Exploratory Data Analysis using Scalable Self-Organising Maps

by

Kakusanda Mudiyanselage Hiran Shyanaka Ganegedara

Thesis

Submitted by Kakusanda Mudiyanselage Hiran Shyanaka Ganegedara

for fulfilment of the requirements for the degree of

Doctor of Philosophy (0190)

Supervisor: Associate Professor Damminda Alahakoon

Associate Supervisor: Dr. Susan Bedingfield

Clayton School of Information Technology
Monash University

August, 2014

To my parents

Contents

List of Tables

List of Figures

Exploratory Data Analysis using Scalable Self-Organising Maps

Kakusanda Mudiyanselage Hiran Shyanaka Ganegedara
[redacted]
Monash University, 2014

Supervisor: Associate Professor Damminda Alahakoon
[redacted]
Associate Supervisor: Dr. Susan Bedingfield
[redacted]

Abstract

Exploratory data analysis is used to derive insights from large volumes of data. Unsupervised learning methods, such as the self-organising map (SOM) and the growing self-organising map (GSOM), have gained popularity as data exploration tools due to the limited nature of the availability of meta-information about real-world datasets. The key advantages of the SOM and the GSOM, in the domain of exploratory data analysis, are their visualisation and summarisation features. However, the application of SOM based techniques for large scale data exploration has been limited due to their high time consumption.

Distributed computing has emerged as a means of providing large amounts of computing power for data and compute-intensive applications. A number of parallel and distributed algorithms have been proposed for SOM based learning. However, none of the current distributed SOM algorithms possess all the desirable features of data-intensive distributed algorithms: a distributed memory model, data parallelism, a horizontal data layout and the ability to process both sparse and dense data.

This thesis presents a distributed SOM model, a distributed memory architecture utilising data parallelism with a horizontal data layout. The distributed SOM model employs a divide and conquer architecture in four stages: data partitioning, SOM training in parallel, redundancy reduction and topographic mapping. Both sparse and dense datasets are used to demonstrate the efficiency and the clustering accuracy of the algorithm which shows up to 99% reduction in processing time while maintaining similar levels of clustering

accuracy. The distributed SOM model has the advantage of using any SOM technique as the learning engine. However, results demonstrate that the GSOM with a dynamic structure represents the dataset better than the SOM with a static structure in exploratory data analysis.

An incremental data integration model for the Distributed GSOM is proposed in order to maintain the currency of the analysis with the availability of new data. The incremental model reuses components of the Distributed GSOM to incorporate new data into an existing network, thus avoiding the need to re-train the entire network. Results indicate that the topographic mappings generated by incremental data presentation are almost identical to the maps generated by the Distributed GSOM on the entire dataset.

The applicability of the distributed SOM model for real-world distributed computing technologies is demonstrated through implementing the Distributed GSOM on Hadoop, a popular MapReduce distributed computing framework. A MapReduce architecture is developed for the Distributed GSOM where combiners are used to further improve the efficiency of the algorithm. The efficiency of the distributed SOM model is further improved by the development of MapReduce processes for the four data partitioning methods.

The effectiveness of the Distributed GSOM is demonstrated by exploring a real-world dataset for electricity consumption profile identification. Twelve gigabytes of smart electricity meter data are processed in order to profile customer behaviours. Heuristic based partitioning is used to improve the quality of the output by incorporating outliers into the analysis process. Electricity consumption profiles are identified over multiple time intervals using the Distributed GSOM. A multi-granular profile generation framework is proposed in order to combine the outcomes of the analysis in short-term, medium term and long-term granularity levels. The results demonstrate that the Distributed GSOM identifies the most prominent and distinctive electricity consumption profiles whilst reducing the time consumption of the analysis process by 95%.

In summary, this thesis presents a practical, distributed SOM model for exploratory analysis of large datasets. The work extends the current knowledge and the technology of self-organising maps and enhances the practical value of exploratory analysis in big data environments. The effectiveness of the model is demonstrated through implementing the Distributed GSOM on Hadoop and using the algorithm for customer profile identification from real-world data.

Exploratory Data Analysis using Scalable Self-Organising Maps

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Kakusanda Mudiyanselage Hiran Shyanaka Ganegedara
August 13, 2014

Acknowledgments

First and foremost, I would like to express my heartfelt gratitude to my supervisor, Associate Professor Damminda Alahakoon, for the guidance, encouragement and support throughout my candidature. Your expertise and advice was a massive assistance to my research and I believe that I could not have asked for a better supervisor than you.

I would also like to thank my associate supervisor, Dr Susan Bedingeld, for the support given to me and for keeping me on track when my main supervisor was not around.

My family is the most important thing in my life. I would like to express my sincere gratitude to my parents for bringing me up in a nurturing surrounding and for guiding and helping me in every endeavour of my life.

Endless thanks should go to my lovely wife, Upuli for supporting me through the past four years. My PhD would have been just a dream if not for your encouraging words and love. The delicious food you cooked no doubt gave me the strength to survive during the hectic days of my PhD.

My friends from Monash University and outside added much needed excitement and variety to my PhD candidature. The trips we went and the parties and games we had would always be cherished and fondly remembered.

I would also like to thank Monash University for granting me two scholarships to assist my studies and providing all resources necessary for successfully conducting my research. My sincere thanks also go to the sta at the Faculty of IT and Monash University in general for their support.

Kakusanda Mudiyanselage Hiran Shyanaka Ganegedara

Monash University

August 2014

Chapter 1

Introduction

The world is said to have transitioned into the information age from the industrial age in the late 1970s (Kluver, 2008). The introduction of the personal computer and subsequent developments in information systems have revolutionised the day-to-day life of humankind. The use of computer software has transformed labour intensive data entry and data analysis operations into semi and fully automated computer-intensive processes. Information systems can store large volumes of data in electronic format which, compared to traditional paper based systems, is efficient in searching and retrieving information.

Data accumulation in information systems naturally creates the need to derive insights from historical data. Analysis of the data can reveal valuable information for decision making such as customer buying patterns, demographic trends and demand shifts. Exploration of data for possible patterns has been a widely researched area over the past several decades. Investigating data for possible patterns is referred to as exploratory data analysis. The need to analyse and interpret massive volumes of data has given rise to the term 'Big Data' in recent years (Manyika et al., 2011). Figure 1.1 shows the main steps of the exploratory data analysis process as given in Fayyad et al. (1996).

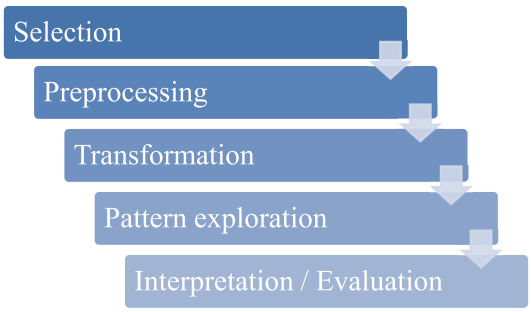


Figure 1.1: The exploratory data analysis process

The primary approaches to data exploration are machine learning (Witten and Frank, 2005), and statistical and probabilistic (Behrens, 1997). It is common practice to apply unsupervised techniques to identify high level patterns and further refine the patterns using supervised techniques with the knowledge acquired in the first stage. The self-organising map (SOM) proposed in Kohonen (1990) and its variants, such as the growing self-organising map (GSOM) proposed in Alahakoon et al. (2000), are widely used in exploratory data analysis as data mining tools. SOMs have become popular in the analysis domain due to:

1. their ability to visualise data in two or three dimensional space
2. their unsupervised learning capabilities
3. their control over the level of detail of the analysis

A key advantage of the SOM is its ability to visualise the dataset in two or three dimensional space. Browsing the lattice reveals the underlying structure of the data. There are a number of visualisation improvements such as U-Matrix, component planes and parallel axis which can be used in addition to the SOM for assisting the analysis task.

A common feature of exploratory data analysis exercises is the lack of knowledge about the underlying structure in data. As a result, defining the expectations of the outcome of the analysis may not be possible at the initial stages of analysis. The SOM algorithm's unsupervised learning capabilities are efficient at eliciting the inherent patterns within data.

A common approach in exploratory data analysis is to start by performing a coarse grained analysis and then to conduct a finer grained analysis based on the initial analysis. This approach is referred to as drilling down and the analysis algorithm should have the capability to produce different grains of analysis outcomes in order to facilitate this drill down. The level of detail of the map can be changed by altering the size of the map. Therefore, a larger map could be used for a finely grained analysis and a smaller map could be used for a coarsely grained analysis.

The SOM has been used for a number of exploratory data analysis applications, as given in Kaski et al. (1998) and Oja et al. (2003). The SOM is primarily used as a visualisation technique in exploratory data analysis where the structure of the dataset is represented as a

two or three dimensional map in an unsupervised manner. However, the time consumption of the SOM is excessive for very large datasets. Kohonen et al. (2000) demonstrated that the SOM was estimated to consume more than six weeks of processing to cluster seven million documents.

Although the structure of the SOM is inherently parallel, the SOM standard algorithm is serial in nature. As a result, the algorithm is incapable of utilising more than one execution thread. The time complexity of the SOM algorithm is linear on the number of input vectors and quadratic on the number of neurons, as given in Roussinov and Chen (1998). Although the time complexity does not have a significant impact for small to medium size datasets (up to a few thousand records), as the number of input vectors increases, the time consumption of the algorithm increases significantly. For very large datasets, the time requirement leads to impractical processing times making the analysis task span long durations. Therefore, improving the efficiency of the SOM is critical to ensuring its application to massive datasets.

The execution time of a serial processor-intensive algorithm is primarily determined by the clock speed of the central processing unit (CPU) of the computer that is executing the algorithm. Traditionally, executing an algorithm on a faster CPU was employed as a means of reducing the running time when the algorithm can no longer be optimised. However, due to silicon chip fabrication limitations, the increase in CPU clock speeds has stagnated in the past decade, according to Sutter (2005). This is evident by observing the clock speeds of the latest desktop computer chips. According to Intel (2013), the fastest desktop computer chip at present has a clock speed of 3.90 GHz. The fastest desktop processor a decade ago, in 2004, had a clock speed of 3.60 GHz, which amounts to an 8.33% increase over 10 years. In order to compensate for the slow improvement in CPU clock speeds, computer chip manufacturers have sought to incorporate multiple computing cores as a means of providing more computing capabilities into modern CPUs. However, the performance gain obtained by hardware advancements has been limited for serial algorithms since the execution process uses only one core of the CPU.

Parallel and distributed computing have been in existence for several decades as a means of facilitating compute-intensive applications by integrating several computers to function as a single unit. In order to fully utilise the computing capabilities offered by

parallel and distributed computing clusters, algorithms that run on them should have the following properties.

Parallelism – The algorithm should consist of multiple processes which can be executed in parallel.

Scalability – Since the number of processing units in the cluster can change across different configurations, the algorithm should be able to modify the number of parallel tasks to fit the cluster.

Synchronisation – The parallel tasks should communicate efficiently amongst themselves such that communication bandwidth is used optimally.

Programmers have to design and implement programs on parallel and distributed systems adhering to the above properties. Due to the complexities involved in inter process communication and data transfers, adoption of distributed and parallel computing on a mass scale has been limited at the early stages. Hadoop (Apache, 2013), developed by the Apache Foundation, provides a framework for distributed application development which provides a simple communication model by abstracting the underlying complex communication and synchronisation processes within a distributed system. Hadoop also provides a distributed file system capable of hosting petabytes of data files which has seen a significant increase in its popularity in large scale data analysis applications.

Hadoop currently provides a range of clustering and statistical data analysis methods vi the Apache Mahout (Owen et al., 2011) project. However, the Mahout project currently does not offer an SOM implementation. The main challenge of implementing the SOM on Hadoop is the development of a suitable architecture for the SOM algorithm that matches the Hadoop programming model.

Smart grids are considered one of the real-world large scale data analysis application areas, as given in Villars et al. (2011). In order to process the gigabytes of electricity consumption data generated on a daily basis, efficient, scalable, distributed data analysis algorithms are required.

1.1 Research motivation

Based on the above discussion, there is a growing need for more efficient SOMs for processing large scale data. Work presented in this thesis was conducted with the aim of developing an efficient distributed algorithm for SOM based exploratory data analysis. Distributed and parallel computing is considered as a means of providing the computing power required for large scale data analysis tasks. Distributed algorithms should have several key properties which make them suitable for large scale data analysis, as given in Zaki (2000).

1. Algorithms should have a distributed memory model which has higher scalability over shared memory models.

2. Algorithms should use data parallelism in order to process massive volumes of data efficiently.

3. The data layout of the algorithm should be horizontal in order to preserve attribute relationships for exploration and to achieve higher levels of scalability.

The current SOM literature does not have a distributed solution which possesses all the above properties. As a result, the application of the SOM for the exploration of large datasets has been hindered by time consumption limitations. The motivation to conduct the work presented in this thesis stemmed from the absence of an efficient distributed memory SOM algorithm with data parallelism having a horizontal data layout. It is expected the new generation of SOM based distributed algorithms proposed in this thesis will widen the application of SOMs in data exploration in the current data intensive environments.

1.2 Research questions

Inspired by the motivation to develop a distributed SOM algorithm, research questions were formulated in order to guide the work conducted. The primary research question can be stated as follows.

> How can the self-organising map be extended via the functionality of a distributed algorithm to enable exploratory analysis with big data?

As the primary research question is broad and abstract, the question was broken down into the different research areas addressed in this thesis. The research areas were identified as algorithmic research on the SOM, distributed computing research and large scale exploratory data analysis research. Specific sub questions were formulated under each research area in order to define the scope of the research. The research questions are listed below.

1.2.1 Research questions on distributed SOM algorithm development

1. How can a distributed memory architecture be developed for the SOM with data parallelism and a horizontal data layout?
2. What are the different partitioning methods to create a horizontal data layout?
3. How can redundancy be determined in SOMs trained on subsets of data and how can redundant neurons be removed?

1.2.2 Research questions on implementing algorithms on distributed computing frameworks

1. How can the distributed SOM algorithm be transformed into a MapReduce pattern?
2. How can partitioning of large datasets utilise a MapReduce algorithm?
3. How can data partitioning ensure that each parallel process receives a sufficient number of records?
4. What measures can be taken to minimise the communication overhead in a distributed computing platform?

1.2.3 Research questions on applying the distributed SOM algorithm onto large datasets

1. How do static and dynamic SOM structures affect data exploration?
2. How can the distributed SOM algorithm be used to explore large electricity meter readings to identify electricity consumption profiles?
3. How can data partitioning be used to improve the quality of the analysis?

4. How can different granularity levels of the analysis add more meaning to the identified electricity consumption profiles?

5. How can new data be integrated into the distributed SOM output without having to re-train the entire network?

1.3 Research objectives

Research objectives were formulated with the aim of answering the research questions. The main objectives of the research are listed below.

1. To develop a distributed memory algorithm for the SOM with data parallelism and a horizontal data layout.

2. To investigate the impact of different partitioning methods on performance and the quality of the output.

3. To develop a measure for redundancy and to develop algorithms to remove redundant neurons.

4. To develop a MapReduce architecture for the distributed SOM algorithm.

5. To create a MapReduce architecture for the data partitioning process.

6. To ensure that the partitions are created such that each parallel process receives a sufficient number of data vectors to ensure proper training of the SOMs trained on each partition.

7. To develop data transfer methods to minimise the communication overhead in a distributed computing environment.

8. To investigate the effect of static and dynamic SOM structures on the exploratory data analysis process.

9. To develop a data analysis model for extracting electricity consumption profiles from large volumes of electricity meter readings.

10. To utilise data partitioning to improve the quality of the output.

11. To identify profiles in different granularity levels in order to increase the level of detail in the analysis.

12. To develop an architecture for the distributed SOM algorithm where new data can be continuously integrated into an established SOM.

1.4 Research contributions

The research presented in this thesis was conducted with the aim of achieving the research objectives. The main contributions of this research are summarised below.

1. Development of a novel parallel algorithm for the SOM which employs a distributed memory model with data parallelism having a horizontal data layout. The new algorithm, called the Distributed GSOM, is significantly faster than the traditional, serial SOM and GSOM.

2. Evaluation of the effect of random and class based partitioning methods for a number of datasets. Results indicate that random partitioning is a suitable option when class information is unavailable.

3. Development of a new redundancy estimation measure called the *redundancy index* which is used to identify and remove redundant neurons created in maps trained on different partitions.

4. Implementation of the Distributed GSOM on Hadoop, a popular distributed computing framework. The Hadoop framework features, such as combiners, were used to improve the efficiency of the Distributed GSOM.

5. Development of novel MapReduce processes for data partitioning where terabytes of data can be efficiently partitioned, in particular when heuristic based partitioning is used.

6. Finetuning of the standard behaviour of the Hadoop framework in order to distribute the workload across the cluster and to ensure that the partitions are created without omitting any records.

7. Modification of the communication structure of the algorithm in order to encapsulate the vectors belonging to a particular neuron within the neuron itself. The

vector encapsulation removes the need to access the dataset during the redundancy reduction process which results in faster performance.

8. Evaluation of the effectiveness of SOMs with a static structure and GSOMs with a dynamic structure for exploratory data analysis. GSOMs have better structure adaptation and have faster performance than SOMs.

9. Proposal of an innovative analysis model to identify electricity consumption profiles from large volumes of smart electricity meter readings. The proposed model tolerates outliers and resilience to varying distributions of records within partitions.

10. Utilisation of an average electricity consumption based heuristic to improve the grouping of records into partitions. The partitioning method demonstrated high tolerance of outliers.

11. Introduction of a novel methodology to identify short-term, medium term and long-term electricity consumption profiles using the Distributed GSOM in order to analyse electricity consumption behaviour in different granularity levels.

12. Development of a new model to continuously integrate new data into an established SOM without the need to re-train the entire network.

1.5 Research methodology and chapter outline

This thesis presents the research contributions in detail in the chapters to follow. Relevant research literature covering exploratory data analysis, self-organising maps and parallel and distributed data mining are discussed in Chapter 2. Figure 1.2 shows the organisation of the rest of this thesis.

Chapter 3 presents a novel distributed algorithm for the SOM learning process. The new algorithm, named the Distributed GSOM, utilises a distributed memory model and data parallelism. The dataset is partitioned into a horizontal layout which can be applied to both sparse and dense data. The algorithm is experimentally evaluated for both efficiency and accuracy which demonstrates that the Distributed GSOM is several orders faster than the serial SOM and GSOM.

The inner workings of the Distributed GSOM are discussed in detail in Chapter 4. The implications of using static and dynamic SOM structures are discussed in the context of

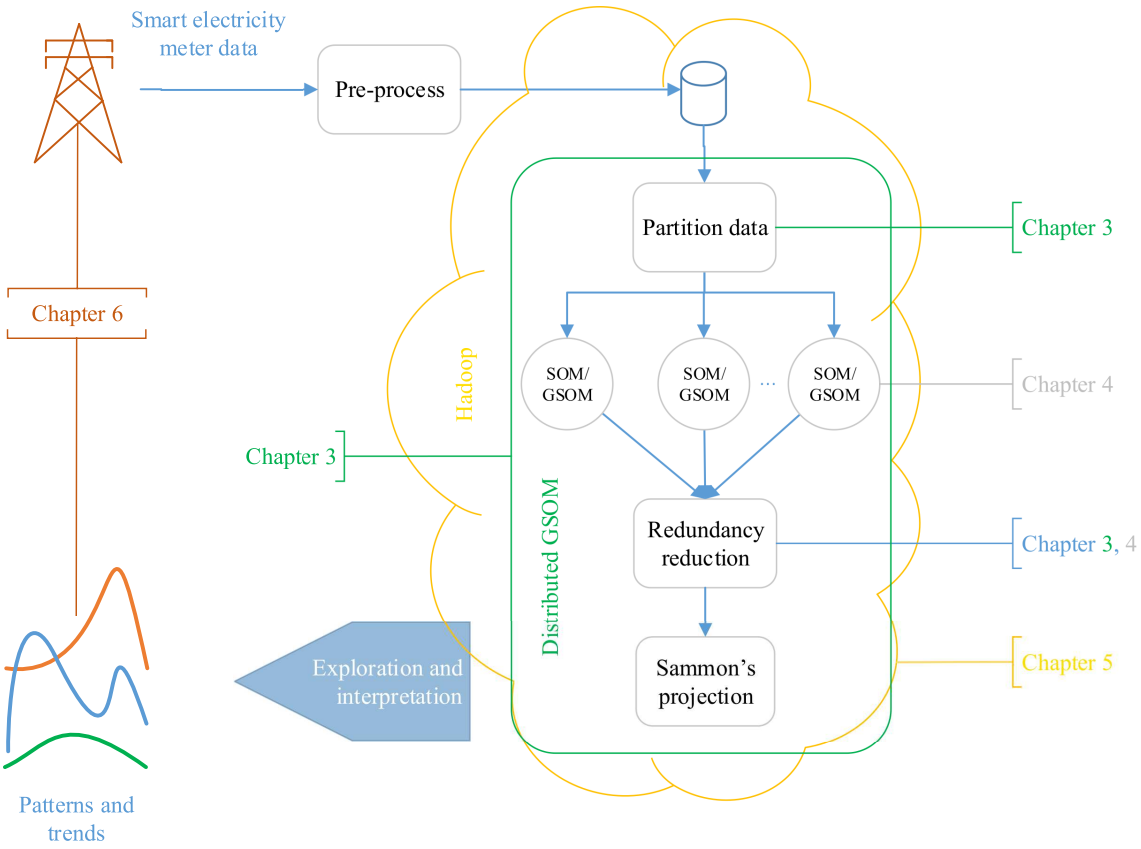


Figure 1.2: Thesis organisation

exploratory data analysis. Results demonstrate that the GSOM with a dynamic structure is more suited for data exploration. Chapter 4 also compares two redundancy reduction methods. How the two redundancy reduction methods can be used in different stages of the data exploration process is also discussed.

Chapter 5 discusses the implications of implementing the Distributed GSOM algorithm on the popular Hadoop distributed computing framework. A new node assignment algorithm is presented which distributes the workload of compute-intensive algorithms across the entire cluster.

An exploratory data analysis using a real-world example of electricity consumption is presented in Chapter 6. The analysis process of gigabytes of electricity meter readings is done through the Distributed GSOM and the common electricity consumption profiles are identified.

Chapter 2

Literature Review

The core research areas and theories related to the work presented in this thesis are discussed in this chapter. Exploratory data analysis is a widely researched topic with a vast amount of published literature and the self-organising map (SOM) is one of the popular techniques used for data exploration. A key limitation of the SOM is the high time consumption of the algorithm when processing large datasets. Parallel and distributed computing has gained popularity as a means of providing computing clusters with massive processing power for data and compute-intensive applications.

The work presented in this thesis focuses on extending the SOM technology such that this widely used and popular technique could be utilised in the new highly data intensive and distributed environments. Several questions have to be answered in order to explore the literature related to SOM based data analysis. These are given below.

1. Exploratory data analysis
   (a) What is exploratory data analysis?
   (b) What are the common algorithms used for exploratory data analysis?
   (c) What is the self-organising map and how and why is it considered a useful tool in exploratory data analysis?
2. Self-organising maps
   (a) How does the self-organising map function?
   (b) What is the growing self-organising map and how does it differ from the self-organising map?

(c) What issues arise when processing large volumes of data with self-organising maps?

3. Distributed data analysis algorithms

(a) How do parallel and distributed data mining handle large volumes of data?

(b) What are the key considerations in developing parallel and distributed algorithms?

(c) What are the currently existing parallel and distributed self-organising map algorithms?

The aim of this literature review chapter is to answer the above questions and to provide context for the problem that is addressed in this thesis. Each of the above questions are answered in subsequent sections of this chapter.

2.1 Exploratory data analysis and clustering

Exploratory data analysis is defined as "detective work – numerical detective work – or counting – or graphical detective work" in Tukey (1977). In other words, exploratory data analysis refers to the process of deriving insights where very little or no information is known about the structure and the patterns within data. The massive growth of information further complicates the analysis process. The rate of expansion of data is discussed in Gantz and Chute (2008). Data analysts have to face a number of challenges in exploratory data analysis exercises. The major challenges are processing power requirements, memory requirements and pattern identification as given in Keim et al. (2006).

Clustering is considered as the main exploratory data analysis method in the work presented in this thesis. Clustering has been widely used as an exploratory data analysis technique from its very early stages (Dubes and Jain, 1980). The primary objective of clustering is to find a number of groups within data such that the similarity of the records within a group are very high and also the differences between the records in two different groups is high. Segmentation achieved from clustering can provide insight into the dataset by grouping similar records. For example, clustering the customer base of a company by demographic attributes could reveal insights such as buying patterns of the customers in different age groups.

Over the years, a significant number of approaches have been researched for clustering. The most popular clustering techniques are discussed briefly below.

Hierarchical algorithms – clusters are created using agglomeration or division. The most popular hierarchical clustering algorithms are single-link (Sneath and Sokal, 1973), complete-link (King, 1967) and minimum-variance (Ward Jr, 1963) methods.

Partitional algorithms – a pre-defined number of clusters are created with some property such as the squared-error. The best example is the K-means algorithm developed by MacQueen (1967).

Fuzzy clustering – a membership function is used to assign data vectors to clusters. Fuzzy theory was first applied to clustering by Ruspini (1969) and the most popular fuzzy clustering algorithm at present is the fuzzy c-means algorithm devised by Bezdek et al. (1984).

Neural networks – neural networks are widely used for clustering such as the self-organising map (Kohonen, 1990) and adaptive resonance systems by Carpenter et al. (1991).

Although all of these techniques have merits and demerits, the work presented in this thesis is focused around the self-organising map (SOM). The SOM is widely used for exploratory data analysis due to its unsupervised learning capabilities and visualisation properties. Astel et al. (2007) have shown that the SOM yields more value to the data analysis process compared to principal component analysis (Wold et al., 1987) and cluster analysis. In addition, Ultsch et al. (1995) have demonstrated that the SOM has greater clustering accuracy over the popular K-means (MacQueen, 1967) algorithm. Unlike most clustering algorithms the SOM creates a visualisation of the dataset which is immensely useful in exploratory data analysis. The strengths of the SOM for data exploration are discussed further in 2.5.

2.2 Self-organising maps

The SOM is an unsupervised learning algorithm which projects high dimensional data onto a low dimensional space. The low dimensional space is usually two or three dimensional

and creates a comprehensible visualisation of the input data space. The self-organisation follows the Hebbian learning principle (Hebb, 1949) which states that neurons which respond to the same stimuli have stronger interconnectivity. The map aspect of the SOM consists of a network of neurons arranged in a lattice. The most common lattice structures are rectangular and hexagonal. The SOM algorithm is an iterative process which is described below.

2.2.1 The SOM algorithm

Step 1. Create a lattice of neurons. Several important decisions would have to be made when creating the lattice.

- What is the structure of the lattice? The structure of the lattice could be either rectangular or hexagonal. The primary functional difference between the two lattice structures is the neighbourhood size of each neuron. A rectangular lattice will permit a maximum of four neighbours whereas a hexagonal lattice will allow up to six neighbours.

- What is the shape of the lattice? The shape of the SOM lattice plays an important role, especially in data analysis (Kohonen, 2001). The shape could be either the most common square and rectangular, or any other shape that is deemed suitable. The shape of the map should match the shape of the dataset in order to create a fair representation. A mismatch in shape would adversely affect the visualisation of the dataset.

- What is size of the map? The size of the map is measured by the number of neurons. The number of neurons in the map determines the level of detail required in the output map. If the number of neurons in the network is insufficient, the dataset may be under represented. Having too many neurons could overrepresent the dataset.

Step 2. Initialise the weight vectors of the neurons. The most common method of initialisation is to assign a random weight vector for each neuron. Additional initialisation methods have been proposed by Su et al. (2002) and Kitajima (1995).

Step 3. Repeat until the entire input dataset is covered for each iteration.

Step 3a. Pick a random vector, i, from the dataset.

Step 3b. Find the neuron n_c, having the closest proximity to i. The distance between n_c and i, d_c is calculated using equation (2.1)

$$d_c(t) = \min_k\{||w_k(t) - i||\}, \tag{2.1}$$

where w_k is the weight vector of neuron n_c. n_c is referred to as the best matching unit (BMU) for i. Several distance measures are used in different applications such as Manhattan distance, Euclidean distance and weighted Euclidean distance. Implications for the SOM learning process from these different similarity measures are discussed in Strehl et al. (2000).

Step 3c. Update the weight vectors of the BMU and neighbourhood neurons (n_k) using

$$w_k(t+1) = w_k(t) + \alpha(t)h_{ck}(t)[i - w_k(t)], \tag{2.2}$$

where α is the learning rate and h_{ck} is the neighbourhood function. In most cases, the standard Gaussian function is used as the neighbourhood function as given in equation (2.3).

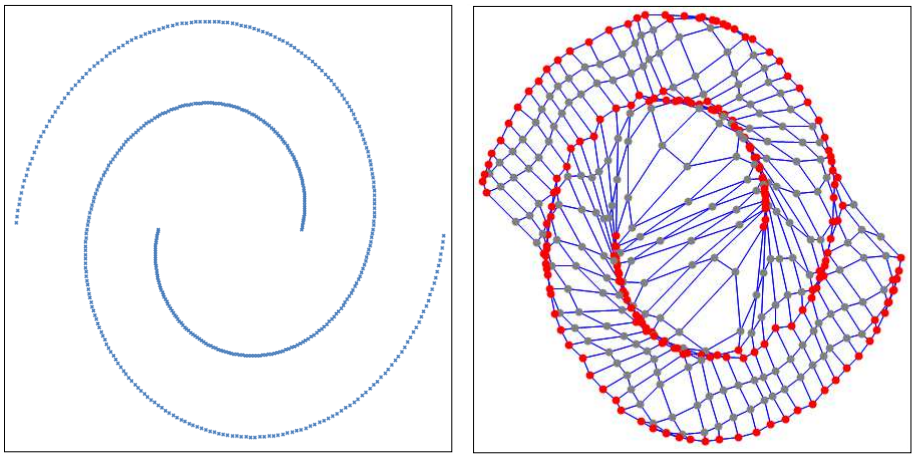
$$h_{ck}(t) = e^{-\frac{|r_k - r_c|}{\sigma(t)}}, \tag{2.3}$$

where, r_k and r_c represent the coordinates of n_c and n_k in the low dimensional space and $\sigma(t)$ is a decreasing function.

The output of the SOM algorithm is a topographically arranged neuronal map where close input vectors in the high dimensional space are mapped onto closer neurons in the low dimensional space. This feature is the key reason for the SOM's popularity in a wide range of applications.

Figure 2.1 shows the SOM for a two dimensional spiral dataset. The created network has the same arrangement as the actual dataset. The summarisation aspect is also visible from the map. Red nodes indicate hit neurons where the neuron is the BMU for at least one data vector. Gray nodes indicate non-hit neurons.

Although the dataset is two dimensional, the ability of the SOM to learn the structure of the data is evident from the results. The results were obtained for an SOM network with a size of 10×10 which was too large for the given dataset. The resulting over representation



(a) The spiral dataset (b) SOM trained on the spiral dataset

Figure 2.1: SOM network for the spiral dataset

of the data is clear between the lines of hit neurons. There are several layers of non-hit neurons forming the boundary between the two spirals. In order to distinguish the boundary, one layer of non-hit neurons is sufficient. Due to the use of a suboptimal network size, an unnecessary level of detail is created.

The growing self-organising map (GSOM) addresses this limitation by dynamically adjusting the size of the network to accommodate the dataset. The GSOM and its advantages are discussed in the following section.

2.3 Growing self-organising map (GSOM)

The GSOM, proposed by Alahakoon et al. (2000), is an extension of the SOM algorithm where, instead of a fixed lattice, the network starts with only four neurons and creates neurons to accommodate the dataset. The learning principle of the GSOM algorithm is the same and due to the creation of nodes, the algorithm has two phases, the growing phase and the smoothing phase.

A key decision in the initialisation of an SOM is the determination of the shape and the size of the network. Using improper attributes could affect the quality of the output. The GSOM overcomes this issue by using only one parameter in the initialisation, the *spread factor*. The spread factor determines the level of detail in the generated neural map. A higher spread factor will result in a detailed map and a lower spread factor will create a less detailed map.

In the growing phase, each vector in the dataset is randomly presented to the network and the BMU is identified. Each BMU accumulates the quantisation error in the selection process. If the BMU is a boundary node and the accumulated quantisation error exceeds a pre-defined threshold called the growth threshold, new neurons are created in the neighbourhood of the BMU. The weight vectors of the neighbouring neurons of the BMU are then updated. This process is repeated over the number of growth iterations.

Once the growing phase is completed, the smoothing phase is carried out. This phase executes all the steps in the growing phase except for the new neuron creation step. Additionally, the learning rate used for the neighbourhood update process is less since, to some extent, the network is already topographically arranged.

2.3.1 The GSOM algorithm

The GSOM algorithm employs the same Hebbian learning principle as the SOM as described in 2.2.1. The key differences in the structures of the algorithms are that the SOM has a single training phase whereas the GSOM has two phases, the growing phase and the smoothing phase. The algorithms for growing and smoothing are discussed in the following sections.

The growing phase

The growing stage is responsible for creating the required number of neurons and expansion of the map. If a neuron accumulates a high quantisation error, it is an indication that the weight vector of the neuron may not be the best match for the accumulated data vectors. In order to reduce the error, the neuron will spawn new neurons with the aim of creating better candidates for the data vectors. New neurons are created only if at least one of the four immediate neighbours of the neuron do not exist.

The algorithm for the growing phase is detailed below.

Step 1. Create a lattice of four neurons and initialise their weight vectors randomly.

Step 2. Calculate the growth threshold (GT) using

$$GT = -D \times ln(spreadfactor) \tag{2.4}$$

Step 3. Repeat until the entire input dataset is covered for each growing iteration.

Step 3a. Select a random vector i from the dataset.

Step 3b. Find the BMU for i using equation (2.1).

Step 3c. Update the quantisation error (QE) of n_c using

$$QE_{n_c}(t+1) = QE_{n_c}(t) + |w_c(t) - i| \tag{2.5}$$

Step 3d. If $QE \geq GT$ AND n_c is a boundary node, then create all missing neighbours for n_c. There are three possible initialisation scenarios each of which is described below.

(i) When a node is present directly opposite the spawning node and the spawned node as shown in Figure 2.2,

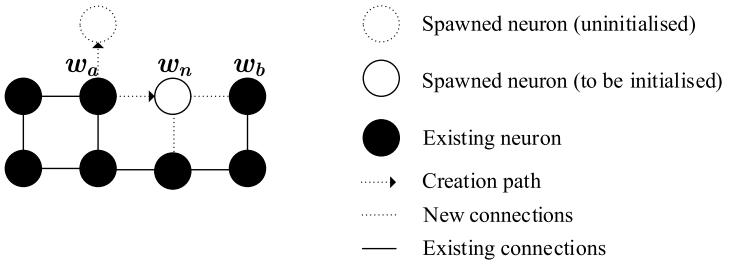


Figure 2.2: Neuron initialisation scenario 1

the new weight w_n is calculated as;

$$w_n = \frac{w_a + w_b}{2}. \tag{2.6}$$

(ii) When two consecutive neighbours exist along the spawning node as shown in Figure 2.3,

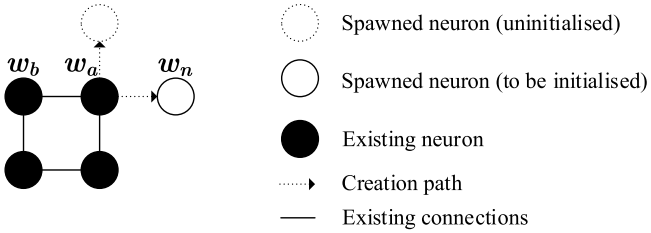


Figure 2.3: Neuron initialisation scenario 2

the new weight w_n is calculated as;

$$w_n = w_a - (w_b - w_a). \tag{2.7}$$

(iii) When two consecutive neighbours do not exist along the spawning node as shown in Figure 2.4,

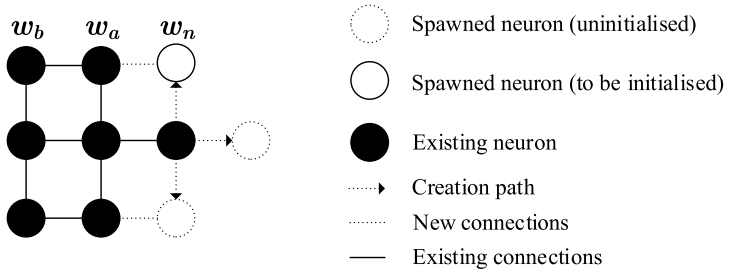


Figure 2.4: Neuron initialisation scenario 3

the new weight w_n is calculated as;

$$w_n = w_a + (w_a - w_b). \tag{2.8}$$

Step 3e. Update the neighbourhood of n_c using equation (2.2).

Once the growing phase is completed, the map would have reached its final size. However, the neurons that were added at the end may not have gone through a sufficient number of neighbourhood adaptation iterations to ensure topology preservation. Therefore, a smoothing phase is carried out in order to improve the topographic arrangement further.

The smoothing phase

The smoothing phase is carried out using a lower learning rate since the network possesses a topographic arrangement generated from the growing phase. Usually the number of smoothing iterations is less than the number of growing iterations.

Step 1. Set the learning rate. It is common practice to set the learning rate to 50% of the growing phase learning rate.

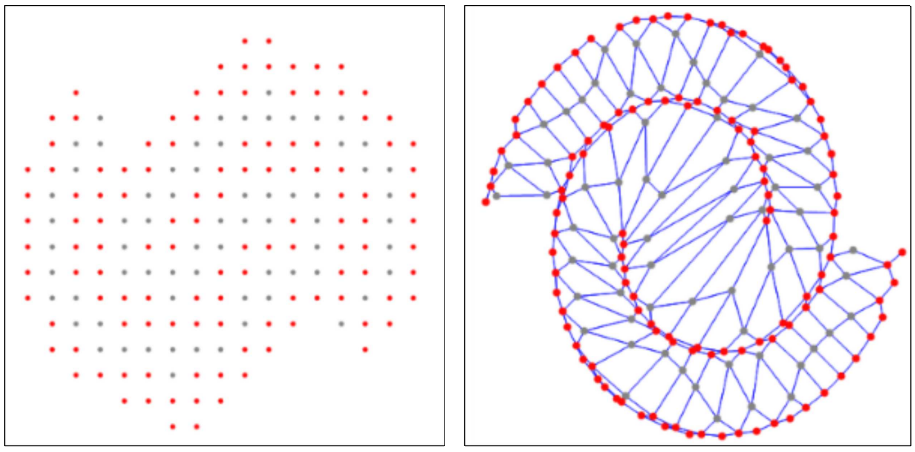
Step 2. Repeat until the entire input dataset is covered for each smoothing iteration.

Step 2a. Select a random vector i from the dataset.

Step 2b. Find the BMU for i using equation (2.1).

Step 2c. Update the neighbourhood of n_c using equation (2.2).

Unlike the traditional SOM algorithm, the layout of the map created by the GSOM is irregular. If the GSOM algorithm is run on the same dataset, two different maps will



(a) The generated GSOM network (b) Weight vectors of the GSOM

Figure 2.5: GSOM layout and the mapping of weight vectors for the dataset in Figure 2.1

be created. Figure 2.5 shows the output of the GSOM for the same dataset shown in Figure 2.1. The GSOM creates fewer non-hit nodes than the SOM and the network itself resembles the actual dataset.

Compared with the SOM in Figure 2.1, the GSOM has created significantly fewer neurons. In addition, the layout of the GSOM does not contain unnecessary neurons at the boundaries. By adapting the structure of the map to that of the dataset, the GSOM creates less confusion in interpretation and having a lesser number of neurons improves the efficiency of the algorithm.

2.4 Applications of the SOMs

The SOM and its variants have been applied in a wide variety of applications as discussed by Kaski et al. (1998) and Oja et al. (2003). The applications of the SOM can be divided into two main categories, pattern recognition and simulation.

Pattern identification

The SOM algorithm has been widely used in a number of application areas for pattern recognition. If the pattern space can be defined in multi-dimensional data vectors, the SOM can be used on the dataset to group common patterns and arrange them in a topographic map. The topology preserving property of the SOM allows users to inspect the structure of high dimensional datasets visually. The topographic organisation would result

in a clustered map which can be interpreted according to different application domains. The clustering properties of the SOM are discussed by Vesanto and Alhoniemi (2000).

A number of additional techniques have been developed which enhance the visualisation capabilities of the SOM. U-Matrix proposed by Ultsch et al. (1995) is one useful technique where inter neuronal distances are visualised by a colour scale. For example, a higher distance will be indicated by a darker colour and a smaller distance by a lighter colour. The U-Matrix enables the identification of cluster boundaries. Another method is component plane visualisation whereby the network is coloured by the distribution of only one attribute at a time.

The SOM algorithm has been used in a wide range of application areas such as image processing (Laaksonen et al., 2000; Haritopoulos et al., 2002; Toivanen et al., 2003; Ganegedara et al., 2012), control systems (Barreto and Araujo, 2004; Kohonen et al., 1996; Walter and Schulten, 1993; Jatmiko et al., 2010), text analysis (Honkela et al., 1997; Roussinov and Chen, 1998; Matharage et al., 2013) and numerous other application areas specified in Kaski et al. (1998) and Oja et al. (2003).

Neural computation and simulation

The SOM algorithm is based on the Hebbian learning principle. As a result, SOMs have been used to simulate the human brain and develop models to understand the function of the brain. Kohonen (1993) discusses the physiological interpretation of the SOM and compares itwith the cortical function. Due to the SOM's ability to mimic brain functions, it has been used to develop a number of cortical models. Sirosh and Miikkulainen (1997) propose a model for the ocular dominance function of the visual cortex which is known as the laterally interconnected synergetically self-organising map (LISSOM). More recently, Pilly and Grossberg (2013) have proposed a spiking neuron based self-organising map for learning spatial and temporal properties of brain cells.

One of the earliest models of the visual cortex has been proposed by Hubel and Wiesel (1968). SOMs have been used by Ramanathan et al. (2010) to develop a concept representation model inspired by the visual cortex. Further applications of the SOM in neural computation are given in Obermayer and Sejnowski (2001).

2.5 The SOM based algorithms as data analysis tools

Although data analysis can be categorised as pattern recognition, data analysis applications of the SOM are discussed separately due their significance to the thesis topic. The SOM and its variants are widely used in exploratory data analysis due to their compression and visualisation capabilities.

