POLITECNICO DI TORINO

ALTA SCUOLA POLITECNICA

Master Degree
in Mathematical Engineering

Master Thesis

Maximum likelihood based clustering via parallel computing

Supervisors
prof. Mauro Gasparini
dr. Lidia Sacchetto

Co-Supervisor
prof. Anna Paganoni

Candidate
Alessandro Barilli

Supervisors' signature

. . . .
. . . .
. . . .

candidate's signature

. . . .

Academic Year 2018 – 2019

Abstract

The following pages contain a review of some recent and ongoing work on model-based clustering, in particular hard and soft assignment. It is analysed up to which point, with modern tools of optimization and parallel computing, it is possible to use basic methods such as maximum likelihood and hard assignment towards automatic identification of the classes and of the class labels of the sampled subjects. The standard soft classification approach using the EM (expectation maximization) algorithm will be compared to the hard assignment approach using maximum likelihood. The latter's limits will be analysed at a computational level, with and without the assumption of local independence. Limits and applications of such algorithms to real datasets will be shown.

In the first chapter I will introduce the problem and background; in the second chapter, the maximum likelihood estimation with the EM algorithm will be explained and implemented in R with the poLCA package; in the third chapter the hard and the soft methods will be compared for the easiest case (binary variables and two clusters); finally, in chapters 4 and 5 the hard assignment approach will be generalised to multiple clusters and different cluster structures (possible correlation among the predictors is taken into account).

Contents

1 Introduction to the problem **3**
- 1.1 Unsupervised learning and clustering . . . 3
  - 1.1.1 Short introduction to Unsupervised Learning . . . 3
  - 1.1.2 Clustering . . . 4
  - 1.1.3 Distances . . . 4
  - 1.1.4 Clustering algorithms using distances . . . 5
  - 1.1.5 Aim of the work . . . 6

2 Maximum Likelihood estimation **7**
- 2.1 Likelihood function . . . 7
- 2.2 Hard versus Soft Estimation . . . 9
  - 2.2.1 General introduction to the two approaches . . . 9
  - 2.2.2 Differences among the two approaches . . . 10
  - 2.2.3 Maximisation issues . . . 11
- 2.3 The EM algorithm . . . 12
  - 2.3.1 Theory of the Expectation Maximization Algorithm . . . 12
  - 2.3.2 Implementation in R for two populations . . . 17
  - 2.3.3 Implementation in R for more than two populations . . . 22

3 Hard estimation for two clusters **26**
- 3.1 The likelihood function for binary variables . . . 26
  - 3.1.1 Likelihood for two populations . . . 28
- 3.2 Implementation of the binary hard assignment in R . . . 29
  - 3.2.1 Parallelized version and performance . . . 39
- 3.3 Laplace smoothing . . . 45
- 3.4 Heuristic methods . . . 47
- 3.5 Applications . . . 56
- 3.6 Comparison with the EM algorithm . . . 58
  - 3.6.1 Symmetric case . . . 60

4 Hard estimation for multiple clusters **63**
- 4.1 The likelihood function for dicotomous answers and more than two populations . . . 63
- 4.2 Implementation in R using dplyr package . . . 64
- 4.3 Heuristic methods . . . 68

5 Hard estimation using DAGs **73**
- 5.1 Introduction . . . 73
- 5.2 Theorical introduction to DAGs . . . 74
- 5.3 bnlearn package . . . 75
- 5.4 Clustering implementation . . . 77

A Binary and non-binary classification **83**

B DAG classification **101**

1 Introduction to the problem

1.1 Unsupervised learning and clustering

1.1.1 Short introduction to Unsupervised Learning

Supposing to have a dataset X of N observations:

$$X = (x_1, ..., x_N)$$

Each x_j is commonly called "predictor", and it represents a characteristic of the observation, for example the weight/height of each individual in case. the dataset refers to people. In general such predictors could be referred to various characteristics: later on, examples will be treated in which the predictors come from a survey, in detail the answers will be qualitative and not quantitative, so each x observation (in this case, answer to r questions) will be of the following form:

$$x_j = (x_{j1}, ..., x_{jr}), \quad j \in \{1, ..., N\}$$

where x_{jl}, with $l \in \{1, ..., r\}$ can take categorical values, mapped to the subset of natural numbers $\{0, ..., n_l - 1\}$, where n_l is the number of possible answers to question l.
This should not confuse and should not lead to the belief that predictors refer to numerical data: they are only a convention: thus the values of $\{0, 1\}$ could refer to a question whose possible outcome is "true" or "false", and a large part of the future dissertation will be focused on such kind of observations.

The focus of the following pages will be on the fact that, unlike classification problems, there will be no labels available for observations, but it will be the outcome of the algorithms to assign the observed objects to a number of k different classes. This is part of what is called "Unsupervised Learning", where "unsupervised" refers to the fact that there is no information available on the labels of the observations, while "learning" refers to the fact that the outcome of these algorithms may lead to the discovery of new information intrinsic to the observed data.

1.1.2 Clustering

One of the classic problems of unsupervised learning is clustering, i.e. the problem of assigning observations to different groups. The number of different possible groups, depending on the methods used, could be defined a priori or a posteriori. We will now show the classic approaches to clustering algorithms and the concept of distance, as opposed to its probabilistic formulation, by using the maximum likelihood to make soft (in particular, using the EM algorithm) and hard classification.

1.1.3 Distances

Let's consider the set of the previously defined N observations:

$$X = (x_1, ..., x_N)$$

where x_j belongs to a certain set X. On this set X it is possible to suppose that a metric function d exists, i.e. by the definition of metric, d should fulfill the following criteria:

1. symmetry: $d(x, y) = d(y, x)$
2. non-negativity: $d(x, y) \geq 0$
3. identity of indiscernibles: $d(x, y) = 0 \leftrightarrow x = y$
4. triangle inequality: $d(x, z) \leq d(x, y) + d(y, z)$

The previous properties must hold for each x, y, z in X.

Supposing quantitative observations in the set R^n, it is possible to define in such set the standard Euclidean distance:

$$d(x_i, x_j)^2 := d_{ij}^2 := \sum_{k=1}^{n} (x_{i,k} - x_{j,k})^2$$

And the matrix of distances is well defined:

$$D := (d_{ij})_{i,j \in \{1,...,N\}} = (d(x_i, x_j))_{i,j \in \{1,...,N\}}$$

If there are categorical data, let's suppose that each component of the x_j takes r possible values, which can then be mapped to the following set:

$$x_{jk} \in \{0, 1, ..., r - 1\}$$

In this case, it makes sense to introduce a dissimilarity distance. For example, if x_{jk} can take the values $\{A, B, C\}$, then $r = 3$ and we can introduce the function that is 1 if $x_{ik} = x_{jk}$ and induces a relative distance (considering the Euclidean distance does not make sense for categorical values).

1.1.4 Clustering algorithms using distances

If the observations are quantitative, the most well known algorithm in the literature is the k-Means. We define a priori the number of groups k in which to divide the data, and we choose a measure of dissimilarity, typically the Euclidean distance defined previously, so as to have the matrix of distances D.

Since the problem of clustering in k groups has combinatorial complexity, a method of minimizing a loss-function is introduced, in this case defined as follows: suppose that the point x_i belongs to the cluster h-th, so we will indicate $C(i) = h$. The loss function is defined as:

$$W(C) = \frac{1}{2} \sum_{h=1}^{k} \sum_{C(i)=h} \sum_{C(j)=h} d_{i,j}$$

where C is the configuration, i.e. the cluster assignments of the elements in X. The minimization of this quantity is carried out in an iterative way and tries to minimize the dissimilarity of the elements within a cluster from the mean of the cluster (here the hypothesis of quantitative data is necessary).

On the other hand, if the data is qualitative, it does not make sense to calculate an average of observations in a cluster, while it appears appropriate to define a "central element" in a cluster, so as to be able to minimise the distances within each cluster between each observation and that central element. This algorithm is called k-Medoids, and can be more robust and less sensitive to outliers than k-Means, since the latter is based on a minimization of Euclidean distances, which are greatly influenced by outliers, since it evaluates squared differences (while for k-Medoids, it will be sufficient to choose a different metric of dissimilarity).

1.1.5 Aim of the work

The following chapters contain an explanation of the soft and hard assignment. In detail, in the chapter 2 is introduced the problem of model-based clustering from a theoretical perspective, introducing the likelihood function. Therefore, in the section 2.2 is presented a first comparison between the hard and soft methodologies, their differences and maximisation issues that may appear in the assignment problem.

The soft methodology and the Expectation Maximization algorithm (EM) are explained and implemented in the section 2.3, using the R package *poLCA*: first, it is introduced the case of two populations, and then the methodology will be generalised.

In the chapter 3 the likelihood function for binary variables is explained, generalising the evaluation to two populations. Then, an exact implementation of the hard methodology is implemented in R, making use of parallel computing. Memory and timing limits of the exact method will be shown, and in the section 3.4 some heuristics will be implemented to tackle those issues. In the section 3.5 the hard methodology is applied to the dataset analysed with the soft approach, and a comparison between the two methods is explained in the section 3.6.

Then, in the chapter 4 the hard assignment is generalised to multiple clusters and non binary answers: a theoretical introduction is included in the section 4.1, and the technical implementation using the *dplyr* R package is explained in the section 4.2. To overcome the previously mentioned issues of timing and memory, some heuristics are implemented in the section 4.3.

Finally, in the chapter 5 the hard assignment is generalised to different clusters' structure using Directed Acyclic Graphs (DAG), taking into account possible correlations among the predictors in the section 5.2. The R package *bnlearn* is introduced in the section 5.3 and the implementation is shown in the chapter 5.4.

2 Maximum Likelihood estimation

2.1 Likelihood function

Suppose to have a random variable X with a density function $f_X(x;\lambda)$ given with a parameter λ. Let's suppose we observe a x_1 from such distribution; we then define the likelihood function as follows:

$$\mathcal{L}(\lambda; x_1) := f(x_1; \lambda)$$

i.e., the likelihood is a function of λ instead of x, as it happens for the density function. We can also define the logarithm of the likelihood function since it would be useful to deal with with maximization problems: actually, it does not make any difference to maximise a function or its logarithm where the function is positive, but we are dealing with probability functions (in case of zeros of the likelihood function, the logarithm would be -Inf, and since the scope is to maximise such function, that would not be computationally a problem). The log-likelihood function will be written as l and defined then in the following way:

$$l(\lambda; x_1) := log(\mathcal{L}(\lambda; x_1))$$

The MLE (Maximum Likelihood Estimation) consists in solving the following optimization problem, finding the optimum λ that will be called $\hat{\lambda}$, such that:

$$l(\hat{\lambda}; x_1) := \max_{\lambda} l(\lambda; x_1)$$

Now suppose we have $X_1, .., X_n$ iid X, and we observe n independent samples of $\{x_1, ..., x_n\}$ from such population. Due to independence, the likelihood function is the product of the individual likelihood functions:

$$\mathcal{L}(\lambda; x_1, ..., x_n) := \prod_{j=1}^{n} f(x_j; \lambda)$$

Then, following from the properties of logarithms, the log-likelihood function l becomes:

$$l(\lambda; x_1, ..., x_n) = \sum_{j=1}^{n} log(f(x_j; \lambda))$$

Notice that λ can be more than one scalar value, for example, for the Normal case we have both μ and σ^2 which define the distribution function. In this case, supposing we already observed $\{x_1, ..., x_n\}$, the log Likelihood function becomes:

$$l(\mu, \sigma^2; x_1, ..., x_n) = -\frac{n}{2} log(2\pi) - \frac{n}{2} log(\sigma^2) - \frac{1}{2\sigma^2} \sum_{j=1}^{n} (x_j - \mu)^2$$

This leads to the following maximisation problem:

$$l(\hat{\mu}, \hat{\sigma}; x_1 ..., x_n) := \max_{\mu,\sigma} \left(-\frac{n}{2} log(2\pi) - \frac{n}{2} log(\sigma^2) - \frac{1}{2\sigma^2} \sum_{j=1}^{n} (x_j - \mu)^2 \right)$$
$$= \left(-\frac{n}{2} log(\sigma^2) - \frac{1}{2\sigma^2} \sum_{j=1}^{n} (x_j - \mu)^2 \right)$$

Setting the partial derivatives (respect to μ and σ^2) equal to zero, the following result is obtained:

$$\hat{\mu} = \frac{\sum_{j=1}^{n} x_j}{n}$$
$$\hat{\sigma^2} = \frac{\sum_{j=1}^{n} (x_j - \mu)^2}{n}$$

It is not always so trivial to solve such optimisation problem, since in this lucky case we can evaluate derivatives and obtain analytically the global maximum, but, in general, this is not feasible, and other strategies are required.

We can further improve this model, assuming the presence of two underlying distribution functions behind our data: in such case, using a "hard" approach (which will be discussed afterwards) each unit may be extracted from one of the two distributions. Let's suppose the independence among $x_1, ..., x_n$ and among the clusters. Then, to find the best approximation of the two distributions it is necessary to find, for each configuration of $x_1, ..., x_n$ in each of the two clusters, the best parameters $\hat{\lambda}$ and $\hat{\sigma}$ for such configuration. The configuration that returns the highest likelihood will be considered as the best one that fits the data.

If the data are not extracted from a normal population, in general, the problem becomes more difficult, since there is no general analytical solution to it. Some approaches will be presented afterwards to face this issue.

2.2 Hard versus Soft Estimation

2.2.1 General introduction to the two approaches

In clustering problems, two methodologies are available: a hard approach and a soft one.

The hard assignment assigns every observed unit to a group (cluster) only [9]: from a theoretical point of view (that will be developed afterwards), a label γ is assigned to each unit such that

$$\gamma_j = k$$

means that the point j is assigned to the cluster k, i.e. the cluster C_k may be defined as the set of all units j such that $\gamma_j = k$. The γ are estimated with the maximum likelihood method, fitting pre-defined distributions over the observed units.

The soft assignment is instead about assigning to each unit a probability (or degree of membership) to belong to each possible cluster [3] [4]. Such probabilities may be estimated using an approximation of the likelihood function, assuming that each point has a probability π_l to belong to each cluster, with a total of K possible populations:

$$f(x) = \sum_{l=1}^{K} \pi_l f(x, \theta_l)$$

Such probabilities may be estimated making use of the Latent Class modeling, i.e. assuming that there exist an unobserved variable (for each unit) that tells from which class the unit is coming from. The EM algorithm will be explained on the basis of this concept, assuming the local independence of the variables.

2.2.2 Differences among the two approaches

In general the hard and soft approaches differ. The differences can be explained by the fact that the two methodologies find the "best" configuration (depending on the method used, it may be a local optimum), maximising a different function, that for the hard approach is the likelihood function, while for the soft is, as said, an approximation of it, assuming that each unit may belong to each cluster with a certain probability. In terms of formulas, assuming for simplicity two clusters (+ and -) and local independence of the variables, the two functions to be maximised are written as follows:

$$\mathcal{L}_H = \prod_{i=1}^{n} f(x_i;\theta^+)^{\gamma_i} f(x_i;\theta^-)^{1-\gamma_i}$$

$$\mathcal{L}_S = \prod_{i=1}^{n} [\eta f(x_i;\theta^+) + (1-\eta) f(x_i;\theta^-)]$$

where η is the global probability to be extracted from the first cluster, and so the individual probability to belong to the first cluster $\hat{\pi}_i$ is:

$$\hat{\pi}_i = \frac{\hat{\eta} f(x_i;\theta^+)}{\hat{\eta} f(x_i;\theta^+) + (1-\hat{\eta}) f(x_i;\theta^-)}$$

With the EM algorithm, the results are found maximising the $\mathcal{L}_S$ function via a two steps approach, called respectively "expectation" and "maximization".

2.2.3 Maximisation issues

As a general statement, the maximisation problem is not immediate, since it may be possible to find trivial maxima of the Likelihood function equal to infinite. Supposing to have observed $\{x_1, ..., x_n\}$, extracted from two gaussian distributions Y_1 and Y_2 with unknown parameters, a trivial solution (with infinite likelihood) would be the following, where one of the two Gaussian is the Dirac's Delta, i.e.:

$$Y_1 \sim N(x_j, 0)$$

For each j and for each possible Y_2, those two distributions solve the maximum likelihood problem, but obviously are not much informative. To overcome this issue, some bounds on the variance will be imposed in the EM algorithm, for example imposing σ_1 and σ_2 bigger than 0.

2.3 The EM algorithm

2.3.1 Theory of the Expectation Maximization Algorithm

In order to solve the maximisation problem, the "expectation maximisation" algorithm has been introduced. The EM algorithm is an iterative process composed by two parts: expectation and maximization.

One application of this algorithm can be used to make soft assignments for values extracted from two Gaussian populations. The case will be later generalized to other families of distributions.

Returning to the example discussed above of data extracted from two Gaussian variables, suppose to define the following two distributions with the following parameters:

$$Y_1 \sim N(\mu_1, \sigma_1)$$
$$Y_2 \sim N(\mu_2, \sigma_2)$$

and let suppose to extract some identically distributed samples from the two populations:

$$X_{11}, ..., X_{n_1 1} \quad iid \quad Y_1$$
$$X_{12}, ..., X_{n_1 2} \quad iid \quad Y_2$$

Clearly a priori it is impossible to know from which of the two distributions the observations come from, so the idea behind the EM algorithm is to add a latent (i.e. not observed, considering it as "missing" data to estimate) random variable taking values into $\{0, 1\}$ (which will be called Δ):

$$\Delta \sim bernoulli(\pi)$$

Where π is the probability for Δ to be equal to 1. Then, once Δ has been defined, it is possible to define the mixture Y of the two underlying Gaussians Y_1 and Y_2:

$$Y := (1 - \Delta)Y_1 + \Delta Y_2$$

So in the case Δ assumes value 1, the corresponding observation Y belongs to class 2. Y is not a gaussian distribution, and its probability density function (pdf) is the following (it can computed by the above definition of Y):

$$f_Y(x; \theta) = (1 - \pi) f_{Y1}(x; \theta) + \pi f_{Y2}(x; \theta)$$

where, in the previous formula, θ is a vector containing all the following parameters introduced:

$$\theta = (\pi, \sigma_1, \sigma_2, \mu_1, \mu_2)$$

and f_1 e f_2 are the pdfs of the gaussians Y_1 and Y_2.

Making a hypothesis of independence among the extractions from the two samples, it is possible to

rewrite the likelihood function as a product, given the observations of the N values extracted from the two populations:

$$\mathcal{L}(\theta; x_1, ..., x_N) = \prod_{j=1}^{N} ((1-\pi)\mathcal{L}_{Y_1}(\theta_1; x_1, ..., x_N) + \pi\mathcal{L}_{Y_2}(\theta_2; x_1, ..., x_N))$$

where $\theta_k = (\mu_k, \sigma_k)$ for $k = 1, 2$. Thus, the log-likelihood can be rewritten as the following sum::

$$l(\theta; x_1, ..., x_N) = \sum_{j=1}^{N} log\left((1-\pi)\mathcal{L}_{Y_1}(\theta_1; x_1, ..., x_N) + \pi\mathcal{L}_{Y_2}(\theta_2; x_1, ..., x_N)\right)$$

So it would be necessary to solve the problem of optimization in the set where θ is defined, since $x_1, ..., x_N$ are observed and l is a function of θ only.

However the problem can be simplified through the following assumption: suppose it is known that $\Delta_j = 1$, then the corresponding observation j would come, as said, from the distribution 2 (vice versa $\Delta_j = 0$ implies that j comes from distribution 1). So let's suppose Δ_j is known, therefore it is possible to rewrite the function of log-likelihood in the following way (instead of π it is sufficient to use Δ_j, supposed as known), and rename it l_0:

$$\begin{aligned} l_0(\theta; &x_1, ..., x_N, \Delta_1, ..., \Delta_N) = \\ &= \sum_{j=1}^{N} [(1-\Delta_j)log(\mathcal{L}_{Y_1}(\theta_1; x_1, ..., x_N)) + \Delta_j log(\mathcal{L}_{Y_2}(\theta_2; x_1, ..., x_N))] + \\ &+ \sum_{j=1}^{N} [(1-\Delta_j)log(1-\pi) + \Delta_j log\pi] \end{aligned}$$

In this way, it is clear that the maximizers of σ_k and μ_k are the sample mean and the sample standard deviation of the variables belonging to the groups 1 and 2.

