Université de Paris 13

Laboratoire d'Informatique de Paris-Nord (LIPN)

Apprentissage Artificiel et Applications (A3)

Thèse

présentée par

Tugdual Sarazin

pour obtenir le grade de

Docteur d'université

Spécialité : Informatique

Apprentissage massivement distribué dans un environnement Big Data.

---

soutenue publiquement le 26 juin 2018

devant le jury composé de

Directeur			
M.	Mustapha Lebbah (MCF-HDR)	-	LIPN, Université Paris 13
Co-encadrement			
Mme	Hanane Azzag (MCF-HDR)	-	LIPN, Université Paris 13
Rapporteurs			
M.	Gilles Venturini (PR)	-	Université de Tours
M.	Cyril de Runz (MCF-HDR)	-	Université de Reims
Examinateurs			
Mme	Salima Benbernou (PR)	-	Université Paris Descartes
M.	Christophe Cerin (PR)	-	LIPN, Université Paris 13

Abstract

In recent years, the amount of data analysed by companies and research laboratories increased strongly, opening the era of BigData. However, these raw data are frequently non-categorized and uneasy to use. This thesis aims to improve and ease the pre-treatment and comprehension of these big amount of data by using unsupervised machine learning algorithms.

The first part of this thesis is dedicated to a state-of-the-art of clustering and bi-clustering algorithms and to an introduction to bigdata technologies. The first part introduces the conception of clustering Self-Organizing Map algorithm [Kohonen, 2001] in big data environment. Our algorithm (SOM-MR) provides the same advantages as the original algorithm, namely the creation of data visualisation map based on data clusters. Moreover, it uses the Spark platform that makes it able to treat a big amount of data in a short time. Thanks to the popularity of this platform, it easily fits in many data mining environments. This is what we demonstrated it in our project "Square Predict" carried out in partnership with Axa insurance. The aim of this project was to provide a real-time data analysing platform in order to estimate the severity of natural disasters or improve residential risks knowledge. Throughout this project, we proved the efficiency of our algorithm through its capacity to analyse and create visualisation out of a big volume of data coming from social networks and open data.

The second part of this work is dedicated to a new bi-clustering algorithm. BiClustering consists in making a cluster of observations and variables at the same time. In this contribution we put forward a new approach of bi-clustering based on the self-organizing maps algorithm that can scale on big amounts of data (BiTM-MR). To reach this goal, this algorithm is also based on a the Spark platform. It brings out more information than the SOM-MR algorithm because besides producing observation groups, it also associates variables to these groups, thus creating bi-clusters of variables and observations.

Keywords: unsupervised machine learning, clustering, bi-clustering, self-organizing-map, big data, map reduce, Spark, distributed machine learning.

Résumé

Lors de ces dernières années les volumes de données analysées par les entreprises et les laboratoires de recherches ont fortement augmentés ouvrant ainsi l'ère du BigData. Cependant ces données brutes sont fréquemment non catégorisées et difficilement exploitables. Cette thèse vise à améliorer et faciliter le pré-traitement et la compréhension de grands volumes de données en fournissant des algorithmes d'apprentissage non supervisés.

La première partie de cette thèse est consacré à un état de l'art des algorithmes de partitionnement et bi-partitionnement ainsi qu'une présentation des technologies du BigData. La première contribution de cette thèse est dédiée à la conception de l'algorithme de clustering Self-Organizing Map ou carte auto-organisatrice [Kohonen, 2001] dans un environnement Big data. Notre algorithme (SOM-MR) fournit les mêmes avantages que l'algorithme de base, à savoir la création de partition de données et leur visualisation sous la forme de carte. De plus il utilise la plateforme Spark, ce qui lui permet à la fois de traiter de grands volumes de données en peu de temps. De part la popularité de cette plateforme il s'intègre facilement dans dans de nombreux environnements de traitement de données. C'est ce que nous avons démontré dans notre projet "Square Predict" réalisé en partenariat avec l'assurance Axa. Ce projet avait pour objectif de fournir une plateforme d'analyse de données en temps réel afin d'évaluer la sévérité d'une catastrophe naturelle ou d'améliorer la connaissance des risques résidentiels. Durant ce projet nous avons démontré l'efficacité de notre algorithme pour analyser et fournir des visualisation à partir de grands volumes de données provenant des réseaux sociaux et d'Open data.

La deuxième contribution de cette thèse est consacrée à un nouvel algorithme de BiClustering. Le BiClustering consiste à réaliser un clustering simultanément sur les observations et les variables. Dans cette contribution nous proposons une nouvelle approche de biclustering basé sur l'algorithme self-organizing maps capable de passer à l'échelle sur de grands volumes de données (BiTM-MR). Pour ce faire il est également basé sur la plateforme des technologies Big data. Mais il apporte davantage d'informations que notre algorithme SOM-MR car en plus de produire des groupes d'observations il associe des variables à ces groupes, formant ainsi des bi-groupes d'observations et variables.

Mots clés: apprentissage non supervisé, clustering, bi-clustering, Self-organizing map, big data, map reduce, Spark, apprentissage distribué.

Acknowledgements

The golem appeared in Jewish mythology where it is an magicaly animated humanoid made of clay. It is a very old myth but still present in our culture, particularly in literature (the monster of Frankestein), on television (X-Files season 4) and in many video games (Minecraft, Diablo, ...). Machine learning is probably the science that best represents the Golem symbol, it arouses both fear and fascination that man can create a machine without conscience but still able to mimic human intelligence.

I've always had a lot of fascination with these intelligent machines, however it took me a long time to understand the connection between creating artificial intelligence and what I had learned at school. But I still remember the wonder that I had during each of the courses where I understood how to make a program able to "learn", particularly the course of Nicolas Kamennoff on genetic algorithms as well as the course of Mustapha Lebbah on SOM algorithms. That why I wanted to make this thesis to learn more on this topic. Of course a thesis is not done alone, I thank very warmly my thesis directors Hanene Azzag and Mustapha Lebbah for all the motivation they gave me (and I'm not someone easy to motivate;)) but also for all the exchanges we had, I learned a lot from them and it is the most important thing of these years of thesis for me. I also thank very much Mohammed Ghesmoune without whom this thesis would never have been finished. Thank you very much too to SmokeWatchers for funding this thesis and the LIPN for supervising it. Many thanks also to all the people who re-read my publications, although they did not understand much to them : Claire Migné, my grandmother and Selena Salkoff. Finally, I thank all my family for telling me to do this thesis "because it's always good to have diplomas ".

Contents

List of Figures

List of Tables

List of publications

International conferences with reading committee

- Tugdual Sarazin, Mustapha Lebbah, and Hanane Azzag. Biclustering using spark- mapreduce. In 2014 IEEE International Conference on Big Data, Big Data 2014, Washington, DC, USA, October 27-30, 2014, pages 58?60, 2014.
- Tugdual Sarazin, Amine Chaibi, Mustapha Lebbah and Hanane Azzag. Feature group weighting and topological biclustering. International Conference on Neural Information Processing (ICONIP) 2014. Kuching, Malaisie, 4 novembre 2014.
- Tugdual Sarazin, Hanane Azzag, and Mustapha Lebbah. 2014. SOM Clustering Using Spark-MapReduce. In Proceedings of the 2014 IEEE International Parallel & Distributed Processing Symposium Workshops (IPDPSW '14). IEEE Computer Society, Washington, DC, USA, 1727-1734. DOI=10.1109/IPDPSW.

French speaking conferences with reading committee

- Tugdual Sarazin, Hanane Azzag, Mustapha Lebbah. Modèle de Biclustering dans un paradigme "Mapreduce". In EGC 2015, vol. RNTI-E-28, pp.467-468

To . . . TODO

Chapter 1

Introduction

1.1 Context of the thesis

The present work proposes to develop data mining tools based on learning techniques for mining Big Data. In a nutshell, this thesis pertains to the fields of Machine Learning (ML) and Big Data. Machine Learning methods traditionally fall into three categories: *supervised*, *semi-supervised*, and *unsupervised* methods [Han et al., 2011]. Clustering is considered as the most important unsupervised learning problem. It is a main task of data mining and the common technique that has been used in many fields, including machine learning, data mining, pattern recognition, web mining, textual document collection, image segmentation, biology, etc. [Everitt et al., 2009].

However, applying data mining techniques, and specifically clustering algorithms, to large datasets (Big Data) raise more challenges and difficulties. Big Data has become recently a new ubiquitous term. Big Data is transforming science, engineering, medicine, healthcare, finance, business, and ultimately our society itself. Currently, the Big Data domain can be characterized by the 5 V's: Volume, Velocity, Variety, Value and Veracity (see chapter 4 for more details).

Given the interest of mining Big Data, organisations are increasingly relying on Big Data to provide the opportunities to discover correlations and patterns in data that would have previously remained hidden, and to subsequently use this new information to increase the quality of their business activities. Learning from Big Data has become a significant challenge and requires development of new types of algorithms. Most machine learning algorithms can not easily scale up to Big

Data. MapReduce is a simplified programming model for processing large datasets in a distributed and parallel manner.

1.2 Our contributions

As already mentioned, the present work is concerned with the modelling of large-scale data within a Big Data framework, using Machine Learning methods, specifically the Self-Organizing Maps approach, and Big Data concepts and technologies.

Chapter 3 surveys clustering and scalable clustering methods implemented with MapReduce. Chapter 4 presents a thorough survey of the state-of-the-art for a range of bi-clustering algorithms. Chapter 2 is devoted to introducing the Big Data ecosystem and the fundamentals for data science.

In the subsequent chapters are our main contributions, summarized as follows:

1. In chapter 5 we present a scalable Self-Organizing Maps method, called **SOM-MR** which consists of a novel re-formalization of the dynamic clusters "nuées dynamiques". The proposed SOM-MR algorithm is implemented with the Spark framework which represents a new way of writing using the MapReduce paradigm. The major research challenge addressed is how to minimize the input and output of primitives (map and reduce) for topological clustering algorithm. So, we show that we can save computation time by changing the (key, value) parameters.

2. After that, in chapter 6, we presented our second contribution consisting of proposing a model for bi-clustering using MapReduce. The proposed model, called **BiTM**, is a distributed algorithm for scalable bi-clustering based on topological maps. We defined a new cost function and so a new formalization of topological bi-clustering. After that, we proposed a model for scalability. This model consists of decomposing the db-clustering problem into the elementary functions, Map and Reduce. The SOM-MR and the BiTM implementations are assured in the Spark platform.

3. In chapter 7, we present an application of our SOM-MR algorithm on the insurance Big Data provided by AXA. The utility of the SOM-MR scalable approach is demonstrated and validated on the insurance Big Data as an

example of unsupervised learning. Afterwards, a predictive and analysis system is proposed by combining the clustering result with decision trees.

The different assessments carried out in this thesis (performance measurements and visualizations) obtained promising results.

The thesis manuscript is organized as follows. Chapter 3 reviews and discusses the state-of-the-art related to both clustering and scalable clustering methods implemented with MapReduce. Chapter 4 presents a thorough survey of the state-of-the-art for a range of bi-clustering algorithms. Chapter 2 gives an introduction to the Big Data ecosystem and discusses the fundamentals that a data scientist needs in order to extract knowledge or insights from large data in various forms. Chapter 5 presents our SOM-MR algorithm concerned with scaling-up the SOM approach to deal with large datasets; experimental validation on benchmark datasets from the clustering literature is reported and discussed. Chapter 6 introduces the BiTM algorithm designed for large-scale bi-clustering. Chapter 7 finally describes the validation results of SOM-MR on the insurance Big Data. Some conclusions and perspectives for further research are presented in chapter 8.

Chapter 2

Fundamentals of Big Data

This chapter gives an introduction to the Big Data ecosystem. Indeed, we will review and discuss the fundamentals that a data scientist needs in order to extract knowledge or insights from large data in various forms, with a focus on the data stream use case.

2.1 Big Data

To our knowledge, the term "Big Data" appeared for first time in 1998 in a Silicon Graphics (SGI) slide deck by John Mashey with the title of "Big Data and the Next Wave of InfraStress" [Mashey, 1998]. It is a term used to identify the datasets that due to their large size and complexity, we can not manage them with our current methodologies or data mining software tools [Fan and Bifet, 2013]. Despite that the "Big Data" has become a new buzz-word, there is no consistent definition for Big Data, or any detailed analysis of this new emerging technology. Most discussions until now have been going in the blogosphere where active contributors have generally converged on the most important features and incentives of the Big Data [Demchenko et al., 2013].

The work presented in [Laney, 2001] was the first one to talk about 3 Vs in Big Data management, i.e., Volume (great volume), Velocity (rapid generation), Variety (various modalities), to which were added Value (huge value but very low density) [Gantz and Reinsel, 2011] and Veracity (consistency and trustworthiness) [Demchenko et al., 2013] more recently proposed. Figure 2.1 resumes the 5 Vs of Big Data [Demchenko et al., 2013]:

- Volume: there is more data than ever before, their size continues to increase, but not the percentage of data that our tools can process
- Velocity: data are arriving continuously as streams of data, and we are interested in obtaining useful information from it in real time
- Variety: data type diversity in a given stream (text, video, audio, static image, etc.); also differences in data processability (structured, semi-structured, unstructured data)
- Value: business value that gives an organization a competitive advantage, due to the ability to make decisions based in answering questions that were previously considered beyond reach
- Veracity: it includes two aspects: data consistency (or certainty) as defined by their statistical reliability; and data trustworthiness that includes data origin, collection and processing methods (trusted infrastructure and facilities).

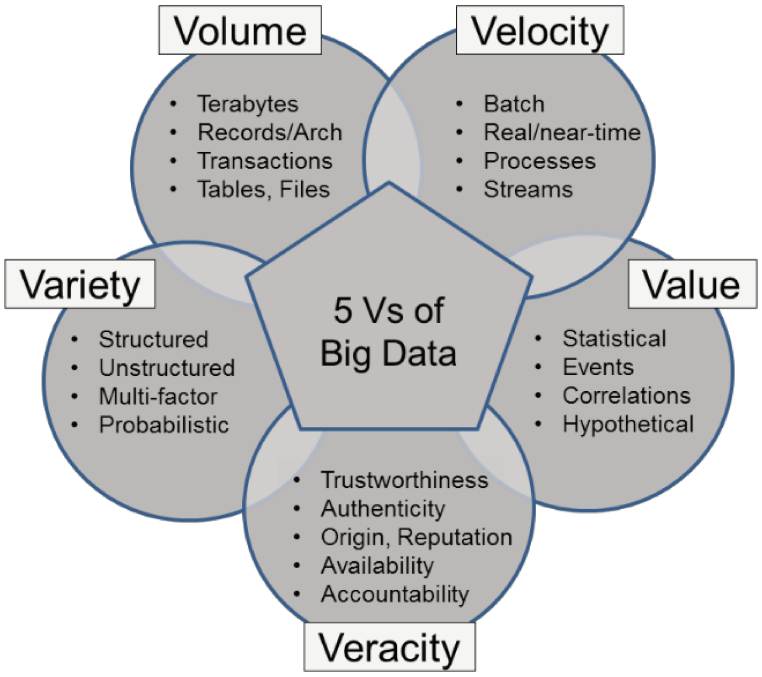


FIGURE 2.1: 5 Vs of Big Data [Demchenko et al., 2013]

We offer an alternative definition: Big Data is also a multidisciplinary area for exchange and collaboration. This definition emphasis the processes involved with Big Data rather than attempting to define intrinsic characteristics like the 5 Vs.

2.2 Distributed data storage systems

2.2.1 Google File System (GFS)

GFS [Ghemawat et al., 2003] uses a simple design with a single *master* server for hosting the entire metadata (the namespace, access control information, the mapping from files to chunks, and the current locations of chunks) and where the data is split into chunks and stored in *chunk-servers*. Files are divided into fixed-size *chunks*. Chunkservers store chunks on local disks and read or write chunk data specified by a chunk handle and byte range. For reliability, each chunk is replicated on multiple chunkservers. However the GFS master is now made fault tolerant using the Chubby [Burrows, 2006] abstraction.

2.2.2 Hadoop Distributed File System (HDFS)

HDFS [Borthakur, 2007] is a distributed file system designed to run on top of the local file systems of the cluster nodes and store extremely large files. HDFS consists of two types of nodes, namely, a namenode called "master" and several datanodes called "slaves". HDFS can also include secondary namenodes. The namenode manages the hierarchy of file systems and director namespace (i.e., metadata). File systems are presented in a form of a namenode that registers attributes, such as access time, modification, permission, and disk space quotas. The file content is split into large blocks, and each block of the file is independently replicated across datanodes for redundancy and to periodically send a report of all existing blocks to the namenode.

HDFS is highly fault tolerant and can scale up from a single server to thousands of machines, each offering local computation and storage. For example, according to Figure 2.2, the record #2 is replicated on nodes A, B, and D. When a process needs this record, it can retrieve it from the node which optimises the response time.

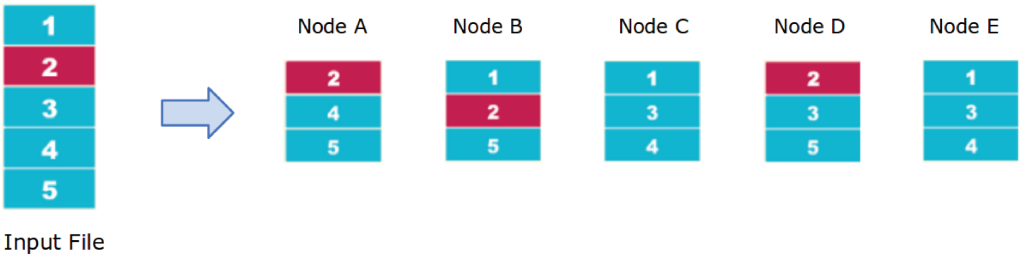


FIGURE 2.2: HDFS Data Distribution

2.3 MapReduce: Basic Concept

MapReduce [Dean and Ghemawat, 2008a] is a simplified programming model for processing large numbers of datasets pioneered by Google for data-intensive applications. The MapReduce model was developed based on GFS [Ghemawat et al., 2003] and is adopted through an open-source Hadoop implementation, which was popularized by Yahoo. MapReduce enables programmers who have no experience with distributed systems to write applications that process huge datasets in a large cluster of commodity machines; it manages data partitioning, task scheduling, and nodes failure.
Indeed, MapReduce allows an unexperienced programmer to develop parallel programs and create a program capable of using computers in a cloud. The MapReduce programming model can be explained as follows. The computation takes a set of input key/value pairs, and produces a set of output key/value pairs. The user of the MapReduce library expresses the computation as two functions: *Map* and *Reduce*.

- *Map*: written by the user, takes an input pair and produces a set of intermediate key/value pairs. For example, given the word count example that it is displayed in Figure 2.3, each mapper takes a line as input and breaks it into words. It then emits a key/value pair of <"word", 1> (<"D", 1>, <"B", 1>, etc.).

- *Shuffle*: is an intermediate step which is done automatically by the system. It starts after finishing the Map step and before the Reduce step. It takes the intermediate data generated by each Map task, sorts this data with intermediate data from other nodes, divides this data into regions to be processed by the reduce tasks.

- *Reduce*: also written by the user, accepts an intermediate key and a set of values for that key. It merges together these values to form a possibly smaller set of values. According to example of Figure 2.3, each reducer sums the counts for each word and emits a single key/value with the word and sum. The final result can be collected into one file that contains each word associated with its frequency (<"A", 2>, <"B", 2>, <"C", 3>, <"D", 2>).

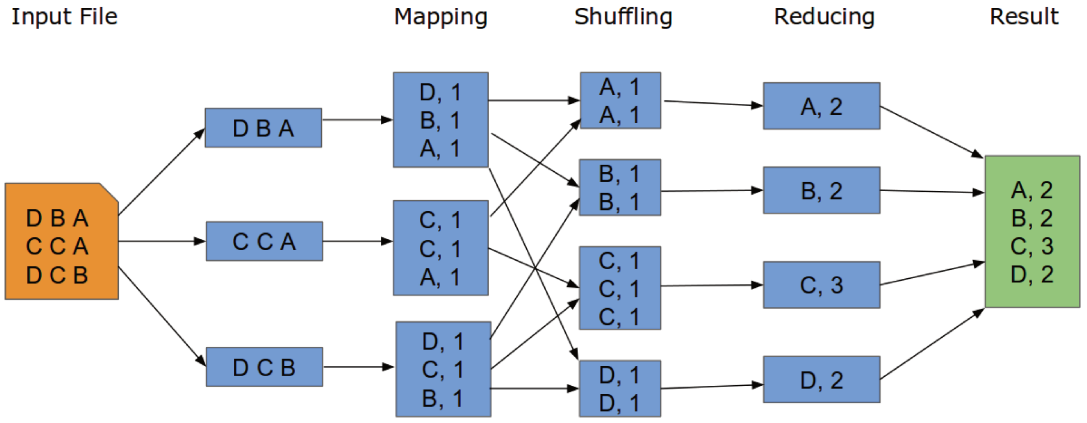


FIGURE 2.3: MapReduce processes for counting the number of occurrences for each word in a document

2.4 Distributed platforms

2.4.1 Hadoop

Apache Hadoop[1] is one of the most well-established software platforms that allow for the distributed processing of large data sets across clusters of computers using simple programming models. It implements the MapReduce paradigm. It is designed to scale up from single servers to thousands of machines, with each offering local computation and storage. Typiquely, the Hadoop framework uses the Hadoop Distributed File System (HDFS) to save large datasets in a distributed manner. Users code their queries and programs using Java. Therefore, the I/O performance of a Hadoop MapReduce job strongly depends on HDFS. Indeed, the HDFS has an non-negligible access time; reads and writes are sufficiently long (because of the use of hard disc to save intermediate data as shown in Figure 2.4)

[1]See http://hadoop.apache.org/

that maching learning processes which generally make many iterations on data are inefficient.

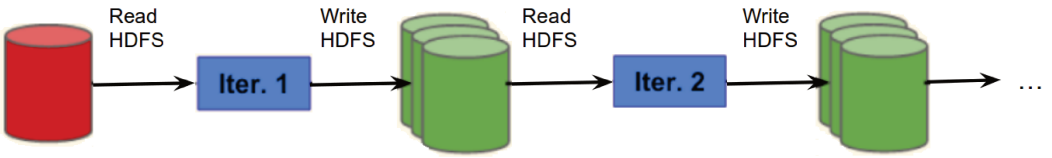


FIGURE 2.4: HDFS reads and writes in iterative machine learning algorithms

In terms of a Hadoop cluster, there are two kinds of nodes in the Hadoop infrastructure: master nodes and worker nodes. The master node takes the input, divides it into smaller sub-problems, and distributes them to worker nodes in the Map step. Afterwards, the master node collects the answers to all the sub-problems and combines them in some way to form the output in the Reduce step.

2.4.2 Spark

Spark is a cluster computing system originally developed by UC Berkeley AMPLab [Zaharia et al., 2012]. Now it is an umbrellaed project of the Apache Foundation[2]. The main abstraction in Spark is that of a resilient distributed dataset (RDD), which represents a read-only collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost. Users can explicitly cache an RDD in memory across machines and reuse it in multiple MapReduce-like parallel operations. RDDs achieve fault tolerance through a notion of lineage: if a partition of an RDD is lost, the RDD has enough information about how it was derived from other RDDs to be able to rebuild just that partition. Indeed, the elements of an RDD need not exist in physical storage; instead, a handle to an RDD contains enough information to compute the RDD from data in reliable storage. This means that RDDs can always be reconstructed if nodes fail. Although RDDs are not a general shared memory abstraction, they represent a sweet-spot between expressivity on the one hand and scalability and reliability on the other hand, and they are well-suited to a variety of applications [Zaharia et al., 2010a].

Spark is intended to be easy to use where users can write their applications quickly in Java, Scala, Python, or R. Figure 2.5 shows that starting from the second iteration, the intermediate data are saved in-memory (RAM) where Spark

[2]See `http://spark.apache.org/`

can retrieve them by accessing the RAM rather than the hard disc, which makes the execution much faster.

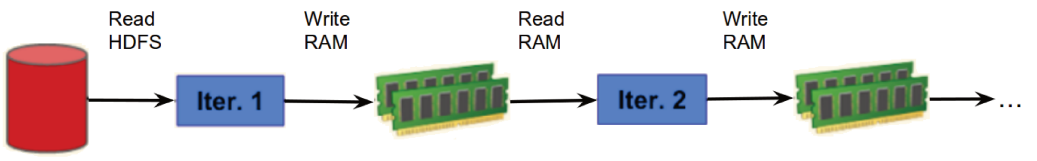


FIGURE 2.5: Iterative machine learning algorithms in Spark

In terms of comparison between Spark and Hadoop, Spark runs programs up to 100× faster than Hadoop MapReduce in memory, or 10× faster on disk, in iterative machine learnig algorithms. As example, Figure 2.6 compares Hadoop and Spark in terms of running time for the k-means and logistic regression algorithms. Another point of comparison between the two frameworks is that Spark offers more choice and flexibility to the programmers because it allows them to write their code in Scala, Java, Python, or R while Hadoop offers only Java.
Given these arguments, it appears that the choice of Spark over Hadoop is obvious.

FIGURE 2.6: Running time of k-means and logistic regression in Hadoop and Spark [Zaharia et al., 2012]

2.5 Conclusion

Big Data are becoming a new technology focus both in data science and in industry. The domain of Big Data gathers all the techniques and algorithms recently proposed where conventional methods fail when applied to data, as well as accompanying platforms and technologies. Indeed, it has been necessary to reconsider platforms, programming languages, and paradigms usually used for statistical learning. Handling large volumes of data and developing scalable models using computing power (clusters and clouds) was not possible with the usually existing platforms. These new platforms are usually deployed after extensive development on traditional platforms e.g. Matlab, R, etc., as they remain attractive for rapid

scientific prototyping.

In the next chapter, we will detail our first contribution, concerning the SOM-MR algorithm which is a scalable clustering method based on the SOM approach and implemented with MapReduce.

Chapter 3

Clustering and Scalable Algorithms

The first part of this chapter reviews and discusses the state of the art related to clustering methods. In the second part, we detail some scalable clustering methods implemented with MapReduce, allowing the reader to have a clear idea on how to scale any data clustering algorithm using the MapReduce paradigm.

There are too many clustering algorithms to cover comprehensively here so we will focus on the algorithms which we have utilised ourselves or those which appear to be most relevant to our work.

3.1 Introduction

Clustering is a key data mining task. This is the problem of partitioning a set of observations into clusters such that observations assigned in the same cluster are similar (or close) and the inter-cluster observations are dissimilar (or distant). The other objective of clustering is to quantify the data by replacing a group of observations (cluster) with one representative observation (prototype).

This chapter reviews and discusses the state of the art related to clustering methods. Even if we do not propose an exhaustive survey, we argue that we present in detail the most well-known data clustering algorithms. Furthermore, we present an understandable section on how to scale traditional clustering algorithms using the MapReduce paradigm.

We assume that a set of n data-points $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ are given, where

$\mathbf{x}_i = (x_i^1, x_i^2, ..., x_i^d)$ is a vector in the $\mathbb{R}^d$ space. We denote by $\mathcal{C}$ the set of clusters produced by the clustering task. Each cluster has a prototype variable, denoted by $\mathbf{w}_c = (w_c^1, w_c^2, ..., w_c^d)$, which represents the position of the cluster in $\mathbb{R}^d$.

3.2 Data clustering algorithms

3.2.1 *k*-means

The most common example of clustering algorithms is k-means [Jain and Dubes, 1988]. Clusters are represented by a mean vector called the weighted vector or prototype $\mathbf{w}_j$, where $j = 1, ..., k$, which may not necessarily be a physical point in the data space. Thus we can re-define the clustering problem as an optimization problem: find the cluster centers such that the intra-class variance is minimized, i.e., the sum of squared distances from each object within a cluster to its corresponding prototype. k-means finds k classes from a set of n observations, by minimizing the following cost function:

$$\mathcal{R}_{k-means}(\phi, \mathcal{W}) = \sum_{i=1}^{n} \sum_{j=1}^{k} \|\mathbf{x}_i - \mathbf{w}_j\|^2 \tag{3.1}$$

The method used for the minimization of the function $\mathcal{R}_{k-means}(\phi, \mathcal{W})$ is an iterative method whose basic iteration has two phases:

- **Assignment step**: it is, in this phase, to minimize the function $\mathcal{R}_{k-means}(\phi, \mathcal{W})$ with respect to the assignment function ϕ assuming that the prototype vectors $\mathcal{W}$ are constant; The minimization is achieved by assigning each observation $\mathbf{x}_i$ to the referent $\mathbf{w}_c$ using the assignment function ϕ:

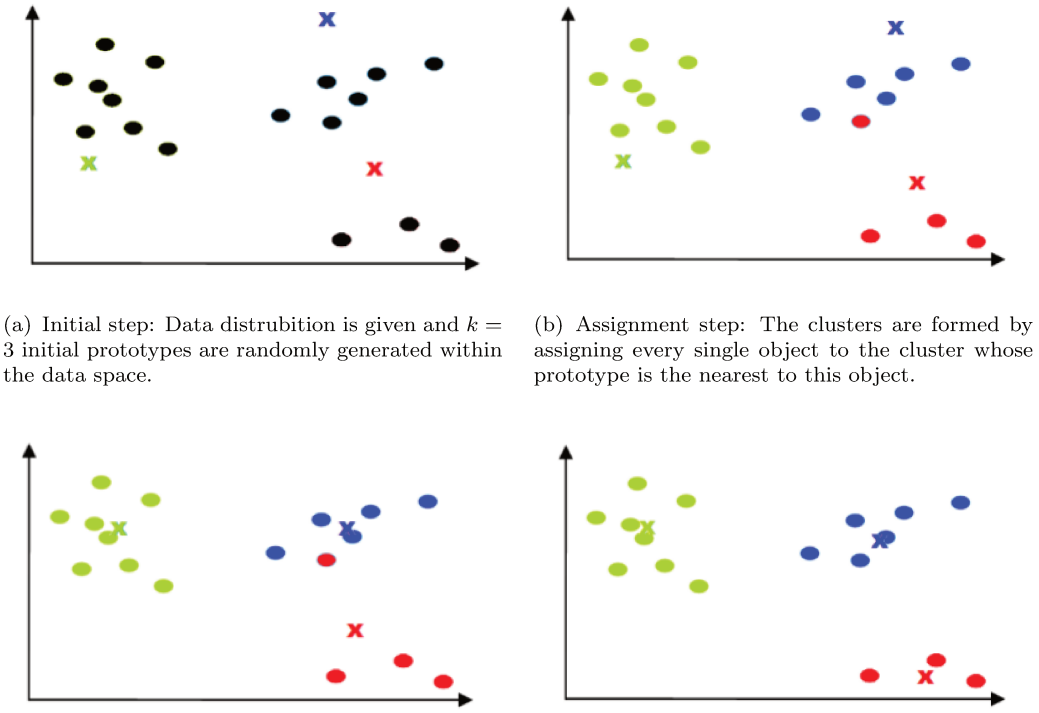
$$\phi(\mathbf{x}_i) = \arg \min_{j=1,...,k} \|\mathbf{x}_i - \mathbf{w}_j\|^2 \tag{3.2}$$

  assign data points to the nearest prototype (best match unit). This assures that the cost function $\mathcal{R}(\phi, \mathcal{W})$ is minimized with respect to the assignment function ϕ assuming that the prototype vectors are constant. Additionally, this step maps data to the network nodes.

- **Minimization step**: the second phase of iteration decreases again $\mathcal{R}_{k-means}(\phi, \mathcal{W})$ according to the set of referents $\mathcal{W}$. It is assumed in this case that ϕ is fixed at the current value. The referents $\mathbf{w}_c$ are calculated using the following equation:

$$\mathbf{w}_c = \frac{1}{n_c} \sum_{i=1}^{n_c} \mathbf{x}_i \tag{3.3}$$

The k-means algorithm is summarised in Figure 3.1 and written in Algorithm **??**.



(a) Initial step: Data distrubition is given and $k = 3$ initial prototypes are randomly generated within the data space.

(b) Assignment step: The clusters are formed by assigning every single object to the cluster whose prototype is the nearest to this object.

(c) Update step: The prototypes are updated and move to the local minimum and to cover data distrubition.

(d) The algorithm repeats until convergence or a stopping criterion has been fulfilled.

FIGURE 3.1: Clustering with k-means

k-means is a heuristic algorithm which has few shortcomings: it can converge to a local optimum, and the result depends on k and the initial prototypes. Therefore, some variants have been developped including topological models for example: SOM and Neural Gas that suffer less from these problems, because of the topological preservation.