Ultsch and Siemon (1990) were among the first to discuss the advantages of using the SOM for exploratory data analysis. A comprehensive study on the desirability of the SOM for data analysis is given in Kaski (1997). The key features of the SOM which assist the data analysis process are:

i. Visualisation

ii. Unsupervised learning

iii. Compression

The core function of the SOM is to create a low dimensional representation of high dimensional data. The dataset can be effectively browsed using the neuron lattice created by the SOM algorithm. The visualisation features of the SOM are further discussed in Vesanto (1999) and Flexer (1999). Additional visualisation methods have been developed in order to enhance the data analysis process such as the U-Matrix, component planes and parallel axes.

Having very little or no knowledge about the patterns in the dataset is a prominent feature of exploratory data analysis. Therefore, data analysis algorithms would have to isolate patterns from large quantities of data. Supervised learning algorithms, such as the back propagation neural network (Bryson and Ho, 1969), would require a labelled dataset for training which contains all the possible patterns. The learning algorithm in the SOM is unsupervised, which implies labelled data is not required. Therefore the SOM is well suited to exploratory data analysis.

Another important feature of the SOM is the compression of the input dataset into a collection of neurons. The SOM trained on a large dataset can be used to describe the dataset since similar records would be mapped onto the same neuron. This summarised view of the dataset is useful for exploratory data analysis since a concise view of the entire dataset is created.

Due to the visualisation and summarisation features of the SOM, a number of application areas have used SOMs for data exploration. Engineering applications of the SOM are discussed in Kohonen et al. (1996). Kiviluoto (1998) used the SOM to predict bankruptcies, and electricity consumption analysis was performed in Lendasse et al. (2002): these are some of the many SOM based data analysis applications.

2.6 Using SOMs for large scale data analysis

For large datasets, the processing power required for analysis increases more than proportionately to the increase in the size of the dataset. The time complexity of the SOM is linear on the number of input vectors and quadratic on the number of neurons. Large datasets could contain a large number of patterns. Therefore, SOMs would have to contain a large number of neurons in order to represent the dataset properly. As a rule of thumb, the number of neurons is set to approximately $5\sqrt{N}$ for N data vectors as given in Vesanto and Alhoniemi (2000). Therefore, the processing power requirement for the SOM for large datasets will be significantly high.

Improving the performance of the SOM for large datasets has been widely researched. Two key approaches have been used to improve the efficiency of SOM algorithms: these are 1) improving the efficiency of the serial algorithm, and 2) creating parallel and distributed algorithms.

A number of serial improvements have been proposed for the SOM algorithm for efficient processing of large datasets. Some of the proposed enhancements include initialisation methods (Su et al., 2002), changes to the structure (Rauber et al., 2002; Ontrup and Ritter, 2006) and changes to the learning algorithm (Alahakoon et al., 2000; Cuadros-Vargas and Romero, 2005; Mulier and Cherkassky, 1995). Such improvements result in slightly lower computational times and can be used effectively for medium sized datasets spanning a few hundred megabytes. However, any improvement to the serial SOM algorithm will be constrained by the amount of resources available in the computer which runs the algorithm.

Although significant advancements have been made in increasing the processing power and data storage in computer hardware, time in delivering the results has also become a very important factor. Kouzes et al. (2009) demonstrate that the size of datasets has

significantly increased into terabytes and petabytes. For an algorithm to work efficiently, the data has to be loaded onto the system memory of the computer. When the size of the dataset exceeds the amount of physical memory, data will have to be transferred back and forth between system memory and storage. According to Gray (2007), using disk or flash based storage is several orders slower than using system memory. Therefore, a regular computer could take months or years to process such massive volumes of data where the relevance of the outcome would have been surpassed in a changing environment. Although a number of supercomputers exist which boast extremely high processing speeds and vast amounts of memory, availability and cost have restricted them to high end research.

Parallel and distributed computing have emerged as a cost effective means of facilitating compute-intensive operations for processing large datasets. The spread of the use of parallel and distributed computing has been assisted by commercially available computing clusters such as *Amazon.com* and *Windows Azure*. Most distributed computing clusters use regular personal computers as computing nodes which reduces the cost and increases the affordability of distributed computing for the masses.

2.7 Advancements in parallel and distributed computing systems

Many advancements have been made in parallel and distributed computing hardware. The cost of hardware has significantly decreased and has led to an increase in the availability of parallel and distributed computing platforms. Parallel computing, distributed computing and more recently, cloud computing resources are commercially offered by internet companies such as Amazon web services (Cloud, 2011) and Windows Azure (Redkar and Guidici, 2011).

Hadoop, a distributed computing framework, developed by the Apache Foundation, has gained popularity in recent years due to its scalability and high storage capacity (White, 2012). The cost of establishing a Hadoop cluster is affordable due to the use of commodity hardware. A range of complementary technologies have been developed on top of Hadoop in order to assist in exploratory data analysis, such as Hive and HBase for data storage, Pig for data transformation, Sqoop for data loading and Mahout for data mining, as given in Zikopoulos and Eaton (2011).

2.8 Parallel and distributed data analysis

In order to harness the computing power offered by parallel and distributed systems, data analysis algorithms would need to execute as parallel processes. There are a number of concerns that have to be addressed when designing a parallel algorithm as listed in Zaki (2000).

i. Shared vs distributed memory models

ii. Data vs task parallelism

iii. Horizontal vs vertical data layouts

Each of these aspects is discussed in the sections to follow.

2.8.1 Shared vs distributed memory models

Barney (2013) note functional and implementation aspects of parallel and distributed memory models. Figure 2.6 shows the differences in the architecture of the two models. While both models have been widely used in exploratory data analysis applications, distributed memory models are better equipped to process large volumes of data.

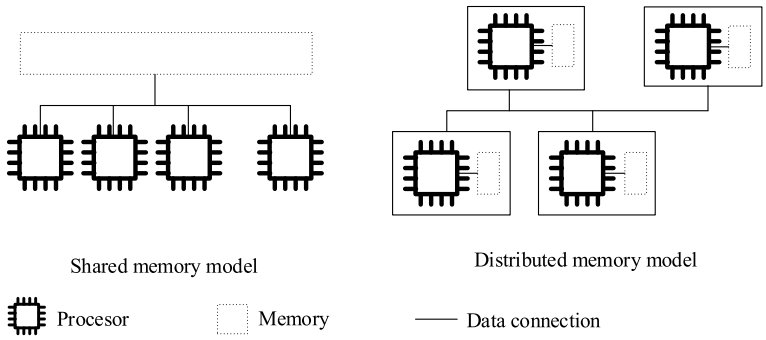


Figure 2.6: Shared and distributed memory models

Shared memory models assume access to physical memory as a global address space for all CPUs. Shared memory systems are therefore located at the same physical location and require specialised hardware in order to connect a large number of processors as a single unit of computing. While shared memory systems have an advantage of shorter memory access times, they suffer from low levels of scalability due to the memory bus being shared across all the parallel processes. Shared memory models would incur high levels of overhead if too many processes are trying to access memory simultaneously. In

addition, the cost of acquiring and maintaining specialised hardware is higher compared to commodity hardware.

CPUs in a distributed memory system on the other hand have local memory. Distributed memory models can operate locally on data and combine the outputs from each process to generate the global output. Writing programs for distributed memory models used to be difficult since programmers have to manage communication between the CPUs. With the introduction of Hadoop, programming has become streamlined since the Hadoop framework manages communication and reliability assurance. Distributed memory models can be scaled to a higher degree than shared memory models making them suitable for large scale data analysis tasks. Most distributed memory systems use commodity hardware which has increased the availability of distributed computing systems as well as bringing down the cost of such systems. The desirability of distributed memory models for data analysis is further discussed in Lawrence et al. (1999).

2.8.2 Data vs task parallelism

Algorithms that utilise data parallelism operate by assigning each parallel process to operate on a subset of the dataset. The output of each process is then combined to form a single output if required. Task parallelism operates differently via multiple processes operating on the same dataset performing different tasks. Data parallelism is well suited for distributed memory systems whereas task parallelism is geared towards shared memory systems.

In the case of large scale exploratory data analysis, the volume of data could span terabytes and petabytes. In such situations, task parallelism is undesirable since providing access to the entire dataset across multiple processes could be impractical or impossible. Data parallelism offers a scalable solution for large volumes of data by partitioning the data and assigning a process for each partition.

2.8.3 Horizontal vs vertical data layouts

Data parallel algorithms need to partition the data among processes. The dataset can be partitioned either horizontally or vertically. Vertical partitioning is where each partition contains all the rows of the dataset with only a subset of attributes. Horizontal partitioning will create partitions with all the attributes for subsets of rows.

Figure 2.7 shows vertical partitioning of a dataset consisting of N data vectors and $k \times m(= M)$ attributes. k partitions are created with n records in each partition having m attributes. x denotes the record index and y denotes attribute indices. In order to create a large number of partitions, the number of attributes have to be high.

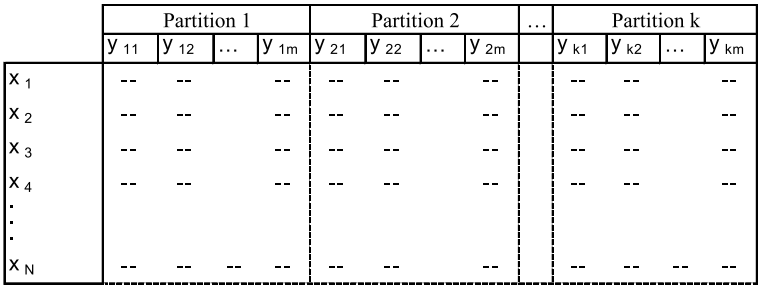


Figure 2.7: Vertical partitioning of a dataset with N records and $k \times m$ attributes

Creating k partitions from a dataset with $k \times n = (N)$ records and M attributes with horizontal data layout is shown in Figure 2.8. For large scale applications, a large number of partitions can be created since $k \times n$ will be a large value.

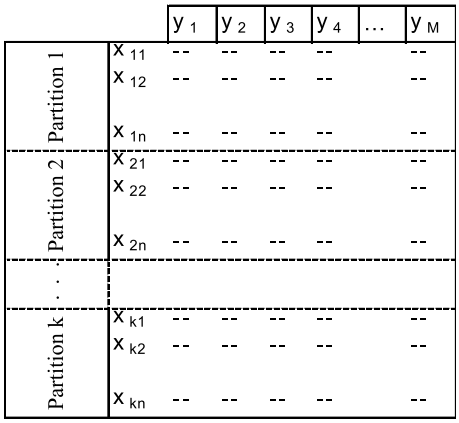


Figure 2.8: Horizontal partitioning of a dataset with $k \times n$ records with M attributes

Data analysis algorithms with vertical data layouts achieve faster performance since each parallel task processes only a subset of attributes in the original data. However, large scale data analysis tasks involve processing billions and trillions of records. In such situations, even though the number of attributes is less, a very high number of rows would have to be processed. That could still result in long processing times for each thread. Therefore, vertical data layouts may not be viable for large scale data analysis tasks.

Most large datasets have a low number of attributes compared to the number of records. Therefore, the scalability of algorithms that utilise vertical partitioning is limited by the number of attributes in the dataset. Another drawback of vertical partitioning is the loss

of attribute relationships across partitions. If closely related attributes are separated into different partitions, the relationships would not surface during data exploration. Taking all these factors into consideration, horizontal data layouts are preferred for large scale exploratory data analysis tasks.

2.9 Parallel and distributed SOM algorithms

A number of parallel algorithms have been proposed in order to address the limitations of the self-organising map encountered when processing large volumes of data. Some of the popular parallel SOM algorithms are discussed below.

2.9.1 The $_{par}$SOM

The $_{par}$SOM algorithm proposed by Rauber et al. (2000) proposes a method that utilises task parallelism to achieve faster performance. The $_{par}$SOM utilises single-instruction multiple data (SIMD) nodes for vectorisation which enables parallel vector operations. In addition, the map is partitioned and distributed across multiple nodes such that each partition contains a subset of neurons in the map. For each input vector, the local BMU for each partition is identified and the master node determines the global BMU. The neighbourhood of the global BMU is adjusted including the neurons belonging to partitions other than the partition of the global BMU. These two synchronisation operations result in a bottleneck especially in distributed memory systems.

The bottlenecks could dampen the performance advantage of the algorithm for very large datasets since the BMU search and neighbourhood update have to be conducted for every input vector. In particular, a distributed memory environment spread across physically separate computers could consume longer processing times due to network latency. Although, theoretically, $_{par}$SOM is capable of operating in distributed memory environments, the communication overhead of the BMU search and neighbourhood adaptation could hinder the benefits of parallel execution. In addition, Tomsich et al. (2000) claim to have optimised the algorithm for distributed memory systems. However, all the experiments have been carried out on shared memory systems which do not depict the true nature of distributed memory systems.

Huntsberger and Ajjimarangsee (1990) have proposed a similar model where the weight update process is executed in parallel. Due to the high overhead from synchronisation, this method possesses the same limitations of the $_{par}$SOM and would result in long processing times for large datasets.

2.9.2 Sparse batch SOM

Lawrence et al. (1999) proposed the sparse batch SOM which employs data parallelism to achieve scalability. The sparse batch SOM algorithm uses the Batch SOM (Mulier and Cherkassky, 1995) as the learning algorithm. A subset of the dataset is assigned as input to each parallel task and a local SOM network is created for each partition. At the end of an iteration, all the SOM networks across parallel tasks are synchronised resulting in a single map.

As the global neighbourhood adaptation occurs only once per iteration, the preservation of the topology of the final output could suffer. In order to minimise the loss of topology, the dataset would have to be sparse. Sparse data would mean that the density of the weight vector is low so that the chance of updating the same set of attributes is low. This approach cannot be used for dense datasets as local weight updates would significantly differ between partitions, thus adversely affecting the quality of the output. In addition, the communication overhead is significant as the networks are synchronised at the end of each iteration.

2.9.3 Graphic processing unit implementation of the SOM

The graphic processing unit (GPU) in a computer consists of an array of vector processors which can operate in parallel. The SOM structure is inherently parallel and ideally the role of each neuron can be thought of as a single processor. The GPU implementation of the SOM proposed by Zhongwen et al. (2005) assigns a single neuron or a group of neurons for each processor of the GPU for parallel operation. GPU processors are tuned for vector operations which results in increased efficiency levels.

However, the GPU memory is slower than system memory and is very expensive as given in Nickolls et al. (2008). In addition, the size of the GPU memory is limited to a few gigabytes which makes them unsuitable for large scale data processing. Although a hybrid approach is possible where data is stored in both system memory and GPU memory, the

overhead of data transfer between the two types of memory would result in impractical processing times for large datasets.

2.9.4 Scalable GSOM

The Scalable GSOM was proposed by Zhai et al. (2006) where data parallelism is used to achieve faster performance. The algorithm uses a high level GSOM to partition the dataset according to high level classes and trains separate GSOMs on each partition. The authors do not specify any specific methods to merge the separate GSOMs to create a singular view of the entire dataset.

The Scalable GSOM suffers from two key limitations. Firstly, the use of the GSOM for the initial data partitioning process will significantly increase the processing time for very large datasets. In addition, the Scalable GSOM can distribute the dataset unevenly across the neurons which will result in unbalanced loads. The Scalable GSOM does not provide a single map for the entire input dataset which will not provide the data analysts with a holistic view of the data.

2.9.5 PartSOM

The PartSOM is a technique that uses data parallelism with vertical partitioning, proposed by Gorgonio and Costa (2008). The algorithm assigns subsets of attributes to computing nodes, trains an SOM at each node and uses another SOM to combine the results. The scalability of the PartSOM depends on the number of attributes in the dataset as partitions are created with subsets of attributes. For example, the maximum number of partitions that can be created for a dataset with 10 attributes is 10. Therefore, the PartSOM will consume excessive amounts of time for a dataset with a fewer number of attributes and millions of records.

Table 2.1 summarises the different features of the above algorithms. An important observation is the absence of a parallel SOM algorithm with all the desired properties for large scale data processing.

This thesis presents a distributed memory algorithm which utilises data parallelism with a horizontal data layout that is capable of processing both sparse and dense data. The new algorithm will have all the desirable features of parallelism that are suited for large scale exploratory data analysis. The author believes that the work presented in

Table 2.1: Parallel and distributed SOM algorithm comparison

Algorithm	Shared vs distributed memory	Task vs data parallelism	Horizontal vs vertical data layout	Sparse vs dense data
Parallel SOM	Shared	Task	N/A	Both
Sparse batch-SOM	Distributed	Data	Horizontal	Sparse
Scalable GSOM	Hybrid	Data	Horizontal	Sparse
partSOM	Distributed	Data	Vertical	Both
GPU implementation	Shared	Task	N/A	Both
parSOM	Shared	Task	N/A	Both

this thesis will result in a new generation of efficient SOM algorithms and increase the popularity of the SOM even further.

Chapter 3

Distributed Self-Organising Maps

This chapter discusses the need for, and the value in extending SOMs to utilise distributed computing platforms. It introduces the distributed architecture for generating SOMs efficiently. Application of the SOM to large scale data analysis has been limited due to the compute-intensive nature of the algorithm. In this chapter, the Distributed GSOM is proposed which can harness the power of parallel and distributed computing platforms while generating an output similar to the SOM. The Distributed GSOM is several orders faster than the traditional SOM algorithm and retains the ability to generate a single map representation of the dataset. The algorithm utilises a distributed memory model with horizontal partitioning and data parallelism to achieve faster performance.

The work presented in this chapter was undertaken with the following research objectives.

1. To develop a scalable distributed architecture for the SOM to address needs identified for large scale exploratory data analysis.

2. To investigate different data partitioning techniques and their effect on sparse and dense datasets.

3. To develop a measure of redundancy for SOMs to identify and prune redundant neurons.

4. To develop a redundancy reduction algorithm so as to remove neurons which represent the same weight vectors arising from different partitions, efficiently in order to optimise the composition of the map.

5. To evaluate the performance of the distributed algorithm to ensure it is viable and acceptable in practical applications.

6. To compare the accuracy of the new distributed method with the traditional SOM to ensure that clustering accuracy is not compromised in the distributed method.

Work done to achieve the research objectives are described in the sections that follow.

3.1 The Distributed GSOM Algorithm

The Distributed GSOM algorithm aims to improve the time consumption of the analysis process by decreasing the time requirement for the training of the SOM. The Distributed GSOM algorithm utilises a divide and conquer approach to efficiently process large datasets. The algorithm operates in four phases as listed below.

i. Data partitioning for generating multiple SOMs

ii. Training multiple SOMs in parallel

iii. Redundancy reduction to prune the SOMs

iv. Merging of multiple maps to generate a single map

The Distributed GSOM has a similar architecture to that of the PartSOM given in Goil et al. (1999). The primary differences between the two methods are the data layout and the merging methods. The Distributed GSOM uses a horizontal data layout as opposed to the vertical data layout of the PartSOM. The Distributed GSOM utilises Sammon's projection (Sammon Jr, 1969) for creating the topographic mapping, whereas the PartSOM uses the SOM algorithm. The differences between using Sammon's projection and the SOM for the merging process are discussed further in 3.5 below.

The first step of the Distributed GSOM algorithm is to partition the large input dataset into a number of non overlapping subsets. Each subset has to be small enough to reduce SOM training time significantly yet large enough to facilitate adequate learning. Several partitioning methods are possible and are discussed in 3.2. An SOM is trained on each of the subsets concurrently. Since the SOMs can operate independently on each partition, the algorithm can utilise a distributed memory model which accommodates higher levels of scalability as given in Lawrence et al. (1999). Since the SOM's execution

time is proportional to the size of the dataset, using only a subset of the data considerably reduces time consumption. The parallel training phase is the primary contributor to the reduction in execution time.

Due to partitioning, the classes within data may get distributed across the partitions. Therefore, two different SOMs trained in parallel may contain neurons that represent the same information. The presence of such neurons creates redundancy within the SOMs and should be removed from the output map. The redundancy reduction process identifies such neurons and removes them before the final map is created.

The main advantage of using an SOM for data exploration is the SOM's data visualisation properties. Due to the creation of multiple SOMs in parallel, visualisation of the entire dataset on a single map is lost. Therefore, once the redundant neurons are minimised, the SOMs have to be combined to provide a holistic view of the dataset. Sammon's projection is used to combine the SOMs and to arrange all the neurons topographically as a single map.

Figure 3.1 shows the architecture of the Distributed GSOM algorithm. Each phase of the algorithm is discussed in detail in the following sections.

3.2 Data Partitioning

The horizontal or vertical data layout of the algorithm is critical to the partitioning process. Horizontal partitioning splits the dataset into smaller groups with the same number of attributes but with fewer vectors in each group. In vertical partitioning, the number of records in each partition is equal to the number of records in the dataset; however, the attribute columns in each partition would be a subset of the attributes in the dataset. If two closely related attributes are assigned to two partitions, the relationship may not surface during the analysis. For exploratory data analysis, preserving the attribute relationships is critical. In addition, for large scale data analysis tasks, horizontal partitioning delivers higher levels of scalability as discussed in Faro et al. (2011). Therefore, horizontal partitioning is used for the data partitioning stage of the Distributed GSOM.

Data partitioning is the first stage of the algorithm where the input dataset is partitioned into smaller non-overlapping subsets. The number of partitions is critical to the performance of the algorithm and the quality of the output. Having a larger number of

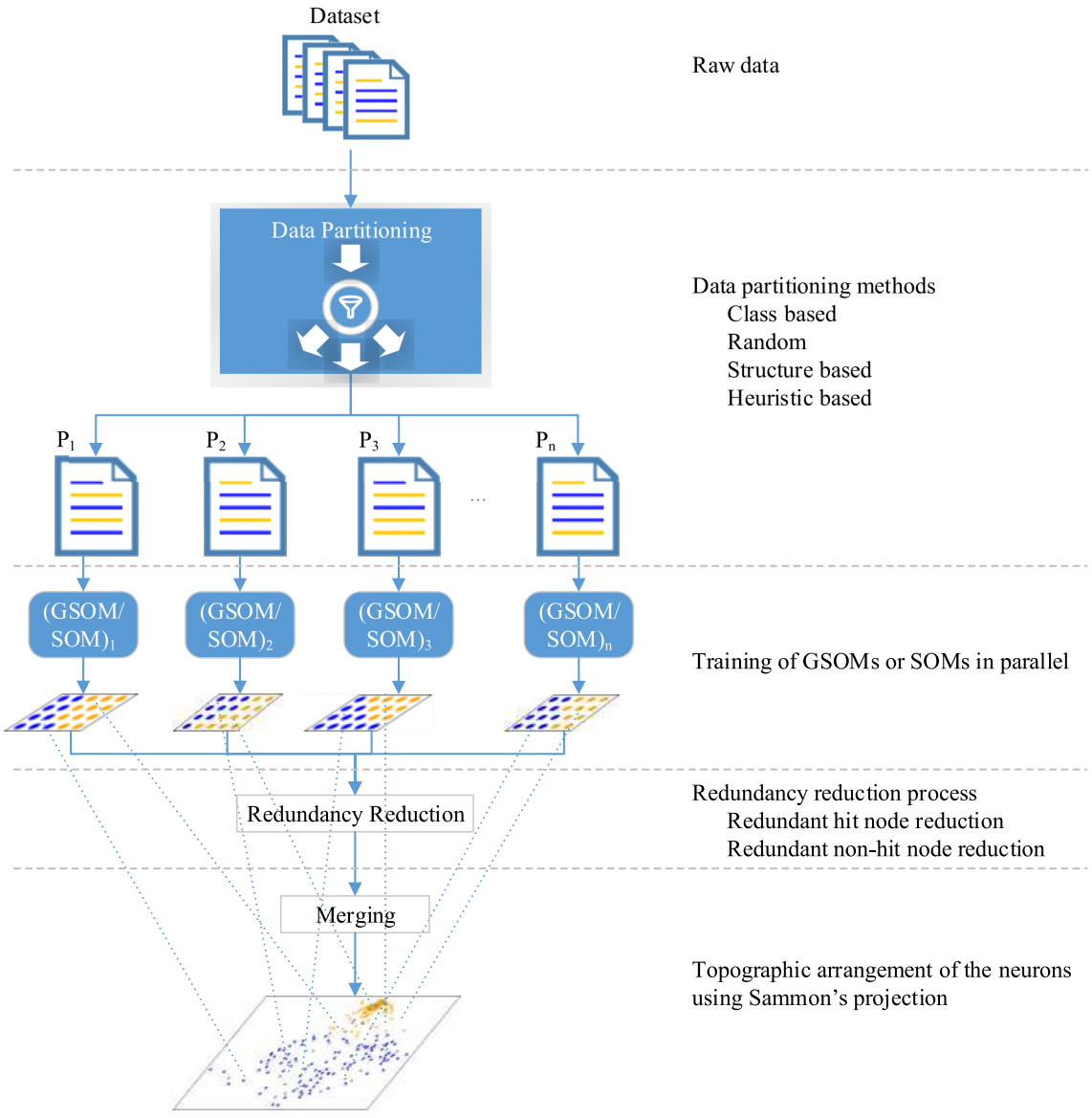


Figure 3.1: The Distributed GSOM algorithm

partitions results in fewer vectors per partition. While fewer input vectors reduces the time consumption of the parallel SOM training process, at the same time, the dataset may be too small for the SOM to achieve convergence. Hush and Horne (1993) suggest that the number of vectors presented to a neural network should be at least ten times the number of attributes in the dataset. This rule of thumb has been used for self-organising maps in Holub et al. (2012). Therefore, the maximum number of partitions (P_{max}) can be derived as

$$P_{max} = \frac{N}{10 \times D} \tag{3.1}$$

where N is the number of vectors in the dataset and D is the number of dimensions.

Ideally, the number of computing nodes should be greater than or equal to the number of partitions so that all the SOMs can be trained in parallel. If the number of computing nodes is less than the number of partitions, making the number of partitions equal to a multiple of the number of computing nodes yields the optimal performance.

The time complexity of the partitioning process is significant to the performance of the Distributed GSOM algorithm. A number of approaches for performing horizontal partitioning have been identified. Random partitioning, class based partitioning, high level cluster based partitioning and structure based partitioning are some examples of horizontal partitioning techniques. Using a high level cluster based partitioning technique, such as in Yang and Ahuja (1999), results in excessive time consumption which is not practical for large scale datasets due to the use of SOMs in the partitioning algorithm. The fastest data partitioning techniques are discussed below in detail.

i. Random partitioning

ii. Class based partitioning

iii. Structure based partitioning

iv. Heuristic based partitioning

3.2.1 Random partitioning

With the absence of any observed or derived information about the underlying patterns in data, the dataset can be randomly partitioned into P groups. Each vector in the dataset is randomly assigned to one of the P partitions. Since random partitioning is a linear

process, massive datasets can be efficiently processed. It can be assumed that the data distribution in the data partitions is a close approximation to the data distribution in the complete dataset.

Although simple and efficient, random partitioning could create higher levels of noise compared to other partitioning techniques. If the underlying cluster distribution is skewed, some partitions may under or over represent the dataset. As a result, cluster boundaries may get distorted in the final output. For clustering exercises with a clear separation of clusters, the distortions created by random partitioning are minimal as discussed in O'Connor and Herlocker (1999). In addition, the experiments conducted with respect to the Distributed GSOM also show that random partitioning does not significantly reduce the accuracy compared to other techniques.

3.2.2 Class based partitioning

If the underlying classes in the dataset are readily available, each partition could represent a class in the data. The primary advantage of partitioning the dataset by classes is the low levels of distortion in the SOMs trained in parallel. If each partition contains records from all the classes, the neurons in each map would have to represent all the classes, whereas if each partition contains records from only one class, the resulting map would consist of a better spread of neurons and less distortion. In addition, during the redundancy reduction phase, class based partitioning tends to create lower levels of redundancy compared to random partitioning.

However, the use of class information for partitioning could create bias towards the classes. If the classes in the dataset are different from the classes used for partitioning, over fitting could result, as discussed in Obradovic and Vucetic (2004).

3.2.3 Structure based partitioning

If the dataset possesses an inherent structure, partitions can be created according to structural formations. The structure could be time based, geography based or based on any other attribute that creates logical groups within data. Structure based partitioning would assist in the exploration of the dataset by creating a layer of maps which enables analysts to drill down through the structure. Structure based partitioning has been used

in clustering ontologies (Stuckenschmidt and Klein, 2004). However, over fitting can result if the structure does not reflect the patterns within the dataset.

For example, a stream of electricity consumption data over an year could be partitioned by months. Such partitioning could enable analysts to incorporate drilling down into the analysis process itself since the combined map would contain data for the whole year whereas the partitions contain maps for each month (an example of over fitting).

3.2.4 Heuristic based partitioning

The dataset may contain internal groupings based on a specific property. For such datasets, a heuristic may exist which could be used to group the dataset into partitions. A heuristic would be a rule of thumb which can be used to group records in a dataset intuitively. The heuristic is determined by the purpose of the analysis task and the features of the dataset; it could be an existing attribute or a calculated property. Heuristic based partitioning has been discussed in detail in Kriegel et al. (2009).

For example, daily electricity consumption analysis could use grouping by average electricity consumption in order to improve the coherence of the data within a partition. Having highly varying electricity consumption values within a single partition could create bias towards higher electricity consumption records due to the higher magnitude of the weight vector. By grouping records by average consumption, the records within a partition are similar in magnitude and would create less bias.

3.3 Parallel network training

Once the partitioning phase is complete, an SOM or a GSOM is trained on each partition in parallel, provided that the number of partitions exceed the number of available computing nodes. The training phase is the primary contributor to the efficiency of the Distributed GSOM. The time complexity of the SOM algorithm is linear on the number of vectors and quadratic on the number of neurons. The relationship between the number of neurons (N) and the size of the dataset (n) is

$$N = 5\sqrt{n} \tag{3.2}$$

as given in Vesanto and Alhoniemi (2000). By reducing the number of each partition, both the number of vectors and the number of neurons in each network are reduced. Therefore, the decrease in time consumption is exponential with the increase in the number of partitions. An SOM or a GSOM trained on a data partition is referred to as a *partition network*. The primary difference between the two techniques is the structure of the map: the SOM has a static structure and the GSOM has a dynamic structure.

The width and the height of the SOMs should be specified which would determine the shape of the map by the width to height ratio and the level of detail by the number of neurons. Since the vectors in the each partition may have different characteristics, a uniform width and height across all the partitions may not be suitable. According to Kim and Han (2001), a theoretical method for determining the optimum number of neurons of a SOM does not exist. Determining the optimum width and height of the map is a subjective process which is usually done by trial and error, as discussed in Koikkalainen and Oja (1990) and Alahakoon et al. (2000). Having too many or too few neurons would over or under represent the input dataset in the map according to Koikkalainen and Oja (1990). In addition, the high time consumption when processing large volumes makes the trial and error approaches impractical.

Due to the dynamic structure of the GSOM, the number of neurons in the map is determined by the level of detail specified by the spread factor. As a result, the structure of the GSOM would have different shapes depending on the shape of the dataset. Consequently, the GSOM has the ability to adapt its structure to accommodate the dataset in different partition configurations. A global spread factor could be used to ensure the same level of detail is created across all partition networks, which is more suitable for data exploration.

The GSOM algorithm is considered to be the preferred learning method for large scale data analysis exercises for the distributed algorithm. Chapter 4 demonstrates the advantages of the GSOM for the distributed algorithm by comparing both the SOM and the GSOM as exploratory data analysis techniques.

3.4 Redundancy Reduction

At the end of the training phase, the partition networks need to be merged to form a single map representing the entire input dataset. However, if at least two partitions contained similar records from the same class (which could result from random partitioning), there is a possibility that some neurons representing the same class of vectors. If two neurons from two partition networks represent the same information, this is considered to be a redundancy. Having redundant neurons in the output map could result in high processing power requirements and lower accuracy levels. Since the time complexity of Sammon's projection is $O(n^2)$, having more neurons as input could lead to a rapid increase in time consumption for the merging process. In addition, with the presence of multiple neurons representing the same set of input vectors, it could be difficult to distinguish cluster boundaries. In order to decrease the level of redundancy in the merged map, a redundancy reduction process is carried out.

Several redundancy reduction methods have been proposed in the literature. Fu et al. (2001) have proposed a redundancy reduction method whereby if two neurons are closer than a pre-defined maximum distance, the neurons are merged. For exploratory analysis, the determination of the maximum distance may be impossible without an initial analysis. In addition, the proposed method requires a distance matrix calculated for all the neurons within intermediate outputs. For a detailed analysis with millions of neurons, the distance matrix calculation process could reduce the efficiency of the Distributed GSOM.

A related redundancy reduction method is proposed in Qiao and Han (2010) where redundant neurons are identified by inspecting the neighbourhood of the neurons. Since the neurons in the partition networks are in different networks, determining a universal neighbourhood is impossible in this phase. It is possible to apply this method on the single map created from using Sammon's projection when the neighbourhoods are defined by topographic ordering. However, due to the presence of redundant neurons in the input to Sammon's projection, the overall efficiency of the algorithm would deteriorate.

In this section, a novel redundancy reduction algorithm is proposed which is suitable for exploratory data analysis tasks. The quantisation error (QE) of the neuron is used to determine redundancy. The QE is an indication of how well the weight vector of the neuron matches the weight vectors of the vectors mapped onto it. A lower QE means a

better match and vice versa. Using the QE as a measure, a redundant neuron is defined as follows.

If there exists a neuron n_i such that the records mapped to n_i result in a lower quantisation error on neuron n_j, it can be concluded that n_j is a better candidate to accommodate the records of n_i. Therefore n_i is considered a redundant neuron.

3.4.1 Notations

The following notations are used for describing the algorithms.

$n_{i,j}$ The neuron j in partition i

$w_{i,j}$ The weight vector of $n_{i,j}$

$I_{i,j}$ The list of input vectors mapped to $n_{i,j}$

$N_{i,j}$ The number of vectors in $I_{i,j}$

$E_{i,j}$ The total quantisation error for $n_{i,j}$ for $I_{i,j}$

$E_{i,j}^{p,q}$ The total quantisation error for $n_{i,j}$ for $I_{p,q}$

GetBMU($w_{i,j}, k$) Returns the closest neuron to $w_{i,j}$ in partitioned network k

IsHitNeuron($n_{i,j}$) Returns true if $n_{i,j}$ is a hit neuron or false otherwise

IsNonHitNeuron($n_{i,j}$) Returns true if $n_{i,j}$ is a non-hit neuron or false otherwise

An SOM or a GSOM training on a single dataset will not create any redundant neurons. Since each partition network trains on a non overlapping subset of the dataset, redundant neurons would not be present within a single partition. Therefore, each partition is compared against other partitions for redundant neurons. In order to determine whether a neuron is redundant, the quantisation error for the neuron would have to be calculated. A neuron will have a quantisation error only if the neuron has at least one input vector mapped onto it. Such a neuron is called a hit neuron where for at least one vector in the input space, the neuron is the best matching unit. On the other hand, a non-hit neuron is one where none of the vectors in the input space are mapped onto it. As a result, redundant non-hit neurons cannot be identified using the quantisation error comparison. In order to identify both redundant hit-neurons and redundant non-hit neurons, redundancy reduction is performed in two stages: redundant hit neuron reduction and redundant non-hit neuron reduction.

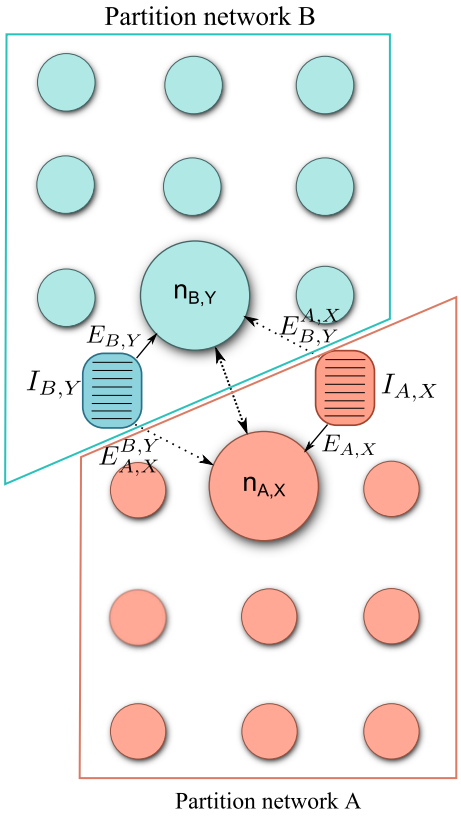


Figure 3.2: The redundancy reduction process

3.4.2 Redundant hit neuron reduction

Figure 3.2 helps explain the redundant hit neuron reduction algorithm. The algorithm starts by selecting a partition at random and for each hit neuron in the map, the closest neurons from the other maps are identified. However, since only two comparisons can be done at a time, only two networks are considered simultaneously. For example, if the two networks are A and B, for each hit neuron $n_{A,X}$, the best matching neuron in network B, $n_{B,Y}$, is identified since the best matching units across networks have the highest potential for creating redundancy. Equation (3.3) is used to identify $n_{B,Y}$.

$$n_{B,Y} = n_i : min\{||w_i - w_{A,X}||\}\ \forall n_i \in B \tag{3.3}$$

In order to check for redundancy, the input vectors $I_{A,X}$ mapped onto $n_{A,X}$ are mapped onto $n_{B,Y}$ and the error $E^{A,X}_{B,Y}$ is calculated using equation (3.4).

$$E^{A,X}_{B,Y} = \sum_{i=1}^{|I_{A,X}|} ||w_{B,Y} - w_{I_{A,X}[i]}|| \tag{3.4}$$

If the original error, $E_{A,X}$ for $I_{A,X}$ on $n_{A,X}$, is greater than $E_{B,Y}^{A,X}$ then $n_{A,X}$ is considered redundant since there exists a better candidate, $n_{B,Y}$, with a lower quantisation error for $I_{A,X}$.

If $E_{A,X}$ is less than $E_{B,Y}^{A,X}$ there could be a possibility that $n_{B,Y}$ is redundant. If $n_{B,Y}$ is a non-hit neuron, the comparison has to stop. However, in the case of $n_{B,Y}$ being a hit neuron and if $I_{B,Y}$ are mapped onto $n_{A,X}$ and $E_{A,X}^{B,Y}$ is less than $E_{B,Y}$, then $n_{B,Y}$ is considered redundant. Otherwise, if the total error for $I_{A,X}$ and $I_{B,Y}$ on a single neuron is less than the combined error $E_{A,X} + E_{B,Y}$, the node with the higher error is considered redundant and is removed.

The redundant hit node reduction algorithm is given in Algorithm 1. The algorithm returns a measure computed based on the average Euclidean distance between each redundant neuron and the corresponding retained neuron in order to determine redundant non-hit neurons. The logic behind calculating the measure is described in the following section.

3.4.3 Redundant non-hit neuron reduction

A non-hit neuron is a neuron which has not become the BMU for any of the vectors in the dataset. As a result, non-hit neurons do not accumulate any quantisation errors. As a result, the QE comparison approach used for the hit neuron reduction algorithm cannot be used for non-hit neurons.

Since the only property of a non-hit neuron is its weight vector, a redundancy measure has to be created around neuron distances. Since the purpose of non-hit neurons is to create spacing between hit neurons, it can be assumed that redundant non-hit neurons would be arranged in clusters in the final map. In order to determine the redundant non-hit neurons, the radius of the redundant neuron clusters has to be calculated. The radius is calculated during the redundant hit neuron reduction phase and is referred to as the *Redundancy Index (RI)*. The redundancy index is calculated as the average distance of the removed neurons and their best matching counterparts, which is given by equation (3.5).

$$RI = e^{SF} \times \frac{\sum |w_{BMU} - w_{redundant}|}{N_R \times D} \tag{3.5}$$

Algorithm 1 Redundant hit neuron reduction algorithm

1:
2: Let $P =$ All partitioned networks
3: $d = 0, n = 0$
4: **for all** partition $P_i \in P$ **do**
5: **for all** hit neuron $n_{i,j} \in P_i$ **do**
6: **for all** partition $P_k \in P$ where $i \neq k$ **do**
7: $n_{k,l} = \text{GetBMU}(w_{i,j}, k)$
8: $E_{k,l}^{i,j} = \sum_{x=0}^{N_{i,j}} |w_{k,l} - I_{i,j}[x]|$
9: $E_{i,j}^{k,l} = \sum_{x=0}^{N_{k,l}} |w_{i,j} - I_{k,l}[x]|$
10: **if** $E_{i,j} > E_{k,l}^{i,j}$ **then**
11: $I_{k,l} = I_{k,l} + I_{i,j}$
12: remove $n_{i,j}$
13: $d = d + |w_{i,j} - w_{k,l}|$
14: $n = n + 1$
15: **else**
16: **if** $E_{k,l} > E_{i,j}^{k,l}$ **then**
17: $I_{i,j} = I_{k,l} + I_{i,j}$
18: remove $n_{k,l}$
19: $d = d + |w_{k,l} - w_{i,j}|$
20: $n = n + 1$
21: **else**
22: $E_C = E_{i,j} + E_{k,l}$
23: $E_1 = E_{i,j} + E_{i,j}^{k,l}$
24: $E_2 = E_{k,l} + E_{k,l}^{i,j}$
25: **if** $E_C > E_1$ & $E_1 < E_2$ **then**
26: $I_{i,j} = I_{k,l} + I_{i,j}$
27: remove $n_{k,l}$
28: $d = d + |w_{i,j} - w_{k,l}|$
29: $n = n + 1$
30: **else if** $E_C > E_2$ & $E_2 < E_1$ **then**
31: $I_{k,l} = I_{k,l} + I_{i,j}$
32: remove $n_{i,j}$
33: $d = d + |w_{i,j} - w_{k,l}|$
34: $n = n + 1$
35: **end if**
36: **end if**
37: **end if**
38: **end for**
39: **end for**
40: **end for**
41: **return** $\frac{e^{SF} \times d}{D \times n}$

where SF is the spread factor, $w_{redundant}$ is the weight vector of each redundant neuron and w_{BMU} is the weight vector of each corresponding best matching unit, N_R is the total number of neurons removed due to redundancy and D is the number of dimensions. The redundancy index is normalised by the number of dimensions in order to compare the level of redundancies across datasets. Algorithm 2 describes the redundant non-hit neuron reduction process.