Then, since we do not have any information a priori about the groups, first of all we have to estimate Δ_j, and this is the first step of the algorithm, the expectation: it is reasonable to approximate at each step the random variable Δ_j with his expected value, that will be defined in the following way (for all j in $\{1, ..., N\}$):

$$\begin{aligned} \gamma_j := &E[\Delta_j|\theta, x_1, ..., x_N] = \quad (since \quad \Delta_j \sim be(\pi)) \\ &= P(\Delta_j = 1|\theta, x_1, ..., x_N) \end{aligned}$$

γ_j is also called "responsibility" for j, and indicates the posterior probability for the sample to belong to the class 2.

Then it is necessary to initialise all the parameters contained in θ, i.e. $(\pi, \sigma_1, \sigma_2, \mu_1, \mu_2)$: it is reasonable to initialise $\pi = 0.5$ since it's not present any information a priori about the belonging

of a variable to the class 2, while for σ_k, μ_k it is possible to use the information contained in the sample, i.e., μ_1 and μ_2 can be initialised as two random observations z_1 and z_2, while σ_k can be estimated as the overall standard deviation (for k in $\{1, 2\}$):

$$\begin{aligned}
\pi^0 &= 0.5 \\
\mu_1^0 &= z_1 \\
\mu_2^0 &= z_2 \\
(\sigma_k^0)^2 &= \frac{1}{N-1}\sum_{j=1}^{N}(x_j - \bar{x})^2
\end{aligned}$$

It is important to notice that the global maximum of the likelihood function for this mixture problem is infinite: in fact, assuming $\mu_1 = x_n$ for some observed x_n, and $\sigma_1 = 0$, an infinite likelihood is obtained, since the gaussian curve tends to approximate the Dirac's Delta function.

In order to avoid this, it is essential to look for solutions that include:

$$\hat{\sigma_1}, \hat{\sigma_2} > 0$$

Keeping in mind that the algorithm will converge to a local maximum (the global maximum is infinite, and not at all useful), the outcome of the algorithm (i.e. the θ) may not be the "best" one, since there can exist multiple local maximums. As the problem grows in dimensionality, this is even more and more likely to happen.

The structure of the EM algorithm, as said, have the following form for the mixture problem:

1. initialization of θ
2. expectation step: updating the responsibilities γ_j for each j in $\{1, ..., n\}$:

$$\hat{\gamma_j} \leftarrow \frac{\pi f_{Y_2}(y_j; \theta_2)}{(1-\pi) f_{Y_1}(y_j; \theta_1) + \pi f_{Y_2}(y_j; \theta_2)}$$

3. maximization step: for this mixture problem, it is sufficient to update sample standard deviations and means (in a "soft" way, considering the already estimated γ_j):

$$\hat{\mu_1} \leftarrow \frac{\sum_{j=1}^{N}(1-\gamma_j)y_j}{\sum_{j=1}^{N}(1-\gamma_j)}$$

$$\hat{\mu_2} \leftarrow \frac{\sum_{j=1}^{N}\gamma_j y_j}{\sum_{j=1}^{N}\gamma_j}$$

$$(\hat{\sigma_1})^2 \leftarrow \frac{\sum_{j=1}^{N}(1-\gamma_j)(y_j-\mu_1)^2}{\sum_{j=1}^{N}(1-\gamma_j)}$$

$$(\hat{\sigma_2})^2 \leftarrow \frac{\sum_{j=1}^{N}\gamma_j(y_j-\mu_2)^2}{\sum_{j=1}^{N}\gamma_j}$$

$$\hat{\pi} \leftarrow \frac{\sum_{j=1}^{N}\gamma_j}{N}$$

Step 2 and 3 are iterated until the algorithm converges with respect to some criteria.

Of course the procedure can be extended to other density functions, introducing latent unobserved variables in the problem (this technique is called "augmentation"). For this general case, we define the set of complete data T as follows:

$$T := Z \cup Z^m$$

where Z is the set of observed data, while Z^m is the set of the latent (or, unobserved or missing) variables. Coming back to the gaussian mixture, it was the case that $Z = \{x_1, ..., x_N\}$ and $Z^m = \{\Delta_1, ..., \Delta_N\}$. The likelihood will be therefore written as $l_0(\theta; T)$. In addition, a function Q (of one variable, θ) is introduced as follows (supposing to be in the n-th step of the algorithm, therefore there is an approximation of θ equal to $\theta^{(n)}$):

$$Q(\theta; \theta^{(n)}) := E[l_0(\theta; T)|Z, \theta^{(n)}]$$

Instead of maximising the likelihood function, it can be shown the equivalence of maximising the previous Q function. For the gaussian mixture case, the l_0 function to evaluate the expected value is:

$$\begin{aligned}
l_0(\theta; & x_1, ..., x_N, \Delta_1, ..., \Delta_N) = \\
& = \sum_{j=1}^{N}[(1-\Delta_j)log(\mathcal{L}_{Y_1}(\theta_1; x_1, ..., x_N)) + \Delta_j log(\mathcal{L}_{Y_2}(\theta_2; x_1, ..., x_N))] + \\
& + \sum_{j=1}^{N}[(1-\Delta_j)log(1-\pi) + \Delta_j log\pi]
\end{aligned}$$

then, Q is simply obtained by changing Δ_j with $\gamma_j(\theta^{(n)})$ (last updated value among the iterations). Then the generalised EM algorithm can be summarised as follows:

1. initialization of $\theta^{(0)}$
2. expectation step: compute the Q function:

$$Q(\theta;\theta^{(n)}) := E[l_0(\theta;T)|Z,\theta^{(n)}]$$

3. maximization step: Q needs to be maximised, updating $\theta^{(n+1)}$:

$$Q(\theta^{(n+1)};\theta^{(n)}) = \max_{\theta} Q(\theta;\theta^{(n)})$$

As the gaussian mixture case, the steps 2 and 3 need to be repeated until convergence. Notice also that the third step generalises the gaussian mixture case which was only about finding the sample mean and the sample standard deviation, the MLEs.

q1	q2	q3	q4	frequency
1	1	1	1	15
1	1	0	1	23
1	1	1	0	7
0	1	1	1	4
1	0	1	1	1
1	1	0	0	7
1	0	0	1	6
0	1	0	1	5
1	0	1	0	3
0	1	1	0	2
0	0	1	1	4
1	0	0	0	13
0	1	0	0	6
0	0	0	1	4
0	0	1	0	1
0	0	0	0	41

Figure 1: Observed profiles from the math test

2.3.2 Implementation in R for two populations

The theory of the EM algorithm previously described has an implementation in R in the poLCA (polytomous latent class analysis) package [8]:

```
library(poLCA)
```

The example dataset that is going to be used is a survey presented in Bartholomew et al. "Analysis of Multivariate Social Science Data" 2011 (p.284-285), that contains data about a math test composed of 4 questions performed by 142 individuals: if the individual answers are grouped by profiles, the dataset shown in Figure 1 is obtained.

The 1 and 0 answers refer to the correctness of the answer, i.e. 1 refers to a correct answer given by a certain number of inidividuals (frequency), while the 0 refers to a wrong answer. The purpose of a Latent Class Analysis (LCA) composed of two classes would be to divide the individuals between "good" and "bad" students, using the expectation maximization algorithm applied to the latent class Δ introduced in the previous chapter. Therefore, the posterior probabilities of belonging to a certain class will be obtained via an iterative process.

In the poLCA package, it is needed to have individual observations rather than the profiles, and the dataset in the appropriate format may be obtained via the following lines:

```
# profiles' dataset reading
data_Bart <- read.table('Bartholomew_dataset.txt',
    sep ='\t', header=T)

# creation of the individual dataset
freq.to.long <- function(x, freq){x[rep(1:length(freq), freq), ]}
data1 <- freq.to.long(data_Bart, data_Bart$Observed_freq)
dataset <- as.matrix(data1[, 3:6])
dim(dataset)

> dim(dataset)
[1] 142   4
```

The algorithm will work under the hypothesis of global independence of $x_1, ..., x_N$ (where $N = 142$, number of individuals participating to the test). In addition, it is essential to make a (strong) assumption about the independence of the answers to the 4 questions (local independence hypotesis). The individual's answers to the questions can be modelled as the following two random variables (one for each group) taking values in $\{0, 1\}^4$:

$$Y_1 = (Y_{11}, Y_{12}, Y_{13}, Y_{14})$$
$$Y_2 = (Y_{21}, Y_{22}, Y_{23}, Y_{24})$$

where $Y_{k,l} \sim be(p_{k,l})$ for each $k = \{1, 2\}$ and $l = \{1, 2, 3, 4\}$. The distribution functions for Y_k will have the following form (assuming local independence):

$$\begin{aligned} f_{Y_k}(a; \theta_k) &= P(Y_k = (a_1, a_2, a_3, a_4)) = \\ &= P(Y_{k1} = a_1)P(Y_{k2} = a_2)P(Y_{k3} = a_3)P(Y_{k4} = a_4) \end{aligned}$$

where $a = (a_1, a_2, a_3, a_4)$, and $\theta_k = (p_{k,1}, p_{k,2}, p_{k,3}, p_{k,4})$ are the parameters of the two distributions. By its definition, it is also given that

$$Y_{k,l} \sim be(p_{k,l})$$

and by hypotesis, each of the $Y_{k,l}$ is independent to each other. Therefore, under all these assumptions, it is possible to rewrite:

$$f_{Y_k}(a;\theta_k) = \prod_{l=1}^{4} p_{k,l}^{a_l}(1 - p_{k,l})^{(1-a_l)}$$

As the two distribution functions have been well defined, it is possible to proceed with the EM algorithm, having introduced the Δ latent variable such that:

$$Y = (1 - \Delta)Y_1 + \Delta Y_2$$

Using the poLCA function, in this case, we built a model without covariates (as expressed with "~ 1" in the function call). It is necessary to set some parameters for the poLCA iterations:

```
nclass = 2 # number of classes
maxiter = 5000 # maximum number of iterations for the EM
nrep = 5 # number of models to evaluate
```

The model is then built using the previous dataset whose names of the columns are V1, V2, V3, V4 (referring to the binary answers). Furthermore a starting seed is set in order to obtain the same results if the function is run more than once.

```
# building and solving the poLCA model assuming no covariates
set.seed(1)
lc <- poLCA(cbind(V1, V2, V3, V4) ~ 1,
  data = as.data.frame(dataset + 1),
  nclass = nclass, nrep = nrep, verbose = T)
```

```
========================================================
Fit for 2 latent classes:
========================================================
number of observations: 142
number of estimated parameters: 9
residual degrees of freedom: 6
maximum log-likelihood: -331.7637

AIC(2): 681.5273
BIC(2): 708.1298
G^2(2): 8.965682 (Likelihood ratio/deviance statistic)
X^2(2): 9.459245 (Chi-square goodness of fit)
```

The posterior probabilities can be found in the lc object, and so also the clustering labels (i.e. the belonging to the + or - population, evaluated with the posterior probability, using a 0.5 threshold). A plot of such probabilities can be seen in fig 2.

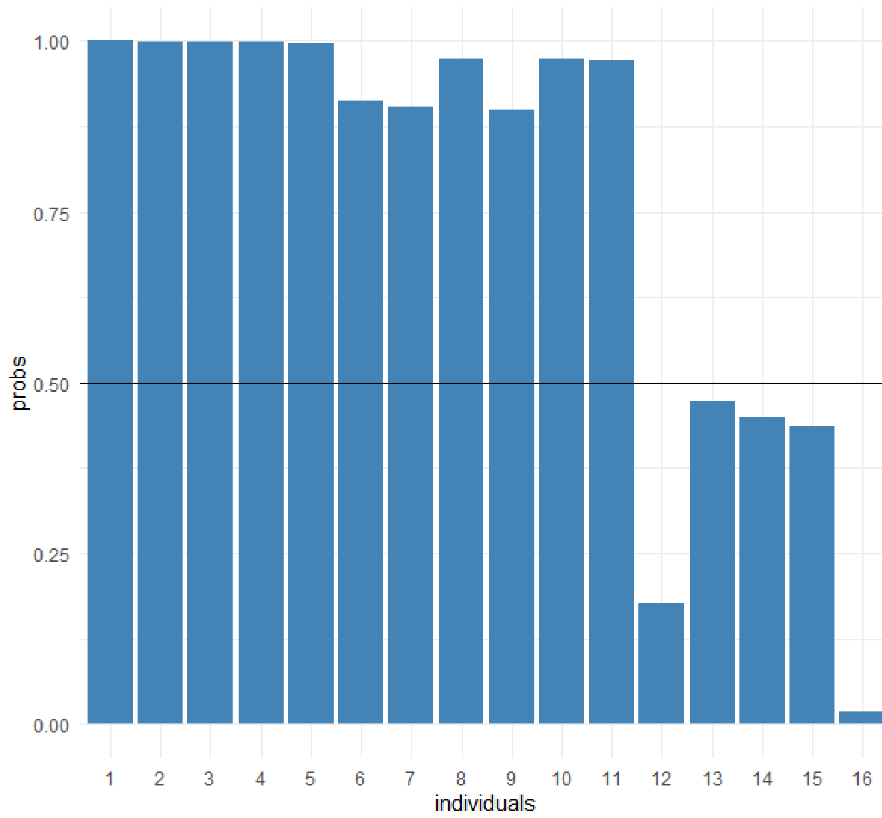

Figure 2: Posterior probabilities estimated with the EM algorithm integrated in the poLCA R package

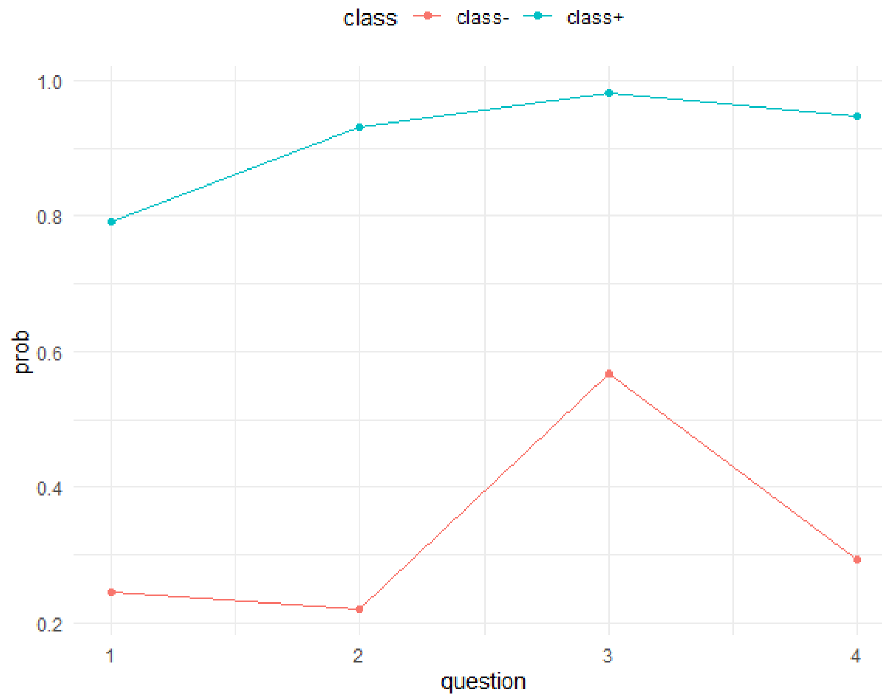

Figure 3: Probability of answering correctly to each question

All the observations below the threshold 0.5 (see horizontal line in the fig 2) will be assigned to the - group, while all the observations above it will be assigned to the + one. It is also informative to see the probability of answering correctly to each question, inside each of the two clusters: what is expected, is that the + class has higher probabilities than the - class. Looking inside the lc structure obtained by the poLCA package, it is possible to retrieve such values which are plotted in the fig 3.

2.3.3 Implementation in R for more than two populations

The poLCA package allows to fit more than two distributions to the data, compare the different outcomes on the basis of the BIC (Bayesian Information Criterium): the BIC is returned for each estimated model, so it could be a good option to look for the "best" model investigating among all the returned BICs.

In detail, the BIC criterium is a value assigned to a model, and it allows to compare models with a different number of predictors, using the Likelihood principle: it is not sufficient to compare different models using only the likelihood, since the more predictors are used, the more the Likelihood increase, and this fact may lead to overfitting. Therefore the BIC is defined as follows:

$$BIC = -2log(\mathcal{L}) + klog(n)$$

where $\mathcal{L}$ is the Likelihood, k the number of parameters of the model, and n is the number of observed values. Then it is clear that, in case the Likelihood is the same, the BIC criterium will assign a better value to the smallest model, since the "best" model is identified with the smallest BIC.

It is therefore possible to build N models that have a number of classes from 2 to N + 2, and choose the best among them. The estimation can be done as follows, assuming k as the number of classes, different for each model tested:

```
# defining the number of models
N_MODELS <- 4

# BIC values for each model
bicVector = rep(0, N_MODELS)

# evaluating each model
for (k in 2:(N_MODELS + 1)){
  lc <- poLCA(cbind(V1, V2, V3, V4) ~ 1,
    data = as.data.frame(dataset + 1),
    nclass = k, nrep = 5, verbose = T)
  bicVector[k - 1] <- lc$bic
}
```

and therefore it is possible to plot the BIC vector (bicVector) containing the BIC value for each model, as shown in Figure 3.

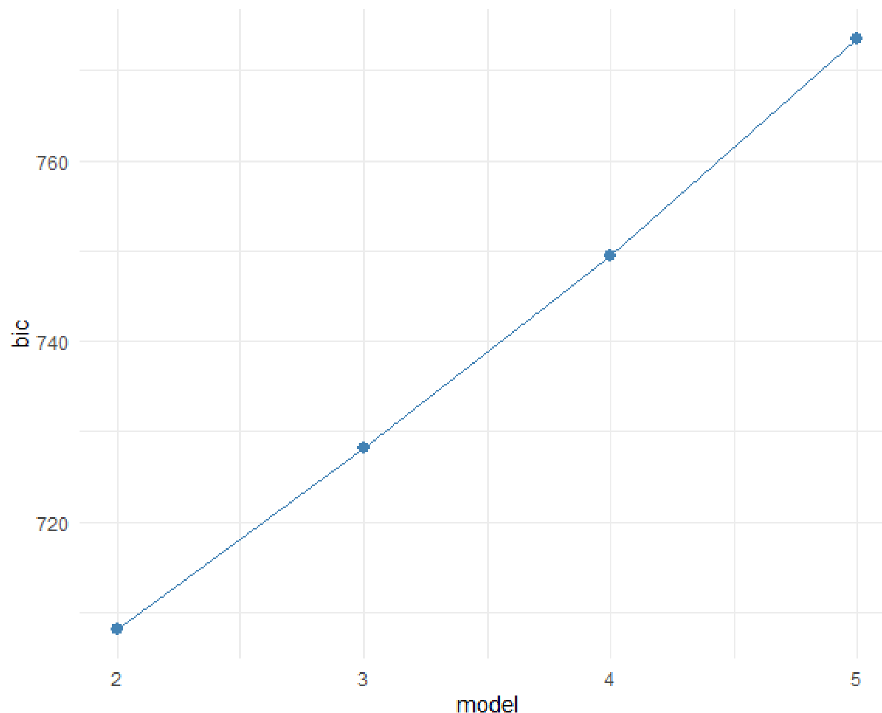

Figure 4: BIC obtained values for each model

As it is clear from the graph, the bigger the parameter set is, the more the BIC is penalised, then the criterium seems to be in favour of the smallest model containing two groups. The interpretability of the model (together with the overfitting) could be also affected by bigger models: in this case it has been said that a model of 2 groups would represent good and non-good students. In case of three groups, it could be used to model "good", "non-good" and "neutral" students. It may be then informative to analyse the $k = 3$ model, despite the BIC seems to prefer the smallest one with $k = 2$:

```
# building and solving the poLCA model for k=3 assuming no covariates
lc_neutral <- poLCA(cbind(V1, V2, V3, V4) ~ 1,
  data = as.data.frame(dataset + 1),
  maxiter <- 2000, nclass = 3, nrep = 30, verbose = T)
```

And the obtained results are the following:

```
========================================================
Fit for 3 latent classes:
========================================================
number of observations: 142
```

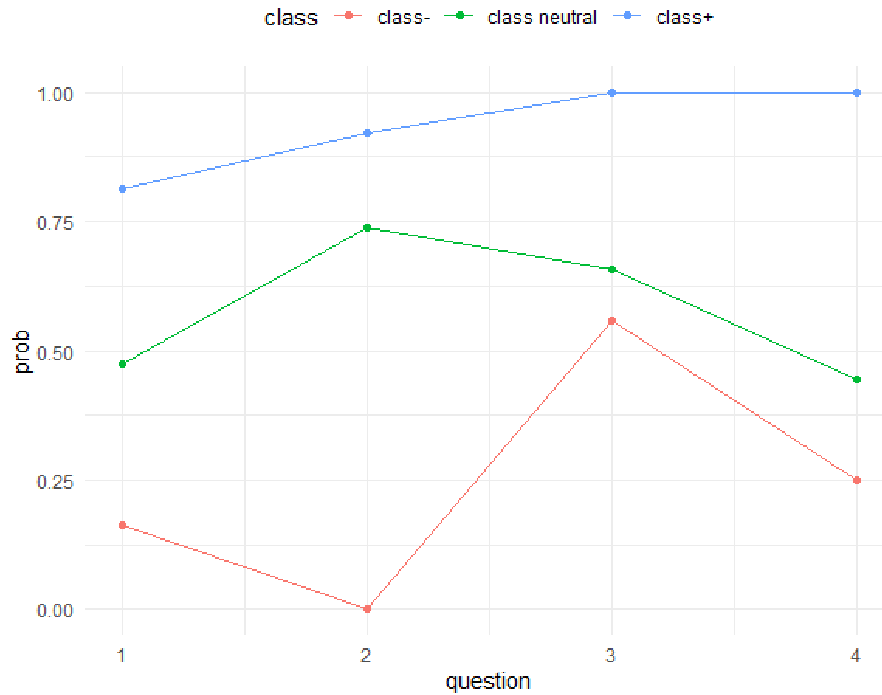

Figure 5: Probability of answering correctly to each question

```
number of estimated parameters: 14
residual degrees of freedom: 1
maximum log−likelihood: −329.3987

AIC(3): 686.7973
BIC(3): 728.1789
G^2(3): 4.235673 (Likelihood ratio/deviance statistic)
X^2(3): 3.993942 (Chi−square goodness of fit)
```

It would be also informative to see now how the probabilities of correctness changed inside each population: as done before, the proportion of correct answers for each question inside each group is presented in figure 5:

As expected, the neutral students have a probability of around 0.5 to answer correctly, while the other two classes are similarly distributed as in the $k = 2$ model. Regarding the proportions of profiles (or individuals) inside each population, the exact value is the following:

```
> sum(lc_neutral$predclass == 2)/142
[1] 0.443662
```

```
> sum(lc_neutral$predclass == 3)/142
[1] 0.1338028

> sum(lc_neutral$predclass == 1)/142
[1] 0.4225352
```

where the first number refers to the proportions of individuals assigned to the + population, the second refers to the neutral population and the last one refers to the - population.

