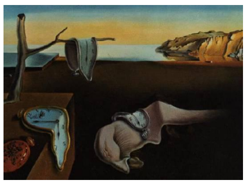

Journal of Statistical Software
April 2012, Volume 47, Issue 3. *http://www.jstatsoft.org/*

The R Package bgmm: Mixture Modeling with Uncertain Knowledge

Przemysław Biecek
University of Warsaw

Ewa Szczurek
Max Planck Institute
for Molecular Genetics

Martin Vingron
Max Planck Institute
for Molecular Genetics

Jerzy Tiuryn
University of Warsaw

Abstract

Classical supervised learning enjoys the luxury of accessing the true known labels for the observations in a modeled dataset. Real life, however, poses an abundance of problems, where the labels are only partially defined, i.e., are uncertain and given only for a subset of observations. Such partial labels can occur regardless of the knowledge source. For example, an experimental assessment of labels may have limited capacity and is prone to measurement errors. Also expert knowledge is often restricted to a specialized area and is thus unlikely to provide trustworthy labels for all observations in the dataset. Partially supervised mixture modeling is able to process such sparse and imprecise input. Here, we present an R package called **bgmm**, which implements two partially supervised mixture modeling methods: soft-label and belief-based modeling. For completeness, we equipped the package also with the functionality of unsupervised, semi- and fully supervised mixture modeling. On real data we present the usage of **bgmm** for basic model-fitting in all modeling variants. The package can be applied also to selection of the best-fitting from a set of models with different component numbers or constraints on their structures. This functionality is presented on an artificial dataset, which can be simulated in **bgmm** from a distribution defined by a given model.

Keywords: mixture modeling, semi-supervised modeling, belief-based modeling, soft-label modeling, R.

1. Introduction

The most fundamental applications of statistical learning include supervised modeling (classification) and unsupervised modeling (clustering). In the first case, also called discrimination analysis, observations in the modeled dataset are all labeled with their known class. In the second case, no information about the classes of the observations is given, and labeling them with their corresponding cluster is the task of the modeling. The number of clusters may be given in the input, but may as well be estimated by the model. Recent developments in statistics offer a range of learning variants, spreading between the fully supervised and unsupervised approaches, and can be divided into two different types.

The first type is semi-supervised modeling, which takes as input a dataset where for a subset of observations the labels are known, and the remaining part is unlabeled (Ambroise, Denœux, Govaert, and Smets 2001; Zhu and Goldberg 2009; Zhu 2005). The second type is implemented by the recently developed partially supervised modeling (Ambroise and Govaert 2000; Côme, Oukhellou, Denœux, and Aknin 2009; Hüllermeier and Beringer 2006; Lawrence and Schölkopf 2001; Bouveyron and Girard 2009). Partially supervised modeling works with datasets, where for a subset of observations labels may be concluded, but the labels are *uncertain*, i.e., determined with some probability. Figure 1 illustrates the different modeling variants with respect to utilizing knowledge.

Partially supervised modeling constitutes the most general variant and is applicable in an abundance of real-life problems. The uncertainty may result from the properties of the knowledge sources from which the labels are derived, e.g., a literature study of similar, but not exactly the same objects, noisy measurements due to technological limitations, or a panel of experts who may disagree in some cases. Depending on the problem, the partially supervised modeling methods may be applied to classification improved with the use of additional unlabeled data, or to clustering by utilizing the labeled observations. To unify the nomenclature we embrace these two applications in a single term *prediction*, as in both cases modeling is applied to predict correct (either class or cluster) labels.

All the described learning variants are developed in the area of mixture modeling. The common point of various mixture modeling methods is the assumption, that observations come from a mixture of components with different distributions. Each component is related to a cluster in unsupervised modeling or to a class in fully supervised modeling. Fitted mixture models are applied to prediction of labels for the partially labeled or unlabeled observations: the observations are assigned the label, which is the most probable according to the model.

In this paper we present an R package called **bgmm** (for belief-based Gaussian mixture modeling) that is available from the Comprehensive R Archive Network (CRAN) at `http://CRAN.R-project.org/package=bgmm`. The package implements different multivariate Gaussian mixture modeling methods, each representing a particular learning variant. Gaussian mixture modeling and its variants are shortly introduced in Section 1.1. The focus of our work is on two partially supervised mixture modeling methods: soft-label modeling, introduced by Côme *et al.* (2009), and our contribution, called belief-based modeling. Belief-based modeling was described theoretically, compared to the soft-label modeling and verified in application to one-dimensional gene expression data in Szczurek, Biecek, Tiuryn, and Vingron (2010). Here, the formulation of the belief-based and soft-label modeling methods is only briefly recapitulated in Section 1.2. Section 1.3 lists existing software that provides implementation of the fully, semi- and unsupervised modeling.

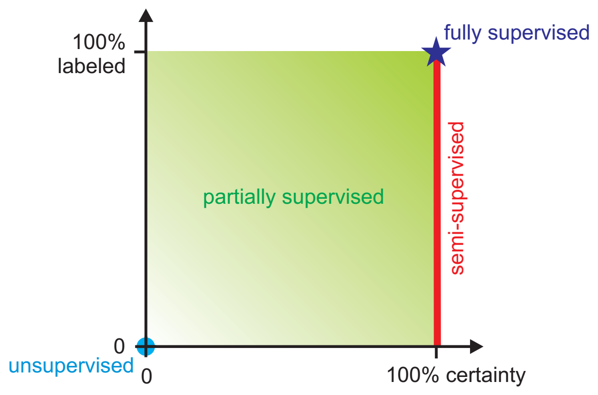

Figure 1: Amount of knowledge required by the different variants of modeling. x-axis: Knowledge certainty. y-axis: The percentage of observations for which knowledge is provided as labels. The modeling method variants are color-coded. Unsupervised modeling (blue dot) requires no input knowledge. On the other hand, fully supervised modeling (violet star) requires 100% certain knowledge about all labels. Semi-supervised modeling (red line) take in the input certain knowledge given for any subset of all observations. The most general partially supervised modeling (green area shaded darker for increasing knowledge) accepts uncertain knowledge about a subset of observations.

Both the partially supervised belief-based and soft-label modeling methods are supported by the **bgmm** package. Fully supervised, semi-supervised, and unsupervised modeling are added to the package for completeness. Section 2 presents the basic usage of the implemented model-fitting functions, model initialization and model-based label prediction in application to genotyping data of Takitoh, Fujii, Mase, Takasaki, Yamazaki, Ohnishi, Yanagisawa, Nakamura, and Kamatani (2007). The **bgmm** package offers automatic selection of the optimal model structure and the number of components. This functionality is presented on a simulated dataset in Section 3.

1.1. From fully supervised to fully unsupervised mixture modeling

Gaussian mixture model is defined by a pair (X, Y) of random variables, where the d-dimensional variable X takes values in $\mathcal{R}^d$, while Y takes values in the set $\mathcal{Y} = \{1, ..., k\}$. The value k defines the number of mixture components. The variable Y follows a multinomial distribution with probabilities $(\pi_1, \pi_2, ..., \pi_k)$, called mixing proportions or prior probabilities. $X|Y = y$ follows a multidimensional Gaussian distribution with parameters $\theta_y = (\mu_y, \Sigma_y)$, and with a density function $f(x, \theta_y)$. The density of the joint distribution of the variables (X, Y) reads

$$f(x, y) \quad = \quad \pi_y f(x, \theta_y),$$

where $x \in \mathcal{R}^d$ and $y \in \mathcal{Y}$ is the y-th Gaussian component. Accordingly, the marginal distribution of X is a mixture of k multivariate Gaussian distributions:

$$f(x) = \sum_{y=1}^{k} \pi_y f(x, \theta_y).$$

Usually the parameters μ_y, Σ_y as well as π_y are unknown. However, additional constraints may be put on the model structure, for example, the equality of the parameters μ_y or Σ_y, for all $y \in \mathcal{Y}$ (see Section 2.1). To estimate the set of mixture model parameters $\Phi = (\pi_1, ..., \pi_k, \theta_1, ..., \theta_k)$, the number of components k needs to be fixed. In real life problems, often the number of components is not known *a priori*. In such a case, mixture models for different values of k can be fitted in order to choose the one that best satisfies given model selection criteria (Section 3.4). The same criteria can be used to compare models having different constraints put on their structures (Section 3.3).

Consider n realizations of the random variables (X, Y), denoted x_i, y_i, for $1 \leq i \leq n$. The values of the variable X are observed. For the variable Y, four different scenarios define four mixture modeling variants, which differ in the amount of knowledge given about y_i:

1. **Fully supervised modeling:** Variable Y is observed for all realizations, i.e., y_i is given for all i.

2. **Semi-supervised modeling:** Variable Y is observed for m realizations $(0 \leq m \leq n)$.

3. **Partially supervised modeling:** Variable Y is observed for m realizations $(0 \leq m \leq n)$, but those observations are uncertain, i.e., their values are given with some probability (see Section 1.2). The **bgmm** package implements two partially supervised mixture modeling methods: belief-based and soft-label mixture modeling.

4. **Unsupervised modeling:** Variable Y is not observed for any realization.

Realizations, for which the values of Y are known, or (in the case of partially supervised modeling) can be pointed as the most probable, are called labeled. Thus, each label assigns to a given observation its mixture model component, which according to the model's application may correspond to a class or a cluster. Without loss of generality, we assume that the labeled observations have indexes $1 \leq i \leq m$, while the remaining observations are indexed with $i > m$.

