UNIVERSITÀ DI PARMA
DIPARTIMENTO DI INGEGNERIA E ARCHITETTURA

Dottorato di Ricerca in Tecnologie dell'Informazione
XXXVII Ciclo

Andrea Fois

OPTIMIZATION MODELS AND METHODS FOR MAXIMUM LIKELIHOOD ESTIMATION OF STATISTICAL PARAMETERS

DISSERTAZIONE PRESENTATA PER IL CONSEGUIMENTO
DEL TITOLO DI DOTTORE DI RICERCA

ANNI ACCADEMICI 2021/2022 - 2023/2024

UNIVERSITÀ DI PARMA
UNIVERSITÀ DEGLI STUDI DI PARMA

Dottorato di Ricerca in Tecnologie dell'Informazione

Ciclo XXXVII

OPTIMIZATION MODELS AND METHODS FOR MAXIMUM LIKELIHOOD ESTIMATION OF STATISTICAL PARAMETERS

Coordinatore:

Chiar.mo Prof. Marco Locatelli

Tutor:

Chiar.mo Prof. Luca Consolini

Chiar.mo Prof. Fabrizio Laurini

Dottorando: *Andrea Fois*

Anni Accademici 2021/2022 - 2023/2024

To my first dog Zuzu

Summary

List of Figures

List of Tables

Introduction

The work presented in this thesis is aimed at providing new models and methods for the Maximum Likelihood Estimation of the parameters of statistical models. In particular, the work is focused on two closely related subfields: the clustering of Mixtures of Gaussians and the Clusterwise Linear Regression.

Clustering Mixtures of Gaussians

Clustering methods aim at grouping similar data points together, while maximizing the dissimilarity between different groups. This is a fundamental task in data analysis and statistical learning, which plays a pivotal role in various domains, including medicine [1], image analysis [2], customer segmentation [3], and social network analysis [4] among others. Traditional model-based clustering algorithms, such as `k-means` [5] and `mclust` [6], have made significant contributions to the field. The theoretical guarantees for the recovery of the clusters with SemiDefinite Programming (SDP) relaxations of the Gaussian Mixture Models have been extensively studied in [7], [8], [9], [10], [11]. However, the ever-increasing complexity and dimensionality of data pose new challenges that call for innovative approaches. In particular, traditional approaches rely on resampling techniques as a part of an *Expectation-Maximization* (EM) algorithm [12], which requires evaluating a large number of subsamples as the model dimension or the number of groups increase. Moreover, only a local optimum can be achieved, which can be substantially far from the global one if one does not consider a sufficient number of subsamples.

In Chapter 1, we propose a novel clustering method that leverages semi-definite programming (see, e.g., [13]) to address these challenges and to enhance the accuracy of model-based clustering methods. Our clustering method, `bclust`, follows the common approach of maximizing the log-likelihood of a Gaussian Mixture Model. We find a (local) maximum by solving a set of Mixed Integer Semidefinite Programming (MISDP) problems. In this way, we are able to refine an initial assignment. We will show that this approach improves the log-likelihood objective function, compared to traditional EM-type algorithms, at the cost of a higher computational burden. Moreover, it offers better solutions in terms of assignments (e.g., in terms of the Adjusted Rand Index (*ARI*; [14])). In principle, the proposed method could also be employed to detect the global optimal solution. However, computing times grow exponentially with respect to the data set cardinality. Hence, in general, finding the global optimum is not possible in real-world applications. Even though the employment of artificial neural networks to solve Mixed-Integer Problems is an active area of research [15] we decided to instead rely on classic approaches to obtain some (local) optimality guarantees. Extensive Monte Carlo simulations and real-world applications demonstrate the effectiveness of our approach in tackling clustering problems.

Chapter 1 is organized as follows. In Section 1.1, we review the relevant literature on model-based clustering under Gaussian assumptions, and we describe existing methods based on semi-definite programming. Section 1.2 details our `bclust` proposal based on MISDP. Section 1.3 highlights its effectiveness through Monte Carlo simulations, and real-world applications are described in Section 1.4.

Clusterwise Linear Regression

Mixture models offer a powerful probabilistic framework for capturing the inherent heterogeneity in a population, where data may arise from unobserved subpopulations with distinct characteristics. In the realm of supervised learning, Clusterwise Regression Models (CRM) provide a valuable tool for uncovering relationships within such heterogeneous populations [16]. They model the relationships between a response and a set of explanatory variables coming from several latent components,

and their diverse applicability extends across various domains, including engineering [17], economics [18], environmental sciences [19], and sociology [20]. Unlike traditional regression models, which assume a single underlying relationship for all data points, CRM techniques aim to group the data into distinct clusters, each governed by its own regression model.

Chapter 2 focuses on clusterwise linear regression [21], a specific class of CRM well-suited for modeling linear relationships within clustered data arising from a Gaussian mixture. Estimating the parameters of such models presents a significant challenge due to the complexity of the parameter space and the potential presence of multiple local optima. Existing optimization methods primarily rely on the Expectation-Maximization (EM) algorithm [22], which, due to its local search nature, typically converges to local optima and does not allow the assessment of the solution quality. To address these issues, several mathematical optimization approaches have been put forward to estimate the parameters of a wide spectrum of models [23].

The focus of this chapter is on the maximization of the likelihood of Gaussian Mixture Models. Existing approaches try to find either a local optimum without any quality guarantees or the global optimum. On the other hand, our approach `pclustreg` finds solutions that, with a desired probability $s < 1$ are guaranteed to be as good as the unknown ground-truth with respect to the value of the objective function. This is achieved under minimal assumptions on the true model parameters. The method is based on a probabilistic branch-and-bound approach that exploits the distribution of the sum of squared residuals to compute good probabilistic bounds. In addition, we show, through experimental tests on synthetic and real-world data, that our approach can also be employed as a competitive heuristic to find good solutions while keeping the computational costs under control.

The content of Chapter 2 has been presented at the conference ODS2024 and will be included in the proceedings, in a volume of the AIRO Springer Series with the prospective title “Operations Research: closing the gap between research and practice”. The version reported in this thesis is reproduced with permission from Springer Nature.

Chapter 2 is structured as follows. Section 2.1 describes the model and its as-

sumptions and formulates the problem. Section 2.2 introduces our approach and presents some theoretical results on its performance. Section 2.3 reports on experiments carried on synthetic datasets. Section 3.3 presents a real-world use-case scenario that highlights the advantages of our approach.

The limitations of the approach described in Chapter 2 are highlighted and dealt with a new proposal in Chapter 3. In particular, the main difference lies in the use of a different probability distribution. Moreover, Chapter 3 presents thorough testing of such improved version of `pclustreg` on synthetic datasets. Chapter 3 is structured as follows: Section 3.1 shows the modified upper bound method and the proof of its correctness, Section 3.2 illustrates the performance of the new version of `pclustreg` with synthetic tests in different scenarios, Section 3.4 reports and discusses relevant implementation details.

Chapter 1

A Mixed Integer Semidefinite Programming Approach for Cluster Analysis in Gaussian Mixture Models

Considerate la vostra semenza:
fatti non foste a viver come bruti
ma per seguir virtute e canoscenza

– Dante Alighieri

1.1 Background

Clustering techniques aim at grouping a set of observations $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^p$ into $k \geq 2$ homogeneous groups in an unsupervised manner. It is customary to assume the presence of multivariate normal components and then derive the *Maximum Likelihood Estimator* (MLE). In this chapter, we focus on the *classification likeli-*

hood approach which, unlike the mixture likelihood, provides a crisp assignment of each unit to a certain component. Namely, in the Gaussian classification likelihood framework we maximize the following log-likelihood:

$$\max_{\mathscr{H},\{\mu_j,\Sigma_j\}_{j=1,\dots,k}} \sum_{j=1}^{k} \sum_{x_i \in H_j} \log\{\pi_j \phi(x_i;\mu_j,\Sigma_j)\}, \tag{1.1}$$

where $\mathscr{H} = \{H_1,\dots,H_k\}$ is a partition of the units labels $\{1,\dots,n\}$ into k clusters, π_j are positive weights such that $\sum_{j=1}^{k} \pi_j = 1$, and $\phi(\cdot;\mu_j,\Sigma_j)$ is the probability density function of a p-variate normal distribution with mean $\mu_j \in \mathbb{R}^p$ and covariance matrix $\Sigma_j \in \mathbb{R}^{p\times p}$. Namely:

$$\phi(x_i;\mu_j,\Sigma_j) = (2\pi)^{-\frac{p}{2}} \det(\Sigma_j)^{-1/2} \exp\left[-\frac{1}{2}(x_i-\mu_j)^T \Sigma_j^{-1}(x_i-\mu_j)\right],$$

where $\det(\cdot)$ denotes the matrix determinant. Hence, clusters are ellipsoidal, centered at μ_j, and their geometric features depend on the covariances Σ_j.

However, without additional constraints on the Σ_j's, the optimization problem in (1.1) may lead to degenerate solutions [24]. Constraints on their determinants or on the scatter matrices themselves could be used [25, 16], which are linked to the well-known Gaussian parsimonious model family [26]. However, they do not guarantee the absence of degenerate solutions (e.g., some eigenvalues of Σ_j may remain close to zero), although all determinants are bounded away from zero. To address this problem, [27] proposed the `tclust` algorithm for clustering with mixture of Gaussians in the presence of outliers, and includes a constraint on the overall maximum ratio between the eigenvalues of all the covariance matrices. A fast algorithm to enforce the constraint is presented in [28]. Moreover, [29] recently proposed to control not only the overall ratio among the eigenvalues of all the covariance matrices Σ_j, but also the ratio of the eigenvalues for each distinct Σ_j. Specifically, consider the decomposition of Σ_j, for $j = 1,\dots,k$, as

$$\Sigma_j = \lambda_j \Omega_j \Gamma_j \Omega_j^T$$

where $\det(\Sigma_j) = \lambda_j^p$, Ω_j is an orthogonal matrix of eigenvectors, Γ_j is a diagonal matrix composed by the elements $\{\gamma_{j1},\dots,\gamma_{jp}\}$, which are proportional to the eigenvalues of Σ_j's, and such that $\det(\Gamma_j) = 1$. Here, the Γ_j's are denoted as shape matrices

since they determine the shape of the resulting clustering partition. Shape constraints are thus enforced as

$$\frac{\max_{h=1,\dots,p} \gamma_{jh}}{\min_{h=1,\dots,p} \gamma_{jh}} \leq r^2$$

for an arbitrary constant $r \geq 1$.

The resulting optimization problem is typically tackled through the so-called *Classification EM* (CEM) algorithm [30, 12].

1.1.1 Clustering and Mixed-Integer SDP approaches

In this subsection, we review the literature on Mixed Integer Semidefinite Programming (MISDP) approaches to model-based clustering, in settings similar to the one discussed in this chapter. In [31], the authors propose a theoretical framework based on the 0-1 SDP reformulation of the Minimal Sum Of Squares Clustering (MSSC for short, which is an instance of the Gaussian Mixture Model where all covariance matrices are set equal to the identity). In [32], the authors propose a linear relaxation of the 0-1 SDP reformulation, which can be used to approximate the optimal solution of MSSC. In [33], the authors extend the work of [31] on the MSSC problem to a branch-and-cut scheme that solves LP relaxations of the 0-1 SDP reformulation with additional cutting planes. The proposed algorithm is able to recover the exact solution to the MSSC problem in a variety of small datasets. In [34], the authors present a novel branch and bound and cut algorithm that is shown to be able to solve exactly instances of real-world datasets of up to 4000 data points. Lower bounds are computed by solving SDP problems through the recent first-order SDP solver `SDPNAL+` [35]. Bounds are also improved through the addition of different cutting planes. In [36], a branch and cut approach based on the solution of SDP relaxations of the cardinality constrained MSSC is proposed and shown to exactly solve instances up to 10 times larger than the ones solved by other state-of-the-art exact solvers. In all these papers the distance between points is the Euclidean one, so that the identified clusters tend to have a spherical shape. A few other approaches deal with ellipsoidal clusters. In [37], the authors propose an approach for a covering problem. They use a MISDP reformulation that is similar to the one proposed in this chapter, but that only minimizes

the sum of the volumes of the covering ellipsoids. They also introduce linear constraints to avoid the degeneration of ellipses into lines. They use a branch and bound approach to solve the problem and precompute a good upper bound through various algorithms, including `k-means` and subsampling. In [38], the LA-SDP (likelihood adjusted SDP) approach is proposed. This approach alternates between the solutions of two maximization problems. The first one is an SDP relaxation with fixed covariance matrices (basically, the same relaxation as the one for MSSC, but with the Mahalanobis distance replacing the Euclidean distance), and an assignment of points to cluster is derived from the solution of the relaxation. The solution of the second maximization problem is given in closed form, and allows one to derive new covariance matrices based on the current cluster assignments.

1.1.2 Statement of contribution

The main contributions of this chapter are the following ones.

- We introduce a reformulation of the clustering problem as a Mixed Integer Convex Semidefinite Programming problem. The reformulation, which considers the Mahalanobis distance (ellipsoidal clusters), is more general than existing ones [33, 31, 32, 34, 36], that consider the Euclidean distance (spherical clusters). With respect to [37], we aim at reconstructing the parameters of a mixture of Gaussian distributions through likelihood maximization, rather than optimizing a covering of the points.

- Based on this reformulation, we propose a new optimization approach, `bclust`, where `b` stands for binary, since it is based on binary variables. This is an alternate maximization approach, where, at each iteration:

  - First, we fix a cluster assignment, and compute optimal covariance matrices and centroids, through the solution of a nonlinear convex SDP problem. To avoid degenerate ellipsoids we impose conditions on the ratio of the eigenvalues of the covariance matrices.

– Then, we explore a neighborhood of the current cluster assignment by changing the association to cluster of some suitably chosen points.

The work of [38] studies a problem that is similar to the one studied in this chapter (it is the same except for the constraints on the eigenvalues of the matrices), but proposes a different approach. Indeed, we solve SDPs to set the values of the continuous variables (covariances and centroids), while, in [38], SDP problems are solved to set the values of the combinatorial variables (assignment of points to clusters).

- The effectiveness of the proposed approach is demonstrated on both synthetic and real-world datasets.

1.2 Our proposal: `bclust`

We consider the special case of (1.1) with equal weights (i.e., $\pi_1 = \pi_2 = \ldots = \pi_k = \frac{1}{k}$). Similarly to the approach in [29], we include the following constraints

$$\lambda_{\max}(\Sigma_j^{-1}) \leq r^2 \lambda_{\min}(\Sigma_j^{-1}), \quad j = 1, \ldots, k,$$

for some $r \geq 1$. If we substitute the definition of ϕ into the objective function, add the constraint on the eigenvalues, and switch from a maximization to a minimization problem by changing the sign of the objective function, we obtain the following optimization problem:

$$\begin{aligned} \min_{\mathscr{H}, \{m_j, \Sigma_j\}_{j=1}^k} \quad & \sum_{j=1}^k \sum_{x_i \in H_j} \log(2\pi) - \log((\det(\Sigma_j))^{-1/2}) + \tfrac{1}{2}(x_i - \mu_j)' \Sigma_j^{-1} (x_i - \mu_j) \\ \text{s.t.} \quad & \lambda_{\max}(\Sigma_j^{-1}) \leq r^2 \lambda_{\min}(\Sigma_j^{-1}) \quad j = 1, \ldots, k \\ & \Sigma_j \succ O \quad j = 1, \ldots, k, \end{aligned}$$

where $\Sigma_j \succ O$ imposes that Σ_j is a positive definite matrix. In order to reformulate it into a Mixed Integer Convex Problem with semidefinite constraints, we first introduce the change of variables $M_j^2 = \Sigma_j^{-1}$, $j = 1, \ldots, k$, with $M_j \in \mathbb{R}^{p \times p}$ and $M_j \succ 0$. This

leads to the equivalent formulation:

$$\begin{array}{lll} \min_{\mathscr{H},\{m_j,M_j\}_{j=1}^k} & \sum_{j=1}^k \sum_{x_i \in H_j} \log(2\pi) - \log\det(M_j) + \frac{1}{2}(x_i-\mu_j)' M_j^2 (x_i-\mu_j) & \\ \text{s.t.} & \lambda_{\max}(M_j) \leq r\lambda_{\min}(M_j) & j=1,\ldots,k \\ & M_j \succ O & j=1,\ldots,k. \end{array}$$

Next, we introduce the new vectors of variables $y_i \in \mathbb{R}^p$, $i = 1,\ldots,n$, and $z_j \in \mathbb{R}^p$, $j = 1,\ldots,k$, defined as follows:

$$y_i = M_j x_i,\ x_i \in H_j$$

$$z_j = M_j \mu_j,\ j = 1,\ldots,k.$$

Since

$$\begin{aligned} (x_i-\mu_j)' M_j^2 (x_i-\mu_j) &= (x_i-\mu_j)' M_j^T M_j (x_i-\mu_j) = (M_j x_i - M_j\mu_j)'(M_j x_i - M_j\mu_j) = \\ &= (y_i - z_j)'(y_i - z_j) = \|y_i - z_j\|^2, \end{aligned}$$

we can replace the vectors of variables μ_j with z_j, $j = 1,\ldots,k$, and reformulate the optimization problem as follows:

$$\begin{array}{lll} \min_{y,z,M,\mathscr{H}} & \sum_{j=1}^k \sum_{x_i \in H_j} \log(2\pi) - \log\det(M_j) + \frac{1}{2}\|y_i - z_j\|^2 & \\ \text{s.t.} & \lambda_{\max}(M_j) \leq r\lambda_{\min}(M_j) & j=1,\ldots,k \\ & M_j \succ O & j=1,\ldots,k \\ & y_i = M_j x_i & \text{if } x_i \in H_j. \end{array}$$

For a fixed partition $\mathscr{H}$, this problem has a convex objective function because both $-\log\det(M_j)$ (see, e.g., Example 3.23 in [39]) and $\|y_i - z_j\|^2$ are convex functions. Moreover, constraint $\lambda_{\max}(M_j) \leq r\lambda_{\min}(M_j)$ is convex since function $\lambda_{\max}(M_j)$ is convex and function $\lambda_{\min}(M_j)$ is concave (see, e.g., Example 3.10 in [39]). Finally, $M_j \succ O$ is a convex positive definite constraint. Note that such constraint can also be relaxed into $M_j \succeq O$, i.e., the requirement that M_j is positive semidefinite, since term $-\log\det(M_j)$ in the objective function prevents from having an optimal solution with

null determinant.

