Bachelor Thesis

Model–based Clustering with Copulas

Dominik Ernst

March 21, 2010

Supervision:
Prof. Dr. Friedrich Leisch

Department of Statistics
Ludwig–Maximilians–University
Munich

Contents

Abstract

Copulas have become a popular tool in multivariate modeling. Applications include actuarial sciences, medical and epidemiological studies and hydrologic engineering. This thesis describes an extension to the *R* package *FlexMix* for copula clustering using finite mixture models. Foremost, the theoretical background is briefly introduced, including copulas, ML-estimation in copulas and estimation in mixture models using the EM-algorithm. Gumbel–Hougaard copulas serve as example throughout the thesis. Then the usage of this implementation is demonstrated in form of a tutorial. Also, the performance of the implementation is tested in a set of simulations, in which the estimates appear to asymptotically converge to their real values. Higher values of the copula parameter θ have a positive effect on the estimation error as well, while the estimation error increases with increasing amounts of clusters.

1 Motivation

Copulas have become a popular tool in multivariate modeling. Copulas are functions that provide a "coupling" between several univariate distribution functions into a joint distribution function. Also, with the choice of a specific copula one can model different kinds of dependence between the margins. This can be used for example in cases where a multivariate joint distribution is not already known and one would want to model a dependence structure between the margins, or to construct a joint distribution from (possibly different) univariate distributions while even accounting for dependency. Formulating multivariate distributions from copulas can also be easier than having to analytically derive a multivariate joint distribution.

Applications can be found in medical and epidemiological studies, actuarial sciences and hydrologic engineering, to name a few. Frees and Valdez [1998] name a list of possible applications, for example "survival of multiple lives" where the joint mortality rate of groups or pairs of individuals is assessed. Conversely, competing risks models can, in medical applictions, model the survival rate of individuals facing several causes of death, or in systems reliability, the failure rate of mechanical devices when single components fail. As for actuarial sciences, Frees and Valdez [1998] demonstrate the fitting of a copula to data containing insurance company indemnity claims, describing the joint distribution of company losses (indemnity payments) and expenses (such as lawyers' fees and investigation expenses).

In this thesis we consider data that consists of multiple clusters, where each cluster follows the same copula distribution with different parameters. This can be expressed as a finite mixture model, where each cluster is assigned a prior probability for an observation to originate from this specific cluster. These prior probabilities are assumed unknown and would have to be estimated along with the parameters of each cluster. For this we use the Expectation–Maximization (EM) algorithm. Conveniently there exists already an *R* package called *FlexMix* [Leisch, 2004], which offers an easily extendible implementation of the EM algorithm. Moreover, the copula package [Yan, 2007] provides support for different families of copulas and means to describe multivariate distributions made up of copulas and arbitrary margins in *R*. The goal of this thesis will then be to develop an M–step driver for *FlexMix*, using the functionality provided by the copula package.

2 Copulas

First of all a short introduction to copulas is given, limiting however to the case of bivariate copulas. Multivariate copulas behave analogously to bivariate copulas in many aspects, therefore bivariate copulas were chosen here for simplicity and easier visualization on two–dimensional paper.

Copulas can be defined as follows [Nelsen, 2007, p. 10]:

Definition 1 (Copula) *A* copula *is a function C from $[0,1]^2$ to $[0,1]$ with the following properties:*

1. *For every u, v in $[0,1]$*

$$C(u,0) = 0 = C(0,v)$$

and

$$C(u,1) = u \quad \text{and} \quad C(1,v) = v$$

2. *For every u_1, u_2, v_1, v_2 in $[0,1]$ such that $u_1 \leqslant u_2$ and $v_1 \leqslant v_2$*

$$C(u_2,v_2) - C(u_2,v_1) - C(u_1,v_2) + C(u_1,v_1) \geqslant 0$$

The first property makes a statement about the value of C at the edges of its domain. With either u or v being zero, C equals zero as well. If either u or v equals one, then the value of C equals v or u respectively. Considering the unit cube $[0,1]^3$ with axes u, v and $w = C(u,v)$, the section between the graph of C and the $u = 1$ or $v = 1$ plane is always the straight line from $[1,0,0]$ or $[0,1,0]$ to $[1,1,1]$. Also the value of $C(1,1)$ obviously equals one.

The second property is also called *2–increasing* or *quasi–monotone*[1]. This is similar to the term monotone increasing for one–dimensional functions as it can be shown that C is nondecreasing in each argument[2].

For example figure 2 shows the graphs of a Farlie–Gumbel–Morgenstern copula and a Gumbel–Hougaard copulas. The latter will be discussed in more detail in section 2.1.

The most fundamental property of copulas, which was first shown by Sklar and is hence named after him, is decribed in theorem 1 [3]:

[1] [Nelsen, 2007, p.8]
[2] [Nelsen, 2007, Lemma 2.1.4]
[3] [Nelsen, 2007, Theorem 2.3.3]

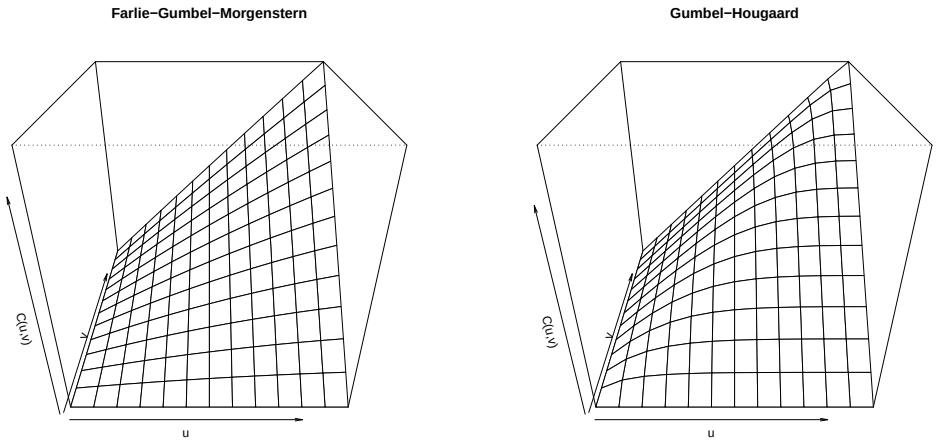

Figure 1: Example surfaces of Farlie–Gumbel–Morgenstern and Gumbel–Hougaard copulas.

Theorem 1 (Sklar's theorem) *Let H be a joint distribution function with margins F and G. Then there exists a copula C such that for all x, y in $\mathbf{R} \cup (-\infty, \infty)$,*

$$H(x, y) = C(F(x), G(y))$$

If F and G are continuous, then C is unique; otherwise, C is uniquely determined on $\mathrm{Ran}F \times \mathrm{Ran}G$*. Conversely, if C is a copula and F and G are distribution functions, then the function H defined by [theorem 1] is a joint distribution function with margins F and G.*

According to theorem 1 any (bivariate) joint distribution function can be expressed as the combination of a copula and its univariate margins. Interesting about this is that, in this notation, all the "ingredients" for a joint distribution can be changed out and replaced at will. For instance the practicioner could replace the copula, and therefore the dependence structure, by another copula which he deems more suitable for his data. The same applies to the marginal distributions as well. This is a rather simple process and even more so, with appropriate software packages (like the *copula* package in *R*).

2.1 Example: Gumbel–Hougaard copulas

As an example Gumbel–Hougaard copulas are presented here. These copulas belong to a class of copulas called *archimedian copulas*. Archimedian copulas

are constructed using a generator function φ:

Theorem 2 *Let φ be a continuous, strictly decreasing and convex function from $[0,1]$ to $[0,\infty]$ with $\varphi(1) = 0$ and $\varphi^{[-1]}$ being the pseudo–inverse of φ. Then the function $C : [0,1]^2 \to [0,1]$ given by*

$$C(u,v) = \varphi^{[-1]}\left(\varphi(u) + \varphi(v)\right)$$

is a copula[4]*.*

This is one of numerous methods of constructing copulas; Nelsen [2007] even devotes a whole chapter to these methods. For archimedian copulas it is however sufficient to find a generator function φ and to apply theorem 2.

Such a function is

$$\varphi_\theta(t) = (-\ln t)^\theta$$

which satisfies the conditions from theorem 2 for $\theta \geqslant 1$ (see appendix A.1) and can therefore be used as a generator function. First, one determines the inverse of φ_θ

$$\begin{aligned} \varphi_\theta(t) = y &= (-\ln t)^\theta \\ y^{1/\theta} &= -\ln t \\ \varphi_\theta^{[-1]}(y) := \exp\left(-y^{1/\theta}\right) &= t \end{aligned}$$

and applies theorem 2 to obtain:

$$\begin{aligned} \varphi_\theta^{[-1]}\left(\varphi_\theta(u) + \varphi_\theta(v)\right) &= \varphi_\theta^{[-1]}\left((-\ln u)^\theta + (-\ln v)^\theta\right) \\ C_\theta(u,v) &:= \exp\left(-\left[(-\ln u)^\theta + (-\ln v)^\theta\right]^{1/\theta}\right) \end{aligned} \qquad (1)$$

The resulting function $C_\theta(u,v)$ is referred to as *Gumbel–Hougaard copula*, whose graph was also shown in figure 2. C_θ is a one–parameter copula, i.e. it is parameterized by a single parameter θ. The value of θ determines the correlation between the margins. This is illustrated in figure 2.1; it shows example scatter plots of Gumbel–Hougaard copulas, where both margins are uniform over $[0,1]$ for different values of θ. Additionally, two–dimensional kernel density estimates[5] of the data depicted in the scatter plots were plotted as contour plots to further visualize the dependence of the data. For $\theta = 1$, the lowest possible value of θ, there is barely any dependence visible in the plots. For larger values of θ the plots show a stronger dependence between the margins and the data points appear to be accumulating around the diagonal line in a roughly elliptical shape.

[4] As derived from Lemma 4.1.2 and Theorem 4.1.4 in Nelsen [2007].