1.2. Partially supervised belief-based and soft-label mixture modeling

Belief-based modeling

In belief-based modeling, first introduced by Szczurek *et al.* (2010), the uncertainty of labels is expressed as a probability distribution over the set of components. Technically, for the first m observations a so called $(m \times k)$ beliefs matrix B is defined. The matrix entry $b_{i,j}$ is interpreted as a belief that observation i belongs to component j. The belief values for a given observation i need to define a probability, i.e., $\sum_{j=1}^{k} b_{i,j} = 1$ (the values in the i-th row of B need to sum up to 1). For the m labeled observations, their belief values replace the prior probabilities and remain fixed during the estimation process.

The log likelihood can be written as

$$\begin{aligned} l(\mathcal{X}, B, \Phi) = \; & \textstyle\sum_{i=1}^{m} \log\left(\sum_{j=1}^{k} b_{i,j} f(x_i, \theta_j)\right) + \\ & \textstyle\sum_{i=m+1}^{n} \log\left(\sum_{j=1}^{k} \pi_j f(x_i, \theta_j)\right), \end{aligned}$$

where $\mathcal{X} = \{x_1, ..., x_n\}$ is the input set of n observations, B is the beliefs matrix, $\Phi = (\pi_1, ..., \pi_k, \theta_1, ..., \theta_k)$ stands for the set of model parameters, and $\theta_i = (\mu_i, \Sigma_i)$ are the Gaussian parameters of the i-th component.

The parameters in Φ can be estimated with an expectation-maximization (EM) method. In the (q+1)-th iteration, the posteriors are calculated in the E step as

$$t_{i,j}^{(q+1)} = \begin{cases} b_{i,j} f\left(x_i, \theta_j^{(q)}\right) / \sum_{l=1}^{k} b_{i,l} f\left(x_i, \theta_l^{(q)}\right) & i \leq m \\ \pi_j^{(q)} f\left(x_i, \theta_j^{(q)}\right) / \sum_{l=1}^{k} \pi_l^{(q)} f\left(x_i, \theta_l^{(q)}\right) & i > m \end{cases} \tag{1}$$

In the M step the model parameters are updated to

$$\pi_j^{(q+1)} = \sum_{i=m+1}^{n} t_{i,j}^{(q+1)} / (n - m) \tag{2}$$

$$\mu_j^{(q+1)} = \left(\sum_{i=1}^{n} x_i t_{i,j}^{(q+1)}\right) / \left(\sum_{i=1}^{n} t_{i,j}^{(q+1)}\right) \tag{3}$$

$$\Sigma_j^{2\ (q+1)} = \left(\sum_{i=1}^{n} \left(x_i - \mu_j^{(q+1)}\right)^\top \left(x_i - \mu_j^{(q+1)}\right) t_{i,j}^{(q+1)}\right) / \left(\sum_{i=1}^{n} t_{i,j}^{(q+1)}\right). \tag{4}$$

Soft-label modeling

The soft-label approach, introduced by Côme *et al.* (2009), works with more general labels than our implementation. Each label can be a subset of the model components. We reduce this general approach to a particular case, where only labels that are single components are allowed. In this formulation, soft-label modeling takes as the input a $(n \times k)$ matrix of plausibilities P, defined for all observations. The plausibility $p_{i,j}$ stands for a weight of the prior probability that the observation i belongs to the component j. Thus, the lack of additional knowledge about a given observation is reflected in equal weights. We add a constraint on plausibilities, i.e., for each observation they have to sum up to 1, i.e., $\sum_{j=1}^{k} p_{i,j} = 1$. This constraint is added for the sake of unified implementation of different methods, and has no effect on the maximum likelihood estimate, since it only rescales the likelihood. The log likelihood is given by

$$l(\mathcal{X}, P, \Phi) = \sum_{i=1}^{n} \log \left(\sum_{j=1}^{k} p_{i,j} \pi_k f(x_i, \theta_j)\right).$$

In the $(q + 1)$-th iteration of the EM algorithm, the posteriors are computed in the E step as

$$t_{i,j}^{(q+1)} = p_{i,j} \pi_j^{(q)} f\left(x_i, \theta_j^{(q)}\right) / \sum_{l=1}^{k} p_{i,l} \pi_l^{(q)} f\left(x_i, \theta_l^{(q)}\right), \quad i \leq m,$$

while the M step the mixing proportions are updated by

$$\pi_j^{(q+1)} = \sum_{i=1}^{n} t_{i,j}^{(q+1)} / n, \tag{5}$$

and the Gaussian parameters are updated using Equations 3 and 4.

Note that the belief-based and soft-label modeling methods differ in the way the uncertain knowledge is incorporated into the models, and beliefs should be interpreted differently than plausibilities. In the belief-based approach beliefs replace priors for the labeled observations, while in the soft-label approach plausibilities work as weights for priors. For example, in the belief-based approach beliefs $(0.33, 0.67)$ represent prior knowledge that the observation belongs to the first component with 0.33 probability and to the second component with the probability 0.67. In contrast, in the soft-label approach plausibilities $(0.33, 0.67)$ represent knowledge that the observation belongs to the second component with twice as high probability than defined by the mixing proportions.

Fixing uniform beliefs incorporates specific knowledge that each component is equally probable, since the beliefs replace the priors for the observations. In contrast to that, fixing equal plausibilities does not incorporate any knowledge (i.e., the user does not know whether the components are equally probable, or whether maybe one of them is much more probable than the others). Thus, when only m observations are labeled, in the case of soft-label modeling the remaining observations are given plausibilities that are uniformly distributed over all components. This is in contrast to beliefs in belief-based modeling, which are not defined for the remaining observations. Szczurek *et al.* (2010) show that due to differences in mixing proportion estimation (Equation 2 vs. 5), belief-based modeling is more suitable for the more realistic problems with only a small subset of observations labeled. Soft-label modeling should rather be applied when the observations are labeled in a large part.

Note also, that partially supervised modeling is more general than semi-supervised modeling. Semi-supervised modeling is implemented by the belief-based modeling when for the labeled observations $b_{i,j} \in \{0, 1\}$ ($i \leq m$ and $j \in \mathcal{Y}$), as well as by soft-label modeling when for the labeled observations $p_{i,j} \in \{0, 1\}$ and for the remaining observations the plausibilities are uniform. Szczurek *et al.* (2010) prove advantage of the partially supervised modeling methods over semi-supervised modeling. In contrast to semi-supervised modeling, which assumes that the input labeling of observations is fixed, partially supervised modeling uses the entire (also unlabeled) data to verify the knowledge about the labels. After the model is fitted, it may predict labels that are different than the ones specified as the most believed or plausible. Such "re-labeling" happens for observations originally labeled with components, which are highly improbable according to the fitted model.

1.3. Implementations of the fully, semi- and unsupervised modeling

The fully supervised modeling variant proceeds in one simple step, by estimating each component's parameters directly from its observations. For a given component $y \in \mathcal{Y}$, the parameters μ_y and Σ_y are estimated by the mean vector, and the covariance matrix of the observations labeled y. The estimator of the prior probability π_y is given by the proportion of the observations labeled y toward all observations. Such estimations can be made with the R package **MASS** (Venables and Ripley 2002). In the general case, supervised modeling is also called quadratic discriminant analysis, and is implemented by the function `qda()` in the **MASS** package. In some applications it is convenient to force $\Sigma_i = \Sigma_j$ for all $1 \leq i, j \leq k$. This case is called linear discriminant analysis and is implemented by the function `lda()`.

Despite the large number of publications in which authors show the advantages of semi-supervised modeling (Bair and Tibshirani 2004; Zhu and Goldberg 2009; Zhu 2005), its implementations are not easily available in statistical packages. In R (see R Development Core Team (2012)) both packages **phyclust** (Chen 2011) and **spa** (Culp 2011) provide some support for semi-supervised modeling. However, neither of them focuses on estimation of Gaussian mixture models. Although the inner procedures of the **phyclust** package do estimate a mixture model, the package works specifically on DNA sequence data and is aimed at reconstruction of phylogenetic trees and phyloclustering. The **spa** package contains functions for semi-supervised regression-like estimation and graph-based estimation.

A broad variety of CRAN packages supporting unsupervised learning and mixture modeling is available. A list of them may be found in the task view "Cluster Analysis and Finite Mixture Models" (Leisch and Grün 2012). One of the most flexible packages for unsupervised mixture modeling is **mclust**, (Fraley and Raftery 2006) which implements an EM algorithm for model fitting, and uses the Bayesian information criterion(BIC, Schwarz 1978) to determine the number of components. Another package, **HDclassif** (Bergé, Bouveyron, and Girard 2012) contains functions both for fully unsupervised and fully supervised modeling. Similarly as our **bgmm**, the packages **mclust** and **HDclassif** allow specifying additional constraints for model parameters. In all three packages the constraints reduce the number of degrees of freedom of the models and result in more stable estimates. With this respect, the approach implemented in **HDclassif** differs from the one provided by **mclust** and **bgmm**. In the latter packages the user can choose one of a few classes for covariance matrices, while in **HDclassif** constraints are defined on the eigenspace of covariance matrix and are more data-guided.

The package **Spider**, (Weston, Elisseeff, Bakir, and Sinz 2010) a comprehensive object-oriented environment for machine learning in MATLAB, provides semi-supervised learning with the support vector machine (SVM) method (see Weston, Leslie, Ie, Zhou, Elisseeff, and Noble (2005) for an example application).

