why has data become massive?

The modern world is full of data sources, including sensors, social media feeds, weather stations, and mobile devices. The proliferation of these technologies and the growing use of the internet has enabled the collection of large volumes of data across many fields of human activity, including research and development, commerce, and government. This abundance of data has created opportunities for new insights and innovations, but also poses challenges for storage, management, and analysis. As a result, the field of Big Data has emerged to address these challenges and enable organizations to extract value from their data,

3.3.3. Big Data today

Businesses collect and use a wide variety of structured and unstructured data. According to the Cisco Connected World Technology Report, the main sources of data, ranked by percentage of companies using them, are: 74 % current data collection, 55 % historical data collection, 48 % data from monitors and sensors, and 40 % real-time data. Regarding real-time data, the highest usage is in India, where 62 % of companies use this type of data, the United States (60 %), and Argentina (58 %). Additionally, 32 % of respondents collect unstructured data, such as video. China leads with 56 % of respondents collecting unstructured data in this area.

The sheer volume of data can be staggering. For example, sensors on aircraft engines can generate 10 terabytes of data in just half an hour. Similarly, drilling equipment and oil refineries produce large data streams. Even a simple service like Twitter, with its 140-character message limit, generates 8 terabytes of data per day. If all of this data is collected and stored for further analysis, the total amount can be measured in petabytes.

The Cisco Connected World Technology Report surveyed 1,800 university students and young professionals between the ages of 18 and 30 across 18 countries to assess the readiness of IT departments to implement Big Data projects. The study found that while many companies collect, record, and analyze data, they face complex business and IT challenges when it comes to Big Data. For instance, 60 % of respondents said that Big Data solutions could improve decision-making processes and increase competitiveness, but only 28 % reported already benefiting from their data.

In addition, the different fields of application are being specified and with them the tools, generating various branches within this field, and tools:

- Distributed databases and file systems: These technologies allow data to be stored and processed across multiple computers, allowing for faster processing of large datasets. Examples include Hadoop, Apache Spark (Section 3.2.4), and Apache Flink.

- Data lakes: A data lake is a central repository for storing and managing large amounts of structured and unstructured data. Examples include Amazon S3 and Microsoft Azure Data Lake.
- NoSQL databases: Unlike traditional relational databases, NoSQL databases are designed for storing and processing large amounts of unstructured data. Examples include MongoDB and Apache Cassandra.
- Real-time streaming platforms: These technologies enable the processing of data as it is generated, allowing for real-time analysis and decision making. Examples include Apache Kafka and Apache Storm.
- Machine learning and artificial intelligence (Section 3.3.5): These technologies are used to automatically identify patterns and insights in large datasets, often with the goal of predicting future events or making decisions. Examples include TensorFlow and PyTorch.

3.3.4. Applications

Big Data refers to the large, complex sets of data that are generated by businesses and other organizations. These data sets are often too large and complex to be processed using traditional data analysis techniques. However, advances in technology have made it possible for businesses to collect, store, and analyze this data in order to gain insights and make better decisions.

Big Data is being used by businesses in a variety of ways. For example, retailers can use Big Data to analyze customer behavior and preferences in order to improve their sales and marketing strategies. Banks and other financial institutions can use Big Data to improve their risk management and fraud detection capabilities. And government agencies can use Big Data to improve the delivery of public services, such as healthcare and education.

Overall, the use of Big Data can help businesses to improve their operations and better serve their customers. By analyzing large amounts of data, businesses can identify trends, patterns, and insights that can help them make more informed decisions and allocate resources more efficiently. This can ultimately lead to increased profitability and competitiveness.

Big Data has become an important tool for retailers to analyze customer behaviour and preferences in order to increase sales and improve customer loyalty. This is done by collecting data from various sources, such as social media and e-commerce platforms, and using it to tailor product offerings and marketing campaigns to individual customers. In addition, Big Data can help retailers reduce costs and reach a wider audience. Despite the potential risks associated with the adoption of new technologies, many companies are implementing Big Data solutions in order to stay competitive in a difficult economic enviroment. The development of mobile technologies and “Omni channel” systems, which allow customers to seamlessly switch between online

and in-store shopping, is also a major trend in the retail industry. Overall, the use of Big Data in the retail industry is expected to continue to grow in the coming years.

Big Data has become an important tool in the retail industry, allowing companies to analyze customer behaviour and preferences in order to increase sales and improve customer loyalty. This is done by collecting data from various sources, such as social media and e-commerce platforms, and using it to tailor product offerings and marketing campaigns to individual customers. Big Data can also help retailers reduce costs and reach a wider audience. Despite the potential risks associated with the adoption of new technologies, many companies are implementing Big Data solutions in order to stay competitive in a difficult economic environment. Mobile technologies are also playing an increasingly important role in the retail industry, as more and more consumers use their smartphones for shopping and other transactions. The development of "Omni channel" systems, which allow customers to seamlessly switch between online and in-store shopping, is also a major trend in the retail sector. Overall, the use of Big Data in the retail industry is expected to continue to grow, as companies seek to improve their operations and better serve their customers.

In addition to the retail industry, Big Data is also being used in other areas, such as public services. For example, government agencies are using Big Data to improve the delivery of services to citizens, such as healthcare and education. By analyzing large amounts of data, public sector organizations can identify trends and patterns that can help them make more informed decisions and allocate resources more efficiently. In the healthcare sector, Big Data is being used to improve patient care, by identifying risk factors and developing personalized treatment plans. In education, Big Data is being used to evaluate the effectiveness of teaching methods and to identify areas where students may need extra support.

Another area where Big Data is being used is finance and banking. Banks and other financial institutions are using Big Data to improve their risk management and fraud detection capabilities. By analyzing large amounts of data, they can identify unusual patterns and transactions that may indicate potential fraud or other risks. In addition, Big Data is being used to improve the customer experience, by providing personalized financial advice and recommendations based on individual needs and preferences. This can help banks and other financial institutions build stronger relationships with their customers and increase loyalty.

3.3.5. Data Science

Big Data is a key component of data science, as it provides the raw material that data scientists use to extract insights and build predictive models. Data science is a interdisciplinary field that combines methods from

statistics, computer science, and domain expertise to analyze and interpret large, complex data sets.

MapReduce is a programming model for data processing that was first introduced by Google in 2004 [16]. It is designed to process large-scale data sets in parallel across a cluster of computers, and it has become a popular choice for data-intensive tasks in various domains, including data science. MapReduce consists of two main phases: the "map"phase, in which data is transformed and processed, and the reduce"phase, in which the results of the map phase are combined and aggregated.

One of the key advantages of MapReduce is its ability to process data in parallel, which allows it to handle large-scale data sets efficiently. This is particularly useful in data science, where data sets can often be very large and complex, and require significant computational resources to process and analyze. By using MapReduce, data scientists can break down a large data set into smaller chunks, and process each chunk independently and in parallel, which can greatly reduce the time and computational resources required for analysis.

There are many applications of MapReduce in data science, including data preprocessing, machine learning, and data visualization. For example, MapReduce can be used to clean and prepare data for analysis, to train and evaluate machine learning models, and to generate visualizations of the data. MapReduce can also be used to perform complex data transformations and manipulations, such as joining multiple data sets together, or aggregating data by various dimensions.

Overall, the use of MapReduce in data science can provide many benefits, including improved efficiency and scalability, and the ability to handle large-scale data sets.

Data scientists use a variety of techniques and tools to extract insights and build predictive models from Big Data using Map Reduce. The data science process generally involves the following steps:

1. Define the problem: Identify the business problem or research question that needs to be addressed.

2. Collect and prepare data: Gather and organize the necessary data for the problem at hand. This may involve obtaining data from various sources, cleaning and preprocessing the data to remove errors or inconsistencies, and transforming the data into a format suitable for analysis. MapReduce can be useful for this step, as it allows data scientists to process and transform large-scale data sets in parallel, which can greatly improve efficiency and reduce the time and resources required for data preparation.

3. Explore and analyze data: Explore the data to gain insights and understand patterns and relationships. This may involve statistical analysis,

visualization, and machine learning techniques. MapReduce can also be useful for this step, as it allows data scientists to train and evaluate machine learning models on large-scale data sets in parallel.

4. Communicate results: Present the findings and insights derived from the data to relevant stakeholders in a clear and concise manner. This may involve creating reports, visualizations, or presentations.

There are several principal types of machine learning, including:

- Supervised learning: In supervised learning, a model is trained on labeled data, where the correct output is provided for each input. The goal is to make predictions on new, unobserved data based on the patterns learned from the training data. Examples of supervised learning tasks include regression (predicting a continuous output) and classification (predicting a categorical output).

- Unsupervised learning: In unsupervised learning, the model is not provided with labeled training examples, and must discover the inherent structure in the data through techniques such as clustering.

- Semi-supervised learning: In semi-supervised learning, the model is provided with a mix of labeled and unlabeled data, and must learn to make predictions based on the patterns in the labeled data and the inherent structure in the unlabeled data.

- Reinforcement learning: In reinforcement learning, an agent learns to interact with an environment in order to maximize a reward. The agent takes actions in the environment and receives feedback in the form of rewards or punishments, and learns to choose actions that maximize the reward over time.

Overall, Big Data plays a critical role in data science, providing the raw material that data scientists use to generate insights and build predictive models. MapReduce can be useful for all of these types of machine learning, as it allows data scientists to train and evaluate models on large-scale data sets in parallel, which can improve the efficiency and scalability of the machine learning process.

Big Data and expert systems are two different technologies that can be used in combination to improve decision making and problem solving in various industries. Big Data refers to the large volumes of structured and unstructured data that are generated by businesses, organizations, and individuals on a daily basis. Expert systems, on the other hand, are computer-based systems that use artificial intelligence algorithms to mimic the decision-making abilities of human experts in a specific domain.

When combined, Big Data and expert systems can provide organizations with valuable insights and recommendations based on the analysis of large data sets. For example, an expert system can use data mining techniques to analyze large amounts of data from various sources, such as customer transactions, market trends, and social media conversations. This information can then be used to identify patterns, trends, and relationships that would be difficult for a human expert to identify manually.

Overall, the relationship between Big Data and expert systems is that they both play a role in the field of data analytics and artificial intelligence, with Big Data providing the raw data that can be used by expert systems to make informed decisions and recommendations.

The relationship between functional programming and Big Data is that functional programming languages, such as Haskell, Scala, and Clojure, can be used to write efficient and scalable algorithms for processing and analyzing large data sets. These languages often provide built-in support for parallelism and distributed computing, which can be useful for working with Big Data in distributed systems.

Functional programming can also help improve the readability and maintainability of Big Data algorithms by promoting the use of pure functions and immutable data structures. This can make it easier for developers to understand and modify the code, reducing the risk of errors and improving the overall reliability of the system.

Overall, functional programming can be a useful tool for working with Big Data, providing a set of powerful abstractions and concepts that can help to simplify the development and deployment of Big Data algorithms.

3.4. Distances

The concept of measure has its origins in the need to calculate and manage areas, lengths, and volumes. The earliest known documents on the subject, the Moscow and Ahmes (or Rhind) papyri, date back to the $19th$ century BC. In the treatise Euclid's Elements ($300BC$), the first mathematical proofs of theorems related to the measurement of areas and volumes appear, along with axioms that would govern the principles of measurement until 1883. It was in that year that G. Cantor proposed the first definition of measure $m(K)$ as an arbitrary (bounded) set $K \subset \mathbb{R}^n$.

The definition of the Lebesgue measure by E. Borel and H. Lebesgue at the beginning of the 20th century marked a turning point in the development of measure theory. This contribution enabled the measurement of objects on the real line and paved the way for other important measures, such as the topological dimension proposed by H. Poincaré and generalized by F. Hausdorff with the dimension or measure that bears his name. In the 1920s, the Russian mathematician A.S. Besicovitch laid the foundations of

the Mathematical Theory of Measurement, a branch of mathematics that grew in popularity, particularly in the second half of the 20th century. This field of study has been essential for the development of various scientific fields, including capacity measurements by G. Choquet, fuzzy measurements introduced by M. Sugeno, the concept of granularity, and measures of necessity and possibility proposed by L.A. Zadeh, as well as specificity measures proposed by R.R. Yager, non-specificity measures discussed by M. Higashi and G.J. Klir, and the concept of minimum specificity from D. Dubois and H. Prade.

3.4.1. Measure, distance, metric spaces and topology

Definition 3.4.1. By measure, in a measurable space $(\Omega, \mathcal{A})$ - with $\mathcal{A}$ algebra o ring - we will understand a non-negative function $\mu : \mathcal{A} \to [0, \infty]$, that satisfies the following axioms:

1. $\mu(\emptyset) = 0$, where $\emptyset$ is the empty set.
2. It is countably additive, that is, for any countable collection of disjoint sets $A_1, A_2, \ldots$, we have Eq. 3.4. This axiom states that the measure of a countable union of disjoint sets is equal to the sum of the measures of the individual sets.

$$\mu\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} \mu(A_i) \tag{3.4}$$

3. For any subset X of $\mathcal{A}$, we have $\mu(X) \leq \mu(\mathcal{A})$, where $\mu(\mathcal{A})$ is called the total measure of $\mathcal{A}$.

   We will call probability to any measurement verifying $\mu(\Omega) = 1$.

Definition 3.4.2. We will call any triplet $(\Omega, \mathcal{A}, \mu)$ a measure spaces, where μ is measure over the $\sigma - algebra$ $\mathcal{A}$ of Ω.

A measure space $(\Omega, \mathcal{A}, \mu)$ is complete if for each $B \subset A$ with $A \in \mathcal{A}$ y $\mu(A) = 0$, then $B \in \mathcal{A}$

Example The Dirac delta measure. Consider $(\Omega, \mathcal{A}, \mu)$, $x \in \Omega$ and for each $E \in \mathcal{A}$, $\mu(E) = 0$, if $x \in E$ and $\mu(E) = 1$, if $x \in E$. It is the probability concentrated at x, or Dirac delta at x, usually, be denoted by δx.

Example The measure of counting. Consider $(\Omega, \mathcal{A}, \mu)$, with the measurable points x, $\{x\} \in \mathcal{A}$ and $\mu(\emptyset) = 0$, $\mu(A) = n$, if A is finite and has $n \in \mathcal{N}$ elements and $\mu(A) = \infty$ in any other case.

Example Hausdorff measures in a metric space. More general than Lebesgue's will allow us to define the area of a surface of space.

We are approaching the objective, which involves distances and metric spaces, which is definitely topology.

Definition 3.4.3. Let $(\mathcal{X}, \rho)$ be a metric space. For any $U \subset X$ and $diam(U) := sup\{\rho(x, y) : x, y \in U\}$, $diam(\emptyset) := 0$

Let S be any subset of $\mathcal{X}$ and $\delta \geq 0$ a real number. Define $H^d_\delta(S) = inf\sum_i^\infty (diamU_i)^d : \bigcup_{i=1}^\infty U_i \supseteq S, diamU_i < \delta$. where the infimum is over all countable covers of S by sets $U_i \subset X$ satisfying $diamU_i < \delta$.

Definition 3.4.4. A metric space [24] is a pair (M, d) where M is a set and d is a metric on M, i.e., a function: $d : MxM \to (R)$ satisfying the next axioms for all points $x, y, z \in M$:

1. $d(x, x) = 0$
2. (Positivity) If $x \neq y$, then $d(x, y) \geq 0$
3. (Symmetry) $d(x, y) = d(y, x)$
4. $d(x, z) \leq d(x, y) + d(y, z)$

Definition 3.4.5. The distance d of a metric space generates a topology in M defined by the open ball $B_r(x) = \{y \in \mathcal{M} : d(x, y) < r\}$, with $x \in M$ and $r \geq 0$ (a natural way define a set of points that are relatively close to x.

In other words, the distance between two points in a metric space is a numerical value that represents the degree of separation between the two points. This distance can be calculated using a specific formula that is defined for the particular metric space in question. For example, in the Euclidean plane, the distance between two points (x_1, y_1) and (x_2, y_2) is given by the Euclidean distance Eq. 3.5.

$$d((x_1, y_1), (x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{3.5}$$

In this case, the distance between the two points is the length of the straight line that connects them. However, different metric spaces may have different distance formulas, depending on the properties of the space and the way in which distances are defined within it.

In measure theory, the distance between two sets is a measure of the difference between them. It is typically defined as the infimum (i.e., the greatest lower bound) of the distances between all possible pairs of points in the two sets. This means that the distance between the sets is the smallest possible distance between any two points, one from each set.

Definition 3.4.6. If we have two sets A and B, their distance $d(A, B)$ is defined as:

$$d(A, B) = \text{ínf}\{d(a, b) \mid a \in A, b \in B\} \tag{3.6}$$

where $d(a, b)$ is the distance between the points a and b. The distance function $d(a, b)$ could be any function that satisfies the properties of a distance metric, such as the Euclidean distance or the Manhattan distance.

Hierarchical clustering [40] is a method of cluster analysis that seeks to build a hierarchy of clusters. This is done by either considering each observation as a single cluster at the outset and then successively merging (agglomerative) or splitting (divisive) clusters, or by considering all possible pairs of clusters and then merging the most similar pairs until only a single cluster remains (agglomerative). In hierarchical clustering, the distance between groups is a crucial factor in determining the structure of the resulting clusters.

There are many different ways to define the distance between two groups of observations, and the choice of distance measure can have a big impact on the resulting cluster hierarchy. Some common alternatives include:

- Maximum distance: This is defined as the maximum distance between any pair of points in the two groups. It is the largest possible distance between any pair of points in the two groups, and is useful for clustering data with a high degree of variability.
- Average distance: This is defined as the average distance between all pairs of points in the two groups. It is the mean distance between all pairs of points in the two groups, and is less sensitive to outliers than maximum or minimum distance measures.
- Median distance: This is defined as the median distance between all pairs of points in the two groups. It is the middle distance between all pairs of points in the two groups, and is less sensitive to outliers than average or maximum distance measures.

  Again, the choice of distance measure will depend on the specific characteristics of the data being clustered, and it may be necessary to try out several different measures to determine which one provides the best results.

3.4.2. Distance Examples

In the context of image processing, the distance between two sets of points can be used to measure the difference between two images. For example, if we have two images of the same scene, we can compute the distance between the sets of pixels in the two images to measure how different they are. This can be useful for applications such as image matching or image compression.

In this thesis, we explore the use of distance in measure theory for the problem of image matching. We investigate different distance metrics and their properties, and develop algorithms for computing the distance between

sets of SOMs representations. We also explore the use of machine learning techniques for image matching, with focus on SOM methods.

There are many different distances that can be used in measure theory and image processing.

Euclidean The Euclidean distance is the most commonly used distance. It is defined as the straight−line distance between two points in Euclidean space. In other words, it is the square root of the sum of the squares of the differences between the coordinates of the two points. In cartesian coordinates, if $a = (a_1, \ldots, a_n)$ and $b = (b_1, \ldots, b_n)$ are two points in a Euclidean n-space, then the euclidean distance (d) is, Eq. 3.7

$$d(a,b) = d(b,a) = \sqrt{(a_1 - b_1)^2 + \cdots + (a_n - b_n)^2} = \sqrt{\Sigma_{i=1}^{n}(a_i - b_i)^2} \tag{3.7}$$

For example, the Euclidean distance between the points $(2,3)$ and $(4,5)$ is: $d((2,3),(4,5)) = \sqrt{(4-2)^2 + (5-3)^2} = \sqrt{2^2 + 2^2} = \sqrt{8} = 2.83$

The Euclidean distance is useful for image matching because it captures the geometric relationships between points in an image. However, it can be computationally expensive to compute for large images since it requires calculating the square root of the sum of the squares of the differences between the coordinates of all pairs of points in the two images.

Manhattan The Manhattan distance is another common distance that is used in image processing. It is defined as the sum of the absolute differences between the coordinates of the two points. In other words, it is the sum of the distances along each coordinate axis. In Cartesian coordinates, if $a = (a_1, \ldots, a_n)$ and $b = (b_1, \ldots, b_n)$ are two points in a Euclidean n-space. Then the Manhattan distance (d) is, Eq. 3.8

$$d(a,b) = d(b,a) = \mid a_1 - b_1 \mid + \cdots + \mid a_n - b_n \mid = \Sigma_{i=1}^{n} \mid a_i - b_i \mid \tag{3.8}$$

For example, the Manhattan distance between the points $(2,3)$ and $(4,5)$ is:

$d((2,3),(4,5)) = \mid 4-2 \mid + \mid 5-3 \mid = 2+2 = 4$

The Manhattan distance is useful for image matching because it is computationally efficient to compute and can capture the geometric relationships between points in an image. However, it can be less accurate than the Euclidean distance for some applications since it does not consider the geometric distances between points.

