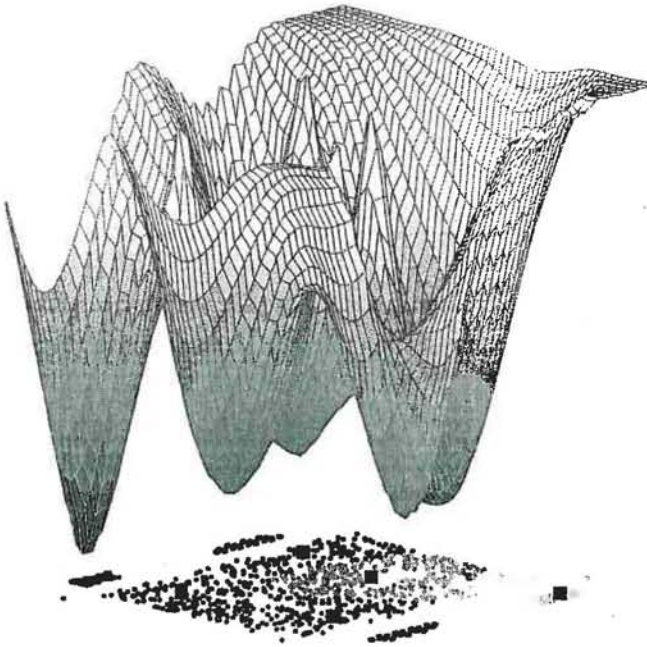


Application of nonsmooth optimisation to data analysis

Thesis by
Julien Ugon

Submitted to the Degree of Doctor of Philosophy



A DISSERTATION

Presented to the School of Information Technology and Mathematical
Sciences

University of Ballarat, Australia

P.O. Box 663

Gear Avenue, Mount Helen

Ballarat, Victoria 3353

Australia

Abstract

Analysing current phenomena and forecasting future ones have always been at the centre of human preoccupations, with strong economical and political impacts. Today, needs becoming too complex and involving too many parameters do not allow the construction of analytical models. Therefore, data analysis, as an alternative, is gaining in importance.

In data analysis measurements are collected. Then an analysis is carried out, in order to extract information, or to forecast the behaviour of unknown data.

While in analytical modelling the result is verified against measurements after being constructed, in data analysis, the model is built on the data: that is, results should be optimised to give the best accuracy: one wants the “*best*” fitting parameters, or the *most* accurate forecasts. Therefore it is natural to reformulate data analysis problems as mathematical optimisation problems, and solve them using adapted tools.

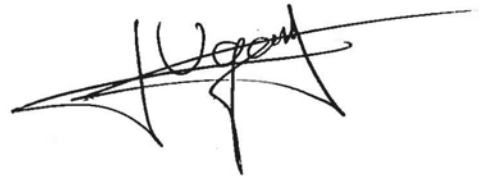
Recent technological progress has boosted the development in both fields: cheaper storing devices allow the collection of large amounts of data, while faster processing power enable the implementation of powerful algorithms. Consequently, while a considerable amount of research exists in both fields, the application of one to solve the problems arising in the other is still in a developmental stage.

The research presented in this thesis is two-fold: on the one hand, major data mining problems are reformulated as mathematical programming problems. These problems should be carefully designed, since from their formulation depends the efficiency, perhaps the existence, of the solvers. On the other hand, optimisation methods are adapted to solve these problems, most of which are nonsmooth and nonconvex. This part is delicate, as the solution is often required to be good and obtained fast. Numerical experiments on real-world datasets are presented and analysed.

Statement of Authorship

This thesis contains no material which has been accepted for the awards of any other degree or diploma in any university and is less than 100,000 words in length excluding tables, maps, footnotes, bibliographies and appendices. To the best of my knowledge and belief this thesis contains no material previously published by any other person except where due acknowledgement has been made.

Julien Ugon
March 2005

A handwritten signature in black ink, appearing to read 'Julien Ugon', with a long horizontal line extending to the right from the end of the signature.

Acknowledgements

Firstly and foremost I would like to thank my supervisor Prof. Rubinov and my associate supervisor Dr. Bagirov. Their availability, their guidance and their enthusiasm have been extremely helpful all along this time as their student.

I am also grateful to the various colleagues I worked with: Dr. Jia, Ms. Kouhbor, Dr. Mammadov, Dr. Soukhoroukova, Pr. Yearwood, Dr. Zhang, among others. A good working environment made working on this PhD a pleasure. I also thank all the anonymous referees who read my papers and provided useful comments. I would also like to thank Fred Ross, who read this thesis and corrected many of my grammatical mistakes.

On a personal level, I am very grateful to my family and friends in France and in Australia, who provided support and understanding: my parents, my brother and sister, Nadya, Tai and Olf, Cindy and David, Zari, and many more.

Finally I would like to thank the University of Ballarat for providing financial support for this research through the International Postgraduate Scholarship.

List of Publications

The work presented in this thesis is based, in parts, on the following papers written by the author:

- A. Rubinov, N. Soukhoroukova, and J. Ugon. A feature selection for clustering. In *Proceedings of ICOTA 6*, Australia, December 2004. CIAO, University of Ballarat.
- A. Bagirov and J. Ugon. An algorithm for solving large scale optimisation problems. submitted.
- A. Bagirov and J. Ugon. Separation of two sets by piecewise linear function. *Applied and Computational Mathematics*, 3(2):117–131, 2004.
- A. Bagirov and J. Ugon. Max-min separability. In *Nonconvex Optimization and Its Applications*, Kluwer Bookseries. Kluwer, 2005. To appear.
- A. Bagirov and J. Ugon. A method for minimising the clustering function. *Journal of Optimization*, 2005. To appear.
- L. Clemow, M. Hannah, D. Nash, M. Mamedov, A. Rubinov, J. Ugon, and J. Yearwood. Feature ordering for small datasets. In L. Caccetta, editor, *Industrial Optimization*. Kluwer Academic, 2005. To appear.
- A. Rubinov, N. Soukhoroukova, and J. Ugon. Minimization of a class of sum of max-min functions. In Jeyakumar and Rubinov, editors, *Continuous Optimization: Current Trends and Applications*. Springer, 2005. To appear.
- A. Rubinov, N. Soukhoroukova, and J. Ugon. Minimization of the sum of minima of convex functions. In *Proceedings of the Continuous Optimization and Optimal Control Workshop*, Melbourne, December 2003.
- A. Rubinov, N. Soukhoroukova, and J. Ugon. A new algorithm to find a shape of a finite set of points. In L. Caccetta and V. Rehbock, editors,

Industrial Optimization, Vol 1, Proceedings of Symposium in Industrial Optimization, Curtin University, Perth, Australia, 2003. Western Australian Centre of Excellence in Industrial Optimisation (WACEIO).

- A. Rubinov, N. Soukhoroukova, and J. Ugon. Classes and clusters in data analysis. *European Journal of Operations Research*, 2005. To be published.

Contents

Acknowledgements	iii
List of publications	iv
Contents	vi
List of Algorithms	xi
List of Figures	xii
List of Tables	xiv
Notations	xvi
Introduction	1
I Methodology	4
1 Data analysis	5
1.1 Introduction	5
1.1.1 Definitions and notations	5
1.1.2 Classification (supervised learning)	7
1.1.3 Clustering (unsupervised learning)	7
1.1.4 Preprocessing	8
1.2 Clustering (unsupervised learning)	9
1.2.1 Mathematical formulation	10
1.2.2 Optimisation problem	14
1.2.3 k -methods	15
1.2.4 Optimisation methods	18
1.2.5 Other methods	19
1.2.6 Evaluation of the clustering	19

1.3	Classification (supervised learning)	21
1.3.1	Evaluation of the algorithm	22
1.3.2	Non-optimisation based approaches	22
1.3.3	Optimisation methods in supervised learning	23
1.3.4	Linear separability	24
1.3.5	Support vector machine	25
1.3.6	Polyhedral separability	28
1.3.7	Classification via clustering	29
1.3.8	Max-min separability	30
1.4	Preprocessing	35
1.4.1	Scaling	36
1.4.2	Feature selection	38
1.4.3	Dataset reduction	39
1.5	Test datasets	41
1.5.1	Supervised learning	41
1.5.2	Unsupervised learning	41
1.6	Summary	42
2	Optimisation methods	44
2.1	Generalities	44
2.1.1	Presentation	44
2.1.2	Notions of mathematical analysis	46
2.2	Analytical considerations	49
2.2.1	Preliminaries	49
2.2.2	Clarke subdifferential	51
2.2.3	Demyanov-Rubinov quasidifferential	52
2.3	Descent methods	53
2.4	Numerical methods	54
2.4.1	Cutting planes method	54
2.4.2	Bundle methods	55
2.4.3	Discrete gradient method	55
2.4.4	Other methods	61
2.5	Global methods	61
2.6	Heuristic methods: simulated annealing	62
2.7	Hybrid methods	64
2.7.1	Global method as an escape method	64
2.7.2	Global method as an improvement of the local search	65
2.8	Large scale optimisation	66
2.9	Choice of the method utilised	67

II	Computational considerations	69
3	Solving large scale nonsmooth optimisation	70
3.1	Introduction	70
3.2	Piecewise partially separability: Definitions and examples . . .	71
3.2.1	Chained and piecewise chained functions	72
3.2.2	Piecewise separable functions	73
3.3	Properties of piecewise partially separable functions	74
3.4	Minimisation of piecewise partially separable functions	80
3.4.1	Remarks on the discrete gradient	82
3.4.2	Discrete gradients of sum of max min functions	82
3.5	Numerical experiments	85
3.5.1	Test problems	85
3.6	Conclusion	93
4	Minimisation algorithms for data analysis problems	95
4.1	Introduction	95
4.2	Solving the Max-Min separation problem	95
4.2.1	Statement of problem	96
4.2.2	Differential properties of the objective function	97
4.2.3	Discrete gradients of the objective function	97
4.3	Solving the clustering problem	99
4.3.1	Differential properties of the objective functions	99
4.3.2	Discrete gradients of the objective function	99
III	Data analysis methods	102
5	Classification using Max-Min separability	103
5.1	Supervised data classification via max-min separability	103
5.2	Results on large datasets	105
5.2.1	Datasets	105
5.2.2	Results and discussion	105
5.3	Restrictions to the current model	107
5.4	Conclusion	108
6	Solving the clustering problem	109
6.1	Introduction	109
6.2	A class of sum-min functions	109
6.2.1	Sum of minima of convex functions	109
6.2.2	Some properties of SMC functions	110

6.3	Step by step approach to clustering	111
6.3.1	An algorithm for solving the intermediate problem	112
6.3.2	Results of numerical experiments	114
6.4	Simultaneous clustering	118
6.4.1	Minimisation of SMC functions	119
6.4.2	Minimisation of the generalised cluster function	121
6.4.3	Numerical experiments	124
6.4.4	Skeletons	131
6.5	Conclusion	139
6.5.1	Optimisation	139
6.5.2	Clustering	140
7	A feature selection algorithm	142
7.1	Introduction	142
7.2	A feature selection approach	143
7.2.1	Unsupervised learning	143
7.2.2	Supervised learning	145
7.3	Numerical experiments	145
7.3.1	Unsupervised learning	145
7.3.2	Supervised learning	147
7.4	Conclusion	148
8	Study of the relations between clusters and classes	150
8.1	Introduction	150
8.2	Pendigits	151
8.2.1	Classes and centres	151
8.2.2	Classes and cluster structures	152
8.2.3	Summary	153
8.3	Letters	155
8.3.1	Classes and centres	155
8.3.2	Classes and cluster structures	158
8.4	Conclusion	160
9	Shape of a finite set of points	165
9.1	Introduction	165
9.2	Geometrical approximation for a finite set of points	165
9.2.1	Sets and shapes	165
9.2.2	Positive definite symmetric matrix approach	166
9.2.3	Positive diagonal matrix approach(PDMA)	166
9.2.4	Structures	167
9.3	Algorithm	168

9.3.1	Find the centre	168
9.3.2	Elimination of the isolated points	168
9.3.3	Finding the shape	169
9.4	Numerical experiments: comments and descriptions	169
9.4.1	Subsets	169
9.4.2	Point elimination	170
9.4.3	Shape of the sets	171
9.5	Conclusion	171
Conclusion and further research		172
Conclusion		172
Further research		174
Index		177
Bibliography		179

List of Algorithms

1.1	The k -methods	16
1.2	Scaling by average	36
1.3	Scaling by average and variance	38
1.4	Scaling in an interval	38
1.5	ϵ -cleaning procedure	40
2.1	A generic descent method	54
2.2	DG: finding a descent direction: Clarke subdifferentials version	59
2.3	Discrete gradient method	60
2.4	Simulated annealing algorithm	63
2.5	Hybrid method of type 1	65
2.6	Multidirectional search descent algorithm	66
6.1	An algorithm for solving the clustering problem	111
6.2	Finding the initial point in the intermediate problem	113
6.3	K means L_n : a quick variation of the K -methods	123
7.1	Feature selection for unsupervised learning	144
7.2	Feature selection for supervised learning	145
9.1	Finding the shape of a set of points	168
9.2	An algorithm for eliminating noise	169

List of Figures

1.1	Visual categorisation and clustering in 2 dimensions	12
1.2	The number of clusters in this dataset is not obvious	12
1.3	Use of different dissimilarity measures	13
1.4	Linear separation of sets	24
1.5	Nonlinear separation through support vector machine	27
1.6	Polyhedral separation of sets	28
1.7	Piecewise linear separation using clustering	30
1.8	Max-min separability.	31
1.9	Several piecewise linear separators for the same sets	35
1.10	Two different scalings for the same dataset	37
1.11	Effects of various scalings on a dataset	38
2.1	Function having 8 local minima and only one global one	47
2.2	Noisy function	48
3.1	Piecewise chained functions: average efficiency	92
3.2	Piecewise chained functions: average gain in efficiency	93
3.3	Piecewise partially separability: average efficiency	94
3.4	Piecewise partially separability: average gain in efficiency . . .	94
5.1	Classification using max-min separability	104
6.1	An example of a dataset containing an erroneous point	113
6.2	Computational effort: Fisher's Iris dataset	116
6.3	Efficiency improvement: Fisher's iris dataset	118
6.4	Computational effort: Image Segmentation dataset	119
6.5	Efficiency improvement: Image Segmentation dataset	120
6.6	Computational effort: TSPLIB3038 dataset	121
6.7	Efficiency improvement: TSPLIB3038 dataset	122
6.8	Computational effort: TSPLIB1060 dataset	123
6.9	Efficiency improvement: TSPLIB1060 dataset	124
6.10	2 nd class for the diabetes database, with 2 hyperplanes	135

6.11	2^{nd} class of the diabetes database, with 3 hyperplanes	136
7.1	A sample dataset containing two almost distinct groups	143
7.2	Projections of the dataset along its features	144
8.1	Structures of the clusters of pendigits	153
8.2	Structures of the clusters in pendigits	154
8.3	Structures of the clusters for the letters dataset	162
8.4	The structures of the clusters	163
9.1	The structure for $L = 5$ for a shape of a finite set	167

List of Tables

1.1	Datasets in use in this thesis	42
3.1	Results for minimising piecewise chained functions	88
3.2	Results for minimising piecewise chained functions	89
3.3	Results for minimising piecewise chained functions	90
3.4	Results for minimising piecewise partially separable functions .	91
5.1	Results of numerical experiments for classifying large datasets.	106
5.2	Results of numerical experiments for classifying large datasets	107
6.1	Results for clustering Fisher's Iris dataset	116
6.2	Results for clustering Image Segmentation dataset	117
6.3	Results for clustering TSPLIB3038 dataset	117
6.4	Results for clustering TSPLIB1060 dataset	118
6.5	Results for DG, uniform initial point, 3 clusters	126
6.6	Results for DG, uniform initial point, 4 clusters	127
6.7	Results for DG+CAM, uniform initial point, 3 clusters	127
6.8	Results for DG+CAM, uniform initial point, 4 clusters	128
6.9	Cluster function: DG, k -means L_1 initial point	128
6.10	Cluster function: DG+CAM, k -means L_1 initial point	129
6.11	Cluster function: DG, ordered initial point	129
6.12	Cluster function: DG+CAM, ordered initial point	129
6.13	Cluster function: DG, uniform-ordered initial point	130
6.14	Cluster function: DG+CAM, uniform-ordered initial point . .	130
6.15	Cluster function: LGO solver	131
6.16	Australian credit card database with 2 hyperplanes skeletons .	134
6.17	Diabetes database with 3 hyperplanes skeletons	134
6.18	Skeleton function: pendigits	137
6.19	Skeleton function: letters	138
7.1	Feature elimination	146
7.2	classification after the first step of the algorithm.	147

7.3	intermediate classification	148
7.4	order of elimination.	148
8.1	Pendigits: Repartition of the classes in the clusters	151
8.2	Pendigits: Repartition of the classes in the clusters	152
8.3	Pendigits: Repartition of the classes by cluster layers	154
8.4	average depth for each class in each cluster	155
8.5	Letters: Repartition of the classes in the clusters	156
8.6	Letters: Repartition of the classes in the clusters	157
8.8	Pendigits: Repartition of the classes by cluster layers	160
8.7	Letters: Repartition of the classes in the clusters	161
8.9	average depth for each class in each cluster	164
9.1	Point elimination for symmetric definite positive matrices . . .	170
9.2	Point elimination for diagonal matrices	170
9.3	Repartition of the points using S_{10} structures	171

Notations

Operators

$\|\cdot\|$ Norm of a vector

$\langle \cdot, \cdot \rangle$ Scalar product

$\lfloor \cdot \rfloor$ floor (largest smaller integer)

Sets

card · cardinality of a (finite) set

co · convex hull of a set

int · interior of a set

\mathbb{R}^n n -dimensional Euclidean space

\mathbb{R}_+^n $\{x \in \mathbb{R}^n : x_i \geq 0\}$

\mathbb{R} \mathbb{R}^1 (1-dimensional Euclidean space)

\mathbb{N} Set of positive integers.

S_α sphere of radius α :

$$S_\alpha = \{g \in \mathbb{R}^n : \|g\| = \alpha\}$$

G set of vertices of the unit hypercube in \mathbb{R}^n :

$$G = \{e \in \mathbb{R}^n : e = (e_1, \dots, e_n), |e_j| = 1, 1 \leq j \leq n\}$$

P Set of univariate positive infinitesimal functions:

$$P = \left\{ (z : \mathbb{R}_+ \rightarrow \mathbb{R}) : z(\lambda) > 0, \forall \lambda > 0; \frac{z(\lambda)}{\lambda} \xrightarrow{\lambda \rightarrow 0} 0 \right\}$$

Other

$C^n(A)$ set of functions n times differentiable defined on A .

D_f domain of definition of the function f .

e_i i -th orth vector:

$$e_i^j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

diag (u) diagonal matrix, whose diagonal is the vector u

diag (A) vector whose coefficients are the diagonal coefficient of the matrix A .

0 origin ($\vec{0}$), or number zero.

Introduction

Mathematical optimisation was always motivated by practical purposes. It has applications in almost every field of interest. Building accurate mathematical models and problems, selecting and applying adapted optimisation algorithms and efficiently using theory is in itself a challenging task.

In the area of data analysis, information is collected on the field and interpreted for various purposes later on. Here again, almost every task can be rewritten as a mathematical programming problem. Unfortunately accurate models are too complex to be solved, and it is often necessary to simplify the models, or design heuristic methods.

There are several reasons for this trend: data analysis problems are usually very difficult to solve, and involve complex structures that cannot be handled correctly by the most common optimisation techniques. On the other hand, more sophisticated methods are too demanding for current computers to solve problems of reasonable size.

Unfortunately this is a never ending problem: as the computing power will increase, data analysis models will become more accurate, but also harder to solve. Databases will increase in size, thereby increasing the difficulty, and optimisation methods will still not be efficient in solving these problems.

Therefore there is a necessity to study more precisely the structure of the problems encountered in data analysis, and to develop new algorithms more adapted to these specific types of problems.

The task of finding a common structure for data analysis is not a simple one: the field is very broad, englobing several distinct objectives such as:

- Analysing information collected during a particular situation, in order for example to find the cause of a problem or an event, or to interpret the reasons for a trend;
- Using the recorded information to build forecasting models.

However, a good understanding of the types of the problems can lead not only to more adapted solvers, but also to the development of more accurate

models. Formulating these problems in mathematical terms is difficult, as the models need to be both accurate and simple.

The goal of this thesis is two-fold. On the one hand, under the view of existing approaches to data mining, a general common structure to a class of such problems is found. Using the knowledge of this structure, an optimisation algorithm is adapted to solve data analysis problems in general. Although this algorithm does not address all the issues generated by these problems, it does provide a very efficient way to solve them.

On the other hand, better methods for responding to the demands of data analysis are elaborated and tested on real-world datasets.

This thesis is divided in three parts. In the first part, the methodology is considered. Chapter 1 presents the different approaches to data analysis. A specific attention is given to optimisation based models, and a discussion on the advantages and drawbacks of these methods is proposed.

In chapter 2, the different existing optimisation solvers are reviewed. In view of solving data analysis problems, the most appropriate ones are selected, and their limitations are discussed.

The second part is devoted to computational considerations. The structure of the problems is used to adapt the methods selected in chapter 2 to our needs. In chapter 3, a new class of functions is introduced. The class of piecewise partially separable functions is very broad, and is particularly representative of large scaled problems. An algorithm to solve these problems is proposed, and results of numerical experiments on test problems from the literature are presented, to show the efficiency of these methods.

In chapter 4, this class of function is linked with the problems modelling data analysis tasks. In general, these problems have a similar structure which corresponds perfectly to the class of piecewise partially separable functions. In particular, two problems are examined, and the algorithm presented in chapter 3 is adapted to these. These two problems are chosen because they will be used in the models developed in the next part.

The final part focuses more on data analysis itself. Attention is given to the construction of models, and to the way they can be evaluated. The practical tools developed in the previous part are put into action, to efficiently solve the problems. The first task examined is the one of classification, or building forecasting methods. In chapter 5, a classification algorithm is developed. This method presents the advantage of having few theoretical limitations. However, it is difficult to implement practically. The implementation proposed in this thesis is tested on large scale datasets, and results are given to show that this algorithm is efficient.

Chapter 6 gives more attention to the task of clustering, that is splitting a dataset into several smaller ones, in order to find its underlying structure.

The modelling of the clustering problem is a particularly hard one, as its purpose is not clearly delimited. Two approaches are proposed. The first one is designed for obtaining an accurate result, without any prior knowledge on the dataset. The other one is more focused on time efficiency.

In chapter 7, a feature selection method is presented. Preprocessing the dataset before solving the problem is very important, as it is often ill-scaled and contains useless and noisy information. While a large number of preprocessing methods exist, very few of them are based on optimisation. A feature elimination scheme adapted to optimisation based data analysis method is necessary for obtaining an accurate result. Moreover unlike most existing methods, the scheme presented in this thesis is designed to work for clustering.

A fundamental discussion is carried out in chapter 8, on how to evaluate the quality of clustering. In particular, attention is given to a common belief that a good clustering method should separate a labelled dataset according to the labels, thus acting as a classification method. It is shown in this chapter that this is not necessarily true, and that the underlying structure of the dataset may be based on other criteria.

A fairly new direction is examined in chapter 9, where the structure of a set of points is studied more carefully. Once a dataset has been divided into smaller clusters, it may be interesting to have some algorithm finding the intrinsic properties of these clusters. In this chapter, an algorithm to find an ellipsoidal envelope of the set is presented. An application of this algorithm for eliminating noisy points from the dataset is developed and applied.

Part I

Methodology

Chapter 1

Data analysis

1.1 Introduction

The second half of the last century witnessed an unprecedented development of the technology. Increasingly sophisticated electronic measurement devices, combined with larger and cheaper storage capacity, permitted the collection of very large data at a high rate. Major scientific projects, like the human genome, will rely heavily on the data collected by experts for many years. Today, hardly any political or financial decision is made without the help of some sort of data, and wars are declared on the basis of “intelligence data”. However, mistakes are made. For this reason, the art of extracting non-trivial information from data is bound to take a major importance in the near future. This is the purpose of data analysis.

1.1.1 Definitions and notations

The information usually takes a very distinct form: for a given individual, a set of *characteristics* or *features* are measured or evaluated. These measurements are carried out on a number of instances.

For each individual, the set of values of all the characteristics is called a *record*, or an *observation* and the set of all the records is called a *dataset* or *database*.

Some obstacles to the measurements are:

- the cost of a particular measurement technique may restrict the number of measures taken;
- the precision of the measurements can not always be ensured, and measurement errors are likely to appear

In [181, page 4, example 4], the author gives an excellent example which helps explain the different concepts in data analysis. The following example is largely inspired from this.

Example 1: Suppose one decides to collect information about a particular sickness. On a number of patients doctors can input material such as age, gender, height, weight, social and professional categories, diet, colour of the eyes, ... and whether the person is sick.

In this case, the material collected for one patient constitutes a *record*, while “age”, “gender” or “diet” are *features*

The collected information must be coded in order to be stored in computers. Such coded information is called *data*. This task may be a delicate one, as several difficulties can appear:

- Most measurements can be stored under different units (time can be stored in centuries, years, minutes, seconds, microseconds, ...)
- Some more abstract information has to be made suitable for a computer format (a colour may take many different variations that may be difficult to code).

Example 2: In the case of the dataset presented in example 1, the height can be recorded in centimetres, metres, feet, etc... Meanwhile, the weight can be recorded in kilogrammes, pounds, or stones. Although these records will in reality represent the same entities, a height in centimetres will “vary” more from one record to another than a weight in stones, while a height in metres will “vary” less than a weight in kilogrammes

Some of the characteristics may also not be very easy to record very accurately: how to describe numerically the profession or the diet of a person?

At this stage, a dataset can be represented as a table, where the columns are the features and the rows are the records.

The purpose of storing information is to enable researchers to find some underlying characteristics. The process of extracting non-trivial information from data is called *data analysis* or *data mining*. The fields of data analysis can be divided into several categories, among which:

- Preparing the dataset for the application of some algorithms.
- Comparing the existing data, in order to find the similarities and differences.
- Building forecasting models from the existing data, in order to assign a new record to one of several categories.

We focus on three particular aspects of data analysis: classification, clustering, and preprocessing. In the next section we will give a brief glance at each of these problems. Then, for each of them, the existing research directions and solutions will be discussed.

1.1.2 Classification (supervised learning)

The supervised classification problem is one of the most widely studied in the field of data analysis. It may be necessary to be able to guess a characteristic using the features. A dataset is separated into several groups (*classes*) according to one of its features. The goal of supervised classification is to elaborate an algorithm to assign a new observation to one or several of the classes.

One can distinguish between single-class classification, where each record may only belong to one class, and multi-class classification, where a record may belong to several classes.

Example 3: Let us come back to example 1 on the previous page. The current test to check whether the patient is sick could be very expensive. In that case, it is interesting to devise a method to assess whether the patient is sick or not according to the other features.

In this case, the feature “Sick or not” represents the class. Obviously here the dataset consists of only two classes, and each record can only belong to one class.

1.1.3 Clustering (unsupervised learning)

The clustering problem is a less definite one than the supervised classification problem. It consists of finding “clusters”, that is to group the records by similarity.

The following definition can be found in [104]:

Cluster Analysis is the organisation of a collection of patterns (usually represented as a vector of measurements, or a point in a multidimensional space) into clusters based on similarity. Intuitively, patterns within a valid cluster are more similar to each other than they are to a pattern belonging to a different cluster.

Example 4: Let us consider the dataset from the example 1 on the preceding page, examining only the patients who are sick.

Then the clusters are likely to reflect the different causes of the disease. For example, one might find that people who are sick can be divided into the following clusters:

- Elderly people,
- Smokers/Drinkers,
- Miners,
- Children growing up in the city.

This would help elaborate an adapted therapy for each of these types of patients.

Notice however that the problem of clustering is very protean and several solutions may appear suitable, as they would separate the datasets according to different but acceptable criteria.

Example 5: Instead of grouping the patients according to the cause of the disease like in example 4 on the previous page, the clustering algorithm may also group them according to the effects it takes:

- Loss of weight, high fever
- Gain of weight, extreme fatigue

Both criteria are acceptable and both are useful for developing adapted remedies or for prevention.

1.1.4 Preprocessing

As highlighted on page 6, a number of factors might influence the algorithms of data analysis. Prior to applying a clustering or a classification algorithm, it is thus necessary to perform preprocessing. When the characteristic can be numerically encoded (as are distances, weights, age, etc...), different units in which the data is entered might induce different behaviours of the classification or clustering algorithms. Indeed, most of the data analysis techniques involve some sort of comparison between the different characteristics, for each particular record, and the results of these comparisons depend strongly on the scales for each characteristic.

Example 6: Suppose one wants to compare the influences of the sickness on the weight of a person and on her body temperature. The temperature is measured in Celsius degrees, and its value may vary by 2 or 3. If the weight is measured in stones its value may change by 3.5. In this case the variations are quite comparable. However, if the weight is measured in kilogrammes, the change could be of 20, which would give more importance to the weight changes than to the temperature ones.

One way to limit the fluctuations of the results is by *rescaling* the dataset: each feature is reduced to a particular scale in order to enable their comparison.

Another problem encountered in data analysis is the *feature selection*. The choice of the data collected is usually made by humans. Aside from financial considerations, it may seem that the most information is recorded, the best it is. This will result in the record of much useless or redundant information.

Example 7: Before the modern days of data mining, people used to believe that the colour of the hair could be associated with witchery. Good feature selections would have helped them correct this misjudgement.

Being able to sort the data through *feature selection* in order to eliminate uninformative features is capital, as these features not only slow down the programs, but also may bring some noise and alter the results.

In the rest of this chapter we will give a review of the existing methods used in data analysis.

1.2 Clustering (unsupervised learning)

The literature about clustering is abundant and dense. The clustering problem appears in a large variety of fields, and was therefore often studied separately, under various forms and with different vocabulary. It is not our goal here to carry out the monumental task of giving an exhaustive overview of the clustering techniques. Excellent starting points for such a review are the report [104] and the book [147]. We also can refer to an older but good introduction such as [182].

We will give here a brief overview of clustering by giving the outlines of the main clustering approaches, and explain more in details the methods we believe are of interest.

1.2.1 Mathematical formulation

The clustering problem, notwithstanding its multifariousness, can be formulated mathematically.

A dataset can be represented as a finite set of points in the n -dimensional space \mathbb{R}^n . The clustering problem can then be reduced to the following partitioning problem.

Given a finite set $A = \{a_1, \dots, a_n\} \subset \mathbb{R}^n$, find A_1, \dots, A_q such that:

$$\bigcup_{k=1}^q A_k = A.$$

Several other rules are usually added, depending on the problem:

- If the number q of clusters is fixed in advance, in some cases no empty cluster is required

$$A_k \neq \emptyset, \forall k = 1, \dots, q.$$

- In the case of *hard clustering* (as opposed to *fuzzy clustering*), there should be no overlapping between the clusters:

$$\bigcap_{k=1}^q A_k = \emptyset.$$

Our work will concentrate on the hard clustering problem.

The purpose of clustering being to group points by similarity. It is necessary to elaborate a tool to evaluate the similarity between two points. Let us consider a mathematical definition of the *similarity* (see [182]).

Similarity is used to compare two records, and therefore should be a function of two vectors into the ordered space \mathbb{R} : $s : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$. Consider the following list of axioms.

- The similarity is the same for any identical pair (x is as similar to x as y is to y):

$$s(x, x) = s_0 \in \mathbb{R} \cup \{+\infty\}, \forall x \in \mathbb{R}^n$$

- A pair of elements cannot be more similar than an identical pair:

$$s(x, y) \leq s_0, \forall (x, y), \forall (x, y) \in \mathbb{R}^n \times \mathbb{R}^n$$

- If two elements are as much similar as they can, they are identical:

$$s(x, y) = s_0 \Rightarrow x = y, \forall (x, y) \in \mathbb{R}^n \times \mathbb{R}^n$$

- **Symmetry:** x is as similar to y as y is to x :

$$s(x, y) = s(y, x), \forall (x, y)$$

Definition 1. We will call *similarity* a function $s : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$ which satisfies all these axioms.

It is possible to write a dissimilarity function:

- If $s_0 \in \mathbb{R}$, $d(a, b) = s_0 - s(a, b)$,
- If $s_0 = +\infty$, $d(a, b) = \frac{1}{s(a, b)}$.

If the dissimilarity function follows the rule:

- $d(x, z) \leq d(x, y) + d(y, z)$

then it is a metric function.

In this context metrics can be used as a dissimilarity measure. Metric functions are a widely studied class of functions, and people are usually more comfortable with measuring the dissimilarity than the similarity (that is point out the differences between two items, instead of their resemblance). Therefore most clustering algorithms are based on the minimisation of the dissimilarity. This is the approach adopted in this thesis.

Remark 1: It should be emphasised that the distance used for the minimisation should not necessarily be a metric distance. In fact, the distance is dependent on the dataset and the clustering goals.

For example, it is quite common in telecommunications to consider the pathloss between an antenna and a user. This pathloss is expressed under the form: $d(a, b) = \alpha \ln(\|b - a\|_2 + \beta)$, for some $\alpha > 0, \beta > 0$. As this is a concave function, it is clearly not a metric.

Inasmuch as an inappropriate distance would lead to meaningless results, for each application, or each dataset, a proper distance function has to be specified by an expert.

Remark 2: Most clustering algorithms actually try to emulate the “visual categorisation” done by the brain in 2-dimensional clustering: suppose one can represent the records on a euclidean plan. Then the brain will naturally try to group the points according to patterns based on geometrical considerations (such as the Euclidean distance).

The figure 1.1(a) shows a dataset, and figure 1.1(b) on the next page shows its natural clustering. In figure 1.1, one can see how the points “close to one another” are grouped together. Notice however that point A is closer to point C from another cluster than to point B from its own cluster.

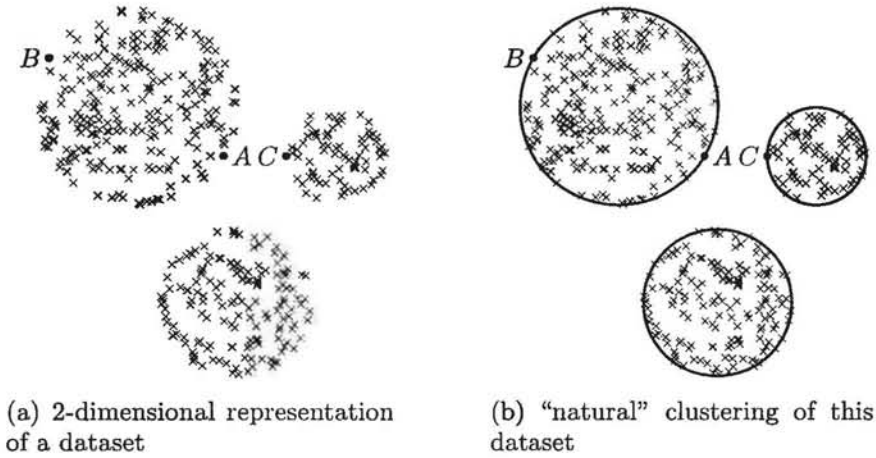


Figure 1.1: Visual categorisation and clustering in 2 dimensions

Remark 3: The difficulty of the clustering problem can easily be grasped when a 2-dimensional example is considered: the number of clusters contained in the dataset represented on figure 1.2 is not obvious, and probably depends on the type of information sought.

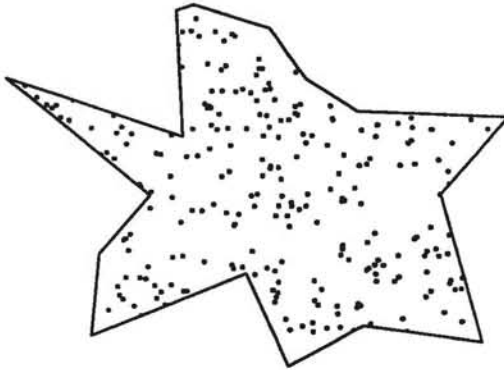


Figure 1.2: The number of clusters in this dataset is not obvious

As distance functions, most methods use the Minkowski metric:

$$d(x, y) = \|x - y\|_p^\gamma,$$

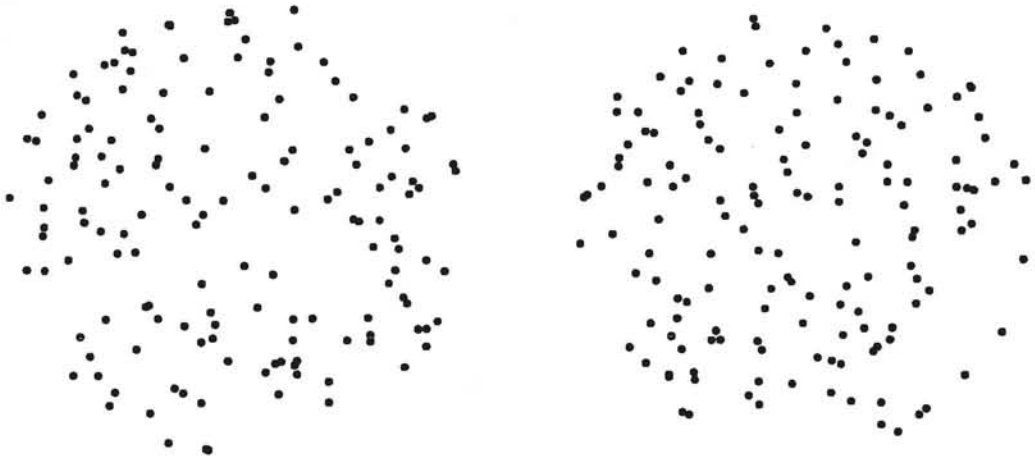
where

$$\|x\|_p = \left(\sum_{i=1}^n x_i^p \right)^{\frac{1}{p}}. \quad (1.1)$$

Remark 4: For $p = 2$ and $\gamma = 1$ this function is the Euclidean distance.

The advantage of such distance functions is multiple:

1. Minkowski metrics, particularly for $p \in \{1, 2, \infty\}$ are based on geometrical considerations and therefore comply with remark 2 on page 11;
2. Minkowski metrics are well known and are often less computationally demanding than less usual distance functions.



(a) Clusters are defined by the Euclidean metric



(b) Clusters are defined by a distance to a central plane

Figure 1.3: Datasets defined by clusters defined by different dissimilarity measures

Figure 1.3 presents two datasets. The one presented on figure 1.3(a) clearly consists of two clusters, which can take the form of balls. Clearly by defining two central points for these clusters, it is possible to characterise this dataset fairly well.

Contrariwise, it is difficult to say how many point-based clusters the set depicted on figure 1.3(b) has. Its description by means of such clusters cannot

clarify its structure. At the same time this structure can be described by three straight lines (hyperplanes).

1.2.2 Optimisation problem

Consider the problem of finding q clusters in the dataset A , containing m observations and n features. The clustering problem can be presented as an optimisation problem, where the objective is to maximise the similarity of the points inside each cluster. A common method is to assimilate each cluster with its centre. We obtain the following formulation: ([32, 33, 182]):

$$\begin{aligned} \text{minimise } \varphi(C, x) &= \frac{1}{\text{card } A} \sum_{i=1}^q \sum_{a \in A^i} d(x^i, a) \\ &\text{subject to} \\ C &\in \overline{C}, x_i = (x^1, \dots, x^q) \in \mathbb{R}^{n \times q}. \end{aligned} \quad (1.2)$$

where $C = \{A^1, \dots, A^q\}$ is a set of clusters, \overline{C} is the set of all possible q -partitions of the set A and x^i is the centre of the cluster A^i , $1 \leq i \leq q$.

This formulation presents an important drawback: the variables of the problems are clusters, or sets of points. It is therefore extremely difficult to devise algorithms to solve such problems. However, it can be rewritten as the following mathematical programming problem:

$$\begin{aligned} \text{minimise } \psi(x, w) &= \frac{1}{\text{card } A} \sum_{i=1}^m \sum_{j=1}^q w_{ij} d(x^j, a^i) \\ &\text{subject to} \\ x &= (x^1, \dots, x^q) \in \mathbb{R}^{n \times q}, \\ \sum_{j=1}^q w_{ij} &= 1, 1 \leq i \leq m, \\ w_{ij} &= 0 \text{ or } 1, 1 \leq i \leq m, 1 \leq j \leq q \end{aligned} \quad (1.3)$$

where w_{ij} is the association weight of pattern a^i with cluster j , given by

$$w_{ij} = \begin{cases} 1 & \text{if pattern } i \text{ is allocated to cluster } j, \forall 1 \leq i \leq m, 1 \leq j \leq q, \\ 0 & \text{otherwise} \end{cases}$$

This problem is a conventional optimisation problem. However it is very heavy, as it is a mixed problem (it has both integer and real variables), and contains a very large number of variables (this number depends on the size of the set A , which can reach several millions). It may nevertheless be reduced to the following nonconstrained nonsmooth problem:

$$\begin{aligned} & \text{minimise } \frac{1}{\text{card } A} \sum_{i=1}^m \min_{1 \leq s \leq q} d(x, a) \\ & \text{subject to} \\ & (x^1, \dots, x^q) \in \mathbb{R}^{n \times q}, \end{aligned} \tag{1.4}$$

This formulation has several advantages: the number of variables is independent from the number of instances in the dataset. Furthermore, it is a nonconstrained problem. Nevertheless, the objective function is nonconvex (for $q > 1$) and nonsmooth, containing a large number of local minima. Most usual optimisation algorithms do not efficiently solve this problem, for several reasons:

- Due to the nonsmoothness of the function, the application of smooth descent algorithms such as Newton's method and its variations are not adapted
- The potentially large number of variables (real world datasets may contain hundreds of features) does not allow the application of global techniques.

Remark 5: It must be noted that the formulation (1.4) gives some freedom: not only the function d does not have to be a metric, but also the variable x does not have to be in \mathbb{R}^n . In [37] an objective function was introduced where the "centres" are not points, but hyperplanes, and thus the variables belong to \mathbb{R}^{n+1} .

1.2.3 k -Methods

General scheme

The k -means method is one of the oldest, yet still one of the most popular clustering methods. It was introduced by MacQueen in 1965 ([138]), and since then many variations have been developed. A generalisation of this method is presented here, and its efficiency is discussed.

The k -methods are grounded on the convexity of metric distance functions, and can thus be applied to any convex dissimilarity measure.

Algorithm 1.1 on the next page presents a general version of the k -methods. It is easy to see that the k -algorithms terminate: there are a finite number of ways to assign the points into q clusters. As the stopping criterion ensures that no repeat is possible, the algorithm terminates after a finite number of steps. The convergence of the algorithm is studied in [177].

Algorithm 1.1: The k -methods**Step 1 Initialisation:**

Select an initial solution x_1^0, \dots, x_q^0 , set $i \leftarrow 0$

repeat

Step 2 $i \leftarrow i + 1$

Step 3 Cluster assignment:

Find clusters A_1^i, \dots, A_q^i such that for any $1 \leq l \leq q$,

$$A_l^i = \{a \in A : d(x_l^{i-1}, a) \leq d(x_p^{i-1}, a), \forall 1 \leq p \leq q\}.$$

Step 4 Cluster update:

For each cluster i solve the convex optimisation problem:

$$\begin{aligned} & \text{minimise } \sum_{a \in A_l^i} d(x, a) \\ & \text{subject to} \\ & x \in \mathbb{R}^n, \end{aligned} \tag{1.5}$$

until $\exists j \in \{1, \dots, i-1\} : \{A_1^i, \dots, A_q^i\} = \{A_1^j, \dots, A_q^j\}$

The particularity of this algorithm is that it is based on the solution of the problem (1.5), which is a convex problem of relatively small size. Moreover this problem is usually smooth (when the distance function is smooth), and thus there are many methods to find the global minimum.

The problem with this algorithm is that it strongly relies on an efficient solver for the convex minimisation problem. Indeed, in most cases, when a theoretical solution to this problem is not known and a local algorithm has to be applied, the k -algorithm is practically not viable. Some interesting cases worth mentioning are:

- The k -means algorithm (see [138]), solving the problem for $d(x, y) = \|y - x\|_2^2$, where $\|\cdot\|_2$ is the Euclidean norm: $\|x\|_2 = (\sum_{i=1}^n x_i^2)^{\frac{1}{2}}$. For such distance the solution of the problem (1.5) is the barycentre of the set A_l^i .
- The k -median algorithm (see [38]), solving the problem for $d(x, y) = \|y - x\|_1$, where $\|\cdot\|_1$ is the 1-norm: $\|x\|_1 = \sum_{i=1}^n |x_i|$. For such distance the solution of the problem (1.5) is the median of the set A_l^i .
- The k -planes algorithm, introduced in [37]. For this method, the variables are hyperplanes ($x \in \mathbb{R}^{n+1}$), and the distance used is $d(x, a) = (\langle l, a \rangle - b)^2$, where $x = (l, b)$, $l \in \mathbb{R}^n$, $b \in \mathbb{R}$. For this method, the analytical solution can be obtained as the solution of a linear system.

Although the k -methods seem quite general and lead to a local solution for any kind of distance, in practice only the k -means and the k -median are fast enough for being applied. Even the k -planes algorithm, for which an analytical solution to the convex problem is calculated, cannot be applied successfully on real-world datasets, as solving a linear system at each iteration appears too expensive.

Notwithstanding the popularity of the k -means method, it has been shown ([25]) that these methods also rely strongly on the initial point, which translates in them often terminating on shallow local minima.

Improvements

To reduce the influence of the initial points on the k -means algorithm, many improvements have been proposed (see [35, 66, 90, 182]). They can arguably be divided into two categories:

1. The search for a well-chosen initial point,
2. The combination with other search methods.

The search for a combination has been particularly proficient. Most algorithms are based on the application of k -means starting from a number of initial points, chosen randomly according to some probabilistic distribution.

On the other hand, the search for an analytical initial point is a hard task, as little is known about the structure of the dataset. Once again most approaches are based on some statistical considerations. Two approaches worth mentioning have been proposed by Hansen and Mladenovic in [87] and [89] (see also [86, 88]).

- The j -means method is a variant of the k -means algorithm where the cluster assignment step 3 is modified as follows: one of the cluster centres is replaced by one of the observations. The selection of the centre and the observations is based on the best improvement of the value of the function (1.4).
- The other one is based on a combination between k -means and a Variable Neighbourhood Search. Variable Neighbourhood Search ([88]) is a metaheuristic algorithm adding perturbations to a local search algorithm.

An approach proposed in [124] uses an incremental implementation of the k -means algorithm: at iteration i , the method solves the k -means N times (each observation being an initial point) for finding the optimal solution for i clusters.

In [89], the authors compare these three methods. They all clearly improve the k -means algorithm. The combination between k -means and VNS is best among them. Nevertheless, while the k -means method is an extremely fast procedure, these methods are computationally more expensive.

1.2.4 Optimisation methods

When the number of clusters is larger than one, the objective function (1.4) is nonsmooth. Most classical methods can therefore not be used to solve this problem. Moreover the number of variables depends on the number of features and on the number of clusters. It can be very large, and consequently make problem (1.4) not solvable through global methods.

Several methods have however been applied on this problem. A good review for these algorithms is given in [86].

Among these methods we can find:

- Dynamic Programming and Branch and Bound methods, which only perform well for a reduced number of features and few clusters (see [105]), but get worse for larger problems ([64, 84, 118]).
- Metaheuristics, such as Tabu Search ([4]), Simulated Annealing ([40, 178, 183]) and Genetic Algorithms ([169]). These methods were successfully applied to solve Problem (1.4). Nevertheless, even for small problems, these methods may be more than thousands times slower than the k -means methods (see [5]).
- A Variable Neighbourhood approach was developed in [88], and an interior point search in [65].

In a general manner, numerical experiments have shown that while some heuristic methods may reach good solutions, they demand too much computational effort to be effective. The solution of the problem is out of reach for global deterministic methods, and general purpose local descent methods are generally slower than the k -means method without guaranteeing a better solution.

Remark 6: It should be noted that the k -methods can be considered as local optimisation methods, as it has been shown that they converge towards a local solution for a particular function. For instance, the k -means method converges towards a local minimum of the function:

$$\begin{aligned} & \text{minimise } \sum_{a \in A} \min_i \|x - a\|_2^2 \\ & \text{subject to} \\ & x \in \mathbb{R}^n \end{aligned}$$

1.2.5 Other methods

Many non-optimisation based clustering methods exist. Among these can be mentioned:

- **Hierarchical methods.** These methods are based on iteratively division or merge of clusters. Their drawback is that the results strongly depends on eventual errors during the previous iterations.
- **Self organising maps.** Self Organising Maps are one of the most popular clustering methods. They are based on the distribution of the data into 2-dimensional cells represented by a vector. Very little theoretical information about these methods is known, and convergence only exists for 1-dimensional data. For more information, see [56, 67, 68, 69, 73, 96, 115, 116, 117, 155].

1.2.6 Evaluation of the clustering

The main criterion to evaluate the quality of the clusters is a function measuring the similarity. Assuming that the dissimilarity measure is adapted to the problem and the dataset under consideration, the average dissimilarity within the clusters, as described in (1.4) is a good indicator.

However the selection of an adapted dissimilarity measure is a perilous task, and because no automatic selection procedure exists, it can often lead to inaccurate results. Other techniques for evaluating a clustering method exist. We present two of them in this section.

Structure of clusters

One possible way to evaluate the quality of unsupervised classification is to check the distribution of the points within the clusters. To this effect, the notion of structure of clusters has been introduced in [24]. The goal is to check how "deep" the points are in the clusters.