Algorithm 2 Redundant non-hit neuron reduction algorithm

Require: RI =Redundancy Index
Let P = All partitioned networks
for all partition $P_i \in P$ **do**
  for all neuron $n_{i,j} \in P_i$ **do**
    if IsNonHitNeuron($n_{i,j}$) **then**
      for all partition $P_k \in P$ such that $i \neq k$ **do**
        for all neuron $n_{k,l} \in P_k$ **do**
          if IsNonHitNeuron($n_{k,l}$) **then**
            if $|w_{i,j} - w_{k,l}| \leq (RI \times D)$ **then**
              remove $n_{k,l}$
            end if
          end if
        end for
      end for
    end if
  end for
end for

At the end of the redundancy reduction phase, partition networks are stripped of redundant neurons and the input vectors that used to be mapped to those redundant neurons are allocated to the preserved neurons. As a result, the number of vectors presented to Sammon's projection is reduced which in turn improves the efficiency of the overall process.

3.5 Merging

The output of the redundancy reduction phase is a collection of neurons in their own partitions. However, in order to provide a holistic view of the input dataset, the neurons have to be combined to form a single map. Having the input represented in a single map has significant advantages over multiple smaller maps. The merged map has to be equivalent to training an SOM on the entire input dataset where the neurons are arranged according to their distance relationships in the high dimensional space.

Since the partition networks have undergone the expensive learning process, the purpose of this phase is the topographic arrangement of the neurons. It is possible to use a learning technique such as the SOM for the topographic projection. The SOM would create a topographic map using the weight vectors of the neurons as input. This would create bias towards averages since the weight vector of a neuron is an approximate average of the vectors mapped onto that neuron. As a result, a dimensionality reduction technique without incorporated learning is used for merging the neurons.

Sammon's projection (Sammon Jr, 1969) is used to create the output map due to its ability to arrange the neurons in the partition networks to form a single map topologically. Using Sammon's projection to project the neurons in an SOM has been presented as a means of improving the visualisation of the SOM in Trnen et al. (1999) and Demartines and Hrault (1997).

The key advantages of Sammon's projection over the SOM for creating a topographic map for the output of the redundancy reduction process are the absence of learning and the better visualisation features of Sammon's projection. These features are described below.

1. Absence of learning – Sammon's non-linear algorithm is a projection technique which does not involve any learning. Therefore, the knowledge accumulated (as weight vectors) during the training phase is preserved during merging. In contrast, a neural network algorithm would create new neurons based on the already trained input neurons. This bias towards the averages of the input vectors could result in limited representation, as discussed in Koskela et al. (1997). In addition, the number of input vectors available for ensuring quality could be insufficient to achieve convergence.

2. Better visualisation – Sammon's projection arranges the vectors in a continuous space whereas SOM based and GSOM based neural networks use a discrete space. A continuous space has the ability to arrange neighbours of a particular network arbitrarily, whereas a discrete space has limitations such as having equally close neighbours. The visualisation advantages of Sammon's projection are discussed in

Trnen et al. (1999). The main advantage is that Sammon's projection can visually show the inter neuronal distances whereas SOM based techniques require other methods such as U-Matrix (Ultsch, 2003).

3.6 Evaluation of the Distributed GSOM

The performance of the Distributed GSOM was measured for efficiency and accuracy. In order to consider a good cross section of the types of data, three datasets were used for the experiments. The datasets and their details are described in the following section. The scalability of the algorithm was evaluated with experiments for different partition configurations for each dataset. Each experiment was conducted five times for each partition configuration and the mean and the standard deviation statistics were calculated.

The experiment set up was in a simulated parallel computing platform using a multi core computer. Each computing node was assigned 2 Gigabytes of RAM and 2 GHz processing power.

3.6.1 Datasets

Wisconsin Breast Cancer (WBC) Dataset

The WBC dataset (Wolberg et al., 1992) is one of the most popular benchmark datasets used to measure the accuracy of classification and clustering algorithms (Wang, 2009). Since the dataset has clear classes, the WBC dataset was primarily used to evaluate the accuracy of the Distributed GSOM. The dataset contained 683 vectors from two classes. Each vector consisted of 9 attributes describing the features of the cells. The distribution of the classes was 444 records from benign cells and 239 records from malignant cells. Since the size of the dataset was small, only two and four partitions were used in the parallel training phase. The WBC dataset has been widely used to benchmark clustering algorithms, as given in Pantazi et al. (2002) and Chen et al. (2011).

Sydney Morning Herald (SMH) News Archive

The SMH news archive contains 19316 news articles published in the *Sydney Morning Herald* in 2009 (Matharage et al., 2013). The main classes in the dataset were considered

to be the categories of the articles, namely, 'National', 'Sport', 'World' and 'Entertainment' where the record percentages were 41%, 29%, 18% and 12%, respectively. Each article was represented as a vector with 3526 attributes using term frequency and inverse document frequency (TFIDF). The generated dataset with 19,316 records each containing 3526 attributes was a sparse dataset which was considered a moderately large analysis task. The SMH dataset was used to evaluate the performance of the Distributed GSOM for large sparse datasets. Experiments were conducted with 4, 8, 12 and 16 partitions in order to evaluate the scalability of the algorithm. Efficiency and accuracy results are analysed for 4, 8 and 16 partition configurations for brevity.

UCI CoverType Dataset

The CoverType dataset from the UCI machine learning repository contains 581,012 records from seven different forest cover types (Blackard et al., 1998). Each record consists of 54 attributes generated from cartographic variables. The distribution of the seven classes was 36.5%, 48.8%, 6.2%, 0.5%, 1.6%, 2.9% and 3.5% as percentages of the total number of records. Due to the large size of the dataset, experiments were run for 8, 12, 16, 20, 24, 28 and 32 partitions. The classification accuracy for the CoverType dataset has been in the range of 70% according to Blackard and Dean (1999). Although the cluster separation is not as clear as the WBC dataset, the CoverType dataset provided a good basis for evaluating the Distributed GSOM on large datasets. The clustering accuracy was evaluated for the largest two clusters consisting of 85.3% of the total records. Efficiency and accuracy results are analysed for 8, 16 and 32 partition configurations for brevity.

3.6.2 Redundancy Statistics

The effectiveness of the redundancy reduction algorithm was evaluated using the number of neurons before and after the redundancy reduction process. Table 3.1 shows the statistics for the SMH dataset. The total number of neurons created in partition networks increases with the number of partitions. Therefore, the level of redundancy should also increase. Results show that the redundancy reduction process removes redundant neurons to create similar sized outputs. The redundancy reduction percentage is calculated as the number of redundant neurons relative to the total number of neurons in the partitioned networks.

Table 3.1: Redundancy reduction (RR) statistics for SMH dataset

Partitions	Neuron count before RR	Redundant neuron %	RI
4 random	295	44.9%	0.0074
8 random	339	44.8%	0.0074
16 random	414	45.7%	0.0075
Class	303	5.5%	0.0122

Table 3.2: Redundancy reduction(RR) statistics for CoverType dataset

Partitions	Neuron count before RR	Redundant neuron %	RI
8 random	14,421	26.3%	0.0110
16 random	19,425	38.1%	0.0126
32 random	26,450	52.8%	0.0154
Class	15,028	41.3%	0.0149

The percentage reduction of neurons is 53%, 61% and 66%, respectively, for 4, 8 and 16 partitions with random partitioning.

Class based partitioning, on the other hand, results in fewer redundant neurons due to the distribution of classes among the partitions. The percentage reduction of neurons is 13%, which is significantly lower than the number of neurons removed with random partitioning.

Similar results can be observed for the CoverType dataset as shown in Table 3.2. It can be observed that class based partitioning has resulted in a significant number of neuron removals. Due to the skewed distribution of the classes, multiple partitions were created for the two largest classes. As a result, the level of redundancy increases, hence the higher number of neuron removals.

A detailed discussion on the implications of random and class based partitioning are presented in section 3.6.5.

3.6.3 Efficiency Analysis

The efficiency of the algorithm was evaluated by comparing the total time consumption for processing the dataset. Due to its smaller size, the time consumption for the WBC dataset is in the order of two seconds. On the other hand, due to the higher number of records, SMH and CoverType datasets were used to benchmark the Distributed GSOM against the serial GSOM. Different partition configurations were used to assess the scalability of the algorithm which will be discussed in section 3.6.4.

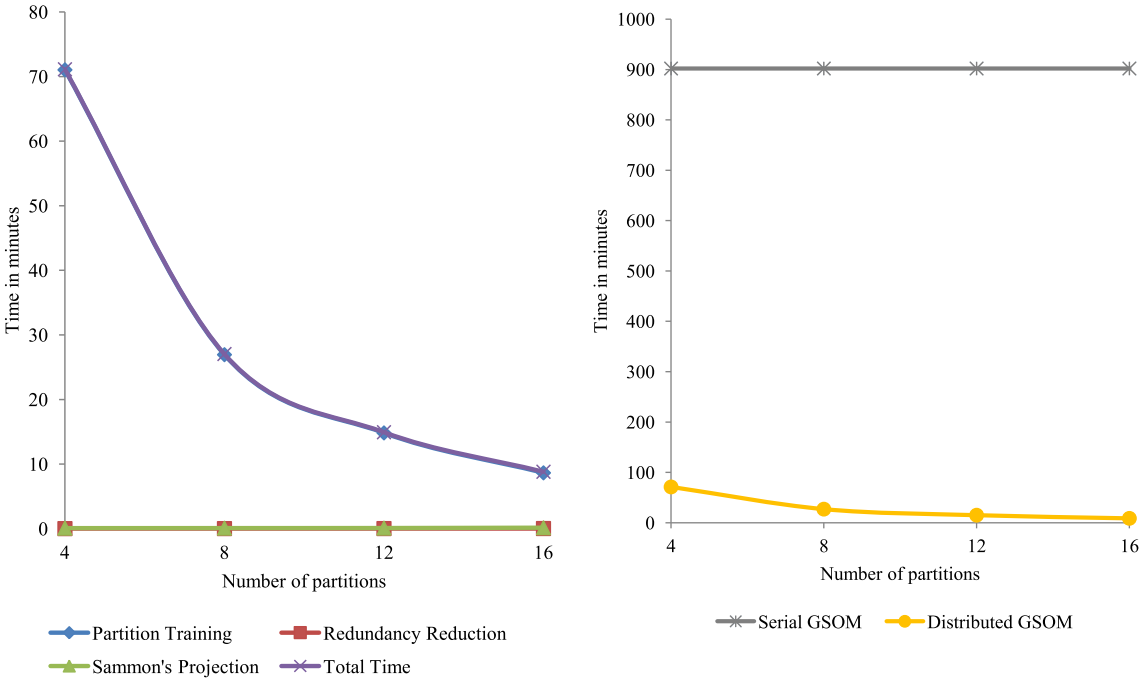


Figure 3.3: Total time consumption of serial GSOM and Distributed GSOM for the SMH dataset. (a) Time consumption of the different stages of the Distributed GSOM algorithm; (b) Time consumption of the Distributed GSOM vs the serial GSOM.

Figure 3.3 shows the time consumption comparison for serial GSOM and Distributed GSOM for the SMH dataset. Average time consumption of the five test runs was used to create the charts. The breakdown of the time consumption of the Distributed GSOM is shown in chart (a). The chart shows a significant reduction in the time consumption of the partition network training process as the number of partitions increase. This effect can be attributed to fewer vectors being used as input to the partition network with a higher number of partitions.

The time consumption values of the redundancy reduction process and Sammon's projection are considerably lower than that of the partition training phase. The difference can be attributed to the high number of dimensions and the low number of neurons in the output. Since the SMH dataset consists of 3526 attributes, the time consumption of the partition network training process is comparatively high. The redundancy reduction statistics are shown in Table 3.1 which displays the size of the input to Sammon's projection. As the number of neurons in the output is consistent, the time consumption of Sammon's projection is approximately constant.

Chart (b) in Figure 3.3 depicts the time consumption of the Distributed GSOM and the serial GSOM. Since the serial GSOM does not scale with the number of computing nodes,

the time consumption is constant. The chart demonstrates that the Distributed GSOM is several orders more efficient than the serial GSOM. The average time consumption of the Distributed GSOM was 7.9%, 3% and 1% of the average time consumption of the serial GSOM algorithm for 4, 8 and 16 partitions, respectively. Thus the Distributed GSOM would result in significantly less turnaround time in exploratory data analysis tasks.

The time consumption values for the CoverType dataset are shown in Figure 3.4. The time consumption breakdown of the different stages of the Distributed GSOM algorithm are shown in chart (a). Compared to the SMH dataset, some key differences can be observed. The time consumption of Sammon's projection is greater than that of partition network training.

The reason for the difference in time consumption can be explained using the redundancy reduction statistics given in Table 3.2. Since the dense CoverType dataset consists of only 54 attributes which is significantly lower than the 3526 attributes of the sparse SMH dataset, the time consumption of the partition network training phase is lower. The input for Sammon's projection on the other hand contains a high number of neurons, in the order of 10,000. The time complexity of Sammon's projection is $O(n^2)$ on the size of the input. The time consumption of Sammon's projection therefore is high due to the higher number of neurons created.

Chart (b) in Fig. 3.4 shows the time consumption comparison of the Distributed GSOM and the serial GSOM. The total average time consumption figures of the Distributed GSOM, as a percentage of the average time consumption for the serial GSOM, were 25%, 22% and 18% for 8, 16 and 32 partitions, respectively. Although creating more than 32 partitions could yield in higher efficiency levels, the maximum number of partitions was set to 32 as the simulated environment had a maximum of 32 nodes.

It is evident from the efficiency analysis that the Distributed GSOM performs significantly faster than its serial counterpart. As a result, the Distributed GSOM would decrease the time consumption of the data processing stage of exploratory data analysis thus increasing the efficiency of the entire process. The Distributed GSOM would enable analysts to perform a deeper analysis of data since the turnaround time is shortened.

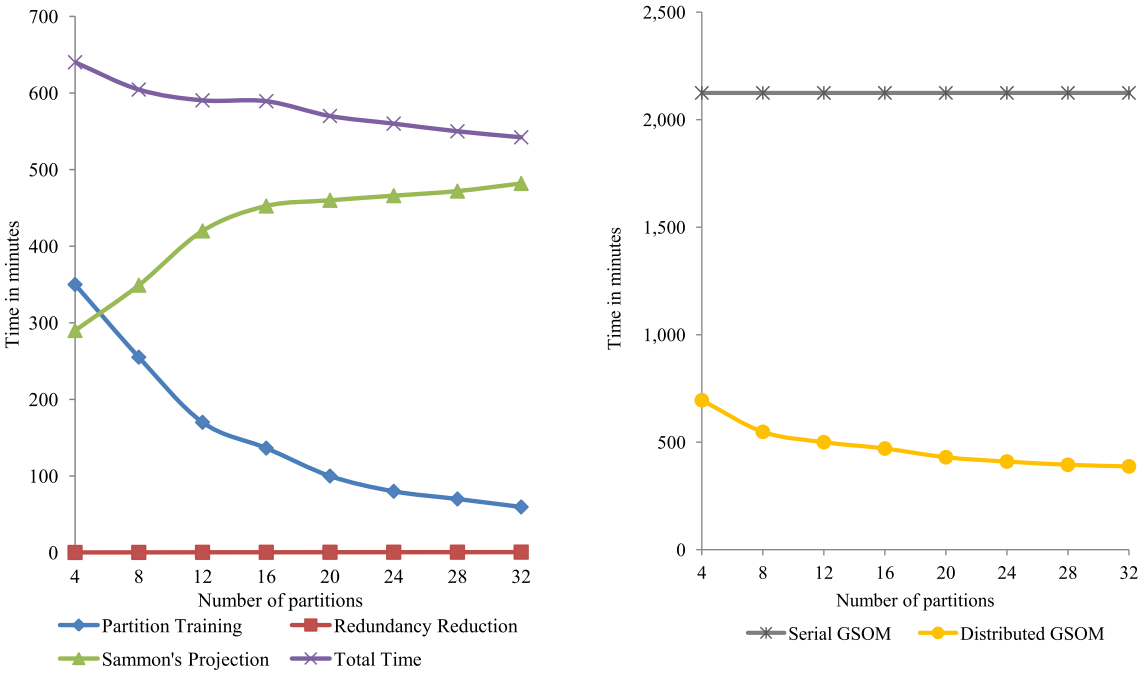


Figure 3.4: Total time consumption of serial GSOM and Distributed GSOM for the Cover-Type dataset. (a) Time consumption of the different stages of the Distributed GSOM algorithm; (b) Time consumption of the Distributed GSOM vs the serial GSOM.

3.6.4 Scalability analysis

Figure 3.3 and Figure 3.4 also show that, with the increase of resources, the Distributed GSOM handles the computational load gracefully. The total time consumption continues to decrease as the number of partitions increases. However, depending the number of records and the attributes in a dataset, there could be restrictions to the maximum number of partitions. In situations where accuracy could be compromised for efficiency, the number of partitions could be increased.

3.6.5 Accuracy analysis

Although the primary goal of the Distributed GSOM is to provide an efficient means of processing data, the accuracy of the analysis would have to be comparable to that of the GSOM. In this section, clustering accuracy of the Distributed GSOM is benchmarked against the GSOM algorithm. In addition, the effects of different partition methods will also be discussed.

Distributed GSOM vs serial GSOM

The accuracy of the algorithms was compared using the clustering accuracy measure, the F-measure. The WBC dataset was used as the basis for comparing accuracy since it has been widely used for evaluating clustering algorithms. Results for the SMH and the CoverType datasets are also listed in order to show that the Distributed GSOM is capable of delivering similar levels of accuracy to the GSOM.

The F-measure values for the two classes in the WBC dataset are shown in Table 3.3.

Table 3.3: F-measure and the standard deviation (σ) of the F-measure for the WBC dataset for benign (B) and malignant (M) classes

Technique	F-measure			σ (F-measure)		
	B	M	Overall	B	M	Overall
GSOM	0.9171	0.9323	0.9256	0.0096	0.0079	0.0084
Random Partitioning, 2 partitions	0.9204	0.9358	0.9290	0.0058	0.0085	0.0073
Random Partitioning, 4 partitions	0.9134	0.9294	0.9222	0.0199	0.0166	0.0178
Class based partitioning	0.9166	0.9433	0.9321	0.0054	0.0064	0.0056

The results show that the Distributed GSOM has higher levels of accuracy than the GSOM for most of the partition configurations, with the exception being random partitioning with four partitions. The reason for these higher levels of accuracy could be attributed to the better topology preservation of Sammon's projection compared to the GSOM. Higher levels of topology preservation would lead to better accuracies in clustering. The standard deviation values demonstrate that the Distributed GSOM maintains a comparable level of stability against the GSOM. Therefore, it could be concluded that the Distributed GSOM delivers similar levels of accuracy to the GSOM.

The minor decrease in accuracy in the case of four random partitions could be attributed to the decreased number of records in each partition. The number of input vectors have to be sufficient to achieve convergence in SOM based data analysis. A smaller number of input vectors could be compensated for by increasing the number of iterations in the training phase. In order to evaluate the Distributed GSOM's performance in standard conditions, the number of iterations was the same across all experiments. As a result, the lower number of records in the case of four partitions could have resulted in the minor decrease in accuracy and the increase in the standard deviation.

Table 3.4: F-measure and the standard deviation (σ) of the F-measure for the SMH dataset for different number of partitions (P) and partitioning methods, random (R) and class based. The classes are news (N), world (W), sport (S) and entertainment (E).

Method	F-Measure					σ(F-Measure)				
	N	W	S	E	Avg.	N	W	S	E	Avg.
GSOM	0.808	0.704	0.647	0.924	0.804	0.006	0.012	0.026	0.003	0.005
R, 4P	0.798	0.700	0.612	0.938	0.795	0.029	0.027	0.027	0.005	0.010
R, 8P	0.784	0.697	0.564	0.930	0.784	0.009	0.028	0.049	0.006	0.011
R, 16P	0.766	0.690	0.536	0.923	0.768	0.009	0.014	0.032	0.007	0.003
Class	0.893	0.853	0.821	0.971	0.898	0.002	0.005	0.003	0.002	0.002

Similar observations were made in the case of the SMH dataset as summarised in Table 3.4. It should be noted that the 19,316 records in the SMH dataset, which has 3526 attributes, poses a challenge for the convergence of the network. As a rule of thumb, Deboeck and Kohonen (1998) suggest that the number of neurons in the network has to be greater than ten times the number of attributes for good training. However, due to the sparsity of the data as well as the selected spread factor, the average number of neurons in the serial GSOM analysis was 176. As a result, the deviations in the F-measure in different partition configurations tend to be higher as the number of partitions increases.

Table 3.5 shows the accuracy results for the CoverType dataset which shows similar accuracy levels to the WBC dataset. All results for different partition configurations result in higher levels of accuracy compared to the serial algorithm. The primary reason for higher levels of accuracy for the CoverType dataset and lower levels of accuracy for the SMH dataset could be that there are fewer input records per partition. As the number of attributes increases, the number of input records required for good training of the network increases. Therefore, finding a healthy balance between the number of partitions and the number of input records is important.

Random partitioning vs class based partitioning

F-measure results show that the difference between random partitioning and class based partitioning for WBC and CoverType is nominal. Both datasets are dense and the number of records relative to the number of attributes is low. As a result, the partitions would have had a sufficient number of records to converge the partition networks. The F-measure values therefore do not deviate significantly between these two methods.

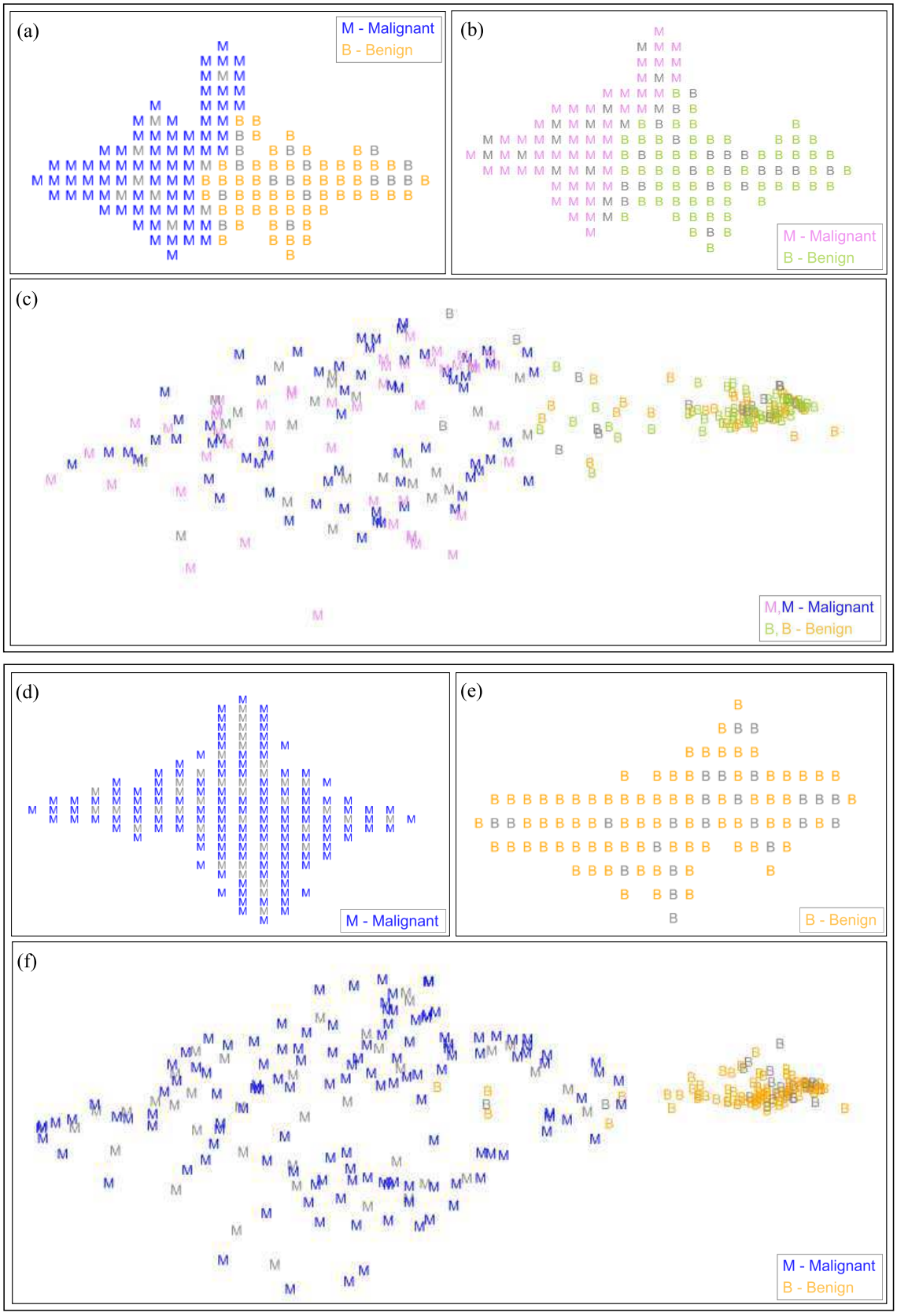


Figure 3.5: Maps generated by random and class based partitioning. (a), (b) Partition network for random partitioning. (c) Output of the Distributed GSOM for random partitioning. (d) Partition network for the malignant (M) class. (e) Partition network for the benign (B) class. (f) Distributed GSOM output for class based partitioning.

Table 3.5: F-measure and the standard deviation (σ) of the F-measure for the CoverType dataset for different number of partitions (P) and methods (M)

Technique	F-measure			σ(F-measure)		
	1	2	Overall	1	2	Overall
GSOM	0.6554	0.7781	0.7278	0.0104	0.0102	0.0072
Random, 8P	0.7548	0.8243	0.7953	0.0015	0.0008	0.0009
Random, 16P	0.7519	0.8224	0.7930	0.0019	0.0018	0.0018
Random, 32P	0.7412	0.8175	0.7860	0.0012	0.0013	0.0011
Class based	0.7492	0.8120	0.7851	0.0019	0.0005	0.0006

However, class based partitioning has yielded higher accuracy levels than both the Distributed GSOM with random partitioning and the serial GSOM. The increase in accuracy could be attributed to the lower number of records relative to the number of attributes. As a result, when the partitions are made up of records in a single class, each partition network would train to be specialised for each class, resulting in lower levels of distortion. Random partitioning, on the other hand, would struggle to converge the partition networks due to the high number of attributes.

3.6.6 Evaluation of visualisation properties

The wide application of the SOM in exploratory data analysis is primarily due to its visualisation properties. The creation of the two or three dimensional mapping of the dataset allows users to observe and isolate patterns within data. Sammon's projection, used in the Distributed GSOM, creates a similar topographic visualisation of the data. The primary difference between the maps created by the SOM and Sammon's projection is the layout. The SOM creates a discrete map whereas Sammon's projection creates a continuous map.

Figure 3.5 shows the maps created for the WBC dataset for random and class based partitioning. The map generated by the random partitioning approach shows the same topology as the map generated by class based partitioning. It can be noted that the two partition networks in random partitioning resemble the shape of Sammon's projection whereas the two class based partitions parallel the two class components of the map. The distribution of the non-hit nodes are shown in gray.

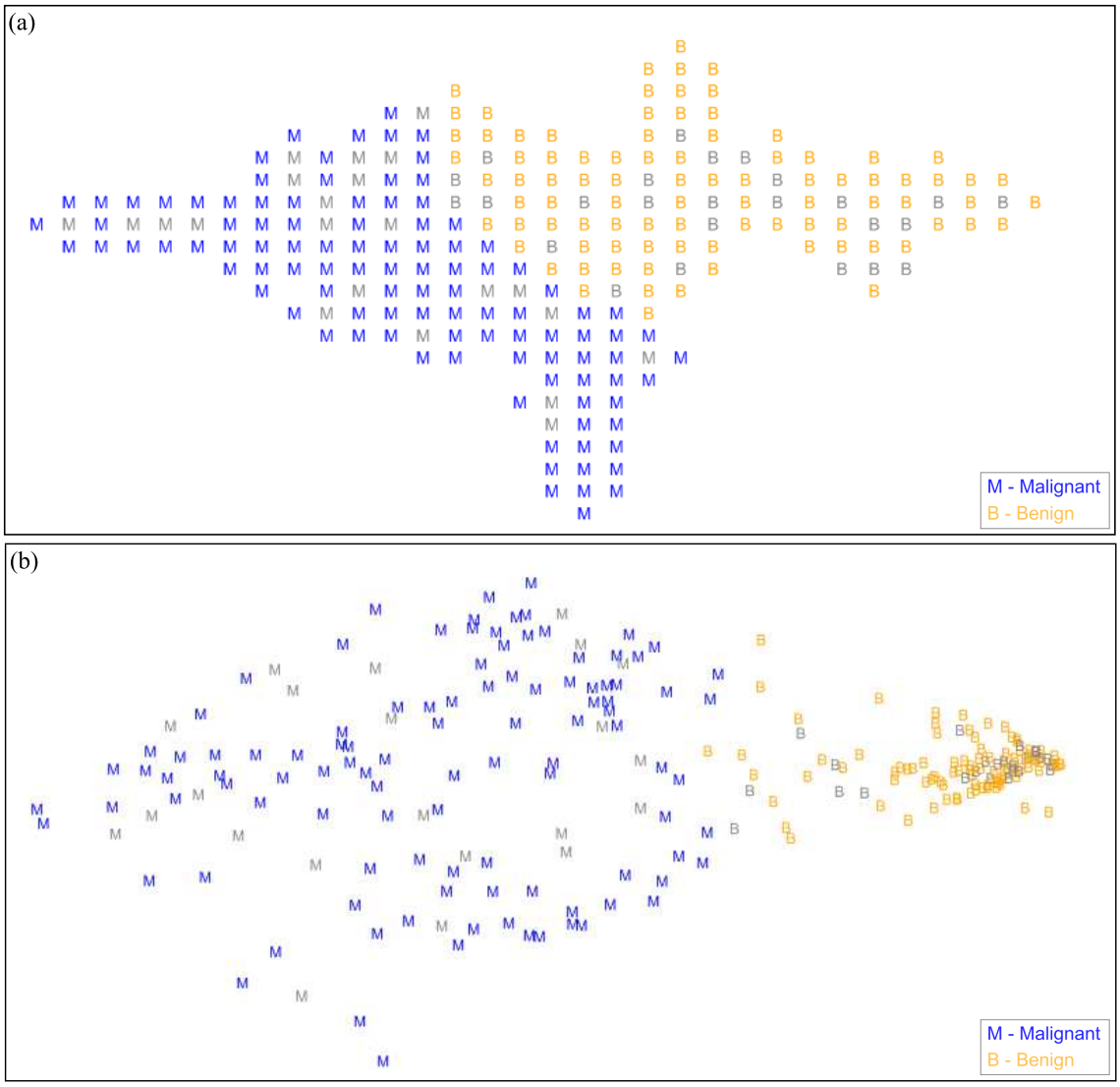


Figure 3.6: The GSOM and Sammon's projection of the GSOM of the WBC dataset. (a) the GSOM. (b) Sammon's projection of the GSOM

Figure 3.6 shows the GSOM and Sammon's projection of the GSOM for the same dataset. It is evident that the Distributed GSOM produces a topologically similar map with a clear separation of the two classes. The effect of the continuous map is also visible where Sammon's projection clearly separates the two classes into two concentrated masses. Thus the need for additional enhancements such as the U-Matrix is avoided.

3.7 Discussion

In this chapter, the Distributed GSOM algorithm is introduced as a novel method of utilising parallel processes to efficiently process large datasets using SOM based learning techniques. The results show a significant reduction in processing times compared to the traditional SOM. In some cases, the time consumption of the Distributed GSOM was 1% of the time consumption of the traditional SOM, which quantifies to a 10,000% increase in efficiency. The efficiency can be increased further with more parallel processes.

With the high levels of performance, the Distributed GSOM does not significantly compromise the clustering accuracy. The reduction in accuracy is minor across different datasets, and in some instances, the Distributed GSOM delivers higher levels of accuracy. In addition, the visualisation advantages of the SOM are preserved in Sammon's projection. Furthermore, Sammon's projection visualises the inter-node distances in a continuous space which integrates the function of the U-Matrix into the projection. The continuous space visualisation of the Distributed GSOM is more useful for exploratory data analysis.

Results show that difference between the clustering accuracy of random and class based partitioning is minor. Therefore, in situations where class information is unavailable, the Distributed GSOM would perform well with random partitioning.

The outcomes of the experiments described in this chapter are as follows.

1. A novel distributed memory algorithm has been proposed with data parallelism to process large volumes of data with SOM techniques efficiently.

2. The data partitioning technique has minimal impact on the quality of the output.

3. A new measure was developed to measure the redundancy between two SOMs called the *redundancy index*.

4. Two redundancy reduction algorithms were proposed, for hit neurons and non-hit neurons.

5. The efficiency of the new algorithms was evaluated by comparing the total time consumption against the serial GSOM. The time consumption of the Distributed GSOM was 1% of that of the serial GSOM in the best case.

6. The partitioning method does not have a significant impact on the accuracy for most datasets.

7. Clusering accuracy comparison of the Distributed GSOM and the traditional GSOM shows the Distributed GSOM outperforms the traditional GSOM for some datasets while the loss of accuracy is minor in the others.

The next chapter discusses the implications of using SOMs and GSOMs as the learning technique; it highlights the advantage of a dynamically structured SOM, introduces a new redundancy reduction method and proposes a new model for incremental data integration to the Distributed GSOM.

Chapter 4

A Deeper Look into the Distributed GSOM

This chapter presents a detailed analysis of the Distributed GSOM algorithm. The main focus of the discussion is to compare, contrast and evaluate the SOM and the GSOM as the learning engine which is then used as the base to propose a new redundancy reduction method, and to parent a novel incremental model integration model for the Distributed GSOM algorithm.

The distributed algorithm proposed in the previous chapter can use either the SOM or the GSOM algorithm as the learning technique. The implications of using the SOM and te GSOM as the learning technique is discussed briefly in Chapter 3. This chapter discusses the impact of the SOM and the GSOM in detail and aims to demonstrate that the GSOM adds more value to the analysis process.

The new redundancy reduction algorithm proposed in this chapter improves the scalability of the Distributed GSOM algorithm. The new algorithm is faster than the redundancy reduction algorithm proposed in 3.4. A detailed analysis of the efficiency and the accuracy of the two redundancy reduction algorithms is presented in the following sections.

This chapter also presents a new model for incrementally integrating data into the Distributed GSOM, which enhances the applicability of the algorithm in the domain of exploratory data analysis. The new model continuously integrates new data into an existing network in an efficient manner. This adds a powerful feature to SOM based exploratory data analysis since new maps have to be generated only for the new data. The effectiveness of the model is demonstrated using a two dimensional dataset.

The research objectives addressed in this chapter are listed below.

1. To compare, contrast and evaluate the SOM and the GSOM as the learning technique for distributed learning, demonstrating the advantage of the GSOM.

2. To develop a new redundancy reduction method by pruning across the partition networks and to compare it with the redundancy reduction method detailed in 3.4.

3. To evaluate and compare the efficiency and the accuracy of two redundancy reduction methods.

4. To develop and demonstrate a model for incrementally integrating new data into the Distributed GSOM efficiently, thus extending the exploratory analytical value of SOMs in general.

4.1 SOM vs GSOM for exploratory data analysis

The Distributed GSOM algorithm has the ability to use either the SOM or the GSOM as the learning technique in the training phase. Although both the SOM and the GSOM use the same learning principle, the structure and the training process have significant differences, as discussed in 2.2 and 2.3. In this section, the SOM and the GSOM are evaluated for their suitability for exploratory data analysis using the distributed approach proposed in Chapter 3. The two methods are compared for efficiency and accuracy using WBC, SMH and CoverType datasets.

4.1.1 SOM for data exploration

The most common lattice structure used for the SOM is a rectangular lattice. The SOM is initialised by defining the dimensions of the map, namely the width and the height of the network. Since the ultimate objective of the SOM is to create a low dimensional representation of the input dataset, the shape (the width to height ratio) of the map should ideally reflect the shape of the dataset. If the shape of the dataset is known, it is possible to specify a shape for the SOM that fits the dataset. However, for most exploratory data analysis applications, the shape information is unknown.

In order to demonstrate the effects caused by the rigidity of the SOM structure, experiments were conducted using an artificial two dimensional dataset with different shapes.

The experiments transformed a square shaped dataset with 1,000 two dimensional vectors incrementally into a rectangular shaped dataset. For each transformed instance, an SOM was trained with the same specification, a 13 × 13 square shaped SOM with a learning rate of 0.2 trained for 1,000 iterations. The number of neurons in each SOM, N, was determined by equation 3.2 in Chapter 3.

$$N = 5 \times \sqrt{1000}$$

$$N \approx 159 N \approx 13 \times 13$$

Figure 4.1 shows the outcome of the experiment. It can be observed that when the shape of the dataset closely resembles to the square shape of the SOM, the network has low levels of distortions as in the case of (a), (b). However, in (c), the structure of the SOM is starting to compress and distortions start to appear. The distortions further increase in (d) and (e) which also creates bias towards the vectors at the centre of the dataset. The neuron density at the centre of the map is higher than the neuron density at the borders. This causes over representation of vectors at the centre and under representation at the borders.

In addition, for all the experiments, the border neurons are adapted towards the centre, which creates under representation of the vectors at the borders. As a result, input vectors at the boundary of the dataset create higher quantisation errors. These effects are further detailed in Kohonen (2001).

In data exploration, where shape information is mostly unavailable, the possibility of distortions is high for the SOM. In the next section, the same dataset is processed using the GSOM and compared with the SOM.

4.1.2 GSOM for data exploration

Section 2.3 describes the GSOM algorithm in detail. In summary, the GSOM starts with four neurons and creates additional neurons in order to accommodate the entire input dataset. New neurons are created only if an existing neuron accumulates a quantisation error greater than the growth threshold. Input vectors which generated the quantisation error are then distributed among the newly created neurons. Therefore, the GSOM has

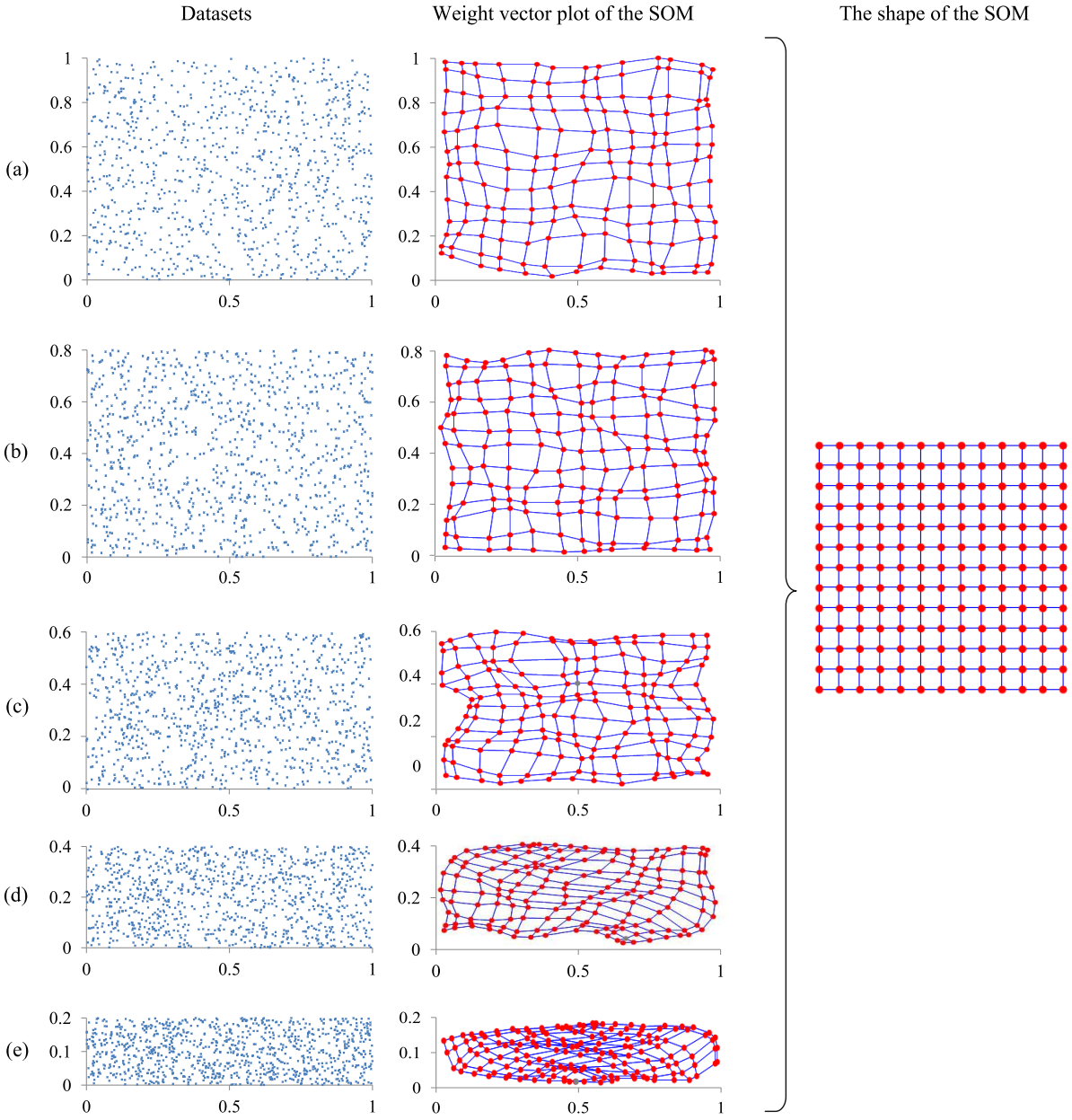


Figure 4.1: How the SOM accommodates differently shaped datasets

the ability to adapt the structure of its map to match the input dataset. Figure 4.2 shows the results using a GSOM for the same dataset used in the previous section.

Figure 4.2 indicates that the GSOM creates an even distribution of neurons compared to the SOM. The main reason for the even distribution of the neurons can be attributed to the growth technique of the GSOM. If the input vectors are densely packed, the neurons in the dense area would generate higher quantisation errors. As a result, such neurons would exceed the growth threshold faster and create more neurons in order to represent the dense set of vectors.

Another difference of the GSOM has to do with how vectors at the dataset boundary are represented in the map. Due to the dynamic nature of the structure of the GSOM, boundary neurons are able to distance themselves from the neurons at the centre of the map by growing outwards from the centre. With a higher distance from the centre, boundary neurons are able to resist neighbourhood adaptation. When compared with the output of the SOM, it is evident that the GSOM map spreads closer to the border which results in better representation of the data.