Hamming The Hamming distance is a distance that is used in image processing when the images are represented as binary strings. It is defined as the number of positions at which the two strings differ. Let $a = (a_1, \ldots, a_n)$ and $b = (b_1, \ldots, b_n)$ be words in $\mathcal{C}$ an subset of $\mathcal{A}^n$ where $\mathcal{A}$ is an alphabet of symbols, Eq. 3.9.

$$d(a, b) = d(b, a) = \#\{i : a_i \neq b_i, i = 1, \ldots, n\} \tag{3.9}$$

For example, the Hamming distance between the strings "1001" and "1011" is:

$d($"1001","1011"$) =\mid 1 - 1 \mid + \mid 0 - 0 \mid + \mid 0 - 1 \mid + \mid 1 - 1 \mid = 0 + 0 + 1 + 0 = 1$

The Hamming distance is useful for image matching because it can capture the similarity between two binary strings, even if they are of different lengths. However, it can only be applied to binary images, and it may not be suitable for some applications that require more detailed information about the differences between the images.

Cosine The Cosine distance is a distance that is used in image processing when the images are represented as vectors in a high-dimensional space. It is defined as the angle between the two vectors, measured in radians Eq. 3.10. For example, the Cosine distance between the vectors $(2, 3, 5)$ and $(4, 5, 6)$ is:

$d((2, 3, 5), (4, 5, 6)) = \arccos \frac{58}{\sqrt{34} * \sqrt{77}} = \arccos \frac{58}{2618} = 0.058\ radians$

$$d(\mathbf{a}, \mathbf{b}) = 1 - \frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}|\,|\mathbf{b}|} \tag{3.10}$$

To use the arccosine function (or its inverse, the cosine function) to calculate the angle between two vectors, one will first need to calculate the dot product of the two vectors and the magnitudes (lengths) of the vectors. Then, one can use the formula for the dot product to express the dot product in terms of the magnitudes and the cosine of the angle between the vectors:

$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}|\,|\mathbf{b}| \cos\theta$$

Where $\mathbf{a}$ and $\mathbf{b}$ are the two vectors, $|\mathbf{a}|$ and $|\mathbf{b}|$ are their magnitudes, and θ is the angle between the vectors.

Once one has obtained this equation, one can solve for θ by dividing both sides by the product of the magnitudes and taking the arccosine of both sides:

$$\cos\theta = \frac{\mathbf{a}\cdot\mathbf{b}}{|\mathbf{a}|\,|\mathbf{b}|}$$

$$\theta = \arccos\frac{\mathbf{a}\cdot\mathbf{b}}{|\mathbf{a}|\,|\mathbf{b}|}$$

The Cosine distance is helpful for image matching because it captures the similarity between two vectors based on their angle. It is also computationally efficient to compute since it only involves multiplying the coordinates of the two vectors and taking the arccosine of the result. However, it can only be applied to images represented as vectors, and it may not be suitable for some applications that require more detailed information about the differences between the images.

Jaccard The Jaccard distance is a distance that is used in image processing when the images are represented as sets of points. It is defined as the complement of the Jaccard index, which is the ratio of the number of points in the intersection of the two sets to the number of points in the union of the two sets. Let $(\mathcal{A}, \mu)$ a measure spaces and $A, B \in \mathcal{A}$, Jaccard distance is defined as, Eq. 3.11.

$$d(A,B) = d(B,A) = \frac{\mu(A \cap B)}{\mu(A \cup B)} \tag{3.11}$$

For example, the Jaccard distance between the sets $2,3,5$ and $4,5,6$ is:

$d(2,3,5,4,5,6) = 1 - (\frac{3}{6}) = 1 - 0.5 = 0.5$

The Jaccard distance is useful for image matching because it captures the similarity between two sets of points based on the overlap between the sets. It is also easy to compute, since it only involves counting the number of points in the intersection and the union of the two sets. However, it can only be applied to images that are represented as sets of points, and it may not be suitable for some applications that require more detailed information about the differences between the images.

Minkowki The Minkowski distance is a distance that is a generalization of the Euclidean distance and the Manhattan distance. It is defined as the $p-th$ root of the sum of the $p-th$ powers of the differences between the coordinates of the two points, Eq. 3.12.

$$d(a,b) = d(b,a) = (\Sigma_{i=1}^{n} \mid (a_i - b_i \mid^p)^{\frac{1}{p}} \tag{3.12}$$

For example, the Minkowski distance with $p = 2$ (which is equivalent to the Euclidean distance) between the points $(2,3)$ and $(4,5)$ is:

$d((2,3),(4,5)) = (\mid 4-2 \mid^p + \mid 5-3 \mid^p)^(\frac{1}{p}) = (2^2+22)^(\frac{1}{2}) = 8^(\frac{1}{2}) = 2.83$

The Minkowski distance is useful for image matching because it is a flexible distance that can capture different types of geometric relationships between points in an image, depending on the value of p. It can be computationally efficient to compute for some values of p, such as p = 1 (which is equivalent to the Manhattan

In addition to the all distances define above, there are many other distances that can be used in measure theory and image processing. Some examples include the Mahalanobis distance, the Pearson correlation distance, and the Kullback-Leibler divergence.

Mahalanobis The Mahalanobis distance is a distance that is used to measure between a point a and a distribution D, is possible use as a distance between two points a and b under this distribution D. In image processing when the images are represented as vectors in a high-dimensional space. It is defined as the square root of the sum of the squares of the differences between the coordinates of the two vectors, divided by the corresponding variances. Eq. 3.13.

$$d(a,b) = \sqrt{(\vec{a}-\vec{b})^T S^{-1} (\vec{a}-\vec{b}} \tag{3.13}$$

where S is the covariance matrix.

For example, the Mahalanobis distance between the vectors $(2,3,5)$ and $(4,5,6)$ with variances $(1,1,1)$ is:

$d((2,3,5),(4,5,6)) = \sqrt{\frac{(4-2)^2+(5-3)^2+(6-5)^2}{(1^2+1^2+1^2)}} = \sqrt{\frac{8}{3}} = 2.83$

The Mahalanobis distance is helpful for image matching because it takes into account the correlations between the coordinates of the vectors, as well as their variances. It is also computationally efficient to compute, since it only involves multiplying the coordinates of the two vectors by the reciprocal of their variances and taking the square root of the sum of the squares of the results. However, it can only be applied to images that are represented as vectors, and it may not be suitable for some applications that require more detailed information about the differences between the images.

Pearson correlation The Pearson correlation distance is a distance that is used in image processing when the images are represented as vectors in a high-dimensional space. It is defined as the complement of the Pearson correlation coefficient, which is a measure of the linear relationship between the coordinates of the two vectors. Let $a = (a_1, \ldots, a_n)$ and $b = (b_1, \ldots, b_n)$ are two points the pearson correlation is Eq.3.14:

$$d(a,b) = r_{ab} = \frac{\Sigma_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\Sigma_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\Sigma_{i=1}^{n}(y_i - \bar{y})}} \tag{3.14}$$

where $\bar{y} = \frac{1}{n}\Sigma_{i=1}^{n} y_i$.

For example, the Pearson correlation distance between the vectors $(2,3,5)$ and $(4,5,6)$ is:

$d((2,3,5),(4,5,6)) = 1 - \frac{(2*4)+(3*5)+(5*6)}{\sqrt{(2^2+3^2+5^2)*(4^2+5^2+6^2)}} = 1 - \frac{58}{\sqrt{2618}} = 1 - 0.022 = 0.978$

The Pearson correlation distance is useful for image matching because it captures the linear relationship between the coordinates of the vectors. It is also computationally efficient to compute, since it only involves multiplying the coordinates of the two vectors and taking the complement of the result. However, it can only be applied to images that are represented as vectors, and it may not be suitable for some applications that require more detailed information about the differences between the images.

Kullback-Leibler The Kullback-Leibler divergence is a distance that is used in image processing when the images are represented as probability distributions. It is defined as the difference between the expected value of the logarithm of the ratio of the two distributions and the logarithm of the ratio of the expected values of the distributions.

Hausdorff The Hausdorff distance is a distance that is used in image processing when the images are represented as sets of points. It is defined as the maximum of the distance from each point in one set to the closest point in the other set. For example, the Hausdorff distance between the sets $2,3,5$ and $4,5,6$ is:

$$d(\{2,3,5\},\{4,5,6\}) = \text{máx}\{\text{mín}\{|\,2-4\,|,|\,2-5\,|,|\,2-6\,|\}\ \text{mín}\{|\,3-4\,|,|\,3-5\,|,|\,3-6\,|\},\text{mín}\{|\,5-4\,|,|\,5-5\,|,|\,5-6\,|\}\} = \text{máx}\{2,1,1\} = 2$$

The Hausdorff distance is useful for image matching because it captures the maximum difference between the two sets of points. It is also easy to compute since it only involves calculating the distances from each point in one set to the closest point in the other set and taking the maximum results. However, it can only be applied to images represented as sets of points. It may not be suitable for some applications that require more detailed information about the differences between the images.

Sorensen The Sorensen-Dice distance is a distance that is used with sets of points. It is defined as the complement of the Sorensen-Dice coefficient, which is the ratio of twice the number of points in the intersection of the two sets to the sum of the number of points in each set.

The Sorensen-Dice distance is useful for image matching because it captures the overlap between the two sets of points. It is also easy to compute, since it only involves counting the number of points in the intersection and the union of the two sets and taking the complement of the result. However, it can only be applied to images that are represented as sets of points, and it may not be suitable for some applications that require more detailed information about the differences between the images.

Wasserstein The Wasserstein distance is a distance that is used with probability distributions. It is defined as the minimum cost of transporting the mass of one distribution to the other, where the cost is the product of the mass and the distance between the points at which the mass is located.

In summary, many different distances can be used in measure theory and image processing, including the Euclidean distance, the Manhattan distance, the Hamming distance, the Cosine distance, the Jaccard distance, the Minkowski distance, the Mahalanobis distance, the Pearson correlation distance, the Kullback-Leibler divergence, the Hausdorff distance, the Sorensen-Dice distance, and the Wasserstein distance. Each distance has its own strengths and weaknesses, and may be more or less suitable for different applications. For example, some distances are more computationally efficient to compute, while others can capture more detailed information about the differences between the images.

Here is a summary of the advantages and disadvantages of some common distances that are used in measure theory and image processing:

The Euclidean distance is a widely used distance that captures the geometric relationships between points in an image. It is defined as the square root of the sum of the squares of the differences between the coordinates of the two points. However, it can be computationally expensive to compute for large images.

The Manhattan distance is a computationally efficient distance that captures the geometric relationships between points in an image. It is defined as the sum of the absolute differences between the coordinates of the two points. However, it can be less accurate than the Euclidean distance for some applications.

The Hamming distance is a distance that is used for binary images. It is defined as the number of positions at which the two strings differ. However,

it can only be applied to binary images, and it may not be suitable for some applications that require more detailed information about the differences between the images.

The Cosine distance is a distance that is used for vectors in a high-dimensional space. It is defined as the angle between the two vectors, measured in radians. However, it can only be applied to images that are represented as vectors, and it may not be suitable for some applications that require more detailed information about the differences between the images.

The Jaccard distance is a distance that is used for sets of points. It is defined as the complement of the Jaccard index, which is the ratio of the number of points in the intersection of the two sets to the number of points in the union of the two sets. However, it can only be applied to images that are represented as sets of points, and it may not be suitable for some applications that require more detailed information about the differences between the images.

The Minkowski distance is a flexible distance that is a generalization of the Euclidean distance and the Manhattan distance. It is defined as the p-th root of the sum of the p-th powers of the differences between the coordinates of the two points. However, it may not be suitable for some applications that require more detailed information about the differences between the images, depending on the value of p.

The Mahalanobis distance is a distance that is used for vectors in a high-dimensional space. It is defined as the square root of the sum of the squares of the differences between the coordinates of the two vectors, divided by the corresponding variances. However, it can only be applied to images that are represented as vectors, and it may not be suitable for some applications that require more detailed information about the differences between the images.

The Pearson correlation distance is a distance that is used for vectors in a high-dimensional space. It is defined as the complement of the Pearson correlation coefficient, which is a measure of the linear relationship between the coordinates of the two vectors. However, it can only be applied to images that are represented as vectors, and it may not be suitable for some applications that require more detailed information about the differences between the images.

The Kullback-Leibler divergence is a distance that is used for probability distributions. It is defined as the difference between the expected value of the logarithm of the ratio of the two distributions and the logarithm of the ratio of the expected values of the distributions. However, it may not be suitable for some applications that require more detailed information about the differences between the images, since it only considers the expected values of the distributions.

The Hausdorff distance is a distance that is used for sets of points. It is defined as the maximum of the distances from each point in one set to

the closest point in the other set. However, it can only be applied to images that are represented as sets of points, and it may not be suitable for some applications that require more detailed information about the differences between the images.

The Sorensen-Dice distance is a distance that is used for sets of points. It is defined as the complement of the Sorensen-Dice coefficient, which is the ratio of twice the number of points in the intersection of the two sets to the sum of the number of points in each set. However, it can only be applied to images that are represented as sets of points, and it may not be suitable for some applications that require more detailed information about the differences between the images.

The Wasserstein distance is a distance that is used for probability distributions. It is defined as the minimum cost of transporting the mass of one distribution to the other, where the cost is the product of the mass and the distance between the points at which the

3.4.2.1. Distance of images

An image can be represented as a matrix, where each element of the matrix corresponds to a pixel in the image. The dimensions of the matrix correspond to the width and height of the image, and the value of each element in the matrix corresponds to the color of the corresponding pixel in the image.

To measure the distance between two images, or between two matrices representing the images, we can use a distance metric such as the Euclidean distance or the cosine distance. The Euclidean distance between two images A and B is defined as:

$$d(A, B) = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} (A_{i,j} - B_{i,j})^2}$$

where m and n are the dimensions of the images, and $A_{i,j}$ and $B_{i,j}$ are the elements of the matrices representing the images at position (i, j).

The cosine distance between two images A and B is defined as:

$$d(A, B) = 1 - \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} A_{i,j} B_{i,j}}{\sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} A_{i,j}^2} \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} B_{i,j}^2}}$$

where m and n are the dimensions of the images, and $A_{i,j}$ and $B_{i,j}$ are the elements of the matrices representing the images at position (i, j).

These distance metrics can be used to measure the similarity between two images, with smaller distances indicating greater similarity. For example, if two images are identical, their distance will be zero, while if they are

completely different, their distance will be the maximum possible value (e.g., 1 for the cosine distance, or ∞ for the Euclidean distance).

3.4.3. Measure Spaces

A measure space is a mathematical concept that is closely related to topology. Formally, a measure space is a triple $(X, \mathcal{F}, \mu)$, where X is a set, $\mathcal{F}$ is a σ-algebra of subsets of X, and μ is a measure defined on $\mathcal{F}$.

The set X is called the underlying set of the measure space. In most cases, X is a topological space, and the elements of X are called points.

The σ-algebra $\mathcal{F}$ is a collection of subsets of X that satisfies certain axioms, such as being closed under complementation and countable unions. Intuitively, the σ-algebra $\mathcal{F}$ specifies which subsets of X are "measurable" in the sense that they can be assigned a measure.

The measure μ is a function that assigns a non-negative real number, or ∞, to each element of $\mathcal{F}$. The measure must also satisfy certain axioms, such as being countably additive and having $\mu(\emptyset) = 0$. The measure μ defines the size of a measurable set, and allows us to compute the size of various combinations of measurable sets.

The relationship between measure spaces and topology is that a topological space $(X, \mathcal{T})$ can be viewed as a measure space by taking $\mathcal{F} = \mathcal{T}$ and defining the measure μ to be the trivial measure that assigns a measure of zero to all sets in $\mathcal{T}$. This is not a very interesting measure space, but it illustrates the connection between measure spaces and topology.

In general, a measure space is more general than a topological space, because it allows us to define the size of subsets of X, whereas a topological space only defines the notion of convergence and continuity. However, a measure space can be thought of as a special kind of topological space, where the open sets are precisely the measurable sets in $\mathcal{F}$.

Chapter 4

Self-Organizing Map

Mi Ovejita Lucera

La Mandrágora

Summary: Self-organizing maps (SOMs) are a type of unsupervised neural network that can be used for dimensionality reduction and clustering. SOMs are trained to preserve the topological relationships between high-dimensional input data, and can be visualized as two-dimensional grids of neurons, each node represents a set of points in the input space. In this section, the motivation for using SOMs in the current study is discussed, as well as an overview of the relevant background information and previous work in the field. This include a brief explanation of how SOMs work and their potential applications in various domains.

Resumen: Las mapas auto-organizativos (SOMs) son un tipo de red neuronal no supervisada que se puede utilizar para la reducción dimensional y el agrupamiento. Los SOMs se entrenan para preservar las relaciones topológicas entre los datos de entrada de alta dimensionalidad, y se pueden visualizar como cuadrículas bidimensionales de neuronas, cada una representando un punto diferente en el espacio de entrada. En esta sección, discutimos la motivación para utilizar SOMs en este estudio, así como una visión general de la información relevante y del trabajo previo en el campo. Esto incluye una breve explicación de cómo funcionan los SOMs y sus posibles aplicaciones en diferentes dominios.

4.1. Introduction

We now present a short review of the original SOM algorithm.

The SOM is a neural network introduced by Teuvo Kohonen in 1986 that allows us to project a high-dimensional vector space onto a low-dimensional topology (typically two) integrated by a set of different nodes or neurons displayed as a grid. This nonlinear projection results in a "feature map" of the high-dimensional space, which may be used to detect and analyze characteristics that allow for the identification of patterns or groups. SOMs have been applied to a number of fields, e.g., document retrieval, financial data analysis, forensic analyses, and engineering applications. Machine learning includes two categories, supervised and unsupervised learning. In the former, there is a division of the variables between inputs and outputs, and the purpose of the analysis is to estimate the input–output relationship and make predictions. In the latter, all variables are on equal footing, and one searches for a better understanding of their multivariate structure, possibly also involving dimensional reduction. With (unsupervised) SOMs, we are capable of identifying features and structures in high-dimensional data. SOMs show a self-organizing behavior with the capacity to detect hidden characteristics within nonlabeled groups when enough map nodes (also called neurons) are used. It also has the perspective of dimensional reduction that seeks to optimize topological conservation [30, 2, 67], that is that the relative locations of the points in the original space are reflected in some way on the map. One of those ways is distance preservation. Thus, it can be used to identify similar objects once the map has been trained.

SOMs differ from other neural networks in that they use unsupervised learning rather than supervised learning. Additionally, SOMs use competitive learning, in which neurons compete to be activated at each iteration, rather than learning through error correction. Since SOMs are unsupervised learning techniques, they are used to discover common structures and patterns in data. They are also useful for dimensionality reduction and for visualizing data in two-dimensional maps. SOMs can be thought of as a type of neural network consisting of a single layer of neurons arranged in two dimensions. The number of neurons, K, in the map is fixed in advance and can be arranged in a grid or hexagonal pattern. These neurons form the output space of the SOM.

As mentioned above, the SOM produces a nonlinear mapping of the high-dimensional space onto a reduced dimension one. Several versions of the original algorithm [30, 37] have been proposed. Suppose $x(t)$ denotes the input vectors and the weight vector of the k-th neuron ($k = 1...K$), which represents the corresponding node in the original input space. The K neurons are arranged in a bidimensional network, which induces a set of corresponding weight vectors. A discrete time (training epoch) index ($t = 1...$) is introdu-

ced so that $x(t)$ is presented to the network at time t, and the $w_k(t)$, which represents the state of the net at that moment, are updated. Before training the map, the weight vectors should be initialized, which is generally performed by taking K vectors, typically by random sampling within the initial data.

Once the map has been trained, any new observation will be assigned to the map node with the lowest euclidean distance between the weight and observation (obviously, in the original space).

Self-Organizing Maps (SOMs) have a wide range of applications, including finance, economics, marketing, speech recognition, text mining, and language processing.

A priori, SOMs need not be related to a stochastic reality: it can be used within a deterministic framework. The first reference to SOMs was the seminal paper by Kohonen [30], based on the work by von der Malsburg [38] and Willsh and von Der Malsburg [76] on competitive learning. The theoretical background was in the monograph by Kohonen [31]. Two interesting and thorough reviews of SOMs can be found in Yin [77] and Van Hulle [28]. The stochastic approaches started with Lutrell [37] and Yin and Alison [78]; a more recent reference was Guo et al. [25]. Deep SOMs were approached from a practical of view by Liu et al. [58]. The relationship between SOMs and the EM algorithm was explored by Yuille [79], Durbin et al. [19], and Utsugi, A. [68]. Mixtures within the Bayesian framework were dealt with by Lau and Green, P. [35], Wang and Dunson, D. [73], and Ormoneit and Tresp, Dahl [15].