3 Hard estimation for two clusters

3.1 The likelihood function for binary variables

Multivariate binary data (or, more generally, categorical) naturally originate from polls, questionnaires, online automated interviews which represent the profiles of n statistical units (subjects, respondents) responding to r questions. Usually, data are collected to investigate if there is a sensible separation into two groups, when the group labels (memberships) are not known. In the next pages I will indicate the two groups with + and -, as done before.

For each unit $j \in 1 : N$, there will be a multivariate $\{0, 1\}$ vector of r dimensions corresponding to the r binary answers, each denoted as 0 or 1. For each individual j, the string of the answers will be written as follows:

$$x^j := (x_1^j, ..., x_r^j)$$

while each answer to a binary question X_k may be modeled as a Bernoulli with parameter p_k, in short:

$$X_k \sim be(p_k)$$

We assume a-priori the local independence of the different questions (a strong but useful hypothesis that may be discarded using a Bayesian network approach, that will be discussed afterwards). The density function of one single unit may then be written as follows:

$$\begin{aligned} f_X(x;p) :&= P(X = x) = \\ &= P(X_1 = x_1, ..., X_r = x_r) = \\ (by \quad local \quad independence) &= \prod_{k=1}^{r} P(X_k = x_k) = \\ (by \quad def \quad X_k) &= \prod_{k=1}^{r} \left[p_k^{x_k} (1 - p_k)^{1-x_k} \right] \end{aligned}$$

Supposing then to have observed more than one unit, i.e. supposing to have observed a sample of N units $\{x^1, ..., x^N\}$, it is possible to write the Likelihood function, taking into account the previous definition of the density function. As it is common practice for random samples, the hypothesis of independence among the observations may be introduced:

$$\begin{aligned}
\mathcal{L}(p; x^1, ..., x^N) &= \\
(by \quad units \quad independence) &= \prod_{j=1}^{N} f_X(x^j; p) = \\
(by \quad local \quad independence) &= \prod_{j=1}^{N} \prod_{k=1}^{r} \left[p_k^{x_k^j} (1 - p_k)^{1 - x_k^j} \right]
\end{aligned}$$

Since the Likelihood function has to be maximised, it may be useful to consider its logarithm:

$$l(p; x^1, ..., x^N) := log\left(\mathcal{L}(p; x^1, ..., x^N)\right)$$

It is a monotone transformation that does not change its maximum points, but it transforms products into sums, which helps the computations:

$$l(p; x^1, ..., x^N) := \sum_{j=1}^{N} \sum_{k=1}^{r} \left[x_k^j log(p_k) + (1 - x_k^j) log(1 - p_k) \right]$$

From a computational point of view, it is interesting to notice that, for the binary case, into the double sum, only one of the two members $log(p_k)$ or $log(1-p_k)$ will be added, since x_k^j can be only 0 or 1; then it will be sufficient only to evaluate $\hat{p_k}$ as the MLE of the Bernoulli distribution, i.e.:

$$\hat{p_k} = \frac{\sum_{j=1}^{N} x^j}{N}$$

Since by definition $\hat{p_k} := P(X_k = 1)$.

3.1.1 Likelihood for two populations

Up to now, only the case in which all the units come from the same distribution has been considered. But in the clustering problem, we suppose that the units come from two different distribution, i.e. the units are grouped into two clusters, that will be called Cl_+ and Cl_-, respectively containing N_+ and N_- units: (indicating y and z the units respectively in the + and - cluster)

$$Cl_+ := \{y^1, ..., y^{N_+}\}$$
$$Cl_- := \{z^1, ..., z^{N_-}\}$$

Using the previous likelihood formula, it is possible to introduce one more hypothesis about the independence of the units between the clusters, so that the total likelihood will be the product of the likelihood inside each cluster:

$$\mathcal{L}(p^+, p^-; y^1, ..., y^{N_+}, z^1, ..., z^{N_-}) = \prod_{j=1}^{N_+} f_Y(y^j; p^+) \prod_{j=1}^{N_-} f_Z(z^j; p^-)$$

Applying the logarithm, the previous formula becomes then the sum of the log-Likelihood inside each cluster, making use of the previous evaluation of the density function, as follows:

$$\begin{aligned} l(p^+,p^-; y^1, ..., y^{N_+}, z^1, ..., z^{N_-}) = \\ = \sum_{j=1}^{N_+} \sum_{k=1}^{r} \left[y_k^j log(p_k^+) + (1 - y_k^j) log(1 - p_k^+) \right] + \\ + \sum_{j=1}^{N_-} \sum_{k=1}^{r} \left[z_k^j log(p_k^-) + (1 - z_k^j) log(1 - p_k^-) \right] \end{aligned}$$

Where p^+ and p^- are the parameters of the Bernoulli distribution, estimated with $\hat{p}^+$ and $\hat{p}^-$ respectively inside the + and - cluster.

3.2 Implementation of the binary hard assignment in R

We wanted to implement the hard assignment and maximum likelihood approach in R and study its limits: we start generating a random dataset to see how far the complete hard estimation can go; using $r = 4$ as number of questions/answers and 250 individuals, with the following R lines the dataset is built:

```
# random data generation
r <- 4
data <- matrix(rbinom(1000, 1, .6), 250, r, byrow = T)
head(data)

> head(data)
     [,1] [,2] [,3] [,4]
[1,]    1    1    0    0
[2,]    0    1    1    1
[3,]    1    1    1    0
[4,]    1    1    1    1
[5,]    1    1    1    0
[6,]    1    1    1    1
```

Then, each row in the data matrix represents an observation (i.e.: 4 binary answers) from a single individual. It is clear that, since 250 individuals are observed, some repetitions of the rows will be present in the data matrix, then it may be useful to introduce "profiles" observations with their corresponding frequency instead of the individual data.

The following function takes as input the individual data and converts to profiles observations combined with their frequency:

```
# to convert individual data to profiles
dataToProfiles <- function(data){
  # obtaining profiles
  # pipeline command %>%:
  #      data is the input of the following line
  data %>%
    unique(margin = 1) -> profiles

  # cbinding the frequencies
  profiles <- cbind(profiles, profiles %>%
    apply(1, function(y) apply(data,1,function(x) all(x==y))) %>%
    colSums() -> frequencies)
  return(profiles)
}
```

Then the previous function is applied to the individual data matrix, and m = profiles are retrieved:

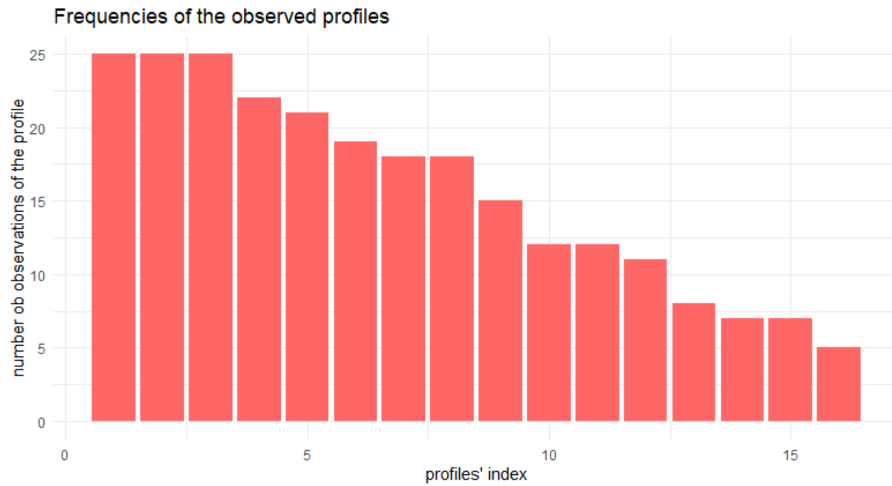

Figure 6: observed frequencies for the 16 observed profiles

```
# converting to profiles
profiles <- dataToProfiles(data)
m <- nrow(profiles)
```

In total, 16 profiles have been found (which is 2^4, i.e. all the possible patterns having 4 questions), with the corresponding frequencies reported in Fig 6:

Since the purpose is to assign every profile (and consequently each individual) to a cluster, all the possible assignments are considered in the following matrix:

```
# m * 2^m assignment matrix evaluation
tmp <- split.data.frame(cbind(rep(0, m), rep(1, m)),rep(1:m))
assignments <- matrix(t(expand.grid(tmp)), nrow=m, ncol=2^m)

# how does it look like: (transpose of the first 8 profiles)
t(assignments[1:8,1:5])

> t(assignments[1:8,1:5])
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    0    0    0    0    0    0    0    0
[2,]    1    0    0    0    0    0    0    0
[3,]    0    1    0    0    0    0    0    0
[4,]    1    1    0    0    0    0    0    0
[5,]    0    0    1    0    0    0    0    0
```

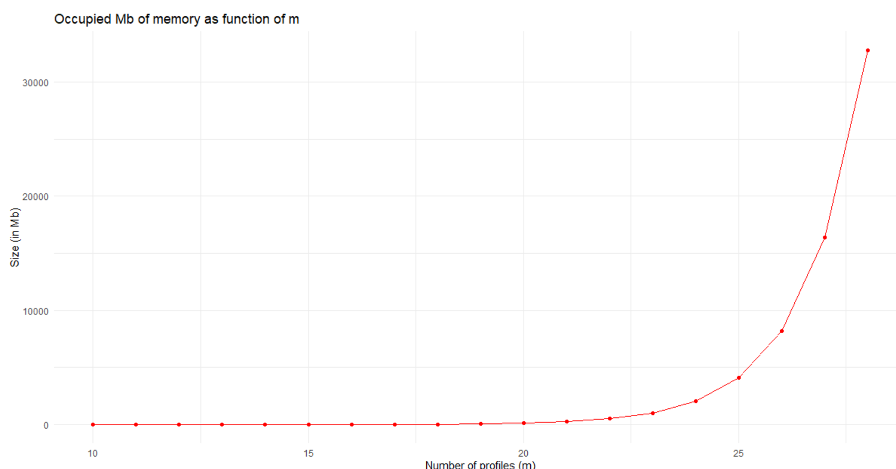

Figure 7: Value of the allocated memory for the assignment matrix as a function of m

We transposed the assignment matrix only for graphical reasons. This means that for each column of the assignments matrix (and therefore each row of the transpose), a configuration of the profiles assigned to the + or - cluster (0 and 1 in this case) is considered. Even if only 16 profiles are present, the dimension of the assignments matrix is already not small, and grows exponentially as the number of profiles increases. In detail, for 16 profiles (the assignments matrix only depends on m) the following memory is allocated:

```
# dimensions of assignments
format(object.size(assignments), units = "Mb")

> format(object.size(assignments), units = "Mb")
[1] "8 Mb"
```

The allocated space for m profiles will follow an exponential function, since each time a profile is added, the allocated space doubles. Moreover, since for $m = 16$ the allocated memory is approximately 8 Mb, the exponential function must be the following:

$$allocated \quad space \quad \approx 2^{m-13} Mb$$

Then the problem becomes easily unsolvable even for few profiles (see fig 7); if we suppose for instance $m = 30$, then 128 gigabites would be allocated only for the matrix of the assignments. For this reason some heuristic methods will be explained, since it is not feasible to explore the entire space of possible configurations.

Therefore a method is needed to compute the log-likelihood for a certain configuration, such that it would be applied to all possible configurations when it is feasible; otherwise the same function

would be used in a heuristic method build over the space of possible configurations. The following function evaluates, for a certain configuration *cl* and for a profiles matrix, the log-likelihood:

```
# for binary data and 2 cluster:
# function to evaluate the log-likelihood Laplace smoothed
# of a certain cluster configuration cl
likelihoodEval01 <- function(cl, profiles){
  # number of questions (columns - 1) in profiles
  r = ncol(profiles) - 1

  # cl is a logical vector
  # 1 if the element is in the + cluster
  # 0 if the element is in the - cluster

  # finding the number of elements in the + cluster
  den_p = sum(profiles[cl, r + 1])

  # finding the number of elements in the - cluster
  den_m = sum(profiles[!cl, r + 1])

  # in case the C+ cluster is composed by one element
  if ((length(cl[cl == TRUE]) > 1) || (length(cl[cl == TRUE]) == 0)) {
    num_p = colSums(profiles[cl, 1:r]*profiles[cl, r+1])
  } else {
    num_p = profiles[cl, 1:r]*profiles[cl, r+1]
  }

  # in case the C- cluster is composed by one element
  if ((length(cl[cl == FALSE]) > 1) || (length(cl[cl == FALSE]) == 0)) {
    num_m = colSums(profiles[!cl, 1:r]*profiles[!cl, r+1])
  } else {
    num_m = profiles[!cl, 1:r]*profiles[!cl, r+1]
  }

  # evaluating probabilities in C+ and C- using Laplace Smoothing (1, 2)
  p_p = (num_p + 1)/(den_p + 2)
  p_m = (num_m + 1)/(den_m + 2)

  # C+ total log-likelihood
  CpL = sum((profiles[cl,1:r]*profiles[cl,r+1])%*%log(p_p) +
             ((1 - profiles[cl,1:r])*profiles[cl,r+1])%*%log(1-p_p))

  # C- total log-likelihood
  CmL = sum((profiles[!cl,1:r]*profiles[!cl,r+1])%*%log(p_m) +
             ((1 - profiles[!cl,1:r])*profiles[!cl,r+1])%*%log(1-p_m))

  # total likelihood
  L = CpL + CmL
  return(L)
}
```

The reported function evaluates at first the number of questions answered in each profile, and the result is stored in the r variable as the number of columns - 1 since it has been added a column (the r + 1 one) containing the frequencies of each profile. Since the variable cl is a logical variable, it is possible to slice the dataframe containing the profiles and evaluate the total number of observations (i.e. the sum of the frequencies) in each class, and this is done evaluating den_p and den_m, where the name "den" stands for denominator.

As an example, supposing that we are evaluating the function for the 100th configuration contained in the assignments matrix, let *cl* be then the following:

```
# loglikeEval for a certain cl
cl <- as.logical(assignments[,100])
as.numeric(cl)

> as.numeric(cl)
 [1] 1 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0
```

which means that the first two profiles and the 6th and the 7th are assigned to the + group, while the remainings are assigned to the - group. In detail, we have the following profiles in the + group (the fifth column represents the frequencies of the observed pattern):

```
> profiles[cl,]
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    1    0    1   25
[2,]    1    0    0    1   20
[3,]    1    0    1    0   13
[4,]    1    1    0    0   17
```

While in the - the following are contained, for this certain configuration:

```
> profiles[!cl,]
      [,1] [,2] [,3] [,4] [,5]
 [1,]    1    1    1    0   17
 [2,]    1    1    1    1   26
 [3,]    0    1    1    1   29
 [4,]    0    1    0    1   13
 [5,]    1    0    1    1   20
 [6,]    0    1    1    0   14
 [7,]    0    1    0    0    9
 [8,]    0    0    0    0    8
 [9,]    0    0    1    1   13
[10,]    1    0    0    0   10
[11,]    0    0    0    1    7
[12,]    0    0    1    0    9
```

Threfore, as said, the den_p and den_m variables will represent the number of observations in each cluster, in this particular case they will assume the following values:

```
> den_p
[1] 75
> den_m
[1] 175
```

Then the "numerator" has to be evaluated in order to compute the likelihood of both clusters with the variables num_p and num_m in the following way:

```
num_p = colSums(profiles[cl, 1:r]*profiles[cl, r+1])
num_m = colSums(profiles[!cl, 1:r]*profiles[!cl, r+1])

> num_p
[1] 75 42 13 45
> num_m
[1]  73 108 128 108
```

The two variables represent the total number of ones appearing in each cluster for each question (this is the reason why they are r-dimensional vectors): for example the individuals answered 1 for 75 times for the first question, inside the first cluster.

Having all those numbers is now possible to evaluate the probability of answering 1 to each question, applying the division between num_p and den_p (and viceversa, num_m and den_m); also a Laplace smoothing has been applied, adding one to the nominator and 2 to the denominator (as detailed explained in section "Laplace smoothing"), and the result is then saved in the following two vectors:

```
p_p = (num_p + 1)/(den_p + 2)
p_m = (num_m + 1)/(den_m + 2)

> p_p
[1] 0.9870130 0.5584416 0.1818182 0.5974026
> p_m
[1] 0.4180791 0.6158192 0.7288136 0.6158192
```

The probability of answering 0 to each question is automatically obtained from those values, since the two possible options for the answers are 0 and 1, as $1 - p$, where p is the probability of answering one. It is now possible to evaluate the log-likelihood inside each cluster using the following formula described in the previous section:

$$l(p^+,p^-;y^1,...,y^{N_+},z^1,...,z^{N-}) =$$
$$= \sum_{j=1}^{N_+}\sum_{k=1}^{r}\left[y_k^j log(p_k^+) + (1-y_k^j) log(1-p_k^+)\right] +$$
$$+ \sum_{j=1}^{N_-}\sum_{k=1}^{r}\left[z_k^j log(p_k^-) + (1-z_k^j) log(1-p_k^-)\right]$$

In detail, the first and the second members of the sum (referring to the + and - cluster) will be evaluated as follows:

```
# C+ total log-likelihood
CpL = sum((profiles[cl,1:r]*profiles[cl,r+1])%*%log(p_p) +
          ((1 - profiles[cl,1:r])*profiles[cl,r+1])%*%log(1-p_p))

# C- total log-likelihood
CmL = sum((profiles[!cl,1:r]*profiles[!cl,r+1])%*%log(p_m) +
          ((1 - profiles[!cl,1:r])*profiles[!cl,r+1])%*%log(1-p_m))
```

Considering the CpL variable as an example, the term

```
(profiles[cl,1:r]*profiles[cl,r+1])*log(p\textunderscore p)
```

refers to the quantity

$$\sum_{j=1}^{N} y_k^j log(p_k^+)$$

in the previous formula, evaluated for each question k. The results for this configuration cl is (each row refers to a question):

```
> (profiles[cl,1:r]*profiles[cl,r+1])%*%log(p_p)
          [,1]
[1,] -27.77104
[2,] -10.56472
[3,] -22.33166
[4,] -10.12652
```

Viceversa, also the probability of answering 0 for each question is taken into account:

```
> ((1 - profiles[cl,1:r])*profiles[cl,r+1])%*%log(1-p_p)
              [,1]
[1,]    -5.016767
[2,]   -20.362312
[3,]   -22.454420
[4,]   -18.878312
```

To obtain the total log-likelihood inside the + cluster, the previous quantities are summed up obtaining a vector of 4 elements, whose sum is the following quantity:

$$\sum_{j=1}^{N_+}\sum_{k=1}^{r}\left[y_k^j log(p_k^+) + (1-y_k^j) log(1-p_k^+)\right]$$

The same steps are also done for the - cluster and the result is stored in CmL. Due to the previous hypotheses of independence, the total log-likelihood of the entire configuration can be evaluated as the sum of the two log-likelihoods inside each cluster, then the function *logLikeEval01* returns the following:

```
# total likelihood
L = CpL + CmL
return(L)
```

which is, for the chosen cl configuration:

```
> likelihoodEval01(cl, profiles)
[1] -591.1238
```

And this is one of the possible 65536 configurations contained in the assignments matrix. Using the *apply* function contained in R, it is possible to vectorize the evaluation of each log-likelihood among the columns of the assignments matrix, storing the resulting vector in the following *results* variable:

```
results <- apply(assignments, 2,
    function(cl) likelihoodEval01(as.logical(cl), profiles))
```

With respect to *results* object, in the figure 8 it is possible to see the sorted log-likelihood obtained for each configuration.

