CONTRIBUTIONS TO PARALLEL AND DISTRIBUTED COMPUTING IN KNOWLEDGE DISCOVERY AND DATA MINING

By

Elio Lozano Inca

A Ph.D. thesis submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTATION AND INFORMATION SCIENCE AND ENGINEERING

UNIVERSITY OF PUERTO RICO

MAYAGÜEZ CAMPUS

August, 2006

Approved by

Dr. Edgar Acuña Fernandez President, Graduate Committee	Date
Dra. Dorothy Bollman Member, Graduate Committee	Date
Dr. Robert Acar Member, Graduate Committee	Date
Dr. José Fernando Vega Member, Graduate Committee	Date
Dr. Dorial Castellanos Representant, Office of Graduate Studies	Date
Dr. Manuel Rodriguez Interim Director, CISE Doctoral Program	Date
Dr. José A. Mari Mutt Director, Office of Graduate Studies	Date

Abstract

Recently databases are increasing continuously without bound, due to new data acquisition technologies. One challenge is how to gain knowledge from these large data sets. In this thesis, we analyze and improve the algorithmic solution of four problems related to knowledge discovery and data mining, making use of parallel computing; we also compare our results with related works. We design two parallel algorithms for outlier detection; the first one is for finding distance-based outliers based on nested loops along with randomization and the use of a pruning rule. The second parallel algorithm is for detecting density-based local outliers. In both cases data parallelism is used. The star coordinates plot is a useful visualization technique, but it has some drawbacks. We enhance the traditional star coordinates plot introducing new parameters that will allow us to visualize the data points in two dimensions as polygons and in three dimensions as polyhedrons. In order to visualize large data sets and reduce its computational time, a parallel algorithm is also designed. We design a new meta-classifier algorithm, and its performance is compared with base classifier algorithms and Bagged based meta-classifier algorithms. Our meta-classifier algorithm gives better results compared to other meta-classifier algorithms. For speeding up its computation time as well as making it suitable for large data sets a parallel algorithm is developed. We develop a meta-clustering algorithm and compare its performance with two Bagged based meta-clustering algorithms, and hypergraph partitioning meta-clustering algorithm. Our proposed meta-clustering algorithm gives results close to the best clustering algorithm, and is more robust to the data dependency problem. A parallel algorithm to compute four meta-clustering algorithm is also designed.

The experimental results of our collection of sequential and parallel programs is tested in two different clusters of Linux-based workstations using real-world databases available in the Machine Learning Repository of the University of California at Irvine.

Resumen

Actualmente las bases de datos están en continuo crecimiento debido a los avances en la recoleción de datos. Los desafíos recientes radican en obtener información útil de bases de datos grandes. En esta tesis analizaremos y mejoraremos la solución algorítmica de cuatro problemas relacionados a descubrimiento del conocimiento y minería de datos haciendo uso de computación paralela. Nuestros resultados son comparados también con otros trabajos relacionados. Se diseño dos algoritmos paralelos, el primero basado en ciclos anidados conjuntamente con aleatorización y una regla de poda. El segundo basado en densidad local. En ambos casos usamos la técnica de paralelismo de datos. Las coordenadas estrella es una técnica de visualización muy útil, pero tiene sus limitaciones. Nosotros mejoramos esta técnica usando nuevos parámetros, la cual nos permite visualizar puntos de datos en dos dimensiones como polígonos y en tres dimensiones como poliedros. Con el objetivo de visualizar datos grandes y reducir su tiempo computacional se desarrolló un algoritmo paralelo. Se diseño un algoritmo meta-clasificador, y su rendimiento es comparado con meta-clasificadores basados en *Bagging*. Nuestro algoritmo meta-classificador obtiene los mejores resultados. Con el objetivo de acelerar su tiempo computacional y trabajar con conjuntos de datos grandes se desarrolla una versión paralela de este algoritmo. En esta tesis desarrollamos un algoritmo meta-conglomerado y su rendimiento es comparado con dos algoritmos meta-conglomerados basados en *Bagging* y un algoritmo meta-conglomerado basado en particionamiento de hypergrafos. Nuestro algoritmo meta-conglomerado obtiene resultados cercanos al mejor algoritmo de clustering y es mas robusto que este frente al problema de dependecia de datos. También se propone un algoritmo paralelo para el cómputo de cuatro algoritmos meta-conglomerados.

Los resultados experimentales de nuestros programas se hizo en dos redes de estaciones de trabajo basados en Linux, usando datos provenientes del repositorio del *"Machine Learning Repository of the University of California at Irvine"*.

Dedicated to God, Virgen Maria, my mother Simeona, my father Samuel, my brothers Rosalio, Rosa, Rafael, Ernesto, Fernando and all my family

Acknowledgment

I would like to thank my advisor Dr. Edgar Acuña for his dedication, suggestions and collaboration in the present investigation; my graduate committee members, for their valuable suggestions; the administrative staff; faculty members; the CASTLE research group of the Mathematics Department; the Doctoral program CISE (Computation and Information Science and Engineering) of University of Puerto Rico, at Mayagüez Campus; and my family and friends, for their appreciation and dedication.

I would like also to thank the Office of Naval Research (ONR) - (grant number. N00014-03-1-0359) for their financial support: without the aid provided, the present work would not have been accomplished.

Contents

List of Tables **xi**

List of Figures **xii**

1 Introduction **1**

1.1 Parallel computing terminology 2

1.2 Parallel Programming Models 3

1.3 Outlier Detection 3

1.4 Data visualization 4

1.5 Meta-classifiers 4

1.6 Meta-clustering 4

1.7 Software and hardware environment 6

1.8 Algorithm Evaluation Criteria 7

1.9 Thesis structure 7

1.10 Ethics 8

2 Distance and Density based Outlier Detection **11**

2.1 Introduction 11

2.2 Distance-based outlier detection 13

2.2.1 The Bay's Algorithm 14

2.2.2 Parallel Bay's Algorithm 15

2.3 Density-based local outlier detection 16

2.3.1 Parallel LOF Algorithm 18

2.4 Experimental results 19
2.4.1 Description of Input and Output Parameters 19
2.4.2 Speedup 20

3 Visualization **22**
3.1 Star Coordinates 23
3.2 3D star Coordinates 25
3.3 Algorithm for 3D star coordinates 27
3.4 Parallel algorithm for 3D star coordinates 28
3.5 Experimental Methodology 33

4 Meta-classifiers **38**
4.0.1 Literature review 39
4.0.2 Our Work 41
4.0.3 Motivation 41
4.0.4 Organization 42
4.1 Supervised Classification 42
4.2 Base Classifier algorithms 44
4.2.1 The C4.5 algorithm 44
4.2.2 Radial Basis Function Networks 45
4.2.3 The Kernel Density Classifier 47
4.2.4 The K- Nearest Neighbors Classifier 50
4.2.5 The Naive Bayes Classifier 51
4.3 Ensemble Methods 51
4.3.1 Combination of generative and non-generative Ensembles . . . 52
4.3.2 Parallel design of the proposed meta-classifier algorithm . . . 54
4.4 Experimental Results 54

5 Meta-clustering **59**
5.1 Introduction 59
5.1.1 Literature Review 60

5.1.2 Motivation . . . 62
5.1.3 Our Work . . . 62
5.1.4 Organization . . . 63
5.2 Clustering Techniques in Data Mining . . . 63
5.2.1 Cluster Definition . . . 63
5.2.2 Similarity and Dissimilarity measures . . . 63
5.2.3 Taxonomy of Clustering algorithms: . . . 64
5.3 Base Clustering Algorithms . . . 65
5.3.1 Gaussian Mixture Models . . . 66
5.3.2 Partition Around Medoids . . . 67
5.3.3 Fuzzy C-means . . . 69
5.3.4 DBSCAN . . . 71
5.3.5 BIRCH . . . 73
5.4 Meta-clustering . . . 77
5.4.1 Bagged Clustering . . . 77
5.4.2 Majority Voting . . . 79
5.4.3 Graph partitioning . . . 79
5.5 Cluster Validation Techniques . . . 81
5.6 Parallel algorithm for Meta-clustering Algorithms . . . 83
5.7 Experimental Evaluation . . . 84

6 Conclusions **89**
6.1 Distance and Density based outliers . . . 89
6.2 Visualization . . . 90
6.3 Meta-classifiers . . . 90
6.4 Meta-clustering . . . 91

7 Future Work **92**

A Data sets and cluster description **101**
A.1 Data sets used in this thesis . . . 101

A.2 Cluster Description 105

B Parallel outlier detection algorithms **106**

B.1 Parallel Algorithms to detect outliers 106

B.1.1 Program parameters 106

C Data visualization **108**

C.1 VTK objects used in 3D star coordinate algorithm 108

C.2 Library dependencies 110

C.3 Building Binaries 111

C.4 Program parameters 111

D Meta-classifier **112**

D.1 Class hierarchy 112

D.2 Libraries and programs used 112

D.3 Building Binaries 113

D.4 Program parameters 113

E Meta Clustering **115**

E.1 Class hierarchy 115

E.2 Libraries and programs used 115

E.3 Building Binaries 117

E.4 Program parameters 117

List of Tables

3.1 Running time (sec.) and speedup of parallel 3 D star coordinate algorithm for the Shuttle data set) 36

4.1 Classification Error rates of Base and Ensemble Algorithms 56
4.2 Ranking of Classifiers and Ensembles 57
4.3 Running time (sec.) and speedup of parallel combined voting algorithm for Landsat data set 58

5.1 Accuracy and Mutual Information (MI) measures 86
5.2 Running time (Sec.) of Parallel Compound Clustering Algorithm for synthetic Parabola data set 88

A.1 Data set Description 104

List of Figures

1.1 Interrelationship between the tasks considered in this thesis 5

2.1 Bay's Algorithm 13
2.2 Parallel Bay's Algorithm 15
2.3 The LOF Algorithm 18
2.4 Parallel LOF Algorithm 19
2.5 Speedup for Parallel Bay's Algorithm 20
2.6 Speedup for Parallel LOF Algorithm 21

3.1 Star coordinate transformation from 8 dimensional data set 25
3.2 Pipeline of the 3D star Coordinate algorithm. 28
3.3 a) Wire frame view in three dimensions. b) Polyhedrons with different opacity in three dimensions. 29
3.4 Wire frame view of two overlapping points. 30
3.5 Polygons with different opacity. 30
3.6 Pipeline of the parallel 3-D star Coordinate algorithm. 32
3.7 2D star coordinates plot of Iris dataset 33
3.8 3D zoom of Iris data set 34
3.9 3D zoom of Iris data set on wire frame view 35
3.10 2D star coordinates of Iris data set with polygons 36
3.11 3D zoom of the first class of Iris data set 37

4.1 *Bagging* Algorithm 53

4.2 Proposed Ensemble Algorithm 53

5.1 PAM algorithm 70
5.2 FCM Algorithm 71
5.3 DBSCAN algorithm 72

A.1 a) Cassini data set. b) Synthetic Parabola data set. 102

Chapter 1

Introduction

Knowledge discovery is a process composed of many steps such as data cleaning, data integration, data selection, data transformation, data mining, pattern evaluation and knowledge presentation. From all these steps, data mining is the most important. Some areas related with knowledge discovery are Machine Learning, Statistics, Databases, and Data Visualization. This research is focused on almost all of these major areas. Two well known definitions for data mining are the following:

1. *Data Mining is the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data* [FGW02].

2. *Data Mining consists in the discovery of interesting, unexpected, or valuable structures in large data sets* [Han98].

Common data mining tasks include classification, clustering, association rule discovery, and regression.

The implementation of data mining ideas in high-performance parallel and distributed computing environments is becoming crucial for ensuring system scalability and interactivity as data continues to grow inexorably in size and complexity.

Databases are increasing continuously, in such a way that in many applications the source of data is physically distributed. Some examples of this phenomena are network intrusion data which has a large number of instances, and genomic data which has a high number of variables. The type of problems we treat in this work can be solved algorithmically in such away that parallelization can be done to reduce the

cost of solution. In Lozano's master thesis [Loz03] parallel computation was applied for computing two meta-classifiers, Bagging and Boosting. Both approaches were applied in kernel density classifier, achieving almost linear speedup. Most of this research is about the application of parallel computing in the solution of complex tasks in data mining and knowledge discovery involving large data sets. In this work we consider the following tasks concerning data mining: outlier detection, meta-classifiers, meta-clustering, and data visualization.

1.1 Parallel computing terminology

The following are terms used in parallel computing environment [WA99]:

- **Task.** It is a set of instructions that can be executed by a processor.
- **Sequential execution.** When the execution of a program is made by using a single processor one instruction at a time.
- **Parallel execution.** When the execution of the program can carry out many tasks simultaneously.
- **Synchronization.** It is a coordination of the parallel tasks in real time.
- **Granularity.** It is the rate between the size of computation and the size of communication. If the size of computation is less than the size of communication, then it is called fine granularity, otherwise it is called coarse granularity.
- **Speedup.** It is a performances measure between a system with many processors and a system with one processor. It is defined as:

$$\begin{aligned} S(n) &= \frac{execution\ time\ using\ one\ processor}{execution\ time\ using\ p\ processors} \\ &= \frac{t_s}{t_p} \end{aligned}$$

where t_s is the execution time using only one processor and t_p is the execution time using p processors. The maximum speedup that can be reached using n_procs is n_procs (linear speedup).

- **Scalability.** It is the ability that a parallel system has when more processors are used. This ability is influenced by the software and hardware.

1.2 Parallel Programming Models

- **Data parallelism.** In this approach the data is divided into different partitions. The same program runs in each partition, and the results are combined.
- **Task parallelism.** In this programming model, the task parallelism is specified explicitly by the programmer, who is responsible for the data partitioning, and communication between processors.
- **Shared memory.** Communication between processors can be simplified if they share a global memory. But, when the number of processors increase, it is difficult to control the conflict when the processors try to write to the same memory location.
- **Message passing.** Each processor has its own memory, and the processors communicate by explicitly sending or receiving messages, where messages are buffers of data with specific length.

1.3 Outlier Detection

Among the techniques to detect outliers [BL94, KN98] distance-based and density-based are the most powerful techniques. One of the major disadvantage of these techniques is that the order of complexity increases when the data becomes very large. We use parallel computing to reduce the order of complexity of these algorithms.

1.4 Data visualization

In order to have a better idea of the structure of a high dimensional data set, an effective visualization technique in either two or three dimensions is needed. Currently many visualization techniques are proposed, some of them are: RADVIZ [HGP01], parallel coordinates [Weg90], star coordinates [Kan01], and others. We propose an extension of the 2D star coordinate algorithm, in such a way that transformed instances can be visualized in two dimensions as a set of polygons and in three dimensions as a set of polyhedrons. Our enhanced star coordinate helps a data miner to detect the best features, identify clusters, and outliers, but with the difference that all data in high dimension is set into a one to one correspondence between data of two or three dimensions. The main advantage of this technique is that each point can be visualized uniquely. When the data sets are very large, parallel computing techniques are used to reduce the order of complexity of these algorithms.

1.5 Meta-classifiers

Meta-classifier algorithms are considered as ensembles of base classifier algorithms, since they improve the efficiency of single classifiers, and also because they are robust. We propose an ensemble to merge effectively five different base classifiers learned from centralized data sets. The results obtained with the proposed algorithm are compared with those obtained with other meta-classifier algorithms. Since meta-classifiers are computationally intensive, parallel computing techniques can help to reduce the order of complexity of the meta-classifier algorithms.

1.6 Meta-clustering

Like in supervised classification problems, ensembles of clustering algorithms offer robustness and accuracy when compared to single clustering algorithms. So there is

a necessity to design a meta-clustering algorithm to improve traditionally proposed algorithms. To reduce the computation time of the proposed ensemble of clustering algorithms, we design a parallel algorithm for our proposed ensemble algorithm and three other ensemble algorithms proposed in [DF03], [Lei99], and [SG02].

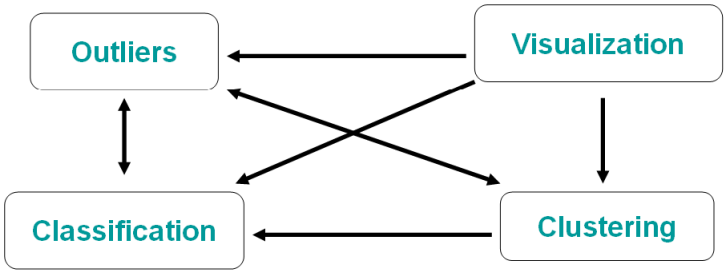

Figure 1.1: Interrelationship between the tasks considered in this thesis

The four data mining tasks mentioned before are chosen, because they are interrelated as shown in Figure 1.1. Visualization can be used not only as a data mining technique, but can also be an effective front-end tool in synergy with other data mining methods within a hybrid interactive and cooperative Knowledge Discovery in Databases (KDD) framework. Visualization plays an important role in the KDD process. At the beginning of a KDD process, Visualization can be used in the initial overall KDD planning, followed by data preprocessing, data mining, and representing results. Also, Visualization can be used in outlier detection, clustering, and classification. Classification algorithms can be used to find outliers. On the other hand, clustering algorithms can be used to find class labels, which is considered a preliminary step in supervised classification. Clustering algorithms can also be used to detect outliers. Removing outliers from data sets is a very important task, because outliers affect adversely the performance of the data mining algorithm.

This doctoral research satisfies the philosophy of CISE (Computing and Information Sciences and Engineering) doctoral program since it falls into the area of information science and scientific computing. The subjects of this thesis include intelligent data analysis, simulation, modelling, databases, data visualization, and

parallel and distributed computing.

In summary, the main contributions of this dissertation are the following:

i. The design of two parallel algorithms for outlier detection based on distance and density;

ii. The enhancement of the two dimensional star coordinates plot to two and three dimensions, and its parallel algorithm implementation, and

iii. The design of meta-classifier and meta-clustering algorithms, and their parallel algorithms.

In the next section, we briefly describe the software and hardware environment used in this research.

1.7 Software and hardware environment

The implementation of the serial and parallel programs proposed in this dissertation is in C++ in conjunction with C and Fortran 77 libraries. Message Passing (MPI) and Visualization toolkit (VTK) libraries are also used, both of them with C++ interface. This programming language and libraries are portable to different architecture environments. A more detailed specification of the software used is shown in the appendix. The computer platform used is two Linux-based clusters, one within the Mathematical Department of University of Puerto Rico (UPRM) and the other one from the High Performance Computing Facilities (HPCF) located in Rio Piedras, Puerto Rico. These clusters are well equipped with all libraries and compilers required to implement the serial and parallel programs for the algorithms proposed in this thesis. Both clusters are fully described in the appendix.

1.8 Algorithm Evaluation Criteria

The serial algorithms used in this work are well known data mining algorithms, and have been studied by many researchers. The proposed visualization algorithm is compared graphically with the traditional algorithm [Kan01]. We show that our proposed algorithm better illustrates high dimensional data sets.

The ensemble algorithms proposed give similar results to the existing ensemble algorithms. The comparison of our proposed algorithms with these ones is based on misclassification error rate and mutual information validation index. The proposed parallel algorithms correctly solve our proposed algorithms while reducing their computation time.

1.9 Thesis structure

This dissertation is composed of 7 chapters and one appendix, which are described briefly below:

In Chapter 1 we give a brief introduction to Data mining and parallel computing.

In Chapter 2 we give a brief introduction to outlier detection and review the most resent research done until now. Bay' s algorithm, and the local density outlier factor algorithm are also described. Finally, we make an experimental overview, and we describe the input parameters of these algorithms, and show their performance.

In Chapter 3 we provide an introduction to data visualization, and the description of the star coordinate algorithm proposed by Kandogan [Kan01]. The enhanced star coordinate algorithm, and its parallel version is also described. Finally the experimental results are shown.

In Chapter 4 we give a literature review, and an introduction to supervised classification. The five base supervised classification algorithms used in this research are described, namely: Decision trees (C4.5),radial basic functions (RBF), kernel

density (KD), K-nearest neighbors (KNN), and naive bayes (NB). Also we propose an ensemble method and its parallel version. The input parameters of the ensemble algorithm are described in the appendix.

In Chapter 5 we provide an introduction to clustering and a literature review of research done up to now. Similarity and dissimilarity measures, as well as the taxonomy of clustering algorithms are described. A description of five base clustering algorithms used in this thesis is given, namely: Partition around medoids (PAM), fuzzy C-means (FCM), gaussian mixtures (EM), density based clustering algorithm (DBSCAN), and balanced iterative reducing hierarchical clustering (BIRCH). Also ensemble clustering, and the clustering validation techniques are studied. Finally a parallel algorithm to compute two ensemble clustering algorithms together with the five base clustering algorithms are designed. In the experimental overview the performance of the ensemble of clustering algorithm is compared with the existing ones. The input parameter of this ensemble algorithm is described in the appendix.

In Chapter 6 we give the main conclusions. In Chapter 7 we list some open problems which arise from this work, and are proposed for future work. In the appendix we include a description of some data sets used in this research, followed by the description of hardware and software environments used in this work. VTK classes used in the implementation of the visualization algorithm are also described. Finally a description of implementation of the meta-classifiers, meta-clustering algorithms, input parameters, installation, libraries, and programs used for the implementation of these ensemble algorithms is given.

1.10 Ethics

Ethics refers to a set of attitudes, values, beliefs and habits that a person or a group displays. Computer ethics are important when it comes to issues related to the profession such as safety, environmental impact, and quality. The aim of ethics in science and engineering is to give future professionals the ability to recognize and solve ethical problems, to accept different ethical perspectives and ethical pluralism.

We are committed to ensure that our research is ethical, which includes protecting ourselves from being harmed, which can be physical, psychological, economical or spiritual. We are conducting this research within laws concerning intellectual property and privacy; therefore, we are protecting our individual and institutional interests.

Our parallel algorithms for detection of outliers can be used by other researchers to find outliers in large data sets. We provide our implementation to researchers that are interested in this specific area. These parallel algorithms were compared to the sequential ones giving the same results. In the same way, the data visualization system that we designed can be used by researchers to explore and find useful patterns in their specific data sets. We will provide our code upon request.

The meta-classifier and meta-clustering algorithms will be used as a tool for comparison with other classification and clustering process. The data sets used for validating our algorithms are available on the web (KDD and UCI repositories). They are public, and do not contain private or harmful data. These data sets are used by many people in the world. We use neither data collection techniques nor data reduction. We rigorously performed the tests and collected the data results, which is then presented as tables and figures.

Knowledge discovery allows considerable insight into data. This insight brings with it the inherent risk that may be inferred, private or ethically sensitive.

We must be careful in mentioning work that has been done and ideas proposed by different authors throughout this entire thesis; because we must avoid plagiarism in the whole sense.

We take care not to fall into ethical mistakes, using appropriate text or ideas, and mentioning previous works, and by also mentioning from where the motivations and principal sources in our thesis research come from.

There are two privacy problems of KDD, the input and the output problem. Sensitive data (in genetics, military, and financial fields) can be used as input to KDD methods. Academic and legal regulations must determine whether an analyst

may access especially sensitive data set and use KDD methods to analyze the data. If data analysis techniques are allowed for pre-existing databases, also KDD methods can be applied to these data sets. In this case, some methods to exclude the re-identification risk of a sensitive data set and preserving the statistical content of data as far as possible can be used to allow KDD methods to be applied in a modified data set. The output problem refers to the results of KDD applications, which can be used for making decisions. KDD ethics must surely be developed outlawing e.g. discrimination, manipulation, or watching of groups. Since ethics alone cannot exclude these applications, legal regulations may be needed.