[5] Estimates were obtained using `kde2d` from *R*'s `MASS` library.

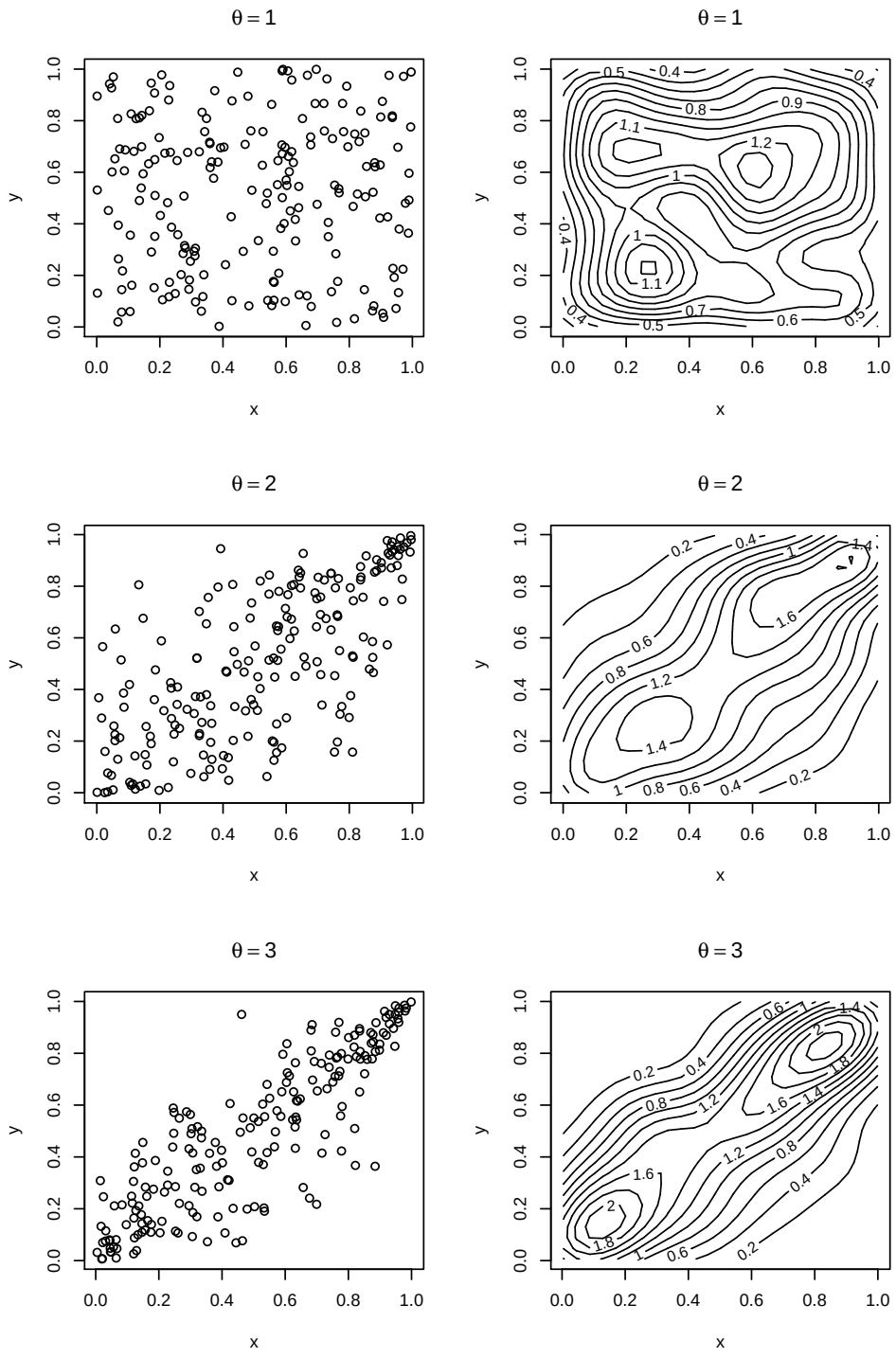

Figure 2: Example scatterplots and estimated densities thereof of Gumbel–Hougaard copulas with $\theta \in \{1, 2, 3\}$.

2.2 Maximum–Likelihood estimation

Parameter estimates of a multivariate distribution constructed from a copula, can be obtained rather easily using Maximum–Likelihood estimation. First, the joint density of such a multivariate distribution (as in theorem 1) is determined by differentiating:

$$\frac{\partial^2 H(x,y)}{\partial x \partial y} = \frac{\partial^2 C\left(F(x), G(y)\right)}{\partial x \partial y} \cdot \frac{\partial F(x)}{\partial x} \cdot \frac{\partial G(y)}{\partial y}$$

Let $h(x,y)$ be the joint density function, $c(u,v)$ the second order derivative of $C(u,v)$ and $f(x), g(y)$ the densities of $F(x), G(y)$ respectively. Then the previous equation can be written more concisely

$$h(x,y) = c\left[F(x), G(y)\right] \cdot f(x)g(y) \ . \tag{2}$$

Then one can formulate the likelihood function:

$$\mathcal{L}(\theta_C, \theta_M | x, y) = \prod_{i=1}^{N} h(x_i, y_i) = \prod_{i=1}^{N} c\left[F(x_i), G(y_i)\right] \cdot f(x_i)g(y_i)$$

where $x = (x_1, \ldots, x_N)$ and $y = (y_1, \ldots, y_N)$ are the data vectors and θ_C is the parameter vector of the copula (or just a scalar for one–parameter copulas) and θ_M the parameter vector of the margins. Consequently, the log–likelihood would then be:

$$\ell(\theta_C, \theta_M | x, y) = \sum_{i=1}^{N} \ln\left(c\left[F(x_i), G(y_i)\right] \cdot f(x_i)g(y_i)\right)$$

Thus one gets the ML–estimator:

$$\hat{\theta}_{\mathrm{ML}} = \arg\max_{\theta} \ \ell(\theta_C, \theta_M | x, y)$$

with $\theta = (\theta_C, \theta_M)$. Estimates can be determined by optimizing the log–likelihood. As a simple extension one can also define the weighted ML–estimator, which will be needed later on

$$\begin{aligned} \hat{\theta}_{\mathrm{WML}} &= \arg\max_{\theta} \ \ell(\theta_C, \theta_M | x, y, w) \\ &= \arg\max_{\theta} \ \sum_{i=1}^{N} w_i \ln\left(c\left[F(x_i), G(y_i)\right] \cdot f(x_i)g(y_i)\right) \end{aligned} \tag{3}$$

where $w = (w_1, \ldots, w_N)$ depicts the vector of weights for each observation. For instance, the weighted ML–estimator for Gumbel–Hougaard copulas is then obtained by inserting its second order derivative (see appendix A.2)

$$\hat{\theta}_{\mathrm{WML}} = \arg\max_{\theta} \sum_{i=1}^{N} w_i \ln \left(\frac{1}{C} \frac{\partial C}{\partial u} \frac{\partial C}{\partial v} \left(1 + (\theta_C - 1)(-\ln C)^{-1} \right) \right)$$

with $C := C_{\theta_C}(F(x_i), G(y_i))$.

3 Model–based clustering

Clustered data can be described in form of a finite mixture model. The whole dataset is said to be composed of several components or clusters, where each component follows the same distribution p whose parameters θ_k vary between components. Also, each component is assigned a prior–probability $\pi_k \geq 0$ with $\sum_{k=0}^{K} \pi_k = 1$. A mixture distribution h with K components is then written as:

$$h(z;\psi) = \sum_{k=1}^{K} \pi_k \; p(z;\theta_k)$$

with $\psi = (\pi_1, \ldots, \pi_K, \theta_1, \ldots, \theta_K)$. In this case p refers to a multivariate joint density constructed from copulas as shown in equation 2. That is:

$$h(x,y;\psi) = \sum_{k=1}^{K} \pi_k \cdot c\left[F(x), G(y)\right] \cdot f(x)g(y) \; .$$

In order to estimate ψ one would try to maximize the corresponding log–likelihood

$$\mathcal{L}(x,y;\psi) = \sum_{i=1}^{N} \ln \left(\sum_{k=1}^{K} \pi_k \cdot c\left[F(x_i), G(y_i)\right] \cdot f(x_i)g(y_i) \right)$$

which is, however, not directly optimizable. Therefore the iterative expectation–maximization algorithm is used, which is implemented by *FlexMix*. Leisch [2004] defines the algorithm as follows:

> **Estimate** the posterior class probabilities for each observation
>
> $$\hat{p}_{nk} = \mathrm{P}(k|x_n, y_n, \hat{\psi})$$
>
> [. . .]
>
> **Maximize** the log-likelihood for each component separately using the posterior probabilities as weights
>
> $$\max_{\theta_k} \sum_{n=1}^{N} \hat{p}_{nk} \log f(y_n|x_n, \theta_k)$$
>
> The E- and M-steps are repeated until the likelihood improvement falls under a a pre-specified threshold or a maximum number of iterations is reached.

For clustering using a specific distribution, one only needs to implement the M-step, which in this case essentially consists of implementing equation 3, therefore providing a weighted ML–estimator for copulas. Appendix B contains the actual implementation.

4 Usage

This section aims to give a short introduction on how to use *FlexMix* for copula clustering using an example with artificial data. First of all one needs to load the *FlexMix* and *copula* packages, which are both available from CRAN, as well as the copula M-step driver available from http://epub.ub.uni-muenchen.de/view/subjects/160103.html (in form of an electronic appendix to this thesis):

```
> library(copula)
> library(flexmix)
> source("flxmccopula.R")
```

The distribution in question, consisting of a copula and its margins, needs to be specified; in this example a two–dimensional Gumbel–Hougaard copula is used with chi–square and poisson distributions as margins:

```
> my.copula <- gumbelCopula(dim = 2, param = 1)
> my.mvdc <- mvdc(copula = my.copula,
+                 margins = c("chisq", "pois"),
+                 paramMargins = list(
+                     list(df = 1),
+                     list(lambda = 1)))
```

A copula object is instantiated using `gumbelCopula()` with dimension 2 and parameter $\theta = 1$, which is passed on to the constructor of the `mvdc` class, which represents a multivariate distribution constructed from a copula. It also takes a vector with the names of the margins (`margins`) and a list of the parameters of the margins (`paramMargins`). Note, that the copula package assembles the names of the respective density, distribution and quantile functions from the entries in `margins`, for instance it expects to find functions named `dchisq`, `pchisq` and `qchisq`. Analogously the elements of `paramMargins` need to be named lists where the names of the list entries equal the parameter names of the corresponding density, distribution and quantile functions (e.g. `dchisq` takes a parameter named `df`).