Algorithm 1 K-means algorithm

initialize randomly K prototypes
repeat
  for $i = 1$ **to** N **do**
    $k = \text{argmin}_{k=1,..,K} \|\mathbf{x}_i - \mathbf{w}_k\|^2$
    $C_k = C_k \cup \mathbf{x}_i$ {assign $\mathbf{x}_i$ to cluster C_k}
  end for
  for $k = 1$ **to** K **do**
    $\mathbf{w}_k = \frac{1}{n_{C_k}} \sum_{j=1}^{n_{C_k}} \mathbf{x}_j$ {update prototype k, where n_{C_k} is the cardinality of cluster C_k}
  end for
until stopping criterion has been fulfilled

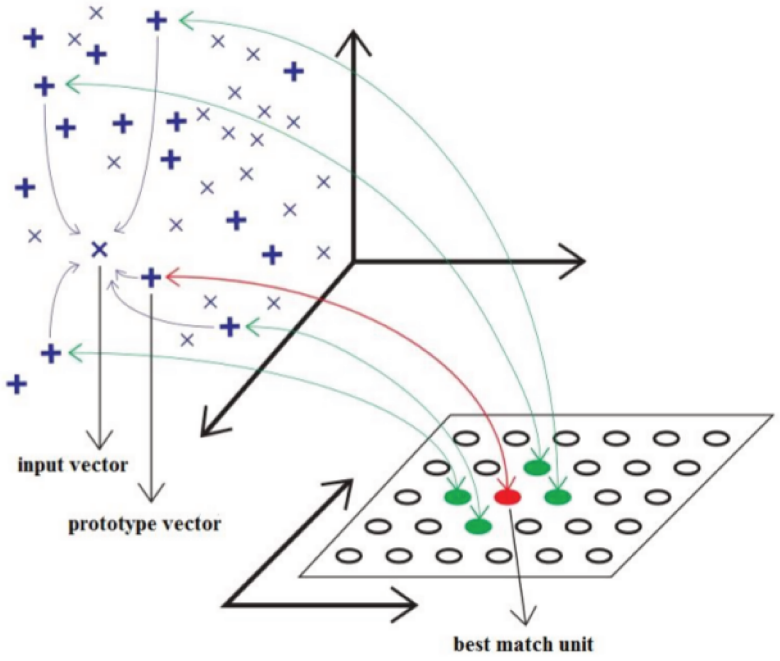
3.2.2 Self-Organizing Map (SOM)

The SOM algorithm, proposed by Kohonen [Kaski et al., 1998], is a type of artificial neural network for unsupervised learning. SOM has the ability of creating spatially organized internal representations of input objects and their abstractions. As in Figure 3.2, SOM produces a low-dimensional (1D or 2D) discretized representation (called a map or network) from the high-dimensional space of the input data. SOM uses a neighborhood function to preserve the topological properties of the input space, and forms a discretely topological map where similar objects are grouped close together and dissimilar ones apart. Like most artificial neural networks [Haykin, 1998], SOM has a two-fold objective:

1. **Training map**: build the topological map using the input data. A map consists of a number of network nodes arranged according to a structure defined a priori. The usual arrangement of the network nodes is a 1D or 2D, hexagonal or rectangular grid. Associated with each network node is a prototype, $\mathbf{w}_c$, of the same dimension as the input data points.

2. **Mapping (quantization)**: put the input data into a non-linear, discrete map. Vector quantization techniques assign a data point to a prototype such that the distance from this point to the best match prototype is the smallest. This process will respect the neighborhood function to preserve data topology. Data points which are similar into the input space will be put onto neighbor network nodes.

At the start of the learning, a discrete topological map of size $p \times q = k$ is initialized. We denote $\mathcal{C} = \{c_1, ..., c_k\}$ where c_i $(i = 1, ..., k)$ is a network node.



(a)

FIGURE 3.2: SOM principles: mapping and quantization

$\mathcal{C}$ is associated with $\mathcal{W} = \{\mathbf{w}_1, .., \mathbf{w}_k\}$ where $\mathbf{w}_i = (w_i^1, ..., w_i^k)$ is the prototype associated with the network node c_i. For each pair of network nodes c_r and c_s in $\mathcal{C}$, their mutual influence is defined by the function $\mathcal{K}^T(\delta(c_r, c_s))$ as in Equation (3.4). A Gaussian function is a common choice for $\mathcal{K}$ that will shrink with time.

$$\mathcal{K}^T(\delta(c_r, c_s)) = e^{\frac{-\delta(c_r, c_s)}{T}} \tag{3.4}$$

where T represents the temperature which decreases the value of T between two values T_{max} and T_{min}, to control the size of the neighborhood influencing a given cell on the map :

$$T = T_{max}\left(\frac{T_{min}}{T_{max}}\right)^{\frac{ith}{N_{iter}-1}} \tag{3.5}$$

N_{iter} is the number of iterations, and $\delta(c_r, c_s)$ is defined as the shortest distance between the two network nodes c_r and c_s.

Due to the use of this function $\mathcal{K}$, in the training the whole neighborhood network nodes move along in the same direction towards the learning data, similar data tend to be put in the adjacent network nodes [Kohonen et al., 2001a].

There are mainly two versions of SOM algorithm: stochastic and batch, both aim to minimize the cost function presented in equation 3.6.

$$\mathcal{R}_{SOM}(\phi, \mathcal{W}) = \sum_{i=1}^{n} \sum_{j=1}^{k} \mathcal{K}^T\big(\delta(\phi(\mathbf{x}_i), c_j)\big) \|\mathbf{x}_i - \mathbf{w}_j\|^2 \tag{3.6}$$

where $\phi(\mathbf{x}_i)$ is the assignment function which returns the network node to which $\mathbf{x}_i$ is assigned:

$$\phi(\mathbf{x}_i) = \arg \min_{j=1,\dots,k} \|\mathbf{x}_i - \mathbf{w}_j\|^2 \tag{3.7}$$

The learning steps are similar to the steps of k-means:

1. **Initialization step**: initialize the map structure, i.e., the number of network nodes (or k clusters), the arrangement shape: hexagonal or rectangular and the initial prototypes.
2. **Assignment step**: assign data points to the nearest prototype (best match unit). This assures that the cost function $\mathcal{R}(\phi, \mathcal{W})$ is minimized with respect to the assignment function ϕ assuming that the prototype vectors are constant. Additionally, this step maps data to the network nodes.
3. **Update step**: re-compute the prototype vectors. The prototypes and their neighbors move along together towards the assigned data such that the map tends to approximate the data distribution. It includes minimizing the cost function $\mathcal{R}(\phi, \mathcal{W})$ with respect to the prototypes vectors assuming that data are all assigned to the best match unit.

Batch SOM

In batch version, the prototypes are updated according to the following equation:

$$\mathbf{w}_c = \frac{\sum_{r=1} \mathcal{K}^T(\delta(c, r)) \sum_{i=1}^{n_r} \mathbf{x}_i}{\sum_{r=1} \mathcal{K}^T(\delta(c, r)) n_r} \tag{3.8}$$

where n_r is the number of data assigned to cluster r. This formula is obtained by fixing ϕ and minimizing $\mathcal{R}$ with respect to $\mathcal{W}$. The assignment function in the batch version is calculated according to the following equation:

$$\phi(\mathbf{x}_i) = \arg \min_{j=1,\dots,k} \mathcal{K}^T(\delta(\mathbf{x}_i, \mathbf{w}_j)) \|\mathbf{x}_i - \mathbf{w}_j\|^2 \tag{3.9}$$

Algorithm 2 Batch SOM version

initialize K prototypes and $\mathcal{W}$
while stopping criteria have not been fulfilled **do**
  for $i = 1 \rightarrow N$ **do**
    $\phi(\mathbf{x}_i) = \text{argmin}_{k=1,..,K} \mathcal{K}^T(d(\mathbf{x}_i, \mathbf{w}_k))\|\mathbf{x}_i - \mathbf{w}_k\|^2$ {Find the best match unit to the current selected vector.}
    $C_{\phi(\mathbf{x}_i)} = C_{\phi(\mathbf{x}_i)} \cup \{\mathbf{x}_i\}$ {Put $\mathbf{x}_i$ into cluster $\phi(\mathbf{x}_i)$}
  end for
  for $k = 1 \rightarrow K$ **do**
    $\mathbf{w}_k = \frac{\sum_{r=1}^{K} \mathcal{K}^T(\delta(C_c, C_r)) \sum_{j=1}^{n_{C_r}} \mathbf{x}_j}{\sum_{r=1}^{K} \mathcal{K}^T(\delta(C_c, C_r)) n_r}$ {Update prototype vectors where n_r is the number of data found in cluster r }
  end for
end while

Stochastic SOM

In the stochastic version, each iteration consists of presenting the SOM map a data point randomly selected. The best match unit (the neatest node) as well as its neighbors move to the input point (see Figure 3.2).

Unlike the batch version, the stochastic version uses the gradient descent method in order to update prototypes:

$$\mathbf{w}_c^t = \mathbf{w}_c^{t-1} - \mu^t \mathcal{K}^T(\delta(c, c_{\phi(\mathbf{x}_i)}))(\mathbf{w}_c^{t-1} - \mathbf{x}_i) \tag{3.10}$$

where μ^t is an adaptation parameter, called "the learning rate" which decreases with time t.

Stochastic SOM

3.2.3 Neural Gas

Neural Gas (NG) [Martinetz and Schulten, 1991] is inspired by the SOM. While the SOM map dimensionality must be chosen a priori; depending on the data distribution, the topological network of neural gas may have a different arrangement. Neural Gas is a more flexible network capable of quantizing topological data and learning the similarity among the input data without defining a network topology. Unlike SOM, the adaptation strength in Neural Gas is constant over time and only

Algorithm 3 Stochastic SOM version

```
initialize K prototypes and 𝒲
while stopping criteria have not been fulfilled do
  for i = 1 → N do
    φ(x_i) = argmin_{k=1,..,K} ‖x_i − w_k‖² {Find the best match unit to the current selected vector.}
    for all C_r is a neighbor of φ(x_i) (including φ(x_i) itself) do
      w_r = w_r + 𝒦^T(δ(C_{φ(x_i)}, C_r))(x_i − w_r) {Update the nodes in the neighborhood of φ(x_i) (including the node φ(x_i) itself) by pulling them closer to the input vector.}
    end for
  end for
end while
```

the best match prototype and its direct topological neighbors are adapted.

Given a network of k clusters $\mathcal{C} = \{c_1, ..., c_k\}$ associated with k prototypes $\mathcal{W} = \{\mathbf{w}_1, ..., \mathbf{w}_k\}$, they are adapted independently of any topological arrangement of the network nodes within the network. Instead, the adaptation step is affected by the topological arrangement within the input space. For each data point $\mathbf{x}_i$ is selected, prototypes will be ajusted by distortions $\mathcal{D}(\mathbf{x}_i, c_j) = \|\mathbf{x}_i - \mathbf{w}_j\|, \forall j = 1, ..., k$. The resulting adaptation rule can be described as a "winner takes most" instead of a "winner takes all" rule [Fritzke, 1991]. The winner network node denoted by j_0 is determined by the assignment function

$$j_0 = \phi(\mathbf{x}_i) = \arg \min_{j=1,...,k} \|\mathbf{x}_i - \mathbf{w}_j\|^2. \tag{3.11}$$

An edge that connects the network node adjacent, denoted by j_1, to the winner node j_0 which is then stored in a matrix $\mathcal{S}$ representing the neighborhood relationships among the input data:

$$\mathcal{S}_{ij} = \begin{cases} 1 & \text{if a connection exists between } c_i \text{ and } c_j \ (\forall i, j = 1, ..., k, i \neq j) \\ 0 & \text{otherwise} \end{cases}$$

When an observation is selected, the prototypes move toward it by adjusting the distortion $\mathcal{D}(\mathbf{x}_i, c_{j_0})$, controlled by a neighborhood function $\mathcal{K}^T$. In [Fritzke, 1991], this function is fixed, e.g. $\mathcal{K}^T = \exp^{knn_j/T}$ where knn_j is the number of neighborhood network nodes of c_j. This affects directly to the adaptation step for $\mathbf{w}_j$ which is determined by:

$$\mathbf{w}_j^t = \mathbf{w}_j^{t-1} - \varepsilon \mathcal{K}^T(\delta(c_j, c_{\phi(\mathbf{x}_i)}))(\mathbf{x}_i - \mathbf{w}_j) \tag{3.12}$$

To capture the topological relations between the prototypes, each time an observation is presented, the connection between j_0 and j_1 is incremented by one. Each connection is associated with an "age" variable. Only the connection between j_0 and j_1 is reset, the other connections of j_0 age, i.e. their age increment. When the age of a connection exceeds a specific lifetime Max_{age}, it is removed. The way to update the age of the connections is to increase with each incoming input object is learnt. Finally, Neural Gas can be summarized by the Algorithm **??**.

Algorithm 4 Neural Gas algorithm

1: Initialize K prototypes and set all $\mathcal{S}_{ij}$ to zero
2: **for all** $\mathbf{x}_i \in \mathcal{X}$ **do**
3: determine the sequence $(C_{k_0}, C_{k_1}, ..., C_{k_{N-1}})$ such that

$$\|\mathbf{x}_i - \mathbf{w}_{k_0}\| < \|\mathbf{x}_i - \mathbf{w}_{k_1}\| < .. < \|\mathbf{x}_i - \mathbf{w}_{k_{K-1}}\|$$

{$\mathbf{w}_{k_0}$ is the best match prototype, i.e the nearest prototype; $\mathbf{w}_{k_1}$ is the second nearest prototype to $\mathbf{x}_i$}
4: **for all** C_j with $\mathcal{S}_{k_0,j} == 1$ **do**
5: $\mathbf{w}_j = \mathbf{w}_j + \epsilon\mathcal{K}^T(\mathbf{x}_i - \mathbf{w}_j)$ {perform an adaptation step for the prototypes}
6: **end for**
7: **if** $\mathcal{S}_{k_0,k_1} == 0$ **then**
8: $\mathcal{S}_{k_0,k_1} = 1$ {create a topological connection between C_{k_0} and C_{k_1}}
9: $age_{k_0,k_1} = 0$ {set age for this connection}
10: **end if**
11: **for all** C_j with $\mathcal{S}_{k_0,j} == 1$ **do**
12: $age_{k_0,j} = age_{k_0,j} + 1$ {increase the age of all connections of k_0 by one}
13: **if** $age_{k_0,j} > Max_{age}$ **then**
14: $\mathcal{S}_{k_0,j} = 0$ {remove all connections of k_0 which exceeded their age }
15: **end if**
16: **end for**
17: **end for**

In this algorithm, stopping criteria can be either:

- a number of iterations
- a threshold for the quantization error.

3.2.4 Growing Neural Gas

Growing Neural Gas (GNG) [Fritzke, 1994] is an incremental self-organizing approach which belongs to the family of topological maps such as Self-Organizing

Maps (SOM) [Kohonen et al., 2001a] or Neural Gas (NG) [Martinetz and Schulten, 1991]. It is an unsupervised clustering algorithm capable of representing a high dimensional input space in a low dimensional feature map. Typically, it is used for finding topological structures that closely reflect the structure of the input distribution. Therefore, it is used for *visualization tasks* in a number of domains [Martinetz and Schulten, 1991, Beyer and Cimiano, 2012] as neurons (nodes), which represent prototypes, are easy to understand and interpret.

The GNG method has no input parameters which change over time and is able to continue learning, adding network units and connections. As an incremental variant of Neural Gas, GNG inherits its principle; however it does not impose the strict network-topology preservation rule. The network incrementally learns the topological relationships inherent in the dataset, and continues until a stopping criterion is fulfilled. Before learning, only $k = 2$ prototypes are initialized. Step by step, after a certain number of iterations (called epoch), a new network node is successively added into the topological network. But how to add a new network node? Now, this relates to quantization error.

In the clustering problem, the goal is always to minimize the quantization error of datasets or data within the clusters. Therefore, the cluster that provides a high value of quantization error is not a good one. We should divide this cluster into smaller clusters. GNG finds the two clusters c_1 and c_2 which have the highest quantization error. Then a new node is inserted halfway between these two nodes by the following expression:

$$\mathbf{w}_{new} = \frac{1}{2}(\mathbf{w}_1 + \mathbf{w}_2) \tag{3.13}$$

The node insertion will be repeated until a stoping criterion is fulfilled.

3.2.5 DBSCAN

Density-based clustering has the ability to discover arbitrary-shape clusters and to handle noise [Amini et al., 2014]. In density-based clustering methods, clusters are formed based on the dense areas that are separated by sparse areas. DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [Ester et al., 1996] is one of the most well-known density-based clustering algorithms.

DBSCAN is developed for clustering large spatial databases with noise, based on connected regions with high density. The density of each point is defined

based on the number of points close to that particular point called the point's neighborhood. The dense neighborhood is defined based on two user-specified parameters: the radius (ε) of the neighborhood (ε-neighborhood), and the number of the objects in the neighborhood ($MinPts$).

The basic definitions in DBSCAN are introduced in the following, where $\mathcal{X}$ is a current set of data points:

- ε-neighborhood of a point: the neighborhood within a radius of ε of a point $\mathbf{x}_p$ is denoted by $N_\varepsilon(\mathbf{x}_p)$:

$$N_\varepsilon(\mathbf{x}_p) = \{\mathbf{x}_q \in \mathcal{X} | dist(\mathbf{x}_p, \mathbf{x}_q) \leqslant \varepsilon\},$$

  where $dist(\mathbf{x}_p, \mathbf{x}_q)$ denotes the Euclidean distance between points $\mathbf{x}_p$ and $\mathbf{x}_q$;

- $MinPts$: the minimum number of points around a data point in the ε-neighborhood;

- core point: a point where the cardinality of its ε-neighborhood is at least $MinPts$ (see Figure 3.3);

- border point: a point is a border point if the cardinality of its ε-neighborhood is less than $MinPts$ and at least one of its ε-neighbors is a core point (see Figure 3.3);

- noise point: a point is a noise point if the cardinality of its ε-neighborhood is less than $MinPts$ and none of its neighbors is a core point (see Figure 3.3);

- directly density reachable: a point $\mathbf{x}_p$ is directly density reachable from point $\mathbf{x}_q$, if $\mathbf{x}_p$ is in the ε-neighborhood of $\mathbf{x}_q$ and $\mathbf{x}_q$ is a core point;

- density reachable: a point $\mathbf{x}_p$ is density reachable from point $\mathbf{x}_q$, if $\mathbf{x}_p$ is in the ε-neighborhood of $\mathbf{x}_q$ and $\mathbf{x}_q$ is not a core point but they are reachable through chains of directly density reachable points;

- density-connected: if two points $\mathbf{x}_p$ and $\mathbf{x}_q$ are density reachable from a core point $\mathbf{x}_o$, $\mathbf{x}_p$ and $\mathbf{x}_q$ are density-connected;

- cluster: a maximal set of density-connected points.

DBSCAN starts by randomly selecting a point and checking whether the ε-neighborhood of the point contains at least $MinPts$ points. If not, it is considered

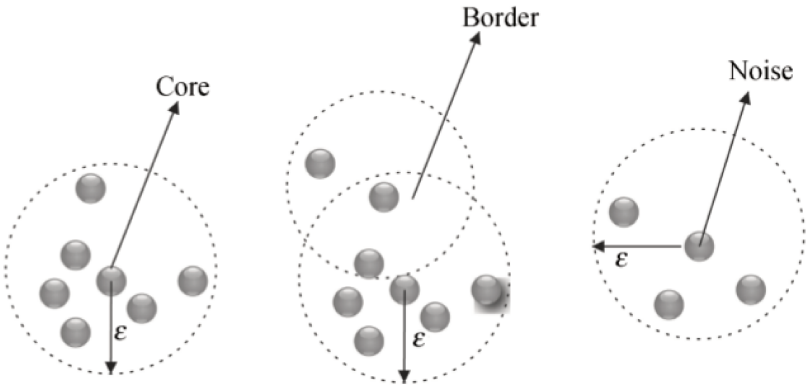


FIGURE 3.3: DBSCAN: core, border, and noise points [Ester et al., 1996].

as a noise point, otherwise it is considered as a core point and a new cluster is created. DBSCAN iteratively adds the data points, which do not belong to any cluster and are directly density reachable from the core points of a new cluster. If the new cluster can no longer be expanded, the new cluster is completed. In order to find the next cluster, DBSCAN randomly selects the unvisited data points and the clustering process continues until all the points are visited and no new point is added to any cluster [Ester et al., 1996, Amini et al., 2014].

3.2.6 EM Algorithm

In the probabilistic approach, data is considered to be a sample independently drawn from a mixture model of several probability distribution [**?**]. Each component of the mixture corresponds to a cluster; additional criteria are used to automatically determine the number of clusters [Fraley and Raftery, 1998]. The Expectation Maximization (EM) algorithm [Dempster et al., 1977a, McLachlan and Krishnan, 2007] is a general approach to maximum likelihood in the presence of incomplete data.

The overall likelihood of the training data is its probability to be drawn from a given mixture model.

$$\mathcal{V}(\mathbf{x}_1, ..., \mathbf{x}_n; \theta) = \prod_{i=1}^{n} \sum_{j=1}^{k} \pi_j \varphi_j(\mathbf{x}_i; \alpha_j) \tag{3.14}$$

with

$\forall j = 1, ..., k, \pi_j \in [0, 1]$ and $\sum_{j=1}^{k} \pi_j = 1$
where:

- $\varphi_j(\mathbf{x}_i; \alpha_j)$ represents the probability density.
- π_j denotes the probability that an element of the sample follows the law φ.
- $\theta = (\pi_1, ..., \pi_k; \alpha_1, ..., \alpha_k)$ represents the unknown parameter of the mixture model.

By introducing the log-likelihood, the Equation (3.14) can be rewritten as follows:

$$\mathcal{L}(\mathbf{x}_1, ..., \mathbf{x}_n; \theta) = \sum_{i=1}^{n} \log \Big(\sum_{j=1}^{k} \pi_j \varphi_j(\mathbf{x}_i; \alpha_j) \Big) \tag{3.15}$$

Log-likelihood serves as an objective function, which gives rise to the EM method. EM is a two-step iterative optimization:

- The Step E estimates probabilities $\varphi_j(\mathbf{x}_i; \alpha_j)$, which is equivalent to a soft reassignment.
- The Step M finds an approximation to a mixture model, given current soft assignments.

This boils down to finding mixture model parameters that maximize log-likelihood. The process continues until log-likelihood convergence is achieved.

In [Attar et al., 2013, El Attar, 2012], the authors have proposed an estimation of probability distribution over a data set which is distributed into subsets located on the nodes of a distributed system. More precisely, the global distribution is estimated by aggregating local distributions which are modelled as a Gaussian mixture.

3.2.7 Computational complexity

In Table 3.1, we report the computational complexity of the data clustering algorithms presented above, where n is the number of data points and k is the number of network nodes (or clusters).

Algorithm	Complexity
k-means	$O(kn)$
SOM	$O(kn \log n)$
NG	$O(kn \log n)$
GNG	$O(kn^2)$
DBSCAN	$O(n^2)$; by using spatial indices, it decreases to $O(n \log n)$
EM	$O(kn)$

TABLE 3.1: Computational complexity of clustering algorithms

3.3 Scalable clustering

In this section, we will describe in details the implementation of some of the most well-known and commonly used clustering methods, using the MapReduce paradigm. This will give the reader a clear idea on how to scale any data clustering algorithm in MapReduce.

As described in chapter 2, MapReduce [Dean and Ghemawat, 2008a, Lämmel, 2008] is a programming model and an associated implementation for processing and generating large datasets that is amenable to a broad variety of real-world tasks. Users specify the computation in terms of a *map* and a *reduce* function, and the underlying runtime system automatically parallelizes the computation across large-scale clusters of machines, handles machine failures, and schedules inter-machine communication to make efficient use of the network and disks.

3.3.1 General Framework

In contrast to the typical single machine clustering, parallel and distributed (scalable) algorithms use multiple machine to speed up the computation and increase the scalability.

Most parallel and distributed clustering algorithms follow the general framework depicted in Figure 3.4 [Januzaj et al., 2004, Sarazin et al., 2014, Zhao et al., 2009]

1. **Partition.** Data are partitioned and distributed over machines.

2. **Local Clustering.** Each machine performs local clustering on its partion of the data.

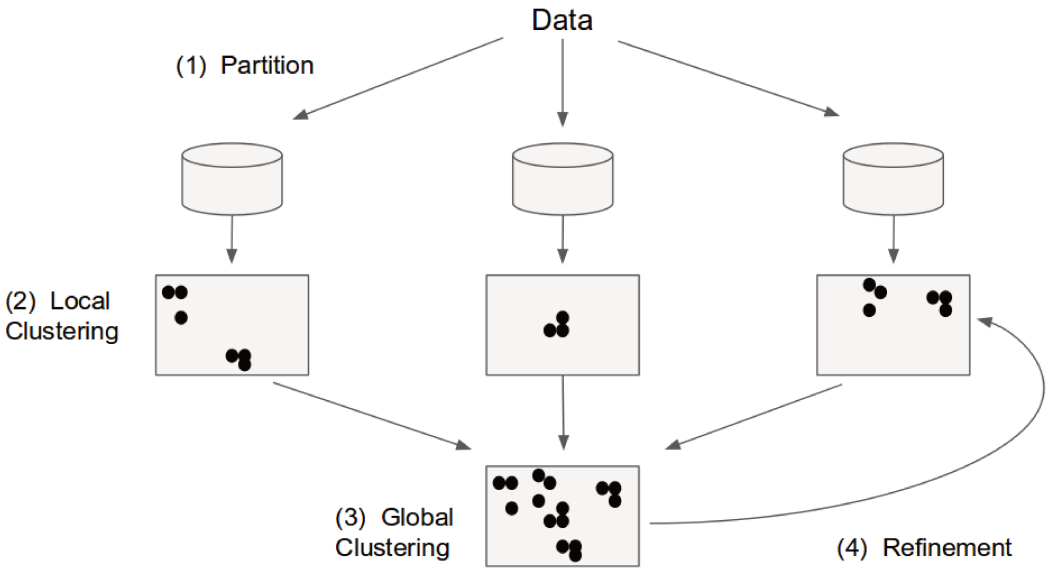


FIGURE 3.4: The general framework of most parallel and distributed clustering algorithms [Aggarwal and Reddy, 2014].

3. **Global Clustering.** The cluster information from the previous step is aggregated globally to produce global clusters.

4. **Refinement of Local Clusters.** Optionally, the global clusters are sent back to each machine to refine the local clusters.

3.3.2 Scalable *k*-means using MapReduce

Zhao et al. [2009] proposed a parallel and distributed implementation of k-means in MapReduce. The proposed algorithm, called PKMeans, is implemented using Hadoop[1] to make the clustering method applicable to large scale data.

Since the most intensive calculation to occur in k-means is the calculation of distances, the idea of PKMeans is to execute in a parallel manner these distance computations between different observations with prototypes. In a nutshell, the map function performs the procedure of assigning each data-point to the closest cluster while the reduce function performs the procedure of updating the new clusters. In order to decrease the cost of network communication, a combiner function is developed to deal with partial combination of the intermediate values with the same key within the same map task.

The input dataset is stored in an HDFS [Shvachko et al., 2010] as a sequence

[1] http://lucene.apache.org/hadoop/

file of $< key, value >$ pairs, each of which represents a record in the dataset. The key is the offset in bytes of this record to the start point of the data file, and the value is a string of the content of this record. The dataset is split and globally broadcast to all mappers. Consequently, the distance computations are executed in parallel. For each map task, PKMeans construct a global variable *clusters* which is an array containing the information about centers of the clusters. Given the information, a mapper can compute the closest cluster for each data-point. The intermediate values are then composed of two parts: the index of the closest cluster and the data-point information [Zhao et al., 2009]. The pseudocode of the map function is shown in Algorithm 5.

Algorithm 5 map(key, value)

Require: Global variable *clusters*, the offset *key*, the data-point *value*
Ensure: $< key', value' >$ pair, where the key' is the index of the closest cluster and $value'$ is a string comprise of data-point information

```
1: Construct the data-point instance from value
2: minDist = Double.MAX_VALUE
3: index = −1
4: for each cluster c_i ∈ C do
5:    dist = ComputeDistance(instance, c_i)
6:    if dist < minDist then
7:       minDist = dist
8:       index = i
9:    end if
10: end for
11: Take index as key'
12: Construct value' as a string comprise of the values of different dimensions
13: output < key', value' > pair
```

In the *combine* function, we partially sum the values of the points assigned to the same cluster. In order to calculate the mean value of the objects for each cluster, we should record the number of data-points in the same cluster in the same map task. This procedure does not consume the communication cost because the intermediate data is stored in local disk of the host. The pseudocode for the combine function is shown in Algorithm 6.

In the *reduce* function, we sum all the data-points and compute the total number of data-points assigned to the same cluster. Therefore, we can obtain the new cluster centers which are used for next iteration. The pseudocode for the reduce function is shown in Algorithm 7.

Algorithm 6 combine(key, V)

Require: *key* is the index of the cluster, V is the list of the data-points assigned to the same cluster

Ensure: $< key', value' >$ pair, where the key' is the index of the cluster and $value'$ is a string comprised of sum of the data-points in the same cluster and the data-point number

Initialize one array to record the sum of value of each dimensions of the data-points contained in the same cluster, i.e. the data-points in the list V Initialize a counter *num* as 0 to record the number of data-points in the same cluster

1: **for** each value $v \in V$ **do**
2: Construct the data-point *instance* from v
3: Add the values of different dimensions of *instance* to the array
4: $num = num + 1$
5: **end for**
6: Take *key* as key'
7: Construct $value'$ as a string comprised of the sum values of different dimensions and *num*
8: output $< key', value' >$ pair

Algorithm 7 reduce(key, V)

Require: *key* is the index of the cluster, V is the list of the partial sums from different host

Ensure: $< key', value' >$ pair, where the key' is the index of the cluster and $value'$ is a string representing a new cluster center

Initialize one array record the sum of value of each dimensions of the data-points contained in the same cluster, e.g. the data-points in the list V Initialize a counter *NUM* as 0 to record the number of data-points in the same cluster

1: **for** each value $v \in V$ **do**
2: Construct the data-point *instance* from v
3: Add the values of different dimensions of *instance* to the array
4: $NUM = NUM + num$
5: **end for**
6: Divide the entries of the array by *NUM* to get the new cluster's coordinates
7: Take *key* as key'
8: Construct $value'$ as a string comprise the cluster's coordinates
9: output $< key', value' >$ pair

3.3.3 Scalable DBSCAN using MapReduce

A recent proposed algorithm is MR-DBSCAN [He et al., 2014] which is a scalable MapReduce-based DBSCAN algorithm. Three major drawbacks are existed in parallel DBSCAN algorithms which MR-DBSCAN is fulfilling [Shirkhorshidi et al., 2014]:

1. They are not successful to balance the load between the parallel nodes
2. These algorithms are limited in scalability because all critical sub procedures are not parallelized
3. Their architecture and design limit them to less portability to emerging parallel processing paradigms.

MR-DBSCAN proposes a novel data partitioning method based on computation cost emission as well as a scalable DBSCAN algorithm in which all critical sub-procedures are fully parallelized. The MR-DBSCAN algorithm consists of three stages: data partitioning, local clustering, and global merging.

The first stage divides the whole dataset into smaller partitions according to spatial proximity. In the second stage, each partition is clustered independently. Then the partial clustering results are aggregated in the last stage to generate the global clusters. Experiments on large datasets confirm the scalability and efficiency of MR-DBSCAN.

3.3.4 Scalable EM using MapReduce

Expectation Maximization (EM) is used to learn the maximum likelihood parameters in the presence of incomplete data.

Many works have been proposed to scale-up the EM algorithm [Das et al., 2007, Cui et al., Basak et al., 2012]. The parallel implementation of EM proposed in [Cui et al.] is coded in Spark.

- Each E-step is a Spark map transformation which runs in parallel mapping each $\mathbf{x}_i$ to a vector of conditional probability densities.

- Each M-step is a reduce action which goes through all the observations in the RDD, aggregating results from E-step.

In their implementation, each iteration consists of two map operations and two reduce operations. In the first map operation, we calculate the responsibility (the log-likelihood, $\mathcal{L}$) of each cluster for each data point, along with the product of the data point and $\mathcal{L}$ and the sum of the products for all clusters. Then we do a reduce operation to calculate the new centers for each cluster. In the las step, we do another map and reduce to calculate the covariance of each cluster.