In the following section, we will describe how SOMs work.

4.2. Operation

If a data set consists of n observations, each represented by a p-dimensional vector (p variables), the SOM will assign each of these observations to a neuron in the output space. These neurons are in the input space R^p. This mapping allows the SOM to represent high-dimensional data in a lower-dimensional space while preserving the topology of the original space.

To map the high-dimensional data onto the output space of the SOM, each neuron in the output space is associated with a p-dimensional weight vector. Through an iterative process of "self-organization,"these weight vectors are modified to best represent the original data set. This process preserves the topology of the original space because the weight vectors of nearby neurons are also modified to this end during the self-organization process.

The self-organization process consists of four phases, which will be described briefly below.

In order to use SOMs, the first step is to determine the number of neurons, K, that will make up the network, as well as the two-dimensional arrange-

ment of these neurons. This number must be chosen carefully to ensure that the map is neither too large nor too small. There are several criteria that can be used to determine the value of K, and the appropriate value will depend on the specific data set being used. However, a commonly used rule proposed by Vesanto [72] is to set $K = 5\sqrt{n}$, where n is the number of observations in the data set.

Once the configuration of the network has been chosen, the weight vectors of each neuron must be initialized. A simple and effective approach is to generate K random vectors in R^p. Alternatively, the weight vectors can be initialized using the first principal components of the data set, which can improve the performance of the SOM at the expense of additional computational cost.

After initialization, the iterative process of self-organization begins. In each iteration, an observation from the input data set is selected and the neuron with the weight vector most similar to the input vector is identified as the "winning"neuron. The weight vectors of the winning neuron and its neighboring neurons are then modified to move closer to the input vector. This process is repeated until the network has converged, at which point the SOM is ready for use.

The specific equations and mathematical formulas used in the self organization process will be presented in the following section.

4.3. Algorithm and Mathematical questions

Despite the intuitive nature of SOMs, mathematically demonstrating their effectiveness is a challenging task. The network or map is defined as $m(t) = (m_1(t), m_2(t), ..., m_k(t))$ with $m_i(t)$ representing the model vector assigned to neuron i at time t. The goal is to show that there exists a state to which the network converges that preserves the topology of the input space.

The construction of SOMs can be divided into two stages: the organization of the network, during which the network becomes ordered, and the fitting stage, during which the model vectors converge to a stable state. As a result, theoretical studies of SOMs are also divided into two parts: investigating the existence of an ordered configuration and demonstrating convergence to a stable state once this configuration is reached.

Studies of SOMs in higher dimensions are limited, as it is difficult to define a single order that ensures an orderly and stable configuration. However, some partial results have been obtained, particularly in one-dimensional spaces, as detailed in [9].

4.3.1. Self-organization in one dimension

In order to demonstrate the effectiveness of SOMs, it is necessary to show that there exists a state to which the network converges that preserves the topology of the input space. This process can be divided into two stages: one in which the network is ordered, and another in which it fits to the data. In the case of a one-dimensional input space, it has been shown that under certain conditions, the algorithm will order the neurons in a finite time, reaching an ordered state. However, the convergence of the algorithm in multidimensional cases is much more difficult to prove and has not been extensively studied.

In this section, we consider a self-organizing map with a single variable input space and a linear arrangement of neurons. The aim is to demonstrate self-organization by finding an absorbing state in which the map converges with probability 1 in a finite amount of time. We define the sets F_k^+ and F_k^- as follows:

$F_k^+ = m \in \mathbb{R} \ / \ 0 < m_1 < m_2 < ... < m_k < 1$

$F_k^- = m \in \mathbb{R} \ / \ 0 < m_k < m_{k-1} < ... < m_1 < 1$

If the neighborhood function h decreases with distance between neurons, the following theorem applies:

Teorema 4.3.1. *(i) F_k^+ and F_k^- are absorbing sets. (ii) If α is constant and the function h decreases with distance, the time τ at which $F_k^+ \cup F^- k$ is reached is almost certainly finite and there exists $\lambda > 0$ such that* $\sup m \in [0,1]^k$ $\mathbb{E}_m(e^{\lambda\tau})$ *is finite where $\mathbb{E}_m$ is the expectation with $m(0) = m$.*

This theorem, proven in [7] using Markov chain techniques, guarantees that under the given conditions, the algorithm will almost certainly order the m_i values in finite time, reaching an ordered state.

4.3.2. Theorical study of convergence

The convergence of the algorithm must be studied after self-organization has been achieved by reaching an ordered state. This ordered state can be represented by either F^+ or F^-. From equation (2.8), it can be determined that the possible limit states must satisfy the equation $h(m) = 0$, where $h(m)$ is defined as the expected value of $H(x, m)$ with respect to the input distribution P. This leads to the ordinary differential equation $\frac{dm}{dt} = -h(m)$, which must be studied to find its equilibrium points.

Teorema 4.3.2. *Let $\alpha(t)$ be a decreasing function defined on $(0, 1)$ that satisfies $\lim_{t\to\infty} t\alpha(t) = +\infty$ and $\lim_{t\to\infty} t^2\alpha(t) < +\infty$. Let, also, h be the neighborhood function and a decreasing function from $[k_0, k-1]$. It is also assumed that the input distribution P has a positive associated density f at $(0, 1)$ and which satisfies that $\ln(f)$ is strictly concave (or only concave if it satisfies $\lim_{x\to 0^+} f(x) + \lim_{x\to 1^-} f(x)$ is positive). Then:*

The function:

1. $h(m) = E(H(x,m)) = \int_{-\infty}^{\infty} H(x,m), dP(x)$ *has at least one zero* m^* *in* F^+.
2. *Every stationary point* m^* *it is attractive. That is, if* $m(0) \in F^*$, $m(t)$ *c.s.* $\rightarrow m^*$.
3. *If the distribution is also uniform, the zero* m^* *is unique.*
4. *In the same sense, the previous theorem is equally applicable in the case of* F^-.

A theorem, given in [14], provides a proof of convergence under certain conditions. This theorem assumes that the adaptation parameter $\alpha(t)$ is decreasing, and that it satisfies certain conditions. It also assumes that the neighborhood function h is a decreasing function and that the input distribution P has a positive density f that satisfies certain conditions. The theorem states that $h(m)$ has at least one zero in F^+, that every stationary point m^* is attractive, and that if the distribution is uniform, the zero m^* is unique. The theorem also applies to the case of F^-.

However, the conditions required by this theorem are strict, and the problem becomes increasingly difficult as the dimension increases (in original space). As a result, there are few rigorous theoretical studies on the convergence of SOMs, despite their increasing use in applications.

4.4. Review of differents SOM alternatives

There are several variations and alternatives to SOMs that have been developed over the years, some of which are outlined below:

Growing Self-Organizing Maps (GSOMs) [66] GSOMs are a variant of SOMs that can handle data sets with varying densities and dimensions. They are able to adapt to the structure of the data by growing and pruning the map as needed. This makes GSOMs well-suited for handling data sets with non-uniform distributions and complex structures. However, the map growth and pruning process can be computationally expensive, and may require a large number of training iterations.

- Advantages: Can handle data sets with non-uniform and complex distributions.
- Disadvantages: Computationally expensive due to map growth and pruning process. May require a large number of training iterations.

- Parallelization alternatives: Map growth and pruning process can be parallelized using techniques such as divide-and-conquer or parallel coordinate descent.

Adaptive Self-Organizing Maps (ASOMs) [56] : ASOMs are a variant of SOMs that can adapt to changes in the data distribution over time. They are able to adjust the map structure and the weights of the neurons to reflect these changes. This makes ASOMs well-suited for handling data sets that are subject to concept drift, where the data distribution changes over time. However, ASOMs may require more training data and more training iterations than traditional SOMs.

- Advantages: Can adapt to changes in the data distribution over time.
- Disadvantages: May require more training data and more training iterations than traditional SOMs.
- Parallelization alternatives: Training process can be parallelized using techniques such as divide-and-conquer or parallel coordinate descent.

Fast Self-Organizing Maps (FSOMs) [9] : FSOMs are a variant of SOMs that use a fast training algorithm to reduce the training time of the map. They are able to achieve faster training times by using a divide-and-conquer approach to training the map. This makes FSOMs well-suited for handling large data sets that require fast training times. However, FSOMs may not be as accurate as traditional SOMs, and may require more training data to achieve good results.

- Advantages: Fast training times.
- Disadvantages: May not be as accurate as traditional SOMs.
- Parallelization alternatives:Divide-and-conquer training approach is inherently parallelizable.

Incremental Self-Organizing Maps (ISOMs) [10] : ISOMs are a variant of SOMs that can handle streaming data, and can update the map incrementally as new data points are received. This makes ISOMs well-suited for handling data sets that are constantly evolving and changing. However, ISOMs may require more training data and more training iterations than traditional SOMs, and may be prone to overfitting if the data distribution changes significantly over time.

- Advantages: Can handle streaming data and update the map incrementally.

- Disadvantages: May require more training data and more training iterations than traditional SOMs. May be prone to overfitting if the data distribution changes significantly over time.
- Parallelization alternatives: Training process can be parallelized using techniques such as divide-and-conquer or parallel coordinate descent.

Hierarchical Self-Organizing Maps (HSOMs) [64] : HSOMs are a variant of SOMs that are organized into a hierarchy, with each level representing a different level of abstraction in the data. This allows HSOMs to capture more complex patterns and relationships in the data. However, HSOMs may require more training data and more training iterations than traditional SOMs, and may be more computationally expensive due to the hierarchical structure of the map.

- Advantages: Can capture more complex patterns and relationships in the data.
- Disadvantages:May require more training data and more training iterations than traditional SOMs. May be more computationally expensive due to the hierarchical structure of the map.
- Parallelization alternatives: Training process can be parallelized using techniques such as divide-and-conquer or parallel coordinate descent.

Bayesian Self-Organizing Maps (BSOMs) [25] : BSOMs are a variant of SOMs that incorporate Bayesian techniques to model uncertainty in the data. They are able to estimate the posterior distribution over the map parameters given the data, which allows them to make more robust predictions and handle uncertainty in the data. However, BSOMs may require more training data and more training iterations than traditional SOMs, and may be more computationally expensive due to the Bayesian inference process.

- Advantages:Can model uncertainty in the data and make more robust predictions.
- Disadvantages:May require more training data and more training iterations than traditional SOMs. May be more computationally expensive due to the Bayesian inference process.
- Parallelization alternatives: Bayesian inference process can be parallelized using techniques such as Markov chain Monte Carlo or variational inference.

Uncertainty-based Self-Organizing Maps (USOMs) : USOMs are a variant of SOMs that incorporate uncertainty estimates into the map

structure and training process. They are able to model and propagate uncertainty through the map, which allows them to handle uncertain and noisy data and make more robust predictions. However, USOMs may require more training data and more training iterations than traditional SOMs, and may be more computationally expensive due to the uncertainty estimates.

- Advantages: Can handle uncertain and noisy data and make more robust predictions.
- Disadvantages: May require more training data and more training iterations than traditional SOMs. May be more computationally expensive due to the uncertainty estimates.
- Parallelization alternatives:Training process can be parallelized using techniques such as divide-and-conquer or parallel coordinate descent.

4.5. Sequential (Original) SOM

One interesting type of self-organizing map (SOM) is the sequential SOM, which is more interesting for us because it can adapt to new observations over time and improve its representation of the data. This can be useful in applications where the data is dynamic and changing over time, such as in real-time monitoring or online learning systems. Additionally, the online nature of sequential SOMs can also make them more efficient and scalable than batch SOMs, which require all observations to be present at once. Overall, the sequential SOM algorithm is a powerful tool for exploring and modeling complex, high-dimensional datasets.

In the original, so-called sequential SOM, observations are introduced one at a time, as opposed to the batch SOM. The operation of SOMs can be broken down into several key stages: initialization, competition, cooperation, and adaptation. Each of these stages is described in more detail below.

For the training phase, a metric is required for the distance between vectors in the input space, and the euclidean distance is typically applied when the variables are continuous and on equal scale [30]. For each input vector introduced, the closest (euclidean distance) weight vector is obtained. The so-called winning vector is denoted by the c subindex. Finally, all weight vectors are updated according to Equation (1).

$$\omega_k(t+1) = \omega_k(t) + \alpha(t)h_{ck}(t)|x(t) - \omega_k(t)| \tag{1}$$

The weight updating process is mainly driven by the learning rate, which should decrease monotonically along time. A standard option is an exponen-

tial decrease:

$$\alpha(t) = \alpha_1 e^{-\frac{t}{\alpha_j}}, \tag{2}$$

where α_1 and α_j are constants.

The neighborhood function $h_{ck}(t)$ provides the conservation of topology through weight updating, usually called the neighborhood Equation (3) function. It thus defines the region—around the winning node—affected by the updating process. It controls the extent to which an input vector adjusts the weight of the neuron k with respect to the winner c by means of the corresponding (in the map) intra-net distance between the two vectors. Moreover, the $x(t) - \omega_k(t)$ term produces a stronger update—within nodes equally distant in the 2D map from the winning one—for that whose weight in the previous iteration is more distant from the new observation. Several functions have been proposed in this direction, and here, we applied a Gaussian neighborhood. This function is symmetric about the winner. Furthermore, it monotonically decreases with time due to $\sigma(t)$ and with distance to the winner.

$$h_{ck}(t) = exp(-|\omega_c - x_k|^2/\sigma^2(t)) \tag{3}$$

$$\sigma(t) = \sigma_1 e^{-\frac{t}{\sigma_j}} \tag{4}$$

Algorithm 1 SOM algorithm.

1-layer SOM

Require: $K \geq 0 \vee S \subset X$ sample $\vee$ epochs number
Ensure: $\omega_k \; \forall K$
  Initialize $\omega_k \; \forall k \in (1...K)$ randomly
  for all epoch **do**
    {random loop over all input vector $x \in X$}
    for each x **do**
      {update weight vector ω_k according to eq. 1}
      Note: Including winner and neighborhood function
    end for
    Note: Not update α and σ because are function of the epoch
  end for

4.6. Quality

Once the self-organizing map (SOM) is built, as in the rest of the methods, it is necessary to evaluate how well the obtained result fits the data, allowing

us to extrapolate the conclusions drawn from the map to the original data. There are different measures that allow us to analyze the quality of the model, but two of the most commonly used and studied are the quantization error and the topographical error.

As in other clustering methods, it is necessary to check if the observations in the input space are well-matched with their projections into the output space (that is, to the reference vectors associated with each neuron) through the quantization error. On the other hand, since the SOM seeks to preserve the topology of the original space, we can measure the quality of the model in this aspect based on the topographical error, which measures how well the features are maintained on the map in terms of their topologies in the input set.

These two properties are related to each other, as adjusting the map to improve the quantization error may worsen the conservation of topological properties and vice versa. Therefore, it is important to maintain a trade-off between the two. Next, we will describe both measurements and how to obtain them.

This paper presents a summary of the various known alternatives for measures, including both their pros and cons. The most commonly used measures are discussed in greater detail.

There are several performance measures [72],[31],[65] that can be used to evaluate the quality of a SOM. These measures can be grouped into two main categories: topology-based measures and density-based measures.

4.6.1. Topology-based measures

They evaluate the topological structure of the SOM and how well it preserves the structure of the original data. Some examples of topology-based measures include:

Topographic error This measure of how well the structure of the input space is modeled by the SOM. Specifically, it evaluates the local discontinuities in the mapping, and is calculated by finding the best-matching and second-best-matching neuron in the map for each input and evaluating their positions. If the two nodes are neighbors, then we say topology has been preserved for this input, otherwise, this is counted as an error. The total number of errors divided by the total number of data points gives the topographic error of the map.

- Pros: Simple to calculate. Provides a clear measure of how well the SOM preserves the topological relationships between the input patterns.
- Cons: Does not take into account the density of the data in the map. Can be affected by the size and shape of the map.

The topographic error (TE) is calculated as follows:

$$TE = \frac{1}{n} \sum_{i=1}^{n} t(x_i) \tag{4.1}$$

where n is the number of data points, $t(x)$ is a binary variable that equals 0 if the best-matching and second-best-matching neurons are neighbors for the input data point x, and 1 otherwise. The function $BMU(x)$ returns the best-matching unit for data point x, and $BMU_0(x)$ returns the second-best-matching unit.

Note that this measure only evaluates how well individual data points are mapped to the nodes and does not account for larger distortions in the manifold.

U matrix This measure visualizes the distance between the neurons in the SOM. A low U-matrix value indicates that the neurons are more similar to their neighbors, while a high value indicates that the neurons are more dissimilar.

- Provides a visual representation of the distance between the neurons in the SOM. Can help identify clusters and outliers in the data.
- Can be affected by the size and shape of the map. Does not take into account the density of the data in the map.

The U matrix can be calculated as follows:

$$U(i,j) = \frac{1}{2} \left[d(w_i, w_j) + \frac{1}{2} \left(d(w_i, w_{i,j+1}) + d(w_j, w_{i,j+1}) \right) \right] \tag{4.2}$$

Where w_i and w_j are the weights of the neurons at positions (i, j) and $(i, j+1)$, and $d(x, y)$ is a distance function between two points x and y.

Neighborhood preserving error : This measure evaluates how well the SOM preserves the local structure of the original data. A low neighborhood preserving error indicates that the SOM is able to capture the local structure of the original data.

- Evaluates how well the SOM preserve the local structure of the original data. Can help identify clusters and outliers in the data.
- Can be affected by the size and shape of the map. Does not take into account the density of the data in the map.

The neighborhood preserving error is defined as the average distance between the data points and their corresponding neurons in the SOM, normalized by the size of the map. It can be calculated as follows:

$$NPE = \frac{1}{N} \sum_{i=1}^{N} \frac{d(x_i, BMU_i)}{\max_{j=1}^{n} d(BMU_j, BMU_{j+1})} \tag{4.3}$$

Where N is the number of data points, x_i is the i-th data point, BMU_i is the best matching unit (neuron) for the i-th data point, and $d(x, y)$ is a distance function between two points x and y.

4.6.2. Density-based measures

Density-based measures evaluate the density of the data in the map and how well the map represents the distribution of the original data. Some examples of density-based measures include:

Quantization error : This measure evaluates the average distance between the data points and their nearest neurons in the SOM. A low quantization error indicates that the SOM is able to accurately represent the distribution of the original data.

- Provides a measure of how well the SOM represents the distribution of the original data. Simple to calculate.
- Does not take into account the topology of the map. Can be affected by the size and shape of the map.

The quantization error is defined as the average distance between the data points and their nearest neurons in the SOM. It can be calculated as follows:

$$QE = \frac{1}{N} \sum_{i=1}^{N} d(x_i, BMU_i) \tag{4.4}$$

Where N is the number of data points, x_i is the i-th data point, and BMU_i is the best matching unit (neuron) for the i-th data point.

Entropy: This measure evaluates the uniformity of the data distribution in the SOM. A low entropy value indicates a more uniform distribution, while a high value indicates a more skewed distribution.

- Provides a measure of the uniformity of the data distribution in the SOM. Can help identify skewed or uneven distributions in the data.
- Does not take into account the topology or density of the map. Can be affected by the size and shape of the map.

$$H = -\sum_{i=1}^{n} p_i \log_2 p_i \tag{4.5}$$

Where n is the number of neurons in the SOM and p_i is the probability of a data point being assigned to the i-th neuron.

Davies-Bouldin index This measure evaluates the compactness and separation of the clusters in the SOM. A low Davies-Bouldin index indicates good separation and compactness of the clusters.

- Provides a measure of the compactness and separation of the clusters in the SOM. Can help identify well-separated and compact clusters in the data.
- Does not take into account the topology or density of the map. Can be affected by the size and shape of the map.

The Davies-Bouldin index can be calculated as follows:

$$DB = \frac{1}{n} \sum_{i=1}^{n} \max_{j \neq i} \left(\frac{s_i + s_j}{d(c_i, c_j)} \right) \tag{4.6}$$

Where n is the number of clusters in the SOM, s_i is the average distance of the data points in the i-th cluster to the cluster centroid, c_i is the centroid of the i-th cluster, and $d(x, y)$ is a distance function between two points x and y.

4.7. Expert system

Self-organizing maps (SOMs) are particularly useful for visualization and analysis of high-dimensional data, such as in cases where the data has many features or variables.

One of the main benefits of using SOMs as part of an expert system is that they can help to identify patterns and relationships in the data that may not be immediately apparent. SOMs can also be used to cluster the data into groups or categories, which can be helpful for making decisions or predictions based on the data.