Szczurek *et al.* (2010) provided initial implementation of the partially supervised belief-based and soft-label mixture modeling methods in the first version of **bgmm** package. This implementation was applied specifically to expression data measured upon single-gene knockouts and other single-measurement experiments. Thus, modeling was limited to one-dimensional data only. Moreover, model selection was not supported. In this paper, we extend the package to contain full functionality. To our knowledge, it is the first exhaustive partially supervised modeling software available open source. For completeness, the package provides also the remaining three modeling variants, namely the fully, semi- and unsupervised mixture modeling methods. Those different variants of mixture modeling are handled in a similar way. For all of them functions for parameter estimation, cross-validation and structure selection are provided. Note that in case of supervised modeling other packages contain more specialized functions like receiver operating characteristic (ROC) curves, different error measures, more flexible cross-validation schemes, etc. The package is available for download from CRAN and from the project website `http://bgmm.molgen.mpg.de/`.

2. Basic model-fitting with the bgmm package

First, we illustrate the basic functionality of the **bgmm** package in application of the implemented mixture modeling methods to single-nucleotide polymorphism (SNP) genotyping data. SNPs are different mutations of the DNA sequence, which appear in the genomes of a given population. Typically, each SNP has two different mutations, called alleles. We analyze the dataset of Takitoh *et al.* (2007), where each observation is represented by two real values, measured by the Invader assay (Ohnishi, Tanaka, Ozaki, Yamada, Suzuki, and Nakamura 2001). Each value represents an intensity of the fluorescence signal corresponding to a given allele of a given SNP. Three types can be distinguished in the measured set of 333 SNPs. The first type occurs only as its first possible allele 1, second only as its second possible allele 2, and third as both alleles. The fluorescent points cluster accordingly. Assigning the SNPs to their correct types based on the fluorescence intensities is called genotyping. In other words, genotyping is a clustering problem, where the clusters to be found in the data correspond to the three types of SNPs that can occur. Thus, the required number of model components is known and equals three. The correct types of the 333 SNPs in the Takitoh *et al.* (2007) dataset are also known and can serve as labels for the supervised modeling methods.

The input data and knowledge for the modeling methods is gathered in the `genotypes` object in the **bgmm** package. To prepare the input for the partially and semi-supervised modeling methods, we separated 15 SNPs, choosing at random five per each cluster, and labeled them with their correct type. For the partially supervised methods, the belief/plausibility values of the most probable type reflect the high certainty of the labels and are set to 0.95, and of the remaining types are equal 0.025. The remaining 318 SNPs are kept unlabeled.

```
R> library("bgmm")
R> data("genotypes")
R> str(genotypes)

List of 4
 $ X     : num [1:318, 1:2] 0.741 0.682 0.797 0.772 0.738 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:318] "1" "2" "3" "4" ...
  .. ..$ : chr [1:2] "allele1" "allele2"
 $ knowns: num [1:15, 1:2] 0.815 0.955 0.833 0.772 0.828 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:15] "known1" "known2" "known3" "known4" ...
  .. ..$ : chr [1:2] "allele1" "allele2"
 $ B     : num [1:15, 1:3] 0.95 0.95 0.95 0.95 0.95 0.025 0.025 0.025 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:15] "known1" "known2" "known3" "known4" ...
  .. ..$ : chr [1:3] "1" "3" "2"
 $ labels: Named int [1:15] 1 1 1 1 1 3 3 3 3 3 ...
  ..- attr(*, "names")= chr [1:15] "known1" "known2" "known3" "known4" ...
```

The object `genotypes` contains four slots with data.frames. First, the data.frame in slot `$X` is filled with fluorescence intensities for allele 1 and allele 2, measured for the 318 unlabeled SNPs. Second, the data.frame in the slot `$knowns` contains intensities for the 15 SNPs that are labeled with their correct type. Third, the data.frame `$B` keeps the beliefs about the assignment of the 15 known SNPs to each type (i.e., each mixture component). In the following usage cases we assume that for those 15 labeled genes the plausibilities are distributed in the same way as the beliefs. Recall from Section 1.2 that the remaining unlabeled SNPs should have plausibilities that are uniformly distributed over the mixture components. The fourth slot `$labels` contains a vector of labels for the 15 known SNPs.

The **bgmm** package provides five model-fitting functions, one per each of the mixture modeling variants. Due to differences in the basic input expected by the modeling variants, these functions differ slightly in the lists of their arguments:

- A function `belief()` takes at least three arguments: `X` – a data.frame with the unlabeled observations, `knowns` – a data.frame with the labeled observations, and `B` – a beliefs matrix with the distribution of beliefs for the labeled observations. Unless explicitly provided, the argument `k` fixing the number of components is by default equal the number of columns of `B`,
- A function `soft()` takes the same arguments as `belief`, but instead of `B`, the argument `P` can be provided. `P` is the matrix of plausibilities, but specified only for the labeled observations. The function assumes that the remaining observations are unlabeled and gives them uniformly distributed plausibilities by default,
- A function `semisupervised()` takes the same arguments as `belief`, but instead of `B`, the argument `class` can be provided. `class` has to contain the vector of classes for the labeled observations,
- A function `supervised()` assumes that all observations are labeled. Thus it requires only two arguments, i.e., `X` with the data and `class` with the labels. The number of components is derived from the number of unique labels,
- A function `unsupervised()` takes only two arguments, i.e., `X` with the observations and `k` with the number of components.

In fact, the implementation of the model-fitting functions is very flexible, and adjusted to allow maximally similar input between the functions. For example, the user may provide argument `B` to the `soft()` function, and it will automatically be treated as the argument `P`. Moreover, providing distributions over components with the arguments `B` or `P` to the `semisupervised()` function is also possible, as they can be used by the function to derive the most probable labels and convert them to classes by default. For more details on the full spectrum of the function parameters refer to the package manual.

Below usage cases for each model-fitting function are presented. Each function returns an object of the class `mModel`. The `mModel` class overloads the `plot()` function. Figure 2 shows the plots for the models fitted with the presented function calls. In case of supervised modeling model is build with use of only 15 labeled observations, in other cases all observations are used. Note that the `genotypes` dataset easily separates into three disjoint clusters, and here it serves only to illustrate the modeling functionality of **bgmm**. Modeling of more complicated data is presented in Section 3.1.

```
R> modelSupervised <- supervised(knowns = genotypes$knowns,
+    class = genotypes$labels)
R> plot(modelSupervised)
R> modelSemiSupervised <- semisupervised(X = genotypes$X,
+    knowns = genotypes$knowns, class = genotypes$labels)
R> plot(modelSemiSupervised)
R> modelBelief <- belief(X = genotypes$X, knowns = genotypes$knowns,
+    B = genotypes$B)
```

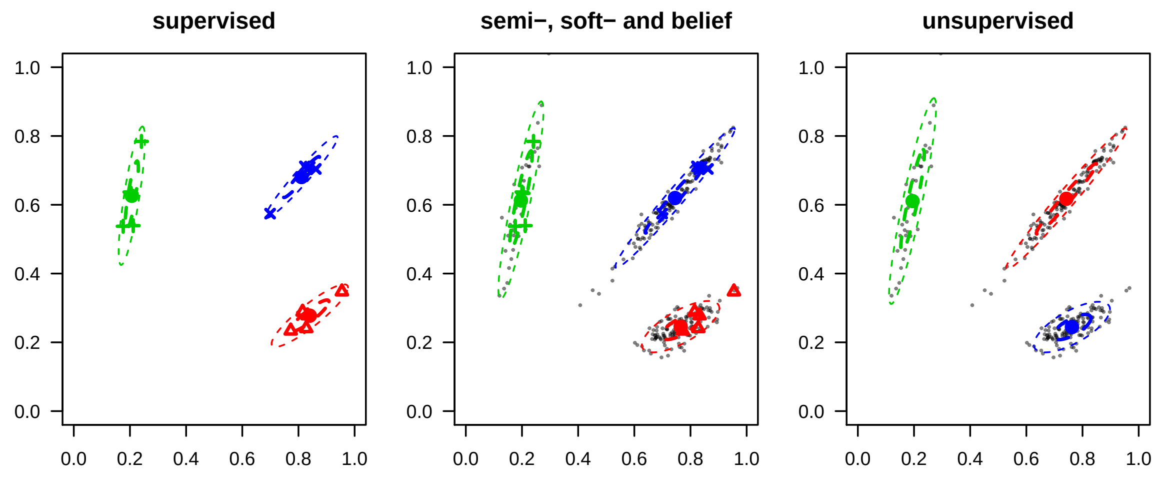

Figure 2: The models fitted with different mixture modeling variants to the two-dimensional SNP genotyping data of Takitoh *et al.* (2007). The axes correspond to fluorescence intensities for allele 1 (x-axis) and allele 2 (y-axis). Black dots, visibly grouping into three clusters, represent observations without labels. Colored symbols code for the three possible SNP types, used as labels for the known observations. Colored ellipses are graphical representations of the fitted Gaussian components, one per each of the SNP types. In this example, the models fitted by the `belief()`, `soft()` and `semisupervised()` functions are identical and thus plotted on a common panel (middle). The colors of the Gaussians fitted with unsupervised modeling (right panel) are arbitrarily assigned to the identified clusters of data. On the contrary, the Gaussian components fitted by the supervised modeling method (left panel) correspond only to the SNP types of the observations that are known to come from those components (see Section 2.3). Thus the colors of those Gaussians match the colors of the label symbols. In this example, the number of labeled observations utilized by the `supervised()` function is much smaller than the number of all observations (utilized by the remaining functions). This is the reason why the left plot has fewer points.

```
R> plot(modelBelief)
R> modelSoft <- soft(X = genotypes$X, knowns = genotypes$knowns,
+    P = genotypes$B)
R> plot(modelSoft)
R> modelUnSupervised <- unsupervised(X = genotypes$X, k = 3)
R> plot(modelUnSupervised)
```