Suppose that we work with a dataset which contains N observations. A clustering method has been applied to this dataset and k centres of clusters have been obtained. Consider a point a from the dataset which belongs to the l -th cluster (with the centre x^l). For this point we determine a value $c(a)$ which can be found as follows:

$$c(a) = \frac{d(a, x^l)}{\min_{\substack{j=1, \dots, k \\ j \neq l}} d(a, x^j)}. \quad (1.6)$$

From the definition of clusters it is easy to conclude that $c(a) \in [0, 1]$. Very often a grid for $c(a)$ such as $0.1, 0.2, \dots, 1$ is used, and intervals are considered rather than the exact value for each $c(a)$. For example,

$$c(a) \in [(i-1) * 10^{-1}, i * 10^{-1}), \quad i = 1, \dots, 9, \quad c(a) \in [0.9, 1] \quad (1.7)$$

or

$$c(a) \in [0, i * 10^{-1}) \quad i = 1, \dots, 9, \quad c(a) \in [0, 1]. \quad (1.8)$$

The value $c(a)$ for each point describes how “deep” this point is inside the cluster. It should be underlined that different values of $c(a)$ do not represent the radii for some spheres centred at the centres of the corresponding clusters. They rather represent some levels of confidence that the chosen point belongs to this cluster but not another one. It is possible that some points which are not “deep” enough inside the corresponding cluster move to another cluster (change their membership). It could happen, for example,

- if we change the dissimilarity measure;
- if we change the location of points (another accuracy to represent numbers in the computer);
- if we change the value for some internal parameters for the optimisation methods.

If $c(a) = 1$ or close to 1 there are two centres such that the distances between the point and these two centres are (almost) the same. In this case some changes within the data may change the membership of the point (the point is “unstable” inside the cluster). If $c(a)$ is close to 0 the level of confidence for the point to keep its membership is high (the point is “stable”).

Suppose that we obtain two different clustering results. First we can compare the value of cluster function in the centres of corresponding clusters. The systems of clusters with the lowest value of cluster function is better from the point of view of cluster function. Second, we check the structure of corresponding clusters by means of (1.6). If for the first clustering result, for most of the points a the values $c(a)$ are smaller than for the second clustering result, we assume that the first collection of the centres is preferable to the second one in the sense of the structure of the clusters (by means of (1.6)). The most important is to investigate the points a with the values $c(a) \in [0.9, 1]$. It is possible that this two approaches do not coincide.

Classes and clusters: purity

The notion of *purity* (see for example [63]) is used in the literature for evaluation of accuracy of clustering methods. It requires classes in a dataset to be known in advance, but not used during the learning process. Assume that we have a dataset A composed of the classes $\{D_1, \dots, D_l\}$ and we apply a clustering procedure for finding clusters $\{C_1, \dots, C_k\}$ in this dataset.

The purity of a set of clusters $\{C_1, \dots, C_k\}$ is calculated as follows:

$$p(\{C_1, \dots, C_k\}) = \frac{1}{\text{card } A} \sum_{i=1}^k \max_{j=1, \dots, l} |C_i \cap D_j|, \quad (1.9)$$

Let us illustrate this notion in the simplest case where a dataset A contains only 2 classes $\{D_1, D_2\}$. Suppose that 4 clusters $\{C_1, C_2, C_3, C_4\}$ have been found in this dataset.

If the majority of points from the j -th cluster ($j = 1, \dots, 4$) is in the i -th class ($i = 1, 2$), we assign the whole j -th cluster to the i -th class. When all the clusters are assigned to one of the classes, the percentage of the correctly classified points for the test set is considered as the classification accuracy.

$$p(\{C_1, \dots, C_4\}) = \frac{1}{\text{card } A} \sum_{i=1}^4 \max(|C_i \cap D_1|, |C_i \cap D_2|)$$

The validity of the notion of purity relies on the expectation that for a labelled dataset, a good clustering method should divide the dataset into clusters corresponding to its classes. Stated differently, the classes in a labelled dataset are the only valid clusters. This is very arguable, as other clusters may arise in a dataset, and in case the dataset is not well constructed, these clusters may even make more sense than the actual classes.

1.3 Classification (supervised learning)

Unlike the clustering problem, the classification problem can be clearly stated. Given some *labelled* observations, one tries to devise a set of rules to assign a label to a new observation. In other terms, the classification algorithm is given a *training set*, where the labels are known, and the output should be a decision rule on a new observation, for which the label is not known in advance.

The labels are also called *classes*.

Here again, the literature is abundant, and an exhaustive list of classification algorithms is beyond the scope of this work. An excellent introduction to classification techniques as well as a good list of references can be found in [145]. Below we give a brief presentation of the main directions and a review on the application of optimisation to classification.

1.3.1 Evaluation of the algorithm

It is fairly easy to evaluate a classification algorithm. Indeed, it suffices to check whether it labels observations well or not.

In order to allow a comparison of algorithms, a number of standard datasets are used in the data mining community. The datasets can be found on repositories such as [150, 170].

The algorithm is applied on a *training set*. Obviously, the evaluation cannot be performed on the training set itself. Because the labels of the points of this set are known and used by the algorithm, it is simple for this algorithm to get the rules to classify correctly these observations.

To evaluate the quality of a classification algorithm it is thus necessary to apply the obtained classification rules to a *test set*, for which the labels are known, but not used during the learning process, but only as a verification during the classification process.

In practice, two major techniques are used

- The training and test sets are given separately. The comparison of the different algorithms is thus made on the same samples.
- The test set is not given. In that case, the *n-fold cross validation* is applied

The *n-fold cross validation* is a simple process: the dataset is divided randomly into n groups. The learning process is then applied n times, each time a different subgroups being the test set and the rest being the training set. The accuracy of the algorithm is then the average accuracy over the n experiments.

1.3.2 Non-optimisation based approaches

Here again, many approaches exist for data classification. Such methods include:

- **Statistical approaches**, which tend to be very sensitive to the size of the data: when the data is too small, their performance may be altered;

- **Machine learning methods.** These methods are based on the elaboration of the classification rules as a tree. It has been shown in [145] that these methods may perform very well on the training data, but very bad on test data. They may also be unpractical, as they are very memory-consuming;
- **Neural networks.** The neural network approach is based on an analogy with the brain. The training data is processed one by one, and the classification rules are corrected or refined progressively. The drawback of these methods is that the classification rules are not explicit, and therefore it is difficult to evaluate whether they will fail.

1.3.3 Optimisation methods in supervised learning

Intuitively, the classification problem is an optimisation problem: the goal is to minimise the number of misclassified points (or to maximise the number of well-classified points).

Mathematically the formulation of the problem takes usually the following shape: the classification rule is formulated as a function depending on a number of parameters. Then the objective function of the problem depends on the misclassified points, and the variables of the problem are the parameters.

In the following subsections, we will introduce various types of optimisation problems for supervised learning. Most of these problems are based on the connected problem of separating two sets \mathcal{A} and \mathcal{B} . Mathematically, this problem is formulated as follows.

Find $f : \mathbb{R}^n \rightarrow \mathbb{R}$ such that:

$$\begin{cases} f(a) < 0 & \forall a \in \mathcal{A} \\ f(b) > 0 & \forall b \in \mathcal{B}, \end{cases}$$

or, without loss of generality, such that:

$$\begin{cases} f(a) \leq -1 & \forall a \in \mathcal{A} \\ f(b) \geq 1 & \forall b \in \mathcal{B}, \end{cases} \quad (1.10)$$

The difficulty of solving this problem is that usually only subsets $A \in \mathcal{A}$ and $B \in \mathcal{B}$ are known. This problem in general is a *feasibility* problem: find a solution satisfying a given set of constraints. Feasibility problems can often be converted to optimisation problem by the creation of a *penalty* function. The function is positive for non feasible solutions, and zero otherwise. Often these functions represent a “distance” to the feasible set, to ease the task of the optimisation method. These functions are called *error functions*.

In the case of the problem (1.10), the variable is a function. This problem is hardly solvable as such, and most optimisation-based classification methods actually restrict the function f to a special class.

1.3.4 Linear separability

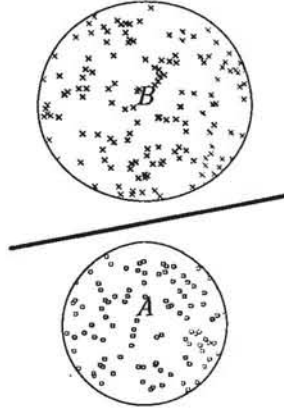


Figure 1.4: Linear separation of sets

Linear Separability is a set-separation technique, where the separating function is affine.

Let A and B be given sets containing m and p n -dimensional vectors, respectively:

$$\begin{aligned} A &= \{a^1, \dots, a^m\}, a^i \in \mathbb{R}^n, 1 \leq i \leq m, \\ B &= \{b^1, \dots, b^p\}, b^j \in \mathbb{R}^n, 1 \leq j \leq p. \end{aligned}$$

The sets A and B are linearly separable if there exists a hyperplane $\{x, y\}$, with $x \in \mathbb{R}^n, y \in \mathbb{R}^1$ such that

1. for any $j \in \{1, \dots, m\}$

$$\langle x, a^j \rangle - y < 0,$$
2. for any $k \in \{1, \dots, p\}$

$$\langle x, b^k \rangle - y > 0.$$

The sets A and B are linearly separable if and only if $\text{co } A \cap \text{co } B = \emptyset$.

In [30] the problem of finding this hyperplane is formulated as the following optimisation problem:

$$\text{minimise } f(x, y), \text{ subject to } (x, y) \in \mathbb{R}^{n+1} \quad (1.11)$$

where

$$f(x, y) = \frac{1}{m} \sum_{i=1}^m \max(0, \langle x, a^i \rangle - y + 1) + \frac{1}{p} \sum_{j=1}^p \max(0, -\langle x, b^j \rangle + y + 1)$$

is the error function.

The authors describe an algorithm for solving problem (1.11). They show that the problem (1.11) is equivalent to the following linear program:

$$\begin{aligned} & \text{minimise } \frac{1}{m} \sum_{i=1}^m t_i + \frac{1}{p} \sum_{j=1}^p z_j \\ & \text{subject to} \\ & t_i \geq \langle x, a^i \rangle - y + 1, 1 \leq i \leq m, \\ & z_j \geq -\langle x, b^j \rangle + y + 1, 1 \leq j \leq p, \\ & t \geq 0, z \geq 0, \end{aligned}$$

where t_i represents the error for the point $a^i \in A$ and z_j represents the error for the point $b^j \in B$.

The sets A and B are linearly separable if and only if $f^* = f(x^*, y_*) = 0$ where (x^*, y_*) is the solution to the problem (1.11). It is proved that the trivial solution $x = 0, y \in \mathbb{R}$ cannot occur.

1.3.5 Support vector machine

Support Vector Machine (SVM) is essentially an extended linear separability technique. First proposed in [191] (see also [192]), it is nowadays one of the most popular classification methods. A good introduction can be found in [42] (see also [193, 194] for comprehensive reviews).

Linear support vector machines

When two sets A and B are linearly separable, it is quite clear that the minimum of problem (1.11) is 0, and that the solution is not unique. In order to select the "best" among these solutions, a new problem is constructed, where the linear separation is set as a constraint.

The linear separability can be written without loss of generality as follows:

$$z_k(\langle x, p_k \rangle - y) - 1 \geq 0, \forall k \quad (1.12)$$

where $p_k \in A \cup B$ is an observation, and

$$z_k = \begin{cases} -1 & \text{if } p_k \in A \\ 1 & \text{if } p_k \in B \end{cases}$$

The goal of SVM is to maximise the *margin* between the separating plane and the points. This margin is represented by $1/\|x\|$, and therefore the goal of the problem is to minimise $\frac{1}{2}\|x\|^2$, subject to (1.12).

Writing the Lagrangian of this problem, using λ_k as Lagrange multipliers, we obtain:

$$L = \frac{1}{2}\|x\|^2 - \sum_{k=1}^K \lambda_k z_k (\langle x, p_k \rangle - y) + \sum_{k=1}^K \alpha_k. \quad (1.13)$$

By differentiating with respect to x and y , we obtain:

$$x = \sum_{k=1}^K \lambda_k z_k p_k \quad (1.14)$$

$$\sum_{k=1}^K \lambda_k z_k = 0, \quad (1.15)$$

which can be substituted in (1.13) to get:

$$L_D = \sum_{k=1}^K \lambda_k - \frac{1}{2} \sum_{k,l} \lambda_k \lambda_l y_k y_l \langle p_k, p_l \rangle. \quad (1.16)$$

The hyperplane is then found by minimising (1.16) subject to (1.15) and the positivity of the λ_k s.

The vectors in A and B for which the Lagrange multipliers are nonzero are called the *support vectors*.

In the non linearly separable case, no feasible solution exists to the initial problem. A variation is therefore written, taking into account errors $\xi_k > 0$ as:

$$z_k (\langle x, p_k \rangle - y) - 1 \geq -\xi_k, \forall k, \quad (1.17)$$

If $\xi_k \geq 1$ then the point p_k is misclassified. In order to reduce the number of misclassified points, the new objective is

$$\frac{\|x\|^2}{2} + K \sum_{k=1}^K \xi_k.$$

The linear support vector machine is a more elegant way of linearly separating two sets than the method presented in section 1.3.4, because it actually selects the "best" (according to some criterion) feasible separating hyperplane. One should expect that by taking a hyperplane as far as possible from any point, it improves the chances of separating the sets \mathcal{A} and \mathcal{B} as well.

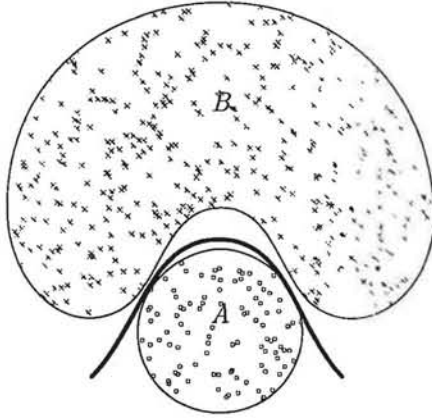


Figure 1.5: Nonlinear separation through support vector machine

Nonlinear support vector machines

Sometimes, the separating function is highly nonlinear. In such a case, the space D can be mapped by a function $\varphi : D \rightarrow H$, where (in general) $\dim(H) \geq \dim(D)$.

Due to the shape of the problem to solve (see function (1.16)), it is not necessary to know φ , but only the function $K : D \rightarrow \mathbb{R}$ such that:

$$K(x, y) = \langle \varphi(x), \varphi(y) \rangle.$$

The function K is called a *Kernel function*.

Classical types of kernel functions are:

- $K(x, y) = (\langle x, y \rangle + 1)^p$
- $K(x, y) = e^{-\|x-y\|^2/2\sigma^2}$
- $K(x, y) = \tanh(\kappa \langle x, y \rangle - \sigma)$

Nonlinear support vector machines can be considered as statistical methods. They need to be provided with a particular distribution (Kernel function), which task is difficult, in practice, to apply rigorously. Choosing the kernel function is a particularly difficult task when this function is very complex.

Considering the problem (1.10) suppose that it is known that $f \in \mathcal{F}$, where \mathcal{F} is a family of functions. Then, one can define the set

$$\Phi = \{(\varphi : \mathbb{R}^n \rightarrow \mathbb{R}^m) : (x \mapsto \langle w, \varphi(x) \rangle) \in \mathcal{F}\},$$

and deduce an associated function K . Then SVM provide an elegant and efficient method for solving the set-separation problem.

In reality, it is very rarely possible to apply this scheme, as the conditions necessary for it are:

- The family \mathcal{F} should be known;
- It should be possible to deduce from it the set Φ (in the case it even exists);
- The function K should be explicitly derived from Φ .

The practical application of SVM is based on arbitrary choices of K , which may range from a “wild guess” to the judgement of an expert.

1.3.6 Polyhedral separability

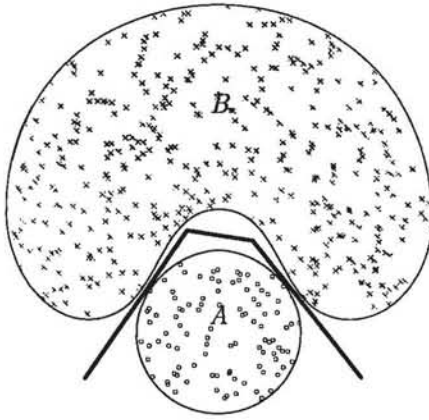


Figure 1.6: Polyhedral separation of sets

The concept of h -polyhedral separability was developed in [8]. The sets A and B are h -polyhedrally separable if there exists a set of h hyperplanes $\{x^i, y_i\}$, with

$$x^i \in \mathbb{R}^n, y_i \in \mathbb{R}^1, 1 \leq i \leq h$$

such that

1. for any $j \in \{1, \dots, m\}$ and $i \in \{1, \dots, h\}$

$$\langle x^i, a^j \rangle - y_i < 0,$$

2. for any $k \in \{1, \dots, p\}$ there exists at least one $i \in \{1, \dots, h\}$ such that

$$\langle x^i, b^k \rangle - y_i > 0.$$

It is proved in [8] that the sets A and B are h -polyhedrally separable, for some $h \leq p$ if and only if

$$\text{co}(A) \cap B = \emptyset.$$

The problem of polyhedral separability of the sets A and B is reduced to the following problem:

$$\text{minimise } f(x, y) \text{ subject to } (x, y) \in \mathbb{R}^{(n+1) \times h} \quad (1.18)$$

where

$$\begin{aligned} f(x, y) = & \frac{1}{m} \sum_{j=1}^m \max \left[0, \max_{1 \leq i \leq h} \{ \langle x^i, a^j \rangle - y_i + 1 \} \right] \\ & + \frac{1}{p} \sum_{k=1}^p \max \left[0, \min_{1 \leq i \leq h} \{ -\langle x^i, b^k \rangle + y_i + 1 \} \right] \end{aligned}$$

is an error function.

Note 1: This function is nonconvex piecewise linear.

It is proved in [8] that $x^i = 0, 1 \leq i \leq h, y \in \mathbb{R}$ cannot be the optimal solution.

Let $\{\bar{x}^i, \bar{y}_i\}, 1 \leq i \leq h$ be a global solution to the problem (1.18). The sets A and B are h -polyhedrally separable if and only if $f(\bar{x}, \bar{y}) = 0$. If there exists a nonempty set $\bar{I} \subset \{1, \dots, h\}$ such that $x^i = 0, i \in \bar{I}$, then the sets A and B are $(h - |\bar{I}|)$ -polyhedrally separable.

In [8] an algorithm for solving problem (1.18) is developed. The calculation of the descent direction at each iteration of this algorithm is reduced to a certain linear programming problem.

1.3.7 Classification via clustering

Independently from linear and polyhedral separability, the problem of separating two sets can be studied using the clustering approach (see [145] for references on the application of unsupervised learning for classification).

Suppose we are given two sets of points A and B . Then the structure of each of these sets can be studied separately. From the structure of the two sets, a separating function can then be deduced.

Suppose we formulate the clustering problem under its (1.4) form.

Separating two sets

An approach for separating two sets of points $A = \{a_i\}$ and $B = \{b_i\}$ based on clustering was proposed in [24]. For each set, the clustering problem is solved, respectively with q_A and q_B clusters. The solutions obtained are $\mathcal{X} = \{x_A^1, \dots, x_A^{q_A}, x_B^1, \dots, x_B^{q_B}\}$. The separating function is then:

$$f(y) = \min_{x \in \mathcal{X}} \left\{ \min_{0 \leq j \leq q_A} \|x_A^j - y\|_p^\gamma, \min_{0 \leq j \leq q_B} \|x_B^j - y\|_p^\gamma \right\} \quad (1.19)$$

If the point y belongs to a cluster of the set A then it belongs to the set A . Otherwise it belongs to the set B .

This technique can be seen as a form of piecewise linear separation of the sets: each pair of centres is separated by a line. The separation between the centres of the set A and those of the set B is thus piecewise linear (see figure 1.7).

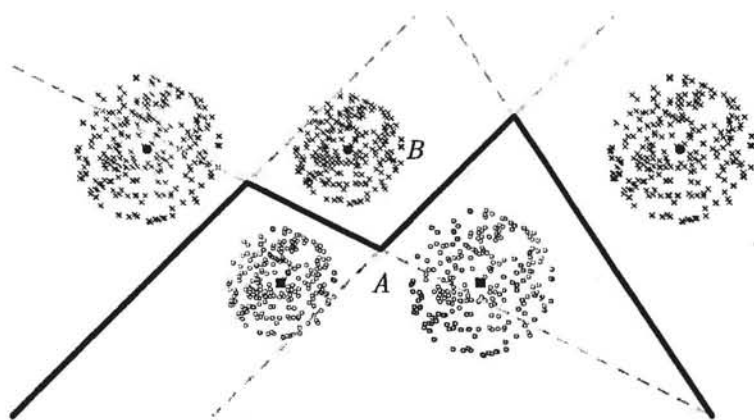


Figure 1.7: Piecewise linear separation using clustering

The advantage of this technique is that it does not restrict the search to only a convex polyhedron, and thus allows both the sets A and B to be nonconvex. One disadvantage, however, is that it only considers the sets separately.

1.3.8 Max-min separability

Max-min separability is a generalisation for the polyhedral separability introduced in [23].

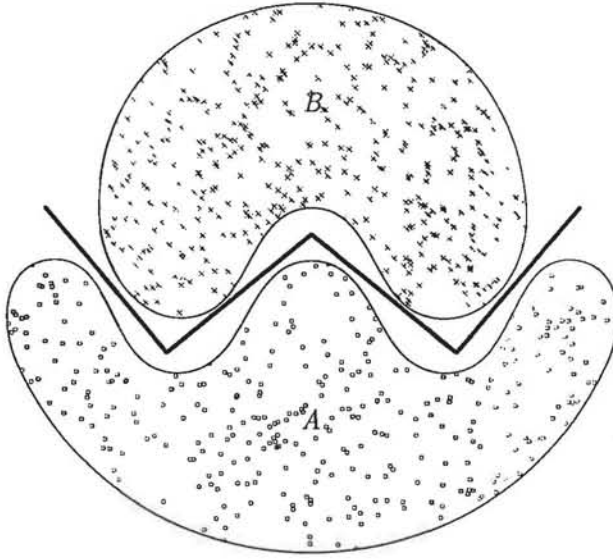


Figure 1.8: Max-min separability.

Definition and properties

Let $H = \{h_1, \dots, h_l\}$, where $h_j = \{x^j, y_j\}$, $1 \leq j \leq l$ with $x^j \in \mathbb{R}^n$, $y_j \in \mathbb{R}^1$, be a finite set of hyperplanes. Let $J = \{1, \dots, l\}$. Consider any partition of this set $J^r = \{J_1, \dots, J_r\}$ such that

$$\left\{ \begin{array}{l} J_k \neq \emptyset, 1 \leq k \leq r; \\ J_k \cap J_j = \emptyset; \\ \bigcup_{k=1}^r J_k = J. \end{array} \right.$$

Let $I = \{1, \dots, r\}$. A particular partition $J^r = \{J_1, \dots, J_r\}$ of the set J defines the following max-min-type function:

$$\varphi(z) = \max_{i \in I} \min_{j \in J_i} \{ \langle x^j, z \rangle - y_j \}, z \in \mathbb{R}^n. \quad (1.20)$$

Let $A, B \subset \mathbb{R}^n$ be two disjoint sets, that is $A \cap B = \emptyset$.

Definition 2. The sets A and B are *max-min separable* (*piecewise linearly separable*) if there exist a finite number of hyperplanes $\{x^j, y_j\}$ with $x^j \in \mathbb{R}^n$, $y_j \in \mathbb{R}^1$, $j \in J = \{1, \dots, l\}$ and a partition $J^r = \{J_1, \dots, J_r\}$ of the set J such that

1. for all $i \in I$ and $a \in A$

$$\min_{j \in J_i} \{ \langle x^j, a \rangle - y_j \} < 0;$$

2. for any $b \in B$ there exists at least one $i \in I$ such that

$$\min_{j \in J_i} \{ \langle x^j, b \rangle - y_j \} > 0.$$

Remark 7: It follows from definition 2 that if the sets A and B are max-min separable then $\varphi(a) < 0$ for any $a \in A$ and $\varphi(b) > 0$ for any $b \in B$, where the function φ is defined by (1.20). Thus the sets A and B can be separated by a function represented as max-min of linear functions. Therefore this kind of separability is called *max-min separability*.

Remark 8: Linear and polyhedral separabilities can be considered as particular cases of the max-min separability. If $I = \{1\}$ and $J_1 = \{1\}$ then we have the linear separability and if $I = \{1, \dots, h\}$ and $J_i = \{i\}, i \in I$ we obtain the h -polyhedral separability.

Proposition 1. (see [23]). *The sets A and B are max-min separable if and only if there exists a set of hyperplanes $\{x^j, y_j\}$ with $x^j \in \mathbb{R}^n, y_j \in \mathbb{R}^1, j \in J$ and a partition $J^r = \{J_1, \dots, J_r\}$ of the set J such that*

1. for any $i \in I$ and $a \in A$

$$\min_{j \in J_i} \{ \langle x^j, a \rangle - y_j \} \leq -1;$$

2. for any $b \in B$ there exists at least one $i \in I$ such that

$$\min_{j \in J_i} \{ \langle x^j, b \rangle - y_j \} \geq 1.$$

Proposition 2. (see [23]). *The sets A and B are max-min separable if and only if there exists a piecewise linear function separating them.*

Remark 9: It follows from proposition 2 that the notions of max-min and piecewise linear separability are equivalent. For this reason, max-min separability is also called *piecewise linear separability*.

Proposition 3. *Assume that the set A can be represented as a union of sets $A_i, 1 \leq i \leq q$:*

$$A = \bigcup_{i=1}^q A_i$$

and for any $1 \leq i \leq q$

$$B \cap \text{co } A_i = \emptyset. \tag{1.21}$$

Then the sets A and B are max-min separable.

Corollary 1. (see [23]). The sets A and B are max-min separable if and only if they are disjoint: $A \cap B = \emptyset$.

In the next proposition it is shown that in most cases the number of hyperplanes necessary for the max-min separation of the sets A and B is limited.

Proposition 4. (see [23]). Assume that the set A can be represented as a union of sets $A_i, 1 \leq i \leq q$ and the set B as a union of sets $B_j, 1 \leq j \leq d$ such that

$$A = \bigcup_{i=1}^q A_i \text{ and } B = \bigcup_{j=1}^d B_j$$

and

$$\text{co } A_i \cap \text{co } B_j = \emptyset, \forall 1 \leq i \leq q, 1 \leq j \leq d. \quad (1.22)$$

Then the number of hyperplanes necessary for the separation of the sets A and B is at most $q \cdot d$.

Remark 10: The only cases where the number of hyperplanes necessary is large are when the sets A_i and B_j contain a very small number of points. This situation appears only in the particular case where the distribution of the points is like a "chessboard".

Error function

Given any set of hyperplanes $\{x^j, y_j\}, j \in J = \{1, \dots, l\}$ with $x^j \in \mathbb{R}^n, y_j \in \mathbb{R}^1$ and a partition $J^r = \{J_1, \dots, J_r\}$ of the set J , we say that a point $a \in A$ is well separated from the set B if the following condition is satisfied:

$$\max_{i \in I} \min_{j \in J_i} \{\langle x^j, a \rangle - y_j\} + 1 \leq 0.$$

Then we can define the separation error for a point $a \in A$ as follows:

$$\max \left[0, \max_{i \in I} \min_{j \in J_i} \{\langle x^j, a \rangle - y_j + 1\} \right]. \quad (1.23)$$

Analogously, a point $b \in B$ is said to be well separated from the set A if the following condition is satisfied:

$$\min_{i \in I} \max_{j \in J_i} \{-\langle x^j, b \rangle + y_j\} + 1 \leq 0.$$

Then the separation error for a point $b \in B$ can be written as

$$\max \left[0, \min_{i \in I} \max_{j \in J_i} \{-\langle x^j, b \rangle + y_j + 1\} \right]. \quad (1.24)$$

Thus, an averaged error function can be defined as

$$f(x, y) = \frac{1}{m} \sum_{k=1}^m \max \left[0, \max_{i \in I} \min_{j \in J_i} \{ \langle x^j, a^k \rangle - y_j + 1 \} \right] + \frac{1}{p} \sum_{t=1}^p \max \left[0, \min_{i \in I} \max_{j \in J_i} \{ -\langle x^j, b^t \rangle + y_j + 1 \} \right] \quad (1.25)$$

where $x = (x^1, \dots, x^l) \in \mathbb{R}^{l \times n}$, $y = (y_1, \dots, y_l) \in \mathbb{R}^l$.

It is clear that $f(x, y) \geq 0$ for all $x \in \mathbb{R}^{l \times n}$ and $y \in \mathbb{R}^l$.

Proposition 5. (see [23]). *The sets A and B are max-min separable if and only if there exists a set of hyperplanes $\{x^j, y_j\}$, $j \in J = \{1, \dots, l\}$ and a partition $J^r = \{J_1, \dots, J_r\}$ of the set J such that $f(x, y) = 0$.*

Proposition 6. (see [23]). *Assume that the sets A and B are max-min separable with a set of hyperplanes $\{x^j, y_j\}$, $j \in J = \{1, \dots, l\}$ and a partition $J^r = \{J_1, \dots, J_r\}$ of the set J . Then*

1. $x^j = 0$, $j \in J$ cannot be an optimal solution;

2. if

(a) for any $t \in I$ there exists at least one $b \in B$ such that

$$\max_{j \in J_t} \{ -\langle x^j, b \rangle + y_j + 1 \} = \min_{i \in I} \max_{j \in J_i} \{ -\langle x^j, b \rangle + y_j + 1 \}, \quad (1.26)$$

(b) there exists $\bar{J} = \{\bar{J}_1, \dots, \bar{J}_r\}$ such that $\bar{J}_t \subset J_t, \forall t \in I$, \bar{J}_t is nonempty at least for one $t \in I$ and $x^j = 0$ for any $j \in \bar{J}_t, t \in I$.

Then the sets A and B are max-min separable with a set of hyperplanes $\{x^j, y_j\}$, $j \in J^0$ and a partition $\bar{J} = \{\bar{J}_1, \dots, \bar{J}_r\}$ of the set J^0 where

$$\bar{J}_t = J_t \setminus \bar{J}_t, t \in I \text{ and } J^0 = \bigcup_{i=1}^r \bar{J}_i.$$

Remark 11: In most cases, if a given set of hyperplanes with a particular partition separates the sets A and B , then there are other sets of hyperplanes with the same partition which will also separate the sets A and B (see figure 1.8). The error function (1.25) is nonconvex and if the sets A and B are max-min separable, then the global minimum of this function $f(x^*, y_*) = 0$ and the global minimiser is not unique (see figure 1.9).

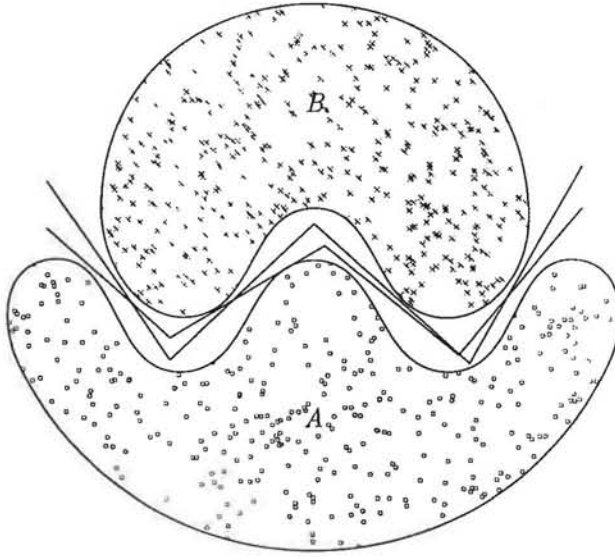


Figure 1.9: Several piecewise linear separators for the same sets

The family of piecewise linear functions is particularly attractive for solving the set separation problem. Indeed, since only approximations of the sets A and B are known, it is in any case impossible to find exactly the separating function. Furthermore, it is well known that it is possible to approximate any continuous function as close as wanted using a piecewise linear function. Therefore, it seems natural to consider this family for set separation.

Unfortunately, the optimisation problem generated is not as simple as the Support Vector Machines, and the problem of selecting the best of two separating functions still has to be considered. Both these methods seem to present advantages and disadvantages, but the max-min separability seems to show more potential for improvements.

1.4 Preprocessing

Most algorithms usually work better under certain conditions. Consequently, the data should be preprocessed in order to fit these conditions the best possible. Since the data is not known beforehand, the processing techniques should be as general as possible.

1.4.1 Scaling

One most common preprocessing need is the scaling. Indeed, it is commonly admitted that polythetic algorithms (The features are simultaneously considered) work best. Hence, the necessity not to have any dominating nor any dominated feature is clear. In such a case, the result would depend only on the dominating feature.

Example 8: The results of most algorithms on the dataset presented on figure 1.10 strongly depend on the preprocessing on this dataset: in the case 1.10(a), the dataset is ill-scaled, and the points A_1 and A_2 have strong chances to be attributed to the wrong clusters (or classes). In contrast, the problems are much easier to solve on the same dataset with a good scaling, like in 1.10(b).

In most cases selecting the best scaling is not easy: the shapes of the clusters or classes are not known in advance, and it could be quite easy to worsen a situation.

Here we give three scaling techniques commonly used. figure 1.11 presents the results of these scaling techniques on the ill scaled dataset shown on figure 1.10(a). Notice that all of them give a satisfactory scaling in this simple case.

Scaling by average

The principle of this scaling is to obtain a dataset for which the average of each feature is the same (usually 1). This scaling is described in Algorithm 1.2.

Algorithm 1.2: Scaling by average

	<i>for each feature do</i>
Step 1	$\text{Let } \mu = \frac{1}{\text{card}(A)} \sum_{a \in A} a.$
Step 2	$\text{For each } a \in A \text{ let } a' = \frac{a}{\mu}.$

One of the possible drawbacks of this approach is that it depends on the “zero” for each feature. For example, the fact that the temperature can be measured in Fahrenheits, in Celsius or in Kelvins may give totally different results, as these scales present different 0.

It may be necessary for this scaling algorithm to take into account the features with means 0. In this case, a slight modification of the algorithm can easily solve this problem.

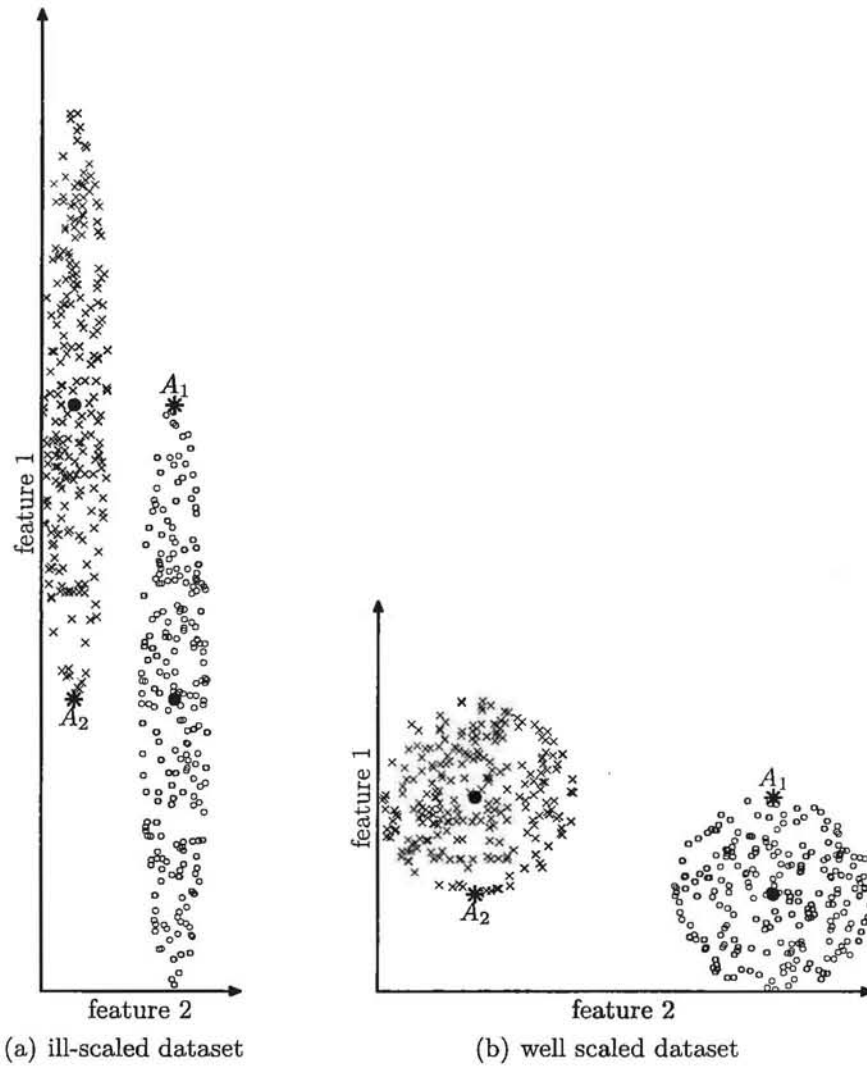


Figure 1.10: Two different scalings for the same dataset

Scaling by average and variance

The idea behind this scaling is that we do not only want all the features to have the same average, but also that the repartition of the points around it is similar. Algorithm 1.3 presents it more in details.

The problem of this algorithm is that the repartition of the points may be disturbed.

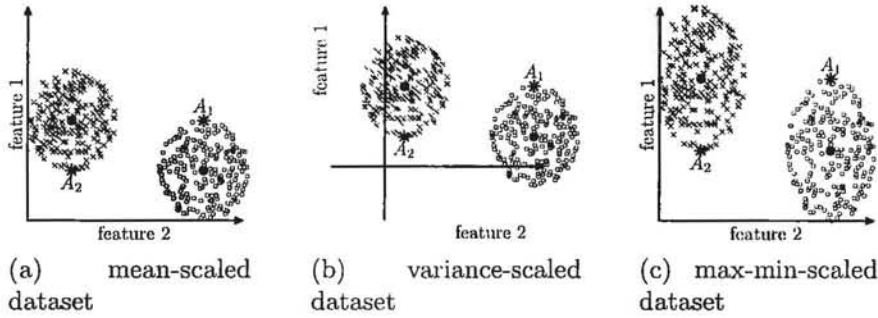


Figure 1.11: Effects of various scalings on a dataset

Algorithm 1.3: Scaling by average and variance

```

for each feature do
Step 1  Let  $\mu = \frac{1}{\text{card}(A)} \sum_{a \in A} a.$ 
Step 2  Let  $\sigma = \sqrt{\sum_{a \in A} (a - \mu)^2}.$ 
Step 3  For each  $a \in A$  let  $a' = \frac{a - \mu}{\sigma}.$ 

```

Scaling in an interval

It may be that some applications necessitate scaling all the features within the same interval, to keep the repartition intact. Algorithm 1.4 presents the pseudo-code for this algorithm.

Algorithm 1.4: Scaling in an interval

```

for each feature do
Step 1  Let  $\alpha_1 = \min_{a \in A} a$  and  $\alpha_2 = \max_{a \in A} a.$ 
Step 2  For each  $a \in A$  let  $a' = \frac{a - \alpha_1}{\alpha_2 - \alpha_1}.$ 

```

The drawback of this method is that it is very sensitive to measurement errors or erroneous points.

1.4.2 Feature selection

Another major issue for the quality of algorithms is the pertinence of the features. Selecting a smaller set of features sufficient to solve the given data analysis problem is important for two reasons:

- The presence of noisy features is likely to influence the quality of the results given by the algorithms. While not bringing any useful information, a feature may increase the dissimilarity between observations in reality close to one another.
- In most optimisation problems in the data analysis field the number of variables is proportional to the number of features. More generally, the complexity of the problems encountered in data analysis presumably depends on this number. Hence reducing it without loss of useful information is expected to accelerate the algorithms and increase the accuracy.

Most feature selections are based on statistical considerations, and the features are usually eliminated according to a correlation between observations and features (see [36, 82]). In [110], an approach based on evolutionary algorithms has been developed, while [9, 20] present solutions based on optimisation techniques.

1.4.3 Dataset reduction

Nowadays, the datasets used in data analysis are often extremely large, counting millions of observations. In some cases, data started to be collected as early as the 70's (for example the ADRAC database). In other cases, like in the Stock Market, information is added daily.

Although the more information the better, algorithms are seldom able to deal with such large numbers of data, and it may be a judicious choice to design an algorithm working well on a subset of the data than a "quick and dirty" algorithm, able to work on the whole dataset, but providing deceiving results.

Even though the number of variables of most optimisation problems we consider in data analysis does not depend on the number of observations in the dataset, too large datasets do influence the efficiency of the algorithms. Particularly when the number of variables is large, it is necessary to evaluate the objective functions many times. The computational effort for these evaluations is directly related to the number of records in the dataset. Therefore, it is often advantageous to apply a dataset reduction scheme before solving the problem.

Additionally, a large number of records in the dataset often leads to objective functions having a large number of local minima, many of them not very deep, and therefore not acceptable at all. Decreasing the number of observations is a way to reduce the difficulty of solving the problems.

An alternative which has been explored recently is to elaborate some datasets which somehow “summarise” the original datasets. The ε -cleaning dataset (see [24]) is a method providing such reduced datasets, as shown in algorithm 1.5.

Algorithm 1.5: ε -cleaning procedure

input : A very large dataset A .
output : A not so large dataset B .

Step 1 Initialisation:
 Select $\varepsilon > 0$. Set $A_1 = A$ and $i = 1$
repeat

Step 2 | Select $b^i \in A_i$
Step 3 | Let $A_{b^i} \leftarrow \{a \in A_i : d(a, b^i) < \varepsilon\}$
Step 4 | Set $w_i \leftarrow \text{card}(A_{b^i})$
Step 5 | Let $A_{i+1} \leftarrow A_i \setminus A_{b^i}$
Step 6 | Set $i \leftarrow i + 1$

until $A_i = \emptyset$

Depending on ε , the set B is much smaller than the set A . Varying ε allows to get a balance between the size of the dataset and the accuracy of the representation. The weights w_i are very important, as they enable the modification of the objective functions accordingly.

The ε -cleaning procedure should provide a good approximation for the function (1.4). It has been shown in [181] that if we use the *generalised cluster function*

$$\tilde{f}(x) = \frac{1}{m} \sum_{i=1}^{m_B} w_i \min_{1 \leq s \leq q} d(x, b) \quad (1.27)$$

instead of the original function f , we get, for all $x \in \mathbb{R}^n$, $|\tilde{f}(x) - f(x)| < \varepsilon$.

The minimisation of data analysis functions for datasets with weights has been studied in [78].

Notice that the selection of the representatives b^i in step 4 of the algorithm can be different. For example it is possible to take the barycentre of the set B_i .

although the final dataset depends on the order in which the observations are processed, results of numerical experiments show that the order does not influence the results very strongly.

1.5 Test datasets

In this section we present all the test datasets that will be used for numerical experiments in this thesis.

1.5.1 Supervised learning

Letters The dataset was introduced by David Stale. It is based on various fonts representation. The dataset consists of 20000 observations, 26 classes, 16 numerical attributes. There are samples of 26 capital letters, printed in different fonts. 20 fonts have been considered and the location of the corresponding samples has been distributed randomly within the dataset (see [145]).

Pen-based recognition of handwritten digits (Pendigits) This dataset was introduced by E. Alpaydin and Fevzi Alimoglu. It contains 10 classes, 10992 observations, 16 attributes. All input attributes are integers between 0 and 9.

The dataset has been created by collecting 250 samples from 44 writers. The samples written by 30 writers are used for the training set and the digits written by the other 14 are used for writer independent testing. These writers are asked to write 250 digits in random order inside boxes of 500 by 500 tablet pixel resolution. The first ten digits are ignored because most writers are not familiar with this type of input device, but subjects are not aware of this (see [145]).

1.5.2 Unsupervised learning

Fisher's iris dataset This dataset contains 50 samples of 4 measurements for 3 types of irises (4 features, 150 observations). The initial goal is to establish rules to differentiate between the types of irises.(see [72]).

Image segmentation dataset This database contains information about 7 outdoors images, segmented to obtain a classification of every pixel. Each instance is a 3x3 region. The dataset contains 19 features, 2310 observations [150].

The travelling salesman problem - TSPLIB1060 2 features, 1060 observations [170].

The travelling salesman problem - TSPLIB3038 2 features, 3038 observations [170].

Heart disease database 13 features, 364 observations [145].

Diabetes database 8 features, 768 observations [145].

Australian credit cards database 14 features, 690 observations [145].

Database	training	test	No. of attributes	No. of classes
Shuttle control	43500	14500	9	7
Letter recognition	15000	5000	16	26
Landsat satellite image	4435	2000	36	6
Pendigits	7494	3498	16	10
Page blocks	4000	1473	10	5
Fisher's iris dataset	150	-	4	-
Image segmentation dataset	2310	-	19	-
TSPLIB1060	1060	-	2	-
TSPLIB3038	3038	-	2	-
Heart disease	364	-	13	-
Diabetes	768	-	8	-
Australian credit cards	690	-	14	-

Table 1.1: Datasets in use in this thesis

1.6 Summary

A large choice of methods is available for the most common data analysis techniques. The variability between these methods mostly stands on their efficiency, their applicability and their portability.

While statistical and heuristic methods are to this day the fastest methods to apply, they do not provide a large freedom of application: if one method works for a particular dataset, there is not guarantee that it will be successful on another dataset.

Statistical methods are more efficient on large datasets, and may not give very good results when the number of records is small. Moreover, for a long period these methods were based on particular distributions. Today, much more efficient and portable statistical methods are designed with no assumption on the distribution of the points in the database.

Heuristic methods are less limiting on the number of records. However, they suffer from the same problem as classical statistical algorithm: they are

not very portable, and a method working well in one case may completely fail in other (not necessarily more complex) cases.

Because optimisation based methods are not limited in the number of variables, nor by a particular black-box scheme, they give more freedom to the expert. The same method can be applied to various fields, under various conditions, just by changing the error function.

However, they necessitate a good optimisation solver. The problems arising in data mining are very demanding, and most current general-purpose optimisation methods may require too much resource to be applied "out of the box". Therefore it is necessary to develop new methods, or to adapt the methods to the types of error functions that arise in classification and clustering.

Chapter 2

Optimisation methods

2.1 Generalities

2.1.1 Presentation

Definition of the problem

The field of optimisation is very large and has many applications. Its purpose is to find the “best” solution to a given problem under particular circumstances, by choosing suitable values for a set of given variables.

It is possible to find excellent reviews for optimisation. Classical results of mathematical analysis provided in [49, 59, 60, 171] are unequalled. Good generalist presentations about optimisation can be found in [54, 74, 94, 95, 97, 98, 157, 160, 161]. Most results presented in this chapter are detailed in these. Proofs of theorems and of the termination of algorithms will be omitted, as they can be easily found in the literature.

In mathematical programming, each variable takes a numerical value and the list of variables can consequently be written as a vector $x \in \mathcal{A} \subset \mathbb{R}^n$, where n is the number of variables. The vector x is also called a *feasible solution*.

The quality of a given set of variables is measured with a function $q : \mathcal{A} \rightarrow \mathbb{R}$. The higher the value of the function q the better the quality of the parameters. The goal of optimisation becomes to *maximise* the quality. The function q is also sometimes called *fitness function*. Alternatively, it is possible to define a *cost function* $c : \mathcal{A} \rightarrow \mathbb{R}$. The goal of optimisation is then to *minimise* the cost. Clearly the two problems are equivalent, as the cost can be defined as the opposite to the fitness: $c = -q$. By convention, most optimisation procedures are designed to minimise.

A very general formulation of the optimisation problems is then:

$$\begin{aligned} & \text{minimise } f(x) \\ & \text{subject to} \\ & x \in \mathcal{C} \end{aligned} \tag{2.1}$$

The function f is also called *objective function*.

Reformulation of the continuous optimisation problem

In the case of constraint optimisation, the set \mathcal{C} is usually defined by a set of inequalities and/or a set of equalities. The general continuous optimisation problem can then be rewritten:

$$\begin{aligned} & \text{minimise } f(x) \\ & \text{subject to} \\ & g_i(x) \leq 0, \quad \forall 1 \leq i \leq n_i \\ & h_j(x) = 0, \quad \forall 1 \leq j \leq n_e \\ & x \in \mathbb{R}^n. \end{aligned} \tag{2.2}$$

If $n_e = n_i = 0$ then problem (2.2) is an unconstrained problem, otherwise it is a constrained problem and the functions g_i , $1 \leq i \leq n_i$ and h_j , $1 \leq j \leq n_e$ are called the *constraints* of the problem. Notice that the convexity of an unconstrained problem only depends on the convexity of the objective function.

Remark 12: There exists a very large number of techniques which study the problem of taking constraints into account.

For a smooth programming problem, the most efficient among these techniques is based on Karush-Kuhn-Tucker theorem (see theorem 1 on the following page).

This theorem is difficult to generalise to the nonsmooth case, and therefore other methods have to be employed. Among such methods one of the simplest is to incorporate the constraints into the objective function using a *penalty function* p such that:

$$f = \begin{cases} p(x) = 0 & \text{if } x \in \mathcal{C}; \\ p(x) > 0 & \text{if } x \notin \mathcal{C} \end{cases}$$

Usually the problem is then reduced to the following unconstrained problem:

$$\text{minimise } f(x) + \lambda p(x)$$

where $\lambda \gg 0$ is a *penalty parameter*.

Theorem 1 (Karush-Kuhn-Tucker). *Considering problem 2.2 on the previous page, where f, g_i and h_j are all differentiable. Then if $x^* \in \mathbb{R}^n$ is a local minimizer, there exists $u_i \in \mathbb{R}^+, v_j \in \mathbb{R}$ such that:*

$$\nabla f(x^*) + \sum_{i=1}^{n_i} u_i \nabla g_i(x^*) + \sum_{j=1}^{n_j} v_j \nabla h_j(x^*) = 0.$$

$$u_i g_i = 0.$$

Definition 3. The function

$$L : (u, v, x) \mapsto f(x) + \sum_{i=1}^{n_i} u_i g_i(x) + \sum_{j=1}^{n_j} v_j h_j(x)$$

is called the Lagrangian .

The most difficult type of problems is the one where the properties of the function and of the domain cannot be exploited, that is nonsmooth nonconvex optimisation.

2.1.2 Notions of mathematical analysis

Convexity

Convexity plays an important role in optimisation.

Definition 4. A set A is convex if for any $x, y \in A$, for any $t \in [0, 1]$

$$tx + (1 - t)y \in A.$$

Definition 5. Consider the function $f : \mathcal{C} \rightarrow \mathbb{R}$. The set

$$epi(f) = \{(x, y) \in \mathcal{C} \times \mathbb{R} : y \geq f(x)\}$$

is called the *epigraph* of the function f .

Definition 6. A function f is convex if its epigraph is convex.