Chapter 2

Distance and Density based Outlier Detection

2.1 Introduction

According to Hawkins [Haw80], *"An outlier is an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism"*. Almost, all the studies that consider outlier identification as their primary objective are in the field of statistics. A comprehensive treatment of outliers appears in Barnett and Lewis [BL94]. They provide a list of about 100 tests for detecting outliers in data following well known univariate distributions. However, real-world data are commonly multivariate with unknown distribution.

Detecting outlier instance in a database with unusual properties is an important data mining task. People in the data mining community have been interested in outliers. Knorr and Ng [KN98] proposed a non-parametric approach to outlier detection based on the distance of an instance to its nearest neighbors. Outlier detection has many applications, among them: Fraud detection and network intrusion, and data cleaning. Frequently, outliers are removed to improve accuracy of the estimators. However, this practice is not recommendable because sometimes outliers can have very useful information. The presence of outliers can indicate individuals or groups that have behaviors very different from a standard situation.

One might think that multivariate outliers can be detected based on the uni-

variate outliers in each feature, but this is not true, because an instance can have values that are outliers in many features, but the whole instance might not be a multivariate outlier. A basic method for detecting multivariate outliers is to observe the outliers that appear in the distribution of the Mahalanobis square distance of all instances. Rocke and Woodruff [RW02] stated that the Mahalanobis distance works well identifying scattered outliers. However, in data with clustered outliers the Mahalanobis distance measure fails in detecting some outliers, because it cannot deal well with the masking and swamping effects. In the Masking effect, an outlier masks a second one that is close by if the latter can be considered an outlier by itself, but not if it is considered along with the first one. In the Swamping effect, an outlier swamps another instance if the latter can be considered outlier only under the presence of the first one. The masking and swamping problem can be solved by using robust estimates of the centroid (location) and the covariance matrix (shape).

There are several methods for detecting multivariate outliers such as: Robust statistical based outlier detection [HR04, RW02, Rod04], outlier detection by clustering [HR04, KR90, RL87], outlier detection by neural networks [HC04], distance-based outlier detection [BS03, KNT00, NH94, RRS00], and density-based local outlier detection [BKNS00].

Hung and Cheung [**?**] propose parallel algorithms for mining distance-based outliers [KNT00, NH94]. They claim that their algorithm can be used to parallelize the density-based local outlier algorithm; but, they did not carry out any experiment. On the other hand, Ruoming and Agrawal [RA01] use locally the nearest neighbor property to parallelize the K-nearest neighbors classifier. In this thesis we use this property to parallelize Bay's algorithm, along with some strategies described by Skillircon [Ski01]. We also parallelize the density based local outlier factor algorithm.

This chapter is organized as follows: Section 2 focuses on distance-based outlier detection. Section 3 of this chapter considers density local-based. The experimental results appear in section 4.

```
Input: k: the number of nearest neighbors;
n: the number of outliers;
D: the dataset ordered randomly;
B: the size of blocks in which D is divided.
distance(x,y) is the Euclidean distance between x and y.
maxdist(x,Y) is the maximum distance between x and an example in Y.
Closest(x,Y,k) returns the k closest examples in Y to x.
Begin
1. c← 0 // the cutoff is initialized to 0
2.O ← ∅ // It is initialized to the empty set
3.while B← get-next-block(D){ // B loads a block from D
4. Neighbors(b)← ∅ for all b in B
5. for each d in D {
6.   for each b in B, b≠ d{
7.    if | Neighbors(b)| < k or
      distance(b,d) < maxdist(b,Neighbors(b)) {
8.     Neighbors(b)← Closest(b ,Neighbors(b)∪d,k)
9.     if (score(Neighbors(b),b)<c){
10.       Remove b from B
11.     }}}}
12. Top(B∪O,n)//Keeps only the top n outliers
13. c←min(score(o)) for all o ∈ O
14.}
15.Return O
end
Output: O, a set of outliers
```

Figure 2.1: Bay's Algorithm

2.2 Distance-based outlier detection

Given a distance measure in a feature space, two different definitions of distance-based outliers are the following:

1. An instance $\mathbf{x}$ in a dataset D is an outlier with parameters p and λ if at least a fraction p of the objects are at a distance greater than λ from $\mathbf{x}$ [KNT00], [NH94]. This definition has certain difficulties such as the determination of λ and the lack of a ranking for the outliers. Thus, an instance with very few neighbors within a distance λ can be regarded as strong an outlier as an instance with more

neighbors within a distance λ. Furthermore, the time complexity of this algorithm is $O(vn^2)$, where v is the number of features and n is the number of instances. Hence it is not adequate to use with datasets having a large number of instances.

2. Given the integer numbers k and n (k<n), the outliers are the top n instances with the largest distance to their k-th nearest neighbor [RW02]. One shortcoming of this definition is that it considers only the distance to the k-th neighbor and ignores information about closer points. An alternative is to use the greatest average distance to the k nearest neighbors. The drawback of this alternative is that it takes longer to calculate.

2.2.1 The Bay's Algorithm

Bay and Schwabacher [BS03] propose a simple nested loop algorithm. It tries to reconcile definitions 1 and 2. This algorithm gives near linear time performance when the data is ordered randomly and a simple pruning rule is used. The order of the algorithm in the worst case is quadratic. The algorithm is described in Figure 2.1.

The main idea in the algorithm is that for each instance in D, it keeps track of the closer neighbors found so far. If an instance is closer to its neighbors, it achieves a score lower than a cutoff parameter. Then, the instance is removed, because it can no longer be an outlier. Bay and Schwabacher use the score function as the sum of the distances to the k neighbors. Also, the average distance as well as the median distance can be considered. As more instances are processed the algorithm finds more extreme outliers and the cutoff increases along with pruning efficiency.

Bay and Schwabacher [BS03] show experimentally that their algorithm is linear with respect to the number of neighbors and that is almost linear with respect to the number of instances. Using six large datasets they find a complexity of order $O(n^{\alpha})$ where α varies from 1.13 to 1.32. Although, this reduction is favorable, it is still costly for large data sets. For this reason we propose to design a parallel algorithm.

2.2.2 Parallel Bay's Algorithm

We construct a parallel algorithm for Bay's algorithm based on the local nearest neighbors property [LA05]. Once the data is distributed uniformly among processes, each process computes its local neighbors and sends its results to the master process, which computes the global neighbors and then finds the top outliers. After that, it sends the cutoff parameter to each process. The proposed algorithm is given in the Figure 2.2.

```
Begin
1. c← 0 // It sets the cutoff for pruning to 0
2.O ← ∅ // It is initialized to the empty set
3.while B← get-next-block(D){//It loads a block from D
4. Neighbors(b)← ∅ for all b in B
5. for each d in LocalD {
  //Each process computes its local neighbors
6.   for each b in B, b≠ d{
7.     if | Neighbors(b)| < k or
    distance(b,d) < maxdist(b,Neighbors(b)) {
8.       Neighbors(b)← Closest(b,Neighbors(b)∪d,k)
9.       if (score(Neighbors(b),b)<c){
10.        Remove b from B
11.      }}}}
12.  Each process sends its local neighbors to the master
13. The master process {
14.  Computes global neighbors from local neighbors
15.  Computes top(B∪O,n)// keeps only the top n outliers
16.  Computes c←min(score(o)) for all o in O
17.  Broadcasts the cutoff parameter }
18.}
19.Return O
end
```

Figure 2.2: Parallel Bay's Algorithm

Bay claim that his algorithm runs in $O(N^2)$ in the worst case, in addition to $N/blocksize * N$ data access. Experimentally, using polynomial regression of empirical runtime, he found an exponent varying from 1.13 to 1.32. This exponent

was found from fitting $t = aN^b$; where t is the total time, and a and b are constants. We can suppose that our algorithm has similar complexity for our experimental data set. We denote this complexity as $O(N^\alpha)$; where $1.13 \leq \alpha \leq 1.32$. Therefore, the total time complexity of our parallel Bay' algorithm is $O((\frac{N^\alpha}{p} + n)t_{comp} + \frac{N}{p}t_{I/O} + N^\alpha k t_{comm})$, where t_{comp} is the computation time, $t_{I/O}$ is the I/O time and t_{comm} is the communication time, and p is the number of processes.

2.3 Density-based local outlier detection

For this type of outlier the density of the neighbors of a given instance plays a key role. Furthermore, an instance is not explicitly classified as either outlier or non-outlier; instead for each instance a local outlier factor (LOF) is computed. This factor gives an indication of how strongly an instance can be considered as an outlier. Breuning et al. [BKNS00] show through an example, the weakness of the distance-based method identifying certain type of outliers.

In order to formalize the algorithm, the following definitions are needed to detect density-based local outliers:

i. *k−distance* of an instance x. For any positive integer k, the *k−distance* of an instance x, denoted by *k−distance*(x), is defined as the distance d(x,y) between x and instance y ∈ D such that:

   (i) for at least k instances y' ∈ D–{x}, d(x,y') ≤ d(x,y),

   (ii) for at most k-1 instances y' ∈ D–{x}, d(x,y') < d(x,y).

ii. *k−distance neighbor of an instance x.* Given the *k−distance* of x, the *k−distance* neighborhood of x contains every instance whose distance from x is not greater than the *k−distance*; i.e.
   $N_{k-distance(x)}(x) = \{q \in D - \{x\} : d(x,q) \leq k-distance(x)\}$

iii. *Reachability distance of an instance x w.r.t. object y.* Let K be a positive integer number. The reachability distance of an instance x with respect to the

instance y is defined as

$reach{-}dist_k(x,y) = max\{k{-}distance(y), d(x,y)\}$

iv. *Local reachability density of an instance x.* It is given by:

$$lrd_{MinPts}(x) = \left[\frac{\sum_{o \in N_{MinPts}(x)} reach{-}dist_{MinPts}(x,o)}{|N_{MinPts}(x)|} \right]^{-1}$$

lrd is the average *reachability distance* based on the *MinPts*-nearest neighbor of the instance x.

v. *Local outlier factor of an instance x.* The local outlier factor of x is defined as:

$$LOF_{MinPts}(x) = \frac{\sum_{o \in N_{MinPts}(x)} \frac{lrd_{MinPts}(o)}{lrd_{MinPts}(x)}}{|N_{MinPts}(x)|}$$

The density-based local algorithm to detect outliers requires only one parameter, *MinPts*, which is the number of nearest neighbors used for defining the local neighbor of the instance. The LOF measures the degree to which an instance x can be considered as an outlier. Breuning et al. [BKNS00] show that for instances deep inside a cluster, their LOF's are close to 1 and should not be labelled as a local outlier. Since LOF is not monotonic, they recommended finding the LOF for each instance of the datasets using MinPts-nearest neighbor, where MinPts assumes a range of values from MinPtsLB to MinPtsUB. They also suggest MINPtsLB=10 and MinPtsUB=20. Once the MInPtsLB and MinPtsUB is found, the LOF of each instance is computed within this range. Finally, all the instances are ranked with respect to the maximum LOF value within the specified range, that is, the ranking of an instance x is based on:

$\max\{\text{LOF}_{MinPts}(\text{x}) \ / \ MinPtsLB \leq MinPts \leq MinPtsUB\}$

The LOF algorithm to detect density-based local outliers is shown in Fig. 2.3. Breuning et al. discuss in detail the time complexity of the LOF algorithm.

The serial LOF algorithm appears in Figure 2.3

Input:
klb and kub the lower and upper bounds of k-distance neighborhoods.
D is a data set of examples.
The number of top outliers
Output: lof, which is a vector with local density factors
kdis-neighbors(D,k) returns a matrix that containts the k-distance neighbors and their k-distances.
reachability(KDNeighbors) returns the local reachability density of each p in D
Begin
1.lof ←NULL
2.for each k in {klb,..., kub} {
3. KDNeighbors ← kdis-neighbors(D,k)
4. lrddata ← reachability(KDNeighbors,k)
5. for each p in KDNeighbors
6. templof[i] ← sum(lrddata[o∈ $N(p)$])/lrddata[i])/ $|N(p)|$
7. lof ←max{lof , templof}}
8.return top(lof)
End
Output: lof

Figure 2.3: The LOF Algorithm

2.3.1 Parallel LOF Algorithm

The major task to be carried out in the serial LOF algorithm relies on the computation of KDNeighbors (which is the matrix that contains the elements of D with its respective k-neighbors). For this reason we attempt to parallelize that step. Each process computes its respective KDNeighbors matrix, and then sends its result to the master process, which collects the results and then computes the reachability and local outlier factor. The proposed parallel algorithm [LA05] is given in Figure 2.4.

```
1.lof ←NULL
2.for each k in {klb,..., kub} {
3. Each process computes its respective KDNeighbors
and sends it to the master
4. The master process collects the partial KDNeighbors
and finds the KDNeighbors matrix
5. The master process {
6.   lrddata ← reachability(KDNeighbors,k)
7.   for each p in KDNeighbors {
8.     templof[i] ← sum(lrddata[o∈ N(p)])/lrddata[i])/ |N(p)|
9.   lof ←max{lof , templof}}
10.  }
11.return lof
```

Figure 2.4: Parallel LOF Algorithm

Parallel LOF algorithm uses the master slave paradigm. The master process sends the data set to the slaves, which calculate distances between its parts. Then the master process collects the results and finds the LOF factor for each observation.

The total upper bound of computation time of LOF algorithm is $Nvt_{I/O} + (kub-klb)((N^2v+2Nk)t_{comp})$. Where N is the number of instances, klb and kub are the lower and upper bounds, v the number of variables, k is the k-distance neighbors. Therefore, the parallel LOF algorithm has a total upper bound of $Nvt_{I/O} + (kup - klb)((N(\frac{N}{p}v + 2Nk)t_{comp} + Nkt_{comm}))$, where p is the number of processes.

2.4 Experimental results

We tested our algorithms using four datasets from UCI(Landsat, Shuttle, Census, Covtype). These data sets are described in the Appendix A.1. The cluster environment used are also described in the Appendix.

2.4.1 Description of Input and Output Parameters

For each $b \in B$ in Bay's algorithm (see line 4 of figure 2.2), we setup with long numbers (of order 10^6) for the initial k distances to the k neighbors. Also, we use

the following parameters: Block size = 1000, number of neighbors = 10, number of top outliers = 10. In The LOF algorithm, we set up k-lower bound = 10, k- upper bound = 20, number of top outliers = 10. For both algorithms we detect outliers only in the first class (class number one).

2.4.2 Speedup

In Figures 2.5 and 2.6 we show the speedups of our parallel algorithms for different number of processors and different data sets. The slowdown on Figure 2.5 for Covtype is mainly due to the high communication cost in the neighbors exchange phase. It occurs because the block size is small for our choice of parameters. For Landsat this slowdown also occurs because the data set is small, but for the other data sets, the speedup is almost linear. The speedup of parallel LOF (Figure 2.6) reaches linear speedup for all data sets.

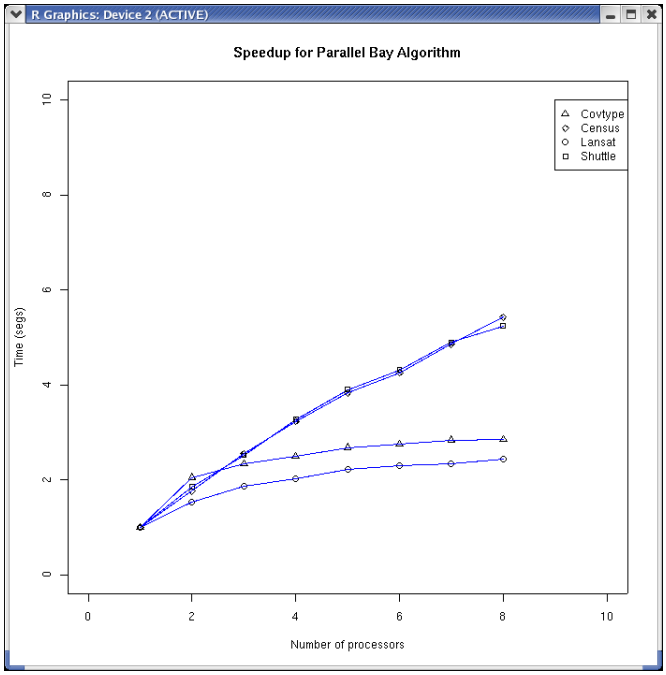

Figure 2.5: Speedup for Parallel Bay's Algorithm

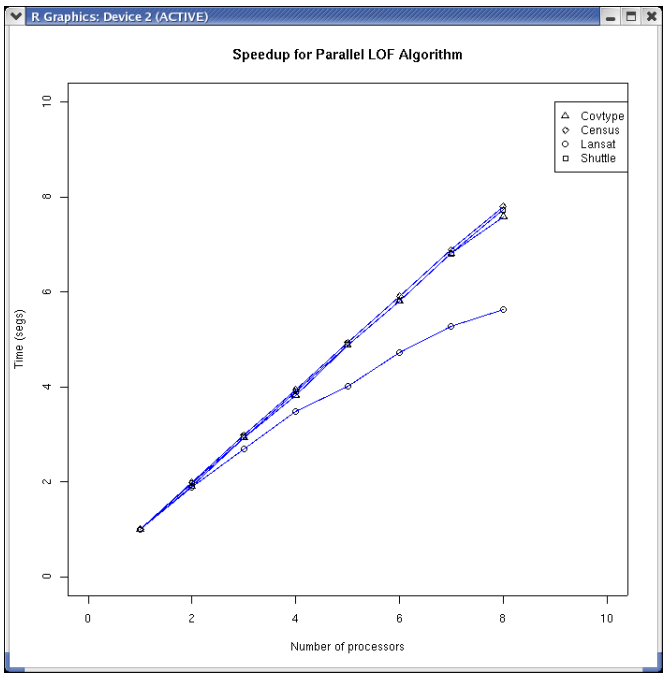

Figure 2.6: Speedup for Parallel LOF Algorithm

Chapter 3

Visualization

Data visualization plays an important role in the process of Data Mining, because our eyes learn more quickly and we can discover some useful patterns such as outliers, distributions, clusters, etc. There are many question about visualization like: How to effectively separate data in high dimension? How can we improve classification accuracy? To answer these questions many algorithms for high dimensional data visualization have been introduced, like RADVIZ [HGP01] and Parallel Coordinates [Weg90]. The latter visualization technique is one of the most widely used.

Kandogan [Kan01] introduced the star coordinates system, where the data is arranged in a circle with axes corresponding to each feature. The data representation is clear when it is used with scaling as well as axis rotation. Scaling the length of an axis allows one to see the contribution of a particular feature on the visualization display, whereas the axis rotation allows one to see the degree in which a particular feature is correlated with other features.

We propose to extend the two dimensional star coordinates algorithm to three dimensions, because the classical technique has some limitations and drawbacks. In order to improve this visualization, we introduce new parameters to make the transformation one to one. We visualize these parameters using polygons in two dimensions and polyhedrons in three dimensions, where the data point is visualized using its original values. The overlapped points can be visualized using a wire frame view or using different opacity levels. Our algorithm can be used for supervised and unsupervised classification. In supervised classification, we colour each data point

according to its pre classified label. In unsupervised classification we cluster similar polyhedrons in the same group. Outliers can be detected easily, because they have different polyhedral forms.

We implement our algorithm using the Tcl/Tk interface and C++ with the VTK [SML96] library, which is an open source visualization toolkit. Another important use of our algorithm can be to visualize outliers using outlier metrics based on distance and local density. Each point might be scaled according to its attribute values. The user can manipulate and display the dataset in different forms using Tcl/Tk interface and C++. These interfaces include several tools such as: zooming, panning, scaling, etc. We used the well known Iris data set from the UCI [BMNH] Machine Learning Repository to test our proposed algorithm. The outliers we find are compared with other outlier detection algorithms such as: LOF [BKNS00] and Bay [BS03] algorithm. When we want to visualize large data sets, the overall computation of the polyhedrons corresponding to each instance becomes computationally heavy. For this reason a parallel version of this algorithm is also introduced (like in Ahrens et al. [ALS+01]). The computation of each polyhedron is independent of each other. So, data parallelism technique can be used for this purpose.

This chapter is organized as follows: In Section 2 we introduce classical star coordinates. Section 3 introduces our enhanced algorithm of star coordinate to improve some limitations and drawbacks. The parallel algorithm and technical specifications are described in Section 4.

3.1 Star Coordinates

Kandogan [Kan01] introduced the star coordinates plot, where data is arranged in a circle with axes corresponding to each feature. His work was motivated from Bertin's permutation matrix which helps users to rearrange rows and columns to discover patterns and clusters from coarse graphical depiction of data. Another inspiration of his work was the parallel coordinates plot in which a vertical line is used for the projection of each dimension or attribute, with the maximum and minimum

values of each dimension usually scaled to the upper and lower boundaries on those vertical lines. A polyline made up of n-1 lines at the appropriate dimensional values connects the axes to represent an n-dimensional point. On the other hand, the star coordinate technique is quite similar to RADVIZ [HGP01], which maps a set of m dimensional points onto two dimensional space. This technique places dimensional anchors (dimensions) around the perimeter of a circle, and then spring constants are used to represent relational values among points - one end of a spring is attached to a dimensional anchor, the other is attached to a data point. The values of each dimension are usually normalized from 0 to 1. Each data point is displayed at the point where the sum of all spring forces equals zero. The position of a data point depends largely on the arrangement of dimensions around the circle.

The star coordinates technique arranges the coordinate axis in a circle on a two dimensional plane with equal angles between the axis and origin at the center of the circle. The points are scaled according to the length of the axis. The mapping of a n dimensional point in a two dimensional Cartesian coordinate, is determined by the sum of all unit vectors in each coordinate multiplied by the value of the data element for that coordinate.

$$P_j(x,y) = (o_x + \sum_{i=1}^{n} u_{xi}(d_{ji} - min_i), o_y + \sum_{i=1}^{n} u_{yi}(d_{ji} - min_i))$$

where d_{ji} is the jth data with ith value, min_i is the minimum value of scaled values in each coordinate, u_{xi} and u_{yi} are the unit vectors in each coordinate direction, and (o_x, o_y) is the origin of the coordinate system. In figure 3.1 an example of calculation of one data point location in 8 dimensional dataset appears.

Kandogan applies transformations to start coordinates, such as scaling and rotation transformation. The data representation is clear when it is used with scaling as well as axis rotation. Scaling the length of an axis allows one to see the contribution of a particular feature on the visualization display. On the other hand, the axis rotation allows one to see the degree in which a particular feature is correlated with other features.

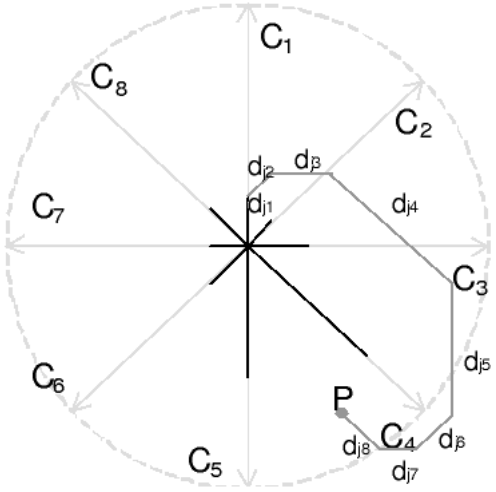

Figure 3.1: Star coordinate transformation from 8 dimensional data set [Kan01]

3.2 3D star Coordinates

We propose an extension of star coordinates to three dimensions, where we can use rotation with axis scaling to find some patterns that do not appear on the two dimensional star coordinate plot.

The 3D star coordinate plot is the result of a non-linear transformation, which maps high dimensional data to a three-dimensional coordinate system. Let $O(x, y, z) = (o_x, o_y, o_z)$ be the origin of the system, and $A_n =< a_1, a_2, ..., a_n >$ be a sequence of n three dimensional vectors representing the axis (cosine directors in three dimensions). These axes follow the direction of the three dimensional unit vectors $\overrightarrow{u_i} = (u_{xi}, u_{yi}, u_{zi})$, $i = 1, ..., n$.