It is now possible to find the best and the worst configurations, selecting the ones with maximum and minimum likelihood with the following filters:

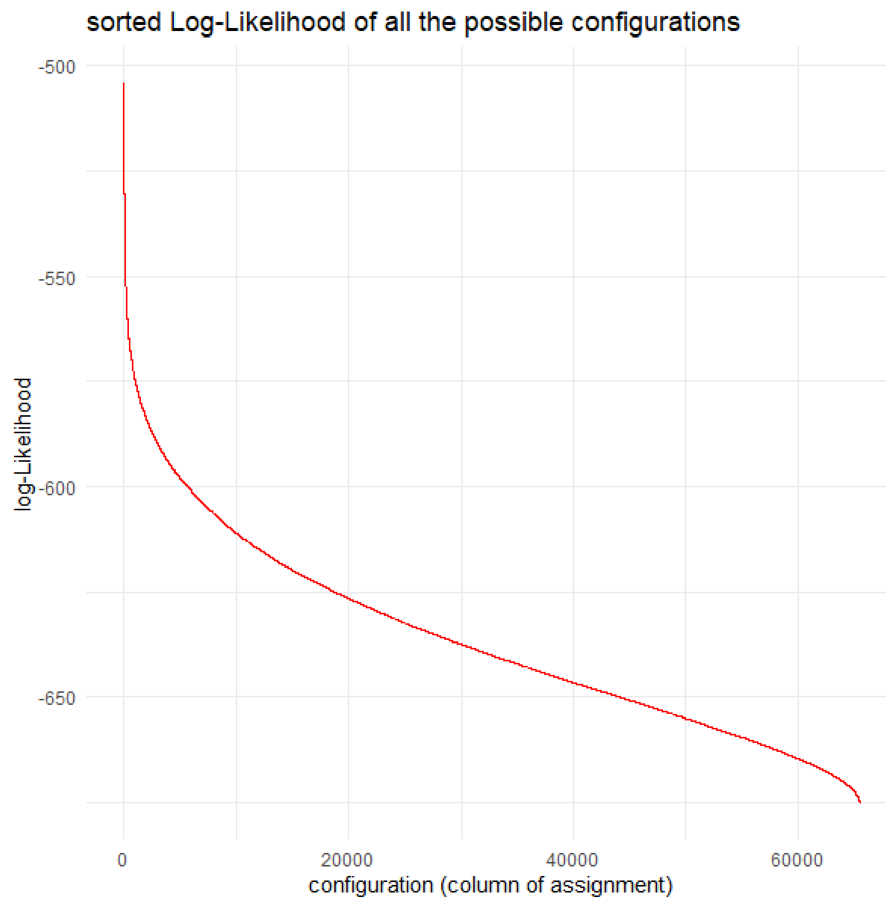

Figure 8: Value stored in *results*, sorted from the maximum to the minimum

```
# best configuration
as.data.frame(cbind(t(assignments), results)) %>%
  filter(results == max(results))

 V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16   results
  1  1  0  0  0  0  1  1  0   0   1   1   0   1   1   0 -504.1661
  0  0  1  1  1  1  0  0  1   1   0   0   1   0   0   1 -504.1661
```

It makes sense that the results are symmetrical, starting from the fact that there is no formal difference between the cluster + and -. Therefore it has been obtained that the maximum likelihood configuration has a logarithmic likelihood of -504.1661.

```
# worst configuration
as.data.frame(cbind(t(assignments), results)) %>%
  filter(results == min(results))

 V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16   results
  0  0  0  0  0  0  0  0  0   0   0   0   0   0   0   0 -675.4791
  1  1  1  1  1  1  1  1  1   1   1   1   1   1   1   1 -675.4791
```

This last result suggests that the configurations in which each observation is contained in the same cluster are the less explicative; in other words, it is meaningful to consider the profiles as separated in two populations.

3.2.1 Parallelized version and performance

Using the package *parallel* provided in R, it is possible to implement the same coded in a parallelised way. The code will be run on a single computer, then the assignment matrix will be split into N parts, where N is the number of cores of the processor. It is possible to use the library and to detect the number of cores in the following way:

```
library(parallel)
df <- profiles
numCores <- detectCores()
numCores

    > numCores
    [1] 4

clusters <- makeCluster(numCores)
clusters

    > clusters
    socket cluster with 4 nodes on host      localhost
```

The makeCluster() function creates a set of copies of R running in parallel and communicating over sockets inside the same machine, so that the code can be distributed. Another way of distributing the code is available only on Unix machines using forking techniques: it allows to parallelize an apply() function only changing apply() with the parlapply() function provided by the package. We have implemented the code only on a Windows machine.

Since each of the 4 Clusters runs independently of each other and represents a new session of R, it is necessary to load all the needed packages inside each running sessions: (in this case *dplyr* and *magrittr* have to exported since the pipeline method has been used previously in the evaluation of the log-likelihood)

```
clusterEvalQ(clusters, {library(dplyr); library(magrittr)})

> clusterEvalQ(clusters, {library(dplyr); library(magrittr)})
[[1]]
[1] "magrittr"  "dplyr"      "stats"
    "graphics"  "grDevices" "utils"
    "datasets"  "methods"    "base"

[[2]]
[1] "magrittr"  "dplyr"      "stats"
    "graphics"  "grDevices" "utils"
    "datasets"  "methods"    "base"

[[3]]
[1] "magrittr"  "dplyr"      "stats"
    "graphics"  "grDevices" "utils"
    "datasets"  "methods"    "base"
```

```
[[4]]
[1] "magrittr"   "dplyr"      "stats"
    "graphics"   "grDevices"  "utils"
    "datasets"   "methods"    "base"
```

In order to use the parallelization, it is necessary to split the matrix of assignment into smaller ones, so that each cluster will manage to find independently the solution inside each sub-matrix, then the result will be deduced from the 4 maxima found by each core. As an example to explain how parallel computing works on our code, a new bigger random profiles matrix is created with the following commands:

```
# random profiles generation

# in order to have always the same random matrices
set.seed(7)

# number of questions
r <- 5

# random units
data <- matrix(rbinom(1000, 1, .6), 200, r, byrow = T)
head(data)

> head(data)
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    1    1    1    1
[2,]    0    1    0    1    1
[3,]    1    1    0    1    1
[4,]    1    1    1    0    1
[5,]    0    1    0    0    0
[6,]    1    0    1    0    1

# conversion to profiles
profiles <- dataToProfiles(data)
profiles <- profiles[1:17,]

# total number of profiles
m <- nrow(profiles)
m
> m
[1] 17
```

In this case, the frequencies of each profile and the number itself of profiles are changed and can be seen from the (previously used) frequency graph in the figure 9.

The generation of the assignment matrix is done as always, and this time its size will be doubled since 17 profiles are observed:

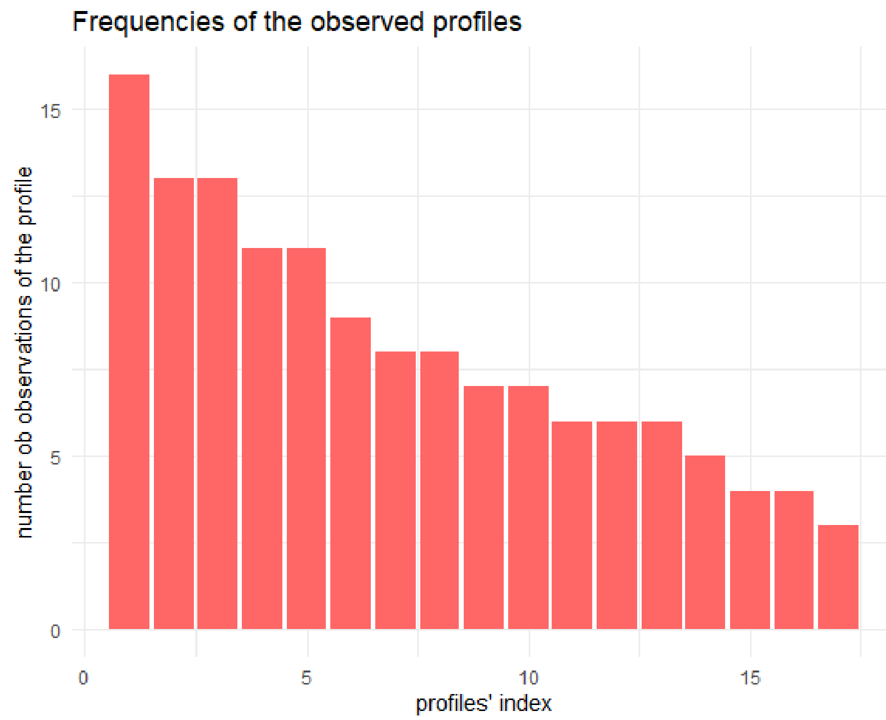

Figure 9: Value stored in $results$, sorted from the maximum to the minimum

```
# assignment matrix generation
df <- profiles
tmp <- split.data.frame(cbind(rep(0, m), rep(1, m)),rep(1:m))
assignments <- matrix(t(expand.grid(tmp)), nrow=m, ncol=2^m)
```

and with the following function it is possible to split the assignment matrix into a list containing N (4 in this case) sub-matrices:

```
# returns a list of N submatrices of mat
splitMat <- function(mat, N){

  # initialising the list of sub-matrices with the first element
  submats = list(mat[,1:round(dim(mat)[2]/N, 0)])

  # appending the remaining matrices
  for (k in 1:(N - 2)) {
    submats = append(submats,
      list(
        mat[,(k*round(dim(mat)[2]/N, 0)):((k + 1)*round(dim(mat)[2]/N, 0))]))
  }

  # appending the last one
  submats = append(submats,
      list(mat[,((N - 1)*round(dim(mat)[2]/N, 0)):dim(mat)[2]]))

  # output
  return(submats)
}
```

The results will be stored inside the following list that will be passed as an argument in parallel to each core:

```
# list containing numCores sub-matrices of assignments
submats <- splitMat(assignments, numCores)
```

Therefore, each core will deal with a smaller matrix with the following dimensions:

```
# dimensions of each sub-matrix
dim(submats[[1]])
```

```
[1]     17  32768
```

which means that each column represents a possible configuration that will give a certain log-likelihood.

Now it is necessary to add the data and the needed functions to each cluster, i.e. the profiles matrix renamed as *df* and the *likelihoodEval01* function in order to evaluate the log-likelihood of

the configurations. Also the *colLike01* function is introduced to simplify the code: it evaluates the log-likelihood of the columns of a sub-assignments matrix in a vectorized way:

```
# likelihood of a column of assignment
# returns the best cls and the relative loglike
colLike01 <- function(submat) {
  loglike <- apply(submat, 2,
    function(cl) likelihoodEval01(as.logical(cl), df))
  return(unique(as.data.frame(cbind(t(submat), loglike)) %>%
                 filter(loglike == max(loglike))))
}

# data and functions exporting
clusterExport(clusters, c("df", "colLike01", "likelihoodEval01"))
```

After having exported all the needed data and functions, with the following lines the results are evaluated, both in parallel and not:

```
# running clusters in parallel
parallelOutput <- parLapply(clusters, submats, colLike01)
parallelOutput

> parallelOutput
[[1]]
V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16 V17    loglike
 0  0  0  1  1  1  1  0  1   1   0   1   1   1   1   0   0 -310.9683

[[2]]
V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16 V17    loglike
 0  1  1  0  1  0  1  1  1   1   0   1   1   0   0   1   0 -327.30

[[3]]
V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16 V17    loglike
 1  0  0  1  0  1  0  0  0   0   1   0   0   1   1   0   1 -327.3063

[[4]]
V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16 V17    loglike
 1  1  1  0  0  0  0  1  0   0   1   0   0   0   0   1   1 -310.9683

# running not in parallel
output <- colLike01(assignments)

# comparison
Reduce("rbind", parallelOutput) %>% filter(loglike == max(loglike))
output

> Reduce("rbind", parallelOutput) %>% filter(loglike == max(loglike))
V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16 V17    loglike
```

```
  0   0   0   1   1   1   1   0   1    1    0    1    1    1    1    0    0  -310.9683
  1   1   1   0   0   0   0   1   0    0    1    0    0    0    0    1    1  -310.9683
> output
V1  V2  V3  V4  V5  V6  V7  V8  V9  V10  V11  V12  V13  V14  V15  V16  V17    loglike
  0   0   0   1   1   1   1   0   1    1    0    1    1    1    1    0    0  -310.9683
  1   1   1   0   0   0   0   1   0    0    1    0    0    0    0    1    1  -310.9683
```

As it is clear, both the implementations give the same results, symmetric as previously described, since there is no intrinsic difference a priori between the two clusters. About the timing both methods require, the solution is launched for each method, obtaining a speedup of more than 2x:

```
# parallel version
start_time = Sys.time()
parallelOutput <- parLapply(clusters, submats, colLike01)
end_time = Sys.time()
end_time - start_time

Time difference of 5.212734 secs

# non parallel version
start_time = Sys.time()
output <- colLike01(assignments)
end_time = Sys.time()
end_time - start_time

Time difference of 12.26025 secs
```

Therefore, parallelizing the code, there is an important improvement in our code; however the memory limit still holds: indeed, even using a parallelized approach, there is still the need to store the assignment matrix in a data structure, and, since it has an exponential complexity, the limit of about 30 profiles still persists.

3.3 Laplace smoothing

It would make sense to wonder why in the Laplace smoothing formula (used to estimate the probabilities of correct answers) the following parameters have been used:

$$p_k^+ = \frac{\sum_{j=1}^{N_+}(x_j) + 1}{N_+ + 2}$$

where k is the answer from $1:r$. In general the +1 and the +2 values have been added inside the fraction so that to avoid the case of having zeros in the denominator [1] [7]. From a probabilistic perspective, adding 1 in the numerator is equivalent to add a "correct" answer, and the 2 in the denominator is equivalent to add a "wrong" answer in addition to the previous one. But, in general, it is possible to define a more general Laplace smoothing, as follows (with a and $b \in R$):

$$p_k^+ = \frac{\sum_{j=1}^{N_+}(x_j) + a}{N_+ + b}$$

Since those parameters have been introduced just to tackle the numerical issue of zeros, nothing would change in the best configuration found, apart from the log-likelihood function which would be of course slightly different. For the last analysed dataset, as expected, nothing changes in the optimal configuration found, comparing different choice of parameters for the smoothing:

```
# default version a = 1, b = 2
start_time = Sys.time()
output_12 <- colLike01(assignments)
end_time = Sys.time()
end_time - start_time

# version a = 2, b = 4
start_time = Sys.time()
output_24 <- colLike01Laplace(assignments, a = 2, b = 4)
end_time = Sys.time()
end_time - start_time

# version a = exp(-10), b = 2*exp(-10)
start_time = Sys.time()
output_ee <- colLike01Laplace(assignments,
    a = exp(-10), b = 2*exp(-10))
end_time = Sys.time()
end_time - start_time

# outputs
output_12
output_24
output_ee
```

```
> output_12
V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16 V17    loglike
 0  0  0  1  1  1  1  0  1   1   0   1   1   1   1   0   0 -310.9683
 1  1  1  0  0  0  0  1  0   0   1   0   0   0   0   1   1 -310.9683
> output_24
V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16 V17    loglike
 0  0  0  1  1  1  1  0  1   1   0   1   1   1   1   0   0 -313.8481
 1  1  1  0  0  0  0  1  0   0   1   0   0   0   0   1   1 -313.8481
> output_ee
V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16 V17    loglike
 0  0  0  1  1  1  1  0  1   1   0   1   1   1   1   0   0 -308.0082
 1  1  1  0  0  0  0  1  0   0   1   0   0   0   0   1   1 -308.0082
```

3.4 Heuristic methods

Since the assignments matrix grows exponentially as a function of the number of profiles, it would not be feasible to evaluate the results for datasets that contain more than a very small number of profiles; it is then necessary to introduce some heuristic methods to find out the best solution inside a subset of the possible configurations, instead of generating a matrix of all the possible ones. The first way to do it is to implement a basic Montecarlo method, simulating totally random configurations in the space of $\{0,1\}^m$ where m is the number of profiles. The matrix *SIM* will contain nSim of those possible configurations: an example is presented using 10 possible configurations (using the same 17 profiles as before)

```
# generating SIM matrix
nSim <- 10
set.seed(5)
SIM <- matrix(sample(0:1, nSim*m, replace = TRUE),
     nrow = m, ncol = nSim)

> SIM
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]    1    1    1    1    0    0    1    1    1     1
 [2,]    0    1    1    1    1    1    0    1    0     0
 [3,]    0    1    1    1    1    0    1    1    0     0
 [4,]    0    0    0    1    1    1    0    1    0     0
 [5,]    0    1    0    1    1    0    0    0    0     1
 [6,]    0    0    0    0    0    0    1    0    1     0
 [7,]    0    0    0    0    1    1    0    1    0     1
 [8,]    0    1    1    1    1    0    1    1    1     1
 [9,]    1    1    1    1    1    1    0    0    0     1
[10,]    0    0    1    0    1    0    1    0    1     1
[11,]    0    1    1    0    0    0    0    1    0     0
[12,]    0    0    1    0    1    1    1    0    1     1
[13,]    0    1    0    1    0    0    0    0    1     1
[14,]    1    1    1    0    0    1    1    0    0     0
[15,]    1    0    1    0    0    0    1    1    0     0
[16,]    0    0    1    0    1    1    1    0    0     0
[17,]    0    1    1    1    1    1    0    0    0     1
```

Therefore, for each column there is a possible configuration of the profiles in the clusters 0 and 1 (which are as usual + and - populations). For this SIM matrix, the obtained best likelihood is the following, using the usual function *colLike01*:

```
# best configuration
colLike01(SIM)

> colLike01(SIM)
V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16 V17   loglike
 1  1  1  1  0  0  1  1  0   0   1   0   0   0   1   0   0 -362.9615
```

which is still far away from the best solution -310.9683 obtained from assignment, but the evaluation of SIM is immediate, so it would be meaningful to increase its size to more elements and iterate its evaluation *nIter* times in order to find better and better solutions over time: a trade-off between nIter and nSim has to be found depending on the characteristics of the computational power used. In this case, a laptop with 4 Gb of RAM will be used, and the code will run with the following parameters:

```
# parameters
set.seed(5)
nIter <- 100
nSim <- 1000

# initialization solution vector (best solution until t step)
solution <- rep(-Inf, nIter)

# initialization best solution (max likelihood found)
best_sol <- -Inf

# initialization best configuration initialization
best_conf = rep(0, m)

# iterations
for (t in 1:nIter){
  SIM <- matrix(sample(0:1, nSim*m, replace = TRUE),
    nrow = m, ncol = nSim)

  # current solution dataframe at iteration t
  current <- (colLike01(SIM)
    %>% filter(loglike == max(loglike)) %>% unique())

  # current max likelihood found at iteration t
  current_sol <- current$loglike %>% unlist() %>% unique()

  # current configuration found at iteration t
  current_conf <- current %>%
    select(-loglike) %>% head(1) %>% as.numeric()

  # if a new maximum is found
  if (current_sol > best_sol){
    best_sol <- current_sol
    best_conf <- current_conf
  }

  solution[t] <- best_sol
}

# results
best_conf
max(solution)
```