The GSOM neuron location plot shows that the GSOM's shape is a rotated representation of the actual dataset. Rotation is caused by the random initialisation of the neurons. The maps show that the actual shapes of the GSOMs are similar to the shape of the dataset. Th ability of the GSOM to take the shape of the dataset creates an additional visualisation advantage for the analysts by showing the structure of the dataset.

Table 4.1 shows the number of neurons and the total quantisation error values for the SOM and the GSOM for the same datasets described above. For each SOM, a learning rate of 0.2 was used and over 1,000 iterations were processed on the dataset.

The GSOMs were trained using a spread factor of 0.15, which generated 171 neurons for dataset (a). Fifty growing iterations and 950 smoothing iterations were used for the GSOM in order to create similar circumstances to that of the SOM. Both algorithms employed a learning rate of 0.2.

It can be observed that when the shape of the dataset matches the shape of the SOM, the total quantisation error is slightly lower than that of the GSOM in the case of dataset (a). However, the GSOM tends to generate a lower quantisation error as the shape of the dataset increasingly becomes rectangular for datasets (b), (c) and (d). In the case of dataset (e), the SOM creates a lower quantisation error due to over representation.

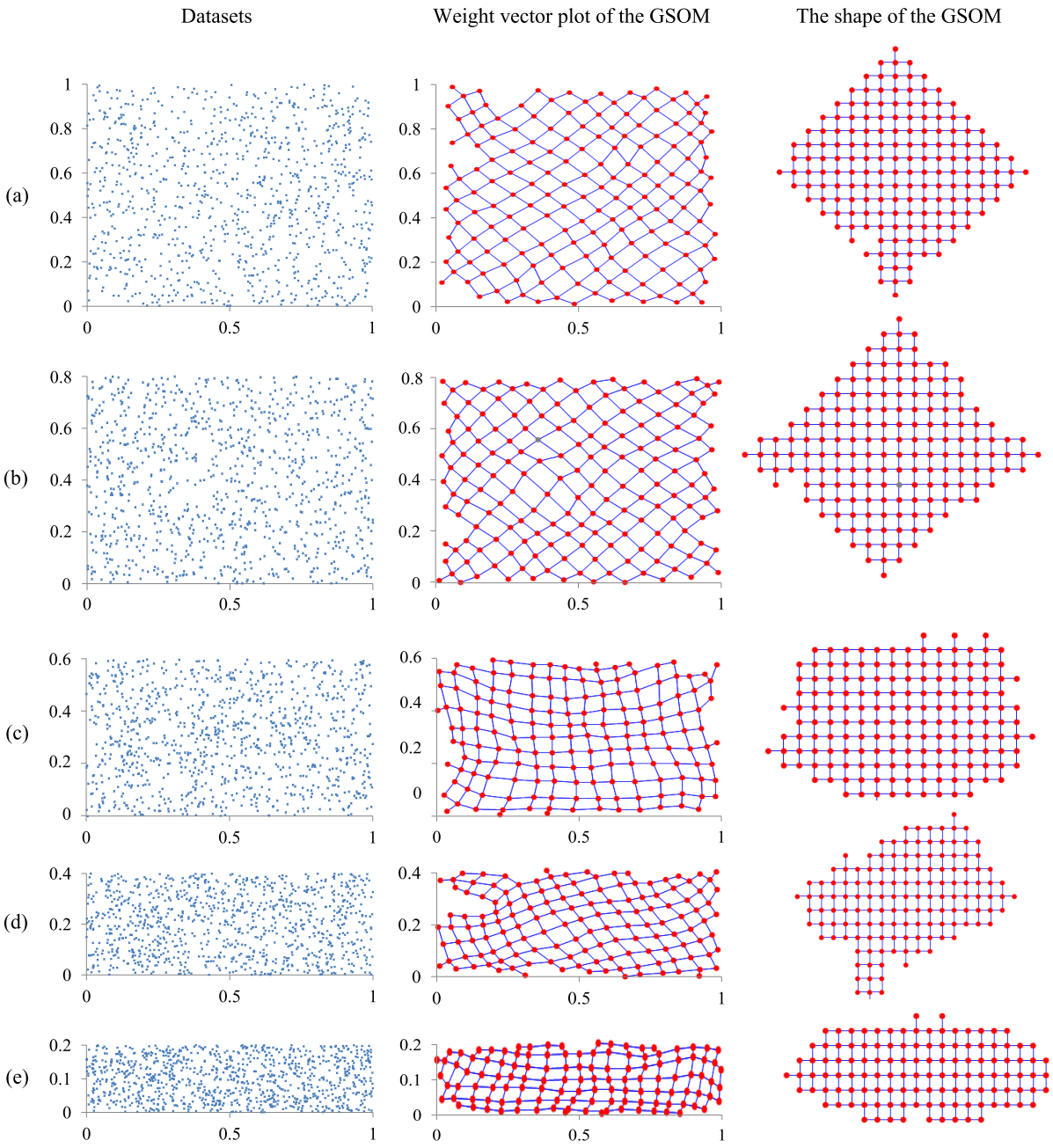


Figure 4.2: How the GSOM accommodates differently shaped datasets

Table 4.1: Neuron count and quantisation error (QE) values for the rectangular dataset for the SOM and the GSOM

	SOM		GSOM	
	Neuron count	QE	Neuron count	QE
Dataset (a)	169	34.2402	171	34.2709
Dataset (b)	169	30.4347	167	28.8775
Dataset (c)	169	27.3300	163	27.3113
Dataset (d)	169	22.9986	140	22.9528
Dataset (e)	169	16.0397	122	16.1538

However, the performance advantage of the GSOM with approximately 30% fewer neurons outweighs the 0.7% increase in the total quantisation error. It is also evident that, as the span of the dataset decreases, the GSOM creates fewer neurons which is an advantage when shape information is unavailable.

The next section describes the implications of using the SOM and the GSOM as the learning technique for the distributed algorithm proposed in this thesis.

4.2 Comparison of the SOM and the GSOM for data exploration using the distributed algorithm

The limitations of the SOM in exploratory data analysis is evident when the shape of the dataset does not match the shape of the SOM. In addition, the SOM tends to create inconsistencies in the representation of the input by over representing low density regions and under representing the high density regions, as given in Haykin (1994) and Su and Chang (2000). However, the GSOM is able to overcome this limitation by dynamically adapting the network to match the spread of the neurons to suit the density of the input.

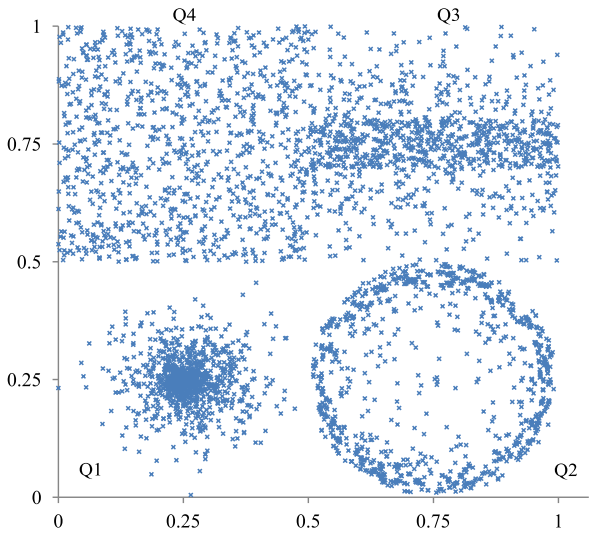


Figure 4.3: The dataset used to compare the SOM and the GSOM

In order to demonstrate the effects of the SOM and the GSOM in the context of the distributed SOM approach discussed in Chapter 3, a varying density dataset was used. The two dimensional artificial dataset used for the experiments is shown in Figure 4.3. The dataset is divided into four quarters which would form the partitions. Each quarter

Table 4.2: Neuron count and quantisation error (QE) values for the four data partitions for the SOM and the GSOM. Additionally, the merged result is also shown.

	SOM		GSOM	
	Neuron count	QE	Neuron count	QE
Q1	169	9.3549	145	8.6814
Q2	169	12.05	130	12.0436
Q3	169	15.447	174	13.0595
Q4	169	17.4016	167	14.6668
Merged	676	54.0196	616	40.9787

contained 1000 two dimensional vectors. Q1 is a Gaussian distribution where the vector density is very high at the centre. Records in Q2 are arranged in an inverse Gaussian distribution where the vector density is higher closer to the borders. Q3 contains a square dataset having a higher density along the line, $y = 0.75$. The records in Q4 are evenly distributed in $x\epsilon[0, 0.5]$ and $y\epsilon[0.5, 0.75]$.

The experiment was conducted as two exercises where the distributed algorithm was executed using SOMs and GSOMs separately. The steps in the experiment included creating partitions for the four quarters, training an SOM and a GSOM on each partition, redundancy reduction and topographic arrangement of the neurons using Sammon's projection. Since each partition contained 1000 records, the size of the SOMs was set to 13 $\times$ 13.

The GSOMs were trained with a spread factor of 0.15 which created similar levels of detail in the GSOMs for the dataset with the even distribution, Q4. Table 4.2 shows the outcomes of the SOM and GSOM for the partitions. It can be observed that the GSOM creates a lower total quantisation error for all the partitions. The GSOM has created fewer total neurons, which has resulted in less time consumption for Sammon's projection compared to the SOM. The total quantisation error for the merged network is lower than the summation of the errors of the partition networks due to partition boundary nodes being assigned to neurons from other partition networks.

The visualisation of the SOM partition networks and the resulting Sammon's projection are shown in Figure 4.4. It can be observed that the visualisation of the irregular shapes are distorted by the SOMs for Q1, Q2 and Q3 datasets. The distortions are carried forward to Sammon's projection resulting in a higher quantisation error.

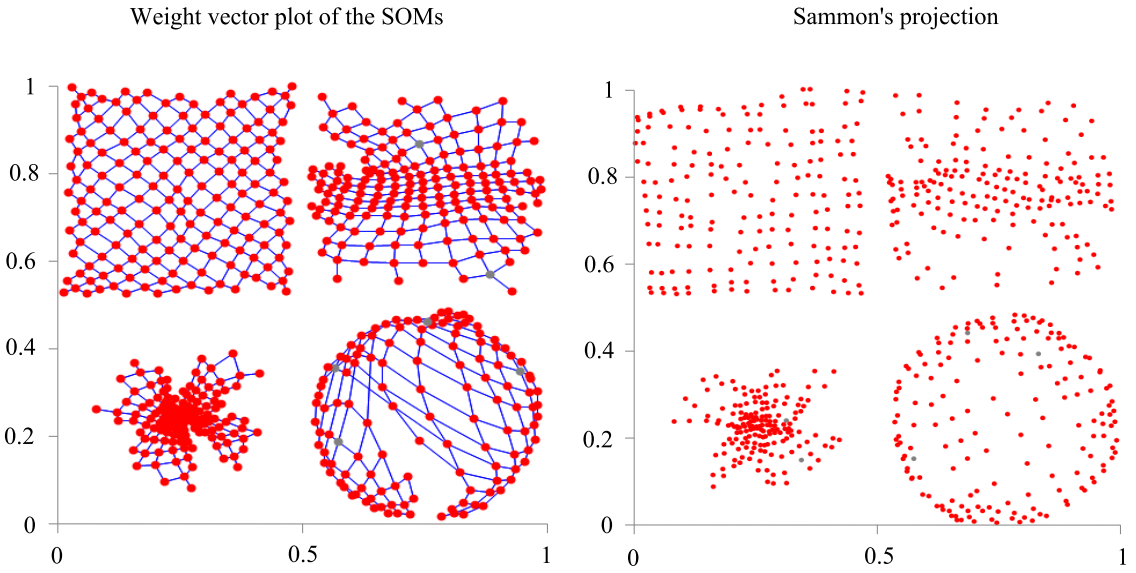


Figure 4.4: Partition networks of the Distributed SOM and the final output

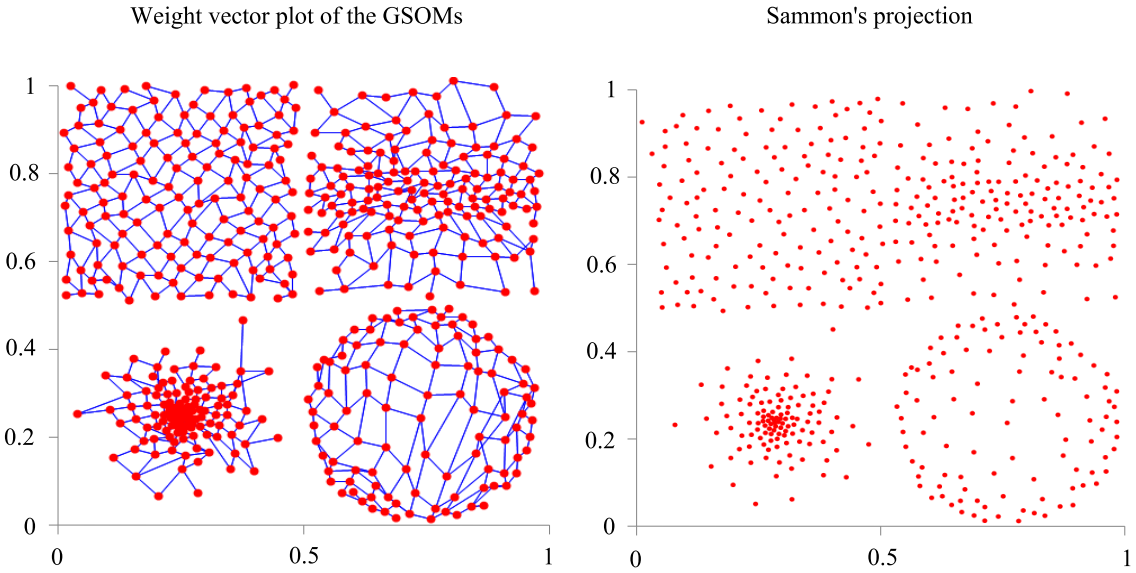


Figure 4.5: Partition networks of the Distributed GSOM and the final output

Figure 4.5 shows the output created using GSOMs for the distributed algorithm. The GSOM output more closely resembles the actual datasets for all four partitions. The output of Sammon's projection has fewer distortions and depicts a structure similar to the actual dataset. Due to the lower levels of distortion, the GSOM output creates a significantly lower quantisation error compared to the SOM.

The total quantisation error of an SOM or a GSOM indicates the degree of mismatch between the input dataset and the weight vectors of the neurons. Observing the statistics shown in Table 4.2, the GSOM creates a 24% lower total quantisation error than the SOM.

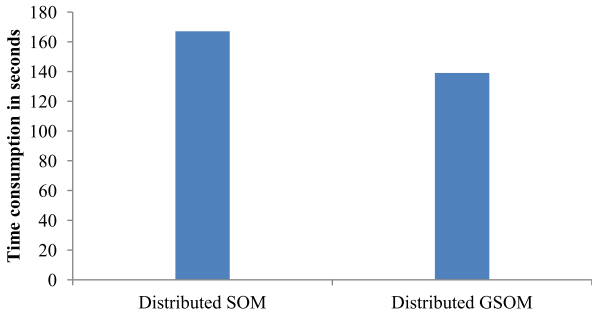


Figure 4.6: Time consumption of Sammon's projection for the Distributed SOM and the Distributed GSOM

Therefore, the GSOM creates a better match for the input dataset than the SOM. Hence, the GSOM creates a more faithful representation of the input dataset.

Furthermore, the GSOM creates fewer neurons than the SOM which in turn reduces the size of the input presented to Sammon's projection. This results in faster performance of the overall algorithm when GSOMs are used as the learning technique. Figure 4.6 shows the time consumption of Sammon's projection process for the Distributed SOM and the Distributed GSOM. It can be observed that the time consumption of Sammon's projection of the Distributed GSOM is 16.8% less than that of the Distributed SOM which improves the efficiency of the overall process.

In real-world exploratory data analysis applications, datasets seldom have regular shapes and even distributions. Specifying the shape of the SOM to match the dataset may be impossible for large scale analysis tasks. Experiments show that GSOMs can better accommodate irregular, varying density datasets over the SOM. Hence, the Distributed GSOM will create a more faithful representation of the dataset compared to the Distributed SOM. In addition, the GSOM creates fewer neurons than the SOM which results in lower time consumption for Sammon's projection. Therefore, it can be concluded that the GSOM is the most appropriate learning technique for the distributed algorithm proposed in Chapter 3 and appropriately named, the *Distributed GSOM*.

4.3 Redundancy reduction

Chapter 3 discusses how neurons in the partition networks trained in parallel may represent the same set of vectors if vectors within the same class are distributed across different

partitions. Such neurons are considered redundant and would be removed by the redundancy reduction process. The approach suggested in 3.4 identifies redundant neurons by comparing the quantisation errors created for subsets of input vectors. If a hit neuron is identified as redundant, its hit vectors are assigned to the BMU for the redundant neuron. As a result, as the redundancy reduction process progresses, the number of vectors assigned to the preserved neurons increases.

With the accumulation of input vectors, fewer redundant neurons are identified as the redundancy reduction process progresses. This effect is caused by the neurons becoming increasingly specialised on the input vectors mapped onto the neurons. With a higher number of partitions, the rate of accumulation increases since overall redundancy increases as the number of partitions increase. As a consequence of the accumulation, the number of neurons presented to Sammon's projection increases, which, in turn, increases the overall time consumption of the Distributed GSOM. This effect is evident from Table 3.1 which shows that the redundant neuron percentage is similar across different partition configurations. However, as the number of partitions increase, the redundant neuron percentage should increase with random partitioning.

Table 3.2 indicates that the number of neurons in the Distributed GSOM output increases with the number of partitions. As a result, the time consumption of Sammon's projection process increases as the number of partitions increases. Although this increase in time consumption is compensated by the reduction of time consumption of the partition network training phase, Figure 3.4 shows that the time consumption of the overall process could reach its minimum value when the curves of the partition network time consumption and Sammon's projection time consumption cross each other. The scalability of the algorithm would thus be limited. This effect is further discussed in Ganegedara and Alahakoon (2012) and Matharage et al. (2013).

4.3.1 A new redundancy reduction method

As a means of avoiding input vector accumulation of the preserved neurons, this section discusses using only the original input vectors of a neuron for redundancy determination. Using the same notations given in 4.3 above, the algorithm for redundant hit neuron reduction, using the original hit items is given in Algorithm 3.

Algorithm 3 Redundant hit neuron reduction algorithm using only the original hit items

Let P = All partitioned networks
$d = 0, n = 0$
for all partition $P_i \in P$ **do**
  for all hit neuron $n_{i,j} \in P_i$ **do**
    for all partition $P_k \in P$ where $i \neq k$ **do**
      $n_{k,l} = \text{GetBMU}(w_{i,j}, k)$
      $E_{k,l}^{i,j} = \sum_{x=0}^{N_{i,j}} |w_{k,l} - I_{i,j}[x]|$
      $E_{i,j}^{k,l} = \sum_{x=0}^{N_{k,l}} |w_{i,j} - I_{k,l}[x]|$
      if $E_{i,j} > E_{k,l}^{i,j}$ **then**
        remove $n_{i,j}$
        $d = d + |w_{i,j} - w_{k,l}|$
        $n = n + 1$
      else
        if $E_{k,l} > E_{i,j}^{k,l}$ **then**
          remove $n_{k,l}$
          $d = d + |w_{k,l} - w_{i,j}|$
          $n = n + 1$
        else
          $E_C = E_{i,j} + E_{k,l}$
          $E_1 = E_{i,j} + E_{i,j}^{k,l}$
          $E_2 = E_{k,l} + E_{k,l}^{i,j}$
          if $E_C > E_1$ & $E_1 < E_2$ **then**
            remove $n_{k,l}$
            $d = d + |w_{i,j} - w_{k,l}|$
            $n = n + 1$
          else if $E_C > E_2$ & $E_2 < E_1$ **then**
            remove $n_{i,j}$
            $d = d + |w_{i,j} - w_{k,l}|$
            $n = n + 1$
          end if
        end if
      end if
    end for
  end for
end for
return $\frac{e^{SF} \times d}{D \times n}$

In order to evaluate the features of the new method, the SMH and CoverType datasets were used to compare the efficiency and accuracy of the output. Through the remainder of this chapter, the redundancy reduction method which examined accumulated hit items introduced in 3.4 will be referred to as *redundancy reduction method 1* and Algorithm 3 will be referred to as *redundancy reduction method 2*.

4.3.2 Experiments and results

The two redundancy reduction methods were compared using WBC, SMH and CoverType datasets, which were discussed in Chapter 3. Since the main concerns of the two methods are on scalability and time consumption, the SMH and CoverType datasets were used to compare efficiency. All three datasets were used to compare accuracy. The following sections discuss the redundancy reduction statistics, efficiency and the accuracy of the two methods.

Redundancy reduction statistics

Table 4.3 shows the redundancy reduction statistics for the SMH dataset. It can be observed that, as the number of partitions increases, the percentage of redundant neurons identified by method 2 increases at a faster rate than in method 1. Therefore, as the number of partitions increases, the number of neurons preserved during the redundancy reduction process increases for method 1. On the other hand, as the number of partitions increases, due to the higher percentage of redundant neuron identification by method 2, the total number of preserved neurons decreases. Consequently, method 1 creates a lower redundancy index (RI) compared to method 2.

Table 4.4 shows that the two redundancy reduction methods create a similar effect for the CoverType dataset. Since the CoverType dataset is a dense dataset, the proportion of redundant neurons is high. Although the identified redundant neuron percentage increases with the number of partitions in method 1, method 2 has a higher rate of increase.

As the number of partitions increase, the average number of hit items assigned per neuron decrease. With a small number of hit items to compare for redundancy, the chance of finding a better replacement for a particular neuron is high. As a result, in method 2, as the number of partitions increases, the number of redundant neurons also increases. Both random partitioning and class based partitioning have higher levels of redundant

Table 4.3: Statistics for the two redundancy reduction methods for the SMH dataset

Partitions	Neuron count	Redundant neurons	Preserved neurons no.	Redundancy index
		Method 1		
4 random	295	44.9%	167	0.0074
8 random	339	44.8%	182	0.0074
16 random	414	45.7%	225	0.0075
Class	303	5.5%	287	0.0122
		Method 2		
4 random	295	61.9%	112	0.0074
8 random	339	76.6%	79	0.0075
16 random	414	85.6%	60	0.0076
Class	303	21.0%	240	0.0104

Table 4.4: Statistics for the two redundancy reduction methods for the CoverType dataset

Partitions	Neuron count	Redundant neurons	Preserved neurons no.	Redundancy index
		Method 1		
8 random	14,421	26.3%	10,628	0.0110
16 random	19,425	38.1%	12,016	0.0126
32 random	26,450	52.8%	12,496	0.0154
Class	15,028	41.3%	8,820	0.0149
		Method 2		
8 random	14,421	40.2%	8621	0.0101
16 random	19,425	56.1%	8525	0.0116
32 random	26,450	72.5%	7364	0.0143
Class	15,028	62.9%	5578	0.0158

neuron identification percentages for method 2 which is consistent for both datasets. As a result, both redundancy reduction methods perform consistently for both random and class based partitioning.

Efficiency comparison of the two redundancy reduction methods

Since the redundancy reduction method 2 creates fewer neurons as input to Sammon's projection, the time consumption of Sammon's projection is less for method 2. As a

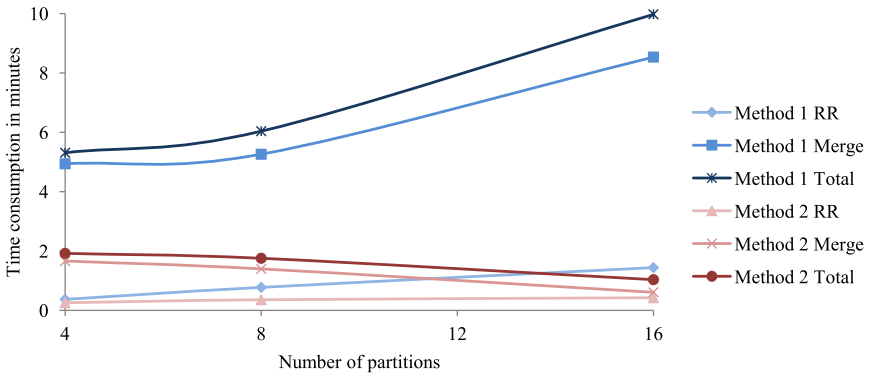


Figure 4.7: Time consumption of the redundancy reduction (RR) process, Sammon's projection and the combined time for the SMH dataset for the two redundancy reduction methods

result, method 2 delivers higher efficiency levels and higher scalability for the Distributed GSOM algorithm.

Figure 4.7 shows the time consumption of the Distributed GSOM for the SMH dataset. It can be observed that the total merging time increases for method 1 and decreases for method 2. This is caused by the reduction of the total number of neurons presented to Sammon's projection as the number of partitions increases. The time consumption of Sammon's projection dominates the time consumption of the merging process and Sammon's projection consumes less time when the number of inputs is less. A similar observation can be made for the CoverType dataset as shown in Figure 4.8.

For both datasets, the total time consumption for the redundancy reduction and merging process combined increases for method 1. The increase in time consumption would limit the scalability of the Distributed GSOM when the increase in time consumption from the merging process outweighs the reduction in time consumption caused by splitting the dataset. Method 2, on the other hand, shows a decreasing trend for the total merging time for both datasets. Therefore, method 2 would have higher levels of scalability compared to method 1.

Accuracy comparison of the two redundancy reduction methods

The F-measure was used in order to investigate the impact of the two redundancy reduction methods on accuracy. All three datasets were used for the accuracy evaluation. Tables 4.5, 4.6 and 4.7 show the accuracy results for method 1 (M1) and method 2 (M2). Due to its taking into account only the original hit items for error calculation, method 2 may

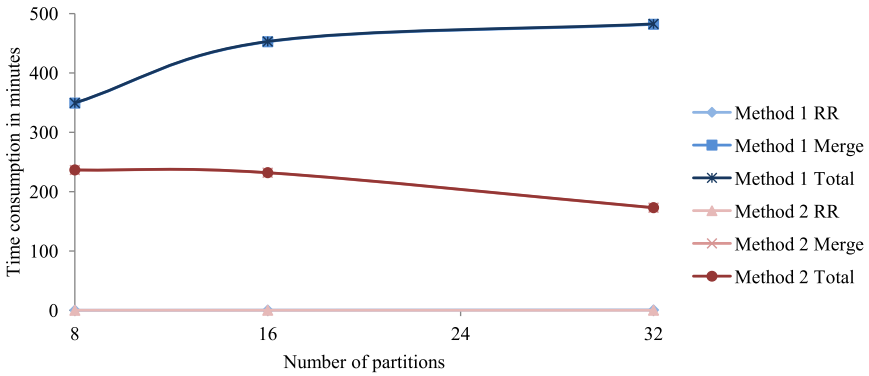


Figure 4.8: Time consumption of the redundancy reduction (RR) process, Sammon's projection and the combined time for the CoverType dataset for the two redundancy reduction methods

Table 4.5: F-measure and the standard deviation (σ) of the F-measure for the WBC dataset for benign (B) and malignant (M) classes showing results for random (R) and class based partitioning

Details	F- Measure			σ (F-Measure)		
	M	B	Avg.	M	B	Avg.
GSOM	0.9171	0.9323	0.9256	0.0096	0.0079	0.0084
R, 2P, M1	0.9204	0.9358	0.9290	0.0058	0.0085	0.0073
R, 2P, M2	0.9215	0.9369	0.9301	0.0056	0.0089	0.0075
R, 4P, M1	0.9134	0.9294	0.9222	0.0199	0.0166	0.0178
R, 4P, M2	0.9075	0.9269	0.9185	0.0235	0.0199	0.0212
Class, M1	0.9166	0.9433	0.9321	0.0054	0.0064	0.0056
Class, M2	0.9166	0.9433	0.9321	0.0054	0.0064	0.0056

remove neurons which could be preserved if accumulated hit items are used. If such neurons existed at cluster boundaries, taking account of the accumulated hit items would result in better clustering accuracy. Redundancy reduction method 1 which incorporates accumulated hit items therefore produces higher accuracy levels overall.

Table 4.5 indicates the F-measure and the standard deviation of the F-measure for the WBC dataset. The results indicate that the differences between method 1 and method 2 are minor where method 2 has higher levels of accuracy for random partitioning with two partitions. Method 1 has higher levels of accuracy for random partitioning with four partitions and accuracy is the same for both methods for class based partitioning. Method 2 shows marginally higher levels of standard deviation, which could be expected as using only the original hit items introduces higher levels of variability.

Table 4.6: F-measure and the standard deviation (σ) of the F-measure for the SMH dataset for news (N), world (W), sport (S) and entertainment (E) classes showing results for random (R) and class based partitioning

Details	F-Measure					σ (F-Measure)				
	N	W	S	E	Avg.	N	W	S	E	Avg.
GSOM	0.808	0.704	0.647	0.924	0.080	0.006	0.012	0.026	0.003	0.005
R, 4P, M1	0.798	0.700	0.612	0.938	0.795	0.029	0.027	0.027	0.005	0.010
R, 4P, M2	0.785	0.688	0.596	0.932	0.788	0.013	0.000	0.000	0.000	0.000
R, 8P, M1	0.784	0.697	0.564	0.930	0.784	0.009	0.028	0.049	0.006	0.011
R, 8P, M2	0.774	0.672	0.557	0.923	0.774	0.014	0.001	0.001	0.000	0.000
R, 16P, M1	0.766	0.690	0.536	0.923	0.768	0.009	0.014	0.032	0.007	0.003
R, 16P, M2	0.742	0.658	0.509	0.907	0.745	0.010	0.001	0.001	0.000	0.000
Class, M1	0.893	0.853	0.821	0.971	0.898	0.002	0.005	0.003	0.002	0.002
Class, M2	0.886	0.850	0.791	0.973	0.891	0.005	0.000	0.000	0.000	0.000

Table 4.6 shows the F-measure and the standard deviation of the F-measure for the two redundancy reduction methods for the SMH dataset. It can be observed that method 2 results in lower clustering accuracy levels than method 1. As the number of vectors considered for redundancy neuron determination is lower in method 2, the neurons at cluster boundaries may get removed. As a result, vectors at the cluster boundaries may get assigned to incorrect clusters. In addition, method 2 shows lower variability caused by the sparsity of the dataset.

Table 4.7 shows similar results for the CoverType dataset where method 2 has moderately lower levels of accuracy with similar levels of variability.

4.3.3 Applications of the two redundancy reduction methods

The exploratory data analysis process usually involves several stages of analysis. At the initial stages, a trial and error approach may be used to determine the best attribute and record configurations. Based on the outcomes of the initial analysis, subsequent methods of analysis would be finetuned to identify specific patterns.

For large scale data, the initial trial and error approach may consume excessive amounts of time which would limit the number of trials. In such situations, method 2 can be used since it offers faster performance. As the initial stages are only used for the guidance of the subsequent analysis, the loss in accuracy is affordable. The faster performance of method

Table 4.7: F measure and the standard deviation (σ) of the F measure for the CoverType dataset for the most frequent two classes (class 1 and class 2) showing results for random (R) and class based partitioning

Details	F measure			σ(F measure)		
	Class 1	Class 2	Avg.	Class 1	Class 2	Avg.
GSOM	0.6554	0.7782	0.7278	0.0104	0.0102	0.0072
R, 8P, M1	0.7548	0.8243	0.7953	0.0015	0.0008	0.0009
R, 8P, M2	0.7413	0.8152	0.7844	0.0027	0.0005	0.0011
R, 16P, M1	0.7519	0.8224	0.7930	0.0019	0.0018	0.0018
R, 16P, M2	0.7300	0.8082	0.7757	0.0041	0.0011	0.0016
R, 32P, M1	0.7412	0.8175	0.7860	0.0012	0.0013	0.0011
R, 32P, M2	0.7138	0.7989	0.7638	0.0026	0.0012	0.0007
Class, M1	0.7492	0.8120	0.7851	0.0019	0.0005	0.0006
Class, M2	0.7312	0.8017	0.7718	0.0008	0.0019	0.0013

2 would allow analysts to perform more trials in order to determine the most suitable analysis configuration.

Once the details of the analysis process are determined using trial and error, method 1 can be used to perform a more accurate analysis. As the detailed analysis is usually performed only once, method 1's higher accuracy in clustering would compensate its slower performance.

4.4 Dynamic data integration into the Distributed GSOM

Depending on the problem under investigation, the results from any exploratory data analysis may have only short-term value; this is because the source data may become out-of-date. With time, historical data may increasingly become irrelevant as new data becomes available. In order to maintain the effectiveness of the analysis, the analysis process would have to be periodically repeated on the most up-to-date data. Therefore, exploratory data analysis can be considered a continuous process.

A key feature of the traditional SOM algorithm is the need to re-train the entire network when new data becomes available. The re-training process has to be completed efficiently if new data arrive frequently. However, for large datasets, the time consumption of re-training the SOM may be impractically excessive. Furao et al. (2007),

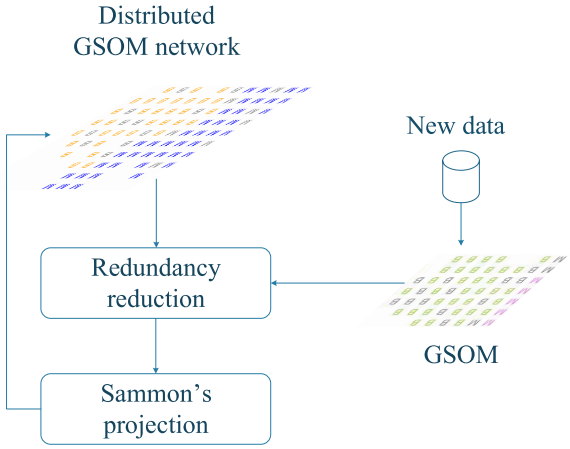


Figure 4.9: An incremental data integration for the Distributed GSOM

Nurnberger and Detyniecki (2006) and Prudent and Ennaji (2005) have proposed incremental SOM algorithms which can integrate new data into the existing network efficiently. The algorithm proposed in Furao et al. (2007) uses a two layer approach which approximately doubles the time consumption of the SOM which limits its applicability on large datasets. The algorithms proposed in Nurnberger and Detyniecki (2006) and Prudent and Ennaji (2005) use dynamic structures to create new nodes dynamically where the new position calculation process is significantly time consuming.

The Distributed GSOM architecture possesses inherent support for incremental data integration. The components of the Distributed GSOM algorithm can be reused for integrating new data into an existing network. This can be performed by training a GSOM on the new data off-line and integrating the neurons in the newly trained GSOM into the existing network by executing a redundancy reduction phase, followed by Sammon's projection. The architecture of this model is shown in Figure 4.9.

The model was evaluated using the dataset given in Figure 4.3 by presenting the different quarters of the dataset incrementally. Two sets of experiments were conducted using different orderings of the presentation of datasets.

Figure 4.10 shows the first set of experiments where the datasets are presented to the Distributed GSOM in the order, Q1, Q2, Q3 and Q4. It can be observed that the output of the Distributed GSOM algorithm creates the correct intermediate output for each increment. Although the output contains minor distortions in the case of three partitions in step (b), the three clusters within the input are accurately identified. The distortions

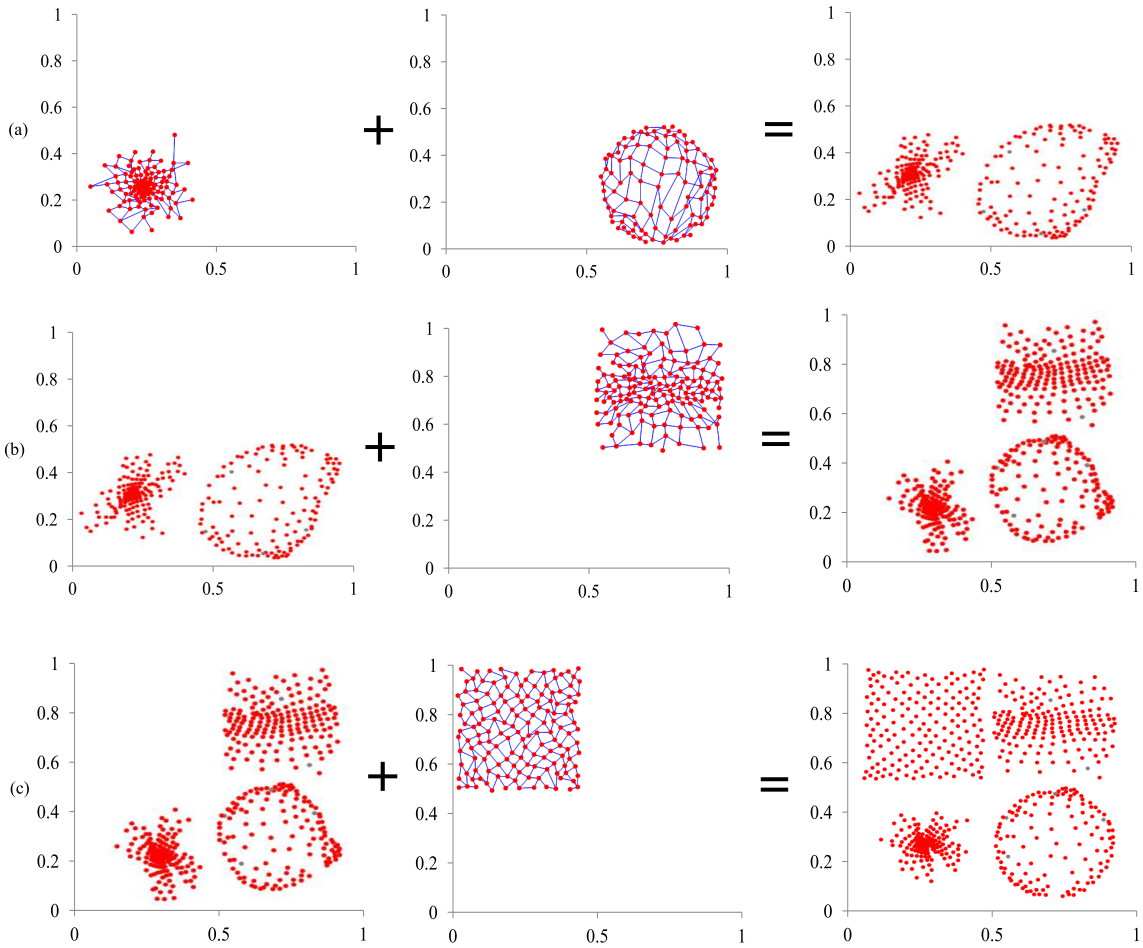


Figure 4.10: Incremental data integration experiment 1

in step (b) can be attributed to the skew of the datasets which contains only three quarters. As a result, Sammon's projection stretches the projection space to accommodate the dataset. As the dataset spans the entire input space in step (c), the output accurately represents the entire input dataset.

Figure 4.11 shows the second set of experiments run where the datasets were presented in the following order: Q4, Q2, Q3, Q1 order. The results indicate that with the skew of the datasets in steps (a) and (b), minor distortions are introduced into the output of the Distributed GSOM. The intermediate outputs are accurately created and the final outputs have a high degree of similarity.

The output of the incrementally trained networks can be compared with the output of the Distributed GSOM created by presenting the entire dataset simultaneously. The final outputs of the incremental integration exercises shown in step (c) in Figure 4.10 and Figure 4.11 can be compared to the Distributed GSOM output shown in Figure 4.5 where the entire dataset was presented simultaneously. The figures indicate that both

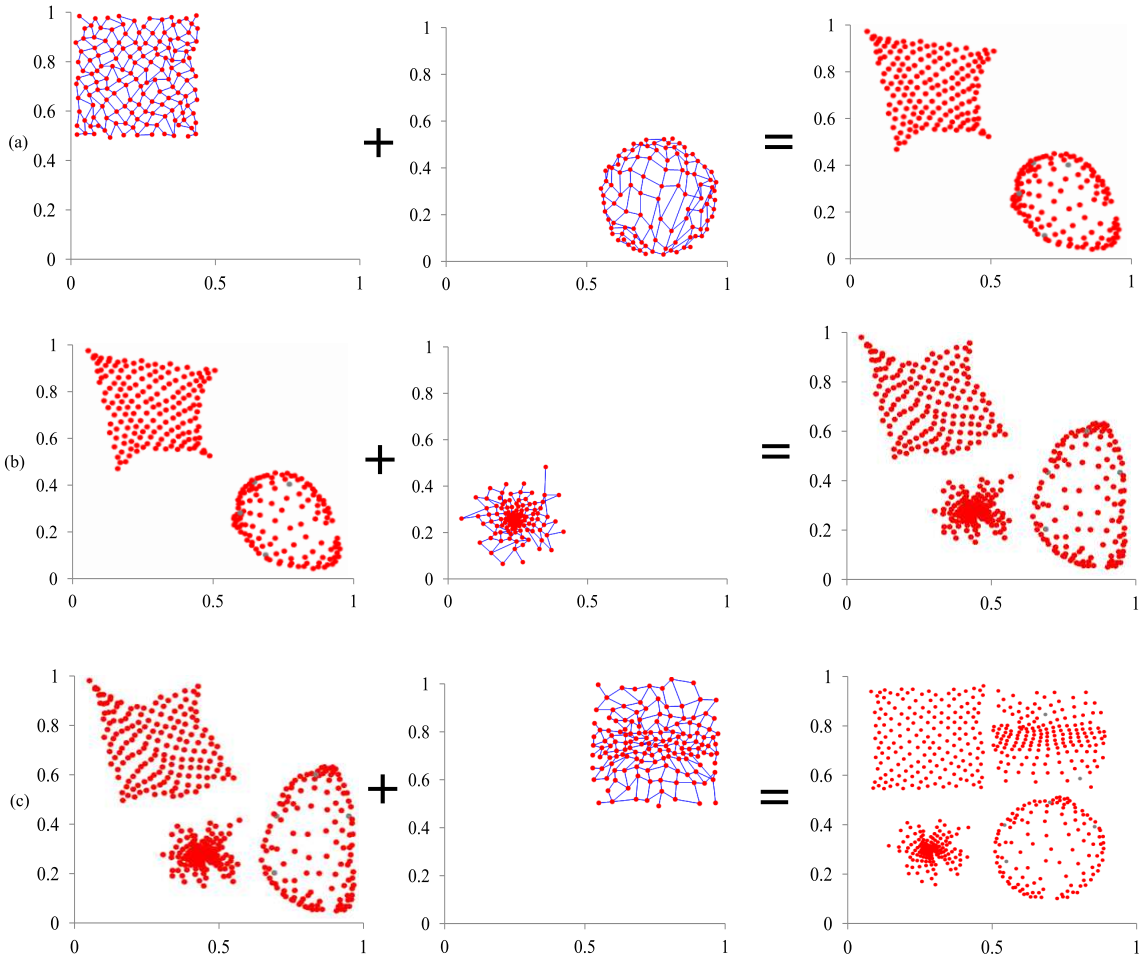


Figure 4.11: Incremental data integration experiment 2

the incremental model and the standard Distributed GSOM algorithm produce nearly identical outputs. Therefore, it can be concluded that the Distributed GSOM would create the same output when the data is presented incrementally or simultaneously.