Objects returned from these functions describe the fitted model:

```
R> str(modelBelief)
```

```
List of 15
 $ pi               : Named num [1:3] 0.456 0.116 0.428
  ..- attr(*, "names")= chr [1:3] "1" "3" "2"
 $ mu               : num [1:3, 1:2] 0.765 0.196 0.745 0.246 0.613 ...
 $ cvar             : num [1:3, 1:2, 1:2] 0.00476 0.00161 0.01137 0.00184 ...
```

```
  ..- attr(*, "dimnames")=List of 3
  .. ..$ : NULL
  .. ..$ : chr [1:2] "allele1" "allele2"
  .. ..$ : chr [1:2] "allele1" "allele2"
 $ B              : num [1:15, 1:3] 0.95 0.95 0.95 0.95 0.95 0.025 0.025 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:15] "known1" "known2" "known3" "known4" ...
  .. ..$ : chr [1:3] "1" "3" "2"
 $ m              : int 15
 $ n              : int 333
 $ k              : int 3
 $ d              : int 2
 $ likelihood     : num 817
 $ n.steps        : num 5
 $ tij            : num [1:333, 1:3] 1 1 1 1 1 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:333] "known1" "known2" "known3" "known4" ...
  .. ..$ : chr [1:3] "1" "3" "2"
 $ X              : num [1:318, 1:2] 0.741 0.682 0.797 0.772 0.738 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:318] "1" "2" "3" "4" ...
  .. ..$ : chr [1:2] "allele1" "allele2"
 $ knowns         : num [1:15, 1:2] 0.815 0.955 0.833 0.772 0.828 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:15] "known1" "known2" "known3" "known4" ...
  .. ..$ : chr [1:2] "allele1" "allele2"
 $ model.structure:List of 4
  ..$ mean   : chr "D"
  ..$ between: chr "D"
  ..$ within : chr "D"
  ..$ cov    : chr "D"
 $ dof            : num 15
 - attr(*, "class")= chr [1:2] "beliefModel" "mModel"
```

The objects returned by all five functions share a common class attribute `mModel`, thus in all cases the generic function `plot()` in fact calls the function `plot.mModel()`.

The object contains slots for the model parameters: a vector with the fitted mixing proportions `$pi`, a matrix `$mu` with the means fitted for all components, and a three-dimensional matrix `$cvar` with the covariance matrices fitted for all components. The first dimension in this array corresponds to the index of Gaussian component, so that `$cvar[i, , ]` results in a two-dimensional covariance matrix of the ith Gaussian component. It holds also slots for the input data and knowledge: `$X` for the unlabeled and `$knowns` for the labeled observations, `$B` for the beliefs matrix (in a `mModel` object returned from the `soft()` function there is a corresponding slot `$P` instead, whereas models fitted with the `supervised()` and `semisupervised()` functions contain a slot `class`), `$n` for the number of all and `$m` for the number of unlabeled observations, `$k` for the number of components and finally, `$d` for the data dimension. The slot `$likelihood` contains the log-likelihood of the fitted model,

`$n.steps` the number of steps performed by the EM algorithm, `$model.structure` the set of constraints kept during fitting process (see Section 2.1 below), `$dof` the number of degrees of freedom and `$tij` the posteriors for each observation. In this example, the likelihood of the model fitted by the `belief()` function equals:

```
R> modelBelief$likelihood
```

```
[1] 817.1028
```

2.1. Constraining the model structure

For some problems, it is possible to *a priori* specify constraints on the structure of the fitted model. For example, it may be known that within each model component the measurements in each dimension are independent and have equal variances (for two-dimensional data this would imply that the Gaussians are circular). The **bgmm** package allows incorporating such prior information into the model estimation procedures. The specified model constraints affect the M step of the EM algorithms used in all mixture modeling variants.

Constraining model structure decreases the number of degrees of freedom of the model, thereby increasing the stability of parameter estimation, and is particularly useful when the number of observations is small. For d-dimensional data, an unconstrained mixture model with k components is defined by $kd + kd(d+1)/2 + k - 1$ degrees of freedom. This total number consists of kd parameters for the component means $\mu_1, ..., \mu_k$, $kd(d-1)/2$ for the covariance matrices $\Sigma_1, ..., \Sigma_k$, and $k-1$ for the mixing proportion parameters $\pi_1, ..., \pi_{k-1}$. Thus, for example, forcing independence and equal variances of the data vector components (corresponding to data dimensions) within each model component reduces the number of degrees of freedom to $k(d+2)-1$. Table 1 presents numbers of degrees of freedom for different constraints of the model structure possible in the **bgmm** package.

Specifying the preferred model structure in **bgmm** can be performed using a function called `getModelStructure()`. By default, the function returns an unconstrained structure. We list the possible arguments and their settings, which allow for introducing constraints:

- A constraint `mean = "E"` forces equality of the means' vectors between the components, i.e., $\forall_{i \in \mathcal{Y}} \mu_i \;=\; \mu_1$,
- A constraint `between = "E"` forces equality of the covariance matrices between the components, i.e., $\forall_{i \in \mathcal{Y}}\ \Sigma_i \;=\; \Sigma_1$,
- A constraint `within = "E"` forces equality of variances within each covariance matrix i to some constant v_i and equality of covariances to some constant w_i, i.e., $\forall_{1 \leq r \leq d} \Sigma_i[r,r] = v_i$, $\forall_{1 \leq r \neq c \leq d} \Sigma_i[r,c] = w_i$,
- A constraint `cov = "0"` forces equality of covariances within each covariance matrix i to 0, i.e., $\forall_{1 \leq r \neq c \leq d} \Sigma_i[r,c] = 0$,

where $\Sigma_i[r,c]$ is the entry in the r-th row and c-th column of the i-th covariance matrix.

For example, below we set all the means' vectors equal with the use of the `mean` argument:

Model structure	# Ind. parameters	Model structure	# Ind. parameters
DDDD	$kd + kd(d+1)/2$	EDDD	$d + kd(d+1)/2$
DDD0	$kd + kd$	EDD0	$d + kd$
DDED	$kd + 2k$	EDED	$d + 2k$
DDE0	$kd + k$	EDE0	$d + k$
DEDD	$kd + d(d+1)/2$	EEDD	$d + d(d+1)/2$
DED0	$kd + d$	EED0	$d + d$
DEED	$kd + 2$	EEED	$d + 2$
DEE0	$kd + 1$	EEE0	$d + 1$

Table 1: Degrees of freedom for different model structures. Each number is given omitting the $k-1$ parameters defining the mixing proportions, e.g., the total number of degrees of freedom for the structure DDDD is $kd + kd(d+1)/2 + k - 1$. The structures are coded with four-letter strings. The letters refer, in order from left to right: first, the relation between the means' vectors of the components, which can either be equal (letter `"E"`) or unconstrained (`"D"`). Second, the relation *between* covariance matrices, which can all either be equal (`"E"`), or unconstrained (`"D"`). Third, the relation between the data vector components (corresponding to data dimensions) *within* each covariance matrix, i.e., each covariance matrix can either have all variances equal to some constant and all covariances equal to some constant (`"E"`) or can be unconstrained (`"D"`). Fourth, the covariances in each covariance matrix, which can either all be forced to equal 0 (`"0"`) or be unconstrained (`"D"`).

```
R> model.structure.m.E <- getModelStructure(mean = "E")
R> str(model.structure.m.E)
```

```
List of 4
 $ mean   : chr "E"
 $ between: chr "D"
 $ within : chr "D"
 $ cov    : chr "D"
```

Model fitting with a predefined structure can be performed by setting the `model.structure` argument in any of the fitting functions. For example, below a belief based model is created with a specified model structure.

```
R> modelBelief <- belief(X = genotypes$X, knowns = genotypes$knowns,
+    B = genotypes$B, model.structure = model.structure.m.E)
R> modelBelief$likelihood
```

```
[1] 222.498
```

Note, that in the likelihood for a model fitted to the `genotypes` data, where the component means are forced to be equal is much lower than the likelihood for an unconstrained model (equal 765.6777; see Section 2). That difference gives evidence against the equality of component's means. Similar observation can be concluded from graphical representations of the two models (compare Figures 3 with 2) or applying model selection criteria to the choice of the optimal model structure (see Section 3.3).

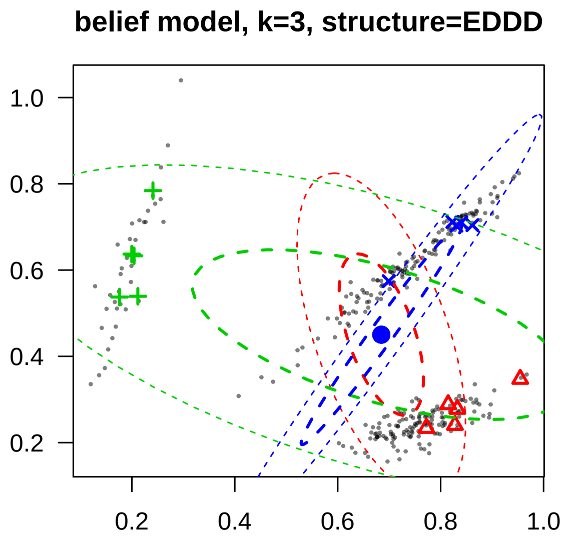

Figure 3: A three-component model fitted to the `genotypes` data by belief-based mixture modeling with forced equality of the component means. The designation of the axes and the data visualization are the same as in Figure 2. Visibly, the fitted model components do not capture the correct data clustering: instead of overlaying the real clusters in the data, the Gaussian ellipses are all centered in the same point between the clusters, and consequently spread also over the spaces between the clusters.