For this example, two clusters are considered:

Cluster	df	λ	θ
1	2	2	2
2	4	4	1

In order to generate the dataset two copies of `my.mvdc` are created (one for each cluster) with updated parameters[6]. These copies are then used to generate the two clusters of size 100 each[7]:

```
> comp1 <- mvdcSetParam(c(2, 2, 2), my.mvdc)
> comp2 <- mvdcSetParam(c(4, 4, 1), my.mvdc)
> data <- rbind(rmvdc(comp1, n = 100), rmvdc(comp2, n = 100))
```

Estimates are then obtained by calling `flexmix()` on the dataset:

```
> m <- flexmix(data ~ 1, k = 2,
+              model = FLXMCcopula(mvdc = my.mvdc))
> m


Call:
flexmix(formula = data ~ 1, k = 2, model = FLXMCcopula(mvdc = my.mvdc))

Cluster sizes:
  1   2
 96 104

convergence after 30 iterations
```

The parameter $k = 2$ tells *FlexMix* to try two estimate two clusters. The result can however contain less clusters if they are not sufficiently distinguishable. For a reference `mvdc` instance, `my.mvdc` is supplied, whose initial parameters are used as starting value for the optimizer, since it is not trivial to find these automatically. In case, `optim` reports an error stating the likelihood of the initial values could not be evaluated, the parameters need to be re–adjusted and the call to `flexmix` needs to be repeated. This usually happens if either one or more parameter value lies outside their supports, or the likelihood at that point is numerically `-Inf`.

[6] See appendix B.1 for function `mvdcSetParam`

[7] The example uses `set.seed(42)` in case one wants to reproduce the results.

After the estimation is done one might want to examine the original and estimated cluster assignments (figure 3):

```
> orig <- c(rep.int(1, 100), rep.int(2, 100))
> est <- clusters(m)
> par(mfrow = c(1, 2))
> plot(data[, 1], data[, 2], col = orig, pch = 1 + orig)
> plot(data[, 1], data[, 2], col = est, pch = 1 + est)
```

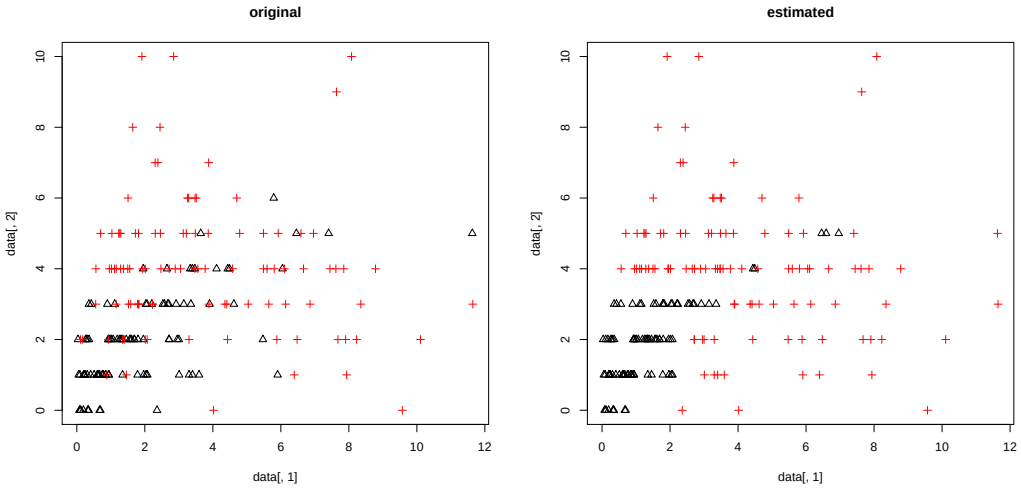

Figure 3: Original and estimated cluster assignments.

One can further examine how many observations were (in)correctly classified:

```
> table(orig, clusters(m))
```

```
    est
orig  1  2
   1 76 24
   2 20 80
```

Both clusters were assigned most of their original observations, only 20 and 24 were incorrectly assigned to the other cluster respectively. With a total of 156 correctly classified observations this yields a ratio of 78% correctly classified observations.

The estimated parameters can be obtained with `parameters()` which returns a list of `mvdc` objects, one for each estimated component. The parameter values can be extracted using `mvdcGetParam` (see appendix B.1):

```
> lapply(parameters(m), function(x) mvdcGetParam(x))

$Comp.1.mvdc
      df   lambda
1.483955 2.159034 2.292523

$Comp.2.mvdc
       df    lambda
4.0176387 3.7696889 0.9500256
```

The parameter values lie within the area of the orginal ones and both components can be recognized from the parameter values alone.

5 Simulations

A number of simulations was run in order to see how well this implementation for copula clustering performs. For this, datasets were generated given a certain copula and margins and different parameters per cluster using the `rmvdc` function from the `copula` package. Then parameter estimates were obtained using *FlexMix*.

Primarily the difference between the original and estimated parameters in each cluster or component is considered. However, "[m]ixture models are only identifiable up to a permutation of the component labels" (Leisch [2004]). That is, the original ordering of the clusters might not be preserved during the estimation step, and for example "Cluster 1" might then be called "Cluster 2". To compare the parameters of corresponding clusters nonetheless, one would have to determine a plausible permutation of the resulting cluster labels, which, in this case, was done by minimizing the sum of the euclidean distances between original and estimated parameter vectors for all clusters:

$$d(\theta^{(O)}, \theta^{(E)}) := \min_{\sigma_i} \sum_{k=1}^{K} \left\| \theta^{(O)}_{\sigma_i(k)}, \theta^{(E)}_{\sigma_i(k)} \right\|_2 \tag{4}$$

where $\theta^{(O)}$ and $\theta^{(E)}$ are original and estimated parameter vectors respectively. σ is the set of all possible permutations, σ_i is the i–th permutation and $\sigma_i(k)$ is then the i–th permutation of k. The distance $d(\theta^{(O)}, \theta^{(E)})$ is considered to be a metric for the difference between the original and estimated parameters.

The underlying copula of all of the following simulations was the two–dimensional Gumbel–Hougaard copula. Also, in about 0.7% of the cases *FlexMix* estimated a lesser number of clusters, than there were originally specified. These models were excluded from the following analysis.

5.1 Simulation 1

For the first three simulations the estimation error for increasing sample sizes was examined. The simulation dataset consisted of two clusters in each case, and the sample size was increased in multiples of 50 starting with 50 up to 500 observations per cluster. Both margins were exponential and the parameters for each cluster were:

Cluster	θ	λ_1	λ_2
1	2	2	3
2	1	4	5

Figure 4 shows an example scatter plot with 200 observations per cluster, as well as the corresponding estimated cluster assignments. Figure 5 shows a

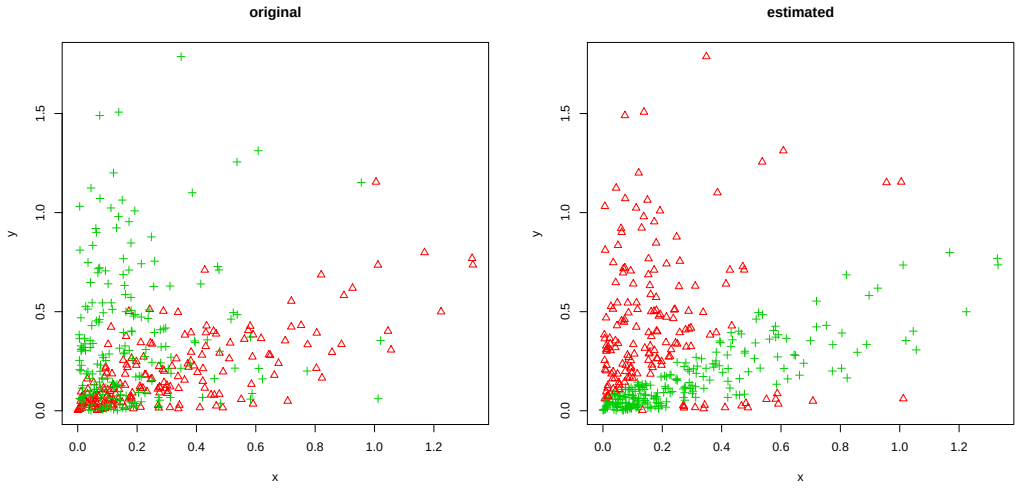

Figure 4: Original vs estimated cluster assignments

plot of the mean distances between original and estimated parameters per total sample size on the left hand side and a plot of the mean ratio of correctly classified observations on the right hand side. Also, the estimated standard deviations were plotted around the means as a symmetric interval using dotted lines. As one would suspect, the estimation error and its deviation decreases with increasing sample sizes. The ratio of correct classifications however only shows slight improvements, which could be attributed to the relatively large overlap area shown in the previous scatter plots. It should be noted though that the ratio starts off at a quite high value of about 60%.