Up to now, we have treated sets H_j, $j = 1,\ldots,k$, as optimization variables. We can make their definition explicit by introducing the binary variables $c \in \{0,1\}^{n\times k}$, where

$$c_{ij} = \begin{cases} 1 & \text{if } i \in H_j \\ 0 & \text{otherwise.} \end{cases}$$

We need to introduce the following linear constraints, enforcing that each point is assigned to exactly one cluster, and that each cluster is not empty, respectively:

$$\begin{aligned} &\textstyle\sum_{j=1}^{k} c_{ij} = 1 \quad i = 1\ldots,n \\ &\textstyle\sum_{i=1}^{n} c_{ij} \geq 1 \quad j = 1,\ldots,k. \end{aligned} \tag{1.2}$$

We also need to introduce n additional vectors of variables $\bar{z}_i \in \mathbb{R}^p$ and n additional matrices of variables $\bar{M}_i \in \mathbb{R}^{p\times p}$ that are to be made equal, respectively, to z_j and M_j of the cluster j if $i \in H_j$. More precisely, we need to impose the following implications:

$$\begin{aligned} c_{ij} = 1 \quad &\Rightarrow \quad \bar{z}_i = z_j \qquad i = 1,\ldots,n,\ j = 1,\ldots,k \\ c_{ij} = 1 \quad &\Rightarrow \quad \bar{M}_i = M_j \quad i = 1,\ldots,n,\ j = 1,\ldots,k. \end{aligned}$$

In general, a logical constraint $b = 1 \Rightarrow a = c$ where b is a binary variable, can be rewritten as a couple of linear constraints with a Big-M reformulation:

$$a - c \leq (1-b)B,\ \ c - a \leq (1-b)B,$$

where B is a sufficiently large constant. Indeed, if $b = 1$ the two constraints reduce to $a - c = 0$, while if $b = 0$ the two constraints are $a - c \leq B$ and $c - a \leq B$, which, for a big enough constant B (in particular, B should be bigger than $|a - c|$ in the optimal solution) do not pose restrictions on the optimal value of the objective function. If we apply this reformulation to the two constraints above, we obtain the following linear

constraints:

$$\begin{aligned}
&\bar{z}_i - z_j \leq (1-c_{ij})B_z && i=1,\dots,n,\ j=1,\dots,k\\
&-\bar{z}_i + z_j \leq (1-c_{ij})B_z && i=1,\dots,n,\ j=1,\dots,k\\
&\bar{M}_i - M_j \leq (1-c_{ij})B_M && i=1,\dots,n,\ j=1,\dots,k\\
&-\bar{M}_i + M_j \leq (1-c_{ij})B_M && i=1,\dots,n,\ j=1,\dots,k.
\end{aligned} \tag{1.3}$$

The choice of B_z and B_M requires special care. Values that are too small could exclude the optimal solution from the feasible region of the problem, and values that are too high may lead to numerical problems. The rule we chose to set these values will be discussed in Section 1.2.3. The model can thus be reformulated as follows, after removing the constant term $n\log(2\pi)$:

$$\begin{aligned}
&\min_{y,z,M,\bar{z},\bar{M},c} && \textstyle\sum_{i=1}^{n}\left[-\log\det(\bar{M}_i) + \frac{1}{2}\|y_i-\bar{z}_i\|^2\right] && &\\
&\text{s.t.} && (1.2),\ (1.3) && &\\
& && \lambda_{\max}(M_j) \leq r\lambda_{\min}(M_j) && j=1,\dots,k\\
& && M_j \succeq O && j=1,\dots,k\\
& && y_i = \bar{M}_i x_i && i=1,\dots,n.
\end{aligned} \tag{1.4}$$

Problem 1.4 is a Mixed-Integer Semidefinite Programming (MISDP) problem with convex objective function and constraints.

Remark

In order to ensure that the mathematical formulation is free of all possible singularities it is necessary to add additional constraints that prevent the determinant of each M_j and $\bar{M}_i$ from approaching infinity. One possibility is to control the relative sizes of all the clusters determinants analogously to [29] (taking into account that $M=\Sigma^{-1}$ and thus the minimum eigenvalue of Σ^{-1} is the maximum one of M and vice versa). A second avenue consists of controlling the relative sizes of the maximum eigenvalues of each M. A third way to remove all singularities is to add linear constraints limiting

the sizes of the maximum eigenvalue of each M. All these constraints, and in particular the second and third proposals are simple to implement and do not change the class of the resulting optimization problem. It is worth noting that the singularity we want to avoid is caused by one of the Σ approaching 0 while maintaining the relative sizes of its eigenvalues. In our formulation this happens only if the corresponding M matrix approaches infinity while maintaining the relative sizes of its eigenvalues. This is prevented by the Big-M constraints 1.3, since B_M is finite. For the sake of simplicity we do not employ either of the suggested solutions in the rest of this manuscript and instead rely on the Big-M constraints to avoid singularities.

1.2.1 Restricted problem and iterative local search algorithm

Current solvers for MISDP use branch-and-bound approaches. In general, their computational cost increases exponentially with the number of integer variables. In particular, Problem (1.4) has nk binary variables c_{ij}. Due to this large number, in real-life problems, the exact solution of Problem (1.4) is computationally unfeasible. To overcome this problem, we present a local search strategy, based on the reduction of the number of binary variables in Problem (1.4). Namely, we solve a sequence of MISDP Problems (1.4). In each problem, we hold most binary variables c_{ij} to the value of the previous iteration. We use a heuristic to decide the subset of variables c_{ij} that we can change in current iteration.

In more detail, assume a feasible solution is available, with binary variables values $\tilde{c}_{ij}$, $i = 1,\dots,n$, $j = 1,\dots,k$. Let $A \subset X$ be the set of assigned points (i.e., the set of points for which we do not want to change the current assignment to a cluster). This can be imposed by adding the following constraints to (1.4):

$$c_{ij} = \tilde{c}_{ij}, \quad i \in A,\ j = 1,\dots,k. \tag{1.5}$$

Note that fixing these binary variables also allows setting variables $\bar{z}_i$ and $\bar{M}_i$ equal to z_j and M_j, respectively, if $\tilde{c}_{ij} = 1$. If we denote by $F = X \setminus A$ the set of 'free' points (i.e., those for which the assignment to a cluster is not fixed), then the number of binary variables is reduced from nk to $|F|k$. Provided that the cardinality of F is chosen

sufficiently small, the resulting problem can be solved by existing solvers in a reasonable time. In summary, for a given binary solution $\tilde{c}$ and $F \subset X$ (or, equivalently, $A = X \setminus F$), the restricted problem is the following:

$$\begin{array}{lll}
\min_{y,z,M,\bar{z},\bar{M},c} & \sum_{i=1}^{n} \left[-\log\det(\bar{M}_i) + \frac{1}{2}\|y_i - \bar{z}_i\|^2 \right] & \\
\text{s.t.} & (1.2),\ (1.3),\ (1.5) & \\
& \lambda_{\max}(M_j) \leq r\lambda_{\min}(M_j) & j = 1,\dots,k \qquad (\mathscr{P}(F,\tilde{c})) \\
& M_j \succeq O & j = 1,\dots,k \\
& y_i = \bar{M}_i x_i & i = 1,\dots,n.
\end{array}$$

In what follows, given an optimal solution $(y^\star, z^\star, M^\star, \bar{z}^\star, \bar{M}^\star, c^\star)$ of problem $\mathscr{P}(F,\tilde{c})$, we set:

$$\begin{array}{l}
\Sigma = (\Sigma_1,\dots,\Sigma_k), \quad \Sigma_j = \left(M_j^\star\right)^{-2}, \quad j = 1,\dots,k \\
\mu = (\mu_1,\dots,\mu_k), \quad \mu_j = \left(M_j^\star\right)^{-1} \gamma_j^\star, \quad j = 1,\dots,k.
\end{array} \tag{1.6}$$

Each binary feasible solution c defines a partition of $\{1,\dots,n\}$ into k clusters $H_1,\dots,H_k$. For a given c, we set $j_i = j$ if $c_{ij} = 1$ (i.e., j_i is the index of the cluster to which point x_i belongs, according to c). Now, the question is how to select the subset F. To this end, we assign a score to each point in X. More precisely, given $x_i \in X$, some feasible binary vector c, and μ, Σ defined according to (1.6), we set:

$$\begin{array}{l}
\texttt{contr}_{i,j} = -\frac{1}{2}(x_i - \mu_j)'\Sigma_j^{-1}(x_i - \mu_j) + \log(\det(\Sigma_j)^{-1/2}) \\
\texttt{score}(i) = -\texttt{contr}_{i,j_i} + \max_{l \in \{1,\dots,k\} \wedge l \neq j_i} \texttt{contr}_{i,l}.
\end{array} \tag{1.7}$$

The value $\texttt{contr}_{\texttt{i,j}}$ is the contribution of the point x_i to the log-likelihood if it is assigned to the cluster j, provided that the mean μ_j and covariance matrix Σ_j do not change (in case j is different from the cluster to which the point x_i is currently assigned). The value $\texttt{score(i)}$ is an estimate of the best change in the log-likelihood if the point x_i is assigned to a different cluster, and $\tilde{l}_i$ is the index of the cluster which gives the best contribution $\texttt{score(i)}$. It should be noted that the value of $\texttt{score(i)}$ is always lower than or equal to the actual change in the log-likelihood because it does

not take into account how the means and covariance matrices would change (and they can only change in a way that further increases the log-likelihood). Because of this, it can happen that the value of `score(i)` is negative (i.e., it predicts a decrease in the value of the log-likelihood), but the value of the log-likelihood actually increases when the point is re-assigned. On the other hand, if `score(i)` > 0, then the log-likelihood certainly increases when the point is re-assigned to the cluster

$$\tilde{l}_i \in \arg \max_{l \in \{1,\ldots,k\} \wedge l \neq j_i} \texttt{contr}_{i,l}.$$

The idea behind the heuristic for the local search is to rank the points according to their scores and, eventually, try to re-assign the points with highest scores. Indeed, they are the ones with the best chances to increase the log-likelihood function.

1.2.2 `bclust` algorithm

We are now ready to describe our iterative clustering algorithm, called `bclust`, which is described in Algorithm 1. First, given an initial valid cluster assignment c_0, we solve problem $\mathscr{P}(\emptyset, c_0)$, and store in Obj the optimal objective value. We define vector μ and the collection of matrices Σ according to (1.6) (line 2). Next, in the outer `while` loop (at lines 4-19), we enter the inner `while` loop (at lines 7-13). Here, at first, we compute vector $\tilde{q}$ containing, for every point, the index of the cluster that yields the best individual contribute for that point (line 8). We solve problem $\mathscr{P}(\emptyset, \tilde{q})$, and compute the new optimal means and covariance matrices (stored in $n\mu, n\Sigma$) from (1.6) (line 9). In case the new solution improves the current one, we update the current solution (line 11), otherwise we exit the inner `while` loop (line 13). Once we exit the inner `while`, which implements the standard classification EM algorithm (see also Remark 1), at line 14, we define, according to some policy that we will discuss later, a collection $\mathscr{F}$ of subset of points. Then, we enter the `for` loop at lines 15-19, where, for each member F of the collection, we solve Problem $\mathscr{P}(F, c)$, and use its solution to compute $n\mu, n\Sigma, nObj, nc$ (line 16). In case the objective function value $nObj$ (the optimal value of problem $\mathscr{P}(F, c)$) is lower than Obj, the best value observed so far (line 17), we update Obj, μ, Σ, c, and record an improvement

(line 18). We iterate the outer `while` loop at lines 4-19 until its execution does not improve the objective function (i.e., `improved` is not set to 1). The output of the algorithm is the current binary vector c, that represents the partition of X into k clusters.

1.2.3 Implementation details

We now briefly discuss the choice of B_z and B_M. When solving the problem $\mathscr{P}(\emptyset, c)$ at lines 2 and 9, there is no need to set specific values for such constants. Indeed, all binary variables are set in advance, and the set of free points is empty (i.e., $F = \emptyset$). Instead, when solving the problem $\mathscr{P}(F, c)$ at line 16, we need to choose suitable values for B_z and B_M. In our experiments we set:

$$B_z = 2 \max_{j=1,\ldots,k} \|z_j\|_\infty, \;\; B_M = 2 \max_{j=1,\ldots,k} \|M_j\|_\infty,$$

where z_j and M_j, $j = 1,\ldots,k$, are those corresponding to the current best known solution. Empirically, we found that this is a good choice, because it avoids numerical problems without cutting away the optimal solution from the feasible region. The rationale is that, when the number of points to be re-assigned is small, the values of z and M can only change by small amounts. It is worthwhile to make the following remark. When solving the full problem (1.4), we need to guarantee that the value chosen for B_z and B_M do not cut away optimal solutions. But when we solve the subproblems $\mathscr{P}(F, c)$ at line 16, cutting an optimal solution does not compromise the execution of the `bclust` algorithm, which may simply be unable to detect an improvement when this is instead possible. Therefore, the choice of suitable values for B_z and B_M is less critical in this case.

Remark 1. *It is worthwhile to remark that the first inner* `while` *loop in* `bclust` *implements the standard classification EM algorithm used by* `tclust` *(in the absence of outliers), but with a different definition of matrices Σ_j and vectors μ_j, $j = 1,\ldots,k$. Indeed, by solving exactly the SDP problem $\mathscr{P}(\emptyset, c)$ for a given assignment c, we are*

Algorithm 1: Routine `bclust`

input : X, c_0, d, D

output: c

$c = c_0$;
Solve problem $\mathscr{P}(\emptyset, c)$ and set Obj equal to its optimal value; compute μ, Σ from (1.6);
Set $improved = 1$;
while $improved$ **do**
    Set $improved = 0$;
    Set $exitWhile = 0$;
    while $not(exitWhile)$ **do**
        Compute $\tilde{q} = (\tilde{q_1}, \ldots, \tilde{q_n})$ with $\tilde{q_i} = \arg\max_{j \in \{1,\ldots,k\}} contr_{i,j}$ for $i \in \{1, \ldots, n\}$;
        Solve problem $\mathscr{P}(\emptyset, \tilde{q})$ and set $nObj$ equal to its optimal value; compute $n\mu, n\Sigma$ from (1.6);
        if $nObj < Obj$ **then**
            Set $c = \tilde{q}$, $Obj = nObj$, $\mu = n\mu$,$\Sigma = n\Sigma$, $improved = 1$;
        if $nObj \geq Obj$ **then**
            $exitWhile = 1$;
    Generate, according to some rule, a collection $\mathscr{F}$ of subsets of points;
    for $F \in \mathscr{F}$ **do**
        Solve problem $\mathscr{P}(F, c)$ and set $nObj$ equal to its optimal value, nc equal to the optimal c vector, and compute $n\mu, n\Sigma$ from (1.6);
        if $nObj < Obj$ **then**
            Set $improved = 1$, $Obj = nObj$, $\mu = n\mu$, $\Sigma = n\Sigma$, $c = nc$;
            Break `For` loop;
return c;

able to compute matrices Σ_j and vectors μ_j, $j = 1,\ldots,k$, which fulfill the constraints on the eigenvalue ratios, and maximize the likelihood, under the current assignment c. The second loop, the inner `for` *loop, implements a heuristic based on the exact solution of restricted MISDP problems, through which one can often escape the local minimum found by the EM algorithm. Because of this, our approach can be seen as a generalization of the classification EM algorithm.*

Remark 2. *Our* `bclust` *proposal takes an initial assignment as the starting point, and tries to refine it by improving the objective function. The initial assignment can be generated randomly, but the most effective way to employ the* `bclust` *algorithm is to initialize it with the solution of another algorithm (in our experiments* `tclust`*; see Sections 1.3-1.4). It is possible to start* `bclust` *from an ensemble of initial random assignments, but this is computationally expensive due to the high number of MISDP problems to be solved. It turns out to be better to run* `bclust` *one or few times, seeding it with carefully selected initial assignments, such as those returned by other clustering algorithms, like* `tclust`*, whose solutions are then refined by* `bclust`.

In `bclust`, we optimize both with respect to continuous variables (Σ_j and μ_j, for $j = 1,\ldots,k$), and with respect to binary variables (c, which define the assignment of points to clusters). Obviously, for a fixed set of binary variables c, the covariance matrices Σ_j and the centroids μ_j for $j \in \{1,\ldots,k\}$ are the optimal ones, under the constraints on the ratio of the eigenvalues. This is true since we solve exactly SDP problems $\mathscr{P}(\emptyset, c)$. Concerning the combinatorial part, we can prove the following local optimality result.

Proposition 1. *Algorithm* `bclust`*, described in Algorithm 1, terminates, and its output c is a (locally) optimal solution of Problem $\mathscr{P}(F,\tilde{c})$, with respect to the neighborhood defined by the collection of subsets $\mathscr{F}$.*

Proof. Proposition 1 By contradiction, assume that `bclust` does not terminate. This implies that `bclust` updates variable Obj an infinite number of times. Denote by O_i the corresponding sequence of values for Obj, and, by c_i, the corresponding values for c. Sequence O_i is monotone strictly decreasing. Hence, the values of sequence c_i

are all distinct. Since c_i belong to a finite set, there does not exist an infinite sequence of distinct values for c_i, leading to a contradiction. The second statement is trivial. Indeed, `bclust` terminates only if none of the possible points reassignments in $\mathscr{F}$ leads to an improvement of the objective function value. □

Choice of r

As said, parameter r constrains the eigenvalues of the covariance matrices of the clusters. A full discussion of the choice of r is outside the scope of this work. However, we briefly discuss a possible way to tackle such choice. One could adopt an automated procedure involving an extension of the BIC [40], introduced in [41]. This can be used to select the value of r by executing the routine `bclust` with multiple values of r, computing the BIC on the obtained solutions, and choosing the value of r which gives the lowest BIC. In fact, rather than selecting a single value r, one could also return all solutions computed by the routine `bclust`, with different values of r. Then, the user could choose among them, possibly exploiting some additional knowledge about the problem.

1.2.4 Hyperparameters overview

In the discussion of `bclust` we have made use of several hyperaparameters. For the sake of clarity and usability of this manuscript we list them in this section and provide a brief description for each one. The purpose of this list is to offer a bird-eye view over all the hyperparameters at play and in-depth discussions presented in previous sections regarding each hyperparameter are not repeated here.

- k: a numerical hyperparameter equal to the number of clusters. If the number of clusters is known beforehand than it is actually a fixed number.
- r: a numerical hyperaparameter that controls the ratio of the maximum eigenvalue to the minimum eigenvalue of each cluster matrix. It must satisfy $r \geq 1$.
- B_z, B_M: Big-M reformulation constants. They must be big enough to not cut the optimal solutions while also small enough to avoid numerical problems.