It has to be noted that Sammon's projections shown in Figures 4.10 and 4.11 show the location plot of the projection. Plotting the weight vectors would reveal the same image as the individual GSOM plots in the images since Sammon's projection does not modify the weight vectors of the input. Therefore, the visualisation advantage of Sammon's projection is clearly demonstrated as it shows the clusters accurately within the data.

The results demonstrate that the Distributed GSOM can be used for integrating new data effectively into the network by positioning the new data accurately within the existing network. It can be assumed that the new data would have a lower level of skewness. If the skewness of the dataset is low, the number of distortions created in the Distributed GSOM will be minimal. As a result, the output of the Distributed GSOM would be similar to the output that would be created on the entire dataset in a single run. Thus the incremental

model prevents the need to retrain the entire network on new data. Furthermore, this approach would also facilitate the analysis of the new data separately by analysing the partitioned network (GSOM) created on the new data in isolation.

4.5 Discussion

In this chapter, three aspects of the Distributed GSOM algorithm were discussed in detail. The implications of using static and dynamic structured SOMs for data exploration were investigated, a new redundancy reduction method was introduced and an incremental data integration mode was proposed.

It was clear that the GSOM with a dynamic structure creates a lower total quantisation error with fewer neurons compared to the SOM with a static structure. The total quantisation error indicates the degree to which the weight vectors of the neurons are different from the actual input data vectors. Experiments indicate that the GSOM creates a lower total quantisation error on average, which demonstrates that the GSOM creates a better representation of the input space compared to the SOM. Furthermore, since the GSOM creates fewer average neurons, the efficiency of Sammon's projection is improved. As a result, it can be concluded that the GSOM is a better learning technique for the distributed algorithm.

The new redundancy reduction method (method 2) proposed in this chapter offer faster performance and higher levels of redundant neuron identification compared to the method proposed in Chapter 3. Since the new method takes into account only the original hit items of neurons for error calculation, as the volume of data increases, the redundancy reduction process scales well. This increases the overall scalability of the algorithm. The new redundancy reduction method achieves faster performance at the cost of accuracy. As a result, method 2 can be used for the initial stages of the exploratory data analysis process where high accuracy levels are not required and method 1 (the accumulated hit item based method) for advanced analysis.

Data analysis algorithms should incorporate new data into the existing analysis in order to maintain the relevance of the outcomes. An incremental data integration model is proposed in this chapter which reuses components of the Distributed GSOM algorithm to fuse new data into the existing network. Experimental results show that the Distributed

GSOM can effectively add new data into an existing network by maintaining cluster separation.

The outcomes of the chapter can be listed as follows:

1. The GSOM creates a better representation of the dataset compared to the SOM when the shape of the dataset is unknown. Therefore, the GSOM is considered to be the preferred learning technique for the distributed algorithm.

2. The new redundancy reduction method, which considers only the original hit items of neurons for redundancy determination, delivers faster performance at the expense of accuracy. The two redundancy reduction methods can be used at different stages of the exploratory data analysis process.

3. The Distributed GSOM can efficiently integrate new data into the existing network by training a GSOM on the new data off-line and integrating the newly trained GSOM. The results show that the output accurately represents the clusters within the data when the data is incrementally presented.

The next chapter discusses the details of the implementation of the Distributed GSOM algorithm on the Hadoop distributed computing framework.

Chapter 5

The Distributed GSOM on Hadoop

Hadoop is one of the leading cloud computing frameworks in existence developed by the Apache Foundation (White, 2012). Hadoop computing clusters offer massive amounts of computing power for data and compute-intensive applications. Although there have been some accounts of SOM implementations on Hadoop such as Goto et al. (2013) and Weichel (2010), it is believed that the work presented in this chapter is the first instance of a clear and detailed implementation of SOM technologies on Hadoop. Therefore, the implementation of the Distributed GSOM on Hadoop is believed to be a contribution towards the practical use of SOM technologies in current big data environments.

Hadoop applications follow a 'divide and conquer' execution model which is referred to as MapReduce, where multiple parallel map processes divide the work load amongst multiple processors and the intermediate results are combined using the reduce process. The Distributed GSOM algorithms follows a similar pattern where GSOMs are created on multiple data partitions which are then combined to form a single map. Therefore, the Distributed GSOM has an ideal architecture for MapReduce implementations. But, developing a Hadoop implementation for the Distributed GSOM would have to overcome a number of programming challenges. The research objectives of this chapter, which are listed below, are based on addressing these challenges.

1. To adapt the Distributed GSOM architecture to fit MapReduce.

2. To develop a technique for splitting multivariate data such that each GSOM receives a sufficient number of vectors to achieve convergence.

3. To develop cascading layers of MapReduce processes to integrate a MapReduce data partitioning process into the Distributed GSOM to improve its efficiency.

4. To minimise dataset access in the reduce process.

5. To employ combiners to improve the efficiency of the overall process.

6. To demonstrate the performance of the Hadoop implementation of the Distributed GSOM using benchmark datasets.

The following sections describe the Hadoop framework and the work done to achieve the research objectives.

5.1 MapReduce

One key barrier for the wide adaptation of distributed and parallel programming had been the complexity that has to be handled by the programmers. Parallel and distributed applications require interfacing with multiple computers and remotely managing, monitoring and synchronising processes running on a multitude of computers. The MapReduce programming model was developed by Google Inc. in order to create a programmer friendly computing environment that abstracts the complex communication and synchronisation aspects involved in parallel and distributed computing (Dean and Ghemawat, 2008).

The purpose of the MapReduce cluster is to provide a scalable computing environment for processing massive volumes of data. At Google, the MapReduce model is complemented by a distributed file system called the Google file system (GFS). The GFS combines all the storage elements within the cluster to create a massive, federated, distributed storage module capable of storing petabytes scale data as discussed in Ghemawat et al. (2003). Unlike early distributed computing platforms, MapReduce was developed to run on clusters made up of commodity hardware. The use of commodity hardware significantly reduces the capital cost of setting up the cluster. As a result, a number of organisations have established MapReduce clusters on site in order to provide analysts with dedicated resources and on demand computing power.

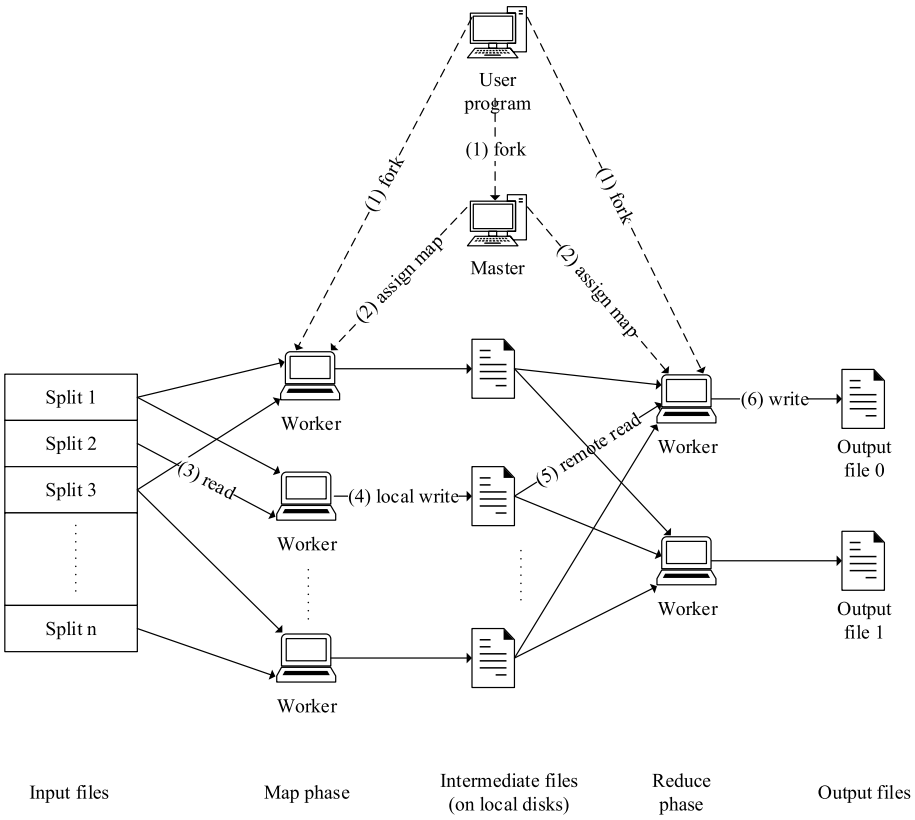


Figure 5.1: MapReduce architecture and control flow

The key elements of the MapReduce model are shown in Figure 5.1. The typical execution of a MapReduce program consists of two phases, the map phase and the reduce phase. In the map phase, multiple tasks are created that would generate partial results for the final output. The reduce process combines the intermediate results generated by the map processes to produce the final output. The coordination of the map and reduce processes and the parallel task execution is performed by the master node. Slave nodes execute the individual map tasks and manage a portion of the GFS.

Due to the simplicity of the MapReduce programming model, programmers have widely adopted the MapReduce architecture in parallel and distributed applications. Currently, the MapReduce implementation at Google is proprietary, and is unavailable for public access. A number of open source and proprietary implementations of MapReduce have been developed such as Hadoop (Apache, 2013), Disco (Mundkur et al., 2011) and Phoenix (Yoo et al., 2009).

The limitations and the strengths of MapReduce are discussed in the sections to follow.

5.2 Hadoop framework

Hadoop was created by Doug Cutting and Mike Cafarella in 2005. The Hadoop project is hosted by the Apache Foundation. Hadoop has been widely adopted compared to the other MapReduce frameworks for several reasons. Hadoop was one of the first MapReduce frameworks offered to the public and the code was written in mainstream Java language. Subsequent introduction of complementary technologies for data storage, data transformation and data extraction have also aided the expansion of the use of Hadoop.

The Hadoop environment follows a master slave architecture. The master is called the namenode and the slaves are called datanodes. The components of the Hadoop MapReduce architecture and their interconnections are shown in Figure 5.2.

The namenode is responsible for maintaining the file system index and the location of the individual blocks of a file. The namenode is the only source of information about the file locations and datanodes rely on the namenode to perform their functions. If the namenode fails, all the file indices would be lost along with access to those files. Therefore, ensuring the correct function of the namenode is essential for the health of the cluster. The secondary namenode creates periodic backups of the contents of the primary namenode which is useful for data recovery.

Datanodes are the workers of the cluster. Each datanode has its own storage which forms a part of the distributed file system. Each datanode also hosts a task tracker which monitors and coordinates the progress of the tasks currently executing. When a MapReduce job is running on the cluster, datanodes would execute the map and reduce processes and perform input output operations.

The job tracker coordinates the execution of a MapRedce job across the entire cluster. The namenode would attempt to distribute the file blocks across the cluster evenly and the job tracker would assign map tasks on the node that hosts data. The job tracker maintains a list of the datanodes assigned to a particular job and performs error recovery in case a datanode fails. Details of the error recovery features of Hadoop are discussed later in this chapter.

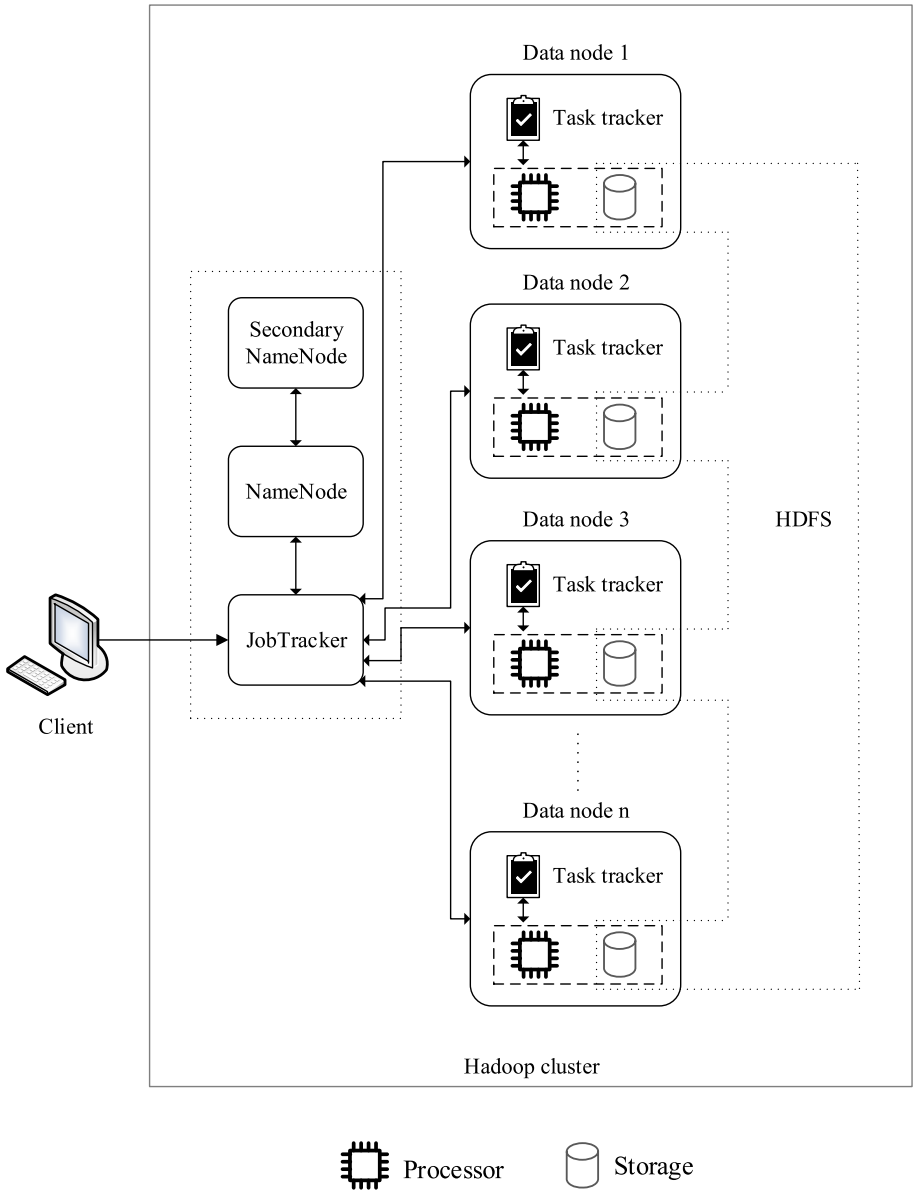


Figure 5.2: Hadoop framework architecture

5.3 Hadoop Distributed File System (HDFS)

The Hadoop distributed file system is designed to hold massive data files. Files are arranged in blocks similar to a disk file system; however, the block size is significantly larger. The default block size on HDFS is 64 megabytes which implies even a text file containing a single character would occupy a 64 megabyte block. The reason behind using larger block sizes is to accommodate large files in gigabyte or terabyte scale. Therefore, it is advantageous to arrange data as large files spanning several blocks.

Since Hadoop clusters are made up of commodity hardware, the likelihood of failure is very high. If one node fails, the cluster should be able to recover the data stored on the failed node. HDFS utilises data replication in order to compensate for possible node failure. For each block in a file, two more copies are created across the cluster by default. Two copies of each block would be hosted on the same server rack and the third copy would be hosted on a different rack.

5.4 Using Hadoop for large scale data analysis

The MapReduce computing model combined with the HDFS creates a well suited environment for large scale data analysis. The Hadoop framework accommodates parallel and distributed program execution while the HDFS enables efficient storage of massive datasets. The following sections discuss the features of Hadoop that have increased its wide adoption for large scale data processing.

5.4.1 Processing power

The Hadoop framework is capable of scaling to thousands of computing nodes. Currently, Yahoo Inc., an internet services company, hosts a Hadoop cluster with over 10,000 nodes (Zawodny, 2008). Sorting terabytes of data has been used to benchmark Hadoop cluster performance and currently Yahoo holds the record, as given in O'Malley and Murthy (2009). Compute-intensive algorithms such as the SOM require a very high degree of processing massive datasets. By distributing the workload among the nodes in the cluster, Hadoop can efficiently perform computations.

Trial and error approaches are a common approach in data exploration. However, the turnaround time of the analytical task should be short enough to produce the analysis outcomes within a practical time frame. It is possible to evaluate different analytical methods and parameter configurations using the Hadoop infrastructure by executing compute-intensive tasks concurrently.

5.4.2 Storage capacity

Two problems would have to be addressed when storing and retrieving large data files. Firstly, the storage capacity of the system should be large enough to hold the file and secondly, data retrieval should be efficient. Considering one terabyte of data stored on a single disk and a disk read speed of 100 megabytes per second, the read operation would require approximately three hours to complete. The write operation would be even slower.

Hadoop is equipped with a federated file system geared to store and retrieve data files spanning terabytes and petabytes efficiently. In addition to the large storage capacity, the HDFS provides faster access to large files. Due to the distribution of the blocks of a file across the network, each node would read a single block and the entire dataset would be read much faster than a single disk system. File reads are performed in parallel, thus the total time required to read large files is significantly reduced.

5.4.3 Hadoop ecosystem

In addition to the data processing and storage features offered by the Hadoop framework, a range of complementary technologies has been developed in order to assist the data analysis process such as Hive, HBase, Sqoop, Pig and Mahout (Monteith et al., 2013). Each technique is suitable for different components of the exploratory data analysis process. Hive and HBase are data stores, Sqoop is a data extraction tool, Pig is a data transformation tool and Mahout is a collection of data analysis algorithms. These technologies are briefly described in the following sections.

HBase and Hive

HBase and Hive use Hadoop and HDFS to create a structured data store for large data volumes. HBase and Hive are widely used as data warehouses. HBase is a column oriented database which stores data in the form of key value pairs. Hive, on the other hand, is

a relational style data store with querying capabilities. Hive query language is used to manipulate data which is an adaptation of structured query language (SQL).

Sqoop

Sqoop is a data transferring tool which is designed to import and export data between Hadoop and relational databases. Sqoop supports incremental loading of tables which is particularly desirable for large scale data processing applications. Sqoop can be used to import data from production databases into HDFS, Hive or HBase. Furthermore, the analytical outcomes can be exported to relational databases for reporting purposes using Sqoop.

Pig

The contents of the data files may need to be transformed into a format compatible with data analysis techniques. For massive volumes of data, MapReduce can be used to improve the efficiency of the transformation. Pig is a data transformation tool which can be used to transform data into different formats. The transformation instructions are given in a language named Pig Latin. The Pig runtime creates MapReduce jobs to execute the instructions in Pig Latin which are executed in parallel.

Mahout

Apache Mahout is a collection of MapReduce data analysis algorithms developed on top of the Hadoop framework. Mahout provides distributed implementations of popular clustering, classification and recommendation algorithms, details of which are given in Owen et al. (2011). Mahout has significantly contributed to the use of Hadoop for large scale data with modular data analysis algorithms which analysts can incorporate into their applications.

5.4.4 Economy

When MapReduce was introduced in 2004, the main difference from the then existing parallel and distributed implementations was that Hadoop could run on commodity hardware. Commodity hardware is much cheaper than specialised parallel computing hardware. If a Hadoop cluster node fails, it can be replaced by any regular computer system. Due to the

use of commodity hardware, setting up and maintaining a Hadoop cluster has become a comparatively low cost operation. Therefore, commercial infrastructure providers such as Amazon and Microsoft offer access to Hadoop at an affordable price. In addition, if a distributed computing cluster is set up at an organisation, the cost of maintaining the cluster could be high due to hardware purchase and for specialist personnel. As a result, many organisations use the services of Amazon, Microsoft and a number of other infrastructure providers due to their low cost.

5.5 Challenges in Hadoop development for processor intensive multi variate data

Although Hadoop clusters offer a massive amount of processing power and computing resources, many applications fail to harness the full potential such systems. According to Das (2009), four types of issues can be identified, imbalance in input splits, imbalance in computations, imbalance in partition sizes, and imbalance in heterogeneous hardware. Although programmers do not have any influence over the hardware, the remaining three issues can be addressed in the program design.

Implementing concurrent algorithms on distributed and cloud computing infrastructure poses a number of challenges. They range from setting up infrastructure to balancing data and computational load, as discussed in Zhang et al. (2010). In the context of exploratory data analysis, which is both data and CPU intensive, the key challenges for the programmers are data loading, splitting, load balancing and efficient processing. These are described in the following sections.

5.5.1 Data loading

Exploratory data analysis is performed on data generated by software applications of an organisation. Massive real-world datasets span terabytes and petabytes of disk space. In most cases, data is accumulated in databases geared to handle massive volumes of data such as Teradata, Oracle, Cassandra and MySQL. Querying production databases is avoided to a large extent in data analysis exercises since running multiple queries on production systems would slow down or crash applications. The best practice is to import data into a data warehouse periodically and use the data warehouse as the source for data

analysis exercises. Importing data from such systems is a key challenge in developing a seamless Hadoop data analysis solution.

In order to ensure fast performance, the data has to be available in the HDFS of the Hadoop cluster. Therefore, the most common choices for the data warehouse are HBase and Hive (Thusoo et al., 2010).

5.5.2 Splitting

Splitting is the process of dividing the job into a number of smaller jobs that will be executed in parallel. The splitting method determines the number of map processes. Determining the number of map processes is critical to ensuring full utilisation of the cluster. The split operations are wrapped inside InputFormat classes in Hadoop. The Hadoop framework supplies three classes to handle input formats: `FileInputFormat`, `TextInputFormat` and `SequenceFileInputFormat`.

The `FileInputFormat` is used to read files in blocks. If the data file spans more than one block in HDFS, the file is split by blocks such that a map process is created for each block in the file. The main disadvantage of splitting by blocks is the inability to decide the exact split point. If the end of the block occurs at the middle of a line, the map process should manage incomplete lines. For data analysis applications, ignoring incomplete lines could lead to missing data records and thus omission of possible patterns. The `FileInputFormat` is suitable for character and sequence search applications. The `FileInputFormat` class creates mappers with file offsets as keys and the contents of the block as text.

The `TextInputFormat` class is used for applications where data rows are separated by new line characters. A mapper is created for each line of the file. The `TextInputFormat` is designed to process text data with long string entries in each line. Using this class for a large volume of short string lines would result in high levels of overhead since a large number of map tasks would be created. The `TextInputFormat` class is unsuitable for multivariate data analysis since datasets usually contain a large number of records.

If the data consists of binary sequences, the `SequenceFileInputFormat` can be used. The mappers have to process the binary values to produce the output. As data in exploratory analysis is mainly textual, the `SequenceFileInputFormat` cannot be used.

It is evident that the three input format classes provided by the framework are not capable of processing multivariate data. The type of input format class suitable for the

Distributed GSOM algorithm would process multivariate data files to create splits at line boundaries, parse attributes and ensure a sufficient number of records are passed into each map task. Since none of the default classes possess such behaviour, a new input format class would have to be defined for SOM based data analysis.

5.5.3 Load balancing and node assignment

During job execution, the Hadoop runtime does not guarantee that all the map tasks are simultaneously executed. Scheduling of the map tasks mainly depends on node availability. Data and processor-intensive applications could suffer from availability based node assignment since improper or unbalanced assignment could become bottlenecks. If two map tasks are assigned to the same node, it is equivalent to running two serial tasks. Performance could suffer if frequent context switches occur. Fischer et al. (2010) have proposed an algorithm which executes the map tasks efficiently to achieve faster performance.

In addition, the homogeneity of the cluster is an important consideration since the nodes in a heterogeneous cluster would have different computing capabilities. Several approaches have been proposed to compensate for the heterogeneity of the cluster, such as using a fitness function for scheduling tasks (Liu et al., 2011) and using multiple queues (Tian et al., 2009).

In order to ensure optimal performance, each node would have to process equivalent computational loads and all the map tasks should execute in parallel. However, making changes to the Hadoop framework may be impossible in some instances. The best possible method is to address the node assignment issue within the Hadoop application.

5.5.4 Processing

Data locality is one of the key aspects of ensuring the efficient execution of a Hadoop job. Data locality refers to using the physical storage of a node to store the data processed by the same node. For data intensive applications, having to access data over the network could hinder the efficiency of the program. Figure 5.3 shows different situations that could arise in data storage.

The performance of the SOM algorithm would depend on data locality. The current Hadoop framework does not guarantee data locality. Modifications to the Hadoop framework have been proposed by Jin et al. (2011), where the task scheduler is modified

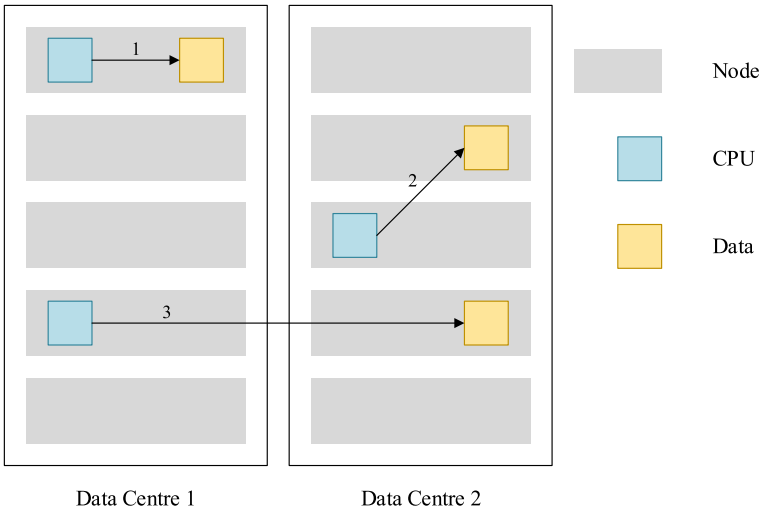


Figure 5.3: Different data locality scenarios. 1) Data processed on the same node as the CPU. 2) Data available over the network at the same data centre. 3) Data located at a different data centre

to assign map tasks data at the same node. Xie et al. (2010) propose a data placement strategy to ensure data is stored locally for map tasks.

5.6 Applications of Hadoop for data clustering

Hadoop has been widely used for clustering due to the compute-intensive nature of clustering algorithms. Zhao et al. (2009) have used a MapReduce implementation of the K-means algorithm to improve te efficiency of the clustering process. Hadoop was used to execute SOMs for clustering demographic data in Nair and Mehta (2011); however, the publication does not provide details of the SOM implementation of Hadoop.

A detailed implementation of the Batch SOM algorithm on Hadoop is given in Weichel (2010). The proposed implementation is similar to the Sparse Batch SOM proposed in Lawrence et al. (1999). The suggested approach creates a distributed map and weight updates for the BMU are performed locally and, at the end of each iteration, the weights of the global network are updated. The Batch SOM would work well for sparse datasets but would create distortions for dense data. Therefore, the clustering accuracy would suffer for dense data using the method in Weichel (2010). In addition, this approach would generate a significant volume of network traffic across the cluster for large datasets and would suffer from the same limitations discussed in section 2.9.2.

This chapter presents an efficient implementation of the SOM on Hadoop in the form of the Distributed GSOM. Since the dataset is partitioned into multiple smaller subsets,

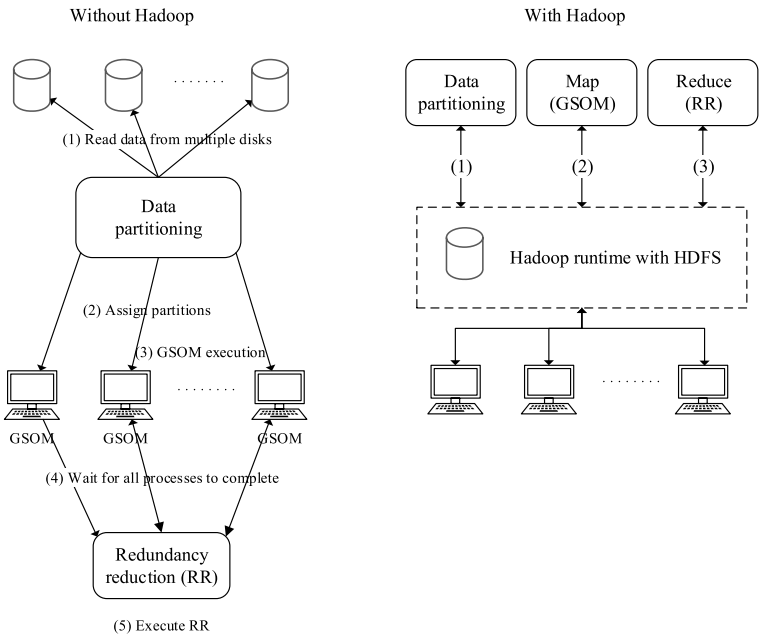


Figure 5.4: Comparison of the implementation of the Distributed GSOM on a distributed cluster by itself and on Hadoop

the computation time is significantly reduced. The details of the Hadoop implementation of the Distributed GSOM are discussed in the following section.

5.7 A MapReduce architecture for the Distributed GSOM

5.7.1 Why Hadoop?

A key concern in implementing distributed algorithms on cloud clusters is the coordination of the complex communication and synchronisation processes. Since Hadoop provides an abstract communication model, the programming complexity is significantly reduced in application development. Figure 5.4 shows the implementation of the Distributed GSOM algorithm on a distributed cluster with and without Hadoop.

If the Distributed GSOM is applied for a large scale data analysis task on a distributed environment, the program would have to handle consolidating data on multiple disks for the partitioning process. The partitions would be created and manually assigned to the computing nodes individually. The partitioning process would also have to transfer the data partitions across the network and ensure correct delivery. A GSOM would be trained on each partition at each computing node. The redundancy reduction process would have to wait for all the parallel GSOM processes to complete before redundancy removal.

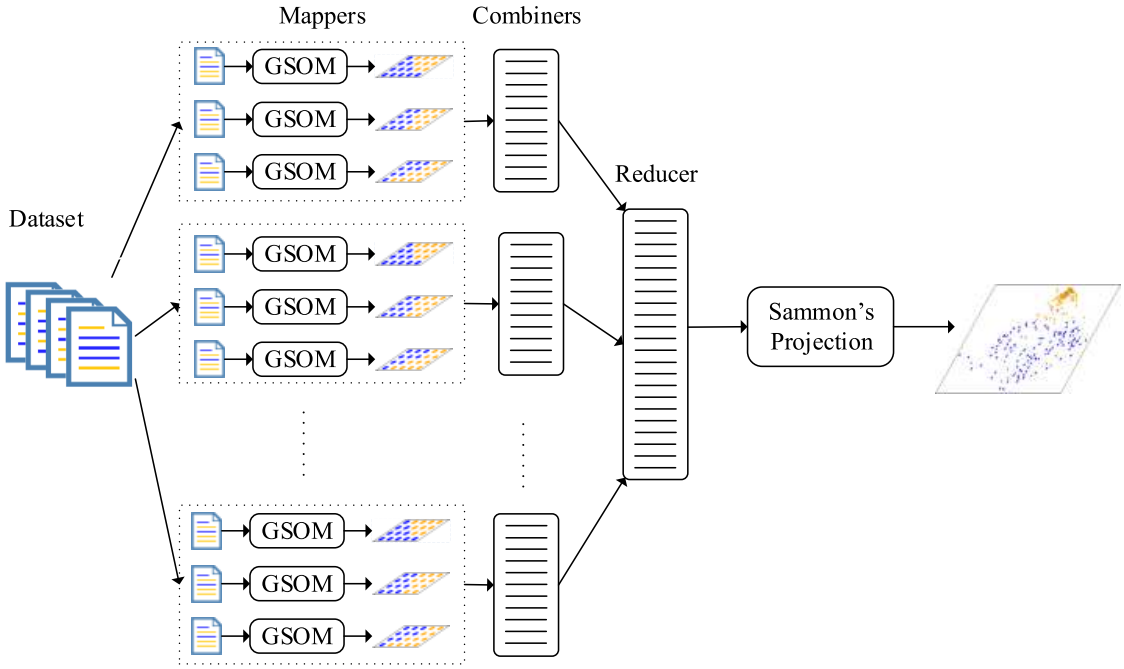


Figure 5.5: A MapReduce architecture for the Distributed GSOM

Therefore, the programmers would have to manage all the complex communication and process synchronisation operations manually.

Hadoop, on the other hand, provides an abstract model for defining distributed program flows. With the HDFS supporting extremely high data volumes, the partitioning process is be able to create partitions on the HDFS itself which would be optimally assigned in the map process. The Hadoop runtime ensures replication and reliability of data transfers and provides the correct inputs for the reducer which would execute the redundancy reduction process. Therefore, the programming time and effort is significantly reduced via Hadoop.

5.7.2 The Distributed GSOM on Hadoop

The MapReduce architecture is built into the Distributed GSOM algorithm. The Data partitioning stage is the map process which initiates the parallel tasks. Each map task is responsible for training a GSOM on a partition. The resulting partition GSOMs are then combined in the redundancy reduction phase which is the reduce process. Although it is possible to integrate the Sammon's projection into the reducer, due to the serial nature of the algorithm, it is best run on a standalone computer. Figure 5.5 depicts the MapReduce architecture of the Distributed GSOM.

The three main components of the algorithm are data partitioning, GSOM execution and redundancy reduction. Each process has to ensure optimal use of the cluster while adhering to the Hadoop programming model. The Hadoop runtime does not ensure that each node is allocated only one map task. If a node is responsible for two map tasks while free nodes exist, the time consumption of the algorithm would increase. Therefore, customisations were made to the standard input output classes in order to optimise the execution of the Distributed GSOM. Details of the modifications are provided in the sections to follow.

5.7.3 Data transformation

The input data processed by the Distributed GSOM can be stored in different formats. Data sources can vary from data warehouses to flat text files. Data warehouses store data in either a relational schema or a star schema. Text files, on the other hand, can store data in widely different formats, both structured and unstructured. The data input to an SOM has to be numeric multivariate data. Therefore, source data would have to be transformed into an SOM compatible format before the Distributed GSOM is executed.

Since the SOM algorithm is an iterative process, accessing the data source for every iteration could result in reduced efficiency levels. In addition, if the data retrieval queries involve joining tables and filtering results, the loss in efficiency would be even greater. Therefore, the data loading process retrieves the data from the data warehouse or production systems and creates a tokenised text file for the dataset on the HDFS. The most common tokenised formats are tab separated and comma separated value formats.

Transformation of massive datasets would require a significant amount of processing power. The intuitive approach would be to develop a data transformation tool running on Hadoop itself. A MapReduce job would extract data from sources, transform the data into an SOM compatible format and save on the HDFS. However, this would require a new MapReduce program to be written for every data transformation task. Pig, a data transformation tool that runs on Hadoop, simplifies this process by automating the MapReduce job creation. Pig requires a script written in Pig Latin which defines the data transformation operation. The Pig runtime converts the Pig Latin script into a collection of MapReduce jobs which efficiently transforms the data into the specified format.

5.7.4 Data partitioning

The data partitioning process creates subsets of data that feeds the GSOMs in the mappers. The data partitioning phase is the starting point of the map process. Two strategies are possible for the data partitioning stage.

1. Create data partitions during the extraction process itself.

2. Extract the entire dataset from the source and create partitions.

For extremely large datasets in the terabyte or petabyte scale, creating a single representation of the dataset before partitioning would result in high time consumption. Integrating the partitioning process into the data transformation process would increase the overall efficiency. If MapReduce is used for data extraction, the efficiency of the process will be even higher. This approach is not applicable for heuristic based partitioning which is based on any metric that is based on the entire dataset. For example, if the partitions are created by the average value of the attributes, averages for the entire dataset have to be calculated in order to group the records.

The partitioning strategy should be decided according to the specific data analysis task. If the node executing the GSOM fails, it should be possible to access a copy of the same data partition to create a new process. Therefore, the output of the data partitioning process should be created as separate files on the HDFS. Otherwise, the contents of the data source may have changed and the same partition may not be created, which would result in inconsistent results.

A MapReduce approach for data partitioning

The time consumption of the data partitioning phase could become a significant proportion of the overall time for massive datasets. The data partitioning process has to read the entire dataset and assign each record to a partition. Reducing the time consumption of the data partitioning phase would improve the efficiency of the overall process.

Random, class based and structure based partitioning create straightforward MapReduce computing tasks where the map process reads the data vectors and outputs the records arranged by their corresponding partition identifiers. Brief descriptions of the MapReduce approaches for creating P partitions are given below.

Random partitioning Each map task would read a block of data and parse the input vectors. For each vector, output key value pairs would be created such that the key is a random integer in [1,P] and the value is the data vector. The reducer would receive the vectors grouped by the keys where each key represents a partition. The data vectors belonging to each partition can be written to the file system within the reducer.

Class based partitioning Similar to the map process in random partitioning, each map task would read a block and determine the class of each data vector. The map process would create key value pairs such that the key is the class name and the value is the data vector. The reducer would then create partitions for each key. If the record distribution is uneven across different classes, the reducer can create multiple partitions for individual classes.

Structure based partitioning Structure based partitioning is similar to class based partitioning where the main difference is that the map process creates keys based on the structural property (time, geography etc.)

Heuristic based partitioning, on the other hand, groups records by a calculated measure in most cases. Partitioning by the magnitude of the data vector, for example requires the entire dataset to be traversed in order to calculate the magnitude of all the vectors. Heuristic based partitioning therefore requires more processing compared to the other partitioning methods. Figure 5.6 shows the architecture of a heuristic based map reduce partitioning process.

It is assumed that the records in each partition are grouped by a calculated heuristic property. Blocks from 1 to n are read in each map process which are then parsed to read data vectors. A heuristic function calculates the property by which the records would be grouped. The map process creates key value pairs such that the key is the calculated heuristic value and the value is the data vector. The shuffle and sort process of the Hadoop framework ensures that the input to the reducer is sorted by the key, which in turn will produce a collection of records sorted by the heuristic property. The reducer then sequentially reads the keys and assigns the records to a partition.

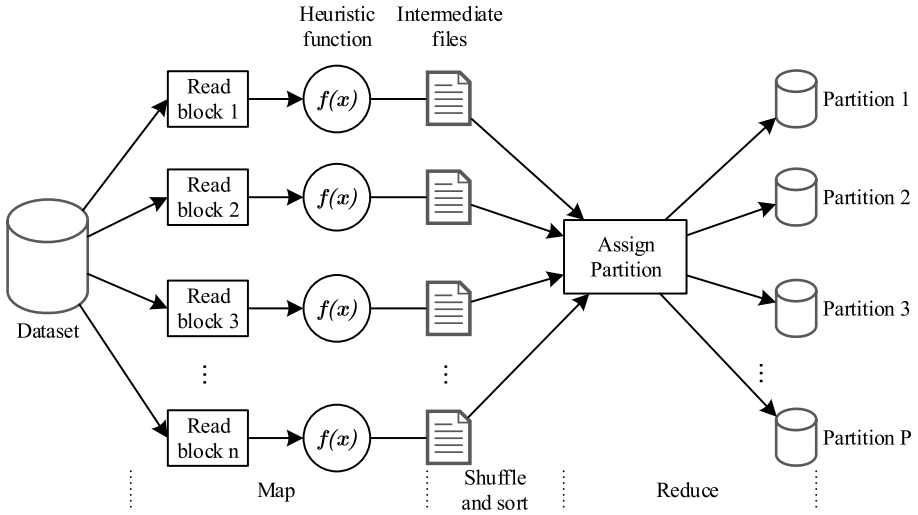


Figure 5.6: A MapReduce example for heuristic based partitioning

5.7.5 Node assignment

Node assignment is critical to the overall performance of the algorithm since the GSOMs are processor-intensive. There would be a significant degradation of efficiency if two or more partitions are assigned to the same node while free nodes are available. The approach used in the Distributed GSOM implementation utilises the split assignment of the Hadoop framework but modifies the assignment to ensure maximum utilisation of the cluster. The node assignment strategy used in this application has two features to preserve data locality while distributing the workload evenly across nodes.

1. Execute the GSOM on the same partition that has the data provided that no other GSOMs are being executed on the same node.

2. If a node hosts more than one partition and is currently executing a GSOM, assign the other partitions to free nodes.

The built-in splitting algorithm in the `FileInputFormat` class attempts to ensure data locality. However, since the compressed data files processed by the partition networks are relatively small in size, it is extremely likely that multiple data partitions on a node would be assigned to process all the partitions. The new splitting algorithm therefore assigns one local partition to a node. The only exception is if the number of partitions exceed the number of nodes. In such situations, the assignment process is repeated across the entire cluster.

Algorithm 4 describes the node assignment algorithm. When a free node is assigned to process data that is located at an occupied node, a data transfer operation occurs. In the worst case, the newly assigned split may already have been assigned to an unoccupied node. Although the algorithm ignores that fact, the benefit of distributing the workload outweighs the communication overhead. This is due to the GSOM execution time being much longer than the data transfer time.

Algorithm 4 Node assignment algorithm

```
1: let originalSplits = get splits from FileInputFormat
2: let newSplits = []
3: repeat
4:    for all node n in the cluster do
5:       let assigned = FALSE
6:       for all split s in originalSplits do
7:          if s.AssignedNode = n then
8:             {s is assigned to the correct node}
9:             add s to newSplits
10:            remove s from originalSplits
11:            assigned = TRUE
12:            break
13:         end if
14:      end for
15:      if NOT assigned then
16:         {n does not have any assigned splits}
17:         assign originalSplits[0] to n
18:         add originalSplits[0] to newSplits
19:         remove originalSplits[0]
20:      end if
21:   end for
22: until originalSplits is empty
23: return newSplits
```

5.7.6 Data reading

The output of the data partitioning process is a list of data files each of which will be processed by a GSOM. The data files are created on the HDFS and would be created in the comma separated value (CSV) format. As discussed in 5.5.2, the standard input format classes in the Hadoop framework are unsuitable for vector based multivariate data processing. A new input format class, `CsvRecordInputFormat`, was developed in order to provide the desired functionality.

