Université Paris 13

Laboratoire d'Informatique de Paris-Nord (LIPN)

Thèse

présentée par

Gaël Beck

pour obtenir le grade de

Docteur d'université

Spécialité : Informatique

Scalable Clustering Applying Local Accretions

Accrétions Locales appliquées au Clustering Scalable et Distribué

Soutenue publiquement le 14 octobre 2019

devant le jury composé de

Abstract

Gaël Beck

Scalable Clustering Applying Local Accretions

Accrétions Locales appliquées au Clustering Scalable et Distribué

This thesis focuses on methods allowing to tackle complexity problem of specific algorithms in order to deal with Big Data. It presents well known algorithms and new ones from various machine learning fields (unsupervised and supervised learning), which use modern algorithms as the Locality Sensitive Hashing to decrease efficiently the algorithmic complexity.

In the first part, we study the problem of scalable clustering algorithm based on Mean Shift algorithm for continuous features. We propose a new design for the Mean Shift clustering using locality sensitive hashing and distributed system. Its variation for categorical features is also proposed based on binary coding and Hamming distance.

In the second part, we introduce scalable Clusterwise method, which is a combination of clustering algorithm and PLS regression. The issue is to find clusters of entities such that the overall sum of squared errors from regressions performed over these clusters is minimized, where each cluster may have a different variance. We improve its time duration and scalability by applying clustering before the regression task. We investigate also in this part of the thesis a feature selection field. We present two efficient distributed algorithms based on Rough Set Theory for large-scale data pre-processing under the Spark framework. The first approach (Sp-RST) splits the given dataset into partitions with smaller numbers of features which are then processed in parallel. The second proposition LSH-dRST use locality sensitive hashing as clustering method to determine appropriate partitions of the feature set.

In the last part, we propose to share as an open source project. This project titled Clustering4Ever offers the possibility to anyone to read the source code and test the different algorithms either via notebooks or calling directly the API. The design enables the generation of algorithms working for many types of data.

Keywords : Clustering, LSH, Mean Shift, Scalability, Distributed Systems

Résumé

Cette thèse porte sur les méthodes dédiées à la manipulation des données massives. Nous présentons de nouveaux algorithmes dans le domaine de l'apprentissage automatique en utilisant des techniques de hashage tel que le Locality Sensitive Hashing (LSH) pour permettre un passage à l'échelle des algorithmes en réduisant leur complexité.

Dans la première partie, nous étudions le problème du passage à l'échelle d'algorithmes de clustering inspirés du Mean Shift pour les données vectorielles continues. Nous proposons un nouvel algorithme utilisant un système de hachage (LSH) tout en bénéficiant du récent paradigme MapReduce appliqué aux systèmes distribués. Nous présentons également la variante de l'algorithme de clustering pour les données catégorielles en utilisant le codage binaire et la distance de Hamming.

Dans la deuxième partie, nous introduisons une amélioration du Clusterwise, qui est une combinaison de l'algorithme de clustering et de la régression. Nous proposons une amélioration de sa complexité en temps d'exécution en appliquant le clustering avant une tâche de régression PLS. Dans cette partie, nous avons étudié le problème de passage à l'échelle dans le domaine de la sélection de variables. Nous présentons deux algorithmes distribués efficaces basés sur la théorie des ensembles pour le prétraitement de données à grande échelle avec le framework Spark.

Dans la dernière partie, nous proposons de partager sous forme d'un projet open source les travaux réalisés. Ce projet intitulé *Clustering4Ever* offre la possibilité d'accèder au code source et de tester les différents algorithmes.

Mots clés : Clustering, LSH, Mean Shift, Scalabilité, Systèmes distribués

Publications

Journals

- 2016. Duong Tarn, Beck Gael, Azzag Hanene, Lebbah Mustapha. "Nearest neighbour estimators of density derivatives, with application to mean shift clustering". Pattern Recognition Letters. DOI
- 2018. Beck Gaël, Azzag Hanane, Bougeard Stéphanie, Lebbah Mustapha, Niang Ndèye, "A New Micro-Batch Approach for Partial Least Square Clusterwise Regression". Procedia Computer Science. Volume 144, Pages 239-250, DOI
- 2019. Beck Gaël, Duong Tarn, Lebbah Mustapha, Azzag Hanane, Cérin Christophe, A (Accepted) "Distributed and Approximated Nearest Neighbors Algorithm for an Efficient Large Scale Mean Shift Clustering". Journal of Parallel and Distributed Computing. arXiv

Conferences

- 2016. Beck Gaël, Duong Tarn, Azzag Hanane and Lebbah Mustapha, "Distributed mean shift clustering with approximate nearest neighbours". International Joint Conference on Neural Networks (IJCNN). Vancouver, BC, 2016, pp. 3110-3115.
  doi: 10.1109/IJCNN.2016.7727595, ResearchGate
- 2017. Beck Gaël, Hanane Azzag, Mustapha Lebbah et Tarn Duong. "Meanshift : Clustering scalable avec les plus proches voisins approximés". Atelier Fouille de Données Complexes @ EGC 2017.7.
- 2017. Chelly Dagdia Zaineb, Zarges Christine, Beck Gaël, Lebbah Mustapha. "A distributed rough set theory based algorithm for an efficient big data preprocessing under the spark framework". BigData 2017: 911-916.
- 2018. Beck Gaël, Duong Tarn, Lebbah Mustapha, Azzag Hanane. "Nearest Neighbor Median Shift Clustering for Binary Data". arXiv
- 2018. Beck Gaël, Azzag Hanane, Lebbah Mustapha, Duong Tarn. "Meanshift : Clustering scalable et distribué". pp.415-425. EGC 2018.
- 2018. Beck Gaël, Azzag Hanane, Bougeard Stéphanie, Lebbah Mustapha, Niang Ndèye, "A New Micro-Batch Approach for Partial Least Square Clusterwise Regression". 3rd INNS Conference on Big Data and Deep Learning (INNS BDDL)
- 2018. Chelly Dagdia Zaineb, Zarges Christine, Beck Gaël, Azzag Hanene, and Lebbah Mustapha. "A Distributed Rough Set Theory Algorithm based on Locality Sensitive Hashing for an Efficient Big Data Pre-processing". IEEE International Conference on Big Data 2018. Dec 10-13 Seattle, WA, USA.

- 2019. Andriantsiory Dina Faneva, Mustapha Lebbah, Hanane Azzag, Gaël Beck. "Algorithms for an Efficient Tensor Biclustering". PAKDD 2019 Workshop. arXiv

Acknowledgements

Over the last three years, I passionately dedicated time and energy diving into the mesmerizing topic of Artificial Intelligence. This part of my personal history can be resumed in one word : data, as each piece of data it is composed of as many features of various inner complexity as we can imagine. This data is a fragment of our universe's information, and it can be visualized by any form of intelligence through the prism of its history, which can be reduced to such properties as the species to which it belongs and its personality. During my studies I learned many interesting concepts, one of which is the nature of features : they can be real numbers or categorical values, sometimes ordered, sometimes unordered. As always with Science, the study of concepts often raises many philosophical questions. Just as with unordered categorical features, I could never rank the people with whom I have shared such good times. I wont even take the risk of naming you all or including private jokes, there are just too many. Thus I wish to thank you all very much for having been, for being, and for becoming what you will forever be, yourselves. Among the many categories of human values that can be ordered, I will always remember each of you as a subtle combination of love, kindness, fun, curiosity, patience, rigor, soft stupidity, humor, sharpness, and so many ineffable others... I deeply know that when you read these lines you will remember the feelings we shared when we were together in good and bad times. I will make a little exception to thank and say how much I love you, my dear parents, for having given me your best at all times, with some mistakes, lets be realistic, we are only human after all... The most common type of data features are real numbers. They describe something with potentially infinite precision and size, because whereas some things can be white or black, others need to be expressed with infinite gray nuances. This is what makes life fun in my opinion, the infinite combinations of dimensions, physical and/or spiritual on which we feel and act in each of our moments. To conclude I would like to use a concept used by many algorithms, the notion of distance, or dissimilarity, more precisely. The closest two things are, the more the value tends to 0. The idea is that two similar objects can be at distance 0 from each other if and only if we consider their common features when measuring their distance. Thus you may not be close to me right now in our 3D euclidean space, you are between a few meters and thousands of kilometers away. However I know and I will never let anyone tell me otherwise, that in the love space, the distance separating us will always tend to 0 as long as we take care of each other.

♡

With all my love
Gaël
KyBe
TremaiDayo

Contents

List of Figures

List of Tables

To the growing cluster of people
that i ♥ 4e

Chapter 1

Introduction

1.1 Context

In our digital era the amount of data has increased exponentially in the last few decades. In the same time many algorithms have been devised to solve a wide range of problems covering various topics such as unsupervised and supervised learning. Dataset size causes serious scalabality problems and exceeds the complexity threshold of many algorithms. To overcome these issues, some heuristics have to be developed to handle what is refereed to Big Data. Most of the contributions achieved in this thesis are focused on scalability. The axes addressed in this thesis are indicated in cyan on Figure 1.1.

Due to the exponential increase of the size of datasets collected to tackle increasingly complex scientific challenges, scalable versions of machine learning methods have become critical. The complexity of machine learning implies that improvements in distributed hardware are insufficient on their own to provide scalable machine learning. In order to optimally exploit the hardware, their mathematical foundations must also be re-formulated with scalability as a priority. Furthermore, existing programming paradigms for dealing with parallelism, such as MapReduce [74] and Message Passing Interface (MPI) [25], have been demonstrated to be the best practical choices for implementing these algorithms. MapReduce allows an unexperienced programmer to develop parallel programs and create a program capable of using computers in a cloud. Indeed the MapReduce paradigm has become popular since data are stored on a distributed file system, which offers data replication, as well as for its ability to execute computations locally on each data node.

The machine learning world is vast and can be divided in different subfields presented in Figure 1.1, we will focus here on two major learning problematics which are unsupervised and supervised learning. Unsupervised learning consist in treating data without any idea about its nature, it often relies on the concept of distance. Two of the most well known unsupervised learning categories are dimensions reduction and clustering. Dimensions reduction objective as its name suggest it is to diminish the dimensionality of a dataset whilst keeping maximum useful information. It is often linked to the curse of dimensionality because it enables to switch from a high dimensional space where most distance are ineffective to a space with much less dimensions which enhance usual applications. One of the most famous technique in this area is the Principal Component Analysis. Clustering consist to assign a cluster membership to unlabeled candidate points where the number and location of these clusters are unknown. Usually clusters are

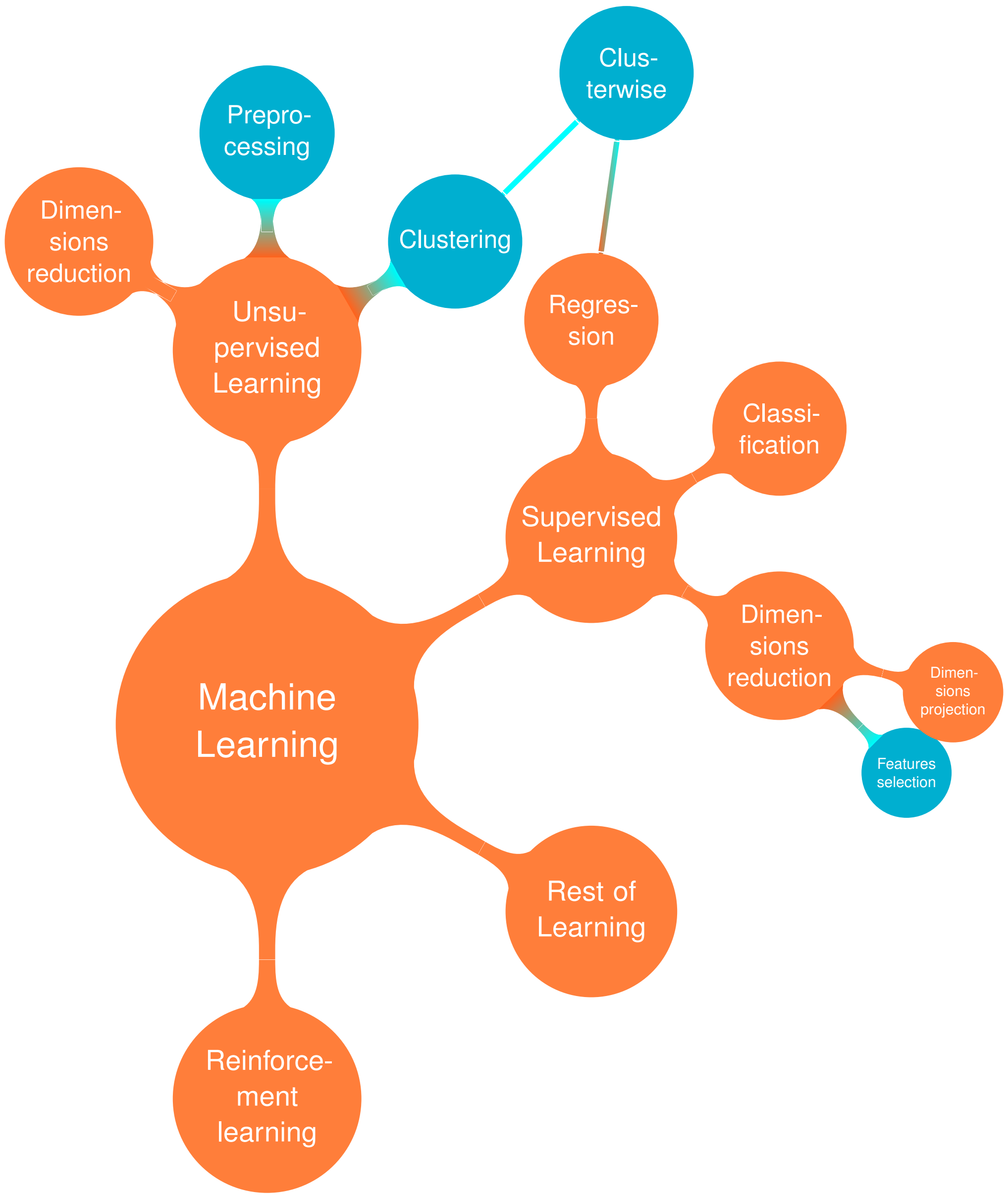


Figure 1.1: Learning Domains

formed from a process which minimizes the dissimilarities inside the clusters and which maximizes the dissimilarities between clusters. Given a set of observations $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n\}$ and a pairwise dissimilarity measure $d(.,.)$, the goal of the clustering problem is to find a partitioning $\Pi = \{C_1, \dots, C_M\}$ of $\boldsymbol{X}$ such that :
$\forall i \in [M], \forall \boldsymbol{x}_u, \boldsymbol{x}_v \in C_i, \forall \boldsymbol{x}_w \in C_j \;\; with \;\; j \in [K] \backslash i,$

$$d(\boldsymbol{x}_u, \boldsymbol{x}_v) < d(\boldsymbol{x}_u, \boldsymbol{x}_w)$$

and

$$d(\boldsymbol{x}_u, \boldsymbol{x}_v) < d(\boldsymbol{x}_v, \boldsymbol{x}_w)$$

Even the formal definition can be overwritten depending on what we consider as a "good" clustering. Imagine for example two close spherical clusters, a small one and a big one. In these conditions the closest pair of points between the two clusters can be less distant than the pair of points farthest from each other within the same cluster, including one of the preceding points. When observations are distributed over a network, clustering algorithms follow the general framework depicted in Figure 1.2 [56, 114, 85, 108].

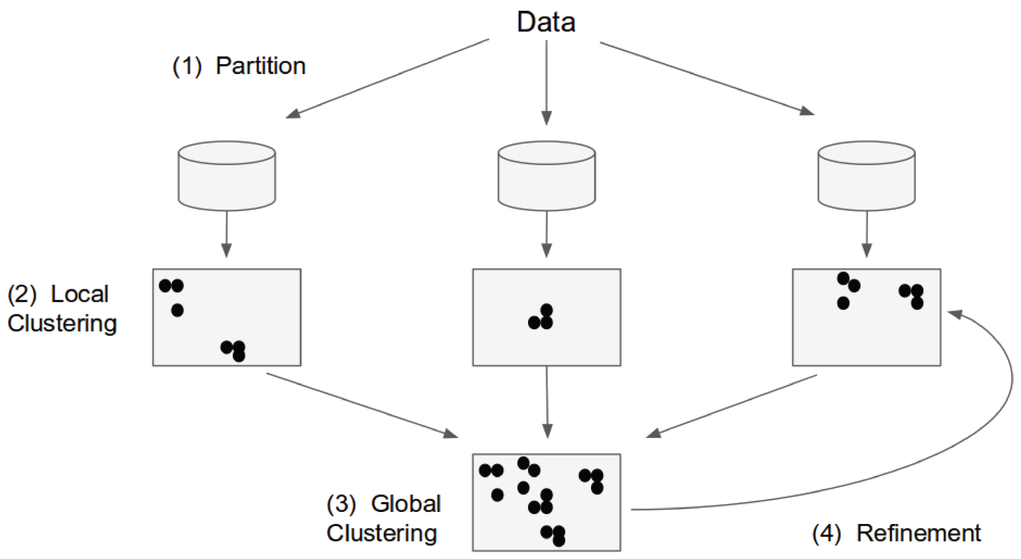


Figure 1.2: The general framework of most parallel and distributed clustering algorithms [108].

1. **Partition.** Distribution and partitioning of data take place over machines.
2. **Local Clustering.** The clustering algorithm is applied on each machine's data independently.
3. **Global Clustering.** Previous clustering results are aggregated into a global clustering.
4. **Refinement of Local Clusters.** Eventually global clustering results can be used to refine the local clusters.

For comprehensive reviews of clustering, [64, 103] present interesting aspects of it. In supervised learning, algorithms learn from labeled data. It consists in learning on data which is divided in two parts. The first one is is the observations described by a set of features. The second is the target information (labels), which drive the learning process. Depending on the case, the second part of data can have different nature. In predictive issues, the objective is to find the missing label of a new object knowing only its raw data through a model learned with a training set, which has the target part (labels). Two categories exist in this area, classification where the feature to predict is a categorical value and regression where it is a continuous value. Other prediction problems exist where the target takes more than one value as we will see in Chapter 5. Reinforcement learning is the third big field of machine learning with supervised and unsupervised learning. It consists of algorithms that optimize a score using the concept of reward.

1.2 Experimental tools

During the experimentation process, we used different hardware and software tools. The hardware part concerned a laptop with an Intel Xeon E3-1535M v6 for small datasets. Scala was the selected programming language to implement our algorithms, which are distributed with the Apache Spark framework. Distributed experiments were operated on the Grid5000 testbed which is a French national large-scale and versatile platform. On this testbed, we used dual 8 core Intel Xeon E5-2630v3 CPUs and 128 GB memory to test the performance of our contributions.

Scala

Scala is a strong typed multi-paradigm programming language born in 2004, it has the advantage to support both object oriented and functional programming. One good point of functional programming is enabling the immutability of objects, which provides a strong guarantee of integrity. Compiled programs are run on the *Java* Virtual Machine (JVM). Thus we can call any optimized *Java* methods.

Two of the most common methods on *Scala* collections, *map* and *reduce*

If we take a look to the *Scala* collection hierarchy, we have GenTraversable, which is a *trait* gathering two of the most used collection methods, *map* and *reduce*. These two functions have the following signatures:

Listing 1 *map* function

```scala
abstract def map[B](f: (A) => B): Traversable[B]
```

- ***map***: The *map* function will work on a *collection[A]* and will return a *collection[B]* by applying a function $f : A \to B$ on each element of the collection.

Listing 2 *reduce* function

```
abstract def reduce[A1 >: A](op: (A1, A1) => A1): A1
```

Listing 3 *map* basic examples

```
val l1 = List(0, 1, 2)

def f(i: Int): Int = i + 1

val l2: List[Int] = l1.map(f) // return List(1, 2, 3)
// Using an anonymous function
val l3: List[Int] = l1.map(_ + 1) // return List(1, 2, 3)

l2 == l3 // return true
```

- ***reduce***: The *reduce* function is also called on a collection[A] and return a single element of type *A* by applying a function $op : (A1, A1) \rightarrow A1$ on each collection's element. Thus, reapplying this process iteratively with previous results until getting only one element of type *A*. An important property is that the function *op* **has to** be associative because as it is specified in the documentation, *the order in which operations are performed on elements is unspecified and may be nondeterministic.* The A1 >: A authorize the reduce to focus on specific information present in parent class of A, we can for example imagine a class $B2$ with two properties $b1$ and $b2$ which inherit from $B1$ which has only information $b1$. We could desire to reduce only $b1$ without being preoccupied by $b2$, in that case we can imagine $op : (B1, B1) \rightarrow B1$.

Listing 4 *reduce* basic examples

```
val l1 = List(0, 1, 2)

def f(a: Int, b: Int): Int = a + b

val s1: Int = l1.reduce(f) // return 3
// Using an anonymous function with 2 arguments
val s2: Int = l1.reduce(_ + _) // return 3

s2 == s3 // true
```

Apache Spark

Apache Spark is originally a Scala framework for Big Data, it allows to spread and operate on large datasets where items are gathered into partitions which are distributed over the cluster of machines. It is important to notice that the whole execution process is resilient and fault-tolerent. A key concept in Spark is the resilient distributed dataset (RDD) which is a read-only collection of objects partitioned across a group of machines which can be rebuilt if necessary from the hierarchy of previous RDD operations. Therefore if a single partition is lost, it can be rebuilt by relaunching computation thanks to the lineage of operations. This last is extracted from the directed acyclic graph (DAG) which is built dynamically. The preceding approach has the advantage to enable fault tolerance. Most of the Map and $Reduce$ operations will be performed on RDDs even if other pure Scala Map and $Reduce$ operations are executed inside each Spark partition. We implement our algorithm using Scala because it is Spark's native language. It allows us to benefit from last advances whilst keeping best performances compared to other language such as Python. As pure Scala collections, Spark has followed the same pattern for its distributed collection API. Thus the Spark *map* and *reduce* follow same signatures to obtain the same results but on distributed collections.

A key concept of Spark resides in its partitioning logic. A partition is the smallest part of a distributed collection. Thus a dataset is decomposed into many partitions, which have to be defined properly by user depending on use cases. Once defined, each node or machine of the Spark cluster will have some partitions depending on its capacity. One strength of Spark is to treat its partitions lazily throughout its executors. An executor is an entity which has a defined amount of resources such as memory and cores. Once the *DAG*(Directed Acyclic Graph) of tasks is generated, Spark's executors will execute tasks applying functions partition by partition in a sequential manner. If a partition is lost, fault tolerance will enable to relaunch specific tasks on a new partition.

1.3 Thesis organization and main contributions

In this thesis, multiple contributions are presented. Before presenting them, chapter 2 introduces major concepts and ideas linked to clustering and other interesting aspects of machine learning.

Chapter 3 will focus on unsupervised algorithms with algorithm gravitating around Mean Shift algorithm for continuous data. We propose a new design for the Mean Shift clustering algorithm to significantly diminish its complexity from quadratic to linear. Knowing that the most time consuming part of the Mean Shift algorithm was the gradient ascent, we applied on it random projection based on the locality sensitive hashing. We also prove empirically that as a preprocessing step, this algorithm enhances other clustering algorithms. Its variation for scalar data will be presented in chapter 4. The scalar Mean Shift can be applied on binary data by replacing computation of the mean by the median center.

In chapter 5, we will introduce a Clusterwise, which is a clustering problem intertwined with regression. The issue is to find clusters of entities such that the overall sum of squared errors from regressions performed over these clusters is minimized, where each cluster may have a different variance. We improve its time duration and scalability by applying clustering before the regression task. Thus we form micro clusters which enhance the algorithm ability to deal with larger datasets. We combine the Clusterwise with the PLS regression to allow regressions simultaneously on multiple goals. In chapter 6 a collaborative work about Rough Set feature selection is presented with the introduction of a meta heuristic on it which can be applied on the classic Rough Set feature selection but also on any heuristic on it. We then proposed a distributed implementation of the algorithm. Applications of contributions in the company Kameleoon is presented in chapter 7. We present also Clustering4Ever which is the open source project born during the thesis. The last chapter 8 will conclude the thesis with final collaboration and various perspectives.

Part I

Unsupervised Learning

Chapter 2

State of the art

2.1 Definitions

Throughout the literature, authors use several words to describe the same concept. Thus, record, object, observation, data point, tuple, and item are used to describe an elementary unit of information. Most of the time data is represented in a high dimensional space so the terms, variable, feature, dimension, or attribute are used to refer to a part of an observation. Lets denote a dataset X as a set of observations $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$, where $\boldsymbol{x}_i = \{\boldsymbol{x}_i^1, \boldsymbol{x}_i^2, \ldots, \boldsymbol{x}_i^d\}^\intercal$.

Distances and similarities are often in the center of the clustering process. We need to choose them wisely if we wish to obtain proper results. Similarity describes how two objects or clusters are similar between them, the higher the score is, the more similar the two inputs are. On the other side the dissimilarity or distance describes the opposite, thus the higher the obtained score is the more the two objects will be different. The most common example is the euclidean distance defined as follows:

$$d(\boldsymbol{x}_1, \boldsymbol{x}_2) = \sqrt{\sum_{i=1}^{d} (\boldsymbol{x}_1^i - \boldsymbol{x}_2^i)^2}$$

where $\boldsymbol{x}_1 = \{\boldsymbol{x}_1^1, \boldsymbol{x}_1^2, \ldots, \boldsymbol{x}_1^d\}^\intercal$ and $\boldsymbol{x}_2 = \{\boldsymbol{x}_2^1, \boldsymbol{x}_2^2, \ldots, \boldsymbol{x}_2^d\}^\intercal$. There exist variants of the euclidean distance. The weighted euclidean distance which gives a user the ability to weight features. The Minkowski distance is considered as a generalization of Manhattan and Euclidean distances, it is expressed as follow :

$$d\left(\boldsymbol{x}_1, \boldsymbol{x}_2\right) = \left(\sum_{i=1}^{d} |\boldsymbol{x}_1^i - \boldsymbol{x}_2^i|^p\right)^{\frac{1}{p}}$$

Centers and prototypes are two common concepts used to describe a specific individual within a cluster, which is considered as the best representative regarding the corresponding metric. Under euclidean distance it can be the center of the sphere. Other times it can be the median or the real or theoretical point minimising distance to all others. Three commonly used terms in this thesis are mean, median and mode, which are represented on Figure 2.1. First the Mean works only in continuous space with any number of dimensions and is $mean = \frac{\sum_{i=1}^{n} x_i}{n}$. Median works on any ordered space and it divides sorted observations in two equal size parts. Mode is a point representing highest density regions inside an ensemble.

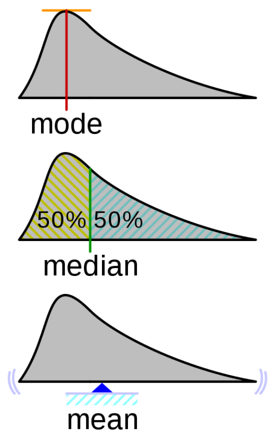


Figure 2.1: Mode, Median, and Mean

2.1.1 Local and distributed algorithms

We consider as local clustering algorithms those which are running on a local environment – a single machine – even if it is really powerful. On the other side of local clustering algorithms, distributed clustering algorithms are designed to deal with multiple machines interconnected through a network to form what is called a cluster, we will express them in the following as cluster of machines to avoid confusion with clusters which are obtained by application of clustering algorithms.

Shuffling is an important concept to build efficient distributed systems. It refers to the exchange of data from some machine to others through the network. Sometimes shuffling is inevitable, often it happens that the design of algorithms or architectures can be optimized to minimise data shuffling. The reason why shuffle has to be minimized is because the network can represent a bottleneck for some applications.

Space partitioning means to divide the exploration space into a number of non-overlaping regions. It is a key point for rendering complex algorithms applicable on Big Data.

Scalability defines the ability of algorithms to compute massive amount of data in a reasonable duration proportionally to the size of data. But this notion is not documented well enough and often depends on user goals. We will define here what we can call a pure scalable algorithm for a specific hyper parameter x as the ones which have a complexity with an upper bound $O(k.x)$ where $k \in \mathbb{R}^+$. Other scalable algorithms that we will define as pseudo scalable are the sub quadratic ones upper bounded by $O(x^2)$, some examples are the $O(x.log(x))$ or $O(\sqrt{x}.x)$ complexities. All other algorithms with quadratic complexity or more are not scalable at all.

It is important to notice that complexity does not take into account which hyper parameter has an important complexity or not. Then many algorithms can be linear with the number of data but quadratic or more on other hyper parameters as the number of dimensions, it is for example the case with the PCA with a cubic complexity over its dimensions' input which prevents any practical applications with data sets with more than 10000 dimensions. This is a bit questionable considering that its goal is to reduce dimensionality.

Speed versus scalability. Let recall that being quadratic does not always mean being slower, a classical trick is to use quadratic sorting algorithm rather than pseudo scalable one on specific and small data sets which gives better performances. Of course when the data set size increases they become irrelevant.

2.2 Data preprocessing

Because there is a gap between raw and learning data, it is important to treat incoming data accordingly with its nature by extracting maximum meaningful information whilst keeping the inner structure of original data. A major step to obtain usable data is preprocessing using projection techniques, feature selection, clustering algorithms. A common problem in machine learning is the curse of dimensionality, which prevents usage of the majority of distances, this is why dimension reduction techniques are developed. There are different forms of reduction techniques such as feature selection, which select the best features on which an algorithm is applied. It has the advantage to keep the meaning of dimensions unlike dimension projections which reduce dimensions by projecting them on a lower dimensional space loosing content readability. Feature selection can also remove noisy dimensions which can diminish performances if they are present. More explanations are given in Chapter 6.

The Principal Component Analysis is a projection technique, which can be used as a dimension reduction technique allowing to extract a reduced space from a real dataset. It takes one parameter $D \in \mathbb{N}$ which is the number of dimensions on which we project the original space. As [65] describes it, *there are two commonly used definitions of PCA*:

- It *can be defined as the orthogonal projection of the data onto a lower dimensional linear space, known as the orthogonal projection of the data onto a lower dimensional linear space, known as the principal subspace, such that the variance of the projected data is maximized (Hotelling, 1933).*
- *It can be defined as the linear projection that maximizes the average projection cost, defined as the mean squared distance between the data points and their projections (Pearson, 1901).*

Complexity is in $\mathbf{O}(\mathbf{d}^2\,\mathbf{n} + \mathbf{d}^3)$ so even if it is linear with the number of data, it growths fast with the number of dimensions.

One of the most recent non-linear projection approaches is UMAP [128], which stands for Uniform Manifold Approximation and Projection is a novel manifold learning technique for dimension reduction. UMAP is a theoretical framework based in Riemannian geometry and algebraic topology. It allows to reduce dimensions in an unsupervised or supervised manner with a sub quadratic time complexity for both the number of data points and the number of dimensions.

2.3 Clustering algorithms

In this section we will essentially focus on algorithms which have scalable equivalents. A fundamental question for those who desire to build efficient clustering algorithm can be as simple as what is a good clustering. First of all, we have to clearly define the goal on which our clustering will give insights. Then define which distance best fits with the problem without forgetting to select the right clustering model.

There are multiple models of clustering exposed on Figure 2.2 each of them is specialized for a specific task. To find which model best fits our use case is an important challenge, the following list exposes main model types with their most prominent algorithms.

2.3.1 K-means like (K-Centers)

K-Centers is a generic algorithm, which has been studied under many names depending on its application domain. The most famous are the K-Means [19], K-Median, K-Modes, and K-Prototypes. The generic definition of this algorithm is as follows: for a given number of clusters K, find a space partitioning where clusters are formed by the belonging to their closest *centers*, also called *prototypes*. In order to reach this goal, two elements have to be well defined. The first one, as often in clustering algorithms, the definition of a dissimilarity measure for the studied space in order to know distance from points to centers. The second is the *center* concept. K-Means defines it as the mean in an euclidean space, and as the median center for the K-Median, and the majority vote for the K-Modes with the hamming distance when data is binary. Some of these algorithms are described in detail below.

2.3.1.1 K-Means

K-Means is probably the most well known clustering algorithm which has two key aspects, its simplicity and its rapidity. It belongs to the category of partitioning clustering algorithms. K-means finds K classes $S = S_1, S_2, ..., S_K$ from a set X of n observations $\mathbf{x}_i$, by minimizing the following cost function:

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} ||\mathbf{x} - \boldsymbol{w}_i||^2$$

where $\boldsymbol{w}_i$ is the center of cluster i (prototype). The famous technique to minimize the cost function is an iterative refinement technique. It starts with the defined K center. The most naive technique is to select them randomly or to pick K data points from the training dataset. Other sophisticated initializations exist as the K-Means++ one [67]. The basic iteration has two phases:

- The assignment phase: assign to each data point the label of its closest mean using Euclidean distance
- The update: update the value of prototype by computing the mean of the generated cluster from assignment step.

K-Means is the most known clustering algorithm which works on $\mathbb{R}^d$, it discovers elliptic clusters.The parameter K defines the number of searched clusters. It is a scalable algorithm with Linear time complexity in $\mathbf{O(k\,t\,n\,d)}$ ($k = K$ is the

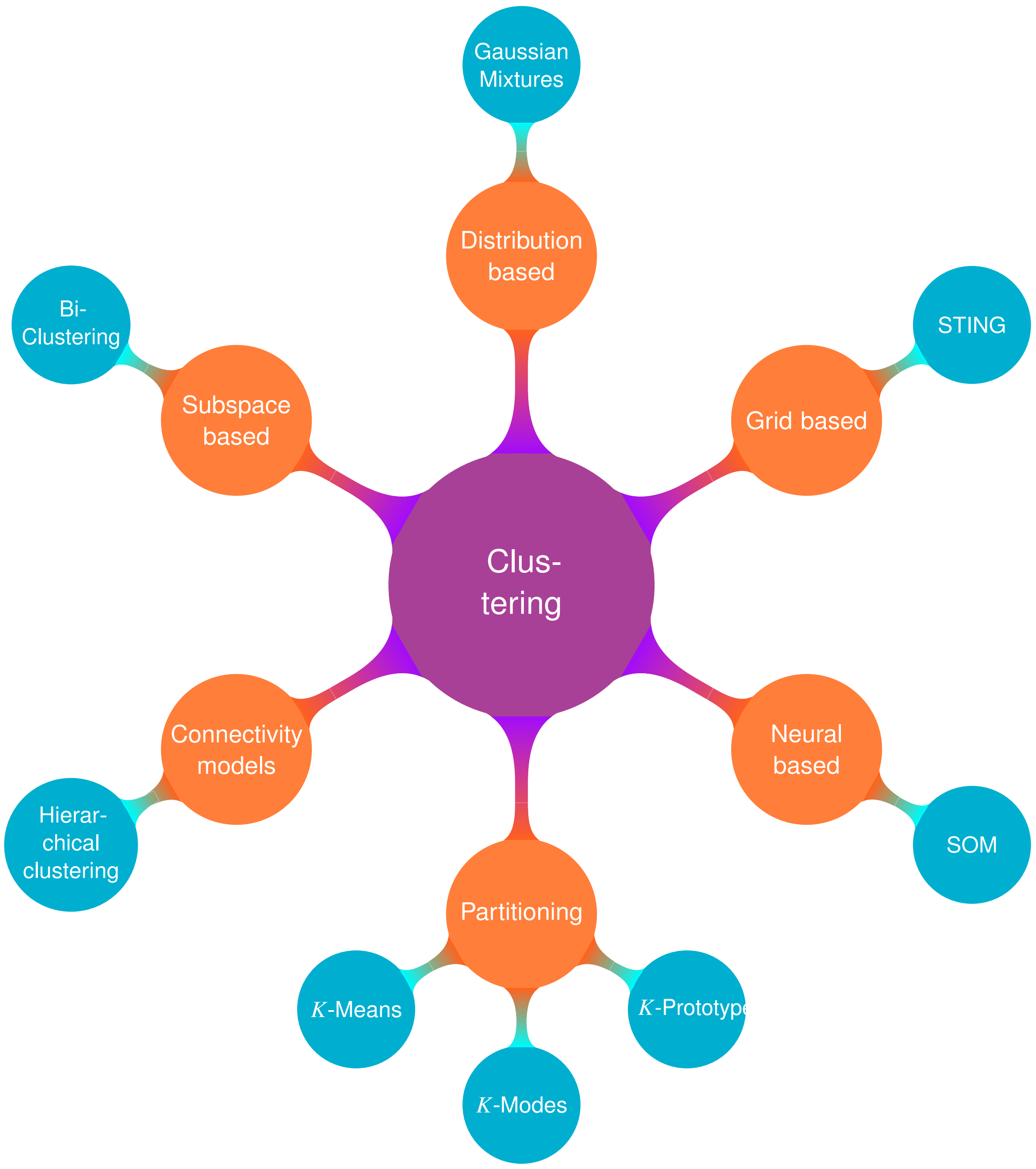


Figure 2.2: Clustering fields

number of clusters, $t \sim 100$ is the maximum number of iterations, n is the number of data points, and d the number of dimensions). A snippet of source code of a generalization of K-Means, the K-Centers, is presented briefly with a tail recursion implementation.

Algorithm 1 K-Means algorithms

```
   Input: $X$, $K$, $iter_{max}$, $\epsilon$
   Output: $K$ centroids
1: Generate $K$ random centroid
   Initialize stopping condition loop $KHaveConverged, i$
2: $KHaveConverged = false$
3: $i = 1$
4: while $i \leq iter_{max}$ && $!KHaveConverged$ do
5:     Associate each point of $X$ to its nearest centroid neighbors
6:     Update centroid by associating them the mean among all its assigned
   points
7:     if for all centroids, $D(\mathbf{w}_{i-1}^{K_i}, \mathbf{w}_i^{K_i}) \leq \epsilon$ then
8:         $KHaveConverged = true$
9:     $i+ = 1$
```

2.3.1.2 K-Modes

K-Modes [30] is the binary equivalent of K-Means ($\mathbf{x} \in \{0, 1\}^d$). The algorithms differ in two aspects: the basic metric used, which is the Hamming distance and it replaces centers with median center that corresponds to the majority vote in order to minimize the following cost function:

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \mathcal{H}(\mathbf{x}, \boldsymbol{w}_i)$$

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} |\mathbf{x} - \boldsymbol{w}_i|$$

It is in linear time complexity with $\mathbf{O(k\ t\ n\ d)}$, where

- $k = K$ is the number of clusters
- $t \sim 100$ is the maximum number of iterations
- n is the number of data points
- d is the number of dimensions

Algorithm 2 K-Modes algorithms

```
    Input: $X$, $K$, $iter_{max}$, $\epsilon$
    Output: $LabelizedData$, $K$ centroids
 1: Generate $K$ random centroid
    Inialize stoping condition loop $KHaveConverged, i$
 2: $KHaveConverged = false$
 3: $i = 1$
 4: while $i \leq iter_{max}$ && $!KHaveConverged$ do
 5:     Associate each vector $\boldsymbol{x}$ of $X$ to its nearest centroid neighbors
 6:     Update centroid by associating them the median center among all its
    assigned points
 7:     if for all centroids, $\mathcal{H}t(\mathbf{w}_{i-1}^{K_i}, \mathbf{w}_i^{K_i}) \leq \epsilon$ then
 8:         $KHaveConverged = true$
 9:     $i+ = 1$
```

2.3.1.3 K-Prototypes

K-Prototypes is the combinations of K-Means & K-Modes. It works on mixed data, and uses a mixed distance to compute distance from points to center. As for updating the centers, it takes both previous algorithms: computation of the mean for the continuous part and of the majority vote for the binary part (median center). It is, in linear time complexity in $\mathbf{O(k\,t\,n\,d)}$ ($k = K$ is the number of clusters, $t \sim 100$ is the maximum number of iterations, and n is the number of data points).

Algorithm 3 K-Prototypes algorithms

```
    Input: $X$, $K$, $iter_{max}$, $\epsilon$
    Output: $LabelizedData$, $K$ centroids
 1: Generate $K$ random centroids
 2: Converged $\leftarrow$ false
 3: $i \leftarrow 1$
 4: while $i \leq iter_{max}$ and Converged $= false$ do
 5:     Associate each vector $\boldsymbol{x}$ of $X$ to its nearest centroid
 6:     Update prototypes by associating them the "center" among all its assigned
    points for binary data and the gravity center for continuous data
 7:     if for all prototypes, $D(\mathbf{w}_{i-1}^{K_i}, \mathbf{w}_i^{K_i}) \leq \epsilon$ then
 8:         $KHaveConverged = true$
 9:     $i+ = 1$
```

2.3.2 *K*-means++

The authors of [67] proposed a specific way of choosing the initial prototypes for the K-means algorithm. Let $dist(\mathbf{x})$ denote the shortest distance from a data point $\mathbf{x}$ to the defined closest prototype. The k-means++ method is described in Algorithm 4. The main idea in the K-means++ algorithm is to choose the centers (prototypes) one by one in a controlled fashion, where the current set of chosen prototypes will stochastically bias the choice of the next prototype. The central drawback of the K-means++ initialization from a scalability point of view is its inherent sequential nature: the choice of the next prototype depends on the current set of prototypes [93].

Algorithm 4 k-means++

```
1: Take one prototype $\boldsymbol{w}_{c_1}$, chosen uniformly at random from the set of data, $\mathcal{X}$
2: Take a new prototype $\boldsymbol{w}_{c_i}$, choosing $\mathbf{x} \in \mathcal{X}$ with probability $\frac{dist(\mathbf{x})^2}{\sum_{\mathbf{x}\in\mathcal{X}} dist(\mathbf{x})^2}$
3: Repeat Step 2 until we have taken $k$ clusters altogether
4: Proceed as with the standard $K$-means method (Algorithm 1)
```

2.3.3 Self Organizing Maps

Proposed by Kohonen [35] the SOM algorithm is a type of artificial neural network for unsupervised learning. It is able to create spatially organized representations of data inputs. Starting from the high-dimensional input space, SOM produces a low-dimensional representation in a discrete space which is generally called network or map presented in Figure 2.3. A neighborhood function is used to preserve the topological properties of the input space, it forms a topological map where close objects are gathered and dissimilar ones are stretched apart. Like most artificial neural networks [32], SOM has a two-fold objective:

1. **Adjusting map**: using input data the topological map is built. A map is composed of nodes arranged in a predefined manner. Usually, the dimensionality of the network is one or two, its topology may differ depending on the needs. A prototype, $\mathbf{w}_c$, of dimension corresponding to input data is associated with each network node.

2. **Mapping (quantization)**: assign input data into a non-linear, discrete map. The aim of vector quantization is to assign a data point to a prototype which minimizes pairwise distance respecting a neighborhood function in order to preserve data topology. Then similar input data points will fall into neighbor network nodes.

The first step is to initialize a discrete topological map of size $p \times q = k$. For a grid $C = \{c_1, ..., c_k\}$ where $\forall i \in [1, k], c_i$'s are network nodes. $\mathcal{G}$ is associated with prototype space $\mathcal{W} = \{\mathbf{w}_1, .., \mathbf{w}_k\}$ where $\forall i \in [1, k], \mathbf{w}_i$ are the prototypes corresponding to network nodes c_i. For each pair of network nodes c_a and c_b in C, their mutual influence is expressed with the function $\mathcal{K}^T(\delta(c_a, c_b))$ as in Equation (2.1). Many functions can be used, a common one is the Gaussian function.

$$\mathcal{K}^T(\delta(c_a, c_b)) = e^{\frac{-\delta(c_a, c_b)}{T}} \tag{2.1}$$

Here, T stands for the temperature which decreases iteratively to control the depth of the neighborhood influence for a given cell:

$$T = T_{max}\left(\frac{T_{min}}{T_{max}}\right)^{\frac{ith}{N_{iter}-1}} \tag{2.2}$$

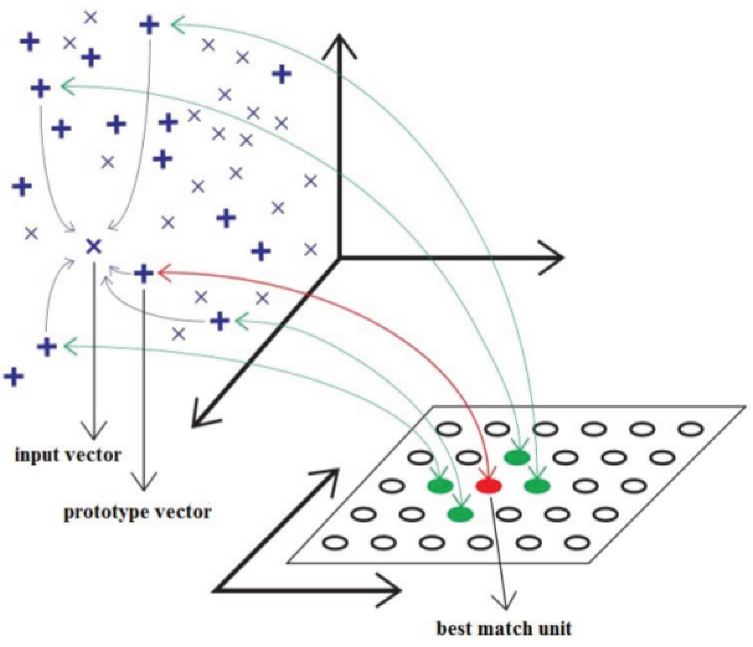


Figure 2.3: SOM principles: mapping and quantization

N_{iter} is the number of iterations, and $\delta(c_a, c_b)$ represents the shortest distance between the pair of network nodes c_a, c_b.