Additionally, SOMs are known for their ability to handle missing or incomplete data, which is often an issue in real-world data sets. SOMs can be a valuable tool for helping to improve the accuracy and effectiveness of an expert system, particularly in cases where the data is complex and high-dimensional.

4.7.1. Inference engine

SOMs can be used as an inference engine [62], [43], [47], [42] in an expert system because they are able to process data and make decisions or predic-

tions based on the patterns and relationships that they identify in the data. Because SOMs are able to analyze and interpret data, they can be used to generate conclusions or recommendations based on the information that is fed into them.

SOMs can also be used as a knowledge base in an expert system because they are able to store and organize information in a structured way. For example, a SOM could be used to store and categorize a large amount of data, such as a collection of medical records or customer data. This information could then be used by the expert system to make decisions or provide recommendations based on the patterns and relationships that are identified in the data.

Overall, SOMs can be used as both an inference engine and a knowledge base in an expert system because they are able to process and analyze data, as well as store and organize information in a structured way. This makes them valuable tools for helping to improve the accuracy and effectiveness of an expert system.

4.7.2. Combining with reinforcement learning

Self-organizing maps (SOMs) can be combined with reinforcement learning [63],[44], [70],[6] techniques in order to improve the performance of an expert system. Reinforcement learning is a type of machine learning that involves training a system to make decisions or take actions in order to maximize a reward or minimize a penalty.

One way to combine SOMs and reinforcement learning is to use SOMs to identify patterns and relationships in the data, and then use reinforcement learning to guide the decision-making process based on those patterns. For example, a SOM could be used to analyze data about customer behavior, and a reinforcement learning algorithm could be used to determine the best actions to take in order to maximize customer satisfaction or sales.

Another way to combine SOMs and reinforcement learning is to use SOMs to cluster the data into groups or categories, and then use reinforcement learning to determine the best actions to take based on the characteristics of each group. For example, a SOM could be used to group customers based on their demographics and purchasing history, and a reinforcement learning algorithm could be used to determine the best marketing strategies to use for each group.

Overall, combining SOMs with reinforcement learning can be a powerful way to improve the performance of an expert system, particularly in cases where the data is complex and high-dimensional. By using SOMs to identify patterns and relationships in the data and reinforcement learning to guide the decision-making process, it is possible to achieve better results and more accurate predictions.

4.8. Relation between SOM, measure spaces and metric spaces

A self-organizing map (SOM) is a type of artificial neural network that is used for unsupervised learning. It is often used to visualize high-dimensional data, such as a 2D grid or a 1D line, in a low-dimensional space.

The relation between SOMs and measure spaces is that SOMs can be viewed as a particular kind of measure space, where the underlying set X is the set of all possible SOMs, the σ-algebra $\mathcal{F}$ is the collection of all measurable subsets of X. The measure μ is a function that assigns a measure to each measurable subset of X.

In practice, the measure μ is often defined in terms of a distance metric, such as the Euclidean distance or the cosine distance. For example, if we are using a SOM to visualize high-dimensional data in a 2D grid, the measure μ might be defined as the average distance between the data points and the corresponding SOM nodes. This measure allows us to compare different SOMs and select the one that best represents the data.

In summary, the relation between SOMs and measure spaces is that SOMs can be viewed as a particular kind of measure space, where the measure is defined in terms of a distance metric and is used to compare different SOMs.

4.9. Cathegory theory and functional programming

Self-organizing maps (SOMs) can be related to category theory, let's consider a simple example of a SOM that is used to visualize a dataset of 2-dimensional points. In this case, the SOM would be organized into a grid of nodes, where each node is associated with a pair of weights (w1, w2) representing the coordinates of a point in the 2-dimensional space.

In category theory, a category is a collection of objects (in our case, the data points in the SOM) that are related to each other by some kind of morphism (in our case, the weights of the SOM nodes). In this context, the SOM can be thought of as a category, where the nodes are the objects and the weights are the morphisms that relate the nodes to each other.

Next, let's consider the concept of a functor. In category theory, a functor is a mapping between two categories that preserves the structure of the categories. In the context of our SOM, we can think of a functor as a function that maps the nodes of the SOM to some other collection of objects (for example, the points in the original space), while preserving the relationship between the nodes (for example, by maintaining the same distances between the nodes in the SOM and their corresponding points in the original space).

Finally, as a monoid is a special type of category where every morphism

has an identity element (i.e. a neutral element that doesn't change the object it is applied to) and every morphism can be composed with any other morphism to produce a new morphism (i.e. the morphisms form a closed set under composition). In the context of our SOM, we can think of a monoid as a special type of category where each node has an identity weight (i.e. a weight that doesn't change the location of the node in the SOM) and the weights of the nodes can be composed with each other to produce new weights (i.e. the weights form a closed set under composition).

This concept of monoid is essential to understand the inferential capacity provided by SOM. Nodes that do not have data sample points assigned are constructed using this monoid as a combination of the neighbourhood weights, in such a way that if there are points in that area of space that was not represented in the sample, we could consider them within our metric space. But what's more, if we had an associated probability distribution, our SOM map was a measurement space, using a similar monoid we could infer the probability of this area of the map that is empty a priori.

Finally, let's consider the concept of a monad. In category theory, a monad is a special type of functor that has an additional property called the "monadic bind"(also known as the "Kleisli composition"). This property allows the functor to combine the effects of multiple morphisms predictably, making it easier to reason about and manipulate the data in the category. In the context of our SOM, we can think of a monad as a special type of functor that allows us to combine the effects of multiple weights on the nodes in a predictable way, making it easier to understand and manipulate the relationships between the nodes in the SOM.

For example, consider a self-organizing map with a two-dimensional grid of neurons, where each neuron is associated with a weight vector that encodes the position of the neuron in the grid, and a label that encodes the class or category of the neuron. We can define a functor from the category of two-dimensional vectors to the category of labeled weight vectors, where the functor maps each input vector to the labeled weight vector of the nearest neuron in the grid.

This functor can be composed with other functors that represent different operations on the input vectors, such as normalization, scaling, or rotation. The resulting composite functor represents the overall self-organizing map model, and its properties and behavior can be analyzed using the tools of category theory.

In particular, category theory can be used to study the consistency and accuracy of the labeling produced by the self-organizing map. For example, we can define a natural transformation between the functors that represent the self-organizing map model with and without labels, and use this natural transformation to measure the quality of the labeling produced by the self-organizing map. This can help to improve the performance and reliability of

the self-organizing map model in labeling tasks.

To sum up, self-organizing maps can be used to visualize and understand the concepts of category, functor, monoid, and monad in a mathematical context. By organizing the data points into a grid of nodes and assigning weights to the nodes that represent the relationships between the data points, the SOM can be thought of as a category, where the nodes are the objects and the weights are the morphisms. The SOM can then be used to define functors, monoids, and monads, which allow us to manipulate and reason about the data in the SOM in a more structured and predictable way.

4.9.1. Real examples

These examples illustrate how concepts from category theory, such as functors and monoids, can be used to analyze and organize data using self-organizing maps (SOMs).

In the first example, we use SOMs as functors in the category of sets to partition input data into clusters based on their proximity on the SOM. This allows us to apply statistical or machine learning algorithms to each cluster to extract meaningful patterns or features.

In the second example, we use SOMs as monoids in the category of monoids to combine images in each cluster and create a new, composite image that represents the overall structure of the data. This allows us to analyze the data in a computationally efficient way while preserving the topological structure of the data.

- Example 1: Using SOM as a Functor.

  Let **Set** be the category of sets. Let X be the input space, and Y be the output space of the SOM. Let $d(x, y)$ be the distance between two points $x, y \in X$, and let $f : X \to Y$ be the function that maps each input vector x to its corresponding point on the SOM.

  We can view X and Y as objects in **Set**, and f as a morphism in **Set**. This gives us a functor $F : \mathbf{Set} \to \mathbf{Set}$ that maps each set S in **Set** to the set $F(S)$ of all points on the SOM that correspond to vectors in S, and each function $g : S \to T$ in **Set** to the function $F(g) : F(S) \to F(T)$ that maps each point on the SOM that corresponds to a vector in S to the point on the SOM that corresponds to the image of that vector under g.

  We can define a partition of the input space X into clusters $C_1, C_2, ..., C_n$, based on their proximity on the SOM. Specifically, we can define C_i as the set of all input vectors x such that $f(x)$ is the i-th closest point to x on the SOM, where the distance between two points on the SOM is defined as the Euclidean distance between their weight vectors.

This partition induces a partition of the input data set D into clusters $D_1, D_2, ..., D_n$, where D_i is the subset of D that corresponds to the points in C_i.

We can then apply statistical or machine learning algorithms to each cluster D_i to extract meaningful patterns or features. For example, we can compute the mean or variance of the images in each cluster, or use a supervised learning algorithm to classify the images in each cluster based on their labels.

- Example 2: Using SOM as a Monoid for Text Clustering.

  Let **Mon** be the category of monoids. Consider a set of text documents $D = d_1, d_2, ..., d_n$, where each d_i is a sequence of words. We can represent each document as a bag-of-words vector v_i, where the j-th entry of v_i represents the frequency of the j-th word in document d_i.

  Let X be the input space of SOM, where each input vector x is a bag-of-words representation of a document. We can train the SOM on X to obtain a set of weight vectors, each of which represents a cluster of similar documents.

  We can view the set M of all possible bag-of-words vectors that can be represented as the combination of the weight vectors on the SOM as an object in **Mon**. Specifically, M is the free monoid generated by the set of weight vectors on the SOM, where the binary operation $*$ is defined as the average of the bag-of-words vectors in each cluster. That is, if w_i is the weight vector of the i-th cluster, then w_i represents the set of bag-of-words vectors V_i in that cluster, and $w_i * w_j$ is defined as:

$$w_i * w_j = \frac{1}{|V_i| + |V_j|} \left(\sum_{x \in V_i} x + \sum_{x \in V_j} x \right)$$

  This operation is a binary operation on M, and it is commutative and associative. The identity element of $*$ is the empty bag-of-words vector.

  By viewing the SOM as a monoid in **Mon**, we can use the algebraic structure of monoids to perform further analysis on the text data. For example, we can compute the average bag-of-words vector of the entire dataset or create a composite bag-of-words vector that represents the overall structure of the data. Using a monoid to represent the SOM allows us to reduce the computational complexity of analyzing the text data while preserving its topological structure.

4.10. Map Reduce

Self-organizing maps (SOMs) can be implemented using the MapReduce programming model. In this approach, the map function is responsible for training a local SOM on each chunk of the input data, and the reduce function is responsible for combining the trained SOMs into a single global SOM.

The input data is split into chunks and distributed to the map functions, which train a local SOM on each chunk in parallel. The trained SOMs are then combined and aggregated by the reduce function, resulting in a single global SOM that has been trained on the entire input data set. This allows MapReduce to handle large-scale data sets efficiently, by distributing the computation across multiple machines. Here is a pseudocode outline of a MapReduce SOM implementation in Scala, Fig. 4.1.

```
def mapReduceSOM(D: RDD[DataPoint], f: DataPoint => SOM, g: (SOM, SOM) => SOM): SOM = {
val chunks = D.map(f)
val result = chunks.reduce(g)
return result
}

def map(c: DataPoint): SOM = {
// train a local SOM on c
return trainedSOM
}

def reduce(result1: SOM, result2: SOM): SOM = {
// combine and aggregate result1 and result2 into a single global SOM
return globalSOM
}
```

Figure 4.1: SOM MapReduce

Chapter 5

One-Layer vs. Two-Layer SOM in the Context of Outlier Identification: A Simulation Study

Summary: In this section, we applied a stochastic simulation methodology to quantify the power of the detection of outlying mixture components of a stochastic model, when applying a reduced-dimension clustering technique such as Self-Organizing Maps (SOMs). The essential feature of SOMs, besides dimensional reduction into a discrete map, is the conservation of topology. In SOMs, two forms of learning are applied: competitive, by sequential allocation of sample observations to a winning node in the map, and cooperative, by the update of the weights of the winning node and its neighbors. By means of cooperative learning, the conservation of topology from the original data space to the reduced (typically 2D) map is achieved. Here, we compared the performance of one- and two-layer SOMs in the outlier representation task. The same stratified sampling was applied for both the one-layer and two-layer SOMs; although, stratification would only be relevant for the two-layer setting to estimate the outlying mixture component detection power. Two distance measures between points in the map were defined to quantify the conservation of topology. The results of the experiment showed that the two-layer setting was more efficient in outlier detection while maintaining the basic properties of the SOM, which included adequately representing distances from the

outlier component to the remaining ones.

RESUMEN: En este sección, aplicamos una metodología de simulación estocástica para cuantificar el poder de detección de componentes de mezclas periféricas de un modelo estocástico, al aplicar una técnica de agrupamiento de dimensiones reducidas como Self-Organizing Maps (SOM). La característica esencial de los SOM, además de la reducción dimensional a un mapa discreto, es la conservación de la topología. En los SOM se aplican dos formas de aprendizaje: competitivo, mediante la asignación secuencial de observaciones muestrales a un nodo ganador en el mapa, y cooperativo, mediante la actualización de los pesos del nodo ganador y sus vecinos. Mediante el aprendizaje cooperativo se logra la conservación de la topología desde el espacio de datos original hasta el mapa reducido (típicamente 2D). Aquí, comparamos el rendimiento de los SOM de una y dos capas en la tarea de representación de valores atípicos. Se aplicó el mismo muestreo estratificado tanto para los SOM de una capa como para los de dos capas; aunque, la estratificación solo sería relevante para la configuración de dos capas, para estimar el poder de detección del componente de la mezcla periférica. Se definieron dos medidas de distancia entre puntos en el mapa para cuantificar la conservación de la topología. Los resultados del experimento mostraron que la configuración de dos capas fue más eficiente en la detección de valores atípicos mientras mantenía las propiedades básicas del SOM, que incluían representar adecuadamente las distancias desde el componente atípico hasta los restantes.

5.1. Two-layer SOM

The SOM algorithm developed belongs to or is included within the unsupervised algorithms. Its objective is to project an original space into a two-dimensional space. This projection is not chaotic, but is governed by a main objective, to maintain the topology, to maintain the positional structure (except for rotations) of the points in the original space on the map [30, 2, 67]. This can be landed on keeping distances from the original space on the map.

However, there are a significant number of studies that have sought different strategies given the perspective that the model could be seen as too biased by the distribution and structure of the data [1, 52].

Our proposal was to apply an assembled SOM linking a single SOM, with different maps generated for random or non-random strata of the data. This allowed us to provide a parallelizable solution while optimizing the process

of the detection and representation of low-frequency areas that we sought to be overrepresented in any of the blocks. We call them two-layer SOM Algorithms 1 and 2.

The process therefore consisted of partitioning the initial dataset into n strata, applying the original SOM algorithm, the one-layer SOM, on each of them, and this was the parallelizable part. We generated a new dataset with all the weights generated in these maps and represented them in a new two-dimensional map, which implies computing a new SOM.

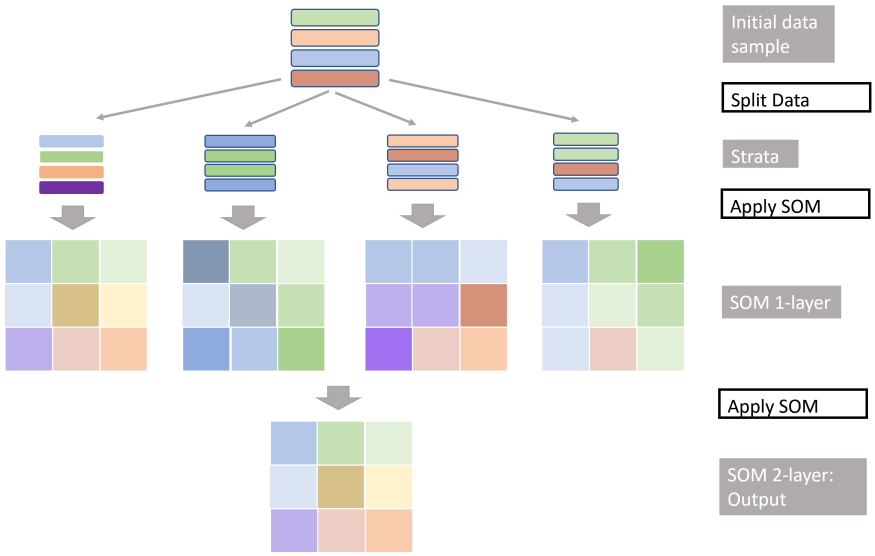


Figure 5.1: Two-layer SOM scheme

5.2. The Computational Experiment

The SOM algorithm application has been extended to various fields: digital marketing [54], recommendation systems [21], visualization [27], and engineering [50]. The SOM allows a visualization of the interpretable sample, a flexible dimensional reduction that can give rise to good inferential models even in situations not previously evaluated.

In our experiment, we wanted to represent a limiting situation that sometimes occurs in some of these applications, a situation in which a small group of a population is nevertheless relevant and possibly narrowly bounded and deviates in some of its characteristics from the general context, being very similar in the rest, for example in the field of customer management of a bank, customer type profiles that vary from these general structures due to the effect of environmental components such as the place where they live. Another example is in the industrial field, in the management of raw mate-

Algorithm 2 Two-layer SOM algorithm

Require: $m, n, p \in \mathbb{N}\ K \geq 0 \vee S \subset X$ sample $\vee$ epochs number
Ensure: $\omega_k^i\ \forall K$
  Split $S = S_1 \cup ... \cup S_p$
  for $S_i\ i \in (1, ..., p)$ **do**
    Initialize $\omega_k\ \forall k \in (1...K)$ randomly
    for all epoch **do**
      {random loop over all input vector $x \in X$}
      for each x **do**
        {update weight vector ω_k according to eq. 1}
        Note: Including winner and neighborhood function
      end for
      Note: Not update α and σ because are function of the epoch
    end for
    return $SOM^i = \omega_k^i \in X, k = 1...m * n$
  end for
  $S := \cup_i SOM^i$
  Apply Algorithm 1 1-layer SOM
  return $SOM = \omega_k \in X, k = 1...m * n$

rials, which occasionally undergo small changes in their basic characteristics due to low-impact weather variations.

We now describe the computational experiment, specifying:

1. The stochastic model used to generate the data in a higher (original) dimension (three in our case);
2. The stratified sampling procedure to generate the data;
3. The structure of the one- and two-layer SOMs used to summarize or cluster the structure in a reduced (2D) dimension map;
4. How the initialization of the map nodes was performed;
5. The measurement of the conservation of topology;
6. Results in the form of visualization.

5.2.1. The Stochastic Gaussian Mixture Model for the Simulations

The stochastic model used for the simulations was a mixture of 3D normal populations, defined by a covariance matrix with low values and a vector of means, selected to create a mesh in the 3D space.

$$x - \mu_k = \begin{bmatrix} x_1 - \mu_{1k} & x_2 - \mu_{2k} & x_3 - \mu_{3k} \end{bmatrix} \tag{5}$$

$$f(x) = \sum_{k=1}^{K} \frac{\psi_k}{(2\pi)^{3/2}\sqrt{M_k}} e^{-\frac{1}{2}(x-\mu)M_k^{-1}(x-\mu)^t}, \tag{6}$$

where f is the density function of a multivariate Gaussian mixture, $\mu = \begin{bmatrix}\mu_{1k} & \mu_{2k} & \mu_{3k}\end{bmatrix}$ are the component mean vectors, ψ_k are the component weights, and M_k the component variance–covariance matrices. A six-component mixture was used with the mean and weights shown in Table 5.1, all lying on the hyperplane $x = z$. Figures 5.2 and 5.3 illustrate this.

Using the z-coordinate, we defined our outlier component and designed the simulation experiment with the following main features:

The outlier component was a low-weight clone of one of the other six components, but located in:

$$z = x - 0.1, \tag{7}$$

i.e., outside the hyperplane where the centroids of the remaining components lie, but very close.