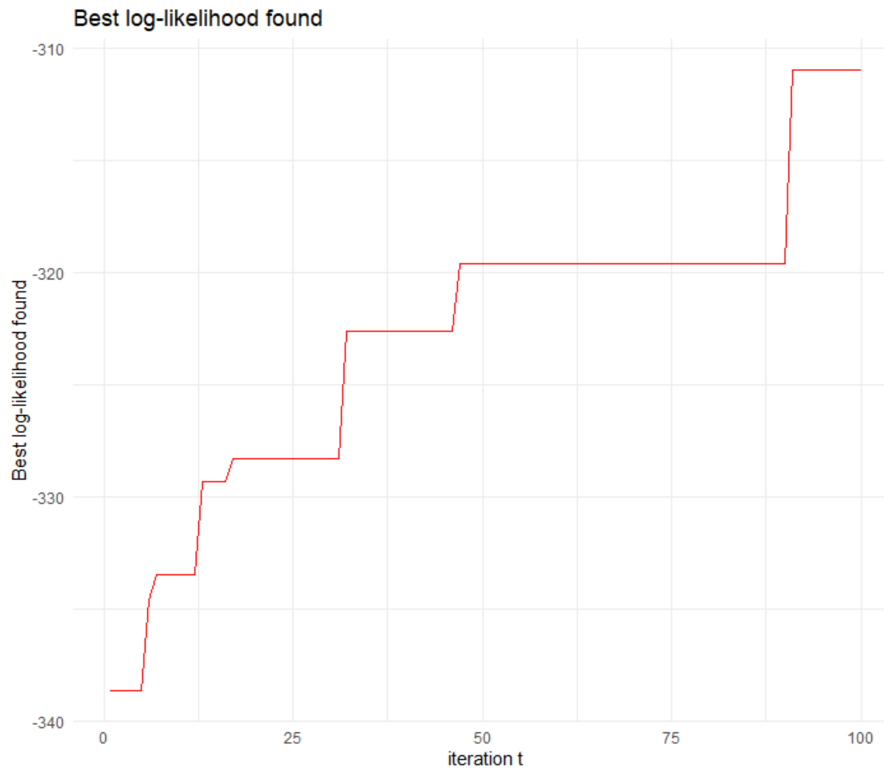

Figure 10: Best log-likelihood obtained at each iteration t

```
> best_conf
 [1] 0 0 0 1 1 1 1 0 1 1 0 1 1 1 1 0 0
> max(solution)
[1] −310.9683
```

which is exactly one of the two (symmetric) best solutions found previously. In detail, for each iteration, the best solution found is shown in Figure 10:

It is clear that the convergence depends on the randomness of the choices of the entries of the SIM matrix, and it may be possible not to reach the global maximum with this fully random approach. To overcome this problem, a simple iterative heuristic can be built in the following way (similarly to the approach developed by LI et al [6] [5]):

1. initialise clusters' configuration = cl
2. compute initial log-likelihood L_0

3. initialise best log-likelihood found $L_{best} = L_0$
4. initialise best clusters' configuration found $cl_{best} = cl$
5. iterate from 1 to nIter:
6.   select a random profile p in a cluster X
7.   move p to cluster Y
8.   compute new log-likelihood L
9.   compute new clusters' configuration cl
10.   if $L > L_{best}$:
11. $L_{best} = L$
12. $cl_{best} = cl$
13. Return cl_{best}, L_{best}

The initialization (step 1 of the previous algorithm) may be random or deterministic (all profiles inside the same cluster). It may be a wiser idea to initialise the first configuration finding a good initial guess of cl made by the random Monte Carlo algorithm previously described: Therefore, it would be sufficient to create a function using the previous code as following:

```
MC <- function(df, nIter = 100, nSim = 1000){

  # retrieving number of profiles and answers
  m <- nrow(df)
  r <- ncol(df) - 1

  # initialization solution vector (best solution until t step)
  solution <- rep(-Inf, nIter)

  # initialization best solution (max likelihood found)
  best_sol <- -Inf

  # initialization best configuration initialization
  best_conf = rep(0, m)

  # iterations
  for (t in 1:nIter){
    SIM <- matrix(sample(0:1, nSim*m, replace = TRUE),
        nrow = m, ncol = nSim)

    # current solution dataframe at iteration t
    current <- (colLike01(SIM) %>%
        filter(loglike == max(loglike)) %>% unique())
```

```
    # current max likelihood found at iteration t
    current_sol <- current$loglike %>% unlist() %>% unique()

    # current configuration found at iteration t
    current_conf <- current %>%
        select(-loglike) %>% head(1) %>% as.numeric()

    # if a new maximum is found
    if (current_sol > best_sol){
      best_sol <- current_sol
      best_conf <- current_conf
    }

    solution[t] <- best_sol
  }

  # results
  output <- list()
  output[[1]] <- best_conf
  output[[2]] <- best_sol
  return(output)
}
```

The output is now stored in a list whose first element is the best configuration found, and the second is the maximum likelihood found:

```
> MC(df)
[[1]]
 [1] 1 1 1 0 0 0 0 1 0 0 1 0 0 0 0 1 1

[[2]]
[1] -310.9683
```

Since with this profiles matrix the Monte Carlo method already easily finds the best solution, it may be meaningful to increase the size of the problem to more profiles, and compare the new heuristic, the Monte Carlo basic random method and the exact method: adding 2 profiles may be already challenging for the exact method, as the following (using the parallel version).

```
# in order to have always the same random matrices
set.seed(7)

# number of questions
r <- 5

# random units
data <- matrix(rbinom(1000, 1, .6), 200, r, byrow = T)
head(data)
```

```
# conversion to profiles
profiles <- dataToProfiles(data)
profiles <- profiles[1:20,]
m <- nrow(profiles)

# assignment matrix generation
df <- profiles
tmp <- split.data.frame(cbind(rep(0, m), rep(1, m)),rep(1:m))
assignments <- matrix(t(expand.grid(tmp)), nrow=m, ncol=2^m)

# subdivision of the assignment matrix into n (= n cores) submatrices
# creating clusters for parallel computing
numCores <- detectCores()
clusters <- makeCluster(numCores)

# list containing numCores sub-matrices of assignments
submats <- splitMat(assignments, numCores)

# exporting the libraries
clusterEvalQ(clusters, {library(dplyr); library(magrittr)})

# data and functions exporting
clusterExport(clusters, c("df", "colLike01", "likelihoodEval01"))

# running clusters in parallel - exact solution
start_time = Sys.time()
parallelOutput <- parLapply(clusters, submats, colLike01)
end_time = Sys.time()
end_time - start_time

    Time difference of 56.9426 secs

# best solution obtained by the exact method
Reduce("rbind", parallelOutput) %>%
  filter(loglike == max(loglike))-> exact_sol

> exact_sol %>% select(loglike) %>% unique()
    loglike
  -359.5729
```

The best log-likelihood found is then -359.5729 after approximately 57 seconds. Regarding the swap heuristic anticipated before, it can be implemented as follows:

```
# simple heuristic with MC initialization
swapMC <- function(df, nIter = 100, MCnSim = 100, MCnIter = 100){
  m <- nrow(df)
  r <- ncol(df) - 1

  # inizialization using Monte Carlo basic method
```

```
  cat("Initializing the first configuration using MC ...\n")
  init <- MC(df, MCnIter, MCnSim)
  cl <- init[[1]]
  L0 <- init[[2]]

  # initialization of the best configuration
  cl_best <- cl

  # initialization of the best log-Likelihood
  L_best <- L0

  # iterations
  for (it in 1:nIter){
    # swap
    cl <- cl_best
    rnd <- sample(1:m,1)
    cl[rnd] <- 1 - cl[rnd]

    # if better, mantain the swap
    if (likelihoodEval01(as.logical(cl), df) > L_best) {
      L_best <- likelihoodEval01(as.logical(cl), df)
      cl_best <- cl
    }
  }

  # results
  output <- list()
  output[[1]] <- cl_best
  output[[2]] <- L_best
  return(output)
}
```

For this particular problem of 20 profiles, the solution obtained by the exact method is easily reached in 2 seconds:

```
# swap heuristic
set.seed(8)
start_time = Sys.time()
swapMC(df)

[[1]]
 [1] 1 1 1 0 0 0 0 1 0 0 1 0 0 0 0 1 1 0 1 0

[[2]]
[1] -359.5729

end_time = Sys.time()
end_time - start_time

Time difference of 1.989992 secs
```

The result is of course highly influenced by the initial conditions, then it would be informative to run the function multiple times, with different initial seeds, in the following lines it will be run with 100 initial different seeds:

```
# how many times maximum is reached
maxima <- rep(0, 100)
for (i in 1:100) {
  cat(paste0(i, "\n"))
  set.seed(i)
  maxima[i] <- swapMC(df)[[2]]
}

# correct solutions
sum(maxima == max(maxima))

> sum(maxima == max(maxima))
[1] 77
```

More in detail, the solutions found can be described via the histogram in fig 11.

Varying the parameters of the heuristic (number of steps for the initialization for the Monte Carlo, number of configurations tested inside each Monte Carlo iteration and number of total swaps, i.e.: nIter) it would be of course possible to change the obtained results.

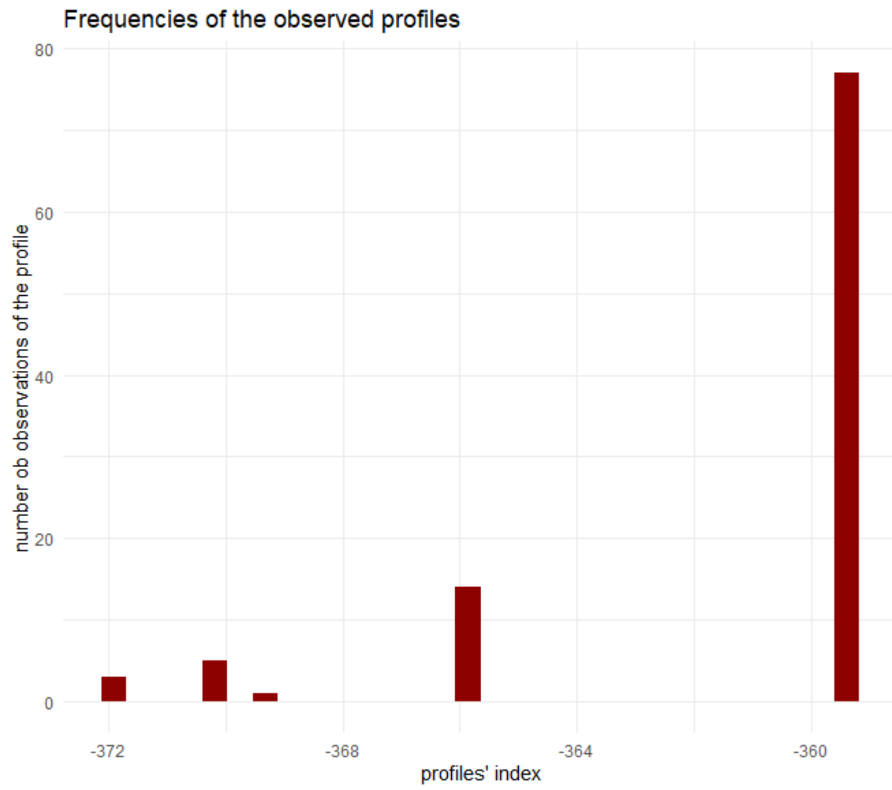

Figure 11: Distribution of the best solutions found by the swap Algorithm

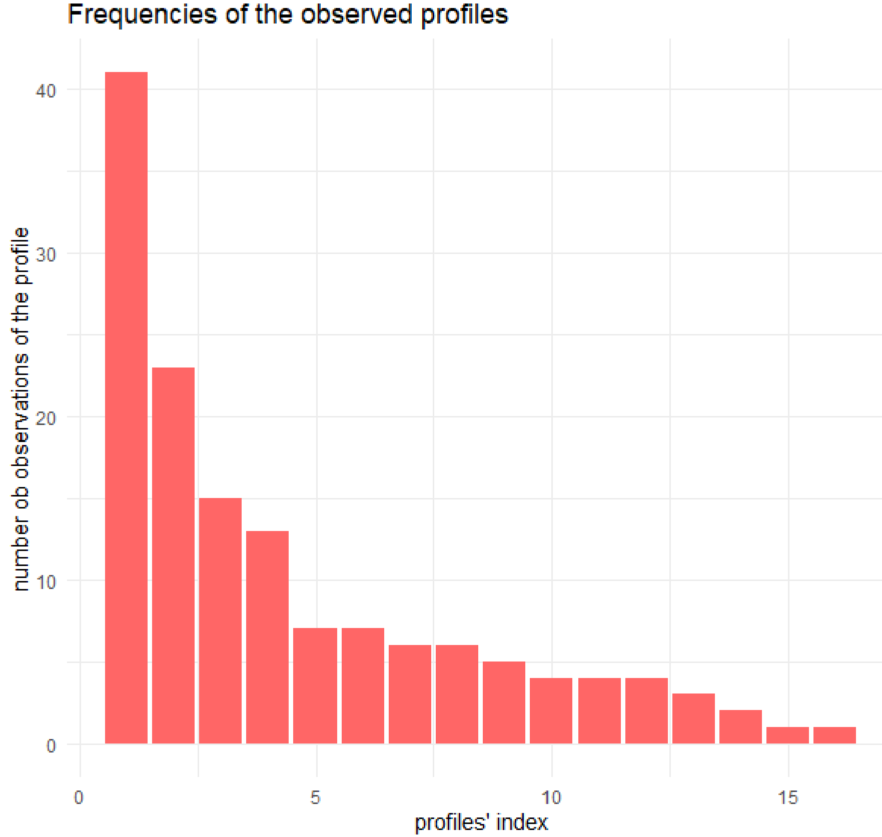

Figure 12: Frequencies of the profiles obtained from the Bartholomew dataset

3.5 Applications

At this point, it would make sense to apply the previously described algorithms to the Bartholomew dataset already introduced in the EM section in order to both evaluate the performance of hard classification by itself and the comparison between the LCA (soft classification) and the hard approach. Loading the individuals from the dataset, 16 profiles are obtained (Considering that in this dataset we have only 16 different profiles the problem could be solve even for the exact method, see fig 12):

The optimal configuration is found making use of the previously described function *colLike()* parallelized and vectorised, with the following output:

```
> exact_sol
V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16    loglike
 1  1  1  1  0  1  0  1  0   1   0   0   1   0   0   0 -240.1591
 0  0  0  0  1  0  1  0  1   0   1   1   0   1   1   1 -240.1591
```

Since the instance of the problem is relatively small, also the heuristic swap method introduced in the latter chapter converges to the optimal solution in a fast way:

```
> end_time - start_time
Time difference of 2.131315 secs
> results_swap[[1]]
 [1] 1 1 1 1 0 1 0 1 0 1 0 0 1 0 0 0
> results_swap[[2]]
[1] -240.1591
```

Profiles	**Frequency**	**Prob.** $\in P_{-}$	**Prob.** $\in P_{+}$	`poLCA`	`hard`
1111	15	0.000	1.000	+	+
1101	23	0.002	0.998	+	+
1110	7	0.002	0.998	+	+
0111	4	0.001	0.999	+	+
1011	**1**	**0.003**	**0.997**	**+**	**-**
1100	7	0.087	0.913	+	+
1001	**6**	**0.095**	**0.905**	**+**	**-**
0101	5	0.025	0.975	+	+
1010	**3**	**0.100**	**0.900**	**+**	**-**
0110	2	0.026	0.974	+	+
0011	**4**	**0.029**	**0.971**	**+**	**-**
1000	13	0.822	0.178	-	-
0100	**6**	**0.526**	**0.474**	**-**	**+**
0001	4	0.550	0.450	-	-
0010	1	0.563	0.437	-	-
0000	41	0.982	0.018	-	-

3.6 Comparison with the EM algorithm

As said, the results between the hard and soft classification are different, can differ. This happens in our example and it is summarized in the table, that contains the observed profiles, their frequency, the probabilities of belonging to the class - and + using *poLCA*, the assignment of poLCA (using 0.5) as a threshold, and the hard assignment.

While for the first 4 and the last 3 observations both the approaches give the same result, for the others there are some differences. Actually, the two models maximise two different functions: for the soft assignment, a likelihood function including latent variables has been maximised iteratively, while the hard approach directly maximises the likelihood function. Both methods rely on the local independence assumption. However, if we calculate the *log-likelihood* fo the configuration found by *poLCA* using the hard approach, we obtain:

```
> likelihoodEval01(as.logical(poLCA_sol), df)
[1] -243.9193
```

which is of course less than the global maximum found by hard:

```
> likelihoodEval01(as.logical(hard_sol), df)
[1] -240.1591
```

Of course this leads to the fact that the estimated classes contain different individuals. On the other hand the balancing of the class is quite similar:

```
# c-binding the dataframe with the solutions found
complete_df <- as.data.frame(cbind(df, hard_sol, poLCA_sol))

# number of individuals
individuals <- complete_df %>%
     select(data_Bart.frequencies) %>%
     sum()

# finding number of individuals inside cluster + for hard
complete_df %>%
  # cluster + for hard
  filter(hard_sol == 1) %>%
  select(data_Bart.frequencies) %>%
  sum() -> np_hard

# number of individuals inside cluster - for hard
nm_hard <-  individuals - np_hard

# finding number of individuals inside cluster + for soft
complete_df %>%
  # cluster + for soft
  filter(poLCA_sol == 1) %>%
  select(data_Bart.frequencies) %>%
  sum() -> np_soft

# number of individuals inside cluster - for hard
nm_soft <-  individuals - np_soft
c(np_hard, np_soft)
c(nm_hard, nm_soft)
```

```
> c(np_hard, np_soft)
[1] 69 77
> c(nm_hard, nm_soft)
[1] 73 65
```

3.6.1 Symmetric case

In case of perfect symmetry the results obtained for the Bartholomew dataset are the same using the soft and the hard approach. For perfect symmetry we mean a dataset that includes all the possible 2^r profiles, where r is the number of questions, observed with the same frequency, i.e. there is basically no distinction among the questions. Then the frequencies are all changed to the same number (5) in the Bartholomew dataset with the following R lines:

```
> data_Bart
   Observed_freq V1 V2 V3 V4
1              5  2  2  2  2
2              5  2  2  1  2
3              5  2  2  2  1
4              5  1  2  2  2
5              5  2  1  2  2
6              5  2  2  1  1
7              5  2  1  1  2
8              5  1  2  1  2
9              5  2  1  2  1
10             5  1  2  2  1
11             5  1  1  2  2
12             5  2  1  1  1
13             5  1  2  1  1
14             5  1  1  1  2
15             5  1  1  2  1
16             5  1  1  1  1
```

It is interesting to see at first that in this case of symmetry, more than one global maximum exists for hard, while before only one was found (in practice 2 were found, but one was the complementary of the other). The global optima are the following:

```
> hardResults
V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16   loglike
 2  2  2  1  2  2  2  1  2   1   1   2   1   1   1   1 -168.2831
 2  2  2  2  1  2  1  2  1   2   1   1   2   1   1   1 -168.2831
 2  2  1  2  2  1  2  2  1   1   2   1   1   2   1   1 -168.2831
 2  1  2  2  2  1  1  1  2   2   2   1   1   1   2   1 -168.2831
 1  2  1  1  1  2  2  2  1   1   1   2   2   2   1   2 -168.2831
 1  1  2  1  1  2  1  1  2   2   1   2   2   1   2   2 -168.2831
 1  1  1  1  2  1  2  1  2   1   2   2   1   2   2   2 -168.2831
 1  1  1  2  1  1  1  2  1   2   2   1   2   2   2   2 -168.2831
```

Due to complementary clusters, 4 optima are found. Interestingly, the solution found by poLCA is one of these, in particular the following:

```
> unique(cbind(dataset, lc$posterior, lc$predclass))
V1 V2 V3 V4
```

1	1	1	1	0.07096035	0.92903965	2
1	1	0	1	0.07096055	0.92903945	2
1	1	1	0	0.07096033	0.92903967	2
0	1	1	1	0.94652944	0.05347056	1
1	0	1	1	0.07096040	0.92903960	2
1	1	0	0	0.07096053	0.92903947	2
1	0	0	1	0.07096060	0.92903940	2
0	1	0	1	0.94652960	0.05347040	1
1	0	1	0	0.07096038	0.92903962	2
0	1	1	0	0.94652943	0.05347057	1
0	0	1	1	0.94652948	0.05347052	1
1	0	0	0	0.07096058	0.92903942	2
0	1	0	0	0.94652958	0.05347042	1
0	0	0	1	0.94652964	0.05347036	1
0	0	1	0	0.94652947	0.05347053	1
0	0	0	0	0.94652962	0.05347038	1

and it matches with the first configuration found by hard. Summarising the results, we obtain the following table with the last four columns that represent the four global optima (i.e. columns 1,2,3,4, in which also the poLCA solution is included):

V1	V2	V3	V4	1	2	3	4
1	1	1	1	2	2	2	2
1	1	0	1	2	2	2	1
1	1	1	0	2	2	1	2
0	1	1	1	1	2	2	2
1	0	1	1	2	1	2	2
1	1	0	0	2	2	1	1
1	0	0	1	2	1	2	1
0	1	0	1	1	2	2	1
1	0	1	0	2	1	1	2
0	1	1	0	1	2	1	2
0	0	1	1	1	1	2	2
1	0	0	0	2	1	1	1
0	1	0	0	1	2	1	1
0	0	0	1	1	1	2	1
0	0	1	0	1	1	1	2
0	0	0	0	1	1	1	1

At first sight, it seems that in each optimal configuration some questions have more importance than others; apparently this is a non-sense because in this example everything is symmetric and all questions have the same weight (and the same a priori probability to be answered in the correct/wrong way). However we found an interesting explanation: since, as said, there is no distinction between the questions (the frequency is the same for all profiles and we have all of the $2^r = 2^3 = 16$ profiles) the optimal configuration is obtained as if it would be built following two easy steps.