Due to the use of $\mathcal{K}$ function, every network node moves, according to the neighborhood function, along the same direction towards the learning data during training step and similar points tend to be assigned to adjacent network nodes [42]. Two versions of the SOM algorithm are mainly exploited, the stochastic and the batch ones, both aim to minimize the cost function presented in equation 2.3.

$$\mathcal{R}_{SOM}(\phi, \mathcal{W}) = \sum_{i=1}^{n} \sum_{j=1}^{k} \mathcal{K}^T(\delta(\phi(\boldsymbol{x}_i), c_j))||\boldsymbol{x}_i - \mathbf{w}_j||^2 \tag{2.3}$$

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

$$\phi(\boldsymbol{x}_i) = \arg \min_{j=1,\ldots,k} ||\boldsymbol{x}_i - \mathbf{w}_j||^2 \tag{2.4}$$

The learning steps are very close to the K-means ones:

1. **Initialization step**: initialize the discrete structure with network nodes and the grid topology to finally initialize prototypes.

2. **Assignment step**: attribute points to their nearest prototype. This process guarantees that the cost function $\mathcal{R}(\phi, \mathcal{W})$ is minimized with respect to the assignment function ϕ presuming that prototype vectors are constant. Moreover, this step binds data to network nodes.

3. **Update step**: re-compute the prototype vectors. Prototypes and their neighborhood shift towards the assigned data whilst the map approximates the data distribution. This includes minimizing the cost function $\mathcal{R}(\phi, \mathcal{W})$ with respect to the prototypes vectors.

Batch SOM

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

$$\mathbf{w}_c = \frac{\sum_{r=1} \mathcal{K}^T(\delta(c,r)) \sum_{i=1}^{n_r} \mathbf{x}_i}{\sum_{r=1} \mathcal{K}^T(\delta(c,r)) n_r} \tag{2.5}$$

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

$$\phi(\mathbf{x}_i) = \arg \min_{j=1,\ldots,k} \mathcal{K}^T(\delta(\mathbf{x}_i, \mathbf{w}_j)) \|\mathbf{x}_i - \mathbf{w}_j\|^2 \tag{2.6}$$

Algorithm 5 Batch SOM version

```
1: Initialize $k$ prototypes and $\mathcal{W}$;
2: while stopping criteria have not been fulfilled do
3:     for $i = 1 \to n$ do
4:         Find the best match unit to the current selected input data according to Equation (2.6) $c_{\phi(\mathbf{x_i})} = c_{\phi(\mathbf{x_i})} \cup \{\mathbf{x_i}\}$ //Put $\mathbf{x_i}$ into cluster $\phi(\mathbf{x_i})$
5:     for $j = 1 \to k$ do
6:         Update prototype vectors according to Equation (2.5)
```

Stochastic SOM

In the stochastic version, each iteration consists of presenting the SOM map with a randomly selected data point. The best match unit (the nearest node) as well as its neighbors move toward the input point (see Figure 2.3).

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

$$\mathbf{w}_c^t = \mathbf{w}_c^{t-1} - \mu^t \mathcal{K}^T(\delta(c, c_{\phi(\mathbf{x}_i)}))(\mathbf{w}_c^{t-1} - \mathbf{x}_i) \tag{2.7}$$

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

Algorithm 6 Stochastic SOM version

```
1: Initialize $k$ prototypes and $\mathcal{W}$
2: while stopping criteria have not been fulfilled do
3:     for $i = 1 \to n$ do
4:         Find the best match unit to the current selected input data according to Equation (2.4)
5:         for all $c_r$ is a neighbor of $\phi(\mathbf{x}_i)$ (including $\phi(\mathbf{x}_i)$ itself) do
6:             Update the nodes in the neighborhood of $\phi(\mathbf{x}_i)$ according to Equation (2.7) (including the node $\phi(\mathbf{x}_i)$ itself) by pulling them closer to the input data
```

2.3.4 Neural Gas

Neural Gas (NG) [24] is inspired by the SOM. While the SOM map dimensionality must be chosen a priori, depending on the data distribution, the topological network of neural gas may have a different arrangement. Neural Gas is a more flexible network capable of quantizing topological data and learning the similarity among the input data without defining a network topology. Unlike SOM, the adaptation strength in Neural Gas is constant over time and only the best match prototype and its direct topological neighbors are adapted.

Given a network of k clusters $C = \{c_1, ..., c_k\}$ associated with k prototypes $\mathcal{W} = \{\mathbf{w}_1, ..., \mathbf{w}_k\}$, they are adapted independently of any topological arrangement of the network nodes within the network. Instead, the adaptation step is affected by the topological arrangement within the input space. For each data point $\mathbf{x}_i$ is selected, prototypes will be ajusted by distortions $\mathcal{D}(\mathbf{x}_i, c_j) = ||\mathbf{x}_i - \mathbf{w}_j||, \forall j = 1, ..., k$. The resulting adaptation rule can be described as a "winner takes most" instead of a "winner takes all" rule [23]. The winner network node denoted by j_0 is determined by the assignment function

$$j_0 = \phi(\mathbf{x}_i) = \arg\min_{j=1,...,k} ||\mathbf{x}_i - \mathbf{w}_j||^2. \tag{2.8}$$

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

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

When an observation is selected, the prototypes move toward it by adjusting the distortion $\mathcal{D}(\mathbf{x}_i, c_{j_0})$, controlled by a neighborhood function $\mathcal{K}^T$. In [23], this function is fixed, e.g. $\mathcal{K}^T = \exp^{knn_j/T}$ where knn_j is the number of neighborhood network nodes of c_j. This directly affects the adaptation step for $\mathbf{w}_j$ which is determined by:

$$\mathbf{w}_j^t = \mathbf{w}_j^{t-1} - \varepsilon\mathcal{K}^T(\delta(c_j, c_{\phi(\mathbf{x}_i)}))(\mathbf{x}_i - \mathbf{w}_j) \tag{2.9}$$

To capture the topological relations between the prototypes, each time an observation is presented, the connection between j_0 and j_1 is incremented by one. Each connection is associated with an "age" variable. Only the connection between j_0 and j_1 is reset, the other connections of j_0 age, i.e. their age increment. When the age of a connection exceeds a specific lifetime Max_{age}, it is removed. The way to update the age of the connections is to increase with each incoming input object learnt. Finally, Neural Gas can be summarized by the Algorithm 7.

In this algorithm, stopping criteria can be either: a number of iterations and a threshold for the quantization error.

2.3.5 Hierarchical clustering

All approches based on hierarchical clustering are divided into two families: Ascending (agglomerative) and descending (divisive). The divisive version starts with the whole observation as a single cluster, which is divided iteratively into

Algorithm 7 Neural Gas

```
1:  Initialize $k$ prototypes and set all $\mathcal{S}_{ij}$ to zero
2:  for all $\mathbf{x}_i \in \mathcal{X}$ do
3:      Determine the sequence $(c_{j_0}, c_{j_1}, ..., c_{j_{n-1}})$ such that
            $$||\mathbf{x}_i - \mathbf{w}_{j_0}|| < ||\mathbf{x}_i - \mathbf{w}_{j_1}|| < ... < ||\mathbf{x}_i - \mathbf{w}_{j_{k-1}}||$$
        $\mathbf{w_{j_0}}$ is the best match prototype, i.e., the nearest prototype; $\mathbf{w_{j_1}}$ is the second nearest prototype to $\mathbf{x_i}$
4:      for all $c_j$ with $\mathcal{S}_{j_0,j} == 1$ do
5:          Perform an adaptation step for the prototypes according to Equation (2.9)
6:      if $\mathcal{S}_{j_0,j_1} == 0$ then
7:          Create a topological connection between $c_{j_0}$ and $c_{j_1}$, i.e., $\mathcal{S}_{j_0,j_1} = 1$
8:          Set age for this connection, i.e., $age_{j_0,j_1} = 0$
9:      for all $c_j$ with $\mathcal{S}_{j_0,j} == 1$ do
10:         Increase the age of all connections of $j_0$ by one, i.e., $age_{j_0,j} = age_{j_0,j} + 1$
11:         if $age_{j_0,j} > Max_{age}$ then
12:             Remove all connections of $j_0$ which exceeded their age, i.e., $\mathcal{S}_{j_0,j} = 0$
```

clusters with the largest variance until K clusters are obtained. At the opposite, the agglomerative approach [3], considers each observation as individual clusters and are iteratively merged until K clusters are formed. In both cases, a central question is: how to define the distance between two clusters and observations ? Many solutions have been proposed such as complete, single and mean linkage clustering.

These clustering methods are also called hierarchical clustering because they provide a hierarchy of clusters. Despite interesting aspects, these algorithms have some drawbacks. The complexity is $O(2^{n-1})$ for divisive algorithms and $O(n^3)$ for agglomerative; both cases prevent any big data applications. Moreover these algorithms suffer from sensibility to outliers which can lead to unwanted additional clusters or merging between proper clusters.

Algorithm 8 Agglomerative hierarchical algorithms

```
Input: $X$, $K$, $iter_{max}$, $\epsilon$
Output: $\tilde{c}(X)$, $K$ centroids
Step 1 Consider each input element as a singleton cluster.
Step 2 For each pair of clusters $c_1$, $c_2$ calculate their distance $d(\boldsymbol{w}_{c_1}, \boldsymbol{w}_{c_2})$
Step 3 Merge the pair of clusters that take the smallest distance depending linkage.
Step 4 Continue the step 2, until the termination criterion is satisfied.
Step 5 The termination criterion most commonly used is a threshold of the distance value
```

2.3.6 DBScan

Density-based clustering has the ability to discover arbitrary-shape clusters and to handle noise [109]. In density-based clustering methods, clusters are formed based on the dense areas that are separated by sparse areas. DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [27] is one of the most well-known density-based clustering algorithms. The density of each observation is defined based on the number of observations close to that particular point called the point's neighborhood. The dense neighborhood is defined based on two user-specified parameters: the radius (ε) of the neighborhood (ε-neighborhood), and the number of the objects in the neighborhood ($MinPts$).

DBSCAN starts by randomly selecting a point and checking whether the ε-neighborhood of the point contains at least $MinPts$ observations. If not, it is considered as a noise point, otherwise it is considered as a core point and a new cluster is created. DBSCAN iteratively adds the data points, which do not belong to any cluster and are directly density reachable from the core points of a new cluster. If the new cluster can no longer be expanded, the new cluster is completed. In order to find the next cluster, DBSCAN randomly selects the unvisited data points and the clustering process continues until all the points are visited and no new observation is added to any cluster.

The complexity is $O(n^2)$ due to similarity matrix constructions. It works on any metric space.

Algorithm 9 DBScan algorithms

```
 1: procedure DBScan($X, \varepsilon, min_{pts}$)
 2:     $C = 0$
 3:     for each unvisited point $P \in X$ do
 4:         mark $P$ as visited
 5:         $spherePoints = regionQuery(P, \varepsilon)$
 6:         if sizeof($spherePoints$) $< min_{pts}$ then
 7:             ignore P
 8:         else
 9:             $C$ = next cluster
10:             expandCluster($P, spherePoints, C, \varepsilon, min_{pts}$)
11: procedure expandCluster($P, spherePoints, C, \varepsilon, min_{pts}$)
12:     add $P$ to cluster $C$
13:     for $eachpointP'inspherePoints$ do
14:         if $P'$ is not visited then
15:             Mark  $P'$ as visited
16:             $spherePoints' = $ regionQuery($P', \varepsilon$)
17:             if sizeof($spherePoints'$) $>= min_{pts}$ then
18:                 $spherePoints = spherePoints \cup spherePoints'$
19:             if $P'$ is not yet member of any cluster then
20:                 add $P'$ to cluster $C$
21: procedure regionQuery($P, \varepsilon$)
22:     return all points within the n-dimensional sphere centered at $P$
23:             with radius $\varepsilon$ (including $P$)
```

2.3.7 Mean Shift

Mean Shift clustering belongs to the class of modal clustering methods where the arbitrarily shaped clusters are defined in terms of the basins of attraction to the local modes of the data density, created by the density gradient ascent paths. Most studies on the Mean Shift clustering have focused on kernel [80] versions, e.g. [78, 73, 117, 115]. The latter authors compared Gaussian, Cauchy and generalized Epanechnikov kernels to study the behaviour of tuning parameters of Mean Shift clustering. Mean-shift algorithm and its variants consist in two major steps as described in Algorithm 10. The first one is the density gradient ascent, it is generally the most computationally intensive. This gradient ascent can be computed in different ways. In the traditional characterization of the Mean Shift, their gradient ascent paths are computed from successive iterations of kernel applications on each data point with following recurrent equations:

$$\boldsymbol{x}^{u+1} = \mathbf{f}(\boldsymbol{x}^{u})$$

$$\mathbf{f}(\boldsymbol{x}) = \sum_{i=1}^{N} \frac{\mathbf{K}'(\left|\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h}\right|^2)}{\sum_{j=1}^{N} \mathbf{K}'(\left|\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h}\right|^2)} \boldsymbol{x}_i$$

where $\mathbf{K}$ is the kernel with bandwidth h and $\mathbf{K}' = \frac{d\mathbf{K}}{dt}$ is its derivative.

Mean Shift algorithms takes its name from the $meanshift$ vector $\mathbf{f}(\boldsymbol{x}) - \boldsymbol{x}$, which is the averaging of point shifts. As we will present in Chapter 3, there are alternative methods to achieve the gradient ascent such as nearest neighbors.

The second step is the actual clustering where we use results from the first step to assign cluster labels to the original data points. Traditionally the model is obtained by computing modes, which gather point inside the bandwidth.

Algorithm 10 Mean Shift principle

```
    **Input:** points $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\}$,
    **Output:** cluster label $\{\tilde{c}(\boldsymbol{x}_1), \ldots, \tilde{c}(\boldsymbol{x}_m)\}$
    **Step 1:** Density gradient ascent;
    **Step 2:** Cluster labeling;
```

2.3.8 Gaussian Mixture EM

Gaussian Mixture can be seen as a generalization of the K-Means where centers are not only described as the mean of a cluster but by a combination of the mean and the standard deviation of its elements over the mean.

The Gaussian Mixtures belongs to distribution-based clustering algorithms. The reason lies as follow. Let's suppose that we have a d dimensional continuous dataset $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n\}$ where $\boldsymbol{x}_i$ are observations of a d dimensional random variable $\boldsymbol{x}$. The assumption behind this is that observations do not come from a single component, or cluster but from multiple ones [63]. The problem is to estimate component's parameters so that they fit optimally the data. Predicting these parameters and the belonging of these points to each component produces a clustering. Another problem is to find the number of components; model based methods have been proposed in that sense [31]. Imagine now that the number of components K is defined, each cluster can be represented by a parametric distribution θ_k which is often called component distribution and the whole data set can be modeled by a mixture of them. This component distribution, which is similar to a probability density function p, of a specific point x_n takes the following form:

$$p(x_n) = \sum_{k=1}^{K} \pi_k p(x_n|\theta_k)$$

where θ_k are sets of parameters associated to kth component and π_k are the mixing probabilities, or weights. $p(x_n|\theta_k)$ is the component distribution. In order to stay in a correct probability world $0 \leq \pi_k \leq 1$ and $\sum_{k=1}^{K} \pi_k = 1$. The most well know mixture models is the Gaussian one in which we replace the component distribution $p(x_i|\theta_k)$ by

$$p(x_i|\pi,\mu,\sigma) = \sum_{k=1}^{K} \mathcal{N}(x_i|\mu_k,\sigma_k)$$

where $\mathcal{N}$ is the Normal or Gaussian law parameterized by μ, the d dimensional mean vector, and σ the $d \times d$ covariance matrix. In order to define θ, one of the most popular methods is to use the expectation maximization algorithm.

Expectation Maximization (EM) algorithm The Expectation Maximization (EM) algorithm [12, 71] is a generic approach to maximum likelihood when the user is faced with incomplete data. The overall likelihood of the training data is its probability to be drawn from a given mixture model.

$$\mathcal{V}(\boldsymbol{x}_1,...,\boldsymbol{x}_n;\theta) = \prod_{i=1}^{n} \sum_{j=1}^{k} \pi_j \varphi_j(\boldsymbol{x}_i;\alpha_j) \tag{2.10}$$

where $\varphi_j(\mathbf{x}_i;\alpha_j)$ represents the probability density. By introducing the log-likelihood, the Equation (2.10) can be rewritten as follows:

$$\mathcal{L}(\boldsymbol{x}_1,...,\boldsymbol{x}_n;\theta) = \sum_{i=1}^{n} \log\Big(\sum_{j=1}^{k} \pi_j \varphi_j(\boldsymbol{x}_i;\alpha_j)\Big) \tag{2.11}$$

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

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

The following process leads to finding mixture model parameters that maximize log-likelihood. It continues until log-likelihood convergence is complete. In [104, 95], the authors have proposed an estimation of probability distribution over a data set which is distributed into subsets located on the nodes of a distributed system. More precisely, the global distribution is estimated by aggregating local distributions which are modelled as a Gaussian mixture. The Gaussian Mixtures algorithm has the following properties:

- works on $\mathbb{R}^d$
- One main parameter K which defines the number of searched clusters
- Linear time complexity in n but quadratic in d, $\mathbf{O(k\ t\ n\ d^2)}$
  - $k = K$ is the number of clusters
  - $t \sim 100$ is the maximum number of iterations
  - n is the number of data points
  - d is the number of dimensions
- Scalable for $d \lessapprox 1000$

2.4 Distributed algorithms

Due to the interest in the MapReduce framework, some studies have used it for scaling clustering algorithms. As examples, we can cite the implementation of the K-means and EM algorithm in MapReduce [85, 69, 90]. Currently, more and more libraries have emerged offering MapReduce-based implementations of machine learning algorithms. **MLlib**[1] is Spark's well known machine learning library. It consists of common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, as well as lower-level optimization primitives and higher-level pipeline APIs.

2.4.1 K-Means

K-Means is probably one of the easiest algorithms to pass at scale thanks to its $O(n)$ time complexity and independence of most of its computations. The authors in [85] proposed a parallel and distributed implementation of k-means in MapReduce. The proposed algorithm, called PKMeans, is implemented using Hadoop[2] to make the clustering method applicable to large scale data. Let's say we have N machines (nodes) and each of them has a part X_i of the dataset X. Each computation of the assignation step is independent, which means that each machine N_i will be able to compute point assignation to its closest prototype c_i for all its data without requiring other information from $N_{j \neq i}$ machine at the exception of prototypes. It is the distributed *map* step. The map function is shown in Algorithm 11.

As for the prototype update, it requires shuffling but it is under control because of computation of mean is a commutative operation. It means that it can happen in any order. A *reduce* operation is used where keys are the cluster keys obtained

[1] http://spark.apache.org/docs/latest/mllib-guide.html
[2] http://lucene.apache.org/hadoop/

Algorithm 11 map(key, value)

```
    Input: Global variable clusters, the offset key, the point value
    Result: $< key', value' >$ pair, where the $key'$ is the index of the closest cluster and $value'$ is a string representing point (features)
 1: Construct the data-point $instance$ from $value$
 2: $minDist$ = Double.MAX_VALUE
 3: $index = -1$
 4: for each cluster $c_i \in C$ do
 5:     $dist = ComputeDistance(instance, c_i)$
 6:     if dist < minDist then
 7:         $minDist = dist$
 8:         $index = i$
 9: Take $index$ as $key'$
10: Construct $value'$ as a string describing features (different dimensions)
11: output $< key', value' >$ pair
```

from *map* step. For each key the sum (*reduce*) of vectors is done on each machine. Thus partial result vectors (maximum $K \times N$) are sent to the master node through network, where they are summed up and finally divided by their cluster cardinality to obtain K updated prototypes.The reduce function is shown in Algorithm 12.

Algorithm 12 reduce(key, V)

```
    Input:key is the index of the cluster, V is the list of the partial sums from different host
    Result: $< key', value' >$ pair, where the $key'$ is the index of the cluster and $value'$ is a string representing a new cluster prototype
 1: Initialize one array record with the sum of value of each dimensions of the points assigned to the same cluster, e.g. the points in the list $V$
 2: Initialize a counter NUM as $0$ to record the cardinality, which is the number of points in the same cluster
 3: for each value $v \in V$ do
 4:     Construct the point instance from $v$
 5:     Add the values of different dimensions of instance to the array
 6:     $NUM = NUM + num$
 7: Divide the entries of the array by NUM to provide the new cluster's prototype
 8: Take key as $key'$
 9: Construct $value'$ as a string comprise the cluster's prototype
10: output $< key', value' >$ pair
```

K-Modes

K-Modes follows same the rules as K-Means replacing mean computation by majority vote rules and using Hamming distance instead of the euclidian distance, as exposed in the local version of K-Modes 2.3.1.2.

K-Prototypes

K-Prototypes follows the same rules as K-Means and K-Modes. The mean and majority vote are respectively used for continuous and binary data, then operations can be distributed easily.

2.4.2 Self Organizing Maps

A distributed version of Self Organizing Maps was introduced in [113]. The way it is distributed is close to the K-Means; we can imagine the K prototypes replaced by as many prototype nodes in the grid. The pseudocode of [113] version is as follows.

1. Initialize the prototypes
2. **Map**: For each point $\mathbf{x}_i \in \mathcal{X}$
   (a) Assign $\mathbf{x}_i$ to its nearest cluster using the Euclidean distance, $dist = ComputeDistance(\mathbf{x}_i, c_i)$
   (b) Compute the numerator and the denominator of the expression (2.5) for each cluster c

$$MapNumerator_c = \mathcal{K}^T(\delta(c, \phi(\mathbf{x}_i)))\mathbf{x}_i$$

$$MapDenominator_c = \mathcal{K}^T(\delta(c, \phi(\mathbf{x_i})))$$

3. **Reduce**: Update the prototypes of all clusters by summing up the output of the Map tasks

$$\mathbf{w}_c = \frac{\sum_c MapNumerator_c}{\sum_c MapDenominator_c} \tag{2.12}$$

2.4.3 DBScan

Many distributed variants [110, 112, 87, 99, 121] have been proposed. A common approach is to divide the space into a grid, and apply a local version of DBScan in each cell. Thus, it gathers cells which works well in low dimension but it becomes exponentially more expensive with the increase of dimensions due to growing number of neighbors cells, increasing the necessary merging steps to compute. The representatives provided in each cell are merged in the global clustering step, a single-machine density-based clustering algorithm. Then the global clusters are sent back to all client nodes, which relabel all observations located on their site independently of each other.

In [110] the authors propose a MapReduce version (MR-DBSCAN) in which all critical sub-procedures are fully parallelized. The MR-DBSCAN algorithm consists of three stages: data partitioning, local clustering, and global merging. The first stage divides the whole dataset into smaller partitions according to spatial proximity. In the second stage, each partition is clustered independently. Then the partial clustering results are aggregated in the last stage to generate the global clusters. Experiments on large datasets confirm the scalability and efficiency of MR-DBSCAN.

2.4.4 PatchWork

PatchWork [123] is a distributed density clustering algorithm with linear computational complexity and linear horizontal scalability. It has desirable properties. It offers natural protection against outliers and noise, it can automatically detect the number of clusters, and it provides arbitrary cluster shapes. The main idea is to divide space with hypercube with edge of size $\varepsilon_d = \{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_d\}$. Thus, a cell is a hypercube in the D-dimensional feature space. A cell has a list of points that it contains and a reference to the cluster it belongs to. ε_d is a user-defined parameter that determines the cell size.

For each point, the *map* function returns the assigned tuple cell as key and 1 as value.

The collection of tuples (cell-id($\mathbf{x}$),1) is then shuffled using cell-id as key, and reduced using a sum operator. The output of the reduction step is a collection of tuples (cell-id,density). Cells with less than the *minPts*, user defined threshold, can be considered as a unique noisy cluster.

The collection of filtered cells is sorted by decreasing density. It creates a first cluster c with the first cell. Neighbor cells c_i are explored, and expand the current cluster if their density is sufficient, according to the formula: $Density(c_i) > Density(c) * Ratio$. This step is repeated until all cells are processed.

One of the limits of this algorithm is the fact it may only work on low dimensional space, because in high dimensional space, the number of cells increase exponentially preventing any scalable applications. In addition, like all density clustering algorithms, the choice of the parameter ε can impact the performance of the algorithm.

2.5 Conclusion

Many clustering algorithms have been proposed and make dealing with very large data a challenging task. The MapReduce paradigm has met with a resounding success in this era of Data Science due to, amongst others, its simplicity. The challenge in scaling clustering algorithm is not only to use the MapReduce paradigm but also to decompose the problem in small functions, the *map* and *reduce* functions. In the next chapter, we begin with the heart of this thesis, that focuses on the Mean Shift clustering algorithm, which combines both an unsupervised preprocessing approach, the gradient ascent and clustering algorithms applied after it. We also propose the use of random projection to decompose the problem for scaling the clustering algorithm.

Chapter 3

Mean Shift Like on continuous data

3.1 Introduction

Mean Shift clustering is a generalization of the k-means clustering which computes arbitrarily shaped clusters as defined as the basins of attraction to the local modes created by the density gradient ascent paths. Despite its potential, the Mean Shift approach is a computationally expensive method for unsupervised learning [6, 7, 26]. In this chapter we present improvements on many Mean Shift clustering aspects.

First of them is about density derivatives which are crucial components for statistical unsupervised learning based on density gradient ascent known as Mean Shift clustering. We will introduce nearest neighbour estimators of the general order derivatives of the probability density function and establish their squared error consistency, and most importantly for data analysis, an automatic, single pass normal scale or 'rule of thumb' selector of the number of nearest neighbours.

Nearest neighbour estimators of the probability density function were introduced in the seminal papers [2, 15] and have been widely used since due to their ease of implementation and interpretation. Derivatives of the density function are important quantities to analyse as they provide supplementary information about the data set, which is not revealed by the density function on its own. Estimators of the first derivative (gradient) have been considered [7], whereas higher order derivatives have not yet been considered. We set up a framework for nearest neighbour estimators for the general r-th order derivatives of multivariate density functions. This is achieved by following recent work in kernel estimators of density derivatives [106] and by exploiting the connection between nearest neighbour and variable kernel estimators [22]. Whilst variable kernel estimators of the density gradient [51, 41] are mathematically similar to their nearest neighbour analogues above, the key difference is that the former suffer from the data sparsity in higher dimensions whereas the latter do not. Modifications of kernel estimators have been proposed to overcome the data sparsity problem via reduced set density estimators [75, 52], though these authors did not consider the extension to density derivatives. We do not pursue this extension as we focus on nearest neighbour estimators.

Furthermore, current Mean Shift clustering algorithms contain computational bottlenecks with both kernel and nearest neighbor approaches: the former is due to the exact evaluation of the kernel function, and the latter due to the exact nearest neighbor searches. We propose a new algorithm NNGA$^+$, which resolves the computational inefficiencies of the nearest neighbor Mean Shift by using Locality

Sensitive Hashing (LSH) [33, 54, 77] for approximate nearest neighbor searches to replace the exact nearest neighbor calculations in the density gradient ascent and in the cluster labeling stages. Compared to kernel approaches to Mean Shift clustering, which are $O(n^2)$ where n is the size of the dataset, our nearest neighbors approach enables a scalable implementation of the gradient ascent and cluster labeling which are both $O(n)$.

Moreover, existing programming paradigms for dealing with parallelism, such as MapReduce [74] and Message Passing Interface (MPI) [25], have been demonstrated the best practical choices for implementing these clustering algorithms. MapReduce paradigm becomes popular and suited for data already stored on a distributed file system, which offers data replication, as well as the ability to execute computations, locally on each data node. Thus, we implement this approximate nearest neighbour Mean Shift clustering algorithm on a distributed Apache Spark/Scala framework [102], which allows us to carry out clustering on Big datasets.

3.2 Density derivative estimation

The nearest neighbour estimator of a density function, as introduced by [2] and elaborated by [15], is

$$\hat{f}(\boldsymbol{x}; k) = 1/[n\delta_{(k)}(\boldsymbol{x})^d] \sum_{i=1}^{n} K((\boldsymbol{x} - \boldsymbol{X}_i)/\delta_{(k)}(\boldsymbol{x})) \tag{3.1}$$

where $\boldsymbol{x} = (x_1, \ldots, x_d), \boldsymbol{X}_i = (X_{i1}, \ldots, X_{id})$, and $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n$ are a d-variate random sample drawn from a common density f. Eq. (3.1) is the mathematically most general form of a nearest neighbour density estimator as the kernel K can be any symmetric multivariate density function. The mathematical analysis of nearest neighbour estimators is simplified if we recast them as variable kernel estimators [22]. A variable multivariate kernel estimator $\tilde{f}$ with a variable bandwidth matrix function $\mathbf{H}(\boldsymbol{x})$ of the density is

$$\tilde{f}(\boldsymbol{x}; \mathbf{H}(\boldsymbol{x})) = n^{-1}|\mathbf{H}(\boldsymbol{x})|^{-1/2} \sum_{i=1}^{n} K(\mathbf{H}(\boldsymbol{x})^{-1/2}(\boldsymbol{x} - \boldsymbol{X}_i)),$$

and of the r-th density derivative is

$$\begin{aligned}&\mathsf{D}^{\otimes r}\tilde{f}(\boldsymbol{x}; \mathbf{H}(\boldsymbol{x})) \\ &\quad = n^{-1}|\mathbf{H}(\boldsymbol{x})|^{-1/2}(\mathbf{H}(\boldsymbol{x})^{-1/2})^{\otimes r} \sum_{i=1}^{n} \mathsf{D}^{\otimes r} K(\mathbf{H}(\boldsymbol{x})^{-1/2}(\boldsymbol{x} - \boldsymbol{X}_i))\end{aligned}$$

where the differentiation of K with respect to $\boldsymbol{x}$ is carried out keeping $\mathbf{H}(\boldsymbol{x})$ constant, and that the dependence on $\boldsymbol{x}$ is only reinstated after differentiation, employing an approach similar to [7]. The $\otimes r$ superscript indicates an r-fold Kronecker product, thus the r-th derivative $\mathsf{D}^{\otimes r}$ is organised as a d^r-vector arising from an r-fold Kronecker product of the differential operator $\mathsf{D} = [(\partial/\partial x_1), \ldots, (\partial/\partial x_d)]$, see [28]. The connection between nearest neighbour and variable kernel estimators [2, 15] appears when $\mathbf{H}(\boldsymbol{x}) = \delta_{(k)}(\boldsymbol{x})^2\mathbf{I}_d$. This implies that the nearest neighbour estimator of the r-th derivative of f follows as

$$\mathsf{D}^{\otimes r}\hat{f}(\boldsymbol{x};k) = n^{-1}\delta_{(k)}(\boldsymbol{x})^{-d-r}\sum_{i=1}^{n}\mathsf{D}^{\otimes r}K(\delta_{(k)}(\boldsymbol{x})^{-1}(\boldsymbol{x}-\boldsymbol{X}_i)). \tag{3.2}$$

Writing nearest neighbour estimators in this form in Eq. (3.2) greatly facilitates the task for optimal selection of the number of nearest neighbours.

The most common criterion utilised for optimal smoothing is the asymptotic mean integrated squared error (AMISE), which is the leading asymptotic term of the integral of the mean squared error between the target quantity and the estimator. We start with the AMISE of the fixed bandwidth kernel estimator $\mathsf{D}^{\otimes r}\tilde{f}(\cdot;\mathbf{H})$, i.e., $\mathrm{AMISE}\,[\mathsf{D}^{\otimes r}\tilde{f}(\cdot;\mathbf{H})]\{1+o(1)\} = \int_{\mathbb{R}^d}\mathbb{E}[\mathsf{D}^{\otimes r}\tilde{f}(\boldsymbol{x};\mathbf{H}) - \mathsf{D}^{\otimes r}f(\boldsymbol{x})]^2\,d\boldsymbol{x}$, as established by [89, Theorem 2]. This can be rewritten as

$$\begin{aligned}\mathrm{AMISE}\,[\mathsf{D}^{\otimes r}\tilde{f}(\cdot;\mathbf{H})] &= n^{-1}|\mathbf{H}|^{-1/2}\,\mathrm{tr}((\mathbf{H}^{-1})^{\otimes r}\mathbf{R}(\mathsf{D}^{\otimes r}K)) \\ &\quad + (-1)^r\tfrac{1}{4}m_2(K)^2\boldsymbol{\psi}_{2r+4}^T(\mathrm{vec}\,\mathbf{I}_{d^r}\otimes\mathrm{vec}\,\mathbf{H}^{\otimes 2})\end{aligned}$$

where $\mathbf{R}(\mathsf{D}^{\otimes r}K) = \int_{\mathbb{R}^d}\mathsf{D}^{\otimes r}K(\boldsymbol{x})\mathsf{D}^{\otimes r}K(\boldsymbol{x})^T\,d\boldsymbol{x}$, and $m_2(K)\mathbf{I}_d = \int_{\mathbb{R}^d}\boldsymbol{x}\boldsymbol{x}^TK(\boldsymbol{x})\,d\boldsymbol{x}$ and $\boldsymbol{\psi}_{2r+4} = \int_{\mathbb{R}^d}\mathsf{D}^{\otimes(2r+4)}f(\boldsymbol{x})f(\boldsymbol{x})\,d\boldsymbol{x}$.

Replacing $\mathbf{H}$ by $\mathbf{H}(\boldsymbol{x}) = \delta_{(k)}(\boldsymbol{x})^2\mathbf{I}_d$ results in a random quantity, so we compute its expectation to derive an AMISE-like quantity for the nearest neighbour density derivative estimator $\mathsf{D}^{\otimes r}\hat{f}(\cdot;k)$,

$$\begin{aligned}&\mathrm{A}[\mathsf{D}^{\otimes r}\hat{f}(\boldsymbol{x};k)] \\ &\quad= \mathbb{E}\{\mathrm{AMISE}\,[\mathsf{D}^{\otimes r}\tilde{f}(\cdot;\delta_{(k)}(\boldsymbol{x})^2\mathbf{I}_d)]\} \\ &\quad= \mathrm{tr}(\mathbf{R}(\mathsf{D}^{\otimes r}K))[v_0f(\boldsymbol{x})]^{(d+2r)/d}n^{2r/d}k^{-(d+2r)/d} \\ &\qquad + (-1)^r\tfrac{1}{4}m_2(K)^2\boldsymbol{\psi}_{2r+4}^T(\mathrm{vec}\,\mathbf{I}_d)^{\otimes(r+2)}[v_0f(\boldsymbol{x})]^{-4/d}n^{-4/d}k^{4/d}\end{aligned} \tag{3.3}$$

where $v_0 = \pi^{d/2}\Gamma((d+2)/d)$ is the hyper-volume of the unit d-ball.

As the first term is the integrated variance and the second term is the integrated squared bias of $\mathsf{D}^{\otimes r}\hat{f}$, the role of k in a bias-variance trade-off is established in Eq. (3.3). So $\mathrm{A}[\mathsf{D}^{\otimes r}\hat{f}(\boldsymbol{x};k)]$ is a suitable basis for an optimality criterion. We obtain in Theorem 1 a closed form expression of

$$k_{\mathrm{A},r} = \int_{\mathbb{R}^d}\Big\{\operatorname*{argmin}_{k>0}\,\mathrm{A}[\mathsf{D}^{\otimes r}\hat{f}(\boldsymbol{x};k)]\Big\}\,d\boldsymbol{x}$$

which serves as an optimal number of the nearest neighbours.

Theorem 1. *An optimal number of the nearest neighbours for* $\mathsf{D}^{\otimes r}\hat{f}$ *is* $k_{\mathrm{A},r} = C_rn^{(2r+4)/(d+2r+4)}$ *where*

$$C_r = v_0\left[\frac{(d+2r)\,\mathrm{tr}(\mathbf{R}(\mathsf{D}^{\otimes r}K))}{(-1)^rm_2(K)^2\boldsymbol{\psi}_{2r+4}^T(\mathrm{vec}\,\mathbf{I}_d)^{\otimes(r+2)}}\right]^{d/(d+2r+4)}.$$

When $K = f = \phi$*, where* ϕ *is the standard normal density, this yields the normal scale selector*

$$k_{\mathrm{NS},r} = v_0\left[\frac{4}{d+2r+2}\right]^{d/(d+2r+4)}n^{(2r+4)/(d+2r+4)}.$$

This $k_{\mathrm{NS},r}$ is closely related to the normal scale bandwidth selector $[4/(d+2r+2)]^{2/(d+2r+4)}\, n^{-2/(d+2r+4)}\mathbf{I}_d$ for the kernel estimator $\mathsf{D}^{\otimes r}\tilde{f}$ in [89, Theorem 6].

Figure 3.1 illustrates the importance of selecting a suitable value of k. The contours of the target density gradient $(r = 1)$ of the standard normal bivariate density is in Figure 3.1(a). The nearest neighbour estimate with the normal scale selector $k = k_{\mathrm{NS},1} = 505$ in Figure 3.1(b) produces a similar structure as the target contours. If a much smaller $k = 50$ is utilised, the resulting estimate in Figure 3.1(c) is considered to be undersmoothed as it is too noisy. If a much larger $k = 1000$ is utilised, the resulting estimate in Figure 3.1(d) is considered to be oversmoothed as it displays insufficient detail. The kernel utilised is the Epanechnikov kernel $K(\boldsymbol{x}) = [(d+2)/(2v_0)](1 - \boldsymbol{x}^T\boldsymbol{x})\mathbf{1}\{\|\boldsymbol{x}\| \le 1)\}$.

The pioneering work of [6, 2] established the oracle local and global mean squared error optimal selectors for density estimators, though these authors did not consider data-based selectors. Perhaps [16] is the first to consider automatic data-based selection for nearest neighbour estimators, in the context of cross validation for regression. Authors who have proposed cross validation selectors for density estimation include [88, 97]. The latter authors [97] also suggest a grid based search for k. We observe that these are multiple passes methods. In contrast, we propose an efficient, single pass fully automatic selector for the nearest neighbour estimator of a general order r of the density derivative in Theorem 1.

3.2.1 Nearest neighbor Mean Shift clustering for continuous data

The Mean Shift clustering proceeds in an indirect manner based on local gradients of the data density, and without imposing an ellipsoidal shape to clusters or that the number of clusters be known, as is the case for k-means clustering. For a candidate point $\boldsymbol{x}$, the theoretical Mean Shift recurrence relation is

$$\boldsymbol{x}_{j+1} = \boldsymbol{x}_j + \frac{\mathbf{A}\mathsf{D}f(\boldsymbol{x}_j)}{f(\boldsymbol{x}_j)} \tag{3.4}$$

for a given positive-definite matrix $\mathbf{A}$, for $j \ge 1$ and $\boldsymbol{x}_0 = \boldsymbol{x}$. The output from Equation (3.4) is the sequence $\{\boldsymbol{x}_j\}_{j\ge 0}$ which follows the density gradient ascent $\mathsf{D}f$ to a local mode of the density function f.

To derive the formula for the nearest neighbor Mean Shift for a random sample $\boldsymbol{X}_1, \dots, \boldsymbol{X}_n$ drawn from a common density f, we replace the density f and density gradient $\mathsf{D}f$ by their nearest neighbor estimates

$$\hat{f}(\boldsymbol{x}; k) = n^{-1}\delta_{(k)}(\boldsymbol{x})^{-d} \sum_{i=1}^{n} \frac{K((\boldsymbol{x} - \boldsymbol{X}_i))}{\delta_{(k)}(\boldsymbol{x})}$$

$$\mathsf{D}\hat{f}(\boldsymbol{x}; k) = n^{-1}\delta_{(k)}(\boldsymbol{x})^{-d-1} \sum_{i=1}^{n} \frac{\mathsf{D}K((\boldsymbol{x} - \boldsymbol{X}_i)}{\delta_{(k)}(\boldsymbol{x}))}$$

where K is a kernel function and $\delta_{(k)}(\boldsymbol{x})$ as the k-th nearest neighbor distance to $\boldsymbol{x}$, i.e. $\delta_{(k)}(\boldsymbol{x})$ is the k-th order statistic of the Euclidean distances $\|\boldsymbol{x} - \boldsymbol{X}_1\|, \dots, \|\boldsymbol{x} -$

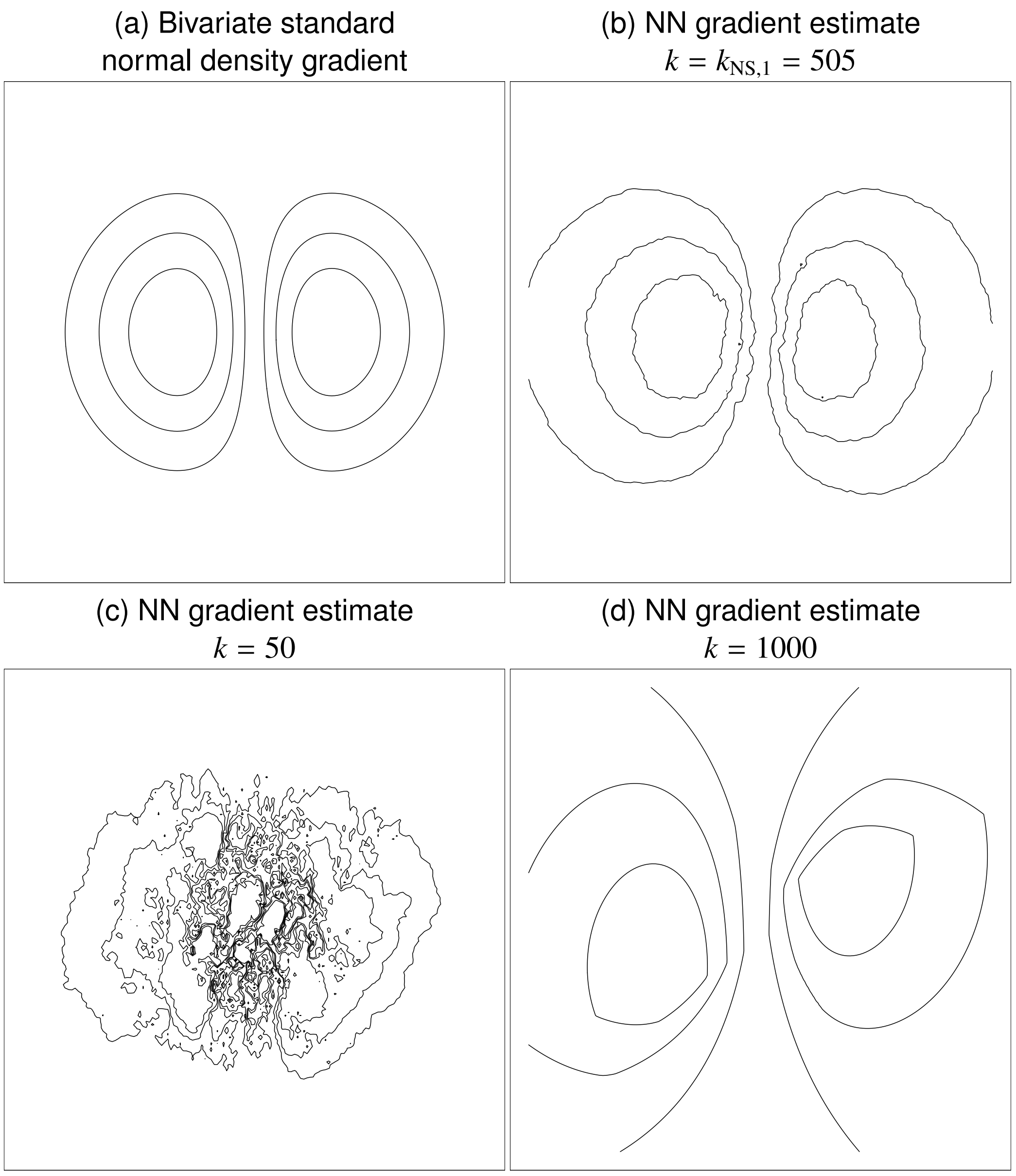


Figure 3.1: Effect of the choice of k for nearest neighbour density gradient estimates for an $n = 1000$ random sample from the bivariate standard normal density. (a) Contours of the target bivariate standard normal density gradient. (b) Nearest neighbour density gradient estimate with $k = k_{\mathrm{NS},1} = 505$. (c) Nearest neighbour density gradient estimate with $k = 50$. (d) Nearest neighbour density gradient estimate with $k = 1000$.