3.3.5 MapReduce-based Models and Libraries

Due to the interest of the MapReduce framework, some studies have used it for scaling clustering algorithms. As examples, we can cite the implementation of the EM algorithm in MapReduce [Das et al., 2007], the parallel version of the k-means++ initialization algorithm [Bahmani et al., 2012], and the work considered in [Ene et al., 2011a] which is a MapReduce implementation of the k-medean problem.

Currently, more and more libraries have emerged offering MapReduce-based implementations of machine learning algorithms:

- **MLlib.**[2] This is Spark's machine learning library. It consists of common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, as well as lower-level optimization primitives and higher-level pipeline APIs.
- **Apache Mahout.**[3] Is a project of the Apache Software Foundation to produce free implementations of distributed or otherwise scalable machine learning algorithms focused primarily in the areas of collaborative filtering, clustering and classification. Currently, the supported algebraic platforms are Apache Spark[4] and H20[5], and Apache Flink[6]. Since April 2014, support for Hadoop MapReduce[7] algorithms is being gradually phased out.

[2] http://spark.apache.org/docs/latest/mllib-guide.html
[3] http://mahout.apache.org/
[4] http://spark.apache.org/docs/latest/index.html
[5] http://www.h2o.ai/
[6] http://hadoop.apache.org/
[7] http://hadoop.apache.org/

3.4 Conclusion

As data clustering has attracted a significant amount of research attention, many clustering algorithms have been proposed in the past decades. However, the engrowing volumes of information made possible by technological advances, makes clustering of very large data a challenging task.

Currently, the MapReduce paradigm has met with a resounding success in this era of Data Science due to, amongst others, its simplicity. The challenge in scaling a data clustering method is not only to use the MapReduce paradigm but also to decompose the problem in small functions, the *map* and *reduce* functions. Usually, scaling an algorithm using MapReduce needs a redefintion of the initial problem.

In the next chapter, we will review and discusse ...

Chapter 4

Bi-Clustering Algorithms

This chapter represents a comprehensive survey on bi-clustering methods. These are algorithms that perform simultaneous clustering on the row and column dimensions of the data matrix. We analyze a large number of existing approaches to biclustering, and classify them in accordance with the type of biclusters they can find, the patterns of biclusters that are discovered, the methods used to perform the search.

4.1 Introduction

In the field of clustering, although most of methods used aim to construct partitions either on the set of observations or on the variables separately, there are other methods of bi-clustering that simultaneously consider the two sets [Govaert, 1983, Abdullah and Hussain, 2006, Govaert and Nadif, 2009, Kwon and Cho, 2010, Ayadi et al., 2012, de França et al., 2013]. Compared to the classical clustering, by not privileging one set over another, bi-clustering is more efficient for discovering homogeneous blocks in a data matrix. In recent years, this family of approaches has attracted great interest in different scientific communities and in various fields such as data mining.

Bi-clustering approaches have become a topic of major interest because of its many applications in the field of data mining. A bi-partitioning method, also called "bi-clustering", co-clustering or cross-classification, is a method of analysis that aims to group data according to their similarity. The traditional strategy of

bi-partitioning methods seeks to find sub-matrices, or blocks that represent subgroups of rows and subgroups of columns in a data matrix.

One of the objectives of a bi-classification method is the search for a pair of partitions, one on the observations (the lines of a data matrix), the other on the columns (columns of a data matrix), such as the "loss of information" due to grouping is minimal; That is to say, so that the difference between the information provided by the initial table and that provided by the table obtained after grouping is minimal.

Since the first bi-partitioning algorithm, called "Block Clustering", proposed by [Hartigan], many techniques have been proposed, such as ([Tanay et al., 2002]), spectral analysis ([Greene and Cunningham, 2010]), ([Shan et al., 2010]) and others ([Angiulli et al., 2006]). The authors of [Charrad et al., 2008] classify the bi-partitioning methods into four major categories: divisive, constructive, probabilistic and partitioning-based methods.

4.2 Partitioning-based methods

K-means algorithms have long been used in bi-partitioning. Indeed, [Govaert, 1983] defined a bi-partitioning algorithm called "Croeuc" which consists in determining a series of pairs of partitions minimizing a cost function on the matrix of the data by applying the K-means alternatively on rows and columns. The Croeuc algorithm is proposed for continuous data.

Let $\mathcal{A}$ a data matrix with N observations and d variables, x_i^j such that $1 < i < N, 1 < j < d$ are the elements of the matrix $\mathcal{A}$. The observations are partitioned into K classes. Similarly, variables are partitioned into L classes. P_k and Q_l represent the row and column partitions, respectively. The optimal partitions P and Q are obtained by means of an iterative algorithm which uses the sum of the Euclidean distances as a measurement of the deviation between the data matrix $\mathcal{A}$.

The goal of the Croeuc algorithm is to find a pair of partitions (P, Q) and g, such that the following criterion is minimized [Jollois, 2003]: [1]

$$W(P, Q, g) = \sum_{k=1}^{K} \sum_{l=1}^{L} \sum_{i \in P_k} \sum_{j \in Q_l} (x_i^j - g_k^l)^2 \tag{4.1}$$

[1] We have included in this part of work the same notation as that used in the thesis of Xavier Jollois [Jollois, 2003].

where

- $P = (P_1, \ldots, P_K)$ is the partition of observations into K classes,
- $Q = (Q_1 \ldots, Q_L)$ is the partition of variables into L classes,
- g_k^l the center of the block $\mathbf{x}_k^l$ (prototype).

It is easy to see that for (P, Q) the optimal values of g_k^l are the averages of each x_i^j belonging to block $\mathbf{x}_k^l$. The main steps of the Croeuc algorithm are:

Algorithm 8 : Croeuc Algorithm

Start from an initial position (P^0, Q^0, g^0)
Calculate $(P^{(c+1)}, Q^{(c+1)}, g^{(c+1)})$ from $(P^{(c)}, Q^{(c)}, g^{(c)})$:

- 2(a) Calculate $(P^{(c)}, Q^{(c)}, g^{'})$ from $(P^{(c)}, Q^{(c)}, g^{(c)})$,
- 2(b) Calculate $(P^{(c+1)}, Q^{(c+1)}, g^{(c+1)})$ from $(P^{(c)}, Q^{(c)}, g^{'})$.

Repeat step 2 until the convergence of the algorithm.

It should be noted that in step 2, the algorithm 8 uses a double K-means (first phase K-means on the lines, second phase K-means on the columns). It is therefore necessary to optimize alternately the following criteria (deduced from 4.1):

$$W(P, g/Q) = \sum_{k=1}^{K} \sum_{i \in P_k} \sum_{l}^{L} |Q_l|(u_i^l - g_k^l)^2 \tag{4.2}$$

where $u_i^l = \frac{\sum_{j \in Q_l} x_i^j}{|Q_l|}$, and

$$W(Q, g/P) = \sum_{l=1}^{L} \sum_{j \in Q_l} \sum_{k=1}^{K} |P_k|(v_k^j - g_k^l)^2 \tag{4.3}$$

where $v_k^j = \frac{\sum_{i \in P_k} x_i^j}{|P_l|}$

Step 2 (a) of algorithm 8 is performed by the K-means algorithm using the matrix u_i^l. Alternatively, step 2 (b) is obtained by the K-means algorithm using the matrix v_k^j. Thus, at convergence, homogeneous blocks are obtained by reorganizing the rows and the columns according to the partitions P and Q. Each block (k, l), defined by the elements x_i^j for $i \; in P_k$ and $j \; in Q_l$ is characterized by $g_k L$.

The advantage of this algorithm was highlighted in comparison with K-means applied separately on the observations and variables of a data matrix [**?**]. Because of its simplicity and speed, the Croeuc algorithm can be applied to large data sets. However, it requires prior knowledge of the number of classes in rows and columns.

4.3 Probabilistic methods

In most cases, probabilistic methods are methods based on mixing models. The models of finite mixtures of probability laws are particularly used in bi-partitioning. Their use, as in clustering, amounts to assuming that the observations to be classified are derived from a model of mixture of which each component represents a class.

The authors assert that the data of a finite mixture of laws of probability $\mathcal{A} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ constitute a sample of n independent realizations of a random variable whose density function can be written as the following 4.4 equation:

$$\forall \mathbf{x}_i \ f(\mathbf{x}_i; \theta) = \sum_{k=1}^{K} \pi_k \varphi_k(\mathbf{x}_i; \alpha_k) \tag{4.4}$$

With:

$$\forall k = 1, \ldots, K, \pi_k \in [0, 1] \ \ et \ \ \sum_{k=1}^{K} \pi_k = 1$$

Where:

- $\varphi_k(\mathbf{x}_i; \alpha_k)$ represents the probability density.
- π_k refers to the probability that an element of the sample follows the law φ.
- $\theta = (\pi_1, \ldots, \pi_K; \alpha_1, \ldots, \alpha_K)$ represents the unknown parameter of the mixing model.

Mixing model for bi-clustering

As mentioned in the article [Govaert and Nadif, 2009], the formulation of the bi-partitioning problem uses the classical mixing model (equation 4.4), in which the

partition of the variables $\mathbf{w}$ is considered as a parameter of the model. The density of the mixture can be written as follows:

$$f(\mathbf{x}_i;\theta) = \sum_k \pi_k \varphi_k(\mathbf{x}_i;\mathbf{w},\alpha) \tag{4.5}$$

$$\varphi_k(\mathbf{x}_i;\mathbf{w},\alpha) = \prod_{j,l} \left(\frac{1}{\sqrt{2\pi\sigma_{kl}^2}} \exp^{-\frac{1}{2\sigma_{kl}^2}(\mathbf{x}_{ij}-\mu_{kl})^2} \right)^{w_{jl}} \tag{4.6}$$

- $\theta = (\pi, w, \alpha)$ represents the parameter of the mixture model which is formed by the proportions $\pi = (\pi_1, \ldots, \pi_g)$,
- the partition of variables $\mathbf{w}$ and the parameters of each component $\alpha = (\mu_{11}, \ldots, \mu_{gm}, \sigma_{11}^2, \ldots, \sigma_{gm}^2)$, where the μ_k and the σ_k represent the mean and the variances of each block.

The log-likelihood is written as follows:

$$L(\theta) = log\ f(\mathbf{x};\theta) = \sum_i log \sum_k \pi_k \varphi_k(\mathbf{x}_i;\mathbf{w},\alpha) \tag{4.7}$$

Let: $\mathbf{z}_k = \sum_i \mathbf{z}_{ik}$ and $\mathbf{w}_l = \sum_j \mathbf{w}_{jl}$ the cardinals of each class, the classifying log-likelihood checks:

$$L_c(\mathbf{z};\mathbf{w},\theta) = \sum_{i,k} z_{ik}\ log\ (\pi_k \varphi_k(\mathbf{x}_i;\mathbf{w},\alpha)) \tag{4.8}$$

In the case where the additive constant is equal to $\frac{nd}{2}\ log\ 2\ \pi$, the equation 4.8 takes the following form:

$$L_c(\mathbf{z};\mathbf{w},\theta) = \sum_k \mathbf{z}_k\ log\ \pi_k - \frac{1}{2} \sum_{i,j,k,l} \mathbf{z}_{ik}\mathbf{w}_{jl} \left(log\ \sigma_{kl}^2 + \frac{1}{\sigma_{kl}^2}(\mathbf{x}_{ij} - \mu_{kl})^2 \right) \tag{4.9}$$

The writing of the classifying log-likelihood L_c, defined for a partition $\mathbf{z}$, can then be extended to the fuzzy partition $\mathbf{s} = (\mathbf{s}_i^k; i = 1, \ldots, n; k = 1, \ldots, \mathbf{g})$ associated to the classification matrix defined by the conditional probabilities [Govaert and Nadif, 2009].

$$L_c(\mathbf{s};\mathbf{w},\theta) = \sum_{i,k} s_{ik}\ log\ (\pi_k \varphi_k(\mathbf{x}_i;\mathbf{w},\alpha)) \tag{4.10}$$

Which can be written:

$$L_c(\mathbf{s};\mathbf{w},\theta) = \sum_k s_k \ log\ \pi_k - \frac{1}{2}\sum_{i,j,k,l} s_{ik}w_{jl}\left(log\ \sigma_{kl}^2 + \frac{1}{\sigma_{kl}^2}(x_{ij}-\mu_{kl})^2\right) \tag{4.11}$$

Where $s_k = \sum_i s_{ik}$.

In [Govaert and Nadif, 2009], the authors used the generalized EM (GEM) algorithm to maximize the likelihood of the observed data in order to estimate the parameters of the model. From an initial position $(w(0), \theta(0))$, the different steps of this EM algorithm are described as follows:

- Step E

This step is reduced to the calculation of conditional a posteriori probabilities, s_{ik}^c

$$s_{ik}^{(c)} = \frac{\pi_k^c \varphi_k(\mathbf{x}_i;\mathbf{w}^{(c)},\alpha^{(c)})}{\sum_{k'} \pi_{k'}^{(c)} \varphi_{k'}(\mathbf{x}_i;\mathbf{w}^{(c)},\alpha^{(c)})} \tag{4.12}$$

These conditional probabilities can be written $s_{ik} = \frac{e^{s_{ik}}}{\sum_{k'} e^{s_{ik'}}}$ where

$$S_{ik} = log(\pi_k \varphi_k(\mathbf{x}_i;\mathbf{w},\alpha)) \tag{4.13}$$

After some algebraic calculations, the term S_{ik} takes the following form:

$$log\pi_k - \frac{1}{2}\sum_l \left(w_l\ log\ \sigma_{kl}^2 + \frac{1}{\sigma_{kl}^2}(e_{il} + w_l(u_{il}-\mu_{kl})^2\right) \tag{4.14}$$

With:
$u_{il} = \frac{\sum_j w_{jl}\ x_{ij}}{w_l}$ et $e_{il} = \sum_j w_{jl}(x_{ij}-u_{il})^2$, easier to calculate than initial probability s_{ik}

- Step M

In this algorithm, we use a Generalized EM algorithm, GEM [Dempster et al., 1977b] to increase the quantity $Q(\theta,\theta^{(c)})$. Knowing that conditional expectation $Q(\theta,\theta^{(c)})$ can also be expressed as the fuzzy classifying log-likelihood $L_c(s^{(c)},w,\theta)$, this function Q can also be written as:

$$\sum_k s_k^{(c)} log\ \pi_k - \frac{1}{2}\sum_{i,j,k,l} s_{ik}^{(c)} w_{jl}\left(log\ \sigma_{kl}^2 + \frac{1}{\sigma_{kl}^2}(x_{ij}-\mu_{kl})^2\right) \tag{4.15}$$

To increase Q, the authors propose iterating until the convergence of the two following steps: maximization of $Q(\theta,\theta^{(c)})$ with regards to $\mathbf{w}$ while s and $\theta(c)$ are

fixed then maximization of $Q(\theta, \theta^{(c)})$ with regards to θ while $\mathbf{w}$ and s are fixed.
Calculation of $\mathbf{w}$:
This step is to maximize $Q(\theta, \theta^{(c)})$ with regards to $\mathbf{w}$. The expression 4.15 of $L_c(\mathbf{s}^{(c)}, \mathbf{w}, \theta)$ can be written as:

$$\sum_k s_k^{(c)} log\ \pi_k + \sum_{j,l} w_{jl}\ T_{jl}^{(c)} \tag{4.16}$$

Where: $T_{jl}^{(c)} = -\frac{1}{2}\sum_{i,k} s_{ik}^{(c)} \left(log\ \sigma_{kl}^2 + \frac{1}{\sigma_{kl}^2}(x_{ij} - \mu_{kl}^2) \right)$. The variable j belongs to the maximizing class $T_{jl}^{(c)}$:

$$w_{jl}^{(c)} = \begin{cases} 1 & \text{si } l = \arg\max_{l'=1,\dots,m} T_{jl'}^{(c)}; \\ 0 & \text{sinon.} \end{cases}$$

As for the calculation of S_{ik}, The authors have shown that the term T_{jl} takes the following form:

$$-\frac{1}{2}\sum_k \left(s_k^{(c)} log\ \sigma_{kl}^2 + \frac{1}{\sigma_{kl}^2}(f_{jk} + s_k(v_{kj} - \mu_{kl})^2 \right). \tag{4.17}$$

Where:
$v_{kj} = \frac{\sum_i s_{ik}\ x_{ij}}{s_k}$ et $f_{jk} = \sum_i s_{ik}(x_{ij} - v_{jk})^2$
Calculation of α from $\mathbf{w}$ and $\mathbf{s}$

This step is to maximize $Q(\theta|\theta^{(c)})$ with regards to π and $\alpha = (\mu_{11}, \dots, \mu_{gm}, \sigma_{11}^2, \dots, \sigma_{gm}^2)$. By writing the classifying log-likelihood in the form:

$$L_c(\mathbf{s}, \mathbf{w}, \theta) = \sum_k\ s_k log\ \pi_k - \frac{1}{2}\sum_{k,l} \left(s_k w_j log\ \sigma_{kl}^2 + \frac{1}{\sigma_{kl}^2}\sum_{i,j} s_{ik} w_{jl}(x_{ij} - \mu_{kl})^2 \right) \tag{4.18}$$

Then: $\pi_k^{(c+1)} = \frac{s_k^{(c)}}{n}$, $\mu_{kl}^{(c+1)} = \frac{\sum_{ij} s_{ik}^{(c)}\ w_{jl}^{(c)} x_{ij}}{s_k^{(c)}\ w_l^{(c)}}$ and $(\sigma_{kl}^2)^{(c+1)} = \frac{\sum_{ij} s_{ik}^{(c)}\ w_{jl}^{(c)} (x_{ij} - \mu_{kl})^2}{s_k^{(c)}\ w_l^{(c)}}$
These calculations can be optimized by using the previously defined v_{jk} and f_{jk} values, which accelerates this step. The center and the variance of each block are:
$\mu_{kl}^{(c+1)} = \frac{\sum_j w_{jl}^{(c)} v_{jk}}{s_k^{(c)}\ w_l^{(c)}}$ et $(\sigma_{kl}^2)^{(c+1)} = \frac{\sum_j w_{jl}^{(c)} \left(f_{jk} + s_k^{(c)}(v_{jk} - \mu_{kl})^2 \right)}{s_k^{(c)}\ w_l^{(c)}}$

Experimentation	Experimenter
Test 1	Experimenter 1
Test 2	Experimenter 2
Test 3	Experimenter 3
Test 4	Experimenter 1

TABLE 4.1: Example of a table containing qualitative variables.

4.4 Topological methodes

The bi-partitioning methods using the self-organizing maps (SOM) ([Kohonen et al., 2001b]) have been defined by several authors (DCC [Busygin et al., 2002], KDISJ [Cottrell et al., 2004], BCDSM [Benabdeslem and Allab, 2012], etc.). This type of methods falls within the category of partitioning-based approaches because they often use simple clustering algorithms applied separately on the rows and columns of a data matrix.

Stanislav et al. [Busygin et al., 2002] proposed the Double Conjugated Clustering (DCC) approach, which allows all rows and all columns to be partitioned using self-organizing maps. The basic principle of this approach and that of linking the two partitions through a bijection associating to each referent of one of the two spaces a referent of the other space called "conjugate". This method has the advantage of relatively rapid convergence and leads to the construction of two partitions, one in the space of the lines and the other in the space of the columns. Each of these partitions is the conjugate of the other.

One of the algorithms that we find frequently in the literature is that introduced by Corttell, called KDISJ [Cottrell et al., 2004]. KDISJ is a variant of topological maps for the processing of qualitative variables in a data table.

KDISJ Algorithm

We recall that a complete disjunctive table consists of coding qualitative variables with the code 1 for the observed modality and 0 for all the other modalities. The complete disjunctive coding thus makes it possible to transform qualitative variables into variables of quantitative type between which it is permissible to calculate correlations.

Experimentation	Experimenter 1	Experimenter 2	Experimenter 3
Test 1	1	0	0
Test 2	0	1	0
Test 3	0	0	1
Test 4	1	0	0

TABLE 4.2: Example of a complete disjunctive table.

KDISJ (Kohonen for Disjonctive Table) [Cottrell et al., 2004] makes it possible to classify simultaneously the observations and the qualitative variables that describe them. Let $\mathcal{A}$ be a data matrix and d_{ij} be the general term of this matrix which can be considered as a contingency table crossing the variable "observation" to N modalities, and the variable 'modalities' to M modalities. The term d_{ij} takes its values in $\{0, 1\}$. The distance χ^2 is used on rows and columns. Then, the modalities are weighted to correct the complete disjunctive table in the following way:

$$d_{ij}^c = \frac{d_{ij}}{\sqrt{d_i.d_j}} \tag{4.19}$$

Where: $d_i = \sum_{j=1}^{M} d_{ij}$ et $d_j = \sum_{i=1}^{N} d_{ij}$

In the case of a complete disjunctive array, d_i is k, whatever i. The term d_j is the number of the modality j. The corrected table is denoted by D^c (corrected disjunctive table). After this transformation, it is possible to use the Euclidean distance on D^c which is equivalent to χ^2 weighted on D. These corrections are equivalent to those used traditionally in the correspondence analysis, which in fact amounts to a weighted principal components, using the simultaneous χ^2 distance on rows and columns. The transition to topological maps is done by using the classical architecture of the SOM model by associating with each referent $\mathbf{w}$ a reference vector C_w formed of $(M + N)$ components, the first M evolve in The space of observations (represented by the lines of D^c), the last N in modal space (represented by the D^c columns).

The notation:

$$C_w = (C_M + C_N)_w = (C_{M,w} + C_{N,x}) \tag{4.20}$$

makes it possible to highlight the structure of the reference vector C_w. The learning steps of the topological map are double. A line of D^c (ie an observation i), then a column (that is, a modality j) are drawn alternately. When an observation i is

pulled, the modality j (i) associated with it. It is defined by:

$$j(i) = \arg\max_{j} \; d_{ij}^{c} \tag{4.21}$$

which maximizes the coefficient d_{ij}^{c}, that is to say the rarest modality in the total population among the modalities corresponding to it. Then, an extended observation vector

$$X = (i, j(i)) = (XM, XN)$$

of dimension $(M + N)$ is created. Then, the procedure searches among the codevectors which is closest, in the sense of the Euclidean distance restricted to the first M components.

Let w_0 be the winning referent. The minimization step is formulated as follows:

$$\begin{cases} w_0 = \arg\min\limits_{w} \; \|X_M - C_{M,w}\| \\ C_w^{(t)} = C_w^{(t-1)} + \varepsilon \; \mathcal{K}(w, w_0)(X - C_w^{(t-1)}) \; . \end{cases}$$

Where ε is the learning step and $\mathcal{K}$ is the neighborhood radius of the map.

When a modality j of dimension N (a column of D^c) is picked, the algorithm of [Cottrell et al., 2004] searches among the codevectors which is closest in the sense of Euclidean distance restricted to the last N components. Let z_0 be the winning unit. The procedure approximates the previous N components of the winning vector-code associated with z_0 and its neighbors with those of the mode vector j, without modifying the first M components. Let Y The column vector of dimension N corresponding to the modality j. This step can be written:

$$\begin{cases} z_0 = \arg\min\limits_{w} \; \|Y - C_{N,w}\| \\ C_{N,u}^{(t)} = C_{N,u}^{(t-1)} + \varepsilon \; \mathcal{K}(w, w_0)(Y - C_{N,u}^{(t-1)}) \; . \end{cases}$$

After convergence, the observations and the modalities are classified in the classes of the obtained map. observations or "close" modalities are classified in the same class or in neighboring classes. The algorithm thus defined is called KDISJ.

4.5 Divisive methods

The basic strategy of this type of method is the iterative division of the database, which makes it possible to find the data blocks which optimize certain criteria. Instead of proposing only a partition in rows and a partition in columns, this type of method proposes a division into homogeneous blocks of the data. One of the oldest and most used algorithms is One-way Splitting [Hartigan]. It is part of the "divisive" algorithms and allows to divide a matrix of data into several sub-matrices corresponding to blocks. The basic principle of this method is to perform permutations of rows and columns in order to define the block structure.

The basic idea of the algorithm is to use only variables with a variance greater than the threshold in a given class to split this class. Let $A(I, J)$ be a data matrix with $1 \leq I \leq N, 1 \leq J \leq d$. The classes in rows $1, 2, \ldots, K$ are constructed so that the I class is determined by the classes that divide it $Min(I)$ and $Max(I)$. For a minimal class (which can no longer be divided), these values represent the first and last rows of the I class. At the end of the algorithm, $V(I)$ is defined as the set of variables that have a variance lower than the threshold in I, and in no other larger class.

The algorithm proceeds by division of successive classes. In the p step, there are p classes $I(1), I(2), \ldots, I(p)$ separating the rows, which are the minimum classes in the set $1, 2, 3, \ldots, 2p-1$. $V[I(J)]$ represents the set of variables with a variance greater than the threshold T for any larger class [Jollois, 2003]. Splitting in half is done on the $I(J)$ classes, using only the variables in $V[I(J)]$ which have a variance greater than the threshold. The two new classes $2p$ and $2p+1$ will have $V(2p) = V(2p+1)$ defined as the set of variables of $I(J)$ which have a variance greater than T in $I(J)$. Therefore, $V[I(J)]$ will be changed to the set of variables of $V[I(J)]$ which will therefore have a variance less than T in $I(J)$. Cutting stops when all $V[I(J)]$ sets contain variables with a variance lower than the threshold in $I(J)$.

4.6 Hierarchical methods

We find in the literature several bi-partitioning approaches that use hierarchical algorithms. We cite the work of [Caldas and Kaski, 2011] [Mao et al., 2005] and [Getz et al., 2000a]. One of the most widely used approaches in this family of

models is CTWC (Coupled Two-Way Clustering) [Getz et al., 2000a]. CTWC is to apply an algorithm of hierarchical classification, the SPC "Super Paramagnetic Clustering" [Getz et al., 2000b], on the columns using all the rows and then on the rows using all the columns. All sub-matrices (I, J), knowing that I is a class on the rows and J a class on the columns are computed. Only sub-matrices that satisfy a certain criterion such as stability or a minimum size are retained [Charrad et al., 2008]. Then, the process is reiterated: row and column classes are extracted from these submatrices. CTWC operates on the set of subsets of the v and on the subsets of the $\{u\}$ variables. Initially, $\{v\} = \{V\}$ and $\{v\} = \{U\}$, the algorithm iteratively selects a subset of $\{V\}^{'} \in v$, and a subset of variables $\{U\}^{'} \in u$; Then the SPC algorithm is applied to $\{V\}^{'}$ and $\{U\}^{'}$. The corresponding algorithm is described as follows:

Algorithm 9 : CTWC Algorithm

Inputs: $\mathcal{A}$: data matrix.
Outputs: the partitions of the observations v and the partitions of the variables u.
Initialization phase:

- $v_1 = \{V\}$, $u_1 = \{U\}$, $v = \oslash$, $u = \oslash$.
- Initialize the hierarchical array H_v to save the clusters of observations.
- Initialize the H_u hierarchical array for saving variable clusters.

While $(u_1 \neq \oslash$ *or* $v_1 \neq \oslash)$ *do For* $(U^{'}, V^{'}) \in (u_1 \times v_1) \cup (u_1 \times v) \cup (u \times_1)$ *do*
Apply the SPC algorithm $(E_{U^{'}V^{'}})$ for the clustering of observations $V^{'}$

- Add to all stable observations v_2
- $H_V[V^{''}] = U^{'}$ for all new clusters $V^{''}$

Apply the SPC algorithm $(E_{U^{'}V^{'}})$ for the clustering of observations $U^{'}$

- Add to all stable observations u_2
- $H_U[V^{''}] = V^{'}$ for all new clusters $U^{''}$

$u = u \cup u_1$, $v = v \cup v_1$
$u_1 = u_1$, $v_2 = v_2$
Return u, v and their hierarchy H_U, H_V.

4.7 Constructive methodes

In this type of approaches, the data blocks are constructed in different ways [Charrad et al., 2008]. For example: by adding and removing rows and columns (δ-biclusters [Cheng and Church, 2000]), by row and column permutation (OPSM [Ben-Dor et al., 2003]), by estimating the parameters of the models (Plaid models [Lazzeroni and Owen, 2000]) or from a bipartite graph (SAMBA [Tanay et al., 2002]), and so on.

The SAMBA algorithm proposed by Tanay et al. [Tanay et al., 2002] represents a matrix of data by a weighted biparti G graph where each node n_i corresponds to a row and each node n_j corresponds to a column. The edge between the node n_i and the node n_j has a weight a_i^j corresponding to the element of the matrix located at the intersection of the line i and The column j. A biclass corresponds to the subgraph (H, J, E) of G and represents a subset I of observations whose value changes significantly under a set of variables J. The objective of the SAMBA algorithm is to look for maximum biclasses in the data. The application of the SAMBA algorithm is carried out in two steps:

1. The data are normalized and represented by a bipartite graph,
2. The algorithm identifies the maximal k bi-cliques.

In a later phase, SAMBA brings local improvements to the biclasses by adding or removing the vertices, and selects the similar biclasses having a large number of vertices in common.

The δ-biclusters approach [Cheng and Church, 2000] is a constructive bi-clustering method. The principle of the δ-biclusters algorithm consists of iteratively deleting rows and columns from the initial matrix until the distance measurement is less than a certain threshold, then adding rows and columns Iteratively without causing an increase in this distance measurement [Charrad et al., 2008]. At each iteration, a biclass is generated and replaced in the initial matrix by random values. One limitation of this approach is that the number of biclasses to be searched must be set by the user just as the δ threshold used for quality measurement. In addition, the quality of the biclasses decreases with each iteration because of the random values added to each iteration.

The authors of [Ben-Dor et al., 2003] defined a block as a submatrix preserving the order of the data. Unlike the [Lazzeroni and Owen, 2000] parameter estimation

methods where the uniformity of the data in the data matrix is considered, they focus instead on the relative order of the columns in the blocks. The objective of OPSM is the identification of large blocks. A submatrix is preservative of the order if there is a permutation of the columns making it possible to have strictly increasing values on each line.

4.8 Matrix decomposition for bi-clustering

Recently, new bi-partitioning approaches based on matrix factorization are proposed ([Long et al., 2005], [Yoo and Choi, 2010], [Labiod and Nadif, 2011], [Shang et al., 2012]). In this type of approach, the bi-partitioning problem can be seen as a matrix approximation problem where the objective is to minimize the approximation error between the data of the original matrix $\mathcal{A}$ And the reconstructed matrix on the basis of the class structures. Given a non-negative matrix $\mathcal{A}$, the general strategy of a bi-partitioning approach in this context is to find a decomposition of $\mathcal{A}$ in the form of three matrices $\mathbf{ZGW}^T$. The matrix $\mathbf{Z}$ represents the partitioning of the $\mathcal{A}$ rows, the $\mathbf{W}$ matrix represents the partitioning of the $\mathcal{A}$ columns and the $\mathbf{G}$ is an intermediate matrix. Most of the algorithms proposed in this sense are iterative. Only the rules for updating the three matrices (chosen optimization method or constraints imposed on the three matrices) may be different.

CUNMTF Algorithm

The Co-clustering Under Nonnegative Matrix Tri-Factorization (CUNMTF) approach by Labiod and Nadif [Labiod and Nadif, 2011] proposes a new formulation of the NMF [Lee and Seung, 1999] To bi-partitioning. The authors propose two approaches that optimize a relaxed formulation of the double K-means criterion in an NMF style. The first is called DNMF and the second ODNMF when the orthogonality constraints on $\mathbf{Z}$ and $\mathbf{Z}$ are considered.

The main idea of this approach is that the latent block structure in a non-negative rectangular data matrix is factored into two factors rather than three: the matrix of the coefficients of the lines $\mathbf{R}$ and the matrix of the coefficients of columns $\mathbf{C}$, which indicate respectively the degree of membership of a row and a column to a cluster. The authors first propose a formulation of the double K-means model, which is called DNMF (Double Nonnegative Matrix Factorization).

Given a matrix $\mathcal{A} = (a_i^j) \in \mathcal{R}^{N\times d}$, the goal of the double K-means is to find simultaneously A partition in K classes $P = \{P_1, \ldots, P_K\}$ of the set of $I = \{1, \ldots, N\}$ rows and a partition $Q = \{Q_1, \ldots, Q_L\}$ into L classes of the set of columns $J = \{1, \ldots, d\}$. The two partitions P and Q naturally induce respectively the classification matrices $Z = (z_i^k) \in \{0,1\}^{N\times K}$ and $W = (w_j^l) \in \{0,1\}^{d\times L}$; $z_i^k = 1$ (resp. $w_j^l = 1$), if the line $\mathbf{a}_i \in P_k$ (resp. the column $\mathbf{a}^j \in Q_l$), and 0 otherwise.