- Choose a question q from 1 to r
- split the profiles considering only the answers to the question q

This is exactly what happens in the previous case, for each question from the first to the fourth, each optimal configuration is built like this. This is just a conjecture, but it is still valid considering a different number of questions (for example 3) and we are confident to be able to prove it, as future development of this work.

V1	V2	V3	1	2	3
1	1	1	2	2	2
1	1	0	2	2	1
0	1	1	1	2	2
1	0	1	2	1	2
1	0	0	2	1	1
0	1	0	1	2	1
0	0	1	1	1	2
0	0	0	1	1	1

4 Hard estimation for multiple clusters

4.1 The likelihood function for dicotomous answers and more than two populations

Introducing the hypothesis of more than 2 populations, and multiple answers (still under the hypotheses of local ad global independence), the log-Likelihood function to maximise would of course change. Instead of having the probability of answering 1 (previously called p_k), each question k will have a certain number of possible categorical answers, defined as $Q(k) = \{0, 1, 2, ...\}$. Then, if x_k^j denotes the answer of the j-th individual to the k-th question, and q is a possible answer in $Q(x)$, the following probabilities are obtained to evaluate the log-likelihood:

$$p_{kq} = \frac{\sum_{j=1}^{N}(x_k^j = q)}{N}$$

where $(\cdot)$ is the indicator function. Therefore, the total log-Likelihood can be stated as follows:

$$l(p; x^1, ..., x^r) := \sum_{j=1}^{N}\sum_{k=1}^{r}\sum_{p\in Q(k)}\Big[(x_k^j = p)log(p_{kq})\Big]$$

and this quantity can be evaluated inside each configuration cl and, by the hypothesis of global independence, the total log-Likelihood is obtained as the following sum:

$$l(p; x^1, ..., x^r) := \sum_{c\in cl}\sum_{j=1}^{N(c)}\sum_{k=1}^{r}\sum_{p\in Q(k)}\Big[(x_k^j = p)log(p_{kq})\Big]$$

where $N(c)$ is the number of units in the population c in the configuration cl. Therefore, the only one thing to be changed in the evaluation of the best configuration, is the function that evaluates the likelihood, presented in the next pages.

4.2 Implementation in R using dplyr package

Since the variables of the problem are increasing (number of populations and number of different outcomes to each question), it is important to keep the code as vectorized as possible in order to avoid loops that increase the time of execution. Then, the functions *apply()*, *lapply()* and *sapply()* will be used. The following random dataset is considered:

```
set.seed(7)

# parameters initialization
k = 3 # number of clusters
r = 4 # numbers of questions

# example matrix
d1 <- sample(c(0, 1), 30, replace = TRUE)
d2 <- sample(c(0, 1), 30, replace = TRUE)
d3 <- sample(c(0, 1, 2), 30, replace = TRUE)
d4 <- sample(c(0, 1, 2, 3), 30, replace = TRUE)
d5 <- sample(c(0, 1, 2, 3), 30, replace = TRUE)
d6 <- sample(c(0, 1, 2, 3), 30, replace = TRUE)
d7 <- sample(c(0, 1, 2, 3), 30, replace = TRUE)
d8 <- sample(c(0, 1, 2, 3), 30, replace = TRUE)
freq <- sample(c(1, 2, 3, 4), 30, replace = TRUE)
df = data.frame(d1, d2, d3, d4, d5, d6, d7, d8, freq)

head(df)

> head(df)
d1 d2 d3 d4 d5 d6 d7 d8 freq
 1  1  0  1  2  0  3  0    1
 0  0  1  0  1  3  1  1    1
 0  0  2  3  2  0  1  3    4
 0  0  2  0  0  0  3  2    3
 0  0  1  2  2  1  2  3    4
 1  1  1  2  1  3  2  1    2
```

In total there are 30 profiles with 8 questions with multiple answers, then it would not be possible to evaluate each configuration since the memory allocation for the assignment matrix would be too big. Nevertheless, the approach is the same as before, i.e. considering in parallel some configurations *cl* and finding the best one. Starting with a simple Monte Carlo method it is possible to find a local optimum: keeping the same names as in the previous section, the matrix SIM will be generated but this time it will contain not only 0 and 1.

```
# SIM matrix, for each column we have a configuration cl
NSIM <- 10000
nProfiles <- dim(df)[1]
SIM <- matrix(sample(1:k, NSIM*nProfiles, replace = TRUE),
              nrow = nProfiles, ncol = NSIM)
```

```
# first configuration considered
cl <- SIM[1:30, 1]

> cl
[1] 1 3 3 2 3 3 2 2 3 2 1 3 1 2 2 1 2 1 3 1 2 2 1 2 3 3 2 1 1 2
```

The first step in order to evaluate the total log-Likelihood is to bind the configuration considered to the dataframe in order to split the entire data into smaller datasets representing each population (in this case, 3):

```
# splitting in list of dataframes
dfList <- cbind(df, cl) %>% split(cl)
dfList[[1]]

   d1 d2 d3 d4 d5 d6 d7 d8 freq cl
1   1  1  0  1  2  0  3  0    1  1
11  0  1  0  1  3  2  1  1    4  1
13  1  1  2  1  3  2  3  2    3  1
16  0  0  1  1  2  2  1  0    2  1
18  0  1  1  0  3  3  3  1    3  1
20  0  1  2  0  3  1  1  3    2  1
23  1  1  0  0  0  1  2  0    4  1
28  0  0  2  3  3  1  0  0    3  1
29  1  0  1  0  0  2  2  2    4  1

dfList[[2]]

   d1 d2 d3 d4 d5 d6 d7 d8 freq cl
4   0  0  2  0  0  0  3  2    3  2
7   0  0  1  2  3  1  2  3    2  2
8   1  1  1  1  0  3  0  2    3  2
10  0  0  1  0  0  3  1  1    2  2
14  0  0  1  1  2  2  2  2    3  2
15  0  1  0  2  3  2  0  2    3  2
17  1  0  1  3  3  1  1  2    1  2
21  1  1  2  2  2  1  2  3    2  2
22  0  1  2  1  2  3  3  3    1  2
24  1  1  2  2  2  2  1  1    2  2
27  1  0  0  2  0  0  0  3    3  2
30  0  0  1  3  3  3  1  2    3  2

dfList[[3]]

   d1 d2 d3 d4 d5 d6 d7 d8 freq cl
2   0  0  1  0  1  3  1  1    1  3
3   0  0  2  3  2  0  1  3    4  3
5   0  0  1  2  2  1  2  3    4  3
6   1  1  1  2  1  3  2  1    2  3
```

9	0	1	1	3	2	3	3	2	4	3
12	0	0	1	1	3	3	2	2	1	3
19	1	1	1	3	3	0	1	0	4	3
25	1	1	1	2	1	1	1	3	4	3
26	0	0	0	2	2	2	3	1	2	3

Since the process of the evaluation of the likelihood is the same for each dataset contained in the list of datasets, the first one can be considered as an example for the following steps, and it will be called dfTmp:

```
# test dataset
dfTmp <- dfList[[1]]
```

The *colFreq* function reported below will be useful to vectorize the process since it will be applied to each "column" of the dataset: it evaluates the Laplace smoothed ($a = 1$, $b = 2$) probability, inside each dataframe in the list, of finding x in the question d, times the frequency of the units that answer x to d:

```
# x is the value d is the column:
# e.g.:
# colFreq(2,4) =
# frequency *(log (laplace-smoothed) probability
# of finding 2 in column 4)
colFreq <- function(x, d, dfTmp) {
  # number of x in question d
  numX <- sum(as.numeric(dfTmp[, d] == x) * dfTmp$freq)

  # returns the log of probabilities times the frequency
  return(numX*log((numX + 1)/(sum(dfTmp$freq) + 2)))
}
```

I reported here the evaluation of colFreq(1,1), which is referring to the answers equal to 1 to the first question:

```
# outcome of colFreq(1, 1, dfTmp)
x <- 1
d <- 1

# number of x in question d
numX <- sum(as.numeric(dfTmp[, d] == x) * dfTmp$freq)
numX
[1] 12

# returns the log of probabilities times the frequency
numX*log((numX + 1)/(sum(dfTmp$freq) + 2))
[1] -9.207062
```

Now let's consider the same question $d = 1$ and evaluate the previous number for all the possible outcomes of the question: a priori, we don't know the possible outcomes, but they can be easily calculated with the following line:

```
# each answer in d = 1
d <- 1
unique(dfTmp[, d])
```

```
[1] 1 0
```

Then it would make sense to find the colFreq(x, 1, dfTmp), where $x \in \{0, 1\}$ (i.e. $x \in Q(d)$, as $Q()$ was introduced previously), applying the following function:

```
# applying to each possible answer in the question d
sapply(unique(dfTmp[, d]), function(x) colFreq(x, d, dfTmp))
```

```
[1] -9.207062 -8.738160
```

Since $d = 1$ has been used, the previous function should be applied to all possible question d, and then the results obtained are summed up (due to local independency) with the pipeline command:

```
# applying to each question
sapply(1:r,
    function(d) sum(sapply(unique(dfTmp[, d]),
                           function(x) colFreq(x, d, dfTmp)))) %>%
  sum()
```

```
[1] -200.7095
```

Then, the previous evaluation has to be applied to each dataset contained in the list of split datasets by *cl*, using the *apply()* function, and then, due to global independence, the results are summed up using the *Reduce("+")* function:

```
# applying to each dfTmp in dfList
Reduce("+",
 lapply(dfList, # applying to each cluster
    function(dfTmp) {
      sapply(1:r, # applying to each column
         function(d) sum(sapply(unique(dfTmp[, d]),
            function(x) colFreq(x, d, dfTmp)))) %>%
      sum()}))
```

```
[1] -619.1202
```

In this way, the likelihood for *cl* has been found, and the previous commands can be used in the function *logLikeEval(cl, df)*. Now it would make sense to proceed as before, i.e. evaluating the log-Likelihood for many *cl*s.

4.3 Heuristic methods

It would make sense to use a Monte Carlo heuristic to find a global optimum to the problem (since the sizes of it are too big to be solved exactly). As a SIM matrix has been generated previously, the best configuration found is the following (using the usual parallelization using the 4 Cores available):

```
# creating clusters for parallel computing
numCores <- detectCores()
numCores
clusters <- makeCluster(numCores)
clusterEvalQ(clusters, {library(dplyr); library(magrittr)})
clusterExport(clusters, c("df", "colLike", "logLikeEval"))

# creating list of numCores submatrices of SIM matrix
submats = list(SIM[,1:round(dim(SIM)[1]/numCores, 0)])
for (k in 1:(numCores - 2)) {
  submats = append(submats,
    list(SIM[,
    (k*round(
    dim(SIM)[2]/numCores, 0)):((k + 1)*round(dim(SIM)[2]/numCores, 0))]))
}
submats = append(submats,
    list(SIM[,
        ((numCores - 1)*round(dim(SIM)[2]/numCores, 0)):dim(SIM)[2]]))

# running clusters in parallel
parallelOutput = parLapply(clusters, submats, colLike)

# stopping clusters
stopCluster(clusters)

# finding the maximum likelihood
do.call("rbind", parallelOutput) %>%
  unique() %>%
  filter(loglike == max(loglike)) %>%
  select(loglike)

  loglike
-581.6811
```

And the best configuration can be found inside *parallelOutput*.

As done in the previous section, it would be anyway convenient to implement also a generalised swap() heuristic, but in this case an element will be swapped from a random cluster X to a random cluster Y (in the latter section, it was swapped from the cluster + to the cluster - and vice versa). The algorithm stays the same as before:

1. initialise clusters' configuration = *cl*
2. compute initial log-likelihood L_0

3. initialise best log-likelihood found $L_{best} = L_0$
4. initialise best clusters' configuration found $cl_{best} = cl$
5. iterate from 1 to nIter:
6. select a random profile p in a cluster X
7. move p to cluster Y
8. compute new log-likelihood L
9. compute new clusters' configuration cl
10. if $L > L_{best}$:
11. $L_{best} = L$
12. $cl_{best} = cl$
13. Return cl_{best}, L_{best}

Again, the initialization can be done with a Monte Carlo instead of a totally random initialization, in order to obtain already a good solution. Using the same logic described before with the MC solution, the initialization gives the following initial configuration and likelihood:

```
> L
[1] −586.8974
> cl
 [1] 2 3 2 3 1 1 1 2 3 2 3 3 2 2 1 2 2 3 2 1 1 2 1 1 1 1 1 1 1 3
```

The results have been obtained via 10 vectorized MC steps, where 100 possible configurations were considered (see the fig 13 for the performances of the heuristic).

Once initialised, the swap steps can be implemented as follows:

```
# number of swaps
nIter = 500

# MC initialisation
cl <− output[[1]]
L0 <− output[[2]]
# initialization of the best configuration
cl_best <− cl
# initialization of the best log−Likelihood
L_best <− L0
solutionSwap <− rep(0, nIter)
```

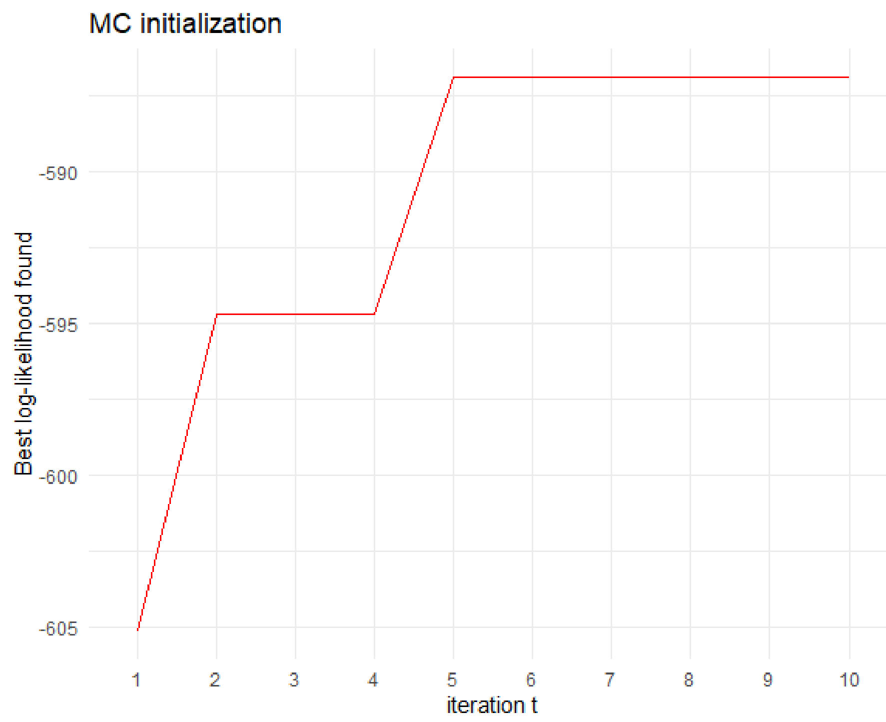

Figure 13: Best log-likelihood obtained at each iteration t

```
# iterations
for (it in 1:nIter){
  cl <- cl_best
  #    changing one random element
  rnd <- sample(1:m,1)
  cl[rnd] <- sample(c(1:k)[c(1:k) != cl[rnd]], 1)

  # if better, mantain the swap
  if (logLikeEval(cl, df) > L_best) {
    L_best <- logLikeEval(cl, df)
    cl_best <- cl
  }
  solutionSwap[it] <- L_best
}

cl_best
L_best

> cl_best
 [1] 3 2 2 2 1 1 1 3 2 2 3 3 3 3 3 3 2 2 2 2 1 2 1 3 1 3 1 2 3 2
> L_best
[1] -537.176
```

The log-Likelihood found has improved the solution found by MC, as expected, similarly to the binary case. The results for each iteration can be found in the Figure 14:

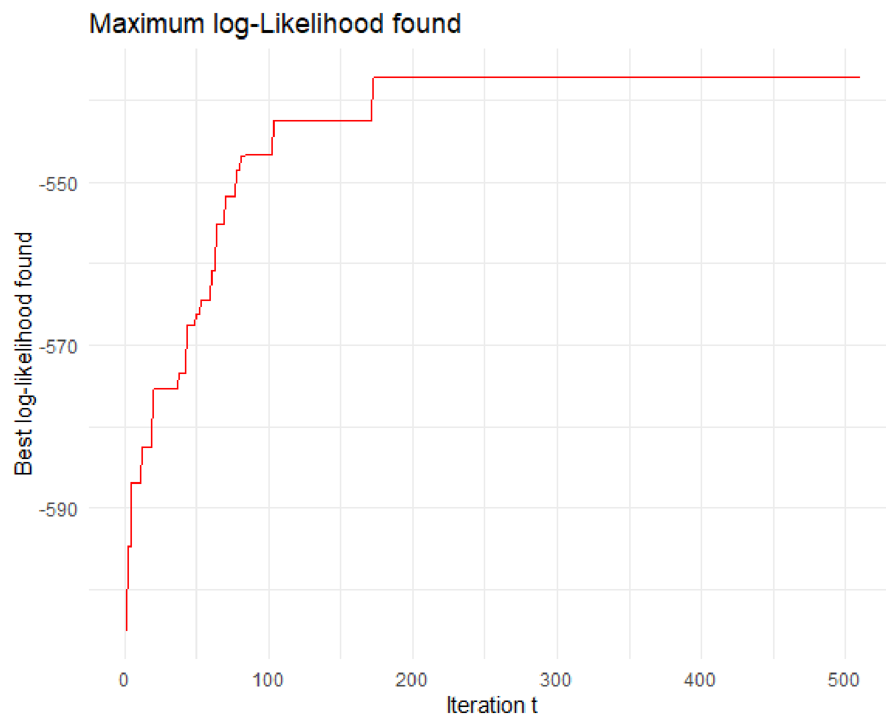

Figure 14: Best log likelihood found at iteration t

5 Hard estimation using DAGs

5.1 Introduction

In this chapter I will use the theory of DAGs to identify the best cluster configurations without the assumption of local independence. The implementation of the algorithm makes use of the R *bnlearn* package developed by Marco Scutari [4]. The observations are supposed to be polytomous (more than two outcomes) and the number of cluster ≥ 2. The assumpion of global independence still holds among the units and among each cluster.