$\boldsymbol{X}_n\|$. These nearest neighbor estimators were introduced by [2] and elaborated by [6, 7] for the Mean Shift. These authors established that the beta family kernels are computationally efficient for estimating f and $\mathsf{D}f$ for continuous data. The uniform kernel is the most widely known member of this beta family, and it is defined as

$$K(\boldsymbol{x}) = v_0^{-1}\mathbf{1}\{\boldsymbol{x} \in B_d(\mathbf{0}, 1)\}$$

where $B_d(\boldsymbol{x}, r)$ is the d-dimensional hyper-ball centered at $\boldsymbol{x}$ with radius r and v_0 is the hyper-volume of the unit d-dimensional hyper-ball $B_d(\mathbf{0}, 1)$. The summation counts the number of data points which fall inside $B_d(\boldsymbol{x}, \delta_{(k)}(\boldsymbol{x}))$, which is equal to k from the definition of $\delta_{(k)}(\boldsymbol{x})$ as the k-th nearest neighbour distance to $\boldsymbol{x}$. The nearest neighbour density estimator in Eq. (3.1) becomes

$$\hat{f}(\boldsymbol{x}; k) = n^{-1} \sum_{i=1}^{n} \mathbf{1}\{\boldsymbol{X}_i \in B_d(\boldsymbol{x}, \delta_{(k)}(\boldsymbol{x}))\} = k/[v_0 n \delta_{(k)}(\boldsymbol{x})^d]$$

The nearest neighbour estimator in Eq. (3.2) for the density gradient becomes

$$\begin{aligned} \mathsf{D}\hat{f}(\boldsymbol{x}; k) &= \hat{f}(\boldsymbol{x}; k) \frac{d+2}{\delta_{(k)}(\boldsymbol{x})^2} \left[\frac{1}{k} \sum_{i=1}^{n} \boldsymbol{X}_i \mathbf{1}\{\boldsymbol{X}_i \in B_d(\boldsymbol{x}, \delta_{(k)}(\boldsymbol{x}))\} - \boldsymbol{x} \right] \\ &= \hat{f}(\boldsymbol{x}; k) \frac{d+2}{\delta_{(k)}(\boldsymbol{x})^2} \left[\frac{1}{k} \sum_{\boldsymbol{X}_i \in k\text{-nn}(\boldsymbol{x})} \boldsymbol{X}_i - \boldsymbol{x} \right] \end{aligned}$$

where $k\text{-nn}(\boldsymbol{x}) = \{\boldsymbol{X}_i : \boldsymbol{X}_i \in B_d(\boldsymbol{x}, \delta_{(k)}(\boldsymbol{x}))\}$ is the set of the k nearest neighbours to $\boldsymbol{x}$, when using the first order beta kernel $K(\boldsymbol{x}; 1) = [(d+2)/(2v_0)](1 - \boldsymbol{x}^T \boldsymbol{x}) \mathbf{1}\{\boldsymbol{x} \in B_d(\mathbf{0}, 1)\}$ is the Epanechnikov (or quadratic) kernel, with derivative $\mathsf{D}K(\boldsymbol{x}; 1) = -[(d+2)/v_0] \boldsymbol{x}\, \mathbf{1}\{\boldsymbol{x} \in B_d(\mathbf{0}, 1)\}$.

Replacing $\mathsf{D}f(\boldsymbol{x})/f(\boldsymbol{x})$ by its estimator $\mathsf{D}\hat{f}(\boldsymbol{x}; k)/\hat{f}(\boldsymbol{x}; k)$ in Eq. (3.4) and the choice $\mathbf{A} = (d+2)^{-1} \delta_{(k)}(\boldsymbol{x}) \mathbf{I}_d$, the nearest neighbor Mean Shift becomes

$$\boldsymbol{x}_{j+1} = \frac{1}{k} \sum_{\boldsymbol{X}_i \in k\text{-nn}(\boldsymbol{x}_j)} \boldsymbol{X}_i \tag{3.5}$$

This nearest neighbor Mean Shift has a simple interpretation since in the Mean Shift recurrence relation, the next iterate $\boldsymbol{x}_{j+1}$ is the sample mean of the k nearest neighbors of the current iterate $\boldsymbol{x}_j$. On the other hand, as these iterations calculate the sample mean, the Mean Shift is not directly applicable to binary data. The gradient ascent paths towards the local modes produced by Eq. (3.5) form the basis of Algorithm 13, our nearest neighbor Mean Shift gradient ascent (NNGA).

The inputs to the NNGA are the data sample $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n$ and the candidate points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$, which we want to cluster (these can be the same as $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n$, but this is not required); and the tuning parameters: the number of nearest neighbors k, the tolerance under which subsequent iterations in the Mean Shift update are considered to be convergent ε_1, the maximum number of iterations $j_{\max}$.

Algorithm 13 NNGA – Nearest Neighbor Gradient Ascent with exact k-nn

```
    Input: ${X_1, ..., X_n}, {x_1, ..., x_m}, k, \varepsilon_1, j_{\max}$
    Output: ${x_1^*, ..., x_m^*}$
1:  Compute similarity matrix and sort each row
2:  for $\ell := 1$ to $m$ do
3:      $j := 0; x_{\ell,0} := x_\ell$;
    /* Search for $k$-$nn$ based on similarity matrix */
4:      $x_{\ell,1}$ := mean of $k$-nn$_k(x_{\ell,0})$
5:      while $||x_{\ell,j+1}, x_{\ell,j}|| > \varepsilon_1$ or $j < j_{\max}$ do
6:          $j := j + 1$
7:          $x_{\ell,j+1}$ := mean of $k$-nn$_k(x_{\ell,j})$
8:      $x_\ell^* := x_{\ell,j}$
```

The classical version of the NNGA introduced in Algorithm 13 requires, for each candidate point that we compute, the distance to all other data points, from which the mean of the k nearest neighbors is set to be the current prototype. The algorithm associates this prototype with the original candidate points. We repeat this step until the prototype moves less than a threshold ε_1 or whenever the algorithm have reached $j_{\max}$ iterations.

The complexity for the exact nearest neighbors search of a single point is $n\log(n)$. Applied to every data point multiple times, this complexity increases to $n^2 j_{\max}\log(n)$, preventing its application on massive datasets.

3.2.2 Approximate nearest neighbors search for density gradient ascent

One promising algorithmic complexity reduction approach relies on computing approximate nearest neighbors rather than exact neighbors. Among the techniques that can be used, Locality Sensitive Hashing (LSH), was introduced in [33, 54]. Many hashing algorithms have been proposed in literature [37, 98, 84], and among these LSH is considered as the most representative and popular one. LSH is presented as a probabilistic similarity-preserving dimensionality reduction method, and based on the adopted distances and similarities, including l_p distance [55], angular distance [47], Hamming distance [34], Jaccard coefficient [29], etc., different types of LSH can be designed; which also depends on the types of the used data [37]. Many variants are developed based on these basic LSH families such as Spectral hashing [79] Kernelized spectral hashing [86], and independent component analysis (ICA) Hashing [91]. These methods aim at learning the hash functions for better fitting the data distribution [116].

In this thesis we are based on LSH presented on [33, 54] which is a probabilistic method based on a random scalar projection of multivariate data point $\boldsymbol{x}$ defined below:

$$L(\boldsymbol{x}; v) = (\boldsymbol{Z}^T\boldsymbol{x} + U)/v$$

where $\boldsymbol{Z} \sim N(0, \mathbf{I}_d)$ is a standard d-variate normal random variable and $U \sim \mathrm{Unif}(0, v)$ is a uniform random variable on $[0, v)$, $v > 0$. The LSH is parametrized by the number of buckets M_1 in the hash table. In our context, we propose to set

$v = 1$, and without loss of generality $L_i \equiv L(\boldsymbol{X}_i; 1)$. These scalar projections are sorted into their order statistics $L_{(1)} < \cdots < L_{(n)}$, and their range is divided into M_1 partition intervals of width $w = (L_{(n)} - L_{(1)})/M_1$ where $I_j = [L_{(1)} + w(j-1), L_{(1)} + wj]$, $\forall j \in \{j = 1, \ldots, M_1\}$. The hash value of $\boldsymbol{x}$ is the index of the interval in which $L(\boldsymbol{x}; 1)$ falls

$$H(\boldsymbol{x}) = j\mathbf{1}\{L(\boldsymbol{x}; 1) \in I_j\}$$

where $\mathbf{1}\{\cdot\}$ is the indicator function. To search for approximate nearest neighbors, the reservoir of potential nearest neighbors is set to the bucket which contains the hash value. This reservoir is enlarged if necessary by concatenating the adjacent buckets. The approximate k nearest neighbors of $\boldsymbol{x}$ are the k nearest neighbors only drawn from the reduced reservoir $R(\boldsymbol{x})$ defined as below:

$$k\text{-}\widetilde{\text{nn}}(\boldsymbol{x}) = \{\boldsymbol{X}_i \in R(\boldsymbol{x}) : \|\boldsymbol{x} - \boldsymbol{X}_i\| \leq \delta_{(k)}(\boldsymbol{x})\},$$

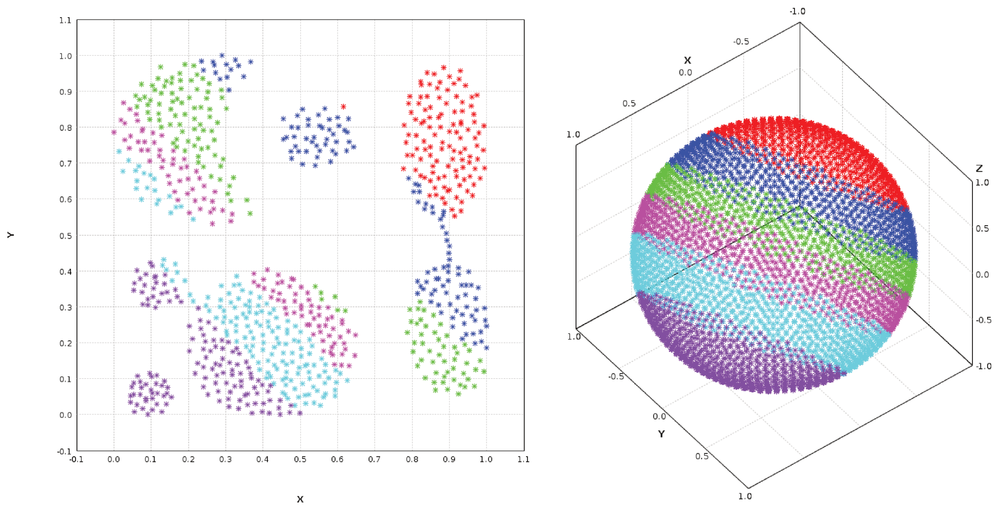
where $\delta_{(k)}(\boldsymbol{x})$ is the kth nearest neighbor distance to $\boldsymbol{x}$. The approximation error in the nearest neighbors to $\boldsymbol{x}$ induced by searching in $R(\boldsymbol{x})$ rather than the full dataset is probabilistically controlled [77]. Our improvement to the classical LSH is based on the following observations:

- it takes into account the properties of the local data space more accurately. Rather than looking exclusively in the bucket where the prototype lies, we also look in the adjacent buckets on both sides. The main advantage of this is to take into account the case where a prototype is at the border of a bucket and so some of its k nearest neighbors mostly likely fall into the adjacent buckets. This is especially important for high values of k. It is computationally more expensive but the cost can be controlled for a fixed bucket size. The memory cost is increased by a factor of 3 per partition due to copying the adjacent layers into the active one.

  The LSH method partitions the data space into buckets of approximately k nearest neighbors, which are delimited by parallel hyperplanes. In practice, the LSH controls the number of neighboring buckets to two, except for the edge buckets which have only one neighbor bucket. This is in contrast to cell based buckets, where the number of neighbor buckets increases exponentially with the number of dimensions. Figure 3.2 illustrates the LSH buckets of approximate nearest neighbors on 2D (Aggregation) and 3D (GolfBall) data examples: the orientation of hyperplanes depends on the random projections utilized to construct the buckets.

- it adds the possibility of allowing the prototype to change buckets during its gradient ascent. In this case, we look for its k_2 nearest neighbors in order to place the prototype within the most representative bucket using a majority voting process. Thus a prototype can pass through multiple buckets before converging to its final position, as illustrated in Figure 3.3.

Algorithm 14 describes the NNGA$^+$, an approximate nearest neighbor search using LSH with the hash function H. The inputs are the data sample $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n$, the candidate points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$, and the tuning parameters: the number of nearest neighbors k_1 and the number of buckets in the hash table M_1. In line 1, the hash



(a) Aggregation with $M_1 = 6$ buckets. **(b)** GolfBall with $M_1 = 6$ buckets.

Figure 3.2: LSH buckets for the Aggregation and GolfBall datasets.

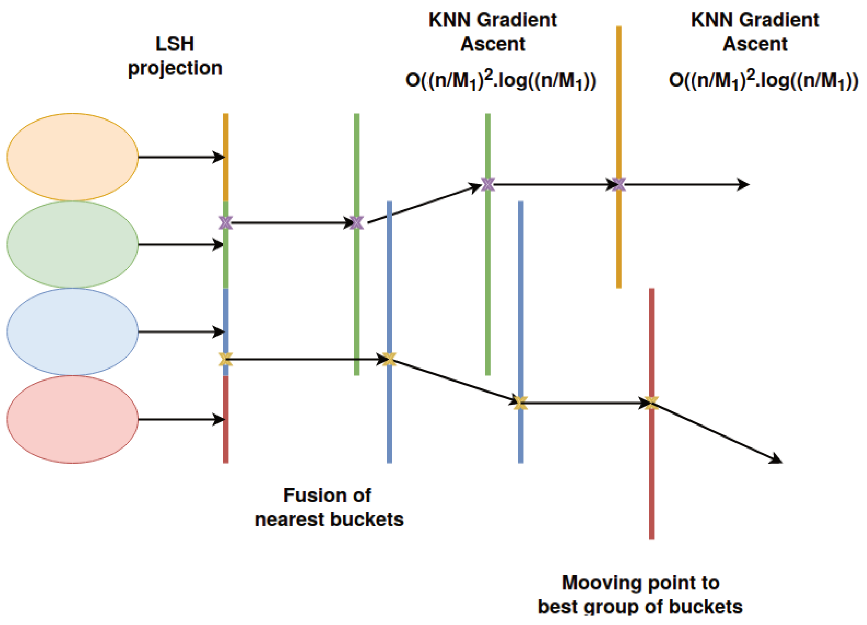


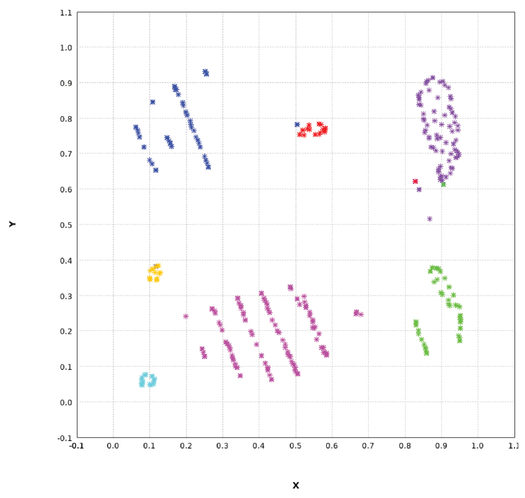
Figure 3.3: Passage of a prototype through different LSH buckets during the gradient ascent; n is the number of data points and M_1 is the number of buckets.

table is created by applying the LSH to the data values $X_1, \dots, X_n$. In lines 2–6, for each candidate point x_ℓ, the approximate k_1 nearest neighbors $k\text{-}\widetilde{\mathrm{nn}}(x_\ell)$ are computed from within the reservoir $R(x_\ell)$.

Figure 3.4 illustrates the effect of including adjacent buckets in NNGA$^+$ versus nearest neighbour gradient ascent without adjacent buckets on the Aggregation dataset. For $k_1 = 20$ without adjacent buckets, in Figure 3.4a, then we observe that the data are artificially forced to follow the hyperplanes which delimit the different buckets. Figure 3.4b shows our algorithmic improvement with adding one layer of adjacent buckets where the underlying structure of data is maintained.

Algorithm 14 NNGA$^+$ – Approximate Nearest Neighbors Gradient Ascent with LSH and adjacent buckets

```
   Input: $\{X_1, \dots, X_n\}, \{x_1, \dots, x_m\}, k_1, M_1$
   Output: $\{k\text{-}\widetilde{\text{nn}}(x_1), \dots, k\text{-}\widetilde{\text{nn}}(x_m)\}$
   /* Create hash table with $M_1$ buckets */
1: for $i := 1$ to $n$ do $H_i := H(X_i)$;
   /* Search for approx $nn$ in adjacent buckets */
2: for $\ell := 1$ to $m$ do
3:     $R(x_\ell) := \{X_i : H_i = H(x_\ell), i \in \{1, \dots, n\}\}$
4:     while $\text{card}(R(x_\ell)) < k_1$ do
5:         $R(x_\ell) := R(x_\ell) \cup$ neighbor bucket;
6:     $k\text{-}\widetilde{\text{nn}}(x_\ell) := k$-nn from $R(x_\ell)$ to $x_\ell$;
```


(a) Aggregation after NNGA$^+$ with number of nearest neighbors $k_1 = 20$, without adjacent buckets.

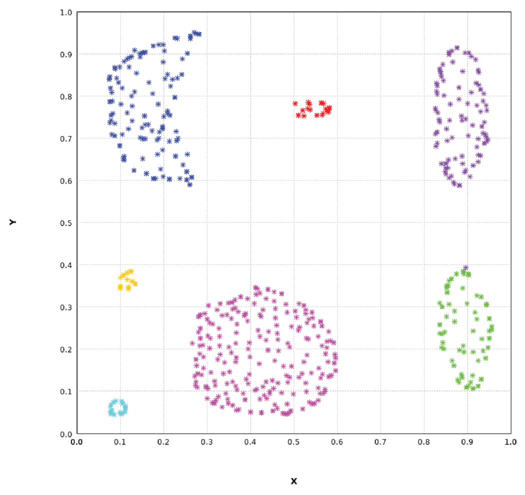

(b) Aggregation after NNGA$^+$ with number of nearest neighbors $k_1 = 20$, with adjacent buckets.

Figure 3.4: NNGA$^+$ without and with adjacent buckets for the Aggregation dataset.

Whilst the use of the LSH to reduce the complexity of kernel Mean Shift clustering was already proposed in [92], these authors did not quantify the reduction in complexity. The complexity of our NNGA$^+$ is reduced to $O((\frac{n}{M_1})^2 \log(\frac{n}{M_1}))$ per bucket with M_1 buckets, and so the total complexity is $O((\frac{n}{M_1})^2 \log(\frac{n}{M_1}))$ for all buckets. Because of this segmentation of the original data space into M_1 sub-spaces, the complexity is inversely proportional to the number of buckets. The trade-off is that the data points in each bucket have to be sufficiently representative of the local properties of the original space. Thus the number of buckets M_1 is a crucial tuning parameter. Despite this, there are no optimal methods for selecting the number of buckets [101]. Consequently, we will examine empirical choices of the number of buckets in the sequel.

3.2.3 Cluster labeling: ε-proximity

NNGA$^+$ carries out the gradient ascent on the data points until they have converged to their prototypes. The question is then how to assign cluster labels to the data points. A first solution is to assign the same cluster to all points sharing the same prototype. As observed in Figure 3.4b, even with a prior good choice of k_1, there are (possibly) hundreds of generated prototypes, so assigning a label to each point according to its closest prototype is not effective because it can generate too many clusters. Applying the density gradient ascent NNGA$^+$ leads to a converged dataset with increased inter-cluster distances and decreased intra-cluster distances as compared to the original dataset. In order to further exploit this property, we propose a new proximity-based approach where points which are under a threshold $\varepsilon > 0$ from each other are considered to belong to the same cluster.

Algorithm 15 illustrates ε-proximity cluster labeling. It consists in exploring the similarity matrix $\mathbf{S}$ which is defined as a map whose objects IDs are the keys and whose pairs (object IDs, distances) are the values. We initialize the process by taking the first object of $\mathbf{S}$ and cluster with it every point whose distance is less than ε. We then apply this exploration process by iteratively adding the k_2 nearest neighbors of these added points until this process terminates. During the process we remove the explored points from $\mathbf{S}$ to avoid repeated calculations. Once the first cluster is generated, we take another object from outside this first cluster from the reduced similarity matrix $\mathbf{S}$ and repeat the above cluster formation, until all objects are assigned to a cluster label.

In order to apply this algorithm, we have to build the similarity matrix which has a $O(n^2)$ time complexity, preventing any Big Data application. A scalable version consists of applying this algorithm in each LSH partition, and merging each bucket its right or left adjacent bucket to maintain the bucket order.

Once this step is completed, we apply a *MapPartitions* procedure to check if two clusters of two different buckets share at least 1 pair of points which are less than ε apart, then these two clusters are considered to form a single cluster. We obtain a dataset which chains common clusters between partitions: all chained clusters are assigned with the same label by generating an undirected graph where each connected subgraph represents a cluster. The search for connected components in a graph is a common problem which can be solved in linear time in the number of vertices.

It is important to bound the number of data points in each bucket because the scalable version of ε-proximity clustering and the check for cluster merging between two buckets have quadratic complexity in this size. Empirically we advise to set the number of buckets M_1 in order to have around 500 to 2000 data points in each bucket.

A notable problem still remains with the choice of the main tuning parameter ε. We set it to be the average of distance from each point to their k nearest neighbors. We compute it as an approximate value in using LSH procedure in order to maintain the scalability property.

Algorithm 15 ε-proximity labeling

```
    Input: $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_m\}, \mathbf{S}, \varepsilon$
    Output: $\{\tilde{c}(\boldsymbol{x}_1), \dots, \tilde{c}(\boldsymbol{x}_m)\}$
 1: needToVisit ← Set(S.head)
 2: $c_{ID}$ ← 0
 3: clusters ← Map.empty[Int, Set[Int]]
 4: clusters += ($c_{ID}$, needToVisit)
 5: while $\mathbf{S}$ has elements do
    /* $p_c$ is the current point of needToVisit */
 6:     for each $p_c$ in needToVisit do
    /* Add all points under $\varepsilon$ to needToVisit */
 7:         needToVisitUpdated ← $\{p \in \mathbf{S}, \text{dist}(p, p_c) \leq \varepsilon\}$
    /* Remove explored point from the similarity matrix */
 8:         $\mathbf{S}$ -= $p_c$
    /* Update points to explore on next iterations */
 9:         needToVisit ← needToVisitUpdated
10:         if neetToVisit is empty then
11:             $c_{ID}$ += 1
12:             needToVisit ← Set(S.head)
    /* Create a new entry in the clusters map */
13:             clusters += ($c_{ID}$, needToVisit)
14:         else
    /* Add new points to cluster $c_{ID}$ */
15:             clusters($c_{ID}$) += needToVisit
```

3.3 Experiments

Our experiments are carried out on the Grid'5000 testbed which is the French national testbed for computer science research. We use a dedicated Spark Linux image optimized for Grid'5000 where Apache Spark is deployed on top of Spark in Standalone mode. Apache Spark is a fast general purpose distributed computing system based on a master-slaves architecture. Only the deployment of the image is automatized. We manually reserve the nodes and provide the Spark cluster with our code to execute the different experiments on a 2×8 core Intel Xeon E5-2630v3 CPUs and 128 Gb RAM setup.

A key concept in Spark is the resilient distributed dataset (RDD) which is a read-only collection of objects partitioned across a group of machines which can be rebuilt if necessary from the hierarchy of previous RDD operations. Most of the *Map* and *Reduce* operations will be performed on RDDs even if other pure Scala *Map* and *Reduce* operations are executed inside each Spark partition. We implement our algorithm in Scala because it is the Spark's native language and thus allows for good performance.

We use a range from 2 to high dimensional datasets with different sizes, as summarized in Table 3.1 [61, 119]. To ensure the comparability of the results across these different datasets, all algorithms are carried out on the normalized version of the datasets: $x_i = (x_i - x_i^{\min})/(x_i^{\max} - x_i^{\min})$ where x_i is the ith component of $\boldsymbol{x}$, and $x_i^{\min}$, $x_i^{\max}$ are respectively the ith marginal minimum and maximum values. We repeat each experiment ten times for robustness.

Dataset	n	d	N
R15	600	2	15
Aggregation	788	2	7
Sizes5	1000	2	4
EngyTime	4096	2	2
Banana	4811	2	2
S3	5000	2	15
Disk6000	6000	2	2
Unbalance	6500	2	8
DS1	9153	2	14
Hepta	212	3	7
Hyperplane	100000	10	5
CovType10	581012	10	7
ScalabiltyDS	140000000	10	–

Table 3.1: Experimental datasets. n is the dataset size, d is the data dimension, N is the number of clusters.

3.3.1 Comparison of NNGA$^+$ with k-means and DBScan

In this section, we compare NNGA$^+$ (Algorithm 14) with ε-proximity labeling (Algorithm 15) to k-means [94] and DBScan [111] presented in previous chapter. Table 3.2 shows the optimal combinations of tuning parameters used in the comparison between the gradient ascent (NNGA$^+$) with ε-proximity, k-means, DBScan and ε-proximity. These optimal combinations are obtained by grid searches over a range of values of each parameter.

To evaluate the quality of the clustering, we use both the Normalized Mutual Information (NMI) [59] and the Rand index [81], as displayed in Table 3.3. The bold values are the highest value within each row. The value of each measure lies between 0 and 1. A higher value indicates better clustering results. The conclusions from both of these indices are similar for all datasets, except for the large datasets Hyperplan $n = 100000$ and CovType10 $n = 581012$. NNGA$^+$ with ε-proximity achieves the highest clustering accuraries for 7 out of the 12 experimental datasets. For those datasets where it is not the best performing, it is close behind, with the only possible exception for Disk6000, whose nested clusters are difficult for NNGA$^+$ to detect accurately.

3.3.2 Effect of tuning parameters on clustering accuracy and execution time

Table 3.4 illustrates clustering quality results with NMI, Rand and time duration exclusively on DS1 dataset. We present three rows, for three parameters ε_1, M_1, and k_1, on which for each of them we fix two parameters and change the third one.

Dataset	NNGA$^+$ with ε-proximity	k-means	DBScan
Aggregation	$k_1 = 50, M_1 = 8, \varepsilon_1 = 30$	$k = 7$	$\varepsilon = 0.05, minPts = 8$
Banana	$k_1 = 40, M_1 = 8, \boldsymbol{\varepsilon_1 = 0.1}$	$k = 2$	$\varepsilon = 0.02, minPts = 3$
Disk6000	$k_1 = 100, M_1 = 8, \boldsymbol{\varepsilon_1 = 0.02}$	$k = 7$	$\varepsilon = 0.02, minPts = 4$
DS1	$k_1 = 50, M_1 = 8, \varepsilon_1 = 30$	$k = 14$	$\varepsilon = 0.03, minPts = 25$
EngyTime	$k_1 = 200, M_1 = 8, \varepsilon_1 = 50$	$k = 2$	$\varepsilon = 0.1, minPts = 100$
Hepta	$k_1 = 20, M_1 = 4, \varepsilon_1 = 10$	$k = 7$	$\varepsilon = 0.1, minPts = 10$
R15	$k_1 = 20, M_1 = 8, \varepsilon_1 = 5$	$k = 15$	$\varepsilon = 0.05, minPts = 25$
S3	$k_1 = 40, M_1 = 8, \varepsilon_1 = 15$	$k = 15$	$\varepsilon = 0.05, minPts = 50$
Sizes5	$k_1 = 20, M_1 = 8, \varepsilon_1 = 5$	$k = 4$	$\varepsilon = 0.08, minPts = 8$
Unbalance	$k_1 = 40, M_1 = 8, \varepsilon_1 = 80$	$k = 8$	$\varepsilon = 0.05, minPts = 20$
Hyperplane	$k_1 = 50, M_1 = 100, \varepsilon_1 = 5$	$k = 5$	$\varepsilon = 0.05, minPts = 8$
CovType10	$k_1 = 50, M_1 = 500, \varepsilon_1 = 20$	$k = 5$	$\varepsilon = 0.05, minPts = 8$

Table 3.2: Optimal tuning parameter choices for clustering comparisons. k_1 is the number of nearest neighbors in the gradient ascent, M_1 is the number of buckets in the LSH, ε_1 is the distance threshold for the ε-proximity labeling, bold values are manually settled ε_1, normal ones correspond to approximate average value of its "ε_1 value" k nearest neighbors distance, for NNGA$^+$ with ε-proximity; k is the number of clusters for k-means; ε is the radius of the hypersphere and $minPts$ is the minimum of number of data points for DBScan.

As expected from the algorithm design the only parameter which influences strongly experiences duration is the number of buckets M_1 used for the LSH. The larger M_1 is the faster one algorithm run becomes. It is due to quadratic complexity operations that remain in each bucket as well as for gradient ascent for local ε-proximity. Increasing the buckets number will decrease number of elements per bucket and then greatly decrease needed computation time per bucket. Concerning link with accuracy scores, they stay relatively stable for Rand and NMI.

We observe that specific combinations of parameters of ε_1 and k_1 perform better than others in Table 3.4. Higher values of k_1 will result in the fusion of closest clusters into bigger ones while smaller values will smooth clusters shapes. ε-proximity clustering algorithm will precisely give results depending of how main parameter ε_1 is settled. Smaller ε_1 values will provide much more clusters because less points tend to be closer from each others with a small ε_1 value. At the opposite bigger ε_1 values will generate less clusters with more points. It is decisive on some indices as the NMI. Too many discovered clusters compared to ground truth classes number will drastically reduce this index score even if there is a belonging logic with original classes, fortunately the Rand index suggests that there is a consistency with the original labeling.

3.3.3 NNGA$^+$ as a data-shrinkage method

As shown in Figure 3.5, our version of the nearest neighbors gradient ascent NNGA$^+$ results in shrinking the data points toward their local modes. DS1 after

Dataset		NNGA$^+$ with ε-proximity	k-means	DBScan
Aggregation	NMI	0.97 ± 0.02	0.83 ± 0.02	$\mathbf{0.98 \pm 0.00}$
	Rand	0.98 ± 0.02	0.91 ± 0.01	$\mathbf{0.99 \pm 0.00}$
Banana	NMI	$\mathbf{1.00 \pm 0.00}$	0.31 ± 0.00	$\mathbf{1.00 \pm 0.00}$
	Rand	$\mathbf{1.00 \pm 0.00}$	0.70 ± 0.00	$\mathbf{1.00 \pm 0.00}$
Disk6000	NMI	0.32 ± 0.00	0.00 ± 0.00	$\mathbf{1.00 \pm 0.00}$
	Rand	0.34 ± 0.00	0.50 ± 0.00	$\mathbf{1.00 \pm 0.00}$
DS1	NMI	0.03 ± 0.01	0.75 ± 0.01	$\mathbf{0.94 \pm 0.00}$
	Rand	0.80 ± 0.01	0.86 ± 0.00	$\mathbf{0.98 \pm 0.00}$
EngyTime	NMI	0.85 ± 0.08	$\mathbf{0.98 \pm 0.00}$	0.75 ± 0.00
	Rand	0.93 ± 0.05	$\mathbf{1.00 \pm 0.00}$	0.90 ± 0.00
Hepta	NMI	0.97 ± 0.04	$\mathbf{0.98 \pm 0.03}$	0.83 ± 0.00
	Rand	0.98 ± 0.04	$\mathbf{0.99 \pm 0.02}$	0.94 ± 0.00
R15	NMI	0.91 ± 0.02	0.96 ± 0.02	$\mathbf{0.99 \pm 0.00}$
	Rand	0.98 ± 0.00	0.99 ± 0.01	$\mathbf{1.00 \pm 0.00}$
S3	NMI	0.74 ± 0.00	$\mathbf{0.78 \pm 0.01}$	0.42 ± 0.00
	Rand	$\mathbf{0.96 \pm 0.00}$	$\mathbf{0.96 \pm 0.00}$	0.52 ± 0.00
Sizes5	NMI	$\mathbf{0.89 \pm 0.01}$	0.81 ± 0.12	0.80 ± 0.00
	Rand	$\mathbf{0.97 \pm 0.00}$	0.88 ± 0.13	$\mathbf{0.97 \pm 0.00}$
Unbalance	NMI	0.98 ± 0.01	0.94 ± 0.05	$\mathbf{0.99 \pm 0.00}$
	Rand	$\mathbf{1.00 \pm 0.00}$	0.97 ± 0.03	$\mathbf{1.00 \pm 0.00}$
Hyperplane	NMI	$\mathbf{0.04 \pm 0.00}$	0.01 ± 0.00	one cluster
	Rand	0.31 ± 0.00	$\mathbf{0.62 \pm 0.00}$	only
CovType10	NMI	$\mathbf{0.09 \pm 0.01}$	0.07 ± 0.006	dataset is
	Rand	0.56 ± 0.04	$\mathbf{0.59 \pm 0.00}$	too massive

Table 3.3: NMI and Rand clustering quality indices for the cluster labeling on the experimental datasets for NNGA$^+$ with ε-proximity labeling, k-means and DBScan. The bold entries indicate the best results within each row, and ± entries are the standard deviations over 10 trials.

DS1	$M_1 = 12, \varepsilon_1 = 30$	$k_1 = 10$	$k_1 = 50$	$k_1 = 200$
NMI		0.0078 ± 0.0003	0.0315 ± 0.0016	0.0206 ± 0.0021
Rand		0.7268 ± 0.0146	0.8190 ± 0.0007	0.8129 ± 0.0029
Execution time (s)		81.1 ± 2.3	84.2 ± 6.5	76.5 ± 2.9
	$k_1 = 50, \varepsilon_1 = 30$	$M_1 = 8$	$M_1 = 12$	$M_2 = 20$
NMI		0.0240 ± 0.0016	0.0315 ± 0.0016	0.0183 ± 0.0020
Rand		0.8039 ± 0.0024	0.8190 ± 0.0007	0.7932 ± 0.0041
Execution time (s)		133.4 ± 4.2	84.2 ± 6.5	43.9 ± 2.0
	$k_1 = 50, M_1 = 12$	$\varepsilon_1 = 5$	$\varepsilon_1 = 30$	$\varepsilon_2 = 100$
NMI		0.1085 ± 0.0017	0.0315 ± 0.0016	0.0034 ± 0.0002
Rand		0.8283 ± 0.0001	0.8190 ± 0.0007	0.7078 ± 0.0006
Execution time (s)		78.3 ± 3.6	84.2 ± 6.5	76.5 ± 2.7

Table 3.4: NMI and Rand clustering quality indices and execution times for NNGA$^+$ with ε-proximity labeling for the DS1 dataset. k_1 is the number of nearest neighbors in the gradient ascent, M_1 is the number of buckets in the LSH, ε_1 is the distance threshold for the ε-proximity labeling. The bold entries indicate the best accuracy results, and $\pm$ entries are the standard deviations over 10 trials.

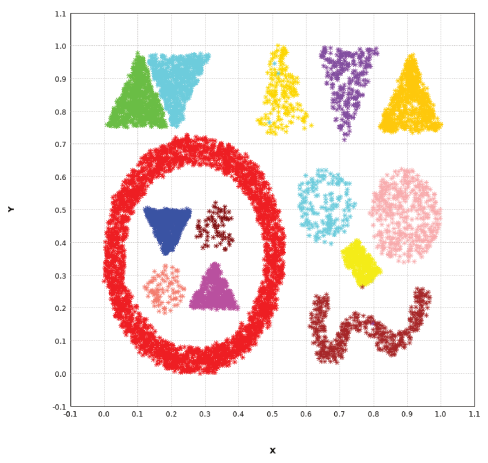
NNGA$^+$ (with $k_1 = 50$ nearest neighbors) presents a more compact version than its original version, maintaining the underlying data structures whilst increasing empty space between clusters; likewise for the Hepta dataset.

Being a data shrinkage method, this implies that we can apply NNGA$^+$ before we apply other clustering algorithms. Figure 3.6a shows an application of k-means on Sizes5 dataset with $k = 4$ clusters without NNGA$^+$. Even if the number of clusters in the k-means on the left is the correct number, these four clusters do not match so closely the original clusters (NMI=0.81, Rand=0.88). Figure 3.6b shows the k-means cluster labeling results applied on NNGA$^+$ output with $k_1 = 20$. We observe that these clusters are more similar to the original clusters (NMI=0.89, Rand=0.95) than with k-means only.

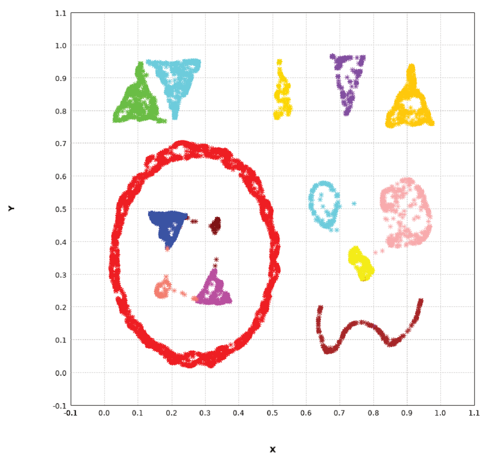
DBScan cluster labeling collates points more efficiently after NNGA$^+$ is applied than without NNGA$^+$, as shown in Figure 3.7. NNGA$^+$ is an efficient way to attach noisy points to their closest cluster. Furthermore, NNGA$^+$ facilitates more robust choices for the DBScan parameters, since it increases the local density which improves the detection of smaller clusters. The NMI and Rand increase after the NNGA$^+$ data shrinkage.

3.3.4 Visual evaluation with image segmentation

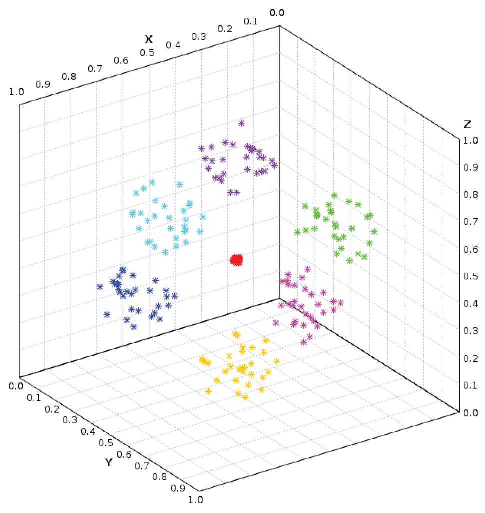
A resurgence in interest in the variant of the Mean Shift algorithm is due to its application to image segmentation [51] where an image is transformed into a color space in which clusters correspond to segmented regions in the original image. The 3-dimensional $L^*u^*v^*$ color space [45, Eqs. 3.5-8a–f] is a common choice. Since an image is a 2-dimensional array of pixels, let (x, y) be the row and column index of a pixel. The spatial and color (range) information of a pixel can be concatenated into a 5-dimensional vector (x, y, L^*, u^*, v^*) in the joint spatial-range



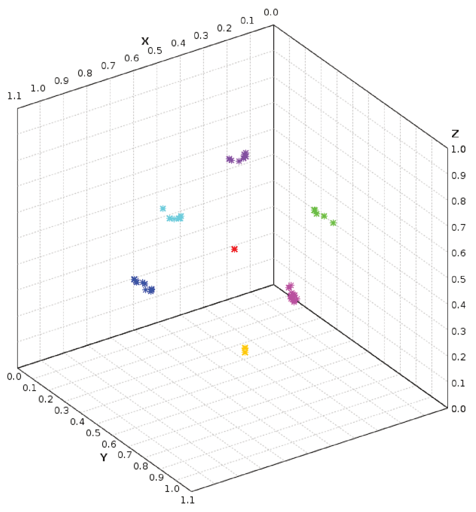
(a) DS1



(b) DS1 after NNGA$^+$ with number of nearest neighbors $k_1 = 50$



(c) Hepta



(d) Hepta after NNGA$^+$ with number of nearest neighbors $k_1 = 20$

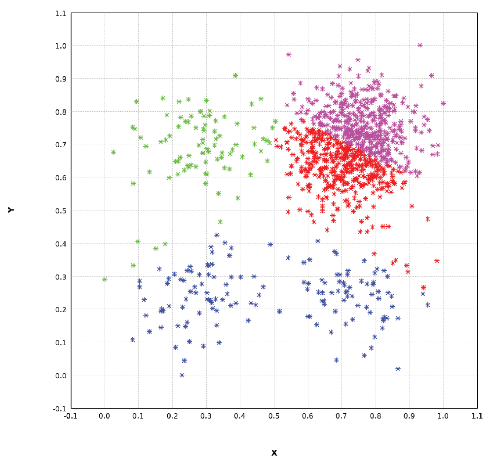
Figure 3.5: Data shrinkage after nearest neighbor gradient ascent NNGA$^+$ on the DS1 and Hepta datasets.

domain. An image segmentation algorithm based on the kernel Mean Shift was introduced in [48] which we adapt for use with NNGA$^+$.

Our test image is image #36 from the colour training set from the Berkeley Segmentation Dataset and Benchmark[1]. Figure 3.8 shows the original RGB 481×321 pixels JPEG image. The tuning parameters for the NNGA$^+$ are $k_1 = 60$, $j_{\max} = 15$, We compute the NNGA$^+$-M_1 with $M_1 = 200, 400, 1000$ buckets.

For the NNGA$^+$-200 and NNGA$^+$-400 in Figures 3.9b and 3.9d where we approximate nearest neighbours with respectively $M_1 = 200$ and $M_1 = 400$ buckets, some finer details are visible, such as the podia. For NNGA-200 and NNGA-400 in Figures 3.9a and 3.9c, the green background color bleeds into the starfish arms. For NNGA-1000 in Figure 3.9e, the starfish is not visible anymore.

[1] http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds


(a) Sizes5 after k-means ($k = 4$)

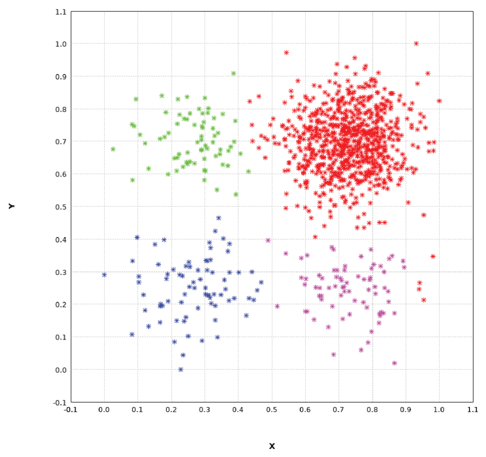
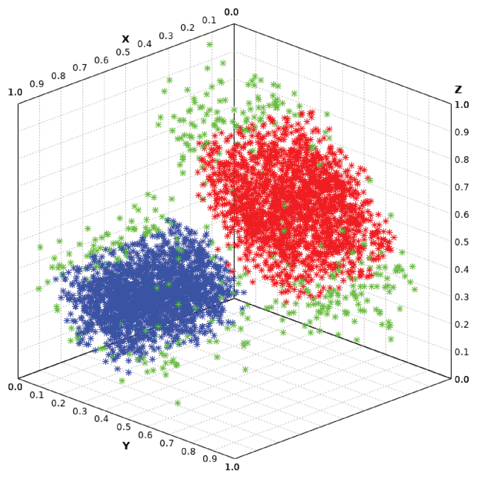

(b) Sizes5 after k-means ($k = 4$) with NNGA$^+$ (number of nearest neighbors $k_1 = 20$)

Figure 3.6: k-means cluster labeling on the Sizes5 dataset without and with NNGA$^+$ data shrinkage.