The 3D star coordinates plot arranges the coordinate axes in a three dimensional sphere, with equal angles between the axes, and same origin at the center of the sphere. The mapping of n-dimensional data point (D_j) to a three dimensional data point (P_j) is determined by the sum of unit vectors mentioned above.

$P_j(x, y, z) = (o_x + \sum_{i=1}^{n} u_{xi}(d_{ji} - min_i), o_y + \sum_{i=1}^{n} u_{yi}(d_{ji} - min_i), o_z + \sum_{i=1}^{n} u_{zi}(d_{ji} - min_i))$

where: $D_j = (d_{j1}, d_{j2}, ..., d_{ji}, ..., d_{jn})$, $|\overrightarrow{u_i}| = \frac{\overrightarrow{|a_i|}}{max_i - min_i}$ $min_i = min\{d_{ji}, 0 \leq j < |D|\}$, $max_i = max\{d_{ji}, 0 \leq j < |D|\}$

Let $S : R^n \rightarrow R^p$ (where $p \leq n$) be the p-dimensional star coordinate projection. We discuss the relation between the distances $|\overrightarrow{x_1}$ - $\overrightarrow{x_2}|_1$ and $|S(\overrightarrow{x_1}) - S(\overrightarrow{x_2})|_1$. We choose norm L_1 to measure distances between high dimensional vectors, because it is more robust than the L_2 norm [Agg01].

$$|\overrightarrow{x_1} - \overrightarrow{x_2}|_1 = |x_{11} - x_{21}| + ... + |x_{1n} - x_{2n}|$$

$$\begin{aligned} |S(\overrightarrow{x_1}) - S(\overrightarrow{x_2})|_1 &= \sum_{j=1}^{p} |\sum_{i=1}^{n} u_{ji}(x_{1i} - x_{2i})| && (3.1) \\ &\leq \sum_{i=1}^{n} (\sum_{j=1}^{p} |u_{ji}|)|x_{1i} - x_{2i}| && (3.2) \\ &\leq |\overrightarrow{x_1} - \overrightarrow{x_2}|_1 && (3.3) \end{aligned}$$

The last inequality is right, because for each $i = 1, ..., n$, $\sum_{j=1}^{p} |u_{ji}| \leq |\overrightarrow{u}_i|_1$

From this result, we can see that the distance in L_1 of the images of two n-dimensional vectors is less than or equal to the distance of the vectors. Therefore, two points in high dimension are projected to p dimensions preserving some class of similarity (at least in norm L_1). In other words, two close points cannot be projected to different locations, because, their distance cannot be larger than the distance of the original vectors. The last inequality does not hold if we choose the Euclidean norm.

On the other hand, we choose to visualize the data points in three dimensions, because there are fewer overlapped points compared to two dimensions. For example the set of points $\{x_j = (x_{j1}, x_{j2}, x_{j1} + c, x_{j2}) : \ x_1, x_2 \in R\}$ is mapped to $(-c, 0)$ using two dimensional star coordinates, but using three dimensional star coordinates system this set of data points is mapped to collinear points.

The overlapping problem continues in three dimensions. For instance, the pair of points $x_1 = (x_{11}, x_{12}, x_{13}, x_{14})$ and $x_2 = (x_{21}, x_{22}, x_{23}, x_{24})$, with $x_{11} = x_{21}, x_{13} =$

$x_{23}, x_{12} = x_{14} + x_{22} - x_{24}$ are mapped to the same point using star coordinates in three dimensions.

To avoid this overlapping problem we introduce new parameters like in [TK99a] to represent uniquely transformed points.

$$P_i = \frac{cP + x_i u_i}{c + x_i}, i = 1, ..., n$$

where P is the projected point by the three dimensional star coordinate transformation. When $c \to 0$, then $P_i \to u_i$; therefore the parameter P_i does not depend on the projected point. On the other hand, if $c \to \infty$, then $P_i \to P$. In order to scale the size of each polyhedron, we vary c from 1000 to 10000.

3.3 Algorithm for 3D star coordinates

We choose to visualize the transformed points using polyhedrons in three dimensions and polygons in two dimensions. Similar shapes represent similar points. The main reason for this choice is to visualize the projected points using their original values. Another reason is that the computation of a polyhedron is less expensive than the computation of spheres around the point (this technique was introduced by Theisel [TK99a]).

The algorithm is as follows: (the VTK classes used in this algorithm are described in the Appendix).

For each observation, a three-dimensional star coordinate transformation is applied. The result is added as input to an instance of the vtkPoints class; next it is added to an instance of the vtkVoxel class, and its result is added to an instance of the vtkUnstructuredGrid class. Finally, it is added as input to an instance of the vtkAppendFiler class, which collects these objects. The output of this collection of objects is added to an instance of the vtkRenderer class. The result is added as an instance of the vtkRenderWindows class. Finally the geometry generated is visualized on the screen.

The pipeline of the proposed algorithm is given in Figure 3.2.

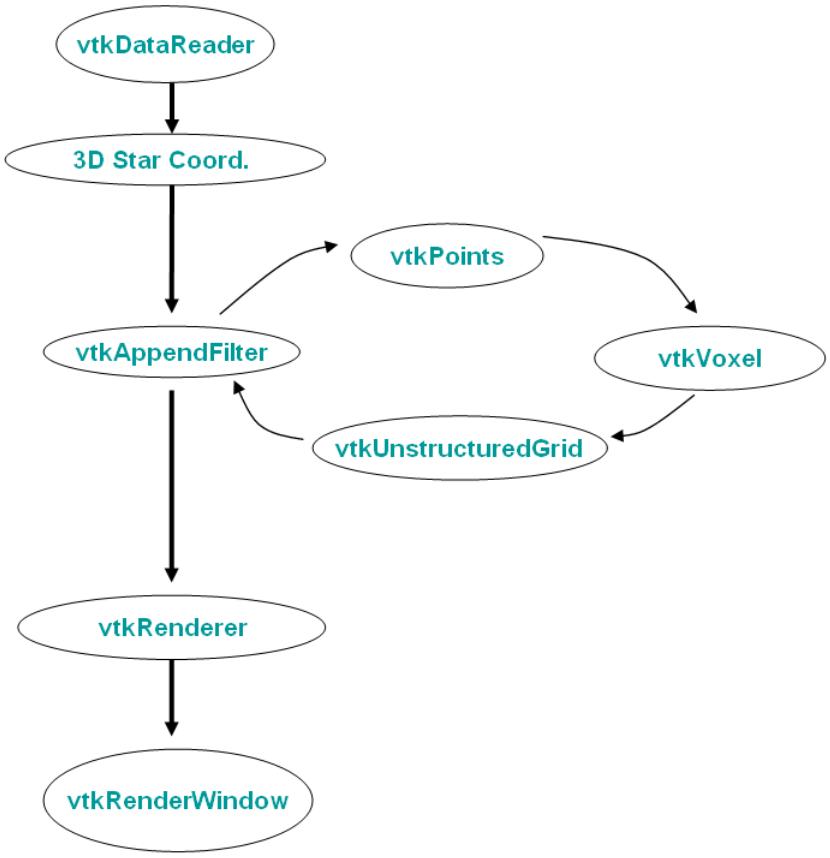

Figure 3.2: Pipeline of the 3D star Coordinate algorithm.

When many points are mapped to the same point the largest polyhedron hides the other ones. The hidden polyhedrons can be visualized using a wire frame view or using different opacity (Figs. 3.4, 3.3, and 3.5). The opacity of a polyhedron is chosen inversely proportional to the sum of their coordinates. A data point with a large coordinate sum has low opacity.

3.4 Parallel algorithm for 3D star coordinates

In order to assure the scalability of this algorithm to large data sets we introduce a parallel algorithm for the proposed three-dimensional star coordinates. This parallel algorithm is based on data parallelism, because the computation of each each

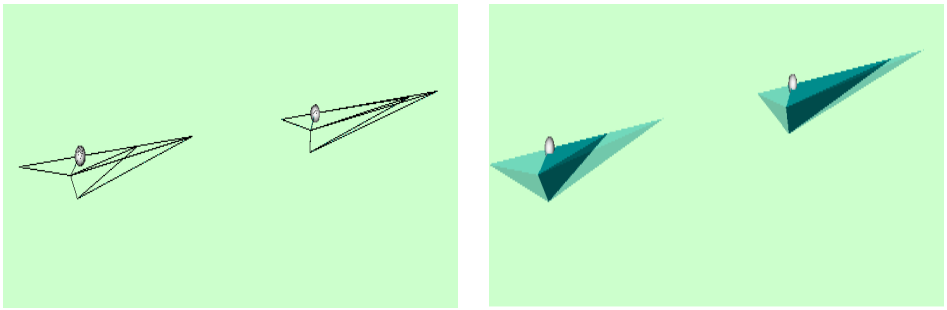

Figure 3.3: a) Wire frame view in three dimensions. b) Polyhedrons with different opacity in three dimensions.

polyhedron is independent of every other.

The proposed parallel algorithm is as follows:

The number of instances is partitioned according to the number of processes. Each process applies a three dimensional star coordinate transformation to its respective observations. Its result is added as input to an instance of the vtkPoints class, next it is added to an instance of the vtkVoxel class, and its result is added to an instance of the vtkUnstructuredGrid class. Finally, it is added as input to an instance of the vtkAppendFiler class, which collects these objects. Each slave process uses ports to send its vtkAppendFilter objects to the master process. Each slave process adds its vtkAppenFilter object to an instance of the vtkOutputPort class to send to the master process, which receives the results as an instance of the vtkInputPort class, and collects each of them in an instance of the vtkAppendFilter class. Finally, the master process adds the output of the vtkAppendFilter object as input to an instance of the vtkRender class, and consequently it is added to an instance of the vtkRenderWindows class, which visualizes the data geometry generated on the screen.

In summary, the parallel algorithm is like this:

i. Each process computes its respective star coordinate transformation and builds its polyhedrons.

ii. Each process sends its work to the master process, which aggregates and dis-

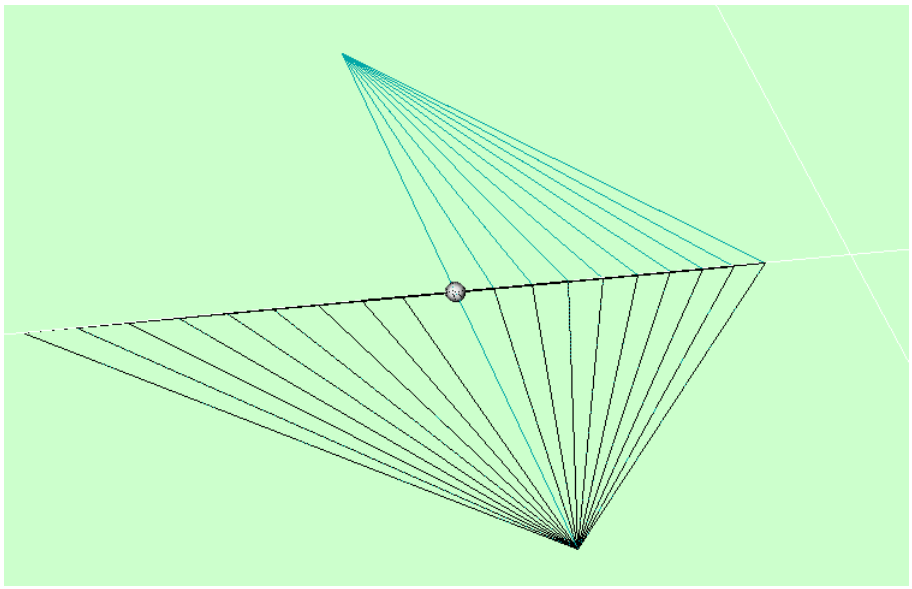

Figure 3.4: Wire frame view of two overlapping points.

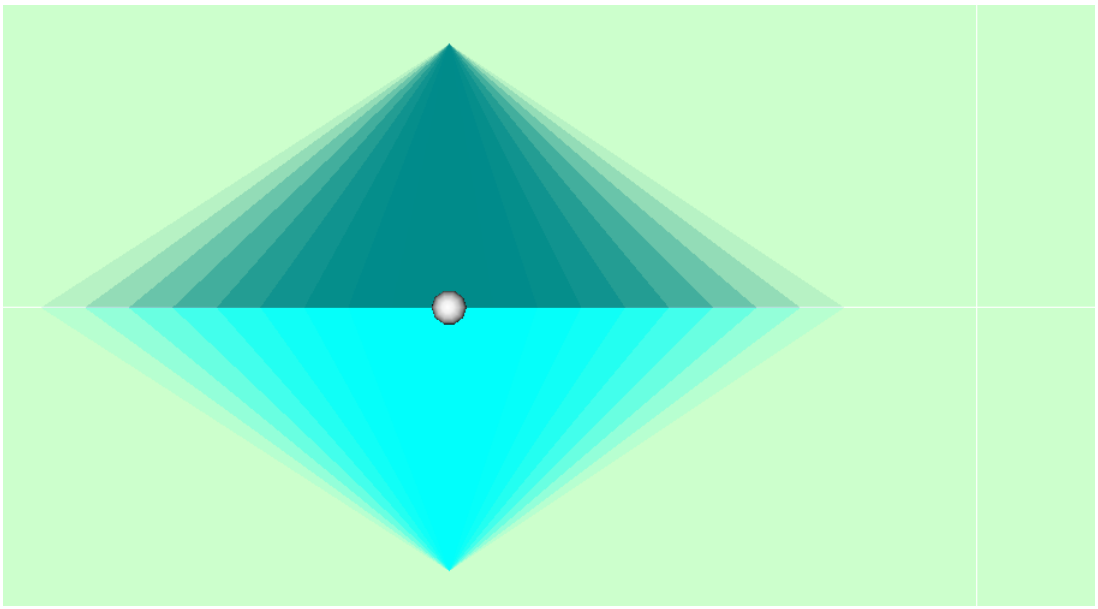

Figure 3.5: Polygons with different opacity.

plays the visualization.

The pipeline sequence of the parallel three dimensional star coordinate algorithm is shown in Figure 3.6.

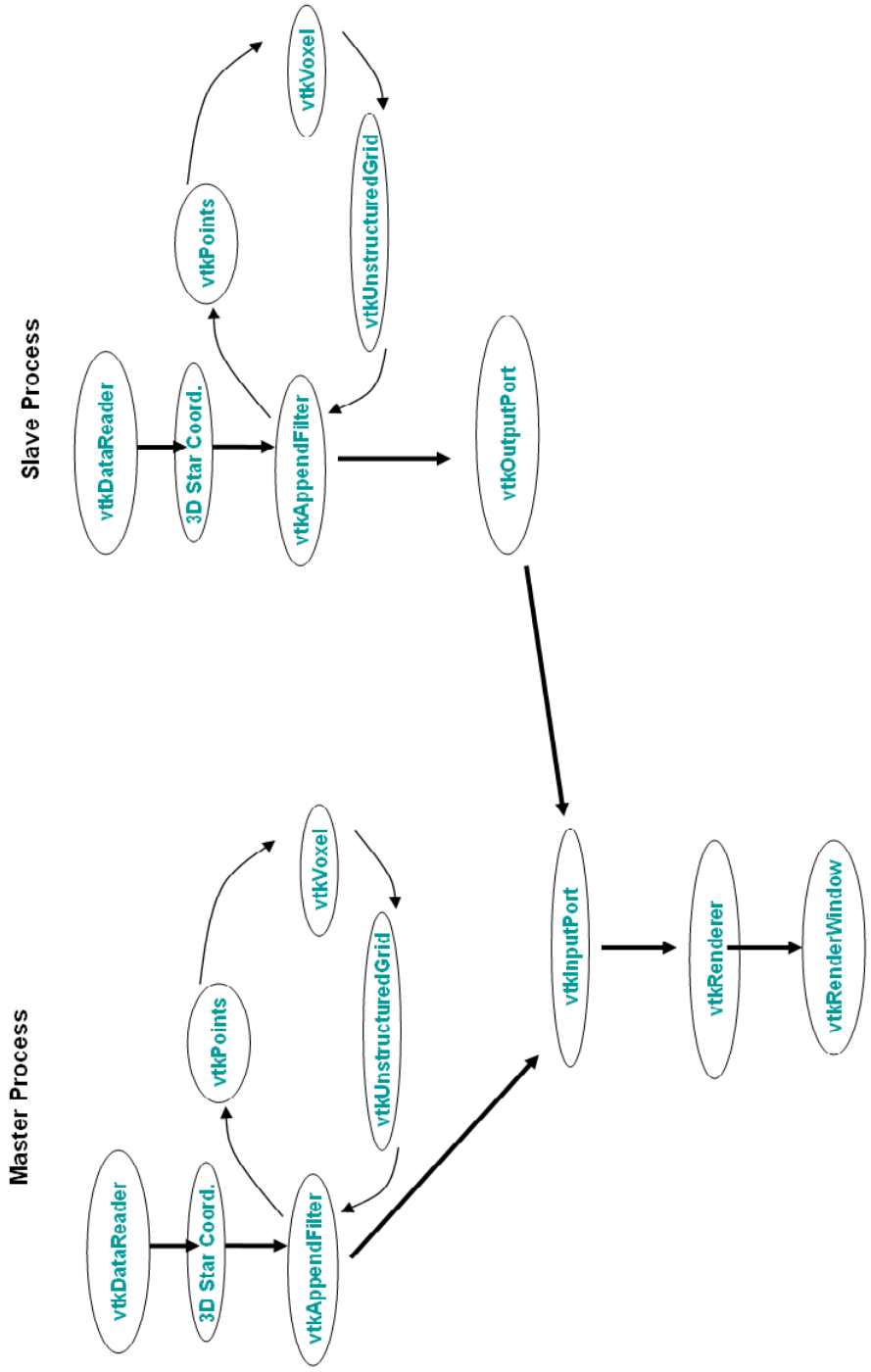

Figure 3.6: Pipeline of the parallel 3-D star Coordinate algorithm.

3.5 Experimental Methodology

Our visualization method was tested using the well-known Iris dataset, which is available at the UCI Machine Learning repository. This dataset contains 150 instances; each of them having 4 attributes and one column of class labels. A display of the 2D star coordinates plot is shown on Figure 3.7 .

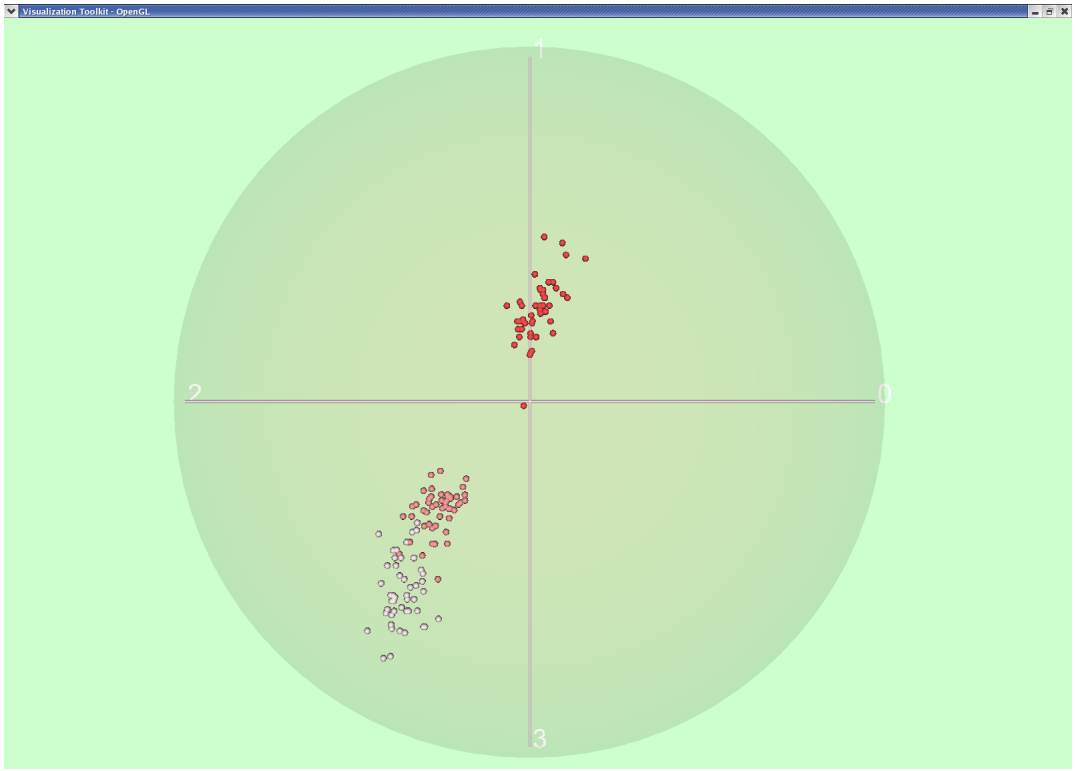

Figure 3.7: 2D star coordinates plot of Iris dataset

The unit vectors are located in a unit sphere. They have the following cosine directors: $u_{ix} = sin(\phi)cos(\theta), u_{iy} = sin(\phi)sin(\theta), u_{iz} = cos(\phi)$, $\theta \in [0, 2\pi]$ and $\phi \in [0, \pi]$.

The visualization was carried out on a Linux-based cluster within the *"High Performance Computing Facility"* (HPCF) located at Rio Piedras, Puerto Rico (see Appendix for a description of the cluster and the software needed to run the application).

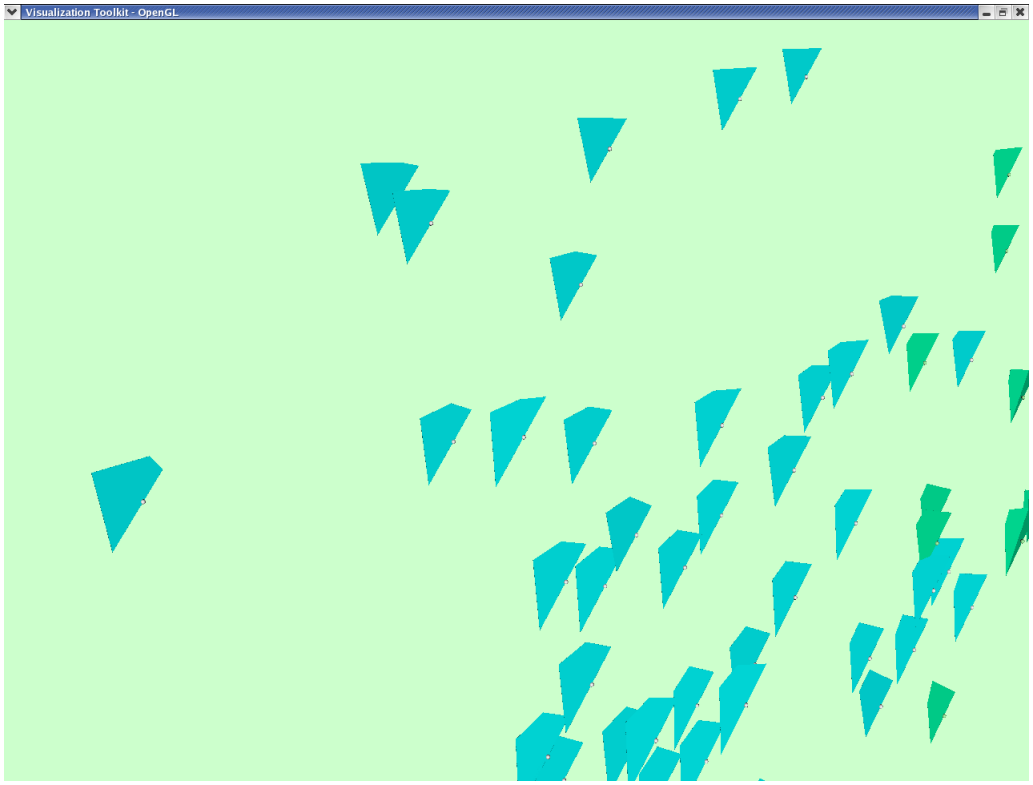

Figure 3.8: 3D zoom of Iris data set

We can distinguish objects in detail, not just as simple points, but as polyhedra. The shape of this objects is directly proportional to the normalized attribute value in its respective dimension. With this representation we can differentiate objects mapped from high dimension to either two or three dimensions.