Proposition 7. A function f is convex if for any x, y , for any t

$$tf(x) + (1 - t)f(y) \geq f(tx + (1 - t)y).$$

Global vs local minimum

Consider a function $f : \mathcal{D}_f \rightarrow \mathbb{R}$.

Definition 7. A vector x is considered a *global minimiser* of f on \mathcal{D}_f if

$$f(x) \leq f(y), \forall y \in \mathcal{D}_f.$$

$f(x)$ is the *global minimum* for f on \mathcal{D}_f .

Definition 8. A vector x is considered a *local minimiser* of f on \mathcal{D}_f if $\exists \varepsilon > 0$ such that:

$$f(x) \leq f(y), \forall y \in \mathcal{D}_f \cap \mathcal{B}(x, \varepsilon).$$

$f(x)$ is a *local minimum* for f on \mathcal{D}_f .

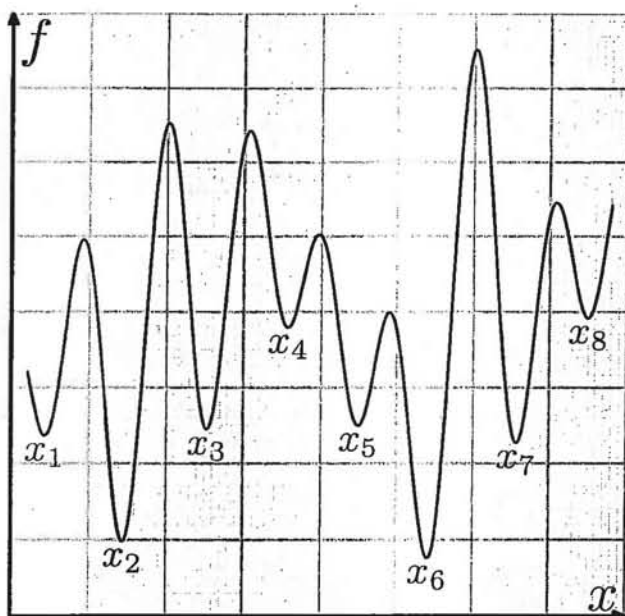


Figure 2.1: Function having 8 local minima and only one global one

The function represented on figure 2.1 has 8 local minima: $x_i, 1 \leq i \leq 8$. For instance, x_1 is clearly a local minimiser, since in its neighbourhood the function is minimal at x_1 . However, because $f(x_2) < f(x_1)$, x_1 is not a global minimum.

The point x_6 is the only global minimum of the function f .

In the case of convex optimisation, any local minimum is also a global minimum. However this is not necessarily true when f or \mathcal{D}_f are not convex.

Finding a local minimum is easier than finding a global minimum, as it is sufficient to verify the local properties of the function around a vector x to know whether it is a solution or not. Accordingly, many efficient methods have been developed for the convex optimisation problem, and most of these methods can be applied to find a local solution to the general problem 2.2.

Although it is sometimes important to reach the global solution, in many practical problems it is satisfactory enough to find a “deep” local minimum. This, however, raises the question of deciding whether a local minimum is “deep” enough, and more generally to be able to evaluate the depth of a local minimum. These questions are in turn related to the problem of knowing the global minimum.

For instance, on figure 2.1, although it is not a global minimiser, the point x_2 is likely to be a satisfactory enough solution for practical purposes. Nonetheless the only reason we can assess that is because we know the global minimum of the function and can compare the function values. On the other hand, the point x_8 is likely to be dismissed as a good solution. However, deciding whether intermediate points like x_5 or x_7 are satisfactory is a difficult task.

Moreover, the function can be *noisy*, that is a slight random perturbation can be introduced due to measurement errors. Figure 2.2 presents a noisy version of the function from figure 2.1.

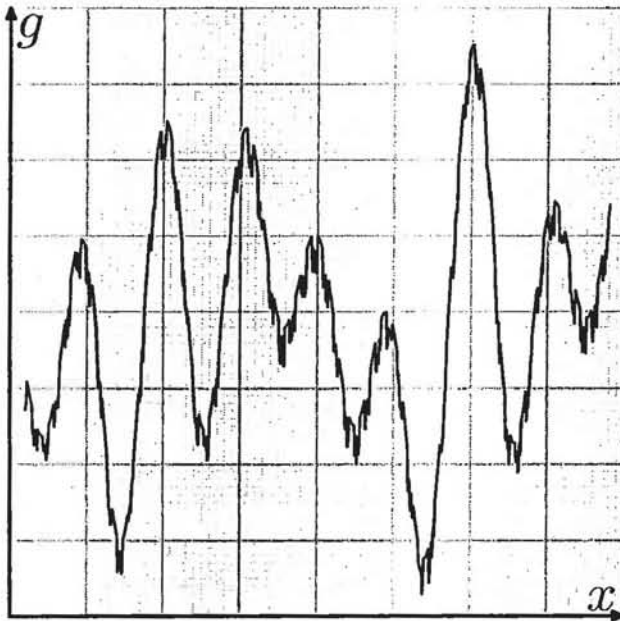


Figure 2.2: Noisy function

Lipschitz continuity

Definition 9. A function f is said to be Lipschitz continuous if

$$\exists L \in \mathbb{R}, \forall (x, y) \in \mathcal{D}_f : |f(y) - f(x)| \leq L\|y - x\|.$$

The smallest L for which f verifies this is called the *Lipschitz constant* of f .

Definition 10. A function f is said to be locally Lipschitz on \mathcal{D} if it is Lipschitz on any bounded subset $\bar{D} \in \mathcal{D}$

2.2 Analytical considerations

Differentiability is a very powerful tool of mathematical analysis. Notwithstanding the efficiency of the optimisation methods based on this tool, few practical problems of optimisation have a smooth structure. This consideration means much research effort has to be done on the generalisation of the notion of gradient. Here we will present two major contributions: the Clarke subdifferential and the quasidifferential in the sense of Demyanov-Rubinov.

2.2.1 Preliminaries

Consider a function f defined on an open set $\Omega \subset \mathbb{R}^n$.

Definition 11. A function h is called *positively homogeneous* if

$$h(\lambda x) = \lambda h(x), \forall x \in \mathbb{R}^n, \forall \lambda > 0.$$

Definition 12. A convex positively homogeneous function is called *sublinear*. A concave positively homogeneous function is called *superlinear*.

Let us denote P (resp. Q) the set of all sublinear (resp. superlinear) functions defined on \mathbb{R}^n .

Definition 13.

$$\partial p = \{u \text{ linear} : u(x) \leq p(x), \forall x \in \mathbb{R}^n\}$$

is the *subdifferential* of the function $p \in P$. Similarly

$$\bar{\partial} q = \{u \text{ linear} : u(x) \geq q(x), \forall x \in \mathbb{R}^n\}$$

is the *superdifferential* of the function $q \in Q$.

The information provided by the directional derivatives is very useful in the field of optimisation. Indeed, they allow one to know whether a direction is descending or ascending. However the set of all directional derivatives is infinite in the multidimensional case. For smooth functions, it can be represented by one vector: the gradient. In the nonsmooth case, we need to create tools that approximate these sets.

Consider the (Dini or Hadamard) directional derivative as a function of the direction: $f'_x(u)$. Assume the continuity of the function f'_x .

Definition 14. A continuous sublinear function p defined on \mathbb{R}^n and such that

$$p(g) \geq f'_x(g), \forall g \in \mathbb{R}^n$$

is called an *upper convex approximation (u.c.a)* of the function f at the point x .

Definition 15. A continuous superlinear function q defined on \mathbb{R}^n and such that

$$q(g) \geq f'_x(g), \forall g \in \mathbb{R}^n$$

is called an *lower concave approximation (l.c.a)* of the function f at the point x .

Definition 16. (see [146]) A function f is said to be *semismooth* at x if it is locally Lipschitz continuous at x and the limit

$$\lim_{\substack{v \in \partial f(x + \alpha g'), \\ g' \rightarrow g, \\ \alpha \rightarrow 0}} \langle v, g \rangle$$

exists for every $g \in \mathbb{R}^n$.

In order to consider generalisations of the gradient, the following definition is capital (see [171]).

Definition 17. Suppose f is a convex function. Then a linear function dominated by f is called a *subgradient* of f of f at $x \in \mathbb{R}^n$. The set of the subgradients of f at x

$$\partial_c f(x) = \{ \xi \in \mathbb{R}^n : f(y) \geq f(x) + \xi^T(y - x), \forall y \in \mathbb{R}^n \}$$

is called the *subdifferential* of f at x .

The notion of subgradient is extremely important in convex analysis. A large majority of practical methods to minimise convex functions revolves around this notion.

2.2.2 Clarke subdifferential

The Clarke subdifferential is one of the simplest generalisations of the notion of gradient. This notion has been studied for example in [48, 49]. It has been developed for locally Lipschitz functions. It is known that these functions are continuous almost everywhere.

Definition 18. The *Clarke subdifferential* is defined as follows:

$$\partial f(x) = \text{co} \left\{ \lim_{i \rightarrow \infty} \nabla f(x_i) : x_i \xrightarrow{i \rightarrow \infty} x \text{ and } x_i \in D_f \right\}. \quad (2.3)$$

Note 2: The Clarke subdifferential of a convex function coincides with its subdifferential.

Proposition 8. $0 \in \partial f(x)$ if and only if x is a stationary point. In particular, if the point $x \in \mathbb{R}^n$ is a minimum of the function f then $0 \in \partial f$.

Proposition 9. The steepest descent at point $x \in \mathbb{R}^n$ is the direction $g^0 \in \mathbb{R}^n$ such that:

$$\|g^0\| = \min_{g \in \partial f(x)} \{\|g\|\} \quad (2.4)$$

Definition 19. The *generalised directional derivative* is the limit:

$$f^0(x, g) = \limsup_{\substack{\alpha \rightarrow 0^+, \\ y \rightarrow x}} \frac{f(y + \alpha g) - f(y)}{\alpha}$$

Note 3: The generalised derivative always exists, and

$$f^0(x, g) = \max_{v \in \partial f(x)} \langle v, g \rangle.$$

Definition 20. The function f is said *Clarke regular* if it is differentiable with respect to any direction $g \in \mathbb{R}^n$ and its directional derivatives and generalised derivatives are equal.

$$f^0(x, g) = f'(x, g), \forall x \in \mathbb{R}^n, \forall g \in \mathbb{R}^n$$

An explicit method to calculate the Clarke generalised gradient of a Clarke regular function is available (see [48]). However the class of Clarke regular functions is very restrictive.

Example 9: The following function $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ is not Clarke regular

$$f(x) = |x_1| - |x_2|.$$

Indeed, for $x^0 = (0, 0)$ and $g = (0, 1)$, $f'(x^0, g) = -1$ while $f^0(x^0, g) = 1$.

The Clarke subgradient of this function at x^0 is

$$\partial f(x^0) = \text{co} \{(1, -1), (1, 1), (-1, 1), (-1, -1)\}$$

The major drawback of the Clarke generalised gradient is that for non Clarke regular functions, there exists no explicit calculus.

2.2.3 Demyanov-Rubinov quasidifferential

Let f be a locally Lipschitz continuous function.

Definition 21. The function f is called *quasidifferentiable* at a point x if it is directionally differentiable and there exists compact, convex sets $\underline{\partial}f(x)$ and $\overline{\partial}f(x)$ such that:

$$f'(x, g) = \max_{v \in \underline{\partial}f(x)} \langle v, g \rangle + \min_{v \in \overline{\partial}f(x)} \langle v, g \rangle.$$

The pair $Df(x) = [\underline{\partial}f(x), \overline{\partial}f(x)]$ is called a *quasidifferential* of the function f at x .

Note 4: The quasidifferential is not unique.

Remark 13: For a convex function, the pair $[\partial f(x), \{0\}]$, where $\partial f(x)$ is the Clarke subdifferential at x , is a quasidifferential at x .

Similarly, for a concave function, the pair $[\{0\}, \partial f(x)]$ is a quasidifferential at x .

Proposition 10. *If the point x is a local minimiser of f on \mathbb{R}^n*

$$-\overline{\partial}f(x) \subset \underline{\partial}f(x) \tag{2.5}$$

A point verifying (2.5) is called inf-stationary point

Proposition 11. *If a point x is not inf-stationary, then the direction:*

$$g^0 = -\frac{v^0 + w^0}{\|v^0 + w^0\|},$$

where

$$\|v^0 + w^0\| = \max_{\overline{\partial}f(x)} \min_{\underline{\partial}f(x)} \|v + w\|,$$

is the direction of steepest descent.

The class of quasidifferential functions is very broad. A particular family of quasidifferentiable functions contains the functions with the following form:

$$f(x) = F(x, g_1(x), \dots, g_k(x)), \quad (2.6)$$

where $F(x, y_1, \dots, y_k)$ is continuously differentiable and the functions $g_i, 1 \leq i \leq k$ are quasidifferential (and semismooth). This is particularly useful if the functions g_i can be concave or convex, as this means for instance that *difference of convex (DC)* functions are quasidifferentiable.

$$f(x) = f_1(x) - f_2(x),$$

where f_1 and f_2 are convex, and the pair $[\underline{\partial}f, \bar{\partial}f]$, where

$$\begin{cases} \underline{\partial}f = \partial f_1, \\ \bar{\partial}f = -\partial f_2, \end{cases}$$

is a quasidifferential of this function.

Example 10: The function given in example 9 is quasidifferentiable, as a difference of convex functions:

$$\begin{cases} f_1(x) = |x_1| \\ f_2(x) = |x_2| \end{cases}$$

A quasidifferential of this function at $(0, 0)$ is:

$$\begin{cases} \underline{\partial}f(0, 0) = \text{co} \{(1, -1), (1, 1)\} \\ \bar{\partial}f(0, 0) = \text{co} \{(-1, 1), (-1, -1)\} \end{cases}$$

2.3 Descent methods

The descent methods are certainly the most popular family of optimisation techniques. They are based on a very simple idea: keep going down until it is not possible anymore. In the case of convex functions, these methods reach the global minimum, but in the general case they only converge to a stationary point (which may or may not be a local minimum).

Their popularity is due to their ability to reach good practical solutions in very acceptable time.

Algorithm 2.1 presents a generic descent algorithm.

Algorithm 2.1: A generic descent method

Step 1 select an initial solution x_0, \dots, x_0 , set $k = 0$
 repeat
 Step 2 Find a descent direction d_k
 Step 3 Find a descent step α_k and set $x_{k+1} \leftarrow x_k + \alpha_k d_k$
 Step 4 $k \leftarrow k + 1$
 Step 5 until *Stopping criterion*

2.4 Numerical methods

Interesting problems of optimisation are the ones presenting complicated objective functions. This means that it may be difficult or impossible to calculate the gradient, the subdifferential or a quasidifferential. For this reason, many practical situations require so-called *finite elements* methods.

The idea behind these methods is as follows: instead of being exactly calculated, the desired object is closely approximated, using only values of the objective function.

Mathematically, in order to approximate $h(x)$, where h can be the gradient, the subgradient or a quasidifferential, a family of functions g_λ is defined, so that

$$g_\lambda(x) \xrightarrow{\lambda \rightarrow 0} h(x).$$

2.4.1 Cutting planes method

The cutting planes method was first independently proposed by Cheney and Goldstein in [47] and by Kelley in [108]. It is based on a well known property of convex functions (see [171]): a convex function is the upper envelope of all the linear functions it dominates

Theorem 2. Consider a convex function f defined on \mathbb{R}^n and the set U of linear functions on \mathbb{R}^n . The function f can be defined exactly using the set

$$U_f = \{u \in U : u(x) \leq f(x), \forall x \in \mathbb{R}^n\}$$

as follows:

$$f(x) = \sup \{u(x) : u \in U_f\}, \forall x \in \mathbb{R}^n$$

This property of convex functions gave rise to the following idea: when convex, the objective function can be approximated by a set of linear functions, and the problem is reduced to a linear programming problem. At each iteration of the cutting planes algorithm, the reduced problem is solved, giving a lower approximation of the actual minimum, and the approximating set is adequately updated.

The cutting planes method presents several drawbacks.

- the choice of the initial points is crucial, as the algorithm may not converge.
- the method only works on convex functions, and may not be successful in finding even a stationary point of a nonconvex function
- at each iteration it is required to evaluate a subgradient of the function. Despite the Clarke regularity of convex functions, it may be a difficult task.
- the convergence of the method has been shown to be very slow in some cases (see [197])

Notice that finite elements approximations of the subgradients cannot be applied here, since the piecewise linear approximation of the function may not be dominated by the objective function anymore.

2.4.2 Bundle methods

Bundle methods were first proposed by Lemaréchal in 1978 in [122] and further developed by Kiwiel in [113]. An excellent presentation of this method can be found in [95]. Other good reviews include [135, section 5] and [139] (see also [134, 137, 140]). Numerical experiments are presented for example in [114].

These methods are an improvement of the Cutting planes method, addressing some of its issues. In order to ensure the convergence and to improve the efficiency, a quadratic term is added to the local linear approximation of the curve, in order to keep the search local.

The bundle methods require a very large amount of memory to store the information. In practice, it is not possible to keep all information, and a balance should be found between efficiency and memory. These methods also necessitate the solution of a quadratic subproblem, and the choice of the quadratic approximation is not simple. All these issues are thoroughly discussed in the literature (see [135, 139]).

2.4.3 Discrete gradient method

Notwithstanding the importance of subdifferentials and quasidifferentials in the domain of nonsmooth analysis, these tools are difficult to calculate. Consequently some research has been oriented at constructing approximations (see [60, 61]). The most successful of them is certainly the discrete gradient,

proposed by Bagirov in 1992 in [10]. We describe here these tools and a corresponding nonsmooth optimisation method.

Definition of the discrete gradient

The discrete gradients allow the construction of continuous approximations to the subdifferential and the quasidifferential. They have been studied in [10, 11, 12, 13] as approximations of subdifferentials and in [19, 22] as approximations of quasidifferentials.

Excellent and thorough reviews can be found in [16, 19].

Define operators $H_i^j : \mathbb{R}^n \rightarrow \mathbb{R}^n$ for $1 \leq i \leq n, 0 \leq j \leq n$ by the formula

$$H_i^j g = \begin{cases} (g_1, \dots, g_j, 0, \dots, 0) & \text{if } j < i, \\ (g_1, \dots, g_{i-1}, 0, g_{i+1}, \dots, g_j, 0, \dots, 0) & \text{if } j \geq i. \end{cases} \quad (2.7)$$

That is

$$H_i^j g - H_i^{j-1} g = \begin{cases} (0, \dots, 0, g_j, 0, \dots, 0) & \text{if } 1 \leq j \leq n, j \neq i, \\ 0 & \text{if } j = i. \end{cases} \quad (2.8)$$

Let $e(\beta) = (\beta e_1, \beta^2 e_2, \dots, \beta^n e_n)$, where $\beta \in (0, 1]$. For $x \in \mathbb{R}^n$ we consider the vectors

$$x_i^j \equiv x_i^j(g, e, z, \lambda, \beta) = x + \lambda g - z(\lambda) H_i^j e(\beta), \quad (2.9)$$

where $g \in S_1, e \in G, i \in I(g, \alpha), z \in P, \lambda > 0, 0 \leq j \leq n, j \neq i$.

It follows from (2.8) that

$$x_i^{j-1} - x_i^j = \begin{cases} (0, \dots, 0, z(\lambda) e_j(\beta), 0, \dots, 0) & \text{if } 1 \leq j \leq n, j \neq i, \\ 0 & \text{if } j = i. \end{cases} \quad (2.10)$$

It is clear that $H_i^0 g = 0$ and $x_i^0(g, e, z, \lambda, \beta) = x + \lambda g$ for all $i \in I(g, \alpha)$.

Definition 22. The discrete gradient of the function f at the point $x \in \mathbb{R}^n$ is the vector $\Gamma^i(x, g, e, z, \lambda, \beta) = (\Gamma_1^i, \dots, \Gamma_n^i) \in \mathbb{R}^n, g \in S_1, i \in I(g, \alpha)$, with the following coordinates:

$$\begin{cases} \Gamma_j^i = \frac{f(x_i^{j-1}(g, e, z, \lambda, \beta)) - f(x_i^j(g, e, z, \lambda, \beta))}{z(\lambda) e_j(\beta)}, 1 \leq j \leq n, j \neq i, \\ \Gamma_i^i = \frac{f(x_i^n(g, e, z, \lambda, \beta)) - f(x) - \sum_{j=1, j \neq i}^n \Gamma_j^i (\lambda g_j - z(\lambda) e_j(\beta))}{\lambda g_i}. \end{cases}$$

For any $j \neq i$, the discrete gradient is a typical finite elements evaluation. The coordinate Γ_i^i allows the following equation to be satisfied for any $g \in S_1, e \in G, i \in I(g, \alpha), z \in P, \lambda > 0$ and $\beta > 0$.

$$f(x + \lambda g) - f(x) = \lambda \langle \Gamma^i(x, g, e, z, \lambda, \beta), g \rangle.$$

Approximations of the subdifferentials and the quasidifferentials

The generalised gradient and the quasidifferentials can be approximated by sets of discrete gradients.

Consider a family of sets $C(x, \varepsilon)$ of set valued mappings depending on $\varepsilon > 0$. We consider the following definition (see [199]).

Definition 23. The *limit* of the family $C(x, \varepsilon)$ is defined by:

$$C(x) = \left\{ \lim_{k \rightarrow \infty} v_k : v_k \in C(x^k, \varepsilon_k), x^k \xrightarrow{k \rightarrow \infty} x, \varepsilon_k \xrightarrow{k \rightarrow \infty} 0 \right\}$$

Definition 24. A family $C_f(x, \varepsilon)$ is a *continuous approximation* to the subdifferential $\partial f(x)$ if

- $C_f(x, \varepsilon)$ is Hausdorff continuous with respect to x for all $\varepsilon > 0$.
- the set $\partial f(x)$ is the convex hull of the limit of the family $C_f(x, \varepsilon)$, that is

$$\partial f(x) = \text{co } C_L f(x)$$

Consider the following sets:

$$D_0 f(x, z, \lambda, \beta) = \text{cl co } \{ \Gamma^i(x, g, e, z, \lambda, \beta) : g \in S_1, e \in G, i \in I(g, \alpha) \}.$$

It is possible to find sequences $\lambda_\varepsilon \xrightarrow{\varepsilon \rightarrow 0} 0, \beta_\varepsilon \xrightarrow{\varepsilon \rightarrow 0} 0, z_\varepsilon$, such that the sets $D_0 f(x, z, \lambda, \beta)$ constitute a continuous approximation of the subdifferentials.

For constructing the quasidifferential of the functions of type 2.6, it is possible to use the gradient of the function F and the structures of the functions g_i .

$$\left\{ \begin{array}{l} D_1 f(x, z, \lambda, \beta) = \text{co } \left\{ \frac{\partial F}{\partial x} + \sum_{i \in I_1} \frac{\partial F}{\partial y_i} v^i, v^i \in D_0 y_i(x, z, \lambda, \beta), i \in I_1(x) \right\}, \\ D_2 f(x, z, \lambda, \beta) = \text{co } \left\{ \frac{\partial F}{\partial x} + \sum_{i \in I_2} \frac{\partial F}{\partial y_i} w^i, w^i \in D_0 y_i(x, z, \lambda, \beta), i \in I_2(x) \right\}, \end{array} \right.$$

where

$$\begin{cases} I_1 = \left\{ i \in I : \frac{\partial F}{\partial y_i} > 0 \right\}, \\ I_2 = \left\{ i \in I : \frac{\partial F}{\partial y_i} < 0 \right\}. \end{cases}$$

In particular, if the function is a difference of convex ($f = f_1 - f_2$), then its quasidifferential can be approximated by the pair

$$[D_0 f_1(x, z, \lambda, \beta), -D_0 f_2(x, z, \lambda, \beta)].$$

The method

Descriptions of this method can be found in [14, 15, 18, 21]

The function f is assumed to be semismooth.

The discrete gradient method is a descent method. Therefore, the main step in this method is the search for the descent direction. Two algorithms are given, one based on the subdifferential, the other one based on the quasidifferentials.

Algorithm 2.2 presents the method for finding a descent direction with approximations of the subdifferentials. It is based on two main steps:

- calculation of a discrete gradient with respect to a given direction.
- search for the direction of the next discrete gradient.

At each iteration, the approximation of the subdifferential is extended by adding a new discrete gradient.

It is shown that the algorithm terminates, and either the latest discrete gradient computed is a descent direction, or $0 \in \overline{D}$, where \overline{D} is the approximation of the subdifferential.

If the function f is a difference of convex, then the discrete gradients can be applied for approximating the quasidifferentials. We note $f = f_1 - f_2$, where f_1 and f_2 are convex functions. The method for finding the descent direction relies on the following proposition:

Proposition 12. *Consider two convex and compact sets A and B and the vectors $v^0 \in A$ and $w^0 \in B$ such that:*

$$\|v^0 - w^0\| = \max_{w \in B} \min_{v \in A} \|v - w\| > 0.$$

and for $g^0 = \frac{v^0 - w^0}{\|v^0 - w^0\|}$, $\delta > 0$:

$$\max_{v \in A} \langle v, g^0 \rangle - \max_{w \in B} \langle w, g^0 \rangle \geq -\delta.$$

Algorithm 2.2: DG: finding a descent direction: Clarke subdifferentials version

Let $z \in P, \lambda > 0, \beta \in (0, 1]$, the number $c \in (0, 1)$ and a small enough number $\delta > 0$ be given.

Step 1 choose any $g^1 \in S_1, e \in G, i \in I(g^1, \alpha)$.

Step 2 set $\bar{D}_0(x) \leftarrow \emptyset$ and $k \leftarrow 1$

repeat

Step 3 Calculate a discrete gradient

$$v^k = \Gamma^i(x, g^k, e, z, \lambda, \beta), i \in I(g^k, \alpha),$$

construct the set $\bar{D}_k(x) = \text{co} \{ \bar{D}_{k-1}(x) \cup \{v^k\} \}$

Step 4 Calculate the vector $\|w^k\| = \min \{ \|w\| : w \in \bar{D}_k(x) \}$.

if

$$\|w^k\| \leq \delta, \tag{2.11}$$

then

 // First stopping criterion

 stop // $w \approx 0$ and $w \in \partial f(x)$: x considered stationary

else

Step 5 Calculate the search direction by $g^{k+1} = -\frac{w^k}{\|w^k\|}$.

if

$$f(x + \lambda g^{k+1}) - f(x) \leq -c\lambda \|w^k\|, \tag{2.12}$$

then

 // Second stopping criterion

 stop. // g is a descent direction

until One of the stopping criteria is satisfied

Then

$$B \subset A + S_\delta$$

As a corollary, if $A = \partial f_2(x)$ and B is a continuous approximation of $\partial f_1(x)$, then a sufficient condition for a point x to be inf-stationary of the function f is

$$\|v^0 - w^0\| < \delta$$

Unfortunately this result cannot be generalised to the case when continuous approximations to both $\partial f_2(x)$ and $\partial f_1(x)$ are used. To our knowledge, no known condition exists for a point to be inf-stationary when continuous approximations are used.

Algorithm 2.3 presents the discrete gradient method using the subdifferentials. The version based on quasidifferentials is very similar.

Algorithm 2.3: Discrete gradient method

Let sequences $\delta_k > 0, z_k \in P, \lambda_k > 0, \beta_k \in (0, 1], \delta_k \rightarrow +0, z_k \rightarrow +0, \lambda_k \rightarrow +0, \beta_k \rightarrow +0, k \rightarrow +\infty$ and numbers $c_1 \in (0, 1), c_2 \in (0, c_1]$ be given.

- Step 1 choose any starting point $x^0 \in \mathbb{R}^n$
and set $k = 0$.
- Step 2 set $s = 0$ and $x_s^k = x^k$.
- repeat
- Step 3 apply algorithm 2.2 for the calculation of a descent direction at $x = x_s^k, \delta = \delta_k, z = z_k, \lambda = \lambda_k, \beta = \beta_k, c = c_1$. As a result we get the set $\overline{D}_m(x_s^k)$ and an element v_s^k such that
- $$\|v_s^k\| = \min \{ \|v\| : v \in \overline{D}_m(x_s^k) \}.$$
- Step 4 **case** $\|v_s^k\| \leq \delta_k$
// x_s^k cannot be improved: the step size is reduced
set $x^{k+1} \leftarrow x^k$ and $k \leftarrow k + 1$.
- case** $f(x_s^k + \lambda_k g_s^k) - f(x_s^k) \leq -c_1 \lambda_k \|v_s^k\|$. // g_s^k is a descent direction
- Step 5 let x_{s+1}^k be the result of a line search starting from x_s^k in the direction g_s^k .
- Step 6 set $s \leftarrow s + 1$.
- until $k = k_{max}$
-

Step 4 in algorithm 2.2 require to solve a quadratic optimisation problem, which can be formulated as the minimal distance between a point (the origin) and a polyhedron (the convex hull of all discrete gradients). This specific problem has been studied, and solutions have been proposed for instance by Wolfe in [198] (see also [75]).

The numerical experiments presented in [16] show that the discrete gradient is an efficient method for finding a stationary point. Furthermore, due to the gradual reduction of the search radius of the direction search, this method avoids many non minimal stationary points and even shallow local minima. Finally, its derivative-free nature is practically advantageous over bundle methods.

2.4.4 Other methods

Among the other methods of nonsmooth local optimisation can be mentioned:

- **Nelder-Mead simplex method** ([151]). This method, one of the oldest nonsmooth methods, is based on the update of a simplex according to the values of the function at its end points. It has been shown that it does not always converge to a local solution (see [142, 143]), and several variations have been proposed in [43, 109, 187].
- **Trust region methods.** Trust region methods are proposed as an alternative to descent methods, where a local approximation of the objective function is a simpler function. Various types of local approximations have been proposed (See [53, 62, 74, 80, 91, 123, 141, 148, 149]). The most efficient trust region method for nonsmooth optimisation is Powell's method (see [162, 163, 164, 165, 166]). Here only values of the objective function are used: It is shown in [166] that this method performs well for problems with less than 160 variables.

2.5 Global methods

Objective functions usually have many local minima, some of them sensibly higher than the global one. For such functions, local algorithms may reach a local yet unsatisfactory solution.

Another large family of methods in the field of optimisation is the one of the global methods: these methods guarantee a convergence towards a global solution.

These methods cannot rely on local characteristics of the functions anymore. Most of them are designed to work with particular types of functions, such as Lipschitz continuous functions.

The most popular global optimisation method is the Branch and Bound technique. This technique has been widely studied (see [85, 99, 107, 152, 168]), and is based on the idea of dividing the domain in subdomains, and to find lower bounds to the function over each subdomain. These lower

bounds allow the elimination of some of these subdomains by comparison with existing solutions. Then the algorithm is applied over each remaining subdomain.

Another global optimisation technique is the Cutting angle method. This method is based on the notion of abstract convexity, developed and studied in [156, 172, 174, 119, 120, 180]. The cutting angle method is then a generalisation of the cutting planes method for abstract convexity (see [6, 7, 17, 156, 174, 175, 176])

2.6 Heuristic methods: simulated annealing

A great majority of heuristic methods is based on the evaluation of randomly generated points. For this reason, the solution obtained for two different runs of the same algorithm can be different. It is almost impossible to obtain theoretical results on the convergence of these methods. The main difficulty in the construction of a heuristic method resides in the way random points are generated. The best results are usually obtained when the previous knowledge is used to generate the new points.

As a rule, heuristic methods work better for discrete optimisation, where the generation of new points is easier (the feasible set may even be finite in the case of constraint programming). However, many heuristic methods can also be applied to continuous optimisation. It is usually possible to define families of heuristic methods. Such families are called *metaheuristics*. We present here one of the most popular ones.

The simulated annealing method is based on an analogy with the natural process of crystal cooling: molecules subjected to a slow decrease of temperature reach the form a pure crystal, which corresponds to a state of minimal energy. This process has been modelled in 1953 in [144] using a Monte-Carlo method. The idea of applying an analogy with this model to optimisation was then developed independently by Kirkpatrick in 1983 in [112] and Černý in 1985 in [46]. It has since then gained a great popularity, and has been widely studied (see for example [1, 2, 31, 34, 39, 51, 55, 57, 71, 76, 77, 101, 102, 103, 106, 132, 173, 179, 189, 190, 196]).

Many good introductions to this method have been written, for instance, see [195], and [202, chapter 2] for an exhaustive review of the literature on the subject. For a good introduction to the application of simulated annealing algorithms to continuous optimisation refer to [131].

Algorithm 2.4 presents a general version of the simulated annealing. Several parameters need to be selected. Their choice is crucial for the efficiency of the method.

Algorithm 2.4: Simulated annealing algorithm

Step 1 select an initial point x^0 , set $k \leftarrow 0$.
 Step 2 select an initial temperature T^0 .
 repeat
 Step 3 sample a point x^{k+1} from the distribution $D_k(\cdot)$.
 Step 4 sample a uniform number $p \in [0, 1]$ and set:

$$y^{k+1} = \begin{cases} x^{k+1} & \text{if } p \leq A_k(y^k, x^{k+1}, t_k) \\ y^k & \text{otherwise} \end{cases}$$

 Step 5 set $t_{k+1} = U_k(t_k)$
 Step 6 record the best value met.
 until *Stopping criterion is met*

- A_k is the probability of acceptance of a point. While in the early iterations (for “high temperature”), the tolerance should be quite large, and most points can be accepted, the algorithm should gradually restrict it to finally only accept descent steps.

The most commonly used function is the so called *Metropolis function*.

$$A_k(y^k, x^{k+1}, t_k) = \min \left\{ 1, \frac{e^{f(y^k) - f(x^{k+1})}}{t_k} \right\}.$$

- D_k is the distribution of the points at the iteration k . This is a crucial function. Most continuous algorithms are based on the uniform generation of a random direction $\theta_k \in \mathbb{R}^n$ and of a random step size $s_k \in (0, s_{\max})$, although the step size is sometimes fixed. Constraint programming problems may require difficult to generate feasible points.
- U_k is the rate of evolution of the temperature. It is usually a decreasing function such as $t_{k+1} = \alpha t_k$, where $0 < \alpha < 1$. The rate of descent must be slow enough to ensure a satisfactory convergence, although too slow a rate would compromise the efficiency.
- the stopping criterion can be determined differently. Instances of applicable criteria are:

1. the number of iterations has exceeded a certain limit,
2. the “temperature” is lower than a certain value.

In a general manner, despite the good results it obtains in discrete optimisation, the simulated annealing method meets some difficulties for solving continuous problems.

The simulated annealing method is a heuristic, and therefore there are numerous difficulties in clearly assessing the quality of the solutions it reaches. The convergence of the method has been studied by Locatelli among others (see [127, 128, 129, 130] and references therein).

2.7 Hybrid methods

Hybrid methods are the most recent family of optimisation method. Their apparition owes to the fact that current global methods are not applicable using current technology for solving real world large problems

Often in practical situations local methods are the only alternative. In order to improve the chances of getting a good solution, these methods are usually run several times starting from several initial points (either randomly chosen, or otherwise deduced from analytical considerations). Unfortunately it is seldom possible to ensure the quality of the solution thus obtained.

On the other hand, global and even heuristic methods are too time and resource consuming to be reasonable options, even though they may guarantee to reach at least a good practical solution.

An ideal method should combine the economical efficiency of a local method with the qualitative efficiency of a global (or heuristic) one. For this purpose hybrid methods have been designed.

A number of combinations have been proposed, most of which can fall in one of the following categories:

1. the global search is used to *escape* from the results reached by the local method.
2. the global search is inserted inside the local search in order to *improve* the search properties of the local method.
3. the global method is applied as a preprocessor, in order to generate a set of starting points for the local method.

A method of the latter category has been applied for instance in [125]. The following subsections bring more light over the other two categories.

2.7.1 Global method as an escape method

This type of hybrid method is very simple. Its core idea lies on the fact that a global minimum is also necessarily a local one. A descending sequence of local minima is constructed by applying at each iteration the local method, starting from a point obtained by the global one.

Algorithm 2.5: Hybrid method of type 1

Step 1 select an initial point x_0 . Set $k \leftarrow 0$
repeat
Step 2 set $k \leftarrow k + 1$.
Step 3 find a local minimum \bar{x}_k by applying the local method starting from x_{k-1} .
Step 4 apply the global method on the problem

$$\begin{aligned} & \text{minimise } f_k(x) = \min\{f(x), f(\bar{x}_k)\} \\ & \text{subject to} \\ & \quad x \in \mathbb{R}^n. \end{aligned}$$

until a point $x_k \in \mathbb{R}^n$ such that $f(x_k) < f(\bar{x}_k)$ is reached, or the stopping criterion of the global method is satisfied.
until $f(x_k) = f(\bar{x}_k)$

This hybrid method has been implemented for instance in [23, 28, 201]. Algorithm 2.5 shows the main steps of this type of methods.

Its great advantage is that it uses the convergence properties of the global method. If this method guarantees the convergence to a global minimum, then it is easy to see that algorithm 2.5 will stop when the function $f_k(x)$ is flat. For this reason, such methods work very well on small problems.

However this method may become very time consuming, as the global method is applied on a problem of the same size as the original one. At least on one occasion (when the function f_k is flat) it is necessary to run the global method until its stopping criterion is met. This means that this hybrid method may not be suitable for very large problems.

2.7.2 Global method as an improvement of the local search

This type of hybrid method is usually more complex to implement, as they are inserted in the local algorithm. For this reason they often are method-dependent. Such hybrid methods based on the simplex algorithm can be found in [92, 93, 121, 167, 200].

Such a hybridisation based on descent methods has been developed in [26, 27]. This method relies on a multidirectional search rather than the usual line search. Algorithm 2.6 on the following page presents this hybrid method.

The step lengths specified in step 5 are usually computed using the bounds of the feasible set. The size of the problem (2.13) is exactly the cardinality of the set X_k . This size can be as small as necessary (in practice 2 or 3).

Although no guarantee of reaching the global solution can be given, this

Algorithm 2.6: Multidirectional search descent algorithm

Step 1 select an initial point x_0 and set $k \leftarrow 0$.
 repeat
 Step 2 find a descent direction d_k at point x_k as well as a set of other directions D_k .
 Step 3 set $k \leftarrow k + 1$
 Step 4 carry out a line search, reaching \bar{x}_k
 Step 5 For each direction $d \in D$, find a maximum step length s_d and set $x_d = x_k + s_d$
 Step 6 let S_k be the convex hull of the points $X_k\{\bar{x}_k, x_d, d \in D\}$.

$$S_k = \left\{ y : y = \sum_{x \in X_k} \alpha_x x, \sum_{x \in X_k} \alpha_x = 1 \right\}$$

Step 7 use the global method to solve the problem:

$$\begin{aligned} & \text{minimise } f(y) \\ & \text{Subject to} \\ & y \in S_k \end{aligned} \tag{2.13}$$

Step 8 let x_k be the solution of this problem
 until the stopping criterion of the local method is met

algorithm works well in practice.

2.8 Large scale optimisation

Many real world problems involve a very large number of parameters to be selected. Most of these problems can be rewritten as mathematical programming problems. Unfortunately the state of the art optimisation algorithms are not applicable to such kinds of problems.

On the other hand, the objective functions of such problems present a particular structure. For instance this is extensively used in linear programming softwares like CPLEX ([100]), where the size of the problem is reduced by an aggressive preprocessing.

Nonlinear problems are much more complex, and an automatic analysis of the function and the constraints seems no longer possible. Nevertheless a few studies have been carried out to deal with the structures of the large dimensional problems.

The basis of most ideas in this domain is the fact that when the number of parameters is very large, the proportion of pairs of parameters that influence one another is small. In the smooth case, this translates to the Hessian of

the objective function being a sparse matrix. It was studied for example in [81, 186]. Another idea, proposed for example in [41, 44, 45, 126, 153, 154], is based on the construction of approximations of the Hessian of the function using previous knowledge.

Much less has been done for nonsmooth functions. To the best of our knowledge, only variants of the latter method has been adapted (see [83]). This is particularly necessary for bundle methods, in order to reduce the memory overflow specific to these methods.

2.9 Choice of the method utilised

The problems arising in data mining require a method able to solve high-dimensional problems. Therefore out of the box global methods and heuristic methods are not suitable for our research. On the other hand, because these problems are nonconvex, it is necessary to select a method dealing efficiently with nonconvex problems.

Functions in data classification and clustering are seldom differentiable. Smooth methods of optimisation are not a good choice for such kind of problems, as their behaviour may be quite unpredictable.

The efficiency of the discrete gradient method to leave a shallow local minimum has been shown by numerical experiments in [16, 17]. Moreover, it has a better capacity to solve nonsmooth high dimensional problems than Powell's or Nelder-Mead's methods (see [29] for comparisons), while the non regularity of the objective functions disqualifies the cutting planes and bundle Methods as good alternatives. Therefore this method will be used in the rest of this thesis.

The version of the discrete gradient method selected is based on Clarke subdifferentials rather than quasidifferentials. Although quasidifferentials constitute a much better local approximation of the function, and many objective functions in data analysis can be rewritten as difference of convex, the stopping criterion for quasidifferentials ($\bar{\partial}f(x) \subset \underline{\partial}f(x)$) is rather difficult to implement numerically. On the other hand the stopping criterion for the Clarke generalised gradient ($0 \in \partial f(x)$) is much simpler to evaluate with the incrementally constructed discrete gradient set (see algorithm 2.2 on page 59). We believe that little change will be necessary to apply the work presented in this thesis to the quasidifferentials based discrete gradient method.

Because the discrete gradient method is a local solver, it is necessary to assess the results it reaches. For comparison of their quality, the hybrid methods described in algorithms 2.5 based on the DG and on simulated an-

nealing and 2.6 based on DG and cutting angle method will also be applied in some occasions. The latter method as well as DG are part of the commercial software *CIAO-GO* (see [188]). Finally come comparisons will also be carried out with the commercial software *LGO* (see [158]), based on the branch and bound method.

Part II

Computational considerations

Chapter 3

Solving large scale nonsmooth optimisation

3.1 Introduction

Supervised and unsupervised learning problem formulations usually present a recurrent difficulty: the size of the problems depend on the number of features in the dataset. Additionally, since the goal of data analysis is to extract information from the datasets, restricting the features is not always advisable. This leads to large scale problems of nonsmooth nonconvex optimisation.

As a rule, large scale problems present a sparse structure: Most parameters are pairwise independent. Furthermore, the parameters can usually be separated in different independent groups of parameters. We will see in this thesis that very few, if any, data analysis problems derogate from that rule.

In case the objective function is smooth, this translates to the Hessian being a sparse matrix. In such a case, matrix calculus can be applied (see section 2.8 for references).

In the general case, though, the Hessian no longer exists, and it is necessary to explore new directions for developing adapted algorithms. In this chapter we will study large scale optimisation problems. We will introduce the class of piecewise partially separable functions, and present an algorithm for their minimisation. This method is based on the discrete gradient method (see section 2.4.3). Results of numerical experiments on test functions will be presented.

3.2 Piecewise partially separable functions: definition and examples

Let f be a scalar function defined on an open set $D_0 \subseteq \mathbb{R}^n$ containing a closed set $D \subseteq \mathbb{R}^n$.

Definition 25. The function f is called partially separable if there exists a family of $n \times n$ diagonal matrices $U_i, 1 \leq i \leq M$ such that the function f can be represented as follows:

$$f(x) = \sum_{i=1}^M f_i(U_i x).$$

Without loss of generality we assume that the matrices U_i are binary, that is they contain only 0 and 1. It is also assumed that the number m_i of non-zero elements in the diagonal of the matrix U_i is much smaller than n .

In other terms, the function f is called partially separable if it can be represented as the sum of functions of a much smaller number of variables. If $M = n$ and $\text{diag}(U_i) = e_i$, then the function f is separable.

Remark 14: Any function f can be considered as partially separable if we take $M = 1$ and $U_1 = I$, where I is the identity matrix. However, we consider situations where $M > 1$ and $m_i \ll n, 1 \leq i \leq M$.

Example 11: Consider the following function

$$f(x) = \sum_{i=1}^n \min\{|x_i|, |x_1|\}.$$

This function is partially separable. Indeed, in this case $M = n, m_i = 2, U_i^{11} = 1, U_i^{ii} = 1$, all other elements of U_i are zeroes for all $1 \leq i \leq n$ and $f_i(U_i x) = \min\{|x_i|, |x_1|\}$.

Definition 26. The function f is said to be *piecewise partially separable* (ppsf) if there exists a finite family of closed sets D_1, \dots, D_m such that $\bigcup_{i=1}^m D_i = D$ and the function f is partially separable on each set $D_i, 1 \leq i \leq m$.

Example 12: All partially separable functions are piecewise partially separable.

Example 13: Consider the following function

$$f(x) = \max_{1 \leq j \leq n} \sum_{i=1}^n |x_i - x_j|.$$

The function f is piecewise partially separable. It is clear that the functions

$$\varphi_j(x) = \sum_{i=1}^n |x_i - x_j|, 1 \leq j \leq n$$

are partially separable with $M = n, m_i = 2$ and $U_i^{ii} = U_i^{jj} = 1$ for all $1 \leq i \leq n$. In this case the sets $D_i, 1 \leq i \leq n$ are defined as follows:

$$D_i = \{x \in \mathbb{R}^n : \varphi_i(x) \geq \varphi_j(x), 1 \leq j \leq n, j \neq i\}.$$

The piecewise partial separability of the function f follows from the fact that the maximum of partially separable functions is piecewise partially separable, which will be proved later on in proposition 17.

3.2.1 Chained and piecewise chained functions

One of the interesting and important classes of partially separable functions is the one of the so-called chained functions.

Definition 27. The function f is said to be k -chained, $k \leq n$, if it can be represented as follows:

$$f(x) = \sum_{i=1}^{n-k+1} f_i(x_i, \dots, x_{i+k-1}),$$

where $f_i : \mathbb{R}^k \rightarrow \mathbb{R}$.

For example, if $k = 2$, the function f is:

$$f(x) = \sum_{i=1}^{n-1} f_i(x_i, x_{i+1}),$$

where $f_i : \mathbb{R}^2 \rightarrow \mathbb{R}$.

Remark 15: Any k -chained function is partially separable. Indeed for k -chained functions $M = n - k + 1, m_i = k$ and the matrices $U_i, 1 \leq i \leq M$ are defined as follows:

$$U_i^{jj} = 1, i \leq j \leq i + k - 1$$

and all other elements of U_i are zeros.

Proposition 13. *Any separable function is 1-chained.*

Definition 28. The function f is said to be piecewise k -chained if there exists a finite family of closed sets D_1, \dots, D_m such that $\bigcup_{i=1}^m D_i = D$ and the function f is k -chained on each set $D_i, 1 \leq i \leq m$.

Remark 16: Any piecewise k -chained function is piecewise partially separable. This directly follows from remark 15.

The following is an example of piecewise 2-chained function.

Example 14 (Chained Crescent I function ([140]):

$$f(x) = \max \{f_1(x), f_2(x)\}$$

where

$$f_1(x) = \sum_{i=1}^{n-1} (x_i^2 + (x_{i+1} - 1)^2 + x_{i+1} - 1),$$

$$f_2(x) = \sum_{i=1}^{n-1} (-x_i^2 - (x_{i+1} - 1)^2 + x_{i+1} + 1).$$

Both f_1 and f_2 are 2-chained functions. We define two sets as follows:

$$D_1 = \{x \in \mathbb{R}^n : f_1(x) \geq f_2(x)\},$$

$$D_2 = \{x \in \mathbb{R}^n : f_2(x) \geq f_1(x)\}.$$

It is clear that the sets D_1, D_2 are closed, $f(x) = f_1(x)$ for $x \in D_1$ and $f(x) = f_2(x)$ for $x \in D_2$. Furthermore $D_1 \cup D_2 = D$. Thus the function f is piecewise 2-chained.

3.2.2 Piecewise separable functions

Definition 29. The function f is said to be piecewise separable if there exists a finite family of closed sets D_1, \dots, D_m such that $\bigcup_{i=1}^m D_i = D$ and the function f is separable on each set $D_i, 1 \leq i \leq m$.

Proposition 14. *Any piecewise separable function is piecewise 1-chained.*

Proof. Since any separable function is 1-chained (proposition 13) the proof is straightforward. \square

Corollary 2. *Any piecewise separable function is piecewise partially separable.*

Proposition 15. All separable functions are piecewise separable. In this case $m = 1$.

Example 15: All piecewise linear functions are piecewise separable. A function $f : D \rightarrow \mathbb{R}^1$ is said to be piecewise linear if there exists a finite family of closed sets Q_1, \dots, Q_p such that $\bigcup_{i=1}^p Q_i = D$ and the function f is linear on each set $Q_i, 1 \leq i \leq p$. Since any linear function is separable the function f is piecewise separable and in this case $m = p$.

Example 16: One of the simplest piecewise separable functions is the following maximum function:

$$f(x) = \max_{1 \leq i \leq n} x_i^2.$$

Here $m = n$ and

$$D_i = \{x \in \mathbb{R}^n : x_i^2 \geq x_j^2, 1 \leq j \leq n, j \neq i\}.$$

$f(x) = x_i^2$ for any $x \in D_i$. It is clear that $\bigcup_{i=1}^n D_i = \mathbb{R}^n$. It should be noted that the function f is neither separable nor piecewise linear.

3.3 Properties of piecewise partially separable functions

In this section we study some properties of piecewise partially separable functions.

Proposition 16. Let f_1 and f_2 be partially separable functions defined on the closed set $D \subset \mathbb{R}^n$. Then the function $f(x) = f_1(x) + f_2(x)$ is also partially separable on D .