Hadoop input format classes employ an implementation of the `RecordReader` class to extract data from files. The `RecordReader` implementation for the `CsvRecordInputFormat` class was named `CsvRecordReader`. The `CsvRecordReader` performs two functions.

1. Read the input file and parse data records.
2. Create a key-value format output as the input to the map process.

Listing 5.1 shows the main component of the `CsvRecordInputFormat` class which is the `next` method. The output of the method creates a list of key-value pairs with the file name as the key and the record list as the value. The `DataVectorList` is a collection of `DataVector` objects which represent the weight vectors of the input dataset.

Listing 5.1: `next` method of the `CsvRecordReader` class

```
@Override
public boolean next(Text key, DataVectorList value) throws IOException {
   // get the next line
   if (!lineReader.next(lineKey, lineValue)) {
      return false;
   }
   List<DataVector> inputList = new ArrayList<DataVector>();
      while (lineReader.next(lineKey, lineValue)) {
      // parse the lineValue which is in the format:
      // record ID, x1, x2, x3,....
      String[] pieces = lineValue.toString().split(",");
         // try to parse double components of value
      double[] weightVector = new double[pieces.length - 1];
      try {
         for (int i = 0; i < weightVector.length; i++) {
            weightVector[i] = Double.parseDouble(pieces[i + 1]);
         }
      } catch (NumberFormatException nfe) {
         throw new IOException(
               "Error parsing floating point value in record");
      }
         // now that we know we'll succeed, overwrite the output objects
```

```
            inputList.add(new DataVector(pieces[0], weightVector));
        }
        key.set(fileName);
        value.set(inputList);
        return true;
    }
```

5.7.7 GSOM execution

The Hadoop runtime creates a map task for each key-value pair generated in the data partitioning stage. The node assignment algorithm ensures the maximum utilisation of the cluster. A GSOM is trained on each data partition concurrently using the standard algorithm. At the completion of the GSOM execution, another list of key-value pairs are created as the output of the mapper.

A distinctive feature of the GSOM in the Hadoop implementation is that the best matching unit always stores the input vectors in order to reduce communication overhead. The redundancy reduction algorithm in the reduce process requires access to the entire dataset in order to calculate quantisation errors. If the reduce process has to access all the partitions in the HDFS, the communication overhead would create a significant loss of efficiency due to network and disk latency.

The output of the GSOM is presented to the combiners and reducers as a list of neurons. Since only one reducer is used, the key is a constant value across all the map processes.

5.7.8 Combiners

The Hadoop runtime does not guarantee the execution of combiners. Therefore, the absence of the combiner should not affect the function of the reducer. Combiners are executed in parallel similar to the map processes. The combiners are used to reduce the redundancy of a subset of the map processes which will improve the efficiency of the overall process if executed. In order to improve the quality of the output, combiners use the redundancy reduction process where the accumulated vectors are used to determine redundancy (method 1 in section 4.3); only the splits assigned to the same server would be used as input to the combiner. Therefore, the amount of data would be a small proportion

compared to the entire dataset. Hence, the performance difference between using method 1 and method 2 in the combiners would be minimal. Thus, the more accurate method 1 is used in the combiner.

5.7.9 Reducer

The input to the reducer is the intermediate output generated by either the map processes or the combiner processes. The intermediate output would consist of lists of neurons with potential redundancy. The primary function of the reducer is to reduce the level of redundancy in the intermediate outputs. While it is possible to use any of the two redundancy reduction methods, method 2, where only the original hit items are used, is preferred since very large datasets would require fewer computations.

Since the all the map processes use the same key, the reducer will receive a collection of neuron lists as input. Each list would correspond to a map process and a GSOM trained on one data partition. Algorithm 1 is used for redundancy reduction.

In order to complete the Distributed GSOM, the output of the reducer has to be topographically arranged. The current projection technique is the Sammon's projection which is a serial process. Running the Sammon's projection on Hadoop would be suboptimal since the parallelism offered by Hadoop would not be used. Therefore, the best strategy would be to write the output of the reducer to the HDFS and use a standalone computer to create the Sammon's projection. Since the Sammon's projection does not require access to the dataset, the resources offered by a standard computer would suffice for this operation.

5.8 Experiments and results

The Distributed GSOM running on a Hadoop cluster was evaluated using the SMH and CoverType datasets. The Hadoop cluster consisted of 32 nodes which accommodated the maximum number used in the experiments for the 32 partitions in the case of the CoverType dataset. Experiments were conducted in order to compare the efficiency results described in Chapter 3 against the experiments using Hadoop. The execution environment in Chapter 3 consisted of a shared memory computing model. As a result, the results also demonstrate the performance of the Distributed GSOM algorithm on shared and distributed memory systems.

The relative advantages and disadvantages of shared and distributed memory models in the context of large scale data analysis are discussed in Chapter 2. The time consumption of the parallel GSOM training phase was expected to have similar time consumption values whereas the redundancy reduction phase of the distributed memory Hadoop implementation was expected to take longer than the shared memory model. However, the time consumption of the redundancy reduction phase is less than 1% of the total time consumption, which does not have a significant impact on the overall performance.

Table 5.1 shows the time consumption statistics for the SMH dataset described in section 3.6.1. The time consumption for the GSOM training phase is similar, with minor differences. The difference can be attributed to the randomness of the GSOM algorithm. The redundancy reduction process involves transferring the intermediate outputs created in the GSOM training phase across the network. Hadoop runtime performs the data transfer across physical nodes whereas the shared memory model performs the data transfer on a shared disk. Therefore the redundancy reduction process of the Hadoop implementation consumes more time compared to the shared memory model. Since the redundancy reduction process consumes significantly less time compared to the overall time, the total time consumption values are dominated by the time consumption of the GSOM training phase. As a result, the overall time consumption is similar across different partition configurations.

Table 5.1: Total time consumption (in minutes) of the Distributed GSOM on the shared memory system and Hadoop for the SMH dataset

	Shared memory			Hadoop		
Partition count	4	8	16	4	8	16
Parallel GSOM training	71.035	26.937	8.642	70.861	27.265	8.259
Redundancy reduction	0.006	0.013	0.024	0.020	0.040	0.070
Sammon's projection	0.082	0.088	0.142	0.093	0.103	0.157
Total time	71.124	27.037	8.808	70.974	27.408	8.486

Similar results were observed for the CoverType dataset (described in section 3.6.1) as given in Table 5.2. Due to a large number of partitions, the Hadoop implementation resulted in longer time consumption values, which can be attributed to the increased levels of network communication. The time consumption of the CoverType dataset experiment is shown in Figure 5.7. It can be observed that the total time consumption of the Hadoop

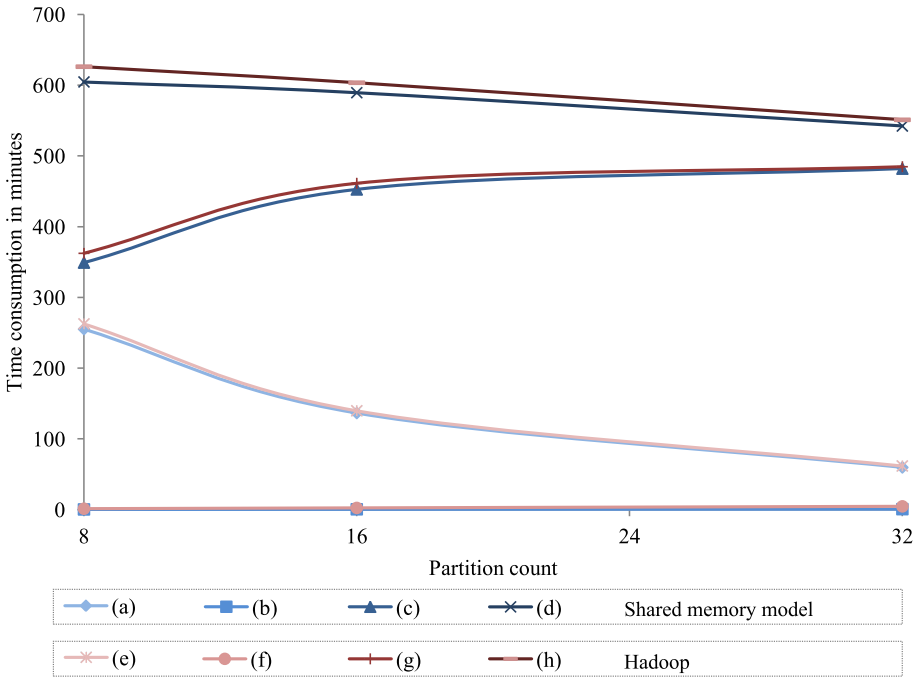


Figure 5.7: Running time comparison of the shared memory model and the Hadoop implementation. (a), (e) GSOM training. (b), (f) Redundancy reduction. (c), (g) Sammon's projection. (f), (h) Total time

implementation is higher than that of the shared memory model. The increase in the time consumption is 3.6%, 2.4% and 1.5%, respectively, for 8, 12 and 32 partition configurations.

Table 5.2: Total time consumption (in minutes) of the Distributed GSOM on the shared memory system and Hadoop for the CoverType dataset

	Shared memory			Hadoop		
Partition count	8	16	32	8	16	32
Parallel GSOM training	255.107	136.403	59.596	262.566	139.757	61.673
Redundancy reduction	0.255	0.423	0.639	1.254	2.365	4.589
Sammon's projection	349.155	452.533	481.939	362.256	461.267	484.698
Total time	604.517	589.359	542.174	626.077	603.389	550.960

The results indicate a minor loss of efficiency in the Hadoop implementation compared with the shared memory model. The lower efficiency of the Hadoop implementation can be attributed to the cost of transferring data across the network that occur at the end of every stage of the Distributed GSOM. However, the delay is comparatively low since the network connecting the nodes in a Hadoop cluster usually consists of a high bandwidth network.

However, the advantages of MapReduce such as failure resilience, abstraction of inter-node communication, scalability to thousands of nodes, storage support for massive data

volumes and a very simple programming model are highly desirable for large scale data analysis applications as given in Dean and Ghemawat (2010). Therefore, MapReduce significantly decreases the development time for data processing applications which, in turn, allows data analysts to produce timely results. Amazon Inc. and Microsoft Inc. currently offer MapReduce solutions as infrastructure as a service (IAAS) which offers affordable computing resources without the cost of maintenance and hardware upgrades. Therefore, the benefits offered by Hadoop outweigh the minor loss of efficiency.

5.9 Discussion

This chapter presents an implementation of the Distributed GSOM algorithm on the popular Hadoop distributed framework. Several programming and design challenges were overcome in order to accommodate the compute-intensive Distributed GSOM algorithm within the standard Hadoop framework. The research contributions include a new node assignment algorithm, a new `InputFormat` class and an architecture to reduce the communication overhead of the program execution.

The research outcomes of the work presented in this chapter can be summarised as follows.

1. A MapReduce architecture for the Distributed GSOM is introduced which is used to implement the Distributed GSOM on Hadoop.

2. A robust splitting algorithm is used to ensure that the records are not partially split. The splitter also ensures each partition network receives a sufficient number of data vectors to facilitate convergence.

3. Hadoop MapReduce processes are proposed for the partitioning of massive datasets. The use of MapReduce for partitioning significantly improves the efficiency of the partitioning process, especially in heuristic based partitioning.

4. In order to minimise the communication overhead of the reduce process, data vectors local to a partition network are transmitted within the neurons. This approach avoids the need for the reducers to access the dataset which significantly reduces the communication overhead.

5. Combiners were used to improve the acurracy and the efficiency of the overall process. By using the accumulated data vector based redundancy reduction approach for the combiners, the accuracy of the analysis is increased. As a result, the reducer is able to use the more efficient original data, vector based, redundancy reduction method.

6. Results indicate that the performance of the Distributed GSOM on Hadoop is more efficient by several orders than the serial algorithm and is comparable to the performance of the shared memory simulation results.

The next chapter introduces a real-world exploratory data analysis application using the Distributed GSOM.

Chapter 6

An Application of the Distributed GSOM for Exploratory Data Analysis

A key motivation for the design and development of a distributed and scalable GSOM was to enable SOM technologies to be used with very large datasets. In this chapter, the Distributed GSOM is used for exploratory analysis of a real-world large dataset to demonstrate its practical application and value.

Smart grid technologies have advanced in the last few decades with the aim of improving the distribution and consumption of electricity. This chapter presents an application of the Distributed GSOM algorithm in smart grids, specifically in the analysis of electricity consumption. Smart electricity meters generate periodic electricity consumption records which are an integral component of smart grids. Due to the massive volume of data generated by smart meters every 15 or 30 minutes, the traditional serial SOMs and GSOMs algorithms cannot be used for the analysis process due to their excessive time consumption.

The Distributed GSOM was used to analyse gigabytes of electricity consumption readings efficiently in order to extract the most common electricity consumption profiles. The work presented in this chapter was conducted in order to achieve the following research objectives.

1. To demonstrate the practical value and usability of the Distributed GSOM with a real life example.

2. To demonstrate how data partitioning is carried out in a real case.

3. To demonstrate how the Distributed GSOM could be used to conduct exploratory analysis in multiple levels of granularity.

4. To demonstrate how multi-granular profiles could be used to link short-term, medium term and long-term electricity consumption profiles based on the the exploratory analysis in 3 above.

5. To develop and demonstrate an efficient model to integrate new data continuously into the existing Distributed GSOM.

6.1 Smart grids

The energy industry has attracted significant attention from the research community due to its importance in the modern society. Current world electricity consumption is heavily dependent on fossil fuels, with more than 80% of the world's electricity production arising from coal, oil and natural gas based electricity generators, according to IEA (2012). Due to the non-renewable nature of fossil fuels and their impact on the environment, optimising the electricity generation and distribution process has become a key research area in the energy industry. Smart grid technologies aim to integrate electricity generation, distribution, information technology, communication and control systems in order to optimise processes in the electricity industry in the most cohesive manner (Fang et al., 2011).
Smart grids comprise three systems;

i. A smart infrastructure system

ii. A smart management system

iii. A smart protection system

A smart infrastructure system is responsible for optimising electricity generation, delivery and consumption by utilising advanced metering technologies and information and communication technologies. Research in smart infrastructure systems has introduced demand side management and time of use pricing methodologies in order to improve the

efficiency of the electricity grids (Palensky and Dietrich, 2011). Smart electricity meters are an integral part of smart grids which enables two way communication between the consumer and the electricity provider. Smart meters are capable of transmitting near real-time electricity consumption readings, usually at intervals of one hour or less.

Demand side management is one of the key smart grid technologies which utilises smart meter readings to determine real-time demand trends to manage electric load effectively, as discussed in Palensky and Dietrich (2011). Time of use pricing is an enabling technology for demand side management which imposes different tariffs on electricity based on the time of use. Smart meters and analysing smart meter data is essential for demand side management as well as for forecasting demand trends and to better understand customers.

Over the years, a number of neural network approaches (Bakirtzis et al., 1995; De Silva et al., 2011a,b; Verd et al., 2006; Lendasse et al., 2002), fuzzy methods (Song et al., 2005; Srinivasan et al., 1999) and statistical techniques (Govindan et al., 2009; Lam et al., 2008) have been introduced for electricity consumption profile analysis. The SOM has distinct advantages in the domain of electricity consumption profiling such as visualisation of the profile distribution and summarisation of the records (Verd et al., 2006).

The influx of data from smart meters could be massive for a utility company with a large customer base. An electricity consumption reading every 15 or 30 minutes results in a high volume data stream to the electricity provider.

It is estimated that a utility company with 2 million customers would generate approximately 22 gigabytes of data every day in a smart grid, as given in Shargal and Houseman (2009).

6.2 Analysis requirements

A multitude of demographic and socio economic factors affect the electricity consumption behaviour of consumers. As a result, these behavioural patterns tend to change over time. In order to maintain the relevance of the analysis outcome, the analysis process should integrate new data and patterns continuously. Furthermore, in order to use demand side management, short-term forecasting and time of use pricing effectively, electricity consumption data has to be analysed with different levels of granularity.

The analysis exercise was performed for a leading utility company in Australia with a 4.3 million customer base (Origin, 2014). The requirements for the analysis are listed as follows.

1. Identify the most common electricity consumption profiles across different time periods;

2. Characterise the electricity consumption profiles using the available data;

3. Investigate the relationship between electricity consumption and temperature.

An electricity consumption profile is defined as a certain behavioural pattern displayed by a group of consumers. For example, working households could consume more electricity during non-working hours and less energy during working hours. On the other hand, retired households or office buildings would consume more electricity during the day than at night.

The business value of profiling lies in the ability to use the profiles to predict the electricity consumption behaviour of new and existing customers. Predicting electricity consumption behaviour would enable the electricity generators to plan the demand better and thus utilise resources efficiently and increase profitability.

The analysis of electricity consumption behaviour could be repeated over a number of time spans. In this exploratory data analysis exercise, daily, weekly and yearly electricity consumption patterns were analysed. Identifying profiles for these different time periods would provide insights into the short-term, medium term and long-term electricity usage patterns of the consumers.

Five research questions were formulated around the above analysis requirements and are listed on the second page of this chapter.

A sample of the dataset was provided by the utility company which contained the electricity meter readings for 10,000 smart meters in Melbourne, Australia, over a year spanning 01 April 2010 to 31 March 2011. Each smart meter recorded the electricity consumption at 30 minute intervals. The dataset was 12 gigabytes in size in raw form with more than 175 million entries. No prior information existed on possible patterns within the data, thus the analysis exercise was a fitting exploratory data analysis task.

6.3 Problem scope

This chapter discusses two key contributions from the work outlined in this thesis which would significantly improve the process of electricity consumption profiling.

1. Distributed GSOM based unsupervised learning to improve the efficiency of the process.

2. An adaptive multi-granular profiling framework

Verd et al. (2006) and Rusitschka et al. (2010) demonstrate the importance of cloud computing for the development of smart grid technologies. The Distributed GSOM provides a scalable solution to traditional SOM based unsupervised learning that is several orders faster, as discussed in Chapter 3. In addition, the Distributed GSOM uses a distributed approach which could harness the computing power of cloud computing platforms to analyse electricity consumption data efficiently.

6.4 The advantages of the Distributed GSOM

Considering the SOM based approaches for achieving the analysis objectives, out of the SOM, GSOM and the Distributed GSOM, the Distributed GSOM was selected for the following reasons.

i. Low time consumption of the Distributed GSOM resulting in shorter turnaround time;

ii. utilisation of heuristic based partitioning to group records based on electricity consumption which improves the spread of the map;

iii. customisability of the spread of partition networks to accommodate even spread across all partition networks;

iv. the ability to integrate data into the analysis process continuously without having to re-train the entire network.

6.4.1 Higher efficiency of the Distributed GSOM

The volume of data available for analysis in the order of several gigabytes and was consisted of several millions of records. It has been shown in Chapter 3 that the Distributed GSOM

delivers higher levels of efficiency while maintaining similar levels of accuracy compared to the GSOM. Processing such large volumes requires significant processing power and resources such as physical memory.

It was estimated that the SOM or the GSOM running on a single computer would take approximately 30 days from start to completion. Having a 30 day turnaround time was considered far too long given the trial end error nature of exploratory data analysis. The Distributed GSOM, on the other hand, was estimated to complete the same task in approximately 36 hours which, significantly was, only 5% of the time to run the serial algorithm.

In addition to the 95% saving of time, the Distributed GSOM has demonstrated similar levels of accuracy compared to the SOM and the GSOM. Therefore, the use of Distributed GSOM would result in increased efficiency without having to compromise accuracy.

6.4.2 Ability to use partitioning to improve the quality of the results

An important feature of the Distributed GSOM for the analysis of electricity consumption data is the possibility of creating partitions by grouping similar records since this would improve the quality of the analysis. For example, the daily electricity consumption analysis resulted in the highest number of records with 3,650,000 records with 48 attributes. Using random partitioning resulted in maps highly biased towards high consumption records. Approximately 94% of the records displayed low electricity consumption of 23.76 kWh a day whereas only 0.01% of the records had a very high daily electricity consumption of 19042.5 kWh.

The very high consumption records could be considered as outliers and discarded. However, in an exploratory data analysis task, outliers could also provide valuable insights into the behaviour of the customers represented by such records. As a result, ignoring the outliers would result in the omission of certain patterns. On the other hand, including outliers would detract from the clustering effect of the SOM, as given in Wu and Chow (2004). Therefore, if the entire dataset is presented to a single SOM, the very high consumption records would detract from the cluster separation. In addition, the growth of the GSOM is driven by the accumulated quantisation error in neurons. Having high consumption vectors (with a high magnitude) would dominate the growth and create significant bias

towards such records. Consequently low electricity consumption records would be under represented in the final output.

In order to reduce any bias towards high electricity consumption records, data partitioning was performed such that records with similar electricity consumption values were grouped within the same partition. The heuristic used was that the total electricity consumption for the day could be used to differentiate between high and low electricity consumption meters. Applying this heuristic, the records were sorted according to the total electricity consumption and partitions were created from sequential records resulting in the grouping of records with similar total electricity consumption.

6.4.3 Customisability of GSOM parameters of partition networks

With total consumption based partitioning, partitions would be created with records having extremely low electricity consumption. It was observed that the dataset was skewed in terms of magnitude and also the attribute variance increased with the increase of the magnitude. Figure 6.1 shows that the dataset is highly skewed, with 93% of the vectors having a magnitude of [0–2] and the highest standard deviation is observed for the vectors with a high magnitude.

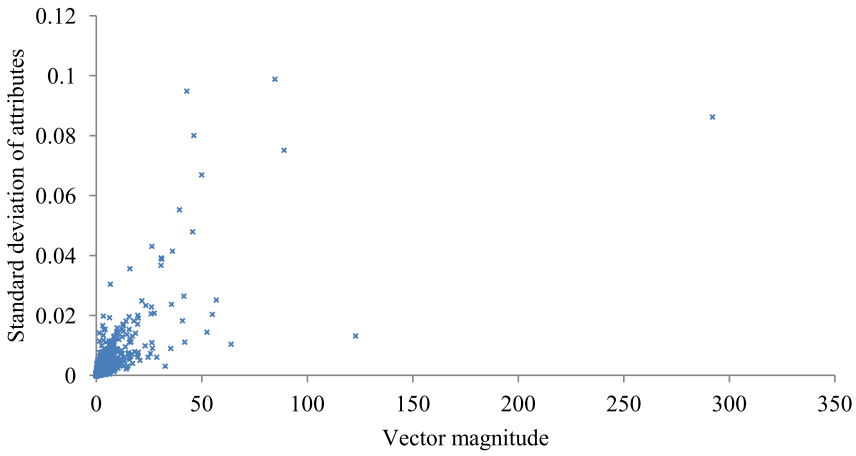


Figure 6.1: The relationship between attribute variance and magnitude

In order for the GSOM to grow nodes, neurons should accumulate a quantisation error greater than the growth threshold. The quantisation error is determined by the difference between the input vector and the prototype vector. Vectors with a lower variance tend to create low quantisation errors as a result of smaller differences in attributes.

Wilson and Martinez (1997) demonstrate that the Euclidean distance creates bias towards attributes with a high variance, resulting in higher quantisation errors for vectors with a high variance. Therefore, it would take a higher number of iterations for low variance vectors to generate an accumulated quantisation error greater than the growth threshold. Hence, partition networks representing low electricity consumption partitions with a low variance tend to contain fewer neurons compared to high consumption partitions. Although standardisation is shown in Wilson and Martinez (1997) to be a means of reducing the bias towards high variance vectors, the time consumption of standardisation would be very high for large datasets.

The Distributed GSOM has the ability to compensate for this effect and ensure even spreads across all the partition networks. By using a higher spread factor for low electricity consumption partition networks and a lower spread factor for high consumption partitions, an even representation of all the consumption patterns could be obtained. The spread factor can be used to adjust the spread of the GSOM as given in Alahakoon (2004). Therefore, two spread factors are used for partitions with high and low consumption partitions in order to ensure even spreads across the partition networks. The low spread factor generates a high growth threshold (GT_{High}) and the high spread factor creates a low growth threshold (GT_{Low}) according to 2.4.

For high quantisations created by the high consumption vectors, the high growth threshold is used which limits the spread of the map. New neurons are created for n when the accumulated quantisation error QE is,

$$QE = \sum_{x} ||n - I[x]|| \geq GT_{High} \tag{6.1}$$

where I is the list of input vectors mapped onto n.

On the other hand, low quantisation errors are created for low consumption vectors. Therefore a lower growth threshold is used to increase the spread of the network. New neurons are created when,

$$QE = \sum_{x} ||n - I[x]|| \geq GT_{Low} \tag{6.2}$$

Having an evenly distributed set of partition networks would ensure uniform representation of the entire dataset in the Distributed GSOM output.

6.4.4 Ability to integrate data continuously

Electricity usage for a meter is determined by several factors, as described below with examples.

i. Time: Electricity usage changes with the time of day as well as the time of year. An office building would consume more electricity during the day compared to other times. Summer and winter tend to have higher electricity consumption compared to spring and autumn.

ii. Environmental factors: Temperature has a clear influence on electricity consumption especially during summer and winter. Houses equipped with air conditioners would consume more electricity during summer on days with high temperatures. On the other hand, houses which use electric heaters would consume more electricity during winter on days with low temperatures.

iii. Demographics: Working households tend to consume more electricity during the early morning and the evening than during the day. Young people may have a high rate of electricity consumption due to their use of computers and video games.

iv. Property features: Features of the property would have a high influence in determining electricity consumption. Houses with gas heating would consume less electricity during winter compared to houses with electric heating.

v. Attitude towards energy saving: Personal attitudes towards energy saving would also have an impact on the electricity consumption of households. A conservative attitude would try to reduce energy consumption whereas a less conservative attitude would use more electricity.

Due to the varying nature of the above factors, patterns of electricity consumption behaviour are likely to change over time. The rate of change would further increase with migration when tenants or home owners relocate from one house to another. Therefore, the profiles identified during the analysis frequently change, as given in Darby (2006). Hence, the relevance of the knowledge acquired in the Distributed GSOM should be maintained by incorporating new patterns into the existing network.

If new data becomes available, the traditional SOM or GSOM algorithms would require complete re-training which could consume significant amounts of time. The Distributed

GSOM, on the other hand, has the ability to integrate data continuously by training a network only on the new data and integrating the new GSOM into the existing map. Further details of the Distributed GSOM's ability to integrate data dynamically is given in 4.4. The GSOM could be integrated by executing the redundancy reduction process between the new GSOM and the existing network, followed by Sammon's projection.

6.5 The analysis process

The use of SOM unsupervised learning techniques has been successfully used for electricity consumption profile identification in Chicco et al. (2003), Valero et al. (2007) and Lendasse et al. (2002). The analysis process consists of several stages, pre-processing, SOM training and clustering. Therefore a Distributed GSOM solution is proposed which significantly decreases the time consumption of the overall process. Figure 6.2 shows the major components of the analysis process.

The process starts by pre-processing the data in order to ensure compatibility with the Distributed GSOM algorithm. Depending on the duration of the profile, aggregation may be required. For example, if half hourly readings are used for annual profiles, each vector would contain 17,520 attributes which provides an unnecessary level of detail while making the analysis process more processor-intensive. Consumption based partitioning is used in order to reduce distortions in the partition networks.

6.5.1 Pre-processing

The data generated by smart electricity meters in raw form consists of tuples with the structure,

<meter identifier, electricity consumption in kWh, day, interval, period, end time >.
The main data values in each tuple could be considered as the meter identifier, day, period and the electricity consumption. The raw data was arranged by meters, and for each meter the time stamp of the reading was recorded along with the block number (1–48) for the day.

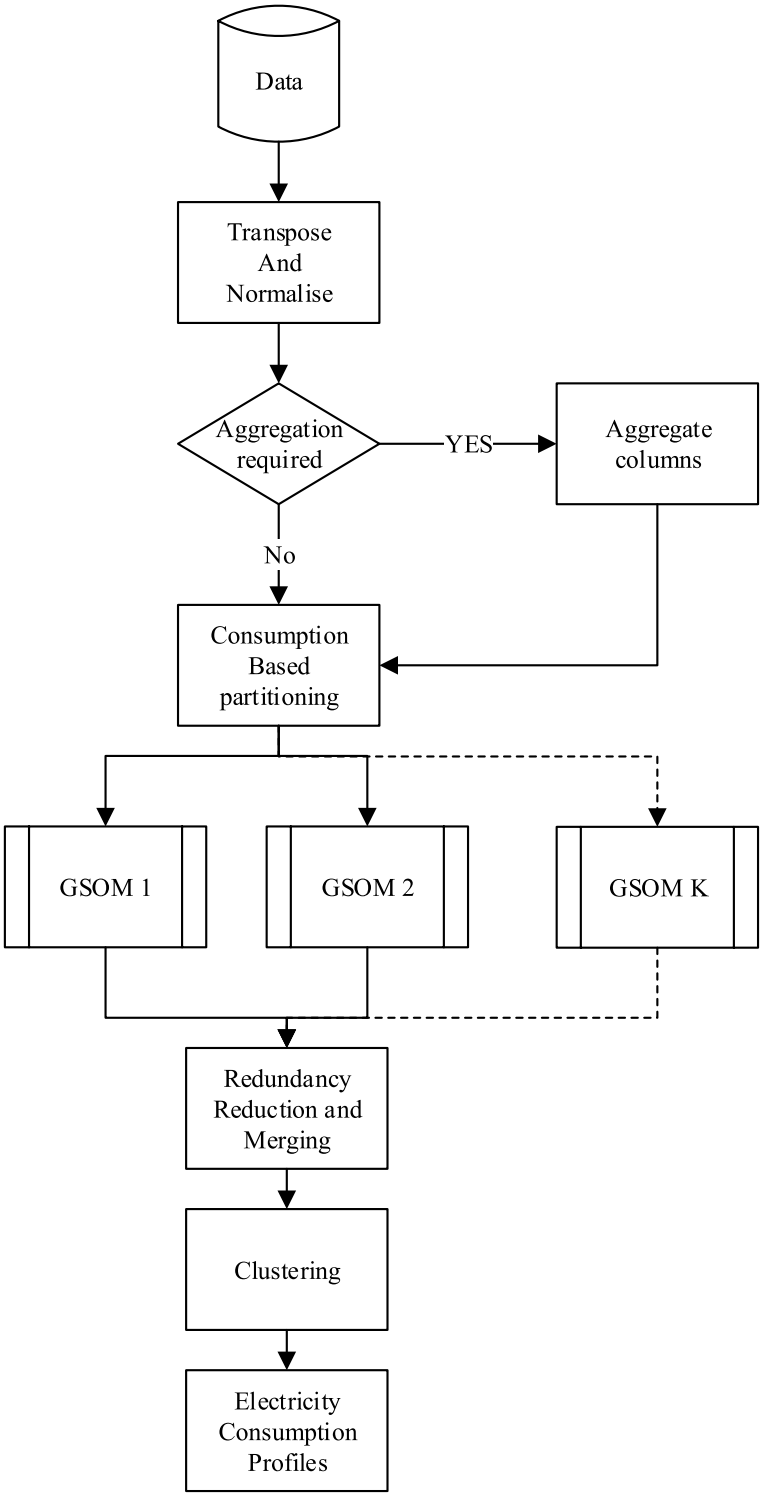


Figure 6.2: The Distributed GSOM algorithm

Since the raw data files contained more than 175 million lines, the dataset was transposed into a multi-dimensional vector space in order to use SOM based algorithms. Normalisation was performed on the generated data file in order to ensure compatibility with the SOM algorithms.

Since the consumption data has a temporal relationship along the columns, normalising by individual columns was deemed to be suboptimal. For each experiment configuration, normalisation was carried out by considering the maximum and the minimum across all the columns. Exploration of the data was carried out in three granularity levels which determined the number of rows and columns of the dataset, as given in Table 6.1.

6.5.2 Data Partitioning

Due to the significant variations of the electricity consumption values in data, if both high and low consumption records are assigned to the same partition, the partition network would be biased towards the high consumption records. Random partitioning was ruled out for that reason. Since any information about the possible classes in the data was unavailable, class based partitioning was also infeasible. A heuristic based partitioning technique was developed, based on the consumption values, in order to avoid bias in partition networks towards high consumption records.

The total consumption for each row was calculated and the entire dataset was sorted by the total consumption figures. Due to high variation in the consumption readings, total consumption based partitioning was used to group records with similar electricity consumption within the same partition. All data records were sorted by the total electricity consumption and the required number of partitions was created. Due to the limitations of the distributed computing environment, 128 partitions were created for each level of granularity. Consumption based partitioning would perform better than random partitioning due to the lower levels of variations within partitions.

6.5.3 Network training

GSOMs are used for unsupervised learning to summarise each partition and create partition networks. Parameters for the GSOMs are specified based on the total electricity consumption with high spread factors for low consumption partitions and low spread factors for high consumption partitions. Due to the differences in electricity consumption

values across the partitions, different spread factors were used to ensure even spreads of the partition networks. The low spread factor was set to 0.3 and the high spread factor was set to 0.5.

6.5.4 Clustering

In order to identify electricity consumption profiles, data has to be clustered where each cluster would represent a profile. The neurons in the Distributed GSOM are clustered using the k-means algorithm (MacQueen, 1967). The cluster centroid would be used as the consumption profile for the records belonging to the cluster. The cluster centroid is the average of all the neuron weight vectors in the cluster. Therefore, the actual weight vectors of the neurons may have minor differences from the cluster centroid. The visualisation feature of SOM based techniques is quite useful for browsing the cluster and determining closely related patterns.

6.6 Analysis outcomes

6.6.1 Dataset

The dataset contained electricity consumption readings from 10,000 smart meters. For each meter, half hourly electricity consumption in kWh was recorded for a year, from 01 Apr 2010 to 31 Mar 2011. Forty eight electricity consumption values were recorded for each day for 365 days. The dataset consisted of 12 gigabytes of data in its raw format which was pre-processed in order to ensure compatibility with the Distributed GSOM.

6.6.2 Data configurations

The analysis was performed for different time periods in order to identify patterns of consumption:

1. Daily consumption,
2. Weekly consumption,
3. Annual consumption.

The number of rows and columns for each configuration is listed in Table 6.1.

Table 6.1: Dataset statistics

Configuration	Rows	Columns
Daily	3,650,000	48
Weekly	510,000	336
Annual	10,000	1440

Analysis of electricity consumption behaviour in terms of daily, weekly and annual time periods have been used in a number of other approaches such as Wood and Newborough (2003), Yao and Steemers (2005) and Badran et al. (2008). In order to investigate the identified three granularity levels, three types of experiments were carried out.

1. Daily electricity consumption analysis
2. Weekly electricity consumption analysis
3. Annual electricity consumption analysis

For each type of analysis, electricity consumption values were arranged in attributes according to their temporal ordering. The daily electricity consumption analysis created 365 vectors for each meter corresponding to each day of the period from 01 April 2010 to 31 March 2011. Each record consisted of 48 attributes corresponding to each 30 minute interval of the day. The first attribute would be the reading for the 30 minute interval from midnight to 00:30 while the 48^{th} attribute would be for the interval from 23:30 to midnight.

Similarly, the weekly analysis was performed on records where each record was based on the weekly electricity consumption for a meter. Fifty two records were generated for each meter where the record started Monday 00:00 to 00:30 and ended on Sunday 23:30 to 00:00. Consequently, 336 (48×7) attributes were created for each record.

Annual analysis resulted in the fewest records with one record per meter. It was evident that some level of aggregation was required, since using the original 30 minute interval readings was deemed unnecessary in the case of annual consumption. In addition, aggregating readings would result in fewer attributes thereby improving the efficiency of the analysis.

The Distributed GSOM was run on a simulated distributed computing environment with a maximum of 128 computing nodes. Each node possessed 2 GHz of processing power and 2 gigabytes of memory.

6.6.3 Daily electricity consumption analysis

Identifying daily electricity consumption profiles is useful for time of use pricing decision making. A mathematical model for daily electricity consumption is given in Paatero and Lund (2006) which generates profiles based on the appliances in the household. However, appliance information would not be available for utility companies, which limits the relevance of this method. Jardini et al. (2000) propose a supervised method which requires standard profiles to be defined. In addition, specialised hardware has been used to record the electricity load. The Distributed GSOM provides an unsupervised method which has the ability to generate profiles without household appliance information.

The objective of the daily electricity consumption analysis was to group the meters by different daily electricity consumption. Each record represented the daily electricity consumption values for a single meter for a single day. In order to capture the patterns as comprehensively as possible, the original granularity of half hourly intervals was used. Therefore, 365 records with 48 attributes were created for each meter and the entire dataset for 10,000 meters contained 3,650,000 records which created the largest dataset for the five configurations.

Hundred and twenty eight partitions were created for the Distributed GSOM based on total daily electricity consumption. In order to ensure even spreads across the partitions, a high spread factor of 0.5 was used for the 64 least consumption partitions and a low consumption spread factor of 0.3 was used for the remaining 64 partitions. The Sammon's projection output was used for analysis purposes and K-means algorithm was used to group the neurons in the Sammon's projection into 10 clusters.

Daily electricity consumption profiles

Figure 6.3 shows the cluster distribution for the daily electricity consumption records for the entire dataset. It can be seen that there are two types of consumption profiles. One is a Gaussian like curve where the peak electricity consumption occurs midday with high consumption between 06:00 and 17:00, as shown by clusters 1, 4, 5 and 7. The other main profile type has approximately constant electricity consumption during the day with only minute variations by time of day as indicated by clusters 2, 3 and 6. It could be said that clusters 8, 9 and 10 possess hybrid profiles of the two main types with

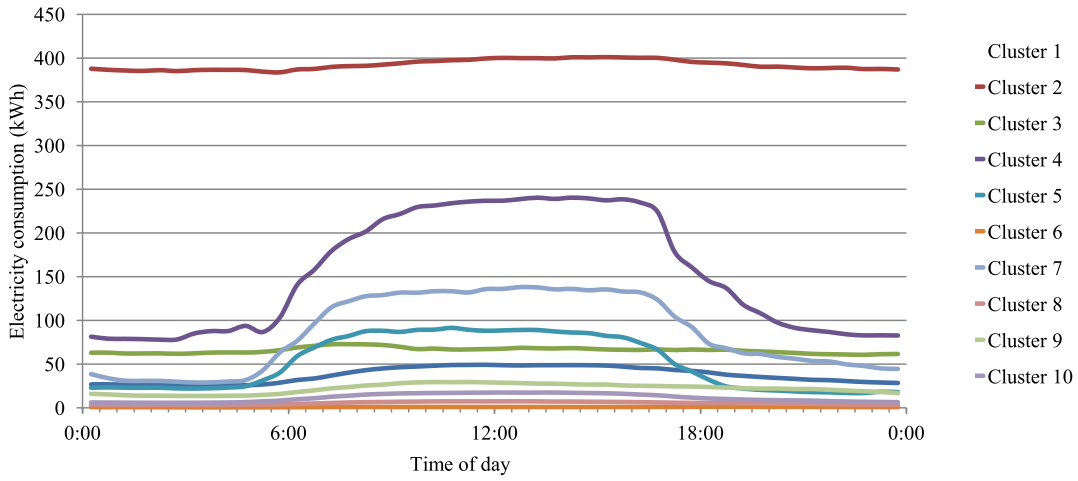


Figure 6.3: Entire dataset: daily electricity consumption by time of day

somewhat constant electricity consumption along with consumption peaking during the day. The main difference between the clusters in these three types is the average electricity consumption which clearly differentiates the profiles. The consumption statistics and the relative sizes of the clusters are given in Table 6.2.

Table 6.2: Entire dataset: daily electricity consumption by no. of records, average kWh, and percentage of total

Cluster	No. of records	Average kWh	Percentage of total
1	5882	1780.89 kWh	0.16%
2	363	19042.50 kWh	0.01%
3	2149	3298.32 kWh	0.06%
4	773	7603.26 kWh	0.02%
5	1962	2424.83 kWh	0.05%
6	3446235	23.76 kWh	94.42%
7	1526	4172.36 kWh	0.04%
8	134311	245.20 kWh	3.68%
9	15842	1035.28 kWh	0.43%
10	40957	541.63 kWh	1.12%
Total	3650000	53.39 kWh	100.00%

It can be seen that the distribution of records across the clusters is quite uneven, with 94% of the records belonging to cluster 6 where the average daily electricity consumption is 23.76 kWh. This is less than 50% of the average for the entire dataset, which is 53.39 kWh. It is interesting to see that the records in cluster 2 have extremely high electricity

consumption where the daily electricity consumption is more than twice the annual consumption of 94.4% of the readings. Clusters 4, 7 and 3 have moderately high electricity consumption values.

The majority of records were mapped to cluster 6 where 94.42% of the records were present. Cluster 6 represents meters with the lowest consumption. Figure 6.4 shows the electricity consumption profile for the cluster. Since the number of records in cluster 6 is significantly large, it is possible that the records within clusters have profiles different from the cluster profile. In order to investigate this further, the records belonging to cluster 6 were normalised and the Distributed GSOM algorithm was run on the data to generate a second level of sub clusters.

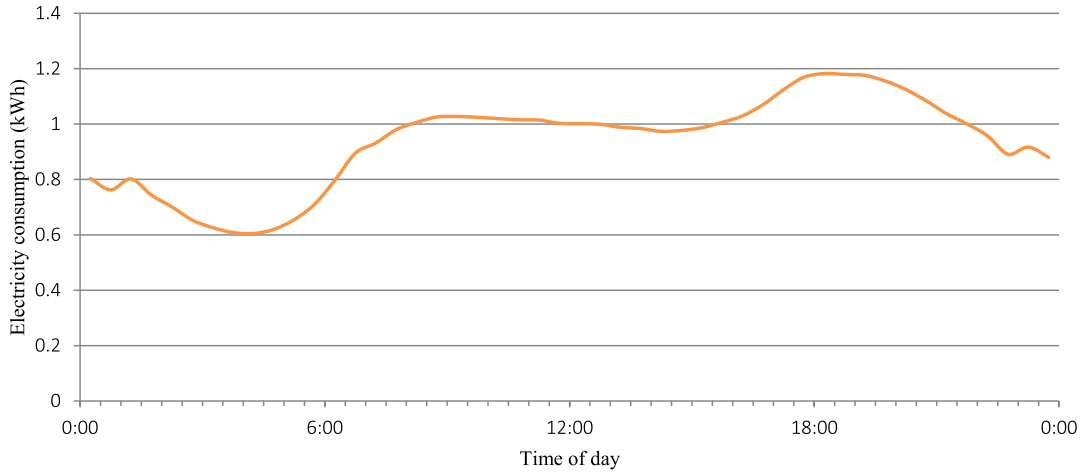


Figure 6.4: Cluster 6: electricity consumption by time of day

As noted, the highest electricity consumption occurs in cluster 2; Figure 6.3 shows that consumption as constant across all time periods. It was concluded that the records in cluster 2 belong to a storage warehouse or some other form of consistent electricity consumption meter. The total number of records in cluster 2 was 363, all of which belonged to the same meter. Therefore, cluster 2 can clearly be identified as an outlier or an anomaly. Average daily consumption was calculated as 19042.5 kWh.