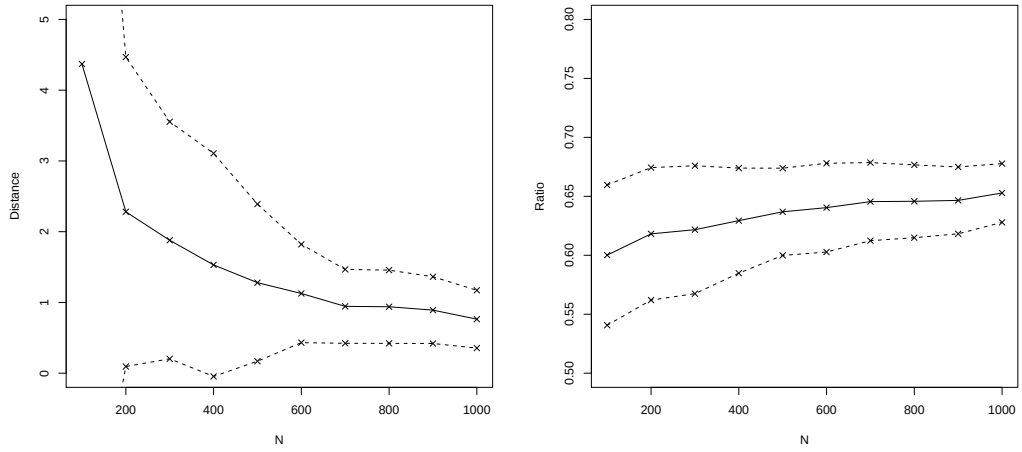

Figure 5: Distances and ratio of correctly classified observations

5.2 Simulation 2

The second simulation was run under the same circumstances as the first, only with different parameters (see the following table). Again the estimation error and the variance improve with increasing sample size while the ratio of correct classifications remains almost the same (figures 6 and 7).

Cluster	θ	λ_1	λ_2
1	2	3	5
2	1	5	3

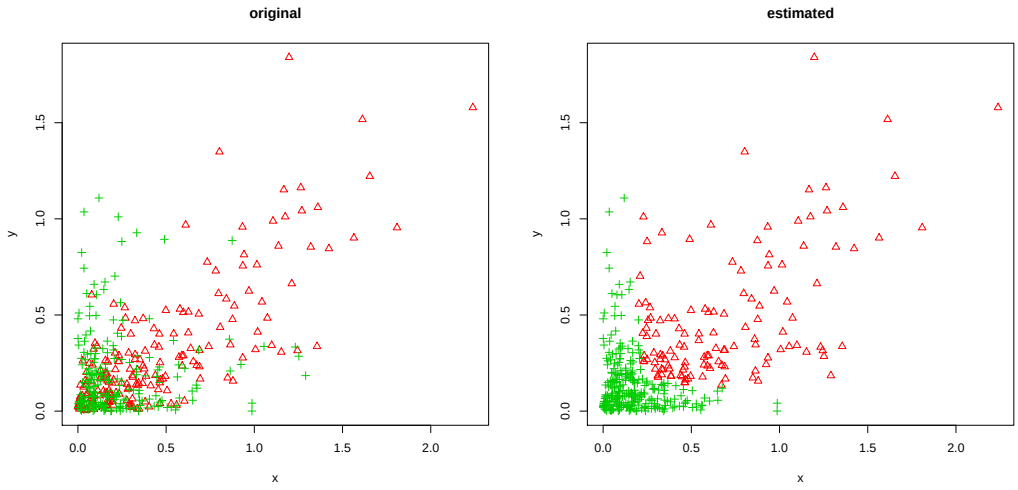

Figure 6: Original vs estimated cluster assignments

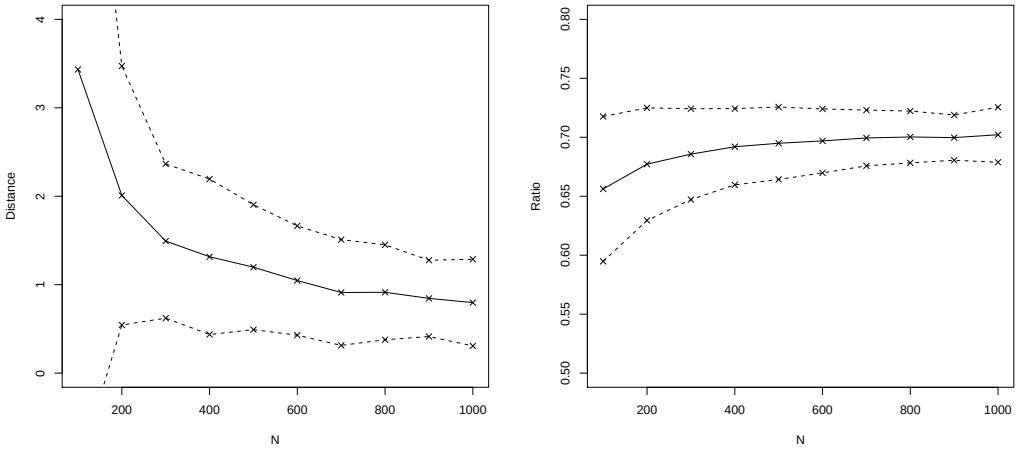

Figure 7: Distances and ratios of correctly classified observations

5.3 Simulation 3

As a variation of the above simulations, the margins were changed to follow a gamma–distribution (see table below for parameters). The mean estimation error declines here as well with increasing sample size. The estimation error seems to be a bit larger in general, as opposed to the previous simulations, which could be attributed to the extra number of parameters and/or the change in marginal distributions (figures 8 and 9).

Cluster	θ	shape1	scale1	shape2	scale2
1	2	2	4	4	5
2	1	5	4	3	4

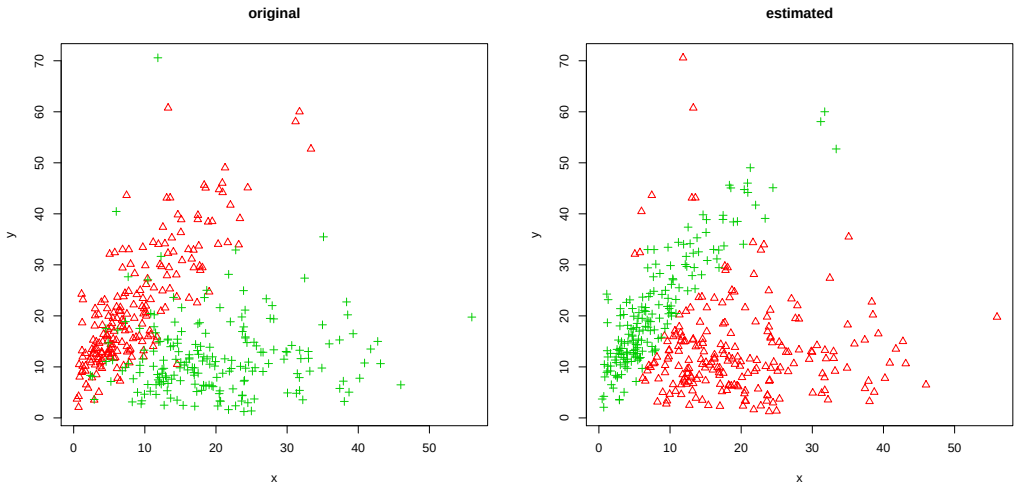

Figure 8: Original vs estimated cluster assignments

5.4 Simulation 4

For the fourth simulation the variation of the parameter θ of the copula was examined. The same margins and parameters as in simulation 2 were used, with $\theta_i \in \{1, 2, 3, 4, 5\}$ and $\theta_1 = \theta_2$. While the estimation error shows no discernible improvement for $\theta \geq 2$, the classification ratio continually increases. Figure 11 shows example scatter plots; for larger values of θ both components become better separated thus explaining the improvement of the classification ratio (figure 10).