(a) EngyTime after DBScan with parameters #minPts=22, #ε=0.07

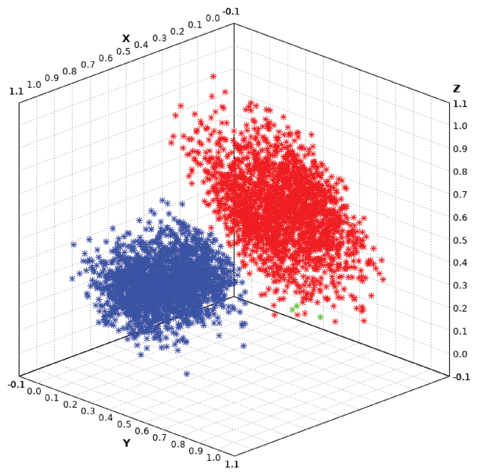

(b) EngyTime after DBScan with parameters #minPts=22, #ε=0.07 after NNGA$^+$ (number of nearest neighbors $k_1 = 200$)

Figure 3.7: DBSCan cluster labeling on the Sizes5 dataset without and with NNGA$^+$ data shrinkage.

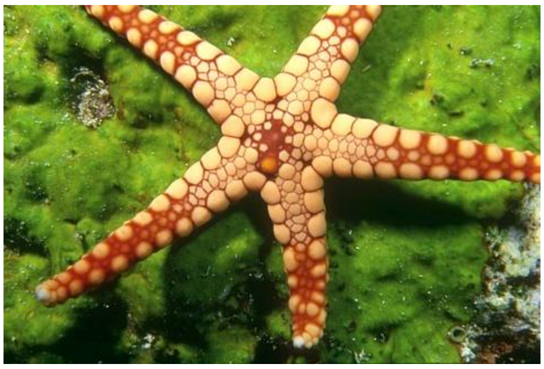


Figure 3.8: Original picture #36.

However, by taking more neighbor layers (NNGA$^+$-1000 $p = 2$) in Figure 3.9f the starfish shape is being delimited from the background.

The Berkeley Segmentation Dataset and Benchmark provides human expert segmentations of their images for comparisons. Figure 3.10a,b depict two edge detections made by Users #1109 and #1119. User #1109 focuses on segmenting the shape of the starfish, whilst ignoring the detail of the podia, whereas User #1119 concentrates on segmenting the individual podia in the foreground. We focus on the NNGA$^+$-200 (Figure 3.9b), which segmentation is closer to User #1119. Whilst this automatic edge detection (Figure 3.10d) in its current state remains too fragmented to be useful for a visual analysis, we anticipate that further application of statistical and image analyses will be able to improve it.

We observe from Figures 3.9 that $M_1 = 200$ buckets is a suitable empirical choice and so we apply it to further images from the Berkeley image database as in Figure 3.11. We apply also the NNGA$^+$ on a picture of 5 millions pixels (taken by ourselves) compared to 150 thousand pixels of Berkeley's pictures. We increase the number of buckets accordingly with the size of the picture to reach approximately 1000 data points per bucket. The image shows a good segmentation between the leafs and the rest of the picture. We distinguish the trunk from the background foliage too without difficulty showing the effectiveness of the algorithm even with big image.

3.3.5 Evaluation of scalability

3.3.5.1 Density gradient ascent

As the data are distributed over all buckets or Spark partitions, this allows for efficient computations in the local environment of a data point. Figure 3.12a shows that the execution time gradually decreases as the number of slaves increases for a dataset of fixed size. In Figure 3.12b, for a fixed number of slaves, if we maintain the constant number of elements per bucket, the execution time grows linearly with the size of the dataset. This indicates that the number of nearest neighbors k_1 needs to be constrained. The number of layers p indicates the adjacent buckets in the LSH which can be searched to find nearest neighbors: p =0 means that no adjacent buckets, $p = 1$ means the immediately adjacent buckets to the left and right, $p = 2$ means that second order adjacent buckets further away etc.

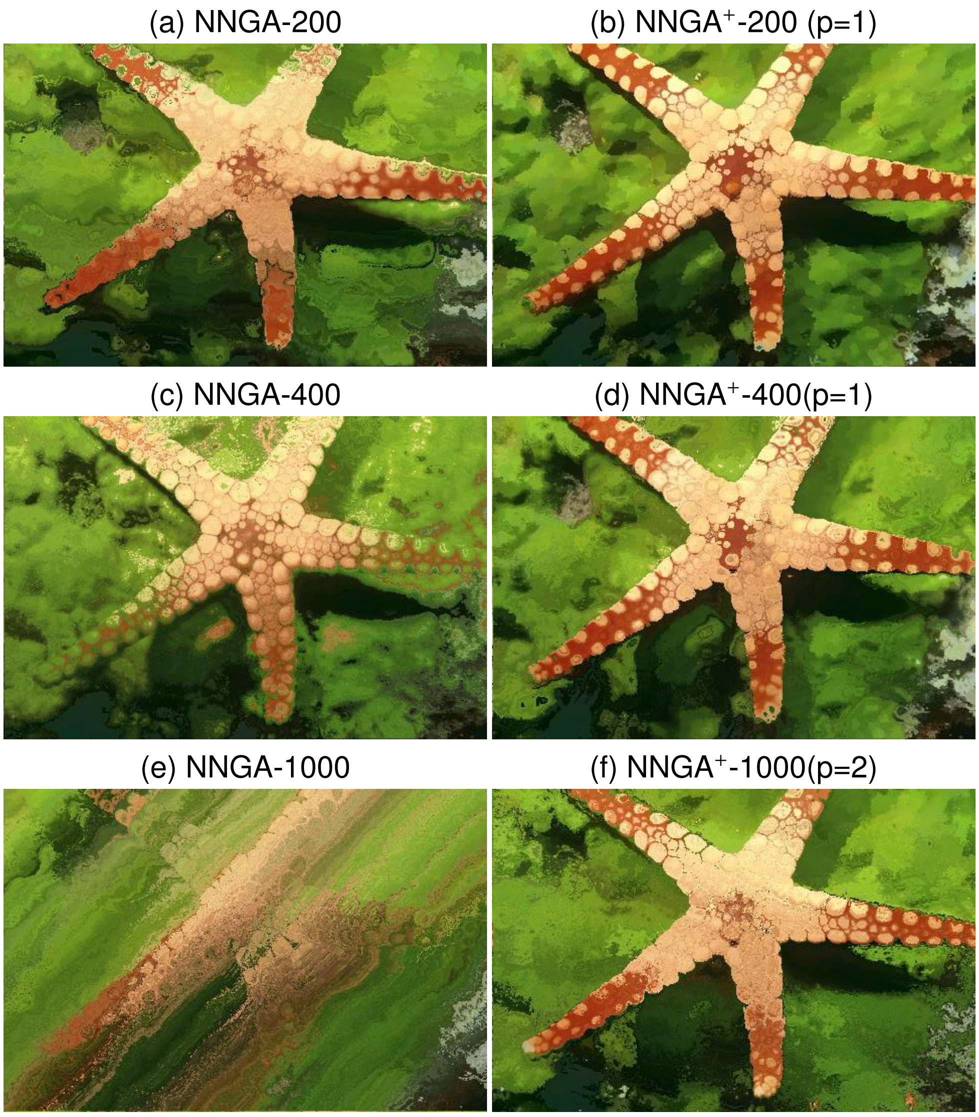


Figure 3.9: Colour image segmentation using in the left NNGA and in the right NNGA^{+} with $M_1 = 200, 400$ and 1000 buckets.

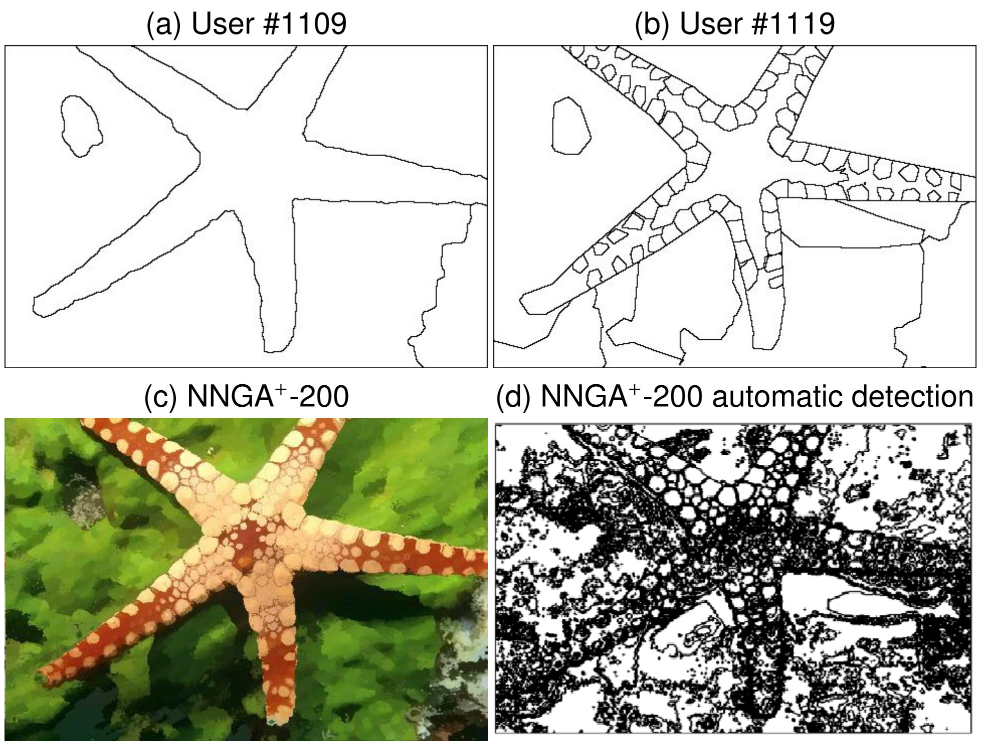


Figure 3.10: Edge detection of segmented images. (a,b) Two human experts: users #1109 and #1119. (c) NNGA$^+$-200. (d) NNGA$^+$-200 with automatic detection.

Since we have an $O((\frac{n}{M_1})^2 \log(\frac{n}{M_1}))$ complexity per bucket for NNGA$^+$, a suitable value for the number of buckets M_1 is to keep the $\frac{n}{M_1}$ ratio approximately equal to a constant C. Then the time complexity of NNGA$^+$ reduces to $O(nC\log(C))$. The scalability is demonstrated by the decrease in execution time with the number of slaves and a linear increase of execution time with the dataset size, reaching 140 million data points (the ScalabilityDS dataset) which is infeasible for the original quadratic algorithm.

3.3.5.2 LSH buckets and neighbor layers

For a fixed number of neighbor layers, we observe in Figure 3.13a that the execution time rapidly decreases and then slows down to reach a plateau, as the number of buckets increases. The observed plateau is due to the quadratic complexity of the NNGA$^+$: more buckets leads to fewer data points within each bucket and so the execution times can quickly reach the minimal plateau after a sufficiently large number of buckets. We also studied the influence of the number of neighbors layers p on the execution time. Whilst NNGA$^+$ has quadratic time complexity in each bucket, if we select an appropriate number of nearest neighbors k_1, then we are able to control the execution time of NNGA$^+$ to be linear with respect to the number of neighbors layers p, as illustrated in Figure 3.13b.

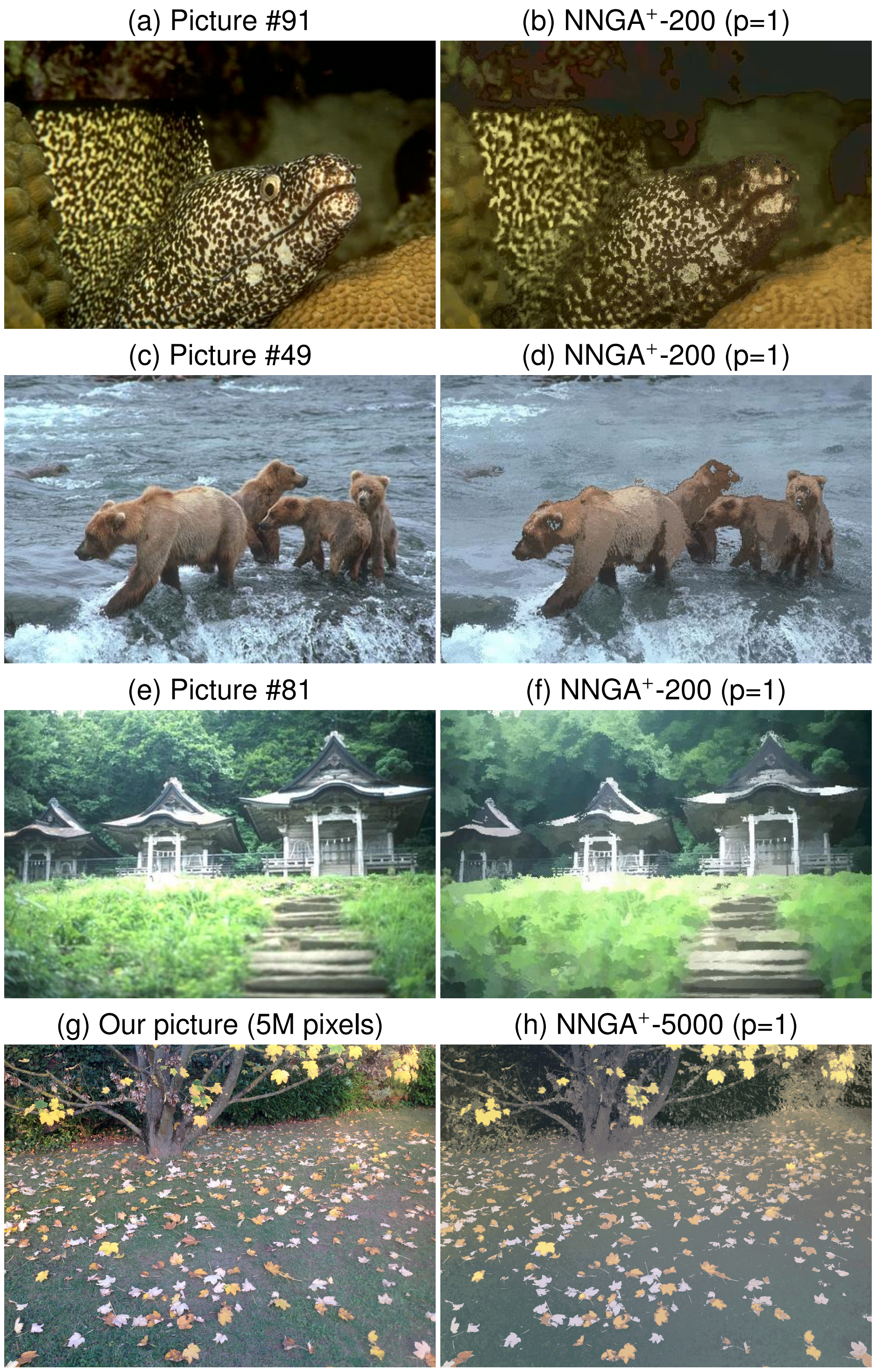
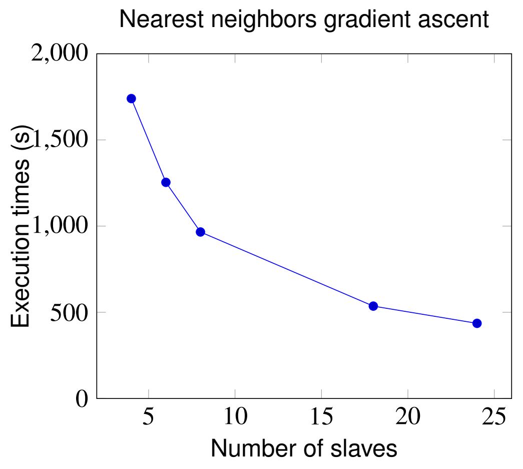
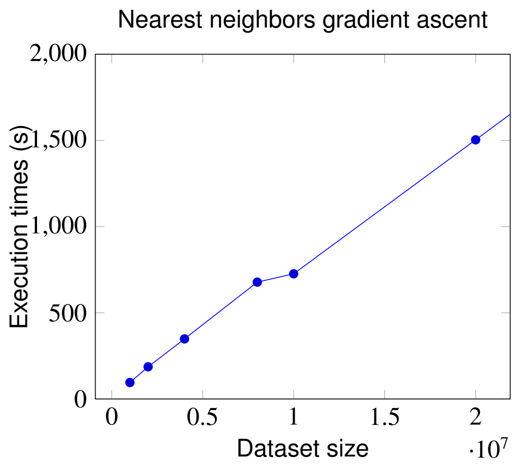


Figure 3.11: Further examples of segmented images with NNGA$^+$-200.

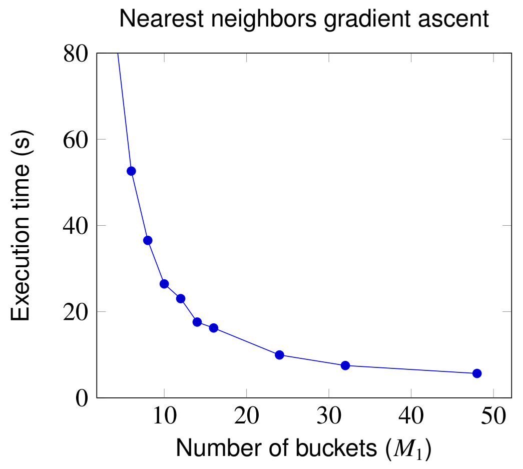


(a) Dataset size $n = 500000$, #buckets $M_1 = 500$, #slaves varies, #layers $p = 1$.

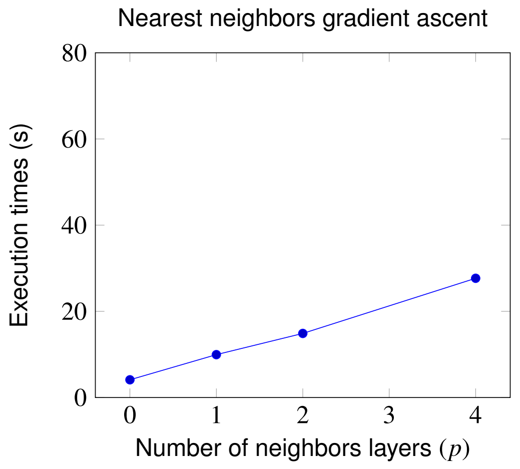


(b) Dataset size n varies, #buckets $M_1 = n/1000$, #slaves = 18, #layers $p = 1$.

Figure 3.12: (a) Execution times for NNGA$^+$ with respect to the number of slaves. (b) Execution times for NNGA$^+$ with respect to the dataset size n.



(a) Dataset size $n = 50000$, #buckets varies, #layers $p = 1$.

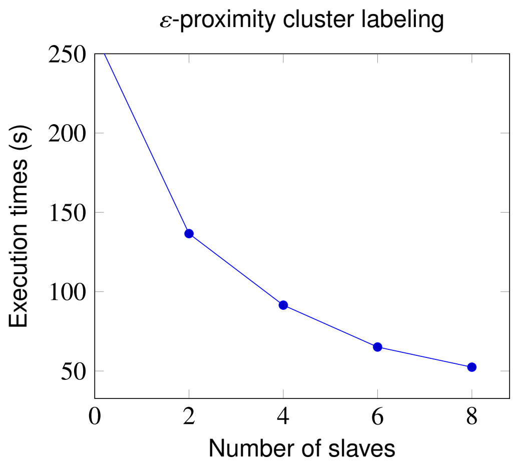


(b) Dataset size $n = 50000$, #buckets $M_1 = 24$, #layers p varies.

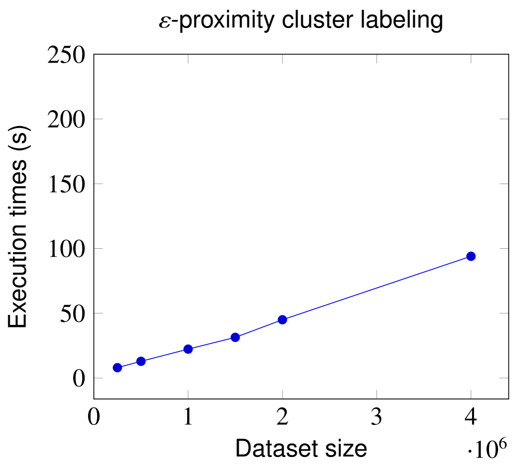
Figure 3.13: (a) Execution times for NNGA$^+$ with respect to the number of buckets (M_1). (b) Execution times for NNGA$^+$ with respect to the number of adjacent layers (p).

3.3.5.3 ε-proximity cluster labeling

Concerning ε-proximity cluster labeling, similar remarks as for the gradient ascent apply here. As Figure 3.14 follows same decrease in the execution time as for the gradient ascent as a function of the number of slaves and data points, we can be confident in the scalability of our approach.



(a) Dataset size $n = 1000000$, #buckets $M_1 = 1000$, #slaves varies, #layers $p = 1$.



(b) Dataset size n varies, #buckets $M_1 = n/1000$, #slaves = 8, #layers $p = 1$.

Figure 3.14: (a) Execution times for ε-proximity cluster labeling with respect to the number of slaves. (b) Execution times for ε-proximity cluster labeling with respect to the dataset size n.

3.4 Conclusion

We have introduced a framework for the mathematical analysis of nearest neighbour estimators of the density function and its derivatives, allowing us to exhibit an automatic, single pass, normal scale selector for the optimal number of nearest neighbours. We apply these results for the density gradient to the Mean Shift for unsupervised learning. Our proposed automatic nearest neighbour Mean Shift clustering NNGA gave good empirical performance for discovering the number, location and shape of non-ellipsoidal clusters for multivariate data analysis and image segmentation. Moreover we have added practical improvements to the standard nearest neighbors gradient ascent used in Mean Shift algorithm. Two of them are based on new usages of Locality Sensitivity Hashing for approximate nearest neighbors during the nearest neighbors gradient ascent (NNGA$^+$), and also during cluster labeling (ε-proximity). The last one is an efficient and scalable implementation of our ideas on a distributed computing ecosystem based on Spark/Scala. We demonstrated that these improvements greatly decrease the execution time whilst maintaining a suitable quality of clustering. We have also shown that using our NNGA$^+$ algorithm, as a pre-processing step in other clustering methods, can improve quality metric evaluations. These improvements open the opportunity to apply our Mean Shift model for Big Data clustering.

Along this chapter, the continuous space has been studied through some algorithms, but it is not the only space on which users would discover insights about their data. We will present in next chapter a new version of modal clustering dealing with binary data and using Hamming distance.

Chapter 4

Mean Shift Like on Binary Data

4.1 Introduction

We describe in this chapter the theory and practice behind a new modal clustering method for binary data. Our approach (BinNNMS) is based on the nearest neighbor median shift. The median shift is an extension of the well-known mean shift, which was designed for continuous data, to handle binary data. Clustering is an important step in the exploratory phase of data analysis, and it becomes more difficult when applied to binary or mixed data. Binary data occupy a special place in many application fields: behavioral and social research, survey analysis, document clustering, and inference on binary images.

A popular clustering algorithm for binary data is the k-modes [30], and it is similar to the k-means clustering [4] wherein the modes are used instead of the means for the prototypes of the clusters. Other clustering algorithms have been developed using a matching dissimilarity measure for categorical points instead of Euclidean distance [66], and a frequency-based method to update modes in the clustering process [40].

We focus here on the mean shift clustering presented in the previous chapter. As explained in in previous chapter, due to its reliance on mean computations, mean shift is not suited to be applied to binary data. Our contribution in this chapter is the presentation of a modified mean shift clustering which is adapted to binary data. It is untitled Nearest Neighbor Median Shift clustering for binary data (BinNNMS). We demonstrate that BinNNMS can discover accurately the location of clusters in binary data with theoretical and experimental analyses. The main novelty is that the cluster prototypes are updated via iterations on the majority vote of their nearest neighbors. We demonstrate that this majority vote corresponds to the median of the nearest neighbors with respect to the Hamming distance [1, 36].

4.2 Nearest neighbor median shift clustering for binary data

A categorical feature, which has a finite (usually small) number of possible values, can be represented by a binary vector, i.e. a vector which is composed solely of zeroes and ones. These categorical features can either be ordinal (which have an implicit order) or can be nominal (no order exists). Table 4.1 presents the two main types of the coding for a categorical feature into a binary vector, additive and disjunctive, for an example of 3-class categorical feature.

Class	Additive coding	Disjunctive coding
1	1 0 0	1 0 0
2	1 1 0	0 1 0
3	1 1 1	0 0 1

Table 4.1: Additive and disjunctive coding for a 3-class categorical feature.

The usual Euclidean distance is not adapted to measuring the dissimilarities between binary vectors. A popular alternative is the Hamming distance $\mathcal{H}$. The Hamming distance between two binary vectors $\boldsymbol{x}_1 = (x_{11}, \dots, x_{1d})$ and $\boldsymbol{x}_2 = (x_{21}, \dots, x_{2d})$, $\boldsymbol{x}_j \in \{0, 1\}^d$, $j \in 1, 2$, is defined as:

$$\begin{aligned}\mathcal{H}(\boldsymbol{x}_1, \boldsymbol{x}_2) &= \sum_{j=1}^{d} |x_{1j} - x_{2j}| \\ &= d - (\boldsymbol{x}_1 - \boldsymbol{x}_2)^\top (\boldsymbol{x}_1 - \boldsymbol{x}_2). \end{aligned} \tag{4.1}$$

Equation (4.1) measures the number of mismatches between the two vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$: as the inner product $(\boldsymbol{x}_1 - \boldsymbol{x}_2)^\top(\boldsymbol{x}_1 - \boldsymbol{x}_2)$ counts the number of elements which agree in both $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, then $d - (\boldsymbol{x}_1 - \boldsymbol{x}_2)^\top(\boldsymbol{x}_1 - \boldsymbol{x}_2)$ counts the number of disagreements.

The Hamming distance is the basis from which we define the median center of a set of observations $\mathcal{X} = \{\boldsymbol{X}_1, \dots, \boldsymbol{X}_n\}, \boldsymbol{X}_i \in \{0, 1\}^d, i = 1, \dots, n.$ Importantly the median center of the set of binary vectors, as a measure of the centrality of the values, remains a binary vector, unlike the mean vector which can take on intermediate values. The median center of $\mathcal{X}$ is a point $\boldsymbol{w} = (w_1, \dots, w_d)$ which minimizes the inertia of $\mathcal{X}$, i.e.

$$\boldsymbol{w} = \operatorname*{argmin}_{\boldsymbol{x} \in \{0,1\}^d} \mathcal{I}(\boldsymbol{x}) \tag{4.2}$$

where

$$\mathcal{I}(\boldsymbol{x}) = \sum_{i=1}^{n} \pi_i \mathcal{H}(\boldsymbol{X}_i, \boldsymbol{x}) = \sum_{i=1}^{n} \sum_{j=1}^{d} \pi_i \mathcal{I}(x_j)$$

and π_i are the weights and

$$\mathcal{I}(x_j) = |X_{ij} - x_j|.$$

Each component w_j of $\boldsymbol{w}$ minimizes $\mathcal{I}(x_j)$.

In the case where all the weights are set to 1, $\pi_i = 1, i = 1, \dots, n$, the w_j can be easily computed since it is the most common value in the observations of the j-th feature. This is denoted as $\mathrm{maj}(\mathcal{X})$, the component-wise majority vote winner among the data points. Hence the median center is the majority vote, $\boldsymbol{w} = \mathrm{maj}(\mathcal{X})$.

If we minimize the cost function in Equation (4.2) using the dynamic clusters [9] then this leads to the k-modes clustering. Like the k-means algorithm, the k-modes operates in two steps: (a) an assignment step which assigns each candidate point $\boldsymbol{x}$ to the nearest cluster with respect to the Hamming distance, and (b) an optimization step which computes the median center as the majority vote. These two steps are executed iteratively until the value of $\mathcal{I}(\boldsymbol{x})$ converges.

Now we show how the median center can be utilized to define a new modal clustering for binary data based on the mean shift paradigm. In Section 3.2.1, the beta family kernels were used in the mean shift for continuous data. The most commonly used smoothing kernel, introduced by [8], for binary data is the Aitchison and Aitken kernel:

$$K_\lambda(\boldsymbol{x}) = \lambda^{d-\boldsymbol{x}^\top\boldsymbol{x}}(1-\lambda)^{\boldsymbol{x}^\top\boldsymbol{x}},\ \boldsymbol{x} \in \{0,1\}^d.$$

Observe that the exponent for λ is the Hamming distance of $\boldsymbol{x}$. The tuning parameter $\frac{1}{2} \le \lambda \le 1$ controls the spread of the probability mass around the origin $\mathbf{0}$. For $\lambda = 1/2$, then $K_{1/2}(\boldsymbol{x}) = (1/2)^d$, which assigns a constant probability to all points $\boldsymbol{x}$, regardless of its distance from $\mathbf{0}$. For $\lambda = 1$, $K_1(\boldsymbol{x}) = \mathbf{1}\{\boldsymbol{x} = \mathbf{0}\}$, which assigns all the probability mass to $\mathbf{0}$. For intermediate values of λ, we have intermediate assignment between points and uniform probability mass.

Using K_λ, the corresponding kernel density estimate is

$$\tilde{f}(\boldsymbol{x};\lambda) = n^{-1}\sum_{i=1}^{n} \lambda^{[d-(\boldsymbol{x}-\boldsymbol{X}_i)^\top(\boldsymbol{x}-\boldsymbol{X}_i)]}(1-\lambda)^{[(\boldsymbol{x}-\boldsymbol{X}_i)^\top(\boldsymbol{x}-\boldsymbol{X}_i)]}. \tag{4.3}$$

Since the gradient of the kernel K_λ is $\mathsf{D}K_\lambda(\boldsymbol{x}) = 2\boldsymbol{x}\log((1-\lambda)/\lambda)K_\lambda(\boldsymbol{x})$, the density gradient estimate is

$$\mathsf{D}\tilde{f}(\boldsymbol{x};\lambda) = 2\log(\lambda/(1-\lambda))n^{-1}\Big[\sum_{i=1}^{n}\boldsymbol{X}_iK_\lambda(\boldsymbol{x}-\boldsymbol{X}_i) - \boldsymbol{x}\sum_{i=1}^{n}K_\lambda(\boldsymbol{x}-\boldsymbol{X}_i)\Big]. \tag{4.4}$$

To progress in our development of a nearest neighbor median shift for binary data, we focus on the point mass kernel $K_1(\boldsymbol{x}) = \mathbf{1}\{\boldsymbol{x} = \mathbf{0}\}$. In order to ensure that it is amenable for the median shift, we modify K_1 with two main changes:

1. K_1 is multiplied by the indicator function $\mathbf{1}\{\boldsymbol{x} \in B_d(\mathbf{0}, 1)\}$
2. the indicator function $\mathbf{1}\{\boldsymbol{x} = \mathbf{0}\}$, which places the point mass at the center $\mathbf{0}$, is replaced by an indicator that places it on $\mathrm{maj}(B_d(\mathbf{0}, 1))$, where $\mathrm{maj}(B_d(\mathbf{0}, 1))$ is the majority vote winner/median center of the data points $\boldsymbol{X}_1, \dots, \boldsymbol{X}_n$ inside of $B_d(\mathbf{0}, 1)$.

This second modification results in an asymmetric kernel as the point mass is no longer always placed in the centre of the unit ball. This modified, asymmetric kernel L is

$$L(\boldsymbol{x}) = \mathbf{1}\{\boldsymbol{x} = \mathrm{maj}(B_d(\mathbf{0}, 1))\}\mathbf{1}\{\boldsymbol{x} \in B_d(\mathbf{0}, 1)\}.$$

Since L is not directly differentiable, we define its derivative indirectly via $\mathsf{D}K_1$ and the convention that $\log(\lambda/(1-\lambda)) = 1$ for $\lambda = 1$. As $\mathsf{D}K_\lambda(\boldsymbol{x})\big|_{\lambda=1} = 2\boldsymbol{x}K_1(\boldsymbol{x})$ then analogously we define $\mathsf{D}L(\boldsymbol{x}) = 2\boldsymbol{x}L(\boldsymbol{x})$. To obtain the corresponding estimators, we substitute $L, \mathsf{D}L$ for $K, \mathsf{D}K$ in $\tilde{f}, \mathsf{D}\tilde{f}$ in Equations (4.3)–(4.4) to obtain $\hat{f}, \mathsf{D}\hat{f}$:

$$\hat{f}(\boldsymbol{x}; k) = n^{-1}\delta_{(k)}(\boldsymbol{x})^{-d}\sum_{i=1}^{n} L((\boldsymbol{x} - \boldsymbol{X}_i)/\delta_{(k)}(\boldsymbol{x}))$$
$$\mathsf{D}\hat{f}(\boldsymbol{x}; k) = 2\delta_{(k)}(\boldsymbol{x})^{-d-1}n^{-1}\Big[\sum_{i=1}^{n} \boldsymbol{X}_i L((\boldsymbol{x} - \boldsymbol{X}_i)/\delta_{(k)}(\boldsymbol{x})) - \boldsymbol{x}\sum_{i=1}^{n} L((\boldsymbol{x} - \boldsymbol{X}_i)/\delta_{(k)}(\boldsymbol{x}))\Big]. \tag{4.5}$$

To obtain a nearest neighbor mean shift recurrence relation for binary data, we substitute $\hat{f}, \mathsf{D}\hat{f}$ for $f, \mathsf{D}f$ is Equation (3.4). For these estimators, the appropriate choice of $\mathbf{A} = \frac{1}{2}\delta_{(k)}(\boldsymbol{x})\mathbf{I}_d$. Then we have

$$\begin{aligned}\boldsymbol{x}_{j+1} &= \boldsymbol{x}_j + \frac{\delta_{(k)}(\boldsymbol{x})}{2}\frac{\mathsf{D}\hat{f}(\boldsymbol{x}_j; k)}{\hat{f}(\boldsymbol{x}_j; k)}\\ &= \frac{\sum_{i=1}^{n} \boldsymbol{X}_i L((\boldsymbol{x}_j - \boldsymbol{X}_i)/\delta_{(k)}(\boldsymbol{x}_j))}{\sum_{i=1}^{n} L((\boldsymbol{x}_j - \boldsymbol{X}_i)/\delta_{(k)}(\boldsymbol{x}_j))}.\end{aligned}$$

We can simplify this ratio if we observe that the scaled kernel is

$$L((\boldsymbol{x} - \boldsymbol{X}_i)/\delta_{(k)}(\boldsymbol{x})) = \mathbf{1}\{\boldsymbol{X}_i \mathrm{maj}(B_d(\boldsymbol{x}, \delta_{(k)}(\boldsymbol{x})))\} \cdot \mathbf{1}\{\boldsymbol{X}_i \in B_d(\boldsymbol{x}, \delta_{(k)}(\boldsymbol{x}))\};$$

and that $B_d(\boldsymbol{x}, \delta_{(k)}(\boldsymbol{x}))$ comprises the k nearest neighbors of $\boldsymbol{x}$, then $\mathbf{1}\{\boldsymbol{X}_i \in B_d(\boldsymbol{x}, \delta_{(k)}(\boldsymbol{x}))\} = \mathbf{1}\{\boldsymbol{X}_i \in k\text{-nn}(\boldsymbol{x})\}$. If m is the number of nearest neighbors of $\boldsymbol{x}_j$ which coincide with the majority vote, then

$$\begin{aligned}\boldsymbol{x}_{j+1} &= \frac{\sum_{\boldsymbol{X}_i \in k\text{-nn}(\boldsymbol{x}_j)} \boldsymbol{X}_i \mathbf{1}\{\boldsymbol{X}_i = \mathrm{maj}(k\text{-nn}(\boldsymbol{x}_j))\}}{\sum_{\boldsymbol{X}_i \in k\text{-nn}(\boldsymbol{x}_j)} \mathbf{1}\{\boldsymbol{X}_i = \mathrm{maj}(k\text{-nn}(\boldsymbol{x}_j))\}}\\ &= \frac{m \cdot \mathrm{maj}(k\text{-nn}(\boldsymbol{x}_j))}{m}\\ &= \mathrm{maj}(k\text{-nn}(\boldsymbol{x}_j)).\end{aligned} \tag{4.6}$$

Therefore in the median shift recurrence relation in Equation (4.6), the next iterate $\boldsymbol{x}_{j+1}$ is the median center of the k nearest neighbors of the current iterate $\boldsymbol{x}_j$. Thus, once the binary gradient ascent has terminated, the converged point can be decoded using Table 4.1, allowing for its unambiguous symbolic interpretation. The gradient ascent paths towards the local modes produced by Equation (4.6) form the basis of Algorithm 16, our nearest neighbor median shift clustering for binary data method (BinNNMS).

The inputs to BinNNMS are the data sample $\boldsymbol{X}_1, \dots, \boldsymbol{X}_n$ and the candidate points $\boldsymbol{x}_1, \dots, \boldsymbol{x}_m$ which we wish to cluster (these can be the same as $\boldsymbol{X}_1, \dots, \boldsymbol{X}_n$, but this is not required); and the tuning parameters: the number of nearest neighbors k_1 used in BGA task, the maximum number of iterations $iter_{\max}$, and the tolerance under which two cluster centres are considered forming a single cluster ε. The outputs are the cluster labels of the candidate points $\{\mathrm{c}(\boldsymbol{x}_1), \dots, \mathrm{c}(\boldsymbol{x}_m)\}$.

The aim of the ε-proximity cluster labeling step is to gather all points which are under a threshold ε. In order to apply this method we have to build the Hamming similarity matrix which has a $O(n^2)$ time complexity. We initialize the process by taking the first point and cluster with it all points whose distance is less than ε. Thus we apply this iterative exploration process by adding the nearest neighbors. Once the first cluster is generated, we take another point from the reduced similarity matrix and repeat the process, until all points are assigned a cluster label. A notable problem still remains with the choice of main tuning parameter ε: we set it to be the average of distance from each point to their k_2 nearest neighbors.

Algorithm 16 BinNNMS – Nearest neighbor median shift clustering for binary data

```
   Input: ${X_1, \dots, X_n}, {\boldsymbol{x}_1, \dots, \boldsymbol{x}_m}, k_1, k_2, iter_{\max}$
   Output: ${c(\boldsymbol{x}_1), \dots, c(\boldsymbol{x}_m)}$
   /* BGA task: compute binary gradient ascent paths */
1: for $\ell := 1$ to $m$ do
2:     $j := 0; \boldsymbol{x}_{\ell,0} := \boldsymbol{x}_\ell$;
3:     $\boldsymbol{x}_{\ell,1} := \mathrm{maj}(k\text{-nn}_{k_1}(\boldsymbol{x}_{\ell,0}))$;
4:     while $j < j_{\max}$ do
5:         $j := j + 1$;
6:         $\boldsymbol{x}_{\ell,j+1} := \mathrm{maj}(k\text{-nn}_{k_1}(\boldsymbol{x}_{\ell,j}))$;
7:     $\boldsymbol{x}^*_\ell := \boldsymbol{x}_{\ell,j}$;
   /* $\varepsilon$-proximity cluster labeling task: create clusters by merging near final iterates*/
8: for $\ell_1, \ell_2 := 1$ to $m$ do
9:     if $\mathcal{H}(\boldsymbol{x}^*_{\ell_1}, \boldsymbol{x}^*_{\ell_2}) \le \varepsilon(k_2)$ then $c(\boldsymbol{x}^*_{\ell_1}) := c(\boldsymbol{x}^*_{\ell_2})$;
```

4.3 Experiments

In this section, we present an experimental comparison of the BinNNMS to the k-modes clustering (as outlined in Section 4.2). Table 4.2 lists the details of the dataset obtained from the UCI Machine learning repository [125]. The Zoo data set contains $n = 101$ animals described with 16 categorical features: 15 of the variables are binary and one is numeric with 6 possible values. Each animal is labelled 1 to 7 according to its class. Using disjunctive coding for the categorical variable with 6 possible values, the data set consists of a 101×21 binary data matrix. The Digits data concerns a dataset consisting of the handwritten numerals ("0"–"9") extracted from a collection of Dutch utility maps. There are 200 samples of each digit so there is a total of $n = 2000$ samples. As each sample is a 15×16 binary pixel image, the dataset consisted of a 2000×240 binary data matrix. The Spect dataset describes the cardiac diagnoses from Single Proton Emission Computed Tomography (SPECT) images. Each patient is classified into two categories: normal and abnormal; there are $n = 267$ samples which are described by 22 binary features.

The Car dataset contains examples with the structural information of the vehicle is removed. Each instance is classified into 4 classes. This database is highly unbalanced since the distribution of the classes is $(70.02\%, 22.22\%, 3.99\%, 3.76\%)$.

Dataset	size (n)	#features (d)	#classes (M)
Zoo	101	26	7
Digits	2000	240	10
Spect	267	22	2
Soybean	307	97	18
Car	1728	15	4

Table 4.2: Overview of experimental datasets.

The Soybean data is about 19 classes, but only the first 15 have been justified as it appears that the last four classes are not well-defined. There are 35 categorical attributes, with both nominal and ordinal features.

4.3.1 Comparison of the k-modes and the BinNNMS

To evaluate the clustering quality, we compare the known cluster labels in Table 4.2 to the estimated cluster labels from BinNNMS and k-modes. For comparability, the k-modes clustering is also based on the binary median center from Equation (4.2). Values of the Adjusted Rand Index (ARAND) [18] and the normalized mutual information (NMI) [50] close to one indicate highly matched cluster labels, and values close to zero for the NMI (less than zero for the ARAND) indicate mismatched cluster labels.

Table 4.3 reports the results in terms of the NMI and ARAND after 10 runs of the BinNNMS and k-modes. Unlike BinNNMS, the k-modes clustering requires an a priori number k of clusters, then we set k to be whichever value between the target number of classes from Table 4.2, or the number of clusters obtained from the BinNNMS clustering gives the highest clustering accuracy. The BinNNMS, apart from the Car dataset, outperforms the k-modes algorithm on Zoo, Digits, Spect, and Soybean datasets. Upon further investigation for the Car dataset, recall that the distribution of the cluster labels is highly unbalanced which leads the BinNNMS giving a single class (i.e. no clustering). These unbalanced clusters also translate into low values of the NMI and ARAND for the k-modes clustering.

4.3.2 Comparison of the tuning parameters for the BinNNMS

Figure 4.1 presents the evolution of the NMI and ARAND scores as a function of the tuning parameters k_1 (in binary gradient ascent BGA task) and k_2 (in the cluster labeling task) for the Digits, Zoo, Soybean, and Spect datasets. The blue dots ($k_1 = 0$) correspond to the application of the cluster labeling task without the gradient ascent. These cases tend to have poor cluster quality values compared to when k_1 is non-zero. Otherwise, that various values of k_1 and k_2 give the highest cluster label accuracy. This indicates that the optimal combination of these tuning parameters remains an open and challenging task.

NMI			
Dataset	k-modes	k	BinNNMS
Digits	0.360 ± 0.011	40	$\mathbf{0.880 \pm 0.000}$
Zoo	0.789 ± 0.023	8	$\mathbf{0.945 \pm 0.000}$
Soybean	0.556 ± 0.000	40	$\mathbf{0.743 \pm 0.000}$
Spect	0.135 ± 0.000	47	$\mathbf{0.145 \pm 0.000}$
Car	$\mathbf{0.039 \pm 0.019}$	**4**	Single class
ARAND			
Dataset	k-modes	k	BinNNMS
Digits	0.166 ± 0.021	40	$\mathbf{0.876 \pm 0.000}$
Zoo	0.675 ± 0.032	8	$\mathbf{0.904 \pm 0.000}$
Soybean	0.178 ± 0.000	40	$\mathbf{0.331 \pm 0.000}$
Spect	$\mathbf{-0.009 \pm 0.055}$	**2**	-0.019 ± 0.000
Car	$\mathbf{0.016 \pm 0.039}$	**4**	Single class

Table 4.3: Comparison of clustering quality indices (NMI and ARAND) for k-modes and BinNNMS. The bold value indicates the most accurate clustering for the dataset.

4.3.3 Comparison of the quantization errors

An important and widely used measure of resolution, the quantization error, is computed based on Hamming distances between the data points and the cluster prototypes:

$$Error = \frac{1}{n} \sum_{m=1}^{M} \sum_{\boldsymbol{x}_j \in C_m} \mathcal{H}(\boldsymbol{x}_j, \boldsymbol{w}_m) \tag{4.7}$$

where $\{C_1, \ldots, C_M\}$ is the set of M clusters, $\boldsymbol{x}$ is a point assigned to cluster C_m, and $\boldsymbol{w}_m$ is the prototype.median center of cluster C_m.