The navigation and interaction on the visualization scene is dynamic and interactive. We can visualize objects from any angle using rotation. Also, it helps to see patterns that are not shown in a traditional star coordinates display. Rotation and zooming help to identify which instances are outliers and which of them are in groups. Observations with different attribute values look different on the visualization scene (see Figures 3.8 and 3.11).

The process of classification and/or detection of outliers are made by the user, who is responsible for this task. For example in figure 3.11, the observation 42 is considered an outlier, because it is away from the others, and has different shape and

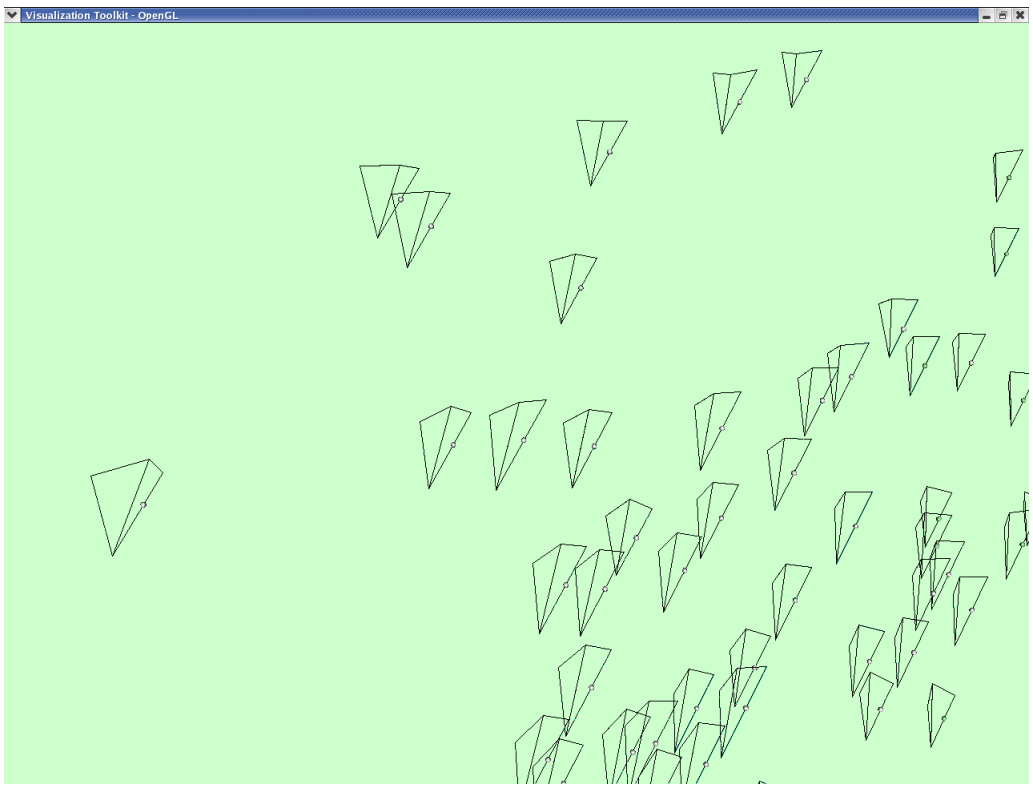

Figure 3.9: 3D zoom of Iris data set on wire frame view

form. A similar result is also achieved using outlier detection algorithms such as: density based method (LOF algorithm) and distance based method (Bay algorithm). Also, from Figure 3.10 we can see that the second feature seems to be the most relevant and the features 3 and 4 are highly correlated.

The computation time of the parallel algorithm includes the I/O time, the computation of the star coordinate transformation, and the creation of the visualization objects until the rendering process. The run time and the speedup of the parallel algorithm for the Shuttle dataset is shown in table 3.1. The parallel algorithm based on data parallelism gives good results. This parallel algorithm does not reach linear speedup because of the following reasons: 1) The data movement between processes at the beginning of the algorithm. 2) The communication cost of the vtkAppendFilter objects. 3) The final rendering process takes considerable computational time, and it is done by one process only.

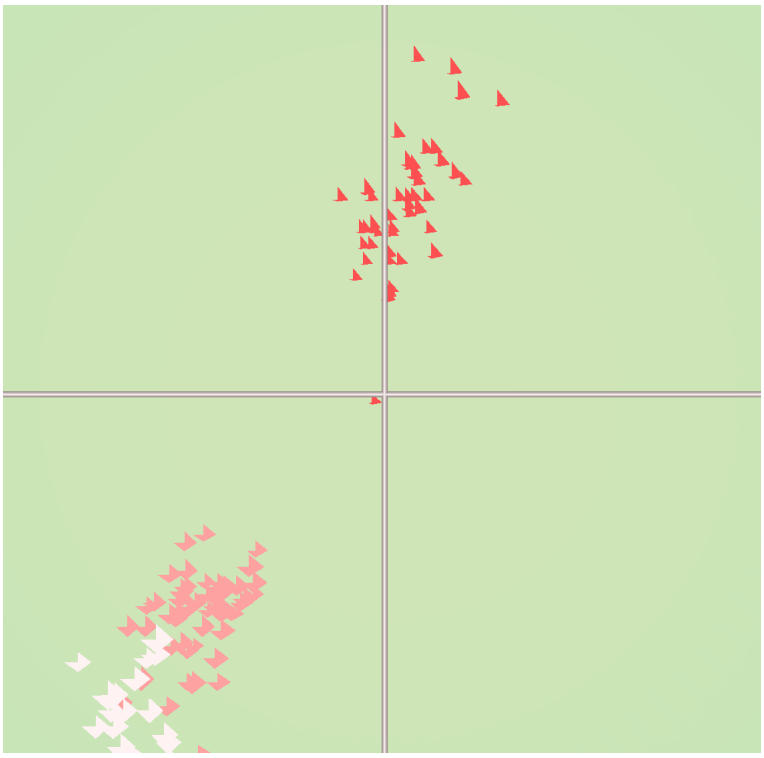

Figure 3.10: 2D star coordinates of Iris data set with polygons

Table 3.1: Running time (sec.) and speedup of parallel 3 D star coordinate algorithm for the Shuttle data set)

# of processors	1	2	3	4	5	6	7	8	9	10
Run time (sec.)	48.50	25.83	17.54	13.5	12.8	12.2	11.7	11.3	11.0	10.8
Speedup	1.00	1.88	2.76	3.60	3.79	3.96	4.15	4.30	4.41	4.50

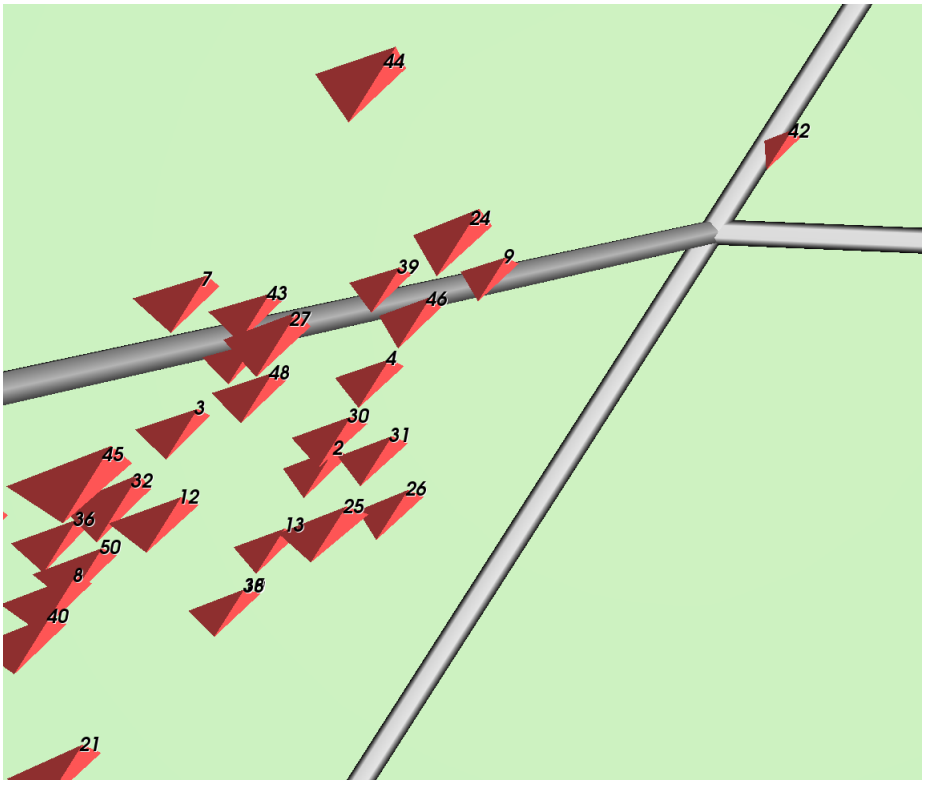

Figure 3.11: 3D zoom of the first class of Iris data set

Chapter 4

Meta-classifiers

Data mining and knowledge discovery play an important role in engineering, scientific, and medical databases, which are reaching gigantic proportions and require both large memory and disk usage, and high speed computing. Data mining will continue to grow in size and complexity. Thus, *Distributed Computing* is needed for ensuring system scalability and interactivity. Distributed Computing for Data Mining refers to designing parallel algorithms for Data Mining tasks which can be mapped efficiently onto parallel computers. The goal is to design and develop algorithms and methods that scale to thousands of attributes and observations.

Data gathering and data warehousing has generated large databases, for instance, medical institutes, financial bank institutions, mega market chains such as Sears, Wall Mart, etc. They have millions of records in their databases ready to mine, and to extract useful information about these databases. Mining information from such databases regards issues such as: (1) data may be in several files as in multi-relational databases, (2) the data set may be spread across several disk or different geographical locations, (3) the statistical data may vary widely, and (4) Data privacy. Usually, learning algorithms are computationally complex and require all data to be memory resident. In some cases the data sets are distributed and cannot be localized in one machine for various reasons such as security, competitive reasons, network bandwidth, storage costs, etc. On the other hand, one can try to assemble the data set in a single file, and then a specific algorithm can be used to sub-sample this file, but useful information might be lost.

4.0.1 Literature review

Ensembles of multiple classifiers [TG00], are found in several fields such as combining estimators in econometrics, evidence in rule-based systems, and multi-sensor data fusion.

Experiments with classifier combining rules are described by Duin and Tax [DT00], and Dietterich [Die00]. The latter uses various fixed and trained combining rules. Six methods for fusing multiple classifiers are presented by Roli et al. [RGV01]. They measure classifiers's diversity and performance of such methods. Prodromidis et al. [PCS00] computed a global classifier from large and inherently distributed databases. It was accomplished by computing multiple classifiers and applying learning programs to a collection of independent and inherently distributed databases.

Meta-learning refers to learning from prediction of base classifiers in a common validation data set. The sequence of this process is as follows:

i. Classifiers are trained from the initial training sets.

ii. Prediction is generated by the learned classifiers in a separate validation set.

iii. A meta-level training set is composed from the validation set and the prediction generated by the classifiers in the validation set.

iv. The final classifier (meta-classifier) is trained from the meta-level training set.

Valentini and Masulli [VM02] cite two general ensemble methods, which are the following:

i. Non-generative methods such as: majority voting, weighted voting, Bayes rule, rules based on the Bayes approach, order statistic operators, minimum, maximum, simple average, product, and weighted average.

ii. Generative methods such as: Bagging [Bre94], Boosting [FS96], cross validation, feature selection, random subspace method, test and select methods, randomized ensemble methods, and linear combiners.

Roli et al. [RGV01], and Tumer and Ghosh [TG00], presente and analyze combiners based on order statistics. They conclude that the combiner's robustness helps to improve the performance of certain individual classifiers. Their experimental results shows that, if there is significant variability among the classifiers, the order statistics-based combiners substantially outperform simple combiners.

Prodromidis et al. [PCS00], introduced a meta-classifier system called JAM (Java agents for meta-learning). They address the efficiency and scalability problems by employing distributed and asynchronous protocols at the architecture level; evaluating and combining only the most essential classifiers. In order to achieve portability across heterogeneous platforms they built a JAM upon existing agent infrastructure available on the internet. Compatibility is obtained by special bridging agents to resolve any differences in the schemata among the distributed databases. Adaptivity is attained by extending the meta-learning techniques to combine both existing and new classifiers. Extensibility is ensured by decoupling JAM from the learning algorithms and by introducing plug and play capabilities through objects.

Duin and Tax [DT00] conclude that combining classifiers trained in different feature sets is very useful. Especially, when the probabilities are well estimated by the classifier in these feature sets. On the other hand, combining different classifiers trained in the same data set may also improve the classifier performance, but it is generally less useful. They conclude that there is no combiner winning rule such as: mean, median, majority in case of correlated errors, and the product of independent errors perform roughly as expected, but others may be good as well. The divide and conquer strategy as well as the independent use of separate feature sets perform efficiently. Difficult data set should not be thrown away, because it might contain important information. The use of randomly selected feature sets appears to give a very good result in their examples, especially for the Bayes-normal-1 classifier. The Nearest neighbor algorithm appears to be useful and stable when used as combiner. They suggest re-mapping the posterior probabilities to distances for the trained combiners, and combining results of combination rules in the different feature-sets.

Dietterich [Die00] concludes that in low-noise cases, Adaboost [FS96] gives a good performance, because it is able to optimize the ensemble without over fitting. However, in high-noise cases, Adaboost puts a large amount of weight on the mislabeled examples, over-fitting badly the classifier. Bagging and Randomization perform well in both the noise and noise-free cases, because they focus on the statistical problem (over-fitting), and noise increases this statistical problem. In very large data sets, randomization can be expected to do better than Bagging, because bootstrap replications of a large training set are similar to the training set itself, hence the learned decision tree may not be very diverse. Randomization creates diversity under all conditions, but at the risk of generating low-quality decision trees. He was interested in seeing if the local algorithms (such as radial basis functions and nearest neighbors methods) can be profitably combined via Adaboost to yield interesting new learning algorithms.

4.0.2 Our Work

In this thesis we introduce a new meta-classifier algorithm based on the combination of two ensemble methods: one of them coming from non-generative methods and the other coming from generative methods. We also compare the accuracy of five well known base classifiers (Radial Basis Function networks, Decision trees C4.5, Kernel Density, Naive Bayes, and K-nearest neighbors). We carry out single Bagging for each of them and their combined Bagging and compare their performance based on the classification error rate. Also, we design a parallel algorithm for our proposed meta-classifier algorithm.

4.0.3 Motivation

Some benefits of Meta-learning are the efficiency of executing in parallel the base-learning processes (each one implemented in a serial program) on a subset of the training data set. Thus we can use the same code without the time consuming process of parallelizing it, and we can learn from a small subset of data that fits

in main memory. On the other hand, distributed data mining systems are oriented to discover and combine information that is distributed across multiple databases. Data mining requires extensive research in all issues concerned to mining large and distributed databases, and there are still open problems in this area.

4.0.4 Organization

This chapter is organized as follows: In Section 1 we introduce some definitions about supervised classification. Section 2 describes the five classifiers used in this thesis. Ensemble methods, and the proposed algorithm with its corresponding parallel algorithm are described in Section 3. Finally, applications and results are presented in Section 4.

4.1 Supervised Classification

Given a finite set of classes $G_1, G_2, ..., G_g$, known a priori, and a p dimensional input vector $\boldsymbol{x}$, the classification problem deals with finding the relation between the values of $\boldsymbol{x}$ and its membership to a group G_i. The probability that a random instance from group G_i is assigned to G_j, $j = 1, 2, ..., g$ by the classifier C is defined by

$$\begin{aligned} e_{ij}(C) &= Prob\left[C(\boldsymbol{x}) = j | \boldsymbol{x} \in G_i\right] && (4.1) \\ &= \int_{R_j} f_i(\boldsymbol{x}) d\boldsymbol{x} \ , && (4.2) \end{aligned}$$

where f_i is the i-th class density function and, $R_j = \{x : \ C(x) = j\}$. Given any member randomly selected from a group G_i, its probability of misclassification is given by

$$\begin{aligned} e_i(C) &= \sum_{i \neq j}^{g} e_{ij}(C) && (4.3) \\ &= \int_{\bar{R}_i} f_i(\boldsymbol{x}) d\boldsymbol{x} && (4.4) \end{aligned}$$

where $\bar{R}_i$ is the complement of R_i $(i = 1, 2, ..., g)$. The error rate of the classifier C is

$$e(C) = \sum_{i=1}^{g} e_i(C)\pi_i \tag{4.5}$$

where π_i represents the prior probability of class G_i.

The training sample is defined as a matrix $\pounds$ whose elements are of the form $(\boldsymbol{x}_j, y_j)$, $(j = 1, ..., n)$. where $\boldsymbol{x}_j$ is a p dimensional observation and y_j is its respective label. The training sample is used to build the classifier.

The object $\boldsymbol{x}$ is assigned to class G_i, if $\pi_i f(\boldsymbol{x}|G_i) > \pi_j f(\boldsymbol{x}|G_j)$ for all $j \neq i$, where $f(\boldsymbol{x}|G_i)$ represents the $i - th$ class conditional density.

The most common estimators of the error rate $e(C)$ are:

- Apparent error. The training sample is used for building as well as for testing the classifier. The error estimate is the proportion of the misclassified instances in the training set.

- Error estimation using a test sample. Given a set of instances, a classifier is built with a percentage of instances and the remaining ones form the test sample. The error estimation is the proportion of misclassified instances in the test sample.

- Cross validation Estimation. In this case, the set of instances is partitioned in m sub-samples. For the $i-th$ $(i = 1, ..., m)$ sub-sample, the classifier is build with the rest of the sub-samples, and then this $i-th$ sub-sample is tested with this classifier. Finally, the error estimation is the average of classification errors for each sub sample.

- Bootstrapping error estimation. The error estimation is the average of classification errors produced by each bootstrap sample (a sample drawn with replacement).

4.2 Base Classifier algorithms

In this thesis, we use the following five classifiers in order to build a meta-classifier algorithm. These are the C4.5 decision trees and Naive Bayes from Machine Learning, Kernel density from statistics, radial basis functions from neural networks, and K-nearest neighbors. We choose these classifiers based on the following reasons: 1) We want to combine classifiers coming from different fields like Statistics and Machine Learning. 2) These classifiers have been studied by many researchers, and they are the most popular classifiers, because they give good results. These algorithms are described below:

4.2.1 The C4.5 algorithm

It is a decision tree algorithm introduced by Quinlan [Qui93]. Given a training sample with known labels; this algorithm constructs a decision tree. At each node a test for each attribute is made. Finally, given an input vector $\boldsymbol{x}$ (unlabeled) the decision tree determines the class to which it is assigned. The following steps explain the C4.5 algorithm.

Let T be the training sample and let $C_1, C_2, ..., C_k$ be the set of possible classes of the instances in T. The decision tree construction is as follows:

i. If T contains instances belonging to a single class, then the decision tree for T is a leaf identifying a class C_j.

ii. If T does not contain any samples, then T is a leaf.

iii. If T contain instances with a mixture of classes. T is partitioned into T_i (the total number of classes). The decision tree of T consists in a decision node.

iv. The same process of tree construction is applied recursively to each non leaf node.

In step iii, the criterion to select an attribute is given by the info-gain measure

$$Info(T) = -\sum_{i=1}^{k}((freq(C_i, T)/|T|)log_2(freq(C_i, T)/|T|)$$

$$Info_X(T) = -\sum_{i=1}^{n}((|T_i|/|T|)Info(T_i))$$

$$Gain(X) = Info(T) - Info_X(T)$$

where X is an instance vector. For discrete attributes, frequencies are used to find the maximum info-gain. For continuous attributes binary tests $Y \leq Z$ and $Y \geq Z$ are defined (Z is a threshold and Y is an attribute value). In the last case the training instances are first sorted; there are only m-1 possible splits (m is the number of different attribute values in the test node), each of them should be examined. Finally the value with the highest information gain is selected.

The time complexity of building a decision tree is $O(dfNlogN)$, where N is the sample size, f is the number of features, and d is the number of nodes.

4.2.2 Radial Basis Function Networks

The radial basis functions (RBF neural networks) are defined as the combination of radially symmetric linear basic functions [Bis95]. These functions transform an input $\boldsymbol{x} \in R^p$ into an C-dimensional space

$$g_j(\boldsymbol{x}) = \sum_{i=1}^{m} w_{ij}\phi_i(\|\boldsymbol{x} - \mu_i\|) + w_{j0}$$

the parameters w_{ij} ($j = 1, ..., C$, C = number of classes) are called the weights and μ_i the centers ($i = 1, ..., m$).

For instance two kind of basic functions are Thin plate $\phi(z) = z^2log(z)$ and gaussian $\phi(z) = exp(-z^2)$.

The centers μ_i can be obtained by the following procedures: 1) Random selection 2) Using clustering algorithms like k-means 3) Gaussian mixtures 4) K- nearest neighbors.

There are three ways to find the number of centers: i) Using cross validation, and then the number of centers are chosen when there is no appreciable increase or decrease of the classification accuracy in the process of cross validation. ii) Monitoring the performance of the algorithm in a separate test sample, and iii) Using the Gaussian mixture algorithm.

The weights w_{ij} can be found using minimum least squares procedure. This procedure is described below:

$$\begin{aligned} E &= \sum_{i=1}^{n} |t_i - g(x_i)|^2 \\ &= \| - T^T + W\Phi^T + w_0 \mathbf{1}^T \| \end{aligned}$$

$$t_{ij} = \begin{cases} 1 & \text{if } x_i \text{ is in } C_j, \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

where E is the square sum of $t_i - g(x_i)$; the matrix T is the target (size n x C); Φ is a nxm matrix, and its rows are the values of the functions evaluated in x_i; W is the weight matrix of size Cxm; w_o is a matrix of size Cx1; and $\mathbf{1}$ is a n dimensional column vector of ones.

If the matrix $\Phi^T\Phi$ is not singular then:

$$W = T^T\Phi^+ \tag{4.7}$$

where $\Phi^+ = \Phi\left(\Phi^T\Phi\right)^{-1}$.

The bias w_0 can be found taking the first derivative of E with respect to w_0. This gives the solution:

$$w_0 = \bar{t} - W\bar{\Phi} \tag{4.8}$$

where

$$\begin{aligned} \bar{t} &= \frac{1}{n}\sum_{i=1}^{n} t_i = \frac{1}{n} T^T \mathbf{1} \\ \bar{\Phi} &= \frac{1}{n}\sum_{i=1}^{n} \Phi(\boldsymbol{x}_i) = \frac{1}{n} \Phi^T \mathbf{1}. \end{aligned}$$

The algorithm is as follows:

1) Assign the form of the radial functions $\phi_i(\cdot)$ which are nonlinear.

2) Give the number of centers.

3) Find the centers using random selection or k-means algorithm.

4) Find the smooth parameter using cross validation (this parameter is a real number that normalizes the $\phi(\boldsymbol{x}_i)$ values).

5) Given the training set $\boldsymbol{x}_i$, $i = 1, ..., n$, find the $\phi_i = \phi(\boldsymbol{x}_i)$, $i = 1, ..., n$

6) Find the weights and bias using minimum least squares.