The reorganization of the rows and columns following P and Q reveals a homogeneous block structure. Each block $\mathcal{A}_k^l$ is therefore defined by $\{(a_i^j)|z_i^k w_j^l a_i^j = 1\}$. On the other hand, $G = (g_k^l) \in \mathcal{R}^{K\times L}$ is the small representative of $\mathcal{A}$ (g_k^l is the centroid of $\mathcal{A}_k^l$).

The detection of the homogeneous blocks in $\mathcal{A}$ can be obtained by searching for the three matrices Z, W and G by minimizing

$$\mathcal{J}(\mathcal{A}, ZGW^T) = ||\mathcal{A} - ZGW^T||^2$$

The term ZGW^T characterizes the information of $\mathcal{A}$ which can be described by a class structure.

This matrix formulation can take the following form:

$$\mathcal{J}(\mathcal{A}, ZGW^T) = \sum_{i,j,kl} z_i^k w_j^l (a_i^j - g_k^l)^2$$

with P_k , Q_l fixed, the general term of G is obtained by:

$$g_k^l = \frac{\sum\limits_{i,j,k,l} z_i^k w_j^l a_i^j}{z_k w_l}$$

Où $z_k = |P_k|$; $w_l = |Q_l|$.

In the context of the double K-means, the objective function to minimize is the distance to the square between each row (each column) of the center. Let $D_z^{-1} \in \mathcal{R}^{K\times K}$ and $D_w^{-1} \in \mathcal{R}^{L\times L}$ two diagonal matrices defined by $D_z^{-1} = Diag(z_1^{-1}, \ldots, z_K^{-1})$ and $D_w^{-1} = Diag(w_1^{-1}, \ldots, w_L^{-1})$. Using the matrices D_z, D_w, $\mathcal{A}$, Z and W, the representation matrix G is written: $G = D_z^{-1} Z^T \mathcal{A} W D_w^{-1}$. If G is integrated in the objective function $\mathcal{J}(\mathcal{A}, ZGW^T)$, then the expression to optimize becomes $||\mathcal{A} - \mathbf{Z}\mathbf{Z}^T\mathcal{A}\mathbf{W}\mathbf{W}^T||^2$, where $\mathbf{Z} = ZD_z^{-0.5}$ and $\mathbf{W} = WD_w^{-0.5}$. The authors assert that this formulation is valid even if $\mathcal{A}$ is not nonnegative, and the approximation $\mathbf{Z}\mathbf{Z}^T\mathcal{A}\mathbf{W}\mathbf{W}^T$ of $\mathcal{A}$ is formed by the same value in each block

$\mathcal{A}_k^l$. More precisely, the matrix $\mathbf{Z}^T\mathcal{A}\mathbf{W}$ acts as a summary of $\mathcal{A}$, and absorbs scale differences $\mathcal{A}$, $\mathbf{Z}$ and $\mathbf{W}$. The matrices $\mathbf{Z}\mathbf{Z}^T\mathcal{A}, \mathcal{A}\mathbf{W}\mathbf{W}^T$ give respectively the vectors of the averages of the row and column classes.

Then, the authors define the CUNMTF model by introducing the objective function:

$$arg \min_{\mathbf{Z},\mathbf{W}\geq 0} ||\mathcal{A} - \mathbf{Z}\mathbf{Z}^T\mathcal{A}\mathbf{W}\mathbf{W}^T||^2$$

and taking into account the constraint of non-negativity. In order to optimize this objective function, the authors use the Karush-Kuhn-Tucker [Boyd and Vandenberghe, 2004] conditions by introducing the Lagrange function:

$$\mathcal{L} = ||\mathcal{A} - \mathbf{Z}\mathbf{Z}^T\mathcal{A}\mathbf{W}\mathbf{W}^T||^2 - Trace(\Lambda\mathbf{Z}^\mathbf{T}) - Trace(\Gamma\mathbf{W}^\mathbf{T})$$

where the matrices Λ and Γ are the Lagrange multipliers introduced to impose the constraint of non-negativity respectively on $\mathbf{Z}$ and $\mathbf{W}$. Let, $X_W = \mathcal{A}\mathbf{W}\mathbf{W}^T$ and $X_Z = \mathbf{Z}\mathbf{Z}^T\mathcal{A}$. This leads to the following update rules:

$$\mathbf{Z} \leftarrow \mathbf{Z} \odot \frac{2\mathcal{A}X_\mathbf{W}^T\mathbf{Z}}{\mathbf{Z}\mathbf{Z}^T X_\mathbf{W} X_\mathbf{W}^T\mathbf{Z} + X_\mathbf{W} X_\mathbf{W}^T\mathbf{Z}\mathbf{Z}^\mathbf{T}\mathbf{Z}} \tag{4.22}$$

$$\mathbf{W} \leftarrow \mathbf{W} \odot \frac{2\mathcal{A}X_\mathbf{Z}^T\mathcal{A}\mathbf{W}}{\mathbf{W}\mathbf{W}^T X_\mathbf{Z} X_\mathbf{Z}^T\mathbf{W} + X_\mathbf{Z} X_\mathbf{Z}^T\mathbf{W}\mathbf{W}^\mathbf{T}\mathbf{W}} \tag{4.23}$$

The authors then propose an algorithm for calculating non-negative relaxation. The algorithm contains the classical steps of the NMF approach. The estimation obtained by this algorithm is improved iteratively by updating the factors with the rules 4.22 and 4.23. To derive the update multiplicative rules under the orthogonality constraints on $\mathbf{Z}$ and $\mathbf{W}$, authors calculate the "natural gradient" on the varieties of Stiefel [Freitas, 1985]. The update rules are therefore:

$$\mathbf{Z} \leftarrow \mathbf{Z} \odot \frac{\mathcal{A}\mathbf{W}\mathbf{W}^T\mathcal{A}^T\mathbf{Z}}{\mathbf{Z}\mathbf{Z}^T\mathcal{A}\mathbf{W}\mathbf{W}^T\mathcal{A}^T\mathbf{Z}}$$

$$\mathbf{W} \leftarrow \mathbf{W} \odot \frac{\mathcal{A}\mathbf{Z}\mathbf{Z}^T\mathcal{A}\mathbf{W}}{\mathbf{W}\mathbf{W}^T\mathcal{A}^T\mathbf{Z}\mathbf{Z}^T\mathcal{A}\mathbf{W}}$$

Long et al. [Long et al., 2005] proposed the non-negative Block Value Decomposition (NBVD) approach of $\mathcal{A}$ based on an iterative alternate least squares optimization procedure. At the convergence, $\mathbf{Z}\mathcal{A}$ is normalized to $\mathbf{Z}\mathcal{A}\mathbf{X}$ ($\mathbf{X}$ is a diagonal matrix) The labels of the column classes are deduced from $\mathbf{X}\mathbf{X}^{-1}\mathbf{W}^T$.

The labels of the line classes are deduced by working on $\mathcal{A}^T$.

4.9 Conclusion

We have presented in this chapter some bi-partitioning approaches that are unsupervised classification techniques. The bi-clustering problem consists of partitioning the rows and columns of a database at the same time. These algorithms are categorized according to the nature of their underlying clustering approach, including partitioning, probabilistic, topological, hierarchical, divisive, and constructive bi-clustering methods. However, all the presented methods do not scale-up since they are based on non-scalable traditional clustering algorithms.

Chapter 5

SOM Clustering using Spark-MapReduce

This chapter presents our first novel contribution, concerned with scaling-up the SOM approach using the MapReduce paradigm. For self-containedness, this chapter begins with a description of the SOM algorithm. Afterwards, the two versions of the SOM MapReduce clustering algorithm are presented. After that, the quality of the proposed method is evaluated in terms of various performance criteria on real-world datasets.

5.1 Introduction

Data clustering is a principal task in a variety of areas: machine learning, data mining, pattern recognition, social network. Consequently, there is a vast amount of research focused on the topic [Jain et al., 1999, Charikar et al., 1997, Matthew Mccutchen and Khuller, 2008]. It is difficult to store and analyse a large volume of datas on a single machine with a sequential algorithm. Thus numerous successful, subspace clustering algorithms or clustering ensemble are proposed to deal with high and large dataset [Parsons et al., 2004, Kriegel et al., 2009].

However, the existing algorithms have difficult to deal with Terabytes and Petabytes of data. Clustering problems have numerous applications and are becoming more challenging as the size of the data increases. Nevertheless, good clustering algorithms are still extremely valuable, because we can (and should) rewrite them for parallel clustering using a new Mapr-Reduce paradigm [Lv et al.,

2010, Lin et al., 2011].

In situations where the amount of data is prohibitively large, the MapReduce (MR) programming paradigm is used to overcome this problem [Dean and Ghemawat, 2008b]. Thus, an increasing number of programmers have migrated to the MapReduce programming model [Ene et al., 2011b, Sul and Tovchigrechko, 2011, Ferreira Cordeiro et al., 2011, Ghoting et al., 2011]. The MR programming model was designed to simplify the processing of large files on a parallel system through user-defined Map and Reduce functions [Karloff et al., 2010]. A MR function consists of two phases: a *Map* phase and a *Reduce* phase. During the Map phase, the user-defined

Map primitive transforms the input data into (key, value) pairs in parallel. These pairs are stored and then sorted by the system so as to accumulate all values for each key. During the Reduce phase, the user-defined Reduce primitive is invoked on each unique key with a list of all the values for that key; usually, this phase is used to perform aggregations. Finally, the results are output in the form of (key, value) pairs. Each key can be processed in parallel during the Reduce phase. Hadoop[1], an open-source implementation of the MR programming model, has emerged as a popular platform for parallelization.

A user can perform parallel computations by submitting MR jobs to Hadoop. While the Hadoop are very popular in their particular domains, we believe that they have a set of limitations that make them ill-suited to the implementation of parallel clustering algorithms. Many common clustering algorithms apply a primitives repeatedly to the same dataset to optimize a parameter. Thus the Map/Reduce primitives need to reload the data, incurring a significant performance penalty.

In this chapter, we are concerned with designing clustering algorithm named Self-organizing Map (SOM, [Kohonen, 2001]) using MapReduce. We use another emerged open-source implementation named Spark[2] [Zaharia et al., 2010b], which is adapted to machine learning algorithms and supports applications with working sets while providing similar scalability and fault tolerance properties to MapReduce. The purpose in this work is not to present a new SOM algorithm, but a new way of writing using the MapReduce paradigm. The major research challenge addressed is how to minimize the input and output of primitives (map and reduce) for topological clustering algorithm. So, we show that we can save computation

[1] www.hadoop.com
[2] http://spark-project.org/

time by changing the (key, value) parameters. We design a complete distributed SOM clustering solution using Spark and Map-Reduce paradigm.

The rest of the chapter is organized as follows: Section 5.2 we provide the self-organizing maps batch algorithm. In Section 5.3, we propose our SOM MapReduce using Spark open source platform. Section 5.4 provides the experimental results and shows the comparisons between two manners to design MapReduce function. Finally, Section 5.5 concludes and provides some future research.

5.2 Self-Organizing Maps (SOM)

Self-organizing maps are increasingly used as tools for visualization, as they allow projection in small spaces that are generally two dimensional. The basic model proposed by Kohonen consists on a discrete set $\mathcal{C}$ of cells called map. The size of the grid $\mathcal{C}$ is denoted by k and must be provided a priori. A variety of self-organizing models is derived from the first original model proposed by Kohonen [Kohonen, 2001, Varsta et al., 2001]. All models are different from each other but share the same idea: depict large data-sets on a simple geometric relationship projected on a reduced topology (1D or 2D). This grid has topological order of k cells. Each cell c has its own cluster denoted Cl_c.

Self-organizing process requires neighbourhood functions to preserve topological relationships between cells. Hence the neighbourhood functions are needed to update prototypes. For each pair of cell c and r on the map, their mutual influence is defined by the function

$$\mathcal{K}^T(\delta(c,r)) = \exp(\frac{-\delta(c,r)}{T})$$

where T represents the temperature which decreases the value of T between two values T_{max} and T_{min}, to control the size of the neighborhood influencing a given cell on the map :

$$T = T_{max}\left(\frac{T_{min}}{T_{max}}\right)^{\frac{t}{t_f-1}} \tag{5.1}$$

t_f is the number of iteration, and $\delta(c,r)$ is defined as the shortest distance between r and c on the grid $\mathcal{W}$. We associate to cluster Cl_c a prototype denoted $\mathbf{w}_c = (w_c^1, w_c^2, ..., w_c^d)$. The cost function of self-organizing tree is expressed as:

$$\mathcal{R}(\phi, \mathcal{W}) = \sum_{\mathbf{x}_i \in \mathcal{A}} \sum_{r=1}^{k} \mathcal{K}^T(\delta(\phi(\mathbf{x}_i), r)) ||\mathbf{x}_i - \mathbf{w}_r||^2 \tag{5.2}$$

where $\mathcal{W} = \cup_{r=1}^{k} \mathbf{w}_r$, ϕ is the assignment function.

Minimizing cost function $\mathcal{R}(\phi, \mathcal{W})$ is a combinatorial optimization problem. In this work we propose to minimize the cost function in the same way as "batch" version performing two steps until stabilization.

1. Minimize $\mathcal{R}(\phi, \mathcal{W})$ with respect to ϕ by fixing $\mathcal{W}$. The expression is defined as follows:

$$\phi(\mathbf{x}_i) = \arg\min_r \|\mathbf{x}_i - \mathbf{w}_r\|^2 \tag{5.3}$$

2. Minimize $\mathcal{R}(\phi, \mathcal{W})$ with respect to $\mathcal{W}$ by fixing ϕ.

$$\mathbf{w}_c = \frac{\sum_{r \in C} \mathcal{K}^T(\delta(c, r)) \sum_{\mathbf{x}_i \in Cl_r} \mathbf{x}_i}{\sum_{r \in C} \mathcal{K}^T(\delta(c, r)) |Cl_r|} \tag{5.4}$$

   where $|Cl_r|$ the data size assigned to each cell r.

The learning algorithm described above allows us to estimate the parameters maximizing the cost function for a fixed neighborhood T. The outline of the algorithm is presented in algorithm 10. This batch approach offers advantage that separates the update expression of prototype (eq. 5.4) into sums (numerator and denominator) that allow parallelization using MapReduce.

Algorithm 10 Outline of SOM batch algorithm

Ensure:
weight vectors initialized
while $t \leq t_f$ **do**
  $t+ = 1$
  for all $\mathbf{x}_i \in \mathcal{A}$ **do**
    compute winning vector according to eq. 5.3
  end for
  for c = 1 :k **do**
    update weight vector $\mathbf{w}_c$ according to eq. 5.4
  end for
  Update the temperature according to eq. 5.1
end while

5.3 Spark-MapReduce and SOM

The increase of data mining on BigData has resulted in the creation of a lot of new parallel programming models like MapReduce, Pregel, and PowerGraph [Malewicz et al., 2010, Low et al., 2012]. To handle this huge amount of data, it is necessary to use distributed architecture. This is not a simple task and several difficulties have to be dealt with, including loading data, failure safety, and algorithm design. The MapReduce implementation on Spark takes care of failure-correction, data management and distribution.

It has become very important in MapReduce to decompose our problem in elementary function. The complexity of writing a MapReduce algorithm is to split the algorithm into atomic parts. Those parts are assigned to Map and Reduce phase. Knowing that $Cl_r = \{\mathbf{x}_i, \phi(\mathbf{x}_i) = r\}$, we can rewrite the quantization phase (eq. 5.4) of SOM algorithm as :

$$\mathbf{w}_c = \frac{\sum_{\mathbf{x}_i \in \mathcal{A}} \mathcal{K}^T(\delta(c, \phi(\mathbf{x}_i))\mathbf{x}_i}{\sum_{\mathbf{x}_i \in \mathcal{A}} \mathcal{K}^T(\delta(c, \phi(\mathbf{x}_i)))} \tag{5.5}$$

In the case of SOM algorithm we identified theses atomic parts:

- Assign each observation $\mathbf{x}_i$ to the best match unit using expression 5.3.
- Accumulate denominator and numerator for each cell $c \in C$
- Update weight vectors $\mathbf{w}_c$ (eq. 5.5)

Hence we can propose two versions of the MapReduce steps. The first one is adopted in literature and the second is our proposition. The details are provided below.

5.3.1 Version 1 of SOM MapReduce

The first version of SOM MapReduce is inspired by K-means MapReduce algorithm [Lv et al., 2010]. It's easy to decompose SOM into MapReduce functions:

- The Map function has an input data vector and computes the best match units and the neighborhood factor for each prototypes $\mathcal{K}^T(\delta(c, \phi(\mathbf{x}_i)))$. The

size of the outputs is equal to the size of the prototypes of the model. The key of the output of the function is the *id* of the winning prototype. The values are the data vector $\mathbf{x}_i$ multiplied by the neighborhood factor $\mathcal{K}^T(\delta(c, \phi(\mathbf{x}_i)))$.

- The Reduce function accumulates each data vector assigned to each prototype and counts them. The new prototype vector is equal to the accumulation divided by the denominator. The different functions are defined as follows :

$$\begin{aligned} MapNumerator(\mathbf{x}_i, c) &= \mathcal{K}^T(\delta(c, \phi(\mathbf{x}_i)))\mathbf{x}_i \\ MapDenom(\mathbf{x}_i, c) &= \mathcal{K}^T(\delta(c, \phi(\mathbf{x}_i))) \\ Reduce(c) &= \frac{\sum_{\mathbf{x}_i \in \mathcal{A}} MapNumerator(\mathbf{x}_i, c)}{\sum_{\mathbf{x}_i \in \mathcal{A}} MapDenom(\mathbf{x}_i, c)} \end{aligned}$$

The batch SOM MapReduce algorithm is shown in algorithm 11.

5.3.2 Version 2 of SOM MapReduce

The main drawback of the first version is the number of outputs. Indeed the number of map outputs is the number of observations multiplied by the number of prototypes $n \times k$. In the second version, map outputs are merged in one value, so the key of the output is not used.

The Map value of the output is a matrix and a neighborhood vector. The matrix is constituted by rows of data vectors $\mathbf{x}_i$ who are themselves multiplied by the neighborhood factors $\mathcal{K}^T(\delta(c, \phi(\mathbf{x}_i)))$. All those neighborhood factors are stored in the neighborhood vector. So the size of the output matrix is the number of prototypes multiplied by the size of the data vectors $(k \times n)$. The size of the neighborhood vector is the number of prototypes.

The reduce function just sums all matrices and all neighborhood vectors together. The new model matrix is computed by dividing the sum of matrices and the sum of the neighborhood vectors.

We denote $\mathcal{H}(k \times n)$ as neighborhood matrix, which elements are defined as follows:

$$\mathcal{H}_{i,j} = \mathcal{K}^T(\delta(i, j)) \tag{5.6}$$

Algorithm 11 SOM MapReduce : version 1

Ensure:

{Random initialization of prototypes}
while $t \leq t_f$ **do**
  $t+ = 1$
  {**MAP :** distributed loop over all input vectors}
  {Compute $\phi(\mathbf{x}_i)$: the best match cell which minimize the distance between its prototype $\mathbf{w}_c$ and the input vector $\mathbf{x}_i$}
  for all $\mathbf{x}_i \in \mathcal{A}$ **do**
    for $c = 1 : k$ **do**
      $D[c] = \|\mathbf{x}_i - \mathbf{w}_c\|^2$
    end for
    {the minimum function provides the index of the minimum value}
    $\phi(\mathbf{x}_i) = min(D)$
    {Compute the numerator and the denominator for each cell c}
    for $c = 1 : k$ **do**
      $MapNumerator_c = \mathcal{K}^T(\delta(c, \phi(\mathbf{x}_i)))\mathbf{x}_i$
      $MapDenom_c = \mathcal{K}^T(\delta(c, \phi(\mathbf{x}_i)))$
    end for
  end for
  {**REDUCE :** distributed sum of all the map outputs (numerator and denominator) for each prototype.}
  for $c = 1 : k$ **do**
    for all $MapOutput_c$ **do**
      $SumNumerator_c+ = MapNumerator_c$
      $SumDenom_c+ = MapDenom_c$
    end for
  end for
  {**UPDATE PROTOTYPES :** Compute new prototypes.}
  for $c = 1 : k$ **do**
    $\mathbf{w}_c = \frac{SumNumerator_c}{SumDenom_c}$
  end for
  Update the temperature according to eq. 5.1
end while

We also consider that $\mathcal{H}_{:,j}$ denotes the column j of the matrix $\mathcal{H}$ and $\mathcal{H}_{i,:}$ denotes the row i of the matrix $\mathcal{H}$.

As the first version, the Reduce function accumulates each data vector assigned to each prototype and counts them. The prototype matrix $\mathcal{W}$ is the accumulation divided by the denominator. Thus Map and Reduce functions are defined as

follows:

$$\begin{aligned} MapNumerator(\mathbf{x}_i) &= \mathcal{H}_{:,\phi(\mathbf{x}_i)} \times \mathbf{x}_i \\ MapDenom(\mathbf{x}_i) &= \mathcal{H}_{:,\phi(\mathbf{x}_i)} \\ \mathbf{R}educe() &= \frac{\sum_{\mathbf{x}_i \in \mathcal{A}} MapNumerator(\mathbf{x}_i)}{\sum_{\mathbf{x}_i \in \mathcal{A}} MapDenom(\mathbf{x}_i)} \end{aligned}$$

The batch SOM MapReduce algorithm is shown in algorithm 12.

Algorithm 12 SOM MapReduce : Version 2

Ensure:

{Random initialization of prototypes}
while $t \leq t_f$ **do**
  $t+ = 1$
  {**MAP :** distributed loop over all input vectors} {Compute $\phi(\mathbf{x}_i)$: the best match cell which minimize the distance between its prototype $\mathbf{w}_c$ and the input vector $\mathbf{x}_i$}
  for all $\mathbf{x}_i \in \mathcal{A}$ **do**
    for $c = 1 : k$ **do**
      $D[c] = \|\mathbf{x}_i - \mathbf{w}_c\|^2$
    end for
    {the minimum function provides the index of the minimum value}
    $\phi(\mathbf{x}_i) = min(D)$
    {Compute the local matrix model numerator}
    $MapNumerator = \mathcal{H}_{:,\phi(\mathbf{x}_i)}\mathbf{x}_i$
    {Compute the local neighborhood factor vector denominator}
    $MapDenom = \mathcal{H}_{:,\phi(\mathbf{x}_i)}$
  end for
  REDUCE : distributed sum of all local matrix models and local neighborhood factor vectors.
  for $c = 1 : k$ **do**
    for all $MapOutput$ **do**
      $SumNumerator+ = MapNumerator$
      $SumDenom+ = MapDenom$
    end for
  end for
  {**UPDATE PROTOTYPES :** Compute new prototypes.}
  $\mathcal{W} = \frac{SumNumerator}{SumDenom}$
  Update the temperature according to eq. 5.1
end while

5.4 Experiments

We implemented our algorithms SOM MapReduce in Spark 7.3 and we compared them on a amazon EC2 cluster of 24 xlarge computers. Each computer has 4 cores and 15GB of RAM, so the total capacity of the cluster is of 96 cores and 360 GB of RAM. The code of the version 2 is available in `https://github.com/TugdualSarazin/spark-clustering`.

Firstly, we will provide the performances of SOM-MapReduce clustering algorithm comparing with SOM serial algorithm based on Matlab toolbox. For most of large datasets, to run a serial algorithm is an impractical task because it would require very long time. Thus secondly we have evaluated its execution time performances and its capacities to scale.

5.4.1 Comparison with SOM not MapReduce algorithm

To evaluate how much MapReduce affects the serial SOM clustering quality [Kohonen, 2001], we used datasets from UCI with known labels [Frank and Asuncion, 2010]. Tables below represents qualities measures of both algorithms (serial SOM and SOM MapReduce). To evaluate the clustering performance, two criterion are used, each of them should be maximized: Accuracy (ACC), Rand measure.

Table 5.1 and table 5.2 depict respectively the ACC and Rand results. We observe that SOM-MapReduce provides equivalent ACC and Rand measure. The objective here is not get better performance than classical SOM clustering approaches, but to show that SOM-MapReduce does not interfere SOM and provides an equivalent performances as a clustering approaches.

5.4.2 Speedup tests

For benchmarking the performance of our MapReduce SOM implementation, we generated 100 millions observations using two gaussian distributions with only two dimensions, only to test the performance. Then, we trained a 10×10 SOM with different core counts.

The figure 5.1 shows the algorithms capacities to scale with respect to the number of computers. We can observe that algorithm 2 is better than the version 1

dataset	SOM-MapReduce	SOM
isolet5	0.435	0.433
Movement Libras	0.453	0.711
Breast	0.970	0.974
Sonar Mines	0.649	0.744
Lung Cancer	0.75	0.906
Spectf 1	0.775	0.716
HorseColic	0.697	0.78
Heart	0.707	0.851
glass	0.664	0.623

TABLE 5.1: Clustering Accuracy performance (Acc)

dataset	SOM-MapReduce	SOM
isolet5	0.920	0.905
Movement Libras	0.907	0.943
Breast	0.551	0.476
Sonar Mines	0.503	0.507
Lung Cancer	0.673	0.425
Spectf 1	0.521	0.403
HorseColic	0.460	0.448
Heart	0.504	0.529
glass	0.740	0.752

TABLE 5.2: Clustering performance comparison using Rand.

whatever the number of computers used. For example the gap between algorithms (version 1 and version 2) is of 7200 seconds using 8 cores and of 400 seconds with 96 cores. For a fixed dataset, speedup captures the decrease in runtime when we increase the number of available cores. The SOM algorithm is linear this means that in a perfect case, the scaling factor is 2 when the number of cores and memory are doubled. In practice with second algorithm (version 2) the scaling factor is 1.7 using 8 to 16 cores and it decrease to 1.14 using 48 to 96 cores. This decrease is explained by the small size of the dataset compared to the high number of cores.

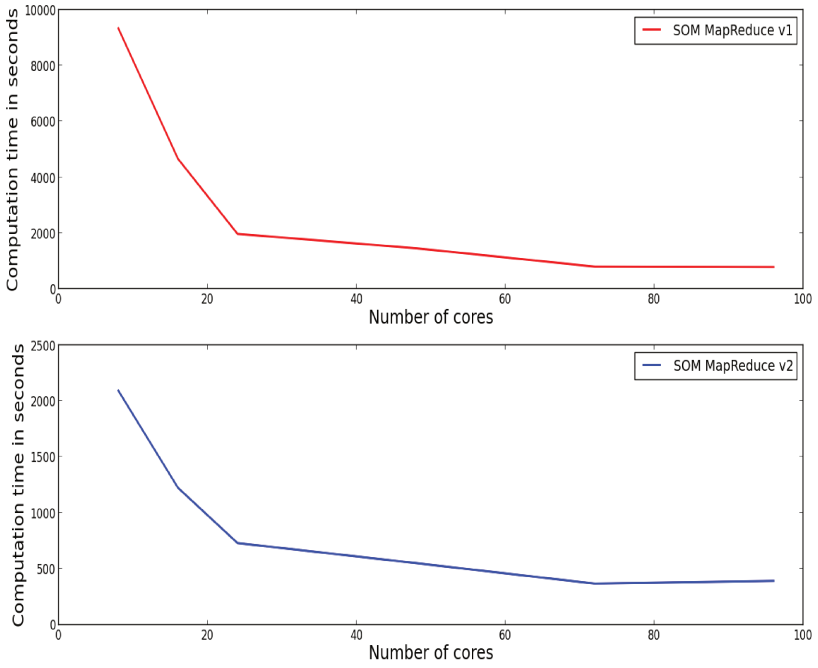


FIGURE 5.1: Speedup for SOM-MR algorithms implemented in Spark. Speedup is relative to execution time by computers.

5.4.3 Variation of the number of observations and variables

The figure 5.2 shows the ratio between execution time and the number of observations. Like in the previous experiment the performances of algorithm 2 are better in all the cases. We furthermore notice that the gap increases more and more as the number of observations increases.

Starting from 1 million the ratio decrease for algorithm 2. This can be explained by the great number of outputs of the Map primitive generated by algorithm 1, which are far superior to those of algorithm 2. When the number of dataset is superior to that of the total RAM of the cluster, Spark store the data on the disk which diminishes the running time.

The figure 5.3 shows the ratio between execution time and the number of variables. Like in the previous test the performances of algorithm 2 are better in all the cases tested, but the curves tend to join together when the number of variables is very high.

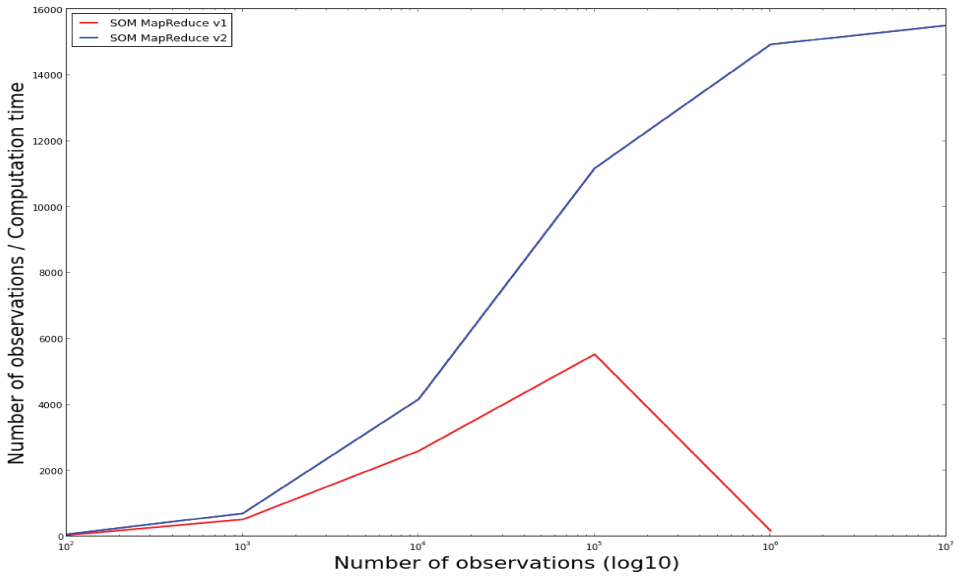


FIGURE 5.2: Ratio between execution time and the number of observations

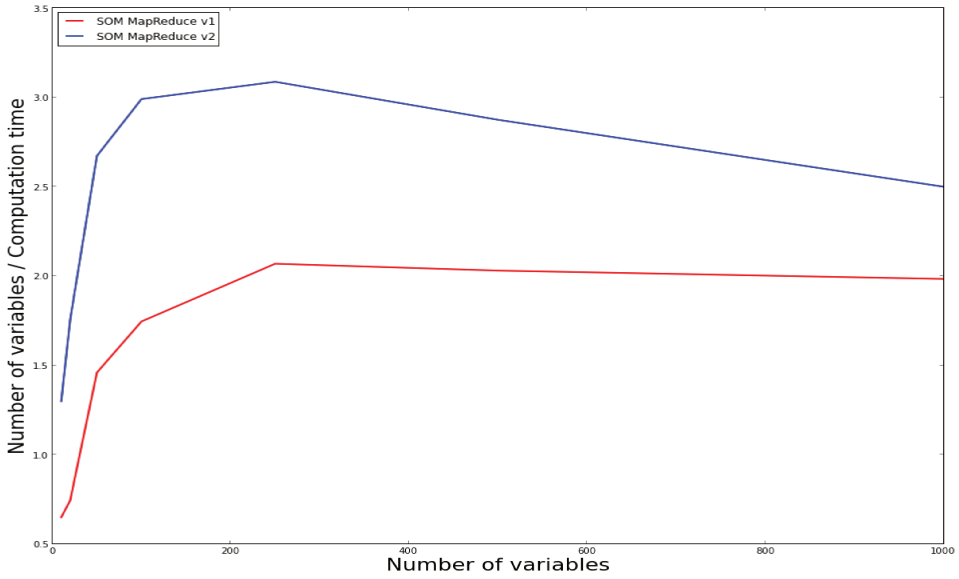


FIGURE 5.3: Ratio between execution time and the number of variables

5.5 Conclusion

We have introduced an adaption of the batch SOM algorithm to the MapReduce programming paradigm using Spark platform. In particular, SOM-MapReduce has been designed to scale to large datasets. The analysis gives a MapReduce algorithm that runs in a constant number of rounds and achieves a constant factor approximation.

The MapReduce paradigm is especially suited for applications within large dataset; this adaption allows SOM to be used in new fields of applications. This

paper aims also to provide a package for clustering algorithm using Spark. The obtained preliminary evidence indicates that the design used for the SOM algorithm can be extended to the other algorithm based on self-organizing map.