Proof. Since the functions f_1 and f_2 are partially separable there exist families of matrices $U_i^1, 1 \leq i \leq M_1$ and $U_j^2, 1 \leq j \leq M_2$ such that

$$f_1(x) = \sum_{i=1}^{M_1} f_{1i}(U_i^1 x),$$

$$f_2(x) = \sum_{j=1}^{M_2} f_{2j}(U_j^2 x).$$

Consider the following sets:

$$\begin{aligned} I &= \{i \in \{1, \dots, M_1\} : U_i^1 \neq U_j^2, \forall j \in \{1, \dots, M_2\}\}, \\ J &= \{j \in \{1, \dots, M_2\} : U_j^2 \neq U_i^1, \forall i \in \{1, \dots, M_1\}\}, \end{aligned} \quad (3.1)$$

$$H = \{(i, j), i \in \{1, \dots, M_1\}, j \in \{1, \dots, M_2\} : U_i^1 = U_j^2\}.$$

It is clear that for any $i \in I$ there is no $j \in \{1, \dots, M_2\}$ such that $(i, j) \in H$ and similarly for any $j \in J$ there is no $i \in \{1, \dots, M_1\}$ such that $(i, j) \in H$. Then the function f can be represented as follows

$$f(x) = \sum_{(i,j) \in H} (f_{1i}(U_i^1 x) + f_{2j}(U_j^2 x)) + \sum_{i \in I} f_{1i}(U_i^1 x) + \sum_{j \in J} f_{2j}(U_j^2 x).$$

This function is partially separable: that is

$$f(x) = \sum_{k=1}^M \bar{f}_k(V_k x),$$

where $M = M_1 + M_2 - \text{card}(H)$, the matrices $V_k, 1 \leq k \leq M$ can be defined as follows:

We define three sets of indices:

$$\begin{aligned} K_1 &= \{1 \leq k \leq \text{card}(H) && (i, j) \in H \}; \\ K_2 &= \{\text{card}(H) + 1 \leq k \leq M_1 && i \in I \}; \\ K_3 &= \{M_1 + 1 \leq k \leq M_1 + M_2 - \text{card}(H) && j \in J \}, \end{aligned}$$

and let

$$V_k = \begin{cases} U_i^1 = U_j^2 & k \in K_1; \\ U_i^1 & k \in K_2; \\ U_j^2 & k \in K_3. \end{cases}$$

and

$$\bar{f}_k(V_k x) = \begin{cases} f_{1i}(U_i^1 x) + f_{2j}(U_j^2 x) & k \in K_1; \\ f_{1i}(U_i^1 x) & k \in K_2; \\ f_{2j}(U_j^2 x) & k \in K_3. \end{cases}$$

Here $\text{card}(H)$ stands for the cardinality of the set H .

□

We say that two partially separable functions f_1 and f_2 have the same structure if $I = J = \emptyset$, where I and J are defined by equation (3.1) on the preceding page. These functions are more interesting from a practical point of view. In this case the function f has the same structure as f_1 and f_2 and

$$f(x) = \sum_{(i,j) \in H} (f_{1i}(U_i^1 x) + f_{2j}(U_j^2 x)).$$

For example, if f_1 and f_2 are k -chained then the function f is also k -chained.

Proposition 17. *If f and g are piecewise partially separable (piecewise k -chained, piecewise separable) continuous functions defined on the closed set D , then*

1. $h(x) = \alpha f(x), \alpha \in \mathbb{R}^1$ is piecewise partially separable (piecewise k -chained, piecewise separable);
2. $h(x) = f(x) + g(x)$ is piecewise partially separable (piecewise k -chained, piecewise separable);
3. $h(x) = \max(f(x), g(x)), h(x) = \min(f(x), g(x))$ and $h(x) = |f(x)|$ are piecewise partially separable (piecewise k -chained, piecewise separable).

Proof. 1. The proof is straightforward.

2. Since the functions f and g are piecewise partially separable there exist families of closed sets

$$D_i^f, 1 \leq i \leq m_1, \bigcup_{i=1}^{m_1} D_i^f = D$$

and

$$D_j^g, 1 \leq j \leq m_2, \bigcup_{j=1}^{m_2} D_j^g = D$$

such that the function f is partially separable on the sets D_i^f and the function g is partially separable on the sets D_j^g . We define a family of sets $Q_{ij}, 1 \leq i \leq m_1, 1 \leq j \leq m_2$ where

$$Q_{ij} = D_i^f \cap D_j^g.$$

It is clear that

$$\bigcup_{i,j} Q_{ij} = D$$

and the sets Q_{ij} are closed. Since the sum of partially separable functions is partially separable we get that $f + g$ is partially separable on each set Q_{ij} .

The proof for piecewise k -chained and piecewise separable functions is similar.

3. Consider the following two sets:

$$P_1 = \{x \in D : f(x) \geq g(x)\}, P_2 = \{x \in D : g(x) \geq f(x)\}.$$

It is clear that $P_1 \cup P_2 = D$. Since the functions f and g are continuous the sets P_1 and P_2 are closed. We define the following families of sets:

$$Q_i^1 = P_1 \cap D_i^f, 1 \leq i \leq m_1, Q_j^2 = P_2 \cap D_j^g, 1 \leq j \leq m_2.$$

These sets are closed. It can be easily shown that

$$\left(\bigcup_i^{m_1} Q_i^1 \right) \cup \left(\bigcup_j^{m_2} Q_j^2 \right) = D.$$

$h(x) = f(x), x \in Q_i^1, 1 \leq i \leq m_1$ and f is partially separable on each set Q_i^1 . Similarly $h(x) = g(x), x \in Q_j^2, 1 \leq j \leq m_2$ and g is partially separable on each set Q_j^2 . Then we get that the function h is piecewise partially separable.

Since $h(x) = \min(f(x), g(x)) = -\max(-f(x), -g(x))$ then we get that h is piecewise partially separable. $h(x) = |f(x)| = \max(f(x), -f(x))$ and both f and $-f$ are piecewise partially separable it follows that the function h is also piecewise partially separable.

Again the proof for piecewise k -chained and piecewise separable functions is similar. □

The problem of computation of Hessians of twice continuously differentiable partially separable functions was discussed by many authors (see, for example, [3, 52] and section 2.8 on page 66 for more references).

Now let us assume that the function f is partially separable and the functions $f_i, 1 \leq i \leq M$ are directionally differentiable. Then the function f is also directionally differentiable and

$$f'(x, g) = \sum_{i=1}^M f'_i(U_i x, U_i g). \quad (3.2)$$

It follows from this formula that if f separable then

$$f'(x, g) = \sum_{i=1}^n f'_i(x_i, g_i) \quad (3.3)$$

where

$$f'_i(x_i, g_i) = \begin{cases} f'_{i+}(x_i) & \text{if } g_i > 0, \\ 0 & \text{if } g_i = 0, \\ -f'_{i-}(x_i) & \text{if } g_i < 0. \end{cases}$$

and $f'_{i+}(x_i), f'_{i-}(x_i)$ are the right and left side derivatives of the function f_i at the point x_i .

Below we study the Lipschitz continuity and directional differentiability of piecewise partially separable functions.

Let f be a piecewise partially separable function defined on the closed convex set $D \subset \mathbb{R}^n$, that is there exists a family of closed sets $D_j, 1 \leq j \leq m$ such that $\bigcup_{j=1}^m D_j = D$, $f(x) = f_j(x), x \in D_j$ and the functions f_j are partially separable on D_j .

Proposition 18. *Let f be continuous and each function f_j be locally Lipschitz continuous on $D_j, 1 \leq j \leq m$. Then the function f is locally Lipschitz continuous on D .*

Proof. We take any bounded subset $\bar{D} \subset D$. Then there exists a subset of indices $\{j_1, \dots, j_p\} \subset \{1, \dots, m\}$ such that

$$\text{co } \bar{D} \cap D_{j_k} \neq \emptyset, 1 \leq k \leq p.$$

Let $L_{j_k} > 0$ be a Lipschitz constant of the function f_{j_k} on the set $\text{co } \bar{D} \cap D_{j_k}, 1 \leq k \leq p$. Let

$$L_0 = \max_{1 \leq k \leq p} L_{j_k}.$$

Now we take any two points $x, y \in \bar{D}$. Then there exist indices $j_{k_1}, j_{k_2} \in \{j_1, \dots, j_p\}$ such that $x \in D_{j_{k_1}}$ and $y \in D_{j_{k_2}}$. If $k_1 = k_2 = k$ then it is clear that

$$|f(x) - f(y)| = |f_k(x) - f_k(y)| \leq L_k \|x - y\| \leq L_0 \|x - y\|.$$

Otherwise we consider the segment $[x, y] = \alpha x + (1 - \alpha)y, \alpha \in [0, 1]$ joining these two points and define the following set:

$$Z_{[x,y]} = \left\{ z \in [x, y] : \exists l_1, l_2 \in \{1, \dots, p\} : z \in D_{j_{l_1}} \cap D_{j_{l_2}} \right\}.$$

It is clear that in this case the set $Z_{[x,y]}$ is not empty. Then there exists a sequence of points $\{z_1, \dots, z_N\} \subset Z_{[x,y]}, N \leq p$ such that

- $\{x, z_1\} \subset D_{j_{k_1}}$;
- $\{z_N, y\} \subset D_{j_{k_2}}$;
- $\forall i \in \{1, \dots, N-1\}, \exists l_i \in \{1, \dots, p\} : \{z_i, z_{i+1}\} \subset D_{j_{l_i}}$.

Then taking into account the continuity of the function f we have:

$$\begin{aligned}
 |f(y) - f(x)| &= \left| f(y) + \sum_{i=1}^{N-1} (f(z_i) - f(z_i)) - f(x) \right| \\
 &= \left| f_{j_{k_2}}(y) + \sum_{i=1}^{N-1} (f_{j_i}(z_i) - f_{j_{i+1}}(z_i)) - f_{j_{k_1}}(x) \right| \\
 &\leq |f_{j_{k_2}}(y) - f_{j_{k_2}}(z_N)| + \sum_{i=1}^{N-1} |f_{j_i}(z_i) - f_{j_i}(z_{i+1})| \\
 &\quad + |f_{j_{k_1}}(z_1) - f_{j_{k_1}}(x)| \\
 &\leq L_{j_1} \|y - z_N\| + \sum_{i=1}^{N-1} L_{j_i} \|z_i - z_{i+1}\| + L_{j_{k_1}} \|z_1 - x\| \\
 &\leq L_0 \left(\|y - z_N\| + \sum_{i=1}^{N-1} \|z_i - z_{i+1}\| + \|z_1 - x\| \right).
 \end{aligned}$$

Then, as all z_i are aligned on the segment $[x, y]$, we get

$$|f(y) - f(x)| \leq L_0 \|y - x\|.$$

Since points x and y are arbitrary it follows that the function f is locally Lipschitz continuous. \square

Corollary 3. *Assume that all conditions of proposition 18 are satisfied. Then the function f is Clarke subdifferentiable.*

Proposition 19. *Assume that for any two points $x, y \in D$ the set $Z_{[x,y]}$ is finite and all functions $f_j, 1 \leq j \leq m$ are directionally differentiable. Then the function f is also directionally differentiable.*

Proof. We take any point $x \in D$ and any direction $g \neq 0$ such that $x + \alpha g \in D, \alpha \in [0, \bar{\alpha}]$ for some $\bar{\alpha} > 0$. By the definition

$$f'(x, g) = \lim_{\alpha \rightarrow +0} \frac{f(x + \alpha g) - f(x)}{\alpha}.$$

Assume that $x \in \bigcap_{k \in K} D_k$, where $K \subset \{1, \dots, m\}$. Let $y = x + \bar{\alpha}g \in D$. Since the set $Z_{[x,y]}$ is finite there exists a finite sequence of numbers $\alpha_1, \dots, \alpha_l$ such that $\alpha_i \in (0, \bar{\alpha})$ and $x + \alpha_j g \in D_{k_j} \cap D_{k_{j+1}}, 1 \leq j \leq l$ and

- $[x, x + \alpha_1 g] \subset D_{k_1}, k_1 \in K;$

- $[x + \alpha_l g, y] \subset D_{k_{l+1}}$;
- $\forall i \in \{1, \dots, l-1\} : [x + \alpha_i g, x + \alpha_{i+1} g] \subset D_{k_{i+1}}$.

This implies that the segment $[x, x + \alpha_1 g] \subset D_{k_1}$. Thus

$$f'(x, g) = f'_{k_1}(x, g).$$

It follows that if the function f is piecewise partially separable then its directional derivative can be calculated using (3.2) and if this function is piecewise separable then its directional derivative is calculated using (3.3). \square

In general piecewise partially separable functions are not regular. The following example demonstrates it.

Example 17: Consider the function

$$f(x_1, x_2) = ||x_1| - |x_2||, (x_1, x_2) \in \mathbb{R}^2.$$

This function is piecewise separable. However it is not regular. Indeed, for the direction $g = (1, 1)$ at the point $x = (0, 1)$ we have

$$f'(x, g) = 0 \text{ and } f^0(x, g) = 2,$$

that is $f'(x, g) < f^0(x, g)$.

This example shows that in general for the subdifferential of piecewise partially separable functions a full calculus does not exist. Therefore in many cases the computation of their subgradients is a quite difficult task.

3.4 Minimisation of piecewise partially separable functions

In this section we will develop an algorithm for minimising one class of piecewise partially separable functions.

We will consider the following unconstrained minimisation problem

$$\text{minimise } f(x) \text{ subject to } x \in \mathbb{R}^n \quad (3.4)$$

where the objective function f is as follows

$$f(x) = \sum_{i=1}^M \max_{j \in J_i} \min_{k \in K_j} f_{ijk}(x) \quad (3.5)$$

and functions f_{ijk} , $1 \leq i \leq M$, $j \in J_i$, $k \in K_j$ are partially separable, that is there exists a family of $n \times n$ matrices U_{ijkt} , $1 \leq t \leq M_{ijk}$ such that

$$f_{ijk}(x) = \sum_{t=1}^{M_{ijk}} f_{ijk}^t(U_{ijkt}x).$$

The function f is piecewise partially separable. If all functions f_{ijk} are l -chained (separable) then the function f is piecewise l -chained (piecewise separable).

Particular cases of this function are the following:

1. The case when the sets J_i , $1 \leq i \leq M$ are singletons

$$f(x) = \sum_{i=1}^M \min_{k \in K_i} f_{ik}(x). \quad (3.6)$$

The clustering function (1.4) serves as an example for this type of functions when $K_i = K$, $\forall i \in \{1, \dots, M\}$ and the functions f_{ik} are separable.

2. The case when $M = 1$

$$f(x) = \max_{i \in I} \min_{j \in J_i} f_{ij}(x). \quad (3.7)$$

As we can see from example 17 even for very simple cases this type of functions may not be regular and therefore sometimes the computation of their subgradients is quite difficult. Therefore, methods based on function evaluations only seem better alternatives to solve problem (3.4).

We will develop a new modified version of the discrete gradient method (see section 2.4.3 on page 55) for solving problem (3.4). Numerical experiments have shown that the first two steps (steps 3 and 4 of algorithm 2.2 on page 59) take most of the CPU time used by the method. We will introduce a new scheme for the calculation of discrete gradients of piecewise partially separable functions represented as a sum of max-min functions. To calculate the discrete gradients we use only values of the objective function. Since the calculation of the objective function in the problem (3.4) can be expensive, such a scheme will allow one to significantly reduce the number of objective function evaluations.

3.4.1 Remarks on the discrete gradient

The structure of the discrete gradient method, presented in section 2.4.3, is suitable for being adapted to minimise piecewise partially separable functions. The following preliminary remarks on the discrete gradient will be used in this adaptation.

Remark 17: It follows from definition 22 on page 56 that for the calculation of the discrete gradient $\Gamma^i(x, g, e, z, \lambda, \beta), i \in I(g, \alpha)$ we define a sequence of points

$$x_i^0, \dots, x_i^{i-1}, x_i^{i+1}, \dots, x_i^n.$$

For the calculation of the discrete gradient it is sufficient to evaluate the function f at each point of this sequence.

Remark 18: The discrete gradient is defined with respect to a given direction $g \in S_1$. We can see that for the calculation of one discrete gradient we have to calculate $(n + 1)$ values of the function f : at the point x and at the points $x_i^j(g, e, z, \lambda, \beta), 0 \leq j \leq n, j \neq i$. For the calculation of the next discrete gradient at the same point with respect to any other direction $g^1 \in S_1$ we have to calculate this function n times, because we have already calculated φ at the point x .

Remark 19: One can see from (2.10) on page 56 that two successive points of the sequence

$$x_i^0, \dots, x_i^{i-1}, x_i^{i+1}, \dots, x_i^n$$

differ by one coordinate only. More precisely, the point x^k can be obtained from the point x^{k-1} by changing only the k -th coordinate.

3.4.2 Calculation of the discrete gradients of the sum of max-min functions

We take any point $x \in \mathbb{R}^n$ and any direction $g \in S_1$. Remark 17 implies that for the calculation of the discrete gradient of f at x with respect to the direction g first we have to define the sequence

$$x_i^0, \dots, x_i^{i-1}, x_i^{i+1}, \dots, x_i^n.$$

It follows from remark 19 that each new point x^p differs from x^{p-1} by one coordinate only. In order to calculate the discrete gradient we have to evaluate the function f at all these points.

The functions f_{ijk} are partially separable and they can be represented as

$$f_{ijk}(x) = \sum_{t=1}^{M_{ijk}} f_{ijk}^t(U_{ijkt}x).$$

We will call f_{ijk}^t *core functions*. The total number of these functions is

$$N_0 = \sum_{i=1}^M \sum_{j \in J_i} \sum_{k \in K_j} M_{ijk}.$$

For one evaluation of the function f we have to compute these functions N_0 times. Since for one evaluation of the discrete gradient we compute the function f $n + 1$ times the total number of computations of core functions for one evaluation of the discrete gradient is

$$N_t = (n + 1)N_0.$$

For $p \in \{1, \dots, n\}$ we introduce

$$\begin{aligned} Q_p^{ijk} &= \{t \in \{1, \dots, M_{ijk}\} : U_{ijkt}^{pp} = 1\}, \\ \bar{Q}_p^{ijk} &= \{t \in \{1, \dots, M_{ijk}\} : U_{ijkt}^{pp} = 0\}. \end{aligned}$$

It is clear that $M_{ijk} = \text{card}(Q_p^{ijk}) + \text{card}(\bar{Q}_p^{ijk})$. One can assume that $\text{card}(Q_p^{ijk}) \ll \text{card}(\bar{Q}_p^{ijk})$. For example, if all functions f_{ijk} are l -chained then

$$\text{card}(Q_p^{ijk}) \leq l \text{ and } \text{card}(\bar{Q}_p^{ijk}) \geq n - l - 1.$$

If these functions are separable then

$$\text{card}(Q_p^{ijk}) = 1 \text{ and } \text{card}(\bar{Q}_p^{ijk}) = n - 1.$$

Then the function f_{ijk} can be calculated at the point x^p as follows:

$$f_{ijk}(x^p) = \sum_{t \in Q_p^{ijk}} f_{ijk}^t(U_{ijkt}x^p) + \sum_{t \in \bar{Q}_p^{ijk}} f_{ijk}^t(U_{ijkt}x^{p-1}) \quad (3.8)$$

that is we compute only functions $f_{ijk}^t, t \in Q_p^{ijk}$ at the point x^p and all other functions remain the same as at the point x^{p-1} . Thus in order to calculate the function f at the point x^p we compute

$$N_s = \sum_{i=1}^M \sum_{j \in J_i} \sum_{k \in K_j} \text{card}(Q_p^{ijk})$$

times the core functions at this point. Since $\text{card}(Q_p^{ijk}) \ll M_{ijk}$ one can expect that $N_s \ll N_0$.

If all functions $f_{ijk}, 1 \leq i \leq M, j \in J_i, k \in K_j$ are l -chained then

$$N_s = l \sum_{i=1}^M \sum_{j \in J_i} \text{card}(K_j).$$

If all these functions are separable then

$$N_s = \sum_{i=1}^M \sum_{j \in J_i} \text{card}(K_j).$$

Thus in order to compute one discrete gradient at the point x with respect to the direction $g \in S_1$ we have to compute the function f at the points x and $x + \lambda g$ using formula (3.5) and at all other points $x_i^p, 1 \leq p \leq n, p \neq i$ it can be computed using the simplified scheme (3.8). In this case the total number of computations of core functions is

$$N_{ts} = 2N_0 + (n - 1)N_s$$

which is significantly less than N_t when n is large.

Now we consider one special case of functions (3.5).

Functions represented as a sum of minimum functions

We consider the following functions:

$$f(x) = \sum_{i=1}^M \min_{k \in \bar{K}} f_{ik}(x^k) \quad (3.9)$$

where $\bar{K} = \{1, \dots, K\}, x^k \in \mathbb{R}^n, x = (x^1, \dots, x^K) \in \mathbb{R}^{K \times n}$ and the functions f_{ik} are separable

$$f_{ik}(x) = \sum_{j=1}^n f_{ijk}(x_j^k).$$

The function (3.9) can be derived from the function (3.5) when

$$J_i = \{1\}, 1 \leq i \leq M, K_j = \{1, \dots, K\}.$$

In order to calculate one discrete gradient of the function (3.9) we have to evaluate functions f_{ijk} $MK(n + 1)$ times. However the use of the simplified scheme reduces this number to $2MK + n - 1$.

3.5 Numerical experiments

A number of numerical experiments have been carried out using large scale nonsmooth optimisation problems.

3.5.1 Test problems

The following test problems have been used in numerical experiments. The description of chained problems can be also found in [83, 133, 136]. We consider unconstrained minimisation problems. Below f_* stands for the minimum value of function f .

Chained problems

Problem 1 (Chained LQ function).

$$f(x) = \sum_{i=1}^{n-1} \max \{ -x_i - x_{i+1}, -x_i - x_{i+1} + (x_i^2 + x_{i+1}^2 - 1) \}, f_* = -(n-1)\sqrt{2}.$$

Problem 2 (Chained CB3 I function).

$$f(x) = \sum_{i=1}^{n-1} \max \{ x_i^4 + x_{i+1}^2, (2 - x_i)^2 + (2 - x_{i+1})^2, 2e^{-x_i + x_{i+1}} \}, f_* = 2(n-1).$$

Problem 3 (Chained CB3 II function).

$$f(x) = \max \left\{ \sum_{i=1}^{n-1} (x_i^4 + x_{i+1}^2), \sum_{i=1}^{n-1} ((2 - x_i)^2 + (2 - x_{i+1})^2), 2 \sum_{i=1}^{n-1} e^{-x_i + x_{i+1}} \right\},$$

$$f_* = 2(n-1).$$

Problem 4 (Nonsmooth generalization of Brown function 2).

$$f(x) = \sum_{i=1}^{n-1} \left(|x_i|^{x_{i+1}^2+1} + |x_{i+1}|^{x_i^2+1} \right), f_* = 0.$$

Problem 5 (Chained Mifflin 2 function).

$$f(x) = \sum_{i=1}^{n-1} \left(-x_i + 2(x_i^2 - x_{i+1}^2 - 1) + 1.75|x_i^2 + x_{i+1}^2 - 1| \right), f_* \text{ varies.}$$

Problem 6 (Chained Crescent I function).

$$f(x) = \max \left\{ \sum_{i=1}^{n-1} (x_i^2 + (x_{i+1} - 1)^2 + x_{i+1} - 1), \sum_{i=1}^{n-1} (-x_i^2 - (x_{i+1} - 1)^2 + x_{i+1} + 1) \right\},$$

$$f_* = 0.$$

Problem 7 (Chained Crescent II function).

$$f(x) = \sum_{i=1}^{n-1} \max \{ x_i^2 + (x_{i+1} - 1)^2 + x_{i+1} - 1, -x_i^2 - (x_{i+1} - 1)^2 + x_{i+1} + 1 \}, f_* = 0.$$

Problem 8 (Chained Wood function).

$$f(x) = \sum_{j=1}^k [100(x_{2j-1}^2 - x_{2j})^2 + (x_{2j-1} - 1)^2 + 90(x_{2j+1}^2 - x_{2j+2})^2 + (x_{2j+1} - 1)^2$$

$$+ 10(x_{2j} + x_{2j+1} - 2)^2 + (x_{2j} - x_{2j+2})^2/10],$$

$$k = (n - 2)/2,$$

$$f_* = 0.$$

Problem 9 (Chained Powell singular function).

$$f(x) = \sum_{j=1}^k [(x_{2j-1} + 10x_{2j})^2 + 5(x_{2j+1} - x_{2j+2})^2 + (x_{2j} - 2x_{2j+1})^4 + 10(x_{2j-1} - x_{2j+2})^4]$$

$$k = (n - 2)/2,$$

$$f_* = 0.$$

Piecewise partially separable problems

Problem 10 (PPSF CB3 I function).

$$f(x) = \sum_{i=1}^n \max \{ x_i^4 + x_1^2, (2 - x_i)^2 + (2 - x_1)^2, 2e^{-x_i+x_1} \}, f_* = 2n.$$

Problem 11 (PPSF CB3 II function).

$$f(x) = \max \left\{ \sum_{i=1}^n (x_i^4 + x_1^2), \sum_{i=1}^n ((2 - x_i)^2 + (2 - x_1)^2), 2 \sum_{i=1}^n (e^{-x_i+x_1}) \right\}, f_* = 2n.$$

Problem 12 (PPSF Nonsmooth generalization of Brown function 2).

$$f(x) = \sum_{i=1}^n \left(|x_i|^{x_i^2+1} + |x_1|^{x_i^2+1} \right), f_* = 0.$$

Problem 13 (PPSF Broyden function).

$$f(x) = \sum_{i=1}^n |(3 - 2x_i)x_i - x_1 - x_2 + 1|^{7/3}, f_* = 0.$$

The objective functions are smooth in Problems (8), (9) and they are piecewise partially separable in Problems (10)-(13). The latter functions are modifications of corresponding test functions.

The code has been written in C++ and numerical experiments have been carried out in PC Intel Pentium 4, 1.6 MHz. Their results are presented in tables 3.1-3.4. In these tables we use the following notations:

- n is the number of variables;
- t the CPU time in seconds;
- N_f the number of evaluations of core functions when the simplified scheme is applied;
- N_S the number of objective function evaluations when the simplified scheme is applied;
- N_g the number of objective function evaluations without application of the simplified scheme;
- x^0 and x^* are the initial point and the minimiser, respectively.

We consider that starting from the point x^0 the algorithm succeeds if for the final point \bar{x} the inequality

$$\frac{f(\bar{x}) - f_*}{|f_*| + 1} < \delta$$

is true. Otherwise we say that it fails. Here the tolerance $\delta = 10^{-4}$.

In the numerical experiments for each problem and n we ran the algorithm starting from 100 randomly chosen points. In the tables we present average values of t , N_f , N_S and N_g/N_S . In the column "Failed" we present the number

	n	t	N_f	N_S	N_g/N_S	Failed	$f(x_0) - f(x^*)$	
							min	max
Chained	2000	44.30	3.23e7	1.62e4	1800.0	0	1.27e05	1.43e05
LQ	1500	23.60	1.84e7	1.23e4	1310.0	0	9.34e04	1.07e05
	1000	11.40	9.28e6	9.29e3	859.0	0	6.32e04	7.27e04
	800	8.31	6.66e6	8.34e3	686.0	0	4.84e04	5.80e04
	500	5.18	4.10e6	8.21e3	416.0	0	2.99e04	3.76e04
	300	3.26	2.38e6	7.95e3	244.0	0	1.79e04	2.22e04
	100	1.76	7.58e5	7.66e3	79.0	0	5.24e03	8.65e03
	50	1.31	3.02e5	6.17e3	39.9	0	2.33e03	4.47e03
	10	0.34	1.10e4	1.22e3	5.6	0	2.54e02	1.00e03
Chained	2000	81.20	3.26e7	1.63e4	1380.0	0	2.73e09	7.80e09
CB3 I	1500	49.00	2.13e7	1.42e4	963.0	0	1.86e09	6.71e09
	1000	25.60	1.14e7	1.14e4	633.0	0	9.85e08	4.39e09
	800	18.70	8.29e6	1.04e4	511.0	0	2.28e08	4.23e09
	500	9.25	4.20e6	8.42e3	309.0	0	3.29e08	3.13e09
	300	3.56	1.81e6	6.04e3	163.0	0	1.17e08	2.26e09
	100	0.75	3.93e5	3.97e3	49.2	0	2.95e06	9.50e08
	50	0.37	1.69e5	3.46e3	24.6	0	9.48e04	6.97e08
	10	0.03	1.44e4	1.60e3	3.0	0	3.96e03	3.55e08
Chained	2000	40.20	2.07e7	1.03e4	1250.0	0	2.92e09	7.65e09
CB3II	1500	18.2	1.06e7	7.04e3	806.0	0	2.01e09	5.76e09
	1000	8.22	5.19e6	5.19e3	488.0	0	9.61e08	4.42e09
	800	5.72	3.62e6	4.53e3	385.0	0	7.16e08	3.84e09
	500	2.93	1.87e6	3.76e3	243.0	0	3.73e08	2.99e09
	300	1.50	9.44e5	3.16e3	150.0	0	5.76e07	2.39e09
	100	0.50	2.37e5	2.39e3	54.0	0	1.05e07	1.40e09
	50	0.29	9.83e4	2.01e3	26.2	0	1.19e05	1.06e09
	10	0.02	1.03e4	1.15e3	3.9	0	4.38e03	2.42e08

Table 3.1: Results for piecewise chained functions

of failures of the algorithm. We also present the minimum and maximum values of the difference $f(x^0) - f(x^*)$ in order to demonstrate how far the initial points are from the solution.

Figures 3.1 and 3.3 show the dependence of N_S on the number of variables n for piecewise chained and piecewise partially separable functions, respectively and figures 3.2 and 3.4 show the dependence of N_g/N_S on the number of variables n for these functions.

As one can see from tables 3.1 - 3.4 the proposed algorithm allows us to solve all problems with a given accuracy except problem 4 (with $n = 2000$), Problem 6 (with $n = 1000, 2000$), problem 7 (with $n = 100 - 2000$) and problem 13 (with $n = 800, 1000, 1500, 2000$). However, it should be noted that all problems have been solved with accuracy $\varepsilon = 10^{-2}$. In the numerical

	n	t	N_f	N_S	N_g/N_S	Failed	$f(x_0) - f(x^*)$	
							min	max
Chained	2000	76.60	1.79e7	8.94e3	1850.0	5	8.86e02	9.37e02
generalised	1500	32.30	7.76e6	5.18e3	1360.0	0	6.56e02	7.09e02
Brown 2	1000	15.30	3.83e6	3.84e3	874.0	0	4.35e02	4.77e02
	800	9.75	2.50e6	3.12e3	689.0	0	3.49e02	3.77e02
	500	4.91	1.15e6	2.30e3	425.0	0	2.14e02	2.42e02
	300	3.24	5.78e5	1.93e3	250.0	0	1.28e02	1.45e02
	100	1.57	1.38e5	1.39e3	82.1	0	4.02e01	4.99e01
	50	2.48	5.56e4	1.14e3	40.3	0	1.86e01	2.64e01
	10	0.03	4.53e3	5.03e2	6.6	0	2.39e00	5.87e00
Chained	2000	10.70	8.84e6	4.42e3	1840.0	5	1.27e05	1.43e05
Crescent I	1500	5.51	4.72e6	3.15e3	1350.0	3	9.48e04	1.09e05
	1000	2.45	2.23e6	2.23e3	875.0	0	6.26e04	7.05e04
	800	1.71	1.56e6	1.95e3	692.0	0	4.87e04	5.72e04
	500	0.99	8.62e5	1.73e3	428.0	0	2.93e04	3.60e04
	300	0.62	4.54e5	1.52e3	256.0	0	1.78e04	2.30e04
	100	0.30	1.19e5	1.20e3	82.9	0	5.37e03	8.04e03
	50	0.23	5.18e4	1.06e3	40.5	0	2.33e03	4.47e03
	10	0.03	5.78e3	6.42e2	6.7	0	2.83e02	9.95e02
Chained	2000	25.80	2.15e7	1.08e4	1760.0	100	1.27e05	1.41e05
Crescent II	1500	11.20	1.07e7	7.17e3	1250.0	99	9.40e04	1.06e05
	1000	4.45	5.14e6	5.15e3	779.0	100	6.20e04	7.23e04
	800	2.83	3.62e6	4.53e3	606.0	98	4.96e04	5.86e04
	500	1.30	1.84e6	3.69e3	361.0	99	3.01e04	3.64e04
	300	0.70	8.95e5	2.99e3	207.0	97	1.76e04	2.25e04
	100	2.48	2.53e5	2.55e3	71.2	61	5.11e03	8.56e03
	50	0.50	1.25e5	2.55e3	39.1	0	2.31e03	4.46e03
	10	0.26	7.51e3	8.34e2	6.0	0	2.61e02	9.23e02

Table 3.2: Results for piecewise chained functions

	n	t	N_f	N_S	N_g/N_S	Failed	$f(x_0) - f(x^*)$	
							min	max
Chained	2000	85.20	7.18e7	3.59e4	1700.0	0	4.69e05	5.16e05
Mifflin	1500	63.50	6.73e7	4.49e4	1140.0	0	3.48e05	3.91e05
	1000	22.50	2.83e7	2.83e4	708.0	0	2.29e05	2.68e05
	800	14.00	1.83e7	2.29e4	549.0	0	1.84e05	2.11e05
	500	6.13	8.08e6	1.62e4	331.0	0	1.13e05	1.33e05
	300	3.15	3.73e6	1.25e4	198.0	0	6.61e04	8.37e04
	100	1.60	7.97e5	8.05e3	75.0	0	1.76e04	3.10e04
	50	1.65	3.37e5	6.88e3	40.0	0	8.59e03	1.81e04
	10	0.91	1.39e4	1.54e3	6.0	0	9.15e02	3.84e03
Chained	2000	62.00	4.27e7	2.14e4	1290.0	0	3.31e08	3.86e08
Wood	1500	34.20	2.44e7	1.63e4	844.0	0	2.49e08	3.05e08
	1000	19.20	1.26e7	1.26e4	466.0	0	1.62e08	1.99e08
	800	14.00	8.77e6	1.10e4	345.0	0	1.22e08	1.72e08
	500	13.20	4.39e6	8.79e3	201.0	0	7.79e07	1.11e08
	300	10.40	2.25e6	7.51e3	117.0	0	4.20e07	6.59e07
	100	11.40	6.19e5	6.25e3	40.6	0	1.08e07	2.77e07
	50	11.10	2.73e5	5.57e3	22.7	0	4.35e06	1.28e07
	10	0.29	2.75e4	3.05e3	3.7	0	5.70e04	3.18e06
Chained	2000	24.10	1.53e7	7.64e3	1040.0	0	1.48e08	1.88e08
Powell	1500	13.90	9.13e6	6.09e3	658.0	0	1.10e08	1.52e08
singular	1000	8.89	5.29e6	5.30e3	402.0	0	7.19e07	1.01e08
	800	13.00	4.13e6	5.17e3	325.0	0	5.08e07	8.58e07
	500	6.42	2.52e6	5.04e3	219.0	0	3.36e07	5.68e07
	300	5.72	1.67e6	5.58e3	142.0	0	1.78e07	3.67e07
	100	5.69	5.40e5	5.45e3	44.4	0	4.33e06	1.36e07
	50	5.44	2.24e5	4.58e3	21.6	0	1.26e06	7.62e06
	10	0.21	3.79e4	4.22e3	3.3	0	2.49e04	1.77e06

Table 3.3: Results for piecewise chained functions

	n	t	N_f	N_S	N_g/N_S	Failed	$f(x_0) - f(x^*)$	
							min	max
PPSF	2000	51.20	3.87e7	1.94e4	854.0	0	3.82e06	8.07e10
CB3 I	1500	27.50	2.19e7	1.46e4	605.0	0	2.80e06	7.70e10
	1000	12.90	1.07e7	1.07e4	384.0	0	1.88e06	2.93e10
	800	9.26	7.85e6	9.82e3	298.0	0	1.49e06	3.99e10
	500	4.93	4.26e6	8.53e3	178.0	0	9.11e05	2.00e10
	300	2.71	2.28e6	7.63e3	103.0	0	5.17e05	1.89e10
	100	1.57	5.24e5	5.29e3	30.5	0	1.33e05	5.91e09
	50	2.26	2.01e5	4.09e3	15.5	0	6.04e04	3.07e09
	10	0.04	1.50e4	1.67e3	3.3	0	3.00e03	9.64e07
PPSF	2000	24.40	2.12e7	1.06e4	777.0	0	3.74e06	9.08e10
CB3 II	1500	11.70	1.08e7	7.20e3	540.0	0	2.76e06	6.83e10
	1000	5.62	5.35e6	5.36e3	341.0	0	1.84e06	2.67e10
	800	4.14	3.98e6	4.98e3	261.0	0	1.47e06	3.80e10
	500	2.24	2.17e6	4.35e3	163.0	0	9.04e05	2.08e10
	300	1.29	1.31e6	4.36e3	97.1	0	5.08e05	1.93e10
	100	0.86	9.97e5	1.01e4	29.8	0	1.36e05	3.98e09
	50	0.25	1.99e5	4.05e3	15.6	0	7.76e04	2.09e09
	10	0.02	1.08e4	1.19e3	3.4	0	3.68e03	4.93e08
PPSF	2000	79.60	3.74e7	1.87e4	975.0	3	5.00e02	1.32e03
generalised	1500	39.80	1.94e7	1.30e4	724.0	0	4.01e02	9.97e02
Brown 2	1000	16.30	8.22e6	8.23e3	473.0	0	2.47e02	6.59e02
	800	10.70	5.38e6	6.74e3	376.0	0	1.97e02	5.27e02
	500	5.10	2.40e6	4.81e3	232.0	0	1.24e02	3.23e02
	300	3.60	1.24e6	4.13e3	138.0	0	7.74e01	1.98e02
	100	2.98	2.74e5	2.77e3	45.9	0	2.28e01	6.68e01
	50	2.89	1.01e5	2.07e3	23.0	0	1.23e01	3.41e01
	10	0.02	6.21e3	6.89e2	4.7	0	1.50e00	6.79e00
	PPSF	2000	54.50	6.18e7	3.09e4	525.0	93	1.25e07
Broyden	1500	19.10	2.44e7	1.63e4	329.0	82	8.85e06	1.72e07
	1000	8.41	1.16e7	1.16e4	199.0	32	6.25e06	1.10e07
	800	5.59	7.99e6	1.00e4	164.0	8	5.05e06	8.75e06
	500	2.48	3.32e6	6.66e3	109.0	0	2.62e06	5.73e06
	300	1.32	1.58e6	5.29e3	70.9	0	1.75e06	3.35e06
	100	0.57	2.96e5	2.99e3	26.5	0	5.27e05	1.36e06
	50	0.62	1.26e5	2.56e3	14.0	0	2.39e05	6.85e05
	10	0.06	1.60e4	1.78e3	3.1	0	1.05e04	1.95e05

Table 3.4: Results for piecewise partially separable functions

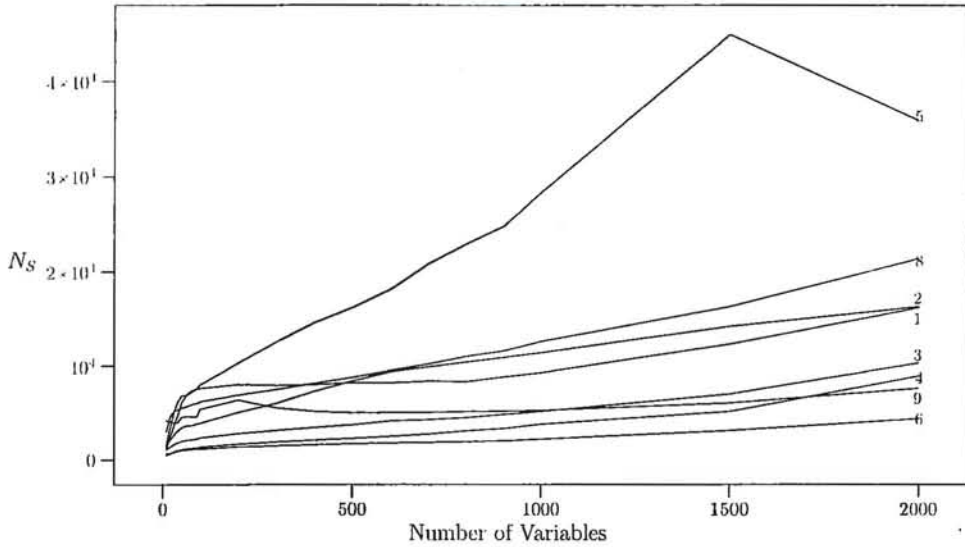


Figure 3.1: Average number of function evaluations for piecewise chained functions

experiments we restricted the maximum number of discrete gradients which can be calculated at each iteration to 200. In all these problems in order to calculate solutions with higher accuracy we have to significantly increase this number. But in this case the CPU time may increase substantially.

Results for CPU time reported in the tables demonstrate that the algorithm is quite fast to find solutions with the given accuracy in problems up to 2000 variables.

The numbers presented in columns for the minimum and maximum values of the difference $f(x^0) - f_*$ show that randomly chosen initial points are not close to the solutions for almost all problems and all n . Therefore one can assert that the number of objective function evaluations N_S is moderate for all problems and n . We can also see from figures 3.1 and 3.3 that the number N_S is almost a linear function of the number of variables for all problems for which the algorithm was successful.

The ratio N_g/N_S increases as the number of variables increases. Figures 3.2 and 3.4 demonstrate that this ratio is a linear function of the number of variables and $N_g/N_S \approx \alpha n$ where $\alpha = 0.30 \div 0.95$.

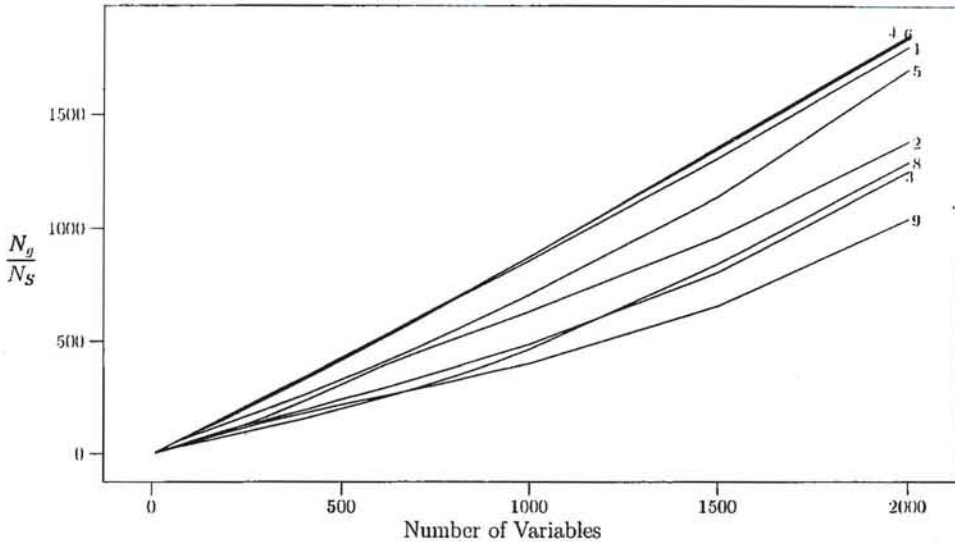


Figure 3.2: Average ratio of the number of function evaluations for general scheme to simplified scheme for chained functions

3.6 Conclusion

In this chapter we have developed an algorithm for solving one class of large scale nonsmooth optimisation. This class contains piecewise partially separable functions. These functions have many practical applications including applications in data mining and information retrieval. The algorithm for minimisation of these functions is the modification of the discrete gradient method. It has been shown that the calculation of discrete gradients can be significantly accelerated. We present results of numerical experiments which demonstrate that the proposed algorithm is effective for solving many large scale nonsmooth optimisation problems up to 2000 variables.

As it was pointed out above in this chapter the discrete gradient method consists of three major steps: the computation of the discrete gradients, the computation of a descent direction by solving a certain quadratic programming problem and a line search. The simplified scheme proposed in this chapter allows one to significantly accelerate the computation of the discrete gradients. However, the acceleration of two other steps taking into the structure of large scale problems may lead to a more efficient algorithm and its application to a broad class of large scale nonsmooth optimisation problems.

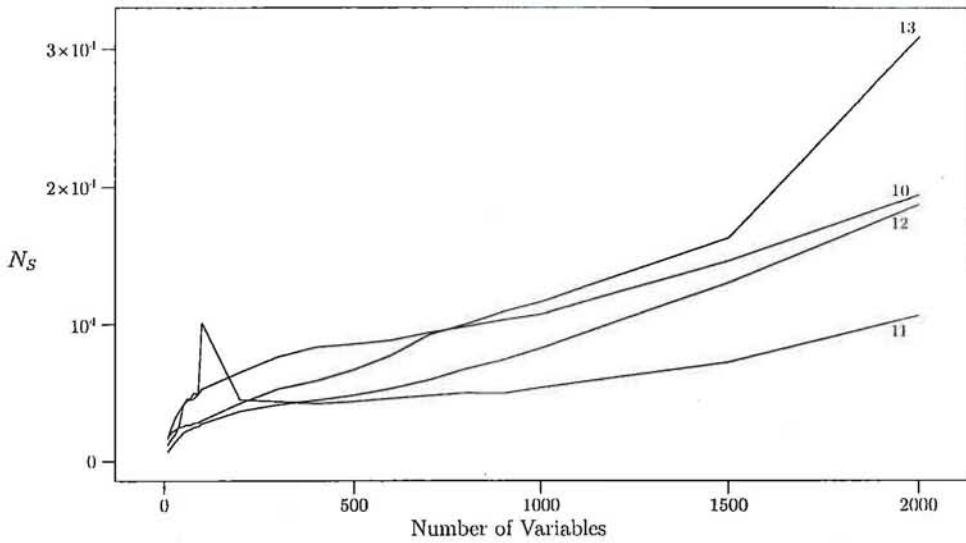


Figure 3.3: Average number of function evaluations for piecewise partially separable functions

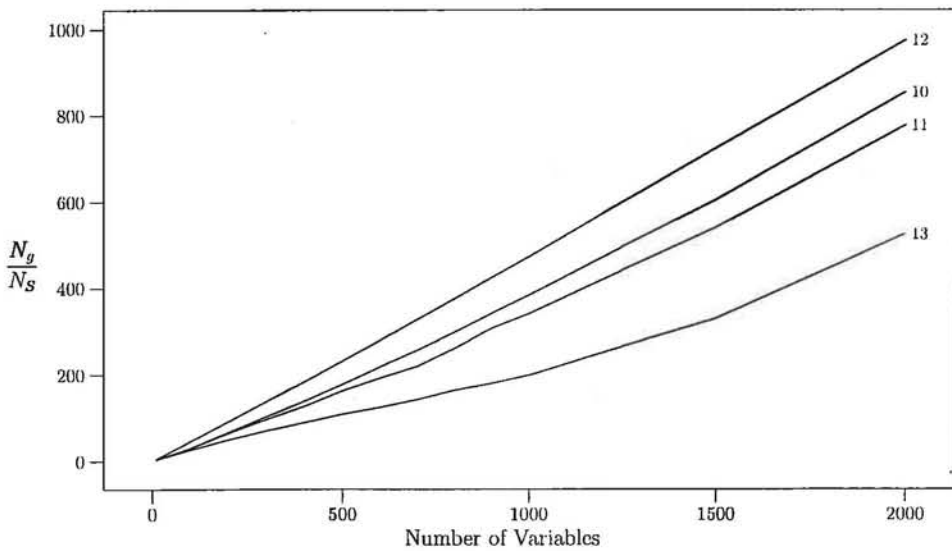


Figure 3.4: Average ratio of the number of function evaluations for general scheme to simplified scheme for PPS functions

Chapter 4

Minimisation algorithms for data analysis problems

4.1 Introduction

Optimisation based formulations of data analysis problems present very generally the same shape: due to the discrete nature of the sets of n -dimensional points representing the datasets, the objective functions are usually based on a set of operators, such as:

sum over the elements of the dataset;

maximum or minimum over the elements of the dataset;

product over the elements of the dataset.

In the case of the sum, the maximum and the minimum, it has been shown in proposition 17 on page 76 that these functions conserve the partial piecewise separability. Functions taking the form of a product are encountered very seldom, and can be transformed using the logarithm operator.

4.2 Solving the Max-Min separation problem

In this section we discuss an algorithm for minimisation of the error function (1.25) on page 34.

4.2.1 Statement of problem

The problem of the max-min separability is reduced to the following mathematical programming problem:

$$\begin{aligned} & \text{minimise } f(x, y) \\ & \text{subject to} \\ & (x, y) \in \mathbb{R}^{(n+1) \times l} \end{aligned} \quad (4.1)$$

where the objective function f has the following form:

$$f(x, y) = f_1(x, y) + f_2(x, y)$$

and

$$f_1(x, y) = \frac{1}{m} \sum_{k=1}^m \max \left[0, \max_{i \in I} \min_{j \in J_i} \{ \langle x^j, a^k \rangle - y_j + 1 \} \right], \quad (4.2a)$$

$$f_2(x, y) = \frac{1}{p} \sum_{t=1}^p \max \left[0, \min_{i \in I} \max_{j \in J_i} \{ -\langle x^j, b^t \rangle + y_j + 1 \} \right]. \quad (4.2b)$$

The problem (4.1) is a global optimisation problem. However, the number of variables in this problem is large and the global optimisation methods cannot be directly applied to solve it. Therefore we will discuss algorithms for finding local minima of the function f .

The functions f_1 and f_2 are nonconvex piecewise linear. These functions are Lipschitz continuous and consequently subdifferentiable in the sense of Clarke (see definition 18 on page 51). Moreover, both functions are semismooth (see definition 16 on page 50). Therefore the function f is also subdifferentiable. The function f_1 contains the following max-min functions:

$$\varphi_{1k}(x, y) = \max_{i \in I} \min_{j \in J_i} \{ \langle x^j, a^k \rangle - y_j + 1 \}, \quad 1 \leq k \leq m$$

and the function f_2 contains the following min-max functions:

$$\varphi_{2t}(x, y) = \min_{i \in I} \max_{j \in J_i} \{ -\langle x^j, b^t \rangle + y_j + 1 \}, \quad 1 \leq t \leq p.$$

The differential properties of max-min functions are studied, for example, in [58, 160]. The functions φ_{1k} , $1 \leq k \leq m$ and φ_{2t} , $1 \leq t \leq p$ are not regular (see definition 20 on page 51). Then the functions f_1 , f_2 and consequently the function f are not regular, too. Therefore the calculation of subgradients of the function f is a difficult task. This implies that the methods of nonsmooth

optimisation which use subgradients at each iteration seem not to be effective for solving problem (4.1).

In the paper [111] optimisation problems with twice continuously differentiable objective functions and max-min constraints were considered and these problems were converted to problems with smooth objective and constraint functions. However, this approach cannot be applied to the problem (4.1), because the function f contains not only max-min-type functions but also min-max-type functions.