7) Classify the data using the discriminant function

$$g_j(\boldsymbol{x}) = \sum_{i=1}^{m} w_{ij}\phi_i(\|\boldsymbol{x} - \mu_i\|) + w_{j0} \quad j = 1, ..., C$$

The classification rule using radial basis function is: Assign $\boldsymbol{x}$ to the class C_i if

$$g_i(x) = \max_j g_j(x) \quad i = 1, ..., C$$

$\boldsymbol{x}$ is assigned to the class for which the discriminant function has the largest value.

The time complexity of this algorithm relies on the singular value computation, which is $O(nn_{hidden}^2 + n^2 n_{hidden} + n_{hidden}^3)$, where n is the number of instances and n_{hidden} is the number of base units in the hidden layer (equal to number of centers).

4.2.3 The Kernel Density Classifier

Given a univariate data set x_1, ... ,x_n, its empirical distribution function can be written as

$$\hat{F}(x) = \frac{\#\text{ observations} \leq x}{n}\,, \tag{4.9}$$

Since the density function of a random variable X is the derivative of its distribution function, the density function can be estimated as

$$\hat{f}(x) = \frac{\hat{F}(x+h) - \hat{F}(x-h)}{2h} \tag{4.10}$$

where $h > 0$. So, $\hat{f}(x)$ is the proportion of observations that fall in the interval $(x-h, x+h)$ divided by $2h$. Equation 4.10 can be written as

$$\hat{f}(x) = \frac{1}{hn} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right) \tag{4.11}$$

where

$$K(z) = \begin{cases} 0 & \text{If } |z| > 1; \\ \frac{1}{2} & \text{If } |z| \leq 1 \end{cases} \tag{4.12}$$

$K(z)$ is called the kernel function and h is the bandwidth or smoothing parameter. Other univariate kernel functions are

i. Gaussian: $K(x) = exp(-x^2/2)/\sqrt{2\pi}$

ii. Epanechnicov:

$$K(x) = \begin{cases} 3(1 - x^2/5)/(4\sqrt{5}) & \text{If } |x| < \sqrt{5}; \\ 0 & \text{Otherwise} \end{cases} \tag{4.13}$$

If the kernel $K(z)$ satisfies $K(z) \geq 0$ and $\int_R K(z)dz = 1$, then the density estimation $\hat{f}(x)$ given previously satisfies the necessary density function conditions, which are $\hat{f}(x) \geq 0$ and $\int_R \hat{f}(x)dx = 1$.

There are several methods to estimate the bandwidth parameter. The standard kernel uses the following:

$$h_{opt} = s\left(\frac{3}{4}\right)^{\frac{1}{5}} n^{-0.2} = s(1.06)n^{-0.2} \tag{4.14}$$

where s is the standard deviation of the vector $\boldsymbol{x}$, The adaptive kernel uses the variable h, changing in each point according to its neighborhood density.

The following steps should be carried out.

i. Find a pilot estimation $\widetilde{f}(x)$ such that $\widetilde{f}(x_i) > 0$ for each i.

ii. Define a bandwidth factor $\lambda_i = \{\widetilde{f}(x_i)/g\}^{-\alpha}$ where g is the geometric mean of $\widetilde{f}(x_i)$ and α is a parameter of sensibility and satisfies $0 \leq \alpha \leq 1$.

iii. Define the estimation by *adaptive kernel* as: $\hat{f}(x) = \sum_{i=1}^{n} \frac{1}{nh\lambda_i} K\{\frac{(x - x_i)}{h\lambda_i}\}$

In the multivariate case ($\boldsymbol{x} \in R^p$), the estimation of the density function is given by

$$\hat{f}(\boldsymbol{x}) = \frac{1}{nh_1h_2 \cdots h_p} \sum_{i=1}^{n} K_p \left(\frac{x_1 - x_{i1}}{h_1}, ..., \frac{x_p - x_{ip}}{h_p} \right) \tag{4.15}$$

where the bandwidth parameters are estimated by

$$h_j = s_j \left\{ \frac{4}{n(p+2)} \right\}^{\frac{1}{p+4}} \tag{4.16}$$

s_j is the deviation standard of the j-th feature. When we are working with a multidimensional data set, we have to take care of dimensionality. This problem requires that the number of observations grows exponentially with the number of variables, for searching observations in the neighborhood of the point where the density is estimated. We consider the kernel product estimator, which is defined by

$$K_p(\boldsymbol{x}) = \prod_{v=1}^{p} K(x_v) \tag{4.17}$$

where $K(\cdot)$ is a univariate density function. The estimation of the density using a fixed bandwidth is

$$\hat{f}(\boldsymbol{x}) = \frac{1}{nh_1h_2...h_p} \sum_{i=1}^{n} \prod_{v=1}^{p} K \left(\frac{x_v - x_{iv}}{h_v} \right) \tag{4.18}$$

In the same way, the estimation of the density function using a variable bandwidth is

$$\hat{f}(\boldsymbol{x}) = \frac{1}{nh_1h_2...h_p} \sum_{i=1}^{n} \left(\frac{1}{\lambda_i} \right)^p \prod_{v=1}^{p} K \left(\frac{x_v - x_{iv}}{h_v\lambda_i} \right) \tag{4.19}$$

where λ_i is calculated as in the univariate case.

An object x is assigned to the class i where the $\pi_i \hat{f}(x/C_i)$ is maximum (the π_is are the priors, and $\hat{f}(x/C_i)$ is the class conditional function estimated by the kernel product).

The complexity of this algorithm is $O(n_{classes}\ n_{test}\ n_{trai}\ n_{var})$, where $n_{classes}$ is the number of classes, n_{test} is the number of instances in the test sample, and n_{var} is the number of features.

4.2.4 The K- Nearest Neighbors Classifier

K-nearest neighbors classifier (KNN) [CH67] is a method for classifying objects based on closest training samples in the feature space. The probability that a point x falls in a volume V centered at a point x is given by

$$\theta = \int_{V(x)} p(x)dx$$

The integration is taken over the volume V. For small samples, $\theta \sim p(x)V$. The probability θ may be approximated by the proportion of samples falling within V. If k is the number of instances out of n falling within V, then $\theta \sim k/n$. Now the density can be approximated by

$$p(x) = \frac{k}{nV}$$

If x_k is the kth nearest neighbor point of x, then V may be taken to be a sphere, centered at x, of radius $\|x - x_k\|$ (the volume of a sphere in an n-dimension space is $2r^n\pi^{n/2}/n\Gamma(n/2)$, where $\Gamma(x)$ denotes the gamma function).

The classification rule of the k-nearest neighbors algorithm is as follows: Given an instance of the testing data sample, the k nearest neighbors of a training data is computed first. Then the testing instance is assigned to the most similar class of its k nearest neighbors. These k neighbors are the near instances to the testing instance with respect to Euclidean distance.

The time complexity of this algorithm is $O(n^2 n_{var})$, where n is the number of samples and n_{var} is the number of features.

4.2.5 The Naive Bayes Classifier

The Naive Bayes classifier relies in the classical Bayes theorem. The class posterior probability given a feature vector $\boldsymbol{x}$, is $f_i(\boldsymbol{x}) = P(C = i|X = \boldsymbol{x})$. But, $P(C = i|X = \boldsymbol{x}) = \frac{P(X = \boldsymbol{x}|C = i)P(C = i)}{P(X = \boldsymbol{x})}$ by Bayes theorem. Therefore, $f_i(\boldsymbol{x}) \propto P(X = \boldsymbol{x}|C = i)P(C = i)$. The Bayesian classifier is defined as:

$$h(\boldsymbol{x}) = arg\max_i P(X = \boldsymbol{x}|C = i)P(C = i) \quad i = 1, ..., g(\#\ of\ classes)$$

When the feature space is high dimensional, the Naive Bayes classifier assumes that features are independent. Therefore, the discriminant function is given by

$$f_i^{NB}(x) = \prod_{j=1}^{n} P(X_j = x_j|C = i)P(C = i) \quad n : number\ of\ features$$

For instance, we can assume that the set of instances with continuous features follows a gaussian distribution.

The time complexity of this algorithm is $O(nn_{var}^3)$.

4.3 Ensemble Methods

We distinguish two types of ensemble methods, one called generative and the other non-generative. The latter does not generate new base classifiers, instead it combines base classifiers in a suitable way to find the ensemble. An explanation of both methods is given below.

- Non-generative Methods

  These methods combine a set of base learning algorithms using a combiner module, which depends in its adaptivity of input and output of its base classifiers. If labels or if continuous outputs are hardened, the majority voting among the results of base classifiers is used. Weights can be assigned to each classifier output to optimize the combined classifier of the training set. Ensembles can be based on the Bayes rule approach. For this purpose the Behavior

Knowledge Space method considers each possible combination of class labels. This method computes the frequency of each class corresponding to each combination of the classifiers, but this technique requires a huge training data set. The base learners can be aggregated using operators such as Minimum, Maximum, Average, Product, and Ordered weight averaging. Another method of combination uses a second level learning machine. This learning algorithm takes the base learner outputs as features in the intermediate space.

- Generative Methods

  Resampling methods may be used to generate different training sets. In order to produce multiple classifiers, base learning algorithms can be applied to these sets. In Bagging we draw samples with replacement, but in Boosting at each iteration we use different distribution or weighting over the training samples. Another method to get training samples is leaving one disjoint subset out. This method is called cross-validation, which is a technique to sample without replacement. Randomized ensemble methods generate classifiers using random initial values to construct the classifier. For instance, in radial basis function, we can randomly initialize the initial weights obtaining different classifiers. In this thesis we use a combination of generative and non-generative methods. A comparison between the single base classifiers, their respective bagging, and the proposed ensemble is carried out.

4.3.1 Combination of generative and non-generative Ensembles

In this thesis our original work resides in the combination of two ensemble methods, majority voting, which is a non-generative method, and Bagging, which is a generative method.

The Bagging algorithm was introduced by Breiman [Bre94]. This algorithm consists of taking B bootstrap samples with replacements $Ł_1, Ł_2, ..., Ł_B$, generated

from a training sample L. A classifier C_i is built for each bootstrap sample L_i. Finally, a classifier C_A is generated, containing the most frequent class estimated by the C_i classifier (Figure 4.1).

1 **Input** $Training\ Sample\ £, Classifier\ C$,$\#\ of\ bootstrap\ samples\ B$
2 $for\ i = 1\ to\ B\ \{$
3 $£' =\ bootstrap\ sample\ from\ £$
4 $C_i = C(£')$
5 $\}$
6 $C_A(x) = \underset{y \in Y}{\operatorname{argmax}} \sum_{i:C_i(x)=y} \mathbf{1}$, $(most\ frequent\ class) \quad Y = \{1, 2, ..., g\}$
7 **Output** $Clasifier\ C_A$.

Figure 4.1: *Bagging* Algorithm

In majority voting an instance is classified in the most frequent class that appears in the classifier output.

Combining Bagging and majority voting consists of generating different base classifiers for each bootstrap sample. Then a majority voting method is applied to these classifiers. The final output is taken as the classifier output for each bootstrap sample in the Bagging algorithm (Figure 4.2).

1 **Input** $Training\ Sample\ £, Classifier\ C$,$\#\ of\ bootstrap\ samples\ B$
2 $for\ i = 1\ to\ B\ \{$
3 $£' =\ bootstrap\ sample\ from\ £$
4 $EC_i = majority\ vote\{BC_j(£')\}$
$where\ BC_j\ is\ a\ base\ Classifier,\ j = 1, ..., \#\ of\ base\ classifiers$
5 $\}$
6 $EC_A(x) = \underset{y \in Y}{\operatorname{argmax}} \sum_{i:EC_i(x)=y} \mathbf{1}$, $(the\ most\ frequent\ class) \quad Y = \{1, 2, ..., g\}$
7 **Output** $Clasifier\ EC_A$.

Figure 4.2: Proposed Ensemble Algorithm

4.3.2 Parallel design of the proposed meta-classifier algorithm

Basically, the parallel algorithm for combined Bagging resides in the master slave paradigm. For each iteration in the Bagging algorithm, the master process generates the bootstrap sample and sends the works to each slave. Finally, the master process collects the partial results combining them to find the solution. The algorithm is as follows:

i. The master process sends the data to slave processes

ii. Each slave builds a classifier and sends its results back to the master process.

iii. The master process continues sending work to slaves until there is no work to send. When there is no more work, the master process receives its last work from the slaves, and sends a message to them to finish their work.

iv. Finally, the master process finds the ensemble classifier.

The time complexity of this parallel algorithm is $O(B \cdot O(RBF + BAYES + C4.5 + KERNEL + KNN + VOT)/P)$, where B is the number of bootstrap samples, $O(RBF + BAYES + C4.5 + KERNEL + KNN)$ is the time complexity of the five base classifiers mentioned before, $O(VOT)$ is the time complexity of voting algorithm, and P is the number of processes.

4.4 Experimental Results

We use eight data sets from Machine learning Database Repository A.1 described in the Appendix. The error rates of classifiers are estimated using 10-fold cross validation technique (where the cross validation technique is applied to a data set partitioned into 10 parts). In table 4.1 are shown these error rates, which are the average of 10 runs. In this table $B^{C4.5}$ is the Bagging of C4.5 algorithm, B^{RBF} is the

Bagging of RBF algorithm, B^{KD} is the Bagging of kernel density algorithm, B^{NB} is the Bagging of the Naive Bayes algorithm, B^{KNN} is the Bagging of K-nearest neighbors algorithm, and B^1 is our proposed meta-classifier algorithm.

RBF and Knn algorithms have their own parameter for each dataset. These parameters are chosen according to their lowest classification error rate by cross validation. Here are the specifications:

1) RBF. This algorithm has hidden nodes (nh, tested from 2 to 30) and a scale parameter (scl, from 1 to 10000) used to reduce and normalize the values of distance matrices: iris (nh =5, scl = 1), diabetes (nh = 5, scl = 1), ionosphere (nh = 6, scl = 1) , breawst(nh = 3, scl = 1) , bupa(nh = 9, scl = 1), vehicle(nh = 20, scl = 1000), segment(nh = 25, scl = 100), and landsat (nh = 36 , scl = 10000).

2) KNN. This algorithm uses the number of neighbors (nn). For iris (nn = 4), diabetes(nn =7), ionosphere(nn = 3), breawst(nn=7), bupa(nn = 7), vehicle(nn = 3), segment(nn= 3), and landsat(nn = 11).

Also, table 4.1 shows the classification error rates for our proposed combined voting algorithm. This table shows that for each classifier the Bagging algorithm tends to reduce the classification error rate. In almost all data sets, our proposed algorithm, gives better results and is more robust (the classification accuracy is more stable) compared to single ones and their ensembles. In the table 4.2, we show a summary of ranking of each base classifier, their ensembles, and our proposed combined voting scheme.

Table 4.3 shows the run time of the parallel ensemble algorithm with different number of processors for the Landsat dataset. We cannot reach linear speedup, because of the following reasons: 1) The data communication cost between processes at the beginning of the computation. 2) The communication cost in each iteration. 3) The majority vote at the end of the Bagging loop is made by only one process.

Table 4.1: Classification Error rates of Base and Ensemble Algorithms

Dataset	C4.5	RBF	KD	NB	KNN	$B^{C4.5}$	B^{RBF}	B^{KD}	B^{NB}	B^{KNN}	B^{1}
Breawst	0.057	0.032	0.049	0.037	0.028	0.041	0.032	0.049	0.037	0.029	0.038
Bupa	0.385	0.359	0.359	0.420	0.359	0.335	0.348	0.379	0.414	0.358	0.368
Diabetes	0.259	0.277	0.264	0.246	0.255	0.242	0.257	0.265	0.247	0.261	0.238
Ionosphere	0.125	0.099	0.108	0.262	0.162	0.080	0.100	0.114	0.262	0.162	0.066
Iris	0.060	0.047	0.046	0.053	0.053	0.053	0.038	0.038	0.048	0.053	0.038
Landsat	0.262	0.189	0.134	0.219	0.207	0.211	0.157	0.129	0.219	0.199	0.161
Segment	0.062	0.142	0.134	0.230	0.111	0.056	0.122	0.136	0.230	0.110	0.067
Vehicle	0.271	0.335	0.373	0.508	0.353	0.262	0.245	0.347	0.507	0.357	0.241

Table 4.2: Ranking of Classifiers and Ensembles

Dataset	1^{st}	2^{nd}	3^{th}	4^{th}	5^{th}	6^{th}	7^{th}	8^{th}	9^{th}	10^{th}	11^{st}
Breawst	KNN	B^{KNN}	RBF	B^{RBF}	NB	B^{NB}	B^{1}	$B^{C4.5}$	KD	B^{KD}	C4.5
Bupa	$B^{C4.5}$	B^{RBF}	B^{KNN}	RBF	KD	KNN	B^{1}	B^{KD}	C4.5	B^{NB}	NB
Diabetes	B^{1}	$B^{C4.5}$	NB	B^{NB}	B^{RBF}	C4.5	KNN	B^{KNN}	KD	B^{KD}	RBF
Ionosfera	B^{1}	$B^{C4.5}$	RBF	B^{RBF}	KD	B^{KD}	C4.5	KNN	B^{KNN}	NB	B^{NB}
Iris	B^{1}	B^{RBF}	B^{KD}	KD	RBF	B^{NB}	NB	KNN	B^{KNN}	$B^{C4.5}$	C4.5
Landsat	B^{KD}	KD	B^{RBF}	B^{1}	RBF	B^{KNN}	KNN	$B^{C4.5}$	NB	B^{NB}	C4.5
Segment	$B^{C4.5}$	C4.5	B^{1}	B^{KNN}	KNN	B^{RBF}	KD	B^{KD}	RBF	NB	B^{NB}
Vehicle	B^{1}	B^{RBF}	$B^{C4.5}$	C4.5	RBF	B^{KD}	KNN	B^{KNN}	KD	B^{NB}	NB

Table 4.3: Running time (sec.) and speedup of parallel combined voting algorithm for Landsat data set

# of processors	1	2	3	4	5	6	7	8	9	10
Run time (sec.)	4795	3315	1691	1206	1023	900	820	780	720	690
Speedup	1.00	1.45	2.83	3.98	4.69	5.33	5.85	6.15	6.66	6.94

Chapter 5

Meta-clustering

5.1 Introduction

In real life, only humans have an incredible capacity to cluster objects, and identify which are similar and which are not. But, when we want to automate the process of clustering objects it becomes a difficult task.

When we do not have training data, the process is called unsupervised classification or clustering. The idea in clustering is how to build groups of similar objects, so that different groups are dissimilar enough. In clustering the principal work resides in how to define the similarity between objects, how to choose the appropriate measure, and how to cluster similar objects together. A clustering process tries to separate the data set into groups such that objects in each cluster are more similar between them than objects that are in other clusters.

Clustering is an important task in the data mining process. It can be used in many fields. In marketing it is used to find groups of customers with similar behaviors given a large database of customer data containing their characteristics, and past buying records. In Biology, clustering is used to classify animal and plants given their features; in libraries, it is used for book ordering; on the world wide web, it is used to classify documents, and to cluster weblog data, discovering groups of similar access patterns; in land use, it is used to study areas of similar land use in a earth observation database; in insurance, it is used to identify groups of motor insurance policy holders with high average claim cost and identifying frauds;

in city planning, it is used to identify groups of houses according to their house type, value, and geographical location; in earthquake studies, it is used to cluster observed earthquake epicenters, and to identify dangerous zones. Finally, in medical diagnosis, it is used to classify tumors or anomalous diseases.

Clustering algorithms are classified by the type of attributes they can handle, scalability to large data sets, ability to work with high dimensional data, ability to find clusters of irregular shape, handling outliers, time complexity, data order dependency, labelling or assignment, reliance on a priori knowledge and user defined parameters, and interpretability of results.

Cluster combination or cluster ensembles, refers to how to combine different clustering results from different base clustering algorithms. The major challenges in clustering ensembles are: How to combine clustering results, since different results can be produced from a single input data, and it is difficult to establish which result is the best and how much it can be trusted.

5.1.1 Literature Review

Among the recent research in cluster ensembles, we refer to the following:

Zeng et al. [ZTGG02] propose a method to extract information from results of different clustering techniques. They use a distance measure technique to represent the statistical signal of each cluster, and show that their approach is able to extract the information efficiently and accurately from an input clustering structure. Clustering results have different cluster size and different boundaries, and this becomes worse when the dimension of data increases. The goal is to make use of all the information combined in the different clustering results to produce a better understanding of data. This goal is achieved comparing the different clustering results in a finer way.

Strehl and Ghosh [SG02] define cluster ensemble problem as an optimization problem and proposed three effective combiners to solve cluster ensembles based on a hyper-graph model. They use three algorithms: a) Cluster-based similarity

partitioning, b) Hypergraph partitioning, and c) Meta-clustering. They introduce two techniques to generate basic clustering results: a) Robust centralized clustering, where each clustering algorithm has access to all features and to all objects and b) Feature distributed clustering, where each clustering algorithm has access to a restricted subset of features. Finally, they claim that cluster ensembles can be used to introduce robustness and dramatically improve sets of subspace clustering for a variety of domains.

On the other hand, Greene et al. [GTBC04] discuss a variety of ensemble generation strategies and integration schemes and suggest an optimal set of parameters for each data set under consideration. They use the following ensemble generation methods: a) Plain, b)Random - K , c)Random - K + , d) Bagging, e) Random sub spacing, f) Random projection, and g) Heterogeneous ensembles. They conclude that the performance of an ensemble clustering algorithm relies on the choice of the base clustering algorithm, generation technique, number of ensemble members, and final meta-clustering algorithm. They suggest that other combination methods strategies can be more successful in exploiting diversity. Quantifying diversity such as pair wise or variance-based could also be investigated.

Januzaj et al. [JKP03] clustered the data locally and extracted suitable representatives out of these clusters. These representatives are sent to a global server site where they are restored as a complete clustering based on the local representatives. This approach is efficient, because the local clustering can be carried out quickly and independently from each other, taking advantage of distributed clustering. The transmission cost is much smaller compared to the size of the complete data set. From the small number of representatives, the global clustering can be done very efficiently. They developed a distributed algorithm based on DBSCAN [EKSX96]. Based on experimental results they showed that their clustering approach yields almost the same clustering as a central clustering in all data.

Leisch [Lei99] proposes a combination of partitioning and hierarchical clustering methods. He obtained a collection of training sets by resampling from the

empirical distribution of the original data, and then ran any partitioning clustering algorithm (K-means, Fuzzy C-means, PAM, etc). The results of the base clustering algorithms are combined into a new data set, which is used as input for a hierarchical clustering algorithm. He concludes that his Bagging algorithm gives better results than single ones.

Dudoit and Friedlyan [DF03], also propose two Bagging clustering algorithms. They apply the clustering procedure repeatedly to each bootstrap sample, and the final partition was obtained by plurality voting. They claim that their algorithms perform better than stand alone algorithms.

5.1.2 Motivation

One of the main advantages of meta-clustering, is its adaptiveness, because it can be applied to different pools of clustering schemes and is able to pick up the most suitable candidate. Combining several clustering results can improve quality and robustness of results. Another advantage is that it can be carried out using distributed computing without the necessity of parallelizing existing serial clustering algorithms.

5.1.3 Our Work

In this thesis we design a meta-clustering algorithm based on majority voting along with relabelling techniques. Also, we compare five different base clustering algorithms: Gaussian Mixtures methods (EM) from the statistics field, Fuzzy C-means (FCM) from Neural Networks, PAM, DBSCAN, and BIRCH from Machine Learning, with ensembles based in hypergraph partitioning (MCLA), Bagging and our proposed voting algorithm. Their performance is compared using clustering validation techniques, which quantify the performance of a clustering algorithms based on its classification accuracy. A parallel algorithm to compute four meta-clustering algorithms altogether is also designed. Two of these algorithms are Bagging based, the MCLA algorithm, and our voting algorithm.