The right hand column in Figure 4.2 shows the evolution of the quantization errors for the BinNNMS with different values of k_1 with respect to the target cluster prototypes. As the quantization errors decrease this implies that the data points converge toward their cluster prototypes, and that the decreasing intra-cluster distance further facilitates the clustering process. Thus at the end of the training phase, the data points converge towards their local mode. In comparison with the ARAND scores in Table 4.3, the magnitude of the decrease in the quantization errors is inversely proportional to the cluster quality indices. That is, the largest decrease for the Digits dataset implies that BinNNMS clustering achieves here the highest ARAND score.

If we run the labeling phase during the BGA phase for a fixed k_1 then we compute the intermediate prototypes $\boldsymbol{w}_m$ of the clusters C_m during the binary gradient ascent BGA task. Since BinNNMS provides clusters as the basins of attraction to the local median created by the binary gradient ascent paths, the left column of Figure 4.2 shows the quantization error with respect to the intermediate median centers/prototypes. In this case we compute at each iteration 7 modes for Zoo dataset, 10 modes for the Digits, 18 modes for Soybean and 2 modes for Spect datasets using ground truth. These quantization errors decrease to an asymptote for all datasets as the iteration number increases.

Visual comparison

Figure 4.3 shows the cluster prototypes provided by k-modes and BinNNMS, displayed as 15×16 binary pixel images. For the k-modes image, the cluster prototype for the “4” digit has been incorrectly associated with the “9” cluster. On the other hand, the BinNNMS image correctly identifies all ten digits from “0” to “9”.

4.4 Conclusion

In this chapter, we have proposed a new and efficient modal clustering method for binary data. We introduced a mathematical analysis of the nearest neighbor estimators for binary data. This was then combined with the Aitchison and Aitken kernel in order to generalize the traditional mean shift clustering to the median shift clustering for binary data (BinNNMS). Experimental evaluation for a number of experimental datasets demonstrated that the BinNNMS outperformed the k-modes clustering in terms of visual criteria, as well as quantitative clustering quality criteria such as the adjusted Rand index, the normalized mutual information and the quantization error. In the next chapter we will present other approaches. It is the combination of supervised and unsupervised models.

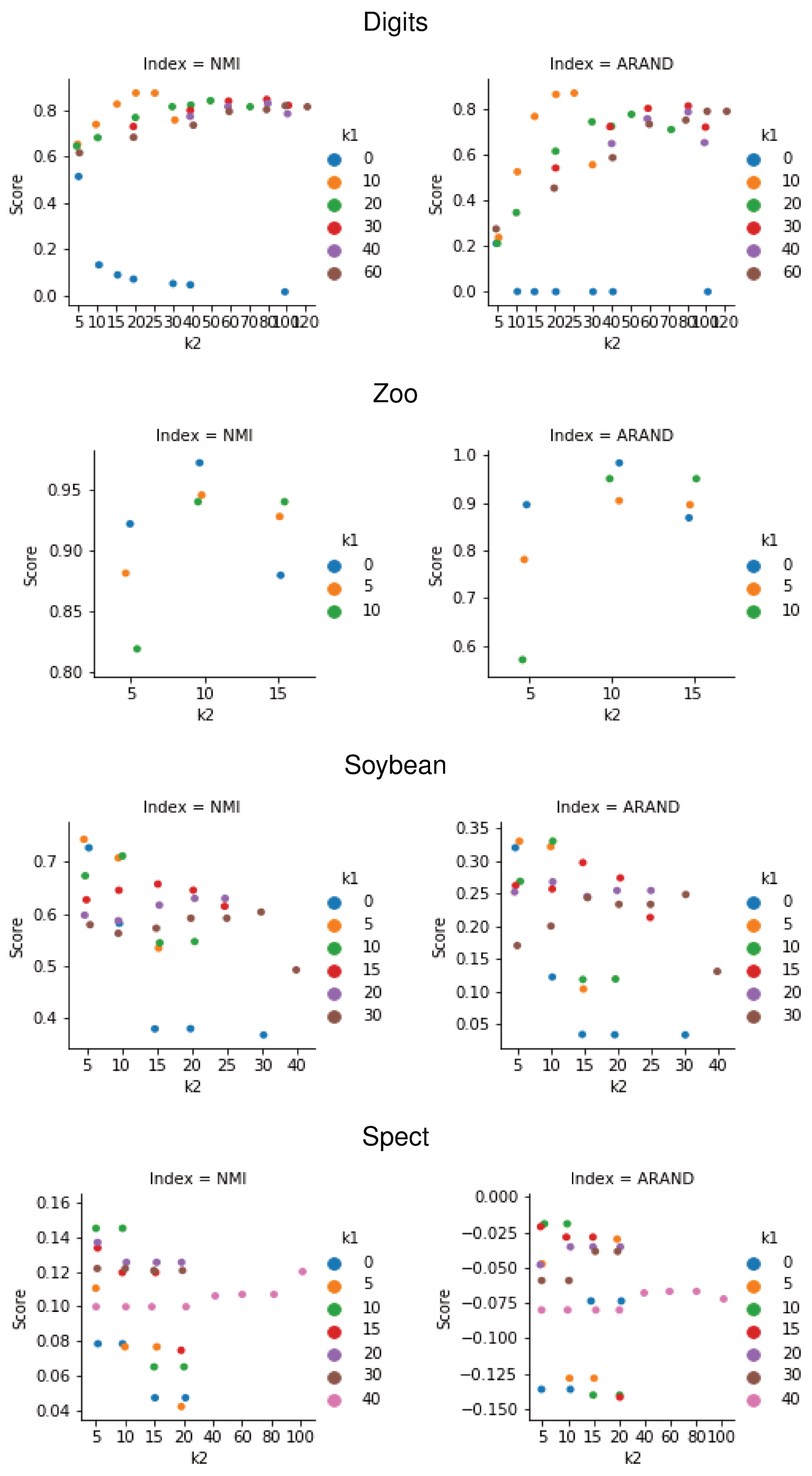


Figure 4.1: Evolution of the cluster quality indices (NMI and ARAND) as functions of the k_1 and k_2 tuning parameters for the BinNNMS for the Digits, Zoo, Soybean and Spect datasets.

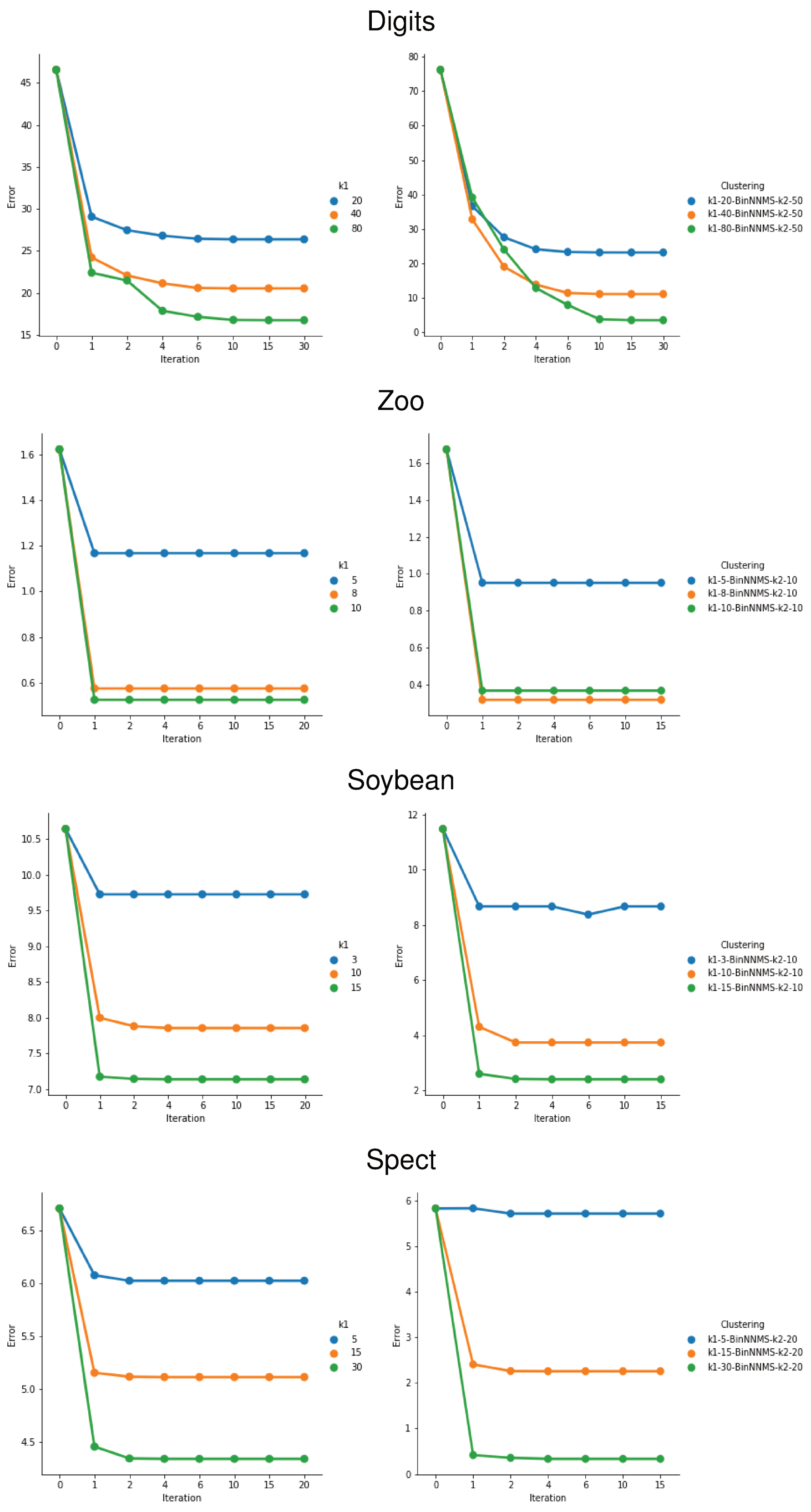


Figure 4.2: Evolution of quantization errors as a function of the k_1 and k_2 tuning parameters in BinNNMS for the Digits, Zoo, Soybean and Spect datasets. Left. Quantization errors between the data points and the target prototypes. Right. Quantization errors between the data points and the intermediate median centers in the BGA task and the cluster prototypes.

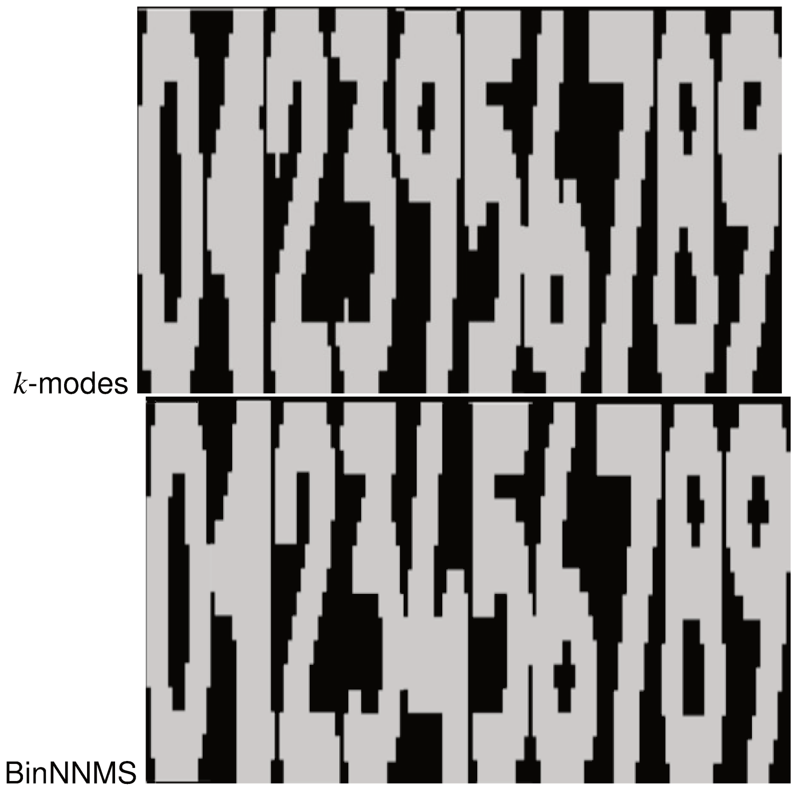


Figure 4.3: Comparison of the k-modes and BinNNMS clustered images for the Digits dataset.

Part II

Model Combination: Enriched Supervised Learning with Clustering

Chapter 5

Clusterwise: Clustering and Partial least squares regression

5.1 Introduction

In modern data analysis, many problems belong to the regression family which consists of explaining one or more variables (known as the response) with respect to other observed variables (known as the explanatory variables). Many types of regression models exist to solve specific problems. A widely used regression method is Multivariate Linear Regression (MLR) where many explanatory variables are linked to a specific response variable by a parametric linear model [44]. MLR is most suited to problems where the number of observations is larger than the number of explanatory variables. On the other hand, when the number of explanatory variables is larger (e.g. high dimensional data), then there tends to be important colinearities between them which implies that MLR is not effective. Partial least squares regression (PLS) is a linear regression model with latent features for high-dimensional data [17, 21]. Standard PLS defines new components by maximizing the covariance between components from two different blocks of variables (data matrix $\mathbf{X}$ and its response matrix $\mathbf{Y}$). In addition, standard PLS does not require any distributional assumptions for the error distributions, since they do not need to be normal with parametric distributions.

Applying PLS to massive high-dimensional data is problematic because the data processing is computationally expensive. This has limited large-scale applications of PLS in practice. To reduce the computational burden, researchers usually apply a specific model design, such as the Clusterwise PLS [127, 17, 21]. In this chapter, we address a Clusterwise problem where the response variables $\mathbf{y}$ are explained by the explanatory variables $\mathbf{x}$, organized into clusters. The statistical model which results from simultaneously treating all observations (i.e. a single macro-cluster) may be of low prediction quality. To overcome this problem we use two nested levels of clustering (i.e. macro- and micro- clusters) as illustrate in Figure 5.1 and compute one model per macro-cluster based on a micro-batch strategy where micro-clusters are shifted from one macro-cluster to another.

A standard approach to obtain clusters within a regression framework is Clusterwise regression (which is also known as typological regression) [20, 14]. Clusterwise regression assumes that there is an underlying clustering structure of the observations and that each cluster can be revealed by the fit of a specific

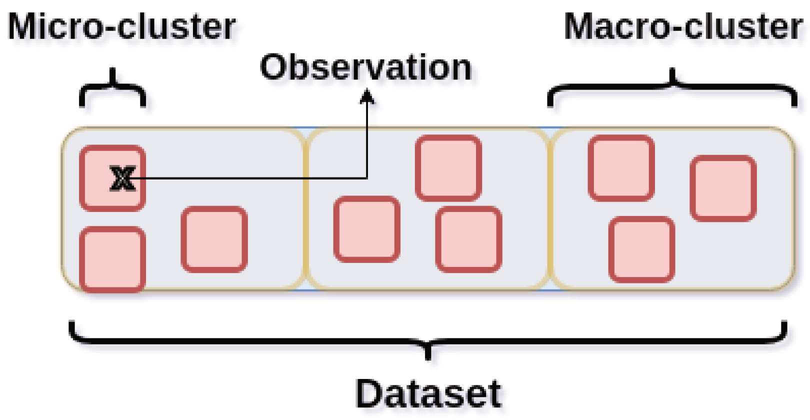


Figure 5.1: Micro and macro clusters

regression model based on micro-clusters. More formally, Clusterwise regression simultaneously looks for a partition of the observations into clusters which minimizes the overall sum of squared error. For Clusterwise methods, a crucial component is to describe the local relationships between the variables measured on the observations within the same cluster. This is handled in this chapter by micro-batch approaches.

5.2 Related works

Existing Clusterwise methods seek clusters within a regression framework while simultaneously minimizing the sum of squared error computed over all the clusters. These methods can be viewed as extensions of the K-means clustering from unsupervised learning to the regression set-up. As in standard regression, ordinary least squares or maximum likelihood estimation can be used to get the quality of the regression coefficients. These K-means like algorithms, based on a least square error criterion, have been proposed by [5, 10]. A multivariate regression for heterogeneous data which takes into account both the between- and the within-cluster variability has also been proposed [118].

To detect categorical differences in underlying regression models on the other hand, Spath [14] developed Clusterwise Regression (CR) which clusters the data points based on the underlying regression model. Related methods exist within the mixture and latent class framework [20]. Other related methods are the principal component regression (PCR) [11] and partial least square regression (PLS) [62, 57] which have also been proposed to deal with multicolinearity, small sample size, or large number of variables. Specifically, PLS reduces the explanatory variables to those which are as maximally related (in terms of squared covariance) as possible to the objective function.

In the framework of component-based path-modeling methods, several Clusterwise methods have been applied in the marketing field (for an early review, refer to [76]). In [46] the authors propose the widely-used finite-mixture PLS (FIMIX-PLS) which assumes multivariate normally distributed data. In [70], fuzzy Clusterwise generalized structured component analysis (FCGSCA) is proposed. In [83] the authors propose REBUS-PLS which came from the hierarchical clustering based on a similarity measure defined from the residuals coming from the same models. [124] proposed PLS-IRRS which identifies homogeneous clusters that have similar residual values.

In the field of multigroup analysis where the groups of observations are known *a priori* Clusterwise simultaneous component analysis (CW-SCA) seeks clusters among groups of observations rather than among observations [96]. It is worth noting that likelihood-based methods are relevant for data exploration or modeling but are unable to be utilized for prediction as dependent values are needed to compute the likelihood.

5.3 Partial Least Square (PLS)

We use the common notation where scalars are defined as italic lower case (x, y), vectors are in bold lower case $(\mathbf{x}, \mathbf{y})$ and matrices as bold upper case $(\mathbf{X}, \mathbf{Y})$.

PLS regression is a technique that generalizes and combines features from principal component analysis and multiple regression. Let $\mathbf{X} = \{\mathbf{x}_1, \dots \mathbf{x}_N\}$ be N vector observations $\mathbf{x_i} = (x_{i1}, \dots, x_{ip}) \in \Re^p$, described by the vector response variables $\mathbf{Y} = \{\mathbf{y}_1, \dots, \mathbf{y}_n\}$ where $\mathbf{y}_i = (y_{i1}, \dots, y_{iq}) \in \Re^q$. Each pair of explanatory and response variables in the model is denoted by the concatenated vector $\mathbf{z}_i = (\mathbf{x}_i; \mathbf{y}_i)$. The matrices $\mathbf{Y}$ and $\mathbf{X}$ are assumed to be pre-whitened (i.e., the sum of each variable is zero and its norm is one). The superscript T denotes the matrix transpose operation, e.g. $\mathbf{X}^T$ and $\mathbf{I}$ the identity matrix.

PLS regression is particularly well-suited when the matrix of explanatory variables $\mathbf{X}$ has more features p than observations, and when there is multicollinearity among the $\mathbf{X}$ values. The goal of PLS regression is to predict $\mathbf{Y}$ from $\mathbf{X}$ and to describe their common structure. When $\mathbf{Y}$ is a vector and $\mathbf{X}$ is sufficiently regular, this goal could be accomplished using ordinary multiple regression. PLS solves the problem that arises when the number of observations (N) is much lower than the number of variables (p).

PLS regression performs a simultaneous decomposition of $\mathbf{X}$ and $\mathbf{Y}$ with the constraint that these components capture as much as possible of the covariance between $\mathbf{X}$ and $\mathbf{Y}$. More formally, $\mathbf{X}$ and $\mathbf{Y}$ are decomposed as follows:

$$\mathbf{X} = \mathbf{T}\mathbf{P}^T + \mathbf{E}, \; x_{ik} = \sum_{j=1}^{r} t_{ij} p_{kj} + e_{ik} \tag{5.1}$$

$$\mathbf{Y} = \mathbf{U}\mathbf{Q}^T + \mathbf{F}, \; y_{im} = \sum_{j=1}^{r} u_{ij} q_{mj} + f_{im} \tag{5.2}$$

where $\mathbf{T}$ and $\mathbf{U}$ are the r-dimensional latent representations of $\mathbf{X}$ and $\mathbf{Y}$ of size of $N \times r$, $\mathbf{P}$ and $\mathbf{Q}$ are the loading matrices with size of $p \times r$ and $q \times r$, $\mathbf{E}$ and $\mathbf{F}$ are the residual matrices. Thus PLS implements the following optimization problem:

$$(\mathbf{p}, \mathbf{q}) = \text{argmax}_{||\mathbf{p}||=||\mathbf{q}||=1} \; cov(\mathbf{T}, \mathbf{U})$$

where $\mathbf{p}, \mathbf{q}$ are columns of the $\mathbf{P}, \mathbf{Q}$ respectively

Once the decomposition of the $\mathbf{X}$ and $\mathbf{Y}$ is carried out, we can continue with computing the regression coefficients. The latter are defined as the minimizers of the residual error

$$\text{argmin}_{\mathbf{B}}||\mathbf{Y} - \mathbf{XB}||^2$$

where $\mathbf{B}$ is a p by q regression coefficient matrix. Suppose there is a linear relationship such that

$$\mathbf{U} = \mathbf{TD} + \mathbf{H}$$

where $\mathbf{D}$ is an $r \times r$ diagonal matrix, then

$$\mathbf{Y} = \mathbf{TC}^T + \mathbf{F}^*$$

where $\mathbf{C}^T = \mathbf{DQ}^T$ and $\mathbf{F}^* = \mathbf{HQ}^T + \mathbf{F}$. Based on the previous expressions, the response matrix is expressed as

$$\mathbf{Y} = \mathbf{XP}(\mathbf{P}^T\mathbf{P})^{-1}\mathbf{CT} + \mathbf{F}^*\mathbf{EP}(\mathbf{P}^T\mathbf{P})^{-1}\mathbf{CT}.$$

The PLS regression coefficient is thus

$$\mathbf{B} = \mathbf{P}(\mathbf{P}^T\mathbf{P})^{-1}\mathbf{C}^T$$

For the brevity, we have discussed the PLS where the variables form a single macro-cluster. When the data consist of non-homogeneous macro-clusters, it is necessary to decompose the regression according to these macro-clusters, for example, using the Clusterwise techniques described in Section 5.2. These techniques have many bottlenecks as the scalability for a large number of observations and variables. One of the main objectives of contribution in this chapter is to integrate the micro-batch processing into the Clusterwise PLS to resolve these scalability issues.

5.4 New model: Micro-Batch Clusterwise PLS

Our proposed approach mb-CW-PLS (micro-batch Clusterwise Partial Least Squares) has the following properties:

- Nested clustering: we combine the PLS with two levels of clustering (macro- and micro-clustering) and micro-batch optimization approaches to create a new Clusterwise method that can find the underlying structure of the observations and provide each macro-cluster of observations with its own set of regression coefficients.
- Micro-Batch processing (divide and conquer strategies): rather than move a single observation from a macro-cluster to another to calculate the regression models in a cross validation approach, we move an entire micro-cluster of points, which we call the micro-cluster shift. These micro-clusters are computed using the k-means clustering, where k is set as the ratio of dataset size to the desired micro-cluster size.

- Scalability: we use a distributed framework based on Apache Spark/Scala to accelerate the initialization process and to compute the regression models. For each cross validation, we distribute initializations over the slave processes. Each slave selects the optimal initialization and submits it to the master process, which in turns selects the optimal one among all the results received from the slaves. This decreases execution times inversely proportional to the number of slaves.

- Usability: the Spark/Scala implementation requires the configuration of a small number of hyper parameters, and can be utilized in a distributed system or even on a stand-alone terminal with minimal effort, which enlarges the scope of usability of the PLS.

Mathematical details

We assume that the N observations are clustered into K micro-clusters (or buckets), $C = \{C_1, \ldots, C_k, \ldots, C_K\}$ where $C_k = \{\mathbf{x}_i, \phi(\mathbf{x}_i) = k\}$. Denote ϕ as the assignment function defined as follows : $\phi : \mathbf{x}_i \in \Re^p \to \{1, \ldots, k, \ldots, K\}$ (eg. euclidean distance). The partition C of the micro-clusters is carried as an initialization before running the PLS models. This step can be done using clustering approaches such as k-means.

Denote a second partition level $\mathcal{P} = \{P_1, ..., P_g, ..., P_G\}$ of the micro-clusters set C into G macro-clusters (and $K \ll G$). Therefore G is the number of regression models that will be considered $(PLS_1, ..., PLS_g, ..., PLS_G)$. The second level assignment function Φ is defined as: $\Phi : \{C_1, \ldots, C_k, \ldots, C_K\} \to \{1, \ldots, g, \ldots, G\}$.

We introduce Micro-Bach Clusterwise PLS methods (mb-CW-PLS) by assuming two phases of clustering: in the first level the N observations are clustered in K fixed micro-clusters, and in the second level, the K micro-clusters are grouped into G macro-clusters where each macro-cluster has a specific PLS regression model. Therefore the mb-CW-PLS algorithm searches for an optimal partition of the K micro-clusters into G macro-clusters as well as the corresponding set of regression coefficient matrices $(\mathbf{B}_1, ..., \mathbf{B}_G)$ that minimize the overall error described in Equation 5.3:

$$\begin{aligned} L(\mathcal{P}, \mathbf{B}) &= \sum_{g=1}^{G} \sum_{\mathbf{x}_i \in P_g} \|\mathbf{Y}_g - \mathbf{X}_g \mathbf{B}_g\|^2 \\ &= \sum_{g=1}^{G} \sum_{C_k \in P_g} \sum_{\mathbf{x}_i \in C_k} \|\mathbf{y}_i - \mathbf{x}_i \mathbf{b}_g^T\|^2 \end{aligned} \tag{5.3}$$

where $\mathbf{X}g$ and $\mathbf{Y}_g$ denote the data matrices of the gth macro-cluster respectively of $\mathbf{X}$ and $\mathbf{Y}$.

The classical Clusterwise PLS does not provide estimators in reasonable time for large N. Based on the traditional sequential algorithm, each observation $\mathbf{x}_i$ is assigned to its optimal cluster and the overall error is updated whenever one observation switches cluster.

In order to overcome this problem and to ensure that the error decreases monotonically at each iteration of the algorithm, we propose to use a new sequential

$$\mathbf{X} = \begin{bmatrix} \mathbf{X}_1 = \begin{bmatrix} x_{11} & \dots & x_{1p} \\ & \vdots & \\ x_{N_1 1} & \dots & x_{N_1 p} \end{bmatrix} \\ \vdots \\ \mathbf{X}_g = \begin{bmatrix} x_{11} & \dots & x_{1p} \\ & \vdots & \\ x_{N_g 1} & \dots & x_{N_g p} \end{bmatrix} \\ \vdots \\ \mathbf{X}_G = \begin{bmatrix} x_{11} & \dots & x_{1p} \\ & v \dots & \\ x_{N_G 1} & \dots & x_{N_G p} \end{bmatrix} \end{bmatrix} \begin{bmatrix} \phi(\mathbf{x}_1) & \Phi(C_{\phi(\mathbf{x}_1)}) \\ \vdots & \vdots \\ \phi(\mathbf{x}_N) & \Phi(C_{\phi(\mathbf{x}_N)}) \end{bmatrix} \quad \text{and} \quad \mathbf{Y} = \begin{bmatrix} \mathbf{Y}_1 = \begin{bmatrix} y_{11} & \dots & y_{1q} \\ & \vdots & \\ y_{N_1 1} & \dots & y_{N_1 q} \end{bmatrix} \\ \vdots \\ \mathbf{Y}_g = \begin{bmatrix} y_{11} & \dots & y_{1q} \\ & \vdots & \\ y_{N_g 1} & \dots & y_{N_g y} \end{bmatrix} \\ \vdots \\ \mathbf{Y}_G = \begin{bmatrix} y_{11} & \dots & y_{1q} \\ & \vdots & \\ y_{N_G 1} & \dots & y_{N_G q} \end{bmatrix} \end{bmatrix}$$

micro-batch algorithm. Each micro-cluster C_k is assigned to its optimal macro-cluster P_g and the overall error is updated whenever a micro-cluster switches to a different macro-cluster.

Therefore the main idea, instead of moving a single observation $\mathbf{x}_i$ from its original macro-cluster to other macro-clusters, we move the entire micro-cluster C_k which contains $\mathbf{x}_i$ and then re-compute the regression model.

Starting with an initial partition of macro-clusters
$\mathcal{P}^{(0)} = \{P_0^{(0)}, \dots, P_G^{(0)}\}$, the algorithm constructs iteratively a sequence $\{\mathcal{P}^{(s)}, \mathbf{B}^{(s)})\}$, $s > 0$, in the following way:

- For each $g \in \{1 \dots, G\}$, $\mathcal{P}^{(0)}$ is given by the least square estimators of the PLS regression using the points of the macro-cluster P_g.

- Given $(\mathcal{P}^{(s)}, \mathbf{B}_g^{(s)})$, then for each macro-cluster

  $P_g \in \{P_1, \dots, P_G\}$,

$$\begin{aligned} \mathcal{P}_g^{(s+1)} = \{(\mathbf{X}_i, \mathbf{Y}_i) : \|\mathbf{Y}_i - \mathbf{X}_i \mathbf{B}_{g_1}^{(s)}\|^2 \\ < \|\mathbf{Y}_i - \mathbf{X}_i \mathbf{B}_{g_2}^{(s)}\|^2, \forall g_1 \neq g_2\}. \end{aligned} \tag{5.4}$$

  The resulting $\mathbf{B}^{(s+1)}$ are the PLS estimators using the data partitioned by $\mathcal{P}^{(s+1)}$. The sequence $\{(\mathcal{P}^{(s)}, \mathbf{B}^{(s)})\}_{s \geq 0}$ is such that

$$L(\mathcal{P}^{(s)}, \mathbf{B}^{(s)}) \geq L(\mathcal{P}^{(s+1)}, \mathbf{B}^{(s+1)}), \ \forall s \geq 0$$

  and so it is convergent.

- Repeat above step for all micro-clusters C_k which are re-assigned to their optimal macro-cluster P_g. This guarantees that the overall error decreases monotonically at each change in assignment.

The mb-CW-PLS algorithm finds simultaneously an optimal partition of the fixed K micro-clusters (buckets) into
$\mathcal{P} = \{P_1, ..., P_g, ..., P_G\}$, $P_g = \{C_k, \Phi(C_k) = g$ and
$\forall \mathbf{x}_i \in C_k \ \phi(x_i) = k\}$ and the regression models associated to each macro-cluster g. Finally the best cross-validated try is selected based on the Root Mean

Squared Error (RMSE) [58] score. This metric evaluates the prediction accuracy of the fitted regression,

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (\hat{y}_i - y_i)^2}{N}}$$

where $\hat{y}_i$ is the predicted value, y_i the true value and N the number of data points. This method is summarized in the Algorithm 17.

Algorithm 17 mb-CW-PLS Clusterwise algorithm

```
1:  procedure Clusterwise($\mathbf{X}, \mathbf{G}, \mathbf{CV}, \mathbf{INIT}$)
    // Generate the cross validated dataset
2:      $\cup_{i=1}^{\mathbf{CV}} \mathbf{X_i} = \mathbf{X}$
     // These steps are distributed over nodes
3:      for i := 1 to CV do
4:          $\mathbf{X_{curr}} = \cup_{i \neq j}^{\mathbf{CV}} \mathbf{X_i}$
5:          for j := 1 to INIT do
            // Generate randomly filled classes
6:              $\cup_{g=1}^{\mathbf{G}} \mathbf{X_g} = \mathbf{X_{curr}}$
7:              Choose randomly $x$ where $x \in \mathbf{X_{curr}}$
8:              for u := 1 to G do Apply $PLS_g$ on $X_u$ with $\mathbf{x} \in X_u$
9:              for u := 1 to G do Apply $PLS_g$ on $X_u$ with $\mathbf{x} \notin X_u$
10:             Choose the class $b$ with the best regression score (least error)
11:             Add $\mathbf{x}$ to $\mathbf{X}_b$
12:         Select the best initialization $\mathbf{b\text{-}init}$
13:         Apply corresponding model on the test set
14:     Select the best model among $\mathbf{CV}$ through RMSE
```

5.5 Experiments

In order to get an efficient distributed implementations, we decided to apply the Clusterwise logic through Spark framework. As explained previously in order to apply a regression in the fastest way, data should be inside the machine which performs linked operations.

Then rather than to distribute pieces of data through nodes in order to avoid shuffling, we put the entire dataset on each node using the *sc.broadcast(dataset)* function. Once this step is done, we distribute simultaneously every initializations which correspond to $CV \times INIT$ homogeneously over nodes. It allows each node to perform sequentially their $\frac{CV \times INIT}{nodes}$ Clusterwise initializations, via a *mapPartitions* function where the number of Spark partitions is $CV \times INIT$. The *Map* part achieved we *aggregateByKey* obtained results where the *Key* is a Cross-Validation index. We decided to use the non-parametric k-nearest neighbors method to assign observations to their closest macro-cluster P_g in order to apply the associated PLS_g. Finally, based on RMSE scores, the best model is selected over Cross-Validations.

Dataset	n	$\mathbf{X}$ size, p	$\mathbf{Y}$ size, q
Yacht Hydrodynamics	308	6	1
Forest Fire	517	10	1
Concrete Compressive Strength	1030	8	1
Wine Quality Red	1599	11	1
Wine Quality White	4898	11	1
SimData	400	10	3

Table 5.1: Overview of data sets.

5.5.1 Prediction accuracy vs micro-cluster sizes

In contrast to the regular PLS regression, micro-batch Clusterwise PLS provides G regression models corresponding to G macro-clusters. The iterative cross validation process is as follows: macro clusters are initiated by assigning them randomly each point, for standard Clusterwise, or each micro-cluster C_k for micro-batch Clusterwise. Then the PLS is applied to each macro-cluster, once it is achieved, one observation or one micro-cluster C_k is moved inside all other macro-clusters P_g in order to compute new PLS regressions. Therefore we run one regression per macro-cluster with and without a specific observation or micro-cluster given $2 \times G$ regressions. Once least squares have been compared over all combinations, the point or the micro-cluster C_k is assigned to the macro-cluster P_g which yields the minimal error. For our experiments different datasets listed in Table 5.1 of various sizes have been used. Most of them came from UCI repository [107].

The evolution of the RMSE in Table 5.3 with the value of micro-cluster size N_k shows that we can efficiently decrease the RMSE with our approach (mb-CW-PLS). We observe a property of Clusterwise methods that increasing number of macro-clusters G reduces the RMSE score. On the other hand, a lower G allows to use larger values of N_k. In Table 5.3, N/A entry indicates that we cannot execute the algorithms under these conditions due to risk of generate empty class which will falsify quality of results. First we seek for the optimal number of macro-clusters G, then we search for the optimal size of the micro-clusters (buckets) N_k. The first performance criterion is the Root Mean Square Error of prediction as evaluated with a ten-fold cross-validation procedure. Not surprisingly, mb-CW-PLS always improves the Root Mean Square Error (RMSE) of prediction while taking into account the cluster size N_k.

We observe a property of Clusterwise methods that an increasing number of macro-clusters G reduces the RMSE score. On the other hand, a lower G allows to use larger values of N_k. In Table 5.3, N/A entry indicates that the choice of tuning parameters does not lead to a well-defined solution e.g. if we set high value of N_k we could inconveniently moving every micro-cluster in a macro-cluster to other macro-clusters. However every macro-cluster has to be present in order to compare score of the different generated models. If this happened, then we re-tried with a different initialization, but a threshold of the number of attempts is fixed before considering tuning parameters ill-posed. The number of clusters K in the k-means clustering is $K = N/N_k$, the number of cross validation classes is $CV = 10$, the number of initializations is $INIT = 20$ and the number of k nearest neighbors for PLS model assignation is 20. Note that for $N_k = 1$ is equivalent to the standard CW-PLS.

Figure 5.2 illustrates results with the three output dimensions **Y** from SimData dataset. We observe that the predicted values (in red) are closer to the true values (in blue) for the mb-CW-PLS than with CW-PLS. The corresponding RMSE is referenced in Table 5.3.

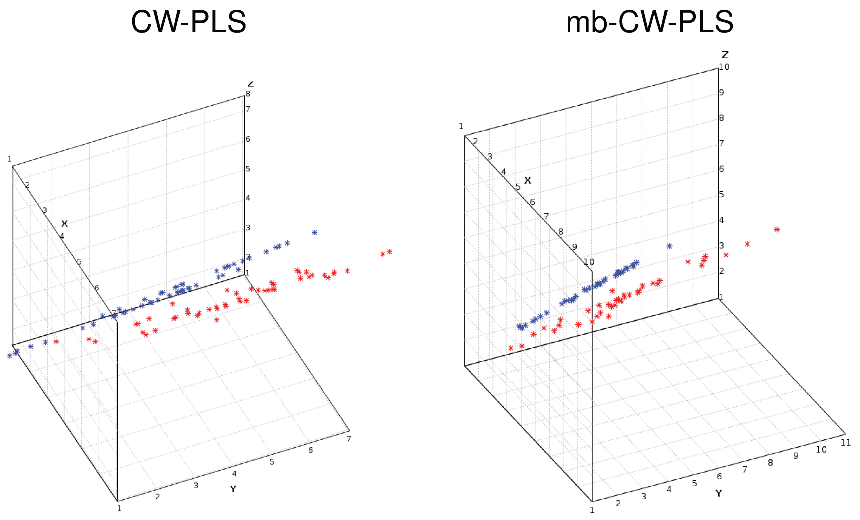


Figure 5.2: Comparison of Clusterwise regression CW-PLS vs mb-CW-PLS for SimData 's **Y**. True responses are in blue and predicted one are in red

5.5.2 Comparison to existing regression methods

To compare our PLS and mb-CW-PLS regressions to other regression methods, we took the experimental data sets with a single response variable ($q = 1$). We then fit models for the OLS, Ridge regression, LASSO and Random Forest using code from the Smile library (*https://haifengl.github.io/smile/*). 500 replicates of 90% random sampled are taken as training data, then the RMSE based on the remaining 10% test data is computed.

The RMSEs are shown in Table 5.2. We observe that PLS provides at least as well as the OLS, Ridge regression and the LASSO, and less well than the Random Forest for the Forest Fire, Concrete Data, Yacht Hydrodynamics and Wine Quality Red. The proposed mb-CW-PLS outperforms these methods, in some cases by a substantial margin, in terms of the RMSE. Furthermore, for the SimData which has 3 response variables ($q = 3$), the PLS and mb-CW-PLS are able to produce results, whereas the other regressions cannot provide results.

Regression method	Forest Fire	Concrete Compressive Strength	Yacht Hydro-dynamics	Wine Quality Red	Sim Data
OLS	0.730 ±0.415	0.626 ±0.000	0.597 ±0.007	0.803 ±0.003	N/A
Ridge regression	0.729 ±0.415	0.626 ±0.000	0.596 ±0.007	0.803 ±0.003	N/A
LASSO	0.729 ±0.416	0.626 ±0.000	0.594 ±0.007	0.803 ±0.003	N/A
Random Forest	**0.715 ±0.362**	0.373 ±0.001	0.289 ±0.008	0.750 ±0.002	N/A
PLS	0.720 ±0.419	0.649 ±0.000	0.594 ±0.007	0.808 ±0.002	0.223 ±0.158
mb-CW-PLS $G = 6$	0.899 ±0.328	**0.00006 ±0.00000**	**0.226 ±0.039**	**0.003 ±0.000**	**0.016 ±0.000**

Table 5.2: Comparison of the different regression methods. The first row indicates the mean RMSE followed by the standard deviation. N/A entry indicates that we cannot execute the algorithms under these conditions

5.5.3 Comparison of prediction accuracy and execution times

Table 5.3 shows the RMSE scores and the execution times for CW-PLS and mb-CW-PLS. Let recall that CW-PLS is equivalent to mb-CW-PLS with $N_k = 1$. Figure 5.4a illustrates some of these results indicating that the higher we set G the better can be our results with a smaller standard deviation. This result is partially intuitive in that sense that the more specialized cluster we build, the more effective will be model built on them. The counter part of this strategy is a risk of overfitting.

For most datasets, mb-CW-PLS presents similar RMSE scores to classical CW-PLS ones. Indeed execution times is much faster as Figure 5.3c highlights it, especially on bigger datasets. The micro-batch processing which produces the micro-clusters allows these decreases in execution time without sacrificing too much of the prediction accuracy as it is exposed on Figure 5.4b-5.4c. In some cases as with the Wine Quality Red dataset, we even observe better results. An interesting thing holds in the modest RMSE evolutions over different N_k values for $N_k > 1$.

5.5.4 Scalability

The results obtained in Table 5.3 were carried out in a local environment (i.e. a stand-alone terminal). We now examine the performance of mb-CW-PLS in a true distributed computing set-up. To execute these experiments, we used the Grid5000 [105] infrastructure which is one of the biggest French's laboratories clusters. We used a set-up with two times 8 core Intel Xeon E5-2630v3 or two

Dataset	CW-PLS	mb-CW-PLS $N_k = 5$	mb-CW-PLS $N_k = 10$	mb-CW-PLS $N_k =$ $custom1$	mb-CW-PLS $N_k =$ $custom2$
Forest Fire $G = 4$ 6 slaves INTEL	**0.06625** ±0.0132 26.3s	0.07771 ±0.00843 16.6s	0.08627 ±0.00952 15.6s	0.0857 ±0.00304 14.9s $N_k = 15$	0.08577 ±0.00728 15.1s $N_k = 20$
Yacht Hydrodynamics $G = 2$ 6 slaves INTEL	**0.0101** ±4.0E-4 15.6s	0.01852 ±3.7E-4 13.2s	0.01888 ±0.00138 12.8s	0.01821 ±0.00103 13.3s $N_k = 15$	0.01899 ±0.00182 13.2s $N_k = 20$
Yacht Hydrodynamics $G = 4$ 6 slaves INTEL	**0.01066** ±4.1E-4 15.0s	0.02002 ±8.2E-4 13.1s	0.01815 ±0.00146 13.1s	0.0167 ±8.0E-4 12.9s $N_k = 15$	0.01435 ±7.9E-4 12.8s $N_k = 20$
Yacht Hydrodynamics $G = 6$ 8 slaves AMD	**0.00146** ±5.0E-5 32.5s	0.01227 ±8.9E-4 28.2s	0.01233 ±6.7E-4 29.2s	N/A $N_k = 15$	N/A
Wine Quality Red $G = 4$ 8 slaves AMD	0.01269 ±4.8E-4 522.1s	**0.00969** ±1.0E-4 **141.6s**	0.00982 ±1.0E-5 **93.5s**	0.00989 ±7.0E-5 **63.1s** $N_k = 20$	0.00971 ±8.0E-5 **50.2s** $N_k = 40$
Wine Quality White $G = 4$ 8 slaves AMD	Not finished after 5h of computation	0.05567 ±3.4E-4 3878.7s	0.05545 ±4.0E-5 2032.0s	0.05489 ±3.9E-4 1059.2s $N_k = 20$	**0.05481** ±2.5E-4 **587.3s** $N_k = 40$
Concrete Compressive Strength $G = 4$ 8 slaves AMD	5.0E-5 ±0.0 126.9s	**5.0E-5** ±0.0 **51.9s**	**5.0E-5** ±0.0 **41.9s**	**5.0E-5** ±0.0 **37.6s** $N_k = 15$	N/A
Sim Data $G = 2$ 8 slaves AMD	**0.02203** ±5.2E-4 40.5s	0.10793 ±0.00558 30.1s	0.10281 ±0.01839 28.7s	0.10541 ±0.00389 28.2 $N_k = 15$	N/A
Sim Data $G = 3$ 8 slaves AMD	**0.02383** ±0.00105 38.3s	0.09959 ±0.00473 30.0s	0.09464 ±0.00611 29.8s	0.09479 ±0.00274 28.7s $N_k = 15$	N/A
Sim Data $G = 4$ 8 slaves AMD	**0.02723** ±0.00156 38.2s	0.10054 ±0.00688 30.0s	0.08356 ±0.00497 29.3s	0.09095 ±0.00731 30.0s $N_k = 15$	N/A
Sim Data $G = 5$ 8 slaves AMD	**0.02951** ±0.00233 42.7s	0.0909 ±0.00376 30.5s	0.0894 ±0.00572 30.5s	0.0707 ±0.00606 31.7 $N_k = 15$	N/A
Sim Data $G = 6$ 8 slaves AMD	**0.02317** ±8.4E-4 36.0s	0.09274 ±0.0058 29.7s	0.07298 ±0.00552 32.0s	0.07436 ±0.00828 36.5 $N_k = 15$	N/A

Table 5.3: Comparison of CW-PLS and mb-CW-PLS. The first

times 12 cores AMD Opteron 6164 HE CPUs and 128 Gb RAM per node.