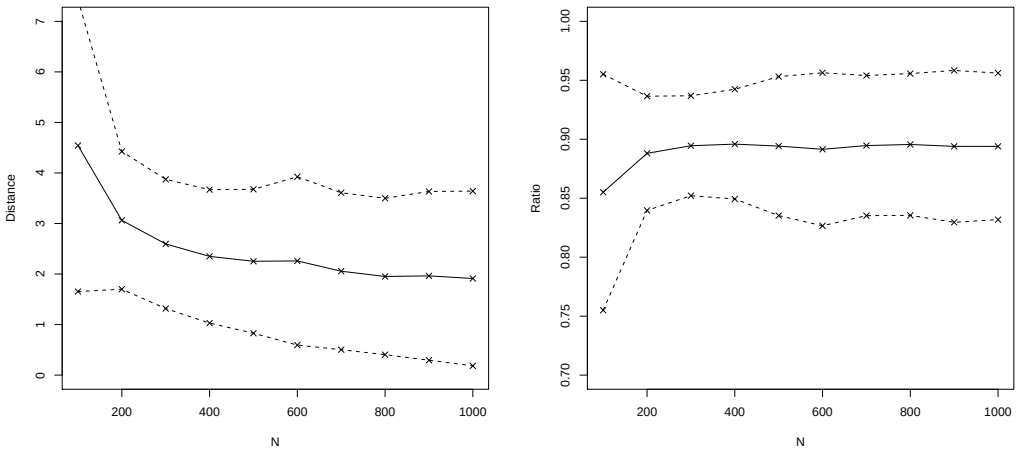

Figure 9: Distances and ratios of correctly classified observations

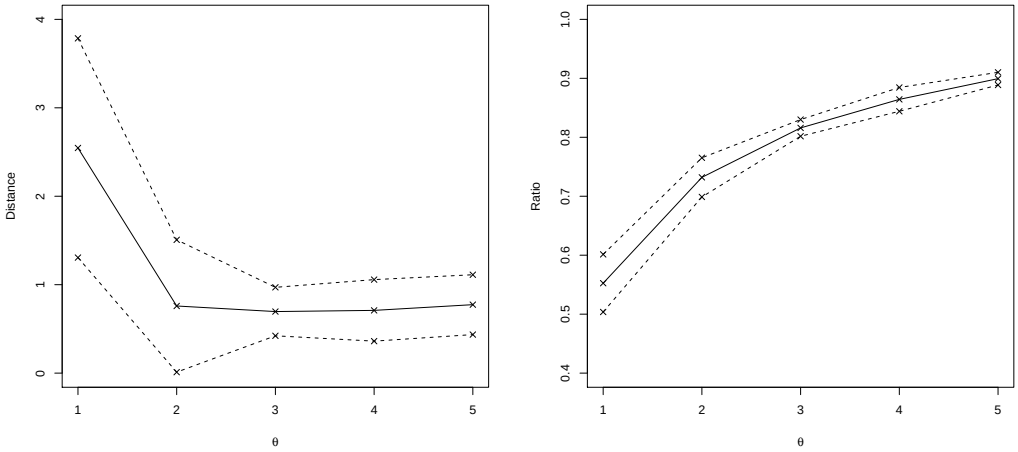

Figure 10: Distances and ratios of correctly classified observations

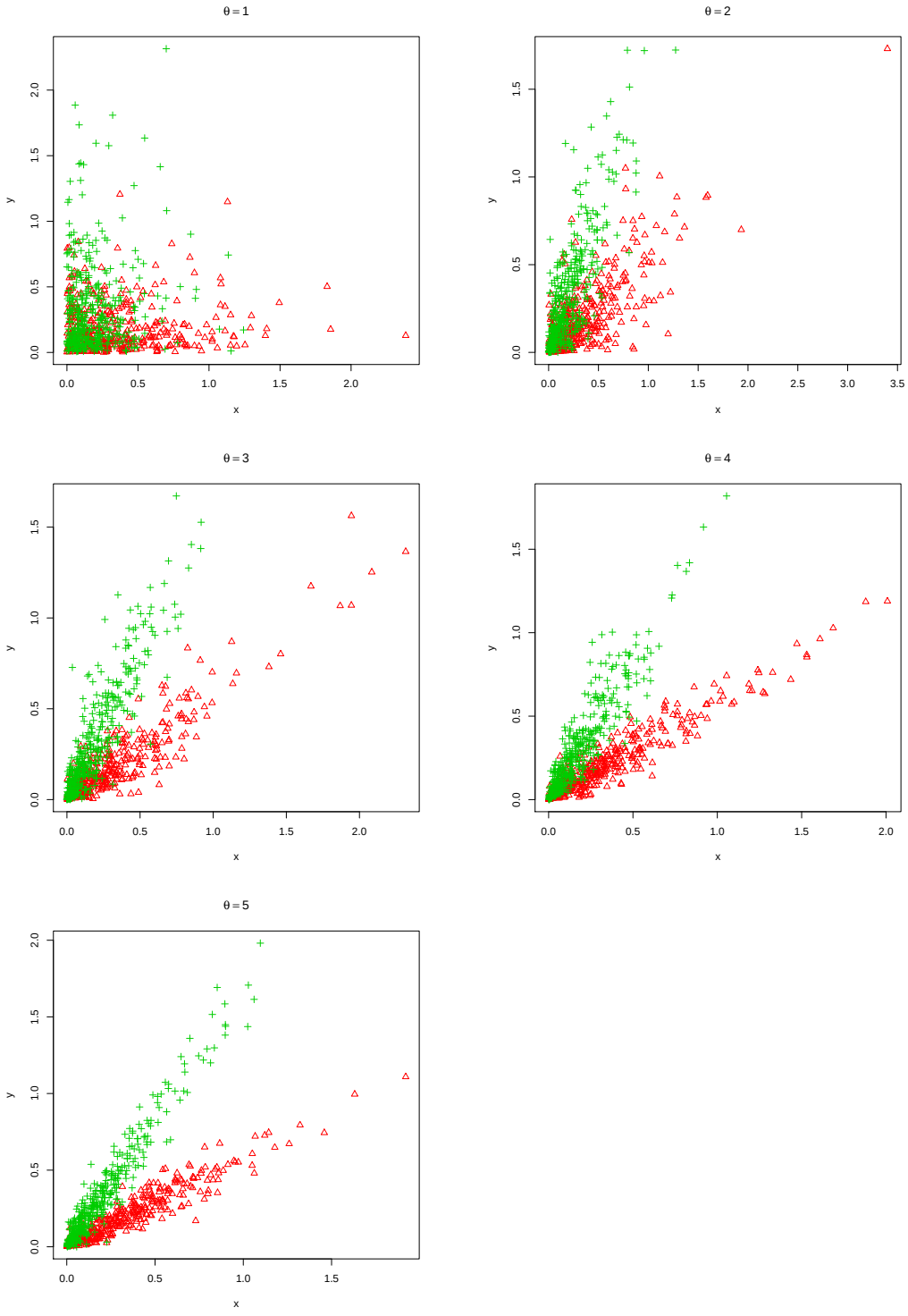

Figure 11: Example scatter plots for different values of θ

5.5 Simulation 5

For the last simulation datasets with varying amounts of clusters, i.e. two to five, were generated. The simulation specified a total of five clusters with exponential margins (see table below) from which $n \in \{2, 3, 4, 5\}$ were randomly chosen without replacement.

Cluster	θ	λ_1	λ_2
1	2	1	5
2	2	2	4
3	2	3	3
4	2	4	2
5	2	5	1

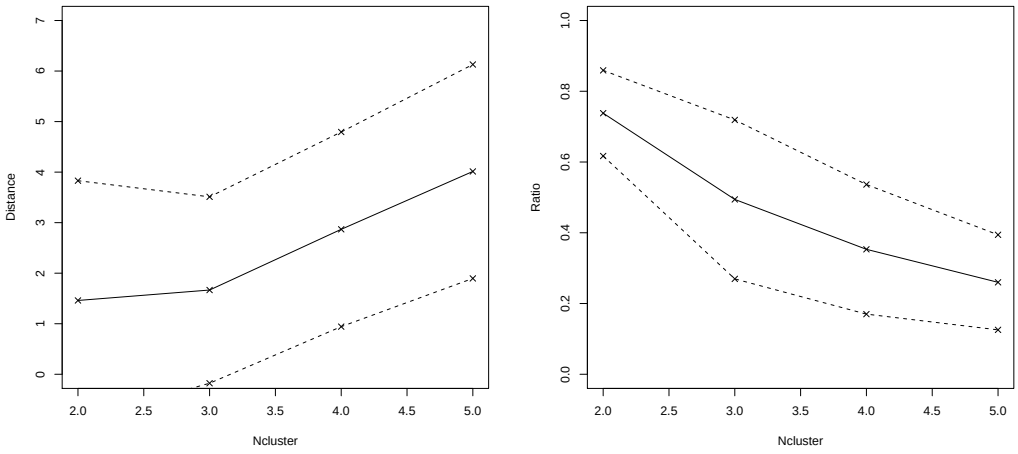

Figure 12: Distances and ratios per amount of clusters

Judging from figure 12 the estimation error increases with the amount of clusters and the ratio of correctly classified observations decreases. Also see figure 13 for example scatter plots with 2–4 clusters.

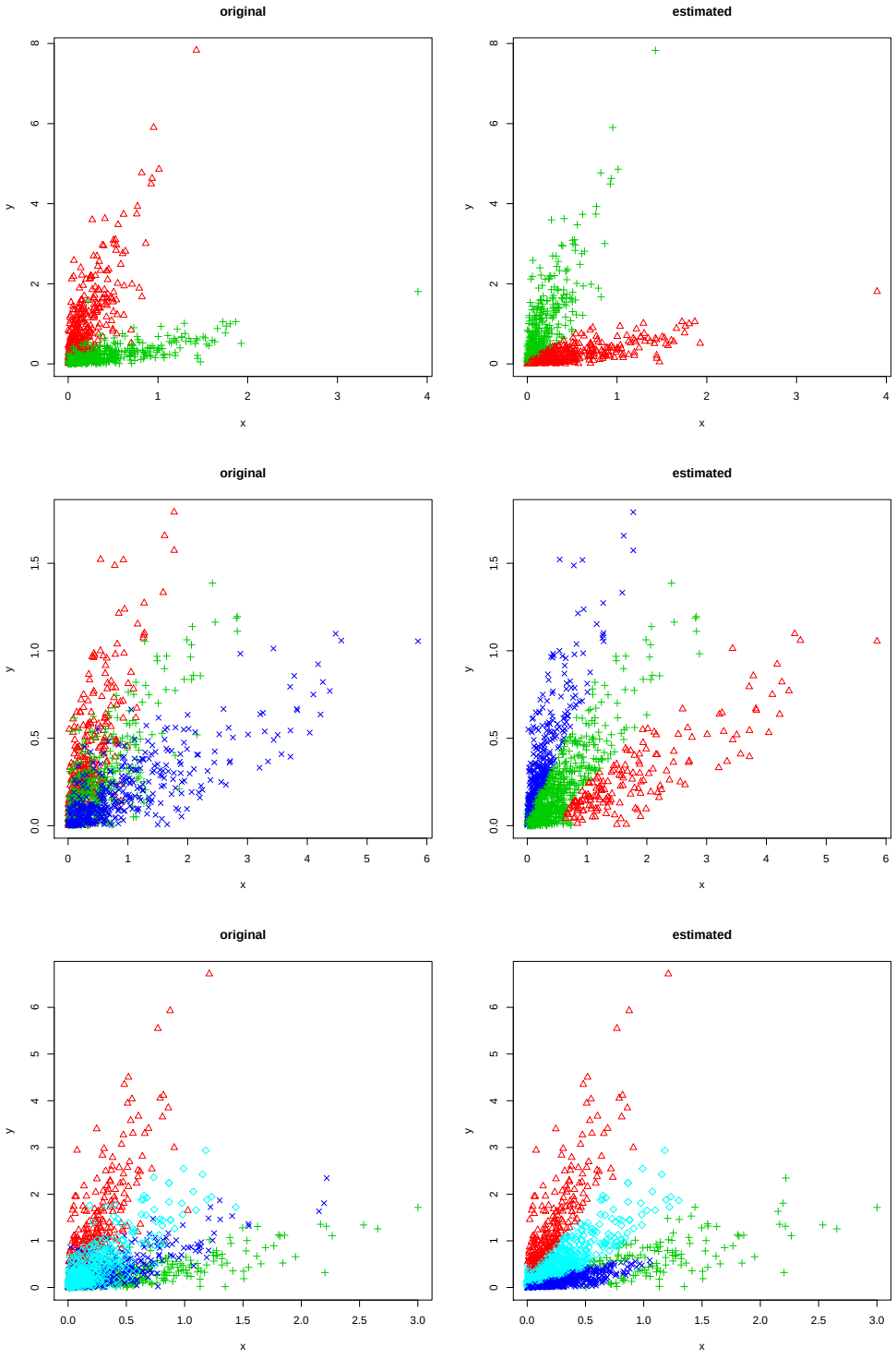

Figure 13: Original vs estimated cluster assignments

6 Conclusion

The objective of the thesis was to develop an M–step driver for copula clustering with *FlexMix*. For this, a brief overview over the theoretical background, i.e. copulas, ML-estimation with copulas and finite mixture models, was given in sections 2 and 3. The implementation itself then mostly consisted of implementing the weighted ML–estimator (equation 3) using the *copula* package and writing the necessary boiler plate code for interfacing with *FlexMix*. As for the type of copulas and margins that can be utilized, no further restrictions had to be made, and as such, all the copulas supplied by the *copula* package should be usable. Similarly any marginal distribution, for which maximum likelihood estimation is applicable and the density, distribution and quantile functions are known, or can be estimated, should be usable as well.