5.1.4 Organization

This chapter is organized as follows: Section 2 shows the taxonomy of clustering algorithms. In Section 3 we talk about the five base-clustering algorithms used in this thesis. Section 4 is concerned with meta clustering algorithms. In Section 5 clustering validation techniques are presented. Parallel clustering algorithm is described in Section 6. Experiments and results are shown in Section 7.

5.2 Clustering Techniques in Data Mining

Clustering is an active research area in several fields such as Statistics, Pattern Recognition, and Machine Learning. A variety of algorithms have been proposed recently and applied to real life problems. Because data mining deals with large data sets, it imposes a challenge to design efficient clustering algorithms.

5.2.1 Cluster Definition

Clustering is defined as a partition of a set using a similarity criterion. Given the data set $X = \{\boldsymbol{x}_i, i = 1, ..., N\}$, let P be the partition of X into m sets $C_j,\ j = 1, ..., m$. These sets are called clusters if they satisfy the following partition condition: 1) $C_i \neq \phi, i = 1, ..., m$. 2) $\cup_{i=1}^{m} C_i = X$. 3) $C_i \cap C_j = \phi, i \neq j, i, j = 1, ..., m$

5.2.2 Similarity and Dissimilarity measures

Similarity measures are used to find the proximity of a pair of objects in a data set X. In contrast, dissimilarity measures are used to find the more distant of a pair of objects in a data set X. There are plenty of similarity and dissimilarity measures. The most frequently similarity measures used by clustering algorithms are the following:

1) The Euclidean distance between two points $\boldsymbol{x} = (x_1, ..., x_n)$ and $\mathbf{y} = (y_1, ..., y_n))$ is defined by $d_2(\boldsymbol{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$, and

2) The Manhatan distance between two points x and y is defined by $d_1(\boldsymbol{x}, \mathbf{y}) = \sum_{i=1}^{n} |x_i - y_i|$.

5.2.3 Taxonomy of Clustering algorithms:

Clustering algorithms are classified by the type of attributes they can handle, scalability to large data sets, ability to work with high dimensional data, ability to find clusters of irregular shape, handling outliers, time complexity, data order dependency, labelling or assignment, reliance in a priori knowledge and user defined parameters, and interpretability of results. Essentially, the clustering algorithms are classified as:

i. **Hierarchical Methods**. These methods proceed in the following way: at each step, an object is assigned to a cluster using information produced in the previous step. There are two types of these algorithms.

   - **Agglomerative Algorithms**. The algorithm begins with a cluster for each object. In each step one object is assigned to the most similar cluster. It is a button-up algorithm.
   - **Divisive Algorithms**. The algorithm begins with a cluster for all data. At each step it splits the cluster using a dissimilarity measure. It is a top-down algorithm.

ii. **Partitioning Methods**. These methods are based on the search of k representative objects of the data set. Once the representative objects are known, each object is assigned to the cluster where the nearest representative object belongs to. Among the partitioning algorithms are the following: Relocation algorithms, Probabilistic clustering, K- medoids methods, K- means methods, Density Based algorithms, Density Based Connectivity clustering, and Density Functions clustering

iii. **Grid based methods**. These methods inherit the topology from the underlying attribute space. Multirectangular segments (also called cube, cell, and region) are considered to bound the search combinations. For instance, DENCLUE, CLIQUE, BAND, GRIDCLUST, Wavecluster and STING are algorithms based on grid methods.

iv. **Methods based on Co-occurrence of Categorical data**. When we have data with categorical variables, conventional clustering algorithms perform poorly and do not work well. Clustering methods based on the idea of co-occurrence of categorical data have been introduced. For instance ROCK and CURE work with categorical variables.

v. **Constraint based clustering**. Frequently, the data set is constrained. In this case, an iterative optimization procedure is used. Clustering partition relies in moving objects to their nearest cluster representatives without violating the constraints.

vi. **Gradient descent and artificial neural networks**. This is similar to the EM algorithm, where exponential probabilities are defined based on Gaussian models. This makes the objective function differentiable with respect to means, allowing the application of general gradient descent method.

vii. **Evolutionary methods**. These methods rely on search such as simulated annealing, tabu search and genetic algorithm.

5.3 Base Clustering Algorithms

In this thesis, we study the combination of five base clustering algorithms. Gaussian Mixture models (EM) from Statistics, Fuzzy C-means (FCM) from Neural Networks, Partition around Medoids (PAM), Density based (DBSCAN), and balance iterative clustering (BIRCH) from Machine Learning.

5.3.1 Gaussian Mixture Models

This technique considers the data to be independently drawn from a mixture model [DLR77] of several probability distributions.

Each observation x is assumed to belong to one and only one cluster, so we can estimate the probabilities of the assignments $Pr(C_j|x)$ to j^{th} model $j = 1, ..., g$. The overall likelihood of the data is its probability to be drawn from a given mixture model.

$$L(X|C) = \prod_{i=1:N} \tau_j Pr(x_i|C_j)$$

where τ_j is the prior of the j class. Log-likelihood $log(L(X|C))$ serves as an objective function, which is maximized using the EM algorithm. In this algorithm, expectation and maximization steps are carried out iteratively, until convergence or certain threshold is satisfied. Step (E) estimates probabilities $Pr(x|C_j)$, which is equivalent to a soft reassignments. Step (M) finds an approximation to a mixture model, given current soft assignments. This process continues until log-likelihood convergence is achieved.

Each cluster is mathematically represented by a parametric distribution such as: Gaussian (continuous), and Poisson (discrete). The entire data set is modelled by a mixture of these distributions. An individual distribution used to model a specific cluster is often referred to as a component distribution.

Suppose that there are g components (clusters). Each component has a Gaussian distribution parameterized by μ_j and Σ_j. The density of component j is

$$f(\boldsymbol{x}; \theta_j) = \frac{1}{\sqrt{(2\pi)^d |\Sigma_j|}} exp(\frac{-(\boldsymbol{x} - \mu_j)^t \Sigma_j^{-1} (\boldsymbol{x} - \mu_j)}{2})$$

where $\boldsymbol{x} \in R^d$. The prior probability (weight) of component j is π_j. Therefore, the mixture density is

$$f(\boldsymbol{x}) = \sum_{j=1}^{g} \pi_j f(\boldsymbol{x}; \theta_j) \quad where\ \theta_j = (\mu_j, \Sigma_j)$$

A mixture model with high likelihood tends to have the following traits: Its Component distributions have high "peaks", because data in one cluster are tight. The mixture model "covers" the data well, because dominant patterns in the data are captured by component distributions. It has available well-studied statistical inference techniques. Also, it has flexibility in choosing the component distributions, obtaining a density estimation for each cluster. Finally, it has available a "soft" classification. The EM algorithm follows two steps until convergence:

E-step

$$w_{ij} = \frac{\pi_j^{(m)} p(x_i|\theta_j^{(m)})}{\sum_k \pi_k^{(m)} p(x_i|\theta_k^{(m)})}$$

M-Step

$$\pi_j = \frac{1}{n} \sum_{i=1}^{n} w_{ij}$$

$$\mu_j = \frac{1}{n\pi} \sum_{i=1}^{n} w_{ij} x_i$$

$$\Sigma_j = \frac{1}{n\pi_j} \sum_{i=1}^{n} w_{ij} (x_i - \mu_j)(x_i - \mu_j)^T$$

The time complexity of this algorithm is $O(nn_{var}^3 n_{iter})$, where n is the number of instances, n_{var} is the number of attributes, and n_{iter} is the number of iteration achieved until the convergence.

5.3.2 Partition Around Medoids

Partitioning Around Medoids [KR90] (PAM) is a clustering algorithm based on the search of representative objects, called medoids. PAM starts from an initial set of medoids and iteratively replaces one of the medoids by one of the non-medoids if it improves the total distance of the resulting clustering. When medoids are selected, clusters are defined as subsets of points close to respective medoids. The objective

function is defined as the average distance or another dissimilarity measure between a point and its medoids.

PAM works effectively for small data sets, but does not scale well for large data sets. There exists other algorithms to deal with large data sets such as CLARA [KR90] and CLARANS [NH94].

The k-medoid algorithm (Figure 5.1) is described below as follows:

i. Arbitrarily select k objects from the data as medoids.

ii. Consider swapping the pair of objects (i,h); where $i \in$ selected objects and $h \in$ non- selected objects. Denote the swap as $i \leftrightarrow h$. Let $d(x_i; x_h)$ be the distance-measure between two objects i and h. Now consider another non-selected object j.

Calculate T_{ih}, "the total swap contribution" for $i \leftrightarrow h$, as

$$T_{ih} = \sum_j C_{jih}$$

Where C_{jih} is the contribution to $i \leftrightarrow h$ from object j, defined below. There are four possibilities to consider when calculating C_{jih}.

- If j currently belongs to the cluster defined by medoid i (denoted cluster i), consider the distance $d(x_j; x_h)$ between object j and object h. If h is farther from j than the second best medoid i' is from j, then the contribution from object j to the swap is

$$C_{jih} = d(x_j, x_{i'}) - d(x_j, x_i)$$

The result of $i \leftrightarrow h$ would be that object j now belongs to cluster i'.

Else if h is closer to j than i' is to j, the contribution from j to the swap is

$$C_{jih} = d(x_j, x_h) - d(x_j, x_i)$$

The result of $i \leftrightarrow h$ would be that object j now belongs to cluster h.

- If j currently belongs to cluster k, where $k \neq i$, check the distance between object j and object h. If h is further from j than the medoid k is from j, then the contribution from object j to the swap is

$$C_{jih} = 0$$

The result of $i \leftrightarrow h$ would be that object j still belongs to cluster k. Else if h is closer to j than k is to j, the contribution from j to the swap is

$$C_{jih} = d(x_j, x_h) - d(x_j, x_i)$$

The result of $i \leftrightarrow h$ would be that object j now belongs to cluster h.

- Let $(i^*, h^*) = \underset{i,h}{\operatorname{argmax}} T_{ih}$. If $T_{i^*h^*} < 0$, then swap $i^* \leftrightarrow h^*$. Now object h belongs to selected objects and i belongs to non-selected objects. Go to step ii.
- Allocate each non-selected object to the cluster defined by the nearest medoid.

The algorithm is shown in Figure 5.1.

The complexity of this algorithm $O(n_{iter}k(n-k)^2)$, because in each iteration the are $k(n-k)$ swaps, and each of them accesses $n-k$ distances.

5.3.3 Fuzzy C-means

This algorithm was first introduced by Dunn [Dun73] in 1973 and subsequently improved by Bezdek [Bez81] in 1981. C-means is a method that allows one instance to belong to two or more clusters. It is based in the minimization of the following objective function:

$$J_m = \sum_{i=1}^{N} \sum_{j=1}^{C} u_{ij}^m \|x_i - c_j\|^2, \quad 1 \leq m < \infty$$

Input:
$D = \{t_1, t_2, ..., t_n\}$ // Set of elements
k // Number of desired clusters
Output
K //Set of clusters.
PAM Algorithm
Build Phase
1. select k initial medoids from D
Swap Phase
2. repeat
3. for each t_h not a medoid do {
4. for each medoid t_i do {
5. calculate TC_{ik} }}
6. find i, h where TC_{ik} is the smallest
7. If $TC_{ik} < 0$ then
8. replace medoid t_i with t_h
9. until $TC_{ih} \geq 0$
10. for each $t_i \in D$
11. assign t_i to K_j where $dis(t_i, t_j)$ is the smallest over all medoids

Figure 5.1: PAM algorithm

where m is a real number greater than 1, u_{ij} is the degree of membership of x_i in the cluster j, x_i is the is the $i-th$ instance of the d-dimensional data set, and c_j is the $d-$dimensional center of the cluster.

In each iteration of the algorithm the membership u_{ij} and the cluster centers c_j are updated by optimization of the objective functions. This gives the following:

$$u_{ij} = \frac{1}{\sum_{k=1}^{C} \left(\frac{\|x_i - c_j\|}{\|x_i - c_k\|} \right)^{\frac{2}{m-1}}} \quad , \quad c_j = \frac{\sum_{i=1}^{N} u_{ij}^m x_i}{\sum_{i=1}^{N} u_{ij}^m}$$

The condition to stop this iteration is when $max_{ij}\{|u_{ij}^{k+1} - u_{ij}^k|\} < \varepsilon$, where ε is the termination criterion ($\varepsilon \in [0, 1]$). This procedure converges to a local minimum

or a saddle point of J_m

The algorithm is described in Figure 5.2.

Input:

D // Set of elements

k // Number of clusters (clusters)

Output

C //Set of clusters.

FCM Algorithm

1. Initialize $U = [u_{ij}]$
2. repeat
3. At step k calculate the center vectors $C^k = [c_j]$ with U^k
4. $c_j = \frac{\sum_{i=1}^{N} u_{ij}^m x_i}{\sum_{i=1}^{N} u_{ij}^m}$
5. Update U^k , U^{k+1}

$$u_{ij} = \frac{1}{\sum_{k=1}^{C} \left(\frac{\|x_i - c_j\|}{\|x_i - c_k\|} \right)^{\frac{2}{m-1}}}$$

6. until $\|U^{k+1} - U^k\| < \varepsilon$

Figure 5.2: FCM Algorithm

The complexity of this algorithm is $O(nc^2p)$, where n is the number of observations, c is the number of clusters, and p is the number of attributes.

5.3.4 DBSCAN

The algorithm DBSCAN [EKSX96] (Density Based Spacial Clustering of Application with Noise) is a clustering algorithm designed to find clusters based on density properties. This algorithm uses two input parameters ξ and $MinPts$. The first one is a ξ-value, which dictates a certain ξ-neighborhood, which is a collection of data points within a specific ξ-radius. The value of the $MinPts$ parameter is the minimum number of points that must be found in the ξ-neighborhood of a core-object data point.

Three measurements are used to describe properties of these data points. These properties are based on whether the point is directly-density-reachable, density-reachable, or density-connected from a core-object. These properties describe relationships between each of the core objects and determine the most appropriate clustering of the core objects. This algorithm is excellent at forming non-round clusters. However, this algorithm suffers from the same problem than the k-means algorithm, because these parameters must be specified before the clusters can be formed. This unfortunately leads to cluster formation being determined by the user.

Input:

D // Set of elements

Eps // Maximum radius of the neighborhood

$MinPts$ // Minimum number of points in an Eps-neighborhood of a point

Output

C //Set of clusters.

O //Set of outliers

DBSCAN Algorithm

1. Arbitrarily select a point p
2. Retrieve all points density-reachable from p wrt Eps and $MinPts$
3. If p is a core point, a cluster is formed
4. If p is a border point, no points are density-reachable from p
5. visit the next point of the data set
6. Continue the process until all of the points have been processed

Figure 5.3: DBSCAN algorithm

The following definitions are used in the DBSCAN algorithm:

i. An ξ neighborhood of a point $\boldsymbol{x}$ is given by $N_{\boldsymbol{x}} = \{\mathbf{y} \in X | d(\boldsymbol{x}, \mathbf{y}) \leq \xi\}$ of the point x.

ii. A core object is a point with a neighborhood consisting of more than $MinPts$ points.

iii. A point **y** is density reachable from a core object $\boldsymbol{x}$ when a finite sequence of core objects between $\boldsymbol{x}$ and **y** exists such that the next core object belongs to an ξ neighborhood of its predecessors.

iv. Two points $\boldsymbol{x}$ and **y** are density connected when they are density reachable from a common core object.

DBSCAN defines a symmetric relation and all the points reachable from core objects can be factorized into maximally connected components serving as clusters. The points that are not connected are declared outliers. The no-core points inside a cluster represent its boundary. Core objects are internal points. Processing is independent of the data ordering. So far, nothing requires any limitations of the dimensionality of the data or attribute types. But an effective computing of the ξ neighborhoods presents a problem. However, in the case of low dimensional spatial data, different effective indexation schemes exist ($O(log(N))$ rather than $O(N)$). DBSCAN relies on R* trees indexation [EKSX96], therefore, in low dimensional spatial data, the theoretical complexity of DBSCAN is $O(Nlog(N))$

Ester et al. [EKSX96] have a detailed definition of DBSCAN algorithm. In Figure 5.3 we give a synthesis of this algorithm.

5.3.5 BIRCH

BIRCH (Balanced Iterative Reducing and Clustering Using Hierarchies) is a clustering algorithm for large data sets introduced by Zhang et al. [ZRL96]. The authors introduce two new concepts to keep track of clusters that are being formed. The first is called the cluster feature, which is a triple containing the number of points in a particular sub-cluster and then the corresponding linear and squared sum of these points. The second feature is a clustering feature (CF) tree, which is a height-balanced tree that stores clustering features.

The BIRCH algorithm performs two procedures. It first builds an initial CF tree, which is stored in memory, by scanning the database. Then, it performs clustering against the leaf nodes of the CF tree. BIRCH performs quite well computationally, however it can not handle clusters that do not have spherical or circular shape very well.

A Clustering Feature is used to represent a sub-cluster by the general statistical information on it. The definition is: $CF = \{N, LS, SS\}$. It has an additive property, for instance, the CF of the cluster obtained by merging C_1 and C_2 is $CF_1 + CF_2 = N_1 + N_2, LS_1 + LS_2, SS_1 + SS_2$. Statistical information used by clustering methods can be calculated easily by using the CFs. For example, by giving the related CFs of two clusters C_1 and C_2, it is not difficult to calculate their centroids X_{0_1} and X_{0_2}, the radius R of a single cluster, and the Euclidean distance D between two cluster centroids. Suppose the two CFs are N_1, LS_1, SS_1 and N_2, LS_2, SS_2, the calculation is as follows:

$$X_{0_1} = \frac{\sum_{\forall X_i \in C_1} X_i}{N_1} = \frac{LS_1}{N_1}$$

$$X_{0_2} = \frac{\sum_{\forall X_i \in C_2} X_i}{N_2} = \frac{LS_2}{N_2}$$

$$RC_1 = \left(\frac{\sum_{\forall X_i \in C_1} \|X_i - X_{0_1}\|}{N_1}\right)^{\frac{1}{2}} = \frac{1}{N_1} \sum_{element} \left(SS_1 - \frac{LS_1^2}{N_1}\right)^{\frac{1}{2}}$$

$$D = \|X_{0_1} - X_{0_2}\| = \|\frac{LS_1}{N_1} - \frac{LS_2}{N_2}\|$$

where, $\sum_{element}\{V\}$ is the summation of all the elements in the vector V . In BIRCH, CFs are stored in a Clustering Feature tree (CF tree), which is used for hierarchical clustering. For building the CF tree, two parameters are given by the users: the branching factor B and the threshold T. These two parameters constrain the growth of the tree. The branching factor B specifies the maximum number of children per non-leaf node. Therefore, a non-leaf node in the tree can have at most

B children. Every non-leaf node stores the sum of CFs from all its children. For a leaf node, it stores the CF calculated from all the objects in the sub-cluster which is represented by the leaf node. The threshold T is used to limit the size of the sub-clusters, which are represented by leaf nodes (the radius of all the sub-clusters should be less than T).

The algorithm for inserting an entry *ent* into the CF tree has three main steps. Here, an entry can be a single data object or a group of data objects. The BIRCH algorithm is the following:

i. Identify the appropriate leaf to insert: Find the leaf node, which represents a subcluster that is closest to the inserted entry. The closeness can be measured by the Euclidean distance between the centroids of the sub-cluster and the entry *ent*. This operation starts from the root and recursively descends the CF tree by always choosing the closest child node.

ii. Modify the leaf node: Suppose the sub-cluster in the leaf node LN_i is closest to *ent*. Check if LN_i can absorb *ent* without violating the threshold constraint T. If so, simply modify the CF in LN_i. If not, add a new leaf node, say LN_j to be a new child of the parent node of LN_i. Then, form a group of objects by all the objects in the sub-group represented by LN_i and all the objects in *ent*. Divide this group to be in two sub-groups by the following methods: First, find two objects that are the farthest away from each other in the group; Suppose these two objects are obj_1 and obj_2, assign all the objects in the group to sub-group 1 if the objects are closer to obj_1 than obj_2, the other objects form the sub-group 2. Now, assign sub-group 1 and 2 to be the sub-clusters represented by LN_i and LN_j respectively.

iii. Modify the path to the root node: The CF in each node in the path from the affected leaf nodes to the root must be modified after *ent* has been inserted. If a split did not happen in step 2, only the affected CFs need be modified. If a split happened, then check if the constraint of the branching factor B is

violated. If so, the parent node has to be split, and this split may affect the nodes in high levels the same way as well. If the root node is split, the height of the tree increases by 1. The way to split a non-leaf node is similar to the way to split a leaf node in step ii, the only difference is that when splitting a non-leaf node, use all its children to form a group to be divided.

Each time a point is added, the CFs in the affected nodes have to be updated. BIRCH builds the CF tree by scanning the total data set once, then obtains the clustering result based on the CF tree, which could be done in main memory of the computer. This algorithm is significantly appropriate for finding clusters in a very large data set. Since the CF tree can be constructed incrementally, any data objects given later can be added to the tree without rebuilding it. So this incremental clustering is another notable advantage of BIRCH. We can see from the BIRCH algorithm how these two parameters B and T control the building of the CF tree. In BIRCH, their choice is mainly determined by memory constraints of the computers. Since the cluster size is limited by these parameters, a cluster found from the CF tree does not always correspond to a nature cluster. This is a problem of BIRCH.

The maximum size of the tree is M/P. The time complexity in phase 1 is as follows: A point is inserted in the tree in $O(1 + log_B(M/P))$. At each node it is necessary examine B entries for the closest one, and each of them is proportional to dimension d. So, inserting all points takes time $O(d * N * B(1 + log_B(M/P)))$. The cost of re-inserting leaf entries is $O(d * (M/ES) * B(1 + log_B(M/P)))$, where there are almost (M/ES) leaf entries. Therefore the total cost in phase 1 is $O(d * N * B(1 + log_B(M/P)) + log_2(N/N_0) * d * (M/ES) * B(1 + log_B(M/P)))$, where $O(log_2(N/N_0))$ is the number of times needed to rebuild the entire tree.

5.4 Meta-clustering

Cluster ensembles provide a consolidation of individual clustering results, and offers quality and robustness, knowledge reuse, and distributed computing.

Different clustering algorithms with different parameter settings may generate different partitions of the same data, due to the exploratory nature of the clustering algorithm. We may combine them to provide robustness and quality.

Combining clustering is a difficult task, because patterns are unlabelled, and thus it is necessary to solve a correspondence problem. There are a variety of consensus functions to combine clusters. They are based on the co-association matrix [FJ02], hypergraph cuts [SG02, KK95], mutual information [TJP03], voting [DF03, FB03], Bipartite Graph Partitioning [ZB04], and soft correspondence [LZY05].

5.4.1 Bagged Clustering

Here we describe two Bagging algorithms for clustering.

i. Bagged Clustering based on combined partitioning and hierarchical method (BAG1). Leisch [Lei99] proposes a combination of partitioning and hierarchical clustering algorithms. He obtains a collection of training sets by re-sampling from the empirical distribution of the original data. Then, he runs a partitioning clustering algorithm. The results of the base methods are combined into a new data set which is then used as input for a hierarchical method.

   1) Construct B bootstrap training samples $X_N^1, ..., X_N^B$ by drawing replacements from the original sample X_N.

   2) Run the base cluster method (K-means competitive learning) in each set resulting in BxK centers $c_{11}, ..., c_{1k}, c_{21}, ..., c_{BK}$, where K is the number of centers used in the base method and c_{ij} is the ith center found using X_N^i.

   3) Combine all centers into a new data set $C^B = C^B(K) = c_{ij}, ..., c_{BK}$.