5.2 Theorical introduction to DAGs

The DAGs are Directed Acyclic Graphs, where each node represents a random variable, and the edges (or arcs) refer to probabilistic dependencies. In the problem of polytomous observations, there are r random variables, then the set of nodes will be indicated as follows:

$$V = \{X_1, ..., X_r\}$$

The hypothesis that holds in this case is the Markov property of Bayesian Networks: first, since there is a structure of a graph, it is defined, for each node X_k, the set of its parents, that will be indicated as $\prod_{X_j}$, which are all the nodes X_p in V such that an arc a_{pj} exists. Having defined the set of parent nodes, it is possible to define the Markov property for Bayesian Networks, similarly to the Markov Chain property:

$$P(X_1, ..., X_r) = \prod_{j=1}^{r} P\left(X_j \middle| \prod X_j\right)$$

The complexity of the maximum likelihood estimation increases, since also the relations (arcs) between each variable has to be found. Some heuristics are already implemented in the *bnlearn* package to find an optimal DAG structure and likelihood. In addition, the clustering problem will be solved fitting a DAG to each cluster, and evaluating the log-Likelihood, for each considered configuration cl.

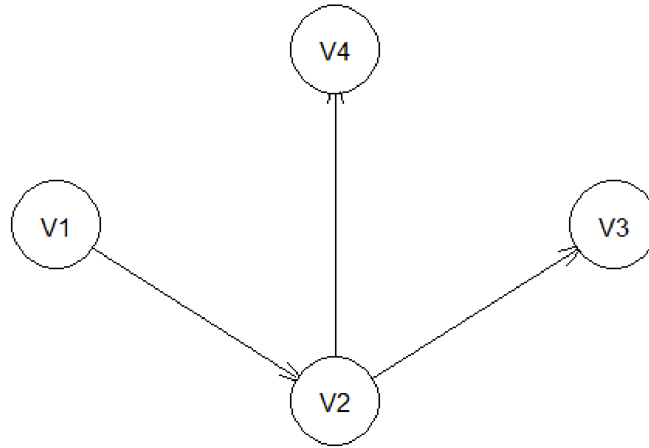

Figure 15: Directed Acyclic Graph (DAG) obtained from the full Bartholomew dataset

5.3 bnlearn package

UIn this part we will use again the standard Bartholomew dataset, composed of 4 questions and 16 profiles. First of all, it is interesting to see how *bnlearn* works without introducing the problem of clustering. Naming the profiles dataset as *df*, the following commands of *bnlearn* find the best DAG:

```
# finding the hierarchies among the columns
res <- hc(df)
res$arcs

> res$arcs
     from to
[1,] "V2" "V4"
[2,] "V1" "V2"
[3,] "V2" "V3"
```

It is also possible to visualize such relations with a graph, as in Figure 15.

And finally also the log-Likelihood is available

```
# log-Likelihood evaluation
score(hc(df), df, type = "loglik")

> score(hc(df), df, type = "loglik")
[1] -336.5064
```

In this way, it is possible to avoid the strong local independence assumption but, instead, a softer assumption of Markov has been made among the questions.

5.4 Clustering implementation

Since the clustering problem using DAGs is even more difficult to solve than the previous ones, it is not feasible to use the complete assignment matrix, even for few profiles, then a Monte Carlo approach will be used. First, a function that evaluates the log-Likelihood of a certain configuration is needed, as done previously, so that it can be therefore vectorized to a simulation matrix containing multiple configurations.

```
logLikeBayesEval <- function(cl, df){
  # retrieving number of questions
  r <- dim(df)[2]

  # creating list of dataframes for each cluster
  dfList <- cbind(df, cl) %>% split(cl)

  # evaluating scores for each cluster
  loglikes <- lapply(dfList, function(dfTmp) {
    res <- hc(dfTmp[,1:r])
    outList <- list()
    outList[[1]] <- res
    outList[[2]] <- score(res, dfTmp[,1:r], type = "loglik")
    return(outList)
    })

  # returning the sum of the log-likes and the structure
  output <- list()
  output[[1]] <- loglikes
  output[[2]] <- Reduce("+", lapply(loglikes,
  function(listTmp) listTmp[[2]]))
  return(output)
}
```

Then, the vectorization is done via the following function, where the output contains not only the log-likelihoods obtained, but also the graph structure estimated by *bnlearn*:

```
colLikeBayes <- function(submat) {
  loglike <- apply(submat, 2,
  function(cl) logLikeBayesEval(cl, df))

  loglikevector <- unlist(lapply(loglike,
  function (listTmp) listTmp[[2]]))

  output <- list()
  unique(as.data.frame(cbind(t(submat), loglikevector)) %>%
           filter(loglikevector == max(loglikevector))) %>%
    head(1) -> output[[1]]

  for (loglikeTmp in loglike) {
    if(loglikeTmp[[2]] == as.numeric(output[[1]] %>% select(loglikevector))){
```

```
      output[[2]] <- loglikeTmp[[1]]
    }
  }
```

As done previously with the binary and non-binary case, the Monte Carlo function is similarly implemented, taking into account also the DAG structure:

```
swapMCBayes <- function(df, k, nIter = 500, MCnSim = 100, MCnIter = 10){
  m <- nrow(df)
  r <- ncol(df) - 1

  # inizialization using Monte Carlo basic method
  cat("Initializing the first configuration using MC ...\n")
  init <- MCBayes(df, k, MCnIter, MCnSim)
  cl <- init[[1]]
  L0 <- init[[2]]

  # initialization of the best configuration
  cl_best <- cl

  # initialization of the best log-Likelihood
  L_best <- L0

  # iterations
  for (it in 1:nIter){
    # swap
    cl <- cl_best
    rnd <- sample(1:m,1)
    cl[rnd] <- sample(c(1:k)[c(1:k) != cl[rnd]], 1)

    # if better, mantain the swap
    if (logLikeBayesEval(cl, df) > L_best) {
      L_best <- logLikeBayesEval(cl, df)
      cl_best <- cl
    }
  }

  # results
  output <- list()
  output[[1]] <- cl_best
  output[[2]] <- L_best
  return(output)
}
```

Now it is possible to apply the previously defined functions to the Bartholomew dataset:

```
# trying clustering - MC
k <- 3
```

```
set.seed(7)
out <- MCBayes(df, k)

# best configurations
unique(cbind(df, bayes_assignment = out[[1]]))
```

And the result of the assignment for three classes is the following:

```
> unique(cbind(df, bayes_assignment = out[[1]]))
   V1 V2 V3 V4 bayes_assignment
1   1  1  1  1                3
2   1  1  0  1                1
3   1  1  1  0                3
4   0  1  1  1                3
5   1  0  1  1                1
6   1  1  0  0                3
7   1  0  0  1                1
8   0  1  0  1                1
9   1  0  1  0                1
10  0  1  1  0                1
11  0  0  1  1                2
12  1  0  0  0                3
13  0  1  0  0                1
14  0  0  0  1                3
15  0  0  1  0                2
16  0  0  0  0                2
```

This result can be considered preliminar and can be improved, becuase it is obtained using only a randomized Monte Carlo methods. It would be essential to consider and develop proper heuristic approaches that considers also the arcs estimated in the DAG. This part will constitue a futere development of this work.

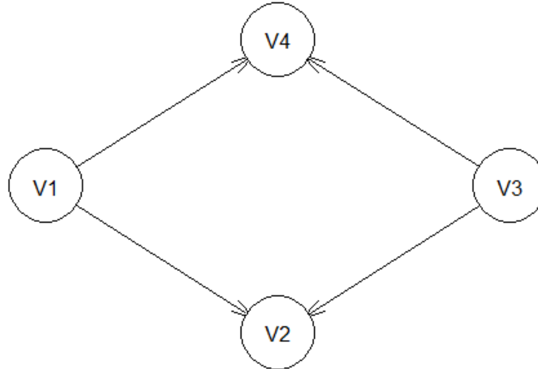

Figure 16: Directed Acyclic Graph (DAG) of the first cluster

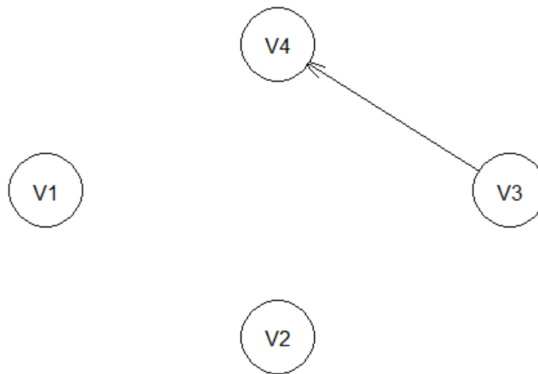

Figure 17: Directed Acyclic Graph (DAG) of the second cluster

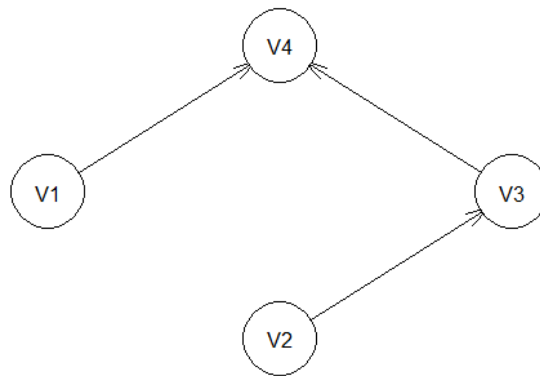

Figure 18: Directed Acyclic Graph (DAG) of the third cluster

Conclusion

Different approaches have been used to solve the clustering problem using the maximization of the Likelihood function, exploiting both hard and soft techniques. Despite the large technological improvement, the hard assignment turned out to be in general a non-solvable problem due to physical memory limitations. Nevertheless, it is possible to obtain approximations of the optimum configurations making use of some heuristics.

The use of the EM algorithm results in a soft assignment of variables through an estimation of the subsequent probability of belonging to each cluster. In general, the algorithm differs from the hard assignment; however, in some specific cases, the two approaches may match, such as in a totally symmetrical case, where also the assignment solutions are symmetrical. The reasons for this behaviour have been discussed and an original conjecture has been formulated.

For very small binary datasets (with a number of profiles less than 25) it may be possible to solve the clustering problem within an acceptable timing: using both the vectorization of the code and the parallelization, so it is possible to leverage the computational potential and solve the problem in an exact way. A comparison of hard and soft approach has been evaluated on a small standard dataset (with four binary questions and 16 different profiles), commonly used to explain LCA models. For bigger problems with multiple possible answers and more than two clusters, the implemented "swap" heuristic is essential, because of memory and timing bounds of the exact solution, since the complexity of the problem is exponential. In particular, if the questions are not binary and many units are observed, it is likely to obtain more and more profiles as the possible outcomes of each question increases, and not many repetitions of the units will be observed. Up to this point our reasoning relies on the strong assumption of local independence among the predictors (i.e. questions). In the last part of the thesis I started to explore ways to model correlations and to develop more complex structures in the data.

The use of Directed Acyclic graphs can overcome the hypothesis of local independence among the predictors, using instead a softer Markov property. However its implementation requires a deeper evaluation of heuristics to find local optima. With this Bayes approach, in fact, the variables of the problem increase, since there are not only the profiles and the assignment matrix, but also, for each possible configuration, there are multiple possible graphs explaining the relations among the predictors. This generates a more difficult problem to solve, and the heuristics used should be also consider the fact that the local independence does not hold anymore. A further analysis of this approach, with a deeper investigation of difficulties and possibe solutions wiil constitue a future development of this work.