In order to provide an efficient algorithm for minimising the error function, the scheme presented in section 3.4 on page 80 will be applied.

4.2.2 Differential properties of the objective function

Proposition 20. *The function f is semismooth.*

Proof. The sum, the maximum and the minimum of semismooth functions are semismooth (see [146]). A linear function, as a smooth function, is semismooth. Thus the function f which is the sum of functions represented as the maximum of 0 and max-min of linear functions, is semismooth. \square

4.2.3 Calculation of the discrete gradients of the objective function

Due to their piecewise linearity, it is quite clear that functions f_1 and f_2 defined in (4.2a) and (4.2b) on the preceding page are piecewise separable. The objective function of problem (4.1), as their sum, is thus piecewise separable.

The scheme proposed in section 3.4.2 can therefore be applied on this function.

The objective function f depends on $(n + 1)l$ variables where l is the number of hyperplanes. The function f_1 contains max-min functions φ_{1k}

$$\varphi_{1k}(x, y) = \max_{i \in I} \min_{j \in J_i} \psi_{1jk}(x, y), 1 \leq k \leq m$$

where

$$\psi_{1jk}(x, y) = \langle x^j, a^k \rangle - y_j + 1, j \in J_i, i \in I.$$

We can see that for every $1 \leq k \leq m$, each pair of variables $\{x^j, y_j\}$ appears in only one function ψ_{1jk} .

For a given $1 \leq i \leq (n + 1)l$ we set

$$q_i = \left\lfloor \frac{i - 1}{n + 1} \right\rfloor + 1, d_i = i - (q_i - 1)(n + 1)$$

We define by X the vector of all variables $\{x^j, y_j\}, 1 \leq j \leq l$:

$$X = (X_1, X_2, \dots, X_{(n+1)l})$$

where

$$X_i = \begin{cases} x_{d_i}^{q_i} & \text{if } 1 \leq d_i \leq n, \\ y_{q_i} & \text{if } d_i = n + 1. \end{cases}$$

We use the vector of variables X to define a sequence

$$X_t^0, \dots, X_t^{t-1}, X_t^{t+1}, \dots, X_t^{(n+1)l}, t \in I(g, \alpha), g \in \mathbb{R}^{(n+1)l}$$

as in remark 17 on page 82. It follows from (2.10) that the points X_t^{i-1} and X_t^i differ by one coordinate only. This coordinate appears in only one linear function $\psi_{1q_i k}$. It follows from the definition of the operator H_t^j that $X_t^t = X_t^{t-1}$ and thus this observation is also true for X_t^{t+1} . Then we get

$$\psi_{1jk}(X_t^i) = \psi_{1jk}(X_t^{i-1}), \forall j \neq q_i.$$

Moreover the function $\psi_{1q_i k}$ can be calculated at the point X_t^i using the value of this function at the point $X_t^{i-1}, i \geq 1$:

$$\psi_{1q_i k}(X_t^i) = \begin{cases} \psi_{1q_i k}(X_t^{i-1}) - z(\lambda)a_{d_i}^k e_i(\beta) & \text{if } 1 \leq d_i \leq n, \\ \psi_{1q_i k}(X_t^{i-1}) + z(\lambda)e_i(\beta) & \text{if } d_i = n + 1. \end{cases} \quad (4.3)$$

In order to calculate the function f_1 at the point $X_t^i, i \geq 1$ first we have to calculate the values of the functions $\psi_{1q_i k}$ for all $a^k \in A, 1 \leq k \leq m$ using (4.3). Then we update f_1 using these values and the values of all other linear functions at the point X_t^{i-1} according to (4.2a). Thus we have to apply a full calculation of the function f_1 using the formula (4.2a) only at the point $X_t^0 = X + \lambda g$.

Since the function f_2 has a similar structure as f_1 we can calculate it in the same manner using a formula similar to (4.3).

Thus for the calculation of each discrete gradient we have to apply a full calculation of the objective function f only at the point $X_t^0 = X + \lambda g$ and this function can be updated at the points $X_t^i, i \geq 1$ using a simplified scheme.

We can conclude that for the calculation of the discrete gradient at a point X with respect to the direction $g^0 \in S_1$ we calculate the function f at two points: X and $X_t^0 = X + \lambda g^0$. For the calculation of another discrete gradient at the same point X with respect to any other direction $g^1 \in S_1$ we calculate the function f only at the point: $X + \lambda g^1$.

Since the number of variables $(n+1)l$ in the problem (4.1) can be large this algorithm allows us to significantly reduce the number of objective function evaluations during the calculation of a discrete gradient.

On the other hand the function f_1 contains max-min-type functions and their computation can be simplified using an algorithm proposed in [70]. The function f_2 contains min-max-type functions and a similar algorithm can be used for their calculation.

Results of numerical experiments show that the use of these algorithms allows one to significantly accelerate the computation of the objective function f and its discrete gradients.

4.3 Solving the clustering problem

Problem (1.4) on page 15 is a nonsmooth optimisation problem. It is very difficult to adapt the simplified scheme to the general form of these functions, since the dissimilarity measure can take any form. We will present here a scheme for minimising the cluster function when the dissimilarity is measured using a Minkowski metric (see formula 1.1 on page 12). Since the objective functions in these problems are represented as a sum of minimum of norms they are Lipschitz continuous. Moreover they are piecewise partially separable functions, and the scheme presented in section 3.4 on page 80 can be applied.

4.3.1 Differential properties of the objective functions

Since function f from (1.4) is locally Lipschitz continuous they are subdifferentiable.

Proposition 21. *The function f is semismooth.*

Proof. The sum and the minimum of semismooth functions are semismooth (see [146]). A norm as a convex function is semismooth. Then the function f which is the sum of functions represented as the minimum of norms is semismooth. \square

The objective function is clearly piecewise partially separable, as the sum of minima of smaller functions.

4.3.2 Calculation of the discrete gradients of the objective function of the clustering problem

The objective function f of the problem (1.4) depends on $n \times q$ variables where q is the number of clusters. This function is represented as a sum of

min-type functions

$$f(x) = \sum_{i=1}^m \psi_i(x)$$

where

$$\psi_i(x) = \min_{1 \leq j \leq q} [\eta_{ij}(x)]^\theta, 1 \leq i \leq m$$

with $\theta = \frac{\gamma}{p}$, and

$$\eta_{ij}(x) = \sum_{u=1}^n |x_u^j - a_u^i|^p.$$

It should be noted that the functions η_{ij} are separable. We can see that for every $1 \leq i \leq m$, each variable x^j appears in only one function η_{ij} .

For a given $1 \leq k \leq nq$ we set

$$r_k = \left\lfloor \frac{k-1}{n} \right\rfloor + 1, d_k = k - (r_k - 1)n.$$

We define by X the vector of all variables $x^j, 1 \leq j \leq q$:

$$X = (X_1, X_2, \dots, X_{nq})$$

where

$$X_k = x_{d_k}^{r_k}.$$

We use the vector of variables X to define a sequence

$$X_t^0, \dots, X_t^{t-1}, X_t^{t+1}, \dots, X_t^{nq}, t \in I(g, \alpha), g \in \mathbb{R}^{nq}$$

as in remark 17. It follows from (2.10) that the points X_t^{k-1} and X_t^k differ by one coordinate only ($0 \leq k \leq nq, k \neq t$). For every $1 \leq i \leq m$ this coordinate appears in only one function η_{ir_k} . It follows from the definition of the operator H that $X_t^t = X_t^{t-1}$ and thus this observation is also true for X_t^{t+1} . Then we get

$$\eta_{ij}(X_t^k) = \eta_{ij}(X_t^{k-1}) \forall j \neq r_k$$

which means that when we change the k -th coordinate of the point X only one function (namely the function η_{ir_k}) changes its value.

Moreover the function η_{ir_k} can be calculated at the point X_t^k using the value of this function at the point $X_t^{k-1}, k \geq 1$:

$$\eta_{ir_k}(X_t^k) = \eta_{ir_k}(X_t^{k-1}) + |x_{d_k}^{r_k} + z(\lambda)e_k(\beta) - a_{d_k}^i|^p - |x_{d_k}^{r_k} - a_{d_k}^i|^p. \quad (4.4)$$

Because η is separable, only one term in the sum changes. Thus we need to add the new term, and subtract the old one. For $p = 2$, (4.3) can be simplified as follows

$$\eta_{ir_k}(X_t^k) = \eta_{ir_k}(X_t^{k-1}) + z(\lambda)e_k(\beta) (2x_{d_k}^{r_k} + z(\lambda)e_k(\beta) - 2a_{d_k}^i).$$

In order to calculate the function f at the point $X_t^k, k \geq 1$ first we have to calculate the values of the functions η_{ir_k} for all $a^i \in A, 1 \leq i \leq m$ using (4.3). Then we update f using these values and the values of all other functions $\eta_{ij}, j \neq r_k$ at the point X_t^{k-1} according to (1.4). Thus we have to apply a full calculation of the function f using the formula (1.4) only at the point $X_t^0 = X + \lambda g$. Hence for the calculation of each discrete gradient we have to apply a full calculation of the objective function f only at the point $X_t^0 = X + \lambda g$ and this function can be updated at the points $X_t^k, k \geq 1$ using a simplified scheme.

We can conclude that for the calculation of the discrete gradient at a point X with respect to the direction $g^0 \in S_1$ we calculate the function f at two points: X and $X_t^0 = X + \lambda g^0$. For the calculation of another discrete gradient at the same point X with respect to another direction $g^1 \in S_1$ we calculate the function f only at the point: $X + \lambda g^1$.

Since the number of variables nq in the problem (1.4) can be large this algorithm allows one to significantly reduce the number of objective function evaluations during the calculation of a discrete gradient.

Part III

Data analysis methods

Chapter 5

Classification using Max-Min separability

In this chapter we present an application of the max-min separation to solve data supervised classification problems. Numerical experiments on real world datasets are presented.

5.1 Supervised data classification via max-min separability

We are given a dataset A containing a finite number of points in \mathbb{R}^n . This dataset contains d disjoint subsets A_1, \dots, A_d where A_i represents a training set for the class i . The aim of supervised data classification is to establish rules for the classification of new observations using these training subsets of the classes. This problem is reduced to d set separation problems.

Each of these problems consists in separating one class from the rest of the dataset. To separate the class i from all others, we separate sets A_i and $\bigcup_{j \neq i} A_j$, with a piecewise linear function by solving problem (1.25) on page 34.

One of the important questions in supervised data classification is the estimation of performance measure. Different performance measures are discussed in [185]. When the dataset contains two classes the classification problem can be reduced to only one separation problem, therefore the classification rules are straightforward. We consider that the separation function obtained from the training set, separates the two classes.

When the dataset contains more than two classes we have more than one separation function. In our case for each class i of the dataset A we have one piecewise linear function φ_i separating the training set A_i from all

other training points $\bigcup_{j \neq i} A_j$. We approximate the training set A_i using the following set

$$\bar{A}_i = \{a \in \mathbb{R}^n : \varphi_i(a) < 0\}.$$

Thus we get the sets $\bar{A}_1, \dots, \bar{A}_d$ which approximate the training sets A_1, \dots, A_d , respectively. Then for each $i \in \{1, \dots, d\}$ we can consider the following two sets:

$$A_i^0 = \bar{A}_i, \bar{A}_i^0 = \bigcup_{j=1, j \neq i}^d \bar{A}_j$$

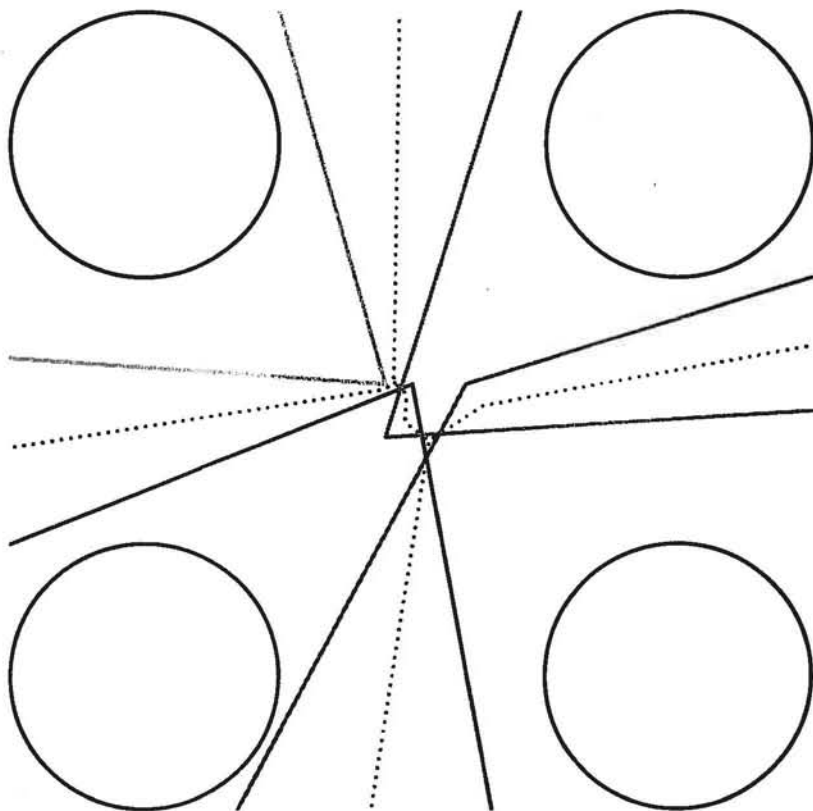


Figure 5.1: Classification using max-min separability

These two sets define the following four sets (see figure 5.1):

1. $A_i^0 \cap (\mathbb{R}^n \setminus \bar{A}_i^0)$
2. $(\mathbb{R}^n \setminus A_i^0) \cap \bar{A}_i^0$
3. $A_i^0 \cap \bar{A}_i^0$

4. $(\mathbb{R}^n \setminus A_i^0) \cap (\mathbb{R}^n \setminus \bar{A}_i^0)$

If a new observation a belongs to the first set we classify it in class i , if it belongs to the second set we classify it not to be in class i . If this point belongs to the third or fourth set in this case if $\varphi_i(a) < \min_{1 \leq j \leq d, j \neq i} \varphi_j(a)$ then we classify it in class i , otherwise we classify it not to be in class i .

In order to evaluate the classification algorithm we use two performance measures. First we present the average accuracy (a_{2c} in tables 5.1 and 5.2) for two classes classification (when one particular class is separated from all others) and the multi-class classification accuracy (a_{mc} in tables 5.1 and 5.2) as described above. First accuracy is an indication of separation quality and the second one is an indication of multi-class classification quality.

5.2 Results on large datasets

5.2.1 Datasets

The datasets used are the Shuttle control, the Letter recognition, the Landsat satellite image, the Pen-based recognition of handwritten and the Page blocks classification databases.

5.2.2 Results and discussion

Our algorithm has been implemented in C++ on a Pentium 4 1.7 GHz. We took $X^0 = 0 \in \mathbb{R}^{(n+1)l}$ as a starting point for solving each separation problem (1.25). The parameters in the calculation of the discrete gradient for all iterations k and all separation problems were chosen as follows: $z_k(\lambda) = \lambda^t$, $t = 3 \div 3.5$, $\lambda_{k+1} = 0.5\lambda_k$, $\lambda_0 = 0.9$, $\beta_k = 1$. At each iteration of the discrete gradient method the line search is carried out by approximation of the objective function using univariate piecewise linear function (see [14]). In each separation problem (1.25) all J_i , $i \in I$ have the same cardinality.

Results of numerical experiments are presented in tables 5.1 and 5.2. In these tables *fct eval* and *DG eval* show respectively the average number of objective function evaluations and of discrete gradient evaluations required to solve an optimisation problem.

From the results presented in these tables we can see that the use of the max-min separability algorithm allows us to achieve a very high classification accuracy for both training and test phases. Results on training sets show that this algorithm provides a very high quality of separation between two sets. In our experiments we used only large-scale datasets. Results on these datasets show that a few hyperplanes are sufficient to separate efficiently

sets with large numbers of points. Since we use a derivative-free method to solve problem (1.25) the number of objective function evaluations is a significant characteristic for estimation of the complexity of the max-min separability algorithm. Results presented in tables 5.1 and 5.2 confirm that the proposed algorithm is effective for solving classification problems on large-scale databases.

$ I $	$ J_i $	Training		Test		fct eval	DG eval
		a_{2c}	a_{mc}	a_{2c}	a_{mc}		
Shuttle control dataset							
1	1	97.61	97.22	97.53	97.00	925	615
2	1	99.44	97.56	99.41	97.42	2148	1676
3	1	99.61	97.57	99.59	97.50	1474	968
2	2	99.68	99.06	99.68	99.07	2723	2196
3	2	99.75	99.67	99.73	99.53	2663	2079
3	3	100.00	99.87	99.97	99.84	2108	1493
Letter recognition dataset							
1	1	88.44	84.09	88.61	85.42	628	336
2	1	94.09	90.29	93.86	92.52	1646	926
3	1	96.29	96.03	96.12	95.08	2902	1707
2	2	95.73	94.15	95.53	94.76	2950	1816
3	2	97.47	97.25	97.18	95.20	3778	2382
3	3	97.59	96.69	97.25	94.56	4017	2544
Landsat satellite image dataset							
1	1	85.63	68.46	84.83	83.35	990	606
2	1	94.52	94.22	92.89	90.30	3178	2075
3	1	94.83	91.52	93.10	89.05	4082	2618
2	2	95.02	93.78	93.49	89.50	3951	2556
3	2	94.96	93.80	93.61	89.65	4125	2625
3	3	95.79	91.00	94.07	82.30	4687	2991

Table 5.1: Results of numerical experiments with Shuttle control, Letter recognition and Landsat satellite image datasets

$ I $	$ J_i $	Training		Test		fct eval.	DG. eval.
		a_{2c}	a_{mc}	a_{2c}	a_{mc}		
Pen-based recognition of handwritten dataset							
1	1	97.27	96.74	96.43	92.37	1597	1146
2	1	99.31	99.44	98.33	96.14	2607	1852
3	1	99.79	99.92	98.89	96.20	3040	2220
2	2	99.80	99.95	98.99	96.03	3083	2306
3	2	99.83	99.87	99.12	95.88	1806	1268
3	3	99.73	99.88	98.96	95.80	2693	1966
Page blocks dataset							
1	1	96.57	92.30	92.29	81.26	5234	3511
2	1	96.93	94.63	93.10	87.17	4359	2840
3	1	97.22	95.28	92.71	86.08	4865	3197
2	2	97.45	95.50	93.09	85.54	5737	3618
3	2	97.45	95.83	93.39	85.61	5289	3189
3	3	97.95	96.28	93.85	85.95	4918	3224

Table 5.2: Results of numerical experiments with Pen-based recognition of handwritten and Page blocks datasets

5.3 Restrictions to the current model

The piecewise linear separation is a very powerful tool which has a number of advantages over the other separation methods. More general than the linear and the polyhedral separations (of which it can be seen as a generalisation), it does not require any assumption made on the distribution of the points, like the Support Vector Machines. If one had to make an analogy between these two methods, one could see the piecewise linear separation as a linear approximation of the separating function, which as rough as it may be, is certainly preferable to an erroneous hand-made approximation proposed in the SVM. It has been shown that the only condition for two sets to be piecewise linearly separable is that they have to be disjoint.

Nevertheless, a number of improvements are needed for this method to be even more powerful. The optimisation problem associated with the piecewise linear separation is a complicated one, with a nonconvex, nonsmooth objective function, with a large number of local minima when the number of hyperplanes is large enough. Global methods of optimisation are generally not efficient to solve this problem for real world large scale datasets, and

therefore only local methods like the discrete gradient method can be used. It is crucial to be able to find a good initial point, in order to avoid this method to get trapped in a shallow local minimum. Such initial points must be guessed from the structure of the dataset, and one idea may be to use the piecewise linear separation via clustering presented in section 1.3.7.

The other crucial point is the guess of the number of hyperplanes needed for the separation. If not enough hyperplanes are used, then the separation will be ineffective. If too many hyperplanes are computed, the problem becomes very large, and the number of local minima increases without use for it. For the problem of clustering, we will present an approach (see section 6.3 on page 111) where incremental algorithms are developed to evaluate automatically how many clusters are needed. An incremental solution seems to carry much hope in the case of the piecewise linear separation, as this solution also solves the problem of the initial point. However the cluster function (1.4) only contains mins of functions, and adding a new centre to an existing solution can not increase the function value. Meanwhile, the Maxmin separability function (1.25) contains max, and adding a new hyperplane may increase the value of the function, making it more difficult to obtain a convergent algorithm.

5.4 Conclusion

In this chapter the max-min separability has been presented. It has been proved that this method can successfully discriminate disjoint finite sets of points, using a max-min of linear functions.

One main application of this algorithm is the data classification, and numerical experiments were carried out on large size real world dataset, demonstrating the effectiveness of this algorithm.

The max-min classification offers advantages over other nonlinear separating methods such as the support vector machine, in that they are not based on any assumption over the distribution of the points in the plane.

Nevertheless, this method necessitates a number of improvements. Firstly, the number of hyperplanes is usually not known in advance. It is therefore necessary to develop a method to find automatically the number of hyperplanes. Problem (1.25) is a global optimisation problem. When the number of hyperplanes is higher than 1, it is nonconvex. As most global methods are computationally not effective for solving this problem in large scale datasets, a local algorithm was developed, and therefore it is crucial to compute a good starting point, in order to improve the computational efficiency and the classification accuracy.

Chapter 6

Solving the clustering problem

6.1 Introduction

The clustering problem as formulated in (1.4) in section 1.2 on page 9 has been widely studied. For small datasets (containing few clusters and few features), it may be possible to apply global optimisation methods, but in general it is not possible. The piecewise partial separability of the objective function of this problems allows us to apply a scheme on the discrete gradient method, in order to find very efficiently a local minimum of this function.

Unfortunately there can be no guarantee that a “false” solution, under the form of a shallow local minimum, will be avoided. Therefore it is crucial to design techniques which ascertain that at least a good solution will be reached.

In this chapter we will describe two techniques. These techniques differ in their approach of the problem, and we will discuss the advantages and disadvantages of both.

6.2 A class of sum-min functions

6.2.1 Functions represented as the sum of minima of convex functions

Consider finite dimensional vector spaces \mathbb{R}^n and \mathbb{R}^m . Let $A \subset \mathbb{R}^n$ be a finite set and let k be a positive integer. Consider a sum of minima of convex functions (SMC) F defined on $(\mathbb{R}^m)^k$ by

$$F(x_1, \dots, x_k) = \sum_{a \in A} \min_{1 \leq i \leq k} (\varphi_k(x_k, a)), \quad (6.1)$$

where $x \mapsto \varphi_i(x, a)$ is a convex function defined on \mathbb{R}^m ($1 \leq i \leq k$, $a \in A$). We do not assume that this function is smooth. Let us denote the class of functions of the form (6.1) by \mathcal{F} .

This class of functions is a subclass of the cluster function (1.4) on page 15, where the dissimilarity is convex. Notwithstanding noticeable exceptions (as in the design of telecommunication networks), this is generally the case.

6.2.2 Some properties of SMC functions

Let $F \in \mathcal{F}$, that is F has the form (6.1). Then F enjoys the following properties:

- F is quasidifferentiable (see section 2.2.3 on page 52). Moreover, F is DC (the difference of convex functions). Indeed, we have (see for example [60, p. 108]):

$$F(x) = f_1(x) - f_2(x), \quad x = (x_1, \dots, x_k),$$

where

$$f_1(x) = \sum_{a \in A} \sum_{i=1}^k \varphi_i(x_i, a)$$

$$f_2(x) = \sum_{a \in A} \max_{i=1, \dots, k} \sum_{j \neq i} \varphi_j(x_j, a).$$

Both f_1 and f_2 are convex functions. The pair $DF(x) = (\partial f_1(x), -\partial f_2(x))$ is a quasidifferential of F at a point x . Here ∂f stands for the convex subdifferential of a convex function f .

- Since F is DC, it follows that this function is locally Lipschitz.
- Since F is DC it follows that this function is semi-smooth.
- Since F is locally Lipschitz, it is subdifferentiable in the sense of Clarke (see section 2.2.2 on page 51).
- Since F is semismooth, the discrete gradient method can be applied.
- The function F is a piecewise partially separable function (see chapter 3 on page 70). As a sum of minimum functions, the scheme described in section 3.4 on page 80 can be applied. Such a scheme has been detailed for Minkowski metrics (see equation (1.1) on page 12) in section 4.3 on page 99.

6.3 Step by step approach to clustering

Basis of the approach

Although the optimisation problem (1.4) on page 15 is in general nonconvex, it should be noticed that there is convexity when only one cluster is sought. In this case, the solution reached by the discrete gradient method is the optimal one.

On the other hand, in practical situations, it is often impossible to fix the number of clusters in advance. This means that several options should be tried until a satisfactory solution is reached.

The core idea of the step by step approach is to combine these two facts by adding the clusters one by one, while updating the sets of existing clusters.

The method

Algorithm 6.1: An algorithm for solving the clustering problem

Step 1 *Initialisation.* Select a tolerance $\varepsilon > 0$.

Select a starting point $x^0 = (x_1^0, \dots, x_n^0) \in \mathbb{R}^n$ and solve the minimisation problem (1.4) for $q = 1$.

Let $x^{1*} \in \mathbb{R}^n$ be a solution to this problem and f^{1*} be the corresponding objective function value.

Set $k = 0$.

repeat

Step 2 set $k \leftarrow k + 1$

Step 3 *Computation of the next cluster centre.* Select a starting point $y^0 \in \mathbb{R}^n$ and solve the following minimisation problem:

$$\begin{aligned} & \text{minimise } \bar{f}^k(y) \\ & \text{subject to} \\ & y \in \mathbb{R}^n \end{aligned} \tag{6.2}$$

where

$$\bar{f}^k(y) = \sum_{i=1}^m \min \{ \|x^{1*} - a^i\|_p^\gamma, \dots, \|x^{k*} - a^i\|_p^\gamma, \|y - a^i\|_p^\gamma \}. \tag{6.3}$$

Step 4 *Refinement of all cluster centres.* Let $y^{k+1,*}$ be a solution to problem (6.2). Take $x^{k+1,0} = (x^{1*}, \dots, x^{k*}, y^{k+1,*})$ as a new starting point and solve the problem (1.4) for $q = k + 1$. Let $x^{k+1,*}$ be a solution to problem (1.4) and $f^{k+1,*}$ be the corresponding objective function value.

Step 5 until *Stopping criterion:*

$$\frac{f^{k*} - f^{k+1,*}}{f^{1*}} < \varepsilon$$

Algorithm 6.1 on the preceding page presents an iterative method for solving the clustering problem. In Step 1 the centre of the entire set A is calculated with respect to a given norm. In this case the problem (1.4) is a convex programming problem. In Step 3 we calculate the centre of the next $(k + 1)$ -th cluster, assuming the previous k cluster centres to be known and fixed. It should be noted that the number of variables in problem (6.2) is n which is substantially less than if we calculate all cluster centres simultaneously. In Step 4 the refinement of all $k + 1$ cluster centres is carried out. One can expect that the starting point $x^{k+1,0}$ calculated in Step 3 is not far from the solution to problem (1.4). Therefore it takes only a moderate number of iterations to calculate it. Such an approach allows one to significantly reduce the computational time for solving problem (1.4).

It is clear that $f^{k*} \geq 0$ for all $k \geq 1$ and the sequence $\{f^{k*}\}$ is decreasing, that is,

$$f^{k+1*} \leq f^{k*}, \forall k \geq 1.$$

The latter implies that after $\bar{k} > 0$ iterations the stopping criterion in step 5 will be satisfied.

One of the important questions when one tries to apply algorithm 6.1 is the choice of the tolerance $\varepsilon > 0$. Large values of ε can result in the appearance of large clusters whereas small values can produce small and artificial clusters. Results presented in [24] show that appropriate values for ε are $\varepsilon \in [10^{-1}, 10^{-2}]$.

An algorithm for solving problems (1.4) is discussed in section 4.3 on page 99. This algorithm is based on the partial piecewise separability of the objective function. Since the function (6.3) has a very similar shape, it is also possible to apply this method to solve problem (6.2). This algorithm is a local one, hence the choice of a good initial guess for problem (6.2) is very important. An algorithm for finding such an initial guess is described below. It should be noted that starting points in problem (1.4) are predetermined in algorithm 6.1.

6.3.1 An algorithm for finding the initial points in problem (6.2)

The main idea behind step 3 in algorithm 6.1 is that a new cluster is added to preexisting ones.

In [87] an improvement for the k -means algorithm, j -means, is given, to avoid so-called *degenerated solutions*, where one cluster is empty: the furthest point from all cluster centres is taken as a new centre. This leads to an interesting idea, however it needs to be further improved: often real-world

datasets contain erroneous records which can be quite far from the rest of the dataset. Taking such an erroneous point as an initial guess may lead to a shallow local minimum. In figure 6.1, the point P_1 is the furthest point from the cluster centres. However if this point is chosen as the centre of a cluster, this cluster will consist of only P_1 . Moreover this initial set of centres, being also a local solution, will not be improved. The point P_2 is closer to the centres, however it would induce a much better cluster.

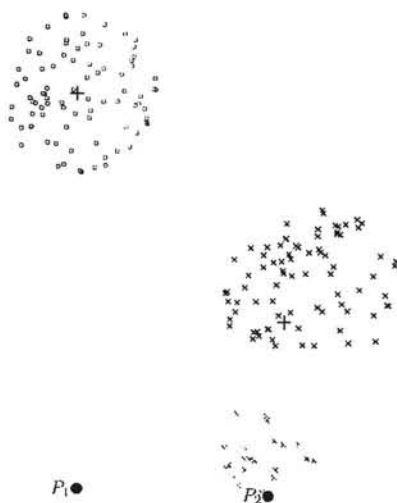


Figure 6.1: An example of a dataset containing an erroneous point

Algorithm 6.2 allows one to avoid this difficulty.

Algorithm 6.2: An algorithm for finding the initial point in problem (6.2).

Step 1 *Initialisation.* Let $C_1, \dots, C_{q-1}, q \geq 2$ be the preexisting centres and $\rho > 0$ a tolerance. Let $A_1 = A$, and $i = 1$.

repeat

Step 2 | Let C be the furthest point in A_i from the centres C_1, \dots, C_{q-1} .

Step 3 | Find the set

$$C = \left\{ a \in A_i : \|C - a\| < \min_{1 \leq j \leq q-1} \|C_j - a\| \right\}.$$

Step 4 | Set $A_{i+1} = A_i \setminus \{C\}$, $i = i + 1$

until $\text{card}(C) > \rho$

Remark 20: Since A contains a finite number of points the initial guess will be found after a finite number of steps. If the tolerance ρ is too small

then the initial guess may be an erroneous point, but if ρ is large then no initial point can be found. Results of numerical experiments show that the tolerance ρ should be chosen in $\left[0.1\frac{m}{q}, 0.4\frac{m}{q}\right]$.

6.3.2 Results of numerical experiments

To verify the effectiveness of the proposed approach a number of numerical experiments with real-world data sets have been carried out on a Pentium-4, 1.7 GHz, PC. The datasets used in numerical experiments are Fisher's iris dataset, the image segmentation dataset, TSPLIB1060 and TSPLIB3038.

In order to implement the discrete gradient algorithm (algorithm 2.3 on page 60) we have to select the sequences $\{\delta_k\}$, $\delta_k > 0$, $\{z_k\}$, $z_k \in P$, $\{\lambda_k\}$, $\lambda_k > 0$, $\{\beta_k\}$, $\beta_k \in (0, 1]$. In the numerical experiments we chose these sequences as follows: $\delta_k = 10^{-9}$, $\beta_k = 1$ for all k , $z_k(\lambda) = \lambda^\alpha$, $\alpha \in [1.5, 4]$, $\lambda_k = d^k \lambda_0$, $d = 0.5$, $\lambda_0 = 0.9$. We take $\lambda_{min} \in (0, \lambda_0)$. If $\lambda_k < \lambda_{min}$ then algorithm 2.3 terminates. In order to get a solution with high accuracy one has to take λ_{min} very small, for example $\lambda_{min} \leq 10^{-5}$. Larger values of λ_{min} may lead to more inaccurate solutions, however algorithm 2.3 calculates such solutions very quickly. In our numerical experiments, unless specified otherwise, we set λ_{min} large.

In the numerical experiments we also take $c_1 = 0.2$ and $c_2 = 0.001$. The starting point for solving problem (6.2) is generated by algorithm 6.2 and the starting point for solving problem (1.4) is generated in algorithm 6.1. In algorithm 6.2 we fix the parameter $\rho = \frac{rm}{q}$ where $r \in [0, 0.5]$.

For the image segmentation dataset we take $\lambda_{min} = 10^{-5}$. For all other datasets this parameter is $\lambda_{min} = 0.01$.

In the numerical experiments we consider the squared Euclidean norm, that is $\gamma = 2$ and $p = 2$.

In order to provide comparison with the best known solutions from the literature we calculate 10 clusters for iris dataset and 50 clusters for all other datasets.

Algorithm 6.1 is a deterministic one. However, it requires the selection of the parameter ρ , which determines the starting points. To see how this parameter influences the results, numerical experiments have been carried out for $r_i = 0.05i$, $0 \leq i \leq 10$.

We present the results of the numerical experiments in tables 6.1-6.4. In these tables k represents the number of clusters. We give the best known function value f_{opt} from the literature corresponding to k clusters ([87, 89]) and the % error E_{best} and E_{mean} of respectively the best value and the average

value obtained by the proposed algorithm. The error E is calculated as

$$E = \frac{(\bar{f} - f_{opt})}{f_{opt}} \cdot 100\%$$

where \bar{f} is the function value obtained by the algorithm. A negative error means that the proposed algorithm improved the best known solution. We also give the average number of calculations of norms for reaching the solutions. N_s and N_g represent the average number of norm evaluations for finding k clusters using the simplified and general schemes as described in section 4.3 on page 99, respectively. In the tables we give the values of N_s and the ratio N_g/N_s . Finally, in these tables we present the average CPU time (t) and its standard deviation (σ_t).

Results presented in tables 6.1-6.4 show that for Fisher's iris, TSPLIB3038 and TSPLIB1060 datasets the proposed algorithm allows one to calculate either the best known solution or a solution which is very close to the best one. For image segmentation dataset the results for large numbers of clusters are not so good. This can be explained by the existence of erroneous points. These results demonstrate that if the algorithm gets stuck in a shallow minimum then it may affect the next iterations. However, results for this dataset show that the algorithm in most cases reaches a solution which is close to the best one. It should be noted that the problem of finding 50 clusters in the image segmentation dataset has 950 variables which is challenging for many global optimisation techniques.

The results for the error of average values presented in these tables show that for Fisher's iris, TSPLIB3038 and TSPLIB1060 datasets the results obtained by the proposed algorithm do not strongly depend on the initial point (that is values of ρ) and they are always close to the best solutions. The results for the image segmentation dataset are more dependent on the initial point. This is an indicator of the existence of erroneous points in this dataset, and shows that if a dataset does not have a good cluster structure the proposed algorithm may lead to different solutions starting from different initial points.

The results for the number N_s show that one can calculate a large number of clusters using a reasonable number of norm evaluations. The ratios N_g/N_s demonstrate that the simplified scheme allows one to significantly reduce the computational effort. This complexity reduction becomes larger as the number of clusters or the number of features increase (see also figures 6.2, 6.4, 6.6 and 6.8).

The results presented in the tables show that the clustering problem can be solved by the algorithm within a reasonable CPU time. CPU time depends

k	f_{opt}	E_{best}	E_{mean}	N_s	N_g/N_s	t	σ_t
2	152.348	0.00	0.00	$1.43 \cdot 10^5$	3.63	0.10	0.02
3	78.851	0.00	0.00	$3.22 \cdot 10^5$	5.00	0.20	0.03
4	57.226	0.00	0.00	$5.83 \cdot 10^5$	5.95	0.34	0.05
5	46.446	0.00	0.68	$8.73 \cdot 10^5$	7.33	0.49	0.05
6	39.040	0.00	0.00	$1.21 \cdot 10^6$	8.23	0.65	0.04
7	34.298	0.00	0.86	$1.57 \cdot 10^6$	9.31	0.82	0.05
8	29.989	0.00	0.11	$2.03 \cdot 10^6$	10.62	1.04	0.07
9	27.786	0.00	2.11	$2.46 \cdot 10^6$	11.86	1.23	0.09
10	25.834	0.52	1.78	$3.00 \cdot 10^6$	13.04	1.47	0.07

Table 6.1: Results for Fisher's Iris dataset

on the parameter λ_{min} . Small values of λ_{min} lead to a better solution, however much more CPU time is required, as in the case of the image segmentation dataset. The result for σ_t show that the CPU time does not strongly depend on starting points (see also figures 6.3, 6.5, 6.7 and 6.9).

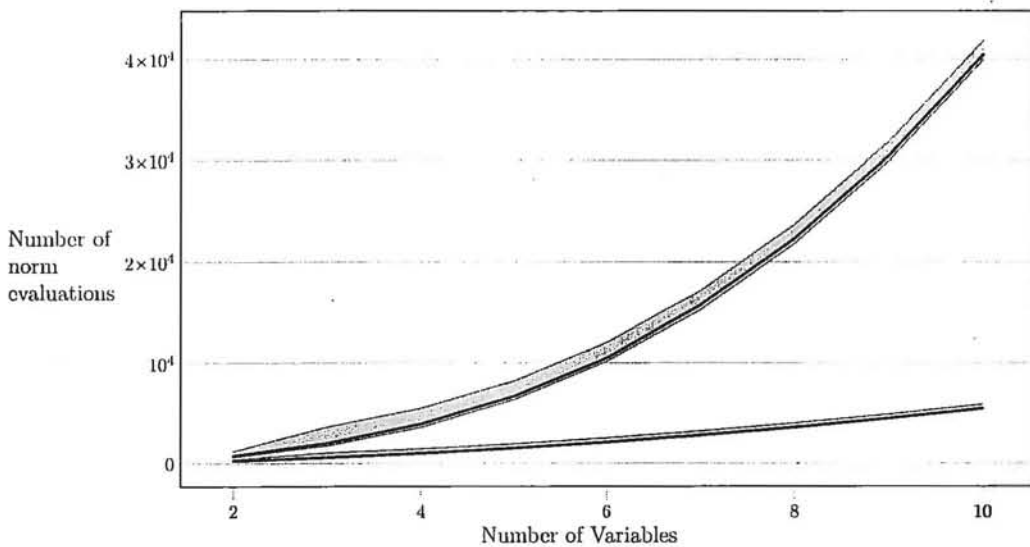


Figure 6.2: Evolution of the computational effort for Fisher's iris dataset

k	f_{opt}	E_{best}	E_{mean}	N_s	N_g/N_s	t	σ_t
2	$3.5606 \cdot 10^7$	-0.01	2.08	$8.14 \cdot 10^6$	20.69	19.43	3.12
3	$2.7416 \cdot 10^7$	-0.02	4.90	$1.86 \cdot 10^7$	27.31	40.05	6.93
4	$1.9456 \cdot 10^7$	-0.03	15.82	$3.60 \cdot 10^7$	33.70	69.37	12.92
5	$1.7143 \cdot 10^7$	-0.03	13.16	$5.45 \cdot 10^7$	40.41	98.53	22.23
6	$1.5209 \cdot 10^7$	-0.03	16.36	$7.73 \cdot 10^7$	47.34	131.88	32.67
7	$1.3404 \cdot 10^7$	0.33	10.50	$1.04 \cdot 10^8$	54.77	169.15	47.53
8	$1.2030 \cdot 10^7$	2.28	14.29	$1.30 \cdot 10^8$	62.96	205.52	49.40
9	$1.0784 \cdot 10^7$	1.36	9.28	$1.61 \cdot 10^8$	70.12	248.53	70.28
10	$9.7952 \cdot 10^6$	1.51	8.72	$1.90 \cdot 10^8$	77.12	289.24	74.52
20	$5.1283 \cdot 10^6$	-0.01	8.76	$2.48 \cdot 10^8$	111.15	897.46	198.27
30	$3.5076 \cdot 10^6$	5.62	10.21	$3.33 \cdot 10^8$	184.89	1815.14	255.73
40	$2.7398 \cdot 10^6$	9.96	13.52	$3.50 \cdot 10^8$	207.80	2396.03	247.34
50	$2.2248 \cdot 10^6$	15.26	19.30	$3.65 \cdot 10^8$	231.72	3011.10	241.75

Table 6.2: Results for Image Segmentation dataset

k	f_{opt}	E_{best}	E_{mean}	N_s	N_g/N_s	t	σ_t
2	$3.1688 \cdot 10^9$	0.00	0.00	$1.78 \cdot 10^6$	1.69	0.74	0.17
3	$2.1763 \cdot 10^9$	0.00	1.51	$4.31 \cdot 10^6$	2.24	1.65	0.19
4	$1.4790 \cdot 10^9$	0.00	0.03	$6.62 \cdot 10^6$	2.82	2.48	0.14
5	$1.1982 \cdot 10^9$	0.00	0.21	$9.56 \cdot 10^6$	3.35	3.50	0.16
6	$9.6918 \cdot 10^8$	0.00	0.05	$1.26 \cdot 10^7$	3.90	4.56	0.16
7	$8.3966 \cdot 10^8$	1.73	1.91	$1.52 \cdot 10^7$	4.54	5.53	0.11
8	$7.3475 \cdot 10^8$	0.00	0.75	$1.94 \cdot 10^7$	5.20	6.99	0.32
9	$6.4477 \cdot 10^8$	0.00	0.44	$2.34 \cdot 10^7$	5.94	8.41	0.40
10	$5.6025 \cdot 10^8$	0.00	0.68	$2.78 \cdot 10^7$	6.73	9.98	0.37
20	$2.6681 \cdot 10^8$	0.00	0.90	$3.40 \cdot 10^7$	10.07	29.35	1.04
30	$1.7557 \cdot 10^8$	0.27	1.55	$4.32 \cdot 10^7$	16.56	61.04	1.35
40	$1.2548 \cdot 10^8$	-0.08	1.48	$5.45 \cdot 10^7$	24.70	104.74	1.72
50	$9.8400 \cdot 10^7$	0.62	1.63	$6.82 \cdot 10^7$	33.88	158.03	3.83

Table 6.3: Results for TSPLIB3038 dataset

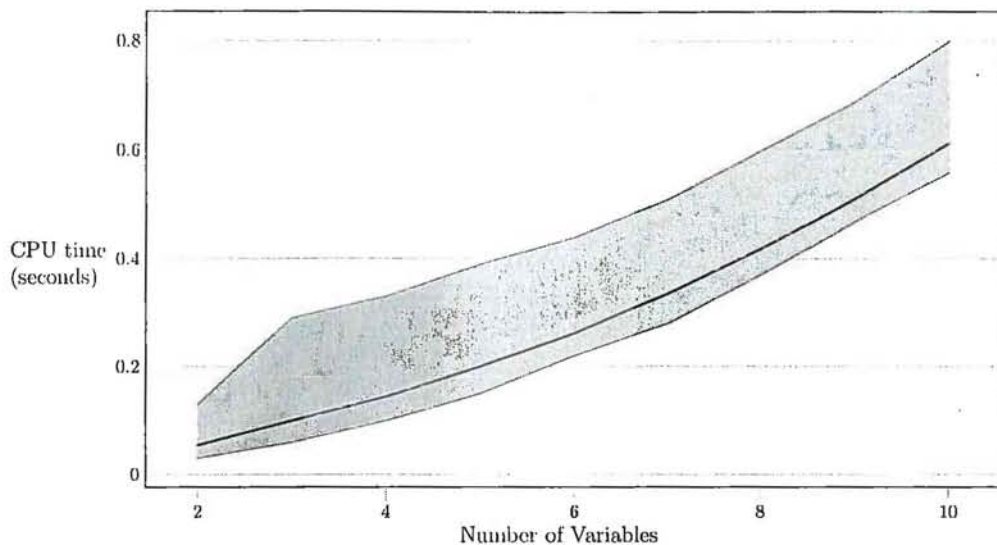


Figure 6.3: Improvement in the computational effort for Fisher's iris dataset

k	f_{opt}	E_{best}	E_{mean}	N_s	N_g/N_s	t	σ_t
10	$1.75484 \cdot 10^9$	0.00	0.18	$1.39 \cdot 10^7$	6.00	4.61	0.42
20	$7.91794 \cdot 10^9$	0.32	3.01	$1.77 \cdot 10^7$	8.74	13.62	1.08
30	$4.81251 \cdot 10^9$	1.29	3.76	$2.46 \cdot 10^7$	14.03	29.02	1.48
50	$2.55509 \cdot 10^9$	1.36	2.46	$4.48 \cdot 10^7$	28.12	81.90	3.72

Table 6.4: Results for TSPLIB1060 Dataset

6.4 Simultaneous clustering

Step by step clustering is a very interesting method, as it tackles the problem of finding the number of clusters, and generates a good initial guess for each of the problems to be solved. However, it may sometimes be necessary to solve the clustering problem very quickly. In such a case, this method, which requires the resolution of many mathematical programming problems, may not be the best option.

In this section we will study the selection of initial points for the clustering problem. When the time efficiency is an issue, it is often advantageous to use the ε cleaning (see section 1.4.3 on page 39), and to minimise the generalised cluster function (1.27). Initial points for this function will therefore be the subject of a particular attention.

We will present a selection of starting points for the discrete gradient method. Very few results exist in the literature for ε -cleaned datasets. In

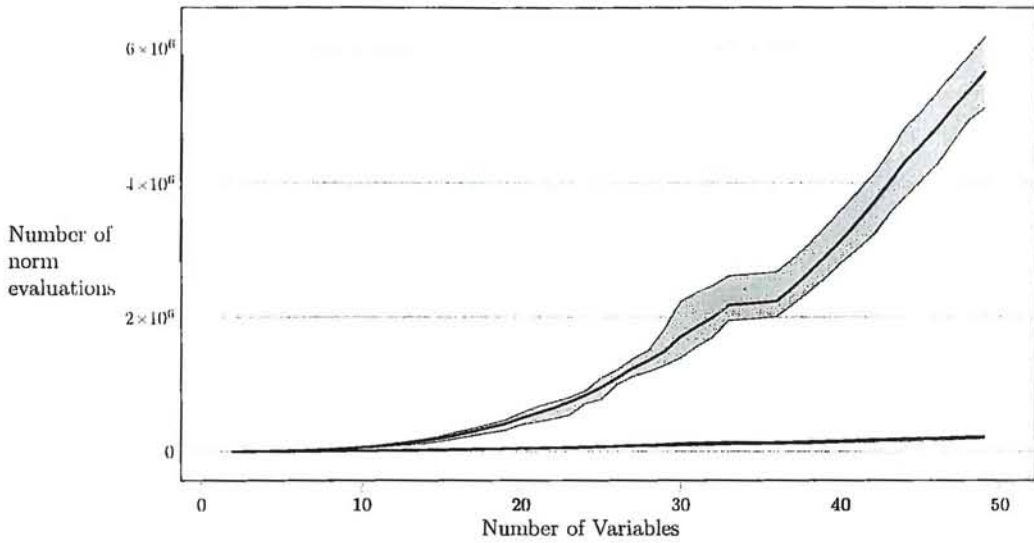


Figure 6.4: Evolution of the computational effort for Image segmentation dataset

order to evaluate the results reached starting from these points, we will apply the hybrid method between DG and the cutting angle method (DG+CAM) (see section 2.7.2 on page 65), and the commercial software GAMS (LGO solver), (see [79, 158]). LGO is based on the branch and bound method.

These methods were applied to the minimisation of the cluster function for various types of dissimilarity measures: the Minkowsky metric (Norm 1), and the skeleton function, which is a variation of the Bradley-Mangasarian function ([37])

We report results of numerical experiments and analyse these results.

6.4.1 Minimisation of SMC functions

Consider function F defined by (6.1):

$$F(x_1, \dots, x_k) = \frac{1}{N} \sum_{a \in A} \min(\varphi_1(x_1, a), \varphi_2(x_2, a), \dots, \varphi_k(x_k, a)),$$

$$x_i \in \mathbb{R}^n; i = 1, \dots, k.$$

where $A \subset \mathbb{R}^n$ is a finite set. This function depends on $n \times k$ variables. In real-world applications $n \times k$ is a large enough number and the set A contains some hundreds or thousands of points. In such a case function F has a huge amount of shallow local minimisers that are very close to each other. The minimisation of such functions is a challenging problem.

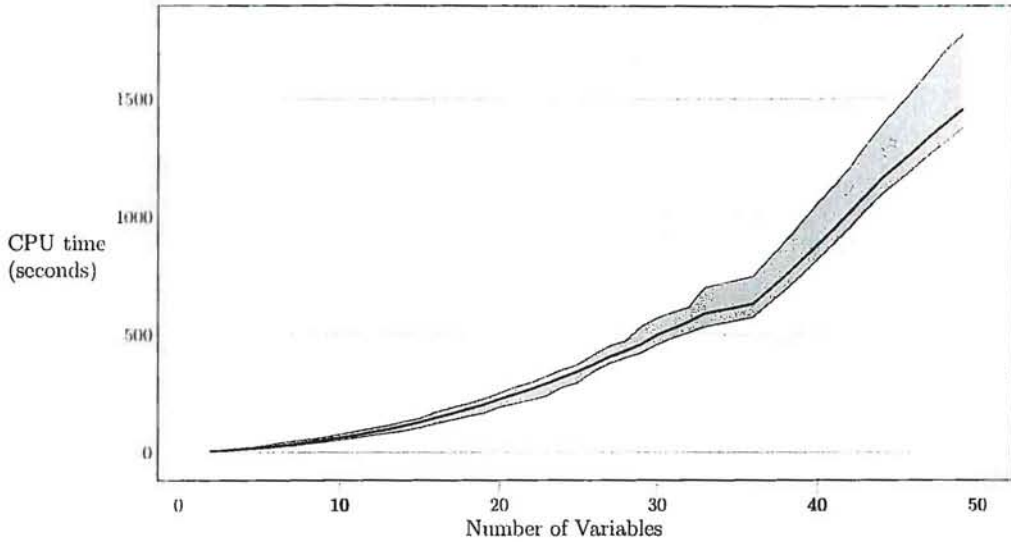


Figure 6.5: Improvement in the computational effort for Image segmentation dataset

This version of the CIAO-GO software (Centre for Informatics and Applied Optimisation-Global Optimisation) allows one to use four different solvers

1. DG,
2. DG multi start,
3. DG+CAM,
4. DG+CAM multi start.

Working with this software users have to input

- an objective function (for minimisation),
- an initial point for optimisation,
- upper and low bounds for variables,
- constraints and a penalty constant (in the case of constrained optimisation), constraints can be represented as equalities and inequalities,
- maximal running time,
- maximal number of iterations.