4) (Optional) Prune the set C^B by computing the partition of X_N with respect to C^B and remove all centers where the corresponding cluster is empty (or below a predefined threshold), resulting in a new set

$$C^B_{prune}(K, \theta) = \{c \in C^B(k) | \#\{x : c = c(x)\} \geq \theta\}$$

Make all members of $C^B_{prune}(K, \theta)$ unique.

5) Run a hierarchical cluster algorithm in $C^B_{prune}(K, \theta)$, resulting in a dendrogram.

6) Let $c(x) \in C^B$ denote the center closest to x. A partition of the original data can be obtained by cutting the dendrogram at a certain level, resulting in a partition $C^B_1, ..., C^B_m, 1 \leq m \leq BK$ of set C^B. Each point $x \in X_N$ is now assigned to the cluster containing $c(x)$.

ii. Bagged Clustering based on voting (BAG2). Dudoit and Fridlan [DF03] propose a bagged clustering algorithm, where they apply a clustering procedure repeatedly to each bootstrap sample, and the final partition is obtained by plurality voting.

The algorithm is as follows:

1) Apply the partitioning clustering procedure P to the original set L to obtain cluster labels $P(x_i; L) = \widehat{y}_i$. for each observation $x_i, i = 1, ..., n$.

2) Form the bootstrap sample $L^b = (x^b_1, ..., x^b_n)$

3) Apply the clustering procedure P to the bootstrap learning set L^b and obtain cluster labels $P(x^b_i; L^b)$ for each observation in L^b.

4) Permute the cluster labels assigned to the bootstrap learning set L^b so that there is a maximum overlapping with the original clustering of these observations. Specifically, let S_k denote the set of all permutations of the integer 1,...,k. Find the permutation $\tau^b \in S_K$ that maximizes.

$$\sum_{i=1}^{n} I(\tau(P(x_i^b; L^b)) = P(x_i^b; L))$$

where $I(\cdot)$ is the indicator function, equaling 1 if the condition in parentheses is true and 0 otherwise.

5) Repeat Step 2-4 B times and assign a bagged cluster label for each observation i by majority vote, that is, the cluster label corresponding to x_i is $argmax_{1 \leq k \leq K} \sum_{b:x_i \in L^b} I(\tau^b(P(x_i; L^b)) = k)$.

Also, record a cluster vote, which is the proportion of votes in favor of the winning cluster assignment, that is,

$$CV(x_i) = \frac{max_{b:x_i \in L^b} I(\tau^b(P(x_i; L^b)) = k)}{|b : x_i \in L^b|}$$

5.4.2 Majority Voting

It consist in combining different base clustering algorithms, and assign each observation to the most frequent cluster. In this thesis we combine five base clustering algorithms: PAM, FCM, EM, DBSCAN, and BIRCH. Since the base clustering yield different labels, we need to relabel them, using a permutation of labels that maximize the classification accuracy with respect to one clustering algorithm. But this relabelling algorithm is of factorial order in the number of clusters. We use the Hungarian algorithm (introduced by Harold Kuhn in 1955 and revised by James Munkres in 1957), which can do this work in cubic polynomial time in the number of clusters.

5.4.3 Graph partitioning

Another way to make cluster combination was introduced by Strehl and Ghosh [SG02]. They propose three meta-cluster algorithms, which are based on graph partitioning algorithms. These algorithms are the following:

- Cluster based similarity partitioning algorithm (CSPA). If two objects are in the same cluster, then they are considered similar. Similarity can be defined as the fraction of clustering in which two objects are in the same cluster. The similarity matrix S of size nxn is computed to a sparse matrix $S = \frac{1}{r}HH^{\dagger}$. Then, S is re-clustered using METIS [KK95]. This algorithm has complexity $O(n^2kr)$.

- Hyper Graph partitioning Algorithm (HGPA). The cluster ensemble problem is formulated as partitioning the hypergraph by cutting the minimal number of hyperedges. All hyperedges and vertices are equally weighted. For this purpose HMETIS [KAKS97] is used. It has complexity of $O(nkr)$

- Meta-clustering Algorithm (MCLA). The main idea is to group and collapse hyperedges and assign each object to the collapsed hyperedge in which it participates most strongly. Thus, the indicator vectors h (hyperedges of H) form the vertices of a regular undirected graph. The edge weight $w_{a,b}$ between two vertices h_a and h_b is defined by the binary Jaccard measure. Next, find matching labels by partitioning the meta-graph into k balanced meta-clusters. This is a clustering of the h vectors in H. Since each vertex in the meta-graph represents a distinct cluster label, a meta-cluster represents a group of corresponding labels. Each meta-cluster collapses the hyperedges into a single meta-hyperedge. Each meta-hyperedge has an associated vector which contains an entry for each object describing its level of association with the corresponding meta-cluster. The level is computed by averaging all indicator vectors h of a particular meta-cluster. Finally, an object is assigned to the meta-cluster with the highest entry in the association vector. This algorithm has complexity of $O(nk^2r^2)$.

In this thesis, we work only with the MCLA algorithm.

5.5 Cluster Validation Techniques

Cluster Validation can be understood as an estimated measure of cluster qualities. In unsupervised clustering algorithms, the real partitions are not known. In fact, using the original data set is one of the few ways to validate the results. Of course, that means that you will never know what the absolute true solution is, but nevertheless some validation methods could help to understand how well the algorithm worked with the data. The use of quality indexes for clusters is an important step that should not be used just as a final analysis but also as a method to review the parameters of previous steps like preprocessing, or clustering analysis.

In this section we present some index validation techniques, to find out clustering accuracy. These techniques show us how similar or dissimilar two clusters are. Some data sets do not have cluster structure, so a measure to identify if these data sets have cluster structure is required. Cluster validity is used to evaluate quantitatively the results of a clustering algorithm. Three different validation techniques are known [TK99b]:

- External validation measures. This approach is based on a previous knowledge of the data set.
- Internal validation measures. This approach is based on the information intrinsic to the data set alone. Measures of robustness, compactness or separation can be partially calculated, but there are some indexes to combine all of these values.
- Relative Criteria. Given a set of parameters associated with a specific clustering algorithm, among the clusters obtained by an algorithm using different values of the parameter, choose the one that best fits the data set.

Consider C and P, two partitions of the same data set. Let $SS = a$ if both vectors belong to the same cluster in C and to the same group in P; $SD = b$ if both vectors belong to the same cluster in C and to different groups in P ; $DS = c$ if

both vectors belong to different clusters in C and to the same group in P; $DD = d$ if both vectors belong to different clusters in C and to different groups in P .

The following validation indices are the more frequently used in unsupervised classification

i. Jaccard Index. This index shows the proportion of pairs that are in the same cluster, in the same partition and those that are either in the same cluster or in the same partition. It is an external validation measure.

$$J = \frac{a}{a+b+c}$$

ii. Silhouette. This index measures the compactness and the disjointness of clusters. Let $a(i)$ be the average dissimilarity between i and all the other objects in cluster C_j. If C_k is different to C_j, and $b(i) = min_{C_i \neq C_j} d(i, C_k)$ $(k = 1, ..., c \ ; k \neq j)$, where $d(i, C_k)$ denotes dissimilarity between instance i and cluster C_k. Then, silhouette width is defined as

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}}$$

The global silhouette width is defined as

$$GS_U = \frac{1}{c} \sum_{j=1}^{c} S_j$$

where S_j is the average silhouette width of cluster C_j, and U is any partition of C. This index is an internal validation measure.

iii. Mutual information. This index is a symmetric measure for the degree of dependency between the clustering and the categorization.

$$\Lambda^M(k, \lambda) = \frac{1}{n} \sum_{l=1}^{k} \sum_{h=1}^{g} n_l^{(h)} \frac{log\left(\frac{n_l^h n}{\sum_{i=1}^{k} n_i^{(h)} \sum_{i=1}^{g} n_l^{(i)}}\right)}{log(k \cdot g)}$$

where k is the number of clusters, g is the number of clusters given by the categorization, and $n_l^{(h)}$ is the number of objects in cluster C_l that are classified to be h. It is an external validation measure.

iv. Hubert's Γ Statistic. It is useful to measure the correlation between two association matrices X and Y, where $X(i,j)$ (or $Y(i,j)$) equals 1 if the pair of vectors (x_i, x_j) are in the same cluster, 0 otherwise.

$$\Gamma = \frac{1}{M}\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} X(i,j)Y(i,j)$$

Assuming that X and Y have binomial distribution it follows that:

$$a + b = \sum_{i=1}^{N-1}\sum_{j=i+1}^{N} X(i,j) \tag{5.1}$$

$$a + c = \sum_{i=1}^{N-1}\sum_{j=i+1}^{N} Y(i,j) \tag{5.2}$$

$$a = \sum_{i=1}^{N-1}\sum_{j=i+1}^{N} X(i,j)Y(i,j) \tag{5.3}$$

$$\tag{5.4}$$

Rewriting the Hubert's Γ statistic we have

$$\Gamma = \frac{Ma - (a+b)(a+c)}{\sqrt{(a+b)(a+c)(M-(a+b))(M-(a+c))}}$$

It is an external validation measure.

In this thesis we work only with Mutual information and Classification accuracy.

5.6 Parallel algorithm for Meta-clustering Algorithms

We parallelize the Bagged clustering proposed by Dudoit and Fridland [DF03], and Leisch [Lei99] in conjuntion with the MCLA algorithm and our proposed ensemble algorithms. The parallel algorithm is as follows:

i. The master process sends tasks to each slave process with a specific tag. Each tag determines if the work is a basic clustering algorithm or if it is a base algorithm of the Bagged algorithm.

ii. Each slave process receives a message from the master process with a specific tag. If the tag is to compute a base clustering algorithm, it sends its labels back to the master as a result. Otherwise, it computes the centers and the labels using a base clustering algorithm, sending them back to the master process.

iii. The master process collects the results according to the tags received. If there are more jobs, it sends another job to the slave process; otherwise it sends a message to each slave process to finish its work.

iv. Finally, the master process computes the ensemble by majority voting, and hierarchical clustering in its collected centers. The MCLA is computed from the base clustering algorithms.

The time complexity of the sequential compound algorithm is
$O(EnsembleAlgorithm) + O(B(O(FCM) + O(k^3)) + O(HC) + O(MCLA)$
where B is the number of bootstrap samples, $O(k^3)$ is the complexity of Hungarian algorithm to relabelling the output labels (k is the number of centers), $O(HC)$ is the time complexity of hierarchical clustering which is quadratic, and $O(MCLA)$ is the time complexity of the MCLA algorithm . Therefore the parallel time complexity is $(O(EnsembleAlgorithm) + B(O(FCM) + O(k^3)))/P + O(HC) + O(MCLA)$, where P is the number of processors.

5.7 Experimental Evaluation

We compare mutual information indices and accuracies of five basic clustering algorithms (EM, FCM, PAM, BIRCH, and DBSCAN) and four meta-clustering algo-

rithms (BAG1, BAG2, MCLA, and VOT). The data sets used for this experiment are described in Appendix A.

In the experimental evaluation we use the following parameters for each data set. The FCM, PAM, BIRCH and EM algorithms use the number of classes as the number of centers for each data set.

The DBSCAN algorithm uses $Eps = 1.6$ and $MinPts = 5$ for the Iris data set; $Eps = 0.18$ and $MinPts = 5$ for the Cassini data set; $Eps = 5.0$ and $MinPts = 7$ for the Breawst data set; and $Eps = 120$ and $MinPts = 50$ for the synthetic Parabola data set.

Bagging algorithm uses $B = 50$ as the number of bootstrap samples for the two Bagging algorithms.

Table 5.1: Accuracy and Mutual Information (MI) measures

	Cassini		Iris		Breastw		Parabola	
	Accuracy	MI	Accuracy	MI	Accuracy	MI	Accuracy	MI
EM	0.999	0.993	**0.967**	**0.899**	0.946	0.728	0.538	0.354
FCM	0.970	0.888	0.893	0.750	0.956	0.730	0.785	0.577
PAM	0.962	0.867	0.893	0.758	0.959	0.741	0.782	0.571
BIRCH	0.993	0.964	0.807	0.699	0.958	0.736	0.778	0.567
DBSCAN	**1.000**	**1.000**	0.687	0.725	0.820	0.709	**0.850**	**0.638**
BAG1	**1.000**	**1.000**	0.913	0.795	**0.964**	**0.744**	0.794	0.583
BAG2	0.971	0.889	0.907	0.782	0.957	0.735	0.781	0.570
MCLA	0.986	0.940	0.900	0.778	0.958	0.736	0.784	0.573
VOT	0.993	0.964	0.953	0.850	0.959	0.741	0.784	0.573

Now, we compare the performance of each clustering algorithm on each data set. Table 5.1 shows the classification accuracy and the mutual information measure of five base clustering algorithms and four ensemble clustering algorithms. These algorithms are ranked by their classification accuracy and mutual information index.

For the Cassini data set, the clustering algorithms are ranked as: BAG1 and DBSCAN first, EM second, our Voting and BIRCH third, MCLA fourth, FCM and BAG2 fifth, and PAM last. For this data set, the accuracy of DBSCAN and BAG1 is perfect, because this data set has instances that can be clearly separated in clusters.

The Iris data set has two overlapping classes, therefore the DBSCAN algorithm joins the second and third classes. The clustering algorithms are ranked as: EM first, our voting algorithm second, BAG1 third, MCLA and BAG2 fourth, FCM and PAM fifth, BIRCH sixth, and DBSCAN last.

For the Breawst data set, these algorithms are ranked as: BAG1 first, our voting algorithm, BAG2, BIRCH and MCLA, PAM and FCM second, EM third, and DBSCAN last.

Finally, the synthetic parabola data set has three separable classes. Therefore the clustering algorithms are ranked as DBSCAN first, BAG1 second, our voting algorithm, FCM, MCLA, PAM and BAG2 third, and BIRCH last.

Table 5.2 shows the run time and the speedup of the parallel algorithm for compound clustering algorithm. We cannot reach linear speedup because of the following reasons: 1) The data communication costs at the beginning of the computation. 2) The communication cost at each iteration in the Bagging algorithm. 3) The hierarchical clustering algorithm, the MCLA algorithm, and our VOT algorithm are made only by one process.

Table 5.2: Running time (Sec.) of Parallel Compound Clustering Algorithm for synthetic Parabola data set

# of processors	1	2	3	4	5	6	7	8	9	10
Run time (sec.)	780	676	425	350	305	255	215	188	170	160
Speedup	1.00	1.15	1.83	2.22	2.55	3.05	3.62	4.14	4.58	4.87

Chapter 6

Conclusions

6.1 Distance and Density based outliers

There are many algorithms for detecting outliers. Two of them are the LOF algorithm and Bay's algorithm, which have good performance both in runtime and outlier detection. We have designed parallel versions of these algorithms. In the first one, we exploit the nearest neighbor property of Bay's algorithm. In the parallel algorithm, each process runs the same Bay algorithm but locally in its own data set. Then, it sends its local neighbors to the master process. The master process receives the partial neighbors and computes the cutoff value and sends it to the slaves for the next iteration. This parallel algorithm reduces computational time of the algorithm.

On the other hand, the heavy work in the LOF algorithm resides in the computation of the k-distance nearest neighbors for each observation. In our proposed parallel LOF algorithm, the master process sends the entire data set to the slave processes, and assigns their tasks. Each slave process computes its respective k-distance nearest neighborhoods of its respective data, sending it back to the master. The master process receives the results from the slaves, and then computes the local reachability and LOF factor for each observation. This parallel algorithm also reduced considerably the computational time of this algorithm.

6.2 Visualization

We studied and analyzed drawbacks and limitation of two dimensional star coordinates. We enhanced this visualization technique, and extended it to three dimensions. We introduced new parameters to improve the star coordinate visualization, making this transformation one to one. These parameters are visualized with polyhedrons or polygons, using their original data values. Overlapping points are visualized on wire frame view or with different opacity. Our visualization algorithm performs well in supervised as well as unsupervised classification. We tested our visualization technique in the well-known Iris data set. In order to visualize large data sets, we proposed a parallel version of our algorithm. This parallel algorithm is based on data parallelism. It was tested in the Shuttle data set, reducing considerably the computational time. The speedup decreases because of the communication costs and the rendering process.

6.3 Meta-classifiers

We compared the performance of five base classifier algorithms (C4.5, Naive Bayes, KERNEL, KNN, and RBF), as well as their ensemble Bagging algorithms. We introduced a hybrid ensemble algorithm (combination of generated and non generated ensembles). This proposed algorithm gives good results, and is robust, compared to their single ones and other ensemble algorithms. The proposed parallel algorithm uses a simple task parallelism to distribute tasks equally to each process. Also, this parallel algorithm reduced the computation time of the proposed meta-classifier algorithm. The parallel algorithm proposed cannot reach linear speedup because communication costs and the voting process is made only by one process.

6.4 Meta-clustering

We compared the performance of five base clustering algorithms, using classification accuracy and cluster validation techniques. Also, a comparison of the ensembles generated by these clustering algorithms was carried out. The ensembles were based on Bagging, majority voting, and graph partitioning. We can conclude that each clustering algorithm performs better on a specific data set. For instance, the best performance of DBSCAN algorithm is obtained in a totally separated non convex data sets like Cassini and Parabola, while, in overlapping data sets, it performs poorly. Bagging algorithm BAG1 performs better than the other clustering algorithms almost in all data sets. Our voting algorithm performs quite well, and its performance is similar to BAG1. Also, BAG2 outperforms the single clustering algorithms. Therefore, we can conclude that combining base clustering algorithms improves the performance, and is more stable than single ones. The parallel algorithm proposed computes two Bagged clustering algorithms in conjunction with our voting based clustering algorithm and the MCLA algorithm. The computational time of this compound algorithm is reduced considerably with this parallel algorithm. The proposed parallel algorithm cannot reach linear speedup because communication costs and the serial processing of the hierarchical clustering, MCLA, and our voting algorithm.

Chapter 7

Future Work

Future work can be addressed in the following directions:

- Designing clustering algorithm ensembles is necessary, because these have more robustness, better quality, and performance than single clustering algorithms.

- Classification algorithms play an important role in the Data Mining process. New classification and clustering algorithms can be designed based on existing algorithms.

  The ensemble of more base classifiers can also be designed. For instance, applying learning algorithms to clean data gives better results than applying these algorithms to unclean data. It will be interesting to design an algorithm that performs cleaning and learning tasks at same time.

- New visualization methods can be designed from existing ones, and their performance must be compared with traditional methods. On the other hand, one can try to enhance visualization methods or combine them to get better results.

- Preprocessing is another challenge in Knowledge Discovery. Future work can be done in the design of an algorithm that works with oversetting, handling missing data, and detecting outliers and clusters in distributed data sets.

- Parallel and Distributed Computing is a necessary tool to ensure scalability to large databases, because these databases will continue to increase in size

in both the number of instances and the number of attributes. On the other hand, some databases are inherently distributed, because of privacy reasons or communication costs. Actual data mining algorithms do not work in distributed databases. Therefore, in order to work in a distributed environment the development and design of new parallel and distributed data mining algorithms is necessary.

Bibliography

[Agg01] Aggarwal, C.C. and Hineburg, A. and Kein, D. On the surprising behavior of distance metrics in high dimensional space. *Proceedings of the Eighth International Conference on Database Theory, Lecture Notes in Computer Science*, 1973:420434, *2001.*

[ALS+01] J. Ahrens, C. Law, W. Schroeder, K. Martin, and M. Papka. A Parallel Approach for Efficiently Visualizing Extremely Large, Time-Varying Datasets. *Los Alamos National Laboratory. Tech. Rep. LAUR-00-1620*, *2001.*

[Bez81] J. C. Bezdek. Pattern Recognition with Fuzzy Objective Function Algoritms. *Plenum Press, New York*, *1981.*

[Bis95] C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, *1995.*

[BKNS00] M Breuning, H. Kriegel, R. T. Ng, and J. Sander. LOF: Identifying density-based local outliers. *ACM SIGMOD International Conference on Management of Data*, pages 93–104, *2000.*

[BL94] V. Barnett and T. Lewis. *Outliers in Statistical Data.* John Wiley, New York, *1994.*

[BMNH] C. L. Blake, C. J. Mertz, D. J. Newman, and C. L. Hettich. UCI Repository of machine learning databases. *Irvine, CA: Univer-*

sity of California, Department of Information and Computer Science. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

[Bre94] L. Breiman. Bagging predictors. *Machine Learning, Technical Report. Department of Statistics, University of California, 1994.*

[BS03] S. D. Bay and M. Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 29–38, *2003.*

[CH67] T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *Institute of Electrical and Electronics Engineers Transactions on Information Theory*, 13:21–27, *1967.*

[DF03] S. Dudoit and J. Fridland. Bagging to improve the accuracy of a clustering procedure . *Bioinformatics*, 19:1090–1099, *2003.*

[Die00] T. Dietterich. Ensemble Methods in Machine Learning. *Lecture Notes in Computer Science*, 1857:1–15, *2000.*

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *JRSSB*, 38:1–38, *1977.*

[DT00] W. Duin and D. Tax. Experiments with Classifier Combining Rules. *Proceedings 1st International Workshop of Multiple Classifier System, Cagliari, Italy. Lecture Notes in Computer Science, Springer-Verlag*, 1857:16–29, *2000.*

[Dun73] J. C. Dunn. A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters. *Journal of Cybernetics*, 3:32–57, *1973.*

[EKSX96] M. Ester, P. Kriegel, J. Sander, and X. Xu. A Density-Based Algorithm for Discovery Clusters in Large Spatial Databases with Noise. *Published in*

Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining , pages 226–231, *1996*.

[FB03] B. Fischer and J. M. Buhmann. Bagging for Path-Based Clustering . *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 25(11):1411–1415, *2003*.

[FGW02] U. M. Fayyad, G. Grinstein, and A. Wierse. *Information visualization in Data Mining and Knowledge Discovery*. Morgan Kaufman Publishers, San. Francisco, *2002* .

[FJ02] A. Fred and A. K. Jain. Evidence Accumulation Clustering Based on the K-Means Algorithm . *Proceeding Structural, Syntactic, and Statistical Pattern Recognition, LNCS*, 2396:442–451, *2002*.

[FS96] Y. Freund and R. Schapire. Experiments with a new booosting algorithm. *Machine Learning, Proceedings of the thirteenth International Conference. San Francisco, Morgan Kauffman*, pages 148–156, *1996*.

[GTBC04] T. Greene, A. Tsymbal, N. Bolshakova, and P. Cunningham. Ensemble Clustering in Medical Diagnostics. *17th IEEE Symp. on Computer-Based Medical Systems. Bethesda, MD, National Library of Medicine/National Institutes of Health, IEEE CS Press*, pages 576–581, *2004*.

[Han98] D. J. Hand. Data mining: statistics and more ? *The American Statistician*, 52:112–118, *1998*.

[Haw80] D. Hawkins. *Identification of Outliers*. Chapman and Hall, London, *1980*.

[HC04] E. Hung and D. Cheung. Parallel Mining of Outliers in Large Database. *Distributed and Parallel Databases*, 12(1):5–26, *2004*.

[HGP01] P. Hoffman, G. Grinstein, and D. Pinkney. Visualizing multi-dimensional clusters trends, and outliers using star coordinates. *Proc. ACM SIGKDD, New York, NY, USA*, pages 107–116, *2001*.

[HR04] J Hardin and D. M. Rocke. Outlier Detection in the Multiple Cluster Setting using the Minimum Covariance Determinant Estimator. *Computational Statistics and Data Analysis*, 44:625–638, *2004*.

[JKP03] E. Januzaj, H. P. Kriegel, and M. Pfeifle. Towards Effective and Efficient Distributed Clustering. *Workshop on Clustering Large Data Sets, 3rd Int. Conf. on Data Mining (ICDM'03), Melbourne, FL*, pages 49–58, *2003*.