In section 5 a set of simple scenarios was simulated. The first three simulations showed, while varying the margins, that the estimation error shrunk for increasing sample sizes. Clusters that contained highly correlated data, as seen in simulation 4, were also better estimated, presumably because of the higher degree of separation between the components. For larger amounts of clusters the estimation error increased (see simulation 5), since more clusters, in this setting, also caused bigger overlaps of observations between the components, which was particularly well visible with 4 or more clusters. The ratio of correctly classified observations in generally depended on the overlap area in the simulated datasets. Less overlap usually resulted in a higher ratio and vice versa.

While being able to actually fit a model to data, one would require a way of assessing the model fit in practice. For copulas, a simple graphical method described in Genest and Favre [2007] would be to compare the empirical data to artificial data generated from the assumed copula and estimated parameters thereof. One could then use Q–Q–plots or K–plots (described in Genest and Favre [2007] as well) for comparison between observed and generated data. This approach could probably be extended to be used along with mixture models. On the other hand there are formal goodness–of–fit tests being discussed. Genest et al. [2009] propose the use of "blanket tests", i.e. tests that neither "require an arbitrary categorization of the data nor any strategic choice of smoothing parameter, weight function, kernel, window". The corresponding p–values are then computed using parametric bootstrap. Kojadinovic et al. [in press] implemented a faster version of this test procedure for the *copula* package. Whether this test can be used for mixture models with copulas could be worth of an investigation.

A Proofs

A.1 $\varphi_\theta(t) = (-\ln t)^\theta$ is a valid generator function

φ_θ needs to satisfy the conditions from theorem 2.

1. φ_θ is continuous, since $\ln t$ is continuous and so is $-\ln t$ and $(-\ln t)^\theta$.
2. φ_θ is strictly decreasing:

$$\frac{\partial \varphi_\theta}{\partial t} = \theta(-\ln t)^{\theta-1}(-\frac{1}{t}) = \theta\frac{(-\ln t)^\theta}{t \ln t} < 0$$

   for $\theta > 0$ and $t \in [0, 1)$ ($\Rightarrow \ln t < 0$).

3. φ_θ is convex:

$$\begin{aligned}\frac{\partial^2 \varphi_\theta}{\partial t^2} &= \frac{\partial}{\partial t}\frac{\theta(-\ln t)^\theta}{t \ln t} = \frac{\theta^2(-\ln t)^\theta - \theta(-\ln t)^\theta(\ln t + 1)}{(t \ln t)^2} = \\ &= \frac{\theta(-\ln t)^\theta\,(\theta - \ln t - 1)}{(t \ln t)^2}\end{aligned}$$

   Then $\frac{\partial^2 \varphi_\theta}{\partial t^2}$ is positive if $\theta(\theta - \ln t - 1) > 0$ and thus if $\theta \geqslant 1$.

A.2 Derivatives of Gumbel–Hougaard copulas

The derivates of Gumbel–Hougaard copulas can be determined as follows (see [Frees and Valdez, 1998, Appendix A]). First $C := C_\theta(u, v)$ is defined for better readability. Then one might want to reorder equation 1:

$$\begin{aligned} C &= exp\left(-\left[(-\ln u)^\theta + (\ln v)^\theta\right]^{1/\theta}\right) \\ (-\ln C)^\theta &= (-\ln u)^\theta + (-\ln v)^\theta \end{aligned}$$

Derive both sides with respect to u and reorder:

$$\begin{aligned} \theta(-\ln C)^{\theta-1}\left(-\frac{1}{C}\right)\frac{\partial C}{\partial u} &= \theta(-\ln u)^{\theta-1}\left(-\frac{1}{u}\right) \\ \frac{(-\ln C)^{\theta-1}}{C}\frac{\partial C}{\partial u} &= \frac{(-\ln u)^{\theta-1}}{u} \\ \frac{\partial C}{\partial u} &= \frac{(-\ln u)^{\theta-1}}{u}\frac{C}{(-\ln C)^{\theta-1}} \\ \frac{\partial C}{\partial u} &= \left(\frac{\ln u}{\ln C}\right)^{\theta-1}\frac{C}{u} \end{aligned}$$

From symmetry one obtains:

$$\frac{\partial C}{\partial v} = \left(\frac{\ln v}{\ln C}\right)^{\theta-1}\frac{C}{v}$$

The second–order derivative then is rather straight–forward:

$$\begin{aligned} \frac{\partial^2 C}{\partial u \partial v} &= \frac{\partial}{\partial v}\left(\frac{\ln u}{\ln C}\right)^{\theta-1}\frac{C}{u} \\ &= \left(\frac{\ln u}{\ln C}\right)^{\theta-1}\frac{1}{u}\frac{\partial C}{\partial v} - \frac{1}{u}(\theta-1)\left(\frac{\ln u}{\ln C}\right)^{\theta-2}\frac{\ln u}{(\ln C)^2}\frac{\partial C}{\partial v} \\ &= \left(\frac{\ln u}{\ln C}\right)^{\theta-1}\frac{1}{u}\frac{\partial C}{\partial v}\left[1 - (\theta-1)\frac{\ln u}{(\ln C)^2}\left(\frac{\ln C}{\ln u}\right)\right] \\ &= \frac{1}{C}\frac{\partial C}{\partial u}\frac{\partial C}{\partial v}\left(1 - (\theta-1)\frac{1}{\ln C}\right) \\ &= \frac{1}{C}\frac{\partial C}{\partial u}\frac{\partial C}{\partial v}\left(1 + (\theta-1)(-\ln C)^{-1}\right) \end{aligned}$$

B Implementation

B.1 Utility functions

The *copula* package defines a class `mvdc`, which is used to describe multivariate distributions constructed from copulas. This class for one contains an instance of the `copula` class, describing the underlying copula as well as its parameter(s). On the other hand it provides a generic way to describe the marginal distributions and its parameters. The M–step driver for copula clustering makes extensive use of the `mvdc` class, therefore a set of utility functions was introduced to facilitate exchanging parameter values with an `mvdc` instance.

Listing 1: mvdc(Get|Set)Param

```
1  mvdcGetParam <-
2  function(mvdc) {
3    mpar <- unlist(mvdc@paramMargins)
4    dpar <- mvdc@copula@parameters
5
6    c(mpar, dpar)
7  }
8
9  mvdcSetParam <-
10 function(param, mvdc) {
11   poff <- 1
12
13   for(i in 1:length(mvdc@paramMargins)) {
14     for(j in 1:length(mvdc@paramMargins[[i]])) {
15       mvdc@paramMargins[[i]][j] <- param[poff]
16       poff <- poff + 1
17     }
18   }
19
20   for(i in 1:length(mvdc@copula@parameters)) {
21     mvdc@copula@parameters[i] <- param[poff]
22     poff <- poff + 1
23   }
24
25   mvdc
26 }
```

The function `mvdcGetParam` (lines 1–7) takes an `mvdc` instance as parameter and returns a vector of the parameters extracted from the `mvdc` instance.

By convention, the vector first contains the parameters of the margins, then the parameters of the copula. The `mvdc` class stores the parameters of the margins as a list of lists: the "outer" list just contains one entry per margin, while the "inner" lists contain the parameters of each margin as a named list, where the key corresponds to the name of the parameter, and the value is the respective value of that parameter. Extracting the parameter values of the margins is then easily done using `unlist` (line 3) while the parameters of the copula can be directly accessed using the appropriate slot (line 4).

`mvdcSetParam` (lines 9–26) takes an `mvdc` instance and a vector of parameters as arguments and returns a copy of this `mvdc` instance, containing the given parameters. Then it is iterated over the elements of the list–of–lists containing the parameters of the margins (lines 13–18), while counting the position in the parameter vector (`poff`). Each iteration step stores one element from the parameter vector. Analogously the parameters of the copula are being copied to the `copula` instance (lines 20–23). The resulting `mvdc` instance is then returned (line 25).

B.2 Weighted Log–Likelihood

The function `copLogLik` implements the log–likelihood from equation 3; it takes a vector of parameters `param`, a matrix of observations `x`, an `mvdc` instance and a vector weights `w` for arguments and returns the log–likelihood for the given observations and parameters. The matrix `x` contains one row per observation and one column for each margin. The vector `w` usually contains one weight per observation, but by default uses `1` for each observation thus resulting in the ordinary log–likelihood.