Table 5.4 presents the average time spent for four runs executing a full Clusterwise workflow, which consists to train and test models to select the best one. Total init $CV \times Init$ corresponds to the total number of initializations made. Figure 5.3a shows that the execution times decreases efficiently with the number of nodes.

# total init	local	2 slaves	4 slaves	6 slaves	8 slaves
260	276.6	170.3	114.6	107.4	93.9
500	509.7	303.6	191.2	162.7	146.7

Table 5.4: Comparison of execution times with different number of initializations and slave processes for the Yacht Hydrodynamics using Intel setup.

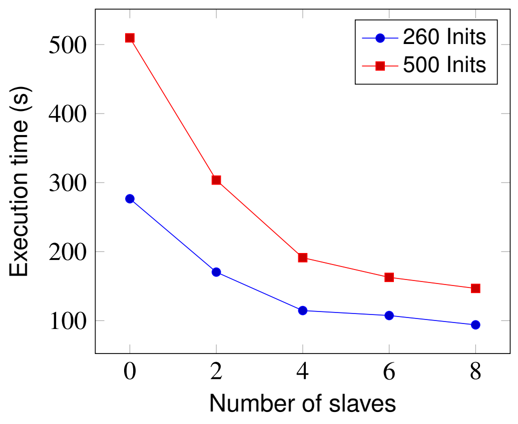
In order to maximize efficiency, we recommend to have 2-3 times more Spark partitions than the total number of cores among every nodes. This approach is not optimal if we have a large data set with few initializations as we do not utilize fully the distributed computational power of Spark. We can also observe that when the number of core nodes is exceeded by the number of initializations, the growth in execution time is almost linear with respect to the number of initializations. As our distributed set-up consists of nodes with 32 cores, a reduction in execution time is observed when the number of initializations is higher than 32.

Figure 5.3b shows the linearity of the problem with the increase of the number of initializations with a set-up of 8 slave nodes. This results illustrate that initializations are well distributed among nodes which allows an efficient computations of the algorithm.

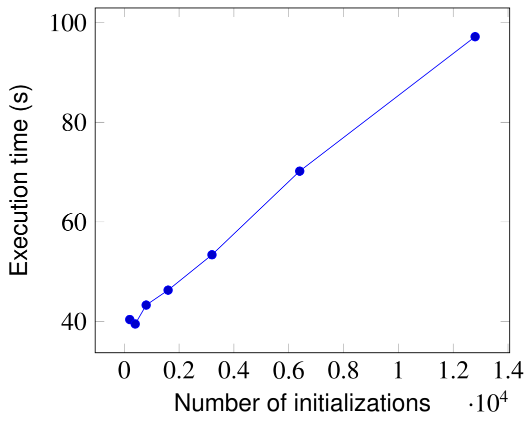
5.6 Conclusion

This work presents a new Clusterwise PLS regression algorithm, which brings multiple improvements. First, the micro-batch processing facilitates a drastic reduction in execution times, keeping the same magnitude of prediction accuracy. Second, the distributed implementation enables the test with a large number of initializations in order to find the optimal cross-validated model. One key aspect of the algorithm is the number of regressions needed for one complete run which is $2 \times G \times CV \times INIT \times N \approx 1000 \times N$ leading to a quadratic time complexity. Our micro-cluster approach reduces exclusively the number of regressions from N to $\frac{N}{N_k}$, but we still compute PLS on the whole dataset preventing better time performance. In our future works we will examine methods to decrease the execution time and also to increase prediction quality. Unfortunately, an inevitable bottleneck is the impossibility to decrease the inner complexity of a regression and Clusterwise strategy is very regression greedy.

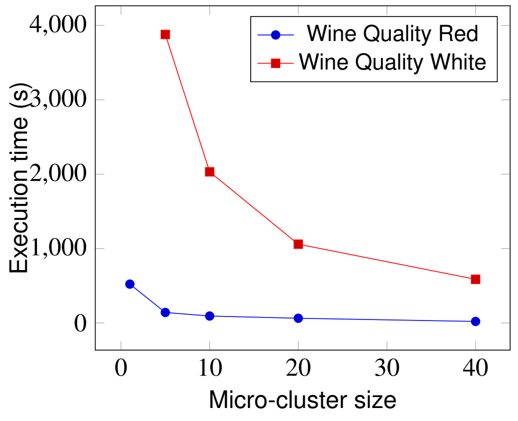
We have seen that combining supervised and unsupervised learning can lead to better result but often the quality of results does not depend exclusively on the



(a) Evolution of the execution times with respect with the number of slaves. Set-up with Intel CPUs

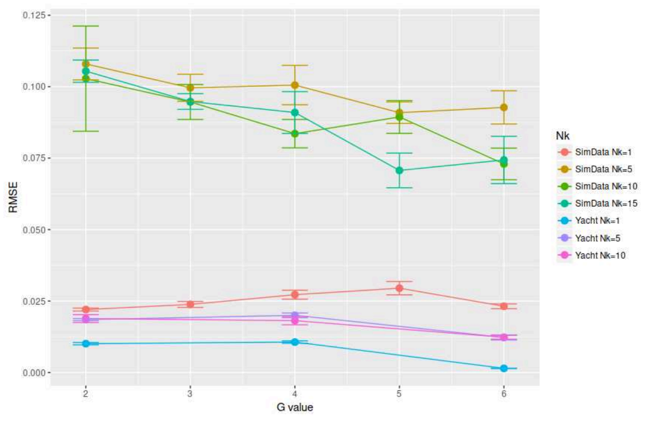


(b) Evolution of the execution times with respect with the number of initialization on Yacht dataset. Set-up with 8 AMD slaves

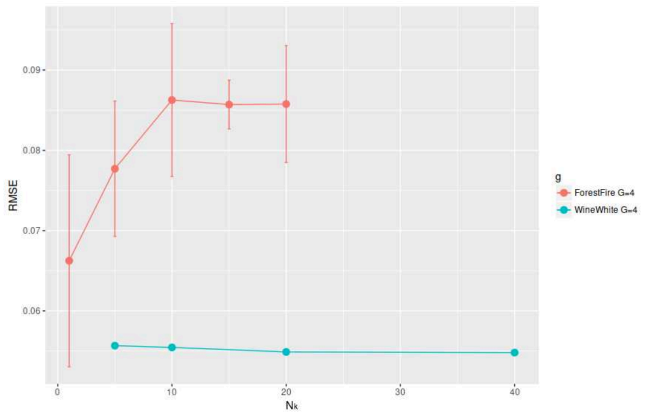


(c) Evolution of the execution times with respect with of the size of micro-cluster for different datasets

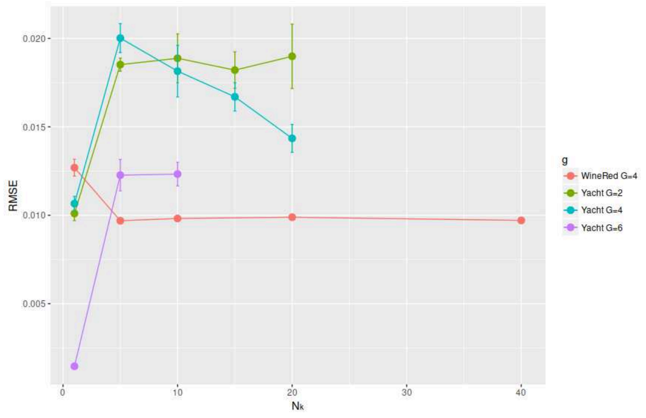
Figure 5.3: Evolution of the execution times



(a) RMSE with standard deviation per G value



(b) RMSE with standard deviation per micro-cluster size



(c) RMSE with standard deviation per micro-cluster size

Figure 5.4: Root Mean Square Error (RMSE)

learning algorithms but on the features composing the training data set. Selecting them wisely becomes a challenge by itself which deserves in depth studies.

Chapter 6

Distributed Features Selection

6.1 Introduction

The use of Rough Set Theory (RST) [72, 100] for feature selection is one approach that has proved successful and more efficient in comparison to a variety of state-of-the-art feature selection methods [82]. However, despite being a powerful feature selection technique, most of the traditional rough set based algorithms are sequential algorithms, computationally expensive and can only deal with small datasets. The computational intensive nature of RST and its incapacity to deal with high dimensional data arise from the necessity to generate all the possible combinations of features at once, process them in turn to finally select the most relevant set. However, as the number of features is increasing this task becomes challenging and this is where the RST inadequacy arises [53]. It is quite unmanageable to generate the set of all possible feature combinations due to hardware and memory constraints. Thus, in this chapter, we present two novel efficient distributed rough set based algorithms, named Sp-RST and LSH-dRST, for large-scale data pre-processing. The first approach is based solely on the MapReduce paradigm and the distributed system. The second proposed solution LSH-dRST, is based on the Locality Sensitive Hashing (LSH) [37] as presented in chapter 3.

With the aim of choosing the most relevant and pertinent subset of features, a variety of dimensionality reduction techniques were proposed within the Apache Spark framework[1] to deal with big data in a distributed way. Among these are several feature extraction methods such as nn-gram, Principal Component Analysis, Discrete Cosine Transform, etc., and very few feature selection techniques which are the VectorSlicer, the RFormula and the ChiSqSelector. To further expand this restricted research, i.e., the development of feature selection parallel methods, lately, some other feature selection techniques were proposed which are based on evolutionary algorithms[2] [120]. These include a generic implementation of greedy information theoretic feature selection methods[3], and an improved implementation of the classical minimum Redundancy and Maximum Relevance feature selection method [60]. This implementation includes several optimizations such as cache marginal probabilities, accumulation of redundancy (greedy approach) and a data-access by columns[4]. However, most of these techniques suffer from some

[1]https://spark.apache.org/docs/2.2.0/ml-features.html
[2]https://github.com/triguero/MR-EFS
[3]https://github.com/sramirez/spark-infotheoretic-feature-selection
[4]https://github.com/sramirez/fast-mRMR

limitations as they involve the user for parameterization, require noise levels to be specified, rank features leaving the user to choose its own subset, require the user to state how many features are to be chosen, or they must supply a threshold that determines when the algorithm should terminate. All of these require the user to make a decision based on its own (possibly faulty) judgment. To overcome these shortcomings, the need for a filter method that does not require any additional information to function properly seems essential. Rough Set Theory (RST) can be used as such a technique [82].

The use of RST for feature selection has proved more efficient in comparison to a variety of state-of-the-art feature selection methods [82]. Over the past years, RST has become a topic of great interest to researchers and has been applied to many domains such as in classification [43], and clustering [49]. This success is due to many aspects of the theory among them the possibility to analyze the facts hidden in data, it does not require any additional information about the data and it is able to find a minimal knowledge representation [39]. This is achieved by making use of the granularity structure of the provided data only. Although RST has been widely used as a powerful filter feature selection technique, most of the traditional rough set based algorithms are sequential algorithms, computationally expensive and can only deal with non-large data sets. The prohibitive complexity of RST comes from the search for an optimal feature subset through the competing of an exponential number of candidate subsets. Although it is an exhaustive method, this is quite impractical for big data as it becomes unmanageable to generate the set of all possible feature combinations.

6.2 Rough Sets for Feature Selection

6.2.1 Basic concepts

In RST, an *information table* is defined as a tuple $T = (U, A)$ where U and A are two finite, non-empty sets, U the *universe* of primitive objects and A the set of attributes. Each attribute or feature $a \in A$ is associated with a set V_a of its value, called the *domain* of a. We may partition the attribute set A into two subsets C and D, called *condition* and *decision* attributes, respectively.

Let $P \subset A$ be a subset of attributes. The indiscernibility relation, denoted by $IND(P)$, is the central concept to RST and it is an equivalence relation which is defined as:

$$IND(P) = \{(x, y) \in U \times U : \forall a \in P, a(x) = a(y)\},$$

where $a(x)$ denotes the value of feature a of object x. If $(x, y) \in IND(P)$, x and y are said to be *indiscernible* with respect to P.

The family of all equivalence classes of $IND(P)$, referring to a partition of U determined by P, is denoted by $U/IND(P)$. Each element in $U/IND(P)$ is a set of indiscernible objects with respect to P. The equivalence classes $U/IND(C)$ and $U/IND(D)$ are called *condition* and *decision* classes, respectively. For any concept $X \subseteq U$ and attribute subset $R \subseteq A$, X could be approximated by the R-*lower* approximation and R-*upper* approximation using the knowledge of R. The lower approximation of X is the set of objects of U that are surely in X, defined as:

$$\underline{R}(X) = \bigcup\{E \in U/IND(R) : E \subseteq X\}.$$

The upper approximation of X is the set of objects of U that are possibly in X, defined as:

$$\overline{R}(X) = \bigcup\{E \in U/IND(R) : E \cap X \neq \emptyset\}.$$

The concept defining the set of objects that can possibly, but not certainly, be classified in a specific way is called the *boundary region* which is defined as:

$$BND_R(X) = \overline{R}(X) - \underline{R}(X).$$

If the boundary region is empty, that is $\overline{R}(X) = \underline{R}(X)$, concept X is said to be R-*definable*; otherwise X is a *rough set* with respect to R. The *positive region* of decision classes $U/IND(D)$ with respect to condition attributes C is denoted by $POS_c(D)$ where $POS_c(D) = \bigcup \overline{R}(X)$. The positive region $POS_c(D)$ is a set of objects of U that can be classified with certainty to classes $U/IND(D)$ employing attributes of C.

Based on the positive region, the *dependency of attributes* is defined as:

$$k = \gamma(C, c_i) = \frac{|POS_C(c_i)|}{|U|}$$

measuring the degree k of the dependency of an attribute c_i on a set of attributes C.

6.2.2 Reduction process

Based on these basics, RST defines two important concepts for feature selection which are the $Core$ and the $Reduct$. In RST, a subset $R \subseteq C$ is said to be a D-*reduct* of C if $\gamma(C, R) = \gamma(C)$ and there is no $R' \subset R$ such that $\gamma(C, R') = \gamma(C, R)$. In other words, the $Reduct$ is the minimal set of selected attributes preserving the same dependency degree as the whole set of attributes. Meanwhile, RST may generate a set of reducts, $RED_D^F(C)$, from the given information table. In this case, any reduct from $RED_D^F(C)$ can be chosen to replace the initial information table. The second concept, the $Core$, is the set of attributes that are contained by all reducts, defined as:

$$CORE_D(C) = \bigcap RED_D(C);$$

where $RED_D(C)$ is the D-reduct of C.

Specifically, the $Core$ is the set of attributes that cannot be removed from the information system without causing collapse of the equivalence-class structure. This means that all attributes present in the $Core$ are indispensable.

6.3 Version based on distributed system

6.3.1 General Model Formalization

Sp-RST creates a Resilient Distributed Dataset (RDD) and formalizes it as a given information table defined as T_{RDD}, where $U = \{x_1, \ldots, x_N\}$ is the universe, the conditional attribute set is $C = \{c_1, \ldots, c_V\}$ and the decision attribute $D = \{d_1, \ldots, d_W\}$ corresponds to the class (label) of each T_{RDD} sample. The conditional attribute set C presents the pool from where the most convenient features will be selected.

In order to make our algorithm scalable with the high number of features, we partition the given T_{RDD} into m data blocks based on splits from the conditional attribute set C defined as:

$$T_{RDD} = \bigcup_{i=1}^{m} (C_r) T_{RDD_{(i)}},$$

where $r \in \{1, \ldots, V\}$. Each $T_{RDD_{(i)}}$ is constructed based on r random features selected from C, where

$$\forall T_{RDD_{(i)}} : \nexists \{c_r\} = \bigcap_{i=1}^{m} T_{RDD_{(i)}}.$$

To ensure scalability, rather than applying Sp-RST to T_{RDD} including the whole C set, the distributed algorithm will be applied to every single $T_{RDD_{(i)}}$. At the end all the intermediate results will be gathered from the different m partitions. In such a way, we can guarantee that Sp-RST can be applied to a computable number of features and hence solving the standard RST computational inefficiencies. Algorithm 18 highlights the pseudo-code of our proposed Sp-RST solution.

Algorithm 18 Sp-RST

```
    Inputs: $T_{RDD}$ the information table
            $m$ number of partitions
            $N$ the number of iterations
    Output: $Reduct$
 1: Calculate $IND(D)$
 2: for each iteration $n \in [1, \ldots, N]$ do
 3:     Generate $T_{RDD_{(i)}}$ based on the $m$ partitions
 4:     for each $T_{RDD_{(i)}}$ partition, $i \in [1, \ldots, m]$ do
 5:         Generate $AllComb_{(C_r)}$
 6:         Calculate $IND(AllComb_{(C_r)})$
 7:         Calculate $DEP(AllComb_{(C_r)})$
 8:         Select $DEP_{max}(AllComb_{(C_r)})$
 9:         Filter $DEP_{max}(AllComb_{(C_r)})$
10:         Filter $NbF_{min}(DEP_{max}(AllComb_{(C_r)}))$
11:     end for
12:     for each $T_{RDD_{(i)}}$ output do
13:         $Reduct_m = \bigcup_{i=1}^{m} RED_{i_{(D)}}(C_r)$
14:     end for
15: end for
16: $Reduct = \bigcap_{n=1}^{N} Reduct_m$
17: return ($Reduct$)
```

In order to further guarantee the Sp-RST performance while avoiding any significant information loss, we apply the algorithm N times on the T_{RDD} m data blocks. More precisely, through all the iterations, the algorithm will first generate the m random $T_{RDD_{(i)}}$ as previously explained. Then, for each partition the distributed Sp-RST tasks, Algorithm 18 line 5 to 10, will be executed.

As seen in Algorithm 18, line 1 presenting the first Sp-RST task is executed before the iteration loop. This is because this task deals with the calculation of the indiscernibility relation of the decision class $IND(D)$ and is independent from the m generated partitions as the result depends on the data items class and not on the features. After the iteration loop, line 12, the output of each partition is either a single reduct $RED_{i_{(D)}}(C_r)$ or a family of reducts $RED^F_{i_{(D)}}(C_r)$.

Based on the RST preliminaries, any reduct of $RED^F_{i_{(D)}}(C_r)$ can be used to represent the $T_{RDD_{(i)}}$ information table. Consequently, if Sp-RST generates only one reduct, for a specific $T_{RDD_{(i)}}$ block, then the output of this feature selection phase is the set of the $RED_{i_{(D)}}(C_r)$ features. These features reflect the most informative ones among the C_r attributes resulting a new reduced $T_{RDD_{(i)}}$, $T_{RDD_{(i)}}(RED)$, which preserves nearly the same data quality as its corresponding $T_{RDD_{(i)}}(C_r)$ that is based on the whole feature set C_r.

On the other hand, if Sp-RST generates a family of reducts then the algorithm randomly selects one reduct among $RED^F_{i_{(D)}}(C_r)$ to represent the corresponding $T_{RDD_{(i)}}$. This random choice is justified by the same priority of all the reducts in $RED^F_{i_{(D)}}(C_r)$. At this stage, each i data block has its output $RED_{i_{(D)}}(C_r)$ referring to the selected features. However, since each $T_{RDD_{(i)}}$ is based on distinct features and with respect to $T_{RDD} = \bigcup_{i=1}^{m}(C_r)T_{RDD_{(i)}}$ a union of the generated selected features is required to represent the initial T_{RDD}; Algorithm 18, line 12 to 14. As previously mentioned, the process of applying Sp-RST will iterate N times generating N $Reduct_m$. Thus, at the end an intersection of all the obtained $Reduct_m$ is needed. By removing irrelevant and redundant features, Sp-RST can reduce the dimensionality of the data from $T_{RDD}(C)$ to $T_{RDD}(Reduct)$.

6.3.2 Distributed Algorithmic Sp-RST Details

The algorithm goes through 7 main jobs in order to generate the final sought $Reduct$. First, Sp-RST has to compute the indiscernibility relation for the decision class $D = \{d_1, \ldots, d_W\}$; defined as $IND(D)$: $IND(d_i)$. More precisely, Sp-RST will calculate the indiscernibility relation for every decision class d_i by gathering the same T_{RDD} data items, which are defined in the universe $U = \{x_1, \ldots, x_N\}$ and which belong to the same class d_i. To do so, Sp-RST processes a $aggregateByKey$ operation where the decision label d_i defines the key and the T_{RDD} data items identifiers id_i of x_i, define the values (see Algorithm 19).

Once the $IND(D)$ is calculated and within a specific partition, Sp-RST creates all the possible combinations of the C_r set of feature; $AllComb_{(C_r)}$. The third Sp-RST job deals with the indiscernibility relation computation for every previously generated combination. As presented in Algorithm 20, Sp-RST aims at grouping all the data items identifiers id_i sharing the same specific combination of features extracted from $AllComb_{(C_r)}$. In order to achieve this, we use the $aggregateByKey$ spark operation where the combination of features defines the key and the id_i as

Algorithm 19 Calculate $IND(D)$

```
    Input: $T_{RDD}$
    Output: $IND(D) : Array[Array[x_i]]$
1:  $IND(d_i) = data.map\{case(id_i, vector, d_i) => (d_i, id_i)\}$
    $.aggregateByKey(ArrayBuffer.empty[Long])(adde, merge)$
2:  $IND(d_i).map\{case(d_i, x_i) => x_i\}.collect$
```

value.

Algorithm 20 Calculate $IND(AllComb_{(C_r)})$

```
    Inputs: $T_{RDD_i}$, $AllComb_{(C_r)}$
    Output: $IND(AllComb_{(C_r)}) : Array[Array[id_i]]$
1:  $IND(AllComb_{(C_r)}) = data.map\ \{case(id_i, vector, d_i) => ((AllComb_{(C_r)_i}, vector),$
    $id_i)\}.aggregateByKey(ArrayBuffer.\ empty[Long])(adder, merger)$
2:  $IND(AllComb_{(C_r)}).map\{case(ListValues, id_i) => id_i\}.collect$
```

Then, the dependency degrees $\gamma(C_r, AllComb_{(C_r)})$ of each feature combination are computed. To do so, the calculated indiscernibility relations $IND(D)$ and $IND(AllComb_{(C_r)})$ as well as the set of all feature combinations $AllComb_{(C_r)}$ are required. The task is to first test if the intersection of every $IND(d_i)$ with each $IND(AllComb_{(C_r)})$ keeps all the latter elements; referring to the lower approximation. If so then a score which is equal to the length of $IND(AllComb_{(C_r)})$ is given, zero otherwise.

As this process is made in a distributed way where each machine is dealing with some feature combinations, a first sum operation of the $IND(d_i)$ scores is operated followed by a second sum operation to record all the $IND(D)$ scores; referring to the dependency degrees $\gamma(C_r, AllComb_{(C_r)})$. The output of this step is the set of dependency degrees $\gamma(C_r, AllComb_{(C_r)})$ of the feature combinations $AllComb_{(C_r)}$ and their associated sizes $Size_{(AllComb_{(C_r)})}$. At this stage, Sp-RST looks for the maximum dependency value among all $\gamma(C_r, AllComb_{(C_r)})$ using the max function operated on the given RDD. The output $MaxDependency$ reflects in one hand the dependency of the whole feature set (C_r) representing the T_{RDD_i} and on the other hand the dependency of all the possible feature combinations satisfying the constraint $\gamma(C_r, AllComb_{(C_r)}) = \gamma(C_r)$. $MaxDependency$ is the baseline value for feature selection.

Once the $MaxDependency$ is generated, Sp-RST keeps the set of all combinations having the same dependency degrees as $MaxDependency$; $\gamma(C_r, AllComb_{(C_r)}) = MaxDependency$. This is achieved by applying a filter function. In fact, at this stage Sp-RST removes in each computation level the unnecessary features that may affect negatively the performance of any learning algorithm.

Finally and based on the output of the previous step, Sp-RST keeps the set of combinations having the minimum number of features, $Size_{(AllComb_{(C_r)})}$, by applying a filter operation and by satisfying the full reduct constraints discussed in Section 6.2; $\gamma(C_r, AllComb_{(C_r)}) = \gamma(C_r)$ while there is no $AllComb'_{(C_r)} \subset AllComb_{(C_r)}$ such that $\gamma(C_r, AllComb'_{(C_r)}) = \gamma(C_r, AllComb_{(C_r)})$. Each combination satisfying this condition is considered as a viable minimum reduct set. The attributes of the reduct set describe all concepts in the original training dataset T_{RDD_i}.

6.4 Version based on distributed system and features partitioning

To perform feature selection in the context of big data, Sp-RST partitions the feature search space in a random way where each partition holds a random set of features. Each partition is dealt with separately in the distributed environment that at the end of the feature selection process all the selected features from each partition are gathered together to generate the final reduced set of features. Eventually, in such implementation design, it is very likely that similar features will belong to different partitions and hence a cut in data dependency will occur. It is very important to highlight that data dependency is a key issue in a distributed environment and in parallel computing. Nevertheless, based on the Sp-RST architecture, data dependency will not be assured as the algorithm uses an arbitrary procedure in partitioning the feature search space.

The second proposed solution, dubbed LSH-dRST, uses LSH presented in chapter 3, Section 3.2.2, which maps similar data instances based on their feature values into the same bucket in low dimensional cases as clustering, and based on this process, LSH-dRST uses the generated buckets to partition the feature search space in a more reliable way and hence guaranteeing data dependency and a lower computational cost.

6.4.1 General Model Formalization

Technically, to deal with high dimensional data sets and to make use of the LSH technique, within a distributed environment, we first generate the appropriate buckets based on LSH, map them into partitions, partition the entire rough set feature selection process into elementary tasks, each executed independently on each generated bucket, and then conquer the intermediate results to finally acquire the ultimate output; the reduct set.

As in the previous approach, we may formalize the latter as a given information table defined as T_{RDD}, where universe $U = \{x_1, \ldots, x_N\}$ is the set of data items, the conditional attribute set $C = \{c_1, \ldots, c_V\}$ contains every single feature of the T_{RDD} information table and the decision attribute D of our learning problem corresponds to the class (label) of each T_{RDD} sample and is defined as $D = \{d_1, \ldots, d_W\}$. The conditional attribute set C presents the pool from where the most convenient features will be selected.

In order to make our algorithm scalable with the high number of features and with respect to data dependency, we partition the given T_{RDD} information table into B data blocks based on the B generated LSH buckets. The buckets are splits from the conditional attribute set C and each bucket covers a specific feature space enclosing all similar and close instances based on their feature values.

Hence, $T_{RDD} = \bigcup_{b=1}^{B} (C_h) T_{RDD_{(b)}}$; where $h \in \{1, \ldots, V\}$. h is a value generated by LSH referring to the number of features per bucket that will be considered to

Algorithm 21 LSH-dRST

```
    Inputs: $T_{RDD}$: information table with $D$ as decision class, $K$: number of nearest neighbors, $B$: number of buckets
    Output: $Reduct$
 1: Generate the $B$ LSH buckets
 2: Calculate $IND(D)$
 3: for each $T_{RDD_{(b)}}$ where $b \in [1, \ldots, B]$ do
 4:     Generate the set $S$ sub-information tables $Cl$ based on $K$
 5:     for each $Cl_s(K)$ where $s \in [1, \ldots, S]$ do
 6:         Generate $AllComb_{(K)}$, Calculate $IND(AllComb_{(K)})$
 7:         Calculate $DEP(AllComb_{(K)})$, Select $DEP_{max}(AllComb_{(K)})$
 8:         Filter $DEP_{max}(AllComb_{(K)})$, Filter $NbF_{min}(DEP_{max}(AllComb_{(K)}))$
 9:     end for
10: end for
11: for each $T_{RDD_{(b)}}$ do
12:     for each $Cl_s(K)$ output do
13:         $Reduct = \bigcup_{b=1}^{B} \bigcup_{s=1}^{S} RED_s$
14:     end for
15: end for
16: return $(Reduct)$
```

create each $T_{RDD_{(b)}}$ data block; and is equal to the size of the feature space C divided by B.

Once the buckets are defined, each $T_{RDD_{(b)}}$ is divided into S automatically created sub-information tables Cl based on the K nearest neighbors approach; where K refers to the number of features per sub-information table and on which LSH-dRST will be applied. Hence, $T_{RDD_{(b)}} = \bigcup_{s=1}^{S} Cl_s(K)$; where $S = C_h/K$.

To ensure scalability, rather than applying LSH-dRST to T_{RDD} including the whole conditional feature set C the distributed algorithm will be applied to every single $Cl_s(K)$, where $s \in \{1, \ldots, S\}$, that at the end all the intermediate results will be gathered from the different Cl sub-information tables of every $T_{RDD_{(b)}}$ partition. In such a way, we can guarantee that LSH-dRST can be applied to a computable number of features while preserving data dependency and hence solving the standard distributed RST limitations. Algorithm 21 highlights the pseudo-code of our proposed LSH-dRST solution.

More precisely, the algorithm will first generate the B partitions using LSH, $T_{RDD_{(b)}}$ while preserving data dependency as previously highlighted. Then, for each partition, the Cl sub-information table will be created in a way that the K nearest neighbors from any data point within the $T_{RDD_{(b)}}$ feature search space form a sub-information table $Cl_s(K)$ (Algorithm 21, line 4). Through all the S Cl sub-information tables, the distributed LSH-dRST tasks, line 5 to 9, will be executed.

As seen in Algorithm 21, line 2 is executed out of the $T_{RDD_{(b)}}$ and the $Cl_s(K)$ iteration loops. The main reason for this implementation is that this task deals with the calculation of the indiscernibility relation of the decision class $IND(D)$. This task is independent from the B generated partitions as the result depends on the data items class and not on the features. Out from the iteration loops, line 10, the output of each $Cl_s(K)$ is either a single reduct $RED_{s_{(D)}}(K)$ or a family of reducts $RED^F_{s_{(D)}}(K)$. Based on the RST preliminaries previously mentioned in Section 6.2, any reduct of $RED^F_{s_{(D)}}(K)$ can be used to represent the $Cl_s(K)$ sub-information table.

Consequently, if LSH-dRST generates only one reduct, for a specific $Cl_s(K)$ sub-information table, then the output of this feature selection phase is the set of the $RED_{s_{(D)}}(K)$ features. These features reflect the most informative ones among the K attributes of $Cl_s(K)$ resulting a new reduced $Cl_s(K)$, $Cl_s(RED)$, which preserves nearly the same data quality as its corresponding $Cl_s(K)$ which is based on the whole feature set K.

On the other hand, if LSH-dRST generates a family of reducts then the algorithm randomly selects one reduct among $RED^F_{s_{(D)}}(K)$ to represent the corresponding $Cl_s(K)$. This random choice is argued by the same priority of all the reducts in $RED^F_{s_{(D)}}(K)$. In other words, any reduct included in $RED^F_{s_{(D)}}(K)$ can be used to replace the K attributes of $Cl_s(K)$. At this stage, each Cl_s sub-information table has its output $RED_{s_{(D)}}(K)$ referring to the selected features. However, since each Cl_s is based on distinct features within different $T_{RDD_{(b)}}$ feature search spaces and with respect to $T_{RDD_{(b)}} = \bigcup_{s=1}^{S} Cl_s(K)$ a union of the generated selected features is required to represent the initial T_{RDD}; defined as $Reduct = \bigcup_{b=1}^{B} \bigcup_{s=1}^{S} RED_s$ (Algorithm 21, line 11 to 15).

By removing irrelevant and redundant features, LSH-dRST can reduce the dimensionality of the data from $T_{RDD}(C)$ to $T_{RDD}(Reduct)$. In what follows, we will elucidate the different LSH-dRST elementary distributed tasks.

6.4.2 Distributed Algorithmic LSH-dRST Details

As previously highlighted, the elementary feature selection LSH-dRST distributed tasks will be executed on every $Cl_s(K)$ sub-information table defined by its K features along the $T_{RDD_{(b)}}$ partitions; except for task 2 in Algorithm 21 dealing with $IND(D)$. The algorithm goes through 10 main jobs in order to generate the final sought $Reduct$.

The first step, is to apply LSH to generate the B buckets based on a hash table. To do so, LSH-dRST creates the hash table based on a set of random vectors following a Gaussian distribution (referred as the $\mathcal{H}$ family of hash functions). The constructed hash table is based on the size of the features of T_{RDD}. After that, the algorithm maps the T_{RDD} to work on each partition separately and on each partition it applies a projection for each vector based on the set of the T_{RDD} mapped feature vectors. As a result the buckets are automatically created; each with a specific index (referred as a hash code). Finally, LSH-dRST performs a sort action to order the buckets with respect to the given number of buckets B. The pseudo-code of this distributed task is given in Algorithm 22.

After that, LSH-dRST has to compute the indiscernibility relation for the decision class $D = \{d_1, \ldots, d_W\}$; defined as $IND(D)$: $IND(d_i)$. More precisely, LSH-dRST will calculate the indiscernibility relation for every decision class d_i by gathering the same T_{RDD} data items which are defined in the universe $U = \{x_1, \ldots, x_N\}$ and which belong to the same class d_i. To do so, LSH-dRST

Algorithm 22 Generate $Buckets(B)$

```
Inputs: $T_{RDD}$, $B$
Output: $T_{RDD_{(b)}}$
1: Generate the hash table based on the $T_{RDD}$ feature size
2: Map the $T_{RDD}$
3: Apply the LSH projection for each vector
4: Sort the buckets by $B$
```

processes a $aggregateByKey$[5] operation where the decision label d_i defines the key and the T_{RDD} data items identifiers id_i of x_i, define the values. The set of gathered data items is only kept as it represents $IND(D)$: $IND(d_i)$. The pseudo-code related to this distributed job is highlighted in Algorithm 19.

At its third step, LSH-dRST has to generate the set S of the $Cl(K)$ sub-information tables based on the number of features K. Let us recall at this stage that LSH gathered all the similar features already within a same specific bucket. On these similar attributes, a further partitioning is required to generate the sub-information tables that can be handled by the LSH-dRST algorithm.

As mentioned in Section 6.2, the standard rough set theory has to generate all the combinations of features at once, process them in turn to finally generate the reduct. As it is infeasible to generate all the combinations of features within the big data context, then the distributed LSH-dRST will operate on the sub-information tables constructed on K number of features; where K is a manageable size that can be handled by the algorithm. Therefore and to achieve this distributed task, for every bucket $T_{RDD_{(b)}}$, LSH-dRST performs a $mapPartitionsWithIndex$[6] operation using the buckets indexes; the already generated hash codes in Algorithm 22. Then, the later result is mapped, where on each partition, the K nearest features to a randomly chosen attribute within the same $T_{RDD_{(b)}}$ hash code, are selected to form a sub-information table. The pseudo-code related to this distributed job is highlighted in Algorithm 23.

Algorithm 23 Generate $Cl(K)$

```
Inputs: $T_{RDD_{(b)}}$, $K$
Outputs: $S$, $Cl(K)$
1: Perform a $mapPartitionsWithIndex$ on every $T_{RDD_{(b)}}$ using its index
2: Map the result of step (1)
   Perform a KNN by looking for the K nearest features within a randomly selected
   attribute within each $T_{RDD_{(b)}}$
3: Generate the set $S$ of the $Cl(K)$ sub-information tables
```

Once the set S of the $Cl(K)$ sub-information tables is generated for all the $T_{RDD_{(b)}}$, LSH-dRST has to perform feature selection for each single $Cl_s(K)$. To do

[5]https://spark.apache.org/docs/0.7.2/api/core/spark/PairRDDFunctions.html

[6]https://spark.apache.org/docs/1.6.2/api/java/org/apache/spark/rdd/RDD.html#mapPartitionsWithIndex(scala.Function2,%20boolean,%20scala.reflect.ClassTag)

so, first, the algorithm has to create all the possible combinations of the K set of features, $AllComb_{(K)}$, using the $flatMap$[7] spark function as shown in Algorithm 24.

Algorithm 24 Generate $AllComb_{(K)}$

```
    Input: $K$
    Output: $AllComb_{(K)}$
 1: $AllComb_{(K)} = K.flatMap(K.combinations)\ .drop(1)$
```

Then, the fifth LSH-dRST job deals with the indiscernibility relation computation for every previously generated combination. As presented in Algorithm 25, LSH-dRST aims at grouping all the data items identifiers id_i sharing the same specific combination of features extracted from $AllComb_{(K)}$. In order to achieve this, we use the $aggregateByKey$ spark operation where the combination of features defines the key and the id_i mapped as value.

Algorithm 25 Calculate $IND(AllComb_{(K)})$

```
    Inputs: $Cl_s(K)$, $AllComb_{(K)}$
    Output: $IND(AllComb_{(K)}) : Array[Array[id_i]]$
 1: $IND(AllComb_{(K)})$ =
    Map the $Cl_s(K)$ sub-information table
    Perfom a $aggregateByKey$ operation based on $\langle AllComb_{(K)}, id_i \rangle$
 2: Map the result of step (1) to keep $id_i$ as follows:
    $IND(AllComb_{(K)}).map\{case(ListValues, id_i) => id_i\}.collect$
```

At this phase, LSH-dRST prepares the set of features that will be selected in the coming steps. In Algorithm 26, the dependency degrees $\gamma(K, AllComb_{(K)})$ of each feature combination are computed. The calculated indiscernibility relations $IND(D)$ and $IND(AllComb_{(K)})$ as well as the set of all feature combinations $AllComb_{(K)}$ are required. The task is to test, first, if the intersection of every $IND(d_i)$ with each $IND(AllComb_{(K)})$ keeps all the latter elements; referring to the lower approximation. If so then a score which is equal to the length of $IND(AllComb_{(K)})$ is given, zero otherwise. As this process is made in a distributed way where each machine is dealing with some feature combinations, a first sum operation of the $IND(d_i)$ scores is operated followed by a second sum operation to record all the $IND(D)$ scores; referring to the dependency degrees $\gamma(K, AllComb_{(K)})$.

The output of this step is the set of dependency degrees $\gamma(K, AllComb_{(K)})$ of the feature combinations $AllComb_{(K)}$ and their associated sizes $Size_{(AllComb_{(K)})}$. Then, LSH-dRST looks for the maximum dependency value $DEP_{max}(AllComb_{(K)})$ among all $\gamma(K, AllComb_{(K)})$ using the max function operated on the given RDD. The output $MaxDependency$ reflects in one hand the dependency of the whole feature set K representing the $Cl_s(K)$ and on the other hand the dependency of all the possible feature combinations satisfying the constraint $\gamma(K, AllComb_{(K)}) = \gamma(K)$. $MaxDependency$ is the baseline value for feature selection.

Once the $MaxDependency$ is generated, LSH-dRST keeps the set of all combinations having the same dependency degrees as $MaxDependency$; i.e.,

[7]https://spark.apache.org/docs/latest/rdd-programming-guide.html

Algorithm 26 Generate $DEP(AllComb_{(K)})$

```
    Inputs: $AllComb_{(K)}$, $IND(D)$, $IND(AllComb_{(K)})$
    Outputs: $\gamma(K, AllComb_{(K)})$, $Size_{(AllComb_{(K)})}$
 1: for i := $AllComb_{(K)}$ do
 2:     for j := $IND(D)$ do
 3:         for v := $IND(AllComb_{(K)})$ do
 4:             if j.intersect(v).length == v.length then v.length
 5:             else 0
 6:         Reduce(_ + _)
 7:     Reduce(_ + _)
```

Algorithm 27 Select $DEP_{max}(AllComb_{(K)})$

```
    Input: RDD[$AllComb_{(K)}$, $Size_{(AllComb_{(K)})}$, $\gamma(K, AllComb_{(K)})$]
    Output: $MaxDependency$
 1: $RDD.max()(Ordering[Int].on(\_.\_3)).\_3$
```

$\gamma(K, AllComb_{(K)}) = MaxDependency$. This is achieved by applying a $filter$ function. In fact, at this stage LSH-dRST removes in each computation level the unnecessary features that may affect negatively the performance of any learning algorithm.

Algorithm 28 Filter $DEP_{max}(AllComb_{(K)})$

```
    Inputs: RDD[$AllComb_{(K)}$, $Size_{(AllComb_{(K)})}$, $\gamma(K, AllComb_{(K)})$], $MaxDependency$
    Output: Filtered-RDD[$AllComb_{(K)}$, $Size_{(AllComb_{(K)})}$, $\gamma(K, AllComb_{(K)})$]
 1: $RDD.filter(\_.\_3 == maxDependancy)$
```

Finally and based on the output of the previous step, LSH-dRST keeps the set of combinations having the minimum number of features, $Size_{(AllComb_{(K)})}$, by applying a $filter$ operation ($NbF_{min}(DEP_{max}(AllComb_{(K)}))$) and by satisfying the full reduct constraints discussed in Section 6.2; $\gamma(K, AllComb_{(K)}) = \gamma(K)$ while there is no $AllComb'_{(K)} \subset AllComb_{(K)}$ such that $\gamma(K, AllComb'_{(K)}) = \gamma(K, AllComb_{(K)})$.

Each combination satisfying this condition is considered as a viable minimum reduct set. The attributes of the reduct set describe all concepts in the sub-information table $Cl_s(K)$.

6.5 Experiments

6.5.1 Benchmark

To demonstrate the effectiveness of our proposed approach we chose the Amazon Commerce reviews data set from the UCI machine learning repository [68] as it was the dataset with the largest number of features that still had a sufficiently large number of data items. This data set was derived from customer reviews on the Amazon commerce website by identifying a set of most active users and with the goal to perform authorship identification. The database includes 1 500

Algorithm 29 Filter $NbF_{min}(DEP_{max}(AllComb_{(K)}))$

```
    Input: RDD[$AllComb_{(K)}$, $Size_{(AllComb_{(K)})}$, $\gamma(K, AllComb_{(K)})$]
    Output: Filtered-RDD[$AllComb_{(K)}$, $Size_{(AllComb_{(K)})}$, $\gamma(K, AllComb_{(K)})$]
 1: $minNbF := RDD.min()(Ordering[Int].on(\_.\_2)).\_2$
 2: $RDD.filter(\_.\_2 == minNbF)$
```

data items described through 10 000 features (linguistic style such punctuation, length of words, sentences, etc.) and 50 distinct classes (authors). Instances are identically distributed across the different classes, i. e., for each class there are 30 items.

The analysis focuses on the scalability of the algorithm that allows it to solve the standard rough set feature selection inadequacy to be applied to Big Data. To do so, we will evaluate the performance using the *speedup*, *sizeup* and *scaleup* criteria introduced in [38]. For the evaluation of Random Forest as classifier, we use the standard set based performance measures which are the the classification error and measures presented in appendix.

Speedup Analysis We keep the size of the data set constant (where size is measured by the number of features, i.e., 10 000 features in our case) and increase the number of nodes. The speedup of a system with m nodes:

$$\text{Speedup(m)} = \frac{\text{runtime on one node}}{\text{runtime on } m \text{ nodes}}$$

Scaleup Analysis The scaleup evaluates the ability to increase the number of nodes and the size of the data set simultaneously:

$$\text{Scaleup(m)} = \frac{\text{runtime for data set of size } s \text{ on 1 node}}{\text{runtime for data set of size } s \cdot m \text{ on } m \text{ nodes}}$$

Sizeup Analysis

The sizeup keeps the number of nodes constant and measures how much the execution time increases as the data set is increased by a factor of m:

$$\text{Sizeup(m)} = \frac{\text{runtime for data set of size } m \cdot s}{\text{runtime for baseline data set of size } s}$$

6.5.2 Sp-RST analysis

The main aim of our experimentation is to demonstrate how our proposed approach Sp-RST speeds up the execution time for large data sets without introducing too much information loss. We investigate different parameters of Sp-RST and analyze how these affect execution time and stability of the feature selection. We then show that the improvement in performance does not decrease the feature selection ability by using a Random Forest classifier on the original dataset and the reduced datasets produced by Sp-RST.