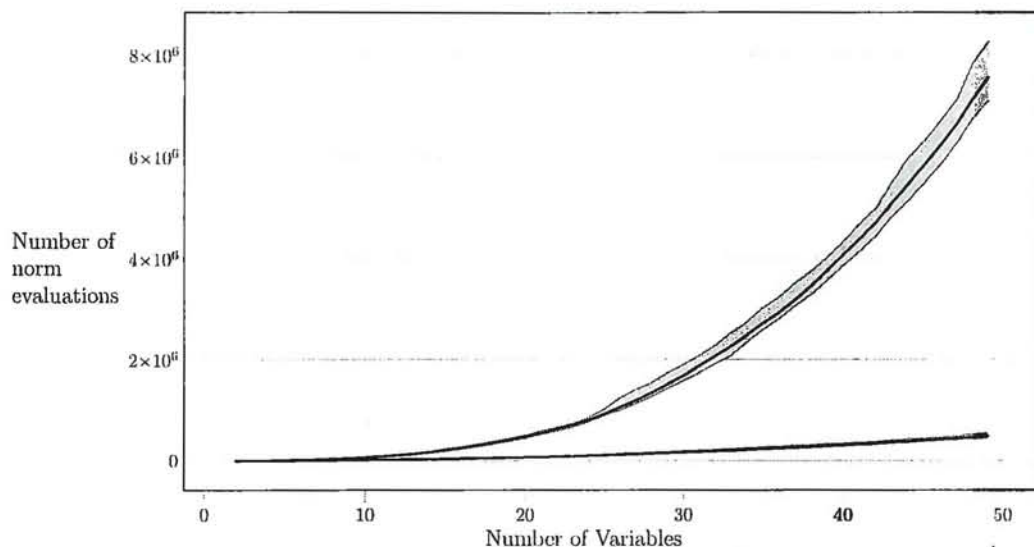


Figure 6.6: Evolution of the computational effort for TSPLIB 3038 dataset

”Multi start” option in CIAO-GO means that the program starts from the initial point chosen by a user and also generates 3 additional initial points. The final result is the best obtained result. The additional initial points are generated by CIAO-GO from the corresponding feasible region.

As a global optimisation technique we use the General Algebraic Modelling System (GAMS), see [79] for more information. We use the Lipschitz global optimiser (LGO) solver [158] from Pinter Consulting Services [159].

6.4.2 Minimisation of the generalised cluster function

In this section we discuss applications of DG, DG+CAM and the LGO solver for minimisation of generalised cluster functions. We propose several approaches for selecting initial points.

Remark 21: Unfortunately, for the ε -cleaning procedure, it is very difficult to know a priori the value for ε which allows one to remove a certain proportion of observations. In our experiments we had to try several values for ε before we found suitable ones.

Initial points

Consider a set $A \subset \mathbb{R}^n$ that contains N points. Assume that we want to find k clusters in A . In this case an initial point is a vector $x \in \mathbb{R}^{n \times k}$. The

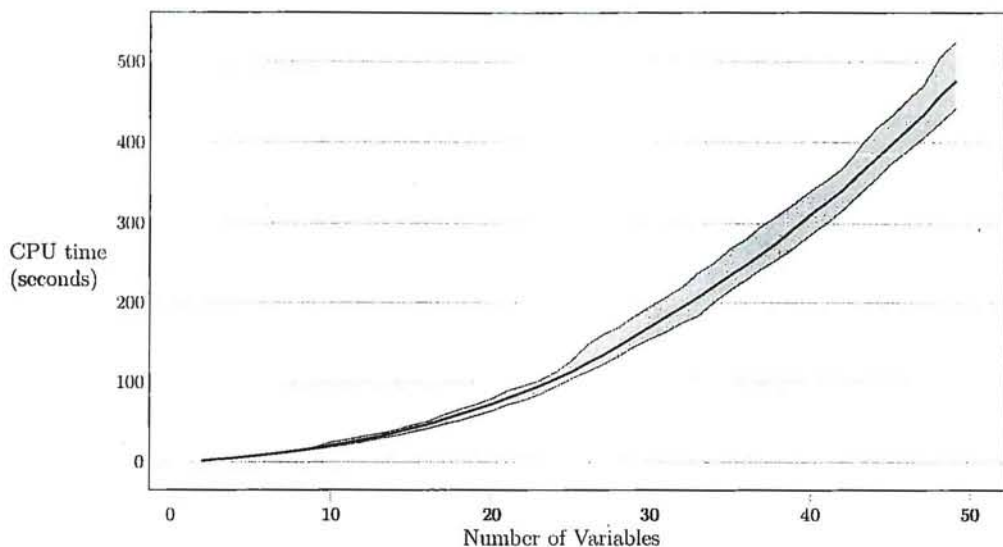


Figure 6.7: Improvement in the computational effort for TSPLIB 3038 dataset

structure of the problem under consideration leads to different approaches to the choice of initial points. We suggest the following four approaches.

k -means L_n initial point The k -means L_n method is a variation of the k -methods (see section 1.2.3 on page 15), which makes use of the speed of the k -means version. In order to obtain a solution quickly, for norm L_n , the barycentre of the points is found instead of step 4 of algorithm 1.1. This method is faster than the classical k -methods (although not so effective), and can be seen as a quick way to improve a random initial solution. The k -means L_n method is represented in algorithm 6.3.

We apply this algorithm on the original dataset A and then the result point $x \in \mathbb{R}^{n \times k}$ is considered as an initial point for minimisation of the generalised cluster function generated by the dataset B .

Uniform initial point In our experiments we apply a scaling procedure before solving the problems. The selected method is the mean scaling described in subsection 1.4.1 on page 36. This means that for each feature, the average value over the whole dataset is 1. In such a case we can choose the point $x = (1, 1, \dots, 1) \in \mathbb{R}^{n \times k}$, as initial guess. We shall call it the *uniform initial point*.

Ordered initial point Recall that w_j indicates the cardinality of the set of points $A_{bj} \in A$, which are represented by a point $b^j \in B$ (see sec-

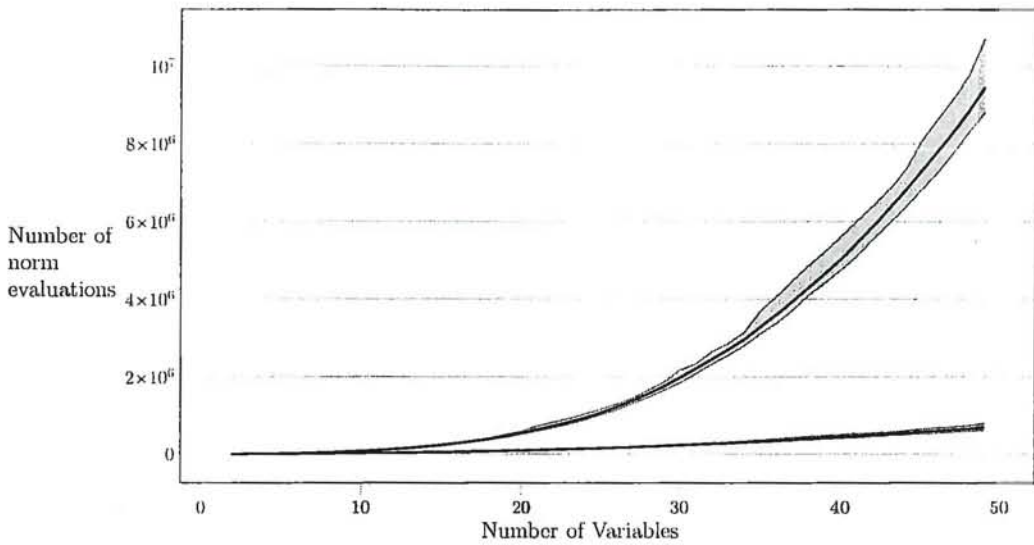


Figure 6.8: Evolution of the computational effort for TSPLIB 1060 dataset

tion 1.4.3 on page 39. It is natural to consider the collection of the heaviest k points as an initial vector for the minimisation of generalised cluster function \tilde{f} . To formalise this, we rearrange the points so that the numbers $w_j, j = 1, \dots, m_B$ decrease and take the first k points from this rearranged dataset. Thus, in order to construct an initial point we choose the k observations with the largest values for weights w_j from the dataset B .

Uniform-ordered initial point This initial point is a combination of the

Algorithm 6.3: $K\text{means}L_n$: a quick variation of the K -methods

Step 1 Initialisation:

Select an initial solution x_1^0, \dots, x_q^0 , set $i=0$

repeat

Step 2 $i \leftarrow i + 1$

Step 3 Cluster assignment:

Find the clusters A_1^i, \dots, A_q^i such that for any $1 \leq l \leq q$,

$$A_l^i = \{a \in A : d(x_l^{i-1}, a) \leq d(x_p^{i-1}, a), \forall 1 \leq p \leq q\}.$$

Step 4 Cluster update:

For each cluster find the barycentre.

until $\exists j \in \{1, \dots, i-1\} : \{A_1^i, \dots, A_q^i\} = \{A_1^j, \dots, A_q^j\}$

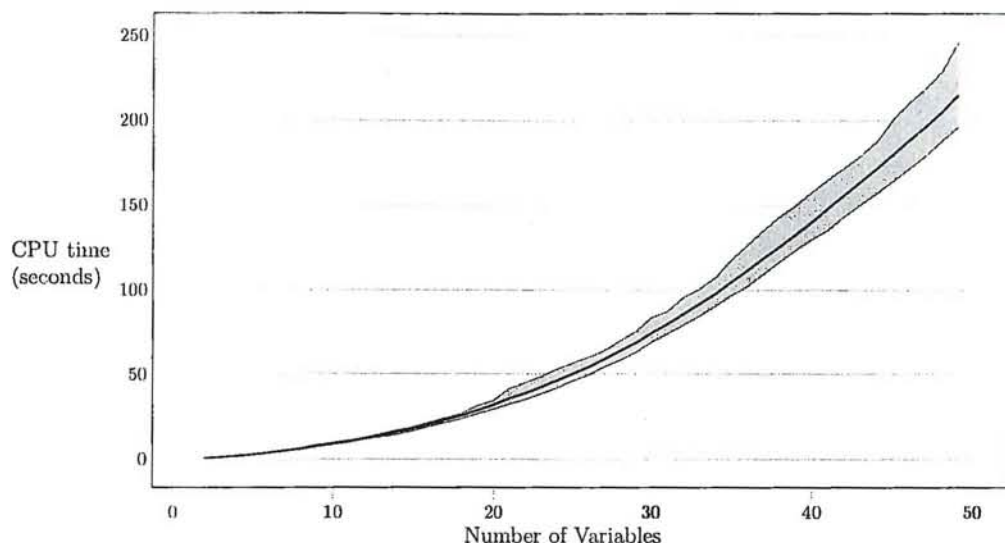


Figure 6.9: Improvement in the computational effort for TSPLIB 1060 dataset

Uniform and the Ordered initial points. It contains the heaviest $k - 1$ observations and the barycentre of the set (each coordinate is 1).

6.4.3 Numerical experiments with the generalised cluster function

For the numerical experiments we use two types of datasets, namely the original dataset A and a small dataset B obtained by the procedure described in section 1.4.3. We compare results obtained for B with the results obtained for the entire original dataset A .

Datasets

The numerical experiments were carried out on the Letters dataset and the Pendigits dataset.

Both Letters and Pendigits datasets have been used for testing different methods of supervised classification (see [145] for details). Since we use these datasets only for construction of generalised cluster function, we consider them as datasets with unknown classes.

Numerical experiments: description

We are looking for three and four clusters in both Letters and Pendigits datasets. For both datasets, the dimension of the optimisation problems is equal to 48 in the case of 3 clusters and 64 in the case of 4 clusters. We consider two small sub-databases of the Letters dataset (Let1, 353 points, approximately 2% of the original dataset; and Let2, 810 points, approximately 4% of the original dataset) and two small sub-sets of the Pendigits dataset (Pen1, 216 points, approximately 2% of the original dataset; and Pen2, 426 points, approximately 4% of the original dataset).

We apply local techniques (the discrete gradient method and a hybrid (described in section 2.7.2 on page 65) between the discrete gradient and cutting angle methods) and a global technique (LGO solver) to the minimisation of the generalised cluster function. Then we need to estimate the results obtained. We can use different approaches for this estimation. One of them is based on the comparison of the values of the cluster function constructed with respect to the centres obtained in the original dataset A and with respect to the centres obtained in its small sub-dataset B . We compare the cluster function values starting from different initial points in original datasets and their approximations, using the following procedure.

Let A be an original dataset and B be its small sub-dataset. First, the centres of clusters in B should be found by an optimisation technique. Then we evaluate the cluster function values in A using the obtained points as the centres of clusters in A . Using this approach we can find out how the results of the minimisation depend on initial points and how far we can go in the process of dataset reduction.

Results for the local optimisation methods

First of all we have to point out that we have two groups of initial points

- Group 1: Uniform initial point and k -means L_1 initial point,
- Group 2: Ordered initial point and Uniform-ordered initial point.

Initial points from Group 1 are the same for an original dataset and for all its reduced versions. Initial points from Group 2 are constructed according to their weights. Points in original datasets have the same weights which are equal to 1.

Remark 22: Because the weights can vary for different reductions of the dataset, the Ordered initial points for Let1 and Let2 do not necessarily coincide. The same is true for the Uniform-ordered initial points. The same

observation applies to the Pendigits dataset and its reduced versions Pen1 and Pen2.

Our next step is to compare results obtained starting from different initial points in the original datasets and in their approximations. In our experiments we minimise the generalised cluster function. This function coincides with the cluster function for original datasets since each point has the same weight which is equal to 1. In the case of reduced datasets we produce our numerical experiments with the corresponding approximations of the original datasets and calculate two different values: the cluster function value and the generalised function value. The **cluster function value** is the value of the cluster function calculated in the corresponding original dataset according to the centres found in the reduced dataset. The **generalised cluster function value** is the value of the generalised cluster function calculated in the reduced dataset according to the centres found in the same reduced dataset. Normally a cluster function value (calculated according to the centres found for reduced datasets) is larger than a generalised cluster function value calculated at the same centres and the corresponding weights, because optimisation techniques have been actually applied to minimise the generalised cluster in the corresponding reduced dataset. In Table 6.5-Table 6.8 we present the results of our numerical experiments obtained for DG and DG+CAM starting from the Uniform initial point.

It is also very important to remember that *a better result in a reduced dataset is not necessarily better for the original one*. For example, in the case of the Pen1 dataset, 3 clusters, the Uniform initial point the generalised function value is lower for DG+CAM than for DG, however the cluster function value is lower for DG than for DG+CAM. We observe the same situation in some other examples.

Dataset	Size	Cluster function value	Generalised cluster function value
Pen1	216	6.4225	5.5547
Pen2	426	6.3844	5.8132
Pendigits	10992	6.3426	6.3426
Let1	353	4.3059	3.3859
Let2	810	4.2826	3.7065
Letters	20000	4.2494	4.2494

Table 6.5: Results for DG, uniform initial point, 3 clusters

Dataset	Size	Cluster function value	Generalised cluster function value
Pen1	216	5.7962	4.8362
Pen2	426	5.7725	5.0931
Pendigits	10992	5.7218	5.7218
Let1	353	4.1200	3.1611
Let2	810	4.0906	3.5040
Letters	20000	4.0695	4.0695

Table 6.6: Results for DG, uniform initial point, 4 clusters

Dataset	Size	Cluster function value	Generalised cluster function value
Pen1	216	6.4254	5.5546
Pen2	426	6.3843	5.8131
Pendigits	10992	6.3426	6.3426
Let1	353	4.3059	3.3859
Let2	810	4.2828	3.7061
Letters	20000	4.2494	4.2494

Table 6.7: Results for DG+CAM, uniform initial point, 3 clusters

Our actual goal is to find clusters in the original datasets, therefore it is important to compare *cluster function values calculated in original datasets* according to obtained centres. Centres can be obtained from our numerical experiments with both types of datasets: original datasets and reduced datasets. It is one of the possible ways to test the efficiency of the proposed approach: substitution of original datasets by their smaller approximations.

Tables 6.9-6.14 represent cluster function values obtained in our numerical experiments starting from the k -means L_1 , Ordered and Uniform-ordered initial point. We do not present the obtained generalised function values because this function can not be used as a measure of the quality of clustering.

Recall that reduced datasets are approximations of corresponding original datasets. Decreasing the number of observations we reduce the complexity of our optimisation problems but obtain less precise approximations. Therefore, our goal is to find a balance between the reduction of the complexity of optimisation problems and the quality of obtained results. In some cases

Dataset	Size	Cluster function value	Generalised cluster function value
Pen1	216	5.7943	4.8353
Pen2	426	5.7718	5.0931
Pendigits	10992	5.7218	5.7218
Let1	353	4.1208	3.1600
Let2	810	4.0909	3.5020
Letters	20000	4.0695	4.0695

Table 6.8: Results for DG+CAM, uniform initial point, 4 clusters

Dataset	Size	Cluster function value	Cluster function value
		3 clusters	4 clusters
Pen1	216	6.4272	5.8063
Pen2	426	6.3840	5.7704
Pendigits	10992	6.3409	5.7217
Let1	353	4.3087	4.1241
Let2	810	4.2816	4.1013
Letters	20000	4.2495	4.0726

Table 6.9: Cluster function: DG, k -means L_1 initial point

(mostly initial point from Group 2, see Remark 22 for more information) the results obtained on larger approximations of original datasets (more precise approximations) are worse than the results obtained on smaller approximations of original datasets (less precise approximations). For example, Pen1 and Pen2 for initial point from Group 2 (3 and 4 clusters).

Summarising the results of the numerical experiments (cluster function, local and improved local techniques, 4 special kinds of initial points) we can draw out the following conclusions

1. DG and DG+CAM applied to the same datasets produce almost identical results if initial points are the same,
2. DG and DG+CAM applied to the same datasets starting from different initial points (4 proposed initial points) produce very similar results in most of the examples,
3. in some cases the results obtained on smaller approximations of original

Dataset	Size	Cluster function value	
		3 clusters	4 clusters
Pen1	216	6.4278	5.8063
Pen2	426	6.3841	5.7723
Pendigits	10992	6.3409	5.7217
Let1	353	4.3087	4.1262
Let2	810	4.2824	4.1014
Letters	20000	4.2495	4.0726

Table 6.10: Cluster function: DG+CAM, k -means L_1 initial point

Dataset	Size	Cluster function value	
		3 clusters	4 clusters
Pen1	216	6.4188	5.8226
Pen2	426	6.6534	5.9047
Let1	353	4.3228	4.2049
Let2	810	4.3843	4.1112

Table 6.11: Cluster function: DG, ordered initial point

Dataset	Size	Cluster function value	
		3 clusters	4 clusters
Pen1	216	6.4171	5.8201
Pen2	426	6.6536	5.9047
Let1	353	4.3228	4.2045
Let2	810	4.3843	4.1107

Table 6.12: Cluster function: DG+CAM, ordered initial point

Dataset	Size	Cluster function value	Cluster function value
		3 clusters	4 clusters
Pen1	216	6.4188	5.7921
Pen2	426	6.6514	5.8718
Let1	353	4.2910	4.1225
Let2	810	4.2828	4.1129

Table 6.13: Cluster function: DG, uniform-ordered initial point

Dataset	Size	Cluster function value	Cluster function value
		3 clusters	4 clusters
Pen1	216	6.4171	5.7945
Pen2	426	6.6492	5.8715
Let1	353	4.2905	4.1233
Let2	810	4.2828	4.1130

Table 6.14: Cluster function: DG+CAM, uniform-ordered initial point

datasets are better than the results obtained on larger approximations of original datasets.

Results for the global optimisation LGO solver

Now we present the results obtained by the LGO solver (global optimisation). We use the Uniform initial point. The results are in Table 6.15.

In almost all the cases (except Pendigits 3 clusters) the results for reduced datasets are better than for original datasets. It means that the cluster function is too complicated for the solver as an objective function and it is more efficient to use generalised cluster functions generated on reduced datasets. It is beneficial to use reduced datasets in the case of the LGO solver from two points of view:

1. computations with reduced datasets allow one to reach a better minimiser;
2. computational time is significantly less for reduced datasets than for original datasets.

Dataset	Size	Cluster function value	Cluster function value
		3 clusters	4 clusters
Pen1	216	6.4370	5.8029
Pen2	426	6.4122	5.7800
Pendigits	10992	6.3426	7.1859
Let1	353	4.3076	4.1426
Let2	810	4.2829	4.1191
Letters	20000	5.8638	4.2064

Table 6.15: Cluster function: LGO solver

It is also obvious that the software failed to reach a global minimum. We suggest that the LGO solver has been developed for a broad class of optimisation problems. However, the solvers included in CIAO-GO are more efficient for minimisation of the sum of minima of convex functions, especially if the number of components in sums is large. In all the experiments, a limit of two hours of computational time has been given. This software may have given a better solution if given more time. Nevertheless, the local algorithms starting from well chosen initial points are much more efficient.

6.4.4 Skeletons

Introduction

The problem of grouping (clustering) points by means of skeletons is not so widely studied as it is in the case of cluster function based models. Therefore, we would like to start with some examples produced in not very large datasets (no more than 1000 observations). In this subsection we formulate the problems of finding skeletons mathematically, discuss applications of the discrete gradient method and the hybrid method described in section 2.7.1 between the discrete gradient method and the simulated annealing method to finding skeletons with respect to $\|\cdot\|_1$ and give graphical implementations of the obtained results (for examples with no more than 3 features).

Skeleton of a finite set of points

We now consider a version of Bradley-Mangasarian function (see remark 5 on page 15), where the distances to hyperplanes are used instead of the squares of these distances. Assume that \mathbb{R}^n is equipped with a norm $\|\cdot\|$. Let A be a finite set of points. Consider vectors l_1, \dots, l_k with $\|l_i\|_* \equiv \max_{\|x\|=1} \langle l_i, x \rangle = 1$

and numbers c_i ($i = 1, \dots, k$). Let $H_i = \{x : \langle l_i, x \rangle = c_i\}$ and $H = \cup_i H_i$. Then the distance between the set H_i and a point a is $d(a, H_i) = |\langle l_i, a \rangle - c_i|$ and the distance between the set H and a is

$$d(a, H) = \min_i |\langle l_i, a \rangle - c_i|. \quad (6.4)$$

The deviation of X from A is

$$\sum_{a \in A} d(a, H) \equiv \sum_{a \in A} \min_i |\langle l_i, a \rangle - c_i|.$$

The function

$$L_k((l_1, c_1), \dots, (l_k, c_k)) = \sum_{a \in A} \min_i |\langle l_i, a \rangle - c_i| \quad (6.5)$$

is of the form (6.1). Consider the following constrained *min-sum-min* problem

$$\min \sum_{a \in A} \min_i |\langle l_i, a \rangle - c_i| \text{ subject to } \|l_j\| = 1, \quad c_j \in \mathbb{R} \quad (j = 1, \dots, k) \quad (6.6)$$

A solution of this problem will be called a *k-skeleton* of the set A . The function in (6.6) is called the *skeleton function*.

More precisely, *k-skeleton* is the union of k hyperplanes $\{x : \langle l_i, x \rangle = c_i\}$, where $((l_1, c_1), \dots, (l_k, c_k))$ is a solution of (6.6). *If the skeletons are known, each point is assigned to the cluster with the nearest skeleton.* It is difficult to find a global minimiser of (6.6), so sometimes we can consider the union of hyperplanes that is formed by a local solution of (6.6) as a skeleton.

Clusters constructed according to skeletons, obtained as a result of the skeleton function minimisation are called *skeleton-based clusters*.

The concept of *shape* of a finite set of points will be introduced and studied in chapter 9 on page 165. By definition, the shape is a minimal (in a certain sense) ellipsoid, which contains the given set. A technique to find an ellipsoidal shape is proposed in this chapter. In many instances the geometric characterisation of a set A can be viewed as the intersection between its shape, describing its external boundary, and its skeleton, describing its internal aspect.

Preliminary experiments

The search for skeletons can be done by solving the constrained minimisation problem (6.6).

Both algorithms are designed for solving unconstrained problems so we use a penalty function in order to convert problem (6.6) to an unconstrained minimisation one. The corresponding unconstrained problem has the form:

$$\begin{aligned} \text{minimise } & \sum_{q \in Q} \min_i |\langle l_i, a^q \rangle - b_i| + R_p \sum_{i=1}^k |||l_i||_1 - 1|, \\ & \text{subject to} \\ & (l_1, b_1) \in \mathbb{R}^{n+1}, \dots, (l_k, b_k) \in \mathbb{R}^{n+1} \end{aligned} \quad (6.7)$$

where R_p is a penalty parameter.

Finally, the algorithms were applied starting from 3 different initial points, and the best solution found was selected. The 3 different points used in the example are:

- $P_1 = \begin{cases} l_i = \frac{1}{\sqrt{N}}(1, \dots, 1) \\ b_i = 1 \end{cases}$
- $P_2 = \begin{cases} l_i = (1, 0, \dots, 0) \\ b_i = 1 \end{cases}$
- $P_3 = \begin{cases} l_i = \frac{1}{\sqrt{N-1}}(0, 1, \dots, 1) \\ b_i = 1 \end{cases}$

The problem has been solved for different sets of points, selected from 3 different well known datasets: the Heart disease database, the Diabetes database and the Australian credit cards database, see also [145] and references therein. Each of these datasets was submitted first to the feature selection method described in [25].

The value of the objective function was considerably decreased by both methods (see tables 6.16 and 6.17). However, the discrete gradient method often gives a local solution which is very close to the initial point, while the hybrid gives a solution which is further and better. In the tables the distance considered is the Euclidean distance between the solution obtained and the initial solution, and the value considered is the value of the objective function at this solution.

The different examples show that although sometimes the hybrid method does not improve the result obtained with the discrete gradient method, in some other cases the result obtained is much better than when the discrete gradient method is used. However the computations times it induces are much greater than the simple use of the discrete gradient method. The diabetes dataset has 3 features, after feature selection (see [25]). This allows us to plot graphically some of the results obtained during the computations.

	Initial point	DG method		hybrid method	
		value	distance	value	distance
Class 1	1	22.9804	10.668	6.11298	7.98738
	2	25.5102	2.81543	13.2263	5.91397
	3	6.10334	4.40741	6.10334	4.40741
Class 2	1	0.473317	5.00549	0.473317	5.00549
	2	3.029	2.14784	0.222154	2.13944
	3	6.87897	6.06736	4.73828	6.74424
computation time		54 sec		664 sec	

Table 6.16: Australian credit card database with 2 hyperplanes skeletons

	Initial point	DG method		hybrid method	
		value	distance	value	distance
Class 1	1	28.5856	6.78624	28.1024	6.79326
	2	39.3925	11.4668	28.2417	11.7711
	3	33.2006	3.09434	31.4624	2.31922
Class 2	1	22.2806	2.3755	22.2806	2.3755
	2	30.346	56.7222	19.5574	8.76914
	3	23.0529	1.61649	22.9495	1.76052
computation time		212 sec		1521 sec	

Table 6.17: Diabetes database with 3 hyperplanes skeletons

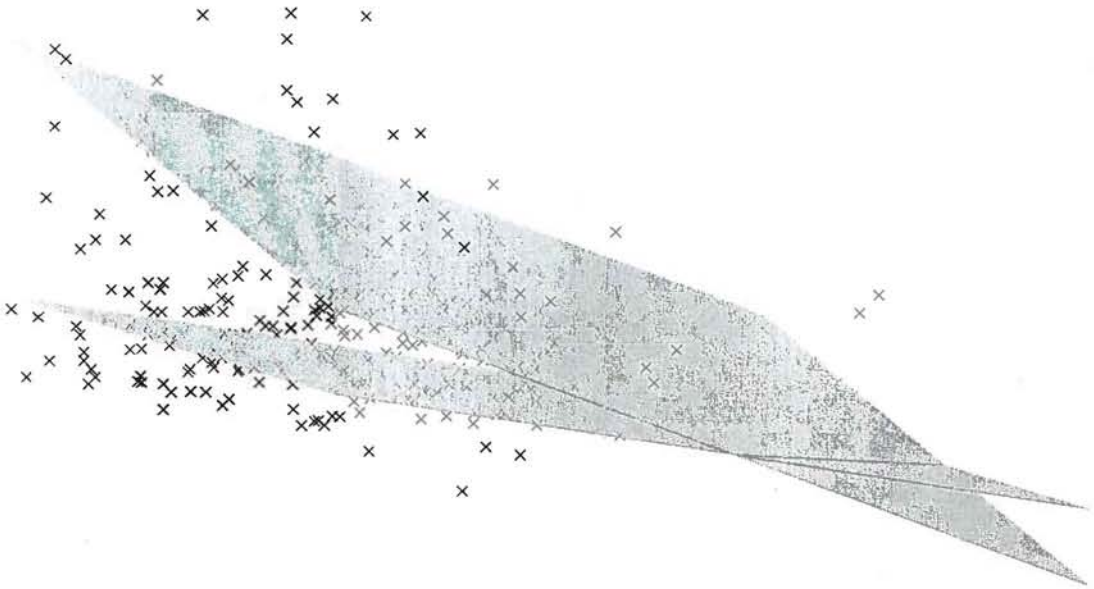


Figure 6.10: 2nd class for the diabetes database, with 2 hyperplanes

We can observe that the hybrid method does not necessarily give an optimal solution. Even with the hybrid method the initial point is very important. The figures 6.10 and 6.11, however, confirm that the solutions obtained are usually very good, and seem to represent correctly the set of points. The set of points studied here is constituted by a big mass of points, and some other points spread around. It is interesting to remark that the hyperplanes intersect around the same place - where the big mass is situated - and take different directions, to be the closest possible to the spread points. This is particularly clear in the case of three hyperplanes, on figure 6.11.

Numerical experiments with large datasets: description

We are looking for three and four clusters in both Letters and Pendigits datasets. The dimension of the optimisation problems is equal to 51 in the case of 3 skeletons and 68 in the case of 4 skeletons. We use the same sub-datasets as in section 6.4.3 (Pen1, Pen2, Let1, Let2).

Remark 23: It is possible to construct the *generalised skeleton function* by using the same idea as for the generalised cluster function.

We apply local techniques (DG and DG+CAM) for minimisation of the generalised skeleton function. Then we use a procedure which is similar to the one we used for the cluster function to estimate the obtained results.

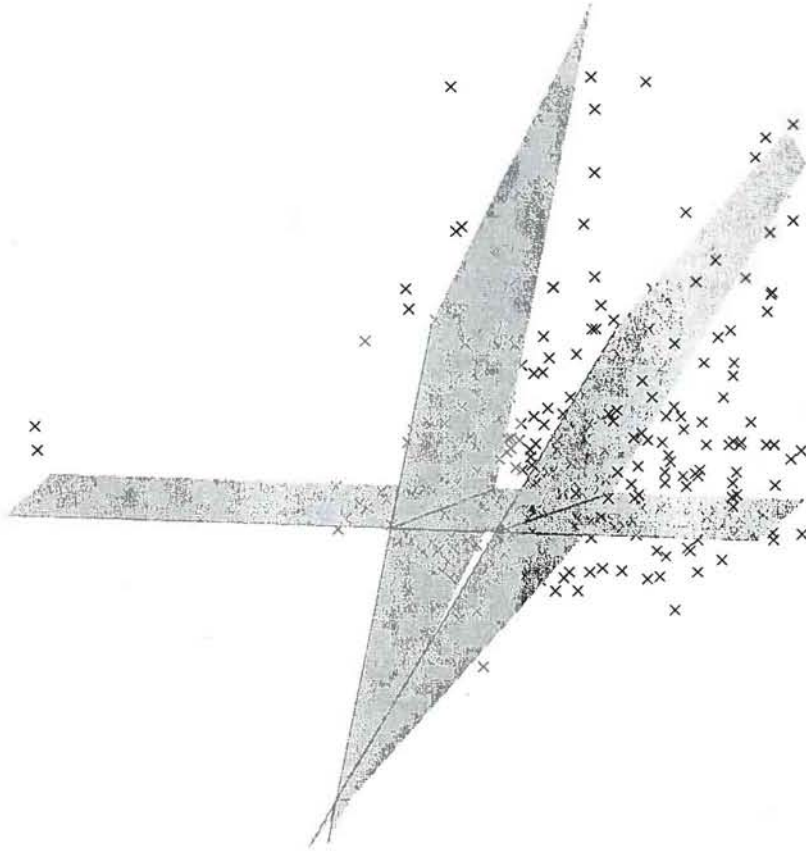


Figure 6.11: 2^{nd} class of the diabetes database, with 3 hyperplanes

N_s	Dataset	Size	Skeleton function values			
			DG	DG multi	DG+CAM	DG+CAM multi
3	Pen1	216	2137.00	1287.58	1832.97	1320.00
	Pen2	426	735.00	735.47	735.47	735.47
	Pendigits	10992	567.20	567.20	567.20	566.55
4	Pen1	216	1223.16	1315.68	1194.65	1180.79
	Pen2	426	1360.16	946.74	1322.46	946.74
	Pendigits	10992	905.56	905.56	905.56	661.84

Table 6.18: Skeleton function: pendigits

First, we find skeletons in original datasets (or in reduced datasets). Then we evaluate the skeleton function values in original datasets using the obtained skeletons.

For the skeleton function the problem of constructing a good initial point has not been studied yet. Therefore, in our numerical experiments as an initial point we choose a feasible point. We also use the "multi start" option to compare results obtained starting from different initial points.

Numerical experiments with large datasets: results

In this subsection we present the results obtained for the skeleton function. Our goal is to find the centres in original datasets, therefore we do not present the generalised skeleton function values. Table 6.18 and Table 6.19 present the values of the skeleton function evaluated in the corresponding original datasets (Pendigits and Letters respectively) according to the skeletons obtained as optimisation results reached in datasets from the first column of the tables. We use two different optimisation methods: DG and DG+CAM and two different types of initial points: "single start" (DG or DG+CAM) and "multi start" (DGMULT or DG+CAMMULT).

The most important conclusion to the results is that in the case of the skeleton function the best optimisation results (the lowest value of the skeleton function) *have been reached in the experiments with the original datasets*. It means that the *proposed cleaning procedure is not as efficient in the case of skeleton function as it is in the case of the clustering function*. However, in the case of the clustering function the initial points for the optimisation methods have been chosen after some preliminary study. It can happen that an efficient choice of initial points leads to better optimisation results for both kinds of datasets: original and reduced.

N_s	Dataset	Size	Skeleton function values			
			DG	DG multi	DG+CAM	DG+CAM multi
3	Let1	353	1548.30	1548.30	1545.58	1545.58
	Let2	810	2201.75	1475.77	2171.01	1608.14
	Letters	20000	1904.71	1904.71	1904.71	964.37
4	Let1	353	1566.69	1566.69	1531.99	1531.99
	Let2	810	2030.20	2030.20	1892.31	1892.31
	Letters	20000	964.37	850.14	850.14	850.14

Table 6.19: Skeleton function: letters

Recall that (6.6) is a constrained optimisation problem with equality constraints. This problem is equivalent to the following constrained optimisation problem with inequality constraints

$$\begin{aligned}
 & \text{minimise } \sum_{a \in A} \min_i |\langle l_i, a \rangle - c_i| \\
 & \text{subject to} \\
 & \|l_j\| \geq 1 \\
 & c_j \in \mathbb{R} \quad (j = 1, \dots, k).
 \end{aligned} \tag{6.8}$$

In our numerical experiments we use both formulations (6.6) and (6.8). In most of the experiments the results obtained for (6.6) are better than for (6.8) but computational time is much higher for (6.6) than for (6.8). It is recommended, however, to use the formulation (6.8) if, for example, experiments with (6.6) produce empty clusters.

Other experiments

The hybrid method described in section 2.7.1 on page 64 was applied on both the generalised cluster function and the skeleton function. This method is based on improvements of the local minima reached by the discrete gradient method by the simulated annealing. Therefore it acts as a (non exhaustive) listing of local minima.

In both cases, the same behaviour was observed: the results show that the hybrid method only reaches results comparable with the local method, yet the algorithm had to leave up to 50 local minima. This can only be explained by a large amount of local minima in the objective function, each close to one another.

6.5 Conclusion

6.5.1 Optimisation

In this chapter, two different methods have been developed to solve the clustering problem. This problem can depend on a very large number of variables, hence only a few methods can be applied. One of them is an adapted version of the discrete gradient method presented in section 4.3. Because this method is a local one, it is necessary to design techniques to generate good starting points.

The first method proposed is an incremental one: clusters are added one by one until a stopping criterion verifying that the result is satisfactory is met. This method has been tested on real world datasets, and results show that it can reach a good solution within a reasonable CPU time. The results also show that the simplified scheme allows one to improve significantly the efficiency of the solver.

The other method proposed is more focused on the speed. When speed is a strong issue, it is often necessary to work on a dataset processed through ϵ -cleaning. four initial points for the discrete gradient method have been proposed. For comparison, numerical experiments have been carried out using commercial softwares LGO and CIAO-GO.

The LGO software failed most of the time to reach even a good solution. This is due to the fact that the objective function has a very complex structure. This method was limited in time, and may have reached the global solution, had it been given a limitless amount of time. Nevertheless, it shows that global solvers cannot reach a satisfactory solution within an acceptable time.

Contrariwise, the local method reached a good solution for at least one of the initial points. The hybrid method was able to reach a good solution for all the initial points. For comparison, both these methods were run starting from randomly chosen initial points, giving unsatisfactory results.

This shows that for such types of functions, presenting a complex structure and many local minima, most global methods will fail. However, well chosen initial points will lead to a deep local minimum. Because the local methods are much faster than global ones, it is more advantageous to start the local method from a set of carefully chosen initial points to reach a global minimum.

The application of the combination between the discrete gradient and the cutting angle methods appears to be a good alternative, as it is not very dependant on the initial point, while reaching a good solution in a limited time.

The second experiment was carried out over the skeleton function. This function having been less studied in the literature, it is harder to draw definite conclusions. However, the experiments show very clearly that the local methods once again strongly depend on the initial point. Unfortunately it is harder to devise a good initial point for this objective function.

6.5.2 Clustering

From the clustering point of view, two different similarity functions have been minimised. The first one is a variation of the widely studied cluster function, where the points are weighted. The second one is a variation of the Bradley-Mangasarian function, where distances from the hyperplanes are taken instead of their square.

The ε -cleaning procedure has been tested on various datasets. Numerical experiments have been carried out for different values of epsilon, leading to very small (2% and 4% of the original size) datasets.

For the generalised cluster function, this method proves to be very successful: even for very small datasets, the function value obtained is very satisfactory. When the method was solved using the global method LGO, the results obtained for the reduced dataset were almost always better than those obtained for the original dataset. The reason is that the larger the dataset, the larger amount of local minima for the objective function. When the dataset is reduced, what is lost in measurement quality is gained by the strong simplification of the function. Because each point in the reduced dataset acts already as a centre for its neighbourhood, minimising the generalised cluster function is equivalent to group these "mini" clusters into larger clusters.

It has to be noted that there is not a monotone correspondence between the value of the generalised cluster function for the reduced and the original dataset. It may happen that a given solution is better than another one for the reduced dataset, and worse for the original. Thus we cannot conclude that the solution can be reached for the reduced dataset. However, the experiments show that the solution found for the reduced dataset is always good.

For the skeletons function, however, this method is not so successful. Although this has to be taken with precautions, as the initial points for this function could not be devised so carefully as for the cluster function, one can expect such behaviour: the reduced dataset is actually a set of tiny cluster centres. The skeleton approach is based on the assumption that the clusters in the dataset can be represented by hyperplanes, while the cluster approach assumes that the clusters are represented by centres.

The experiments show the significance of the choice of the initial point to reach good clusters. While random points did not allow any method to reach a good solution, all initial points selected upon the structure of the dataset lead the combination DG-CAM to the solution, while incrementally constructed ones were enabling the local method to reach a good solution.

Since for the cluster function we are able to provide some good initial points, but not for the skeleton function, unless the structure of the dataset is known to correspond to some skeletons, we would recommend to use the centre approach.

Finally the comparison between the results obtained by the two different methods has to be relativised: experiments having shown the importance of initial points, it is difficult to draw definitive conclusions from the results obtained for the skeleton approach.

Chapter 7

A feature selection algorithm

7.1 Introduction

Among various preprocessing tasks, the selection of relevant features is a particularly important problem, because it is goal dependent: a feature may be significant when the goal is to find clusters, but be noisy when one wants to classify the data into known classes. As a rule, when the data is collected, the choice is to be as exhaustive as possible, thereby creating datasets with a large number of features, many of which will probably be irrelevant for a given task. Not only a useless feature complicates the problem by adding more parameters and variables, and slowing down the programs, but the quality of the solution obtained may also suffer from the noise introduced by these features in the data. The feature selection problem may also be cost-orientated: often some measurement are very expensive, and it may be interesting to get, from a first small set of data, an evaluation of which features are not needed.

The clustering (unsupervised learning) problem requests finding a small (comparatively with the number of records) number of groups of similar data in a dataset. This problem is a very flexible one, and usually several solutions may be acceptable. This means that applying a feature selection for clustering is particularly challenging: how to carry out a problem-dependent task when the given problem is so imprecise?

In this chapter we present a feature selection algorithm which can be applied for both supervised and unsupervised classification. This approach is based on nonsmooth optimisation and clustering (and its applications to supervised learning, as described in subsection 1.3.7 on page 29). The results of numerical experiments show that the proposed algorithm for feature selection works efficiently on the datasets used in the numerical experiments.

7.2 A feature selection approach

7.2.1 Unsupervised learning

Consider the dataset pictured on figure 7.1. This dataset is constituted of two almost distinct sets of points. If one projects this dataset on each of the features, the results obtained are pictured on figure 7.2. It is quite clear that feature 1 discriminates very well between the groups, while feature 2 is almost inefficient. Therefore it can be deduced that the second feature of this dataset is redundant.

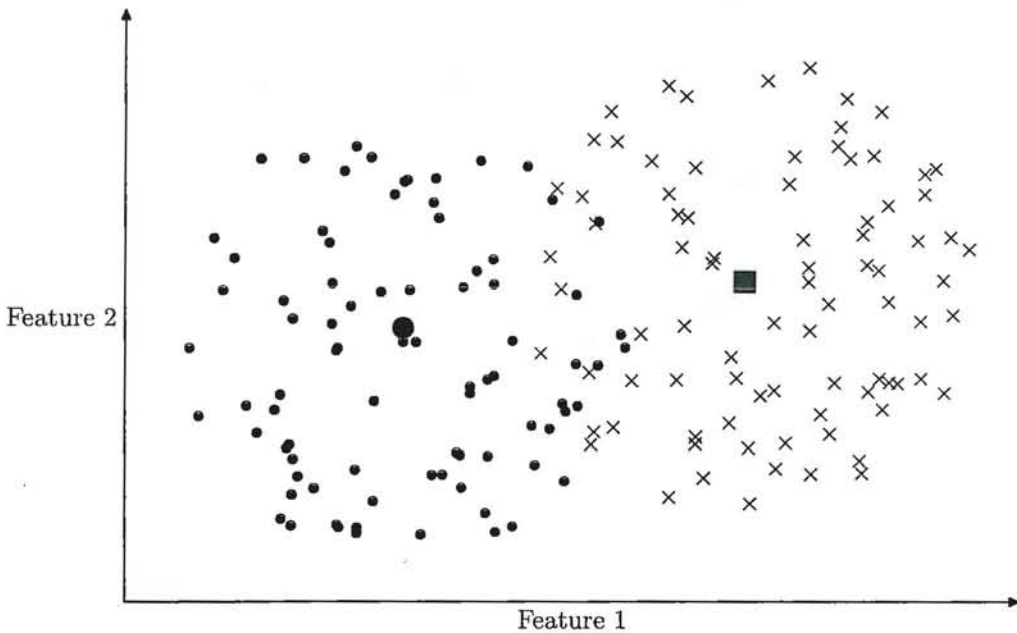


Figure 7.1: A sample dataset containing two almost distinct groups

Comparing the centres of these two sets of points, and their projections, the same deduction can be made: while the two centres have almost the same value for feature 2, their value differs much for feature 1. Therefore we can conclude that the more a coordinate varies between centres, the more significance it has in the dataset.

The basis of this approach is to divide the dataset into small groups by finding their centres, and comparing these centres coordinate-wise. If the variation is small for a given feature, then it is considered that this feature is uninformative.

The feature selection approach presented in this section is based on the idea that the set of centres found during the clustering phase represent the

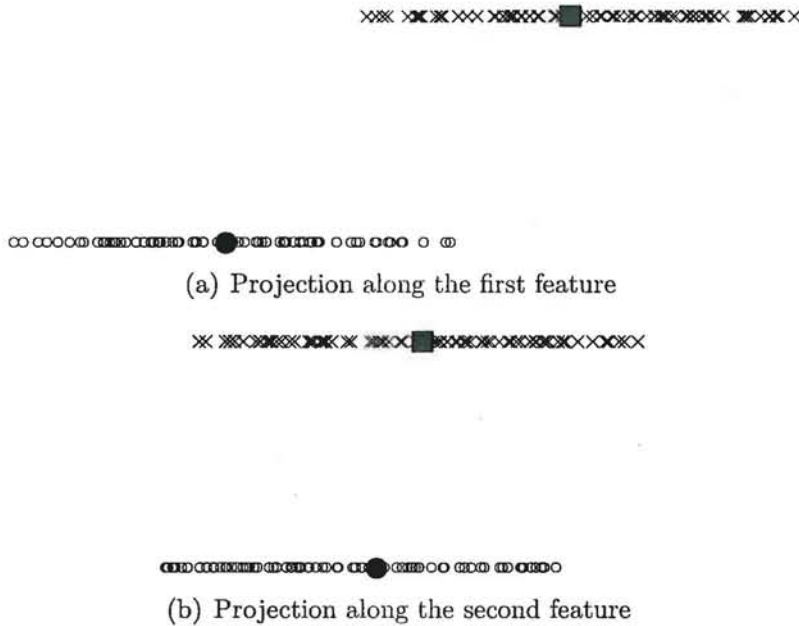


Figure 7.2: Projections of the dataset along its features

sets under consideration. Algorithm 7.1 presents this method.

Algorithm 7.1: Feature selection for unsupervised learning

- Step 1** Divide the dataset in q clusters, by minimising function (1.4) The centres obtained are $\{x^1, \dots, x^q\}$.
- repeat**
- Step 2** Order the features using the following rule:
- $$f_u <_F f_v \Leftrightarrow \max_{1 \leq j \leq q} x_{f_u}^j - \min_{1 \leq i \leq q} x_{f_u}^i > \max_{1 \leq j \leq q} x_{f_v}^j - \min_{1 \leq i \leq q} x_{f_v}^i$$
- Step 3** Remove the “lowest” features according to $<_F$, under a given threshold and divide the dataset in n clusters using the remaining features
until the structure of the clusters has changed beyond a certain tolerance
- Step 4** the latest feature removed, as well as all the remaining ones, are informative.
-

The main step in this algorithm is Step 2. The features are ordered by informativeness. It is considered that if for a given coordinate the centres vary much, the feature is informative. Conversely, if there is little variation for a given coordinate, the feature is uninformative. By evaluating the variation in the coordinates it is possible to order the features by informativeness. In Step 3 it is then verified that by removing the less informative features

the structures of the clusters is not modified. In case of modification, the features are considered as informative.

The number n does not need to be very large. In our experiments we took 2 or 3 clusters, in order to keep the calculations fast enough. Although most datasets contain a larger number of clusters, experiments show that it is sufficient to keep this number low for the feature selection.

7.2.2 Supervised learning

In the case of supervised classification, the algorithm is very similar (see algorithm 7.2). Instead of comparing all the centres, the comparison is carried out only between clusters of different classes, and the stopping criterion is based on the classification accuracy. It can be seen that while the clusters of a same class may have a similar value for a given coordinate, the coordinate may be very different in another class. In that case, the feature is very informative for the classification problem, but useless for the clustering problem.

Algorithm 7.2: Feature selection for supervised learning

Step 1 Divide each class i of the dataset in n clusters, by minimising function (1.4)

The centres obtained are $\{x^{i1}, \dots, x^{in}\}$.

repeat

Step 2 Order the features using the following rule:

$$f_u <_F f_v \Leftrightarrow \max_{\substack{1 \leq j \leq n \\ 1 \leq k \leq n_c}} x_{f_u}^{kj} - \min_{\substack{1 \leq i \leq n \\ l \neq k}} x_{f_u}^{li} > \max_{\substack{1 \leq j \leq n \\ 1 \leq k \leq n_c}} x_{f_v}^{kj} - \min_{\substack{1 \leq i \leq n \\ l \neq k}} x_{f_v}^{li}$$

Step 3 Remove the "lowest" features according to $<_F$, under a given threshold and divide the dataset in n clusters using the remaining features

until *The classification accuracy got worse*

Step 4 the latest feature removed, as well as all the remaining ones, are informative.

7.3 Numerical experiments

7.3.1 Unsupervised learning

In our study we use two test datasets: the Pendigits dataset and the Letters dataset. We use two methods: the discrete gradient method (DG), and a hybrid method between the discrete gradient and the simulated annealing methods (HM) (see subsection 2.7.2 on page 65).

In our numerical experiments DG and HM. For the Pendigits dataset we apply the following techniques to obtain a rough division of the dataset:

1. 2 clusters, reduced dataset (Pen02, 216 points), HM;
2. 3 clusters, original dataset, DG.

There are 2 non-informative features (the same features for both cases): 2, 13.

For the Letters dataset we use the following constructions:

1. 2 clusters, reduced dataset (Let02, 353 points), HM;
2. 3 clusters, original dataset, DG.