- the algorithm to compute $\mathscr{F}$: how to obtain the family of sets of points defining the neighborhood of the current solution to search into for improvements in the objective function.

- the algorithm used to obtain the starting point: it can be a random starting point or it can be obtained through other methods such as `tclust`, and in the latter case whe should also consider the hyperparameters of the method used.

1.3 Numerical experiments on synthetic data

As said, `bclust` can be considered a generalization of the classification EM algorithm. The proposed local search procedure allows one to improve the log-likelihood with respect to an initial assignment. In our numerical experiments, we used `tclust` to find an initial cluster assignment. Then, we ran `bclust` to improve the log-likelihood. In the following, we first present the clustering methods that we compared. Then, we introduce the metrics used to evaluate such methods. Finally, we present four different numerical experiments. We ran the experiments on a machine with an Intel Xeon Gold 6238R CPU with 64 Gigabytes of RAM.

1.3.1 Compared methods

We compared the following methods:

- `bclust`: this is built considering a collection $\mathscr{F}$ of subsets of points made up of singletons. Namely,

$$\mathscr{F} = \big\{\{x_{i_j}\} \ : \ j = 1,\dots,h\big\},$$

  i.e., $\mathscr{F}$ contains h singletons, each one containing one of the first h points ordered according to the scores. In particular, we set $h = \frac{n}{10}$, i.e., we considered the best 10% of points according to the score value. This way, in the `For` loop at lines 15-19 of the `bclust` algorithm, we perform an exhaustive search for a point to re-assign among the best 10% points.

- `tclust`: it is a clustering algorithm presented in [29] and implemented in the FSDA MATLAB toolbox presented in [42]. Since it is a robust clustering algorithm it can detect outliers. Our model can be naturally extended to support the detection of outliers by treating them as an additional cluster with points not contributing to the objective function and a cardinality constraint to fix its size. This extension is left out of this work and is planned for future ones. Since in our settings outliers are not considered, we disabled this functionality. The restriction factor in `tclust` allows imposing the same constraints on the eigenvalues of the shape matrices as in `bclust`.

- A `random` cluster assignment which we used as a baseline for the value of the objective function.

- An `oracle` benchmark which relies on the knowledge of the true assignments. This is an unfeasible estimation strategy which is only used as a benchmark.

Choice of $\mathscr{F}$ in `bclust`

In `bclust`, the choice of family $\mathscr{F}$ is particularly critical. By Proposition 1, if we increase the cardinality of $\mathscr{F}$, we enlarge the local search set, and improve the quality of the final solution. However, obviously, we also increase the computational time. We tried different strategies to choose $\mathscr{F}$. Essentially, we considered two alternative policies. In Policy 1, given an integer constant $u > 1$, we set $\mathscr{F}$ equal to the family of h sets of cardinality u, whose score sum is highest, where the score of a set I is equal to $\sum_{i \in I} \texttt{score(i)}$. In Policy 2, we chose a family $\mathscr{F}$ composed of h singletons, consisting in the h points with the highest score. Note that Policy 2 corresponds to Policy 1 with $u = 1$. With this choice, Problem $\mathscr{P}(F, \tilde{c})$ has only k binary variables. Higher values of h lead to better results, but also to higher computational times.

Overall, Policy 2 gives the best results, in terms both of computational times, and quality of the obtained solution. In particular, the computational time of Policy 1 is larger because the corresponding elements of $\mathscr{F}$ have a larger cardinality.

In general, the number of possible assignments and, thus, the time needed to solve Problem $\mathscr{P}(F,\tilde{c})$, grow exponentially with respect to the number of free assignments.

Also, in general, for equal values of h, Policy 2 gives solutions of better quality with respect to Policy 1. This is due to the fact that, in Policy 1, most subsets of cardinality $u > 1$ in $\mathscr{F}$ contain just different combinations of the points with highest score. Hence, overall, Policy 1 tests less points for reassignment with respect to Policy 2. This makes more difficult to improve the current solution.

A third possible policy consists in using Policy 1, with values of u from 1 up to an upper cardinality $\bar{u}$. In general, this policy improves the quality of the solution, since it increases the cardinality of $\mathscr{F}$, but at the price of very large computational times.

Based on the previous discussion, we decided to run our tests with Policy 2. We tried different strategies to choose h. First, we simply set $h = n$. In this way, we choose the point to re-assign by performing an exhaustive search among all points. Obviously, this is the choice that allows us to find the best value for the objective function, but at the cost of a large computational time. At the end, as said, we set $h = \frac{n}{10}$. Overall, according to our experiments, this choice is a good compromise between clustering performance and computational time.

1.3.2 Performance metrics

To compare different clustering methods, we used the following metrics:

- The value of the objective function (Obj);
- The Adjusted Rand Index (ARI) [14], computed with the `RandIndexFS` function of FSDA, which can only be used in scenarios in which the true labels are already known (i.e., only for benchmarking) since its computation involves their values, it is a value between -1 and 1 in which 1 is perfect recognition and 0 is the expected value for a random assignment;
- The computational time in seconds ($t(s)$).

1.3.3 Generation of synthetic data

We ran some experiments on synthetic clustering problems, based on random data obtained from a mixture of Gaussian distributions. The number of points is the same in each test, and is equal to $n = 200$. We varied the following parameters: the number of clusters k, the dimensionality of the dataset p, the restriction factor r^2, the degree of overlap between different clusters $\overline{\omega}$. For each combination of parameters, we generated the dataset with a mixture of Gaussian distributions using FSDA functions `MixSim` and `simdataset`. First, we used `MixSim` to generate the mixture of Gaussian distributions with a "*restriction factor*" of r^2 (the maximum ratio between the eigenvalues of the covariance matrices of the Gaussians) and the *average overlap* $\overline{\omega}$. Then, we used `simdataset` function to generate n points, and their corresponding labels, according to the distribution generated with `MixSim`. We conducted experiments in four different scenarios. In each scenario, one of the parameters of the simulation was changed while the others were kept fixed. For each combination of parameters, we generated 100 runs with different random number generator seeds. Tables 1.1-1.4 contain simulation results for the four scenarios of interest. Here, for each of the considered methods and performance metrics, we report their means (with standard deviations in parenthesis) across the 100 random runs.

Simulation scenarios

We consider the following simulation scenarios:

- *Scenario 1: varying dimensionality.* We let the dimensionality of the dataset p take the values of $2,4,6,8,10$, with the other parameters fixed: $k = 4$, $r^2 = 10$, $\overline{\omega} = 0.01$.

- *Scenario 2: varying number of clusters.* We let the number of clusters k take the values of $2,3,4,5,6$ with the other parameters fixed: $p = 8$, $r^2 = 10$, $\overline{\omega} = 0.01$.

- *Scenario 3: varying restriction factor.* We let the restriction factor r take the values of $r^2 = 1,4,6,10,30$ with the other parameters fixed: $p = 8$, $k = 4$, $\overline{\omega} = 0.01$. Importantly, the value of r used by `bclust` (see Problem $\mathscr{P}(F,\tilde{c})$) and

Table 1.1: Means and standard deviations of the performance metrics for Scenario 1

Algorithm	p	Obj	ARI	$t(s)$
`random`	2	-339.0557 (134.1729)	0.0016 (0.0073)	1.209e-05 (1.120e-06)
`tclust`	2	-1006.0851 (262.1487)	0.9521 (0.0257)	1.138 (0.188)
`bclust`	2	**-1006.2542 (262.1063)**	**0.9512(0.0277)**	**36.546 (14.222)**
`oracle`	2	-1001.6165 (262.0300)	1 (-)	- (-)
`random`	4	-1223.0374 (181.3359)	8.9679e-05 (0.0074)	1.333e-05 (9.217e-07)
`tclust`	4	-1825.0235 (258.3451)	0.9474 (0.0293)	1.957 (0.196)
`bclust`	4	**-1825.1927 (258.2970)**	**0.9467 (0.0285)**	**42.330 (14.818)**
`oracle`	4	-1819.4029 (258.0915)	1 (-)	- (-)
`random`	6	-2107.1684 (237.5296)	-0.0003 (0.0072)	1.33e-05 (1.605e-06)
`tclust`	6	-2697.648 (326.4845)	0.9378 (0.0333)	2.528 (0.155)
`bclust`	6	**-2698.1714 (326.2492)**	**0.9358 (0.0314)**	**52.276 (19.082)**
`oracle`	6	-2691.8776 (326.3165)	1 (-)	- (-)
`random`	8	-2938.1447 (246.5744)	-0.0002 (0.0070)	1.264e-05 (9.694e-07)
`tclust`	8	-3501.8552 (281.8193)	0.8791 (0.0715)	2.722 (0.122)
`bclust`	8	**-3509.0665 (278.1132)**	**0.9166 (0.0491)**	**93.638 (51.321)**
`oracle`	8	-3502.7442 (278.3736)	1 (-)	- (-)
`random`	10	-3806.7318 (282.8318)	0.0013 (0.0080)	1.332e-05 (1.230e-06)
`tclust`	10	-4333.3133 (321.1212)	0.6972 (0.1374)	2.652 (0.182)
`bclust`	10	**-4366.5921 (312.6707)**	**0.8101 (0.1342)**	**225.473 (141.876)**
`oracle`	10	-4373.7093 (306.9716)	1 (-)	- (-)

`tclust` is the set at the true population parameter r that was used to generate the data through the `MixSim` function.

- *Scenario 4: varying overlap probability.* In the fourth and last scenario, we let the probability of overlap $\overline{\omega}$ take the values of $0.001, 0.005, 0.01, 0.05, 0.1$ with the other parameters fixed: $p = 8$, $k = 4$, $r^2 = 10$.

Table 1.2: Means and standard deviations of performance metrics for Scenario 2

Algorithm	k	Obj	ARI	$t(s)$
`random`	2	-2944.7469 (399.9652)	0.0003 (0.0066)	1.395e-05 (2.409e-06)
`tclust`	2	-3232.8964 (401.669)	0.9668 (0.0318)	1.461 (0.108)
`bclust`	2	**-3232.9726 (401.6389)**	**0.9674 (0.0317)**	**37.936 (11.664)**
`oracle`	2	-3230.5837 (401.6715)	1 (-)	- (-)
`random`	3	-2991.2339 (352.6260)	0.0005 (0.0073)	1.370e-05 (1.872e-06)
`tclust`	3	-3448.3227 (387.2504)	0.9444 (0.0325)	2.221 (0.141)
`bclust`	3	**-3448.7562 (387.1510)**	**0.9482 (0.0321)**	**46.896 (21.362)**
`oracle`	3	-3443.7465 (387.4816)	1 (-)	- (-)
`random`	4	-2938.1447 (246.5744)	-0.0002 (0.0070)	1.275e-05 (1.789e-06)
`tclust`	4	-3501.8552 (281.8193)	0.8791 (0.0715)	2.705 (0.124)
`bclust`	4	**-3509.0665 (278.1132)**	**0.9166 (0.0491)**	**93.178 (50.911)**
`oracle`	4	-3502.7442 (278.3736)	1 (-)	- (-)
`random`	5	-2944.4348 (210.9665)	0.0012 (0.0080)	1.346e-05 (1.132e-06)
`tclust`	5	-3551.2352 (261.1739)	0.7246 (0.1158)	3.029 (0.141)
`bclust`	5	**-3576.4917 (253.7583)**	**0.7930 (0.1182)**	**198.854 (109.576)**
`oracle`	5	-3581.5550 (247.9066)	1 (-)	- (-)
`random`	6	-2916.8989 (195.2045)	-0.0019 (0.0062)	1.368e-05 (1.836e-06)
`tclust`	6	-3557.5160 (242.9329)	0.6102 (0.1114)	3.373 (0.136)
`bclust`	6	**-3591.0901 (239.9369)**	**0.6714 (0.1293)**	**278.271 (143.878)**
`oracle`	6	-3616.2225 (224.5270)	1 (-)	- (-)

Table 1.3: Means and standard deviations of performance metrics for Scenario 3

Algorithm	r^2	Obj	ARI	$t(s)$
`random`	1	-2571.0681 (573.0849)	-0.0002 (0.0071)	1.155e-05 (9.886e-07)
`tclust`	1	-3337.6301 (749.0263)	0.8896 (0.2457)	0.832 (0.197)
`bclust`	1	**-3337.7672 (748.7749)**	**0.8898 (0.2462)**	**54.416 (23.782)**
`oracle`	1	-3320.4803 (790.8764)	1 (-)	- (-)
`random`	4	-2931.2380 (231.9498)	-0.0002 (0.0070)	1.173e-05 (1.062e-06)
`tclust`	4	-3518.6125 (287.4919)	0.9262 (0.0399)	2.664 (0.198)
`bclust`	4	**-3519.0583 (287.3277)**	**0.9296 (0.0350)**	**58.192 (24.278)**
`oracle`	4	-3511.6567 (288.6207)	1 (-)	- (-)
`random`	6	-2946.8508 (233.3785)	-0.0002 (0.0070)	1.159e-05 (1.843e-06)
`tclust`	6	-3522.127 (274.0248)	0.9213 (0.0436)	2.764 (0.141)
`bclust`	6	**-3523.1632 (273.8423)**	**0.9283 (0.0420)**	**63.878 (21.864)**
`oracle`	6	-3516.2452 (274.3089)	1 (-)	- (-)
`random`	10	-2938.1447 (246.5744)	-0.0002 (0.0070)	1.188e-05 (1.855e-06)
`tclust`	10	-3501.8552 (281.8193)	0.8791 (0.0715)	2.706 (0.121)
`bclust`	10	**-3509.0665 (278.1132)**	**0.9166 (0.0491)**	**92.862 (50.801)**
`oracle`	10	-3502.7442 (278.3736)	1 (-)	- (-)
`random`	30	-2903.739 (262.7939)	-0.0002 (0.0070)	1.168e-05 (1.974e-06)
`tclust`	30	-3430.2028 (286.4097)	0.6551 (0.1461)	2.301 (0.195)
`bclust`	30	**-3462.6273 (281.2338)**	**0.7638 (0.1644)**	**181.609 (104.776)**
`oracle`	30	-3480.8847 (281.0250)	1 (-)	- (-)

Table 1.4: Means and standard deviations of performance metrics for Scenario 4

Algorithm	$\overline{\omega}$	Obj	ARI	$t(s)$
`random`	0.001	-3294.4927 (254.8724)	-0.0002 (0.0070)	1.175e-05 (1.850e-06)
`tclust`	0.001	-4056.1572 (323.7116)	0.9927 (0.0098)	2.774 (0.207)
`bclust`	0.001	**-4056.2167 (323.6546)**	**0.9918 (0.0107)**	**48.586 (6.346)**
`oracle`	0.001	-4055.3233 (323.8120)	1 (-)	- (-)
`random`	0.005	-3071.8463 (251.9810)	-0.0002 (0.0070)	1.152e-05 (1.159e-06)
`tclust`	0.005	-3703.0714 (297.7634)	0.9506 (0.0311)	2.782 (0.133)
`bclust`	0.005	**-3705.2903 (296.1618)**	**0.9603 (0.0254)**	**63.421 (24.765)**
`oracle`	0.005	-3701.4243 (296.5269)	1 (-)	- (-)
`random`	0.01	-2938.1447 (246.5744)	-0.0002 (0.0070)	1.173e-05 (1.797e-06)
`tclust`	0.01	-3501.8552 (281.8193)	0.8791 (0.0715)	2.726 (0.125)
`bclust`	0.01	**-3509.0665 (278.1132)**	**0.9166 (0.0491)**	**92.913 (50.721)**
`oracle`	0.01	-3502.7442 (278.3736)	1 (-)	- (-)
`random`	0.05	-2459.4900 (227.4940)	-0.0002 (0.0070)	1.164e-05 (1.059e-06)
`tclust`	0.05	-2844.0493 (234.0999)	0.4624 (0.1085)	2.364 (0.127)
`bclust`	0.05	**-2858.9216 (232.2707)**	**0.5111 (0.1188)**	**180.960 (96.630)**
`oracle`	0.05	-2851.5901 (233.0359)	1 (-)	- (-)
`random`	0.1	-2109.7376 (231.2522)	-0.0002 (0.0070)	1.174e-05 (1.454e-06)
`tclust`	0.1	-2438.7245 (230.3583)	0.2630 (0.0800)	2.169 (0.104)
`bclust`	0.1	**-2451.7286 (230.1846)**	**0.2807 (0.0867)**	**215.312 (122.836)**
`oracle`	0.1	-2417.0747 (230.7355)	1 (-)	- (-)

1.3.4 Simulation results

Results for simulation scenarios 1-4 are provided in Tables 1.1-1.4. We remark that Obj, ARI and $t(s)$ respectively represent the value of the objective function, the Adjusted Rand-Index, and computing time (in seconds). In each of the following tables we highlighted in bold the method that obtaines the best objective function.

For the first simulation scenario, Table 1.1 shows that `bclust` tends to outperform `tclust` as p increases. In particular, for small values of p, both methods perform very well and remain quite close to the oracle solution. At times, they even provide a better value of the objective function as they adapt to the empirical data distribution (unlike the oracle, whose ARI is equal to one by construction). However, for $p > 8$, `bclust` performs considerably better than `tclust`, both in terms of Obj and ARI, and its average computational cost remains under 4 minutes.

Table 1.2 shows that, albeit less markedly, similar conclusions hold for the second simulation scenario. Indeed, also here `bclust` outperforms `tclust` as we consider a number of components $k > 3$.

Moreover, Table 1.3 shows that `bclust` is more effective than `tclust` to tackle problems that are further from a spherical case (i.e., those ones with larger restriction factor r^2). This is likely due to the fact that `bclust` solves a sequence of SDP problems to optimality. However, also here we notice an increase in its computing time, which is still reasonable (approximately 3 minutes for $r^2 = 30$).

Table 1.4 shows that similar conclusions hold also for simulation scenario 4. Here, for very small values of the average overlap $\overline{\omega}$, clusters are far apart and very distinguishable and hence `bclust` and `tclust` perform comparably. However, for larger levels of overlap, `bclust` outperforms `tclust` in terms of objective function value and ARI.

Overall, our simulations results show that `bclust` typically improves, and often considerably, both the value of the objective function and the ARI with respect to `tclust`. This comes at the cost of a higher computational time. We remark that the gap between the two approaches increases with the dimensionality p, the number of clusters k, and the restriction factor r^2. Nevertheless, such gap is more stable for varying levels of average overlap probability $\overline{\omega}$.

1.4 Real-world applications

In this section, we present some experimental results on real-world datasets. We consider the same clustering methods used in our Monte Carlo simulations (see Section 1.3). For the assignment $\bar{c}$ returned by each algorithm, we report the value of the objective function computed by solving problem $P(\emptyset, \bar{c})$. For each dataset, we set $r^2 = 10$ and we normalize each variable.