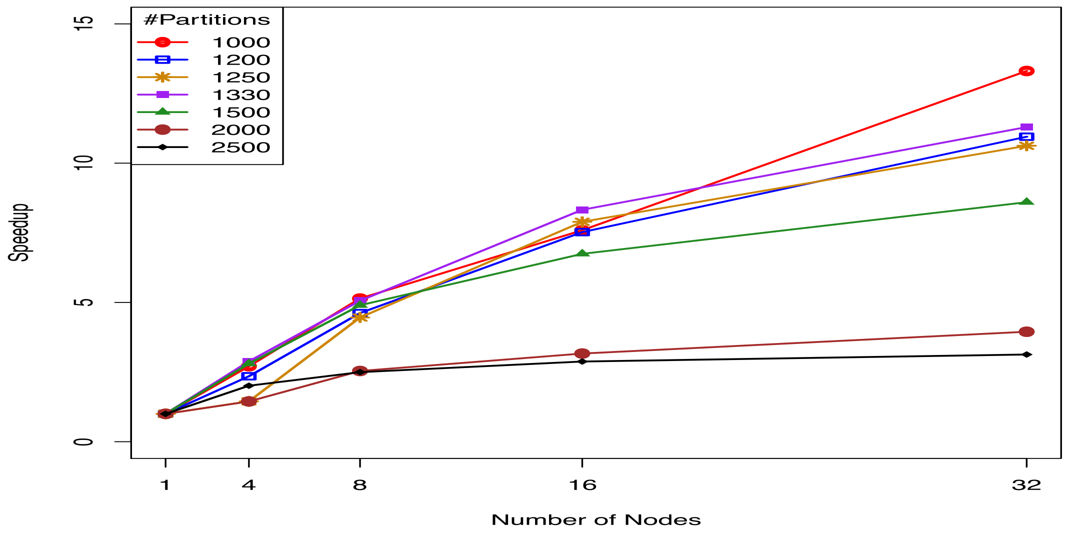


Figure 6.1: Speedup for different numbers of nodes and partitions.

We use the Random Forest implementation provided in the Mlib/Spark framework, with the following parameters: maxDepth=6, numTrees=300, featureSubsetStrategy='all' and impurity='gini'. The algorithm automatically identifies categorical features and indexes them. Preliminary results revealed that a maximum of 10 features per partition is the limit that can be processed by Sp-RST. We therefore perform experiments for 1000, 1200, 1250, 1330, 1500, 2000 and 2500 partitions in Algorithm 18.

We run all settings on 1, 4, 8, 16, and 32 nodes on Grid5000. For the purpose of this study we set the number of iterations in Algorithm 18 to 10 (based on preliminary experiments). We will evaluate the performance of Sp-RST using the *speedup* and *sizeup* criteria.

Speedup

We first consider the speedup of Sp-RST. We keep the size of the dataset constant (where size is measured by the number of features in each partition) and increase the number of nodes. We plot the average time needed to run a single iteration within Algorithm 18 (over the 10 iterations executed) and the respective speedups in Figures 6.1 and 6.2, respectively.

An ideal parallel algorithm has linear speedup, which is, however, difficult to achieve in practice due to communication cost and the fact that the slowest slave dominates the total execution time. From Figure 6.1 we see that our method has a good speedup for settings with fewer partitions. The more the size of the database, i. e., the number of attributes per partition, increases, the closer the speedup gets to linear. This can be explained by the fact that fewer partitions imply that each partition has more features. As discussed previously the execution time grows exponentially in the number of features and thus, using more nodes is more beneficial in cases with many features. We obtain good speedup if the number of features per partition is between 7 and 10 (1000 to 1330 partitions), but for 4 or 5 features (2000 and 2500 partitions, respectively) the speedup quickly stagnates. This observation is also supported by the execution times (Figure 6.2).

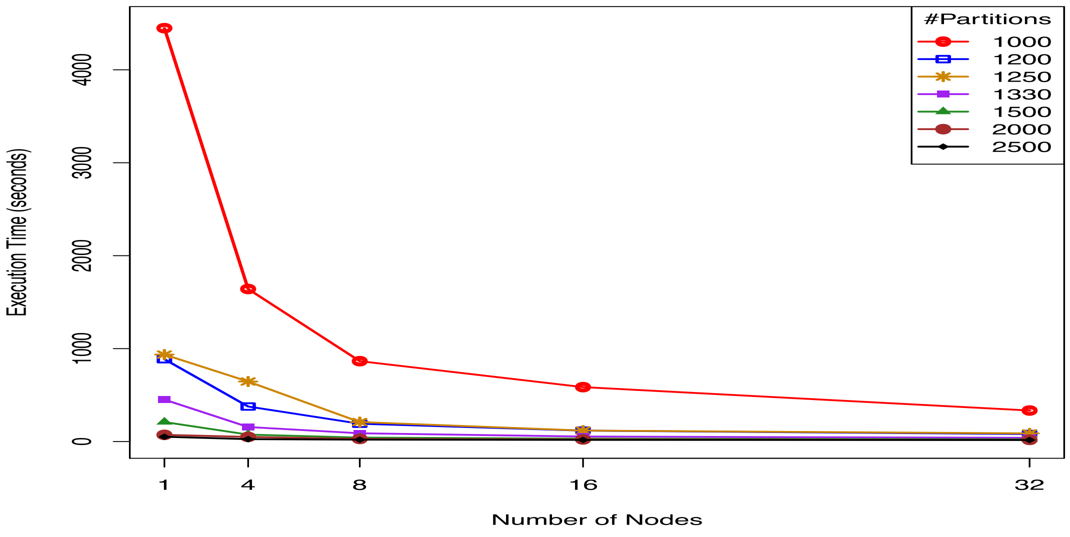


Figure 6.2: Average execution times for a single iteration of Algorithm 18 and for different numbers of nodes and partitions.

For few partitions the execution time quickly decreases with increasing number of nodes while for many partitions we observe hardly any improvement.

Sizeup

We use Sp-RST with 2500 partitions as baseline as this is the smallest dataset in our experiments and calculate the m values for the other settings based on the number of features per partition. We then plot the sizeup in Figure 6.3. We see that the sizeup of Sp-RST grows very quickly as m increases, but gets better as the number of nodes increases. Recall that we define the size of the database as the number of features per partition. Thus, this behavior was expected due to the runtime properties of Sp-RST previously discussed. We conjecture that using the classic definition of size as the number of features in the whole database would yield a good sizeup, i. e., that our method is able to process large datasets efficiently while keeping the number of nodes constant and increasing the size of the data.

Together with the above discussion of the speedup, we see that there is some trade-off between the number of partitions and the number of nodes used. If only few nodes are available, it may be advisable to use a larger number of partitions to reduce execution times while the number of partitions becomes less important if a high degree of parallelization can be afforded. It should be noted that using a larger number of partitions has the potential to increase information loss during the data-preprocessing step. Our following experimentation and analysis shows that this, however, is not the case.

Classification Error With and Without Sp-RST

To validate the suitability of our method with respect to classification, we investigate the influence of Sp-RST on the classification error of the Random Forest classifier. We present results of 100 independent runs on the original dataset with 10 000

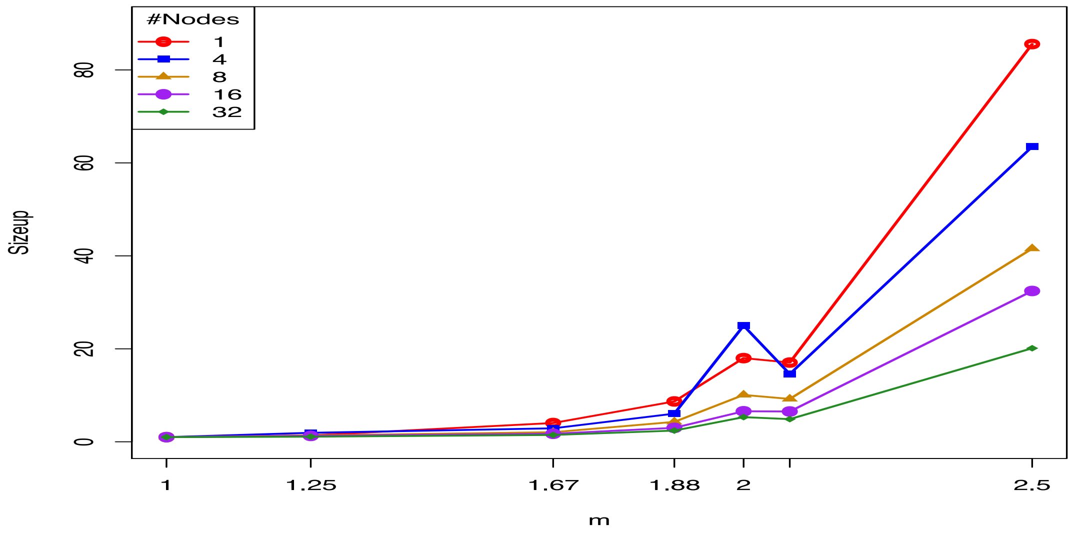


Figure 6.3: Sizeup for different numbers of nodes and values of m.

features and the datasets derived by Sp-RST with different numbers of partitions in Table 6.1. While the overall error seems high, it should be noted that the used database contains 50 distinct classes. Thus, a naive baseline classifier making random guesses would have a classification error of 98%, which is significantly higher than for our approach.

The median error of Random Forest on the original dataset is larger than the corresponding value for all Sp-RST settings. Considering averages, only 1000 partitions and 2000 partitions are slightly higher than Random Forest without Sp-RST. We perform statistical tests as described previously and find that the error difference by Sp-RST with 1330, 1500, and 2500 partitions is statistically significant at confidence level 0.05 while for the other settings no statistically significant difference could be found. We, therefore, conclude that Sp-RST introduces no significant information loss as results are at least comparable.

Partitions	1	1000	1200	1250	1330	1500	2000	2500
Average	**50.50%**	51.00%	**49.69%**	**49.25%**	**47.14%**	**42.18%**	51.44%	**45.16%**
Median	**52.53%**	**51.38%**	**50.77%**	**49.83%**	**46.59%**	**41.51%**	**51.51%**	**45.40%**
Standard deviation	11.09%	11.13%	9.98%	9.79%	11.77%	11.65%	10.71%	10.36%
p-value (comparison with ‘1’)	-	0.4897	0.1852	0.1945	**0.009471**	**7.221e-07**	0.6261	**0.000485**

Table 6.1: Average, median and standard deviation of the classification error over 100 independent runs of Random Forest depending on the number of partitions (rounded to 2 decimal places). The case of ‘1 partition’ corresponds to the original dataset without performing Sp-RST.

6.5.3 LSH-dRST analysis

The main aim of our experimentation is to demonstrate that our proposed approach LSH-dRST preserves data dependency within the same generated buckets and within the distributed environment. We will show that by using a more intelligent partitioning of the universe, via the use of LSH, a more reliable process of gathering similar data instances based on their feature values can be reached; and hence better classification results can be obtained. Indeed, we will show that LSH-dRST is not only scalable but also more reliable for feature selection; making it more relevant to big data pre-processing.

We, therefore, investigate different parameters of LSH-dRST and analyze how these affect execution time and stability of the feature selection; hence data dependency. We then show that the improvement in performance does not decrease the feature selection ability by using a Random Forest classifier on the original data set, the reduced data sets produced by LSH-dRST and some other feature selection techniques as described below. We use the scikit-learn Random Forest implementation[8] with the following parameters: n_estimators $= 1000$, n_jobs $= -1$, and oob_score = True. A Stratified 10-Folds cross-validator[9] is used for all our conducted experiments.

Based on the conducted experiments in Sp-RST Section, a maximum of 10 features per sub-information table Cl is used that can be processed by LSH-dRST. We therefore perform experiments for 2, 5, 10, 25 and 50 buckets (B), in Algorithm 21, each comprising sub-information tables of 4, 5, 8 and 10 features (F); where F refers to the K parameter in Algorithm 21. For instance, for bucket $(B = 2)$ and for a number of 4 features $(F = 4)$ per sub-information table the algorithm generates 1250 Cl. We run all settings on 1, 2, 4, 8, and 16 nodes on the Grid5000 testbed[10] which is a French national large-scale and versatile platform.

Our analysis first focuses on the scalability of the algorithm. We evaluate the performance of LSH-dRST using the *speedup*, *sizeup*, and *scaleup* criteria introduced in previous section and define the combined runtime for LSH and dRST as the execution time of our method.

We perform model evaluation using a Random Forest classifier to evaluate the quality of our feature selection and compare it with other common techniques, namely the Sum Squares Ratio as implemented in Smile[11] as well as Information Gain, Gain Ratio and Chi Squared as provided in Weka 3.6.15[12]. For the Sum Squares Ratio, we set the number of features to be selected to a value comparable with LSH-dRST, i.e., the average number of features selected for each parameter setting of F. All other methods were run with 'Ranker' as search method and a threshold of 0. We determine the sets of features selected by these methods and

[8]http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[9]http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html

[10]https://www.grid5000.fr/mediawiki/index.php/Grid5000:Home

[11]https://haifengl.github.io/smile/feature.html

[12]https://www.cs.waikato.ac.nz/~ml/weka: classes ChiSquaredAttributeEval, GainRatioAttributeEval, InfoGainAttributeEval

then perform 10 runs of the above Random Forest implementation on the new feature set. We use the standard measures which are the precision, the recall, the accuracy and the F1 score defined in appendix to report our results.

LSH-dRST makes use of randomisation in several places, e.g., LSH uses random projections, the construction of the sub-information tables starts with a randomly selected feature, and we select one reduct among the generated family of reducts randomly. For this reason, we always perform multiple runs of the algorithm and report appropriate statistics.

Speedup

We plot the speedup in Figure 6.4 and see that the speedup for most parameter settings is very similar. However, setting $F = 8$ improves the speedup considerably—independently of the setting for B (where $B = 5$ and $B = 10$ yield the best overall results). Overall, we conclude that the number of buckets does not have a significant influence on the speedup, but the number of sub-information tables F does. The latter is expected since the execution time grows exponentially in the number of features and thus, using more nodes is more beneficial in cases with many features.

It is therefore somewhat surprising that $F = 10$ does not exhibit a larger speedup. Note that an ideal parallel algorithm has linear speedup, which is, however, difficult to achieve in practice due to startup and communication cost as well as interference and skew.

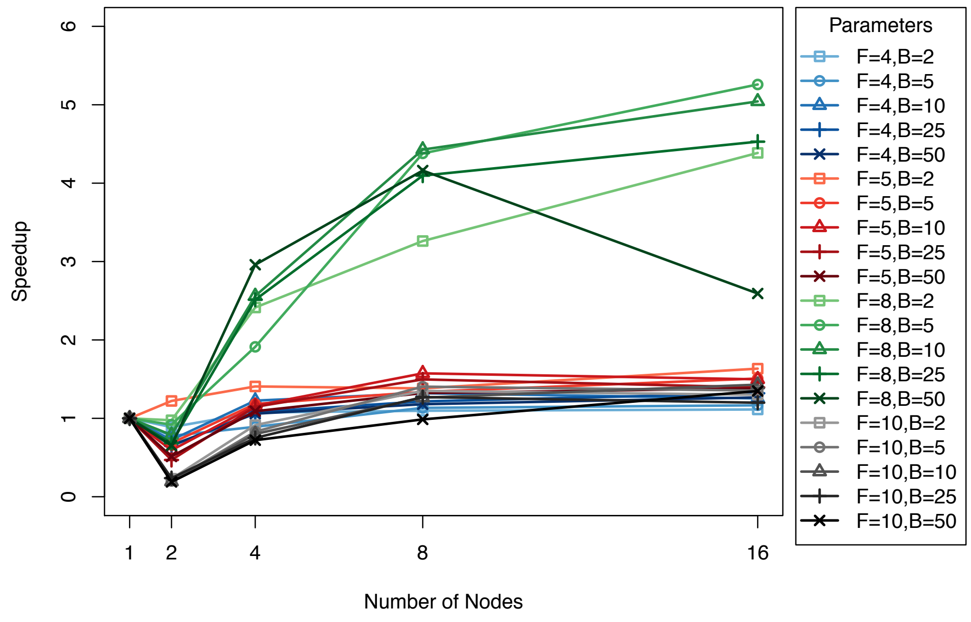


Figure 6.4: LSH-dRS. Speedup for Amazon with 10 000 features for 16 nodes.

Sizeup Analysis

To measure the sizeup we have created smaller databases by selecting random features from the original Amazon 10 000 database. We use 1 000 features as a

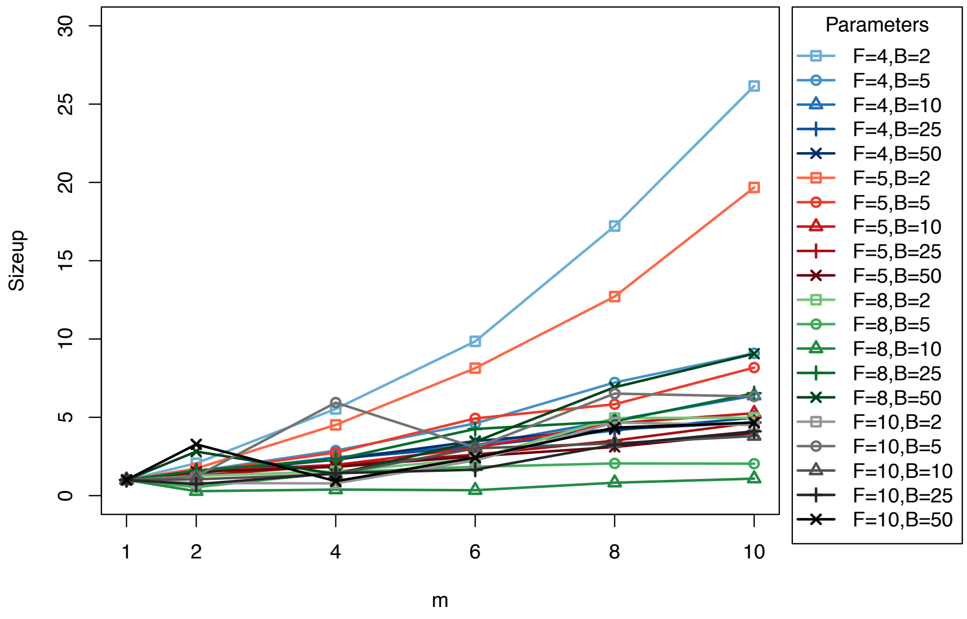


Figure 6.5: LSH-dRS. Sizeup for Amazon with 10 000 features for 16 nodes.

baseline and consider 2 000, 4 000, 6 000, 8 000, and 10 000 features, respectively. We plot the sizeup for 16 nodes (the largest number of nodes we consider) in Figure 6.5 and see that our method has sub-linear sizeup for most parameter settings, i. e., for a 10-times larger data set it requires less than 10 times more time.

The only two exceptions are $F = 4$ and $F = 5$ (i. e., small numbers of features) with only two buckets ($B = 2$). Looking closer into our results we can observe that for some parameter settings the LSH part of LSH-dRST is more time-consuming than the rough set part, but for others it is less time-consuming.

Scaleup Analysis

We use the sub-data sets previously described with 1 000 features as a baseline and plot the results in Figure 6.6. It should be noted that a scaleup of 1 implies linear scaleup, which similarly to linear speedup is difficult to achieve. Our scaleup is clearly smaller than 1 for all parameter settings, but fluctuates between 0.2 and 0.4 for most settings and 8 nodes, including the ones that exhibit the best speedup. The best scaleup is achieved for $F = 5$ and large values for B.

Comparison with Other Feature Selection Methods

We show the results of 10 runs of random forest on the different reduced data sets in Figures 6.7 and 6.8. We observe that LSH-dRST outperforms the Chi Squared, Information Gain and Gain Ratio selection methods and has comparable performance to the other methods. Moreover, we see that the classification result is quite stable with respect to the parameter settings in LSH-dRST. This is observed for all evaluation metrics, i.e., accuracy, recall, precision, and F1 score.

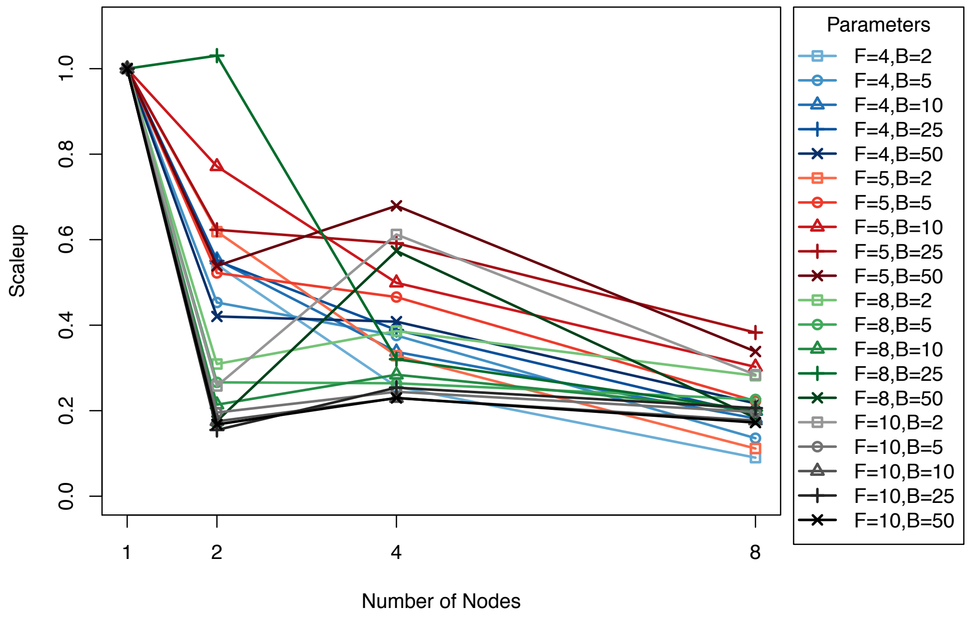


Figure 6.6: LSH-dRS. Scaleup.

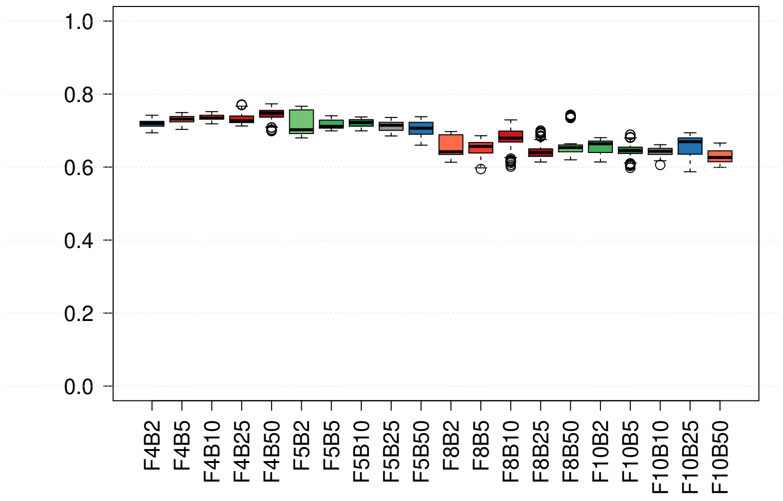
6.6 Conclusion

We have presented here two novel efficient distributed algorithms based on Rough Set Theory for large-scale data pre-processing under the Spark framework. To reduce the computational effort of the rough set computations, the first approach (Sp-RST) splits the given dataset into partitions with smaller numbers of features which are then processed in parallel. The second proposition LSH-dRST use locality sensitive hashing as clustering method to determine appropriate partitions of the feature set. This second approach improves (Sp-RST) method, which use random partitions.

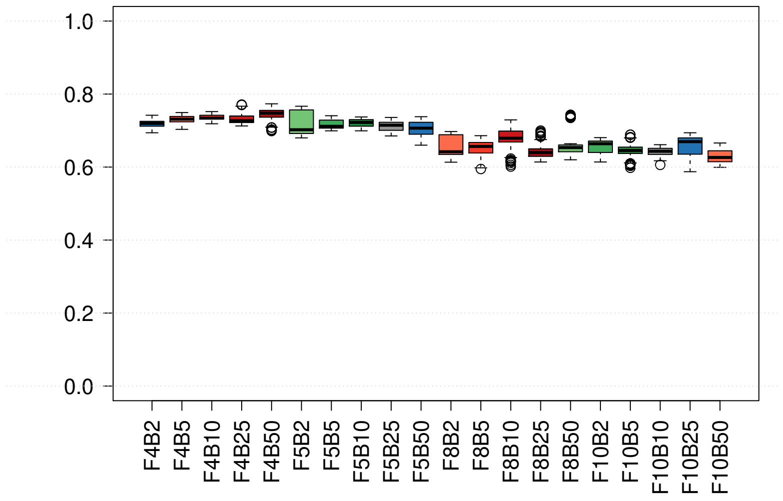
We have demonstrated its effectiveness using the Amazon Commerce reviews data set from the UCI machine learning repository, a dataset with 10 000 features and 1 500 data items equally spread over 50 classes. A detailed experimentation reveals that our proposed Sp-RST method achieves a good speedup, but in order to also achieve good sizeup a large number of partitions is necessary.

The experiments demonstrate also that LSH-dRST scales well in terms of three commonly used evaluation criteria: speedup, sizeup, and scaleup. We investigate different parameter settings and show that LSH-dRST is robust with respect to the number of buckets used in LSH while there is a clear trade-off between quality of the feature selection and speedup with respect to the second parameter (the number of features in each sub-information table). Configured appropriately the mean accuracy achieved by a random forest classifier on the reduced data sets is better than for the unreduced data set and comparable to the results obtained by Sp-RST, however, LSH-dRST exhibits a much smaller variance in the feature selection process and thus, is considered more reliable.

Based on the conducted experiments and results, we can clearly see the benefit and impact of using the locality sensitive hashing as clustering method in our proposed solution as it partitions the high dimensional feature search space in a more reliable and intelligent way and hence guaranteeing data dependency in the distributed environment, and ensuring a lower computational cost.

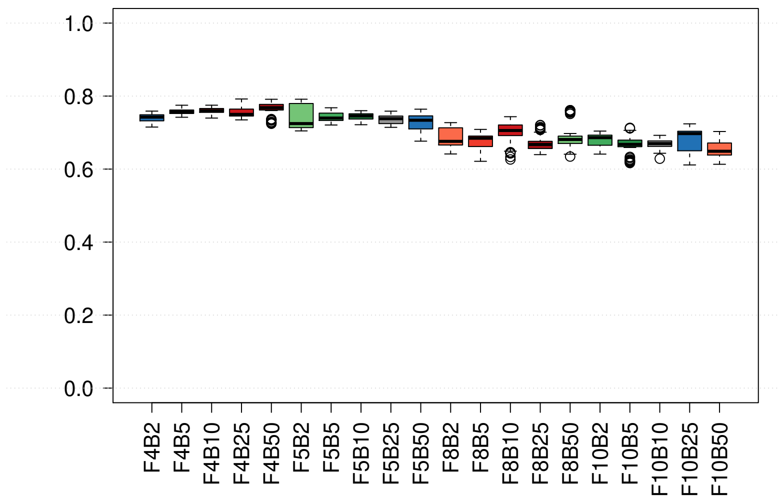


(a) Accuracy

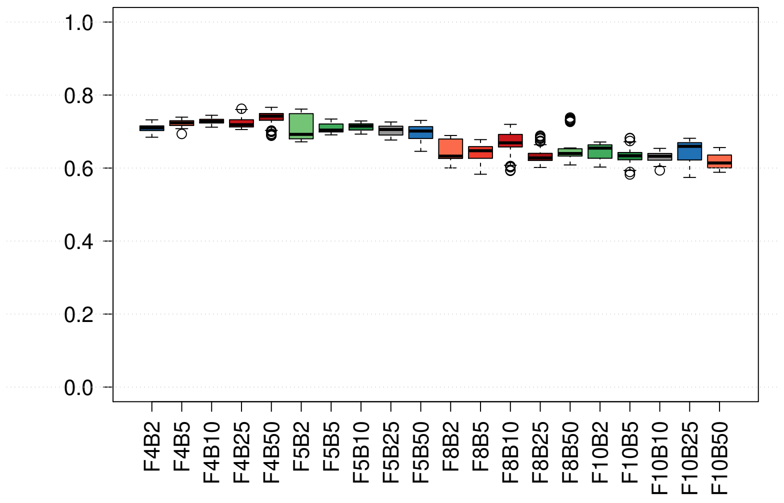


(b) Recall

Figure 6.7: The classification results for different parameters of LSH-dRST.(First part)

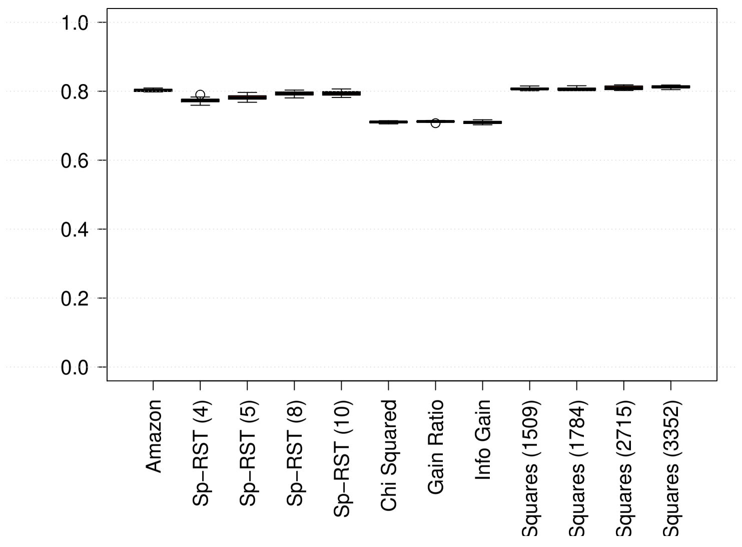


(a) Precision

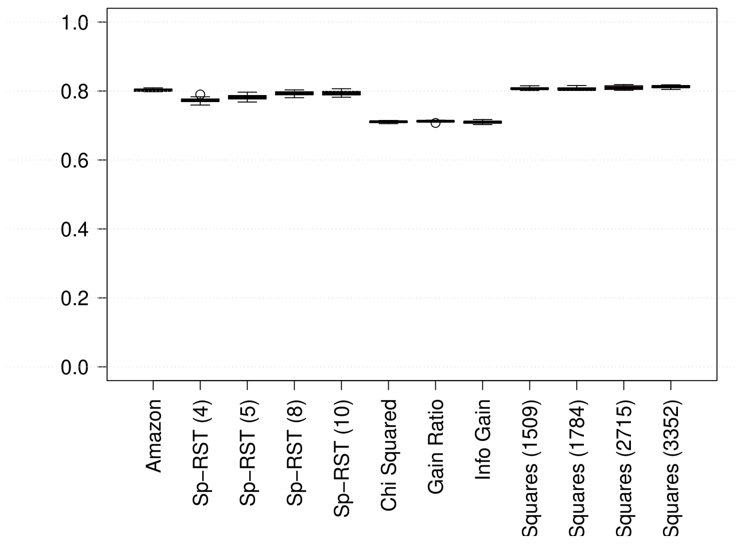


(b) F1 Score

Figure 6.7: The classification results for different parameters of LSH-dRST.(Second part)

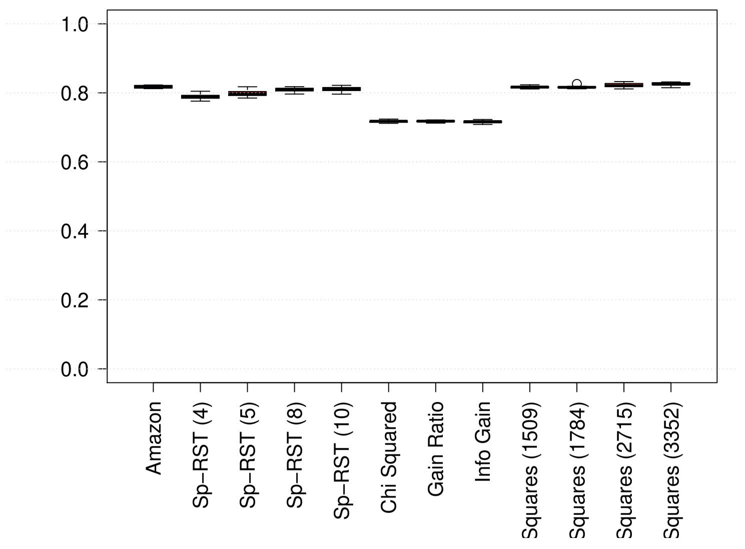


(a) Accuracy

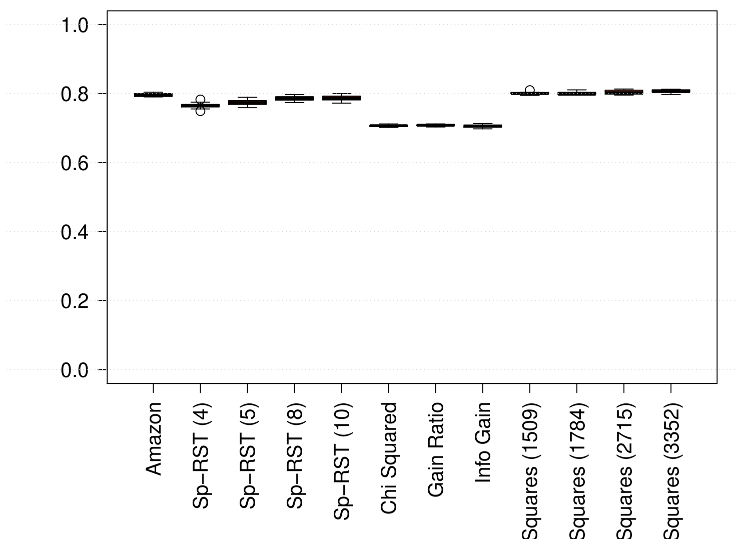


(b) Recall

Figure 6.8: The classification results for the original data set and other feature selection methods. (First part)



(a) Precision



(b) F1 Score

Figure 6.8: The classification results for the original data set and other feature selection methods. (Second part)

Part III

Applications

Chapter 7

Applications and Open Source

7.1 Kameleoon

Kameleoon is an A/B testing and personalization company which offers to its customers the ability to know which version of their websites performs the best giving specific goals as reach a gender dedicated section or click on the buy button. The objective of most websites is to propose the best navigating experience in order to maximise goals realization, which goes from coming back to the website to buy something passing from subscribing to the last newsletter.

In order to help with these issues, Kameleoon started to propose A/B Testing then personalization solutions. A/B Testing enables to test which version (A or B) of a web-page performs better for a specific objective, after having exposed a sufficient number of visitors to both A and B, the client knows which one performs best and can then implement this variation as the default version of their website. Personalization, on the other hand, proposes to offer a specific experience, such as a dedicated theme, to users sharing commons attributes. The idea is that if the users feel more comfortable with their navigating experience they will have greater chance of converting goals.

Describing visits

A web visitor can be discretized as a succession of elementary units that are called events. There exists a wide variety of events. Some contain static anonymous user data such as OS or browser information, others represent user interactions with the website such as clicks or scrolls. When zooming out, the concatenation of events of a specific visitor over a short time period will form the concept of visit.

Predicting future behaviours

A major objective of the company is to be able to predict, with sufficient accuracy and as early as possible, the probability that a visit will convert to a specific goal. This allows the client to trigger pre-defined strategies such as exposing the visitor to a specific personalization (an alternate version of a page for example), in order to increase the visit's conversion probability.

Observing past behaviours

Another interesting aspect is the ability to have a view at specific moments of distributions of visitor features. Clustering is the application which fits with this objective.

7.2 Data, Preprocessing and vectorization

At its lowest granularity level, information is composed of what we call "events". These events contain a variety of information, ranging from anonymous user navigation information, such as browser version, to user actions such as clicking the "purchase" button. The nature of the data is varied, containing such features as integers, scalar, booleans and categorical data. We are confronted, as often in the industry, with a mixed data space.

We have also an object called the *visit*, which is the aggregation of all events referring to a visitor from their arrival on the website until 30 minutes of inactivity. It is with this object that we will interact.

We have to transform our raw data, from the visit, into a suitable vector fitting with metric associate space. It is what we call the vectorization process. Most of the problem resides here, because it exists a combinatorial number of possibilities with the number of usable features. Moreover, depending on the clustering algorithms we use, only specific vector types can be employed.

Binary vector

We have essentially one hot encoded all categorical data. We also have partitionated numerical values, i.e. defining a category for various consecutive ranges of numerical data. For integer values such as number of pages seen and number of previous visits, we decide to employ an additive coding described in chapter 4.

Scalar vector

We have few continuous data, so we choose to use binary vectors and apply dimension reduction techniques such as PCA, and UMAP[128].

Visual interpretation

Again comparing different techniques depends on interpretations, despite that fact we propose few visualisations between PCA and UMAP reduction dimension techniques. Let precise that these two dimensions reduction techniques have been applied on one hot encoded datasets.

We have tried many feature combinations to obtain what in our opinion is the best result we had with the PCA on Figure 7.1, which is two parallel plane where red points are the ones which do not convert a specific goal and blue points are those that convert. As observed on Figure 7.1 two clusters have approximately the same ratio of converted points compared to non converted ones. Unfortunately this kind of data representation does not give any useful information because both clusters have an average converted rate close to the average one and even if a K-Means set with two clusters will give a good partitioning. It will have no exploitable value.

On the other side, results given by UMAP on Figure 7.2 are much more interesting. The Hamming distance has been used thanks to UMAP ability to use any metric. We can observe many elliptic clusters with diverse concentrations of converted points (in blue). A nice clustering application will be to isolate each cluster where some of them have a null converted rate when others have good ones. A K-Means could work but we need to look into the visualisation to set the right number of clusters. Moreover we should set the initialisation properly.

For this case the ε-proximity from chapter 3 works better by finding automatically the right number of clusters as exposed on Figure 7.3 with an ε set to 5% of the average distance between points (5% represent a good starting point in our various experiments). The clusters of same khaki colors are a visualization trade off because the visualization tool does not have enough colors to show each clusters.

7.3 Enhance visits understanding

One objective in the company was to determine visit properties giving the best insights on specifics goals. To achieve this, one strategy was to exhaustively explore every feature combination. Unfortunately due to the combinatorial complexity of exploring every feature combination, this was rapidly limited to exploring restricted feature space in order to make the task feasible.

An interesting proposed technique is to combine clustering with decision trees or random forest. The first step consists in applying a clustering algorithm to partition the space into regions with various common properties. The most interesting aspect being that the standard deviation of the conversion rate per cluster is high, thus indicating the presence of clusters with conversion rates significantly higher or lower than the average conversion rate.

Once the clustering is obtained, a decision tree, or a random forest, is applied with clusters as objective classes. The resulting tree gives us various combinations of features modalities leading to specific clusters. Each branch representing a combination of feature modalities, which tends to fall in a cluster which converts worse, the same, or better than the average conversion rate. Once all these combinations that we called segment are gathered, we have to check exhaustively if they belong to each visit given final results.

This workflow allows to select a restricted number of segments, enabling to explore many goals rather than only one when there are too many segments. Moreover by controlling the depth of decision trees we can choose the upper bound of features composing a segment knowing that usually segments bigger than size four are more complicated to interpret than others.

Applying this combination of clustering and decision trees allows us to discover interesting segments as well in size as in conversion rate with some correct size

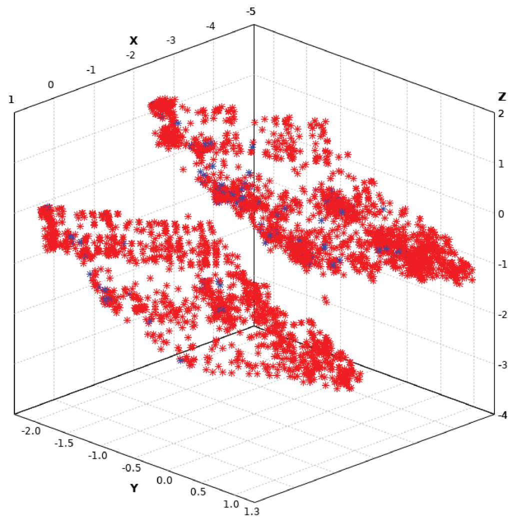


Figure 7.1: PCA

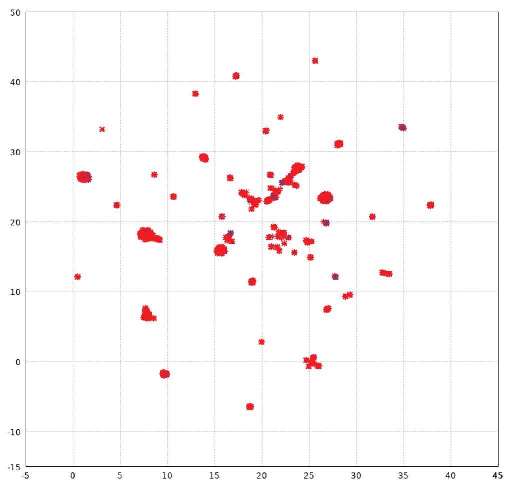


Figure 7.2: UMAP

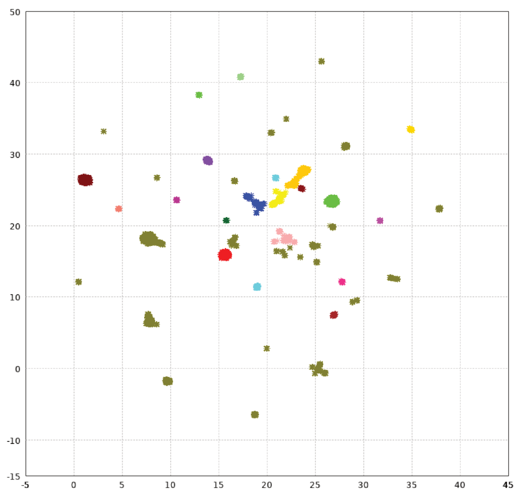


Figure 7.3: ε-proximity on UMAP

segment. This segment has a conversion rate 50% higher than the average conversion rate as well as segments with very low conversion rate.

7.4 Enhance predictive model accuracy with clustering

One rising question could be how can clustering help in a supervised problem ? One answer to that question resides in the fact that a clustering provides specific latent information over data. This information is linked to the nature of the algorithm applied and can eventually help the supervised process.

Experiments consisted in applying a clustering on starting visits composed essentially of static features (the ones always coming with regular users). The clustering results (labels) are encoded and added to the other features. Then the model is trained and tested on split datasets.

The used supervised technique is the logistic regression. We tried to represent the new clustering features under different forms, as an integer and others, which gives better performance with the accuracy even if these results are not enough significant to be considered interesting.

7.5 Clustering4Ever, an Open Source Project

During this thesis, we were often confronted to many papers without access to the source code and without enough implementation details in order to implement properly the proposed algorithms. Thus, for reproducibility we decided to share as an easy to use open source project the source code of different algorithms. This project untitled Clustering4Ever offers the possibility to anyone to read source code and test different algorithms either via notebooks as Spark-Notebook[1] or calling directly the API's methods by adding two lines in the typical configuration file (build.sbt) of their *scala/java* project. The design allows to generate algorithms working for many types of data (continuous, binary and mixed data). Therefore someone desiring to conceive a new algorithm will have the possibility to write a generic implementation using dedicated *trait*, similar to Java *Interface*, where depending algorithm operations, could work on any data type through final specific implementations.

During the process other members of the laboratory join the project to enhance the API with various advises and algorithms. This work is passed from a personal objective to a collective one which increased both quality and fun developing it.

Clustering algorithms

- Jenks Natural Breaks
- K-Centers
  - K-Means, K-Modes, K-Prototypes
- Tensor Biclustering
  - Folding-Spectral, Unfolding-Spectral, Thresholding Sum Of Squared Trajectory Length, Thresholding Individuals Trajectory Length, Recursive Biclustering, Multiple Biclustering
- Self Organizing Maps
- G-Stream
- PatchWork
- Random Local Area
- Clusterwise
- ε-proximity
- DBSCAN

[1] http://spark-notebook.io/

Dimensions Reduction And Selection algorithms

- Dimensions Reduction
  - PCA
  - UMAP
- Dimensions Selection
  - Sp-RST

Clustering indices

- Internal Indices
  - Davies Bouldin
  - Ball Hall
- External Indices
  - Multiple Classification
    - Mutual Information, Normalized Mutual Information
    - Purity
    - Accuracy, Precision, Recall, fBeta, f1, RAND, ARAND, Matthews correlation coefficient, CzekanowskiDice, RogersTanimoto, FolkesMallows, Jaccard, Kulcztnski, McNemar, RusselRao, SokalSneath1, SokalSneath2
  - Binary Classification
    - Accuracy, Precision, Recall, fBeta, f1

7.6 Conclusion

A major objective of Kameleoon company is to be able to predict, with sufficient accuracy and as early as possible, the probability that a visit will convert to a specific goal. In this chapter we have proposed some experiments and typical applications in the company. We also had the opportunity to make all our algorithms available in open source. We have also provided an easy-to-use API. The C4E project currently brings together all doctoral students interested in its big data technology and several contributors.