2.2. Prediction

The model fitted with a chosen function from the **bgmm** package can be used for classification or clustering to the most probable component. This maximum a posteriori probability (MAP) prediction of components is implemented by the overloaded function `predict()`. The first argument of this function is the fitted model, the second argument is the data.frame with the observations for which the predictions should be made. The resulting object has two slots, `tij` with the posterior probabilities calculated for the observations, and `class` with the index of the most probable component.

```
R> preds <- predict(modelSoft, X = genotypes$X, knowns = genotypes$knowns,
+    B = genotypes$B)
R> str(preds)
```

```
List of 4
 $ tij.X       : num [1:318, 1:3] 1 1 1 1 1 1 1 1 1 1 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:318] "1" "2" "3" "4" ...
  .. ..$ : NULL
 $ tij.knowns  : num [1:15, 1:3] 1 1 1 1 1 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:15] "known1" "known2" "known3" "known4" ...
  .. ..$ : NULL
 $ class.X     : Named int [1:318] 1 1 1 1 1 1 1 1 1 1 ...
  ..- attr(*, "names")= chr [1:318] "1" "2" "3" "4" ...
 $ class.knowns: Named int [1:15] 1 1 1 1 1 3 3 3 3 3 ...
  ..- attr(*, "names")= chr [1:15] "known1" "known2" "known3" "known4" ...
```

2.3. Parameter initialization

All model-fitting functions in the **bgmm** package implement an EM algorithm, which needs to be properly started from an initial set of parameters (except for `supervised()` where direct estimation is possible, see Section 2). The package provides two initialization procedures, both via a function called `init.model.params()`. The argument `method` allows to choose from the following:

1. With `method = "knowns"` the initialization is based only on the labeled observations, i.e., observations with certain or probable components assigned. The initial model parameters for each component are estimated in one step from those observations which are assigned to this component, as described for the fully supervised model estimation in Section 1.3.

2. With `method = "all"` (default) the initialization is based on all observations. In this case, to obtain the initial set of model components, we start by clustering the data using 10 iterations of the k-means algorithm (Venables and Ripley 2002). For one dimensional data the clusters are identified by dividing the data into intervals of equal sizes. Each resulting cluster corresponds to one initial model component. For the partially and semi-supervised methods, this correspondence rises a technical problem: the cluster corresponding to component y should be as close as possible to the set of observations labeled $y \in \mathcal{Y}$. Note that for the unsupervised modeling this problem is irrelevant and any cluster may be used to initialize any component.

   To label the clusters with model components, we use a greedy heuristic. The heuristic calculates weighted distances between all possible pairs (Cluster$_i$, Label$_j$), where 'Cluster$_i$' denotes a mean value of observations in cluster i while 'Label$_j$' denotes a mean value of observations with the known label $_j$. In each step, the pair with a minimal distance is chosen (in case of ties, one pair is chosen at random). For the pair with smallest distance, the observations from cluster i are labeled with the label j. Next, the cluster i is removed from the set of considered clusters and label j is removed from the set of labels. These steps are repeated until the set of labels is empty. Finally, having all clusters labeled, we provide this labeling to the fully supervised estimation of the initial model.

In the example below the initial values for model parameters are initialized based on labeled observations only. It is assumed that for every component at least one labeled observation is specified:

```
R> initial.params <- init.model.params(X = genotypes$X,
+    knowns = genotypes$knowns, class = genotypes$labels, method = "knowns")
R> str(initial.params)
```

```
List of 3
 $ pi  : num [1:3] 0.333 0.333 0.333
 $ mu  : num [1:3, 1:2] 0.84 0.206 0.811 0.278 0.626 ...
 $ cvar: num [1:3, 1:2, 1:2] 0.00464 0.00055 0.00412 0.00279 0.00191 ...
```

The obtained initial model parameters can be explicitly specified in all model-fitting functions, except for the `supervised()` function, using an argument called `init.params`, e.g:

```
R> model <- soft(X = genotypes$X, knowns = genotypes$knowns,
+    P = genotypes$B, init.params = initial.params)
```

Without this argument specified, by default the functions themselves run the initialization based on all observations (i.e., with `method = "all"`).

3. Choosing the model structure and number of components

Depending on the nature of a given mixture modeling problem, two cases are possible: the number of components and the model structure may either be (i) easy or (ii) impossible to determine *a priori*. In the second case, model selection criteria should be used to identify the best fitted model from a set of models with different structures or component numbers. We show that both these cases are supported by the **bgmm** package. Section 2 illustrates the first case, giving straightforward instructions of fitting mixture models with a user-defined number of components or constraints on the model structure. To illustrate the second case, in the following we introduce the principles of model selection in the **bgmm** package (Section 3.2). We exemplify on simulated data (Section 3.1) how to use the package to select the model that under a specified criterion is optimal from a set of models varying over a range of component numbers or a set of model structures (Sections 3.3–3.5).

3.1. Simulated dataset

The `simulateData()` function in the **bgmm** package simulates data from a given model with a specified structure. Simulated datasets are useful for the investigation of properties of different modeling methods. If the "true model" used to generate the dataset is known, the performance of the modeling methods is easy to evaluate and compare.

The `simulateData()` function takes the following arguments specifying the observations to be generated:

- An argument `d` sets the dimension of the data,
- An argument `n` sets the total number of observations, both labeled and unlabeled,
- An argument `k` sets the number of the model components,
- An argument `mu` sets a matrix with `k` rows and `d` columns, which defines the means' vectors for the corresponding model components. If not specified, its content is generated from a normal distribution $\mu_i = (\mu_{i,1}, ..., \mu_{i,d})$, $\mu_{i,j} \sim \mathcal{N}(0, 7^2)$, $\;i = 1, ..., k,\; j = 1, ..., d,$

- An argument `cvar` sets a three-dimensional array with the dimensions (`k`, `d`, `d`). If not specified, for each mixture component i, its covariance matrix is generated in three steps:

  1. Generate 2*`d` realizations of a `d`-dimensional normal distribution $k_{i,j} \sim \mathcal{N}(0, I_d)$.
  2. Calculate a (`d` $\times$ `d`) covariance matrix $K_i^\top K_i$ for these realizations, where $K_i = (k_{i,1}^\top, ..., k_{i,2d}^\top)^\top$.
  3. Scale the resulting sample covariance matrix by a factor generated from an exponential distribution $\tau_i \sim \mathcal{E}xp(1)$, i.e., $\Sigma_i = \tau_i K_i^\top K_i$.

- Arguments `mean`, `within`, `between`, `cov` set constraints of the model structure. By default, all are equal to `"D"`. If other values are set, the parameters `mu` and `cvar` are adjusted to match the constraints,

- An argument `s.pi` sets a vector of `k` probabilities, which specify a multinomial distribution over the components, from which the observations are generated. By default this distribution is uniform.

In addition, the `simulateData()` function calculates also a beliefs matrix (which can be used also as the matrix of plausibilities) for a specified number of observations. The beliefs are calculated based on the true components from which the observations were generated, and on the following function arguments:

- An argument `m` sets the number of the observations, for which the beliefs are to be calculated,

- An argument `n.labels` sets the number of components used as labels, defining the number of columns in the resulting beliefs matrix. By default `n.labels` equals `k`, but the user can specify a smaller number. For example, using this argument the user can define a scenario in which the data are generated from a mixture of three components, but only two of them are used as labels in the beliefs matrix (applied below),

- An argument `b.min` sets the belief that an observation does not belong to a component. Formally, the belief b_{ij} for the observation i to belong to component j is equal `b.min` if i is not generated from component j. Thus, the belief that i belongs to its true component is set to `1 - b.min * (n.labels - 1)`, and `b.min` is constrained that `b.min < 1/n.labels`. By default `b.min = 0.02`.

In the example below, a dataset with `k = 3` components is generated, the model parameters `mu` and `cvar` are chosen randomly, yet are given constraints that within each component the variances are all equal and the covariances are 0. The resulting dataset is presented in Figure 4 (left panel). A beliefs matrix is generated for a set of `m = 60` observations, with the columns corresponding to the first `n.labels = 2` components:

```
R> set.seed(1313)
R> simulated <- simulateData(d = 2, k = 3, n = 300, m = 60, cov = "0",
+    within = "E", n.labels = 2)
R> str(simulated)
```