The tests and the routines are written in MATLAB and executed in MATLAB r2022a [43], using the latest (as of May 2023) release of the YALMIP MATLAB library [44] and of the commercial SDP solver MOSEK (free for academic purposes) [45]. We performed the analyses on a machine with an Intel 7700hq CPU and 16 Gigabytes of RAM.

1.4.1 Benchmarking datasets

We consider the following well-known and publicly accessible datasets:

- *UCI Wine dataset* [46]. It has 178 data points with 13 continuous variables. Each variable describes one property of an Italian wine (e.g., the alcohol content, the alcalinity of the ash, etc.). The dataset has 3 classes, corresponding to three different wine types.

- *UCI Seeds dataset* [47]. It has 210 data points with 7 continuous variables, describing some geometric properties (e.g., area, perimeter, etc.) of some wheat kernels, and 3 classes.

- *UCI Iris dataset* [48]. It has 150 data points with 4 continuous variables, describing some geometric properties (e.g., sepal length, sepal width, etc.) of iris plants, and 3 classes for the 3 species.

- *UCI Algerian Forest Fires dataset* [49]. It consists of 244 data points split equally into two Algerian regions (Bejaia and Sidi Bel-abbes) with 12 attributes, describing the date and meteorogical measurements (e.g., temperature,

Table 1.5: Results for the application studies. Name of the dataset, number of points (n), number of variables (p), number of clusters (k), algorithm, value of the objective function (Obj), Adjusted Rand Index (ARI), and computing time (in seconds).

Dataset	n	p	k	**Algorithm**	Obj	ARI	$t(s)$
UCI Wine	178	13	3	`random`	-2.7198e+03	-0.0067	5.690e-05
				`tclust`	-3.3183e+03	0.9471	4.099
				`bclust`	**-3.3262e+03**	**0.9817**	**124.146**
				`oracle`	-3.3261e+03	1	-
UCI Seeds	210	7	3	`random`	-1.5877e+03	-0.0065	1.700e-05
				`tclust`	-2.3766e+03	0.8358	2.493
				`bclust`	**-2.3771e+03**	**0.8486**	**88.050**
				`oracle`	-2.3496e+03	1	-
UCI Iris	150	4	3	`random`	-259.9473	-0.0058	1.280e-05
				`tclust`	-658.5462	0.8858	1.965
				`bclust`	**-658.8447**	**0.9039**	**25.899**
				`oracle`	-653.8608	1	-
UCI Algerian 1	244	4	2	`random`	-1.5285e+03	-0.0072	1.043e-04
				`tclust`	-1.8820e+03	0.7529	1.847
				`bclust`	**-1.8843e+03**	**0.6965**	**43.101**
				`oracle`	-1.8542e+03	1	-
UCI Algerian 2	244	4	2	`random`	-2.1712e+03	-0.0033	3.680e-05
				`tclust`	-2.5649e+03	0.5897	2.230
				`bclust`	**-2.5651e+03**	**0.6153**	**34.536**
				`oracle`	-2.4727e+03	1	-
Star	240	4	6	`random`	-338.7210	0.0018	2.140e-05
				`tclust`	-1.7712e+03	0.6377	6.335
				`bclust`	**-1.8289e+03**	**0.6524**	**80.730**
				`oracle`	-1.9095e+03	1	-

relative humidity, etc.), and 2 classes: fire and non-fire. We provide one table with the results for each of the two regions.

- *Star dataset* (`https://www.kaggle.com/datasets/deepu1109/star-dataset`). It has 240 data points with 4 continuous variables (temperature, luminosity, radius, absolute magnitude) and 2 discrete variables (color and spectral class, which we discarded). Moreover, we have the 6 classes which represents the star type.

1.4.2 Results

In general, we observe from the results in Table 1.5 that the computing times of `bclust` are larger than those of `tclust`. But at the same time, `bclust` outperforms `tclust` both in terms of objective function value and *ARI*. In particular, for the UCI Wine dataset, the results show that `bclust` provides a clustering that is considerably better than that provided by `tclust`. In fact, the *ARI* is almost 1 and the value of the objective function is almost equal to that of the `oracle`. The UCI Algerian 2 dataset is an exception because, although the value of the objective function is improved by `bclust`, the *ARI* is decreased because if the clusters are not well separated the improvement of the value of the objective function does not necessarily lead to an improvement of the *ARI* and this is a limitation of all model-based clustering techniques. For all other datasets, the results show that `bclust` achieves better values with respect to `tclust` both in terms of *ARI* and in terms of the value of the objective function. Overall, we can state that the proposed algorithm `bclust` is capable of improving the solution computed by `tclust` in real-world applications, although at the cost of an increase in computation time.

1.5 Code availability

The code that implements `bclust` and that was used for our numerical experiments is publicly available at

https://github.com/andreaf96/bclust-paper-material.

Chapter 2

Probabilistic branch-and-bound for clusterwise linear regression

Questo creduto fu; che 'l miser suole
dar facile credenza a quel che vuole.

– Ludovico Ariosto

2.1 Problem formulation

Consider a set of n points x_i, y_i with $x_i \in \mathbb{R}^p$ and $y_i \in \mathbb{R}$. The aim of Clusterwise Linear Regression (CLR) is to fit the data with k regression hyperplanes in order to maximize the log-likelihood under appropriate conditions that allow to avoid spurious/degenerate solutions. We denote with $C = \{C_1, C_2, \ldots, C_k\}$ a partition of the indexes $\{1, 2, \ldots, n\}$ of the points into k groups. Let $\sigma = [\sigma_1, \sigma_2, \ldots, \sigma_k]^T \in \mathbb{R}^k_+$ and $\beta = [\beta_1, \beta_2, \ldots, \beta_k] \in \mathbb{R}^{p \times k}$. We assume that, for some unknown $\overline{C}, \overline{\beta}, \overline{\sigma}$ (the ground truth – which we consider unknown),

$$y_i = x_i^T \overline{\beta}_j + \varepsilon_i,\ \varepsilon_i \sim \mathcal{N}(0, \overline{\sigma}_j^2)\ \ \forall j, \forall i \in \overline{C}_j.$$

We assume that $\forall j \in \{1,2,\ldots,k\} : |\overline{C}_j| \geq p+1$. Moreover, we assume that the matrix of the independent variables X_j comprised of the points $x_i, i \in \overline{C}_j$ is full rank for all $j \in \{1,2,\ldots,k\}$. Moreover, we assume $\overline{\sigma}_j > 0, \forall j \in \{1,2,\ldots,k\}$. We refer to $\overline{C}$ as the ground-truth assignment (of points to clusters).

In a partial assignment some points are left unassigned, i.e., it is a partition of a subset of $\{1,2,\ldots,n\}$. For any (possibly partial) assignment C of points to clusters, we denote with C_j the set of indexes of the points assigned to cluster $j \in \{1,2,\ldots,k\}$ according to C. We denote with $n_j(C)$ the cardinality of cluster j according to C, i.e., $n_j(C) = |C_j|$.

The likelihood of point (x,y) belonging to cluster j for certain values of the parameters β_j, σ_j is

$$f_j(x,y) = \frac{1}{\sigma_j\sqrt{2\pi}} \exp\left(-\frac{1}{2\sigma_j^2}(x^T\beta_j - y)^2\right).$$

Now, let us assume that a full assignment C of points to clusters is given. Then, the likelihood of the data is

$$L(C) = \prod_{j=1}^{k} \prod_{i \in C_j} \pi_j f_j(x_i, y_i),$$

where π_j ($\sum_{j=1}^{k} \pi_j = 1$) are the mixing proportions of the clusters. We focus on the following optimization problem:

$$\max_{C,\beta,\sigma,\pi} \log(L(C)) = \max_{C,\beta,\sigma,\pi} \sum_{j=1}^{k} \sum_{i \in C_j} \left\{\log(\pi_j) + \log(f_j(x_i,y_i))\right\}.$$

Where $\log(L(C))$ is the log-likelihood and with the restriction that C is a partition such that $\log(L(C))$ is bounded and $\sigma_j > 0 \wedge \pi_j > 0 \; \forall j \in \{1,2,\ldots,k\}$. In particular, we have that C must be such that $|C_j| \geq p+1 \; \forall j \in \{1,2,\ldots,k\}$. After substituting the value of $f_j(x_i,y_i)$ in $\log(L(C))$, we can obtain

$$\log(L(C)) \quad = \quad -n\log\left(\sqrt{2\pi}\right) + \sum_{j=1}^{k} \left\{ n_j \log(\pi_j) - n_j \log(\sigma_j) - \frac{\sum_{i \in C_j}\left(x_i^T\beta_j - y_i\right)^2}{2\sigma_j^2} \right\}.$$

Therefore, for the given assignment C, the values of π_j, β_j, σ_j which maximize the log-likelihood, are the following (see [21, 50])

$$\hat{\pi}_j(C) = \frac{n_j(C)}{n},$$

$$\hat{\beta}_j(C) = X_j^+ Y_j,$$

$$\hat{\sigma}_j(C) = \sqrt{\frac{\mathrm{SSR}_j(C)}{n_j(C)}},$$

where X_j, Y_j are the matrices of points assigned to cluster j; $X_j^+ = (X_j^T X_j)^{-1} X_j^T$ is the left pseudoinverse of X_j and

$$\mathrm{SSR}_j(C) = \sum_{i \in C_j} (x_i^T \hat{\beta}_j(C) - y_i)^2. \tag{2.1}$$

We substitute these values into $\log(L(C))$ and drop the explicit dependency of n_j, β_j, SSR_j on C for ease of readability. Thus, we obtain

$$\begin{aligned}
\log(\hat{L}) &= -n\log(\sqrt{2\pi}) + \textstyle\sum_{j=1}^k \left\{ n_j \log(\frac{n_j}{n}) - n_j \log\left(\sqrt{\frac{\mathrm{SSR}_j}{n_j}}\right) - \frac{\mathrm{SSR}_j}{2\frac{\mathrm{SSR}_j}{n_j}} \right\} \\
&= -\tfrac{n}{2}\log(2\pi) + \textstyle\sum_{j=1}^k \left\{ n_j \log(n_j) - n_j \log(n) - \frac{n_j}{2}\log(\mathrm{SSR}_j) + \frac{n_j}{2}\log(n_j) - \frac{n_j}{2} \right\} \\
&= -\tfrac{n}{2}\log(2\pi) - \tfrac{n}{2} - n\log(n) + \textstyle\sum_{j=1}^k \left\{ -\frac{n_j}{2}\log(\mathrm{SSR}_j) + \frac{3}{2} n_j \log(n_j) \right\} \\
&= -\tfrac{n}{2}\left[\log(2\pi) + 1\right] - n\log(n) + \tfrac{1}{2}\textstyle\sum_{j=1}^k \left\{ 3n_j \log(n_j) - n_j \log(\mathrm{SSR}_j) \right\}.
\end{aligned}$$

Hence, the computationally hard part of the optimization problem is the discrete part, i.e., finding the best assignment of points to clusters $\hat{C}$. The full optimization problem is the following:

$$\max_C \ \log(\hat{L}(C)). \tag{2.2}$$

It is crucial to notice that the (unknown) ground-truth, which is what the model aims to reconstruct, is not necessarily the global optimum of Problem 2.2, as it happens in cases with overlapping densities. For this reason, the goal of this work is not to find the global optimal solution of Problem 2.2, rather we are satisfied with the detection of any solution that is at least as good as the ground-truth in terms of log-likelihood. Theorem 2.2.4 proves that the proposed approach is able to return such solution with a desired probability $s < 1$.

2.2 Our approach: `pclustreg`

The approach we propose is called `pclustreg` (probabilistic clusterwise regression). In addition to the data matrices X (independent variables) and y (dependent variables) and the number of regression hyperplanes k, the routine takes the following parameters as input.

- A desired probability of success $s < 1$ (for example, 0.999).
- An underestimate of the minimum cluster cardinality $\widetilde{n}_{\text{MIN}} \leq \min_j n_j(\overline{C})$.
- An underestimate of the minimum standard deviation of the underlying Gaussian noises $\widetilde{\sigma}_{\text{MIN}} \leq \min_j \overline{\sigma}_j$.

The value of $\widetilde{n}_{\text{MIN}}$ can be set to $p+1$. However, higher underestimates are preferred if available. The methods for obtaining $\widetilde{\sigma}_{\text{MIN}}$ are not discussed in this work and are instead left for future studies. In the rest of this work we assume that the values $\widetilde{\sigma}_{\text{MIN}}, \widetilde{n}_{\text{MIN}}$ are obtained by either previous knowledge of the domain of the problem or by other means. In Section 4.2 we suggest a possible way to compute $\widetilde{\sigma}_{\text{MIN}}$ from the data and a possible way to get rid of $\widetilde{\sigma}_{\text{MIN}}$ entirely, although the development of such ideas is left for future research. Analogously, we do not discuss the selection of k in this work but we mention some possibilities for its informed choice in Section 4.2 as additional future research avenues.

We will show that `pclustreg` finds a solution with log-likelihood at least as high as that of the (unknown) ground truth, with probability at least s. This is obtained by a probabilistic branch-and-bound method.

The proposed routine `pclustreg` also supports setting a limit to the number of nodes expanded. This feature allows the algorithm to be employed as a heuristic method that finds good solutions while keeping under control the execution time. Of course, in this case the aforementioned guarantees on the value of the objective function found by `pclustreg` do not hold.

2.2.1 Outline of the algorithm

Firstly, pclustreg performs a random permutation of the indexes of the points x, y. The ordering of points defined by this permutation is the one relative to which the points will be indexed in the rest of this and the next chapter. In particular, the sets of indexes C_j are defined relative to it. After that, the probabilistic branch-and-bound algorithm starts. The branching happens on the points to clusters assignment variables. In the root node all points are unassigned. In every intermediate node the first l points in the order given by the random shuffling are assigned, the remaining $n-l$ points are unassigned. Note that in branch-and-bound approaches fixing in advance the sequence of branching operations is not customary, rather branching operations are usually chosen adaptively. However, this choice is crucial for the proof of correctness of the upper bound with a known probability, as it will be explained in detail in Section 2.2.3. In particular, it guarantees that, in every intermediate node with partial assignment $\widetilde{C}$ such that $\forall j \in \{1,2,\ldots,k\} : S_j(\widetilde{C}) \subseteq S_j(\overline{C})$, the sets $S_j(\widetilde{C})$ and $S_j(\overline{C}) \setminus S_j(\widetilde{C})$ are both independent random subsamples of points taken from $S_j(\overline{C})$. When an intermediate node is expanded, it creates k new nodes. The children nodes share the assignments of the first l points with the parent node. For every children $j \in \{1,2,\ldots,k\}$, the point with index $l+1$ is assigned to cluster j. The remaining (last) $n-l-1$ points are left unassigned. The nodes are kept in a priority queue ordered by decreasing upper bounds. At every iteration the node with the highest upper bound is extracted from the queue. If its upper bound is lower than or equal to the current lower bound, the algorithm terminates and returns the best solution found. If the upper bound is greater than the current lower bound, then it is expanded into k children as explained above. For each child, the partial assignment is used to compute a new feasible solution according to the procedure described in Section 2.2.2. If the objective function value of the feasible solution is better than the current lower bound, the latter is updated. The computation of the probabilistic upper bound for each child is discussed in detail in Section 2.2.3. New nodes are cut from the search space when the current lower bound is higher than their upper bound.

2.2.2 Lower bound computation

Given a partial assignment C, the lower bound is computed as follows.

1. Ensure that each cluster has at least $p+1$ points assigned to it by adding random unassigned points to the clusters with less than $p+1$ points.
2. Fit the points in each cluster with standard least squares linear regression to obtain an estimate of the parameters $\hat{\beta}_j$ and $\hat{\sigma}_j$ for each cluster j.
3. Perform a modified Classification Expectation Maximization (CEM) algorithm to convergence.

The modification applied to the CEM consists of partially updating the values of $\hat{\pi}_j$ and $\hat{\sigma}_j$ while the points are reassigned instead of keeping them fixed. This change has yielded better lower bounds in practice in the conducted experiments. For more details about the implementation of the CEM used, see Section 3.4.2.

2.2.3 Upper bound computation

We first recall the definition (2.1) of $\mathrm{SSR}_j(C)$, the sum of the squared residuals for the points assigned to cluster j by the (possibly partial) assignment C when they are fitted with standard linear regression. We denote with $\mathscr{X}^2_\varepsilon(T)$ the quantile of order ε of the distribution $\mathscr{X}^2(T)$ when $T>0$ and $\mathscr{X}^2_\varepsilon(T)=0$ when $T\leq 0$.
For $\widetilde{n}_j \geq \max\{p+1, n_j(\widetilde{C})\}$, we define $\widetilde{f}_j(\widetilde{n}_j) = 3\widetilde{n}_j \log(\widetilde{n}_j) - \widetilde{n}_j \log(g_j(\widetilde{n}_j))$, where:

$$g_j(\widetilde{n}_j) := \begin{cases} \widetilde{\sigma}^2_{\mathrm{MIN}} \mathscr{X}^2_\varepsilon(\widetilde{n}_j - p) & \text{if } n_j(\widetilde{C}) < p+1 \\ \mathrm{SSR}_j(\widetilde{C}) + \widetilde{\sigma}^2_{\mathrm{MIN}} \mathscr{X}^2_\varepsilon(\widetilde{n}_j - n_j(\widetilde{C}) - p) & \text{otherwise.} \end{cases} \tag{2.3}$$

We consider the following problem, with $c_n = -\frac{n}{2}[\log(2\pi)+1] - n\log(n)$:

$$\begin{array}{lll} \max_{\widetilde{n}\in\mathbb{Z}^k} & c_n + \frac{1}{2}\sum_{j=1}^k \widetilde{f}_j(\widetilde{n}_j) & \\ s.t. & \widetilde{n}_j \geq \max(n_j(\widetilde{C}), \widetilde{n}_{\mathrm{MIN}}) & \forall j \in \{1,2,\ldots,k\} \\ & \sum_{j=1}^k \widetilde{n}_j = n & \end{array} \tag{2.4}$$

We will adopt the optimal value of (2.4) as upper bound in our probabilistic branch-and-bound approach. Now, let us consider an intermediate node that is on the path from the root node to the leaf node representing the ground-truth assignment $\overline{C}$. We will denote the (partial) assignment in such node with $\widetilde{C}$. It is important to notice that the points that are assigned in $\widetilde{C}$ are assigned to the same clusters as they are assigned to in $\overline{C}$ ($\forall j \in \{1,2,\ldots,k\}:\ S_j(\widetilde{C}) \subseteq S_j(\overline{C})$). In what follows we prove that at such node, the optimal value of (2.4) is at least as large as

$$\log(\hat{L}) = c_n + \frac{1}{2}\sum_{j=1}^{k} f_j(\overline{C}),$$

the estimated log-likelihood of the ground-truth, at least with a given probability.