Chapter 8

Conclusion and Perspectives

8.1 Research work achieved during the thesis

The main aim of this thesis was to allow scalability of algorithms around clustering problematic. Mean Shift clustering is a generalization of the k-means clustering which computes arbitrarily shaped clusters defined as the basins of attraction to the local modes created by the density gradient ascent paths. Despite its potential, the Mean Shift approach is a computationally expensive method for unsupervised learning. Thus, we introduce two contributions aiming to provide clustering algorithms with a linear time complexity, as opposed to the quadratic time complexity for the exact Mean Shift clustering. Firstly we propose a scalable procedure to approximate the density gradient ascent. Secondly, our proposed scalable cluster labeling technique is presented. It is a DBScan derivative with the ε-epsilon proximity clustering algorithm. Both propositions are based on Locality Sensitive Hashing (LSH) to approximate nearest neighbors.

Afterwards, in the third contribution, we describe the theory and practice behind a new modal clustering method (mean-shift-like) for binary data. Our approach (BinNNMS) is based on the nearest neighbor median shift. The median shift is an extension of the well-known mean shift, which was designed for continuous data, to handle binary data. We demonstrate that BinNNMS can accurately discover the location of clusters in binary data.

In this thesis, we propose contributions done in a collaborative context. We revisit Clusterwise methods for regression. When applied to massive data, they either have prohibitive computational costs or produce models that are difficult to interpret. We introduce a new implementation Micro-Batch Clusterwise Partial LeastSquares (mb-CW-PLS), which consists of two main improvements: (a) a scalable and distributed computational framework and (b) a micro-batch Clusterwise regression using micro-clusters. The last contributions, we propose new distributed versions of RST (Rough Set Theory). The first approach (Sp-RST) splits the given dataset into partitions with smaller numbers of features which are then processed in parallel. The second approach is based on distributed system and on Locality Sensitive Hashing (LSH used as clustering method), named LSH-dRST, for big data pre-processing. LSH-dRST uses LSH to match similar features into the same bucket and maps the generated buckets into partitions on which Sp-RST is applied on.

Towards the end, we propose to share as an open source project, because reproducibility is an important objective of science experiments. This project titled Clustering4Ever offers the possibility for anyone to read the source code and test the different algorithms either via notebooks or calling directly the API's. The design enables the generation of algorithms working for many types of data from continuous to binary or any other kind of data.

This thesis was funded by Kameloon as a CIFRE grant. We presented research work and also implementations at the company. We also presented the performance of the various algorithms we proposed.

8.2 Perspectives

Several perspectives can be considered as a result of this thesis:

- Future work includes exploring the utility of NNGA$^+$ for clustering algorithms with very high dimensional datasets as this is one of the most important advantages of nearest neighbor methods over kernel and other dense methods, different dissimilarity measure on the ε-proximity clustering and different hashing methods [116, 129] other than the simple hashing we have utilized. We aim also to extend the Median-shift algorithm. In [122] authors present the median computation in near linear time which seems to be a good manner to deal with massive datasets. We also consider extending the Mean Shift to mixed data. Thus, we wish to extend this scalable algorithm to address mixed and streaming data.

- Another aspect we would like to pursue is the combination of supervised and unsupervised methods for classification, more specifically clusterwise or typological regression. The proposed Clusterwise design has enabled to moderately increase its speed and its usage on bigger dataset, whilst keeping satisfying results. Another improvement will be to transform it in a more generic way to enable any regression or classification models.

- The other area of research we want to develop is the visualization of large amounts of data. A lot of research has been done on the extraction of relevant parameters to be visualized using clustering. Indeed, this is an increasingly important problem as the amount of information continues to grow up. Many visualization techniques need to be rethought to handle huge data. One of the objectives of this research axis is to propose new approaches that make it possible to use visual information on a density zone using the algorithms described above in order to visualize continuous or categorical data. We have worked with Duong Tarn on a specific visualization aspect, which is the kernel density estimation [126]. It works for 2D points, but we are planning to extend it to higher dimensional spaces.

Part IV

Appendices

Appendix A

Evaluation metrics

A.0.1 Internal clustering indices

We are focusing here on most known internal indices. Some interesting indices are cited in this cran-r repository.

A.0.1.1 Davies-Bouldin

The most famous one is the Davies-Bouldin index [13], which is grossly the ratio between intra-cluster distance, ie distance from cluster points to their centroid, and extra-cluster distance, ie distance between centroids.

Important properties

- linear complexity in n, scalable for decent k values
- quadratic complexity in k, scalable for $k \approx\leq 10000$
- fit well with elliptic clustering

A.0.1.2 Silhouette

The silhouette index provides how similar point is close to its own cluster compared to others clusters. The range of values is within $[-1, 1]$, that high values represent clusters where points are deeply linked to them whilst being dissimilar from others clusters.The silhouette can be calculated with any distance metric. Unfortunately the time complexity of this index is $O(n^2)$.

A.0.2 External clustering indices

External indices require two partitions of labels in order to return a score, the first one is the ground truth, the other is either the result of a clustering or the predictions from a classification algorithm.

A.0.2.1 Mutual Information (MI)

In information theory, the mutual information measure mutual dependencies between two random variable. It describe the quantity of information which can

	1	0
1	TP	TN
0	FN	FP

Table A.1: Confusion matrix

be obtained about one random variable by observing the other. It's general formulation for continuous random variable is as follow:

$$\mathrm{I}(X;Y) = \int_{\mathcal{Y}} \int_{\mathcal{X}} p(x,y) \log\left(\frac{p(x,y)}{p(x)\,p(y)}\right) dx\,dy,$$

It is non negative: $\mathrm{I}(X;Y) \geq 0$ and symetric $\mathrm{I}(X;Y) = \mathrm{I}(Y;X)$

Normalized Mutual Information: It exist different normalized version of mutual information two of the most popular are

$$NMI = \frac{\mathrm{I}(X;Y)}{\sqrt{\mathrm{H}(X)\mathrm{H}(Y)}}\ .$$

and

$$NMI = \frac{\mathrm{I}(X;Y)}{\max(\mathrm{H}(X),\mathrm{H}(Y))}\ .$$

A.0.2.2 Contingency matrix for binary classification

Contingency matrix is a tool, which enable to know how two labeled partitions are linked. It represent essentially four items which are true positive (TP) when ground truth and prediction are equal to the realized objective. True negative (TN) when both value present the unrealized objective. False positive (FP) when ground truth indicate a negative example and model predict a positive one. Finally false negative will occur when ground truth indicates negative outcome and model predict a positive output. It exist two main version of this matrix which are the binary classes one and the multi-classes, both can be compute in $O(n)$ which

Binary classification: It is a 2×2 matrix containing as shown on table A.1 results of prediction knowing the the ground truth.

Multi class classification: Multi class contingency matrix is a generalisation of binary matrix for any number of classes knowing that they can differ between ground truth and prediction.

$X \backslash Y$	Y_1	Y_2	$\ldots$	Y_s	Sums
X_1	n_{11}	n_{12}	$\ldots$	n_{1s}	a_1
X_2	n_{21}	n_{22}	$\ldots$	n_{2s}	a_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
X_r	n_{r1}	n_{r2}	$\ldots$	n_{rs}	a_r
Sums	b_1	b_2	$\ldots$	b_s	

The matrix is used to determined for previous values using following rules.

$$TP + FP = \sum_{i=1}^{s} \binom{\sum_{j=1}^{r} n_{ji}}{2}$$

$$TP + FN = \sum_{i=1}^{r} \binom{\sum_{j=1}^{s} n_{ij}}{2}$$

$$TP = \sum_{i=1}^{s} \sum_{j=1}^{r} \binom{n_{ij}}{2}$$

$$FP = TP + FP - TP$$

$$FN = TP + FN - TP$$

$$TN = \sum_{i=1}^{s} \sum_{j=1}^{r} \binom{n_{ij}}{2} - TP - FP - FN$$

From there, many external indices exists defined from these four values as Rogers Tanimoto A.0.2.4 , Jaccard A.0.2.4, RAND A.0.2.4 and many others.

A.0.2.3 Binary class classification

Recall: It represents the True Positive Rate

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN}$$

Precision: it exposes the Positive Predictive Value

$$PPV = \frac{TP}{TP + FP}$$

F_β **measure:** F_β measure average both precision and recall into a single measure.

$$F(\beta) = \left(1 + \beta^2\right) \cdot \left(\frac{PPV \cdot TPR}{\beta^2 \cdot PPV + TPR}\right)$$

A.0.2.4 Multi-class classification

RAND: The RAND index is a measure of similarity between two clustering giving values between $[0, 1]$ $R = \frac{a+b}{a+b+c+d} = \frac{a+b}{\binom{n}{2}}$

Adjusted RAND It use the contingency matrix as follow

$$\overbrace{ARI}^{\text{Adjusted RAND Index}} = \frac{\overbrace{\sum_{ij}\binom{n_{ij}}{2}}^{\text{Index}} - \overbrace{[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}]/\binom{n}{2}}^{\text{Expected Index}}}{\underbrace{\frac{1}{2}[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}]}_{\text{Max Index}} - \underbrace{[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}]/\binom{n}{2}}_{\text{Expected Index}}}$$

$$FolkesMallows = \frac{TP}{\sqrt{(TP+FN)*(FP+TP)}}$$

Jaccard: The Jaccard coefficient measures similarity between two finite clustering. It is defined as the intersection of sets size divided by the size of sample sets union.

$$Jaccard = \frac{TP}{TP+FN+FP} = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Rogers-Tanimoto: The Rogers-Tanimoto clustering index came from the comemberships of the observations where comembership is defined as the pairs of observations that are put into same cluster.

$$RogersTanimoto = \frac{TP+TN}{TP+(2*(FN+FP))+TN}$$

Measure by label: Label sets L is defined as follows:

$$L = \{\ell_0, \ell_1, \ldots, \ell_{M-1}\}$$

knowing that

$$\hat{\mathbf{y}}_0, \hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_{N-1} \in L$$

are the estimate classes and

$$\hat{\delta}(x) = \begin{cases} 1 & \text{if } x = 0, \\ 0 & \text{otherwise.} \end{cases}$$

be a modified $\hat{\delta}$ function. With theses tool we can define multi class classifications measures.

Recall: Before define a global recall score, we have to set a recall by label with the one versus all strategy.

$$TPR(\ell) = \frac{TP}{P} = \frac{\sum_{i=0}^{N-1} \hat{\delta}(\hat{\mathbf{y}}_i - \ell) \cdot \hat{\delta}(\mathbf{y}_i - \ell)}{\sum_{i=0}^{N-1} \hat{\delta}(\mathbf{y}_i - \ell)}$$

Then we can average the whole results into a weighted recall

$$TPR_w = \frac{1}{N} \sum_{\ell \in L} TPR(\ell) \cdot \sum_{i=0}^{N-1} \hat{\delta}(\mathbf{y}_i - \ell)$$

Precision: The process for precision is the same by defining precision by label

$$PPV(\ell) = \frac{TP}{TP+FP} = \frac{\sum_{i=0}^{N-1} \hat{\delta}(\hat{\mathbf{y}}_i - \ell) \cdot \hat{\delta}(\mathbf{y}_i - \ell)}{\sum_{i=0}^{N-1} \hat{\delta}(\hat{\mathbf{y}}_i - \ell)}$$

Then obtain a weighted precision.

$$PPV_w = \frac{1}{N} \sum_{\ell \in L} PPV(\ell) \cdot \sum_{i=0}^{N-1} \hat{\delta}(\mathbf{y}_i - \ell)$$

F_β **measure:** F-measure follow the same pattern with the f-measure by label

$$F(\beta,\ell) = \left(1+\beta^2\right)\cdot\left(\frac{PPV(\ell)\cdot TPR(\ell)}{\beta^2\cdot PPV(\ell)+TPR(\ell)}\right)$$

To finally get a weighted F-measure

$$F_w(\beta) = \frac{1}{N}\sum\nolimits_{\ell\in L} F(\beta,\ell)\cdot\sum_{i=0}^{N-1}\hat{\delta}(\mathbf{y}_i-\ell)$$

Accuracy $ACC = \frac{TP}{TP+FP} = \frac{1}{N}\sum_{i=0}^{N-1}\hat{\delta}\left(\hat{\mathbf{y}}_i-\mathbf{y}_i\right)$

A.0.3 Regressions evaluation metrics

This subsection highlight some of principal regression metrics for supervised learning because of Chapter 5 which present a combination of clustering and supervised learning

A.0.3.1 Mean Absolute Error (MAE)

Mean absolute error is a measure defining difference between two continuous variables. It is the mean of the difference between the true value ($\hat{y}$) and its estimate (y)

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n}|\hat{y}_i-y_i|}{n} = \frac{\sum_{i=1}^{n}|e_i|}{n}.$$

A.0.3.2 Root Mean Square Error (RMSE)

RMSE also measures the average magnitude of the error. It is the square root of the average of squared differences between prediction and observation.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(\hat{y}_i-y_i)^2}{N}}$$

Similarities and differences between MAE and RMSE As MAE, RMSE expresses prediction error in variable of interest unit. The two metrics are defined on $\mathcal{R}^+$ and do not take into account errors directions, the lower is the value the better is the score. Because errors are squared before they are averaged, the RMSE gives higher weight to large errors which could be useful if users desire to penalise use case with large errors.

Appendix B

Technical details about Map/Reduce

In a synthetic way, some technical details

Transformations are operations that could be chained without triggering any computation. A simple way to distinguish *transformations* from *actions* is to look at the definitions of methods. Those that return an *RDD, Dataframe*, and a *Dataset* are transformations, others that return a non distributed object are *actions*.

map The *map* method is the most basic transformation on *RDD* which transform an *RDD[A]* toward and *RDD[B]* by applying a function *f:* $A \to B$ on each element of type *A* from the RDD on which *map* is applied.

Listing 5 *map* function definition

```
def map[B](f: (A) => B): RDD[B]
```

reduce The *reduce* method is one of the basic action which as its *Scala* equivalent reduces a collection of *A* to an unique element of type *A*. Each *Spark* partition will apply a reduce on its iterator of data, then processed partitions are sent to the master node in order to be reduced on it.

Listing 6 *reduce* function definition

```
def reduce(op: (A1, A1) => A1): A1
```

fold The *fold* method is a generalisation of reduce also present in scala, its specificity resides in an extra argument witch is the neutral element associated to the reduce operation.

ByKey operations Many operations in *Spark* are *ByKey* ones, they all are transformations working on *Key-Value RDD* which apply specific operations on all elements sharing keys which are *equal* to return a new *Key-Value RDD* with as elements as distinct *Key*. We suppose here that we are working with a *Key-Value RDD[(K, V)]*.

reduceByKey is a reduce which is applied on each element sharing same keys. Its definition is as follow :

Listing 7 *reduceByKey* function definition

```
def reduceByKey(func: (V, V) => V): RDD[(K, V)]
```

foldByKey is a fold applied on each element sharing same keys. The definition of *foldByKey* is as follows :

Listing 8 *foldByKey* function definition

```
def foldByKey(zeroValue: V)(func: (V, V) => V): RDD[(K, V)]
```

aggregateByKey gathers elements sharing same keys into a single partition in a tuple *Key-U* where U is the type of the aggregate. The definition is as follows :

Listing 9 *aggregateByKey* function definition

```
def aggregateByKey[U](zeroValue: U)(seqOp: (U, V) => U, combOp: (U, U) =>
 ↪  U)(implicit arg0: ClassTag[U]): RDD[(K, U)]
```

combineByKey is the most general operation which will transform each value into a new value type then combine obtained values into a new one of the same type, it can take the role as well as a *aggregateByKey* as a *reduceByKey* or a *foldByKey*. Its definition is as follows :

Listing 10 *combineByKey* function definition

```
def combineByKey[C](createCombiner: (V) => C, mergeValue: (C, V) => C,
 ↪  mergeCombiners: (C, C) => C): RDD[(K, C)]
```

Listing 11 K-Centers spark heart using tail recursion

```
protected final def obtainCenters[O, Cz[Y, Z <: GVector[Z]] <:
 ↪  Clusterizable[Y, Z, Cz]](data: RDD[Cz[O, V]])(implicit ct:
 ↪  ClassTag[Cz[O, V]]): immutable.HashMap[Int, V] = {

  data.persist(persistanceLVL)

  val unSortedCenters = if(customCenters.isEmpty)
   ↪  kmppInitializationRDD(data.map(_.v), k, metric) else customCenters
  val centers = mutable.ArrayBuffer(unSortedCenters.toSeq:_*).sortBy(_._1)

  @annotation.tailrec
  def go(cpt: Int, haveAllCentersConverged: Boolean, centers:
   ↪  mutable.ArrayBuffer[(Int, V)]): mutable.ArrayBuffer[(Int, V)] = {
    val preUpdatedCenters = mutable.ArrayBuffer(
          data.map( cz => (obtainNearestCenterID(cz.v, centers, metric),
           ↪  cz.v) )
            .reduceByKeyLocally{ case (v1, v2) =>
             ↪  ClusterBasicOperations.obtainCenter(Seq(v1, v2), metric) }
              .toArray
        :_*).sortBy(_._1)
        val alignedOldCenters = preUpdatedCenters.map{ case (oldClusterID,
         ↪  _) => centers(oldClusterID) }
        val updatedCenters = preUpdatedCenters.zipWithIndex.map{ case
         ↪  ((oldClusterID, center), newClusterID) => (newClusterID,
         ↪  center) }
        val shiftingEnough = areCentersNotMovingEnough(updatedCenters,
         ↪  alignedOldCenters, minShift, metric)
        if(cpt < maxIterations && !shiftingEnough) {
                go(cpt + 1, shiftingEnough, updatedCenters)
        }
        else {
                updatedCenters
        }
  }
  immutable.HashMap(go(0, false, centers):_*)
}
```

Listing 12 K-Centers heart using tail recursion

```
@annotation.tailrec
def go(cpt: Int, haveAllCentersConverged: Boolean, centers:
 ↪  mutable.ArrayBuffer[(Int, V)]): mutable.ArrayBuffer[(Int, V)] = {
  val preUpdatedCenters = mutable.ArrayBuffer(
    data.groupBy( cz => obtainNearestCenterID(cz.v, centers, metric) )
      .map{ case (clusterID, aggregate) =>
      (
        clusterID,
        ClusterBasicOperations.obtainCenter(aggregate.map(_.v), metric)
      )
      }.seq.toSeq
  :_*).sortBy(_._1)
  val alignedOldCenters = preUpdatedCenters.map{ case (oldClusterID, _) =>
   ↪  centers(oldClusterID) }
  val updatedCenters = preUpdatedCenters.zipWithIndex.map{ case
   ↪  ((oldClusterID, center), newClusterID) => (newClusterID, center) }
  val shiftingEnough = areCentersNotMovingEnough(updatedCenters,
   ↪  alignedOldCenters, minShift, metric)
  if(cpt < maxIterations && !shiftingEnough) {
    go(cpt + 1, shiftingEnough, updatedCenters)
  }
  else {
    updatedCenters
  }
}
```

Bibliography

[1] R. W. Hamming. “Error detecting and error correcting codes”. In: *Bell Syst. Tech. J.* 29 (1950), pp. 147–160. ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1950.tb00463.x.

[2] D. O. Loftsgaarden and C. P. Quesenberry. “A Nonparametric Estimate of a Multivariate Density Function”. In: *Ann. Math. Statist.* 36 (June 1965), pp. 1049–1051. DOI: 10.1214/aoms/1177700079. URL: https://doi.org/10.1214/aoms/1177700079.

[3] Stephen C. Johnson. “Hierarchical clustering schemes”. In: *Psychometrika* 32.3 (1967), pp. 241–254. ISSN: 1860-0980. DOI: 10.1007/BF02289588. URL: https://doi.org/10.1007/BF02289588.

[4] J. MacQueen. “Some methods for classification and analysis of multivariate observations”. In: *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*. Berkeley, USA: University of California Press, 1967, pp. 281–297. URL: https://projecteuclid.org/euclid.bsmsp/1200512992.

[5] Bock HH. “The equivalence of two extremal problems and its application to the iterative classification of multivariate data.” In: (1969).

[6] K. Fukunaga and L. Hostetler. “Optimization of k-nearest-neighbor density estimates”. In: *IEEE Trans. Inform. Theory* 19 (1973), pp. 320–326.

[7] K. Fukunaga and L. Hostetler. “The estimation of the gradient of a density function, with applications in pattern recognition”. In: *IEEE T. Inform. Theory* 21 (1975), pp. 32–40. ISSN: 0018-9448. DOI: 10.1109/TIT.1975.1055330.

[8] J. Aitchison and C. G. G. Aitken. “Multivariate Binary Discrimination by the Kernel Method”. In: *Biometrika* 63 (1976), pp. 413–420. ISSN: 00063444.

[9] E. Diday and J. C. Simon. “Clustering Analysis”. In: *Digital Pattern Recognition*. Ed. by King Sun Fu. Berlin: Springer, 1976, pp. 47–94. DOI: 10.1007/978-3-642-96303-2_3. URL: https://doi.org/10.1007/978-3-642-96303-2_3.

[10] Diday E. “Classification et selection de paramètres sous contraintes.” In: (1976). Rapport de recherche IRIA-LABORIA, 188.

[11] Charles C. “Regression typologique et reconnaissance des formes.” PhD thesis. University of Paris IX. PhD thesis. 1977.

[12] Arthur P Dempster, Nan M Laird, and Donald B Rubin. “Maximum likelihood from incomplete data via the EM algorithm”. In: *Journal of the royal statistical society. Series B (methodological)* (1977), pp. 1–38.

[13] David L. Davies and Donald W. Bouldin. “A Cluster Separation Measure”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 1.2 (Feb. 1979), pp. 224–227. ISSN: 0162-8828. DOI: 10.1109/TPAMI.1979.4766909. URL: http://dx.doi.org/10.1109/TPAMI.1979.4766909.

[14] Spath H. “Clusterwise linear regression.” In: *Computing* 22 (1979), pp. 367–373.

[15] Y. P. Mack and M. Rosenblatt. “Multivariate k-nearest neighbor density estimates”. In: *J. Multivariate Anal.* 9 (1979), pp. 1–15.

[16] K. C. Li. “Consistency for cross-validated nearest neighbour estimates in nonparametric regression”. In: *Ann. Stat.* 12 (1984), pp. 230–240.

[17] Wold H. “Partial Least Squares.” In: *Encyclopedia of Statistical Sciences.* 6 (1985), pp. 581–591.

[18] L. Hubert and P. Arabie. “Comparing partitions”. In: *J. Classif.* 2 (1985), pp. 193–218.

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

[20] DeSarbo WS and Cron WL. “A maximum likelihood methodology for Clusterwise linear regression.” In: *Journal of Classification* 5 (1988), pp. 249–282.

[21] Lohmoller JB. “Latent variables path modeling with partial least squares.” In: *Physica-Verlag* (1989).

[22] M. C. Jones. “Variable kernel density estimates and variable kernel density estimates”. In: *Aust. J. Stat.* 32 (1990), pp. 361–371.

[23] Bernd Fritzke. “Unsupervised Clustering With Growing Cell Structures”. In: *In Proceedings of the International Joint Conference on Neural Networks*. IEEE, 1991, pp. 531–536.

[24] T. Martinetz and K. Schulten. “A "Neural-Gas" Network Learns Topologies”. In: *Artificial Neural Networks* I (1991), pp. 397–402.

[25] Message P Forum. *MPI: A Message-Passing Interface Standard*. Tech. rep. Knoxville, TN, USA, 1994.

[26] Y. Cheng. “Mean Shift, Mode Seeking, and Clustering”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 17 (1995), pp. 790–799. ISSN: 0162-8828. DOI: 10.1109/34.400568. URL: http://dx.doi.org/10.1109/34.400568.

[27] Martin Ester et al. “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”. In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA*. 1996, pp. 226–231.

[28] B. Holmquist. “The d-variate vector Hermite polynomial of order k”. In: *Linear Algebra Appl.* 237/238 (1996), pp. 155–190.

[29] A. Broder. “On the Resemblance and Containment of Documents”. In: *Proceedings of the Compression and Complexity of Sequences 1997*. SEQUENCES ’97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 21–. ISBN: 0-8186-8132-2.

[30] Z. Huang. “Clustering large data sets with mixed numeric and categorical values”. In: *The First Pacific-Asia Conference on Knowledge Discovery and Data Mining*. 1997, pp. 21–34.

[31] Chris Fraley and Adrian E Raftery. “How many clusters? Which clustering method? Answers via model-based cluster analysis”. In: *The computer journal* 41.8 (1998), pp. 578–588.

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

[33] P. Indyk and R. Motwani. “Approximate nearest neighbors: Towards removing the curse of dimensionality.” In: *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*. 1998, pp. 604–613.

[34] Piotr Indyk and Rajeev Motwani. “Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality”. In: *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*. STOC ’98. Dallas, Texas, USA: ACM, 1998, pp. 604–613. ISBN: 0-89791-962-9. DOI: 10.1145/276698.276876.

[35] Samuel Kaski, Jari Kangas, and Teuv Kohonen. “Bibliography of Self-Organizing Map (SOM) Papers: 1981-1997”. In: *Neural computing surveys* 1 (1998), pp. 102–350.

[36] F. Leisch, A. Weingessel, and E. Dimitriadou. “Competitive Learning for Binary Valued Data”. In: *ICANN 98*. Ed. by L. Niklasson, M. Bodén, and T. Ziemke. London: Springer, 1998, pp. 779–784. ISBN: 978-1-4471-1599-1.

[37] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. “Similarity search in high dimensions via hashing”. In: *VLDB*. 1999, pp. 518–529.

[38] Xiaowei Xu, Jochen Jäger, and Hans-Peter Kriegel. “A fast parallel clustering algorithm for large spatial databases”. In: *High Performance Data Mining*. Springer, 1999, pp. 263–290.

[39] Ivo Düntsch and Günther Gediga. “Rough set data analysis”. In: *Encyclopedia of Computer Science and Technology* 43.28 (2000), pp. 281–301.

[40] M. Lebbah, F. Badran, and S. Thiria. “Topological map for binary data”. In: *ESANN 2000, 8th European Symposium on Artificial Neural Networks, Bruges, Belgium, April 26-28, 2000, Proceedings*. 2000, pp. 267–272.

[41] D. Comaniciu, V. Ramesh, and P. Meer. “The variable bandwidth mean shift and data-driven scale selection”. In: *Proceedings of the Eighth IEEE International Conference on Computer Vision*. Vol. 1. 2001, pp. 438–445.

[42] T. Kohonen, M. R. Schroeder, and T. S. Huang, eds. *Self-Organizing Maps*. 3rd. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2001. ISBN: 3540679219.

[43] Pawan Lingras. “Unsupervised rough set classification using GAs”. In: *Journal of Intelligent Information Systems* 16.3 (2001), pp. 215–228.

[44] D. C. Montgomery, E. A. Peck, and G. G. Vining. “Introduction to linear regression analysis.” In: (2001).

[45] W. K. Pratt. *Digital Image Processing: PIKS Inside*. 3rd. New York: John Wiley and Sons, 2001.

[46] Hahn C et al. “Capturing customer heterogeneity using finite mixture PLS approach.” In: *Schmalenbach Business Review* 54 (2002), pp. 243–269.

[47] Moses S. Charikar. “Similarity Estimation Techniques from Rounding Algorithms”. In: *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*. STOC ’02. Montreal, Quebec, Canada: ACM, 2002, pp. 380–388. ISBN: 1-58113-495-9. DOI: 10.1145/509907.509965.

[48] D. Comaniciu and P. Meer. “Mean shift: a robust approach toward feature space analysis”. In: *IEEE T. Pattern Anal.* 24 (2002), pp. 603–619.

[49] Pawan Lingras. “Rough set clustering for web mining”. In: *Fuzzy Systems, 2002. FUZZ-IEEE’02. Proceedings of the 2002 IEEE International Conference on*. Vol. 2. IEEE. 2002, pp. 1039–1044.

[50] A. Strehl and J. Ghosh. “Cluster ensembles – a knowledge reuse framework for combining multiple partitions”. In: *J. Mach. Learn. Res.* 3 (2002), pp. 583–617.

[51] D. Comaniciu. “An algorithm for data-driven bandwidth selection”. In: *IEEE T. Pattern Anal.* 25 (2003), pp. 281–288.

[52] M. Girolami and C. He. “Probability Density Estimation from Optimally Condensed Data Samples”. In: *IEEE T. Pattern Anal.* 25 (2003), pp. 1253–1264.

[53] Isabelle Guyon and André Elisseeff. “An introduction to variable and feature selection”. In: *Journal of machine learning research* 3 (2003), pp. 1157–1182.

[54] M. Datar et al. “Locality-sensitive hashing scheme based on p-stable distributions.” In: *Proceedings of the 20th Annual Symposium on Computational Geometry*. 2004, pp. 253–262.

[55] Mayur Datar et al. “Locality-sensitive Hashing Scheme Based on P-stable Distributions”. In: *Proceedings of the Twentieth Annual Symposium on Computational Geometry*. SCG ’04. Brooklyn, New York, USA: ACM, 2004, pp. 253–262. ISBN: 1-58113-885-7. DOI: 10.1145/997817.997857.

[56] Eshref Januzaj, Hans-Peter Kriegel, and Martin Pfeifle. “DBDC: Density based distributed clustering”. In: *Advances in Database Technology-EDBT 2004*. Springer, 2004, pp. 88–105.

[57] Preda C and Saporta G. “Clusterwise PLS regression on a stochastic process.” In: *Computational Statistics and Data Analysis* 49 (2005), pp. 99–108.

[58] Willmott C and Matsuura K. “Advantages of the Mean Absolute Error (MAE) over the Root Mean Square Error (RMSE) in assessing average model performance.” In: *Climate Research* 30 (2005), pp. 79–82.

[59] Leon Danon et al. “Comparing community structure identification”. In: *J. Stat. Mech.: Theory E.* 2005.09 (2005), P09008. URL: http://stacks.iop.org/1742-5468/2005/i=09/a=P09008.

[60] Hanchuan Peng, Fuhui Long, and Chris Ding. “Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy”. In: *IEEE Transactions on pattern analysis and machine intelligence* 27.8 (2005), pp. 1226–1238.

[61] A. Ultsch. “Clustering with SOM: U*C”. In: *Proceedings of the Workshop on Self-Organizing Maps*. 2005, pp. 75–82.

[62] Vinzi VE, Lauro CN, and Amato S. “PLS typological regression.” In: *New developments in classification and data analysis* (2005), pp. 133–140.

[63] Pavel Berkhin. “A survey of clustering data mining techniques”. In: *Grouping multidimensional data*. Springer, 2006, pp. 25–71.

[64] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. New York: Springer-Verlag, 2006.

[65] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006. ISBN: 0387310738.

[66] T. Li. “A Unified View on Clustering Binary Data”. In: *Mach. Learn.* 62 (2006), pp. 199–215. ISSN: 1573-0565. DOI: 10.1007/s10994-005-5316-9. URL: https://doi.org/10.1007/s10994-005-5316-9.

[67] David Arthur and Sergei Vassilvitskii. “K-means++: The Advantages of Careful Seeding”. In: *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA ’07. New Orleans, Louisiana: Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035. ISBN: 978-0-898716-24-5. URL: http://dl.acm.org/citation.cfm?id=1283383.1283494.

[68] Arthur Asuncion and David Newman. *UCI machine learning repository*. 2007.

[69] Abhinandan S Das et al. “Google news personalization: scalable online collaborative filtering”. In: *Proceedings of the 16th international conference on World Wide Web*. ACM. 2007, pp. 271–280.

[70] Hwang H, DeSarbo SW, and Takane Y. “Fuzzy Clusterwise generalized structured component analysis.” In: *Psychometrika* 72 (2007), pp. 181–198.

[71] Geoffrey McLachlan and Thriyambakam Krishnan. *The EM algorithm and extensions*. Vol. 382. John Wiley & Sons, 2007.

[72] Zdzisław Pawlak and Andrzej Skowron. “Rudiments of rough sets”. In: *Information sciences* 177.1 (2007), pp. 3–27.

[73] Kuo-Lung Wu and Miin-Shen Yang. “Mean Shift-based Clustering”. In: *Pattern Recogn.* 40 (2007), pp. 3035–3052.

[74] Jeffrey Dean and Sanjay Ghemawat. “MapReduce: Simplified Data Processing on Large Clusters”. In: *Commun. ACM* 51.1 (Jan. 2008), pp. 107–113. ISSN: 0001-0782. DOI: 10.1145/1327452.1327492. URL: http://doi.acm.org/10.1145/1327452.1327492.

[75] Z. Deng and S. Chung F.-L.and Wang. “FRSDE: Fast reduced set density estimator using minimal enclosing ball approximation”. In: *Pattern Recogn.* 41 (2008), pp. 1363–1372.

[76] Sarstedt M. “A review of recent approaches for capturing heterogeneity in partial least squares path modelling.” In: *Journal of Modelling in Management* 3 (2008), pp. 140–161.

[77] M. Slaney and M. Casey. “Locality-sensitive hashing for finding nearest neighbors”. In: *IEEE Signal Proc. Mag.* 2008, pp. 128–131.

[78] Andrea Vedaldi and Stefano Soatto. “Quick Shift and Kernel Methods for Mode Seeking”. In: *Proceedings Part IV of the 10th European Conference on Computer Vision 2008, Marseille, France*. Ed. by David Forsyth, Philip Torr, and Andrew Zisserman. 2008, pp. 705–718.

[79] Yair Weiss, Antonio Torralba, and Rob Fergus. “Spectral Hashing”. In: *Proceedings of the 21st International Conference on Neural Information Processing Systems*. NIPS’08. Vancouver, British Columbia, Canada: Curran Associates Inc., 2008, pp. 1753–1760. ISBN: 978-1-6056-0-949-2.

[80] Marco Cuturi. “Positive Definite Kernels in Machine Learning”. In: *arXiv e-prints*, arXiv:0911.5367 (2009), arXiv:0911.5367. arXiv: 0911.5367 [stat.ML].

[81] Jorge M. Santos and Mark J. Embrechts. “On the Use of the Adjusted Rand Index as a Metric for Evaluating Supervised Classification”. In: *ICANN (2)*. Ed. by Cesare Alippi et al. Springer, 2009, pp. 175–184.

[82] K Thangavel and A Pethalakshmi. “Dimensionality reduction based on rough set theory: A review”. In: *Applied Soft Computing* 9.1 (2009), pp. 1–12.

[83] Vinzi VE et al. “REBUS-PLS: a response-based procedure for detecting unit segments in PLS Path Modeling.” In: *Applied stochastic Models in Business and Industry* 24 (2009), pp. 439–458.

[84] Yair Weiss, Antonio Torralba, and Rob Fergus. “Spectral hashing”. In: *Advances in neural information processing systems*. 2009, pp. 1753–1760.

[85] Weizhong Zhao, Huifang Ma, and Qing He. “Parallel k-means clustering based on mapreduce”. In: *Cloud computing*. Springer, 2009, pp. 674–679.

[86] Junfeng He, Wei Liu, and Shih-Fu Chang. “Scalable Similarity Search with Optimized Kernel Hashing”. In: *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’10. Washington, DC, USA: ACM, 2010, pp. 1129–1138. ISBN: 978-1-4503-0055-1. DOI: 10.1145/1835804.1835946.

[87] Slava Kisilevich, Florian Mansmann, and Daniel Keim. “P-DBSCAN : A density based clustering algorithm for exploration and analysis of attractive areas using collections of geo-tagged photos”. In: *First publ. in: Proceedings of the 1st International Conference and Exhibition on Computing for Geospatial Research & Application, COM.Geo 2010, Washington, DC, USA, June 21 - 23, 2010 / Lindi Liao (Ed.). New York, N.Y. : Association for Computing Machinery, 2010, Article No.: 38* (Jan. 2010). DOI: 10.1145/1823854.1823897.

[88] G. Biau et al. “A weighted k-nearest neighbor density estimate for geometric inference”. In: *Electron. J. Stat.* 5 (2011), pp. 204–237.

[89] J. E. Chacón, T. Duong, and M. P. Wand. “Asymptotics for general multivariate kernel density derivative estimators”. In: *Stat. Sinica* 21 (2011), pp. 807–840.

[90] Alina Ene, Sungjin Im, and Benjamin Moseley. “Fast clustering using MapReduce”. In: *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2011, pp. 681–689.

[91] Junfeng He et al. “Compact Hashing with Joint Optimization of Search Accuracy and Time”. In: *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*. CVPR ’11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 753–760. ISBN: 978-1-4577-0394-2. DOI: 10.1109/CVPR.2011.5995518.

[92] G. Zheng Y. Cui K. Cao and F. Zhang. “An adaptive mean shift algorithm based on LSH”. In: *Procedia Engineering* (2011), pp. 265–269.

[93] Bahman Bahmani et al. “Scalable K-means++”. In: *Proc. VLDB Endow.* 5.7 (Mar. 2012), pp. 622–633. ISSN: 2150-8097. DOI: 10.14778/2180912.2180915. URL: http://dx.doi.org/10.14778/2180912.2180915.

[94] Bahman Bahmani et al. “Scalable K-means++”. In: *Proc. VLDB Endow.* 5 (2012), pp. 622–633.

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

[96] De Roover K, Ceulemans C, and Timmerman ME. “Clusterwise simultaneous component analysis for analyzing structural differences in multivariate multiblock data.” In: *Psychological methods* 17 (2012), pp. 100–119.

[97] Y.-H. Kung, P.-S. Lin, and C.-H. Kao. “An optimal k-nearest neighbor for density estimation”. In: *Stat. Probabil. Lett.* 82 (2012), pp. 1786–1791.

[98] Wei Liu et al. “Supervised hashing with kernels”. In: *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE. 2012, pp. 2074–2081.

[99] M. M. A. Patwary et al. “A new scalable parallel DBSCAN algorithm using the disjoint-set data structure”. In: *SC ’12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. 2012, pp. 1–11. DOI: 10.1109/SC.2012.9.

[100] Zdzisław Pawlak. *Rough sets: Theoretical aspects of reasoning about data*. Vol. 9. Springer Science & Business Media, 2012.

[101] P. Indyk S. Har-Peled and R. Motwani. "Approximate nearest neighbor: Towards removing the curse of dimensionality". In: *Theory Comput.* (2012), pp. 321–350.

[102] Matei Zaharia et al. "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing". In: *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012*. 2012, pp. 15–28.

[103] Charu C. Aggarwal and Chandan K. Reddy. *Data Clustering: Algorithms and Applications*. 1st. Chapman & Hall/CRC, 2013. ISBN: 1466558210, 9781466558212.

[104] Ali El Attar, Antoine Pigeau, and Marc Gelgon. "Robust estimation of a global Gaussian mixture by decentralized aggregations of local models". In: *Web Intelligence and Agent Systems* 11.3 (2013), pp. 245–262. DOI: 10.3233/WIA-130273. URL: http://dx.doi.org/10.3233/WIA-130273.

[105] Daniel Balouek et al. "Adding Virtualization Capabilities to the Grid'5000 Testbed". In: *Cloud Computing and Services Science*. Ed. by Ivan I. Ivanov et al. Vol. 367. Communications in Computer and Information Science. Springer International Publishing, 2013, pp. 3–20. ISBN: 978-3-319-04518-4. DOI: 10.1007/978-3-319-04519-1_1.

[106] J. E. Chacón and T. Duong. "Data-driven density estimation, with applications to nonparametric clustering and bump hunting". In: *Electron. J. Stat.* 7 (2013), pp. 499–532.

[107] M. Lichman. *UCI Machine Learning Repository*. 2013. URL: http://archive.ics.uci.edu/ml.

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

[109] Amineh Amini, Ying Wah Teh, and Hadi Saboohi. "On Density-Based Data Streams Clustering Algorithms: A Survey". In: *J. Comput. Sci. Technol.* 29.1 (2014), pp. 116–141.

[110] Yaobin He et al. "MR-DBSCAN: a scalable MapReduce-based DBSCAN algorithm for heavily skewed data". In: *Frontiers of Computer Science* 8.1 (2014), pp. 83–99. ISSN: 2095-2236. DOI: 10.1007/s11704-013-3158-3. URL: https://doi.org/10.1007/s11704-013-3158-3.

[111] Yaobin He et al. "MR-DBSCAN: a scalable MapReduce-based DBSCAN algorithm for heavily skewed data". In: *Front. Comp. Sc.* 8 (Feb. 2014), pp. 83–99.

[112] Maitry Noticewala and Dinesh Vaghela. "MR-IDBSCAN: Efficient Parallel Incremental DBSCAN algorithm using MapReduce". In: *International Journal of Computer Applications* 93 (May 2014), pp. 13–18. DOI: 10.5120/16202-5391.

[113] T. Sarazin, H. Azzag, and M. Lebbah. “SOM Clustering Using Spark-MapReduce”. In: *2014 IEEE International Parallel Distributed Processing Symposium Workshops*. 2014, pp. 1727–1734. DOI: 10.1109/IPDPSW.2014.192.

[114] Tugdual Sarazin, Hanane Azzag, and Mustapha Lebbah. “SOM Clustering Using Spark-MapReduce”. In: *2014 IEEE International Parallel & Distributed Processing Symposium Workshops, Phoenix, AZ, USA, May 19-23, 2014*. 2014, pp. 1727–1734.

[115] Daniel Schugk, Anton Kummert, and Christian Nunn. *Adaptation of the Mean Shift Tracking Algorithm to Monochrome Vision Systems for Pedestrian Tracking Based on HoG-Features*. Tech. rep. 2014. URL: https://doi.org/10.4271/2014-01-0170.

[116] Jingdong Wang et al. “Hashing for Similarity Search: A Survey”. In: *CoRR* abs/1408.2927 (2014). arXiv: 1408.2927. URL: http://arxiv.org/abs/1408.2927.

[117] Miguel Á. Carreira-Perpiñán. “A review of mean-shift algorithms for clustering”. In: *CoRR* abs/1503.00687 (2015). arXiv: 1503.00687. URL: http://arxiv.org/abs/1503.00687.

[118] Martella F, Vicari D, and Vichi M. “Partitioning predictors in multivariate regression models.” In: *Statistics and Computing* 25 (2015), pp. 261–272.

[119] Pasi Fränti. *Clustering basic benchmark*. http://cs.uef.fi/sipu/datasets. 2015.

[120] Daniel Peralta et al. “Evolutionary feature selection for big data classification: A mapreduce approach”. In: *Mathematical Problems in Engineering* 2015 (2015).

[121] Yanwei Yu et al. “Cludoop: An Efficient Distributed Density-Based Clustering for Big Data Using Hadoop”. In: *International Journal of Distributed Sensor Networks* 2015 (June 2015), pp. 1–13. DOI: 10.1155/2015/579391.

[122] Michael B. Cohen et al. “Geometric Median in Nearly Linear Time”. In: *CoRR* abs/1606.05225 (2016). arXiv: 1606.05225. URL: http://arxiv.org/abs/1606.05225.

[123] Frank Gouineau, Tom Landry, and Thomas Triplet. “PatchWork, a Scalable Density-grid Clustering Algorithm”. In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. SAC ’16. Pisa, Italy: ACM, 2016, pp. 824–831. ISBN: 978-1-4503-3739-7. DOI: 10.1145/2851613.2851643. URL: http://doi.acm.org/10.1145/2851613.2851643.

[124] Schlittgen R et al. “Segmentation of PLS path models by iterative reweighted regressions.” In: *Journal of Business Research* 69 (2016), pp. 4583–4592.

[125] D. Dheeru and E. Karra Taniskidou. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[126] Artur Gramacki and JarosÅĆaw Gramacki. “FFT-Based Fast Computation of Multivariate Kernel Density Estimators With Unconstrained Bandwidth Matrices”. In: *Journal of Computational and Graphical Statistics* 26.2 (2017), pp. 459–462. DOI: 10.1080/10618600.2016.1182918.

[127] Bougeard S et al. “Clusterwise analysis for multiblock component methods. Advances in Data Analysis and Classification.” In: *Springer Berlin Heidelberg* (2017). DOI: https://doi.org/10.1007/s11634-017-0296-8.

[128] Leland McInnes, John Healy, and James Melville. “UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction”. In: *arXiv e-prints*, arXiv:1802.03426 (2018), arXiv:1802.03426. arXiv: 1802.03426 [stat.ML].

[129] Anne Morvan et al. *On the Needs for Rotations in Hypercubic Quantization Hashing*. 2018. arXiv: 1802.03936 [cs.LG].