Listing 2: copLogLik

```
1  copLogLik <-
2  function(param, x, mvdc, w=1) {
3    ll <- -Inf
4
5    mvdc <- mvdcSetParam(param, mvdc)
6
7    tryCatch({
8        dens <- dmvdc(mvdc, x)
9
10       if(any(is.na(dens)))
11         return(-Inf)
12       if(any(dens <= 0))
13               return(-Inf)
14
```

```
15        ll <- sum(w * log(dens))
16      }
17    )
18
19    return(ll)
20  }
```

First, the parameters of the given `mvdc` instance are updated to the position where the log–likelihood is to be evaluated (line 5). Then, an attempt is made to calculate the densities for each observation in `x` (line 8) using `dmvdc` from the `copula` package.

The call to `dmvdc` may cause an error, usually because the optimizer tried to obtain the log–likelihood for an invalid parameter combination, for example where at least one parameter lies outside its domain (simple examples would be a negative variance for normal distributions or a negative rate parameter for exponential distributions). The optimizer just assumes each parameter lives on the real line and is not told otherwise[8]. Instead, eventual errors are intercepted with `tryCatch` and `-Inf` is returned. Additionally, if `dmvdc` did not fail but returned densities which are either `NA`, negative or equal to zero, `-Inf` is returned as well (lines 10–13) because the following call to `log` would cause an error.

If no error occured, the log–likelihood is calculated and returned (lines 15 and 19).

B.3 ML–estimator

The function `copMLE` implements the Maximum–Likelihood estimator from equation 3; it takes a matrix of observations `x`, an `mvdc` instance and a vector of weights as parameters and returns an `mvdc` instance with parameters set to the coordinates of a maximum. This function is essentially a wrapper for `optim`, which in turn requires a starting point for its optimization. Since it is not trivial to automatically determine such a starting point, the parameters set in the `mvdc` instance are used. Therefore the user needs to initialize this instance with reasonable defaults.

The parameters `x`, `mvdc` and `w` are then just passed on to `copLogLik` via `optim`.

[8] It might in theory be possible to use an optimizer with box constraints, such as `L-BFGS-B`, which would however require the task of specifying these constraints by hand from the user. Therefore it seemed easier and more practical to just accept anything as input and handle error case appropriately.

Listing 3: copMLE

```
copMLE <-
function(x, mvdc, w=1, method="Nelder-Mead") {
  st <- mvdcGetParam(mvdc)

  ret <- optim(st, copLogLik,
               x=x, mvdc=mvdc, w=w,
               method=method,
               control=list(fnscale=-1))
  mvdcSetParam(ret$par, mvdc)
}
```

B.4 FlexMix–Interface

Finally, `FLXMCcopula` implements the M–step driver for FlexMix. It takes a reference `mvdc` instance, which was created and initialized by the user as parameter and returns a `FLXMC` instance which is then used by FlexMix.

Listing 4: FLXMCcopula

```
FLXMCcopula <-
function(formula=.~., mvdc=NULL) {
  z <- new("FLXMC",
           weighted=TRUE,
           formula=formula,
           dist="copula",
           name="model-based␣copula␣clustering")

  if(is.null(mvdc))
    stop("mvdc␣object␣missing")

  z@defineComponent <- expression({
    logLik <- function(x,y) {
      log(dmvdc(mvdc, y))
    }

    predict <- function(x, ...) {
      stop("predict()␣not␣implemented")
    }

    new("FLXcomponent",
        parameters=list(mvdc=mvdc),
        df=1,
```

```
24          logLik=logLik,
25          predict=predict)
26    })
27
28
29    z@fit <- function(x, y, w) {
30      mvdc <- copMLE(y, mvdc, w)
31
32      para <- list(mvdc=mvdc)
33      with(para, eval(z@defineComponent))
34    }
35
36    z
37  }
```

At first, an `FLXMC` object is instantiated and initialised with the formula parameter (line 5) and the notion that weighted estimation is being used (line 4) as well as some metadata (lines 6–7).

Next, some slots of the newly–created `FLXMC` instance have to be initialised; the `fit` slot is assigned a wrapper function around `copMLE` that returns an instance of `FLXcomponent` (lines 29–34). The parameters `x` and `y` correspond to the respective values in the formula parameter earlier. `defineComponent` then stores a function that creates an `FLXcomponent` object and expects to find an `mvdc` instance in its environment that contains the parameters estimated in `fit` (lines 12–26). Also, the `FLXcomponent` object is supplied with functions `logLik` and `predict`. The former computes the log–likelihood per observation, while the latter was left unimplemented. `predict` would usually predict the target variable given a dataset, in this case however only clustering was of interest and since there is no target variable, prediction would not have been very meaningful.

The end user would then usually call `flexmix()` with the `FLXMC` object and thus obtain an object of class `flexmix` which encodes the estimated model. `parameters()` returns a list of `mvdc` instances, one per estimated component, where the parameters of each component can be extracted with `mvdcGetParam`.

C Simulation Code

This section contains the code that was used to run the simulations from section 5. The underlying idea was to do the simulations on a number of computers, which would each repeatedly run one simulation and store the results locally in a file. In each simulation run, one kind of a simulation and a corresponding parameter (e.g. cluster size in simulations 1–3) was chosen randomly, so there would not have to be any coordination between the computers. Also, this way simulations could be terminated while only affecting the current model and one did not have to wait until a set of simulations was complete. The data files were then (incrementally) copied to a central location where the results were post–processed (such as calculating the parameters' estimation error) and stored in a more convenient format that can be read by `read.table`.

First off a couple of utility functions; `cluster.distance` implements the sum part of equation 4. `origParam` and `estParam` are each lists of parameter vectors (original and estimated respectively). `perm` is a vector denoting the permutation that should be applied to the cluster labels.

Listing 5: cluster.distance

```
1 cluster.distance <-
2 function(origParam, estParam, perm) {
3   sm <- 0
4   for(i in 1:length(origParam)) {
5     dif <- origParam[[i]] - estParam[[ perm[i] ]]
6     sm <- sm + sqrt(sum(dif*dif))
7   }
8   return(sm)
9 }
```

`cluster.order` tries to find the permutation that minimizes the sum in equation 4. It determines all possible permutations using the equally named function `permutations` from the `gtools` package, calculates the estimation difference for each permutation and then returns the permutation with the least difference.

Listing 6: cluster.order

```
1 cluster.order <-
2 function(origParam, estParam) {
3   stopifnot(length(origParam) == length(estParam))
4
5   lngth <- length(origParam)
```

```
6     mperm <- permutations(lngth,lngth)
7     perm <- lapply(seq.int(nrow(mperm)),
8                    function(x) mperm[x,])
9
10    diflist <- double(length(perm))
11
12    for(i in 1:length(perm)) {
13      diflist[i] <- cluster.distance(origParam,
14                                     estParam,
15                                     perm[[i]])
16    }
17
18    return(perm[[which.min(diflist)]])
19  }
```

`cluster.classification` determines the amount of correctly classified observations given the permutation `perm`. `orig` and `est` are each vectors where the i–th entry contains the assigned cluster number of the i–th observation.

Listing 7: cluster.classification

```
1  cluster.classification <-
2  function(orig, est, perm) {
3    est <- sapply(est, function(x) perm[x])
4    Ncorrect <- length(which(orig == est))
5    return(Ncorrect/length(orig))
6  }
```

The following five functions construct a number of objects needed for a certain kind of simulation without actually running the simulation; that is, a reference `mvdc` instance is created, containing the starting values for the optimizer. Then for each cluster an `mvdc` instance of the same structure is created, with the cluster's parameters, as well as a vector containing the sample sizes of each cluster. Each of these functions represent one simulation type from section 5.

Listing 8: sim.exp2345

```
1  sim.exp2345 <-
2  function(n=NULL) {
3    if(is.null(n))
4      n <- sample(x=c(1:10 * 50), size=1)
5
```

```
  my.mvdc <- mvdc(copula = gumbelCopula(param=1,dim=2),
                  margins  = c("exp", "exp"),
                  paramMargins = list(
                      list(rate=1),
                      list(rate=1)
                  )
  )

  clusterList <- c(
    mvdcSetParam(c(2,3, 2), my.mvdc),
    mvdcSetParam(c(4,5, 1), my.mvdc)
  )

  clusterSizes <- c(n, n)

  list(
    name         = "exp2345",
    clusterList  = clusterList,
    clusterSizes = clusterSizes,
    startMvdc    = my.mvdc
  )
}
```

Listing 9: sim.exp3553

```
sim.exp3553 <-
function(n=NULL) {
  if(is.null(n))
    n <- sample(x=c(1:10 * 50), size=1)

  my.mvdc <- mvdc(copula = gumbelCopula(param=1,dim=2),
                  margins  = c("exp", "exp"),
                  paramMargins = list(
                      list(rate=1),
                      list(rate=1)
                  )
  )

  clusterList <- c(
    mvdcSetParam(c(3,5, 2), my.mvdc),
    mvdcSetParam(c(5,3, 1), my.mvdc)
  )

  clusterSizes <- c(n, n)
```

```
20
21   list(
22     name         = "exp3553",
23     clusterList  = clusterList,
24     clusterSizes = clusterSizes,
25     startMvdc    = my.mvdc
26   )
27 }
```

Listing 10: sim.gamma24455434

```
1  sim.gamma24455434 <-
2  function(n=NULL) {
3    if(is.null(n))
4      n <- sample(x=c(1:10 * 50), size=1)
5
6    my.mvdc <- mvdc(copula = gumbelCopula(param=1,dim=2),
7                    margins  = c("gamma", "gamma"),
8                    paramMargins = list(
9                        list(shape=2, scale=2),
10                       list(shape=2, scale=2)
11                   )
12   )
13
14
15   clusterList <- c(
16         mvdcSetParam(c(2,4, 4,5, 2), my.mvdc),
17         mvdcSetParam(c(5,4, 3,4, 1), my.mvdc)
18   )
19
20   clusterSizes <- c(n, n)
21
22   list(
23     name         = "gamma24455434",
24     clusterList  = clusterList,
25     clusterSizes = clusterSizes,
26     startMvdc    = my.mvdc
27   )
28
29 }
```