We first state the following lemma.

Lemma 2.2.1. *Let $\widetilde{C}$ be the (partial) assignment at a node along the path from the root node to the leaf corresponding to the ground-truth $\overline{C}$. Then, for any $j \in \{1,\ldots,k\}$ it holds that $\widetilde{f}_j(n_j(\widetilde{C})) \geq f_j(\overline{C})$ with probability at least $1-\varepsilon$.*

Proof. We consider the two cases $n_j(\widetilde{C}) < p+1$ and $n_j(\widetilde{C}) \geq p+1$.

Case $n_j(\widetilde{C}) < p+1$. First, notice that $\mathrm{SSR}_j(\overline{C}) \sim \overline{\sigma}_j^2 \mathscr{X}^2(n_j(\overline{C})-p)$ (see, for example, [51]). In this case cluster j does not have enough points to fit with nonzero errors. Since $\mathrm{SSR}_j(\overline{C}) \sim \overline{\sigma}_j^2 \mathscr{X}^2(n_j(\overline{C})-p)$, the value $\overline{\sigma}_j^2 \mathscr{X}_\varepsilon^2(n_j(\overline{C})-p)$ underestimates $\mathrm{SSR}_j(\overline{C})$ with probability at least $1-\varepsilon$. Since $\widetilde{\sigma}_{\mathrm{MIN}} \leq \overline{\sigma}_j$, then $\widetilde{\sigma}_{\mathrm{MIN}}^2 \mathscr{X}_\varepsilon^2(n_j(\overline{C})-p) \leq \mathrm{SSR}_j(\overline{C})$ with probability at least $1-\varepsilon$. Since $-n_j(\overline{C})\log(.)$ is decreasing with respect to the argument of the logarithm we have that $\widetilde{f}_j(n_j(\overline{C})) \geq f_j(\overline{C})$ with probability at least $1-\varepsilon$.

Case $n_j(\widetilde{C}) \geq p+1$. In this case cluster j has enough points to fit with nonzero errors. We have

$$\begin{aligned}
\mathrm{SSR}_j(\overline{C}) &= \textstyle\sum_{i\in S_j(\overline{C})}(y_i - x_i^T\hat{\beta}_j(\overline{C}))^2 = \\
&= \textstyle\sum_{i\in S_j(\widetilde{C})}(y_i - x_i^T\hat{\beta}_j(\overline{C}))^2 + \sum_{i\in S_j(\overline{C})\setminus S_j(\widetilde{C})}(y_i - x_i^T\hat{\beta}_j(\overline{C}))^2 =
\end{aligned}$$

Firstly, we notice that

$$\sum_{i \in S_j(\widetilde{C})} (y_i - x_i^T \hat{\beta}_j(\overline{C}))^2 \geq \mathrm{SSR}_j(\widetilde{C}),$$

by definition of standard linear regression. Thus,

$$\mathrm{SSR}_j(\overline{C}) \quad \geq \mathrm{SSR}_j(\widetilde{C}) + \textstyle\sum_{i \in S_j(\overline{C}) \setminus S_j(\widetilde{C})} (y_i - x_i^T \hat{\beta}_j(\overline{C}))^2 \quad .$$

Secondly, we notice that $S_j(\overline{C}) \setminus S_j(\widetilde{C})$ is a random subsample of $n_j(\overline{C}) - n_j(\widetilde{C})$ points taken from $S_j(\overline{C})$ because the points have been randomly shuffled at the beginning and the points in $S_j(\widetilde{C})$ are simply the first $n_j(\widetilde{C})$ points assigned to cluster j in the ground-truth assignment $\overline{C}$ according to that random order. We denote with $\widetilde{C}'$ the complementary partial assignment with respect to $\overline{C}$, i.e., $S_j(\widetilde{C}') = S_j(\overline{C}) \setminus S_j(\widetilde{C})$ and so $n_j(\widetilde{C}') = n_j(\overline{C}) - n_j(\widetilde{C})$ for all j. We have that, again, by definition of standard linear regression,

$$\sum_{i \in S_j(\overline{C}) \setminus S_j(\widetilde{C})} (y_i - x_i^T \hat{\beta}_j(\overline{C}))^2 \geq \mathrm{SSR}_j(\widetilde{C}').$$

Moreover, recall that, when $n_j(\widetilde{C}') \geq p+1$, since $n_j(\widetilde{C}') = n_j(\overline{C}) - n_j(\widetilde{C})$,

$$\mathrm{SSR}_j(\widetilde{C}') \sim \overline{\sigma}_j^2 \mathscr{X}^2(n_j(\overline{C}) - n_j(\widetilde{C}) - p).$$

So we have that, since $\widetilde{\sigma}_{\mathrm{MIN}} \leq \overline{\sigma}_j$ and $n_j(\widetilde{C}') = n_j(\overline{C}) - n_j(\widetilde{C})$,

$$\sum_{i \in S_j(\overline{C}) \setminus S_j(\widetilde{C})} (y_i - x_i^T \hat{\beta}_j(\overline{C}))^2 \geq \widetilde{\sigma}_{\mathrm{MIN}}^2 \mathscr{X}_\varepsilon^2(n_j(\widetilde{C}') - p)$$

with probability at least $1-\varepsilon$. When $n_j(\widetilde{C}') \leq p$ the inequality is trivially true since, by our definition, $\mathscr{X}_\varepsilon^2(T) = 0$ when $T \leq 0$. Since the logarithm is an increasing function, we have that $\widetilde{f}_j(n_j(\overline{C})) \geq f_j(\overline{C})$ with probability at least $1-\varepsilon$ or, in other terms, $\widetilde{f}_j(n_j(\overline{C})) \geq f_j(\overline{C})$ with probability at least $1-\varepsilon$. □

As a consequence of the previous result, we can also prove the following.

Lemma 2.2.2. *Let $\widetilde{C}$ be the (partial) assignment at the node N of level t along the path from the root node to the leaf corresponding to the ground-truth $\overline{C}$. Then, for node N and all its ancestors,*

$$\widetilde{f}_j(n_j(\overline{C})) \geq f_j(\overline{C}) \ \ \forall\, j \in \{1, \ldots, k\}, \tag{2.5}$$

with probability not lower than $1-(k+t)\varepsilon$.

Proof. We prove the result by induction. It is true at the root node, i.e., when $t = 0$. Indeed, by Lemma 2.2.1 it holds that $\widetilde{f}_j(n_j(\overline{C})) \geq f_j(\overline{C})$ at least with probability $1-\varepsilon$ for a single value j, so that it holds for all $j \in \{1,\ldots,k\}$ at least with probability $1-k\varepsilon$.

Now, let us assume that the statement is true for the node at level t and we prove it also holds for its child at level $t+1$. Only one of the clusters changes when moving from a parent node to a child node because only one new point is assigned to one of the clusters. If such cluster is the r-th one, then only the value $\widetilde{f}_r(n_r(\overline{C}))$ can change. By the inductive assumption,

$$\widetilde{f}_j(n_j(\overline{C})) \geq f_j(\overline{C}) \;\; \forall\, j \neq r,$$

at least with probability $1-(k+t)\varepsilon$. Moreover, by Lemma 2.2.1 with probability at least $1-\varepsilon$ it holds that

$$\widetilde{f}_r(n_r(\overline{C})) \geq f_r(\overline{C}).$$

Then, by the Boole-Bonferroni inequality [52] for the two events, the probability of their intersection is at least $1-(k+t+1)\varepsilon$. □

An immediate consequence of this result is the following.

Lemma 2.2.3. *Let $\widetilde{C}$ be the (partial) assignment at the node N of level t along the path from the root node to the leaf corresponding to the ground-truth $\overline{C}$. Then, for node N and all its ancestors, the optimal value of (2.4) is at least as large as the estimated log-likelihood* $\log(\hat{L})$ *of the ground-truth with probability not lower than* $1-(k+t)\varepsilon$.

Proof. It is enough to observe that $[n_1(\overline{C}), n_2(\overline{C}), \ldots, n_k(\overline{C})]$ is a point inside the feasible region of the optimization problem (2.4), since $\widetilde{C}$ is a partial assignment of $\overline{C}$. Since, by Lemma 2.2.2, (2.5) holds with probability not lower than $1-(k+t)\varepsilon$, then the optimal value of (2.4) is at least as large as the log-likelihood $\log(\hat{L}(\overline{C}))$ of the ground-truth at least with the same probability. □

We are now ready for the main result.

Theorem 2.2.4. *Let $\varepsilon = \frac{1-s}{n+k}$. Then, the proposed probabilistic branch-and-bound approach is able to detect a solution with log-likelihood at least as good as* $\log(\hat{L}(\overline{C}))$*, the log-likelihood of the ground-truth, with probability at least s.*

Proof. If a solution with log-likelihood at least as good as the one of the ground-truth is detected before reaching the leaf node corresponding to the ground-truth, then we are done. Otherwise, by Lemma 2.2.3 no node along the path from the root node to the ground-truth leaf is cut with probability $1-(n+k)\varepsilon = s$, and the ground-truth is directly evaluated at the leaf node. Our method cuts nodes without exploring them when the lower bound is higher than their (probabilistic) upper bound. □

Solution of Problem (2.4)

Regarding the solution of Problem (2.4), notice that since the variables of Problem (2.4) are all integers, then only the values assumed by $\widetilde{f}_j$ on integer points are relevant. We substitute $\widetilde{f}_j$ with $\widetilde{f}_j^*$, a piecewise-linear function that over-estimates $\widetilde{f}_j$ in integer points. The function $\widetilde{f}_j^*$ is obtained exploiting the concavity of the logarithm in $\widetilde{f}_j$ and then taking the piecewise-linear interpolation of the integer points. We do not have a formal theoretical proof that $\widetilde{f}_j^*$ is convex but `pclustreg` numerically checks its convexity by testing that the slopes are increasing before computing each upper bound. Now, if the function is convex, since the feasible region of Problem (2.4) is a polytope (more precisely, a simplex) with integer vertices, its maximum value is attained at a vertex of the feasible region (see, e.g., [53]). The number of such vertices is equal to k. In our tests $\widetilde{f}_j^*$ was always found to be convex. However, if the check fails we can still solve Problem (2.4) through a dynamic programming algorithm that runs in $\mathcal{O}(kn^2)$. We do not discuss such algorithm since it was never used in practice because $\widetilde{f}_j^*$ was always checked to be convex by `pclustreg` in our tests.

2.2.4 Hyperparameters overview

In the discussion of `pclustreg` we have made use of several hyperaparameters. For the sake of clarity and usability of this manuscript we list them in this section

and provide a brief description for each one. The purpose of this list is to offer a bird-eye view over all the hyperparameters at play and in-depth discussions presented in previous sections regarding each hyperparameter are not repeated here.

- k: a numerical hyperparameter equal to the number of regression hyperplanes. If the number of hyperplanes is known beforehand than it is actually a fixed number.

- $\widetilde{\sigma}^2_{\text{MIN}}$: a numerical hyperparameter that must be an underestimate of the minimum variance among the regression clusters.

- s: the desired probability of success, i.e., the probability that the solution found by the algorithm (when it is not used as an heuristic) has a log-likelihood that is at least as high as the log-likelihood of the (unknown) ground truth.

- $\widetilde{n}_{\text{MIN}}$: an underestimate of the minimum cluster cardinality. It must be at least equal to p (the number of dimensions of the points) plus 1.

- The algorithm for solving Problem 2.4. Our choice and an alternative are discussed in 2.2.3.

- The number of times the CEM algorithm is started on a (partly) random initialization for each node of the branch and bound tree.

- The details and hyperparameters of the CEM algorithm employed. See 3.4.2 for more insights about the CEM algorithm variant employed in this work.

2.3 Numerical experiments on synthetic data

In this section we present and discuss the numerical experiments carried out on synthetic data. Notice that we employed `pclustreg` as a heuristic by limiting the number of nodes expanded (see Section 2.2). We compare three methods.

2.3.1 Compared methods

1. Our proposed approach `pclustreg`.
2. The implementation of `tclustreg` [50, 54] provided in the FSDA robust statistical package [55] without trimming and without restriction factors on the ratios of standard deviations.
3. The `oracle`, i.e., the ground-truth. This is an unfeasible method that we employ as a benchmark for the other methods.

We adjusted the node expansion limit of `pclustreg` to obtain similar computation times as the ones obtained by `tclustreg` to perform a fair comparison. In particular, we used a limit to the number of nodes expanded by `pclustreg` of 100. We used 3000 as the number of subsamples used by `tclustreg`. We ran `pclustreg` with $\widetilde{n}_{\text{MIN}} = p + 1 = 3$ and with $\widetilde{\sigma}_{\text{MIN}} = 1$.

2.3.2 Performance metrics

We compare the methods on the following metrics.

1. The value of the log-likelihood $\log(\hat{L})$ obtained. It is worth reiterating that higher values mean better fitness of the model to the data.
2. The computational time $t[s]$, expressed in seconds.

2.3.3 Generation of synthetic data

The synthetic data is obtained through the use of the `MixSimreg` [50, 56] and `simdatasetreg` [56, 57] functions of the MATLAB FSDA [55] package. Given the number of regression hyperplanes k and a desired average overlap the function `MixSimreg` creates the corresponding distribution with the values for $\overline{\pi}_j, \overline{\sigma}_j^2, \overline{\beta}_j$. Then, `MixSimreg` uses these values to generate the data and returns the $X, Y, \overline{C}$. We ran the algorithms on 100 randomly generated datasets with $k = 5$, $p = 10$, $n = 1000$ and average overlap of 0.05.

Figure 2.1: Boxplot for the log-likelihood over 100 runs on synthetic datasets with $k = 5$, $p = 10$, $n = 1000$ and average overlap of 0.05.

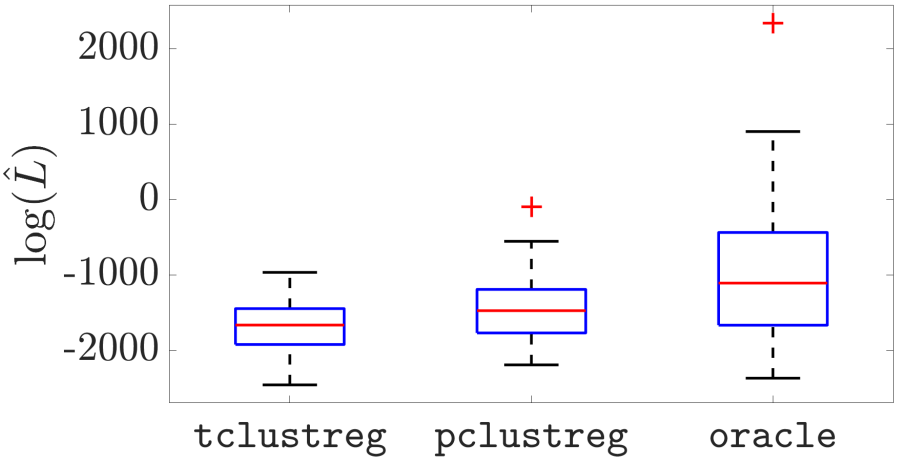
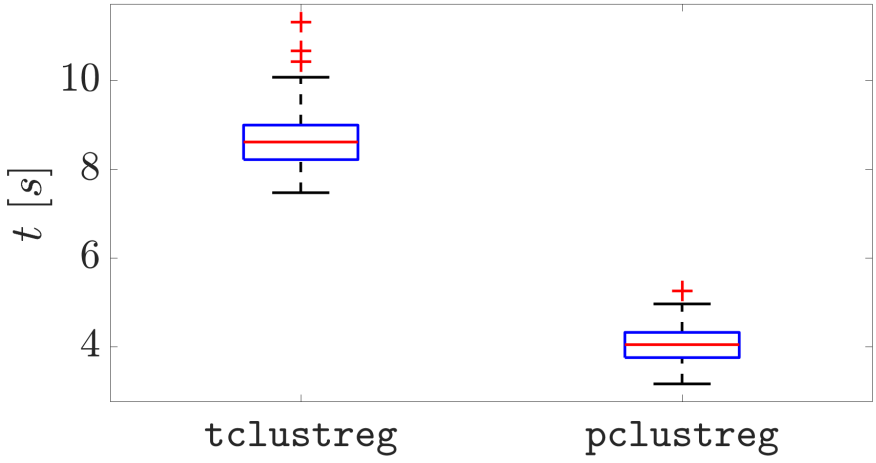


Figure 2.2: Boxplots for the computation times over 100 runs on synthetic datasets with $k = 5$, $p = 10$, $n = 1000$ and average overlap of 0.05. The computation time of the `oracle` is uninformative and thus omitted.



2.3.4 Simulation results and discussion

Results of our experiments are shown in Fig. 2.1 and 2.2. Fig. 2.1 shows that `pclustreg` outperforms `tclustreg` in terms of log-likelihood of the results. The `oracle` is shown as a benchmark. Fig. 2.2 shows that the computation times of `pclustreg` are comparable to those of `tclustreg`.

Chapter 3

Improving `pclustreg` with the $\mathscr{F}$ distribution

Se la vita è sventura,
perché da noi si dura?

– Giacomo Leopardi

As seen in chapter 2, using the $\mathscr{X}^2$ distribution poses a serious limitation to the application of `pclustreg` due to the requirement of the parameter σ_{MIN}. In particular, the problems are two.

- Although it may be reasonably possible to provide an extremely low underestimate for the parameter, it would impact the core idea of the algorithm, making it regress to the standard branch and bound approach.
- The fact that the parameter is, by definition, an underestimate of the *minimum* of the $\overline{\sigma}_j$, makes `pclustreg` not suitable for cases in which the standard deviations of the underlying noises differ appreciably among different clusters.

We will show that it is possible to employ the distribution $\mathscr{F}$ in place of the distribution $\mathscr{X}^2$ to address both of these problems.