We plan to investigate the applicability of the recent work on real and more difficult dataset in order to propose a complete package of clustering and bi-clustering in Spark.

In the next chapter, we will present in details the second contribution which is a bi-clustering algorithm. This algorithm is implemented using MapReduce paradigm under Spark.

Chapter 6

A new Topological Biclustering at scale

In this chapter, we will introduce our second contribution about the "BiTM" algorithm for simultaneous clustering of observations and their features. We start by defining a new cost function and so a new formalization of topological biclustering. After that, we propose a scalable model for biclustering. This model consists of decomposing the biclustering problem into the elementary functions, Map and Reduce. Its implementation is assured in the Spark MapReduce platform.

6.1 Introduction

Biclustering refers to simultaneous clustering of observations and their features. Biclustering dataset is a principal task in a variety of areas of machine learning, data mining, such as text mining, gene expression analysis and collaborative filtering. The term biclustering was first used by Cheng and Church [Cheng and Church, 2000] in gene expression data analysis. Terms such as co-clustering, bidimensional clustering and subspace clustering, among others, are often used in the literature to refer to the same problem formulation.

Different formulations of the biclustering problem have been proposed, such as partitioning model [Hartigan, 1972], bayesian biclustering [Shan et al., 2010], spectral analysis [Greene and Cunningham, 2010], greedy [Angiulli et al., 2006], exhaustive enumeration [Tanay et al., 2002], self-organizing [Benabdeslem and Allab, 2012]. In the direct clustering approach (Block Clustering) [Hartigan, 1972],

the data matrix is divided into several sub-matrices corresponding to blocks. The division of a block depends on the variance of its values.

Indeed, more the variance is low, more the block is constant. Biclustering provides local patterns representing subsets of similar observations and features. Note that biclusters can cover just part of rows or columns. Biclustering is significantly useful and considerably harder problem than traditional clustering. Whereas a cluster is a set of observations with similar values over the entire set of attributes, a bicluster can be composed of observations with similarity over only a subset of attributes.

Recently, biclustering approaches based on matrix decomposition formulation have been proposed as in [Long et al., 2005, Labiod and Nadif, 2011]. In [Long et al., 2005], the authors propose a method named NBVD, which factorizes the data matrix into three components: the row coefficient matrix, the block value matrix and the column coefficient matrix. In [Labiod and Nadif, 2011], authors propose an approach named "Co-clustering Under Nonnegative Matrix Tri-Factorization" (CUNMTF), which generalizes the idea of NMF [Lee and Seung] to factorize the original matrix into three nonnegative matrices. Govaert et al introduce a k-mean like biclustering algorithm titled "Croeuc" to discover all biclusters at the same time. The author of [Govaert, 1983] defines three algorithms for continuous, binary and contingency tables that proceed by optimizing partitions of rows and columns using an iterative k-means procedure. In [Labiod and Nadif, 2011], authors prove that the double k-means is equivalent to algebraic problem of NMF under some suitable constraints. Other probabilistic model-based biclustering have been also proposed in [Govaert and Nadif, 2008, Priam et al., 2008].

Biclustering problems have numerous applications and are becoming more challenging as the size of the data increases. Nevertheless, good clustering algorithms are still extremely valuable and we can (and should) rewrite them for parallel clustering using a new Map-Reduce paradigm [Lv et al., 2010, Lin et al., 2011].

In situations where the amount of data is prohibitively large, the MapReduce (MR) programming paradigm is used to overcome this problem [Dean and Ghemawat, 2008b]. Thus, in recent years, an increasing number of programmers have migrated to the MapReduce programming model [Ene et al., 2011b, Sul and Tovchigrechko, 2011, Ferreira Cordeiro et al., 2011, Ghoting et al., 2011]. The MR programming model was designed to simplify the processing of large files on a parallel system through user-defined Map and Reduce functions [Karloff et al., 2010]. A MR function consists of two phases : a *Map* phase and a *Reduce* phase.

During the Map phase, the user-defined Map primitive function transforms the input data into distributed pairs (key, value). These pairs are then sorted by the system so as to accumulate all values for each key. During the Reduce phase, the user-defined Reduce primitive is invoked on each unique key with a list of all the values for that key; usually, this phase is used to perform aggregations. Finally, the results are output in pairs (key, value). Each key can be processed in parallel during the Reduce phase.

Hadoop[1], an open-source implementation of the MR programming model, has emerged as a popular platform for parallelization. A user can perform parallel computations by submitting MR jobs to Hadoop. While the Hadoop are very popular in their particular domains, we believe that they have a set of limitations that make them ill-suited to the implementation of parallel clustering algorithms. Many common clustering algorithms apply primitive functions repeatedly to the same dataset to optimize a parameter. Thus the Map/Reduce primitive functions need to reload the data, incurring a significant performance penalty.

In this chapter, we are concerned with designing new serial biclustering algorithm and new formalization using MapReduce. We use another emerged open-source implementation named Spark[2] [Zaharia et al., 2010b], which is optimized to machine learning algorithms and supports applications with working sets while providing similar scalability and fault tolerance properties to MapReduce [Sparks et al., 2013].

This chapter proposes a comprehensive new biclustering algorithm based on self–organizing model and distributed biclustering solution from the data to the end clusters using Spark MapReduce. We develop BiTM (Biclustering using Topological Maps) using Spark, an open source package which includes a freely available implementation of MapReduce and has been widely embraced by both commercial and academic worlds. The contributions of this chapter are:

- A new formalization of topological biclustering associated to a new cost function.
- A complete distributed Biclustering solution using Spark MapReduce (we demonstrate its scalability).

The major research challenge addressed is how to minimize the new cost function and the input and output of primitive function (Map and Reduce) for topological

[1] www.hadoop.com
[2] http://spark-project.org/

biclustering algorithm.

The rest of this chapter is organized as follows: Section 6.2 we propose a new biclustering approach (BiTM) based on topological map (self-organizing maps). In Section 6.2.3, we propose our BiTM MapReduce using Spark. Section 6.3 provides the experimental results and shows the comparisons between BiTM and Croeuc which is k-mean as for biclustering. Finally, Section 6.4 concludes and provides some future research.

6.2 A new Topological biclustering: BiTM Model

6.2.1 BiTM Model

Throughout this chapter, we denote a matrix by bold capital letters such as $\mathbf{H}$. Vectors are denoted by small boldface letters such as $\mathbf{g}$ and matrix and vector elements are represented respectively by small letters such as g_i^j and h_i^j. Table 6.1 lists all notations used in BiTM.

As traditional self-organizing maps, which is increasingly used as tools for clustering and visualization, BiTM (Biclustering using Topological Maps) consists of a discrete set of cells $\mathcal{C}$ called map with K cells. This map has a discrete topology defined as an undirected graph, it is usually a regular grid in 2 dimensions. For each pair of cells (c,r) on the map, the distance $\delta(c,r)$ is defined as the length of the shortest chain linking cells r and c on the grid. For each cell c this distance defines a neighbor cell.

Let $\Re^d$ be the euclidean data space and $\mathbf{D}$ the matrix of data, where each observation $\mathbf{x}_i = (x_i^1, x_i^2, ..., x_i^j, .., x_i^d)$ is a vector in $\mathbf{D} \subset \Re^d$. The set of rows (observations) is denoted by $I = \{1, ..., N\}$. Similarly, the set of columns (features) is denoted by $J = \{1, ..., d\}$. We are interested in simultaneously clustering observation I into K clusters $\{P_1, P_2, ..., P_k, .., P_K\}$, where $P_k = \{\mathbf{x}_i, \phi_z(\mathbf{x}_i) = k\}$ and features J into L clusters $\{Q_1, Q_2, ..., Q_l, .., Q_L\}$ where $Q_l = \{\mathbf{x}^j, \phi_w(\mathbf{x}^j) = l\}$. We denote by ϕ_z the assignment function of row (observation) and ϕ_w the assignment function of column (feature).

The main purpose of BiTM is to transform a data matrix $\mathbf{D}$ into a block structure organized in a topological map does. In BiTM, each cell $r \in \mathcal{C}$ is associated with a prototype $\mathbf{g}_k = (g_k^1, g_k^2..., g_k^l, ..., g_k^L)$, where $L < d$ and $g_k^l \in \Re$. To facilitate formulation, we define two binary matrices $\mathbf{Z} = [z_i^k]$ and $\mathbf{W} = [w_j^l]$ to save the

assignment associated respectively to observations and features:

$$z_i^k = \begin{cases} 1 & \text{if } \mathbf{x}_i \in P_k, \\ 0 & \text{else} \end{cases}$$

$$w_j^l = \begin{cases} 1 & \text{if } \mathbf{x}^j \in Q_l \\ 0 & \text{else} \end{cases}$$

To cluster $\mathbf{D}$ into K and L clusters in both observations and features, we propose the new following objective function to optimize in the biclustering process:

$$\begin{aligned} \mathcal{R}(\mathbf{W}, \mathbf{Z}, \mathbf{G}) &= \sum_{k=1}^{K}\sum_{l=1}^{L}\sum_{i=1}^{N}\sum_{j=1}^{d}\sum_{r=1}^{K} \mathcal{K}^T(\delta(r,k)) \\ &\times z_i^k \times w_j^l \times (x_i^j - g_r^l)^2 \end{aligned} \tag{6.1}$$

We can detect the block or bicluster of data denoted by $B_k^l = \{(x_i^j | z_i^k \times w_j^l = 1\}$. $\mathbf{G} = \{\mathbf{g}_1, ..., \mathbf{g}_k\}$ denotes the set of prototype, Typically the neighborhood function $\mathcal{K}^T(\delta) = \mathcal{K}(\delta/T)$ is a positive function which decreases as the distance between two cells in the latent space $\mathcal{C}$ increases and where T controls the width of the neighborhood function.Thus T is decreased between two values T_{max} and T_{min}. In practice, we use the neighborhood function defined as $\mathcal{K}^T(\delta(c,r)) = \exp\left(\frac{-\delta(c,r)}{T}\right)$ and $T = T_{max}(\frac{T_{min}}{T_{max}})^{\frac{t}{t_f - 1}}$, where t is the current epoch and t_f the number of epoch.

The objective function (Eq. 8.1) can be locally minimized by iteratively solving the following three minimization problems:

- Problem 1: Fix $\mathbf{G} = \hat{\mathbf{G}}$ and $\mathbf{W} = \hat{\mathbf{W}}$, solve the reduced problem $\mathcal{R}(\hat{\mathbf{W}}, \mathbf{Z}, \hat{\mathbf{G}})$;
- Problem 2: Fix $\mathbf{G} = \hat{\mathbf{G}}$ and $\mathbf{Z} = \hat{\mathbf{Z}}$, solve the reduced problem $\mathcal{R}(\mathbf{W}, \hat{\mathbf{Z}}, \hat{\mathbf{G}})$;
- Problem 3: Fix $\mathbf{W}$ and $\mathbf{Z}$, solve the reduced problem $\mathcal{R}(\hat{\mathbf{W}}, \hat{\mathbf{Z}}, \mathbf{G})$.

In order to reduce the computational time, we assign each observation and feature without using neighborhood cell as the traditional topological map.

Problem 1 is solved by defining z_k^j as:

$$z_i^k = \begin{cases} 1 & \text{if } \mathbf{x}_i \in P_k, k = \phi_z(\mathbf{x}_i) \\ 0 & \text{else} \end{cases}$$

TABLE 6.1: Table of symbols.

$\mathcal{C}$	Topological map
$\mathbf{D}$	data matrix $N \times d$
$\mathbf{x}_i$	observation vector ($\mathbf{x}_i = (x_i^1, x_i^2, ..., x_i^d), i \in I = 1, 2, ...N$)
$\mathbf{x}^j$	feature vector ($(\mathbf{x}^j)^T = (x_1^j, x_2^j, ..., x_N^j), j \in J = 1, 2, ...d$)
K	the size of observation partition
L	the size of feature partition
$\mathbf{G}$	prototype Matrix $K \times L$
$\mathbf{g}_r$	prototype vector ($\mathbf{g}_r = (g_r^1, g_i^2, ..., g_r^L), L < d$
P_k	cluster of observation (rows)
Q_l	cluster of features (column)
$\mathbf{Z}$	binary matrix after row assignment $\phi_z(\mathbf{x}_i)$
$\mathbf{W}$	binary matrix after column assignment $\phi_w(\mathbf{x}^j)$
B_k^l	bicluster $B_k^l = \{(x_i^j \| z_i^k \times w_j^l = 1\}$
$\mathbf{H}$	neighborhood matrix $K \times K$, $h_i^j = \mathcal{K}^T(\delta(i,j))$
$\mathbf{h}^j$	column j of the matrix $\mathbf{H}$
$\mathbf{h}_i$	row i of the matrix $\mathbf{H}$

Where each observation $\mathbf{x}_i$ is assigned to the closest prototype $\mathbf{g}_k$ using the assignment function, defined as follows:

$$\phi_z(\mathbf{x}_i) = \arg\min_c \sum_{j=1}^{d} \sum_{l=1}^{L} w_j^l (x_i^j - g_c^l)^2 \tag{6.2}$$

Problem 2 is solved by defining w_k^j as

$$w_j^l = \begin{cases} 1 & \text{if } \mathbf{x}^j \in Q_l, l = \phi_w(\mathbf{x}^j) \\ 0 & \text{else} \end{cases}$$

Where each feature $\mathbf{x}^j$ is assigned to the closest prototype $\mathbf{g}^l$ using the assignment function, defined as follows:

$$\phi_w(\mathbf{x}^j) = \arg\min_l \sum_{i=1}^{N} \sum_{k=1}^{K} z_i^k (x_i^j - g_r^l)^2 \tag{6.3}$$

Problem 3 is resolved for the numerical features by :

$$g_r^l = \frac{\sum_{k=1}^{K}\sum_{i=1}^{N}\sum_{j=1}^{d}\mathcal{K}^T(\delta(k,r)z_i^k \times w_j^l \times x_i^j}{\sum_{k=1}^{K}\sum_{i=1}^{N}\sum_{j=1}^{d}\mathcal{K}^T(\delta(k,r))z_i^k \times w_j^l}$$

this value is obtained by resolving the gradients $\frac{\partial \mathcal{R}}{\partial g_r^l} = 0$

$$g_r^l = \frac{\sum_{k=1}^{K}\mathcal{K}^T(\delta(k,r))\sum_{i=1}^{N}\sum_{j=1}^{d} z_i^k \times w_j^l \times x_i^j}{\sum_{k=1}^{K}\mathcal{K}^T(\delta(k,r))\sum_{i=1}^{N}\sum_{j=1}^{d} z_i^k \times w_j^l}$$

$$g_r^l = \frac{\sum_{k=1}^{K}\sum_{x_i^j \in B_k^l}\mathcal{K}^T(\delta(k,r))x_i^j}{\sum_{k=1}^{K}\sum_{x_i^j \in B_k^l}\mathcal{K}^T(\delta(k,r))}$$

We can rewrite the equation as follows:

$$g_r^l = \frac{\sum_{k=1}^{K}\mathcal{K}^T(\delta(k,r))\sum_{x_i^j \in B_k^l} x_i^j}{\sum_{k=1}^{K}\mathcal{K}^T(\delta(k,r))\sum_{x_i^j \in B_k^l}} \tag{6.4}$$

The main phases of BiTM algorithm are presented in Algorithm 13.

6.2.2 BiTM Model vs. Croeuc

The decomposition of the cost function $\mathcal{R}$ (Eq. 8.1) that depends on the value of T, permits to rewrite its expression as follows:

$$\mathcal{R}(\mathbf{W},\mathbf{Z},\mathbf{G}) = \sum_{k=1}^{K}\sum_{l=1}^{L}\sum_{\mathbf{x}_i \in P_k}\sum_{\mathbf{x}^j \in Q_l}\sum_{r=1}^{K}\mathcal{K}^T(\delta(r,k))(x_i^j - g_r^l)^2$$

$$\mathcal{R}(\mathbf{W},\mathbf{Z},\mathbf{G}) = \mathcal{R}_1(\mathbf{W}|Q.,\mathbf{Z}|P.,\mathbf{G}) + \mathcal{R}_2(\mathbf{W}|Q.,\mathbf{Z}|P.,\mathbf{G})$$

Algorithm 13 : BiTM Algorithm

1: **Inputs:**

- The data $\mathbf{D}$, prototypes $\mathbf{G}$ (Initialization).
- t_f : the maximum number of iterations.

2: **Outputs:**

- Assignment matrix $\mathbf{Z}, \mathbf{W}$. Prototypes $\mathbf{G}$

3: **while** $t \leq t_f$ **do**
4: **for all** $\mathbf{x}_i \in \mathbf{D}$ **do**
5: **Observation assignment phase**: Each observation $\mathbf{x}_i$ is assigned to the closest prototype $\mathbf{g}_k$ using the assignment function, defined in equation 8.2
6: **Features assignment phase**: Each feature $\mathbf{x}^j$ is assigned to the closest prototype $\mathbf{g}^l$ using the assignment function, defined in equation 8.3
7: **Quantization phase**: The prototype vectors are updated using following expression defined in equation 6.4
8: **end for**
9: Update T
{ T varies from T_{max} until T_{min}}
10: t++
11: **end while**

where

$$\begin{aligned}\mathcal{R}_1(\mathbf{W}|Q_\cdot, \mathbf{Z}|P_\cdot, \mathbf{G}) &= \sum_{k=1}^{K}\sum_{l=1}^{L}\sum_{\mathbf{x}_i \in P_k}\sum_{\mathbf{x}^j \in Q_l}\sum_{r=1, r\neq k}^{K} \mathcal{K}^T(\delta(r,k)) \\ &\times \ (x_i^j - g_r^l)^2\end{aligned}$$

and

$$\mathcal{R}_2(\mathbf{W}|Q_\cdot, \mathbf{Z}|P_\cdot, \mathbf{G}) = \mathcal{K}^T(\delta(k,k)) \sum_{r=1}^{K}\sum_{l=1}^{L}\sum_{\mathbf{x}_i \in P_k}\sum_{\mathbf{x}^j \in Q_l} (x_i^j - g_r^l)^2$$

where $\delta(k,k) = 0$

The cost function $\mathcal{R}$ is decomposed in two terms. In order to maintain the topological order between blocks, minimizing the first term $\mathcal{R}_1(\mathbf{W}|Q_\cdot, \mathbf{Z}|P_\cdot, \mathbf{G})$ will bring the block corresponding to neighboring cells. Indeed, if c and r are neighbors on the map $\mathcal{C}$, the value of $\delta(r,k)$ is low and in this case the value of $\mathcal{K}^T(\delta(r,k))$ is high. Thus, minimizing the first term has for effect to reduce

the value of the cost function. Minimizing the second term $\mathcal{R}_2(\mathbf{W}|Q., \mathbf{Z}|P., \mathbf{G})$ corresponds to the minimization of the local inertia of component assigned to block B_r^l as follows

$$\mathcal{R}_2(\mathbf{W}|Q., \mathbf{Z}|P., \mathbf{G}) = \sum_{l=1}^{L} \sum_{\mathbf{x}_i^j \in B_r^l} (x_i^j - g_r^l)^2$$

Where $\mathcal{R}_2(\mathbf{W}|Q., \mathbf{Z}|P., \mathbf{G})$ presents the cost function proposed by Govaert [Govaert, 1983].

Hence, for different values of temperature T, each term of the cost function has a relative relevance in the minimization process. We can define two steps in the operating of the algorithm:

- The first step corresponds to high T values where the first term $\mathcal{R}_1(\mathbf{W}|Q., \mathbf{Z}|P., \mathbf{G})$ is dominant and in this case, the priority is to preserve the topology.
- The second step corresponds to small T where the second term $\mathcal{R}_2(\mathbf{W}|Q., \mathbf{Z}|P., \mathbf{G})$ is considered in the cost function. Therefore, the adaptation is very local and BiTM algorithm converge to Croeuc algorithm.

The computational cost of our BiTM model is more expensive than Croeuc, because the neighborhood of the topological map increases the number of output pairs (k, r) of map function.

6.2.3 BiTM and MapReduce

To handle this huge amount of data, it is necessary to use distributed architecture. This is not a simple task and several difficulties have to be dealt with, including loading data, failure safety, and algorithm design. The MapReduce implementation on Spark takes care of failure-correction, data management and distribution.

It has become very important in MapReduce to decompose our problem in elementary functions. The idea is to initiate two Map-Reduce functions for row and column iterations, and a synchronization to update the parameters $\mathbf{G}, \mathbf{W}, \mathbf{Z}$. In the case of BiTM algorithm we identified these atomic parts:

- Assign each observation $\mathbf{x}_i$ to the best match unit using expression (Eq. 8.2).
- Assign each feature $\mathbf{x}^j$ to the best match unit using expression (Eq. 8.3).
- Accumulate denominator and numerator for each prototype $\mathbf{g}_r$ (Eq. 6.4)
- Update prototype vectors $\mathbf{g}_r, \forall r \in C$ (Eq. 6.4)

Pseudo code below describes the implementation of the BiTM algorithm with MapReduce Spark. In most of the cases in big dataset the number of observations is bigger than the number of features ($N >> d$). Thus, we consider that a column vector $\mathbf{x}^j$ (column) of the data matrix $\mathbf{D}$ couldn't be used as an observation vector $\mathbf{x}_i$ (row). Hence for the row assignment function we define row map function (Algorithm 17), which computes for each $\mathbf{x}_i$ the best match unit. In order to reduce the time complexity we compute the denominator and the numerator of the prototype $\mathbf{g}$ (Eq. 6.4). This allows the row reduce function to focus only on the sum of different numerators and denominators provided by each row map function.

For the column assignment, we define a map function (15) and a reduce function (Algorithm 18) in order to reduce the memory consumption. Thus the column assignment function splits the column distance of one column into multiple outputs ($d \times K$). The map function computes the distance between each element x_i^j and all prototypes. Hence the column reduce function sums the "distances" provided by each column map function. At the end of this MapReduce state the best match prototype for features is computed in the main function (Algorithm 14). Therefore, the update of the prototype is computed.

The majority of the algorithm algorithm functions are executed in Map and Reduce. They are executing in parallel on all the machines of the Spark clusters. That is why the algorithm could scales easily (Chapter 6.3.2 Speedup tests).

6.3 Experiments

Our performance study is based on the synthetic datasets and on several datasets extracted from UCI repository [Frank and Asuncion, 2010]. Table 6.2 lists public dataset, the number of real class and the map size used in the learning phase. We used 4 synthetic datasets generated with 1 million and 2 million observations with

Algorithm 14 BiTM - Main

Initilization
{Random initialization of prototypes}
{Random initialization of columns assignements $\mathbf{W}$}
{**Main loop**}
while $t \leq t_f$ **do**

{**Assignment of columns** }
for all $\mathbf{x}_i \in \mathbf{D}$ **do**
$\langle (J, L); V \rangle \leftarrow \text{ColMapper}(\mathbf{x}_i)$
end for
$\langle (J, L); V \rangle \leftarrow \text{ColReducer}(\langle (J, L); V \rangle)$
for each column reduce value $j \in J$ **do**
$\phi_w(\mathbf{x}^j) = \arg\min(\langle (j, L); V \rangle)$
end for

{**Assignment of rows** }
for all $\mathbf{x}_i \in \mathbf{D}$ **do**
$\langle (\mathbf{CM}, \mathbf{CN}) \rangle = \text{RowMapper}(\mathbf{x}_i)$
end for
$(\mathbf{CMs}, \mathbf{CNs}) = \text{RowReducer}(\langle (\mathbf{CM}, \mathbf{CN}) \rangle)$

{**Update prototypes**}
$\mathbf{G} = \mathbf{CMs}/\mathbf{CNs}$
$t+ = 1$
end while

Algorithm 15 ColMapper($\mathbf{x}_i$)

for each column $j = 1..d$ **do**
for each feature prototype $l = 1..L$ **do**
emit $\left\langle (j, l); (x_i^j - g_r^l)^2 \right\rangle$
end for
end for

Algorithm 16 ColReducer($key(j, l)$, V)

$s_v = 0$
for each map value $v \in V$ **do**
$s_v+ = v$
end for
emit $\langle (j, l); s_v \rangle$

Algorithm 17 RowMapper($\mathbf{x}_i$)

{**MAP rows distance:** distributed loop over all input vectors ($\mathbf{x}_i = (x_i^1, x_i^2, ..., x_i^d)$) }
for each $k = 1..K$ **do**
  $bmu(k) = \| \mathbf{x_i} - \mathbf{g_k} \|^2$
end for
$\phi_z(\mathbf{x}_i) = \arg\min(\mathbf{bmu})$
{Contrsuct a new compressed vector : **cm** with $1 \times L$ dimensions}
$\mathbf{cm}(l) = \sum_{x_i^j \in B^l_{\phi_z(\mathbf{x}_i)}} x_i^j$
$\mathbf{CM} = \mathbf{h}^{\phi_z} \times \mathbf{cm}$
{Contrsuct a new compressed vector : **cm** with $1 \times L$ dimensions}
$\mathbf{cn}(l) = \sum_{\mathbf{x}^j \in B^l_{\phi_z(\mathbf{x}_i)}} 1$
$\mathbf{CN} = \mathbf{h}^{\phi_z} \times \mathbf{cn}$
{So **CM** and **CN** are matrices of size $K \times L$}
emit $\langle(\mathbf{CM}, \mathbf{CN})\rangle$

Algorithm 18 RowReducer($\mathbf{V}(\mathbf{CM}, \mathbf{CN})$)

Initialize $\mathbf{CMs} \leftarrow 0, \mathbf{CNs} \leftarrow 0$
for each $(\mathbf{CM}, \mathbf{CN}) \in \mathbf{V}$ **do**
  $\mathbf{CMs} += \mathbf{CM}$
  $\mathbf{CNs} += \mathbf{CN}$
end for
emit $\langle(\mathbf{CMs}, \mathbf{CNs})\rangle$

20 and 40 features each.

An open source Spark library that include the BiTM and our implementation of SOM (self-organizing map) is available on Github (`https://github.com/TugdualSarazin/spark-clustering`).

6.3.1 BiTM versus Croeuc

Firstly, we're interested in comparing our model with algorithms that possess similar architecture to ours, such as Croeuc. We report the Acc (accuracy) and NMI, and plot the quantization error vs.the number of iterations, a common measure for evaluation clustering and biclusteing methods [Strehl et al., 2002].

For the setup of experiments, we have to consider various parameters. For both methods (BiTM and Croeuc), we choose a fixed number of clusters (size of the map), indicated in table 6.2. For both algorithms we randomly select 10 initial cluster centers with 40 epochs. Thus the values indicate the average of 10 measures for each dataset.

datasets	# obs	# feat	Map size	# classes
Sonar mines	208	60	6×6	2
Lung Cancer	32	56	4×4	2
Spectf 1	349	44	4×4	2
Cancer Wpbc Ret	198	33	6×6	2
Horse Colic	300	27	5×5	2
Heart	270	13	5×5	2
Isolet5	1559	617	12×12	26
Breast	699	10	7×7	2
Glass	214	9	5×5	7
Waveform	5000	40	10×10	3

TABLE 6.2: Public datasets description (# obs: number of observation, # feat: number of features)

6.3.1.1 Clustering quality

To measure the quality of clustering, we use two different metrics: Accuracy (Acc) and Normalized Mutual Information (NMI); each should be maximized. They are used to evaluate clustering quality and are only applicable to data sets with ground-truth classes. NMI is particularly useful when the number of clusters is different from that of ground-truth classes.

The clustering quality of both algorithms is shown in Table 6.3 and 6.4. In practice BiTM provides better values of quality measures than Croeuc in most of selected datasets with small standard deviation. Particularly, for Cancer Wpbc Ret and Isolet, our method provides less Accuracy and NMI than Croeuc. BiTM is like a Croeuc with a topological order. These tests demonstrate that the addition of this topological order doesn't modify the general process of the algorithm.

6.3.1.2 Quantization error

In this experiment, our purpose is to study how BiTM evolves over time in term of quantization error. The average of quantization error versus the number of iterations for 10 runs are summarized in Figure 6.1.

Unlike the previous tests (NMI and Acc), quantization error measures that Croeuc results are equivalent with BiTM in most of cases. The quantization error is not a measure of clustering's performance, it only describes the learning of

Databases	BiTM		CROEUC	
	Acc	Std	Acc	Std
Sonar mines	**0.7365**	0.0174	0.6837	0.0124
Lung cancer	**0.7906**	0.0296	0.7469	0.0231
Spectf 1	**0.7817**	0.0096	0.7358	0.0069
Cancer Wpbc Ret	**0.7626**	0.0000	**0.7626**	0.0000
Horse colic	**0.6770**	0.0062	0.6707	0.0021
Heart	**0.8130**	0.0084	0.8007	0.0072
Isolet5	**0.5640**	0.0179	0.4184	0.0455
Glass	**0.5692**	0.0206	0.3799	0.0121
Breast	**0.9725**	0.0026	0.9700	0.0027
Waveform	**0.7313**	0.0054	0.7094	0.0081

TABLE 6.3: Clustering Accuracy performance (Acc).

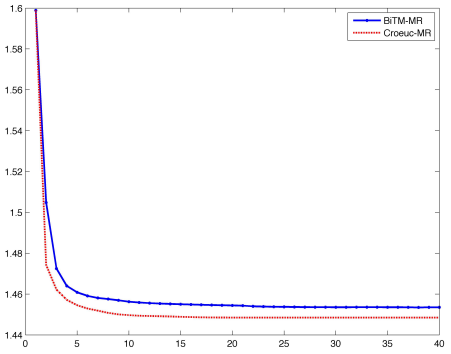
Databases	BiTM		CROEUC	
	NMI	Std	NMI	Std
Sonar mines	**0.1316**	0.0211	0.0923	0.0140
Lung cancer	**0.1673**	0.0363	0.1045	0.0402
Spectf 1	**0.1464**	0.0088	0.1336	0.0060
Cancer Wpbc Ret	0.0205	0.0002	**0.0238**	0.0050
Horse colic	**0.0237**	0.0023	0.0103	0.0040
Heart	**0.2143**	0.0057	0.2129	0.0089
Isolet5	0.5811	0.0090	**0.5887**	0.0170
Glass	**0.2743**	0.0071	0.1089	0.0163
Breast	**0.4128**	0.0057	0.4627	0.0101
Waveform	**0.2458**	0.0022	0.27934	0.0042

TABLE 6.4: Clustering performance comparison using NMI.

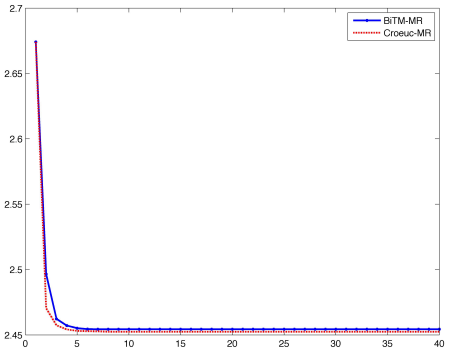
algorithms. They have the same process, they decrease quickly during the first iterations and stabilized after.

The main difference between them is the weak decrease of BiTM during the stabilization phase. We can see this phenomenon on the figure 6.1. This difference could be explained by the topological order added in BITM. BitM as Self-Organinzing Map has two phase: self-organizing step and quantization phase.