Figure 6.3 shows cluster 4 as another high electricity consumption cluster where the electricity consumption starts to rise around 05:00 and peaks midday. Consumption starts to decrease around 17:00. It was concluded that the records in cluster 4 belonged to offices where high electricity consumption occurs during normal working hours. Only 4 meters indicated the consumption pattern in cluster 4.

Table 6.3: Cluster 6: 10 sub clusters by number of records, average kWh and percentage of total

Sub cluster	Total No. of records	Average kWh	Percentage of total
1	237517	48.10 kWh	6.89%
2	4492	136.42 kWh	0.13%
3	78385	70.31 kWh	2.27%
4	49017	87.15 kWh	1.42%
5	39093	128.05 kWh	1.13%
6	34023	111.11 kWh	0.99%
7	692826	26.94 kWh	20.10%
8	64033	70.80 kWh	1.86%
9	2018305	9.34 kWh	58.57%
10	228544	40.41 kWh	6.63%
Total	3446235	23.76 kWh	100.00%

Expansion of cluster 6

Table 6.3 outlines the statistics for 10 sub clusters in cluster 6. While cluster 6's overall profile has a very even outline (Figure 6.4), the records contained in the cluster have a wide variety of patterns at the sub cluster level. The importance of the second level of clustering is therefore evident.

Figure 6.5 shows that sub clusters 5, 7, 8 and 9 have the closest resemblance to the top level cluster 6 profile. The primary difference is the average electricity consumption with, 128 kWh, 27 kWh, 71 kWh and 9 kWh for sub clusters 5, 7, 8 and 9 respectively. All four clusters have peak electricity consumption between the hours of 18:00 and 21:00. It could be assumed that working households may have such behaviour since peak electricity consumption would occur after the working members return from work. sub clusters 5, 7, 8 and 9 contain 88.5% of the records, which indicates that the majority of these electricity consumers could have such profiles.

Since cluster 9 contains the clear majority of the records, a more detailed examination may reveal electricity consumption patterns of working households more clearly. Figure 6.6 does this by outlining a profile for sub cluster 9 in a smaller scale. It can be seen that the electricity consumption starts to increase around 04:00, which would indicate the time of day some households wake up and start using appliances and lights. Electricity consumption increases until approximately 07:00 when it starts to decline. The reason for the decline could be attributed to the households leaving for work and school. Electricity

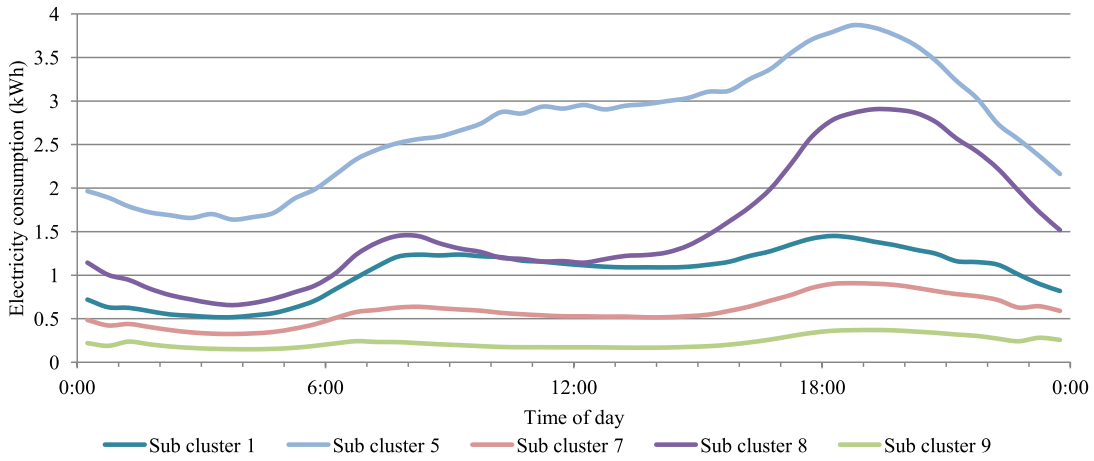


Figure 6.5: Cluster 6: electricity consumption profiles of sub clusters 1, 5, 7, 8 and 9

consumption remains low during the day and starts to rise again, starting at approximately 16:00 which is the time people are returning from school or work. Electricity consumption peaks for the day around 19:00 possibly due to the use of cooking appliances, televisions and computers.

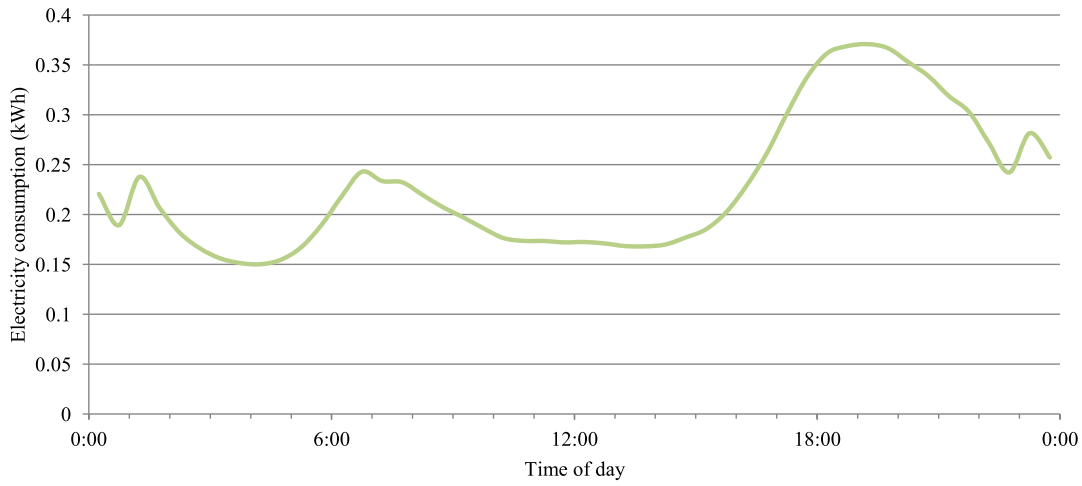


Figure 6.6: Sub cluster 9: more detail on electricity consumption by time of day

It is worth noting, in Figure 6.6, that electricity consumption fluctuates between 22:30 and 02:00. Although insufficient data is available to determine the exact cause, it could be due to electricity consumption by appliances set to a delayed start such as washing machines and dishwashers. Delayed starts could be due to the customer being on a two rate (peak/off peak) plan where energy intensive appliances such as washers, dryers and dishwashers are used during the off peak period when the tariff is low.

6.6.4 Weekly electricity consumption profiles

The Distributed GSOM was run on weekly electricity consumption data in order to generate the weekly electricity consumption profiles. Weekly profiles also indicate the electricity consumption patterns of customers on different days of the week, mainly the differences in electricity consumption on weekdays and weekends. Figure 6.7 shows the weekly electricity consumption profiles identified for the entire dataset.

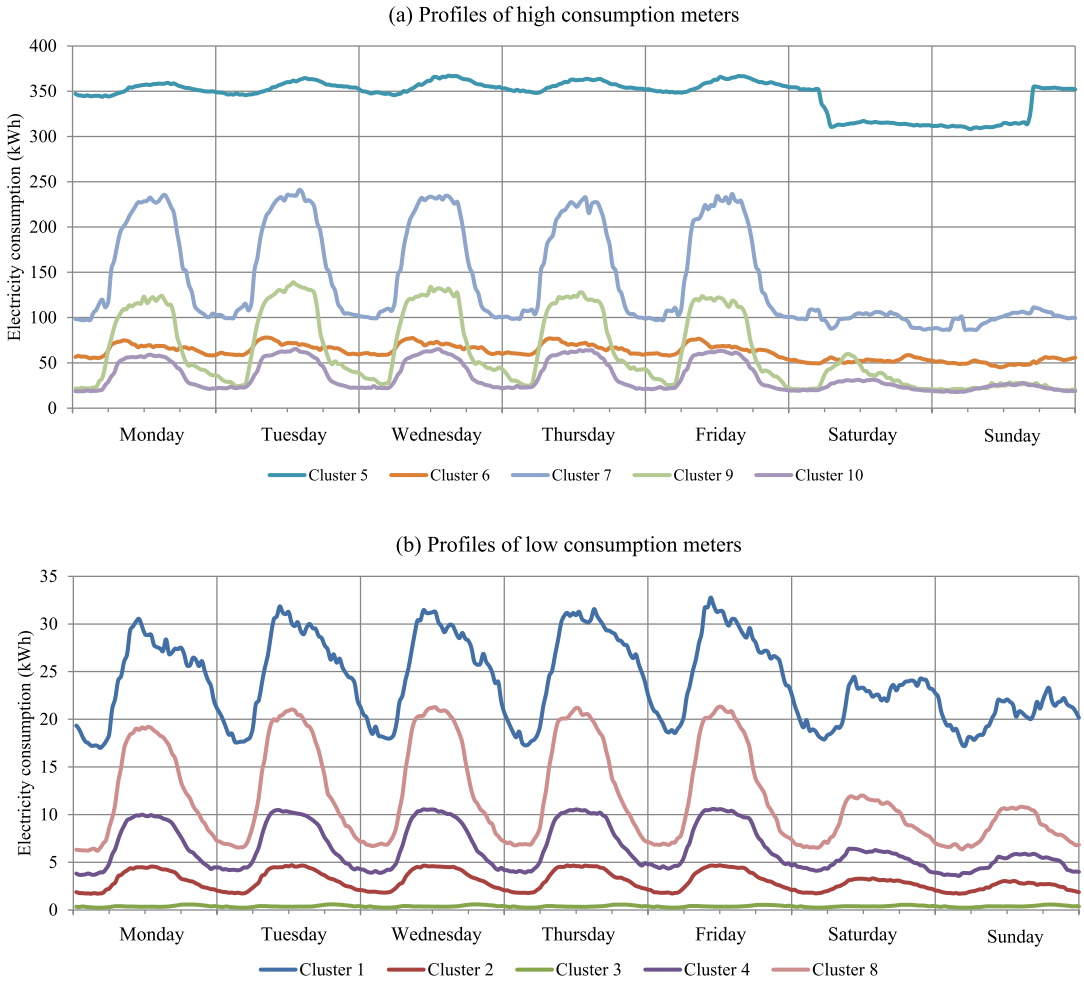


Figure 6.7: Entire dataset: weekly electricity consumption by day of week. (a) Profiles of high consumption meters. (b) Profiles of low consumption meters

It can be seen that weekdays, from Monday to Friday, have similar electricity consumption behaviour while that on Saturday and Sunday changes significantly. Similar to daily electricity consumption profiles, two main types of profiles can be identified where one has a low variance between the maximum and the minimum electricity consumption peaks, as in clusters 5 and 6, whereas the other clusters show varying consumption based on the day of week and time of day.

Table 6.4 summarises the statistics for each cluster. The primary differentiator between the profiles is the average weekly electricity consumption. As cluster 3 contained 90% of the records, a further level of analysis was performed using the Distributed GSOM. The second level sub clusters indicated a result similar to the sub cluster daily profile analysis for the entire dataset.

Table 6.4: Entire dataset: weekly electricity consumption profile statistics

Cluster	Total No. of Records	Average kWh	Percentage of total
1	2182	7512.04 kWh	0.43%
2	28851	922.71 kWh	5.66%
3	461184	144.49 kWh	90.43%
4	11619	2161.91 kWh	2.28%
5	51	132342.30 kWh	0.01%
6	370	22425.77 kWh	0.07%
7	150	45596.58 kWh	0.03%
8	4388	3934.01 kWh	0.86%
9	331	20370.12 kWh	0.06%
10	874	11848.48 kWh	0.17%
Total	510000	374.54 kWh	100.00%

6.6.5 Annual electricity consumption profiles

Clustering the electricity consumption records for the entire year revealed the annual electricity consumption profiles. Annual electricity consumption profiles indicate the variations in electricity consumption for longer time periods such as months and seasons. Figure 6.8 shows the annual electricity consumption profiles identified.

It can be seen that all the clusters have similar electricity consumption patterns throughout the year with high electricity consumption during summer and low consumption during winter. In addition, the weekly electricity consumption behaviour is also visible, with high electricity consumption (the peaks) on weekdays and low consumption (the troughs) on weekends. The main difference between each cluster is primarily the average annual electricity consumption as given in Table 6.5.

Table 6.5 shows that distribution between clusters is uneven with 63% of the records mapped on to cluster 6. Figure 6.9 shows the profile of cluster 6 with maximum and minimum temperature overlay.This reveals a profile where electricity consumption increased,

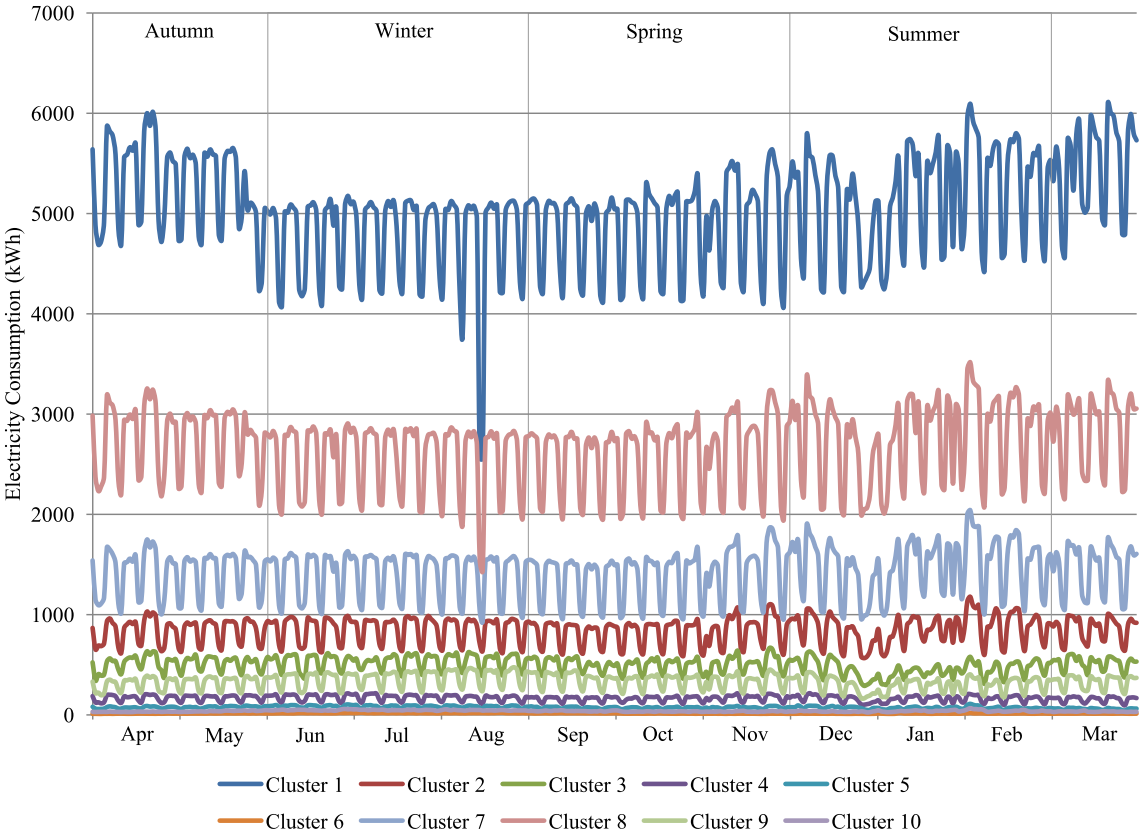


Figure 6.8: Entire dataset: annual electricity consumption profiles by season

Table 6.5: Entire dataset: annual electricity consumption profile statistics by cluster

Cluster	Total No. of records	Average kWh	Percentage of total
1	6	2802051.00 kWh	0.06%
2	50	308143.80 kWh	0.50%
3	75	183168.10 kWh	0.75%
4	359	62662.88 kWh	3.59%
5	641	27021.22 kWh	6.41%
6	6306	4547.90 kWh	63.06%
7	24	525388.70 kWh	0.24%
8	17	958796.80 kWh	0.17%
9	172	115801.10 kWh	1.72%
10	2350	13450.83 kWh	23.50%
Total	10000	19488.89 kWh	100.00%

with a drop in spring and a spike during summer. The maximum and minimum temperatures elaborate the relationship further with a negative correlation with electricity consumption during winter months and a positive correlation with electricity consumption during summer.

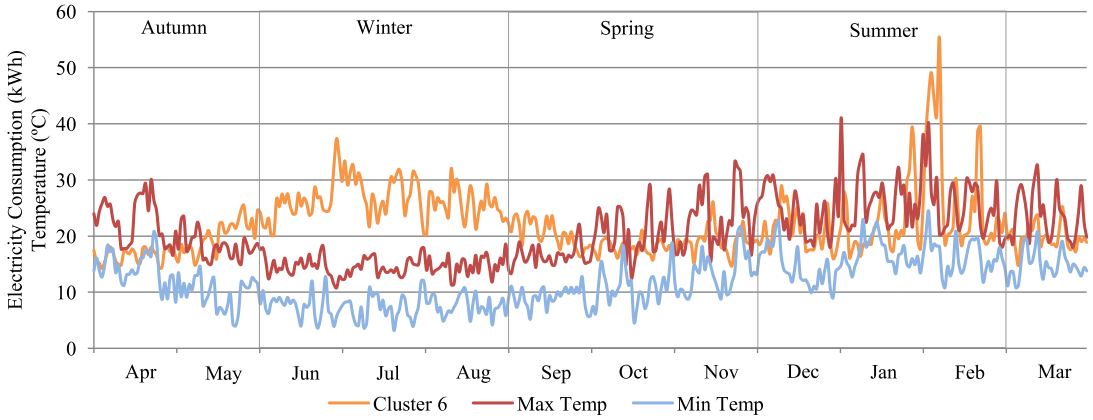


Figure 6.9: Cluster 6 electricity consumption profile by season and month, with temperature (max and min) overlay

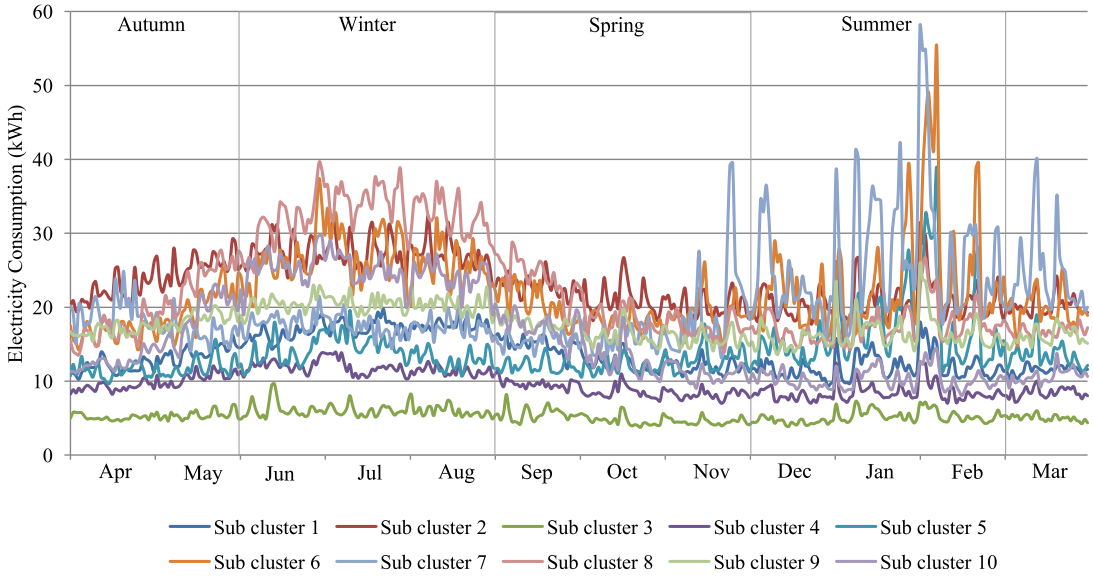


Figure 6.10: Cluster 6: annual electricity consumption profiles of 10 sub clusters

In order to investigate the profiles in cluster 6 further, a second level of clustering was performed on the records in cluster 6. These second level, sub clusters revealed further electricity consumption profiles which are shown in Figure 6.10. The profiles in combination generate the profile of the top level cluster 6 as shown in Figure 6.9.

The sub clusters with the closest profiles to that of the parent cluster are sub clusters 1, 2, 3, 4 and 9, as shown in Figure 6.11. The average electricity consumption differentiates the profiles with 13.50 kWh, 9.49 kWh, 15.77 kWh, 21.73 kWh and 4.65 kWh for profiles 1, 2, 3, 4 and 9, respectively. Considering the maximum and minimum temperature values as shown in Figure 6.9, the profiles in Figure 6.11 indicate high electricity consumption during

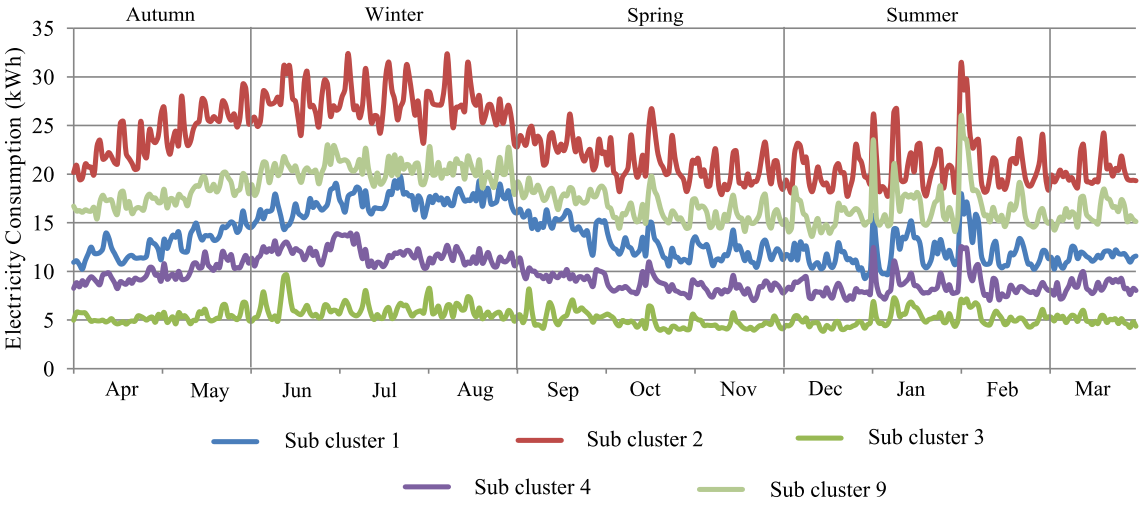


Figure 6.11: Cluster 6: annual electricity consumption profiles of sub clusters 1, 2, 3, 4 and 9

winter and spikes in electricity consumption on high temperature days during summer. This behaviour could be explained by the high electricity consumption during winter due to the use of electric heaters during days with low temperatures coupled with the use of air conditioners on high temperature days during summer.

Figure 6.12 shows the electricity consumption profiles of sub clusters 5 and 7 where electricity consumption displays low electricity usage with minor fluctuations during winter and without any seasonal trend. Electricity consumption is significantly high during summer especially in sub cluster 5 profile where the high electricity consumption trend spans the entire summer. The two types of profiles could be characterised by the electricity consumption of households with no electric heating during winter possibly due to their having gas heating. The high consumption in summer could be due to the use of air conditioning during the summer on high temperature days.

In contrast, sub clusters 8 and 10 display the opposite behaviour to that of sub clusters 5 and 7, with high electricity consumption during the winter and low consumption during summer. The energy consumption profiles for sub clusters 8 and 10 are shown in Figure 6.13. The reason for such behaviour could be due to the increased consumption of electricity as a result of the use of electric heating during winter coupled with absence of using air conditioning during summer. The minor surge in electricity consumption during high temperature days could be attributed to the use of electric fans or evaporative coolers which are less energy intensive compared to air conditioners.

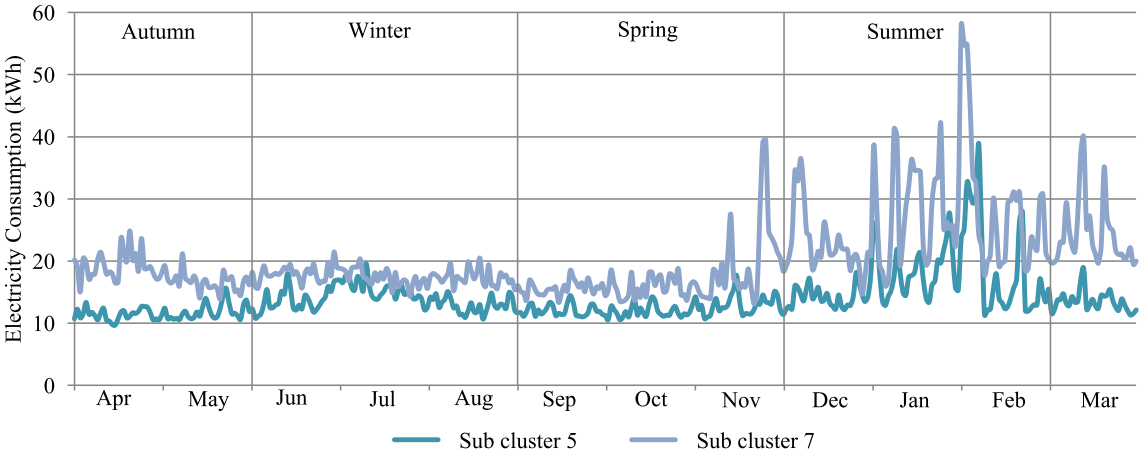


Figure 6.12: Cluster 6: annual electricity consumption profiles of sub clusters 5 and 7

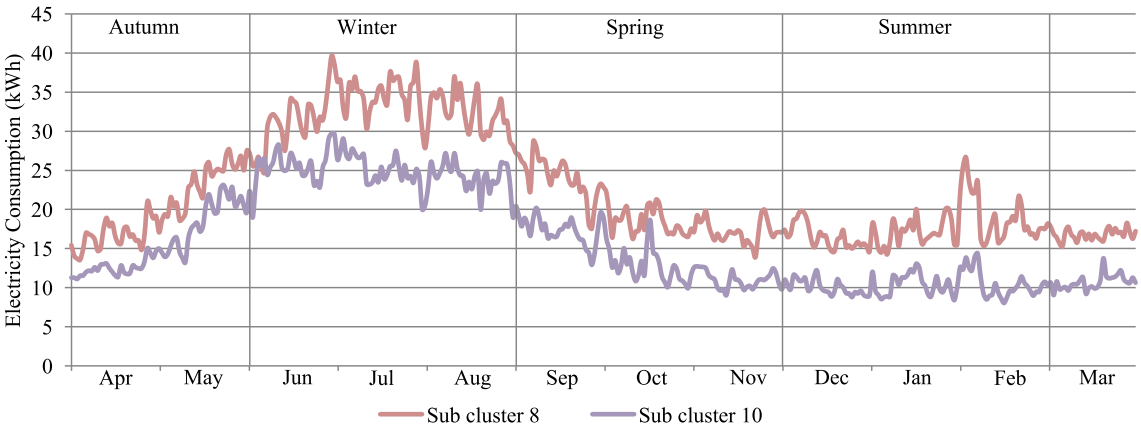


Figure 6.13: Cluster 6: annual electricity consumption profiles of sub clusters 8 and 10

6.7 A Multi-Granular Profile (MGP) analysis framework

Electricity consumption readings made by smart meters form the primary source of data for the electricity consumption profiling. The real-time electricity consumption data recorded by smart meters provide 15 or 30 minute electricity consumption values. Analysing electricity consumption patterns in daily, weekly and annual time intervals enables analysts to understand customers better. However, questions arise as to how daily electricity consumption patterns predict annual electricity consumption and how external factors (weather and socio economic factors) affect electricity consumption behaviour.

To address these concerns, a novel framework is proposed which produces multi-granular profiles which create a complete solution for analysing electricity consumption data. Figure 6.14 shows a diagrammatic representation of the framework.

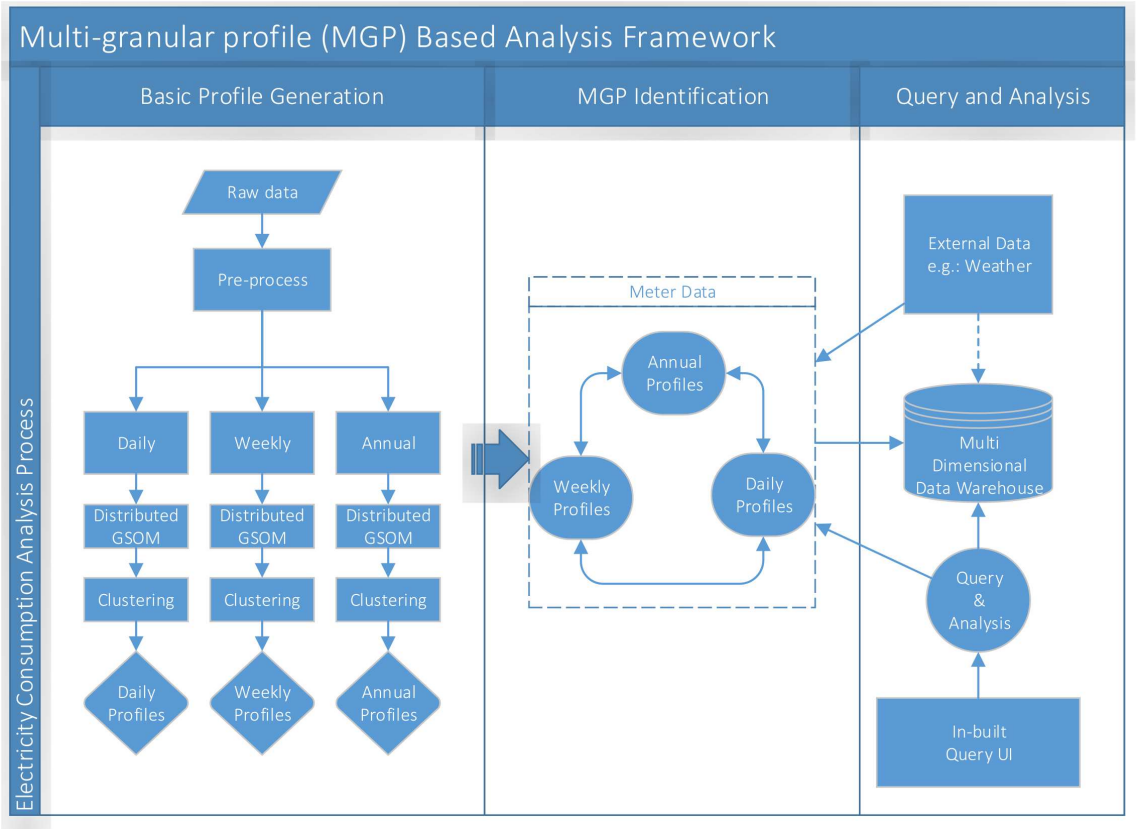


Figure 6.14: The multi granular profile framework

The first stage of the framework generates the daily, weekly and annual electricity consumption profiles. Raw data is transformed into daily, weekly and annual electricity consumption models. The Distributed GSOM is run on each of the three data models to summarise and visualise outcomes for daily, weekly and annual electricity consumption configurations. The output of the Distributed GSOM is clustered to identify the profiles for each configuration.

Once the profiles are generated for daily, weekly and annual electricity consumption, the relationships among the configurations are examined to generate the multi-granular profiles (MGP). The relationship structure is generated by examining the distribution of identifiers in the clusters at each level. For example, if meter A has an annual consumption of type 1, the distribution of meter A records in weekly analysis clusters is investigated. Repeating this process for all meters would reveal the relationship with annual profiles and weekly profiles. A similar approach could be used to generate the relationship between weekly and daily electricity consumption profiles.

The multi-granular profiles can then be used for various types of analysis exercises. A data warehouse could be built to store the MGPs and querying could be performed either

directly on the MGPs or through the data warehouse. External data such as temperature and rainfall could be linked in order to integrate another dimension into the analysis. The querying process could be improved by developing a user interface (UI) to interact with underlying information.

6.7.1 Profile generation

Smart electricity meter data could span several years. Therefore, electricity consumption profiles could cover a wide range of cyclic time spans. Analysing the data with different levels of granularity enables utilities companies to observe and understand consumer behaviour from multiple perspectives.

At the coarsest level of granularity, the profile could span the entire time period which would reveal electricity consumption over a number of years or decades. Having access to long-term electricity consumption patterns would enable utilities companies to identify trends and anticipate long-term implications of the demand for electricity.

Analysing the medium term electricity profiles would indicate the electricity consumption patterns over a period of less than a year. Annual profiles contain information about seasonal and monthly electricity consumption patterns. The predictability of medium term electricity usage patterns would enable utilities companies to plan for the electricity demand according to seasonal variations.

Short-term energy consumption profiles over daily and hourly time intervals would indicate any short-term electricity usage trends. Short-term trends have high relevance to smart grid technologies such as demand side management and time of use pricing. The ability to respond quickly to changing electricity usage trends could lead to more efficient energy generation and utilisation.

Profiles are generated by clustering the energy consumption records by the selected level or granularity. Each cluster would represent an electricity consumption profile showing the electricity consumption pattern for the members of the cluster.

6.7.2 MGP extraction

Multi-granular profiles are created by identifying the relationships between the multiple granularity levels. MGPs pave the basis for exploring the electricity consumption patterns of the consumers from a new perspective. MGPs create a fusion of long-term, medium

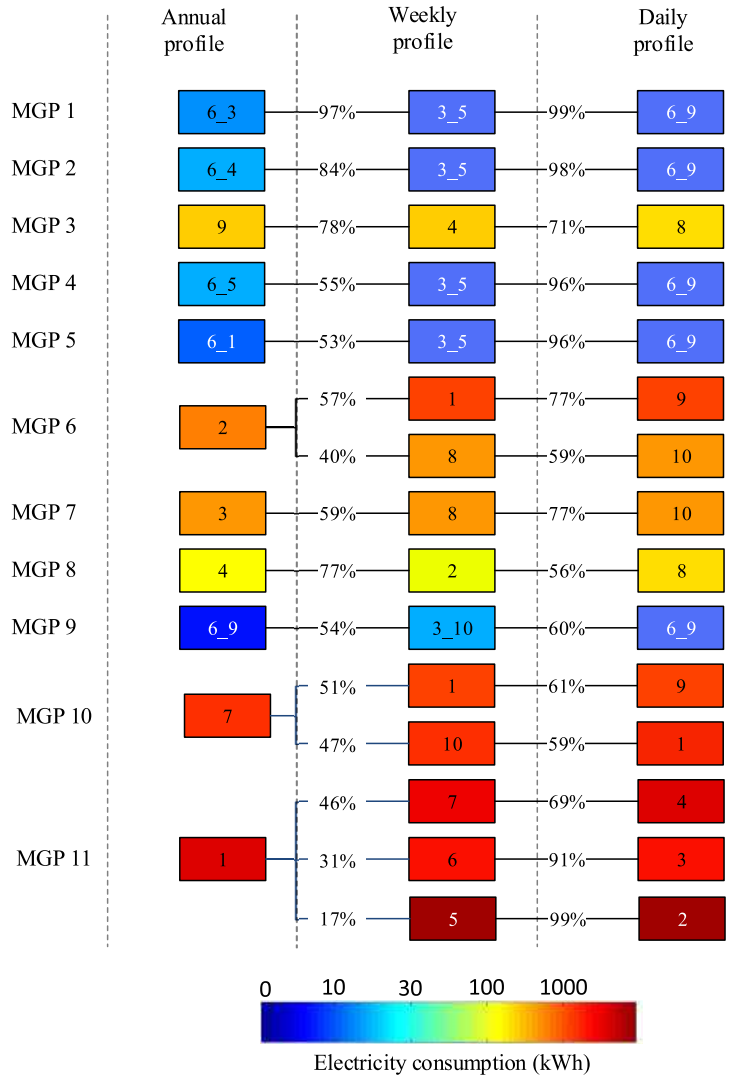


Figure 6.15: Multi-granular profile structure. Profile identifiers are in <top level cluster identifier >[_ <second level cluster identifier >] format.

term and short-term electricity consumption profiles which could be used for effective decision making.

MGPs enable analysts to predict the long-term electricity consumption behaviour of a customer using short-term electricity consumption readings. MGPs also provide a platform for evaluating customers' behavioural changes by analysing the movement of meters from one MGP to another.

6.7.3 Identified multi-granular profiles

The profiles generated for daily, weekly and annual electricity consumption were combined across different granular levels in order to create the MGPs. Figure 6.15 shows the most prominent MGPs.

Table 6.6: MGP categorisation rules

Range in kWh	Classification
> 1000	Ultra high
> 100	Very high
> 30	High
> 10	Average
< 10	Low

The MGPs could be categorised according to average daily electricity consumption. Categorisation rules are given in Table 6.6. Examples from each section are discussed in the following sub sections.

Low electricity consumption MGPs

Figure 6.16 shows the annual, weekly and daily consumption patterns of MGP 1. MGP 1 has an annual profile having high electricity consumption during winter and high consumption during summer with low electricity consumption during autumn and spring. 97% of the meters with such annual profiles displayed high consumption during weekdays and Saturdays with relatively low consumption on Sundays. 99% of the records having this weekly profile displayed a daily profile where the highest electricity consumption occurs at night, approximately at 19:00, with low consumption during the day and moderately high consumption could be observed early morning. It could be inferred from the MGP that the profile relates to working households with low electricity consumption with electric heating and air conditioning facilities.

It can be observed that the daily electricity consumption is repeated over the week irrespective of which day of the week. This information would be taken into consideration for time of use based pricing. In addition, electricity consumption during the winter is relatively low compared to the other seasons of the year. Seasonal discounts could be offered to fully utilise the electricity generation.

6.8 Discussion

The Distributed GSOM based analysis process proposed in this chapter provides an efficient alternative to SOM based electricity consumption profiling. The process further

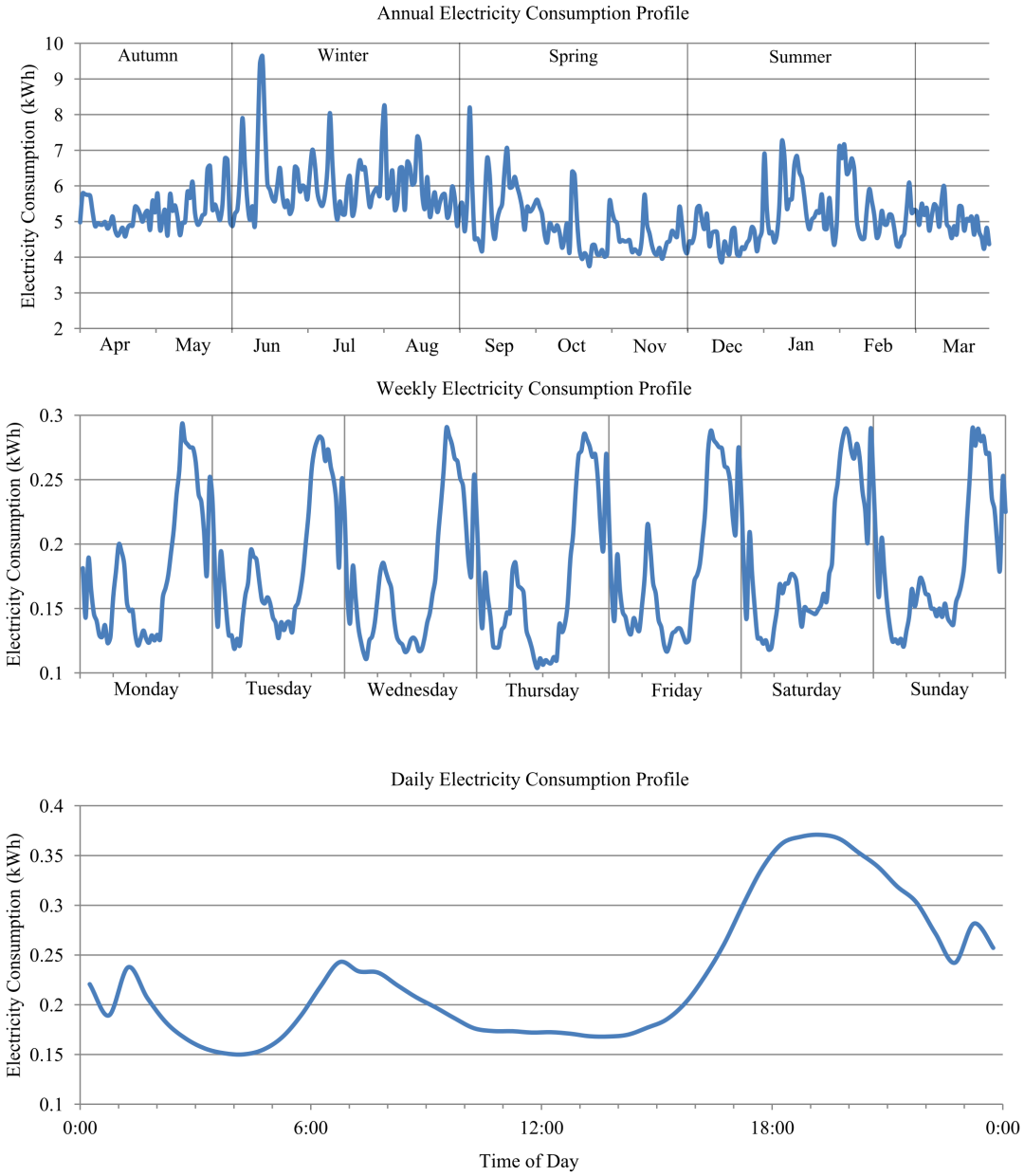


Figure 6.16: Annual, weekly and daily electricity consumption patterns of MGP 1

improves the analysis process by incorporating additional features such as reducing the level of distortion and providing an efficient technique to integrate new data continuously.