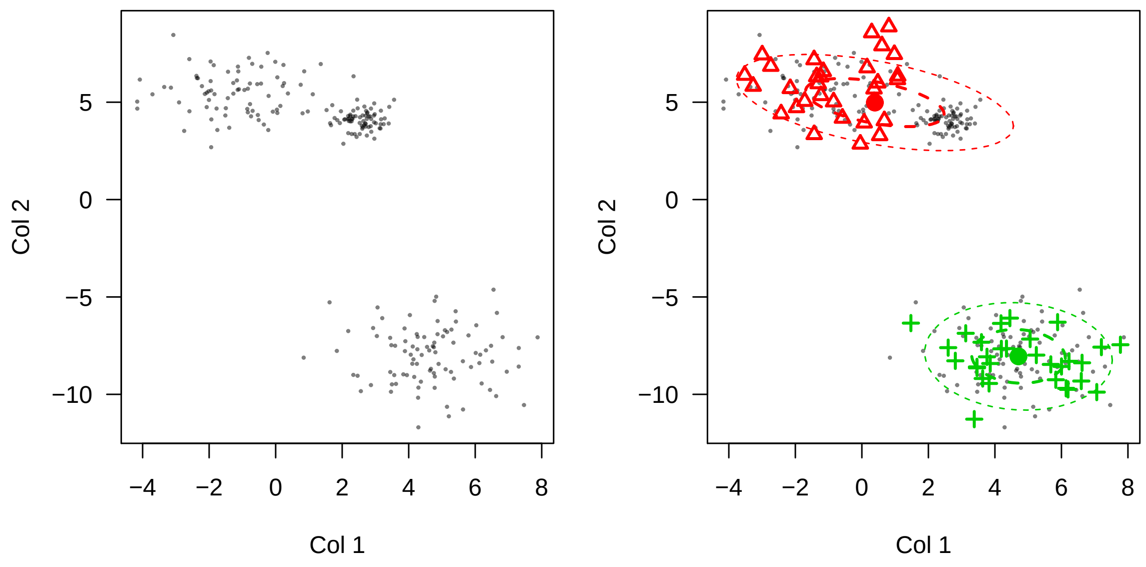

Figure 4: The left plot presents the `simulated` dataset, which was generated from a mixture of three Gaussians. The right plot presents a two-component model fitted to this dataset. The axes correspond to the two dimensions of the data. In the `simulated` dataset the column names (corresponding to the dimensions of the data) are `"Col 1"` and `"Col 2"` respectively. These names are presented as axes labels. Colored symbols stand for the component labels of the observations, for which the beliefs were generated in the `simulated` dataset. Black dots, which visibly group into three clusters, represent the remaining observations in the dataset. One of the fitted Gaussian components (the red ellipse) covers two true clusters in the data. Only one of those two clusters was generated from a component that was used as a label.

```
List of 5
 $ X           : num [1:240, 1:2] 3.42 0.37 4.28 3.89 4.27 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:2] "Col 1" "Col 2"
 $ knowns      : num [1:60, 1:2] 0.2953 6.1983 -0.0476 3.4602 0.9769 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:2] "Col 1" "Col 2"
 $ B           : num [1:60, 1:2] 0.98 0.02 0.98 0.02 0.98 0.98 0.02 ...
 $ model.params:List of 7
  ..$ pi  : num [1:3] 0.333 0.333 0.333
  ..$ mu  : num [1:3, 1:2] -0.918 4.665 2.588 5.478 -8.147 ...
  ..$ cvar: num [1:3, 1:2, 1:2] 1.572 2.165 0.219 0 0 ...
  ..$ m   : num 60
  ..$ n   : num 300
  ..$ k   : num 3
  ..$ d   : num 2
 $ Ytrue       : int [1:300] 1 2 1 2 1 1 2 2 1 2 ...
```

The result is an object of a class `simulated`, and contains the `$X, $knowns` and `$B` slots, which are the same as for objects of the `mModel` class, the slot `$model.params`, which holds the values of the above described arguments that define the simulated data, and, finally, the `$Ytrue` slot, which is a vector specifying the true components from which the observations in `$X` were generated. Any model-fitting function in the **bgmm** package (Section 1.1) can be applied to model a dataset obtained from the `simulateData()` function. Here, we apply the `belief()` function:

```
R> model <- belief(X = simulated$X, knowns = simulated$knowns,
+    B = simulated$B)
R> plot(model)
```

By default, `belief()` assumes that the number of model components equals the number of columns in the beliefs matrix specified by the argument `B`, which in this case is wrong (here, the number of components is equal to three while the number of columns in the belief matrix is equal to two). Consequently, the fitted model presented using the `plot()` function in Figure 4 (right panel), illustrates the mistake which is made when fitting only a two-component model to a dataset generated from a mixture of three Gaussians. Knowing the correct number of model components *a priori*, we could avoid this mistake by setting the argument `k = 3` in the `belief()` function. Otherwise, we can apply the model selection criteria in the **bgmm** package, which we present in application to the `simulated` dataset in the following sections.

3.2. Model selection criteria

For model comparison and selection, the **bgmm** package uses a broad class of generalized information criteria (GIC; Konishi and Kitagawa 1996). The criteria are based on a penalized likelihood and evaluate the goodness of fit of the model to the data. They allow for comparison of different models taking into the account their number of degrees of freedom. Recall from Section 2.1 and Table 1 that the number of degrees of freedom depends on the number of model components, data dimension and model structure. In its general form, the formula for the GIC scores reads:

$$GIC(\Phi) = -2l(\mathcal{X}^-, \Phi) + p|\Phi|, \tag{6}$$

where $l(\mathcal{X}^-, \Phi)$ is the likelihood of the model defined by the set of model parameters Φ, given the data $\mathcal{X}^-$, while $|\Phi|$ stands for the number of degrees of freedom of the model and p is called the penalty parameter of the GIC score. In our application, $\mathcal{X}^-$ is the set of unlabeled observations, which allows for a comparison of the models fitted by unsupervised modeling with models fitted by the semi- or partially supervised methods, putting the belief of plausibility distributions aside. We are aware that in some applications it might be advantageous to use all observations rather than only unlabeled observations in the calculation of the GIC coefficient. Thus in our functions related to GIC usage of all observations can be forced by setting parameter `whichobs = "all"`.

The lower the GIC score, the better the model. Note that for $p = 2$, the GIC score is equivalent to the Akaike's information criterion (AIC; Akaike 1974). For $p = \ln(n - m)$ it is equivalent to the Bayesian Information Criterion (BIC; Schwarz 1978; used in the **mclust** package), where $n-m$ is the number of observations in $\mathcal{X}^-$. Thus, leaving p as a parameter in the GIC formula allows the user to freely choose between applying the popular AIC, BIC, or any own or existing criteria like AIC3, AICc, AICu, CAIC, CLC, ICL-BIC, AWE (see Fonseca

2008 for list of popular criteria and references). In our simulations, models selected with BIC meet our expectations most frequently. Refer to Steele and Raftery (2009) for a discussion about the performance of different model selection criteria in the context of Gaussian mixture models.

The GIC scores computed using the **bgmm** package can be applied to select the model which fits best to a given dataset from a list of models with different structures or component numbers.

A generic function `mModelList()` in the package fits a collection of models with a given list of structures and component numbers. The following function arguments can be specified:

- An argument `funct` points to a function which will be used for model-fitting. By default is set to `belief`, other possible settings are `semisupervised`, `soft` and `unsupervised`,
- Arguments `X`, `knowns`, `B`, `P`, `class`, `init.params` are passed to the function specified by `funct`. Refer to Section 1.1 for the basic input arguments for each model-fitting function,
- An argument `kList` points to a list or a vector of component numbers for the fitted models, specifying the numbers of components to consider (exemplified in Section 3.4),
- Arguments `mean`, `between`, `within`, `cov` together define a list of model structures for the fitted models. By default the value of all these arguments is `c("D", "E")`, except for `cov`, which by default is set to `c("D", "0")`. Thus, the value of each argument specifies whether in the fitted models the corresponding constraint should be set (e.g., for `mean = "E"`), or not (`mean = "D"`), or both (`mean = c("D", "E")`). The function `mModelList()` fits a list of models with all combinations of those constraint settings. By default, models with all sixteen possible structures are fitted.

While `mModelList()` can run any model-fitting variant given by `funct`, in the **bgmm** package specialized functions for each particular modeling variant are also available: `beliefList()` fits a list of models using the belief-based method, whereas `softList()`, `semiList()` and `unsupervisedList()` use the soft-label, semi-supervised and unsupervised methods, respectively. In other words, `beliefList()` is the function `mModelList()` with the argument `funct = belief`, etc.

The `mModelList()` function and its specialized versions return an object of the class `mModelList`, with the following slots:

- An element `$models` refers to a list of the fitted models, each of the class `mModel`,
- An element `$loglikelihoods` refers to a vector with log likelihoods estimated for each of the `$models`,
- An element `$params` refers to a vector with the numbers of parameters calculated for each of the `$models`,
- An element `$names` refers to a vector with names for the `$models`. Each name defines the number of components and the structure of the model, coded by a four-letter string as described in Table 1.

On an object of the class `mModelList` the following functions can be applied:

- A function `plot()` plots all models from the list of models in the slot `$models`,
- A function `plotGIC()` draws a chart presenting the GIC scores for the list of models. Takes an argument `penalty`, which specifies the penalty parameter in the GIC formula. By default `penalty = 2`, which corresponds to the AIC score. Both numeric values and criteria names are allowed settings of the `penalty` parameter. Valid criteria names are `penalty = "AIC"`, `"AIC3"`, `"AIC4"`, `"AICc"`, `"AICu"`, `"CAIC"`, `"BIC"`, `"MDL"`, `"CLC"`, `"ICL-BIC"`, `"AWE"`.
- A function `chooseModels()` returns a subset of models, specified by two arguments: a vector `kList` with model component numbers, and a vector `struct` with four-letter strings describing the model structures (see Table 1),
- A function `chooseOptimal()` returns the model with the lowest GIC score. Takes an argument `penalty`, which specifies the penalty parameter in the GIC formula. By default `penalty = 2`, which corresponds to the AIC score.