Listing 11: sim.theta12345

```
1  sim.theta12345 <-
```

```
function(theta=NULL) {
  n <- 400

  if(is.null(theta))
    theta <- sample(seq.int(5), size=1)

  my.mvdc <- mvdc(copula = gumbelCopula(param=1.5,dim=2),
                  margins  = c("exp", "exp"),
                  paramMargins = list(
                      list(rate=1),
                      list(rate=1)
                  )
  )

  clusterList <- c(
    mvdcSetParam(c(3,5, theta), my.mvdc),
    mvdcSetParam(c(5,3, theta), my.mvdc)
  )

  clusterSizes <- c(n, n)

  list(
    name         = "theta12345",
    clusterList  = clusterList,
    clusterSizes = clusterSizes,
    startMvdc    = my.mvdc
  )

}
```

Listing 12: sim.Ncluster

```
sim.Ncluster <-
function(n=NULL) {
  if(is.null(n)) {
    n <- sample(1+seq.int(4), size=1)
  }

  my.mvdc <- mvdc(copula = gumbelCopula(param=1.5,dim=2),
                  margins  = c("exp", "exp"),
                  paramMargins = list(
                      list(rate=1.5),
                      list(rate=1.5)
                  )
```

```
  )

  theta <- 2

  clusterList <- c(
    mvdcSetParam(c(1,5, theta), my.mvdc),
    mvdcSetParam(c(2,4, theta), my.mvdc),
    mvdcSetParam(c(3,3, theta), my.mvdc),
    mvdcSetParam(c(4,2, theta), my.mvdc),
    mvdcSetParam(c(5,1, theta), my.mvdc)
  )

  clusterList <- sample(clusterList, size=n, replace=F)

  clusterSizes <- rep(400, n)

  list(
    name         = "Ncluster",
    clusterList  = clusterList,
    clusterSizes = clusterSizes,
    startMvdc    = my.mvdc
  )
}
```

`sim.once` runs just one simulation, calling one of the functions above. It calls one of the functions above, generates the dataset and original cluster assignments. Then `flexmix` is called and the result stored along with some metadata.

Listing 13: sim.once

```
sim.once <-
function(simname=NULL, n=NULL) {
  if(is.null(simname)) {
    simList <- c("sim.exp3553", "sim.exp2345",
                 "sim.gamma24455434",
                 "sim.theta12345", "sim.Ncluster")
    sim <- sample(simList, size=1)
  } else {
    sim <- paste("sim.", simname, sep="")
  }

  ret <- eval(call(sim, n))

```

```
14  simName      <- ret$name
15  clusterList  <- ret$clusterList
16  clusterSizes <- ret$clusterSizes
17  startMvdc    <- ret$startMvdc
18  Ncluster     <- length(clusterList)
19
20  fname <- sprintf("data/sim%d_%d",
21                   as.integer(Sys.time()),
22                   as.integer(runif(1,1,9999999)))
23  outfname <- paste(fname, ".Rout", sep="")
24  fname <- paste(fname, ".RData", sep="")
25
26
27  N <- sum(clusterSizes)
28
29  cat("Name:", simName, "\n")
30  cat(class(startMvdc@copula), "dim␣=",
31      startMvdc@copula@dimension, "␣param␣=",
32      startMvdc@copula@parameters, "\n")
33  cat("Margins:", startMvdc@margins, "\n")
34  cat(N, ":", clusterSizes, "\n")
35  cat("\n")
36
37  origCluster <- c()
38  data <- NULL
39
40  for(i in 1:Ncluster)  {
41        origCluster <- c(origCluster,
42                         rep.int(i, clusterSizes[i]))
43
44        dataCluster <- rmvdc(clusterList[[i]],
45                             clusterSizes[i])
46        if(is.null(data))
47          data <- dataCluster
48        else
49          data <- rbind(data, dataCluster)
50  }
51
52  time.start <- Sys.time()
53  mod <- NULL
54  try(mod <-
55      flexmix(data~1, k=Ncluster,
56              model=FLXMCcopula(mvdc=startMvdc)))
```

```
57    time.end <- Sys.time()
58
59
60    out <- list(
61             data       = data,
62             name       = simName,
63             orig       = origCluster,
64             mvdcList   = clusterList,
65             sizes      = clusterSizes,
66             mvdc       = startMvdc,
67             model      = mod,
68             startTime  = time.start,
69             endTime    = time.end,
70             sysInfo    = Sys.info()
71    )
72
73    save(out, file=fname, compress="bzip2")
74  }
```

`makedata.R` is script that iterates over all the data files created by `sim.once` and then stores the results of the simulation using `write.table`. Mostly the estimation error, the ratio of correctly classified observations and simulation covariates, such as the sample sizes of the clusters, the amount of clusters or the value of θ (depending on simulation type) is stored.

Listing 14: makedata.R

```
1  library(copula)
2  library(flexmix)
3  source("flxmccopula.R")
4  source("sim.R")
5
6
7  files <- Sys.glob("data/*.RData")
8  Nfiles <- length(files)
9
10 olddata <- NULL
11 try(olddata <- read.table("data/sim.txt",
12                             header=T, as.is=T))
13 if(!is.null(olddata)) {
14   if(nrow(olddata) == 0)
15     olddata <- NULL
16 }
```

```
17
18 cnt <- 0
19 data <- NULL
20
21 start <- Sys.time()
22
23 for(fn in files) {
24   cnt <- cnt+1
25   out <- NULL
26   cat(sprintf("%5d/%5d␣%2.1f%%␣>>␣%s\n",
27               cnt, Nfiles, cnt/Nfiles*100, fn))
28   newid <- substr(fn, 9, nchar(fn)-6)
29
30   if(!is.null(olddata)) {
31     if(nrow(subset(olddata, id == newid)) > 0) {
32       next
33     }
34   }
35
36   load(fn)
37
38   if(!is.null(out) & !is.null(out$model)) {
39     if(length(parameters(out$model)) ==
40        length(out$mvdcList)) {
41       param.orig <- lapply(out$mvdcList, mvdcGetParam)
42       param.est  <- lapply(parameters(out$model), mvdcGetParam)
43
44       perm <- cluster.order(param.orig, param.est)
45       stat <- cluster.distance(param.orig, param.est, perm)
46       ratio <- cluster.classification(out$orig,
47                                       clusters(out$model),
48                                       perm)
49     } else {
50       perm <- NA
51       stat <- NA
52       ratio <- NA
53     }
54
55     duration <- as.double(out$endTime - out$startTime,
56                           units="secs")
57
58     theta <- NA
59     Ncluster <- 2
```

```
60
61      if(out$name == "theta12345") {
62        theta <- out$mvdcList[[1]]@copula@parameters
63      } else if(out$name == "Ncluster") {
64        Ncluster <- length(out$mvdcList)
65      }
66
67      newdata <- c(newid, out$name,
68                   stat, ratio,
69                   sum(out$sizes), Ncluster,
70                   theta, duration,
71                   out$sysInfo[["nodename"]] )
72
73      if(is.null(data)) {
74        data <- matrix(nrow=1, data=newdata)
75      } else {
76        data <- rbind(data, newdata)
77      }
78    }
79  }
80
81  end <- Sys.time()
82  print(as.double(end-start, units="secs"))
83
84  rownames(data) <- NULL
85  df <- as.data.frame(data, stringsAsFactors=F)
86  colnames(df) <- c("id", "name", "stat", "ratio",
87                    "N", "Ncluster",
88                    "theta", "duration",
89                    "nodename")
90
91  if(!is.null(olddata)) {
92    if(all(colnames(df) == colnames(olddata))) {
93      df <- rbind(df, olddata)
94    }
95  }
96
97  write.table(df, file="data/sim.txt")
```

References

I. N. Bronstein, K. A. Semendjajew, G. Musiol, and H. Mühlig. *Taschenbuch der Mathematik*. Thun, Frankfurt am Main: Verlag Harri Deutsch, 6 edition, 2005.

Edward W. Frees and Emiliano A. Valdez. Understanding relationships using copulas. *North American Actuarial Journal*, 2:1–25, 1998.

Christian Genest and Anne C. Favre. Everything you always wanted to know about copula modeling but were afraid to ask. *Journal of Hydrologic Engineering*, 12(4):347–368, 2007.

Christian Genest, Bruno Rémillard, and David Beaudoin. Goodness-of-fit tests for copulas: A review and a power study. *Insurance: Mathematics and Economics*, 44(2):199–213, April 2009. URL `http://ideas.repec.org/a/eee/insuma/v44y2009i2p199-213.html`.

Ivan Kojadinovic, Jun Yan, and Mark Holmes. Fast large–sample goodness-of-fit tests for copulas. *Statistica Sinica*, in press.

Ivan Kondofersky. Modellbasiertes Clustern mit der Beta-Binomialverteilung. Bachelor's thesis, Ludwig–Maximilians–Universität München, 2008.

Friedrich Leisch. Flexmix: A general framework for finite mixture models and latent class regression in R. *Journal of Statistical Software*, 11(8):1–18, 10 2004. ISSN 1548-7660. URL `http://www.jstatsoft.org/v11/i08`.

Uwe Ligges. *Programmieren mit R*. Springer-Verlag Berlin Heidelberg, 2007. URL `http://ebooks.ub.uni-muenchen.de/8279/`.

Roger B. Nelsen. *An Introduction to Copulas (Springer Series in Statistics)*. Springer, 2nd edition, 10 2007. ISBN 9780387286594. URL `http://amazon.com/o/ASIN/0387286594/`.

Jun Yan. Enjoy the joy of copulas: with a package copula. *Journal of Statistical Software*, 21(4):1–21, 2007.

Affidavit

I, Dominik Ernst, hereby declare that this bachelor–thesis in question was written single–handed and no further as the denounced resources and sources were employed.

Munich, March 21, 2010

(Dominik Ernst)

Eidesstattliche Erklärung

Hiermit versichere ich, Dominik Ernst, dass ich die vorliegende Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 21.03.2010

(Dominik Ernst)