Appendix

A Binary and non-binary classification

```
# to convert individual data to profiles
dataToProfiles <- function(data){
  # obtaining profiles
  data %>%
    unique(margin = 1) -> profiles

  # cbinding the frequencies
  profiles <- cbind(profiles, profiles %>% # obtaining frequencies
    apply(1, function(y) apply(data,1,function(x) all(x==y))) %>%
    colSums() -> frequencies)
  return(profiles)
}

# for binary data and 2 cluster:
# function to evaluate the log-likelihood laplace smoothed
# of a certain cluster configuration cl
likelihoodEval01 <- function(cl, profiles){
  # number of questions (columns - 1) in profiles
  r = ncol(profiles) - 1

  # cl is a logical vector
  # 1 if the element is in the + cluster
  # 0 if the element is in the - cluster

  # finding the number of elements in the + cluster
  den_p = sum(profiles[cl, r + 1])

  # finding the number of elements in the - cluster
  den_m = sum(profiles[!cl, r + 1])

  # in case the C+ cluster is composed by one element
  if ((length(cl[cl == TRUE]) > 1) || (length(cl[cl == TRUE]) == 0)) {
    num_p = colSums(profiles[cl, 1:r]*profiles[cl, r+1])
  } else {
    num_p = profiles[cl, 1:r]*profiles[cl, r+1]
  }

  # in case the C- cluster is composed by one element
  if ((length(cl[cl == FALSE]) > 1) || (length(cl[cl == FALSE]) == 0)) {
    num_m = colSums(profiles[!cl, 1:r]*profiles[!cl, r+1])
  } else {
```

```
    num_m = profiles[!cl, 1:r]*profiles[!cl, r+1]
  }

  # evaluating probabilities in C+ and C- using Laplace Smoothing (1, 2)
  p_p = (num_p + 1)/(den_p + 2)
  p_m = (num_m + 1)/(den_m + 2)

  # C+ total log-likelihood
  CpL = sum((profiles[cl,1:r]*profiles[cl,r+1])%*%log(p_p) +
              ((1 - profiles[cl,1:r])*profiles[cl,r+1])%*%log(1-p_p))

  # C- total log-likelihood
  CmL = sum((profiles[!cl,1:r]*profiles[!cl,r+1])%*%log(p_m) +
              ((1 - profiles[!cl,1:r])*profiles[!cl,r+1])%*%log(1-p_m))

  # total likelihood
  L = CpL + CmL
  return(L)
}

# Laplace smoothing parameters as input
likelihoodEval01Laplace <- function(cl, profiles, a = 1, b = 2){
  # number of questions (columns - 1) in profiles
  r = ncol(profiles) - 1

  # cl is a logical vector
  # 1 if the element is in the + cluster
  # 0 if the element is in the - cluster

  # finding the number of elements in the + cluster
  den_p = sum(profiles[cl, r + 1])

  # finding the number of elements in the - cluster
  den_m = sum(profiles[!cl, r + 1])

  # in case the C+ cluster is composed by one element
  if ((length(cl[cl == TRUE]) > 1) || (length(cl[cl == TRUE]) == 0)) {
    num_p = colSums(profiles[cl, 1:r]*profiles[cl, r+1])
  } else {
    num_p = profiles[cl, 1:r]*profiles[cl, r+1]
  }

  # in case the C- cluster is composed by one element
  if ((length(cl[cl == FALSE]) > 1) || (length(cl[cl == FALSE]) == 0)) {
    num_m = colSums(profiles[!cl, 1:r]*profiles[!cl, r+1])
  } else {
```

```
    num_m = profiles[!cl, 1:r]*profiles[!cl, r+1]
  }

  # evaluating probabilities in C+ and C- using Laplace Smoothing (1, 2)
  p_p = (num_p + a)/(den_p + b)
  p_m = (num_m + a)/(den_m + b)

  # C+ total log-likelihood
  CpL = sum((profiles[cl,1:r]*profiles[cl,r+1])%*%log(p_p) +
              ((1 - profiles[cl,1:r])*profiles[cl,r+1])%*%log(1-p_p))

  # C- total log-likelihood
  CmL = sum((profiles[!cl,1:r]*profiles[!cl,r+1])%*%log(p_m) +
              ((1 - profiles[!cl,1:r])*profiles[!cl,r+1])%*%log(1-p_m))

  # total likelihood
  L = CpL + CmL
  return(L)
}

# returns a list of N submatrices of mat
splitMat <- function(mat, N){

  # initialising the list of sub-matrices with the first element
  submats = list(mat[,1:round(dim(mat)[2]/N, 0)])

  # appending the remaining matrices
  for (k in 1:(N - 2)) {
    submats = append(submats,
      list(mat[,(k*round(dim(mat)[2]/N, 0)):((k + 1)*round(dim(mat)[2]/N, 0))]))
  }

  # appending the last one
  submats = append(submats,
      list(mat[,((N - 1)*round(dim(mat)[2]/N, 0)):dim(mat)[2]]))

  # output
  return(submats)
}

# likelihood of a column of assignment
# returns the best cls and the relative loglike
colLike01 <- function(submat) {
  loglike <- apply(submat, 2,
  function(cl) likelihoodEval01(as.logical(cl), df))
  return(unique(as.data.frame(cbind(t(submat), loglike)) %>%
```

```
                filter(loglike == max(loglike))))
}

# colLike01() with a and b Laplace smoothing parameters as input
colLike01Laplace <- function(submat, a = 1, b = 2) {
  loglike <- apply(submat, 2,
  function(cl) likelihoodEval01Laplace(as.logical(cl), df, a = a, b = b))
  return(unique(as.data.frame(cbind(t(submat), loglike)) %>%
                filter(loglike == max(loglike))))
}

MC <- function(df, nIter = 100, nSim = 1000){

  # retrieving number of profiles and answers
  m <- nrow(df)
  r <- ncol(df) - 1

  # initialization solution vector (best solution until t step)
  solution <- rep(-Inf, nIter)

  # initialization best solution (max likelihood found)
  best_sol <- -Inf

  # initialization best configuration initialization
  best_conf = rep(0, m)

  # iterations
  for (t in 1:nIter){
    SIM <- matrix(sample(0:1, nSim*m, replace = TRUE), nrow = m, ncol = nSim)

    # current solution dataframe at iteration t
    current <- (colLike01(SIM) %>% filter(loglike == max(loglike)) %>% unique())

    # current max likelihood found at iteration t
    current_sol <- current$loglike %>% unlist() %>% unique()

    # current configuration found at iteration t
    current_conf <- current %>% select(-loglike) %>% head(1) %>% as.numeric()

    # if a new maximum is found
    if (current_sol > best_sol){
      best_sol <- current_sol
      best_conf <- current_conf
    }

    solution[t] <- best_sol
```

```
  }

  # results
  output <- list()
  output[[1]] <- best_conf
  output[[2]] <- best_sol
  return(output)
}

##### parallelization 01 case #####

# in order to have always the same random matrices
set.seed(7)

# number of questions
r <- 5

# random units
data <- matrix(rbinom(1000, 1, .6), 200, r, byrow = T)
head(data)

# conversion to profiles
profiles <- dataToProfiles(data)
profiles <- profiles[1:17,]

# total number of profiles
m <- nrow(profiles)
m

# hist of frequencies
frequencies <- profiles[,r + 1]
df = data.frame(frequencies = sort(frequencies, decreasing = TRUE), index = 1:m)
ggplot(data = df, aes(x = index, y =  frequencies)) +
  geom_bar(stat = "identity", fill = "#FF6666")+
  theme_minimal() +
  labs(title = "Frequencies of the observed profiles") +
  ylab("number ob observations of the profile") +
  xlab("profiles' index")

# assignment matrix generation
df <- profiles
tmp <- split.data.frame(cbind(rep(0, m), rep(1, m)),rep(1:m))
assignments <- matrix(t(expand.grid(tmp)), nrow=m, ncol=2^m)

# subdivision of the assignment matrix into n (= n cores) submatrices
# creating clusters for parallel computing
```

```
numCores <- detectCores()
numCores
clusters <- makeCluster(numCores)

# list containing numCores sub-matrices of assignments
submats <- splitMat(assignments, numCores)

# dimensions of each sub-matrix
dim(submats[[1]])

# exporting the libraries
clusterEvalQ(clusters, {library(dplyr); library(magrittr)})

# data and functions exporting
clusterExport(clusters, c("df", "colLike01", "likelihoodEval01"))

# running clusters in parallel
parallelOutput <- parLapply(clusters, submats, colLike01)
parallelOutput

# running not in parallel
output <- colLike01(assignments)

# comparison
Reduce("rbind", parallelOutput) %>% filter(loglike == max(loglike))
output

NITER = 10

# parallel version
start_time = Sys.time()
parallelOutput <- parLapply(clusters, submats, colLike01)
end_time = Sys.time()
end_time - start_time

# non parallel version
start_time = Sys.time()
output <- colLike01(assignments)
end_time = Sys.time()
end_time - start_time
##### Laplace smoothing varying parameters #####

# default version a = 1, b = 2
start_time = Sys.time()
output_12 <- colLike01(assignments)
end_time = Sys.time()
```

```
end_time - start_time

# version a = 2, b = 4
start_time = Sys.time()
output_24 <- colLike01Laplace(assignments, a = 2, b = 4)
end_time = Sys.time()
end_time - start_time

# version a = exp(-10), b = 2*exp(-10)
start_time = Sys.time()
output_ee <- colLike01Laplace(assignments, a = exp(-10), b = 2*exp(-10))
end_time = Sys.time()
end_time - start_time

# outputs
output_12
output_24
output_ee

##### MC basic #####
df # make sure there are the 17 profiles of before
m
r

# generating SIM matrix
nSim <- 10
set.seed(5)
SIM <- matrix(sample(0:1, nSim*m, replace = TRUE), nrow = m, ncol = nSim)

# best configuration
colLike01(SIM)

# parameters
set.seed(5)
nIter <- 100
nSim <- 1000

# initialization solution vector (best solution until t step)
solution <- rep(-Inf, nIter)

# initialization best solution (max likelihood found)
best_sol <- -Inf

# initialization best configuration initialization
best_conf = rep(0, m)
```

```
# iterations
for (t in 1:nIter){
  SIM <- matrix(sample(0:1, nSim*m, replace = TRUE), nrow = m, ncol = nSim)

  # current solution dataframe at iteration t
  current <- (colLike01(SIM) %>% filter(loglike == max(loglike)) %>% unique())

  # current max likelihood found at iteration t
  current_sol <- current$loglike %>% unlist() %>% unique()

  # current configuration found at iteration t
  current_conf <- current %>% select(-loglike) %>% head(1) %>% as.numeric()

  # if a new maximum is found
  if (current_sol > best_sol){
    best_sol <- current_sol
    best_conf <- current_conf
  }

  solution[t] <- best_sol
}

# results
best_conf
max(solution)

# plot
dfTmp <- data.frame(best_log_likelihood = solution, iteration = 1:length(solution))
ggplot(data = dfTmp, aes(x = iteration, y = best_log_likelihood, group=1)) +
  geom_line(color = "red") +
  theme_minimal() +
  labs(title = "Best log-likelihood found") +
  ylab("Best log-likelihood found") +
  xlab("iteration t")

##### MC Swap #####

# in order to have always the same random matrices
set.seed(7)

# number of questions
r <- 5

# random units
data <- matrix(rbinom(1000, 1, .6), 200, r, byrow = T)
head(data)
```

```
# conversion to profiles
profiles <- dataToProfiles(data)
profiles <- profiles[1:20,]
m <- nrow(profiles)

# assignment matrix generation
df <- profiles
tmp <- split.data.frame(cbind(rep(0, m), rep(1, m)),rep(1:m))
assignments <- matrix(t(expand.grid(tmp)), nrow=m, ncol=2^m)

# subdivision of the assignment matrix into n (= n cores) submatrices
# creating clusters for parallel computing
numCores <- detectCores()
clusters <- makeCluster(numCores)

# list containing numCores sub-matrices of assignments
submats <- splitMat(assignments, numCores)

# exporting the libraries
clusterEvalQ(clusters, {library(dplyr); library(magrittr)})

# data and functions exporting
clusterExport(clusters, c("df", "colLike01", "likelihoodEval01"))

# running clusters in parallel - exact solution
start_time = Sys.time()
parallelOutput <- parLapply(clusters, submats, colLike01)
end_time = Sys.time()
end_time - start_time

# best solution obtained by the exact method
Reduce("rbind", parallelOutput) %>%
  filter(loglike == max(loglike))-> exact_sol
exact_sol %>% select(loglike) %>% unique()

# swap heuristic
set.seed(8)
start_time = Sys.time()
swapMC(df)
end_time = Sys.time()
end_time - start_time

# how many times maximum is reached
maxima <- rep(0, 100)
for (i in 1:100) {
```

```
  cat(paste0(i, "\n"))
  set.seed(i)
  maxima[i] <- swapMC(df)[[2]]
}

# correct solutions
sum(maxima == max(maxima))

dfTmp <- data.frame(occurrencies = maxima)
ggplot(dfTmp, aes(x = occurrencies)) +
  geom_histogram(fill = "darkred") +
  theme_minimal() +
  labs(title = "Frequencies of the observed profiles") +
  ylab("number ob observations of the profile") +
  xlab("profiles' index")

##### Bartholomew dataset #####
source("DEF_functions.R")
source("DEF_graphics.R")
library(dplyr)
library(pryr)
library(ggplot2)
library(parallel)

data_Bart <- read.table('Bartholomew_dataset.txt', sep ='\t', header=T)
data_Bart <- data.frame(frequencies = data_Bart$Observed_freq,
                        q1 = data_Bart$V1,
                        q2 = data_Bart$V2,
                        q3 = data_Bart$V3,
                        q4 = data_Bart$V4)
t(data_Bart)
m <- dim(data_Bart)[1]

# histogram
df = data.frame(frequencies = sort(data_Bart$frequencies, decreasing = TRUE), index = 1
ggplot(data = df, aes(x = index, y =  frequencies)) +
  geom_bar(stat = "identity", fill = "#FF6666")+
  theme_minimal() +
  labs(title = "Frequencies of the observed profiles") +
  ylab("number ob observations of the profile") +
  xlab("profiles' index")

# m x 2^m assignment matrix evaluation
tmp <- split.data.frame(cbind(rep(0, m), rep(1, m)),rep(1:m))
assignments <- matrix(t(expand.grid(tmp)), nrow=m, ncol=2^m)
```

```
# profiles evaluation
profiles <- data.frame(data_Bart$q1, data_Bart$q2, data_Bart$q3, data_Bart$q4,
                       data_Bart$frequencies) %>%
            as.matrix()
df <- profiles

# exact method
numCores <- detectCores()
clusters <- makeCluster(numCores)
submats <- splitMat(assignments, numCores)
clusterEvalQ(clusters, {library(dplyr); library(magrittr)})
clusterExport(clusters, c("df", "colLike01", "likelihoodEval01"))
start_time = Sys.time()
parallelOutput <- parLapply(clusters, submats, colLike01)
end_time = Sys.time()
end_time - start_time
Reduce("rbind", parallelOutput) %>%
  filter(loglike == max(loglike))-> exact_sol
exact_sol %>% dplyr::select(loglike) %>% unique()
exact_sol

# comparison with EM

# different solutions found
hard_sol <- as.numeric(exact_sol[1,] %>% dplyr::select(-loglike))
poLCA_sol <- c(1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0)

# log-likelihoods
likelihoodEval01(as.logical(hard_sol), df)
likelihoodEval01(as.logical(poLCA_sol), df)

# c-binding the dataframe with the solutions found
complete_df <- as.data.frame(cbind(df, hard_sol, poLCA_sol))

# number of individuals
individuals <- complete_df %>%
    dplyr::select(data_Bart.frequencies) %>%
    sum()

# finding number of individuals inside cluster + for hard
complete_df %>%
  # cluster + for hard
  filter(hard_sol == 1) %>%
  dplyr::select(data_Bart.frequencies) %>%
  sum() -> np_hard
```

```
# number of individuals inside cluster - for hard
nm_hard <- individuals - np_hard

# finding number of individuals inside cluster + for soft
complete_df %>%
  # cluster + for soft
  filter(poLCA_sol == 1) %>%
  dplyr::select(data_Bart.frequencies) %>%
  sum() -> np_soft

# number of individuals inside cluster - for hard
nm_soft <- individuals - np_soft
c(np_hard, np_soft)
c(nm_hard, nm_soft)

# heuristic
set.seed(7)
start_time = Sys.time()
results_swap <- swapMC(df)
end_time = Sys.time()
end_time - start_time
results_swap[[1]]
results_swap[[2]]

# exact method
results_exact <- apply(assignments,
2, function(cl) likelihoodEval(as.logical(cl), profiles))


dfResults = as.data.frame(cbind(t(assignments), results_exact))
dfResults %>% head()

# best configuration
dfResults %>% filter(results_exact == max(results_exact))

# worst configuration
dfResults %>% filter(results_exact == min(results_exact))

# heuristic method swap
df <- profiles
results_swap <- swapMC(df)



##### Generalization #####
# used libraries
source("DEF_functions.R")
```

```
source("DEF_graphics.R")
library(dplyr)
library(pryr)
library(ggplot2)
library(parallel)

# example of dataframe
set.seed(7)

# parameters initialization
k = 3 # number of clusters
r = 4 # numbers of questions

# example matrix
d1 <- sample(c(0, 1), 30, replace = TRUE)
d2 <- sample(c(0, 1), 30, replace = TRUE)
d3 <- sample(c(0, 1, 2), 30, replace = TRUE)
d4 <- sample(c(0, 1, 2, 3), 30, replace = TRUE)
d5 <- sample(c(0, 1, 2, 3), 30, replace = TRUE)
d6 <- sample(c(0, 1, 2, 3), 30, replace = TRUE)
d7 <- sample(c(0, 1, 2, 3), 30, replace = TRUE)
d8 <- sample(c(0, 1, 2, 3), 30, replace = TRUE)
freq <- sample(c(1, 2, 3, 4), 30, replace = TRUE)
df = data.frame(d1, d2, d3, d4, d5, d6, d7, d8, freq)

head(df)

# SIM matrix, for each column we have a configuration cl
NSIM <- 10000
nProfiles <- dim(df)[1]
SIM <- matrix(sample(1:k, NSIM*nProfiles, replace = TRUE),
              nrow = nProfiles, ncol = NSIM)

# first configuration considered
cl <- SIM[1:30, 1]

# splitting in list of dataframes
dfList <- cbind(df, cl) %>% split(cl)
dfList[[1]]
dfList[[2]]
dfList[[3]]

# number of questions
r <- dim(df)[2] - 1

# test dataset
```

```r
dfTmp <- dfList[[1]]

# x is the value d is the column:
# e.g.:
# colFreq(2,4) =
# frequency *(log (laplace-smoothed) probability of finding 2 in column 4)
colFreq <- function(x, d, dfTmp) {
  # number of x in question d
  numX <- sum(as.numeric(dfTmp[, d] == x) * dfTmp$freq)

  # returns the log of probabilities times the frequency
  return(numX*log((numX + 1)/(sum(dfTmp$freq) + 2)))
}

# outcome of colFreq(1, 1, dfTmp)
x <- 1
d <- 1

# number of x in question d
numX <- sum(as.numeric(dfTmp[, d] == x) * dfTmp$freq)
numX

# returns the log of probabilities times the frequency
numX*log((numX + 1)/(sum(dfTmp$freq) + 2))

# each answer in d = 1
d <- 1
unique(dfTmp[, d])

# applying to each possible answer in answer d
sapply(unique(dfTmp[, d]), function(x) colFreq(x, d, dfTmp))

# applying to each question
sapply(1:r,
    function(d) sum(sapply(unique(dfTmp[, d]),
                           function(x) colFreq(x, d, dfTmp)))) %>%
  sum()

# applying to each dfTmp in dfList
Reduce("+",
 lapply(dfList, # applying to each cluster
    function(dfTmp) {
      sapply(1:r, # applying to each column
          function(d) sum(sapply(unique(dfTmp[, d]),
              function(x) colFreq(x, d, dfTmp)))) %>%
      sum()}))
```

```
# function to be used
logLikeEval(cl, df)

# creating clusters for parallel computing
numCores <- detectCores()
numCores
clusters <- makeCluster(numCores)

# exporting the libraries, the dataframe and the functions inside each cluster
clusterEvalQ(clusters, {library(dplyr); library(magrittr)})
clusterExport(clusters, c("df", "colLike", "logLikeEval"))

# creating list of numCores submatrices of SIM matrix
submats = list(SIM[,1:round(dim(SIM)[1]/numCores, 0)])
for (k in 1:(numCores - 2)) {
  submats = append(submats,
  list(SIM[,(k*round(dim(SIM)[2]/numCores, 0)):((k + 1)*round(dim(SIM)[2]/numCores, 0))
}
submats = append(submats,
list(SIM[,((numCores - 1)*round(dim(SIM)[2]/numCores, 0)):dim(SIM)[2]]))

# running clusters in parallel
parallelOutput = parLapply(clusters, submats, colLike)

# stopping clusters
stopCluster(clusters)

# finding the maximum likelihood
do.call("rbind", parallelOutput) %>%
  unique() %>%
  filter(loglike == max(loglike)) %>%
  select(loglike)

# swap heuristic - MC initialization

# initialization

set.seed(8)
k = 3
nIter = 10
nSim = 100

# retrieving number of profiles and answers
m <- nrow(df)
r <- ncol(df) - 1
```

```
# initialization solution vector (best solution until t step)
solution <- rep(-Inf, nIter)

# initialization best solution (max likelihood found)
best_sol <- -Inf

# initialization best configuration initialization
best_conf = rep(0, m)

# iterations
for (t in 1:nIter){
  SIM <-  matrix(sample(1:k, nSim*m, replace = TRUE), nrow = m, ncol = nSim)

  # current solution dataframe at iteration t
  current <- (colLike(SIM) %>% filter(loglike == max(loglike)) %>% unique())

  # current max likelihood found at iteration t
  current_sol <- current$loglike %>% unlist() %>% unique()

  # current configuration found at iteration t
  current_conf <- current %>% select(-loglike) %>% head(1) %>% as.numeric()

  # if a new maximum is found
  if (current_sol > best_sol){
    best_sol <- current_sol
    best_conf <- current_conf
  }

  solution[t] <- best_sol
}

# results
output <- list()
output[[1]] <- best_conf
output[[2]] <- best_sol

# plot(solution)
dfTmp <- data.frame(best_log_likelihood = solution,
iteration = as.factor(1:length(solution)))
ggplot(data = dfTmp, aes(x = iteration, y = best_log_likelihood, group=1)) +
  geom_line(color = "red") +
  theme_minimal() +
  labs(title = "MC initialization") +
  ylab("Best log-likelihood found") +
  xlab("iteration t")
```

```
# swap

# number of swaps
nIter = 500

# MC initialisation
cl <- output[[1]]
L0 <- output[[2]]

# initialization of the best configuration
cl_best <- cl

# initialization of the best log-Likelihood
L_best <- L0

solutionSwap <- rep(0, nIter)

# iterations
for (it in 1:nIter){
  cl <- cl_best
  # changing one random element
  rnd <- sample(1:m,1)
  cl[rnd] <- sample(c(1:k)[c(1:k) != cl[rnd]], 1)

  # if better, mantain the swap
  if (logLikeEval(cl, df) > L_best) {
    L_best <- logLikeEval(cl, df)
    cl_best <- cl
  }
  solutionSwap[it] <- L_best
}

cl_best
L_best

# plot(c(solution, solutionSwap))
dfTmp <- data.frame(best_log_likelihood = c(solution, solutionSwap),

iteration = 1:length(c(solution, solutionSwap)))
ggplot(data = dfTmp, aes(x = iteration, y = best_log_likelihood, group=1)) +
  geom_line(color = "red") +
  theme_minimal() +
  labs(title = "Maximum log-Likelihood found") +
  ylab("Best log-likelihood found") +
```

```
xlab("Iteration t")
```

B DAG classification

```
##### functions #####
logLikeBayesEval <- function(cl, df){
  # retrieving number of questions
  r <- dim(df)[2]

  # creating list of dataframes for each cluster
  dfList <- cbind(df, cl) %>% split(cl)

  # evaluating scores for each cluster
  loglikes <- lapply(dfList, function(dfTmp) {
    res <- hc(dfTmp[,1:r])
    outList <- list()
    outList[[1]] <- res
    outList[[2]] <- score(res, dfTmp[,1:r], type = "loglik")
    return(outList)
    })


  # returning the sum of the log-likes and the structure
  output <- list()
  output[[1]] <- loglikes
  output[[2]] <- Reduce("+", lapply(loglikes, function(listTmp) listTmp[[2]]))
  return(output)
}

colLikeBayes <- function(submat) {
  loglike <- apply(submat, 2, function(cl) logLikeBayesEval(cl, df))

  loglikevector <- unlist(lapply(loglike, function (listTmp) listTmp[[2]]))

  output <- list()
  unique(as.data.frame(cbind(t(submat), loglikevector)) %>%
            filter(loglikevector == max(loglikevector))) %>%
    head(1) -> output[[1]]

  for (loglikeTmp in loglike) {
    if(loglikeTmp[[2]] == as.numeric(output[[1]] %>% dplyr::select(loglikevector))){
      output[[2]] <- loglikeTmp[[1]]
    }
  }

  return(output)
}
```

```r
MCBayes <- function(df, k, nIter = 10, nSim = 100){

  # retrieving number of profiles and answers
  m <- nrow(df)
  profiles <- dataToProfiles(df)
  m_prof <- profiles %>% nrow()
  r <- ncol(df)

  # to create individual observations
  freq.to.long <- function(x, freq){x[rep(1:length(freq), freq), ]}

  # initialization solution vector (best solution until t step)
  solution <- rep(-Inf, nIter)

  # initialization best solution (max likelihood found)
  best_sol <- -Inf

  # initialization best configuration initialization
  best_conf = rep(0, m)

  # iterations
  for (t in 1:nIter){

    SIM <- matrix(sample(1:k, nSim*m_prof, replace = TRUE),
           nrow = m_prof, ncol = nSim)
    SIM <-freq.to.long(SIM, profiles[,r+1])

    # current solution dataframe at iteration t
    sol <- colLikeBayes(SIM)
    current <- (sol[[1]] %>%
                  filter(loglikevector == max(loglikevector)) %>%
                  unique())

    # current max likelihood found at iteration t
    current_sol <- current$loglike %>% unlist() %>% unique()

    # current configuration found at iteration t
    current_conf <- current %>%
        dplyr::select(-loglikevector) %>%
        head(1) %>% as.numeric()

    # if a new maximum is found
    if (current_sol > best_sol){
      best_sol <- current_sol
      best_conf <- current_conf
```

```
      best_graph <- sol[[2]]
    }

    solution[t] <- best_sol
  }

  # results
  output <- list()
  output[[1]] <- best_conf
  output[[2]] <- best_sol
  output[[3]] <- best_graph
  return(output)
}

##### DAGs #####

library(bnlearn)
data_Bart <- read.table('Bartholomew_dataset.txt', sep ='\t', header=T)
freq.to.long <- function(x, freq){x[rep(1:length(freq), freq), ]}
data1 <- freq.to.long(data_Bart, data_Bart$Observed_freq)
df <- as.data.frame(data1[, 3:6])
df <- apply(df, 2, as.factor)
df <- as.data.frame(df)

# one cluster

# finding the hierarchies among the columns
res <- hc(df)
res$arcs
plot(res)

# log-Likelihood evaluation
score(hc(df), df, type = "loglik")

# trying clustering - MC
k <- 3
set.seed(7)
out <- MCBayes(df, k)

# best configurations
unique(cbind(df, bayes_assignment = out[[1]]))

# graph plotting
plot(out[[3]]$`1`[[1]]) # first cluster
plot(out[[3]]$`2`[[1]]) # second cluster
plot(out[[3]]$`3`[[1]]) # third cluster
```

References

[1] Agresti, Alan, and Brent A. Coull. *Approximate is better than "exact" for interval estimation of binomial proportions.* The American Statistician 52.2 (1998): 119-126.

[2] Bartholomew, David J., Fiona Steele, and Irini Moustaki. *Analysis of multivariate social science data.* Chapman and Hall/CRC, 2008.

[3] Dempster, Arthur P., Nan M. Laird, and Donald B. Rubin. *Maximum likelihood from incomplete data via the EM algorithm.* Journal of the Royal Statistical Society: Series B (Methodological) 39.1 (1977): 1-22.

[4] Friedman J, Hastie T and Tibshirani R. *The elements of statistical learning.* Springer Series in Statistics, New York, 2001.

[5] Li, Tao. *A unified view on clustering binary data.* Machine Learning 62.3 (2006): 199-215.

[6] LI, Tao; MA, Sheng; Ogihara, Mitsunori. *Entropy-based criterion in categorical clustering* In: Proceedings of the twenty-first international conference on Machine learning. ACM, 2004. p. 68.

[7] Jurafsky, Daniel, and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 2008.

[8] Linzer, Drew A., and Jeffrey B. Lewis. *poLCA: An R package for polytomous variable latent class analysis.* Journal of statistical software 42.10 (2011): 1-29.

[9] K. V. Mardia, J. T. Kent, J. M. Bibby. *Multivariate Analysis (Probability and Mathematical Statistics)* 1979, Academic Press Inc

[10] Scutari, Marco. *Learning Bayesian networks with the bnlearn R package.* arXiv preprint arXiv:0908.3817 (2009).