3.3. Selecting the model structure

First, we illustrate model structure selection applying the generic `mModelList()` function (Section 3.2) to the example `simulated` dataset (Section 3.1). We fit a list of models with fixed three components and all possible structures using the belief-based method:

```
R> models1 <- mModelList(X = simulated$X, knowns = simulated$knowns,
+    B = simulated$B, kList = 3, mean = c("D", "E"), between = c("D", "E"),
+    within = c("D", "E"), cov = c("D", "0"), funct = belief)
```

Two functions may be used to visualize the resulting list of fitted models. The overloaded function `plot()` visualizes the components of all models (see Figure 5):

```
R> plot(models1)
```

The function `plotGIC()` plots the GIC scores with a specified penalty (by default equal 2) for each fitted model on a dotchart (see Figure 6):

```
R> plotGIC(models1, penalty = "BIC")
```

Here, a penalty equal $\ln(240) \approx 5.5$ is used, which corresponds to applying the BIC scores to the 240 unlabeled observations in the `simulated` dataset. Out of the set of models with all possible structures, the BIC score is distinctively the lowest for the correct model with each component having equal variances and covariances forced to 0 and other parameters unconstrained.

To select the object corresponding to the best-fitting model with respect to BIC, the `chooseOptimal()` function can be applied:

```
R> bestModel <- chooseOptimal(models1, penalty = "BIC")
```

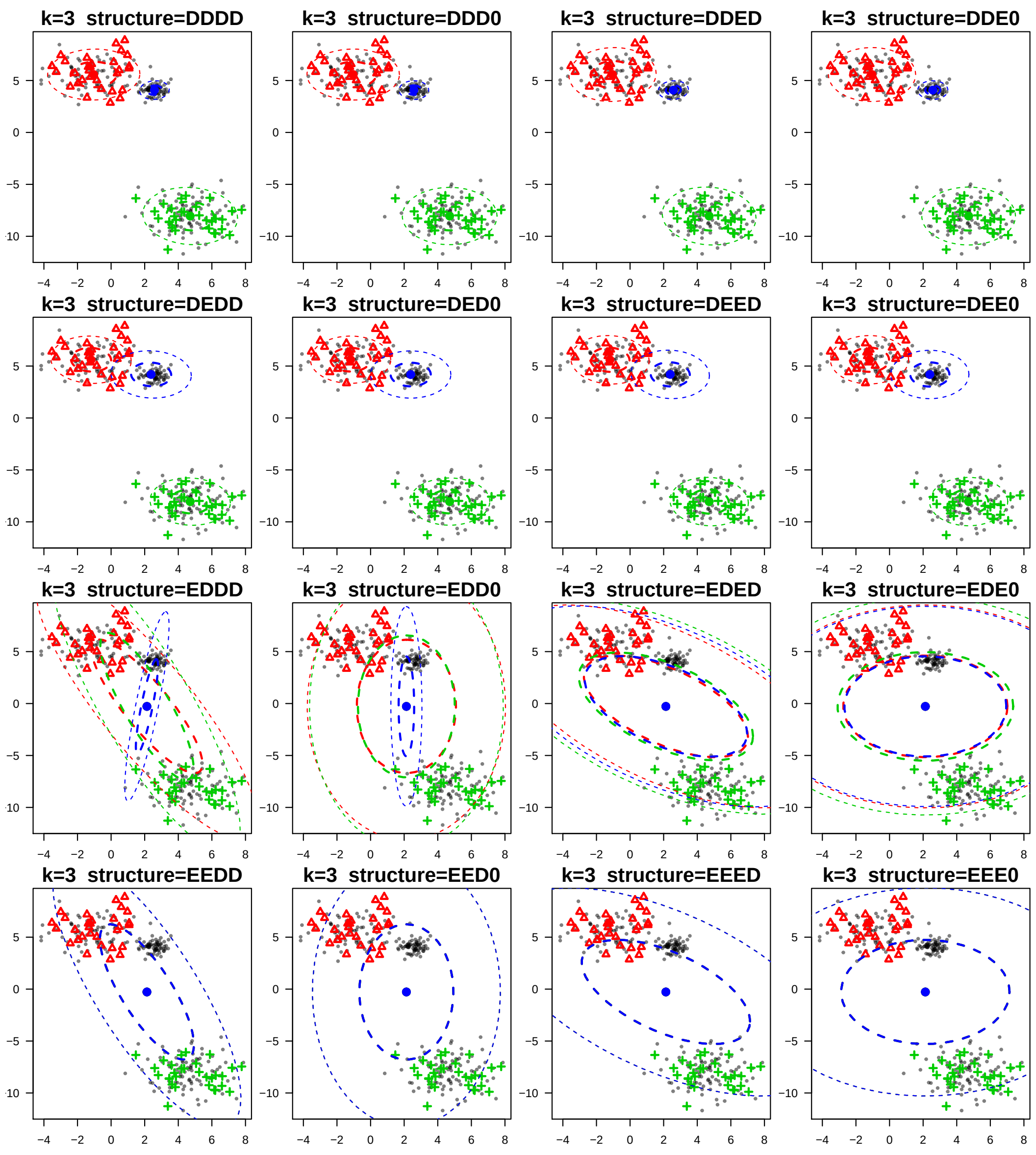

Figure 5: Models with three components and all possible model structures fitted to the `simulated` dataset. The plot axes correspond to the two dimensions of the data. Colored symbols stand for the component labels of the observations, for which the beliefs were generated in the `simulated` dataset. Black dots, which visibly group into three clusters, represent the remaining observations in the dataset. The titles state the component number and the structure of each model. The structures are coded by four-letter strings, stating, from left to right, the `mean`, `between`, `within` and `cov` constraints as described in Table 1. With the constraint `mean = "E"` (plots in the third and fourth row) the different Gaussian components have the same mean point. With `between = "E"` (second and fourth row) the components have identical covariance matrices, which implies that they have the same shape. With `within = "E"` (third and fourth column) each component has variances and covariances equal to some constant, thus the width of each corresponding eclipse is equal to its height. Finally, with the constraint `cov = "0"` (second and fourth column), each Gaussian component has all covariances equal to 0, thus the semimajor and semiminor eclipse's axes are parallel to the plot axes. Thus, with both `within = "E"` and `cov = "0"`, the Gaussians are circular (last column).

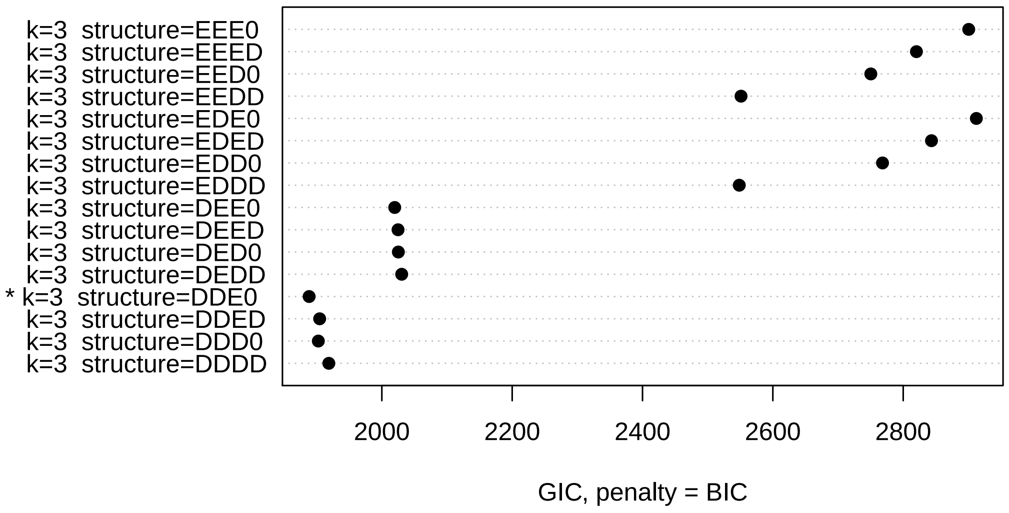

Figure 6: The GIC scores (x-axis) plotted by the `plotGIC()` function for a set of models of the `simulated` dataset. The models have three components and all possible structures (listed in the y-axis). The structures are coded by four-letter strings, stating, from left to right, the `mean`, `between`, `within` and `cov` constraints, as described in Table 1. The penalty parameter is set to the BIC scores. The optimal score is assigned to a model with the structure `"DDE0"` (marked by a star), which is the correct model from which the `simulated` dataset was generated.

The model may be validated with the k-fold cross-validation method implemented in the `crossval()` function. In the presented example dataset is randomly divided into 60 folds. Proportions between labeled and unlabeled observations are kept for each fold. For each fold model parameters are refitted base on observations from remaining folds. Fitted model is applied to the fold and posteriors for observations from the fold are calculated. The error for given fold is calculated as mean absolute difference between posteriors and beliefs for labeled cases from this fold. In this examples slot `$errors` contains 60 values with errors calculated for 60 folds.

```
R> amodel = belief(X = simulated$X, knowns = simulated$knowns,
+    B = simulated$B, k = 4)
R> summary(crossval(model = amodel, folds = 60)$errors)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.01000 0.01000 0.01000 0.04583 0.01000 0.42950
```

3.4. Selecting the number of model components

Another application of the GIC scores (see Equation 6) in the **bgmm** package is evaluation and comparison of models with different component numbers. In the case of unsupervised and semi-supervised modeling, the application of the GIC scores is straightforward. In the case of belief-based and soft-label modeling, a technical problem of scaling beliefs and plausibilities