Table 5.1: Gaussian mixture mean parameter definition

Component	**X**	**Y**	**Z**	ψ
1	0.88	0.71	0.88	3000
2	0.82	0.11	0.82	3000
3	0.46	0.21	0.46	3000
4	0.12	0.11	0.12	3000
5	0.19	0.81	0.19	3000
6	0.49	0.81	0.49	3000
Outlier	0.49	0.81	0.39	90

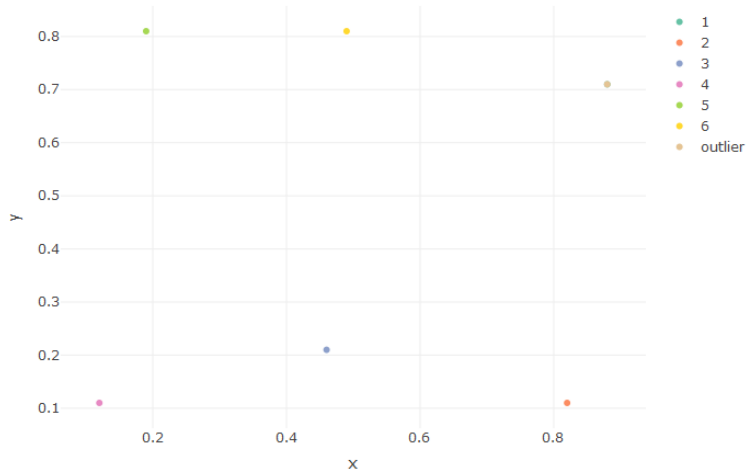


Figure 5.2: Mean vector of each component, in x, y coordinates.

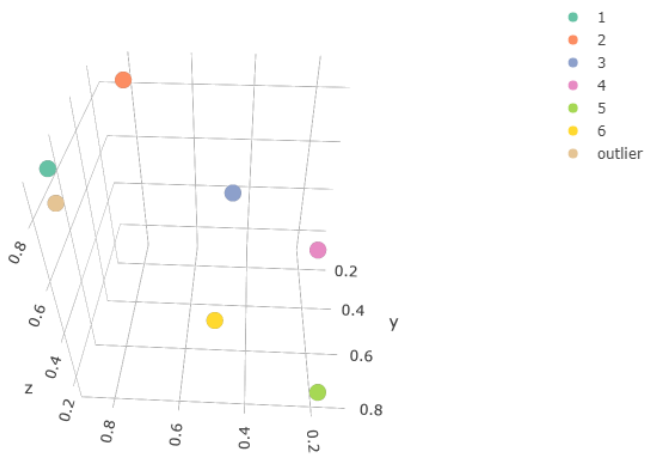


Figure 5.3: Mean vector of each component, in x, y, z coordinates.

As mentioned below, the variance-covariance matrices have the same low values for the six mixture components. Since z is equal to x (except for the outlier), Equation (8) represents the covariance matrix used by all of them for the x and y coordinates, but also for z which is a transform of x.

$$\sigma = \begin{pmatrix} 0.008 & 0.0064 \\ 0.0064 & 0.008 \end{pmatrix} \tag{8}$$

Figure 5.4, above, shows the distribution of the Gaussian components in xy-coordinates. The same covariance matrix generates similar densities, lying in different regions. We just found similarity between Component 1 and the outlier one, the only component with a lower frequency than the others. Figure 5.5, above, shows better how the outlier component lies on a parallel hyperplane, following Equation (7).

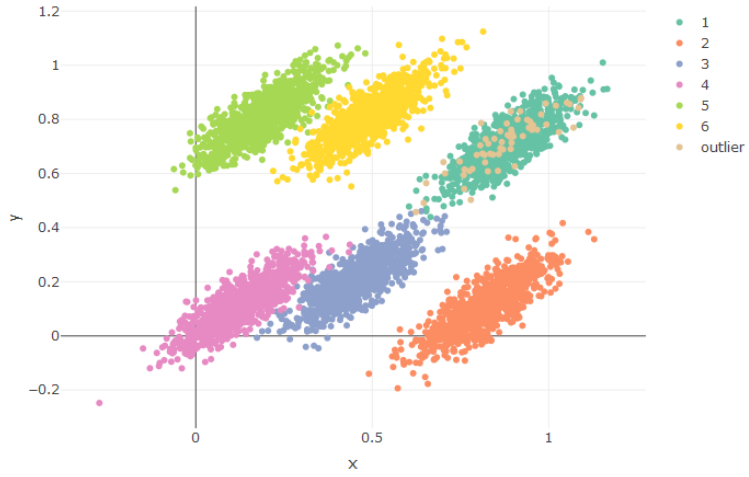


Figure 5.4: Generated components.

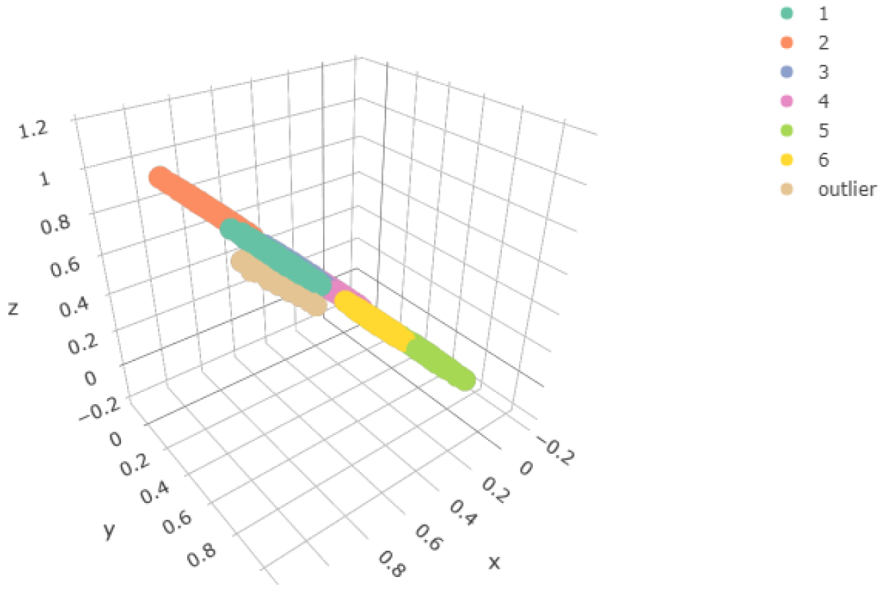


Figure 5.5: Population used in the experiment, x, y, z coordinates

5.2.2. The Sampling Procedure and Strata Configuration

Once we fixed the model parameters, we sampled from the data-generating process in accordance with the component weights. This sample would be the input data to the one-layer and two-layer SOMs, i.e., the same sample was be used for both cases. The size of the sample of each component was 3000 except for the outlier, for which it was 90; see Figure 5.5.

We defined different strata that simulated situations in which prior knowledge allowed us to segment into blocks with less unbalanced localized frequencies and fewer alternative components. The allocation of mixture components to strata is given in Table 5.2. Note that the relative frequencies of the components in some of the strata were significantly different from their global frequency.

Table 5.2: Allocation of mixtures components to strata.

Component/Strata	**First**	**Second**	**Third**	**Fouth**
1	3000	0	0	0
2	0	1200	1200	600
3	2400	0	0	600
4	0	1500	1500	0
5	0	0	3000	0
6	3000	0	0	0
Outlier	0	45	0	45

The stratification was our means to elicit prior information, which should have some relevant content, but not be too specific. We proposed a situation in which we had incomplete a priori information on an alternative grouping of individuals in which the measured characteristics (x, y, z coordinates) were not taken into account. If it were specific, we would be dishonest by putting too much information, which in real problems is not available in that detail. Usually, we do have some information, but not conclusive.

For the simulation, we considered a nontrivial number of strata (higher than two). In a real situation, the outlier can be distributed in many different ways along the strata. For illustration, we selected one of them, the one specified in Table 5.2, in which some strata did not include any outlier individual, some included the outlier with some (not very similar) other components, and another one where the outlier could be confused with other components because it was quite close to them. We worked with the minimum number of strata necessary to represent all three possibilities.

This situation can occur in various examples:

Medicine: such as the measurement of various symptoms (x, y, z coordinates) in patients at different stages of evolution of a disease (each group or mixture component) measured in different hospital centers (stratum). In each hospital (stratum), we can find patients at different stages of evolution (group), and each stage of evolution includes patients from different hospitals.

The outlier group corresponds to a small proportion of patients that have an extremely advanced stage of evolution. A more frequent group of patients has some similarities (symptoms x, y, x) with the outlying ones.

Marketing: clients on whom one measures different behavioral indicators (x, y, z) with a different state of engagement (each of the components of the mixtures or groups) extracted from samples stratified by premises with different locations (strata).

5.2.3. Structure and Parameters of One- and Two-Layer SOMs

The one-layer SOM was a 10×10 map. For the two-layer SOM, four 10×10 maps were built in the first layer, one for each stratum. Then, the 400 nodes of these maps were used as the sample for the second (final layer) SOM, which produced a 10×10 map as well, in such way that the comparison of the one-layer SOM was performed on equal footing. The topology in both cases was hexagonal.

Euclidean distances were used to select winning nodes. The values of the parameters were used for weight updating, in accordance with $\alpha_1 = 0.05$ and $\sigma_1 = 0.5$

5.2.4. SOM Node Initialization

Node initialization was performed in two ways:

1. Each of the four maps was initialized from the data in one of the four strata, i.e., one map per stratum; this would give equal weight in the initialization to all strata, regardless of their relative weights in the stochastic model;

2. Initialization taking into account the weights of the strata assigning numbers of nodes proportional to the sizes of the strata, i.e., strata with double the number of points assigned would have their data present in the initialization of twice the number of nodes.

5.3. Conservation of Topology

The SOM is intended to cluster the data and at the same time project them onto a lower dimensional space while preserving topology, i.e., in such way that points that are close in the original space belong to the same or close nodes (clusters) in the map and likewise (ideally) for those that are far apart or in-between.

The SOM has thus to pursue optimality in two simultaneous directions: cluster homogeneity and conservation of topology. It should be highlighted that a bicriterion optimization would result in being less optimal for the individual criteria than that which would be attained if only one criterion were pursued. As for cluster homogeneity, optimal means that clusters are as homogeneous internally as possible and, consequently, heterogeneous from one

to the other. Thus, for the given total sample variability, within-cluster variability should be minimized and, consequently, between-cluster variability maximized. For instance, it was expected that k-means would be more successful in cluster homogeneity, given that it can focus on it without "having to worry" about the conservation of topology.

Regarding the second criterion, in the preservation of topology, there are different perspectives: to preserve the topology and how to achieve this, although collaborative learning with the neighborhood structure always ensures a certain representativeness in this sense.

In our case, we chose the option of preserving the metric space, that is seeking to maintain the distances between the elements of the original space from their map centroids (weights). It is important to realize that dimension reduction makes a fully accurate representation impossible.

Given a sample of size n, there is an $n \times n$ symmetric distance matrix in the original space and a corresponding SOM distance matrix of equal size. Optimal would then mean minimizing the distance between these two distance matrices.

The distance matrix in the original space is rather straightforward: euclidean distances could be a reasonable solution, and they were, after all, the measure adopted for the SOM competition stage.

The SOM distances are more complex. To start with, if only distances between the node integer pairs in a 2D map were taken into account, then the node centroids would not be considered, and relevant information would be neglected. A sound criterion should thus take both distances into account, i.e., the 2D pairs of integers and the between-centroid (original space) ones. Introducing the structure of the map in the distance measurement is a challenge, to start with the two distances would be on different scales, it also adds one more factor to take into account when combining both. Here, we used a distance defined in terms of the two (distances) and considered that not only the winning nodes of the points have to be taken into account, but also their neighborhoods. In a nutshell, we measured the distance between the neighborhoods of the corresponding winning nodes.

In order to take into account the topological structure of the map when measuring the validity of the SOM, we worked with two distances that took into account the weights and topological structure: image-based and graph-based.

5.3.1. Image-Based Distance

This is based on creating an image map for each of the two points x_i, x_j involved; the dimensions of each image are the same as those of the SOM under study. The distance is a weighted average of the distances between the centroids of all possible pairs of nodes from the two images, the first element

in the pair stemming from the x_i image map and the second from the x_j map. The weights of the average depend on the distance between the node in question and the winning node. Two possible weights were contemplated:

(a) The first would be the product of two factors, a monotonic function of the distance from the winning node to the node in question in the first (x_i) map, and the second likewise for the second (x_j) one;

(b) The second would be interpreted as the joint probability of the two nodes, which would depend on how the nodes are seen for each image map. This would involve the product of four factors, the first two for the weights of the first (x_i) node from the point of view of both images and the second likewise for the (x_j) node.

Here, we applied option (a).

The image-based distance, which came from [68], was thus formally defined as follows:

1. Let $x_i, x_j \in \mathbb{R}^{\mathrm{n}}$ be the sample individuals:

   a) $n_{pq} \in som_{map}$ the pq-th node in the map:

   b) We call $\varpi_{pq} \in W \subset \mathbb{R}^n$ the centroids (projection-representative in the original space);

   c) We call the neighborhood of N_{pq} as $V(N_{pq}) = \{N_{rs}/d_2((p,q),(r,s)) <= \alpha\}$ with $p, q, r, s \in \mathbb{N}$;

   d) We denote by $som_{map} = (NxN, W)$, where $N \subset \mathbb{N}, W \subset \mathbb{R}^{\kappa}$ and $som_{map}(x_i) = (N_{V(N_i)}xN_{V(N_i)}, W_{V(N_i)})$, the projection of the original map onto the neighborhood space for the node assigned to point x_i;

   e) A map may be considered an image, a pixel matrix where each pixel has been assigned the weight vector of the centroid of each node. We shall now lay out for our problem the modifications on distances used in such cases;

   f) Let us define the global distance, which combines the spatial distance with the pixel intensity:

      1) $GD(A,B) = \frac{2}{N^2(N^2-1)} \sum_{A_{ij}} \sum_{B_{lm}} \delta(A_{ij}, B_{lm})$;
      2) $d^D((i,j),(l,m)) = \frac{|i-l|+|j-m|}{N}$;
      3) $\delta(A_{ij}, B_{lm}) = \alpha d^D((i,j),(l,m)) + \beta GR(a_{ij}, b_{lm})$ with $0 \leq \alpha, \beta \leq 1,\ \alpha + \beta = 1$;
      4) $GR(a_{ij}, b_{lm}) = |a_{ij} - b_{lm}|/max(a_{ij,}b_{lm})$. This is the definition for black and white images. In our case, we worked in $W \subset \mathbb{R}^{\kappa}$, so we replaced it with the normalized euclidean distance;

   g) Once a framework has been established, we specified image distances within the framework. We now considered the "images",

the submaps of the neighborhood assigned to each point, and we assigned a variable to each image, an Iintensity matrix that contains the values determined by the neighborhood:

1) $I/I_{ij}(x_i) = f(N_{ij}, N_{nl}) \in (0,1)$, where $N_{n,l} = N(x_i)$
2) f may be defined in several ways, but here we have chosen: Equation (11)

$$f(N_{mk}, N_{nl}) = \begin{cases} 1\, winner \\ 0.75\, first - level\ neighborhood \\ 0.5\, second - level \\ 0.25\, other\ cases \end{cases} \tag{11}$$

3) We use the distance above to compare the two images, by means of the following transformation:
4) We define $dNodes(N_{nl}, N_{mk}) = \lambda_{ij}(N_{nl}, N_{mk})\delta(N_{nl}, N_{mk})$ and then $GD(SOM(x_i), SOM(x_j)) = \frac{2}{N^2(N^2-1)} \sum_{SOM_i} \sum_{SOM_j} dNodes_{ij}(A_{nk}, B_{lm})$
5) $\lambda_{ij}(N_{mk}, N_{nl}) = I_{mk}(x_i) I_{nl}(x_j)$
6) A hierarchy within the distances is thus defined:

$$\lambda_{ij}(N_{mk}, N_{nl}) = \begin{cases} 1\, both\ winners \\ 0.75\, winner\ and\ 1 - level \\ 0.565\, both\ 1 - level\ neighborhood \\ 0.5\, winner\ and\ 2 - level\ neighborhood \\ 0.375\ 1 - level\ and\ 2 - level \\ 0.2\, both\ 2 - level \end{cases} \tag{10}$$

5.3.2. Graph-Based Distance

We took all paths that went from the (x_i) winning node to the (x_j) winning node and obtained the shortest one by application of the Kruskal algorithm [34]. For this path, we added up all the distances between the centroids of the consecutive nodes. Neighborhoods were needed for (x_i) and (x_j), since whether two adjacent nodes will be considered consecutive in a path depends on howthe neighborhood is defined (diagonal nodes could be considered nonconsecutive). Figure 5.6 illustrates this.

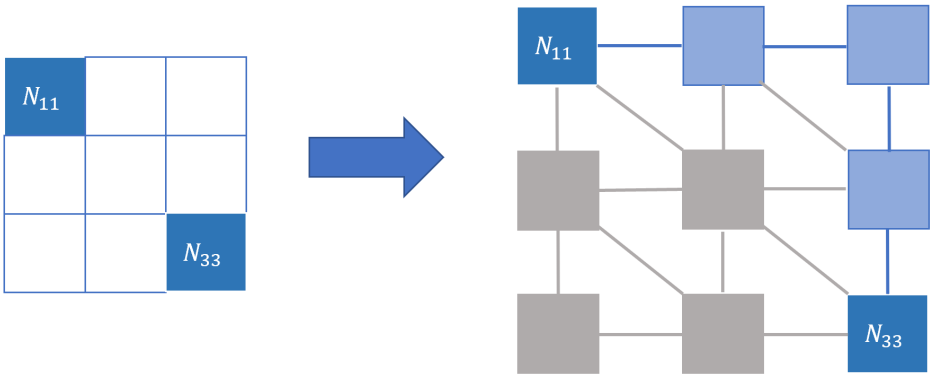


Figure 5.6: In this case, the distance comes from considering our map as a graph where the nodes are the nodes of the map and the edges represent the chosen neighborhood. The distance between winning nodes, representing the points in the new space, will be the minimum path in that graph, in this case, represented with the nodes and edges in blue.

The graph-based distance is thus formally defined as follows:
Let $x_i, x_j \in \mathbb{R}^n$ be the individuals in the sample:

1. $n_{pq} \in som_{map}$ the pq-node in the map:
   a) Let us call $\varpi_{pq} \in \mathbb{R}^n$ the projection-representative in the original space; generally, it is approximated as: $\alpha < 1$, $\omega_{pq} = \sum_{x_i/N(x_i)=N_{pq}} x_i + \sum_{x_j/N(x_j)\in V(x_i)} \alpha x_j / card(V(N_{pq}))$;
   b) The neighborhood of N_{pq} is defined as follows $\alpha < 1$: $V(N_{pq}) = \{N_{rs}/d_2((p,q),(r,s)) <= c\} p,q,r,s \in \mathbb{N}$;
2. Let $N(x_i) = \arg\min_{N_{pq}\in som_{map}}(d(x_i, w_{pq}))$;
3. Let $G = (V, E)$ be a nondirected graph, where $V = \{N \in som_{map}\}$ and $E = \{(N, M), N, M \in som_{map}/N \in V(M), M \in V(N)\}$;
4. We define for each side its weight $p(e_{(N,M)}) = d(N, M) = d(w_N, w_M)$;
5. Let us define a path from N to M as a sequence $(e_1, e_2, e_3, ...e_m)$ of sides such that e_i ends where e_{i+1} begins and e_1, e_m begins or ends in N, M, respectively;
6. $\mathcal{C}_{\mathcal{NM}}$ the set of all paths beginning in N and ending in M;
7. By application of Kruskal's algorithm, one can find the minimal path from any node N to any other one, defined as $C_{NM} = argmin_{c\in\mathcal{C}}\{\sum_{e_i\in C} p(ei)\}$;
8. The distance between the points in the original space is thus: $d(x_i, x_j) = length(C_{N(x_i),N(x_J)})$.

5.3.3. Toy Example

Toy example to illustrate image and graphs distances. Let us consider the following map where the 3D vectors in brackets are the weights (centroids) of the nodes.

$$\begin{bmatrix} (9,8,9) & (8,8,8) & (8,7,7) \\ (8,9,8) & (8,7,8) & (7,8,6) \\ (7,7,7) & (7,6,5) & (5,5,5) \end{bmatrix} \tag{11}$$

Let us compute the distances between two points x_i and x_j such that their winning nodes are (1,1) (upper left) and (3,3) (lower right), respectively.