3.1 Modified upper bound computation

Our approach consists of changing the definition of the function $g_j(\widetilde{n}_j)$ from Equation 2.3 to

$$g_j(\widetilde{n}_j) := \begin{cases} \widetilde{\sigma}^2_{\text{MIN}} \mathscr{X}^2_\varepsilon(\widetilde{n}_j - p) & \text{if } n_j(\widetilde{C}) < p+1 \\ \text{SSR}_j(\widetilde{C}) + \text{SSR}_j(\widetilde{C}) \mathscr{F}_\varepsilon(\widetilde{n}_j - n_j(\widetilde{C}) - p, n_j(\widetilde{C}) - p) & \text{otherwise.} \end{cases}$$

The first case is unchanged. In the second case we changed $\widetilde{\sigma}^2_{\text{MIN}} \mathscr{X}^2_\varepsilon(\widetilde{n}_j - n_j(\widetilde{C}) - p)$ to $\text{SSR}_j(\widetilde{C}) \mathscr{F}_\varepsilon(\widetilde{n}_j - n_j(\widetilde{C}) - p, n_j(\widetilde{C}) - p)$. Since the definition of the function $g_j(\widetilde{n}_j)$ is directly used only in Lemma 2.2.1, we only need to change the proof of such lemma to retain the optimality guarantees of Theorem 2.2.4.

Lemma 3.1.1. *Let $\widetilde{C}$ be the (partial) assignment at a node along the path from the root node to the leaf corresponding to the ground-truth $\overline{C}$. Then, for any $j \in \{1, \ldots, k\}$ it holds that $\widetilde{f}_j(n_j(\overline{C})) \geq f_j(\overline{C})$ with probability at least $1 - \varepsilon$.*

Proof. We consider the two cases $n_j(\widetilde{C}) < p+1$ and $n_j(\widetilde{C}) \geq p+1$.

Case $n_j(\widetilde{C}) < p+1$. First, notice that $\text{SSR}_j(\overline{C}) \sim \overline{\sigma}^2_j \mathscr{X}^2(n_j(\overline{C}) - p)$ (see, for example, [51]). In this case cluster j does not have enough points to fit with nonzero errors. Since $\text{SSR}_j(\overline{C}) \sim \overline{\sigma}^2_j \mathscr{X}^2(n_j(\overline{C}) - p)$, the value $\overline{\sigma}^2_j \mathscr{X}^2_\varepsilon(n_j(\overline{C}) - p)$ underestimates $\text{SSR}_j(\overline{C})$ with probability at least $1 - \varepsilon$. Since $\widetilde{\sigma}_{\text{MIN}} \leq \overline{\sigma}_j$, then $\widetilde{\sigma}^2_{\text{MIN}} \mathscr{X}^2_\varepsilon(n_j(\overline{C}) - p) \leq \text{SSR}_j(\overline{C})$ with probability at least $1 - \varepsilon$. Since $-n_j(\overline{C}) \log(.)$ is decreasing with respect to the argument of the logarithm we have that $\widetilde{f}_j(n_j(\overline{C})) \geq f_j(\overline{C})$ with probability at least $1 - \varepsilon$.

Case $n_j(\widetilde{C}) \geq p+1$. In this case cluster j has enough points to fit with nonzero errors. We have

$$\begin{aligned} \text{SSR}_j(\overline{C}) &= \textstyle\sum_{i \in S_j(\overline{C})} (y_i - x_i^T \hat{\beta}_j(\overline{C}))^2 = \\ &= \textstyle\sum_{i \in S_j(\widetilde{C})} (y_i - x_i^T \hat{\beta}_j(\overline{C}))^2 + \sum_{i \in S_j(\overline{C}) \setminus S_j(\widetilde{C})} (y_i - x_i^T \hat{\beta}_j(\overline{C}))^2 = \end{aligned}$$

Firstly, we notice that

$$\sum_{i \in S_j(\widetilde{C})} (y_i - x_i^T \hat{\beta}_j(\overline{C}))^2 \geq \mathrm{SSR}_j(\widetilde{C}),$$

by definition of standard linear regression. Thus,

$$\mathrm{SSR}_j(\overline{C}) \quad \geq \mathrm{SSR}_j(\widetilde{C}) + \textstyle\sum_{i \in S_j(\overline{C}) \setminus S_j(\widetilde{C})} (y_i - x_i^T \hat{\beta}_j(\overline{C}))^2.$$

Secondly, we denote with $\widetilde{C}'$ the complementary partial assignment with respect to $\overline{C}$, i.e., $S_j(\widetilde{C}') = S_j(\overline{C}) \setminus S_j(\widetilde{C})$ and so $n_j(\widetilde{C}') = n_j(\overline{C}) - n_j(\widetilde{C})$ for all j. By definition of standard linear regression we have that

$$\textstyle\sum_{i \in S_j(\widetilde{C}')} (y_i - x_i^T \hat{\beta}_j(\overline{C}))^2 \quad \geq \sum_{i \in S_j(\widetilde{C}')} (y_i - x_i^T \hat{\beta}_j(\widetilde{C}'))^2 \quad = \mathrm{SSR}_j(\widetilde{C}').$$

Thus,

$$\mathrm{SSR}_j(\overline{C}) \quad \geq \mathrm{SSR}_j(\widetilde{C}) + \mathrm{SSR}_j(\widetilde{C}').$$

Thirdly, we notice that $S_j(\widetilde{C})$ and $S_j(\widetilde{C}')$, due to the fixed order of assignment of the points, are independent (and disjoint) random subsamples of $n_j(\widetilde{C})$ and $n_j(\widetilde{C}')$ points taken from $S_j(\overline{C})$.

Thus, we have

$$\begin{aligned} \mathrm{SSR}_j(\widetilde{C}) \quad &\sim \overline{\sigma}_j^2 \mathscr{X}^2(n_j(\widetilde{C}) - p). \\ \mathrm{SSR}_j(\widetilde{C}') \quad &\sim \overline{\sigma}_j^2 \mathscr{X}^2(n_j(\widetilde{C}') - p). \end{aligned}$$

Thus,

$$\frac{\mathrm{SSR}_j(\widetilde{C}')}{\mathrm{SSR}_j(\widetilde{C})} \quad \sim \frac{\overline{\sigma}_j^2 \mathscr{X}^2(n_j(\widetilde{C}') - p)}{\overline{\sigma}_j^2 \mathscr{X}^2(n_j(\widetilde{C}) - p)} \quad = \mathscr{F}(n_j(\widetilde{C}') - p, n_j(\widetilde{C}) - p).$$

Consequently,

$$\mathrm{SSR}_j(\widetilde{C}') \quad \sim \mathrm{SSR}_j(\widetilde{C}) \mathscr{F}(n_j(\widetilde{C}') - p, n_j(\widetilde{C}) - p).$$

So we have that,

$$\mathrm{SSR}_j(\widetilde{C}') \quad \geq \mathrm{SSR}_j(\widetilde{C}) \mathscr{F}_\varepsilon(n_j(\widetilde{C}') - p, n_j(\widetilde{C}) - p).$$

with probability at least $1 - \varepsilon$. Thus,

$$\mathrm{SSR}_j(\overline{C}) \quad \geq \mathrm{SSR}_j(\widetilde{C}) + \mathrm{SSR}_j(\widetilde{C}) \mathscr{F}_\varepsilon(n_j(\widetilde{C}') - p, n_j(\widetilde{C}) - p).$$

with probability at least $1-\alpha$.

When $n_j(\widetilde{C}') \leq p$ the inequality is trivially true since, by our definition, $\mathscr{X}^2_\alpha(T) = 0$ when $T \leq 0$. Since the logarithm is an increasing function, we have that $\widetilde{f}_j(n_j(\overline{C})) \geq f_j(\overline{C})$ with probability at least $1-\alpha$ or, in other terms, $\widetilde{f}_j(n_j(\overline{C})) \geq f_j(\overline{C})$ with probability at least $1-\alpha$. □

3.1.1 Hyperparameters overview

We do not repeat the list of hyperparameters used by this implementation of `pclustreg` because they are the ones listed in 2.2.4. The only difference is that the importance of $\widetilde{n}^2_{\text{MIN}}$ is greatly diminished since it is only used in nodes in which less than $p+1$ points have been assigned to some clusters and only for those clusters.

3.2 Synthetic Benchmarks

In this section we discuss the benchmarks we conducted on synthetic data. All tests were ran on a machine equipped with an Intel Core i7 7700HQ processor and 16GB (8+8) of RAM running Windows 10. Implementation details for `pclustreg` are presented in Section 3.4.

3.2.1 Compared Methods

We compare the following methods.

- The `oracle`, i.e., the ground truth. It is possible to use it because we are generating synthetic data.
- The MATLAB FSDA `tclustreg` function.
- Our method: `pclustreg` used as a heuristic method.

Since both `tclustreg` and `pclustreg` are heuristic methods we try to set their stopping criteria so that they complete their execution in comparable times. `pclustreg` provides the “nsamp" parameter, i.e., the number of random starts for

the Classification Expectation Maximization (CEM) algorithm to use. `pclustreg` provides the "node_limit" parameter, which is the maximum number of nodes expanded in the branch and bound. Since each node expansion triggers k (the number of clusters) number of executions of the CEM algorithm (for the computation of the lower bounds), it would be reasonable to take “nsamp" divided by k as the value for "node_limit". However, since the CEM algorithm actually implemented in `pclustreg` is nonstandard, the two are not fully comparable. See subsection 3.4.2 for more information. We found that setting "node_limit" to “nsamp" divided by $2k$ is a more reasonable choice for this parameter for the tests we conducted.

3.2.2 Synthetic datasets generation

We generated the synthetic datasets using the following method.

1. Firstly, generate the ground truth parameters using the MATLAB FSDA `MixSimreg` function. We used the following parameters to set up the distribution.
   - the number of clusters k.
   - the dimension of the points p.
   - the average overlap between clusters $\overline{\Omega} = 0.05$
   - the lowest a priori probability allowed $\pi_{\text{LOW}} = \frac{1}{3k}$.
2. Secondly, sample the generated distribution to construct the dataset with n points using the FSDA `simdatasetreg` function.

3.2.3 Comparison metrics

We compare `pclustreg` and `tclustreg` using the following metrics.

1. The computational time t expressed in seconds.
2. The value of the objective function of the solution $\log(\hat{L})$ (the log-likelihood).
3. The Adjusted Rand Index *ARI* [14], computed with the `RandIndexFS` function of FSDA, which is explained below.

The Adjusted Rand Index

The Adjusted Rand Index is a number between -1 and 1 that expresses how similar two assignments of points to clusters are, taking into account the possibility of relabeling of the clusters (e.g., that the same cluster might be "cluster 1" in the first assignment and "cluster 2" in the second assignment). Random assignments have $ARI \approx 0$, the perfect assignment has $ARI = 1$ and assignments that are worse than random chance have $ARI < 0$. We obtain these values by computing the *ARI* with the assignment provided by both `pclustreg` and `tclustreg` against the ground-truth assignment. It is worth noticing that with real-world data it is not possible to employ this metric since the ground-truth assignment is unknown. In this case, since we are working with synthetic data, we are able to compute the value of this metric.

3.2.4 Test scenarios

We devised three testing scenarios that test the performance of `tclustreg` and our proposal `pclustreg` as different parameters change.

- As the number of clusters k changes from 2 to 10.
- As the dimension of the points p changes from 2 to 10.
- As the number of subsamples (starting points of the CEM) “nsamp" (NS in the graphs, for brevity) takes the values $\{100, 200, 400, 800, 1600, 3200\}$.

3.2.5 Scenario 1: Varying k

In this scenario we show the results as k takes the values in $\{2,3,4,5,6,7,8,9,10\}$ with fixed $p = 10$, “nsamp" $= 3200$. In Figure 3.1 we see that for all the values of k tested the computation times of `pclustreg` are below those of `tclustreg`. In Figure 3.2 we see that the values of the objective function is comparable among the methods for $k \in \{1,2,3,4\}$. Starting from $k = 5$ `pclustreg` shows improvements with respect to `tclustreg`. The same phenomenon is noticed also in the values of the *ARI* in Figure 3.3.

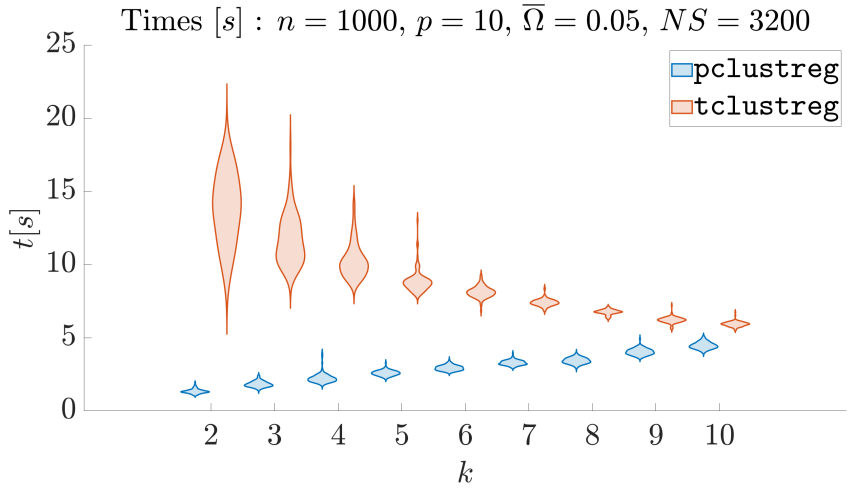


Figure 3.1: violin plots of the computation times for `pclustreg` and `tclustreg` with $n = 1000$, $p = 10$, $\overline{\Omega} = 0.05$, "nsamp" $= 3200$ as k changes from 2 to 10. The computation time of the `oracle` is not meaningful.

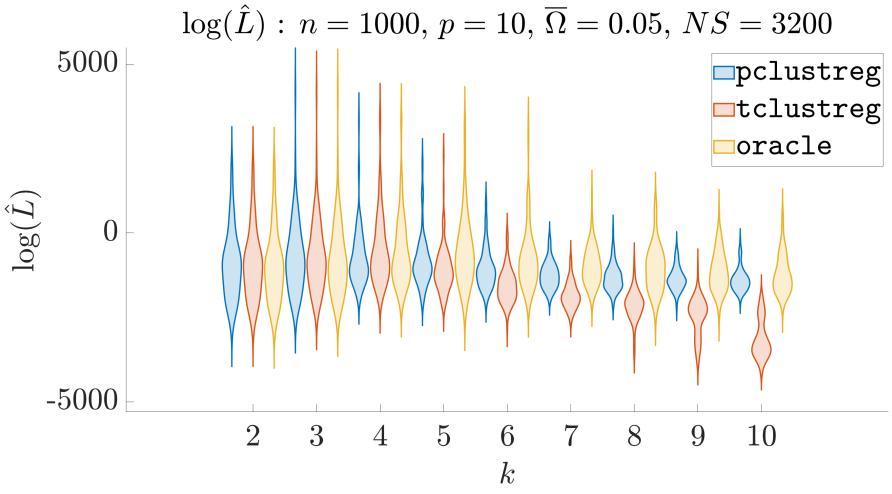


Figure 3.2: violin plots of the objective function $\log(\hat{L})$ for `pclustreg`, `tclustreg` and `oracle` with $n = 1000$, $p = 10$, $\overline{\Omega} = 0.05$, "nsamp" $= 3200$ as k changes from 2 to 10.

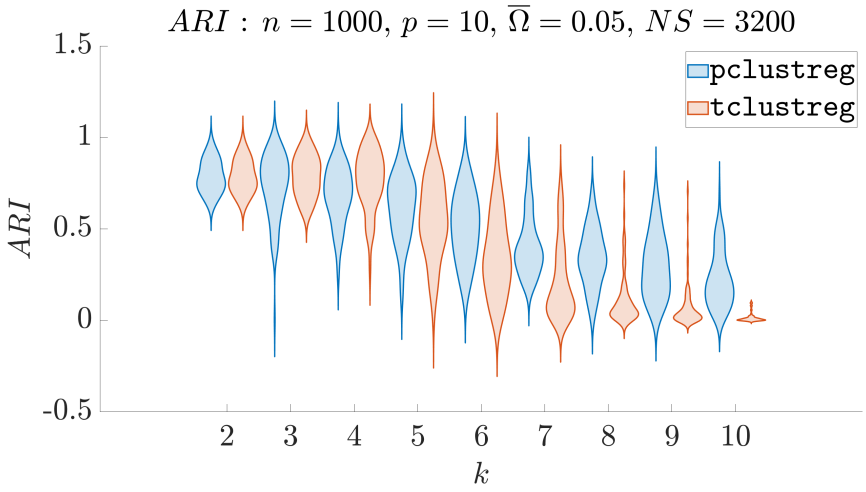


Figure 3.3: violin plots of the *ARI* for `pclustreg` and `tclustreg` with $n = 1000$, $p = 10$, $\overline{\Omega} = 0.05$, "nsamp" $= 3200$ as k changes from 2 to 10. The *ARI* of the `oracle` is not reported in the figure as it is the constant 1.

3.2.6 Scenario 2: Varying p

In this scenario we show the results as p takes the values in $\{2,3,4,5,6,7,8,9,10\}$ with fixed $k = 10$, "nsamp" $= 3200$ In Figure 3.4 we see that for $p \leq 8$ the computation times of `pclustreg` are higher than those of `tclustreg` while they are lower for $p \geq 9$. In Figure 3.5 we see that the values of the objective function is comparable among the methods for $p = 2$. Starting from $p = 3$ `pclustreg` shows improvements with respect to `tclustreg`. The same phenomenon is noticed also in the values of the *ARI* in Figure 3.6.

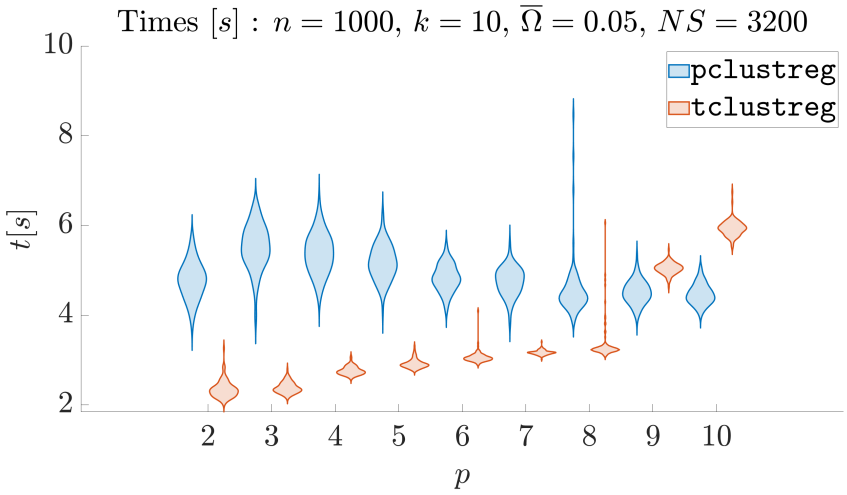


Figure 3.4: violin plots of the computation times for `pclustreg` and `tclustreg` with $n = 1000$, $k = 10$, $\overline{\Omega} = 0.05$, "nsamp" $= 3200$ as p changes from 2 to 10. The computation time of the `oracle` is not meaningful.