In the first case we eliminate 5 features (6, 9, 12, 14, 16), in the second case we eliminate 7 features (6, 9, 10, 11, 12, 14, 16). For this dataset we choose 5 features to eliminate (6, 9, 12, 14, 16).

We apply DG (the initial point is the point obtained by the HM) to the whole dataset after feature elimination and compare the obtained clusters with the results without feature elimination (in this case we remove the coordinates corresponding to the set of the eliminated features).

Dataset	n_c	Features eliminated	Changes of centres (norm $\ \cdot \ _1$ distance)
Pendigits	3	2, 13	.03, .05, .06
10992	4	2, 13	.09, .08, .04, .14
Letters	3	6,9,12,14,16	.62, .34, .07
20000	4	6,9,10,11,12,14,16	.34, .76, .07, .67

Table 7.1: Feature elimination

In Table 7.1 we compare the centres obtained in the datasets after feature elimination and the centres obtained in the original datasets (when the centres in the original datasets have been obtained we remove the coordinates which correspond to the eliminated features).

In the table, the changes of centres are represented in terms of distance (according to 1-norm) between the final centres and the projection of the initial centres over the set of selected features.

We obtain almost the same clusters for the datasets before and after feature elimination. Therefore, we can conclude that the eliminated features are not significant in the sense of clustering and such a procedure of feature elimination for unsupervised classification is quite efficient.

Norm 1	11	7	26	6	16	22	17	20	14	24	32	1	23	10	19	12	2	29	28	25	27	18	3	21	5	4	9	8	31	13	30	15
Norm 2	26	6	1	17	16	11	14	22	24	10	20	2	7	23	19	28	25	27	29	3	32	18	12	21	8	4	13	9	30	5	31	15

Table 7.2: classification after the first step of the algorithm.

7.3.2 Supervised learning

The main interest of the application of this method to supervised learning is for very small datasets, when classical statistical methods are ineffective. In some areas, measuring information is extremely costly, and an algorithm permitting to eliminate from a small sample the measurements that are ineffective is very useful.

The algorithm has been applied on a biomarkers dataset, measuring the presence level of various types of chemical elements inside different environmental bodies (such as grass, soil, etc. . .). The observations are the bodies, while the features are the levels of chemical elements. For reasons of confidentiality, this dataset cannot be published.

The dataset under consideration in this subsection contains 32 features and 29 observations. The dataset is divided into 3 classes (Cow Dung, Grass and Soil), and algorithm 7.2 is applied, for two norms (Euclidean and 1 norms), in order to compare. Because one cluster was found per class, the clustering problem is convex, and therefore the discrete gradient method can reach the global minimum.

By comparing the results for different norms, it is possible to establish levels of information: we can consider a group of features F_1 to be more informative than a group of features F_2 if for all the norms:

$$f_i <_f f_j, \forall f_i \in F_1, \forall f_j \in F_2$$

By dividing the set of features into smaller groups following that rule, it is possible to create levels of information.

It is shown in these results that not only this method is a good feature elimination method, but also it generates a very robust ordering of the features by degree of information.

The tables present the result, exposing the features from the least informative to the most informative. Three different results are presented. Table 7.2 is the first ordering of the features, when no feature has yet been eliminated. Table 7.3 presents the same ordering, after a few iterations of the algorithm. Finally table 7.4 presents the order in which the features have been eliminated.

In table 7.4, for each norm, two features were selected, and the level 1 of

Norm 1	22	20	17	2	32	23	25	1	29	4	28	27	3	21	18	5	9	13	8	30	31	15
Norm 2	11	2	25	23	1	19	28	29	32	12	18	3	27	21	9	30	8	4	13	5	31	15

Table 7.3: intermediate classification

Norm 1	11	6	7	19	26	12	16	24	10	14	20	17	22	23	32	25	29	1	28	21	27	2	18	3	5	4	31	9	13	8
Norm 2	26	17	22	6	11	20	7	16	14	24	10	23	19	1	25	29	27	28	3	12	32	18	21	2	5	13	8	9	31	4

Table 7.4: order of elimination.

information is thus not present in this table. The informative features found by the algorithm are {15,30} for each norm. These features are thus considered as the set of informative features, and be the first level of information.

It is possible to find 5 levels of features:

- Level 1 composed from the selected features {15,30}
- Level 2 composed from the features {4,5,8,9,13,31}
- Level 3 composed from the feature {21}
- Level 4 composed from the features {3,18}
- Level 5 composed from the remaining features.

These results were compared in [50] with another feature selection technique based on a totally different idea, the *FDM*. This methods also divides the features in levels. The features of the first level of the *FDM* are distributed among the levels 1-3 of the presented method. Of these three levels, only one feature (feature 30) is not present in the first level of the *FDM*. This shows that the algorithms obtain very similar results on this dataset.

Remark 24: In the case of the level 4, the rule is not strictly applied, but the two features 3 and 18 seem important enough for each parameter to be considered as forming a 4-th level. A strict application of the rule would lead to only 4 levels, the last one being the union of the levels 4 and 5. Table 1 shows that this definition gives a good idea of the degree of interest each feature can have. One may want to work with only the first 3 levels (9 features instead of 32). The feature selection algorithm can then be applied.

7.4 Conclusion

Several feature selection approaches for supervised classification have been developed recently. These approaches are mostly based on a fact that the

researchers have some information about the class distribution. In the case of unsupervised learning this kind of information is not available and therefore these approaches are not applicable. The area of feature selection for unsupervised learning is not broadly studied.

In this chapter we developed some feature elimination methods for clustering (unsupervised learning). The proposed procedure has been tested on two real-world datasets. The experiments show that the procedure to eliminate noisy features is efficient. This procedure involves application of HM, therefore it requires an appropriate cleaning procedure in the case of large-scale datasets.

The approach presented in this chapter works very well in the case of clustering models based on the minimisation of the cluster function. It is possible that for some other clustering models this approach is not very efficient. Therefore the development of new feature selection approaches for unsupervised learning is very important and needs to be continued.

It is also shown in this chapter that this method works well also in the case of classification, even in extreme cases (very small datasets). It is noticeable that this method is very robust, and does not depend strongly on the norm.

Chapter 8

Study of the relations between clusters and classes

8.1 Introduction

An important characteristic of methods for supervised and unsupervised classification is their accuracy. There are different approaches to *classification accuracy* that can be used for comparison different classifiers (see for example, [145]). We mention here only n -fold cross-validation. This approach is very popular, however it cannot always give a good comparison of classifiers (see [25] for discussion). For example, the classification accuracy obtained for the same dataset in the case of n_1 -fold cross-validation and n_2 -fold cross-validation are not necessarily the same if $n_1 \neq n_2$. The estimation of accuracy of clustering methods is much more difficult than classification methods.

Due to the more fuzzy character of clustering, devising a measure for the quality is difficult. In section 1.2.6 on page 19, two techniques were proposed. One is based on the structure of the clusters, measuring how “deep” points are inside the clusters, while the other one is evaluating the clusters according to the “purity”, that is the proportion of the most represented class in the cluster. In this chapter we show that comparison of classes and clusters is not always appropriate, hence this comparison cannot be used for assessment of a clustering technique. Indeed, it is possible that the points have been grouped by this technique according to some other characteristic rather than the classes. A simple and interesting example of such a case can be found in [104].

In this chapter we show that a similar situation can appear in a real-world datasets. We consider two real-world datasets with classes (“Pendigits” and “Letters”) and compare classes and clusters for these datasets.

	1	2	3	4	5	6	7	8	9	10	size	% class
1	470	0	0	0	0	107	2	1	1	0	581	80.89
2	1	1122	334	5	1	1	0	0	6	1	1471	76.27
3	0	15	634	26	0	13	0	123	146	31	988	64.17
4	0	0	2	1051	5	28	0	79	0	1	1166	90.13
5	11	0	1	29	1049	3	1	0	1	0	1095	95.79
6	247	0	0	0	0	989	0	12	0	0	1248	79.24
7	183	0	0	0	1	0	625	0	3	0	812	76.97
8	8	0	82	32	0	2	191	715	0	2	1032	69.28
9	69	2	1	0	0	0	0	0	962	0	1034	93.03
10	66	5	89	1	0	0	236	125	23	1020	1565	65.17
size	1055	1144	1143	1144	1056	1143	1055	1055	1142	1055		
	44.54	98.07	55.46	91.87	99.33	86.52	59.24	67.77	84.23	96.68		

Table 8.1: Pendigits: Repartition of the classes in the clusters obtained by step by step clustering

We examine clusters in the framework of the point-based clustering model. In this model we can use different optimisation techniques for the search for clusters. We also apply the notions of cluster function and the structure of clusters (see [24]) in order to check the quality of clusters obtained by this technique. The goal of this investigation is two-fold. First we show that the clusters and classes are very different in these datasets. We also compare different optimisation techniques for real world datasets.

8.2 Pendigits

8.2.1 Classes and centres

The experiments are first carried out on the Pendigits dataset. Although this dataset is quite large, it is usually well handled by most classification methods. The first experiment is to carry out a step by step clustering (see section 6.3 without any knowledge of the classes and compare these clusters with the classes. Table 8.1 shows the number of points from each class in each cluster.

As a result it is noticeable that a great majority of points in each cluster belongs to the same class. This means that the clusters seem to coincide with classes. The purity is 78.58%.

The second experiment carried out is to find for each class one centre. The points obtained are then considered as cluster centres and the same analysis as previously is applied. Table 8.2 presents the results.

	1	2	3	4	5	6	7	8	9	10	size	% class
1	561	0	0	0	0	142	14	1	0	0	718	78.13
2	0	1101	325	9	2	1	0	0	7	1	1446	76.14
3	0	37	651	18	0	12	1	126	148	31	1024	63.57
4	0	0	3	1060	9	79	0	34	0	1	1186	89.37
5	42	0	0	11	1043	25	3	0	0	0	1124	92.79
6	157	0	0	0	0	876	0	8	0	0	1041	84.14
7	67	0	0	0	1	0	613	0	0	0	681	90.01
8	6	0	75	45	0	8	180	770	0	2	1086	70.9
9	154	1	0	0	0	0	0	0	968	0	1123	86.19
10	68	5	89	1	1	0	244	116	19	1020	1563	65.25
size	1055	1144	1143	1144	1056	1143	1055	1055	1142	1055		
	53.17	96.24	56.95	92.65	98.76	76.64	58.1	72.98	84.76	96.68		

Table 8.2: Pendigits: Repartition of the classes in the clusters defined by class centres

A similar - but more expectable - observation is made about this table: the correspondence between clusters and classes is very strong. The purity is 78.81%.

The class centres are used as an initial point to the local optimisation method to find a local minimum to the cluster function. In the case of Pendigits the solution obtained is the same as the one reached by the step by step method.

Because the same result was reached by two different methods, it can be expected that it is a very good local minimum. For comparison, the objective function value for a solution reached by the k -means method was more than 2% larger.

The conclusion of this is that even by applying a clustering method without any knowledge of classes, the solution obtained is the one which corresponds at best to the classes. The classification accuracy is relatively high. In the Pendigits dataset the classes and cluster seem to be perfectly equivalent.

Indeed when the number of clusters is increased to 20 (twice the number of classes) for the step by step clustering, the purity becomes 87%. This result, obtained by a method which does not use any knowledge of classes, is comparable to most specifically designed methods presented in [145].

8.2.2 Classes and cluster structures

Figure 8.1 represents the structure of the clusters. The darker the square the larger amount of points are present in the layer. All the clusters present a similar structure: the points are deep, showing a good clustering.

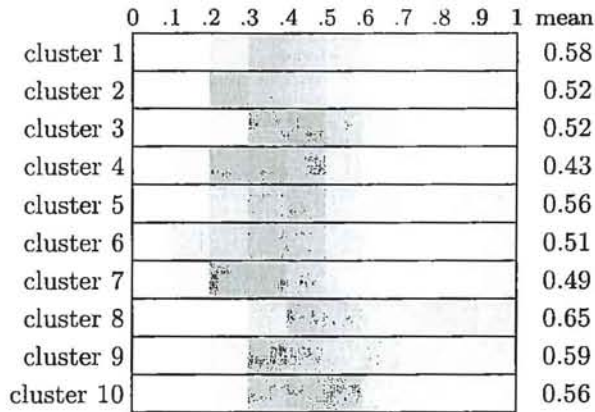


Figure 8.1: Structures of the clusters of pendigits

Cluster structures have been presented in [24] (see section 1.2.6 on page 19). They constitute a tool to evaluate the quality of the clustering. The Pendigits dataset has been shown to present a strong correlation between clusters and classes. It may be interesting to consider a deeper relation, by finding the layers of the clusters.

Figure 8.2 shows the depth of the classes in each cluster. Each disk represents one cluster, and each slice of this disk represents one class. The darker one circle is, the larger the proportion of points from this class is inside the corresponding layer.

Table 8.3 shows the repartition of the main class and the other points in each cluster. All the clusters present the same characteristic: although the points of each clusters are quite deep for all the clusters the main class is deeper than the others. This means that not only the cluster centres represent the classes well, but also the “misclassified points” do not belong so strongly to the cluster.

Table 8.4 shows the average depth of each class inside each cluster. The results presented in this table confirm that in each cluster the predominant class is the deepest.

8.2.3 Summary

- The relationships between classes and clusters are very explicit
- The points are “deep” inside the clusters, and therefore the clustering is considered of high quality
- The correctly classified points are deeper.

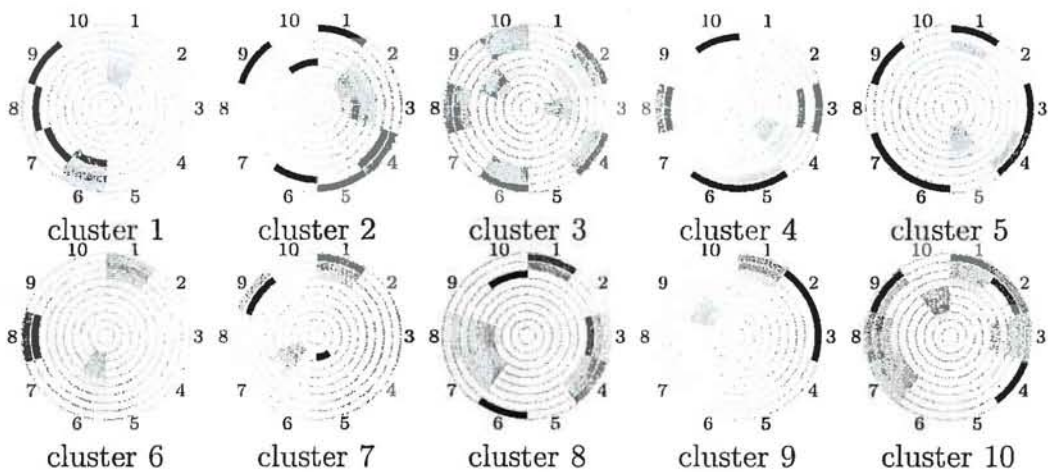


Figure 8.2: Structures of the clusters in pendigits

Cluster		[0,0.1]	[0.1,0.2]	[0.2,0.3]	[0.3,0.4]	[0.4,0.5]	[0.5,0.6]	[0.6,0.7]	[0.7,0.8]	[0.8,0.9]	[0.9,1]
1	Main class	0	0	6.8	22.12	18.72	16.17	11.06	11.27	6.17	0
	Other classes	0	0	0	0	0	3.6	37.83	18.01	24.32	0
2	Main class	0	0.08	5.97	22.01	26.73	18.36	13.72	8.55	3.83	0
	Other classes	0	0	1.43	14.61	27.79	16.61	15.47	9.74	7.73	0
3	Main class	0.31	11.35	18.61	23.34	17.5	12.77	6.46	5.36	2.52	0.31
	Other classes	0	0	2.25	6.49	16.94	12.42	6.49	15.53	19.2	0
4	Main class	0	1.23	10.56	17.6	21.4	13.32	14.93	8.65	7.61	0
	Other classes	0	0	0	0	0	0	0.86	12.17	38.26	0
5	Main class	0	5.71	22.11	25.64	20.59	13.34	7.14	3.62	1.42	0
	Other classes	0	0	0	0	0	0	0	10.86	19.56	0
6	Main class	0	4.24	23.55	19.61	18.7	12.84	8.08	5.66	3.74	0
	Other classes	0	0	0	0	0.77	3.47	10.42	22.39	33.59	0
7	Main class	0	7.68	26.08	26.24	19.36	13.44	4.32	1.76	0.96	0
	Other classes	0	0	0.53	0	0	1.6	6.95	16.57	28.34	0
8	Main class	0	0	0.97	11.74	20.27	17.34	13.28	10.48	14.82	0
	Other classes	0	0	0.31	3.47	10.72	13.56	14.51	26.81	16.71	0
9	Main class	0	0.1	4.67	17.15	23.38	22.03	14.55	9.04	5.5	0
	Other classes	0	0	0	0	0	0	1.38	15.27	36.11	0
10	Main class	0	0.49	13.23	26.17	23.62	18.82	8.43	4.7	3.13	0
	Other classes	0	0	0	0.18	1.28	6.42	14.67	22.2	26.42	0

Table 8.3: Pendigits:Repartition of the classes by cluster layers

	1	2	3	4	5	6	7	8	9	10
1	0.55	-	-	-	-	0.76	0.77	0.87	0.91	-
2	0.99	0.51	0.56	0.91	0.98	0.88	-	-	0.96	0.56
3	-	0.81	0.41	0.84	-	0.88	-	0.83	0.5	0.78
4	-	-	0.85	0.53	0.93	0.93	-	0.87	-	0.85
5	0.92	-	0.91	0.91	0.41	0.99	0.97	-	0.99	-
6	0.82	-	-	-	-	0.45	-	0.89	-	-
7	0.86	-	-	-	0.26	-	0.38	-	0.88	-
8	0.93	-	0.81	0.86	-	0.97	0.62	0.63	-	0.76
9	0.87	0.96	0.99	-	-	-	-	-	0.54	-
10	0.86	0.9	0.77	0.98	-	-	0.76	0.86	0.94	0.47

Table 8.4: average depth for each class in each cluster

8.3 Letters

The experiments described and applied on the Pendigits dataset in the previous section are here applied on the Letters dataset. The results are presented and discussed.

8.3.1 Classes and centres

The Letters dataset contains 26 classes. Therefore 26 clusters are found using the step by step algorithm. The size of the intersection of each cluster with each class is then evaluated. Table 8.5 presents the results.

Unlike for the Pendigits dataset, where each cluster is strongly associated with a class (and where all classes are represented by one cluster), the repartition of the classes in the clusters of the Letters dataset is much more scattered. The purity is very low: 28.25%.

Table 8.6 presents the repartition of the classes when the cluster centres considered are one centre per class. It is noticeable that here again the repartition of the points is very disseminate. The purity is 23.94%.

Finally these class centres are used as an initial point for the local algorithm to minimise the cluster function. While in the case of the Pendigits dataset the result of this experiment was similar to the solution found by step by step clustering, this time the centres are quite far apart and another solution is reached. Once again, the repartition, shown in table 8.7 is very diffuse.

A possible interpretation of these results could be that the clustering algorithm does not reach meaningful clusters. The number of clusters is fairly large, and the cluster function possess a very large number of local minima. Its minimisation is a very difficult task, even for a powerful method

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26			
1	242	0	140	0	0	0	0	0	0	0	0	0	187	0	0	0	1	0	0	0	0	0	104	34	0	0	708	34	
2	123	0	90	0	0	0	7	0	0	0	0	35	107	0	0	0	81	0	0	0	0	0	97	41	95	0	676	18	
3	16	84	32	69	27	40	62	2	67	34	36	71	15	53	32	25	39	44	15	0	59	10	21	40	47	9	949	8	
4	0	276	0	0	17	0	0	0	0	2	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	3	301	91	
5	115	50	125	63	61	11	103	0	111	0	28	74	79	16	43	73	111	1	74	0	0	15	0	5	83	57	1298	9	
6	0	21	7	3	274	2	0	0	0	91	0	8	7	0	0	0	11	125	1	0	105	157	68	0	0	6	886	30	
7	0	0	0	155	0	171	0	159	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	485	35	
8	0	28	6	24	3	27	31	116	4	24	1	30	6	29	18	5	0	16	0	76	35	2	5	36	33	0	555	20	
9	0	0	0	31	0	40	0	44	0	0	0	14	0	0	0	0	0	0	0	25	37	0	0	0	0	0	191	23	
10	0	0	0	0	0	0	0	0	0	291	0	0	0	0	0	0	0	212	0	0	2	198	0	0	0	0	703	41	
11	3	0	15	3	0	1	0	8	3	0	567	1	0	0	4	5	0	0	1	0	0	0	0	3	0	2	616	92	
12	0	56	0	43	0	36	88	0	93	0	0	32	0	153	134	0	45	0	0	0	66	0	21	99	14	0	880	17	
13	0	28	0	40	0	27	85	0	169	1	0	35	23	266	138	13	8	7	0	0	45	14	70	150	3	15	1137	23	
14	23	66	52	69	52	27	80	13	65	45	44	76	66	11	120	34	55	76	16	18	20	50	34	66	111	26	1315	9	
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	324	0	0	0	0	0	0	0	0	0	0	324	100	
16	104	55	98	72	33	1	89	0	114	0	1	93	116	0	1	5	122	0	30	0	0	2	0	1	98	12	1047	11	
17	0	7	0	0	4	0	1	0	4	0	59	0	1	0	0	60	0	0	350	0	0	0	0	0	4	190	680	51	
18	0	5	0	0	5	0	0	0	0	0	0	0	0	0	158	1	0	229	0	0	0	0	0	0	0	190	588	38	
19	0	0	0	2	8	127	0	239	18	26	13	1	0	6	0	1	0	1	0	327	26	0	8	1	0	1	805	40	
20	0	10	0	34	10	251	11	203	9	27	21	20	1	7	2	1	0	1	1	304	41	3	17	8	0	2	984	30	
21	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	169	0	0	0	0	0	174	97	
22	15	26	41	40	79	7	0	2	0	84	1	92	44	0	0	2	69	119	2	0	126	127	87	3	0	12	978	12	
23	0	68	0	3	163	5	0	2	0	137	0	9	3	0	0	0	4	146	0	2	28	174	40	0	0	0	784	22	
24	0	7	0	50	0	9	148	0	123	0	1	119	16	210	187	9	128	2	4	0	54	5	123	249	23	2	1469	16	
25	93	16	142	28	39	1	53	4	25	2	17	29	97	2	104	46	112	36	29	0	0	39	41	37	255	52	1299	19	
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	168	168	100
size	734	803	748	734	775	783	758	792	805	764	789	739	768	753	783	761	787	786	755	752	813	796	736	773	766	747			
	32	34	18	21	35	32	19	30	20	38	71	16	24	35	23	42	16	26	46	43	20	24	16	32	33	25			

Table 8.5: Letters: Repartition of the classes in the clusters obtained by step by step clustering

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26			
1	113	1	50	0	9	0	0	0	1	0	0	0	36	0	16	15	0	0	12	0	0	1	6	3	4	16	283	39	
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
3	0	242	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	242	100	
4	220	2	273	7	8	0	84	0	5	0	0	59	77	0	19	17	165	1	11	0	0	8	8	14	179	22	1179	23	
5	17	112	13	0	320	1	0	0	0	77	0	0	18	0	0	0	16	127	14	0	0	194	0	0	0	16	925	34	
6	5	0	7	1	0	0	8	0	0	0	0	0	0	0	0	7	1	0	0	0	0	0	0	0	0	1	30	26	
7	33	5	54	27	0	0	78	0	38	0	0	5	36	0	4	2	34	0	5	0	0	0	0	0	54	0	375	20	
8	3	39	5	11	10	17	28	0	123	1	16	2	1	9	11	57	30	5	23	0	11	0	0	7	13	19	441	27	
9	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	100	
10	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	50	
11	0	5	3	0	0	6	0	0	0	28	0	0	0	0	0	0	1	2	0	0	0	1	0	0	0	0	46	60	
12	0	3	1	6	19	0	0	0	0	346	0	44	5	0	0	0	51	177	6	0	22	86	2	0	0	0	768	45	
13	15	0	5	4	0	0	0	0	0	44	0	3	37	0	0	0	2	28	35	0	0	0	0	0	5	0	178	24	
14	15	0	31	6	0	1	18	0	42	1	455	0	0	2	3	92	131	0	37	0	2	0	0	7	1	64	908	50	
15	0	0	0	0	0	4	0	0	0	0	0	52	0	0	0	0	0	0	0	0	0	0	0	0	0	0	56	92	
16	0	0	0	0	0	0	0	0	0	5	0	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	22	77	
17	0	21	0	26	0	28	40	15	2	1	0	1	0	44	43	0	0	0	0	0	0	23	0	1	30	2	4	281	15
18	42	76	78	299	20	216	219	188	312	28	128	134	457	473	53	27	83	3	32	399	82	215	372	247	54	4265	11		
19	13	72	10	39	56	25	46	0	111	0	136	3	1	96	41	262	43	0	138	3	0	0	0	8	25	77	1205	21	
20	1	0	0	4	0	1	0	0	0	0	0	50	10	0	0	16	54	0	15	0	0	0	0	1	31	183	29		
21	140	165	194	157	116	7	96	1	139	40	39	142	205	76	115	74	207	118	54	0	13	54	63	110	198	89	2612	7	
22	0	44	0	120	41	471	135	586	32	110	115	181	0	48	11	18	9	6	3	717	126	7	42	41	37	9	2909	24	
23	0	0	2	24	5	6	0	2	0	4	0	19	8	2	0	0	5	1	0	0	190	0	46	0	0	44	358	53	
24	9	15	10	0	171	0	0	0	0	15	0	0	5	0	0	147	6	201	347	0	2	358	66	0	0	225	1577	22	
25	108	1	12	0	0	0	5	0	0	0	0	0	195	0	0	1	0	0	4	0	0	0	282	83	0	76	767	36	
26	0	0	0	3	0	0	1	0	0	60	0	32	0	19	47	0	5	37	48	0	25	5	5	98	0	0	385	25	
size	734	803	748	734	775	783	758	792	805	764	789	739	768	753	783	761	787	786	755	752	813	796	736	773	766	747			
	29	30	36	40	41	60	28	73	38	45	57	24	26	60	60	34	26	25	45	95	49	44	38	48	32	30			

Table 8.6: Letters: Repartition of the classes in the clusters defined by class centres

like the step by step method.

However neither the improved class centres nor the class centres themselves seem to represent the classes well. Moreover the value of the function at the point reached by the step by step method is sensibly lower than the one at the other points.

The distribution of the points of the dataset in clusters independent from their class may be due to the fact that some more information is contained in the dataset (for example the font of the letters), and this information may bring some noise for an eventual classification.

The latter interpretation is confirmed by the fact that when the number of clusters is decreased to 4, thus making the problem easier to solve, the repartition of the classes remains scattered.

8.3.2 Classes and cluster structures

Figure 8.3 shows the structure of the clusters. These clusters present generally a similar structure: the majority of points is deep inside the cluster. This shows that the clustering created clusters of high quality.

Experiments show that the distribution of the classes inside the clusters is very diffuse for the Letters dataset. This means of course that the classes do not represent the most obvious point-based clustering of this dataset. Let us consider the structures of the clusters and examine the distributions of the classes among the layers.

Figure 8.4 and table 8.8 show the distribution of the classes in each cluster. Clearly there are several types of clusters. Some of them contain only points from a few classes. The majority of the clusters, however, contains points belonging to many classes. In a general manner, it can be noticed that the repartition in layers for all the classes present in the cluster is quite similar. This result emphasises the conclusion that a cluster cannot be associated with a particular class in the Letters dataset.

Cluster		[0,0.1]	[0.1,0.2]	[0.2,0.3]	[0.3,0.4]	[0.4,0.5]	[0.5,0.6]	[0.6,0.7]	[0.7,0.8]	[0.8,0.9]	[0.9,1]
1	Main class	0	0	0	2.06	25.61	29.33	13.63	9.91	5.78	0
	Other classes	0	0	0	0.21	15.02	16.95	16.3	16.52	21.67	0
2	Main class	0	0	0.81	16.26	24.39	6.5	28.45	10.56	4.06	0
	Other classes	0	0	0	0.36	2.71	10.3	19.16	20.79	19.71	0
3	Main class	0	0	0	0	1.19	15.47	32.14	20.23	17.85	0
	Other classes	0	0	0	0.11	1.15	8.67	14.68	18.26	26.47	0
4	Main class	0	0	0	6.15	19.56	18.11	11.95	10.5	17.02	0
	Other classes	0	0	0	0	0	0	0	4	24	0

5	Main class	0	0	0	0	6.4	21.6	13.6	16.8	18.4	0
	Other classes	0	0	0	0.17	2.89	7.33	12.36	17.81	25.31	0
6	Main class	0	0	0	0	2.55	6.2	14.23	27	25.91	0
	Other classes	0	0	0	0.16	1.63	8.49	11.27	20.09	25.81	0
7	Main class	0	0	11.69	45.61	26.9	12.28	2.33	0.58	0	0
	Other classes	0	0	0	0	6.36	22.61	37.26	16.87	11.78	0
8	Main class	0	0	0	0.86	5.17	19.82	25.86	18.1	16.37	0
	Other classes	0	0	0	1.13	12.3	16.85	16.17	16.62	17.53	0
9	Main class	0	0	25	40.9	13.63	4.54	9.09	6.81	0	0
	Other classes	0	0	21.08	10.88	7.48	8.84	16.32	5.44	16.32	0
10	Main class	0	0	0	1.37	7.9	23.02	27.49	13.74	14.43	0
	Other classes	0	0	0	4.85	12.13	11.4	17.71	23.05	18.68	0
11	Main class	0	0	1.76	16.22	19.57	20.28	14.99	8.81	10.58	0
	Other classes	0	0	0	0	0	0	6.12	14.28	22.44	0
12	Main class	0	0	0	21.56	26.79	26.14	9.8	3.26	4.57	0
	Other classes	0	0	0	2.33	4.67	12.37	20.08	20.22	20.63	0
13	Main class	0	0	0.37	1.5	9.02	22.55	24.06	14.28	12.4	0
	Other classes	0	0	0	0.11	1.95	8.49	13.31	19.51	25.02	0
14	Main class	0	0	0	0	0	0.83	16.66	24.16	26.66	0
	Other classes	0	0	0	0	1.08	7.78	17.65	22.25	25.77	0
15	Main class	0	0.3	23.45	11.72	13.88	13.27	12.65	12.34	8.33	0
	Other classes	0	0	0	0	0	0	0	0	0	0
16	Main class	0	0	0	0	4.91	18.85	17.21	15.57	20.49	0
	Other classes	0	0	0.32	0.75	6.81	14.91	14.27	19.02	21.4	0
17	Main class	0	0	0	0	0	18.85	38.28	25.42	10.28	0
	Other classes	0	0	0	5.75	10	14.24	14.24	14.24	18.18	0
18	Main class	0	0	0	12.66	19.65	16.15	25.32	11.35	6.98	0
	Other classes	0	0	0.27	0.27	7.52	14.76	17.27	19.49	22.84	0
19	Main class	0	0	0	0	4.58	22.32	26.6	20.48	15.59	0
	Other classes	0	0	0	0.2	3.34	6.69	15.06	20.08	23.84	0
20	Main class	0	0	0	0.32	2.96	20.39	22.03	20.72	16.77	0
	Other classes	0	0	0	0.14	4.41	17.05	18.67	17.79	17.94	0
21	Main class	0	7.1	20.71	32.54	17.75	14.2	5.91	1.77	0	0
	Other classes	0	0	0	0	0	0	0	0	0	0
22	Main class	0	0	0	0	0	2.36	17.32	22.04	31.49	0
	Other classes	0	0	0	0	1.52	9.87	16.68	20.79	26.43	0
23	Main class	0	0	0	1.72	12.06	13.21	22.41	25.28	12.64	0
	Other classes	0	0	0	0.49	2.45	10.49	22.62	30.49	12.95	0
24	Main class	0	0	0	0	3.61	10.04	21.28	21.68	23.69	0

	Other classes	0	0	0	0.24	1.55	6.31	11.88	23.52	27.95	0
25	Main class	0	0	0	0	0.39	3.92	7.45	15.68	43.92	0
	Other classes	0	0	0	0.19	3.16	7.27	13.4	20.3	28.06	0
26	Main class	0	5.35	14.88	20.83	17.85	9.52	10.71	9.52	4.16	0
	Other classes	0	0	0	0	0	0	0	0	0	0

Table 8.8: Pendigits: Repartition of the classes by cluster layers

Table 8.9 presents the average depth of each class in each cluster. All the classes are at the same depth inside the clusters. Moreover this depth - generally between 0.3 and 0.8 - shows that the points are quite far from the boundary of each cluster. An interesting case is the 8-th cluster: most classes of the dataset are very deep inside the cluster. This can be seen as a strong belonging to the clusters, and it can be concluded that although the cluster is strongly constituted, it does not permit discrimination between classes.

8.4 Conclusion

The main conclusion to this research is that clusters do not necessarily coincide with classes. Several factors can cause such results.

- The chosen clustering model does not match the classification structure. (For example, it is possible that some classes in Letters dataset are not point based, while it is the point based model we use for clustering.)
- The dataset contains a high proportion of noisy records and/or possible mistakes, which may have appeared, for example, at the stage of data collection;
- Some characteristics link points more strongly than their belonging to classes.

The notion of purity cannot always be used for evaluation of accuracy of clustering methods. If the purity is high enough, we can conclude that the chosen clustering method is efficient for the dataset under consideration. However, we can not make any conclusion regarding the efficiency of clustering methods, if purity is low. In such a case the classes and clusters do not coincide. There can be different reasons for this: either the applied clustering technique does not work very well, or there might be some other "hidden" characteristics, which link the records together and which are independent from the classes.

The purity can then be seen as a measure of the quality of the dataset rather than of the cluster method.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26			
1	114	0	97	0	0	0	15	0	0	0	0	48	107	0	0	3	92	0	2	0	0	2	108	63	109	0	760	15	
2	242	0	183	0	0	0	0	0	0	0	0	0	192	0	3	0	7	0	0	0	0	0	85	42	94	0	848	28	
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
6	19	90	37	54	36	36	61	1	72	41	25	68	15	59	45	36	44	46	26	0	63	23	31	43	45	16	1032	8	
7	32	41	24	110	35	10	71	1	84	0	21	89	57	11	20	8	51	9	12	0	9	25	1	5	78	13	817	13	
8	0	223	1	1	344	0	0	0	0	26	0	2	1	0	0	0	2	89	2	0	58	131	71	0	0	2	953	36	
9	1	51	26	39	21	41	65	101	23	42	11	60	26	32	49	23	13	51	1	79	49	12	18	67	74	2	977	10	
10	0	0	0	160	0	178	0	181	0	0	0	0	0	0	0	0	0	0	0	0	169	0	0	0	0	0	0	688	26
11	79	29	76	65	18	3	73	0	100	0	1	72	100	0	0	1	80	0	9	0	4	1	1	2	84	6	804	12	
12	0	0	0	0	0	1	0	0	0	385	0	0	0	0	0	0	0	307	0	0	6	300	0	0	0	0	0	999	38
13	3	0	25	3	0	13	0	12	7	0	364	2	0	2	4	13	0	0	7	0	0	0	5	0	15	475	76		
14	0	0	0	0	0	0	1	0	0	0	299	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	301	99	
15	0	53	0	70	22	72	69	20	79	0	0	37	0	133	125	0	40	0	45	15	107	0	22	56	10	0	975	13	
16	0	90	0	30	0	25	99	0	175	0	0	45	19	283	130	15	13	6	0	0	38	16	77	151	23	14	1249	22	
17	94	57	111	55	66	11	73	9	55	25	38	60	108	7	176	55	101	79	38	0	5	53	58	72	134	67	1607	10	
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	326	0	0	0	0	0	0	0	0	0	0	0	326	100
19	146	54	135	32	49	0	75	0	70	0	0	64	103	11	36	101	154	3	88	0	0	19	4	3	62	63	1272	12	
20	0	3	0	0	5	0	0	0	0	0	0	0	0	0	0	168	0	0	374	0	0	0	0	0	0	0	285	835	44
21	0	8	0	31	10	238	12	196	9	22	20	18	1	6	2	0	0	1	1	313	40	2	17	8	0	1	956	32	
22	0	1	0	5	6	137	0	270	15	32	9	2	0	6	1	1	0	1	0	343	26	0	11	2	0	2	870	39	
23	3	49	27	23	162	10	0	1	0	190	0	61	22	0	0	0	61	192	6	2	171	205	103	0	0	41	1329	15	
24	1	54	6	56	1	8	144	0	116	1	1	111	17	203	192	10	129	2	3	0	68	7	129	254	53	4	1570	16	
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	100	
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	141	0	0	0	0	0	0	214	355	60
size	734	803	748	734	775	783	758	792	805	764	789	739	768	753	783	761	787	786	755	752	813	796	736	773	766	747			
	32	27	24	21	44	30	18	34	21	50	46	15	25	37	24	42	19	39	49	45	21	37	17	32	17	38			

Table 8.7: Letters: Repartition of the classes in the clusters defined by improving the class centres

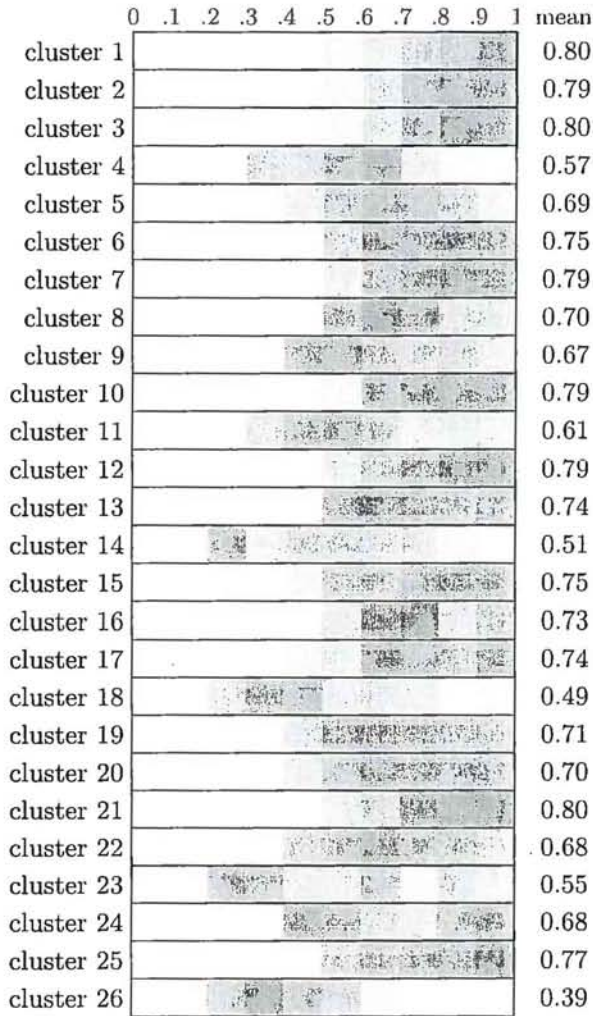


Figure 8.3: Structures of the clusters for the letters dataset

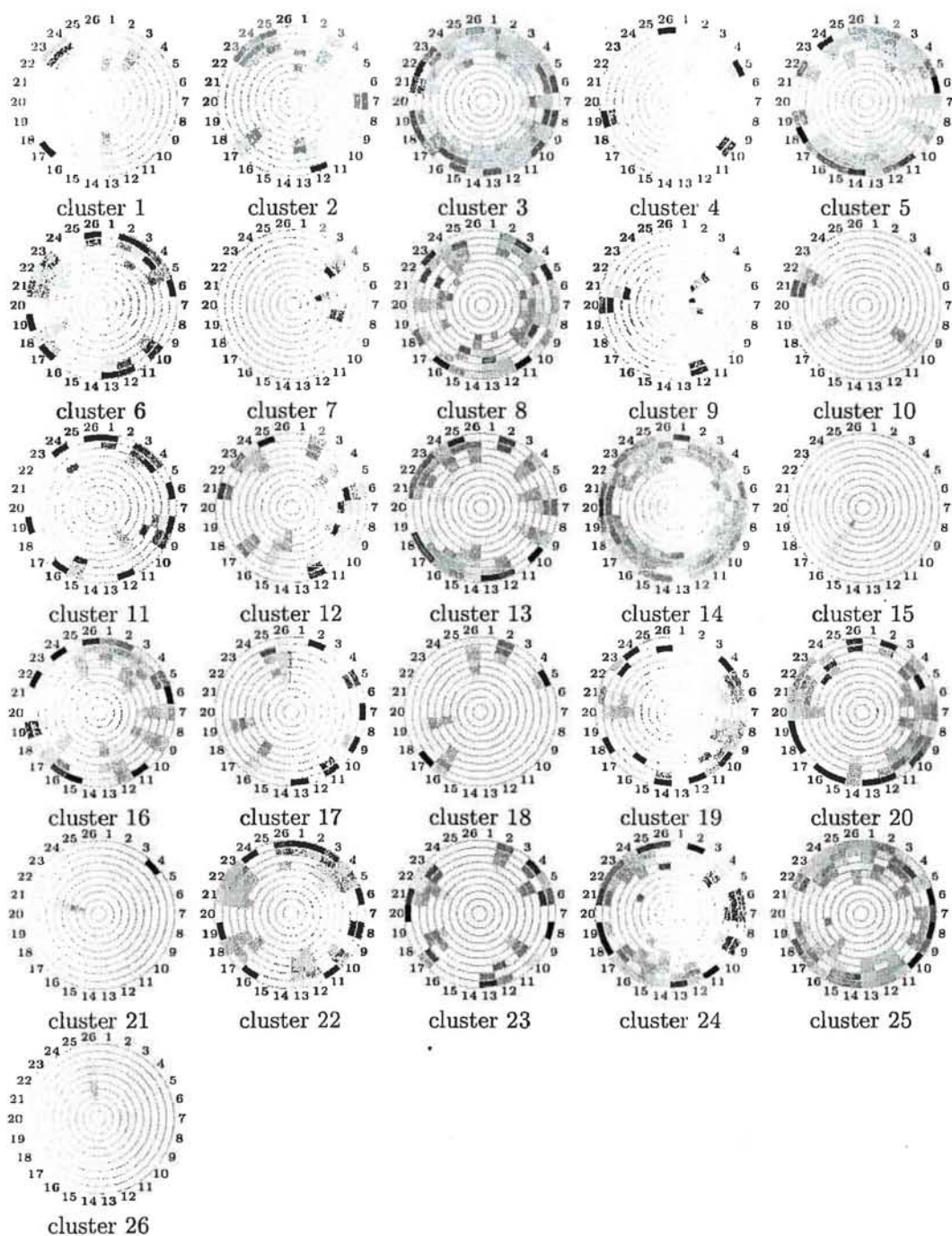


Figure 8.4: The structures of the clusters

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
1	0.63	-	0.68	-	-	-	-	-	-	-	-	-	0.6	-	-	-	0.97	-	-	-	-	-	0.86	0.84	-	-
2	0.59	-	0.74	-	-	-	0.91	-	-	-	-	0.87	0.65	-	-	-	0.73	-	-	-	-	-	0.87	0.82	0.83	-
3	0.87	0.73	0.8	0.77	0.82	0.9	0.72	0.91	0.75	0.89	0.84	0.78	0.85	0.71	0.87	0.81	0.83	0.85	0.9	-	0.85	0.9	0.88	0.76	0.73	0.92
4	-	0.67	-	-	0.92	-	-	-	-	0.89	-	-	-	-	-	-	-	-	0.93	-	-	-	-	-	-	0.96
5	0.83	0.82	0.74	0.79	0.86	0.95	0.74	-	0.77	-	0.88	0.8	0.72	0.89	0.88	0.88	0.8	0.98	0.86	-	-	0.89	-	0.89	0.71	0.9
6	-	0.88	0.86	0.89	0.79	0.93	-	-	-	0.87	-	0.91	0.92	-	-	-	0.9	0.72	0.95	-	0.85	0.77	0.85	-	-	0.94
7	-	-	-	0.74	-	0.41	-	0.61	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
8	-	0.83	0.93	0.76	0.88	0.67	0.7	0.71	0.79	0.68	0.94	0.72	0.79	0.62	0.75	0.92	-	0.79	-	0.68	0.6	0.73	0.83	0.71	0.77	-
9	-	-	-	0.45	-	0.29	-	0.43	-	-	-	0.91	-	-	-	-	-	-	-	0.89	0.71	-	-	-	-	-
10	-	-	-	-	-	-	-	-	-	0.68	-	-	-	-	-	-	-	-	0.61	-	-	0.91	0.8	-	-	-
11	0.92	-	0.85	0.92	-	0.93	-	0.88	0.84	-	0.59	0.96	-	-	0.83	0.95	-	-	0.94	-	-	-	-	0.85	-	0.97
12	-	0.84	-	0.6	-	0.81	0.71	-	0.7	-	-	0.89	-	0.55	0.62	-	0.74	-	-	-	0.9	-	0.86	0.75	0.89	-
13	-	0.9	-	0.77	-	0.7	0.85	-	0.77	0.99	-	0.88	0.92	0.69	0.78	0.83	0.94	0.92	-	-	0.79	0.85	0.83	0.79	0.91	0.74
14	0.88	0.81	0.83	0.83	0.74	0.81	0.77	0.76	0.82	0.8	0.84	0.81	0.69	0.8	0.83	0.75	0.78	0.81	0.83	0.91	0.86	0.76	0.8	0.78	0.77	0.78
15	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.52	-	-	-	-	-	-	-	-	-	-
16	0.83	0.87	0.72	0.68	0.86	0.95	0.77	-	0.74	-	0.95	0.78	0.65	-	0.97	0.93	0.75	-	0.88	-	-	0.96	-	0.93	0.65	0.91
17	-	0.95	-	-	0.9	-	0.92	-	0.95	-	0.9	-	0.9	-	-	0.7	-	-	0.7	-	-	-	-	-	0.75	0.65
18	-	0.87	-	-	0.93	-	-	-	-	-	-	-	-	-	-	0.78	0.95	-	0.6	-	-	-	-	-	-	0.69
19	-	-	-	0.98	0.86	0.79	-	0.78	0.8	0.9	0.8	0.99	-	0.93	-	0.78	-	0.91	-	0.7	0.77	-	0.89	0.97	-	0.89
20	-	0.93	-	0.82	0.83	0.73	0.84	0.69	0.83	0.88	0.77	0.89	0.95	0.81	0.95	0.97	-	0.97	0.98	0.73	0.82	0.85	0.74	0.78	-	0.91
21	-	-	-	0.98	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.38	-	-	-	-	-
22	0.9	0.9	0.84	0.77	0.82	0.95	-	0.91	-	0.85	0.99	0.74	0.76	-	-	0.97	0.69	0.8	0.99	-	0.73	0.82	0.8	0.96	-	0.87
23	-	0.84	-	0.92	0.74	0.88	-	0.97	-	0.71	-	0.82	0.89	-	-	-	0.83	0.69	-	0.97	0.9	0.7	0.83	-	-	-
24	-	0.93	-	0.73	-	0.78	0.83	-	0.85	-	1	0.84	0.91	0.73	0.77	0.73	0.81	0.95	0.92	-	0.89	0.71	0.86	0.76	0.91	0.8
25	0.81	0.77	0.78	0.82	0.72	0.91	0.86	0.92	0.83	0.93	0.79	0.84	0.8	0.9	0.88	0.68	0.82	0.84	0.66	-	-	0.83	0.77	0.77	0.83	0.74
26	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.5

Table 8.9: average depth for each class in each cluster

Chapter 9

Shape of a finite set of points

9.1 Introduction

Although the general definition of data analysis is generally quite broad, in practice only a few fields, such as clustering or classification, have received much attention.

It may sometimes be necessary to extract information from data without needing to subdivide it, for example when an explanation for a phenomenon is sought. In this chapter, we develop an algorithm to study the shape of data. We assume that points have been obtained using a clustering algorithm such as those presented in chapter 6 on page 109, and study each cluster separately (in other terms, the centre of each set has been found). We develop an algorithm to find a geometrical object containing a finite set of points. In this research we focus on a special class of geometrical objects: the ellipsoids.

In this algorithm an auxiliary procedure to eliminate noisy points is developed. This procedure, itself based on ellipsoidal shapes and cluster structures, is quite important, and has many practical applications.

Our algorithm requires the solution of a sequence of optimisation problems. We use the discrete gradient method to solve these problems.

9.2 Geometrical approximation for a finite set of points

9.2.1 Sets and shapes

Suppose that we have a finite set of points

$$A = \{a_i \in \mathbb{R}^n | i = 1, N\}.$$

Our task is to find a shape containing the set. We would like to construct a geometrical object from a certain class which contains this finite set.

We propose an approach to find ellipsoidal approximations for given sets. In order to describe this algorithm we need some definitions.

Definition 30. An ellipsoid which includes the finite set of points A is called a *shape* of the set A .

Definition 31. A shape is *suitable* if at least one of the points from the set A is on the boundary of this geometrical body (shape).

We present several approaches to find suitable shapes for the finite set.