5.3.3.1. Image Distance

For node (1,1), the weight matrix is:

$$\begin{bmatrix} 1 & 0.75 & 0.5 \\ 0.75 & 0.5 & 0.25 \\ 0.5 & 0.25 & 0.25 \end{bmatrix}$$

and likewise for (3,3):

$$\begin{bmatrix} 0.25 & 0.25 & 0.5 \\ 0.25 & 0.5 & 0.75 \\ 0.5 & 0.75 & 1 \end{bmatrix}$$

The δ matrix for all nine combinations of nodes in both maps is:

Translating the point distance to the map space as follows:

$$d^D(map, map) = \tfrac{1}{N}\left[d^D((i,j),(l,m))\right]_{(i,j),(l,m)}$$

$$d^D(map, map) = \tfrac{1}{9}\begin{bmatrix} & (1,1) & (1,2) & (1,3) & (2,1) & (2,2) & (2,3) & (3,1) & (3,2) & (3,3) \\ (1,1) & 0 & 1 & 2 & 1 & 2 & 3 & 2 & 3 & 4 \\ (1,2) & & 0 & 1 & 2 & 1 & 2 & 3 & 2 & 3 \\ (1,3) & & & 0 & 1 & 2 & 1 & 2 & 3 & 2 \\ (2,1) & & & & 0 & 1 & 2 & 1 & 2 & 3 \\ (2,2) & & & & & 0 & 1 & 2 & 1 & 2 \\ (2,3) & & & & & & 0 & 1 & 2 & 1 \\ (3,1) & & & & & & & 0 & 1 & 2 \\ (3,2) & & & & & & & & 0 & 1 \\ (3,3) & & & & & & & & & 0 \end{bmatrix}$$

$$d(map, map) = \begin{bmatrix} & (1,1) & (1,2) & (1,3) & (2,1) & (2,2) & (2,3) & (3,1) & (3,2) & (3,3) \\ (1,1) & 0 & 1.41 & 2.44 & 1.73 & 1.73 & 3.6 & 3 & 4.89 & 6.4 \\ (1,2) & & 0 & 1.41 & 1 & 1 & 2.23 & 1.73 & 3.74 & 5.19 \\ (1,3) & & & 0 & 2.23 & 1 & 1.73 & 1 & 2.45 & 4.13 \\ (2,1) & & & & 0 & 2 & 2 & 2.45 & 4.35 & 5.83 \\ (2,2) & & & & & 0 & 2.45 & 1.41 & 3.31 & 4.69 \\ (2,3) & & & & & & 0 & 1.41 & 2.23 & 3.74 \\ (3,1) & & & & & & & 0 & 2.23 & 3.46 \\ (3,2) & & & & & & & & 0 & 2.23 \\ (3,3) & & & & & & & & & 0 \end{bmatrix}$$

The expression for the distance is:

$$GD(som_{11}, som_{33}) = \lambda_{11}(N_{11}, N_{12}) * \lambda_{33}(N_{11}, N_{12}) * \delta(N_{11}, N_{12}) +$$
$$+\lambda_{11}(N_{11}, N_{13}) * \lambda_{33}(N_{11}, N_{13}) * \delta(N_{11}, N_{13}) + \ldots$$
$$+\lambda_{11}(N_{11}, N_{33}) * \lambda_{33}(N_{11}, N_{33}) * \delta(N_{11}, N_{33}) + \ldots$$
$$+\lambda_{11}(N_{33}, N_{33}) * \lambda_{33}(N_{33}, N_{33}) * \delta(N_{33}, N_{33})$$
$$= 1 * 0.25 * (\alpha 0 + \beta 0) + 0.75 * 0.25 * (\alpha 2 + \beta 2.44) + \ldots$$
$$\cdots + 1 * 1 * (\alpha 4 + \beta 6.4) + \cdots + \ 1 * 0.25 * (\alpha 0 + \beta 0)$$

Figure 5.7 schematizes the example described.

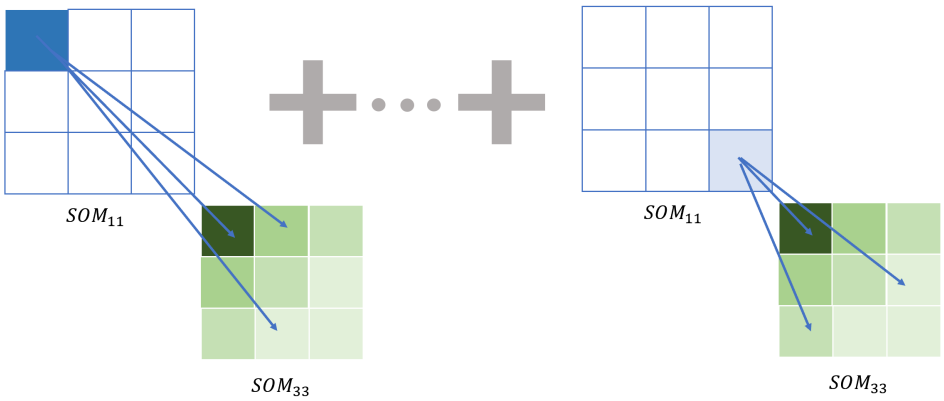


Figure 5.7: In this graph, we summarize the idea of the metric. We measure the distance between the maps associated with each point. It will generally be the same map except for the winning node and its neighborhood. Here, we can observe that when comparing the winning node N11 of the SOM11 map (node in intense blue) with the nodes of the SOM33 map whose winning node is 33, we have to take this spatial relationship into account. The gradient of the colors represents the reduction of the effect of the distance between these nodes in the global distance, which is calculated with the lambda function.

5.3.3.2. Graph Distance

We obtained, for illustration, the distance for one of all possible paths between the two points. The Kruskal algorithm selects, within all possible paths, the one that gives the smallest distance and that will be taken as the final graph distance between the two points.

The neighborhood structure determines if, for instance, the path (1,1), (2,2), (3,3) is valid or if, on the contrary, going from (1,1) to (2,2), one should first pass through (1,2) or (2,1).

Let us choose the following path:

$$P = \{(1,1),\ (1,2),\ (1,2), (1,3), (2,3), (3,3)\}.$$

$d_{graph}(x_i, x_j) = d_{graph}(map(1,1), map(3,3)) = \sum_{v_i \in P} d(v_i, v_{i+1})$
$d_{graph}((9,8,9),(5,5,5)) = d((9,8,9),(8,8,8)) + d((8,8,8),(8,7,7)) + \ldots$
$+d((8,7,7),(7,8,6)) + d((7,8,6),(5,5,5)) =$
$= 1.4142 + 1.4142 + 1.73 + 3.74 = 8.2984$

5.4. Results

The results of the full simulation exercise are given separately for the one-layer and two-layer maps and then compared.

The component centroids are plotted in section 5.2: The Computational Experiment, and it can be observed that the first six lay in a plane, while the outlier center was outside it.

The sample points are represented in Figure 5.5, where, since the variances of the three variables in the components were small, the clouds of the different components are relatively tightly displayed along the plane.

5.4.1. One-Layer SOM

As mentioned above, the number of iterations was 2000, i.e., we ran the algorithm over the full dataset 2000 times.

Through the following figures, we can understand its application and draw different conclusions.

In Figure 5.8, we represent the values of each of the coordinates of the centroids of the nodes of the resulting SOM using a color scale. Notice that the maps referring to the x (first) and z (third) coordinates match exactly.

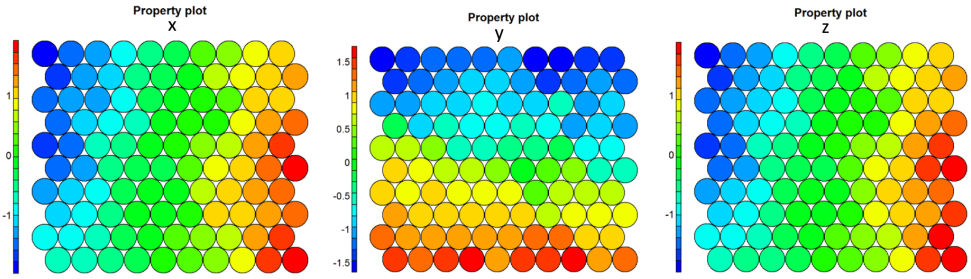


Figure 5.8: 1-layer Final SOMs for coordinates x, y, z, respectively.

Figure 5.9 represents in the three-dimensional space x, y, z (original space) the mean value of each of the mixtures together with the weights of the different nodes of the one-layer map. Notice how the only mean that is not overlapped by the centroids is that of the outlier component. It is the only component not correctly represented.

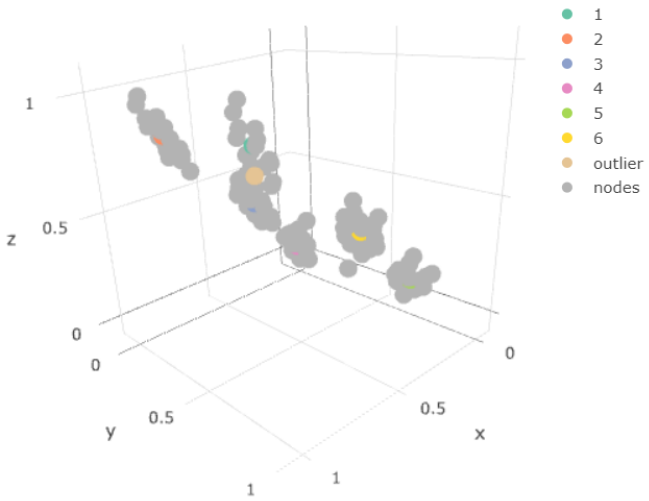


Figure 5.9: Final SOMs for the original spaces.

Figure 5.10 shows all the points belonging to the outlier component and the centroids of the nodes of the one-layer map. Again, we observe how there is no type of overlap.

The components included in the hyperplane $z = x$ were well represented in the SOM, and there were many nodes whose weights were located in the original space close to the component means.

However, our outlier component, due to its proximity to one of the components included in the hyperplane $z = x$, was not represented by any of the nodes Figure 5.9. This was also observed when comparing the original x, y, z spaces SOM property plots, which represent each of the nodes in the map, as shown in Figure 5.8: The outlier values were not represented in the weights of the map nodes, as expected. This behavior is particularly striking in the z-property plot.

In Figure 5.9, the node weights of the one-layer SOM were added to the component mean plot, confirming the above-mentioned situation. We observe in Figure 5.10 that the sample of points of the outlier component did not coincide with any of the weights obtained.

In Figure 5.11, we can see in three dimensions the points of the sample together with the weights of the nodes obtained (node tag), eliminating the component closest to the outliers. We validated the statement again by checking that the weights were located in the area of the component, the closest to the outlier component, but no weights corresponded to the outlier itself.

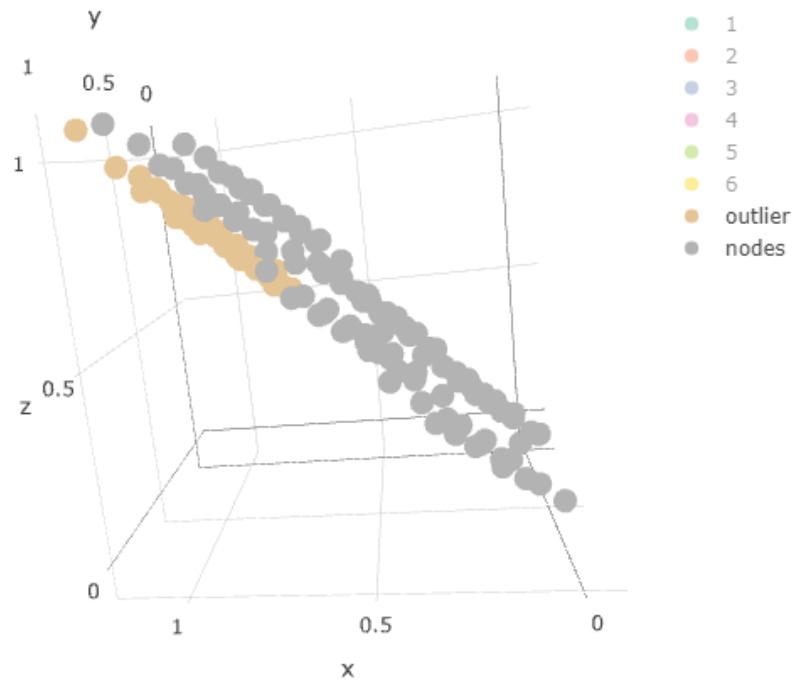


Figure 5.10: Weights over outlier samples.

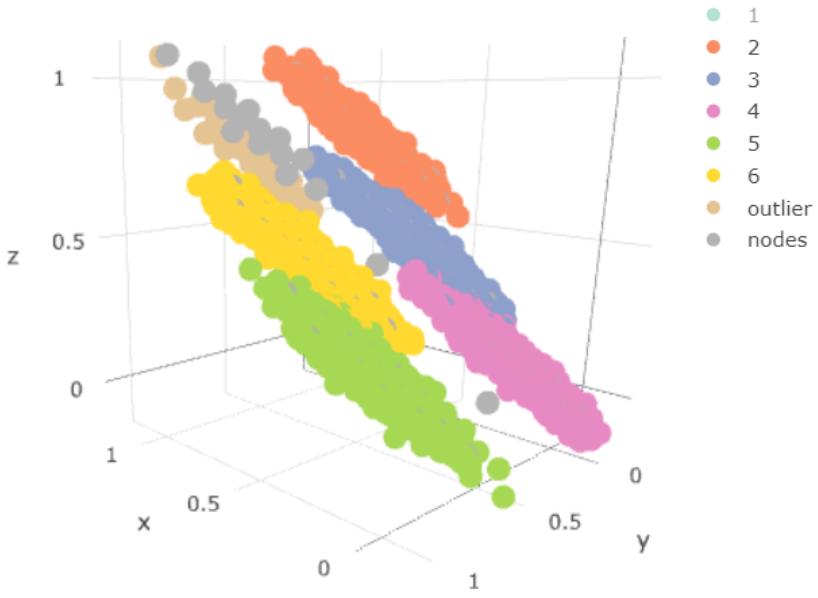


Figure 5.11: Three-dimensional representation of the data obtained and the weights, without the sample of the component.

In Figure 5.12, we represent the x and y coordinates of the population together with the x and y coordinates of the nodes obtained. We show the projection of Figure 5.9 onto the xy plane. It was observed that the outlier points were distributed more or less evenly along the remaining components'points projections, which obviously did not include the z coordinate.

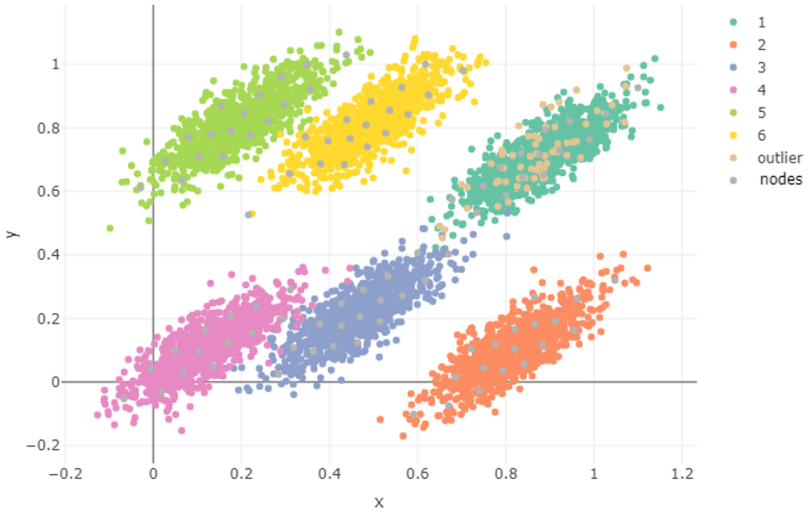


Figure 5.12: Projection.

5.4.2. Two-Layer SOM

We defined in Section 5.1 the same distribution of components for all strata. Each stratum included values from three components, and each component was included in more than one stratum. The frequency of appearance of each of the components in each stratum was similar except in the second one, where the outlier component maintained the overall frequency rate.

5.4.2.1. First (Intermediate) Layer Results

Each map was generated using one stratum as the input. In each case, all components were well represented and kept the topology, because intermediate centroids appeared between far-apart components.

If we focused on the two intermediate maps where the outlier component was included, we may observe that if the relative frequencies of the outlier and the remaining components were balanced, there was a larger number of nodes in the map with nearby centroids in the original space. This increased the probability that they appeared in the last layer.

To illustrate this, we show results for some of the strata.

In the case of Figure 5.13, the first stratum did not contain any sample point of the outlier component. However, it is interesting to view how it generated a smoother transition. The points in the "circle" in blue in Figure 5.14 are the map nodes that were generated to maintain the topology of the system, establishing, as pointed out by [2, 67], the distance.

In Figure 5.15, we show the node weights for Stratum 4, which contained data from the 2, 3 components and the outlier component. Note that the outlier points now appeared, as opposed to the situation in the single-layer map, but also that intermediate points were generated to make the transition smoother. This gave us a clue about what would happen to an even more separate outlier.

In Figure 5.16, we show the 2D projection of the first-layer SOM for Stratum 4; the points are scattered around two straight lines because the outlier points are aligned with those of Component 3; the orange one (outlier centroid) is in (0.46, 0.21); it is more difficult for it to move "continuously" from green to orange or green to blue than from blue to orange where there is continuity.

In Figure 5.17, we show the property plots for the first-layer SOM of Stratum 4; note the z plot where the outlier nodes are on the lower right corner. We obtained what we expected: by reducing the sample size and keeping only some of the less neighboring components, it became easier to identify the elements of the low-frequency component. Let us look at how this could be extended to the second layer.

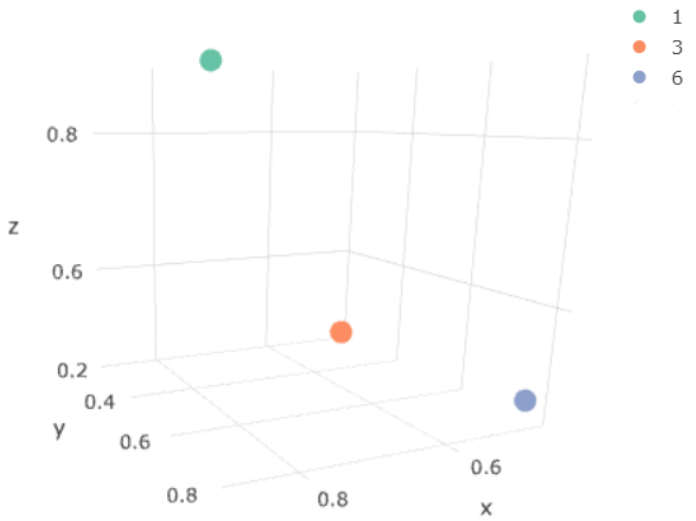


Figure 5.13: The centroids of the first-layer SOM for the first stratum.

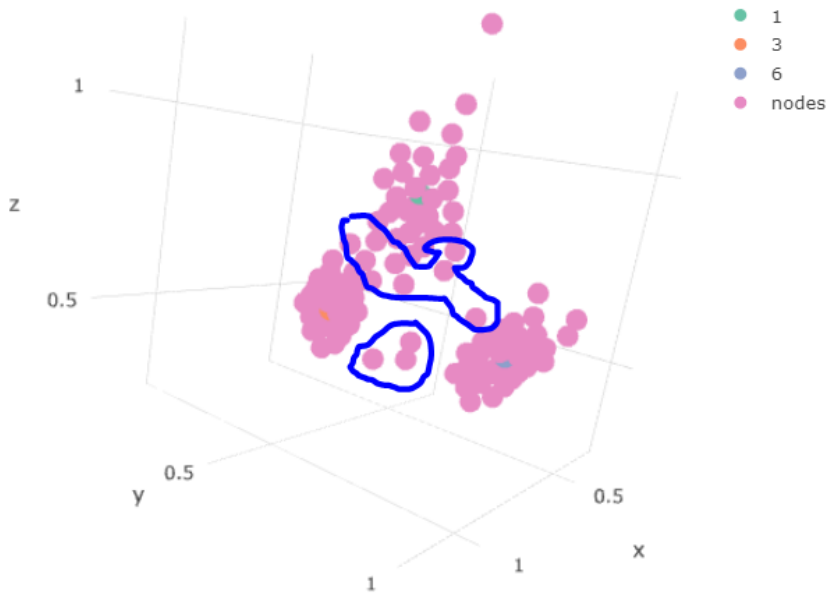


Figure 5.14: Status of the weights associated with the map generated for the first stratum.