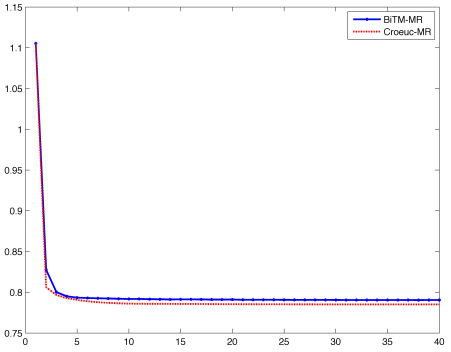
FIGURE 6.1: Quantization error



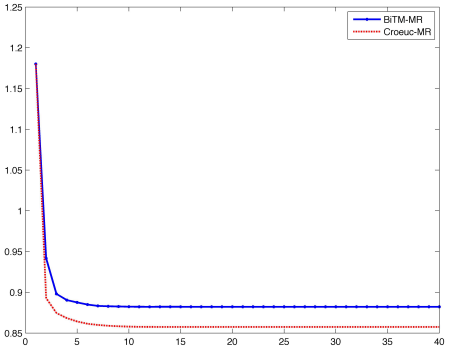
(a) Sonar



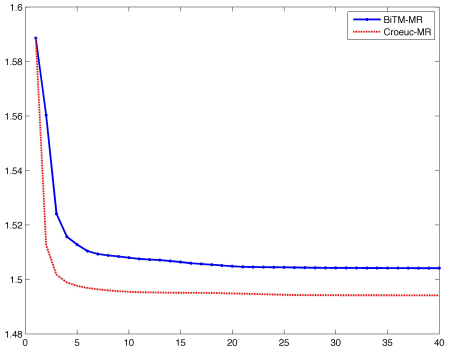
(b) Lung cancer



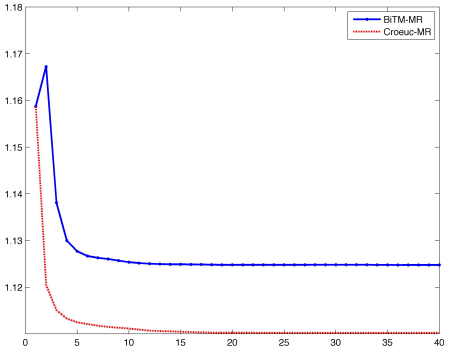
(c) Specif



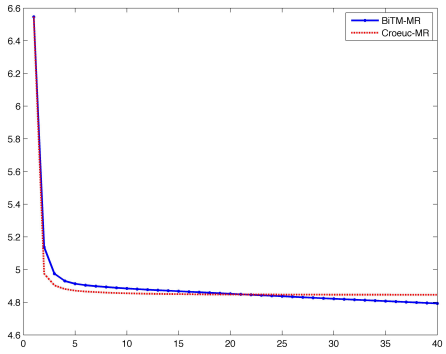
(d) CancerWpbcRet



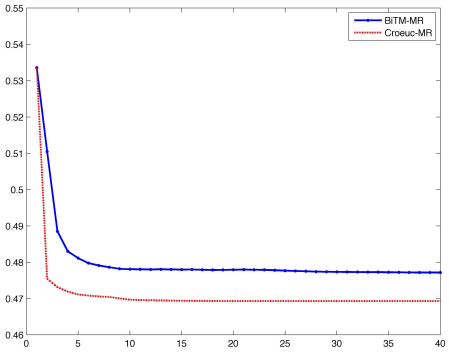
(e) HorseColic



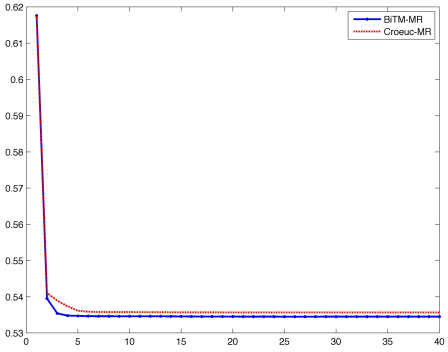
(f) Heart



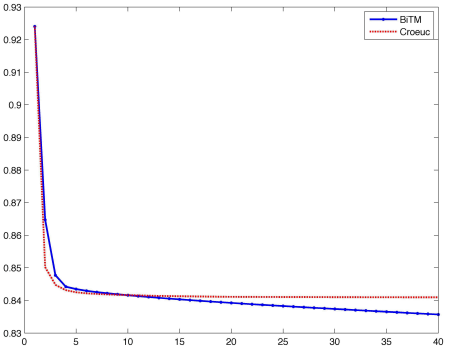
(g) Isolet



(h) Breast



(i) Glass



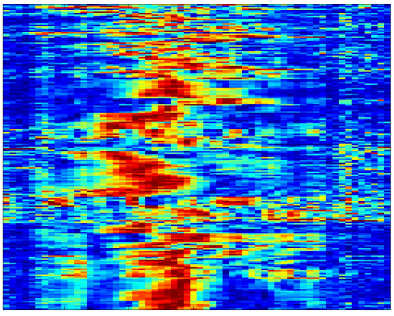
(j) Waveform

6.3.1.3 Visualization

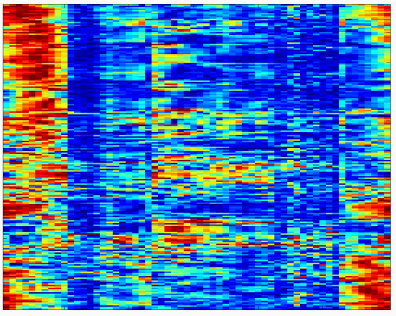
Figure 6.3 shows how the proposed method BiTM provides further information than other clustering approaches. The main advantage is to provide a simultaneous clustering with topological order as seen in Figures 6.2(m), 6.3(c). We used Matlab as the framework to visualize different figures. This simplifies data exploration by offering friendly visualization comparing to global presentation of biclustering presented in 6.2(l), 6.3(b). The same analysis could be done with the rest of the dataset. Our approach tries to improve the standard the visualization by building simultaneous clustering with topological order.

Figures 6.2(l) and 6.3(b) show the topological organization of bicluster on respectively SonarMines dataset 6.2(k), waveform datasets 6.2(k). In BiTM, each cell of the map represents a subset of data, organized according to the cluster of observations and features. This organization is illustrated by different colors. Weak features are indicated with blue and dominant features are indicated with red. Colors are relatively similar when the features are correlated and mapped

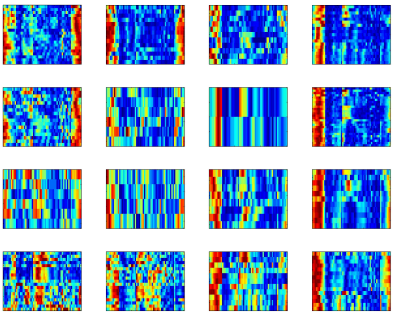
in neighborhood cells. This is due to the use of the neighborhood relationship in BiTM learning.



(k) Data set



(l) Dataset organized according the cluster of the observations and the features



(m) BiTM Map. bicluster organized in a map according the features and observation cluster

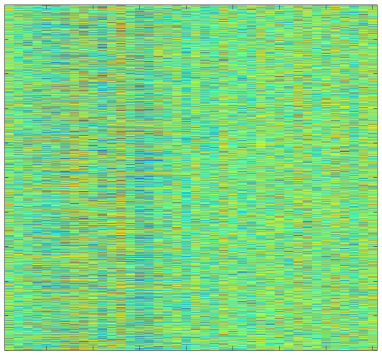
FIGURE 6.2: SonarMines visualization

6.3.2 Speedup tests

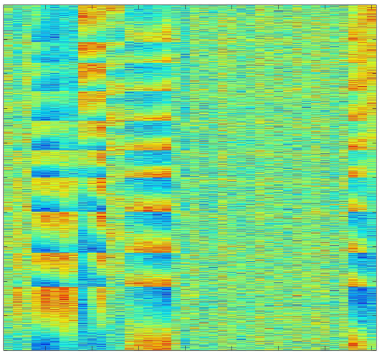
We made scaling experiments on the Magi cluster (http://www.univ-paris13.fr/calcul/wiki/) with 1 to 10 machines of the cluster. Each machine on this cluster has 12 cores (two Xeon X5670 at 2.93GHz), 24GB of RAM and they are connected by an InfiniBand network. All the experiments are done in Spark Platform. We generate different dataset using the Scala random normal function. We apply a different factor for each class. $x = a + \mathcal{N} * b$.

In each of the two classes, a and b are different. But neither the number of class nor the data generation function influctes significantly the execution time of the algorithm.

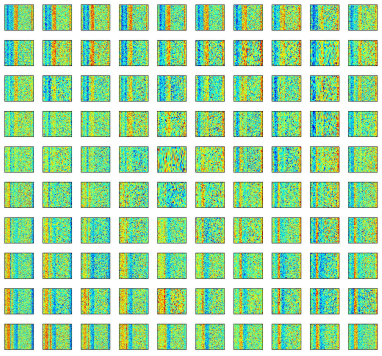
For benchmarking the performance of our BiTM MapReduce implementation,



(a) Data set



(b) Dataset organized according the cluster of the observations and the features

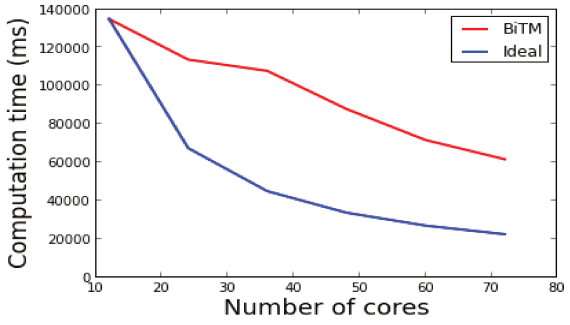


(c) BiTM Map. bicluster organized in a map according the features and observation cluster

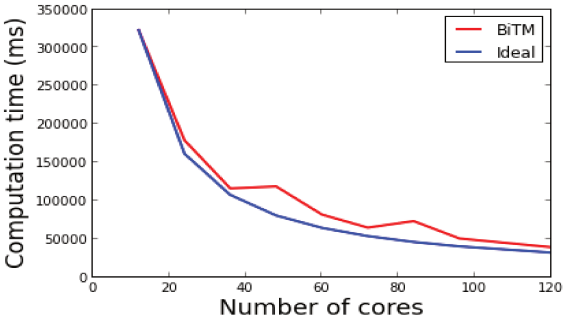
FIGURE 6.3: Waveform visualization

we generated 1 and 2 million observations of 10 to 40 dimensions each. Then, we trained a 5×5 BiTM map with different core counts in each run. The run time is indicated in milliseconds. In all figures below we added an ideal curve. It had been determined by the execution time for one machine divided by the number of machines. This curve represents a perfect scaling architecture.

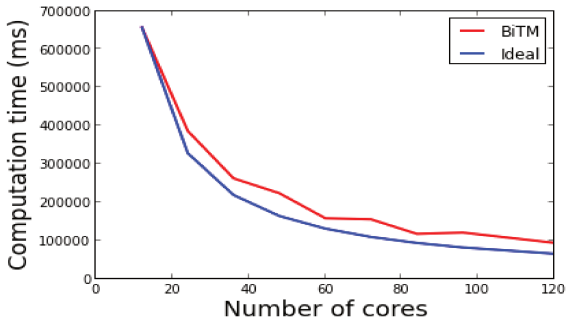
In the figure (6.4(a)) the ideal time is much lower than in the following figures. That could be explained by the low size of the data matrix (1 million elements). When the size of the data matrix is low the performances of the initialization of the system decrease. But with a huge data matrix the initialization tends to be insignificant proportionally to the global run time. As we can see in the following figures 6.4(b), 6.4(c), 6.4(d), 6.4(e). The implementation exhibited excellent linear scaling, and is close to ideal times.



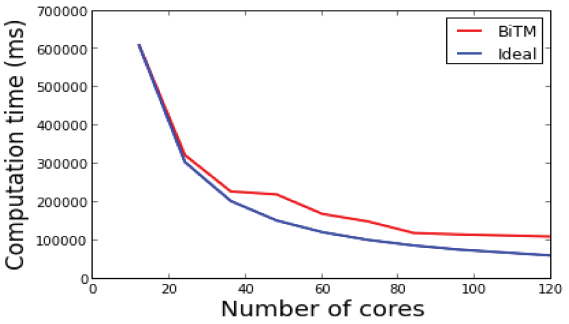
(a) 1 millions observations, 10 features



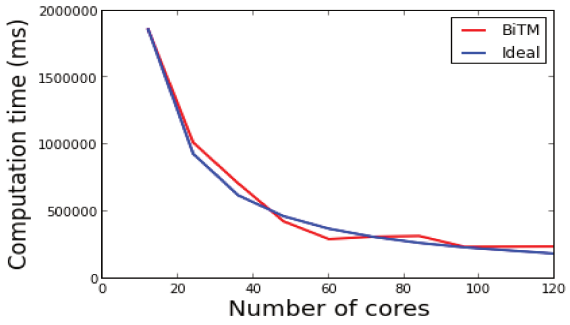
(b) 1 millions observations, 20 features



(c) 1 millions observations, 40 features



(d) 2 millions observations, 20 features



(e) 2 millions observations, 40 features

FIGURE 6.4: BiTM Spark execution times

6.4 Conclusion

In this chapter, we considered a new topological biclustering approach, which is based on self-organizing map and Croeuc algorithm. We have presented some experiments to promote the new biclustering algorithm.They reveal that the BiTM maintains good accuracy and NMI performances. We designed scalable implementations using MapReduce with the Spark platform. The MapReduce paradigm is especially suited for applications within large dataset; this adaptation allows BiTM to be used in new fields of application.

This chapter aims also to provide a library for clustering and biclustering algorithms using Spark. The obtained preliminary results indicate that the design used for the BiTM algorithm can be extended to the other biclustering algorithms. In the future, we plan also to develop a method that can automatically divide features into cluster using the weighted biclustering process.

Chapter 7

Application to Insurance Dataset

This chapter is devoted to explain our work carried in the context of the Big Data project, named Square Predict [1]. We illustrate the utility of the SOM-MR algorithm, presented in the chapter 5, as an unsupervised learning for an insurance Big Data.

7.1 Introduction

Organisations are increasingly relying on Big Data to provide the opportunities to discover correlations and patterns in data that would have previously remained hidden, and to subsequently use this new information to increase the quality of their business activities.

Classification and regression trees (CART) are a useful technique for creating easily interpretable decision rules, see [Hastie et al., 2009]. The R package `rpart` provides a convenient interface to classification and regression trees and whose plotting is extended by the `rpart.plot` package.

In this chapter we prensent an analysis combining an unsupervised learning with a supervised method. The SOM-MR algorithm, which is presented in the chapter 5, is used as an unsupervised method while the regression trees are used to explain the clusters produced by the SOM-MR approach.

[1] http://ns209168.ovh.net/squarepredict/

7.2 Exploratory data analysis of SOM clusters

The 2012 AXA insurances payouts data consists of 2130114 contracts enriched with open data from the INSEE and ONDRP. A SOM (self-organising map) clustering was carried out by Arrow on these data, resulting in 20 clusters. The goal of is to construct a decision tree model of these enriched data in determining the payouts made for water damage (DDE) claims `charge_dde` and for the payouts made for fire damage (INC) claims `charge_inc` within each of the SOM clusters.

As `charge_inc` and `charge_dde` are continuous variables, a regression tree analysis is appropriate. For each of the SOM clusters, a regression true with the response variable being `charge_inc` or `charge_dde`, and the covariates being the other variables. For the fire damages claims, these regressions trees are displayed in Figures 7.1–7.2, in decreasing order of the total sum of payouts per cluster. In each tree, the node labels contain two values: inside the lozenge is the total payouts, and below it is the number of claims. At each binary split, the left and right branches indicate the rule applied to the splitting variable.

All the decision trees begin with the decision `nbsin_inc` < 0.5 which separates all the contracts with/without any damage claims at the root node. All the contracts without any claims becomes a terminal node, as they also do not contribute to the payouts. All the claims are then decomposed with further decision rules based on the covariates. For example, if we focus on the SOM NumCluster=9 in the middle panel in Figure 7.1, we observe that the terminal node (labelled internally 55) has a payout total greater than €180K from only 8 contracts. Its complete decision rules are

```
Rule number: 55 [charge_inc=189513.375 cover=8 (0%)]
  nbsin_inc>=0.5
  constAvant49Prob< 0.8953
  dept=3,5,6,12,15,16,17,18,19,21,24,32,43,47,54,55,56,65,70,71,78,79,82,87,88,90
  constAvant49Prob>=0.6254
  dept=17,43,82
```

showing that the geographical location (`dept`) and the age of construction (`constAvant49Prob`) are used to construct this leaf node: its 8 extracted contracts are

```
nbsin_inc charge_inc NumCluster dept constAvant49Prob ...
```

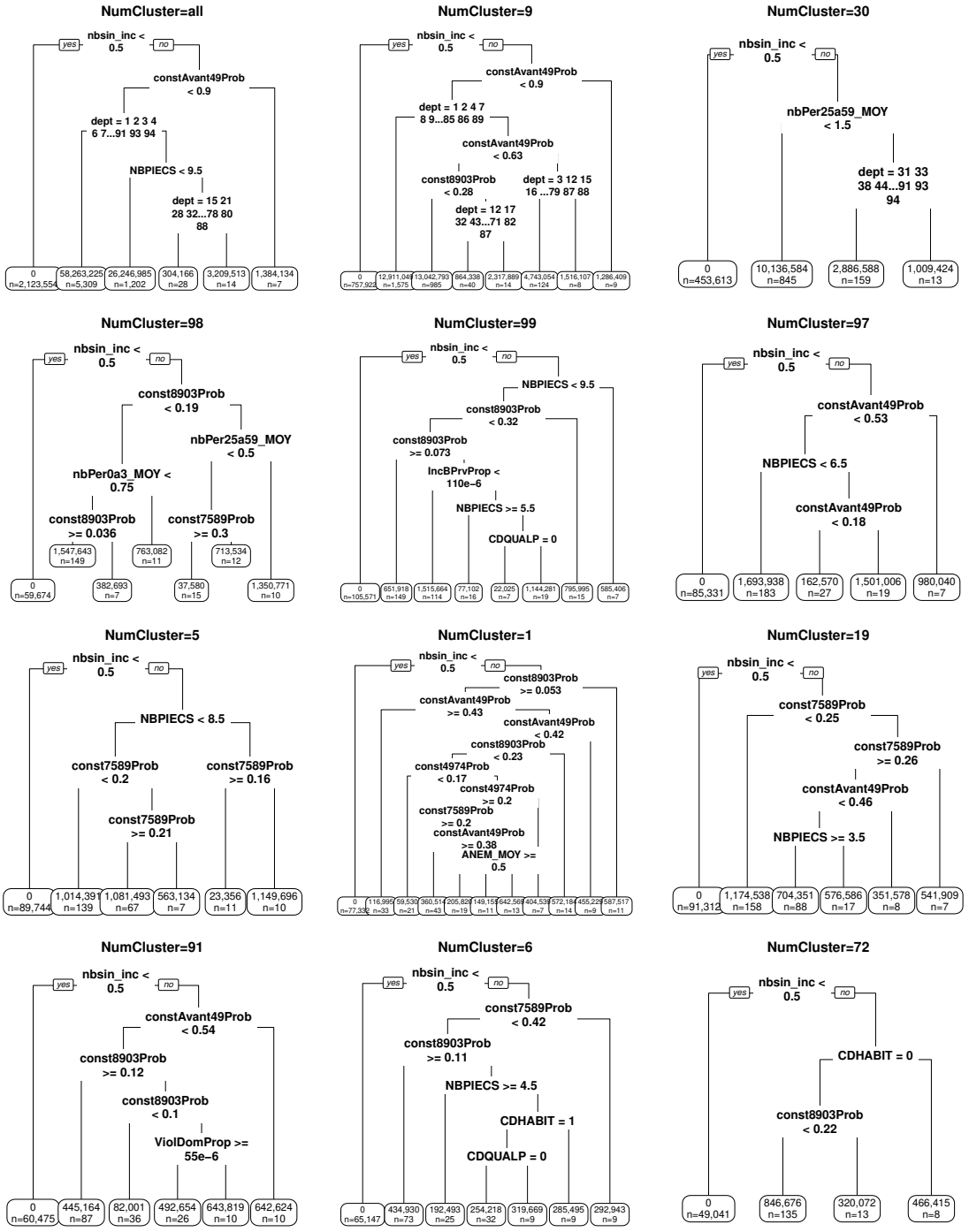


FIGURE 7.1: Decision trees for the enriched AXA data for `charge_inc` (fire damages) payouts, sorted by SOM cluster payouts: NumCluster = all, 9, 30, 98, 99, 97, 94, 5, 1, 19, 91, 6, 72.

1	550000	9	17	0.62667
1	363	9	17	0.70476
1	21816	9	17	0.68493
1	859	9	17	0.71795
1	600000	9	43	0.64327

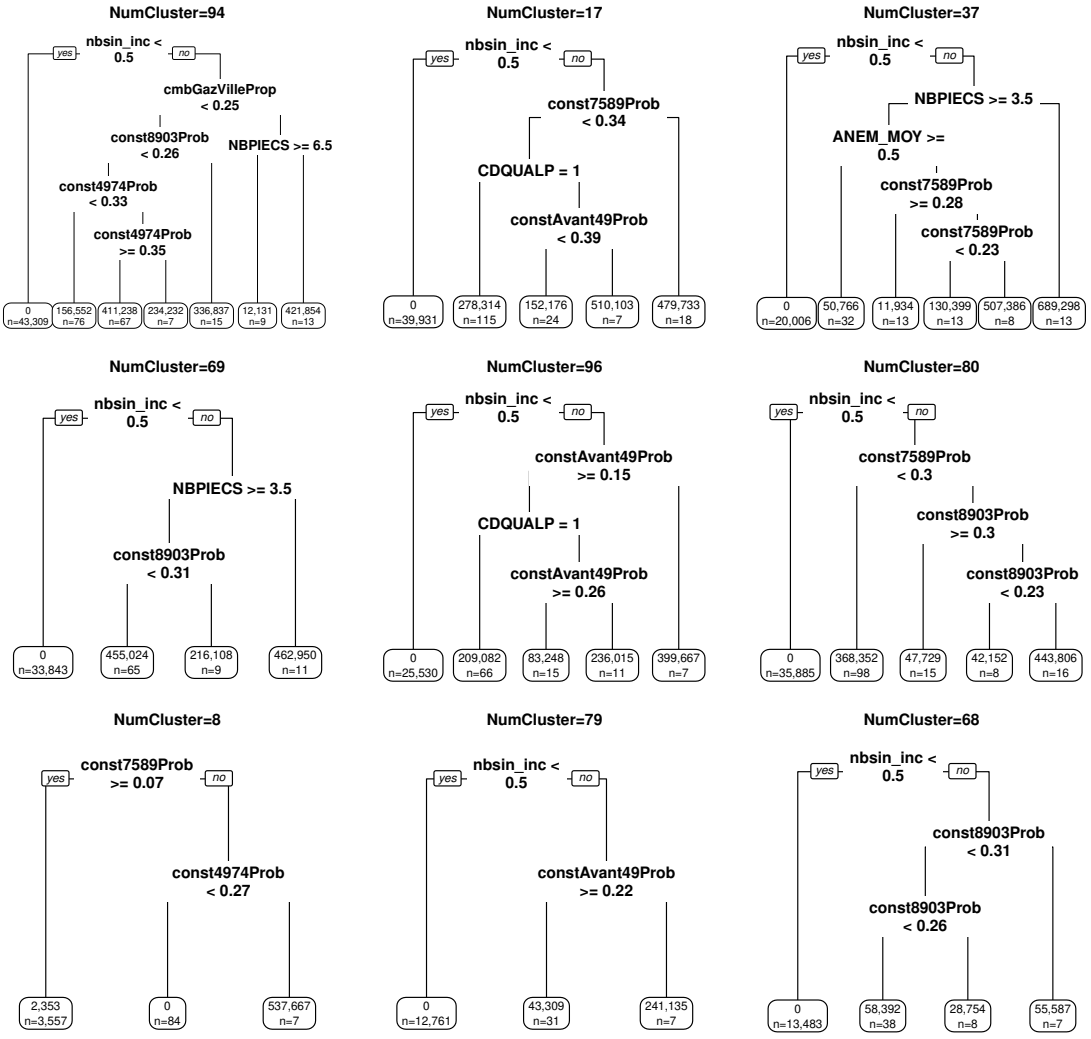


FIGURE 7.2: Decision trees for the enriched AXA data for `charge_inc` (fire damages) payouts, sorted by SOM cluster payouts: NumCluster = 94, 17, 37, 69, 96, 80, 8, 79, 68.

1	1459	9	43	0.84706
1	1841	9	43	0.77477
1	339769	9	82	0.68750

A similar analysis can be carried out with the `nbsin_dde` decision trees.

7.3 Supervised learning of SOM clusters

In the previous section, we examined decision trees for the exploratory analysis of the SOM clusters. In this section, we examine decision trees for the prediction of these SOM cluster labels in a supervised learning context. The response variable is

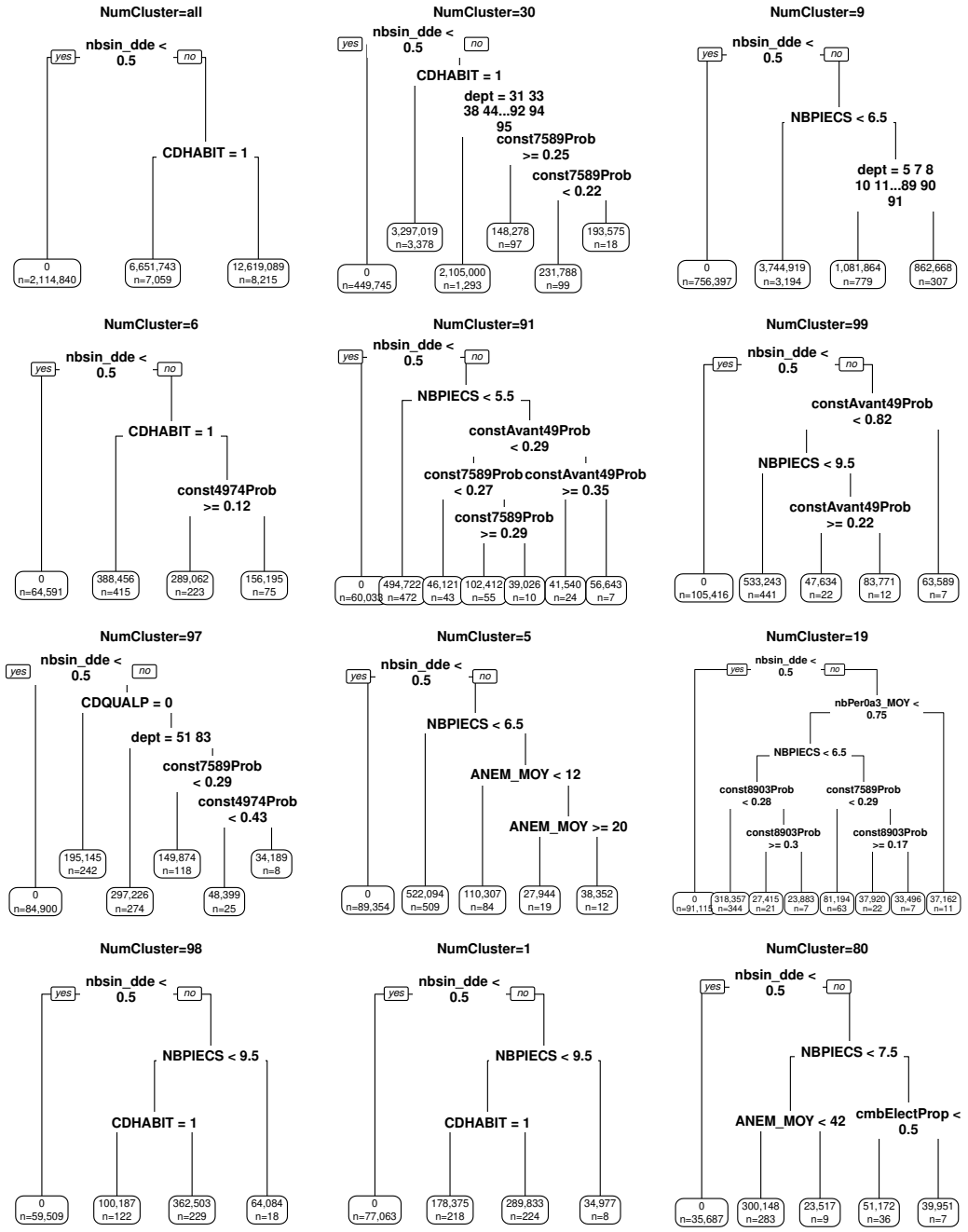


FIGURE 7.3: Decision trees for the enriched AXA data for `charge_dde` (water damages) payouts, sorted by SOM cluster payouts: NumCluster = all, 30, 9, 6, 91, 99, 97, 5, 1, 19, 98, 1, 80.

the SOM cluster label `NumCluster`, which is a categorical variable, a classification tree is appropriate.

The categorical department variable `dept` (94 levels) causes a combinatorial explosion when used in conjunction with the categorical response `NumCluster` (20 levels) in a decision tree. The `dept` variable is not well-suited as an ordinal

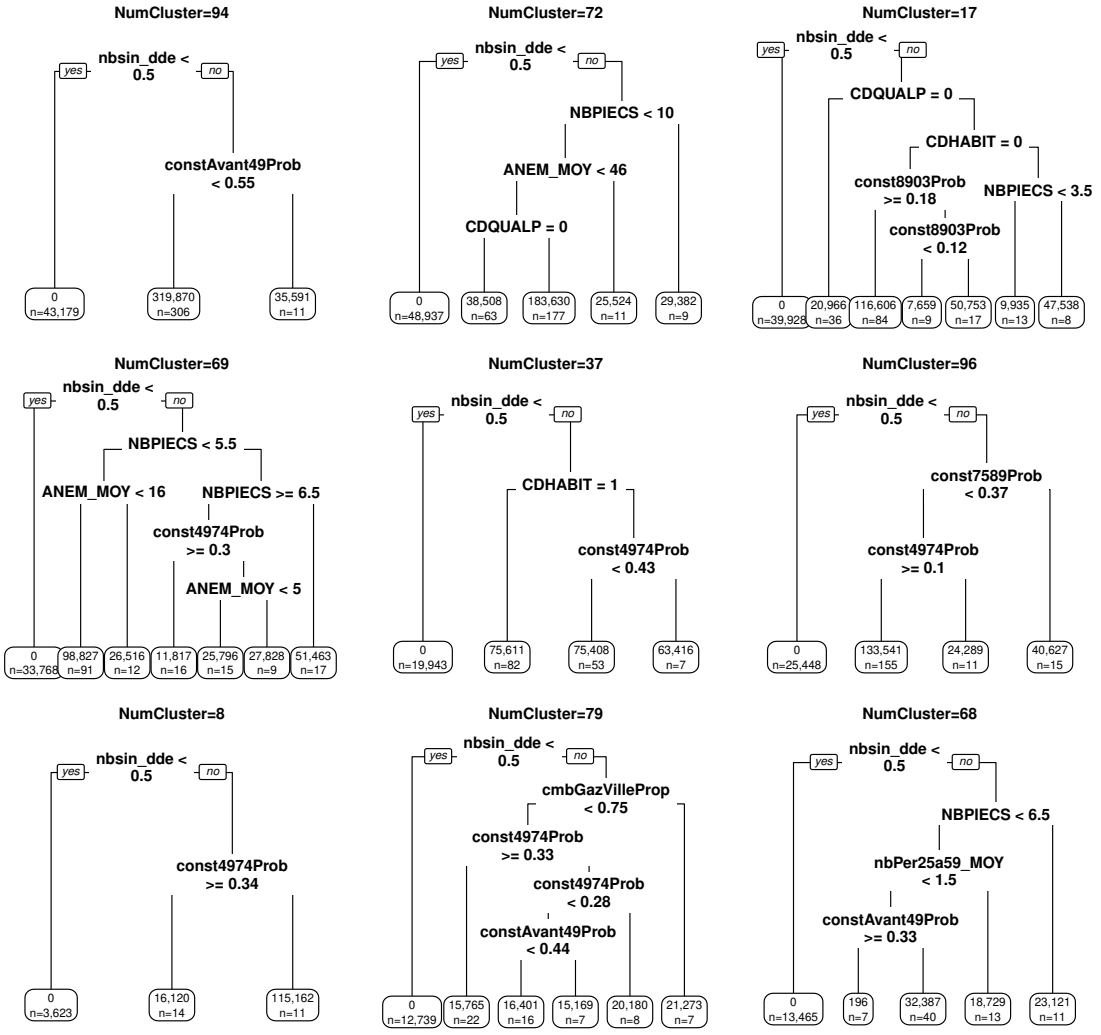


FIGURE 7.4: Decision trees for the enriched AXA data for `charge_dde` (water damages) payouts, sorted by SOM cluster payouts: NumCluster = 94, 72, 17, 69, 37, 96, 8, 79, 68.

variable either, e.g. `dept` < 75 would include 75 (Paris: (`longitude`, `latitude`) = (2.34, 48.86)) with the geographically distant 74 (Haute-Savoie: (6.12, 45.90)) but exclude the geographically close 93 (Seine-Saint-Denis: (2.20, 48.90). Longitude and latitude are better adapted as, say `longitude` < 2.50, has a geographical meaning. So `dept` was replaced by the longitude and latitude of the prefecture of each department (`longitude`, `latitude`), obtained from `SP532_villes_france.csv` and `depts2015.txt`. We compute three decision trees: one with both the INSEE and ONDRP variables (Figure 7.5), one with the INSEE variables only (Figure 7.6), and one where the commune level INSEE variables are replaced by their departmental means in order to be comparable to the ONDRP variables (Figure 7.7).

For the decision tree with both added INSEE and ONDRP variables in Figure 7.5, the geographical variables `longitude` and `latitude` are important as `dept` previously, though the ONDRP crime variables are more important here than the INSEE housing variables.