[KAKS97] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: Applications in VLSI domain . *In Proceedings of the 34th Conference, Design automation, ACM*, pages 526–529, *1997*.

[Kan01] R. Kandogan. Visualizing Multi-Dimensional Clusters Trends, and Outliers using Star Coordinates. *Proceedings ACM SIGKDD '01*, pages 107–116, *2001*.

[KK95] G. Karypis and V. Kumar. Analysis of Multilevel Graph Partitioning. *Technical Report 95-037 Published in Supercomputing*, *1995*.

[KN98] E. Knorr and R. Ng. Algorithms for mining distance-based outliers in large datasets. *In Proc. 24th Int. Conf. Very Large Data Bases VLDB*, pages 392–403, *1998*.

[KNT00] E. Knorr, R. Ng, and V. Tucakov. Distance-based outliers: Algorithms and applications. *VLDB Journal: Very Large Data Bases*, 8:237–253, *2000*.

[KR90] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley, New York, *1990*.

[LA05] E. Lozano and E. Acuña. Parallel algorithms for distance-based and density-based outliers . *Proceeding of the Fifth IEEE International Conference on Data Mining, Houston, Texas, USA*, pages 729–732, *2005* .

[Lei99] F. Leisch. Bagged Clustering. *Working paper Series No 51. Adaptative Information Systems and Modelling in Ecnomics and Management sci-*

ence, Institut für Informations verarbeitung, Abt. Produktions management, Wien, 1999.

[Loz03] E. Lozano. Density Estimation by Kernel and its Applications using Parallel Programming. *M.S. thesis at Mathematics Department UPR Mayagüez, 2003.*

[LZY05] B. Long, Z. Zhang, and P. Yu. Combining Multiple Clustering by Soft Correspondence . *Published on International Conference on Data Mining ICDM' 05*, pages 282–289, *2005* .

[NH94] R. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. *Proc. 20th Int. Conf. on Very Large Databases. Morgan and Kaufmann Publishers, San Francisco*, 8:44–155, *1994.*

[Pac97] P. Pacheco. *Parallel Programming with MPI.* Morgan Kauffmann Publishers Inc., *1997.*

[PCS00] A. Prodromidis, P. Chan, and S. Stolfo. Meta-learning in distributed data mining systems: Issues and Approaches. *In Advances in Distributed and parallel knowledge discovery. Chapter 3, AAAI/MIT Press, 2000.*

[Qui93] R. Quinlan. C4.5 A Program for Machine Learning. *Morgan Kaufmann Series in Machine Learning, 1993.*

[RA01] J. Ruoming and G. Agrawal. A Middleware for developing parallel data mining applications. *In Proceedings of the First SIAM Conference on Data Mining, Chicago IL, April, 2001.*

[RGV01] F. Roli, G. Giacinto, and G. Vernazza. Methods for Designing Multiple Classifier Systems. *Proc. of MCS 2001, Cambridge, UK, LNCS 2096 (Kittler and Roli Eds.), Springer*, pages 78–87, *2001.*

[RL87] P. J. Rousseeuw and A. Leroy. *Robust Regression and Outlier Detection.* John Wiley, *1987.*

[Rod04] C. Rodriguez. A Computational Environment for Data Preprocessing in Supervised Classification. *M. S. thesis at Mathematics Department UPR Mayagüez, 2004.*

[RRS00] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. *In Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 427–438, *2000.*

[RW02] D. Rocke and D. Woodruff. Computational Connections Between Robust Multivariate Analysis and Clustering. *In COMPSTAT 2002 Proc. in Computational Statistics, Wolfgang Härdle and Bernd Rönz eds. Heidelberg: PhysicaVerlag*, pages 255–260, *2002.*

[SG02] A. Strehl and J. Ghosh. Cluster Ensembles - A Knowledge Reuse Framework for Combining Partitionings. *Journal of Machine Learning Research*, 3:583–617, *2002.*

[Ski01] D. B. Skillicorn. Strategies for Parallel Data Mining. *IEEE Concurrency*, 4(7):36–35, *2001.*

[SML96] W. J. Schroeder, K. M. Martin, and E. M. Lorensen. An Object Oriented Approach to 3D Graphics. *Prentice Hall, 1996.*

[TG00] K. Tumer and J. Ghosh. Robust Order Statistics-based ensembles for distributed data mining. *Advances in Distributed and parallel knowledge discovery. AAAI/MIT Press*, pages 185–210, *2000.*

[TJP03] A. Topchy, A. K. Jain, and W. F. Punch. Combining Multiple Weak Clusterings. *Proceedings IEEE Intl. Conf. on Data Mining 2003, Melbourne, Fl*, pages 331–338, *2003.*

[TK99a] H. Theisel and M. Kreusel. An Enhanced Spring Model for Information Visualization. *In Proceedings Eurographics Ferreira and M. Gobel*, 17(3):335–344, *1999.*

[TK99b] S. Theodoridis and K. Kotroumbas. Pattern Recognition. *Academic Press , 1999 .*

[VM02] G. Valentini and F. Masulli. Ensembles of Learning Machines. *in M. Marinaro and R. Tagliaferri, editors, Neural Nets WIRN Vietri-02, Series Lecture Notes in Computer Sciences, Springer-Verlag, Heidelberg (Germany), Springer-Verlag, Heidelberg (Germany) (invited review)*, 2486:3–19, *2002.*

[WA99] B. Wilkinson and M. Allen. *Parallel Programming Techniques and Applications using Networked Workstations and Parallel Computers.* Prentice-Hall, Inc, *1999.*

[Weg90] E. J. Wegman. Hyperdimensional Data Analysis Using Parallel Coordinates. *Journal of the American Statistical Association*, 85(411):664–675, *1990.*

[ZB04] X. Zhang and C. E. Brodley. Solving Cluster Ensemble Problems by Bipartite Graph Partitioning . *Proceedings of the 21 st International Conference on Machine Learning, Banff, Canada* , page 36, *2004* .

[ZRL96] T. Zhang, R. Ramakrishnan, and M. Livni. BIRCH: an efficient data clustering method for very large databases . *Proc. ACM-SIGMOD Int. Conf Management of Data, Montreal, Canada* , pages 103–114, *1996* .

[ZTGG02] Y. Zeng, J. Tang, J. Garcia-Frias, and G. R. Gao. An Adaptive Meta-clustering Approach: Combining the Information from Different Clustering Results. *Bioinformatics Conference*, 00:276, *2002.*

Appendix A

Data sets and cluster description

A.1 Data sets used in this thesis

In the experimental evaluation we consider data sets, which are available in the Machine Learning database repository at the University California, Irvine [BMNH]. These data sets are the following:

- Breastw. This data set comes from the University of Wisconsin Hospital. It contains clinical cases, which were collected in the period 1989-1991. This data set contains 699 instances; each instance has 9 continuous attributes and one categorical attribute with two classes (benign and malignant). In this thesis, we use only 683 instances, because 16 instances have missing values.

- Bupa. This medical data set was created by BUPA Medical Research Ltd. It has 7 attributes: 1. mcv (mean corpuscular volume). 2. alkphos (alkaline phosphotase) 3. sgpt (alamine aminotransferase). 4. sgot (aspartate aminotransferase) 5. gammagt (gamma-glutamyl transpeptidase) 6. drinks (number of half-pint equivalents of alcoholic beverages drunk per day). 7. selector field used to split data into two sets. It has 345 instances and two classes.

- Cassini. This data set comes from the R package called "CLUE". It consists of 1000 points in a 2-dimensional space which are drawn from the uniform distribution in 3 structures. The two outer structures are banana-shaped; the "middle" structure between them is a circle (Figure A.1).

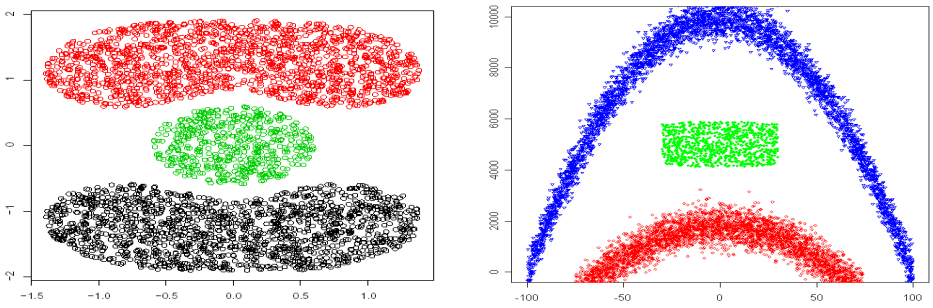

Figure A.1: a) Cassini data set. b) Synthetic Parabola data set.

- Census. This data set consists of 14 features measured in 32561 subjects. The features try to classify the subjects in two classes: "income $<$ 50k", and "income $\geq$ 50k". A 7% of subjects have incomplete information and we have deleted them.

- Covtype. This data set comes from the US Forest Service (USFS). It contains measures of seven types of soils with 581,012 instances, and 54 attributes.

- Diabetes. This data set is also known as the Pima Indians Diabetes data set. It was introduced by National Institute of Diabetes and Digestive and Kidney Diseases. The diagnostic, binary-valued variable investigated is whether the patient shows signs of diabetes. All patients are females at least 21 years old of Pima Indian heritage. This data set has 8 attributes all numeric-valued, 768 instances, and two classes.

- Ionosphere. This data set comes from Johns Hopkins University. This radar data was collected by a system in Goose Bay, Labrador, Nfld, Canada. This system consists of a phased array of 16 high-frequency antennas with a total transmitted power in the order of 6.4 kilowatts. The targets were free electrons in the ionosphere. "Good" radar returns are those showing evidence of some type of structure in the ionosphere. "Bad" returns are those that do not; their signals pass through the ionosphere. There were 17 pulse numbers for the Goose Bay system. Instances in this database are described by two attributes

per pulse number.

The original data set contains 34 predictors, but the two first features were eliminated, because the first feature has the same value in one of the classes and the second feature assumes the value 0 in all observations. The data set contains 351 instance and 2 classes: "Good", and "Bad".

- Iris. This data set contains 3 classes with 50 instances in each class. The class refers to a variety of iris: Iris Setose, Iris Versicolor, and Iris Virginica.

- Landsat. It consists of 4435 instances, each instance represents a satellite image with 36 features. The data is classified in six classes.

- Parabola. Synthetic data set, generated from two concentric parabolas (4000 observations each) and one ellipse (1000 observations). These observations come from a three normal distribution with zero mean and constant variance.

- Segment. This Image Segmentation data was created by the Vision Group, University of Massachusetts. The instances were drawn randomly from a database of 7 outdoor images. The images were hand segmented to create a classification for every pixel. Each instance is a 3x3 region. It has 19 continuous attributes, 2310 instances and 7 classes.

- Shuttle. The NASA Shuttle (Catlett, 1991) contains 43500 training instances, with 9 continuous attributes. It has 7 classes.

- Vehicle. This data set comes from the Turing Institute at Glasgow, Scotland. Originally gathered by JP Siebert 1986-87. The original purpose was to find a method of distinguishing 3D objects within a 2D image by application of an ensemble of shape feature extractors to the 2D silhouettes of the objects. It has 18 continuous attributes, 846 instances, and 4 classes.

Table A.1: Data set Description

Data set	Instances	Classes	Features
Breastw	683	2	9
Bupa	345	2	6
Cassini	1000	3	2
Census	32561	2	14
Covtype	581012	7	54
Diabetes	768	2	8
Ionosphere	351	2	32
Iris	150	3	4
Landsat	4435	6	36
Parabola	9000	3	2
Segment	2310	7	16
Shuttle	43500	7	9
Vehicle	846	4	18

A.2 Cluster Description

The computer environment where we tested our programs consists of two different clusters.

i. A cluster of 4 nodes HP Itanium 2 6M zx6000 (IA64 Architecture). Each node has 2 processors running at 1.5 GHz with 4 GB of main memory, each running Red Hat Advanced Workstation 2.1. We have implemented our algorithms in C++ (gcc version 3.2.3) using some libraries of LAM MPI version 7.0 [Pac97].

ii. A cluster called espresso.hpcf.upr.edu, which is a distributed memory Linux cluster (Atipa Model), This cluster has Intel Pentium4 Xeon processors 1with 2.4 GigaHertz. It is composed of 85 compute nodes (total number of processors is 172) + 1 master node. It uses a Gigabit Ethernet backplane. The total amount of memory is 86 Gigabytes. The total amount of disk space is (85x40 Gigabytes) + (1x600Gigabytes). The current Operating System is SDSC's Rocks clustering system, with compiler gcc version 3.2.3.

Appendix B

Parallel outlier detection algorithms

B.1 Parallel Algorithms to detect outliers

In this section we discuss the input parameters of the two parallel algorithms for outlier detection.

B.1.1 Program parameters

The following input parameters are used for these programs:

$-pbay\_dfile$

The name of the input file

$-pbay\_rows$

The number of rows of the input data

$-pbay\_cols$

The number of columns of the input data

$-plof\_dfile$

The name of the input file

$-plof\_rows$

The number of rows of the input data

$-plof\_cols$

The number of columns of the input data

The distribution of the source code for these programs is available on demand (please send an email to elozanoi@gmail.com). The binaries can be built by executing the following commands:

> tar -zxvf ElioLozano_PhdThesis_Source.tar.gz

> cd ElioLozano_PhdThesis_Source/baylof

> make

These commands will generate two executables: pbay and plof. For instance, one of these binaries can be executed as:

> mpirun -np 8 ./pbay -pbay_dfile dfile -pbay_rows x1 -pbay_cols x2

where $dfile$ is the input file, x1 and x2 are the number of rows and columns of the data set.

Appendix C

Data visualization

Now, we give details of the implementation of the visualization algorithm.

C.1 VTK objects used in 3D star coordinate algorithm

In this section we discuss the classes used in the 3D star coordinate algorithm.

- vtkPoints. It represents 3D points. The data model for vtkPoints is an array of vx-vy-vz triplets accessible by (point or cell) id.

- vtkVoxel. It is a concrete implementation of vtkCell to represent a 3D orthogonal parallelepiped. Unlike vtkHexahedron, vtkVoxel has interior angles of 90 degrees, and sides are parallel to coordinate axes. This results in large increases in computational performance.

- vtkUnstructuredGrid. It is a data object that is a concrete implementation of vtkDataSet. vtkUnstructuredGrid represents any combinations of any cell types. This includes 0D (e.g., points), 1D (e.g., lines, polylines), 2D (e.g., triangles, polygons), and 3D (e.g., hexahedron, tetrahedron).

- vtkAppendFiler. This class is a filter that appends one of more datasets into a single unstructured grid. All geometry is extracted and appended, but point attributes (i.e., scalars, vectors, normals, field data, etc.) are extracted and

appended only if all datasets have the point attributes available. (For example, if one dataset has scalars but another does not, scalars will not be appended.)

- vtkRenderer. It provides an abstract specification for renderers. A renderer is an object that controls the rendering process for objects. Rendering is the process of converting geometry, a specification for lights, and a camera view into an image. vtkRenderer also performs coordinate transformation between world coordinates, view coordinates (the computer graphics rendering coordinate system), and display coordinates (the actual screen coordinates on the display device). Certain advanced rendering features such as two-sided lighting can also be controlled.

- vtkRenderWindows. This class is an abstract object to specify the behavior of a rendering window. A rendering window is a window in a graphical user interface where renderers draw their images. Methods are provided to synchronize the rendering process, set window size, and control double buffering. The window also allows rendering in stereo. The interlaced render stereo type is for output to a VRex stereo projector. All of the odd horizontal lines are from the left eye, and the even lines are from the right eye. The user has to make the render window aligned with the VRex projector, or the eye will be swapped.

- vtkOutputPort. It connects the pipeline in this process to one in another process. It communicates all the pipeline protocol so that the fact you are running in multiple processes is transparent. The output port is placed at the end of the pipeline (an output for a process). It can have multiple corresponding input ports in other processes that receive its data. Updates in a port are triggered asynchronously, so filter with multiple inputs will take advantage of task parallelism.

- vtkInputPort. It connects the pipeline in this process to one in another process. It communicates all the pipeline protocol so that the fact that it is running in

multiple processes is transparent. An input port is used as a source (input to a process). One is placed at the start of a pipeline, and has a single corresponding output port in another process (specified by RemoteProcessId).

The implementation of 3D star coordinates was in C++, using visualization toolkit VTK. In the next section we discuss the library dependencies needed by this implementation.

C.2 Library dependencies

In order to build the 3D star coordinate binaries, it is necessary to install the following libraries:

- TCL/TK libraries (`http://www.tcl.tk/software/tcltk/`). Once these libraries are downloaded, follow the installation steps. Usually these steps are:
  # > ./configure
  # > make
  # > make install

- VTK library [SML96](`http://public.kitware.com/VTK/`)

  We have used library versions 4.0 and 4.2. This library is installed using cmake utility. This library can be installed executing the following commands:
  # > cmake -i
  On the cmake utility window type "configure"
  Then, on the same window type "generate"
  Finally, type "q" to quit from the cmake utility
  # > make
  # > make install

C.3 Building Binaries

Now it is time to build the star coordinate binaries. These binaries are generated executing the following commands:

> tar -zxvf ElioLozano_PhdThesis_Source.tar.gz

> cd ElioLozano_PhdThesis_Source/graphics

> cmake .

> make

These commands will generate two executables: star and pstar.

C.4 Program parameters

The followings are the program parameters used in the visualization algorithm.

$-star\_dfile$

The name of the input file

$-star\_rows$

Number of rows of the input data

$-star\_cols$

Number of columns of the input data

$-star\_factor$

A constant factor to set the size of the polyhedron

For instance, one of these binaries can be executed as:

> mpirun -np 8 ./pstar -star_dfile dfile -star_rows x1 -star_cols x2 -star_factor f

where $dfile$ is the input file, x1 and x2 are the numbers of rows and columns of the data set, and f is real number used to scale the size of the objects.

Appendix D

Meta-classifier

D.1 Class hierarchy

The class hierarchy of the implementation of the meta-classifier algorithm is given below:

$SupervisedClassifier$

$|$

$\vdash RBF$

$\vdash KERNEL$

$\vdash KNN$

$\vdash C45$

$\vdash BAYES$

$|$

$\vdash MetaClassifier$

D.2 Libraries and programs used

The proposed classifier ensemble algorithm was implemented using the following libraries:

i. C4.5 Decision trees. We have written a C++ wrapper for the C4.5 C source code. The C4.5 software was developed by Quinlan (Copyright © J.R. Quinlan, 1987, 1988, 1989, 1990, 1991, 1992). It is available on the author's web page.

ii. CPPLapack. It is a C++ class wrapper for BLAS and LAPACK. We have used this wrapper to find the pseudo inverse of a matrix, calling general svd procedures from LAPACK library. This wrapper is available on `http://cpplapack.sourceforge.net/`.

D.3 Building Binaries

The binaries of the meta-classifier program can be built executing the following commands:
> tar -zxvf ElioLozano_PhdThesis_Source.tar.gz
> cd ElioLozano_PhdThesis_Source/ClassEnsemble
If you want to make all the programs:
> make
If you want to make stand alone programs:
> make program_name

where program_name is the name of the stand alone program (mclass, rbf, kernel, knn, c45, bayes)
The binary of the parallel algorithm is mclass.

D.4 Program parameters

The input parameters for the meta-classifier program are the following:

$-mclass\_dfile$

The name of the input file

$-mclass\_rows$

Number of rows of the input data

$-mclass\_cols$

Number of columns of the input data

$-mclass\_nclass$

The number of classes

$-rbf\_nhidden$

The number of hidden layers

$-c45\_dnames$

The data names file as in C4.5 algorithm

$-knn\_kneigh$

The number of k-neighbors

For instance, these binaries can be executed as:

> mpirun -np 8 ./mclass -mclass_dfile dfile -mclus_rows x1 -mclus_cols x2 -mclass_nclass nc -mclus_nhidden nh -mclus_kneigh kn

where *dfile* is the input file, x1 and x2 are the number of rows and columns of the data set, nc is the number of classes, nh is the number of hidden layers, and kn is the number of k-nearest neighbors.

Appendix E

Meta Clustering

E.1 Class hierarchy

The class hierarchy of the implementation of meta clustering algorithm is given below:

$UnsupervisedClassifier$

$|$

$\vdash EM$

$\vdash PAM$

$\vdash DBSCAN$

$\vdash BIRCH$

$\vdash FCM$

$|$

$\vdash MetaClustering$

E.2 Libraries and programs used

In order to implement the proposed clustering ensemble algorithm, we have written wrappers for existing libraries. These libraries and programs are listed below.

i. DBSCAN. We used ltilib library to implement the DBSCAN algorithm. This library was implemented by Pablo Alvarado, Ulle Canzler, Jochen Wickel, and Suat Akyol (Copyright © 1998-2005 by Chair of Technical Computer Science, RWTH-Aachen University). It is available at `http://ltilib.sourceforge.net/doc/homepage/index.shtml`.

ii. PAM. We wrote a C++ wrapper to implement the PAM algorithm. This wrapper calls C source code (translated from fortran with f2c translator) available in R cluster package.

This package was first developed by Peter Rousseeuw, Anja Struyf and Mia Hubert (Swiss Federal Institute of Technology in Zurich), posteriorly, it was extended and maintained by Martin Maechler. This package is available at the CRAN web page (`http://cran.r-project.org/`).

iii. EM. We have written a C++ wrapper to implement EM algorithm. This wrapper calls Fortran code available in the MCLUST R package. This package was developed by Fraley, Raftery, and Ron Wehrens (Copyright © 1991-2005 Dept. of Statistics, University of Washington). It is also available on the CRAN web page and at `http://www.stat.washington.edu/mclust`.

iv. BIRCH. We have used the Birch library to implement the Birch algorithm. This library was developed by Tian Zhang (Copyright © 1995 CS Dept., Univ. of Wisconsin-Madison). The original source code is available at `http://www.cs.wisc.edu/ vganti/birchcode/`

v. FCM. We wrote a C++ wrapper to implement the fuzzy C-means algorithm. This wrapper calls C source code written by Kurt Hornik (Copyright © by all authors of e1071 R package, Vienna University of Technology). This package is available at the CRAN web page.

vi. MCLA. We have used the Metis library to implement the MCLA algorithm. This library was developed by George Karypis (Copyright © 1998, Regents

of the University of Minnesota). This library is available on the author's web page.

E.3 Building Binaries

The binaries of the meta-clustering program are generated executing the following commands:

> tar -zxvf ElioLozano_PhdThesis_Source.tar.gz

> cd ElioLozano_PhdThesis_Source/ClusterEnsemble

If you want to make all programs:

> make

If you want to make a stand alone program:

> make program_name

where program_name is the name of the stand alone program (mclus, em, pam, dbscan, birch, cmeans). The binary of the parallel compound program is mclus.

E.4 Program parameters

The following input parameters are used for the meta-clustering program.

$-mclus\_dfile$

The name of the input file

$-mclus\_rows$

The number of rows of the input data

$-mclus\_cols$

The number of columns of the input data

$-mclus\_nclus$

The number of clusters

−em_maxiter

The maximum number of iterations

−dbscan_minpts

The minimum number of points in a core point's ξ-neighborhood

−birch_para

The parameter file, which is needed by the Birch algorithm

−birch_scheme

The scheme file, which is needed by the Birch algorithm

−birch_proj

The project file, which is needed by the Birch algorithm

For instance, one of these binaries can be executed as:

```
# > ./pam -pam_dfile dfile -pam_rows x1 -pam_cols x2 -pam_nclus c
```

where $dfile$ is the input file, x1 and x2 are the numbers of rows and columns of the data set, and c is the number of clusters.