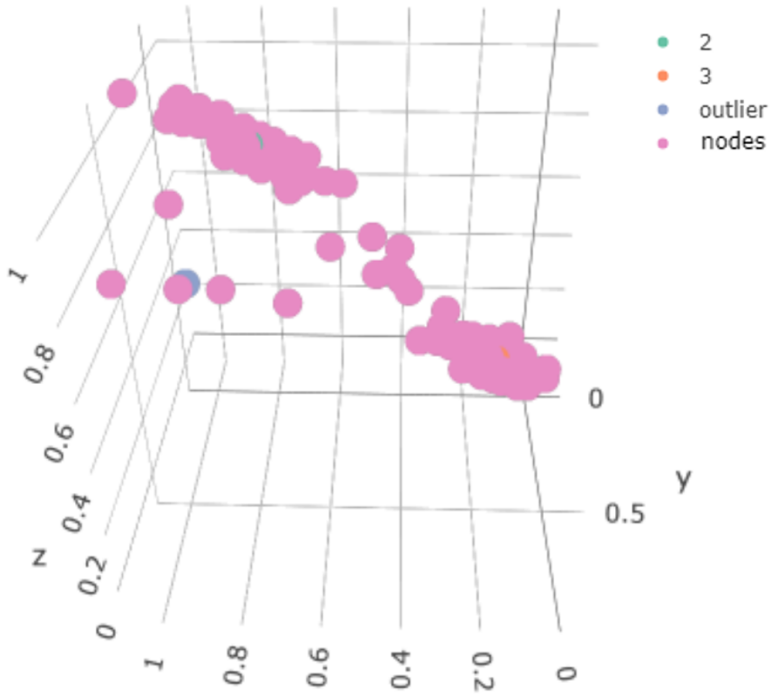


Figure 5.15: Status of the weights associated with the map generated for the second stratum.

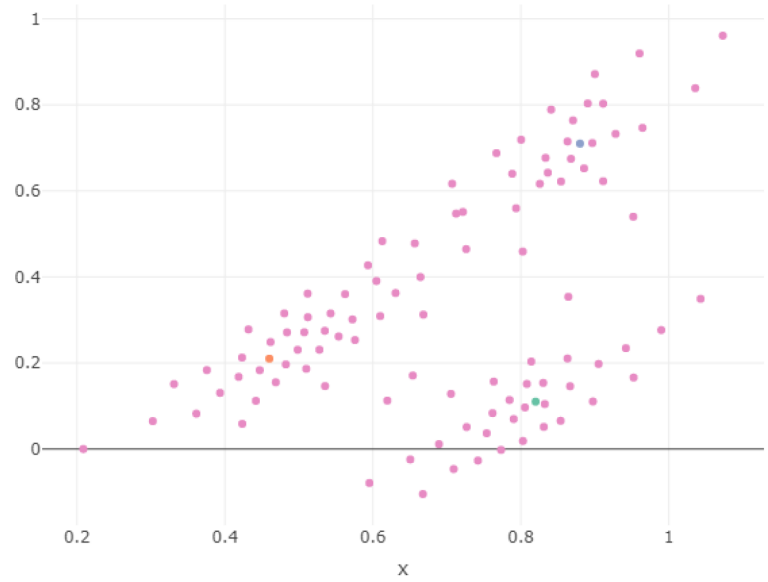


Figure 5.16: Two-dimensional plot of the weights in the second stratum SOM.

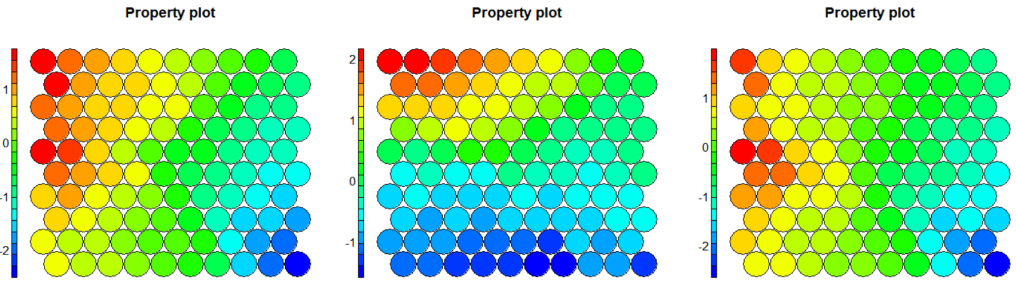


Figure 5.17: Property plots of Stratum 4.

5.4.2.2. Second-Layer Results

The results of this last layer substantially improved the single-layer alternative. We verified that the outlier component was well represented; several centroids of the map nodes were located in the original space in the density zone of the outlier component.

We show the property graphs for the final SOM layer in Figure 5.18: if we focus on the z graph, we can observe the presence of the outlier nodes, as well as the relatively smoother transition than in the one-layer case of the atypical nodes to those of the remaining components. In Figure 5.19, we verify that, now, some of the map weights were close to the centroid of the outlier component. This made this model more representative of the

situation in the original space, and this can also be observed in the results from the SOM metrics in Section 5.4.3.

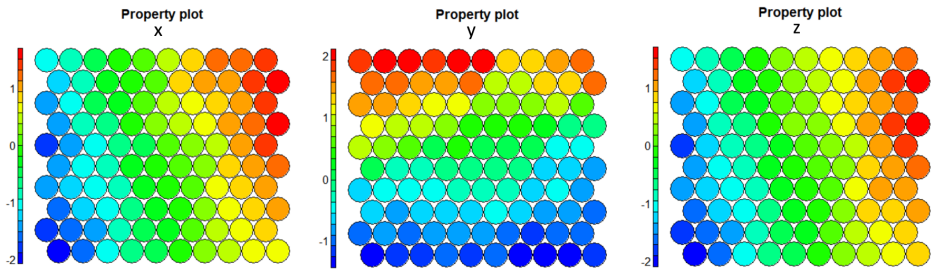


Figure 5.18: Two-layer SOM, last layer, final maps for coordinates x, y, z, respectively.

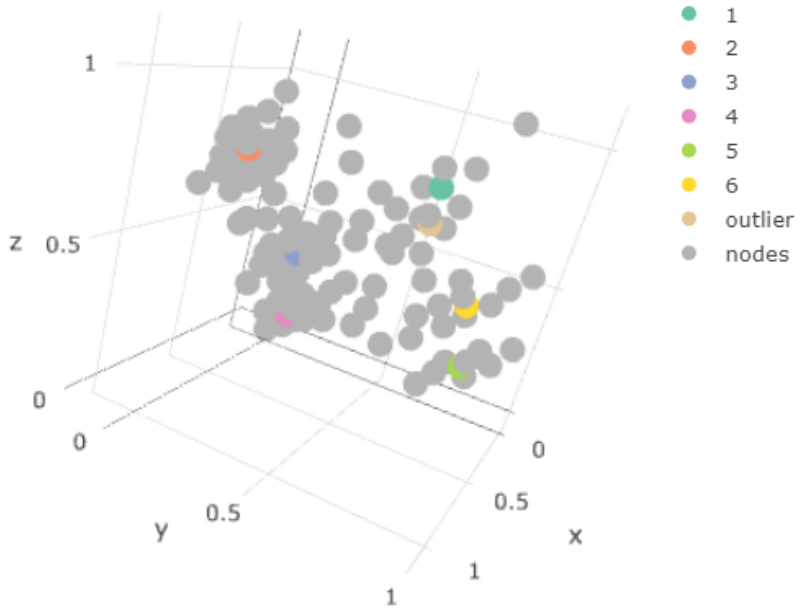


Figure 5.19: Final 3D result.

5.4.3. Conservation of Topology

In Table 5.3, we show the image and graph distances for the one- and two-layer SOM in the two examples.

We sought to validate the preservation of the topological space by checking that the metric space was maintained. We checked that the distances between the points in the original space were preserved when projecting them on the map. We show the distance between the matrices of the distances between the points of the original space and the matrix of the distances

between their representatives on the SOM in the original space. We used the euclidean distance and, to measure the distance between the weights (representatives), the graph-based metrics (where we considered the nodes linked to their neighborhood as representative) or on images (where we considered the map with its topological structure as representative).

This is one of the greatest properties of the SOM compared to other clustering algorithms such as K-means, DBSCAN, or dimensional reduction such as T-SNE or UMAP [33]. The main difference is to introduce "fictitious" areas on the map, to which there are no assigned points in the original space, to preserve topology. In this way, it is more robust when making inferences based on topology even on points in space that, however, have not yet appeared in the sample.

Table 5.3: Results of the different distance metrics of the algorithms used.

Model	Example	Image Distance	Graph Distance
one-layer SOM	1	32456	5032
two-layer SOM	1	22456	4625
one-layer SOM	2	22456	4048
two-layer SOM	2	17456	3740

5.5. Interpretation of the Results and Concluding Discussion

The classic single-layer model obviates low-frequency components located close to similar higher frequency ones.

That is why we developed the multilayer alternative with data segmentation in layers, the two-layer SOM. If it adequately represents the total set of points in the sample in addition to maintaining a better topological structure, it is a valid solution for its application in various areas where we can find these disparate low-frequency components, even more so if they can only be fuzzily delimited.

We encountered several challenges. First was to define a stochastic model through which one can illustrate the contribution of our methodology, for example the low-frequency group of the population needed to be represented because, in the one-layer strategy, it was absorbed by higher frequency neighboring groups. The second was that an adequate stratified sampling procedure had to be developed. Some, but not excessive, prior information was required for adequate outlier group detection. Third, the computational complexity had to be controlled; parallelization was essential to this end. Fourth was the definition of one or more adequate distance measures to quantify conservation of topology: traditional distances measures were not

satisfactory for our specific problem because some low-frequency regions of the population were not accounted for.

From the results obtained in this thesis, it can be concluded that:

1. The classic single-layer model ignored low-frequency components located between similar higher frequency ones. This was reflected in the distance matrix, as can be seen in the developed metrics. One was not adequately representing the original space or translating its topology. Not only was one ignoring one set of the sample, but this affected the representation of other points in the original space;

2. The stratification of the data (which may be given by prior knowledge) allowed, in two-layer SOMs, generating a map in which these components were also well represented;

3. For the two-layer map, stratification would result in the generation of intermediate-layer nodes that would represent points lying between components far apart, but included in the same stratum, and as inputs to the last layer, they would end up appearing to generate a more representative map, also in a topological sense.

Therefore, the methodology developed for the bilayer SOM was proposed to solve the problem of the representativeness of low-frequency elements that can be assigned to other components of the system, as well as a better strategy to maintain the topological structure understood as distances.

Conclusiones y Trabajo futuro

Resumen: En este apartado se presenta un resumen que aclara las conexiones entre las propuestas incluidas en esta tesis y los objetivos planteados al inicio de la misma, a fin de cumplir con su evaluación.

No es posible llevar a cabo esta tarea sin establecer un vínculo entre los ideas y objetivos con las contribuciones desarrolladas en esta memoria, y que de alguna manera sustentan el trabajo de investigación presentado, de cara a la comunidad investigadora.

Para ello, se seguirá la estructura de objetivos expuesta en el Capítulo 1, Sección 1.1, tratando de relacionar las diferentes aportaciones con los objetivos abordados.

Conclusiones

En conclusión, la investigación se ha centrado en el uso de la programación funcional y la teoría de categorías en el desarrollo de sistemas expertos, con un enfoque específico en el Self-Organizing Map (SOM) como herramienta para la representación del conocimiento. El principal desafío encontrado fue la representación incompleta del espacio probabilístico original al aplicar un one-layer SOM. Se definió un experimento computacional usando mixtura de gaussiana para ilustrar los desafíos de usar un one-layer SOM para la representación del conocimiento. Para validar la preservación de la topología, se han incluido algunas definiciones de distancias basadas en imágenes y gráficos, que se utilizaron en el algoritmo two-layer SOM. Los resultados mostraron que el uso de un SOM de two-layer puede mejorar la representación del espacio probabilístico.

El two-layer SOM tiene el potencial de ser un funtor, que podría explorarse en trabajos futuros. Esto podría implicar investigar el uso de la teoría de categorías para modelar las relaciones entre diferentes componentes del SOM y desarrollar algoritmos que aprovechen las propiedades funcionales

del SOM. Además, se podría explorar el uso de la metodología bayesiana en el desarrollo de sistemas expertos, incluido el desarrollo de algoritmos para medir y representar la incertidumbre.

En general, el uso de la programación funcional y la teoría de categorías en sistemas expertos tiene el potencial de mejorar la representación y el análisis de conocimientos complejos, y es probable que futuras investigaciones en esta área arrojen conocimientos valiosos y aplicaciones prácticas.

Relación entre objetivos propuestos y contribuciones

***Objetivo 1**: Realizar implementaciones con procesamiento en paralelo sobre datos distribuidos*

En la sección de background metodológico (Sección 3), se han introducido en primer lugar, herramientas Big Data y algunos paradigmas de programación funcional diferentes, como Spark (Section 3.2.4) para trabajar con procesamiento paralelo sobre datos distribuidos (Section 3.3), paradigma en se engloban los distintos proyectos que han sustentado algunas de las contribuciones. Spark ha sido específicamente utilizado para mejorar el performance de Two-layer SOMs [51] (Section 3.3.5), que presenta, frente a otras soluciones, la desventaja de la complejidad computacional. Este resultado se puede encontrar en the One-layer vs Two layer (Chapter 5) que se corresponde con la principal contribución (**Contribution** [69]).

1. *Comparar los resultados de rendimiento sobre diferentes configuraciones y sistemas de optimización desde el punto de vista teórico.*

   En el Capítulo 5, Sección 5.2 Üna capa vs Dos capas", se analiza la posibilidad de aumentar el número de capas y de mapas por capa, concluyendo que la estructura del algoritmo y el diseño computacional permiten esta extensión de manera sencilla.

2. *Generar procedimientos gráficos que permitan la visualización de los resultados.*

   La visualización de datos implica, por definición, efectos secundarios, por lo que el uso de la programación funcional difícilmente proporciona valor añadido. Es por eso que hemos utilizado R para producir visualizaciones interactivas de los resultados de la modelización y los algoritmos (Sección 5.4). Esto contribuye a considerar a SOM como un sistema experto completo, ya que proporciona al usuario no experto una visión fácilmente interpretable del espacio de datos original y le permite interactuar y explorar diferentes áreas, aprovechando la base de conocimientos y representación proporcionada por el SOM de dos capas.

Objetivo 2: *Evolución del algoritmo a una estructura SOM multicapa que permitiera identificar estructuras topológicas jerárquicas en el espacio original y disminuir el efecto de infrarrepresentación de zonas de baja frecuencia. Sokolovska [58].*
Al usar SOM como una base de conocimientos, es necesario tener una visualización útil, y todas las situaciones previstas en el espacio original deben aparecer en el mapa. Aquí nos encontramos con el problema de que a veces solo se encuentra una solución en el SOM de una sola capa tradicional. Se ha encontrado un medio para la representación y simulación de este problema, a través del cual se puede validar científicamente nuestra solución alternativa propuesta. Esto se puede encontrar en la **Contribución** [69] y en el (Capítulo 5, Sección 5.4).

1. Diseño de métricas de comparación de estructura de distribución de puntos en estratos con respecto a la población original Yin [77] and Van Hulle [28].

2. Métrica de evaluación de mantenimiento de topología mediante ordenamiento de vecindarios, generación de estructuras de grafo.

   En el análisis del estado del arte, se ha comprobado que la evaluación de la conservación topológica en SOM, es un campo abierto de estudio. También comprobamos la necesidad de definir métricas alternativas para evaluar la conservación de la topología desde el punto de vista de la preservación de las distancias asociadas al espacio topológico derivado del espacio métrico. La relación entre los espacios topológicos y métricos se puede encontrar en la sección 3. Algunas medidas de evaluación topológica se pueden encontrar en la Sección 4, y muestras medidas de distancia basadas en imágenes/gráficos se presentan en el Capítulo 5, Sección 5.3.

3. Estudio de efectos de diversas formas de estratificación en el resultado.

   Se ha construido un proceso de simulación basado en mixtura gaussianas, que permite validar el espacio probabilístico original de generación de datos y la ubicación de estos puntos del estrato de baja frecuencia en este espacio, mostrando así que el two-layer SOM se comporta satisfactoriamente en el situaciones evaluadas. Esto da como resultado un modelo que puede ser considerado un funtor soble la categoría del espacio probabilístico, y, por lo tanto, como una sólida base de conocimiento para un sistema experto. Capítulo 5, Sección 5.3.

4. Justificación metodológica y experimental de la paralelización [74]

   Se ha comprobado que nuestro esquema de paralelización no distorsiona el rendimiento de la propuesta two-layer, aunque los resultados pueden sufrir ligeros cambios debido a la aleatoriedad en la selección

de los nodos y la distribución de los estratos a lo largo de los nodos computacionales.

Publicaciones derivadas de la tesis

En esta sección se proporciona una lista de las publicaciones que se han derivado de esta tesis, incluyendo los títulos, autores y las revistas o conferencias en las que se han publicado. A través de esta sección, los lectores pueden obtener una comprensión más profunda del impacto y alcance de la investigación realizada, y también pueden explorar el contexto más amplio en el que se han difundido y discutido estos hallazgos.

Contribución 1: González-Pérez, B., Núñez, C., Sánchez, J., Valverde, G. (2021). : *Expert System to Model and Forecast Time Series of Epidemiological Counts with Applications to COVID-19* Mathematics. 9. 1485. 10.3390/math9131485.(JCR)

Contribución 2: López, V., Urgeles, D., Sanchez, O., Valverde, G. (2019): *Big Data in Healthcare and Social Sciences: Bip4Cast as a CAD System.* 10.4018/978-1-5225-7501-6.ch046 (JCR)

Contribución 3: López, V., Valverde., G., Anchiraico, J., Urgeles, D. (2016). *Specification of CAD Prediction System for Bipolar Disorder* ,In Proceedings of Conference on Uncertainty Modelling in Knowledge Engineering and Decision Making (FLINS 2016), 162-167. 10.1142/97898131469760028

Contribución 4: López, V., Miñana, G., Sanchez, O., González-Pérez, B., Valverde, G. (2015): *Big+Open Data: Some applications for a Smartcity.* 384-389. 10.1109/PIC.2015.7489874.

Futuras líneas de investigación

La mayor parte del trabajo presentado en esta tesis es fruto de la investigación realizada para su publicación en dos revistas JCR y otros recursos durante el tiempo dedicado al Programa de Doctorado en IMEIO. Muchos de los resultados se cierran definitivamente en estas páginas, pero aún quedan algunas líneas de investigación abiertas.

El uso de nuevas metodologías de procesamiento de Big Data y aprendizaje por ensamblaje en combinación con mapas SOM ofrece oportunidades para estudiar mediciones de incertidumbre e incorporar metodologías bayesianas e información de expertos. Este enfoque no ha sido ampliamente

estudiado desde una perspectiva metodológica.

Además, el concepto de SOM como mónada permite la creación de mapas de probabilidad bidimensionales de pertenencia de nodos, lo que permite su aplicación en varios entornos industriales donde un desafío clave es el impacto de agregar nuevas variables o muestras al proceso MLOps en la estructura de datos.

A continuación se presentan algunas líneas de investigación pendientes que podrían desarrollarse en trabajos futuros. Los hemos organizado según la estructura de la tesis. Tenga en cuenta que algunos de estos puntos ya están muy avanzados en el momento de escribir esta tesis.

- El estudio de modelos categóricos de SOM y sus propiedades.
  - La investigación podría centrarse en el uso de la teoría de categorías para formalizar y unificar los enfoques existentes para los SOM
  - omprensión más profunda de los fundamentos matemáticos de los SOM y, potencialmente, conducir a nuevos conocimientos y desarrollos en el campo.
  - Uso de mapas autoorganizados (SOM) como mónada para incrustar datos
- Aplicación de SOM de dos capas como representación del conocimiento y modelo inferencial en sistemas expertos en medicina (enfermedad celíaca), marketing y ciencias sociales. Se están trabajando varios proyectos relacionados, marketing digital en el ámbito bancario, psiquiatría y procesamiento del lenguaje natural.
- Evolución del algoritmo SOM, desde el punto de vista de la medida de incertidumbre.
  - Incluye el desarrollo de una implementación estocástica que facilita el uso inferencial del mapa resultante desde las perspectivas frecuentista y bayesiana.
  - Incluye justificación de la metodología de ambas perspectivas, definición de verosimilitud.
  - Estudio e implementación de diferentes formas de cálculo de distancias y representación en base a incertidumbre y probabilidad en Self-Organizing Map (SOM).
  - Estudio de los beneficios en términos de medición de la incertidumbre
  - Aplicación en la representación de estados en el aprendizaje por refuerzo.

- Aplicación del bayesian two-layer SOM como representante del conocimiento y modelo inferencial en sistemas expertos.

Conclusions and Future Work

Summary: In this section, a summary is necessary to clarify the connections between the proposals included in this report and the objectives outlined at the beginning of it, in order to fulfill its evaluation. It is not possible to carry out this task without establishing a link between the approaches and objectives with the contributions that are included in this report and that somehow support the research work presented, facing the research community.

To do this, the structure of objectives outlined in Chapter 1, Section 1.1 will be followed, trying to relate the different contributions to the addressed objectives.