9.2.2 Positive definite symmetric matrix approach(PDSMA)

Any ellipsoid can be described by a centre and a positive definite matrix. Let M^+ be the set of positive definite matrices. In order to find a suitable shape we intend to determine a positive definite symmetric matrix M , such that:

$$\|c - a_i\|_M \leq 1 \quad \forall i \in \{1, \dots, N\},$$

where there exists an $a_j \in A$ such that $\|c - a_j\|_M = 1$, c is the centre of the finite set obtained by an appropriate method, and

$$\|x\|_M^2 = \langle x, Mx \rangle.$$

As an approximation of the set A we then consider the object

$$\{x \in \mathbb{R}^n : \|c - x\|_M \leq 1\}. \quad (9.1)$$

9.2.3 Positive diagonal matrix approach(PDMA)

It is reasonable to only consider the subset $D_+ \subset M^+$ of positive diagonal matrices. In order to find a suitable shape we intend to determine a positive definite diagonal matrix $D = \text{diag}(d_1, \dots, d_n)$, such that:

$$\|c - a_i\|_D \leq 1 \quad \forall i \in \{1, \dots, N\}$$

where there exists an $a_j \in A : \|c - a_j\|_D = 1$, c is the centre of the finite set obtained by an appropriate method, and

$$\|x\|_D = \left(\sum_{i=1}^n d_i |x_i|^j \right)^{1/j}, \quad j = 1, 2.$$

As an approximation of the set A we then consider the set

$$\{x \in \mathbb{R}^n : \|c - x\|_D \leq 1\}. \quad (9.2)$$

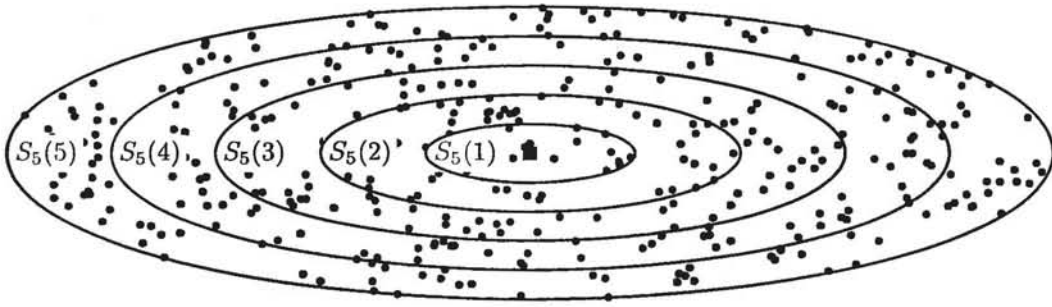


Figure 9.1: The structure for $L = 5$ for a shape of a finite set

Remark 25: The PDSMA is a generalisation of the PDMA for the Euclidean norm, but the dimension of the problem appearing in the PDMA is significantly less.

9.2.4 Structures

Once a shape has been found for A , we would like to study the distribution of the points within this shape. Let $L \in \mathbb{N}$. We want to divide the shape into L layers from the centre to the boundary, and analyse the distribution of the points within these layers. Figure 9.1 shows an illustration of the division of an ellipse into 5 layers. The points represent the finite set approximated by the ellipse.

$$\left\{ \begin{array}{l} S_L(k) = \left\{ a_i \mid \frac{k-1}{L} \leq \frac{\|a_i - c\|_M}{\max_j \|a_j - c\|_M} < \frac{k}{L} \right\} \text{ for } 1 \leq k \leq L - 1 \\ S_L(L) = \left\{ a_i \mid \frac{L-1}{L} \leq \frac{\|a_i - c\|_M}{\max_j \|a_j - c\|_M} \leq 1 \right\} \end{array} \right.$$

Remark 26: In this structure $\text{card}(S_L(L)) \geq 1$.

Definition 32. The point a_{i_0} is *isolated* if:

$$a_{i_0} \in S_L(L) \quad \text{and} \quad S_L(L - 1) = \emptyset.$$

Very often the set we would like to study consists of two parts. The first part contains a lot of points which are close to each other (the so called *solid part*). The other part contains few points spaced out around the solid part. The points from the spaced out part could be interpreted as noise. We can use these structures in order to create an algorithm to separate the solid part from the points considered to be noisy.

9.3 Algorithm

We propose a new algorithm to find an ellipsoidal shape for the finite set of points. This algorithm has three phases.

Algorithm 9.1: Finding the shape of a set of points

Step 1 Find the centre of the set

Step 2 Eliminate some isolated points that are considered as noise

Step 3 Find the ellipsoidal shape

In the following subsections we will explain each step more precisely.

9.3.1 Find the centre

To find the centre of the set A we solve the clustering problem (1.4) on page 15.

9.3.2 Elimination of the isolated points

A point which is too far from the rest of the points, and will thus induce some noise in the process of finding the shape has to be eliminated from the set A . For that, we need to find a shape in which such a point is isolated. This means that for such a shape, most of the points will be near the centre, while the ones to be eliminated will be on the boundary of this ellipsoid.

To find such an ellipsoid, we need the points to be globally the furthest possible from the boundary of the shape, i.e. closer to the centre c . This leads naturally to the optimisation problem of determining M such that:

$$\begin{aligned} & \text{minimise } \sum_{a \in A} \|a - c\|_M \\ & \text{such that} \\ & \max_{a \in A} \|a - c\|_M = 1; M \in M^+. \end{aligned} \tag{9.3}$$

To eliminate the noise, we have to solve the problem (9.3) repeatedly until the noisy data has been removed. Let $L \in \mathbb{N}$, and apply the following algorithm.

In our examples, the points were eliminated with both $L = 10$ and $L = 5$ before the next step.

Algorithm 9.2: An algorithm for eliminating noise

Step 1 Set $A_0 \leftarrow A$ and $i \leftarrow 0$
repeat
Step 2 | solve (9.3)
 | if $S_L(L-1) = \emptyset$ then
 | | $A_{i+1} = A_i \setminus S_L(L)$
Step 3 | Set $i = i + 1$
 | until $A_i = A_{i-1}$

9.3.3 Finding the shape

Once no more noise exists, we can determine the shape. Now we want the boundary of the ellipsoid to be as close to the group of points as possible. We obtain the following optimisation problem for M :

$$\begin{aligned}
 & \text{maximise } \sum_{a \in A} \|a - c\|_M \\
 & \text{such that} \\
 & \max_{a \in A} \|a - c\|_M = 1, M \in M^+.
 \end{aligned} \tag{9.4}$$

The restriction on the set of positive definite symmetric matrices could be implemented using the Cholesky factorisation (for details see [184]). It means that there exists a unique lower triangular $n \times n$ dimensional matrix G with a positive prime diagonal $g_{i,i} > 0$, $i = 1, \dots, n$, such that $M = GG^T$. Thus there exists a bijection between the set of positive definite symmetric matrices and the set of lower triangular matrices with positive leading diagonals.

9.4 Numerical experiments: comments and descriptions

9.4.1 Subsets

The algorithm was applied on the Diabetes dataset. This dataset contains 2 classes (500 observations in the first class and 268 observations from the second class). All features (1-8) are continuous. After application of a feature selection algorithm, we obtain the subset (1,2,8) of the most informative features.

We present some results for different sets of points. We find 3 clusters in the first class and 3 clusters in the second one (Sets 1-6). We also study each class more precisely (Sets 7 and 8).

9.4.2 Point elimination

We present some results obtained by the algorithm for noise elimination. We use the structures of the order $L = 10$ and $L = 5$.

Set	Initial size	Number of iterations	Points eliminated
1	137	3	119;127;51
2	226	6	156;13;46;80;181;77;85;104;43
3	66	2	19
4	85	2	44
5	137	2	42
6	117	1	none
7	500	1	none
8	268	1	none

Table 9.1: Point elimination for symmetric definite positive matrices

Set	Initial size	Number of iterations	Points eliminated
1	137	3	119,127
2	226	6	156;13;46;80;181;77;85;104
3	66	2	19
4	85	2	44
5	137	1	none
6	117	1	none
7	500	1	none
8	268	1	none

Table 9.2: Point elimination for diagonal matrices

Remark 27: No actual elimination occurs during the last iteration.

Remark 28: The results for elimination obtained by diagonal and positive definite symmetric positive matrices are slightly different, but in the case of diagonal matrices we have much fewer variables in the optimisation problem. It is therefore reasonable to use diagonal matrices in the elimination process.

Remark 29: If no point is eliminated in a set, the elimination may need to be refined by running the algorithm with another value of L .

9.4.3 Shape of the sets

For each set after point elimination we found a suitable shape and the repartition of the points within this new shape. We present results for the three biggest sets in our example (Sets 2,7 and 8).

Set	$S_{10}(1)$	$S_{10}(2)$	$S_{10}(3)$	$S_{10}(4)$	$S_{10}(5)$	$S_{10}(6)$	$S_{10}(7)$	$S_{10}(8)$	$S_{10}(9)$	$S_{10}(10)$
2	5	16	24	23	10	4	3	21	80	31
7	30	112	139	74	42	26	32	19	12	14
8	4	11	35	70	52	31	30	19	8	8

Table 9.3: The repartition of the points using $S_{10}(k)$ structures, $k = 1, \dots, 10$.

In the case of the Set 2, we observe that many points are close to the boundary of the suitable shape, and therefore the repartition of the points is quite regular. For the Sets 7 and 8 we could not obtain such results. It is possible that we still have some noisy points.

9.5 Conclusion

In this chapter, an algorithm to find the shape of a finite set of points has been developed. This algorithm is based on the solution of several nonsmooth optimisation methods.

Using the shape, it is possible to eliminate a number of noisy points from the dataset. This may be very useful to obtain more accurate results.

This research should be applied to other types of shapes, and it could be interesting to study the intersections of various shapes. The intersection between the shapes and the skeletons as a characterisation of the sets should also be studied.

Conclusion and further research

Conclusion

In this thesis, tools for addressing data analysis goals have been developed. The research has taken two different, complementary directions.

It has been observed that the optimisation based approach to data analysis is one of the most promising. The modularity of the methods gives the experts more freedom compared to other approaches. However, these methods are based on large-scale mathematical programming problems with nonsmooth, nonconvex objective functions, and the vast majority of general purpose softwares were unable to solve these problems.

Based on one of the most adapted algorithms, the discrete gradient method, an approach particularly adapted to data analysis optimisation problems has been designed. First, theoretical considerations on piecewise partially separable functions have been presented, and an algorithm has been developed to minimise these function. Numerical results have shown that this algorithm is very efficient, and notably that for this class of functions it improves the speed of original discrete gradient method. This algorithm has then been successfully applied to several problems of data analysis.

Although the algorithm proposed addresses a number of issues, it is a descent method, which can only guarantee termination on a stationary point. Unfortunately global methods are not applicable to problems on large scale datasets. In order to obtain a satisfactory solution it is necessary to design techniques that provide a good initial guess, thereby ensuring that the solution reached is good.

The first problem to which this optimisation method has been applied is the one of supervised learning of data. There exists an extremely large number of solutions to this problem, some of which are based on optimisation. The most popular among these is the so-called Support Vector Machine, which maps the data via a nonlinear projection over a larger dimensional set before attempting to separate it using a linear function. Notwithstanding the popularity or the efficiency of this technique, we suggest one limitation: the

nonlinear mapping has to be specified by the user and therefore it may happen that the Support Vector Machine does not reach a good solution. On the other hand, another approach, based on piecewise linear separation of sets, seemed rather promising, although based on a more complex optimisation problem.

We used the algorithm developed previously on this problem, and adapted this set-separating approach to a multi-class classification method. Numerical experiments on large scale datasets show that this method is very efficient and deserves to be further improved. In particular, a method to obtain the number of planes necessary in the linear function should be provided.

Another problem on which we applied our method is the one of unsupervised learning. The most common —and arguably the best— approach is the one which consists of minimising the average dissimilarity among clusters. This is a very complex nonsmooth, nonconvex optimisation problem. Two approaches for solving this problem are proposed.

The first approach is based on an incremental algorithm: the clusters are added one by one, and at each iteration, all the clusters are fixed while the added cluster is placed. Then the clusters are refined. This method is very efficient, as it constructs the points step by step, and always ensures that the optimisation method is given a good initial guess.

The second approach is better suited for larger datasets and for situations where a quick (but good) solution is privileged over a better one. In particular, this approach does not minimise over the whole dataset but on a summary of it, specifically the “ ϵ -cleaned” version. Several initial points are proposed, and 3 different minimisation methods are applied. The numerical experiments show that the initial points ensure that all three methods reach a good solution. An interesting observation is that when a general-purpose optimisation software is applied, it reaches better solutions over the cleaned dataset than over the original one, because the latter generates a too heavy problem for these softwares.

The algorithm is also applied on the minimisation of “skeletons”, but the results show that it is not so efficient. In particular, the combination with the ϵ -cleaning does not provide as good results as for the point-based clustering.

Very few methods for preprocessing the data for unsupervised learning have been designed so far. In this thesis, a new method, based on the clustering algorithm designed previously, is proposed. This method is applied on test datasets, and it is shown that it works well. Moreover it can easily be adapted to supervised learning, which is very useful in the case of datasets of small size.

The penultimate chapter examines the common belief that clustering a labelled dataset should necessarily comply with the distribution into classes.

This belief has given rise to a measure of the quality of clusters called purity. However, it is shown here that it is not necessarily true. While for some datasets, the results obtained strongly correspond to the classes — which certainly show that the clustering method is very good — in some other cases, there is absolutely no relationship between these two. Although in these cases, other measures for the quality of the clustering give satisfactory results, the purity is very low. Hence the purity may be a measure of the quality of the dataset rather than one of the clustering method.

Finally, a new type of data analysis problem has been examined: discovering the geometrical properties of a set of points. An algorithm for finding an ellipsoidal shape has been proposed and applied on a test dataset. The results are promising, and in particular, this algorithm can be used to eliminate successfully noisy points from the dataset.

Further research

The version of the discrete gradient method presented in this thesis is an important step in the application of this method to solve data analysis problems. The computational cost can be drastically reduced when the scheme is applied.

It is necessary to continue the research in this direction, and to enhance even more the efficiency of the method. In the future, the size of datasets will increase, and we have seen that it may be advantageous to solve several optimisation problems (for example in the step by step clustering method). Two directions can (and should) be followed for this purpose.

- The main bottleneck of the method is now on step 4 of the algorithm 2.2 on page 59. It consists of solving a smooth quadratic problem: the minimal distance between the origin and a polyhedron. This task is carried out by applying Wolfe's method. While this method is quite efficient, it does not use the previous knowledge about the polyhedron which is available in the algorithm. A new method using this knowledge would probably improve the efficiency of the overall method.
- The other improvement can be carried out on the line search (step 5). While this step is not itself very time consuming, the quality of its results is directly related to the number of iterations, that is the number of times the discrete gradients are computed and the Wolfe algorithm applied. Improving the line search could lead to a substantial improvement.

The discrete gradient method based on quasidifferentials has not been applied throughout this thesis. Using the quasidifferentials has advantages, inasmuch as it provides a more accurate approximation to the function than the subdifferential. However there is no formal proof of termination for this algorithm. Applying the simplified scheme to this version of the discrete gradient method may nevertheless prove worthwhile, since it may require fewer iterations, and therefore less computationally expensive than the subdifferential version.

From the point of view of data analysis the algorithms presented in this thesis all gave promising results on test datasets. The supervised learning method based on piecewise linear separation provides a theoretical security: if the number of hyperplanes is large enough, then the separating functions should separate the classes correctly. Moreover any separating surface can be approximated by a piecewise linear function. Following the results presented in this thesis, a method should be devised to find the number of hyperplanes necessary. Moreover, a good initial point should be provided to the local minimisation algorithm.

An idea which may lead to a solution to this problem is by constructing the separating function step by step, adding hyperplanes one by one, possibly using the classification through clustering in conjunction with the step by step clustering.

The clustering methods should be further tested, using other dissimilarity functions. In particular, it may be interesting to modify the problem in order to take into account the dissimilarity between clusters. (That is take a combination of two criteria: the average similarity with the cluster centre, and the average dissimilarity from the other cluster centres).

Finally, the shapes of clusters open the door to a new research direction, which should be studied. Many improvements can be studied for this research:

- Studying the shapes as ellipsoids may be too restrictive, and other types of shapes should be studied. In particular, the intersections between various shapes should give more accurate approximation of the envelope of the sets.
- The point elimination procedure should be further studied. In particular, experiments should be carried out to see how this elimination influences the results of clustering and classification methods
- One interesting application of the ellipsoidal shapes is to the recombination of features: using the ellipsoid, it is possible to see how the

features interact with each other, and to process the dataset by multiplying each observation by the resulting matrix. Experiments should be made to see whether this improves the results of classification and clustering algorithms.

Index

B

bundle methods, 55

C

characteristic, 5

Clarke generalised gradient, 52

Clarke regular, 51

Clarke subdifferential, 51, 51–52

class, 7, 21

classification, 7, 21–35

clustering, 7–8, 9–19

 Mathematical formulation, 10–14

constraints, 45

convex function, 46

convex set, 46

core function, 82

cost function, 44

cutting planes, 54–55

D

data preprocessing, 8–9, 35–43

database, 5

dataset, 5

DC, 53

Demyanov-Rubinov quasidifferential,
52–53

descent methods, 53

difference of convex, 53

directional derivative

 generalised, 51

discrete gradient method, 55–59

E

epsilon cleaning, 39–40

error function, 23

F

feasible solution, 44

feature, 5, 6

feature selection, 9, 38–39

finite elements, 54

fitness function, 44

fuzzy clustering, 10

G

generalised cluster function, 40

global minimiser, 47

global minimum, 47

H

hard clustering, 10

I

inf-stationary point, 52

K

k-means, 15–18

k-methods, 15–18

k-planes, 15–18

Karush-Kuhn-Tucker, 46

L

lagrangian, 46

learning

 supervised, *see* classification

 unsupervised, *see* clustering

Lipschitz constant, 49
Lipschitz continuity, 49
local minimiser, 47
local minimum, 47
lower concave approximation, 50

M

max-min separability, 31
metaheuristics, 62
minimiser
 global, 47
 local, 47
minimum
 global, 47
 local, 47

O

objective function, 45
observation, 5
optimisation methods, 44–67
optimisation problem, 45

P

penalty function, 45
penalty parameter, 45
piecewise linear separability, 31
piecewise linear separability, 32
piecewise partially separable function,
 70
positively homogeneous function, 49
preprocessing, *see* data preprocessing

Q

quasidifferentiable, 52
quasidifferential, 52, 52–53

R

record, 5, 6
regular
 Clarke, 51

S

scaling, 9, 36–38

semismooth, 50
shape, 165
similarity, 10, 11
subdifferential, 49, 50
subgradient, 50
suitable shape, 165
superdifferential, 49
supervised learning, *see* classification
support vector machine, 25

T

test set, 22
training set, 21, 22

U

unsupervised learning, *see* clustering
upper convex approximation, 50
upper convex approximation, 50

Bibliography

- [1] E.H.L. Aarts and J. Korst. *Simulated Annealing and Blotzman Machines*. John Wiley and Sons, 1989.
- [2] E.H.L. Aarts and P.J.M. Van Laarhoven. Simulated annealing: a pedestrian review of the theory and some applications. In P A Devijver and J Kittler, editors, *Pattern Recognition Theory and Applications*, pages 179–192. Springer-Verlag, Berlin New-York Tokyo, 1986.
- [3] B.M. Aberick, C.H. Bicshof, A. Carle, J. More, and A. Griewank. Computing large sparse jacobian matrices using automatic differentiation. *SIAM Journal on Scientific and Statistical Computing*, 15:285–294, 1994.
- [4] K.S. Al-Sultan. A tabu search approach to the clustering problem. *Pattern Recognition*, 28:1443–1451, 1995.
- [5] K.S. Al-Sultan and M.M. Khan. Computational experience on four algorithms for the hard clustering problem. *Pattern Recognition Letters*, 17:295–308, 1996.
- [6] M.Y. Andramonov, A.M. Rubinov, and B.M. Glover. Cutting angle for minimizing increasing convex-along-rays functions. Research Report 7/97, SITMS, University of Ballarat, Australia, 1997.
- [7] M.Y. Andramonov, A.M. Rubinov, and B.M. Glover. Cutting angle method in global optimization. *Applied Mathematics Letters*, 12:95–100, 1999.
- [8] A. Astorino and M. Gaudioso. Polyhedral separability through successive lp. *Journal of Optimization Theory and Applications*, 112(2):265–293, February 2002.
- [9] A. M. Bagirov, A.M. Rubinov, and J. Yearwood. A heuristic algorithm for feature selection based on optimisation technique. Research Report 01/2, University of Ballarat, Ballarat, Australia, 2001.

-
- [10] A.M. Bagirov. A method of approximating a subdifferential. *Russian Journal of Computational Mathematics and Mathematical Physics*, 32(4):561–566, 1992.
- [11] A.M. Bagirov. Continuous approximation to a subdifferential of a function of a maximum. *Cybernetics and System Analysis*, 4:626–630, 1994.
- [12] A.M. Bagirov. Continuous subdifferential approximation and its construction. *Indian Journal of Pure and Applied Mathematics*, 1:17–29, 1998.
- [13] A.M. Bagirov. A method for minimizing convex functions based on the continuous approximations to subdifferential. *Optimization Methods and Software*, 9(1-3):1–17, 1998.
- [14] A.M. Bagirov. Derivative-free methods for unconstrained nonsmooth optimization and its numerical analysis. *Investigacao Operacional*, 19:75–93, 1999.
- [15] A.M. Bagirov. Minimization methods for one class of nonsmooth functions and calculation of semi-equilibrium prices. In A. Eberhard et al., editor, *Progress in Optimization: Contribution from Australasia*, pages 147–175. Kluwer Academic Publishers, 1999.
- [16] A.M. Bagirov. Numerical methods for minimizing quasidifferentiable functions: a survey and comparison. In Demyanov and Rubinov [59], chapter 2, pages 33–71.
- [17] A.M. Bagirov. *The cutting angle method and its applications*. PhD thesis, ITMS, University of Ballarat, Australia, September 2001.
- [18] A.M. Bagirov. A method for minimization of quasidifferentiable functions. *Optimization Methods and Software*, 17:31–60, 2002.
- [19] A.M. Bagirov. Continuous subdifferential approximations and their applications. *Journal of Mathematical Sciences*, 115:2567–2609, 2003.
- [20] A.M. Bagirov, B. Ferguson, S. Ivkovic, G. Saunderris, and J. Yearwood. New algorithms for multi-class cancer diagnosis using tumour gene expression signatures. *Bioinformatics*, 19(14):1800–1807, 2003.
- [21] A.M. Bagirov and N.K. Gadjiev. A monotonous method for unconstrained lipschitz optimization. In R. De Leone, A. Murli, P.M. Pardalos, and G. Toraldo, editors, *High Performance Algorithms and Soft-*

- ware in *Nonlinear Optimization*, number 24 in Applied Optimization, pages 25–42. Kluwer Academic Publisher, Dordrecht, 1998.
- [22] A.M. Bagirov and A.A. Gasanov. A method of approximating a quasidifferential. *Russian Journal of Computational Mathematics and Mathematical Physics*, 35:403–409, 1995.
- [23] A.M. Bagirov and A.M. Rubinov. Cutting angle method and a local search. *Journal of Global Optimization*, 27:193–213, 2003.
- [24] A.M. Bagirov, A.M. Rubinov, N. Soukhoroukova, and J. Yearwood. Supervised and unsupervised data classification via nonsmooth and global optimisation. *TOP: Spanish Operations Research Journal*, 11:1–93, 2003.
- [25] A.M. Bagirov, A.M. Rubinov, and J. Yearwood. A global optimisation approach to classification. *Optimization and Engineering*, 3:129–155, 2002.
- [26] A.M. Bagirov, A.M. Rubinov, and J.P. Zhang. Local optimization with global multidimensional search for descent. *Journal of Global Optimization*, 2004.
- [27] A.M. Bagirov, A.M. Rubinov, and J.P. Zhang. A new multidimensional descent method for global optimization. *Computational Optimization and Applications*, 2005.
- [28] A.M. Bagirov and J. Zhang. Hybrid simulated annealing and discrete gradient method for global optimisation. In L. Caccetta and V. Rehbock, editors, *Industrial Optimisation, Vol 1, Proceedings of Symposium in Industrial Optimisation, Western Australian Centre of Excellence in Industrial Optimisation (WACEIO)*, Perth. Australia, 2003. Curtin University.
- [29] G. Beliakov, A. Bagirov, and J. E. Monsalve. Parallelization of the discrete gradient method of non-smooth optimization and its applications. In P.M.A. Sloot, D. Abramson, A.V. Bogdanov, J. Dongarra, A.Y. Zomaya, and Y.E. Gorbachev, editors, *Proceedings of the 3rd International Conference on Computational Science, part III*, volume 2659, pages 592–601. Springer-Verlag, 2003.
- [30] K.P. Bennett and O.L. Mangasarian. Robust linear programming discrimination of two linearly inseparable sets. *Optimization Methods and Software*, 1:23–34, 1992.

-
- [31] G.L. Bilbro and W.E. Snyder. Optimization of functions with many minima. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(4):840–849, 1991.
- [32] H.H. Bock. *Automatische Klassifikation*. Vandenhoeck & Ruprecht, Gottingen, 1974.
- [33] H.H. Bock. Clustering and neural networks. In *Advances in Data Science and Classification*, pages 265–277. Springer-Verlag, Berlin, a.rizzi, m. vichi and h.h. bock (eds) edition, 1998.
- [34] I.O. Bohachevsky, M.E. Johnson, and M.L. Stein. Generalized simulated annealing for function optimization. *Technometrics*, 28:209–217, 1986.
- [35] Paul S. Bradley and Usama M. Fayyad. Refining initial points for k-means clustering. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 91–99. Morgan Kaufmann Publishers Inc., 1998.
- [36] P.S. Bradley and O.L. Mangasarian. Feature selection via concave minimization and support vector machines. In J. Shavlik, editor, *Machine Learning Proceedings of the Fifteenth International Conference ICML'98*, pages 82–90, San Francisco, California, 1998. Morgan Kaufmann.
- [37] P.S. Bradley and O.L. Mangasarian. k-Plane Clustering. Technical Report MP-TR-1998-08, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1998.
- [38] P.S. Bradley, O.L. Mangasarian, and W.N. Street. Clustering via Concave Minimization. In Michael C. Mozer, Michael I. Jordan, and Thomas Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 368–374. The MIT Press, 1997.
- [39] D.G. Brooks and W.A. Verdini. Computational experience with generalized simulated annealing over continuous variables. *American Journal of Mathematical and Management Sciences*, 8:425–449, 1988.
- [40] D.E. Brown and C.L. Entail. A practical application of simulated annealing to the clustering problem. *Pattern Recognition*, 25:401–412, 1992.

-
- [41] A.G. Buckley and A. LeNir. Qn-like variable storage conjugate gradients. *Mathematical Programming*, 27:155–175, 1983.
- [42] C.J.C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.
- [43] D. Byatt, I. Coope, and C. Price. A convergent variant of the nelder-mead algorithm. *Journal of Optimization Theory and Applications*, 113(1):5–19, 2002.
- [44] R.H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal of Scientific Computing*, 16(5):1190–1208, 1995.
- [45] R.H. Byrd, J. Nocedal, and R.B. Schnabel. Representation of quasi-newton matrices and their use in limited memory methods. *Mathematical Programming*, 63:129–156, 1994.
- [46] V. Černý. Thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.
- [47] E.W. Cheney and A.A. Goldstein. Newton’s method for convex programming and tchebycheff approximation. *Numerische Mathematik*, 1:253–268, 1959.
- [48] F.H. Clarke. Generalized gradients and applications. *Transactions of the American Mathematical Society*, 205, 1975.
- [49] F.H. Clarke. *Optimization and Nonsmooth Analysis*. Wiley, New York, 1983.
- [50] L. Clemow, M. Hannah, D.Nash, M.Mamedov, A. Rubinov, J.Ugon, and J.Yearwood. Feature ordering for small datasets. In L. Caccetta, editor, *Industrial Optimization*. Kluwer Academic Publishers, Dordrecht, 2005.
- [51] H. Cohn and M. Fleming. Simulated annealing: searching for an optimal temperature schedule. *Journal of Global Optimization*, 9(3):779–802, 1999.
- [52] B. Colson and Ph.L. Toint. A derivative-free algorithm for sparse unconstrained optimization problems. In A.H. Siddiqi and M. Kocvara,

- editors, *Trends in Industrial and Applied Mathematics*, pages 131–147, Dordrecht, 2002. Kluwer Academic Publishers.
- [53] A.R. Conn, N.I.M. Gould, and Ph.L. Toint. *Trust Region Methods*. SIAM, Philadelphia, 2000.
- [54] A.O. Converse. *Optimization*. Hold, Rinehart and Winston, New York, 1970.
- [55] A. Corana, M. Marchesi, C. Martini, and S. Ridella. Minimizing multimodal functions of continuous variables with the “simulated annealing” algorithm. *ACM Transactions of Mathematical Software*, 13:262–280, 1987.
- [56] M. Cotrell and J.C. Fort. Étude d’un processus d’auto-organisation. *Annales de l’Institut Poincar’e*, 23(1):1–20, 1987.
- [57] A. Dekkers and E.H.L. Aarts. Global optimization and simulated annealing. *Mathematical Programming*, 50:367–393, 1991.
- [58] A.V. Demyanov, V.F. Demyanov, and V.N. Malozemov. Min-max-min problems revisited. *Optimization Methods and Software*, 17(5):783–804, 2002.
- [59] V. Demyanov and A. Rubinov, editors. *Quasidifferentiability and Related Topics*, volume 43 of *Nonconvex Optimization and Its Applications*. Kluwer Academic, Dordrecht/Boston/London, 2000.
- [60] V.F. Demyanov and A.M. Rubinov. *Constructive Nonsmooth Analysis*. Peter Lang, Frankfurt am Main, 1995.
- [61] V.F. Demyanov, G.E. Stavroulakis, L.N. Polyakova, and P.D. Panagiotopoulos. *Quasidifferentiability and Nonsmooth Modelling in Mechanics, Engineering and Economics*. Water Science and Technology Library. Kluwer Academic Publisher, Dordrecht, 1996.
- [62] J.E. Dennis, D.M. Gay, and R.E. Welsch. An adaptive nonlinear least squares algorithm. *ACM Transactions on Mathematical Software*, 7(3):348–368, 1981.
- [63] I.S. Dhillon, J. Fan, and Y. Guan. Efficient clustering of very large document collections. In V. Kumar R. Grossman, C. Kamath and R. Namburu, editors, *Data Mining for Scientific and Engineering Applications*, chapter 20. Kluwer Academic Publishers, Dordrecht, 2001.

- [64] G. Diehr. Evaluation of a branch and bound algorithm for clustering. *SIAM Journal Scientific and Statistical Computing*, 6:268–284, 1985.
- [65] O. du Merle, P. Hansen, B. Jaumard, and N. Mladenovic. An interior point method for minimum sum-of-squares clustering. *SIAM Journal on Scientific Computing*, 21:1485–1505, 2001.
- [66] R. Dubes and A.K. Jain. Clustering techniques: the user’s dilemma. *Pattern Recognition*, 8:247–260, 1976.
- [67] E. Erwin, K. Obermayer, and K. Schulten. Convergence properties of self-organizing maps. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks*, pages 409–414. Elsevier, Amsterdam, Netherlands, 1991.
- [68] E. Erwin, K. Obermayer, and K. Schulten. Self-organizing maps: Ordering, convergence properties and energy functions. *Biological Cybernetics*, 67(1):47–55, 1992.
- [69] E. Erwin, K. Obermayer, and K. Schulten. Self-organizing maps: Stationary states, metastability and convergence rate. *Biological Cybernetics*, 67(1):47–55, 1992.
- [70] Y.G. Evtushenko. A numerical method for finding best guaranteed estimates. *USSR Journal of Computational Mathematics and Mathematical Physics*, 2:109–128, 1972.
- [71] M. Fielding. Simulated annealing with an optimal fixed temperature. *SIAM Journal of Optimization*, 11(2):289–307, 2000.
- [72] R.A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, VII part II:179–188, 1936. Reprinted in R.A. Fisher, *Contributions to Mathematical Statistics*, Wiley, 1950.
- [73] J.A. Flanagan. Self-organisation in the one-dimensional som with a reduced width neighbourhood. In *Proceedings of WSOM’97, Workshop on Self-Organizing Maps*, pages 268–273, Espoo, Finland, 1997. Helsinki University of Technology, Neural Networks Research Centre.
- [74] R. Fletcher. *Practical Methods of Optimization*. Wiley, Chichester - New York - Brisbane - Toronto, second edition, 1987.
- [75] M. Frank and P.H. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3:95–110, 1956.

- [76] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 6(6):721–741, 1984.
- [77] N. Gershenfeld. *The Nature of Mathematical Modelling*. Cambridge University Press, Cambridge, 1999.
- [78] R. Ghosh, A.M. Rubinov, and J. Zhang. Optimisation approach for clustering datasets with weights. *Optimization Methods and Softwares*, 20, 2006.
- [79] GAMS Software GmbH. General Algebraic Modeling System (GAMS).
- [80] S.M. Goldfeldt, R.E. Quandt, and H.F. Trotter. Maximization by quadratic hill-climbing. *Econometrica*, 34:541–551, 1966.
- [81] A. Griewank and P.L. Toint. Partitioned variable metric updates for large structured optimization problems. *Numerische Mathematik*, 39:119–137, 1982.
- [82] I. Guyon and A. Elisseeff. An introduction to Variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.
- [83] M. Haarala, K. Miettinen, and M.M. Mäkelä. New limited memory bundle method for large-scale nonsmooth optimization. *Optimization Methods and Software*, 19(6):673–692, December 2004.
- [84] P. Hanjoul and D. Peeters. A comparison of two dual-based procedures for solving the p -median problem. *European Journal of Operational Research*, 20:387–396, 1985.
- [85] E.R. Hansen. *Global Optimization Using Interval Analysis*. Dekker, New York, 1992.
- [86] P. Hansen and B. Jaumard. Cluster analysis and mathematical programming. *Mathematical Programming*, 79(1-3):191–215, 1997.
- [87] P. Hansen and N. Mladenovic. j -means: a new heuristic for minimum sum-of-squares clustering. *Pattern Recognition*, 4:405–413, 2001.
- [88] P. Hansen and N. Mladenovic. Variable neighborhood decomposition search. *Journal of Heuristic*, 7:335–350, 2001.
- [89] P. Hansen, E. Ngai, B.K. Cheung, and N. Mladenovic. Analysis of global k -means, an incremental heuristic for minimum sum-of-squares clustering. *Les Cahiers du Gerad*, August 2002. submitted.

-
- [90] D.M. Hawkins, M.W. Muller, and J.A. ten Krooden. Cluster analysis. In *Topics in Applied Multivariate Analysis*, pages 337–340. Cambridge University press, Cambridge, 1982.
- [91] M.D. Hebden. An algorithm for minimization using exact second derivatives. Technical Report T.P. 515, AERE Harwell Laboratory, Harwell, Oxfordshire, England, 1973.
- [92] A.R. Hedar and M. Fukushima. Hybrid simulated annealing and direct search method for nonlinear unconstrained global optimization. *Optimization Methods and Software*, 17(5):891–912, 2002.
- [93] A.R. Hedar and M. Fukushima. Simplex coding genetic algorithm for the global optimization of nonlinear functions. Technical report, Department of Applied Mathematics and Physics, Kyoto University, 2002.
- [94] J.B. Hiriart-Urruty and C. Lemaréchal. *Convex Analysis and Minimization Algorithms, Part 1: Fundamentals*. Grundlehren der mathematischen Wissenschaften. Springer-Verlag, Berlin, 1993.
- [95] J.B. Hiriart-Urruty and C. Lemaréchal. *Convex Analysis and Minimization Algorithms, Part 2: Advanced Theory and Bundle Methods*. Grundlehren der mathematischen Wissenschaften. Springer-Verlag, Berlin, 1993.
- [96] R. Horowitz and L. Alvarez. Self-organizing neural networks: convergence properties. In *Proceedings of ICNN'96, IEEE International Conference on Neural Networks*, volume 1, pages 7–12, New York, USA, 1996. IEEE.
- [97] R. Horst and P.M. Pardalos, editors. *Handbook of Global Optimization*. Kluwer Academic Publishers, Dordrecht, Netherlands, 1995.
- [98] R. Horst, P.M. Pardalos, and N.V. Thoai. *Introduction to Global Optimization*. Kluwer Academic Publishers, Dordrecht, Netherlands, 1995.
- [99] R. Horst and H. Tuy. *Global Optimization: Deterministic Approaches*. Springer-Verlag, Berlin, 3rd edition edition, 1996.
- [100] ILOG. www.ilog.com.
- [101] L. Ingber. Very fast simulated re-annealing. *Mathematical and Computer Modelling*, 12:97–118, 1989.

-
- [102] L. Ingber. Simulated annealing: Practice versus theory. *Mathematical and Computer Modelling*, 18(11):29–57, 1993.
- [103] L. Ingber. Adaptive simulated annealing (asa): lessons learned. *Control and Cybernetics*, 25(1):33–54, 1996.
- [104] A.K. Jain, M.N. Murty, and P.J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.
- [105] R.E. Jensen. A dynamic programming algorithm for cluster analysis. *Operations Research*, 17:1034–1057, 1969.
- [106] A.E.W. Jones and G.W. Forbes. An adaptive simulated annealing algorithm for global optimization over continuous variables. *Journal of Global Optimization*, 6:1–37, 1995.
- [107] R.B. Kearfott. *Rigorous Global Search: Continuous Problems*. Kluwer Academics Publisher, Dordrecht, Netherlands, 1996.
- [108] J.E. Kelley. The cutting plane method for solving convex programs. *journal of the SIAM*, 8:703–712, 1960.
- [109] C.T. Kelly. Detection and remediation of stagnation in the nelder-mead algorithm using a sufficient decreasing condition. *SIAM Journal of Optimization*, 10:43–55, 1999.
- [110] Y.S. Kim, W.N. Street, and F. Menczer. Feature Selection in Data Mining. In J. Wang, editor, *Data Mining: Opportunities and Challenges*, pages 80–105. Idea Group Publishing, Hershey, 2003.
- [111] C. Kirjner-Neto and E. Polak. On the conversion of optimization problems with max-min constraints to standard optimization problems. *SIAM Journal of Optimization*, 8(4):887–915, 1998.
- [112] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.
- [113] K.C. Kiwiel. Methods of descent for nondifferentiable optimization. In *Lecture Notes in Mathematics*, volume 1133. Springer-Verlag, Berlin, 1985.
- [114] K.C. Kiwiel. Efficiency of proximal bundle methods. *Journal of Optimization Theory and Applications*, 104:589–603, 2000.

- [115] T. Kohonen. Self-organized formation of generalized topological maps of observations in a physical system. Report TKK-F-A450, Helsinki University of Technology, Espoo, Finland, 1981.
- [116] T. Kohonen. Self-organizing formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69, 1982.
- [117] T. Kohonen. *Self-Organizing Maps*, volume 30 of *Springer Series in Information Sciences*. Springer, Berlin, Heidelberg, 1995. (Third Extended Edition 2001).
- [118] W.L.G. Koontz, P.M. Narendra, and K. Fukunaga. A branch and bound clustering algorithm. *IEEE Transactions on Computers*, 24:908–915, 1975.
- [119] S.S. Kutateladze and A.M. Rubinov. Minkowski duality and its applications. *Russian Mathematical Surveys*, 27:137–191, 1972. In Russian.
- [120] S.S. Kutateladze and A.M. Rubinov. Minkowski duality and its applications. Nauka, Novosibirsk, 1976. In Russian.
- [121] V. Kvasnicka and J. Pospichal. A hybrid of simplex method and simulated annealing. *Chemometrics and Intelligent Laboratory Systems*, 39:161–173, 1997.
- [122] C. Lemaréchal. Nonsmooth optimization and descent methods. Technical Report RR-78-004, International Institute for Applied System Analysis, Laxemburg, Austria, 1978.
- [123] K. Levenberg. A method for the solution of certain problems in least squares. *Quarterly Journal on Applied Mathematics*, 2:164–168, 1944.
- [124] A. Likas, N. Vlassis, and J.J. Verbeek. The Global K-Means Clustering Algorithm. *Pattern Recognition*, 36(2):451–461, 2003.
- [125] K.F. Lim, G. Beliakov, and L.M. Batten. Predicting molecular structures: Applications to the cutting angle method. *Physical Chemistry Chemical Physics*, 5:3884–3890, 2003.
- [126] D.C. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45:503–528, 1989.
- [127] M. Locatelli. Convergence properties of simulated annealing for continuous global optimization. *Journal of Applied Probability*, 33:1127–1140, 1996.

-
- [128] M. Locatelli. Convergence of a simulated annealing algorithm for continuous global optimization. *Journal of Global Optimization*, 18:219–233, 2000.
- [129] M. Locatelli. Simulated annealing algorithms for continuous global optimization: convergence conditions. *Journal of Optimization Theory and Applications*, 104:121–133, 2000.
- [130] M. Locatelli. Convergence and first hitting time of simulated annealing algorithms for continuous global optimization. *Mathematical Methods of Operations Research*, 54(2):171–199, 2001.
- [131] M. Locatelli. Simulated annealing algorithms for continuous global optimization. In *Handbook of Global Optimization II*, pages 179–230. Kluwer Academic Publishers, Dordrecht, 2002.
- [132] S. Lucidi and M. Piccioni. Random tunneling by means of acceptance-rejection sampling for global optimization. *Journal of Optimization Theory and Applications*, 62:255–277, 1989.
- [133] L. Luksan and J. Vlcek. Sparse and partially separable test problems for unconstrained and equality constrained optimization. Technical Report 767, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, 1999.
- [134] L. Lukšan. Numerical methods for unconstrained optimization. Technical Report DMSIA 97/12, Università degli Studi di Bergamo, Bergamo, Italy, 1997.
- [135] L. Lukšan and J. Vlček. Introduction to nonsmooth analysis. Theory and algorithms. serie didattica DMSIA 00/1, Università degli Studi di Bergamo, Bergamo (Italy), 2000.
- [136] L. Lukšan and J. Vlček. Test problems for nonsmooth unconstrained and linearly constrained optimization. Technical Report 798, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, 2000.
- [137] L. Lukšan and J. Vlček. NDA: Algorithms for nondifferentiable optimization. *Transactions on Mathematical Software*, 27:193–213, 2001.
- [138] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297. University of California Press, 1965.

-
- [139] M. Mäkelä. Survey of bundle methods for nonsmooth optimization. *Optimization Methods and Softwares*, 17(1):1–29, 2002.
- [140] M.M. Mäkelä and P. Neittaanmäki. *Nonsmooth Optimization*. World Scientific, Singapore, 1992.
- [141] D. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *SIAM Journal on Applied Mathematics*, 11:431–441, 1963.
- [142] K.I.M. McKinnon. Two strictly convex examples where the nelder-mead simplex method converges to a non-stationary point. Technical report, Department of Mathematics and Computer Sciences, University of Edinburgh, Edinburgh, 1995.
- [143] K.I.M. McKinnon. Convergence of the nelder-mead simplex method to a nonstationary point. *SIAM Journal on Optimization*, 9(1):148–158, 1999.
- [144] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller. Equation of state calculation by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.
- [145] Donald Michie, D. J. Spiegelhalter, and C. C. Taylor, editors. *Machine Learning, Neural and Statistical Classification*. Series in Artificial Intelligence. Ellis Horwood, London, 1994.
- [146] R. Mifflin. Semismooth and semiconvex functions in constrained optimization. *SIAM Journal on Control and Optimization*, 15(6):959–972, 1977.
- [147] B. Mirkin. *Mathematical Classification and Clustering*. Kluwer Academic Publishers, Dordrecht, 1996.
- [148] J.J. Moré. The levenberg-marquardt algorithm: implementation and theory. In *Numerical Analysis, Dundee 1977*, number 630 in Lecture Notes in Mathematics, pages 105–116. Springer-Verlag, Berlin, 1978.
- [149] J.J. Moré and D.C. Sorensen. Computing a trust region step. *SIAM Journal on Scientific and Statistical Computing*, 4(3):553–572, 1983.
- [150] P.M. Murphy and D.W. Aha. UCI repository of machine learning databases. Technical report, Department of Information and Computer Science, University of California, Irvine, 1992.

-
- [151] J.A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.
- [152] A. Neumaier. *Interval Methods for Systems of Equations*. Cambridge University Press, Cambridge, England, 1990.
- [153] J. Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of Computation*, 35(151):773–782, 1980.
- [154] J. Nocedal. Large scale unconstrained optimization. In *The State of the Art in Numerical Analysis.*, pages 311–338. Oxford University Press, Oxford, 1997.
- [155] N.R. Pal, J.C. Bezdek, and E.C.-K. Tsao. Generalized clustering networks and kohonen’s self organizing scheme. *IEEE Transactions on Neural Networks*, 4:549–557, 1993.
- [156] D. Pallaschke and S. Rolewicz. *Foundations of Mathematical Optimization: Convex Analysis Without Linearity*, volume 388 of *Mathematics and Its Applications*. Kluwer Academic Publishers, New York, February 1997.
- [157] J.D. Pintér. *Global Optimization in Action*. Kluwer, Dodrecht, Netherlands, 1995.
- [158] Pintér consulting services. LGO Software Development.
- [159] Pintér consulting services. www.pinterconsulting.com.
- [160] E. Polak. *Optimization: Algorithms and Consistent Approximations*. Springer, New York, 1997.
- [161] B. T. Polyak. *Introduction to Optimization*. Optimization Software, Inc., Publications Division, New York, 1987.
- [162] M.J.D. Powell. Direct search algorithms for optimization calculations. *Acta Numerica*, 7:287–336, 1998.
- [163] M.J.D. Powell. UOBYQA: unconstrained optimization by quadratic approximation. *Mathematical Programming*, 92(3):555–582, 2002.
- [164] M.J.D. Powell. On the use of quadratic models in unconstrained minimization without derivatives. Technical report, Cambridge University, Cambridge, 2003.

- [165] M.J.D. Powell. On trust region methods for unconstrained minimization without derivatives. *Mathematical Programming*, 97:605–623, 2003.
- [166] M.J.D. Powell. The NEWUOA software for unconstrained optimization without derivatives. Technical report, Cambridge University, Cambridge, 2004.
- [167] W.H. Press and S.A. Teutolsky. Simulated annealing optimization over continuous spaces. *Computers in Physics*, 5(4):426–429, 1991.
- [168] H. Ratschek and J.G. Rokne. Interval methods. In Horst and Pardalos [97], pages 751–828.
- [169] C.R. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems*. Wiley & Sons, Incorporated, John, Blackwell, London, 1993.
- [170] G. Reinelt. Tsp-lib-a traveling salesman library. *ORSA Journal on Computing*, 3:319–350, 1991.
- [171] R.T. Rockafellar. *Convex Analysis*. Princeton University Press, Princeton, New Jersey, 1970.
- [172] S. Rolewicz. Convex analysis without linearity. *Control and Cybernetics*, 23:247–256, 1994.
- [173] H.E. Romeijn and R.L. Smith. Simulated annealing for constrained global optimization. *Journal of Optimization Theory and Applications*, 191(2):403–427, 1994.
- [174] A.M. Rubinov. *Abstract Convexity and Global Optimization*. Kluwer Academic Publishers, New York, 2000.
- [175] A.M. Rubinov. Abstract convexity: examples and applications. *Optimization*, 47:1–33, 2000.
- [176] A.M. Rubinov and M.Y. Andramonov. Minimizing increasing star-shaped functions based on abstract convexity. *Journal of Global Optimization*, 15:19–39, 1999.
- [177] S. Selim and M. Ismail. K-means-type algorithms: A generalized convergence theorem and characterization of local optimality. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(1):81–87, 1984.

- [178] S.Z. Selim and K.S. Al-Sultan. A simulated annealing algorithm for the clustering. *Pattern Recognition*, 24(10):1003–1008, 1991.
- [179] P. Siarry, G. Berthiau, F. Durbin, and J. Haussy. Enhanced simulated annealing for globally minimizing functions of many continuous variables. *ACM Transactions on Mathematical Software*, 23(2):209–228, 1997.
- [180] I. Singer. *Abstract Convex Analysis*. Wiley-Interscience Publication, New York, 1997.
- [181] N. Soukhoroukova. *Data classification through nonsmooth optimization*. PhD thesis, ITMS, University of Ballarat, Australia, Australia, 2003.
- [182] H. Späth. *Cluster Analysis Algorithms for Data Reduction and Classification of Objects*. Ellis Horwood Limited, Chichester, 1980.
- [183] L.X. Sun, Y.L. Xie, X.H. Song, J.H. Wang, and R.Q. Yu. Cluster analysis by simulated annealing. *Computers and Chemistry*, 18:103–108, 1994.
- [184] Ivar Tammeraid, Jüri Majak, Seppo Pohjolainen, and Tero Luodeslampi. Application of Linear Algebra.
- [185] J. Thorsten. *Learning to Classify Text using Support Vector Machines*. Kluwer Academic Publishers, Dordrecht, 2002.
- [186] P.L. Toint. On sparse and symmetric matrix updating subject to a linear equation. *Mathematics of Computation*, 31(140):954–961, 1977.
- [187] P. Tseng. Fortified-descent simplicial search method: A general approach. *SIAM Journal of Optimization*, 10(1):269–288, 1999.
- [188] University of Ballarat and Pintér consulting. www.ciao-go.com.au.
- [189] P.J.M. Van Laarhoven. *Theoretical and Computational Aspects of Simulated Annealing*. Centrum voor Wiskunde en Informatik, Amsterdam, 1988.
- [190] P.J.M. Van Laarhoven and E.H.L. Aarts. *Simulated Annealing: Theory and Applications*. Kluwer Academic Publisher, Dordrecht, 1987.
- [191] V.N. Vapnik. *Estimation of Dependencies Based on Empirical Data*. Nauka, Moscow, 1979. In Russian.

-
- [192] V.N. Vapnik. *Estimation of Dependencies Based on Empirical Data*. Springer Verlag, Berlin, 1982. English translation.
- [193] V.N. Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, New-York, 1995.
- [194] V.N. Vapnik. *Statistical Learning Theory*. John Wiley and Sons, New York, 1998.
- [195] R.V.V. Vidal, editor. *Applied Simulated Annealing*, volume 396 of *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, Berlin, Heidelberg, New York, 1993.
- [196] P.P. Wang and D.S. Chen. Continuous optimization by a variant of simulated annealing. *Computational Optimization and Applications*, 6:59–71, 1996.
- [197] P.H. Wolfe. A method of conjugate subgradients for minimizing nondifferentiable functions. *Mathematical Programming Studies*, 3:145–173, 1975.
- [198] P.H. Wolfe. Finding the nearest point in a polytope. *Mathematical Programming*, 11(2):128–149, 1976.
- [199] H. Xu, A.M. Rubinov, and B.M. Glover. Continuous approximations to generalized jacobians. *Optimization*, 46:221–246, 1999.
- [200] J. Yen, J.C. Liao, B. Lee, and D.S. Chen. A hybrid approach to modeling metabolic systems using a genetic algorithm and simplex method. *IEEE Transactions on Systems, Man, and Cybernetics, part B*, 28(2):173–191, 1998.
- [201] K.F.C. Yiu, Y. Liu, and K.L. Teo. A hybrid descent method for global optimization. *Global Optimization*, 27:229–238, 2003.
- [202] J. Zhang. *Derivative-free hybrid methods in global optimization and their applications*. PhD thesis, ITMS, University of Ballarat, Australia, 2005.