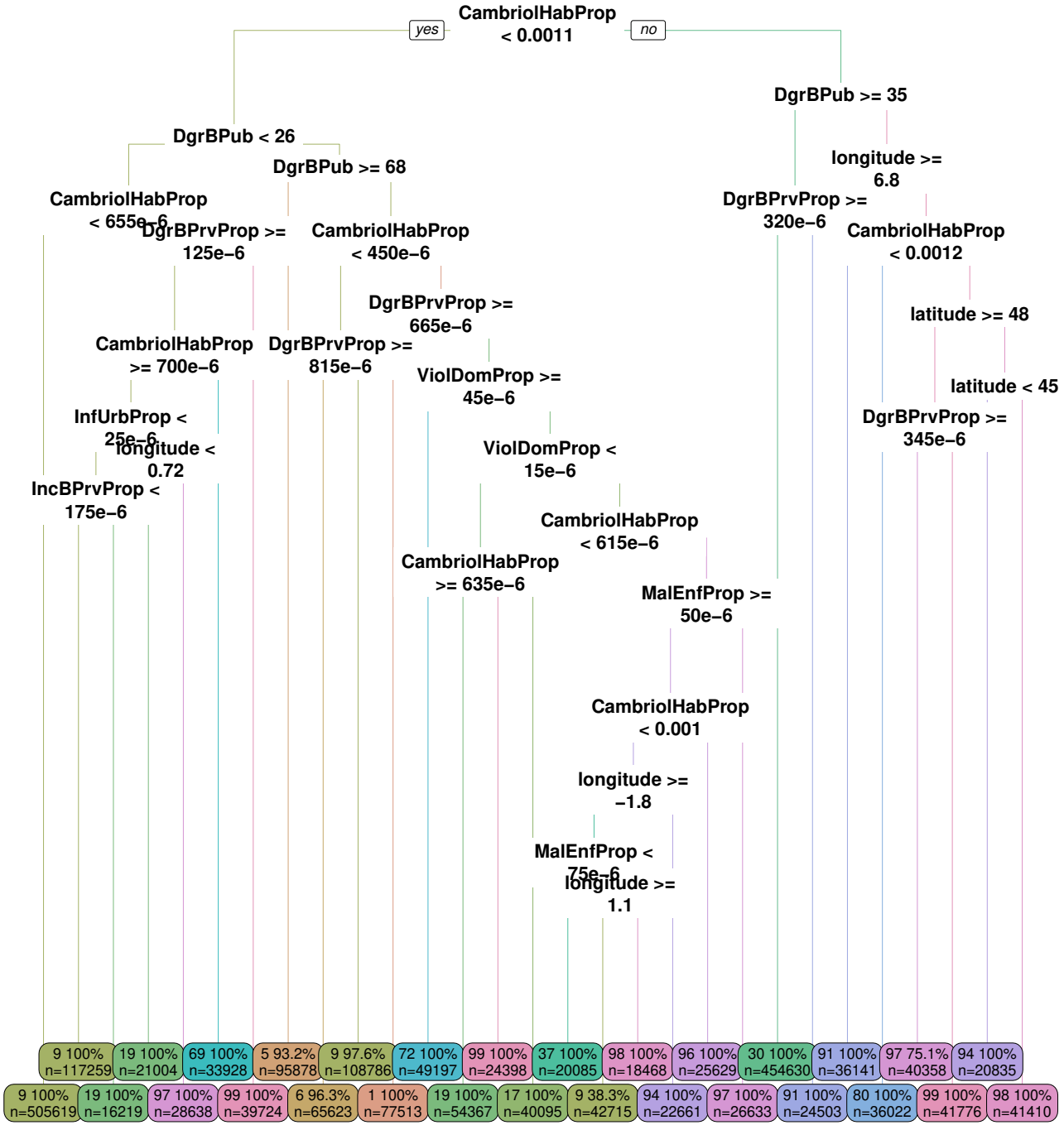


FIGURE 7.5: Decision tree for predicting the SOM cluster labels of the enriched AXA data: INSEE and ONDRP variables.

Each leaf node has a colour-coded label which denotes the estimated cluster label obtained by following this decision tree. For each leaf node is annotated with

the percentage of these contracts whose original SOM cluster label coincide with the estimated label and the number of contracts contained in the node (n).

The decision tree with the added INSEE variables only is shown in Figure 7.6, but only the AXA variables appear in the rules.

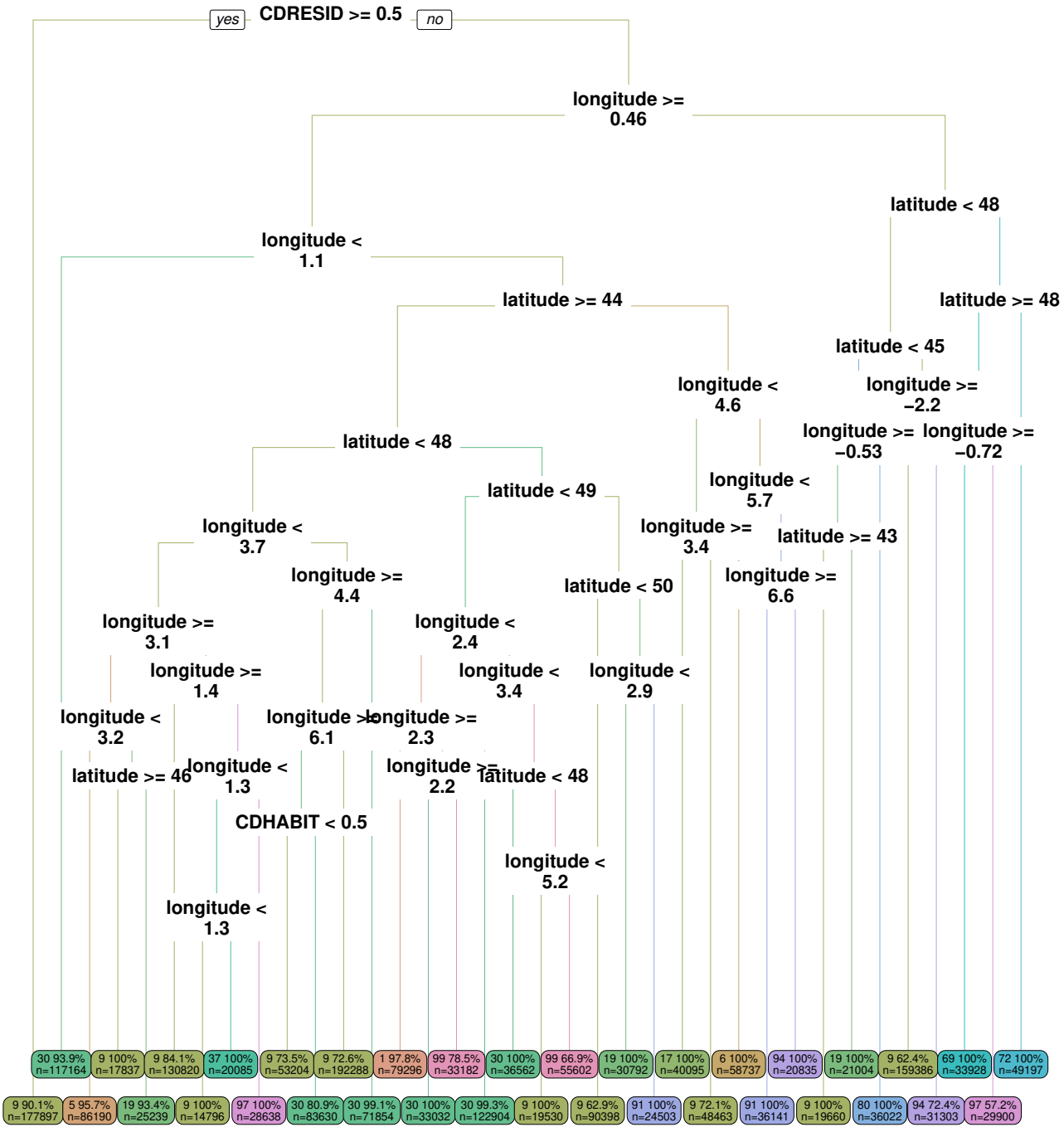


FIGURE 7.6: Decision tree for predicting the SOM cluster labels of the enriched AXA data: INSEE variables only.

Removing the ONDRP variables is too drastic in order to assess the influence of the INSEE variables. The ONDRP variables are available at the departmental level whereas the INSEE variables at the commune (INSEE code) level. In terms of finding groups of similar values, it is more likely to occur for the more aggregated ONDRP variables than the lower level INSEE ones. To remedy this, we replace the commune level INSEE variables with their mean aggregated at the department level with the suffix `_MOYD` indicating the *moyenne départementale*. The resulting decision tree is displayed in Figure 7.7, which shows that the INSEE (`cmbGazVilleProp_MOY`, `constAvant49_Prob_MOY` etc.), ONDRP (`cambiolHabProp`, `DgrBPub` etc.) and AXA (`longitude`, `latitude`) variables all play a role in the decision rules to predict the SOM cluster labels.

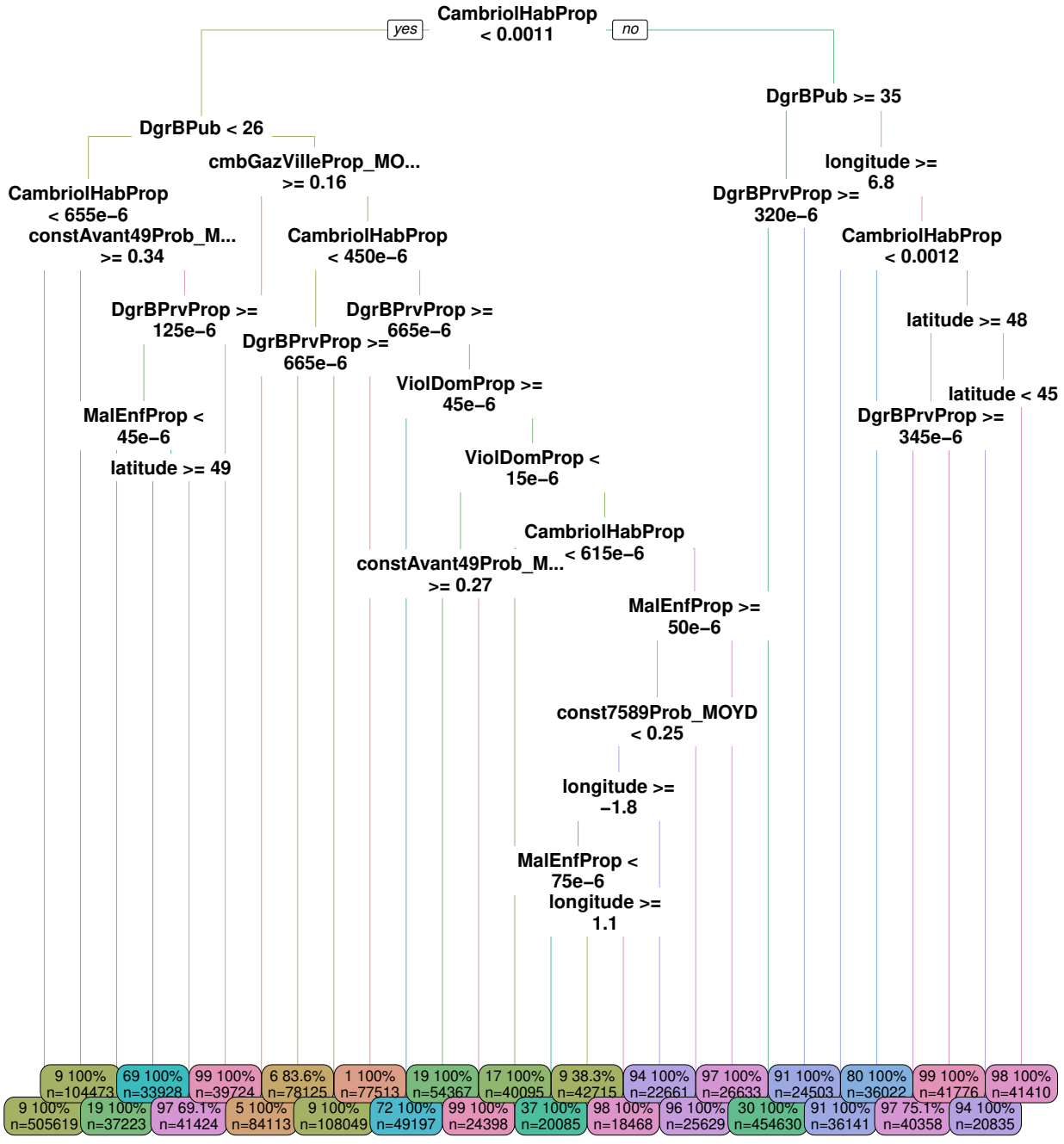


FIGURE 7.7: Decision tree for predicting the SOM cluster labels of the enriched AXA data: departemental mean of INSEE variables.

7.4 Validation of SOM clusters

The accuracy of these decision trees are quantified in the cross classification tables in Tables 7.1–7.3. Each row corresponds to the true SOM cluster label, and the columns to the estimated cluster label induced by the classification tree. The overall misclassification rates are the decision tree with the INSEE and ONDRP variables is 0.022 (Table 7.1), with the INSEE variables only (Table 7.2) 0.137, and with the INSEE departmental mean 0.042 (Table 7.3). Most of the non-zero entries are on the main diagonal, indicating that in the vast majority of cases the estimated labels from the decision tree coincide with the true SOM cluster labels. Notable departures are for classes 8, 68 and 79 which are never estimated by any of the rules for any of the decision trees. These three classes are the smallest by number of contracts (3 648, 13 536 and 12 799) but contain some of the largest individual payouts. So they are difficult to estimate when combined with the larger clusters e.g. clusters 9 and 19 with 760 677 and 454 630 contracts respectively.

True SOM cluster labels \ Estimated SOM cluster labels	1	5	6	8	9	17	19	30	37	68	69	72	79	80	91	94	96	97	98	99	Total
1	77513	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	77513
5	0	89311	667	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	89978
6	0	2097	63207	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	65304
8	0	0	1018	0	2630	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3648
9	0	4470	731	0	745414	0	0	0	0	0	0	0	0	0	0	0	0	10062	0	0	760677
17	0	0	0	0	0	40095	0	0	0	0	0	0	0	0	0	0	0	0	0	0	40095
19	0	0	0	0	0	0	91590	0	0	0	0	0	0	0	0	0	0	0	0	0	91590
30	0	0	0	0	0	0	0	454630	0	0	0	0	0	0	0	0	0	0	0	0	454630
37	0	0	0	0	0	0	0	0	20085	0	0	0	0	0	0	0	0	0	0	0	20085
68	0	0	0	0	13536	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13536
69	0	0	0	0	0	0	0	0	0	0	33928	0	0	0	0	0	0	0	0	0	33928
72	0	0	0	0	0	0	0	0	0	0	0	49197	0	0	0	0	0	0	0	0	49197
79	0	0	0	0	12799	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12799
80	0	0	0	0	0	0	0	0	0	0	0	0	0	36022	0	0	0	0	0	0	36022
91	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60644	0	0	0	0	0	60644
94	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	43496	0	0	0	0	43496
96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25629	0	0	0	25629
97	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	85567	0	0	85567
98	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	59878	0	59878
99	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	105898	105898
Total	77513	95878	65623	0	774379	40095	91590	454630	20085	0	33928	49197	0	36022	60644	43496	25629	95629	59878	105898	2130114

TABLE 7.1: Cross classification table for true and estimated SOM cluster labels from decision tree for enriched AXA data: INSEE and ONDRP variables. Overall misclassification rate is 0.022.

As the decision tree for the 2012 SOM clusters of the enriched AXA data with the departmental mean of INSEE variables, in Figure 7.7 and Table 7.3, is preferred as it utilises the AXA, INSEE and ONDRP variables, we validate it on test data that was not used in the training: the enriched contracts from previous years 2010, 2011.The same procedure for merging with the departmental mean of INSEE variables is carried out, and these merged data are classified according to the decision tree. The estimated SOM cluster labels are merged to these original data.

True SOM cluster labels \ Estimated SOM cluster labels	1	5	6	8	9	17	19	30	37	68	69	72	79	80	91	94	96	97	98	99	Total
1	77513	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	77513
5	0	82499	0	0	7479	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	89978
6	0	0	58737	0	6567	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	65304
8	0	0	0	0	3648	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3648
9	0	0	0	0	712044	0	0	23083	0	0	0	0	0	0	0	0	0	12786	0	12764	760677
17	0	0	0	0	0	40095	0	0	0	0	0	0	0	0	0	0	0	0	0	0	40095
19	0	0	0	0	16219	0	75371	0	0	0	0	0	0	0	0	0	0	0	0	0	91590
30	0	0	0	0	14028	0	0	440602	0	0	0	0	0	0	0	0	0	0	0	0	454630
37	0	0	0	0	0	0	0	0	20085	0	0	0	0	0	0	0	0	0	0	0	20085
68	0	0	0	0	13536	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13536
69	0	0	0	0	0	0	0	0	0	0	33928	0	0	0	0	0	0	0	0	0	33928
72	0	0	0	0	0	0	0	0	0	0	0	49197	0	0	0	0	0	0	0	0	49197
79	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12799	12799
80	0	0	0	0	0	0	0	0	0	0	0	0	0	36022	0	0	0	0	0	0	36022
91	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60644	0	0	0	0	0	60644
94	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	43496	0	0	0	0	43496
96	1783	3691	0	0	17030	0	1664	1461	0	0	0	0	0	0	0	0	0	0	0	0	25629
97	0	0	0	0	39815	0	0	0	0	0	0	0	0	0	0	0	0	45752	0	0	85567
98	0	0	0	0	59878	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	59878
99	0	0	0	0	34035	0	0	0	0	0	0	0	0	0	0	8642	0	0	0	63221	105898
Total	79296	86190	58737	0	924279	40095	77035	465146	20085	0	33928	49197	0	36022	60644	52138	0	58538	0	88784	2130114

TABLE 7.2: Cross classification table for true and estimated SOM cluster labels from decision tree for enriched AXA data: INSEE variables only. Overall misclassification rate is 0.137.

True SOM cluster labels \ Estimated SOM cluster labels	1	5	6	8	9	17	19	30	37	68	69	72	79	80	91	94	96	97	98	99	Total
1	77513	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	77513
5	0	84113	5865	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	89978
6	0	0	65304	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	65304
8	0	0	3648	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3648
9	0	0	3308	0	720933	0	0	0	0	0	1393	0	0	0	7421	0	2584	23294	0	1744	760677
17	0	0	0	0	0	40095	0	0	0	0	0	0	0	0	0	0	0	0	0	0	40095
19	0	0	0	0	0	0	91590	0	0	0	0	0	0	0	0	0	0	0	0	0	91590
30	0	0	0	0	0	0	0	454630	0	0	0	0	0	0	0	0	0	0	0	0	454630
37	0	0	0	0	0	0	0	0	20085	0	0	0	0	0	0	0	0	0	0	0	20085
68	0	0	0	0	13536	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13536
69	0	0	0	0	1393	0	0	0	0	0	32535	0	0	0	0	0	0	0	0	0	33928
72	0	0	0	0	0	0	0	0	0	0	0	49197	0	0	0	0	0	0	0	0	49197
79	0	0	0	0	12799	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12799
80	0	0	0	0	0	0	0	0	0	0	0	0	0	36022	0	0	0	0	0	0	36022
91	0	0	0	0	7421	0	0	0	0	0	0	0	0	0	53223	0	0	0	0	0	60644
94	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	43496	0	0	0	0	43496
96	0	0	0	0	2584	0	0	0	0	0	0	0	0	0	0	0	23045	0	0	0	25629
97	0	0	0	0	446	0	0	0	0	0	0	0	0	0	0	0	0	85121	0	0	85567
98	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	59878	0	59878
99	0	0	0	0	1744	0	0	0	0	0	0	0	0	0	0	0	0	0	0	104154	105898
Total	77513	84113	78125	0	760856	40095	91590	454630	20085	0	33928	49197	0	36022	60644	43496	25629	108415	59878	105898	2130114

TABLE 7.3: Cross classification table for true and estimated SOM cluster labels from decision tree for enriched AXA data: departmental mean of INSEE variables. Overall misclassification rate is 0.042.

Since we do not have the true cluster labels for 2010, 2011, to validate the decision tree on these data, we compare the summary statistics for the number of contracts (n) and the total of the claims for fire (INC) and water (DDE) damages per SOM cluster. In Table 7.4 are four groups of three columns: the rightmost are the 2012 data with the true SOM clusters which serves as our baseline, the third are the same 2012 data but re-classified using the decision tree, the second and first are for the 2011 and 2010 data.

A comparison of the distributions of the summary statistics is easier with the graphical bar charts in Figure 7.8. Within each cluster, there are four bars: the violet are the 2012 data with the true SOM clusters, the turquoise is the 2012 data re-classified using the decision tree, the green is for 2011, and the orange is for 2010.

SOM cluster labels	2010 estimated			2011 estimated			2012 estimated			2012 true		
	n	INC	DDE	n	INC	DDE	n	INC	DDE	n	INC	DDE
1	71417	2208521	288517	70375	3453512	311125	77513	3554052	503184	77513	3554052	503184
5	0	0	0	0	0	0	84113	3805694	609413	89978	3832070	698697
6	113808	3044936	888294	111498	2733776	616970	78125	2421022	1109152	65304	1779748	833714
8	0	0	0	0	0	0	0	0	0	3648	540020	131282
9	587543	26013920	3685580	584201	28928093	2564389	760856	36991021	5558205	760677	36681639	5689450
17	36636	1517202	148858	36383	405406	193108	40095	1420326	253458	40095	1420326	253458
19	81504	1745568	420214	82415	2937252	393204	91590	3348962	559426	91590	3348962	559426
30	342172	13098008	3278232	334491	10695066	2005539	454630	14032596	5975661	454630	14032596	5975661
37	18157	613664	132921	17941	1187235	63110	20085	1389783	214435	20085	1389783	214435
68	0	0	0	0	0	0	0	0	0	13536	142733	74433
69	30224	556372	129254	30357	523178	119306	33928	1139292	234007	33928	1134082	242247
72	47858	1764099	165154	47206	1065009	168921	49197	1633163	277043	49197	1633163	277043
79	0	0	0	0	0	0	0	0	0	12799	284444	88787
80	32849	986640	308077	32448	854509	274218	36022	902039	414788	36022	902039	414788
91	48696	1674154	625144	47989	1981868	470894	60644	2306262	780464	60644	2306262	780464
94	39945	1911866	393761	39455	1173618	351561	43496	1572844	355461	43496	1572844	355461
96	24081	710802	262052	23473	601661	213970	25629	730088	205032	25629	928012	198457
97	82983	2845194	785122	81852	3987037	740369	108415	5256038	931430	85567	4337554	724834
98	56417	2158410	323377	55273	3027478	256419	59878	4795303	526774	59878	4795303	526774
99	74744	3242670	423559	74276	3178022	369088	105898	4109538	762898	105898	4792391	728237
Total	1689034	64092025	12258117	1669633	66732721	9112191	2130114	89408023	19270831	2130114	89408023	19270831

TABLE 7.4: Validation of SOM cluster labels from decision tree for enriched AXA 2012 data with departmental mean of INSEE variables. Validation data are contracts from 2010, 2011 and 2012. INC is fire damages claims (euros), DDE is water damages claims (euros). Summary statistics are the number of contracts n, total fire damages and total water damages claims.

From Table 7.4, for 2010, 2011 there are 1.60 and 1.68 million contracts, €64.1M and €66.7M of fire damages, and €12.2M and €9.11M of water damages. In comparison for 2012, there are 2.1 million contracts, €89.4M of fire and €19.3M of water damages. So we can expect that the heights of the bars for the former to be around 20%, 20% and 40–50% lower than the latter. Taking these into account, the match between the number of contracts is good, especially for the highest bars in clusters 9 and 30. For the fire damages, cluster 30 is proportionally over-represented for 2010, 2011, but nonetheless does not exceed the 2012 level. For the water damages, clusters 9, 30 for 2011 appears to be under-represented and cluster 9 is over-represented for 2010, in comparison to 2012. Overall the SOM cluster labels from 2012 are validated for clustering the 2010, 2011 data in terms of the number of contracts and fire damages, but less so for the water damages.

7.5 Analysis of the insurance big data using SOM-MR

To further analyze clusters, we use the following 3 indicators: rate of claims, payouts per contract, and loss per contract.

$$Rate\_of\_claims = \frac{Number\_of\_claims}{Number\_of\_contracts} \tag{7.1}$$

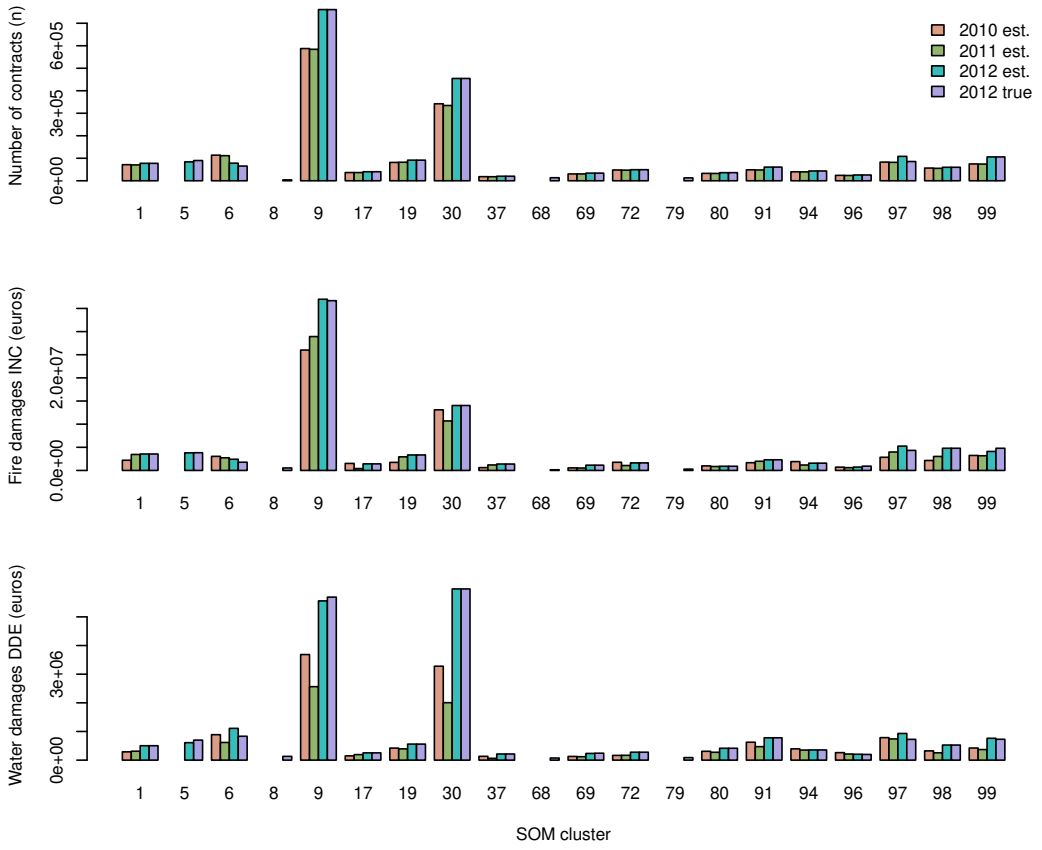


FIGURE 7.8: Validation of SOM cluster labels from decision tree for enriched AXA 2012 data with departmental mean of INSEE variables. Validation data are contracts from 2010 (orange), 2011 (green) and 2012 (turquoise). The 2012 data with true SOM clusters are violet. INC is fire damages claims (euros), DDE is water damages claims (euros). Summary statistics are the number of contracts n, total fire damages and total water damages claims.

$$Payout\_per\_claim = \frac{Sum\_of\_claim\_amounts}{Number\_of\_claims} \tag{7.2}$$

$$Loss\_per\_contract = Rate\_of\_claims * Payout\_per\_claim \tag{7.3}$$

Regarding these indicators, especially the maximum and minimum values, the insurance company can focuse its analysis on the corresponding clusters. Thus, a model based on the features of assigned data can be defined. Using this model, the insurance company can predict the payouts for a new customer within a cluster and so propose more personalised insurance contracts for its customers.

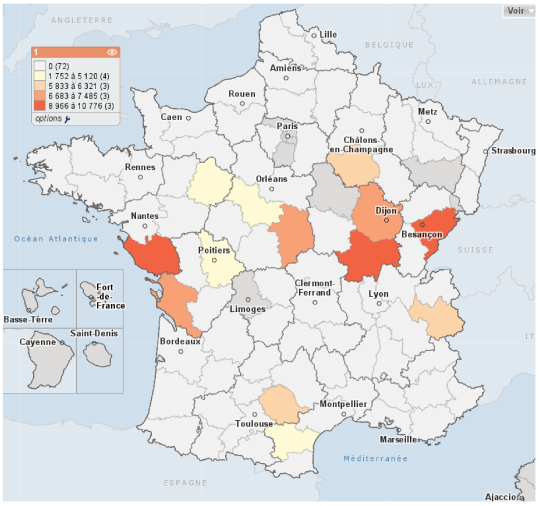
Table 7.5 summarizes the values of these indicators for clusters: 1, 66, 55, 21, and 47. We distinguish two types of claims: claim water damage (WAT) and claim

fire damage (FIR).

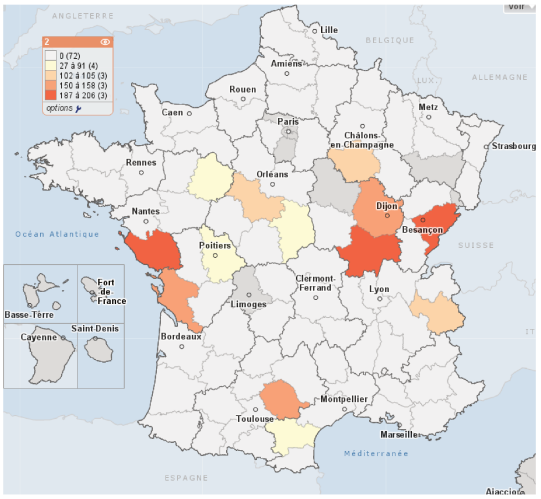
Cluster	Rate_WAT	Rate_FIR	Payout_WAT	Payout_FIR	Loss_WAT	Loss_FIR
1	0.0126	0.0029	1706.4	22305	21.45	64.25
66	0.0235	0.0022	874.91	6249.2	20.58	13.71
55	0.0407	0.0016	886.3	4676.9	36.11	7.33
21	0.0277	0.0027	1276	12552	35.34	33.51
47	0.040	0.0029	1066.3	2658.9	42.64	7.80

TABLE 7.5: Rate of claims, Payout per claim, and Loss per contract for batch-Stream clusters for insurance data

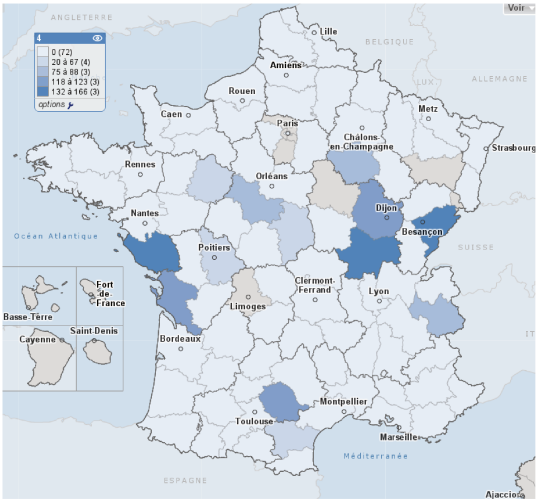
Figues 7.9 and 7.10 show a visualization of contracts, assigned to clusers 21 and 55, on the map of France by department.