Conclusions

In conclusion, the research has focused on the use of functional programming and category theory in the development of expert systems, with a specific focus on the self-organizing map (SOM) as a tool for knowledge representation. The main challenge encountered was the incomplete representation of the original probabilistic space when applying a single layer SOM. A computational experiment has been defined using mixture Gaussian to illustrate the challenges of using a single layer SOM for knowledge representation. In order to validate the preservation of topology, some image-based and graph-based definitions of distances have been included, which were used in the two-layer SOM algorithm. The results showed that using a two-layer SOM can improve the representation of the probabilistic space.

The two-layer SOM has the potential to be a functor, which could be explored in future work. This could involve investigating the use of category theory to model the relationships between different components of the SOM, and developing algorithms that take advantage of the functorial properties of the SOM. Additionally, the use of Bayesian methodology in the development

of expert systems could be explored, including the development of algorithms for measuring and representing uncertainty.

Overall, the use of functional programming and category theory in expert systems has the potential to improve the representation and analysis of complex knowledge, and further research in this area is likely to yield valuable insights and practical applications.

Relation between proposed tasks and contributions

***Objetive 1**: Perform implementations with parallelized processing on distributed data*

In the background section 3, we have introduced first, some different functional programming paradigms, such as Spark (Section 3.2.4) in order to work with Big Data systems (Section 3.3) and second that these systems have been applied in a set of projects. Spark has been specifically applied to improve the performance of the Two-layer SOMs [51] (Section 3.3.5), which present, versus other solutions, the drawback of computational complexity. This can be found in the One-layer vs Two layer (Chapter 5), and in the corresponding paper (**Contribution** [69]).

1. *Compare the performance results on different configurations and optimization systems from the theoretical point of view.*

   In the One-layer vs Two-layer (Chapter 5 Section 5.2), we dwell on the possibility of increasing the number of layers and of maps per layer, concluding that the algorthim structure and computational design allow straightforwardly for this extension.

2. *Generate graphic procedures that allow the visualization of the results.*

   Data visualisation implies, by definition, side effects, given that the use of functional programming hardly provides added value. This is why we have used R instead to produce interactive visualisation of the results of the modelling and algorithms (Section 5.4). This contributes to considering SOM as a full expert system since it provides the non-skilled user with an easily interpretable vision of the original data space and allows him/her to interact, and explore different areas, profiting from the knowledge and representation basis provided by the two-layer SOM.

***Objetivo 2**: Evolution of the algorithm to a multilayer SOM structure that would allow to identify hierarchical topological structures in the original space and reduce the effect of underrepresentation of low frequency areas. Sokolovska [58].*

In using SOM as a knowledge base, it is necessary to have a helpful visualization, and all situations envisaged in the original space should appear in the

map. Here we encounter the problem of only sometimes finding a solution in the traditional one-layer SOM. A means for representation and simulation have been found for this problem, through which to scientifically validate our proposed alternative solution. This can be found in the **Contribution** [69] and (Chapter 5, Section 5.4).

1. Design of metrics for comparison of the structure of the distribution of points in strata with respect to the original population Yin [77] and Van Hulle [28].

   A stratification of the original space has been designed, which finds a trade-off between usefulness and incorporating only a little prior information.

2. Topology maintenance evaluation metric through neighbourhood ordering and generation of graph structures.

   In the state of art analysis, it has been checked the assessment of topological conservation in SOM, is an open field for study. We also check the need to define alternative metrics to evaluate the conservation of topology from the point of view of preserving the distances associated to the topological space derived from the metric space. The relationship between the topological and metric spaces can be found in Section 3 Background measures. Some topological evaluation measures can be found in the SOm section, and our images/graph-based distance measures are presented in Chapter 5, Section 5.3.

3. Study of the effects of various forms of stratification on the result.

   A simulation framework has been built based on Gaussian mixtures, which allows for the validation of the original data generation probabilistic space, and the location of these low-frequency stratum points in this space, thus showing that the two-layer SOM performs satisfactorily in the situations evaluated. This results in a model which can be considered a functor on the probabilistic space, as a category and, therefore, as a sound knowledge base for an expert system. Chapter 5, Section 5.3.

4. Methodological and experimental justification of parallelization [74]

   It has been verified that our parallelization scheme does not distort the performance of the two-layer proposal, although the results may suffer slight changes due to the randomness in the selection of the nodes and the distribution of the strata along the computational nodes.

Publications derived from the thesis

This section provides a list of the publications that have emerged from this thesis, including their titles, authors, and the journals or conferences in which they were published. Through this section, readers can gain a deeper understanding of the impact and scope of the research conducted, and can also explore the wider context in which these findings have been disseminated and discussed.

Contribution 1: González-Pérez, B., Núñez, C., Sánchez, J., Valverde, G. (2021). : *Expert System to Model and Forecast Time Series of Epidemiological Counts with Applications to COVID-19* Mathematics. 9. 1485. 10.3390/math9131485.(JCR)

Contribution 2: López, V., Urgeles, D., Sanchez, O., Valverde, G. (2019): *Big Data in Healthcare and Social Sciences: Bip4Cast as a CAD System.* 10.4018/978-1-5225-7501-6.ch046 (JCR)

Contribution 3: López, V., Valverde., G., Anchiraico, J., Urgeles, D. (2016). *Specification of CAD Prediction System for Bipolar Disorder* ,In Proceedings of Conference on Uncertainty Modelling in Knowledge Engineering and Decision Making (FLINS 2016), 162-167. 10.1142/97898131469760028

Contribution 4: López, V., Miñana, G., Sanchez, O., González-Pérez, B., Valverde, G. (2015): *Big+Open Data: Some applications for a Smartcity.* 384-389. 10.1109/PIC.2015.7489874.

Future research lines

Most of the work presented in this dissertation is the result of the research carried out for publication in two JCR journals and other resources during the time dedicated to the Doctoral Program in IMEIO. Many of the results are definitively closed in these pages, but there are still some open lines of research.

The use of new Big Data processing methodologies and ensembling learning in combination with SOM maps offers opportunities for studying uncertainty measurements and incorporating Bayesian methodologies and expert information. This approach has not been extensively studied from a methodological perspective.

Additionally, the concept of SOM as a monad allows for the creation of two-dimensional probability maps of node belonging, enabling its application in various industrial settings where a key challenge is the impact of

adding new variables or samples to the MLOps process on data structure maintenance.

Here are some pending lines of research that could be pursued in future work. We have organized them according to the structure of the thesis. Note that some of these points are already well advanced at the time of writing this dissertation.

- The study of categorical models of SOMs and their properties.
  - Research could focus on using category theory to formalize and unify existing approaches to SOMs
  - Deeper understanding of the mathematical foundations of SOMs and potentially lead to new insights and developments in the field
  - Use of Self-Organizing Maps (SOMs) as a monad for embedding data
- Application of two-layer SOM as a knowledge representation and inferential model in expert systems in medicine (celiac disease), marketing, and social sciences. Several related projects are being worked on, digital marketing in the banking field, psychiatry and natural language processing.
- Evolution of the algorithm SOM, from a measure of uncentainty point of view.
  - Include development of stochastic implementation that facilitates the inferential use of the resulting map from both frequentist and Bayesian perspectives,
  - Include justification of the methodology of both perspectives,definition of likelihood.
  - Study and implementation of different ways of calculating distances and representation based on uncertainty and probability in Self-Organizing Map (SOM).
  - Study of the benefits in terms of uncertainty measurement
  - Application in the representation of states in reinforcement learning.
- Application of bayesian two-layer SOM as a knowledge representation and inferential model in expert systems.

Bibliography

El que lee mucho y anda mucho, ve mucho y sabe mucho.

Miguel de Cervantes Saavedra

[1] C. A. Astudillo and B. J. Oommen. Self-organizing maps whose topologies can be learned with adaptive binary search trees using conditional rotations. *Pattern Recognition*, 47(1):96–113, 2014.

[2] J. Azorin-Lopez, M. Saval-Calvo, A. Fuster-Guillo, H. Mora-Mora, and V. Villena-Martinez. Topology Preserving Self-Organizing Map of Features in Image Space for Trajectory Classification. In *Bioinspired Computation in Artificial Systems*, pages 271–280, Cham, 2015. Springer International Publishing.

[3] A. Barr, E. A. Feigenbaum, and P. R. Cohen. *The handbook of artificial intelligence*, volume 1. William Kaufmann, 1981.

[4] J. M. Bishop. Artificial intelligence is stupid and causal reasoning won't fix it. *CoRR*, abs/2008.07371, 2020.

[5] B. G. Buchanan and E. H. Shortliffe. Rule-based expert systems: the mycin experiments of the stanford heuristic programming project. 1984.

[6] R. Calvo. Self-organizing maps and reinforcement learning for intelligent agents in dynamic environments. *International Journal of Advanced Intelligence Paradigms*, 1(4):373–382, 2009.

[7] E. Castillo and E. Alvarez. *Expert systems: uncertainty and learning.* WIT Press, 1991.

[8] J. Chen, O. Bukhres, and A. K. Elmagarmid. Ipl: A multidatabase transaction specification language. In *[1993] Proceedings. The 13th International Conference on Distributed Computing Systems*, pages 439–448. IEEE, 1993.

[9] J. Chen and L. Chen. Fast self-organizing maps. *Neural Computing and Applications*, 15(2):121–128, 2006.

[10] J. Chen and L. Chen. Incremental self-organizing maps. *Neural Computing and Applications*, 16(6):661–669, 2007.

[11] S. Chen. Estimating customer lifetime value using machine learning techniques. In *Data mining*, volume 17, 2018.

[12] X.-W. Chen and X. Lin. Big data deep learning: challenges and perspectives. *IEEE access*, 2:514–525, 2014.

[13] P. Chiusano and R. Bjarnason. *Functional programming in Scala*. Simon and Schuster, 2014.

[14] G. Cottrell, A. Girard, J. Herault, J. Matas, W. Rif, and M. Verleysen. Theoretica: A general theory for adaptive learning. *Neural Networks*, 11(4-5):837–863, 1998.

[15] D. Dahl. Modal clustering in a class of product partion models. *Bayesian Statistics*, 4:243–264, 2009.

[16] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2004.

[17] R. Duda, J. Gaschnig, and P. Hart. Model design in the prospector consultant system for mineral exploration. In *Readings in Artificial Intelligence*, pages 334–348. Elsevier, 1981.

[18] D. Dumitriu and M. A.-M. Popescu. Artificial intelligence solutions for digital marketing. *Procedia Manufacturing*, 46:630–636, 2020. 13th International Conference Interdisciplinarity in Engineering, INTER-ENG 2019, 3 4 October 2019, Targu Mures, Romania.

[19] R. Durbin, R. Szeliski, and A. Yuille. An Analysis of the Elastic Net Approach to the Traveling Salesman Problem. *Neural Computation*, 1(3):348–358, 1989.

[20] J. Durkin and J. Durkin. *Expert systems: design and development*. Prentice Hall PTR, 1998.

[21] S. Gabrielsson, S. Gabrielsson, S. F. Mican, D. S. NÃ¤th, E. Ab, and E. A. Jansson. The use of self-organizing maps in recommender systems a survey of the recommender systems field and a presentation of a state of the art highly interactive visual movie recommender system.

[22] B. González-Pérez, C. Núñez, J. L. Sánchez, G. Valverde, and J. M. Velasco. Expert system to model and forecast time series of epidemiological counts with applications to covid-19. *Mathematics*, 9(13):1485, 2021.

[23] M. Grabisch. k-order additive discrete fuzzy measures and their representation. *Fuzzy Sets Systems*, 92(2):167–189, 1997.

[24] M. Gromov, M. Katz, P. Pansu, and S. Semmes. *Metric structures for Riemannian and non-Riemannian spaces*, volume 152. Springer, 1999.

[25] X. Guo, H. Wang, and D. H. Glass. Bayesian self-organizing map for data classification and clustering. *International Journal of Wavelets, Multiresolution and Information Processing*, 11(05):1350037, 2013.

[26] S. Gupta, D. Hanssens, B. Hardie, W. Kahn, V. Kumar, N. Lin, N. Ravishanker, and S. Sriram. Modeling customer lifetime value. *Journal of service research*, 9(2):139–155, 2006.

[27] J. Himberg. A som based cluster visualization and its application for false coloring. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, volume 3, pages 587–592 vol.3, 2000.

[28] M. V. Hulle. Self-organizing Maps, 2012.

[29] L. N. Kanal and J. F. Lemmer. *Uncertainty in artificial intelligence*. Elsevier, 2014.

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

[31] T. Kohonen. *Self-organizing maps*. Springer, 3 edition, 1997.

[32] T. Kohonen and T. Honkela. Kohonen network. *Scholarpedia*, 2(1):1568, 2007.

[33] M. Kratochvíl, A. Koladiya, J. Balounova, V. Novosadova, R. Sedlacek, K. Fišer, J. Vondrášek, and K. Drbal. Som-based embedding improves efficiency of high-dimensional cytometry data analysis. *bioRxiv*, 2019.

[34] J. B. Kruskal. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. In *Proceedings of the American Mathematical Society, 7*, 1956.

[35] J. W. Lau and P. J. Green. Bayesian Model-Based Clustering Procedures. *Journal of Computational and Graphical Statistics*, 16(3):526–558, 2007.

[36] V. López, D. Urgelés, Ó. Sánchez, and G. Valverde. Big data in healthcare and social sciences: Bip4cast as a cad system. *International Journal of Information Systems and Social Change (IJISSC)*, 8(3):1–16, 2017.

[37] S. P. Luttrell. A Bayesian Analysis of Self-Organizing Maps. *Neural Computation*, 6(5):767–794, 1994.

[38] C. Malsburg. Self-organization of orientation sensitive cells in the striate cortex. *Kybernetik*, 14(2):85–100, 1973.

[39] T. Murofushi and S. Soneda. Techniques for reading fuzzy measures (III): interaction index. In *Proceedings of* 9^{th} *Fuzzy System Symposium*, pages 693–696, Sapporo, Japan, 1993. 9^{th} Fuzzy System Symposium.

[40] F. Murtagh and P. Contreras. Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1):86–97, 2012.

[41] P. Nicolas. *Scala for Machine Learning: Data processing, ML algorithms, smart analytics, and more*. Packt Publishing, 2017.

[42] S. Pal and P. Sahoo. A self-organizing map based expert system for diagnosis of heart diseases. *International Journal of Medical Informatics*, 75(12):829–838, 2006.

[43] S. Pal and P. Sahoo. A self-organizing map based expert system for remote sensing image interpretation. *International Journal of Applied Earth Observation and Geoinformation*, 8(3):200–213, 2006.

[44] S. Pal and P. Sahoo. A self-organizing map based reinforcement learning approach for robot navigation. *International Journal of Advanced Robotic Systems*, 7(1):1–10, 2010.

[45] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan kaufmann, 1988.

[46] J. Pearl and D. Mackenzie. *The Book of Why: The New Science of Cause and Effect*. Basic Books, Inc., USA, 1st edition, 2018.

[47] M. Petrovic and M. Petrovic. A self-organizing map-based expert system for classification of remotely sensed data. *International Journal of Remote Sensing*, 24(12):2523–2537, 2003.

[48] J. Qiu, Q. Wu, G. Ding, Y. Xu, and S. Feng. A survey of machine learning for big data processing. *EURASIP Journal on Advances in Signal Processing*, 2016(1):1–16, 2016.

[49] A. Rajaraman and J. D. Ullman. *Mining of massive datasets*. Cambridge University Press, 2011.

[50] A. Sanjurjo-De-No, B. Arenas-RamÃrez, J. Mira, and F. Aparicio-Izquierdo. Driver pattern identification in road crashes in spain. *IEEE Access*, 8:182014–182025, 2020.

[51] T. Sarazin, H. Azzag, and M. Lebbah. Som clustering using spark-mapreduce. In *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*, pages 1727–1734. IEEE, 2014.

[52] P. Sarlin and T. Eklund. Fuzzy Clustering of the Self-Organizing Map: Some Applications on Financial Time Series. In J. Laaksonen and T. Honkela, editors, *Advances in Self-Organizing Maps*, pages 40–50, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[53] D. Schoech, H. Jennings, L. L. Schkade, and C. Hooper-Russell. Expert systems: Artificial intelligence for professional decisions. *Computers in Human Services*, 1(1):81–115, 1985.

[54] A. Seret, T. Verbraken, S. Versailles, and B. Baesens. A new som-based method for profile generation: Theory and an application in direct marketing. *European Journal of Operational Research*, 220(1):199–209, 2012.

[55] G. Shafer. *A mathematical theory of evidence*, volume 42. Princeton university press, 1976.

[56] H. Siemon and A. Ultsch. Adaptive self-organizing maps. *Neural Networks*, 12(1):57–68, 1999.

[57] K. Slavakis, G. B. Giannakis, and G. Mateos. Modeling and optimization for big data analytics:(statistical) learning tools for our era of data deluge. *IEEE Signal Processing Magazine*, 31(5):18–31, 2014.

[58] N. Sokolovska, N. T. Hai, K. Clément, and J.-D. Zucker. Deep self-organising maps for efficient heterogeneous biomedical signatures extraction. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 5079–5086, 2016.

[59] G. Steele. *Common LISP: the language*. Elsevier, 1990.

[60] L. Stevens. *Artificial Intelligence, the Search for the Perfect Machine*. Prentice Hall, 1985.

[61] S. Thompson. *Type theory and functional programming*. 01 1991.

[62] J. Triesch, F. W.ºrg.ºtter, and J. Austin. A self-organizing map-based inference system for diagnosing multiple sclerosis. *Neural Networks*, 15(4):523–534, 2002.

[63] J. Triesch, F. W.ºrg.ºtter, and J. Austin. Reinforcement learning with self-organizing maps. *Neural computation*, 17(8):1691–1706, 2005.

[64] A. Ultsch and A. Rauber. Hierarchical self-organizing maps. *Neural Computing and Applications*, 11(1):1–14, 2002.

[65] A. Ultsch and H. Ritter. Self-organizing maps. In H. K. H. Lau, M. K. H. Leung, and J. Y. S. Lui, editors, *Encyclopedia of biostatistics*, volume 6, pages 3583–3590. John Wiley Sons, 2003.

[66] A. Ultsch and H. Siemon. Growing self-organizing maps. *Neural Networks*, 9(8):1453–1461, 1996.

[67] E. A. Uriarte and F. Dìaz-Martìn. Topology Preservation in SOM. *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 2(9):3192–3195, 2008.

[68] A. Utsugi. Hyperparameter Selection for Self-Organizing Maps. *Neural Computation*, 9(3):623–635, 03 1997.

[69] G. A. Valverde Castilla, J. M. Mira McWilliams, and B. González-Pérez. One-layer vs. two-layer som in the context of outlier identification: A simulation study. *Applied Sciences*, 11(14):6241, 2021.

[70] M. van der Meer, A. Engelbrecht, and P. Botha. Self-organizing maps and reinforcement learning for continuous control. *Journal of Artificial Intelligence Research*, 29:599–620, 2007.

[71] S. Verma, R. Sharma, S. Deb, and D. Maitra. Artificial intelligence in marketing: Systematic review and future research direction. *International Journal of Information Management Data Insights*, 1(1):100002, 2021.

[72] J. Vesanto and E. Alhoniemi. Clustering of the self-organizing map. *IEEE Transactions on Neural Networks*, 11(3):586–600, 2000.

[73] L. Wang and D. B. Dunson. Fast Bayesian Inference in Dirichlet Process Mixture Models. *Journal of Computational and Graphical Statistics*, 20(1):196–216, 2011.

[74] C. Weichel. Adapting self-organizing maps to the mapreduce programming paradigm. In *STeP*, pages 119–131, 2010.

[75] S. M. Weiss and C. A. Kulikowski. A practical guide to designing expert systems. In *A practical guide to designing expert systems.* Chapman & Hall, 1983.

[76] D. J. Willshaw and C. V. D. Malsburg. How Patterned Neural Connections Can Be Set Up by Self-Organization. *Proceedings of the Royal Society of London. Series B, Biological Sciences*, 194(1117):431–445, 1976.

[77] H. Yin. The Self-Organizing Maps: Background, Theories, Extensions and Applications. In J. Fulcher and L. C. Jain, editors, *Computational*

Intelligence: A Compendium, pages 715–762, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[78] H. Yin and N. Allinson. Bayesian self-organising map for gaussian mixtures. *IEE Proceedings: Vision, Image and Signal Processing*, 148(4):234–240, 2001.

[79] A. L. Yuille. Generalized Deformable Models, Statistical Physics, and Matching Problems. *Neural Computation*, 2(1):1–24, 1990.

[80] L. A. Zadeh. The role of fuzzy logic in the management of uncertainty in expert systems. *Fuzzy sets and systems*, 11(1-3):199–227, 1983.