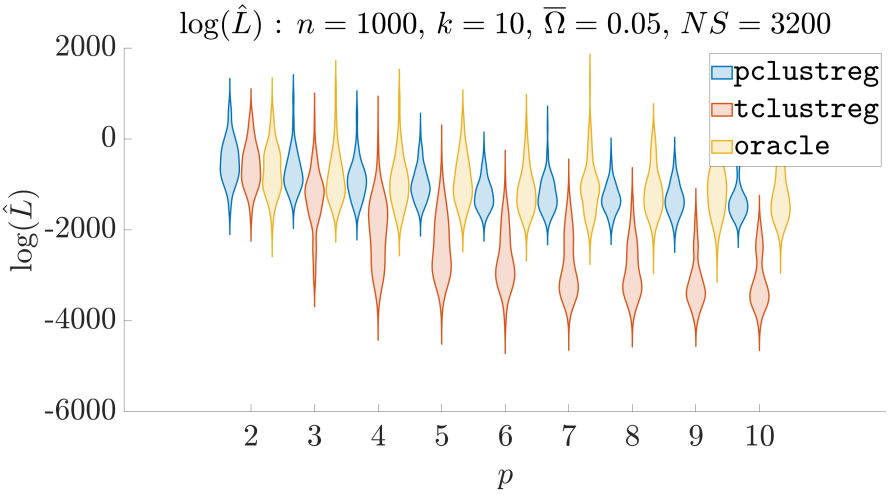


Figure 3.5: violin plots of the objective function $\log(\hat{L})$ for `pclustreg`, `tclustreg` and `oracle` with $n = 1000$, $k = 10$, $\overline{\Omega} = 0.05$, "nsamp" $= 3200$ as p changes from 2 to 10.

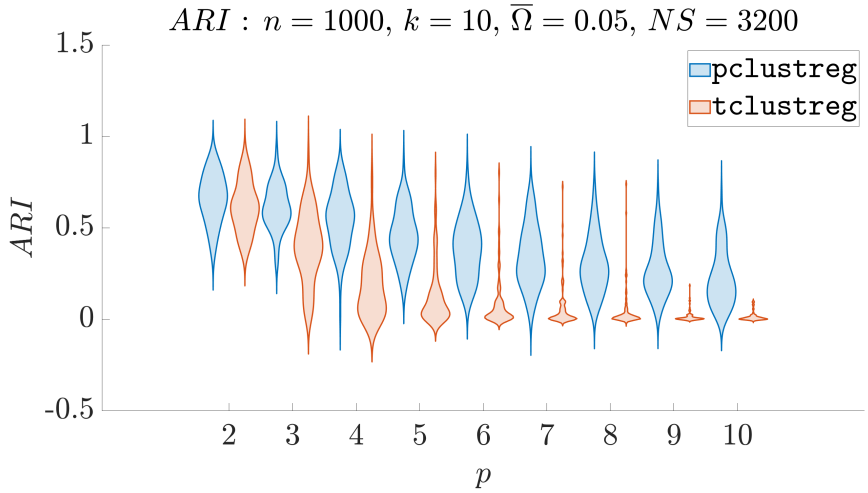


Figure 3.6: violin plots of the *ARI* for `pclustreg` and `tclustreg` with $n = 1000$, $k = 10$, $\overline{\Omega} = 0.05$, "nsamp" $= 3200$ as p changes from 2 to 10. The *ARI* of the `oracle` is not reported in the figure as it is the constant 1.

3.2.7 Scenario 3: Varying “nsamp"

In this scenario we show the results for “nsamp" $\in \{100, 200, 400, 800, 1600, 3200\}$ with fixed $k = 10$, $p = 10$. In Figure 3.7 we see that for “nsamp" $\in \{100, 200, 400\}$ the computation times of `pclustreg` are similar to those of `tclustreg` while they are lower for “nsamp" ≥ 800. In Figure 3.8 we see that the values of the objective function are higher for all values of “nsamp" tested. The same can be noticed also in the values of the *ARI* in Figure 3.9.

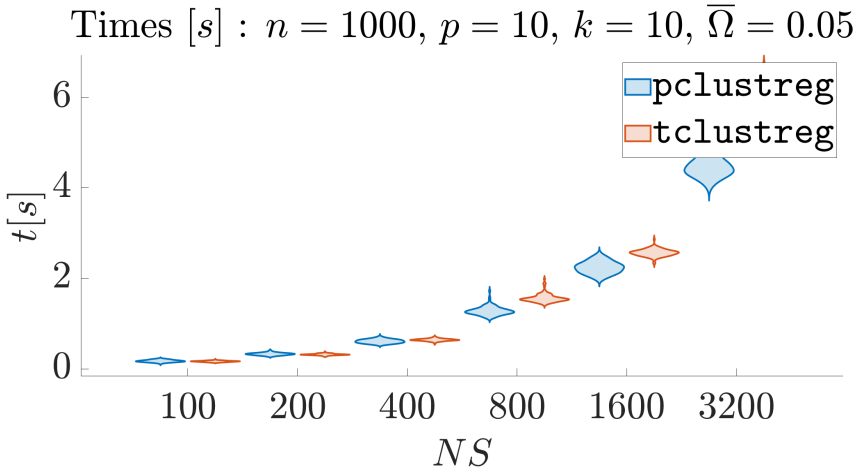


Figure 3.7: violin plots of the computation times for `pclustreg` and `tclustreg` with $n = 1000$, $k = 10$, $\overline{\Omega} = 0.05$, $p = 10$ as “nsamp" takes the values in $\{100, 200, 400, 800, 1600, 3200\}$. The computation time of the `oracle` is not meaningful.

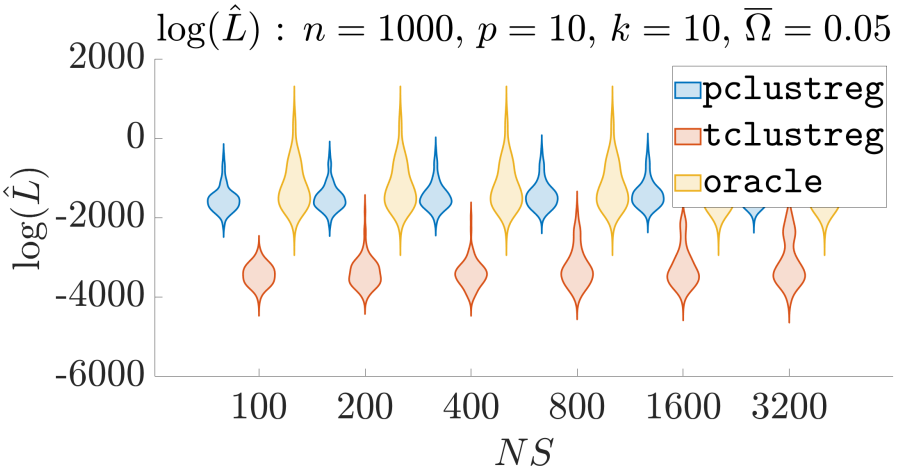


Figure 3.8: violin plots of the objective function $\log(\hat{L})$ for `pclustreg`, `tclustreg` and `oracle` with $n = 1000$, $k = 10$, $\overline{\Omega} = 0.05$, $p = 10$ as "nsamp" takes the values in $\{100, 200, 400, 800, 1600, 3200\}$.

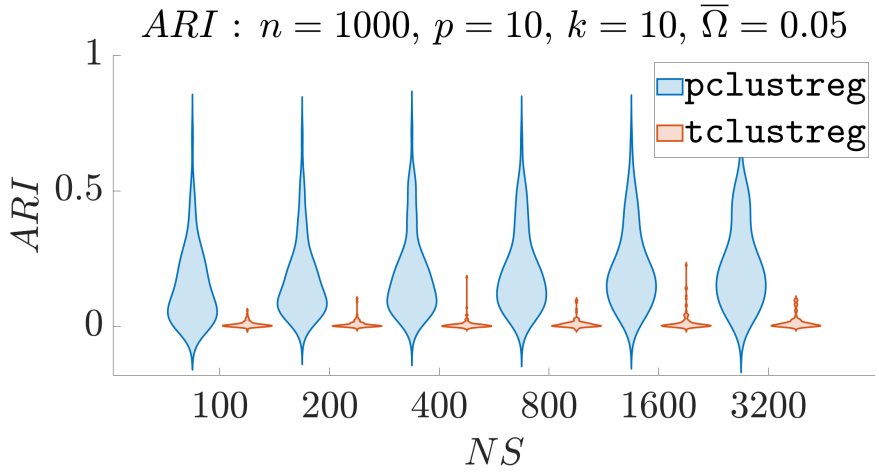


Figure 3.9: violin plots of the *ARI* for `pclustreg` and `tclustreg` with $n = 1000$, $k = 10$, $\overline{\Omega} = 0.05$, $p = 10$ as "nsamp" takes the values in $\{100, 200, 400, 800, 1600, 3200\}$. The *ARI* of the `oracle` is not reported in the figure as it is the constant 1.

3.2.8 Experimental results discussion

The results obtained for all scenarios show that `pclustreg` is competitive with `tclustreg` and is capable of outperforming it in terms of objective function and *ARI* in the same or less time in particular for the most challenging scenarios (high values of p and k).

3.3 Real-world application

In this section we present the tests we conducted on the FSDA [55] Girdles dataset. The dataset consists of real trade data. In particular, it contains the price and quantity of 153 imports of girdles and panty girdles from Israel to Austria. Since we allow an intercept term and there is only one independent variable, $p = 2$.

Since the dataset does not contain the ground-truth the `oracle` method is not feasible, thus we only compare `pclustreg` in its variants `pclustreg1` (described in Chapter 2.1) and `pclustreg2` (described in this chapter) against `tclustreg`.

We tested the dataset using $k = 3$ and $k = 4$. We repeated the tests increasing the `pclustreg` node limit and the `tclustreg` number of subsamples to keep the computation times approximately equal in the most challenging scenario of $k = 4$. This was done in order to have a fair comparison. In particular, we used a node limit of $100, 500, 10000$ and corresponding `tclustreg` number of subsamples of $300, 3000, 60000$. We call these three configurations scenario A, B, and C respectively.

We ran `pclustreg` with $\widetilde{n}_{\text{MIN}} = p + 1 = 3$ and with $\widetilde{\sigma}_{\text{MIN}} = 1$. Since we are using `pclustreg` as a heuristic, one could also perform a search on the parameter $\widetilde{\sigma}_{\text{MIN}}$. Such search is not being performed in this experiment. We repeated the experiment 100 times with 100 different random number generator seeds.

3.3.1 Results and discussion

The results are shown in Figures 3.10, 3.11,3.12 and 3.13. We can see that for $k = 3$ both `tclustreg`, `pclustreg1` and `pclustreg2` usually find the same solution

regardless of their execution time. For $k = 4$ `pclustreg2` performs slightly better than `pclustreg1` and more in line with `tclustreg`, altough at a much lower computation time. In general, the execution times of `pclustreg1` and `pclustreg2` are lower than those of `tclustreg` while achieving comparable or better results.

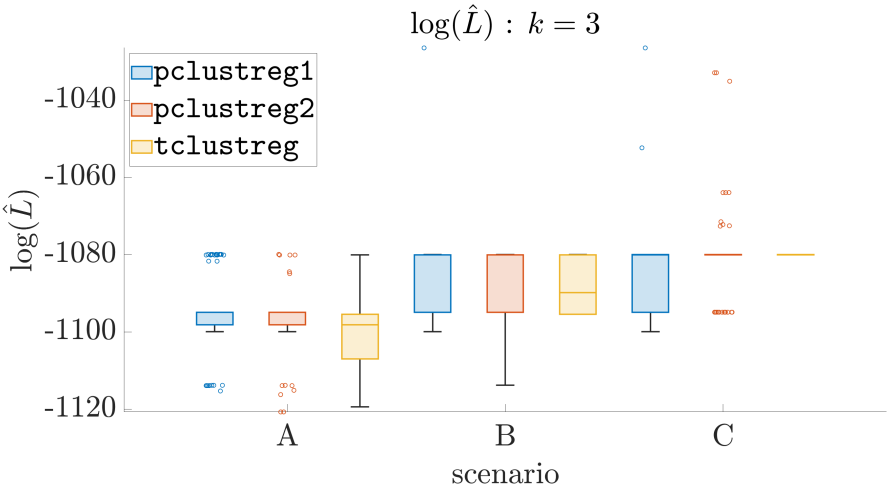


Figure 3.10: Boxcharts of the loglikelihoods obtained by `pclustreg1`, `pclustreg2` and `tclustreg` with $k = 3$.

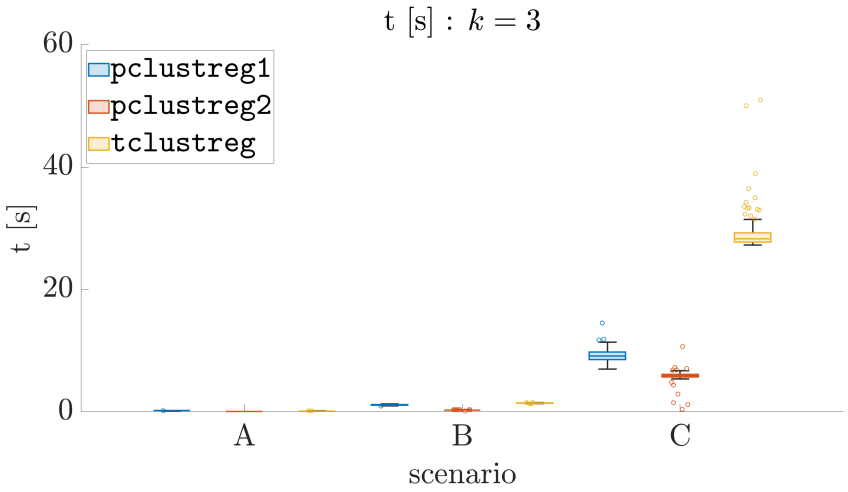


Figure 3.11: Boxcharts of the computation times for `pclustreg1`, `pclustreg2` and `tclustreg` with $k = 3$.

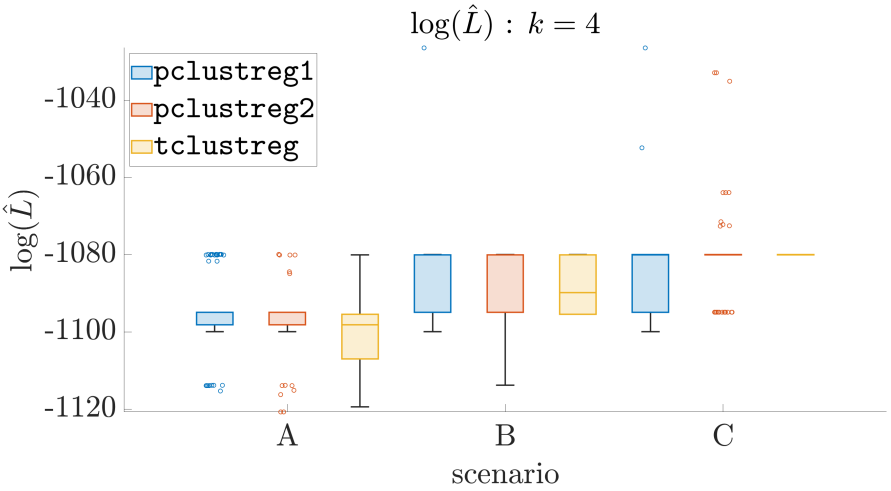


Figure 3.12: Boxcharts of the loglikelihoods obtained by `pclustreg1`, `pclustreg2` and `tclustreg` with $k = 4$.

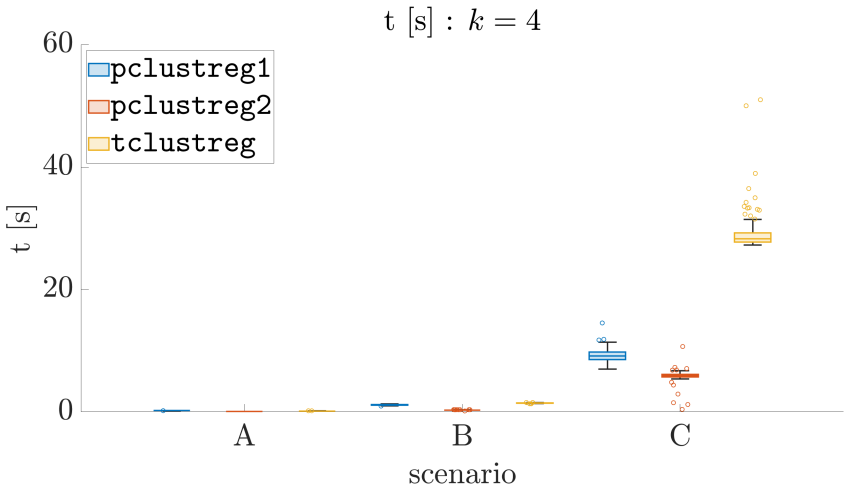


Figure 3.13: Boxcharts of the computation times for `pclustreg1`, `pclustreg2` and `tclustreg` with $k = 4$.

3.4 Implementation details

In this section we discuss some of the relevant implementation details. The implementation of `pclustreg` discussed in this work is written in C and compiled with GCC [58] (MinGW under Windows) with the -O3 optimization flag. The method we employed to call the routine from MATLAB consists of exposing a C header file with the declaration of the function and then a Dynamically Linked Library (DLL on Windows) containing the implementation. This can be done by employing the MATLAB functions `loadlibrary`, which allows the user to load a shared library and `calllib`, which allows to call a function contained in the shared library.

3.4.1 Libraries used

In order to speed up the development of the implementation of `pclustreg` we employed some C libraries other than the libc. The libraries used are listed below.

- The GNU Scientific Library (GSL, `https://www.gnu.org/software/gsl/`) [59] is used for random number generation and for the computation of the quantiles of the probability distributions $\mathscr{X}^2, \mathscr{F}$.

- The Basic Linear Algebra Subprograms (BLAS, `https://netlib.org/blas/`) [60][61] library is used for vector and matrix operations.