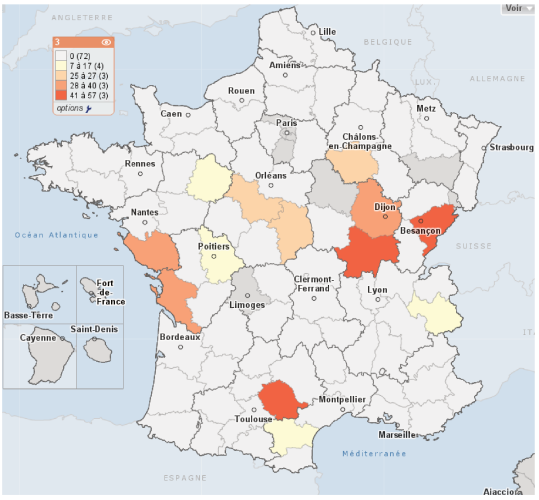
(a) All contrats



(b) Claimed contracts

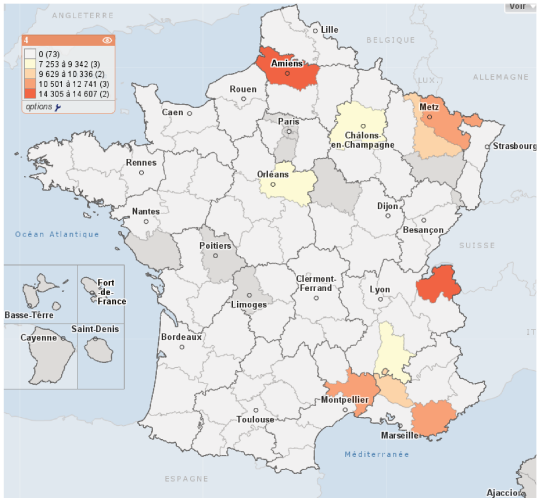


(c) Claimed contracts WAT

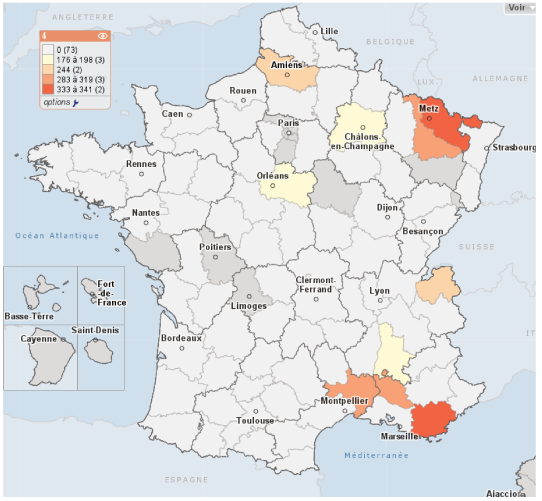


(d) Claimed contracts FIR

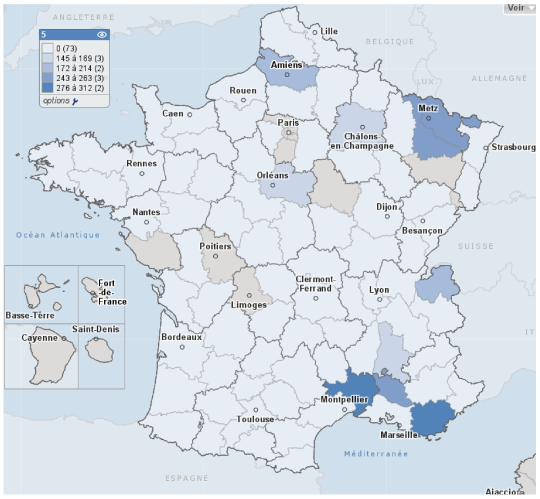
FIGURE 7.9: Visualtisation of contracts assigned to cluster #21



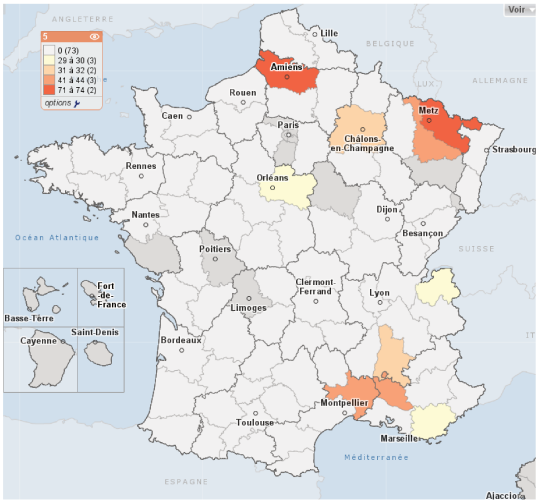
(a) All contrats



(b) Claimed contracts



(c) Claimed contracts WAT



(d) Claimed contracts FIR

FIGURE 7.10: Visualtization of contracts assigned to cluster #55

7.6 Conclusion

In this chapter, we presented our work carried in the context of an insurance Big Data project. We have illustrated the utility of the SOM-MR algorithm (which is presented in the chapter 5) as an unsupervised learning for the insurance Big Data was presented. We plan in the future to extend SOM-MR to deal with binary, categorical, and mixed data, and also to make our algorithm as autonomous as possible.

Chapter 8

Conclusion and perspectives

After summarizing the key issues touched upon this work, the next section discusses the main research avenues open for further work.

8.1 Summary

The first chapters were devoted to giving a state-of-the-art on both clustering and scalable methods using the MapReduce paradigm and clustering data streams as well as a survey on bi-clustering methods and an introduction to the Big Data ecosystem.

Our first contribution is concerned with extending the SOM method for scalability seeks. The proposed SOM-MR algorithm is implemented with the Spark framework which represents a new way of writing using the MapReduce paradigm. The major research challenge addressed is how to minimize the input and output of primitives (map and reduce) for topological clustering algorithm. So, we show that we can save computation time by changing the (key, value) parameters.

Afterwards, in the second contribution, we presented the BiTM distributed algorithm for scalable bi-clustering based on topological maps. We defined a new cost function and so a new formalization of topological bi-clustering. After that, we proposed a model for scalability. This model consists of decomposing the db-clustering problem into the elementary functions, Map and Reduce.

Then, we presented our work carried in the context of an insurance Big Data project [1]. We applied the SOM-MR method to cluster the insurance dataset

[1] http://ns209168.ovh.net/squarepredict/

merged with open data. After that, we illustrated the utility of the SOM-MR algorithm as an unsupervised learning by analyzing the cluster results and combining them with a supervised method.

8.2 Perspectives

8.2.1 Biclustering and feature group weighting

In the following, we denote a matrix by bold capital letters such as $\mathbf{G}$. Vectors are denoted by small boldface letters such as $\mathbf{g}$ and matrix and vector elements are represented respectively by small letters such as g_i^j. As traditional self-organizing maps, which is increasingly used as tools for clustering and visualization, wBiTM consists of a discrete set of cells $\mathcal{C}$ called map with K cells. This map has a discrete topology defined as an undirected graph, it is usually a regular grid in 2 dimensions. For each pair of cells (c, r) on the map, the distance $\delta(c, r)$ is defined as the length of the shortest chain linking cells r and c on the grid. For each cell c, this distance defines a neighbor cell.

Let $\Re^d$ be the euclidean data space and $\mathbf{D}$ the matrix of data, where each observation $\mathbf{x}_i = (x_i^1, x_i^2, ..., x_i^j, .., x_i^d)$ is a vector in $\mathbf{D} \subset \Re^d$. The set of rows (observations) is denoted by $I = \{1, ..., N\}$. Similarly, the set of columns (features) is denoted by $J = \{1, ..., d\}$. We are interested in simultaneously clustering observation I into K clusters $\{P_1, P_2, ..., P_k, .., P_K\}$, where $P_k = \{\mathbf{x}_i, \phi_z(\mathbf{x}_i) = k\}$ and features J into L clusters $\{Q_1, Q_2, ..., Q_l, .., Q_L\}$ where $Q_l = \{\mathbf{x}^j, \phi_w(\mathbf{x}^j) = l\}$. We denote by ϕ_z the assignment function of row (observation) and ϕ_w the assignment function of column (feature).

The main purpose of wBiTM is to transform a data matrix $\mathbf{D}$ into a block structure organized in a topological map. In wBiTM, each cell $r \in \mathcal{C}$ is associated with a prototype $\mathbf{g}_k = (g_k^1, g_k^2..., g_k^l, ..., g_k^L)$, and weight vector $\pi_k = (\pi_k^1, \pi_k^2..., \pi_k^l, ..., \pi_k^L)$ where $L < d$ and $g_k^l \in \Re$. $\mathbf{G} = \{\mathbf{g}_1, ..., \mathbf{g}_k\}$ and $\mathbf{\Pi} = \{\pi_1, ..., \pi_k\}$ denotes respectively the set of prototype and the weight vector. To facilitate formulation, we define two binary matrices $\mathbf{Z} = [z_i^k]$ and $\mathbf{W} = [w_j^l]$ to save the assignment associated respectively to observations and features:

$$z_i^k = \begin{cases} 1 & \text{if } \mathbf{x}_i \in P_k, \\ 0 & \text{else} \end{cases}$$

$$w_j^l = \begin{cases} 1 & \text{if } \mathbf{x}^j \in Q_l \\ 0 & \text{else} \end{cases}$$

Without using the weight parameter, biclustering topological maps, proposes to minimize the following cost function:

$$\mathcal{R}_{BiTM}(\mathbf{W}, \mathbf{Z}, \mathbf{G}) = \sum_{k=1}^{K}\sum_{l=1}^{L}\sum_{i=1}^{N}\sum_{j=1}^{d}\sum_{r=1}^{K} \mathcal{K}^T(\delta(r,k)) z_i^k w_j^l (x_i^j - g_r^l)^2$$

wBiTM extends the cost function with additional parameter π to control the feature group weights at each iteration of the biclustering process. To cluster $\mathbf{D}$ into K and L clusters in both observations and features, we propose the new following objective function to optimize in the biclustering process of wBiTM:

$$\mathcal{R}_{wBiTM}(\mathbf{W}, \mathbf{Z}, \mathbf{G}, \mathbf{\Pi}) = \sum_{k=1}^{K}\sum_{l=1}^{L}\sum_{i=1}^{N}\sum_{j=1}^{d}\sum_{r=1}^{K} \mathcal{K}^T(\delta(r,k)) z_i^k w_j^l (\pi_r^l x_i^j - g_r^l)^2 \quad (8.1)$$

This cost function can be written as follows:

$$\mathcal{R}_{wBiTM}(\mathbf{W}, \mathbf{Z}, \mathbf{G}, \mathbf{\Pi}) = \sum_{k=1}^{K}\sum_{l=1}^{L}\sum_{x_i^j \in B_k^l}\sum_{r=1}^{K} \mathcal{K}^T(\delta(r,k)) (\pi_r^l x_i^j - g_r^l)^2$$

We can detect the bicluster of data denoted by $B_k^l = \{x_i^j | z_i^k w_j^l = 1\}$. Typically the neighborhood function $\mathcal{K}^T(\delta) = \mathcal{K}(\delta/T)$ is a positive function, which decreases as the distance between two cells in the latent space $\mathcal{C}$ increases and where T controls the width of the neighborhood function.Thus T is decreased between two values T_{max} and T_{min}. In practice, we use the neighborhood function defined as $\mathcal{K}^T(\delta(c,r)) = \exp\left(\frac{-\delta(c,r)}{T}\right)$ and $T = T_{max}(\frac{T_{min}}{T_{max}})^{\frac{t}{t_f - 1}}$, where t is the current epoch and t_f the number of epoch.

The objective function (Eq. 8.1) can be locally minimized by iteratively solving the following three minimization problems:

- Problem 1: Fix $\mathbf{G} = \hat{\mathbf{G}}$, $\mathbf{W} = \hat{\mathbf{W}}$ and $\mathbf{\Pi} = \hat{\mathbf{\Pi}}$, solve the reduced problem $\mathcal{R}_{wBiTM}(\hat{\mathbf{W}}, \mathbf{Z}, \hat{\mathbf{G}}, \hat{\mathbf{\Pi}})$;
- Problem 2: Fix $\mathbf{G} = \hat{\mathbf{G}}$, $\mathbf{Z} = \hat{\mathbf{Z}}$ and $\mathbf{\Pi} = \hat{\mathbf{\Pi}}$, solve the reduced problem $\mathcal{R}_{wBiTM}(\mathbf{W}, \hat{\mathbf{Z}}, \hat{\mathbf{G}}, \hat{\mathbf{\Pi}})$;

- Problem 3: Fix $\mathbf{W}$, $\mathbf{Z}$ and $\mathbf{\Pi} = \hat{\mathbf{\Pi}}$, solve the reduced problem $\mathcal{R}_{wBiTM}(\hat{\mathbf{W}}, \hat{\mathbf{Z}}, \mathbf{G}, \hat{\mathbf{\Pi}})$;
- Problem 4: Fix $\mathbf{W} = \hat{\mathbf{W}}$, $\mathbf{Z} = \hat{\mathbf{Z}}$ and $\mathbf{G} = \hat{\mathbf{G}}$, solve the reduced problem $\mathcal{R}_{wBiTM}(\hat{\mathbf{W}}, \hat{\mathbf{Z}}, \hat{\mathbf{G}}, \mathbf{\Pi})$;

In order to reduce the computational time, we assign each observation and feature without using neighborhood cell as the traditional topological map.

Problem 1 is solved by defining z_i^k as:

$$z_i^k = \begin{cases} 1 & \text{if } \mathbf{x}_i \in P_k, k = \phi_z(\mathbf{x}_i) \\ 0 & \text{else} \end{cases}$$

Where each observation $\mathbf{x}_i$ is assigned to the closest prototype $\mathbf{g}_k$ using the assignment function, defined as follows:

$$\phi_z(\mathbf{x}_i) \quad = \quad \arg\min_c \sum_{j=1}^{d} \sum_{l=1}^{L} w_j^l (\pi_c^l x_i^j - g_c^l)^2 \tag{8.2}$$

Problem 2 is solved by defining w_j^l as

$$w_j^l = \begin{cases} 1 & \text{if } \mathbf{x}^j \in Q_l, l = \phi_w(\mathbf{x}^j) \\ 0 & \text{else} \end{cases}$$

Where each feature $\mathbf{x}^j$ is assigned to the closest prototype $\mathbf{g}^l$ using the assignment function, defined as follows:

$$\phi_w(\mathbf{x}^j) \quad = \quad \arg\min_l \sum_{i=1}^{N} \sum_{k=1}^{K} z_i^k (\pi_r^l x_i^j - g_r^l)^2 \tag{8.3}$$

Problem 3 is resolved for the numerical features by :

$$g_r^l = \frac{\sum_{k=1}^{K} \sum_{i=1}^{N} \sum_{j=1}^{d} \mathcal{K}^T(\delta(k,r) z_i^k w_j^l \pi_r^l x_i^j}{\sum_{k=1}^{K} \sum_{i=1}^{N} \sum_{j=1}^{d} \mathcal{K}^T(\delta(k,r)) z_i^k w_j^l}$$

This value is obtained by resolving the gradients $\frac{\partial \mathcal{R}_{wBiTM}}{\partial g_r^l} = 0$

$$g_r^l = \frac{\sum_{k=1}^{K} \mathcal{K}^T(\delta(k,r)) \sum_{i=1}^{N} \sum_{j=1}^{d} z_i^k w_j^l \pi_r^l x_i^j}{\sum_{k=1}^{K} \mathcal{K}^T(\delta(k,r)) \sum_{i=1}^{N} \sum_{j=1}^{d} z_i^k w_j^l} \tag{8.4}$$

For the problem 4, the component π_r^l of $\Pi = (\pi_r^1, \pi_r^2, ..., \pi_r^l, ..., \pi_r^d)$ is computed as follows:

$$\pi_r^l = \frac{\sum_{i=1}^{N} \sum_{j=1}^{d} \mathcal{K}^T(\delta(r, \phi_z(\mathbf{x}_i))) w_j^l x_i^j g_r^l}{\sum_{i=1}^{N} \sum_{j=1}^{d} \mathcal{K}^T(\delta(r, \phi_z(\mathbf{x}_i))) w_j^l (x_i^j)^2} \tag{8.5}$$

This value is obtained by resolving the gradients $\frac{\partial \mathcal{R}_{wBiTM}}{\partial \pi_r^l} = 0$.

8.2.2 Conclusion

In this perspective work, we have proposed a new feature group weighting using biclustering topological maps approach. The main novelty of our model is the use of topological model to organize the data matrix into homogeneous biclusters by considering simultaneously rows and columns, and learning new parameter of feature group weighting. A series of experiments are conducted to validate the proposed method. Experimental results demonstrate that our algorithm is promising and identify meaningful biclusters. Our algorithm inherits all the classical visualization of topological maps and provides a new visualizations to better data understanding. In future work, we will test and improve our method on further real applications. Further investigation is necessary to understand the relationship between weight feature group and feature group selection.

Bibliography

T. Kohonen. *Self-organizing Maps*. Springer Berlin, 2001.

Yuri Demchenko, Paola Grosso, Cees De Laat, and Peter Membrey. Addressing big data issues in scientific data infrastructure. In *Collaboration Technologies and Systems (CTS), 2013 International Conference on*, pages 48–55. IEEE, 2013.

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012*, pages 15–28, 2012.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA*, pages 226–231, 1996.

Charu C. Aggarwal and Chandan K. Reddy. *Data Clustering: Algorithms and Applications*. CRC Press, 2014.

Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

Brian S. Everitt, Sabine Landau, and Morven Leese. *Cluster Analysis*. Wiley Publishing, 4th edition, 2009. ISBN 0340761199, 9780340761199.

John R. Mashey. Big data and the next wave of infrastress problems, solutions, opportunities. 1998.

Wei Fan and Albert Bifet. Mining big data: current status, and forecast to the future. *ACM sIGKDD Explorations Newsletter*, 14(2):1–5, 2013.

Douglas Laney. 3D data management: Controlling data volume, velocity, and variety. Technical report, META Group, February 2001.

John Gantz and David Reinsel. Extracting value from chaos. *IDC iview*, 1142: 1–12, 2011.

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *ACM SIGOPS operating systems review*, volume 37, pages 29–43. ACM, 2003.

Mike Burrows. The chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 335–350. USENIX Association, 2006.

Dhruba Borthakur. The hadoop distributed file system: Architecture and design. *Hadoop Project Website*, 11(2007):21, 2007.

Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008a.

Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010a. USENIX Association.

Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988. ISBN 0-13-022278-X.

Samuel Kaski, Jari Kangas, and Teuv Kohonen. Bibliography of self-organizing map (som) papers: 1981-1997. *Neural computing surveys*, 1:102–350, 1998.

Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1998. ISBN 0132733501.

T. Kohonen, M. R. Schroeder, and T. S. Huang, editors. *Self-Organizing Maps*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 3rd edition, 2001a. ISBN 3540679219.

T. Martinetz and K. Schulten. A "Neural-Gas" Network Learns Topologies. *Artificial Neural Networks*, I:397–402, 1991.

Bernd Fritzke. Unsupervised clustering with growing cell structures. In *In Proceedings of the International Joint Conference on Neural Networks*, pages 531–536. IEEE, 1991.

Bernd Fritzke. A growing neural gas network learns topologies. In *NIPS*, pages 625–632, 1994.

Oliver Beyer and Philipp Cimiano. Online semi-supervised growing neural gas. *Int. J. Neural Syst.*, 22(5), 2012.

Amineh Amini, Ying Wah Teh, and Hadi Saboohi. On density-based data streams clustering algorithms: A survey. *J. Comput. Sci. Technol.*, 29(1):116–141, 2014.

Chris Fraley and Adrian E Raftery. How many clusters? which clustering method? answers via model-based cluster analysis. *The computer journal*, 41(8):578–588, 1998.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977a.

Geoffrey McLachlan and Thriyambakam Krishnan. *The EM algorithm and extensions*, volume 382. John Wiley & Sons, 2007.

Ali El Attar, Antoine Pigeau, and Marc Gelgon. Robust estimation of a global gaussian mixture by decentralized aggregations of local models. *Web Intelligence and Agent Systems*, 11(3):245–262, 2013. doi: 10.3233/WIA-130273. URL http://dx.doi.org/10.3233/WIA-130273.

Ali El Attar. *Estimation robuste des modèles de mélange sur des données distribuées*. Theses, Université de Nantes, July 2012. URL https://tel.archives-ouvertes.fr/tel-00746118.

Ralf Lämmel. Google's mapreduce programming model—revisited. *Science of computer programming*, 70(1):1–30, 2008.

Eshref Januzaj, Hans-Peter Kriegel, and Martin Pfeifle. Dbdc: Density based distributed clustering. In *Advances in Database Technology-EDBT 2004*, pages 88–105. Springer, 2004.

Tugdual Sarazin, Hanane Azzag, and Mustapha Lebbah. SOM clustering using spark-mapreduce. In *2014 IEEE International Parallel & Distributed Processing Symposium Workshops, Phoenix, AZ, USA, May 19-23, 2014*, pages 1727–1734, 2014.

Weizhong Zhao, Huifang Ma, and Qing He. Parallel k-means clustering based on mapreduce. In *Cloud computing*, pages 674–679. Springer, 2009.

Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10. IEEE, 2010.

Yaobin He, Haoyu Tan, Wuman Luo, Shengzhong Feng, and Jianping Fan. Mr-dbscan: a scalable mapreduce-based dbscan algorithm for heavily skewed data. *Frontiers of Computer Science*, 8(1):83–99, 2014.

Ali Seyed Shirkhorshidi, Saeed Aghabozorgi, Teh Ying Wah, and Tutut Herawan. Big data clustering: a review. In *Computational Science and Its Applications–ICCSA 2014*, pages 707–720. Springer, 2014.

Abhinandan S Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*, pages 271–280. ACM, 2007.

Henggang Cui, Jinliang Wei, and Wei Dai. Parallel implementation of expectation-maximization for fast convergence.

Aniruddha Basak, Irina Brinster, and Ole J Mengshoel. Mapreduce for bayesian network parameter learning using the em algorithm. *Proc. of Big Learning: Algorithms, Systems and Tools*, 2012.

Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii. Scalable k-means++. *Proceedings of the VLDB Endowment*, 5(7): 622–633, 2012.

Alina Ene, Sungjin Im, and Benjamin Moseley. Fast clustering using mapreduce. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 681–689. ACM, 2011a.

G. Govaert. *Classification croisée*. PhD thesis, Université Paris 6, France, 1983.

Ahsan Abdullah and Amir Hussain. A new biclustering technique based on crossing minimization. *Neurocomputing*, 69(16):1882–1896, 2006.

Gérard Govaert and Mohamed Nadif. Un modèle de mélange pour la classification croisée d'un tableau de données continue. In *CAP'09, 11e conférence sur l'apprentissage artificiel*, pages 287–302, 2009.

Bongjune Kwon and Hyuk Cho. Scalable co-clustering algorithms. In *International Conference on Algorithms and Architectures for Parallel Processing*, pages 32–43. Springer, 2010.

Wassim Ayadi, Mourad Elloumi, and Jin-Kao Hao. Pattern-driven neighborhood search for biclustering of microarray data. *BMC bioinformatics*, 13(7):1, 2012.

Fabrício Olivetti de França, Guilherme Palermo Coelho, and Fernando J Von Zuben. Predicting missing values with biclustering: A coherence-based approach. *Pattern Recognition*, 46(5):1255–1266, 2013.

J. A. Hartigan. Direct clustering of a data matrix. *Journal of the American Statistical Association*, 67(337):123–129.

Amos Tanay, Roded Sharan, and Ron Shamir. Discovering statistically significant biclusters in gene expression data. In *In Proceedings of ISMB 2002*, pages 136–144, 2002.

D. Greene and P. Cunningham. Spectral co-clustering for dynamic bipartite graphs. In *Workshop on dynamic networks and knowledge discovery at ecml'10, barcelona, spain*, 2010.

Hanhuai Shan, , and Arindam Banerjee. Residual bayesian co-clustering for matrix approximation. In *SDM*, pages 223–234, 2010.

Fabrizio Angiulli, Eugenio Cesario, and Clara Pizzuti. A greedy search approach to co-clustering sparse binary matrices. In *ICTAI*, pages 363–370. IEEE Computer Society, 2006.

Malika Charrad, Y. Lechevallier, G. Saporta, and M Ben Ahmed. Le bi-partitionnement: Etat de l'art sur les approches et les algorithmes. 2008.

Xavier Jollois. *Contribution de la classification automatique à la Fouille de Données*. Dordrecht, France, 2003.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977b.

T. Kohonen, M. R. Schroeder, and T. S. Huang, editors. *Self-Organizing Maps*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 3rd edition, 2001b.

Stanislav Busygin, Gerrit Jacobsen, Ewald Kremer, and Contentsoft Ag. Double conjugated clustering applied to leukemia microarray data. In *In 2nd SIAM ICDM, Workshop on clustering high dimensional data*, 2002.

Marie Cottrell, Smail Ibbou, and Patrick Letrémy. Som-based algorithms for qualitative variables. *Neural Netw.*, 17(8-9):1149–1167, October 2004. ISSN 0893-6080. doi: 10.1016/j.neunet.2004.07.010. URL http://dx.doi.org/10.1016/j.neunet.2004.07.010.

K. Benabdeslem and K. Allab. Bi-clustering continuous data with self-organizing map. Neural Computing and Applications, 2012.

José Caldas and Samuel Kaski. Hierarchical generative biclustering for microrna expression analysis. *Journal of Computational Biology*, 18(3):251–261, 2011.

DQ Mao, Yi Luo, JH Zhang, and Jun Zhu. A new strategy of cooperativity of biclustering and hierarchical clustering: A case of analyzing yeast genomic microarray datasets. *Front. Biosci*, 10:1619–1627, 2005.

G Getz, E Levine, and E Domany. Coupled two-way clustering analysis of gene microarray data. *Proceedings of the National Academy of Sciences of the United States of America*, 97(22):12079–12084, 2000a. URL http://arxiv.org/abs/physics/0004009.

G. Getz, E. Levine, E. Domany, and M. Q. Zhang. Super paramagnetic clustering of yeast gene expression profiles, 2000b.

Yizong Cheng and George M. Church. Biclustering of expression data, 2000.

Amir Ben-Dor, Benny Chor, Richard Karp, and Zohar Yakhini. Discovering local structure in gene expression data: the order-preserving submatrix problem. *Journal of computational biology*, 10(3-4):373–384, 2003.

Laura Lazzeroni and Art Owen. Plaid models for gene expression data. *Statistica Sinica*, 12:61–86, 2000.

Bo Long, Zhongfei (Mark) Zhang, and Philip S. Yu. Co-clustering by block value decomposition. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, KDD '05, pages 635–640, New

York, NY, USA, 2005. ACM. ISBN 1-59593-135-X. doi: 10.1145/1081870.1081949. URL http://doi.acm.org/10.1145/1081870.1081949.

Jiho Yoo and Seungjin Choi. Orthogonal nonnegative matrix tri-factorization for co-clustering: Multiplicative updates on stiefel manifolds. *Inf. Process. Manage.*, 46(5):559–570, September 2010. ISSN 0306-4573. doi: 10.1016/j.ipm.2009.12.007. URL http://dx.doi.org/10.1016/j.ipm.2009.12.007.

Lazhar Labiod and Mohamed Nadif. Co-clustering under nonnegative matrix tri-factorization. In *Proceedings of the 18th international conference on Neural Information Processing - Volume Part II*, ICONIP'11, pages 709–717, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-24957-0. doi: 10.1007/978-3-642-24958-7_82. URL http://dx.doi.org/10.1007/978-3-642-24958-7_82.

Fanhua Shang, L. C. Jiao, and Fei Wang. Graph dual regularization non-negative matrix factorization for co-clustering. *Pattern Recogn.*, 45(6):2237–2250, June 2012. ISSN 0031-3203. doi: 10.1016/j.patcog.2011.12.015. URL http://dx.doi.org/10.1016/j.patcog.2011.12.015.

D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788, 1999.

Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Raphaël K Freitas. *K-théorie réelle des variétés de Stiefel sans torsion*. PhD thesis, Lille 1, 1985.

A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Comput. Surv.*, 31(3):264–323, September 1999. doi: 10.1145/331499.331504.

Moses Charikar, Chandra Chekuri, Tomás Feder, and Rajeev Motwani. Incremental clustering and dynamic information retrieval. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, STOC '97, pages 626–635, New York, NY, USA, 1997. ACM. ISBN 0-89791-888-6. doi: 10.1145/258533.258657.

Richard Matthew Mccutchen and Samir Khuller. Streaming algorithms for k-center clustering with outliers and with anonymity. In *Proceedings of the 11th international workshop, APPROX 2008, and 12th international workshop, RANDOM 2008 on Approximation, Randomization and Combinatorial Optimization: Algorithms and Techniques*, APPROX '08 / RANDOM '08, pages 165–178, Berlin, Heidelberg, 2008. Springer-Verlag.

Lance Parsons, Ehtesham Haque, and Huan Liu. Subspace clustering for high dimensional data: a review. *SIGKDD Explor. Newsl.*, 6(1):90–105, June 2004. ISSN 1931-0145.

Hans-Peter Kriegel, Peer Kröger, and Arthur Zimek. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM Trans. Knowl. Discov. Data*, 3(1):1:1–1:58, March 2009. ISSN 1556-4681.

Zhenhua Lv, Yingjie Hu, Haidong Zhong, Jianping Wu, Bo Li, and Hui Zhao. Parallel k-means clustering of remote sensing images based on mapreduce. In *Proceedings of the 2010 international conference on Web information systems and mining*, WISM'10, pages 162–170, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-16514-1, 978-3-642-16514-6.

Chao Lin, Yan Yang, and Tonny Rutayisire. A parallel cop-kmeans clustering algorithm based on mapreduce framework. In Yinglin Wang and Tianrui Li, editors, *Knowledge Engineering and Management*, volume 123 of *Advances in Intelligent and Soft Computing*, pages 93–102. Springer Berlin Heidelberg, 2011.

Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008b. ISSN 0001-0782. doi: 10.1145/1327452.1327492.

Alina Ene, Sungjin Im, and Benjamin Moseley. Fast clustering using mapreduce. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '11, pages 681–689, New York, NY, USA, 2011b. ACM. ISBN 978-1-4503-0813-7.

Seung-Jin Sul and Andrey Tovchigrechko. Parallelizing blast and som algorithms with mapreduce-mpi library. In *IPDPS Workshops'11*, pages 481–489, 2011.

Robson Leonardo Ferreira Cordeiro, Caetano Traina, Junior, Agma Juci Machado Traina, Julio López, U. Kang, and Christos Faloutsos. Clustering

very large multi-dimensional datasets with mapreduce. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '11, pages 690–698, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0813-7.

Amol Ghoting, Prabhanjan Kambadur, Edwin Pednault, and Ramakrishnan Kannan. Nimble: a toolkit for the implementation of parallel data mining and machine learning algorithms on mapreduce. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '11, pages 334–342, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0813-7. doi: 10.1145/2020408.2020464.

Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *SODA'10*, pages 938–948, 2010.

Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010b. USENIX Association.

Markus Varsta, Jukka Heikkonen, Jouko Lampinen, and José Del R. Millán. Temporal kohonen map and the recurrent self-organizing map: Analytical and experimental comparison. *Neural Process. Lett.*, 13:237–251, July 2001. ISSN 1370-4621. doi: 10.1023/A:1011353011837.

Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: A system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 135–146, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0032-2. doi: 10.1145/1807167.1807184.

Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M. Hellerstein. Distributed graphlab: A framework for machine learning and data mining in the cloud. *Proc. VLDB Endow.*, 5(8):716–727, April 2012. ISSN 2150-8097.

A. Frank and A. Asuncion. Uci machine learning repository. *Technical report, School of Information and Computer Sciences, available at :http ://archive.ics.uci.edu/ml*, 2010.

J. A. Hartigan. Direct Clustering of a Data Matrix. *Journal of the American Statistical Association*, 67(337):123–129, 1972. ISSN 01621459. doi: 10.2307/2284710. URL http://dx.doi.org/10.2307/2284710.

Daniel D. Lee and H. Sebastian Seung. Learning the parts of objects by nonnegative matrix factorization. *Nature*, 401:788–791, 1999.

G. Govaert and M. Nadif. Block clustering with Bernoulli mixture models: Comparison of different approaches. *Computational Statistics and Data Analysis*, 52: 3233–3245, 2008.

R. Priam, M. Nadif, and G. Govaert. The block generative topographic mapping. In *The Third International Workshop on Artificial Neural Networks in Pattern Recognition, Lecture Notes in Artificial Intelligence (LNCS)*, number 5064, pages 13–23, Berlin Heidelberg, September 2008. Springer.

Evan R. Sparks, Ameet Talwalkar, Virginia Smith, Jey Kottalam, Xinghao Pan, Joseph E. Gonzalez, Michael J. Franklin, Michael I. Jordan, and Tim Kraska. Mli: An api for distributed machine learning. *CoRR*, abs/1310.5426, 2013.

Alexander Strehl, Joydeep Ghosh, and Claire Cardie. Cluster ensembles - a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3:583–617, 2002.

T.J. Hastie, R.J. Tibshirani, and J.J.H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer-Verlag New York, 2009. ISBN 9780387848587.