The multi-granular profile framework enables analysts to view the profiles in different perspectives in order to link profiles with different levels of granularity. The MGPs can be used to predict long-term electricity consumption behaviour from daily or weekly electricity usage trends.

The data warehouse, external data integration and the query component of the MGP framework needs to be further enhanced by incorporating a graphical user interface for ease of use.

The outcomes of the work described in this chapter are as follows.

1. The Distributed GSOM significantly improved the efficiency of the analysis process by reducing the total time consumption to only 5% compared to that of traditional SOMs.

2. The Distributed GSOM was used effectively to cluster and visualise electricity consumption patterns from gigabytes of smart electricity meter readings. The outliers were included in the analysis using heuristic based partitioning thereby increasing the value of the analysis.

3. Heuristic based partitioning was used to improve clustering accuracy.

4. Average electricity consumption based partitioning was used to group records with similar electricity consumption which resulted in less distortion in clusters.

5. A multi-granular profile structure was developed linking daily, weekly and annual electricity consumption profiles: this can be used to predict electricity consumption behaviour within a practical and reasonable time frame.

6. Harnessing the scalable and the parallel nature of the new algorithm, a model was developed to integrate new electricity consumption data continuously without re-training the entire network.

Chapter 7

Conclusion

This thesis has described a novel distributed algorithm for efficiently exploring large datasets using self-organising maps. The scope of the research was highlighted in Chapter 1 where the research questions and research objectives were discussed. The literature relevant to the work presented in this thesis was presented in Chapter 2. Chapter 3 introduced the Distributed GSOM algorithm and the conceptual as well as the empirical justification for using the GSOM as the learning engine was discussed in Chapter 4. The implementation of the algorithm on the Hadoop distributed computing framework was presented in Chapter 5. A real-world exploratory data analysis application, using the Distributed GSOM to demonstrate the practical applicability of the work, was demonstrated in Chapter 6 where gigabytes of smart electricity meter readings are explored for electricity consumption profiles.

This chapter concludes this thesis by discussing how the research objectives have been achieved.

The primary research question addressed in this thesis was:

> *How can the self-organising map be extended via the functionality of a distributed algorithm to enable exploratory analysis with big data?*

This thesis reports the work conducted in order to answer the primary research question. This chapter describes the summary of the research contributions and outlines the answers to the main research questions. This chapter concludes the thesis by discussing future research directions.

7.1 Summary of contributions

In order to answer the main research question, a distributed algorithm was developed with a divide and conquer architecture. The algorithm, named the Distributed GSOM, encompasses all the desirable features of scalable data and compute-intensive distributed algorithms. The algorithm was evaluated using three datasets to determine its efficiency and accuracy. The Distributed GSOM was implemented on the Hadoop distributed computing framework addressing the practical concerns. The practical usefulness of the algorithm was demonstrated using a real-world exploratory data analysis application.

Summaries of the research contributions presented in each chapter are outlined below.

The field of exploratory data analysis and SOMs was discussed in Chapter 2. The SOM is a widely used exploratory data analysis algorithm due to its visualisation and summarisation features. The GSOM is an extension of the SOM with a dynamic structure which is more suitable for data exploration. A key limitation of both the SOM and the GSOM is their excessive time consumption when processing large scale datasets. Distributed computing is considered to be an effective approach for distributing the computational load for large scale data analysis applications. A number of parallel algorithms have been proposed for the SOM; however, none of the current parallel SOM algorithms possess all the desirable properties of data and compute-intensive distributed algorithms.

The Distributed GSOM, a scalable, parallel algorithm with a distributed memory model was introduced in Chapter 3. The Distributed GSOM possesses all the key features of data and compute-intensive distributed algorithms, a distributed memory model, data parallelism, a horizontal data layout and the ability to process both sparse and dense data. The algorithm utilises a divide and conquer approach by partitioning the dataset, training GSOMs in parallel on each data partition and merging the outputs of the GSOMs to form a singular representation of the entire dataset. Experiments indicated that the Distributed GSOM results in significantly higher levels of efficiency by reducing the total time consumption by several orders compared to the serial counterparts. In addition, the Distributed GSOM created similar levels of clustering accuracy compared to the serial GSOM.

Several key features of the distributed algorithm were further analysed, justified and presented in detail in Chapter 4. Initially, the use of the GSOM in place of the traditional

SOM was analysed and justified conceptually as well as empirically. Next, the redundancy method and incremental data integration features were discussed. Comparison of the effects of static and dynamic structured SOMs demonstrated that, when the shape of the dataset is unknown, the GSOM with a dynamic structure accommodates the data better. Chapter 4 introduced a new redundancy reduction method that improves the scalability of the algorithm. The strengths of the two redundancy reduction methods can be used in different stages of the exploratory data analysis process. Chapter 4 also introduced a new incremental data integration model that reuses the components of the Distributed GSOM algorithm to integrate new data efficiently into an existing network.

Implementation of the Distributed GSOM on the Hadoop distributed computing framework is discussed in Chapter 5. A MapReduce architecture was developed for the Distributed GSOM in order to deploy it in the popular Hadoop distributed computing framework. MapReduce processes were developed for the data partitioning methods which further improve the performance of the Distributed GSOM. The components of the Distributed GSOM were optimised for Hadoop in order to ensure load balancing and minimal communication.

The advantages of the Distributed GSOM were demonstrated using a real-world exploratory data analysis application in Chapter 6. The Distributed GSOM was used to explore gigabytes of smart electricity meter reading data in order to identify electricity consumption profiles. Due to the use of the Distributed GSOM, the total time consumption of the analysis was decreased by 95%, significantly shortening the turnaround time. An incremental, multi-granular electricity consumption profile analysis framework is presented in Chapter 6 which was utilised to identify short-term, medium term and long-term profiles.

7.2 Addressing the main research questions

This section describes the research contribution of this thesis for each research question formulated in Chapter 1.

The main research question, *How can the self-organising map be extended with the functionality of a distributed algorithm to enable exploratory analysis with big data?*, was composed of three sub-questions: questions on distributed SOM algorithm development,

implementation of distributed SOM algorithms on distributed computing platforms and the application of distributed SOM algorithms for real-world exploratory data analysis tasks. Research contributions made for each research question are outlined in the sections below.

7.2.1 Research questions on distributed SOM algorithm development

1. *How can a distributed memory architecture be developed for the SOM with data parallelism and a horizontal data layout?*

   The Distributed GSOM algorithm is proposed in Chapter 3 as a solution to the high time consumption of the SOM algorithm for large scale exploratory data analysis applications. The algorithm utilises data parallelism in order to achieve higher levels of scalability for large scale data processing. Data parallelism is achieved by partitioning the dataset horizontally which preserves the attributed relationships during the training process. As the SOMs or GSOMs are trained in parallel on data partitions, the Distributed GSOM has a distributed memory model. Since the Distributed GSOM does not use the sparsity of the dataset in order to achieve parallelism, both sparse and dense data can be processed by the algorithm.

   Experiments conducted using the WBC, SMH and CoverType datasets show that the Distributed GSOM is up to 100 times faster than the serial algorithm while creating similar levels of accuracy. Hence, the Distributed GSOM is an effective distributed SOM algorithm that can be used for large scale data exploration.

2. *What are the different partitioning methods to create a horizontal data layout?*

   The primary horizontal data partitioning methods are discussed in Chapter 3. These are random partitioning, class based partitioning, structure based partitioning and heuristic based partitioning. Random partitioning does not require any meta information from the data and can be applied for any dataset. Class based partitioning can be used if the underlying classes in the data are known and each partition will contain records pertaining to only one class. If the dataset is structured by an inherent property such as time or geography, partitions can be created based on the structure. If the partitions can be created based on a heuristic property of the dataset, heuristic based partitioning can be used.

Although structure and heuristic based partitioning are specific to data analysis applications, random and class based partitioning can be applied to most datasets. Experiments show that the class based partitioning creates marginally higher levels of accuracy compared to random partitioning. The minor difference in accuracy in the two techniques indicate that random partitioning is a suitable alternative to class based partitioning where class information is unavailable.

3. *How can redundancy be determined in SOMs trained on subsets of data and how can redundant neurons be removed?*

   Redundant neurons arising due to partitioning are identified by comparing quantisation errors of neurons for sets of input vectors. The neuron with the greater error is removed from the partition network. Two redundancy reduction methods are available, one where preserved neurons accumulate the hit items of the removed neurons (introduced in Chapter 3) and the other which considers only the original hit items for error calculation (introduced in Chapter 4). Experiments showed that the method that uses accumulated hit items yields higher levels of accuracy whereas the method that uses original hit items results in faster performance and higher levels of scalability.

   The redundancy index is a measure of redundancy across SOMs which indicates the average attribute distance of redundant neurons. The redundancy index was used to identify redundant non-hit neurons which do not incur a quantisation error.

7.2.2 Research questions on implementing algorithms on distributed computing frameworks

1. *How can the distributed SOM algorithm be transformed into a MapReduce pattern?*

   The Distributed GSOM has an inherent MapReduce structure where the partitioning and parallel network training phases form the map stage and the redundancy reduction and merging processes form the reduce stage. Chapter 5 proposes a MapReduce architecture for the Distributed GSOM. The map processes execute in parallel and distribute the workload among the nodes of the computing cluster. The node assignment algorithm strives to achieve the optimum node assignment by ensuring that all datanodes are occupied prior to assigning a second map task to a datanode. The

MapReduce model was implemented on the Hadoop distributed computing framework, one of the most popular cloud computing platforms in existence. Furthermore, combiners were used to reduce the time consumption of the redundancy reduction stage.

2. *How can partitioning of large datasets utilise a MapReduce algorithm?*

   MapReduce data partitioning modes were proposed in Chapter 5 for the four data partitioning methods proposed in Chapter 3. Random partitioning, class based partitioning and structure based partitioning were performed by assigning each record in the input dataset to the designated partition in the map process and combining the records for each partition in the reduce process. Heuristic based partitioning involved further processing since the dataset may need to be ordered by the heuristic property. The sort and shuffle process of Hadoop was utilised for heuristic based partitioning without the need for an explicit sorting function.

3. *How can data partitioning ensure that each parallel process receives a sufficient number of records?*

   The `CsvRecordReader` class developed for the Distributed GSOM implementation ensured that the records were accurately parsed and a sufficient number of records were assigned for each partition. Each line in the data files was parsed in order to extract the records. The minimum number of records per partition required for SOMs or GSOMs is outlined in Chapter 3. The partitioning process ensured that the number of records assigned for a partition is greater than the required minimum.

4. *What measures can be taken to minimise the communication overhead in a distributed computing platform?*

   Outputs of the partition networks were modified such that the input vectors are encapsulated within the neurons. Having the input vectors within the neurons overcomes the need to access the dataset during redundancy reduction. The reduce process, therefore, can operate independent of the dataset and the volume of communication in the computing cluster is minimised.

7.2.3 Research questions on applying the distributed SOM algorithm onto large datasets

1. *How do static and dynamic SOM structures affect data exploration?*

   The SOM with a static structure resulted in a greater quantisation error compared to the GSOM with a dynamic structure when the dataset does not match the shape of the network, as discussed in Chapter 4. If the shape of the dataset is known, static structures can be used effectively. However, in exploratory data analysis, the shape of the dataset is rarely known. Therefore, GSOMs create a more accurate representation of the dataset compared to the SOM.

   SOMs created higher levels of inconsistency in representing varying density data whereas the level of inconsistency in GSOMs was low. In addition, the GSOM created fewer neurons in order to achieve lower levels of quantisation errors compared to that of the SOM. Therefore, GSOMs created a better representation of the dataset while improving the performance of the merging process.

2. *How can the distributed GSOM algorithm be used to explore large electricity meter readings to identify electricity consumption profiles?*

   The Distributed GSOM algorithm was used to identify electricity consumption profiles from gigabytes of smart electricity meter readings in Chapter 6. A model was proposed to identify profiles of different granularity levels enabling the analysis of the data from multiple perspectives. External factors were integrated into the analysis in order to add more value to the analysis. The time consumption of the analysis process was reduced by 95% by using the distributed algorithm.

3. *How can data partitioning be used to improve the quality of the analysis?*

   Heuristic based partitioning was employed to group the data based on total electricity consumption. The high consumption records within the dataset were confined to one partition thereby ensuring that the partition networks are unaffected by outliers. Due to the absence of distortions created by the outliers, cluster separation was consistent. The clusters revealed clear electricity consumption profiles which can be attributed to characteristics of households.

4. *How can different granularity levels of the analysis add more meaning to the identified electricity consumption profiles?*

   Profiles were identified for short-term, medium term and long-term electricity consumption behaviour. Short-term profiles indicated the electricity consumption for 24 hours and represented daily electricity consumption behaviours. The medium term electricity consumption behaviour was determined using the weekly profiles which showed the change in electricity consumption by days of the week. Annual electricity consumption behaviour was considered long-term and the profiles showed the patterns of electricity consumption in different seasons of the year.

   The short-term, medium term and long-term profiles were combined in order to create multi-granular profiles that characterise the behaviour of consumers at multiple levels of granularity. Prediction of long-term electricity consumption behaviour using short-term electricity consumption data was a key feature of the multi-granular profile analysis.

5. *How can new data be integrated into the distributed SOM output without having to re-train the entire network?*

   An incremental data integration model was presented in Chapter 4, in order to facilitate the incorporation of new knowledge into an existing map with the availability of new data. The incremental model operates by training a GSOM on the new data and integrating the GSOM into an existing network by performing the redundancy reduction and Sammon's projection steps. The results demonstrated that the map generated by incrementally presenting the dataset is almost identical to the map generated by presenting the entire dataset simultaneously.

7.3 Future work directions

This thesis presents a scalable distributed SOM model for increasing the efficiency of the exploratory data analysis process. The work reported in the thesis demonstrates that the proposed Distributed GSOM algorithm is efficient and effective in clustering massive data volumes. Further research would improve the accuracy and applicability of the algorithm for large scale data exploration applications in the modern day.

The map merging process can be improved by developing a parallel architecture for generating the topographic map. A merging stage that can concurrently generate a single map would improve the efficiency of the overall algorithm. In addition, cascading MapReduce models can be implemented on Hadoop resulting in a seamless end to end process.

An application in electricity consumption profiling is presented for the Distributed GSOM. It would be interesting to scale the algorithm to thousands of nodes for processing hundreds of gigabytes or terabytes of data. The incremental data integration can be utilised to create an evolving self-organising map that continuously learns and maintains its currency. A pruning process can be integrated into the architecture to reduce the level of obsolete knowledge within the network.

In conclusion, the work presented opens up many interesting avenues for future research.

7.4 Concluding remarks

The need to analyse massive volumes of data is becoming increasingly common for most organisations. The SOM is a popular, unsupervised data exploration technique. A key issue of the SOM is the high time consumption of the algorithm when processing large datasets. With the computing power of cloud and distributing computing platforms at our disposal, efficient scalable, parallel algorithms are in high demand for data exploration.

This thesis proposes the Distributed GSOM as a means of delivering the data analysis power of SOM based techniques for large scale data analysis. It also demonstrates the application of the algorithm for analysing large real-world datasets. The Distributed GSOM is several orders faster than the SOM and the total time consumption can be reduced by up to 99%.

It is believed that the work reported in this thesis will give rise to a new generation of efficient distributed SOM algorithms for large scale data analysis applications.

Vita

Publications arising from this thesis include:

Ganegedara, H. and Alahakoon, D. (2011), Scalable data clustering: A Sammon's projection based technique for merging GSOMs. In *The 18th International Conference on Neural Information Processing* Shanghai, China

Ganegedara, H. , et al. (2012), Self-organising map based region of interest labelling for automated defect identification in large sewer pipe image collections. In *International Joint Conference on Neural Networks* Brisbane, Australia

Ganegedara, H. and Alahakoon, D. (2012), Redundancy reduction in self-organising map merging for scalable data clustering. In *International Joint Conference on Neural Networks* Brisbane, Australia

Matharage, S. , Ganegedara, H. and Alahakoon, D. (2013), A Scalable and Dynamic Self-Organizing Map for Clustering Large Volumes of Text Data. In *International Joint Conference on Neural Networks* Dallas, USA

Permanent Address: Clayton School of Information Technology

Monash University

Australia

This thesis was typeset with LaTeX 2$_\varepsilon$[1] by the author.

[1] LaTeX 2$_\varepsilon$ is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society. The macros used in formatting this thesis were written by Glenn Maughan and modified by Dean Thompson and David Squire of Monash University.

References

Alahakoon, D., Halgamuge, S. and Srinivasan, B. (2000). Dynamic self-organizing maps with controlled growth for knowledge discovery, *Neural Networks, IEEE Transactions on* **11**(3): 601–614.

Alahakoon, L. D. (2004). Controlling the spread of dynamic self-organising maps, *Neural Computing & Applications* **13**(2): 168–174.

Apache (2013). Hadoop.
URL: *http://hadoop.apache.org/, last accessed 08 February 2013*

Astel, A., Tsakovski, S., Barbieri, P. and Simeonov, V. (2007). Comparison of self-organizing maps classification approach with cluster and principal components analysis for large environmental data sets, *Water Research* **41**(19): 4566–4578.

Badran, I., El-Zayyat, H. and Halasa, G. (2008). Short-term and medium-term load forecasting for jordan's power system, *American Journal of Applied Sciences* **5**(7).

Bakirtzis, A., Theocharis, J., Kiartzis, S. and Satsios, K. (1995). Short term load forecasting using fuzzy neural networks, *Power Systems, IEEE Transactions on* **10**(3): 1518–1524.

Barney, B. (2013). Introduction to parallel computing.

Barreto, G. A. and Araujo, A. F. (2004). Identification and control of dynamical systems using the self-organizing map, *Neural Networks, IEEE Transactions on* **15**(5): 1244–1259.

Behrens, J. T. (1997). Principles and procedures of exploratory data analysis, *Psychological Methods* **2**(2): 131.

Bezdek, J. C., Ehrlich, R. and Full, W. (1984). Fcm: The fuzzy c-means clustering algorithm, *Computers and Geosciences* **10**(2): 191–203.

Blackard, J. A., Dean, D. and Anderson, C. (1998). The forest covertype dataset.

Blackard, J. A. and Dean, D. J. (1999). Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables, *Computers and Electronics in Agriculture* **24**(3): 131–151.

Bryson, A. and Ho, Y.-C. (1969). Applied optimal control. 1969, *Blaisdell, Waltham, Mass* .

Carpenter, G. A., Grossberg, S. and Rosen, D. B. (1991). Fuzzy art: Fast stable learning and categorization of analog patterns by an adaptive resonance system, *Neural networks* **4**(6): 759–771.

Chen, H.-L., Yang, B., Liu, J. and Liu, D.-Y. (2011). A support vector machine classifier with rough set-based feature selection for breast cancer diagnosis, *Expert Systems with Applications* **38**(7): 9014–9022.

Chicco, G., Napoli, R. and Piglione, F. (2003). Application of clustering algorithms and self organising maps to classify electricity customers, *Power Tech Conference Proceedings, 2003 IEEE Bologna*, Vol. 1, IEEE, p. 7 pp. Vol. 1.

Cloud, A. E. C. (2011). Amazon web services, *Retrieved November* **9**: 2011.

Cuadros-Vargas, E. and Romero, R. A. F. (2005). Introduction to the sam-s m* and mam-s m* families, *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, Vol. 5, IEEE, pp. 2966–2970.

Darby, S. (2006). The effectiveness of feedback on energy consumption, *A Review for DEFRA of the Literature on Metering, Billing and direct Displays* **486**: 2006.

Das, D. (2009). Howto hadoop.

URL: *http://trac.nchc.org.tw/cloud/ raw-attachment/Fwiki/HadoopWorkshop/hadoop-assembled.pdf*

De Silva, D., Yu, X., Alahakoon, D. and Holmes, G. (2011a). A data mining framework for electricity consumption analysis from meter data, *Industrial Informatics, IEEE Transactions on* **7**(3): 399–407.

De Silva, D., Yu, X., Alahakoon, D. and Holmes, G. (2011b). Semi-supervised classification of characterized patterns for demand forecasting using smart electricity meters, *Electrical Machines and Systems (ICEMS), 2011 International Conference on*, IEEE, pp. 1–6.

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

Dean, J. and Ghemawat, S. (2010). Mapreduce: a flexible data processing tool, *Communications of the ACM* **53**(1): 72–77.

Deboeck, G. and Kohonen, T. (1998). *Visual explorations in finance: with self-organizing maps*, Vol. 2, Springer London.

Demartines, P. and Hrault, J. (1997). Curvilinear component analysis: A self-organizing neural network for nonlinear mapping of data sets, *Neural Networks, IEEE Transactions on* **8**(1): 148–154.

Dubes, R. C. and Jain, A. K. (1980). Clustering methodologies in exploratory data analysis, *Advances in Computers* **19**(11).

Fang, X., Misra, S., Xue, G. and Yang, D. (2011). Smart gridthe new and improved power grid: a survey.

Faro, A., Giordano, D. and Maiorana, F. (2011). Mining massive datasets by an unsupervised parallel clustering on a grid: Novel algorithms and case study, *Future Generation Computer Systems* **27**(6): 711–724.

Fayyad, U., Piatetsky-Shapiro, G. and Smyth, P. (1996). From data mining to knowledge discovery in databases, *AI magazine* **17**(3): 37.

Fischer, M. J., Su, X. and Yin, Y. (2010). Assigning tasks for efficiency in hadoop, *Proceedings of the 22nd ACM symposium on Parallelism in algorithms and architectures*, ACM, pp. 30–39.

Flexer, A. (1999). *On the use of self-organizing maps for clustering and visualization*, Springer, pp. 80–88.

Fu, T.-c., Chung, F.-l., Ng, V. and Luk, R. (2001). Pattern discovery from stock time series using self-organizing maps, *Workshop Notes of KDD2001 Workshop on Temporal Data Mining*, Citeseer, pp. 26–29.

Furao, S., Ogura, T. and Hasegawa, O. (2007). An enhanced self-organizing incremental neural network for online unsupervised learning, *Neural Networks* **20**(8): 893–903.

Ganegedara, H. and Alahakoon, D. (2012). Redundancy reduction in self-organising map merging for scalable data clustering, *Neural Networks (IJCNN), The 2012 International Joint Conference on*, IEEE, pp. 1–8.

Ganegedara, H., Alahakoon, D., Mashford, J., Paplinski, A., Muller, K. and Deserno, T. M. (2012). Self organising map based region of interest labelling for automated defect identification in large sewer pipe image collections, *Neural Networks (IJCNN), The 2012 International Joint Conference on*, IEEE, pp. 1–8.

Gantz, J. F. and Chute, C. (2008). The diverse and exploding digital universe: An updated forecast of worldwide information growth through 2011, IDC.

Ghemawat, S., Gobioff, H. and Leung, S.-T. (2003). The google file system, *ACM SIGOPS Operating Systems Review*, Vol. 37, ACM, pp. 29–43.

Goil, S., Nagesh, H. and Choudhary, A. (1999). Mafia: Efficient and scalable subspace clustering for very large data sets, *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 443–452.

Gorgonio, F. L. and Costa, J. (2008). Combining parallel self-organizing maps and k-means to cluster distributed data, *Computational Science and Engineering Workshops, 2008. CSEWORKSHOPS'08. 11th IEEE International Conference on*, IEEE, pp. 53–58.

Goto, Y., Yamada, R., Yamamoto, Y., Yokoyama, S. and Ishikawa, H. (2013). Som-based visualization for classifying large-scale sensing data of moonquakes, *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2013 Eighth International Conference on*, IEEE, pp. 630–634.

Govindan, S., Choi, J., Urgaonkar, B., Sivasubramaniam, A. and Baldini, A. (2009). Statistical profiling-based techniques for effective power provisioning in data centers, *Proceedings of the 4th ACM European conference on Computer systems*, ACM, pp. 317–330.

Gray, J. (2007). Tape is dead, disk is tape, flash is disk, ram locality is king, *Gong Show Presentation at CIDR* .

Haritopoulos, M., Yin, H. and Allinson, N. M. (2002). Image denoising using self-organizing map-based nonlinear independent component analysis, *Neural Networks* **15**(8): 1085–1098.

Haykin, S. (1994). *Neural networks: a comprehensive foundation*, Prentice Hall PTR.

Hebb, D. (1949). The organization of behavior, *A Neuropsychological Theory* .

Holub, J., Richardson, T., Dryden, M., La Grotta, S. and Winer, E. (2012). Contextual self-organizing map visualization to improve optimization solution convergence, *Proceedings of the 14th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, AIAA*.

Honkela, T., Kaski, S., Lagus, K. and Kohonen, T. (1997). Websomself-organizing maps of document collections, *Proceedings of WSOM*, Vol. 97, pp. 4–6.

Hubel, D. and Wiesel, T. (1968). Receptive fields and functional architecture of monkey striate cortex, *The Journal of Physiology* **195**(1): 215.

Huntsberger, T. and Ajjimarangsee, P. (1990). Parallel self-organizing feature maps for unsupervised pattern recognition, *International Journal of General Systems* **16**(4): 357–372.

Hush, D. R. and Horne, B. G. (1993). Progress in supervised neural networks, *Signal Processing Magazine, IEEE* **10**(1): 8–39.

IEA, I. E. A. (2012). Key world energy statistics 2013.
URL: *http://www.iea.org/publications/freepublications/publication/KeyWorld2013.pdf*

Intel (2013). 4th generation intel core i7 processor.
URL: *http://ark.intel.com/products/family/75023, last accessed 08 February 2013*

Jardini, J. A., Tahan, C., Gouvea, M., Ahn, S. U. and Figueiredo, F. (2000). Daily load profiles for residential, commercial and industrial low voltage consumers, *Power Delivery, IEEE Transactions on* **15**(1): 375–380.

Jatmiko, W., Azurat, A., Wibowo, A., Marihot, H., Wicaksana, M., Takagawa, I., Sekiyama, K. and Fukuda, T. (2010). Self-organizing urban traffic control architecture with swarm-self organizing map in jakarta: Signal control system and simulator, *International Journal on Smart Sensing and Intelligent Systems* **3**(3).

Jin, J., Luo, J., Song, A., Dong, F. and Xiong, R. (2011). Bar: an efficient data locality driven task scheduling algorithm for cloud computing, *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, IEEE, pp. 295–304.

Kaski, S. (1997). Data exploration using self-organizing maps, *ACTA POLYTECHNICA SCANDINAVICA: MATHEMATICS, COMPUTING AND MANAGEMENT IN ENGINEERING SERIES NO. 82*, Citeseer.

Kaski, S., Kangas, J. and Kohonen, T. (1998). Bibliography of self-organizing map (som) papers: 1981-1997, *Neural computing surveys* **1**(3-4): 1–176.

Keim, D. A., Mansmann, F., Schneidewind, J. and Ziegler, H. (2006). Challenges in visual data analysis, *Information Visualization, 2006. IV 2006. Tenth International Conference on*, IEEE, pp. 9–16.

Kim, K.-S. and Han, I. (2001). The cluster-indexing method for case-based reasoning using self-organizing maps and learning vector quantization for bond rating cases, *Expert Systems with Applications* **21**(3): 147–156.

King, B. (1967). Step-wise clustering procedures, *Journal of the American Statistical Association* **62**(317): 86–101.

Kitajima, N. (1995). A new method for initializing reference vectors in lvq, *Neural Networks, 1995. Proceedings., IEEE International Conference on*, Vol. 5, IEEE, pp. 2775–2779.

Kiviluoto, K. (1998). Predicting bankruptcies with the self-organizing map, *Neurocomputing* **21**(1): 191–201.

Kluver, R. (2008). Globalization, informatization, and intercultural communication.

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

Kohonen, T. (1993). Physiological interpretation of the self-organizing map algorithm, *Neural Networks* **6**(7): 895–905.

Kohonen, T. (2001). *Self-organizing maps*, Vol. 30, Springer.

Kohonen, T., Kaski, S., Lagus, K., Salojarvi, J., Honkela, J., Paatero, V. and Saarela, A. (2000). Self organization of a massive document collection, *Neural Networks, IEEE Transactions on* **11**(3): 574–585.

Kohonen, T., Oja, E., Simula, O., Visa, A. and Kangas, J. (1996). Engineering applications of the self-organizing map, *Proceedings of the IEEE* **84**(10): 1358–1384.

Koikkalainen, P. and Oja, E. (1990). Self-organizing hierarchical feature maps, *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, IEEE, pp. 279–284. Complexity of SOM.

Koskela, T., Varsta, M., Heikkonen, J. and Kaski, K. (1997). *Time series prediction using recurrent SOM with local linear models*, Helsinki University of Technology.

Kouzes, R. T., Anderson, G. A., Elbert, S. T., Gorton, I. and Gracio, D. K. (2009). The changing paradigm of data-intensive computing, *Computer* **42**(1): 26–34.

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

Laaksonen, J., Koskela, M., Laakso, S. and Oja, E. (2000). Picsomcontent-based image retrieval with self-organizing maps, *Pattern Recognition Letters* **21**(13): 1199–1207.

Lam, J. C., Tang, H. and Li, D. H. (2008). Seasonal variations in residential and commercial sector electricity consumption in hong kong, *Energy* **33**(3): 513–523.

Lawrence, R., Almasi, G. and Rushmeier, H. (1999). A scalable parallel algorithm for self-organizing maps with applications to sparse data mining problems, *Data Mining and Knowledge Discovery* **3**(2): 171–195.

Lendasse, A., Lee, J., Wertz, V. and Verleysen, M. (2002). Forecasting electricity consumption using nonlinear projection and self-organizing maps, *Neurocomputing* **48**(1): 299–311.

Liu, Y., Li, M., Alham, N. K., Hammoud, S. and Ponraj, M. (2011). Load balancing in mapreduce environments for data intensive applications, *Fuzzy Systems and Knowledge Discovery (FSKD), 2011 Eighth International Conference on*, Vol. 4, IEEE, pp. 2675–2678.

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations, *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Vol. 1, California, USA, p. 14. K-means.

Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C. and Byers, A. H. (2011). Big data: The next frontier for innovation, competition, and productivity.

Matharage, S., Ganegedara, H. and Alahakoon, D. (2013). A scalable and dynamic self-organizing map for clustering large volumes of text data, *Neural Networks (IJCNN), The 2013 International Joint Conference on*, IEEE, pp. 1–8.

Monteith, J. Y., McGregor, J. D. and Ingram, J. (2013). Hadoop and its evolving ecosystem, *Proceedings of the Fifth International Workshop on Software Ecosystems*.

Mulier, F. and Cherkassky, V. (1995). Self-organization as an iterative kernel smoothing process, *Neural Computation* **7**(6): 1165–1177.

Mundkur, P., Tuulos, V. and Flatow, J. (2011). Disco: a computing platform for large-scale data analytics, *Proceedings of the 10th ACM SIGPLAN workshop on Erlang*, ACM, pp. 84–89.

Nair, S. and Mehta, J. (2011). Clustering with apache hadoop, *Proceedings of the International Conference & Workshop on Emerging Trends in Technology*, ACM, pp. 505–509.

Nickolls, J., Buck, I., Garland, M. and Skadron, K. (2008). Scalable parallel programming with cuda, *Queue* **6**(2): 40–53.

Nurnberger, A. and Detyniecki, M. (2006). Externally growing self-organizing maps and its application to e-mail database visualization and exploration, *Applied Soft Computing* **6**(4): 357–371.

Obermayer, K. and Sejnowski, T. J. (2001). *Self-organizing map formation: foundations of neural computation*, Vol. 93, The MIT Press.

Obradovic, Z. and Vucetic, S. (2004). Challenges in scientific data mining: Heterogeneous, biased, and large samples, *Technical report*, Citeseer.

O'Connor, M. and Herlocker, J. (1999). Clustering items for collaborative filtering, *Proceedings of the ACM SIGIR workshop on recommender systems*, Vol. 128, Citeseer.

Oja, M., Kaski, S. and Kohonen, T. (2003). Bibliography of self-organizing map (som) papers: 1998-2001 addendum, *Neural computing surveys* **3**(1): 1–156.

O'Malley, O. and Murthy, A. C. (2009). Winning a 60 second dash with a yellow elephant, *Proceedings of sort benchmark* .

Ontrup, J. and Ritter, H. (2006). Large-scale data exploration with the hierarchically growing hyperbolic som, *Neural networks* **19**(6-7): 751–761.

Origin (2014). Origin energy - who we are.
URL: *http://www.originenergy.com.au/1758/Who-we-are, last accessed 18 February 2014*

Owen, S., Anil, R., Dunning, T. and Friedman, E. (2011). *Mahout in action*, Manning.

Paatero, J. V. and Lund, P. D. (2006). A model for generating household electricity load profiles, *International journal of energy research* **30**(5): 273–290.

Palensky, P. and Dietrich, D. (2011). Demand side management: Demand response, intelligent energy systems, and smart loads, *Industrial Informatics, IEEE Transactions on* **7**(3): 381–388.

Pantazi, S., Kagolovsky, Y. and Moehr, J. R. (2002). Cluster analysis of wisconsin breast cancer dataset using self-organizing maps, *Studies in health technology and informatics* pp. 431–436.

Pilly, P. K. and Grossberg, S. (2013). Spiking neurons in a hierarchical self-organizing map model can learn to develop spatial and temporal properties of entorhinal grid cells and hippocampal place cells, *PloS one* **8**(4): e60599.

Prudent, Y. and Ennaji, A. (2005). A new learning algorithm for incremental self-organizing maps, *ESANN*, Citeseer, pp. 7–12.

Qiao, J.-f. and Han, H.-g. (2010). An adaptive fuzzy neural network based on self-organizing map (som), *InTech* .

Ramanathan, K., Shi, L. and Chong, T. (2010). A hubel weisel model for hierarchical representation of concepts in textual documents, *The Annual Congress of the Cognitive Science Society*.

Rauber, A., Merkl, D. and Dittenbach, M. (2002). The growing hierarchical self-organizing map: exploratory analysis of high-dimensional data, *Neural Networks, IEEE Transactions on* **13**(6): 1331–1341.

Rauber, A., Tomsich, P. and Merkl, D. (2000). parsom: A parallel implementation of the self-organizing map exploiting cache effects: making the som fit for interactive high-performance data analysis, *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, Vol. 6, IEEE, pp. 177–182.

Redkar, T. and Guidici, T. (2011). *Windows Azure Platform*, Apress.

Roussinov, D. and Chen, H. (1998). A scalable self-organizing map algorithm for textual classification: A neural network approach to thesaurus generation, *Communication Cognition and Artificial Intelligence* **15**(1-2): 81111.

Rusitschka, S., Eger, K. and Gerdes, C. (2010). Smart grid data cloud: A model for utilizing cloud computing in the smart grid domain, *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on*, IEEE, pp. 483–488.

Ruspini, E. H. (1969). A new approach to clustering, *Information and control* **15**(1): 22–32.

Sammon Jr, J. (1969). A nonlinear mapping for data structure analysis, *Computers, IEEE Transactions on* **100**(5): 401–409.

Shargal, M. and Houseman, D. (2009). The big picture of your coming smart grid, *Smart Grid News* **5**.

Sirosh, J. and Miikkulainen, R. (1997). Topographic receptive fields and patterned lateral interaction in a self-organizing model of the primary visual cortex, *Neural Computation* **9**(3): 577–594.

Sneath, P. H. and Sokal, R. R. (1973). *Numerical taxonomy. The principles and practice of numerical classification.*

Song, K.-B., Baek, Y.-S., Hong, D. H. and Jang, G. (2005). Short-term load forecasting for the holidays using fuzzy linear regression method, *Power Systems, IEEE Transactions on* **20**(1): 96–101.

Srinivasan, D., Tan, S. S., Cheng, C. and Chan, E. K. (1999). Parallel neural network-fuzzy expert system strategy for short-term load forecasting: system implementation and performance evaluation, *Power Systems, IEEE Transactions on* **14**(3): 1100–1106.

Strehl, A., Ghosh, J. and Mooney, R. (2000). Impact of similarity measures on web-page clustering, *Workshop on Artificial Intelligence for Web Search (AAAI 2000)*, pp. 58–64.

Stuckenschmidt, H. and Klein, M. (2004). *Structure-based partitioning of large concept hierarchies*, Springer, pp. 289–303.

Su, M.-C. and Chang, H.-T. (2000). Fast self-organizing feature map algorithm, *Neural Networks, IEEE Transactions on* **11**(3): 721–733.

Su, M. C., Liu, T. K. and Chang, H.-T. (2002). Improving the self-organizing feature map algorithm using an efficient initialization scheme, *Tamkang Journal of Science and Engineering* **5**(1): 35–48.

Sutter, H. (2005). The free lunch is over: A fundamental turn toward concurrency in software, *Dr. Dobbs Journal* **30**(3): 202–210.

Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Zhang, N., Antony, S., Liu, H. and Murthy, R. (2010). Hive-a petabyte scale data warehouse using hadoop, *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, IEEE, pp. 996–1005.

Tian, C., Zhou, H., He, Y. and Zha, L. (2009). A dynamic mapreduce scheduler for heterogeneous workloads, *Grid and Cooperative Computing, 2009. GCC'09. Eighth International Conference on*, IEEE, pp. 218–224.

Toivanen, P. J., Ansamki, J., Parkkinen, J. and Mielikinen, J. (2003). Edge detection in multispectral images using the self-organizing map, *Pattern Recognition Letters* **24**(16): 2987–2994.

Tomsich, P., Rauber, A. and Merkl, D. (2000). Optimizing the parsom neural network implementation for data mining with distributed memory systems and cluster computing, *Database and Expert Systems Applications, 2000. Proceedings. 11th International Workshop on*, IEEE, pp. 661–665.

Trnen, P., Kolehmainen, M., Wong, G. and Castrn, E. (1999). Analysis of gene expression data using self-organizing maps, *FEBS letters* **451**(2): 142–146.

Tukey, J. W. (1977). Exploratory data analysis, *Reading, Ma* **231**.

Ultsch, A. (2003). *U*-matrix: a tool to visualize clusters in high dimensional data*, Fachbereich Mathematik und Informatik.

Ultsch, A. and Siemon, H. (1990). Kohonen's self organizing feature maps for exploratory data analysis.

Ultsch, A., Vetter, C. and Vetter, C. (1995). *Self-Organizing-Feature-Maps versus statistical clustering methods: a benchmark*, Fachbereich Mathematik.

Valero, S., Ortiz, M., Senabre, C., Alvarez, C., Franco, F. and Gabaldon, A. (2007). Methods for customer and demand response policies selection in new electricity markets, *Generation, Transmission and Distribution, IET* **1**(1): 104–110.

Verd, S. V., Garcia, M. O., Senabre, C., Marn, A. G. and Franco, F. J. G. (2006). Classification, filtering, and identification of electrical customer load patterns through the use of self-organizing maps, *Power Systems, IEEE Transactions on* **21**(4): 1672–1682.

Vesanto, J. (1999). Som-based data visualization methods, *Intelligent data analysis* **3**(2): 111–126.

Vesanto, J. and Alhoniemi, E. (2000). Clustering of the self-organizing map, *Neural Networks, IEEE Transactions on* **11**(3): 586–600.

Villars, R. L., Olofson, C. W. and Eastwood, M. (2011). Big data: What it is and why you should care, *White Paper, IDC* .

Walter, J. A. and Schulten, K. (1993). Implementation of self-organizing neural networks for visuo-motor control of an industrial robot, *Neural Networks, IEEE Transactions on* **4**(1): 86–96.

Wang, J.-Y. (2009). Data mining analysis(breast-cancer data).

Ward Jr, J. H. (1963). Hierarchical grouping to optimize an objective function, *Journal of the American statistical association* **58**(301): 236–244.

Weichel, C. (2010). Adapting self-organizing maps to the mapreduce programming paradigm, *STeP*, pp. 119–131.

White, T. (2012). *Hadoop: the definitive guide*, O'Reilly.

Wilson, D. R. and Martinez, T. R. (1997). Improved heterogeneous distance functions, *arXiv preprint cs/9701101* .

Witten, I. H. and Frank, E. (2005). *Data Mining: Practical machine learning tools and techniques*, Morgan Kaufmann.

Wolberg, W. H., Street, W. N. and Mangasarian, O. L. (1992). Breast cancer wisconsin (diagnostic) data set, *UCI Machine Learning Repository [http://archive. ics. uci. edu/ml/]* .

Wold, S., Esbensen, K. and Geladi, P. (1987). Principal component analysis, *Chemometrics and intelligent laboratory systems* **2**(1): 37–52.

Wood, G. and Newborough, M. (2003). Dynamic energy-consumption indicators for domestic appliances: environment, behaviour and design, *Energy and Buildings* **35**(8): 821–841.

Wu, S. and Chow, T. W. (2004). Clustering of the self-organizing map using a clustering validity index based on inter-cluster and intra-cluster density, *Pattern Recognition* **37**(2): 175–188.

Xie, J., Yin, S., Ruan, X., Ding, Z., Tian, Y., Majors, J., Manzanares, A. and Qin, X. (2010). Improving mapreduce performance through data placement in heterogeneous hadoop clusters, *Parallel and Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, IEEE, pp. 1–9.

Yang, M. and Ahuja, N. (1999). A data partition method for parallel self-organizing map, Vol. 3, IEEE, pp. 1929–1933 vol. 3.

Yao, R. and Steemers, K. (2005). A method of formulating energy load profile for domestic buildings in the uk, *Energy and Buildings* **37**(6): 663–671.

Yoo, R. M., Romano, A. and Kozyrakis, C. (2009). Phoenix rebirth: Scalable mapreduce on a large-scale shared-memory system, *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, IEEE, pp. 198–207.

Zaki, M. J. (2000). *Parallel and distributed data mining: An introduction*, Springer, pp. 1–23.

Zawodny, J. (2008). Yahoo! launches worlds largest hadoop production application, *Yahoo! Developer Network Blog* .

Zhai, Y., Hsu, A. and Halgamuge, S. (2006). Scalable dynamic self-organising maps for mining massive textual data, Springer, pp. 260–267.

Zhang, Q., Cheng, L. and Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges, *Journal of Internet Services and Applications* **1**(1): 7–18.

Zhao, W., Ma, H. and He, Q. (2009). *Parallel k-means clustering based on mapreduce*, Springer, pp. 674–679.

Zhongwen, L., Hongzhi, L., Zhengping, Y. and Xincai, W. (2005). Self-organizing maps computing on graphic process unit.

Zikopoulos, P. and Eaton, C. (2011). *Understanding big data: Analytics for enterprise class hadoop and streaming data*, McGraw-Hill Osborne Media.