- The Linear Algebra PACKage (LAPACK, `https://netlib.org/lapack/`) [62] [63] library is used to perform the Least Squares fitting. In particular, the Cholesky decomposition of $X_j^T X_j$ is first computed using the function `dpotrf` (`d`ouble `po`sitive definite `tr`iangular `f`actor) and then used to solve the system $X_j^T X_j \beta_j = X_j^T Y_j$ with the function `dpotrs` (`d`ouble `po`sitive definite `tr`iangular `s`olve). Also the implementation of `tclustreg` used in this work employs the Cholesky decomposition to solve the Least Squares problem.

- The Intel® oneAPI Math Kernel Library (oneMKL, `https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl.html`)

[64] is used for the function `vdLn`, which provides a vectorized (SIMD) implementation of the function $\log(x)$. This library is also used by MATLAB (as are BLAS and LAPACK), and thus by `tclustreg`.

3.4.2 Classification Expectation Maximization implementation

Although the Classification Expectation Maximization (CEM) [30] algorithm is not thoroughly discussed in this work, we need to at least outline its declination in the context of the problem at hand to better delineate the difference between our implementation and the standard ones. The standard implementation of the CEM comprises the following steps.

1. For every cluster j, fit the points and obtain estimates $\hat{\sigma}_j, \hat{\beta}_j, \hat{\pi}_j \forall j$.
2. for every point (x_i, y_i) in the dataset and for every j, compute $log(\hat{\pi}_j f_j(x_i, y_i))$ using $\hat{\sigma}_j, \hat{\beta}_j$ to compute $f_j(x_i, y_i)$. This value is the log-likelihood contribution of the point for every cluster it might be assigned to.
3. assign the point to the cluster with the highest $\log(\hat{\pi}_j f_j(x_i, y_i))$.
4. repeat from step 1 until no point is reassigned to a different cluster (or the objective function does not increase).

As we can notice, the values of $\hat{\sigma}_j, \hat{\beta}_j, \hat{\pi}_j$ remain fixed while the points are being reassigned in steps 2-3.

While performing CEM in the computation of the lower bounds, our method `pclustreg` instead computes the contribution of each point by implicitly partially re-estimating $\hat{\sigma}_j, \hat{\pi}_j$. While the parameters $\hat{\beta}_j$ and the sum of squared residuals SSR_j are not recomputed as the points are being reassigned, the values n_j (which denote the number of points assigned to each cluster j) are updated to keep track of the number of points in each cluster. This causes the estimate of the contribution of each point to each cluster used to decide the reassignment to be slightly more accurate at the cost of a higher number of evaluations of the function $\log(x)$.

Chapter 4

Conclusions and future research

The work done for this thesis has followed two closely related research lines: the clustering of Mixtures of Gaussians, discussed in Chapter 1, and the clusterwise linear regression problem, discussed in Chapters 2 and 3.

4.1 Clustering Mixtures of Gaussians

Regarding the clustering of Mixtures of Gaussians, in this work, we introduced a MISDP reformulation of the clustering problem for Gaussian Mixtures with ellipsoidal components. MISDP problems are iteratively solved to estimate unknown covariance matrices and centroids of the ellipsoidal clusters, as well as the assignment of each point to a given cluster. Such reformulation is at the core of our `bclust` proposal, which allows us to achieve a local optimum and to improve the solutions provided by traditional classification EM algorithms. The `bclust` algorithm iteratively refines its solution, starting from an initial assignment (which could be random or the output of other clustering algorithms). Extensive simulation and application studies show that `bclust` is very promising and typically outperforms a well-known existing method for model-based clustering such as `tclust`, albeit at the cost of a larger, but still reasonable, computational burden.

Our current proposal could be extended in various directions.
Firstly, we are particularly interested in problems where components may have different sizes (i.e., where the mixing proportions $\pi_1, ..., \pi_k$ are not necessarily equal). In principle, this can be achieved with additional optimization variables and linear constraints, and we are developing new heuristics to limit the computational burden.
Secondly, we are also exploring clustering problems affected by data contamination (i.e., noise and/or outliers). They can be tackled through additional binary and linear constraints, to ensure that contaminants are treated as an additional cluster that does not contribute to the objective function.
Thirdly, we are investigating more effective strategies to improve the solutions of MISDP subproblems and limit their computational cost. For instance, we are currently exploring a custom implementation of a branch and bound algorithm that directly calls the MOSEK API without relying on the YALMIP library, as the latter brings a considerable overhead.
Finally, we are currently studying various ways to choose the family of sets $\mathscr{F}$ which is used to define the neighborhood of the current solution, to provide a good trade-off between computational burden and optimality of the solution.

4.2 Clusterwise Linear Regression

For what concerns the clusterwise linear regression problem, in Chapter 2 we have provided a reformulation of the maximization of the log-likelihood for this setting. We then proposed a new algorithm, `pclustreg`. This algorithm, given two underestimates $\widetilde{\sigma}^2_{\text{MIN}}$ and $\widetilde{n}_{\text{MIN}}$ of the minimum variance and minimum cluster cardinality of the ground truth, returns a solution that, with a desired probability $s < 1$, is at least as good as the (unknown) ground-truth in terms of log-likelihood. Moreover, we showed with numerical experiments that `pclustreg` can be effectively used as a heuristic method by limiting the number of nodes expanded. Our tests showed that `pclustreg` is capable of outperforming, as a heuristic, the state-of-the-art implementation `tclustreg` in terms of quality of the result with comparable computation times.

In Chapter 3 we highlighted the limitations of the parameter $\widetilde{\sigma}^2_{\text{MIN}}$ and proposed a change in the method to address its drawbacks by exploiting the $\mathscr{F}$ distribution instead of the $\mathscr{X}^2$. In Section 3.2 we tested the refined implementation of `pclustreg` obtained in this way in three different scenarios with synthetic data obtaining promising results. In Section 3.3 we showed that `pclustreg` is competitive with `tclustreg` als on real-world datasets.

There are several avenues for further research on the approach used by `pclustreg`. We are currently investigating: (i) Estimating the value of $\widetilde{\sigma}_{\text{MIN}}$ by employing a robust trimming-based algorithm for regression or multivariate analysis [65] which aims to identify a single component and $n - \widetilde{n}_{\text{MIN}}$ outliers. (ii) Finding the optimal number of regression hyperplanes k with a linear search using some information criterion [66].

Bibliography

[1] Inke R König, Oliver Fuchs, Gesine Hansen, Erika von Mutius, and Matthias V Kopp. What is precision medicine? *European Respiratory Journal*, 50(4), 2017.

[2] Fan Yang, Heng Fan, Peng Chu, Erik Blasch, and Haibin Ling. Clustered object detection in aerial images. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8311–8320, 2019.

[3] Vesela Mihova and Velisar Pavlov. A customer segmentation approach in commercial banks. In *AIP Conference Proceedings*, volume 2025. AIP Publishing, 2018.

[4] Shazia Tabassum, Fabiola SF Pereira, Sofia Fernandes, and João Gama. Social network analysis: An overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8(5):e1256, 2018.

[5] Hans-Hermann Bock. Clustering methods: a history of k-means algorithms. pages 161–172. Springer, 2007.

[6] Luca Scrucca, Michael Fop, T. Brendan Murphy, and Adrian E. Raftery. Mclust 5: clustering, classification and density estimation using Gaussian finite mixture models. *The R Journal*, 8(1):289–317, 2016. URL: `https://doi.org/10.32614/RJ-2016-021`.

[7] Dustin G Mixon, Soledad Villar, and Rachel Ward. Clustering subgaussian mixtures by semidefinite programming. *Information and Inference: A Journal of the IMA*, 6(4):389–415, 2017.

[8] Xiaohui Chen and Yun Yang. Cutoff for exact recovery of gaussian mixture models. *IEEE Transactions on Information Theory*, 67(6):4223–4238, 2021.

[9] Damek Davis, Mateo Diaz, and Kaizheng Wang. Clustering a mixture of gaussians with unknown covariance. *arXiv preprint arXiv:2110.01602*, 2021.

[10] Sanjoy Dasgupta. Learning mixtures of gaussians. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 634–644. IEEE, 1999.

[11] Rong Ge, Qingqing Huang, and Sham M Kakade. Learning mixtures of gaussians in high dimensions. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 761–770, 2015.

[12] Geoffrey J McLachlan and Thriyambakam Krishnan. *The EM algorithm and extensions*. John Wiley & Sons, 2007.

[13] Henry Wolkowicz, Romesh Saigal, and Lieven Vandenberghe. *Handbook of Semidefinite Programming: Theory, Algorithms, and Applications*, volume 27. Springer Science & Business Media, 2012.

[14] William M Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.

[15] Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid Von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, et al. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2020.

[16] Geoffrey J. McLachlan and D. Peel. *Finite Mixture Models*. Wiley Series in Probability and Statistics. Wiley, 2004.

[17] Naveen Veeramisti, Alexander Paz, Mukesh Khadka, and Cristian Arteaga. A clusterwise regression approach for the estimation of crash frequencies. *Journal of Transportation Safety & Security*, 13(3):247–277, 2021.

[18] Anders Skrondal and Sophia Rabe-Hesketh. *Generalized latent variable modeling: Multilevel, longitudinal, and structural equation models*. Crc Press, 2004.

[19] Jean-Michel Poggi and Bruno Portier. Pm10 forecasting using clusterwise regression. *Atmospheric Environment*, 45(38):7005–7014, 2011.

[20] Michel Wedel and Wagner A Kamakura. *Market segmentation: Conceptual and methodological foundations*. Springer Science & Business Media, 2012.

[21] Luis Angel García-Escudero, Alfonso Gordaliza, Agustin Mayo-Iscar, and Roberto San Martín. Robust clusterwise linear regression through trimming. *Computational Statistics & Data Analysis*, 54(12):3057–3069, 2010.

[22] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the {R*oyal {Statistical {Society: {Series {B, 39(1):1–22, 1977.

[23] Qiang Long, Adil Bagirov, Sona Taheri, Nargiz Sultanova, and Xue Wu. Methods and applications of clusterwise linear regression: A survey and comparison. *ACM Transactions on Knowledge Discovery from Data*, 17(3):1–54, 2023.

[24] Neil E Day. Estimating the components of a mixture of normal distributions. *Biometrika*, 56(3):463–474, 1969.

[25] Jeffrey D Banfield and Adrian E Raftery. Model-based Gaussian and non-Gaussian clustering. *Biometrics*, pages 803–821, 1993.

[26] Gilles Celeux and Gérard Govaert. Gaussian parsimonious clustering models. *Pattern Recognition*, 28(5):781–793, 1995.

[27] Luis García-Escudero, Alfonso Gordaliza, Carlos Matrán, and Agustin Mayo. A general trimming approach to robust cluster analysis. *The Annals of Statistics*, 36:1324–1345, 06 2008. doi:10.1214/07-AOS515.

[28] Heinrich Fritz, Luis A. García-Escudero, and Agustín Mayo-Iscar. A fast algorithm for robust constrained clustering. *Computational Statistics & Data Analysis*, 61:124–136, 2013. URL: https://www.sciencedirect.com/

science/article/pii/S0167947312004203, doi:10.1016/j.csda.2012.11.018.

[29] Luis Angel García-Escudero, Agustin Mayo-Iscar, and Marco Riani. Model-based clustering with determinant-and-shape constraint. *Statistics and Computing*, 30:1363–1380, 2020.

[30] Gilles Celeux and Gérard Govaert. A classification EM algorithm for clustering and two stochastic versions. *Computational Statistics & Data Analysis*, 14(3):315–332, 1992.

[31] Jiming Peng and Yu Xia. A new theoretical framework for k-means-type clustering. *Foundations and Advances in Data Mining*, pages 79–96, 2005.

[32] Jiming Peng and Yu Wei. Approximating k-means-type clustering via semidefinite programming. *SIAM Journal on Optimization*, 18(1):186–205, 2007.

[33] Daniel Aloise and Pierre Hansen. A branch-and-cut SDP-based algorithm for minimum sum-of-squares clustering. *Pesquisa Operacional*, 29:503–516, 2009.

[34] Veronica Piccialli, Antonio M Sudoso, and Angelika Wiegele. SOS-SDP: an exact solver for minimum sum-of-squares clustering. *INFORMS Journal on Computing*, 34(4):2144–2162, 2022.

[35] Liuqin Yang, Defeng Sun, and Kim-Chuan Toh. Sdpnal+: a majorized semismooth newton-cg augmented lagrangian method for semidefinite programming with nonnegative constraints. *Mathematical Programming Computation*, 7(3):331–366, 2015.

[36] Veronica Piccialli and Antonio M Sudoso. Global optimization for cardinality-constrained minimum sum-of-squares clustering via semidefinite programming. *arXiv preprint arXiv:2209.08901*, 2022.

[37] Romy Shioda and Levent Tunçel. Clustering via minimum volume ellipsoids. *Computational Optimization and Applications*, 37(3):247, 2007.

[38] Yubo Zhuang, Xiaohui Chen, and Yun Yang. Likelihood adjusted semidefinite programs for clustering heterogeneous data. *arXiv preprint arXiv:2209.15097*, 2022.

[39] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[40] Gideon Schwarz. Estimating the dimension of a model. *The annals of statistics*, pages 461–464, 1978.

[41] Andrea Cerioli, Luis Angel García-Escudero, Agustin Mayo-Iscar, and Marco Riani. Finding the number of normal groups in model-based clustering via constrained likelihoods. *Journal of Computational and Graphical Statistics*, 27(2):404–416, 2018.

[42] Marco Riani, Domenico Perrotta, and Francesca Torti. FSDA: A MATLAB toolbox for robust analysis and interactive data exploration. *Chemometrics and Intelligent Laboratory Systems*, 116:17–32, 2012. URL: `https://www.sciencedirect.com/science/article/pii/S0169743912000974`, `doi:10.1016/j.chemolab.2012.03.017`.

[43] The MathWorks Inc. MATLAB version: 9.12.0 (r2022a), 2022. URL: `https://www.mathworks.com`.

[44] J. Löfberg. YALMIP : A Toolbox for Modeling and Optimization in MATLAB. In *In Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004.

[45] MOSEK ApS. *The MOSEK optimization toolbox for MATLAB manual. Version 10.0.21*, 2022. URL: `https://docs.mosek.com/10.0/toolbox/index.html`.

[46] Stefan Aeberhard and M. Forina. Wine. UCI Machine Learning Repository, 1991. DOI: https://doi.org/10.24432/C5PC7J.

[47] Magorzata Charytanowicz, Jerzy Niewczas, Piotr Kulczycki, Piotr Kowalski, and Szymon Lukasik. Seeds. UCI Machine Learning Repository, 2012. DOI: https://doi.org/10.24432/C5H30K.

[48] R. A. Fisher. Iris. UCI Machine Learning Repository, 1988. DOI: https://doi.org/10.24432/C56C76.

[49] Miscellaneous. Algerian Forest Fires Dataset. UCI Machine Learning Repository, 2019. DOI: https://doi.org/10.24432/C5KW4N.

[50] Francesca Torti, Domenico Perrotta, Marco Riani, and Andrea Cerioli. Assessing trimming methodologies for clustering linear regression data. *Advances in Data Analysis and Classification*, 13(1):227–257, 2019.

[51] George AF Seber and Alan J Lee. *Linear regression analysis*. John Wiley & Sons, 2012.

[52] András Prékopa. Boole-bonferroni inequalities and linear programming. *Operations Research*, 36(1):145–162, 1988.

[53] Reiner Horst and Hoang Tuy. *Global optimization: Deterministic approaches*. Springer Science & Business Media, 2013.

[54] Luis Angel García-Escudero, Alfonso Gordaliza, Francesca Greselin, Salvatore Ingrassia, and Agustín Mayo-Iscar. The joint role of trimming and constraints in robust estimation for mixtures of gaussian factor analyzers. *Computational Statistics & Data Analysis*, 99:131–147, 2016.

[55] Marco Riani, Domenico Perrotta, and Francesca Torti. Fsda: A matlab toolbox for robust analysis and interactive data exploration. *Chemometrics and Intelligent Laboratory Systems*, 116:17–32, 2012.

[56] Ranjan Maitra and Volodymyr Melnykov. Simulating data to study performance of finite mixture modeling and clustering algorithms. *Journal of Computational and Graphical Statistics*, 19(2):354–376, 2010.

[57] Robert B Davies. The distribution of a linear combination of $\chi 2$ random variables. *Journal of the Royal Statistical Society Series C: Applied Statistics*, 29(3):323–333, 1980.

[58] Arthur Griffith. *GCC: the complete reference*. McGraw-Hill, Inc., 2002.

[59] Brian Gough. *GNU scientific library reference manual*. Network Theory Ltd., 2009.

[60] Chuck L Lawson, Richard J. Hanson, David R Kincaid, and Fred T. Krogh. Basic linear algebra subprograms for fortran usage. *ACM Transactions on Mathematical Software (TOMS)*, 5(3):308–323, 1979.

[61] Jack J Dongarra, Jeremy Du Croz, Sven Hammarling, and Iain S Duff. A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software (TOMS)*, 16(1):1–17, 1990.

[62] James Demmel. Lapack: A portable linear algebra library for supercomputers. In *IEEE Control Systems Society Workshop on Computer-Aided Control System Design*, pages 1–7. IEEE, 1989.

[63] Edward Anderson, Zhaojun Bai, Christian Bischof, L Susan Blackford, James Demmel, Jack Dongarra, Jeremy Du Croz, Anne Greenbaum, Sven Hammarling, Alan McKenney, et al. *LAPACK users' guide*. SIAM, 1999.

[64] Endong Wang, Qing Zhang, Bo Shen, Guangyong Zhang, Xiaowei Lu, Qing Wu, Yajuan Wang, Endong Wang, Qing Zhang, Bo Shen, et al. Intel math kernel library. *High-Performance Computing on the Intel® Xeon Phi™: How to Fully Exploit MIC Architectures*, pages 167–188, 2014.

[65] Anthony C Atkinson, Marco Riani, and Andrea Cerioli. Cluster detection and clustering with random start forward searches. *Journal of Applied Statistics*, 45(5):777–798, 2018.

[66] Francesca Torti, Marco Riani, and Gianluca Morelli. Semiautomatic robust regression clustering of international trade data. *Statistical Methods & Applications*, 30:863–894, 2021.

Acknowledgements

I am grateful to my significant other, Giulia for always being by my side.
I am grateful to RetailTune S.r.l for sponsoring my Ph.D..
I am grateful to my family for supporting me throught the years.
I am grateful to my Ph.D. advisors Luca Consolini and Fabrizio Laurini, as well as to Marco Locatelli, Marco Riani and Luca Insolia for providing guidance, reassurement, valuable insights and in general for being helpful during the research that led to this thesis.
I am grateful to my friends for the many experiences we shared over the years.
I am grateful to my teachers and professors, from elementary school to the Ph.D. courses I took, for making me enjoy learning.