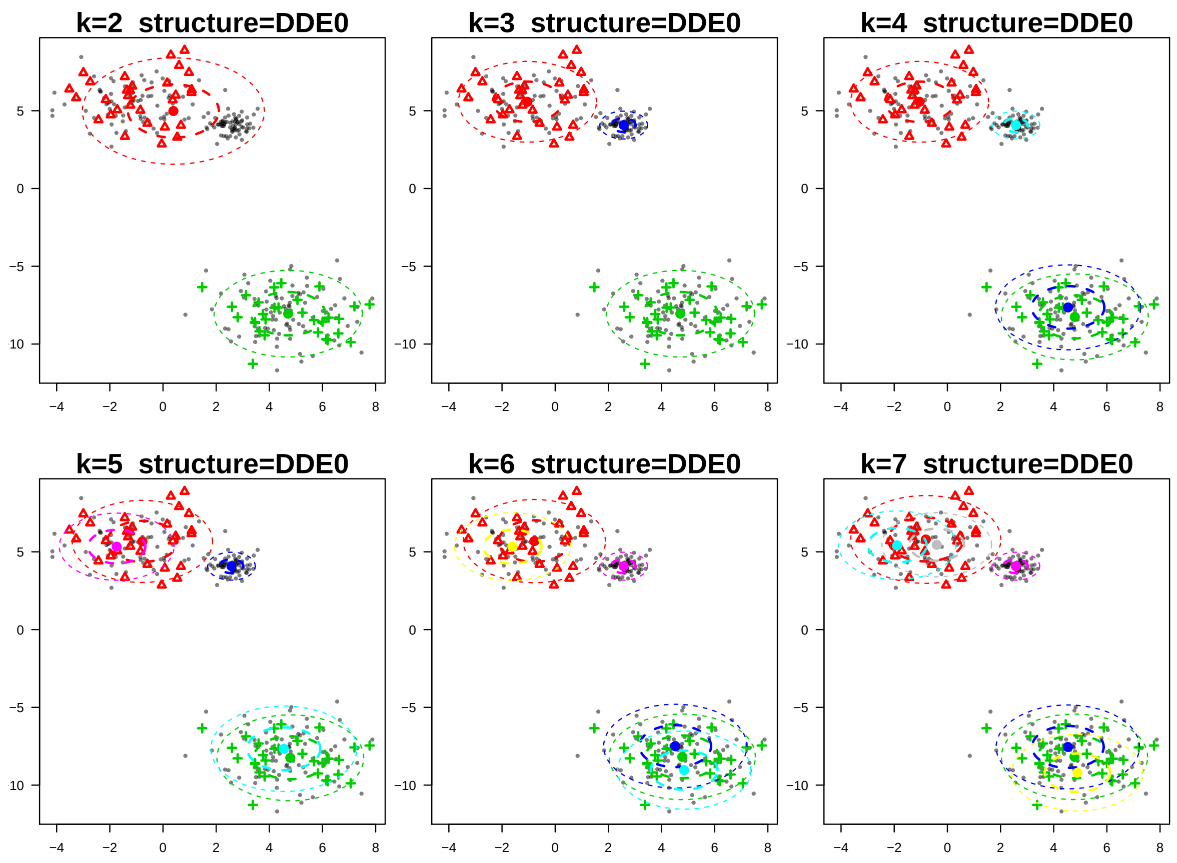

Figure 7: Models with a range of component numbers from two to seven and a fixed structure, fitted to the `simulated` dataset. Plot designation as in Figure 5. The true number of components is three. In the model with only two components (top left panel) one fitted Gaussian corresponds to two true components. In the models with more than three components, there are extra, overlapping fitted Gaussians for one true component.

between the models with different numbers of components arises. For example, in the dataset `genotypes`, the beliefs are defined for three components. Thus if one wants to fit a model with four components the beliefs distribution needs to be scaled. The user can specify beliefs for four components explicitly by providing a belief matrix. However, in an automated model selection a default procedure of scaling the original beliefs matrix to a higher number of components is required. The one implemented in the **bgmm** package is to set the belief values for the extra components to 0. This corresponds to the trivial fact that in the original labeling the observations are not believed to belong to the extra components, and neither the set of labeled observations nor the beliefs about them should change with the scaling. Scaling of plausibilities proceeds in the same way.

Note that quite another problem is to scale given beliefs or plausibilities to a smaller number of components. In the **bgmm** package it is not supported in the automated scaling, but still the user can provide their own beliefs/plausibilities matrix for a lower number of components.

In the following example, we apply the `beliefList()` function to fit a list of models to the

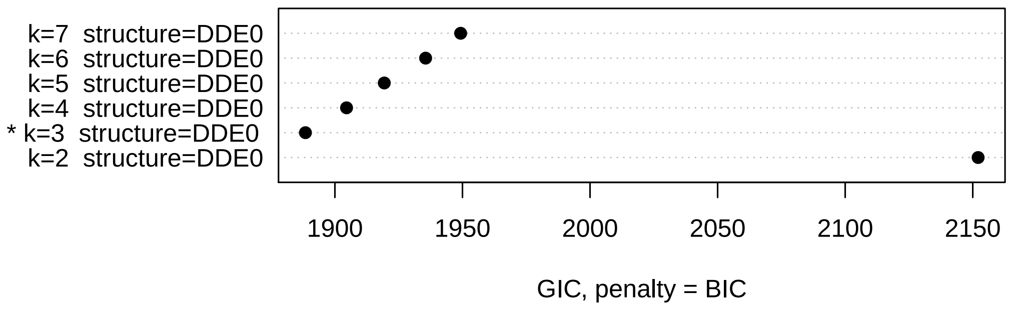

Figure 8: The GIC scores computed for models with a range of component numbers from two to seven, and with a fixed structure, fitted to the `simulated` dataset (the models are visualized in Figure 7). Plot designation as in Figure 6. The GIC scores with the penalty 5.5 (equivalent to the BIC scores) are the lowest for the model with the correct number of three components (marked by a star).

`simulated` dataset (Section 3.1). The models have the number of components ranging from two to seven, and their structure is fixed to the correct structure from which the dataset was generated. The minimum number of components that can be fitted is defined by the number of columns in the original beliefs matrix, which in the `simulated$B` matrix is equal two. For the models with higher component numbers the beliefs matrix is scaled automatically.

```
R> models2 <- beliefList(X = simulated$X, knowns = simulated$knowns,
+    B = simulated$B, kList = 2:7, mean = "D", between = "D",
+    within = "E", cov = "0")
R> plot(models2)
R> plotGIC(models2, penalty = "BIC")
```

The data in the `simulated` dataset is generated from a three-component model, and therefore groups into three clusters. Visualization of the fitted model components with the `plot()` function (Figure 7) shows that for the number of components smaller than the correct number, the fitted Gaussians may embrace more than one true cluster in one component (visualized already in Figure 3). On the contrary, for higher component numbers, extra components may be fitted to one true cluster.

Figure 8 shows that the BIC scores plotted with the `plotGIC()` function favor the model with the correct component number (equal three) over the models with an insufficient or excessive number of components.

3.5. Selecting both the model structure and the number of components

Finally, we show how to use the **bgmm** package to iterate over models with both a given list of possible model structures and a given range of component numbers. Here, an example call of the `beliefList()` function fits a list of models with the component numbers ranging from two to seven, and all possible model structures restricted only not to have equal means (together $6 \times 8 = 48$ models, a combination of six possible component numbers and eight model structures):

```
R> models3 <- beliefList(X = simulated$X, knowns = simulated$knowns,
+    B = simulated$B, kList = 2:7, mean = "D")
```

The `plotGIC()` function with the `plot.it=FALSE` argument allows to see the GIC scores for each model visualised in a table with columns corresponding to the fitted component numbers and rows corresponding to the fitted structures. The structures are coded with four-letter strings, stating, from left to right, the `mean`, `between`, `within` and `cov` constraints as described in Table 1:

```
R> plotGIC(models3, penalty = "BIC", plot.it = FALSE)
```

```
                    k=2      k=3      k=4      k=5      k=6      k=7
structure=DDDD 2071.408 1926.469 1937.027 1960.299 1988.562 2018.308
structure=DDD0 2111.656 1908.782 1930.103 1953.502 1975.321 1992.256
structure=DDED 2132.403 1910.837 1931.207 1957.607 1979.959 1996.528
structure=DDE0 2155.221 1893.123 1910.766 1928.549 1944.893 1955.601
structure=DEDD 2097.053 2035.156 2034.276 2033.243 2032.296 2036.758
structure=DED0 2124.772 2029.511 2028.761 2027.241 2029.938 2035.450
structure=DEED 2135.221 2029.017 2031.810 2035.389 2037.301 2031.719
structure=DEE0 2157.661 2023.406 2025.587 2028.336 2032.454 2030.202
```

To access a subset of twelve out of the 48 fitted models for visualization and evaluation, we apply the `chooseModels()` function:

```
R> models4 <- chooseModels(models3, kList = 2:5,
+    struct = c("DDDD", "DDED", "DDE0"))
R> plot(models4)
R> plotGIC(models4, penalty = "BIC")
```

The output of the `plot()` function in Figure 9 visualizes the chosen models. The three models with the structures `"DDDD"`, `"DDED"`, and `"DDE0"`, and with the correct number of three components (second row of Figure 9) seem to better capture the true clusters in the `simulated` dataset than models with less or more components. Still, only by the visual inspection it is hard to tell which of those models fits best to the dataset. Here, the BIC scores prove very useful in correctly identifying the best-fitting model (see output of the `plotGIC()` function in Figure 10).

3.6. Summary and discussion

The **bgmm** package offers mixture modeling variants spreading the entire range from unsupervised to supervised modeling. To our knowledge, although the most general and applicable in many real-life problems, partially supervised modeling was until now not supported by any open-access software. **bgmm** is also the only R package, which implements semi-supervised modeling. The availability of all variants allows for a comparison analysis between estimates obtained with different models. For such tests, the package offers simulation of data from the user-defined model parameters. Fitted models of up to two-dimensional data can be visualized on plots.

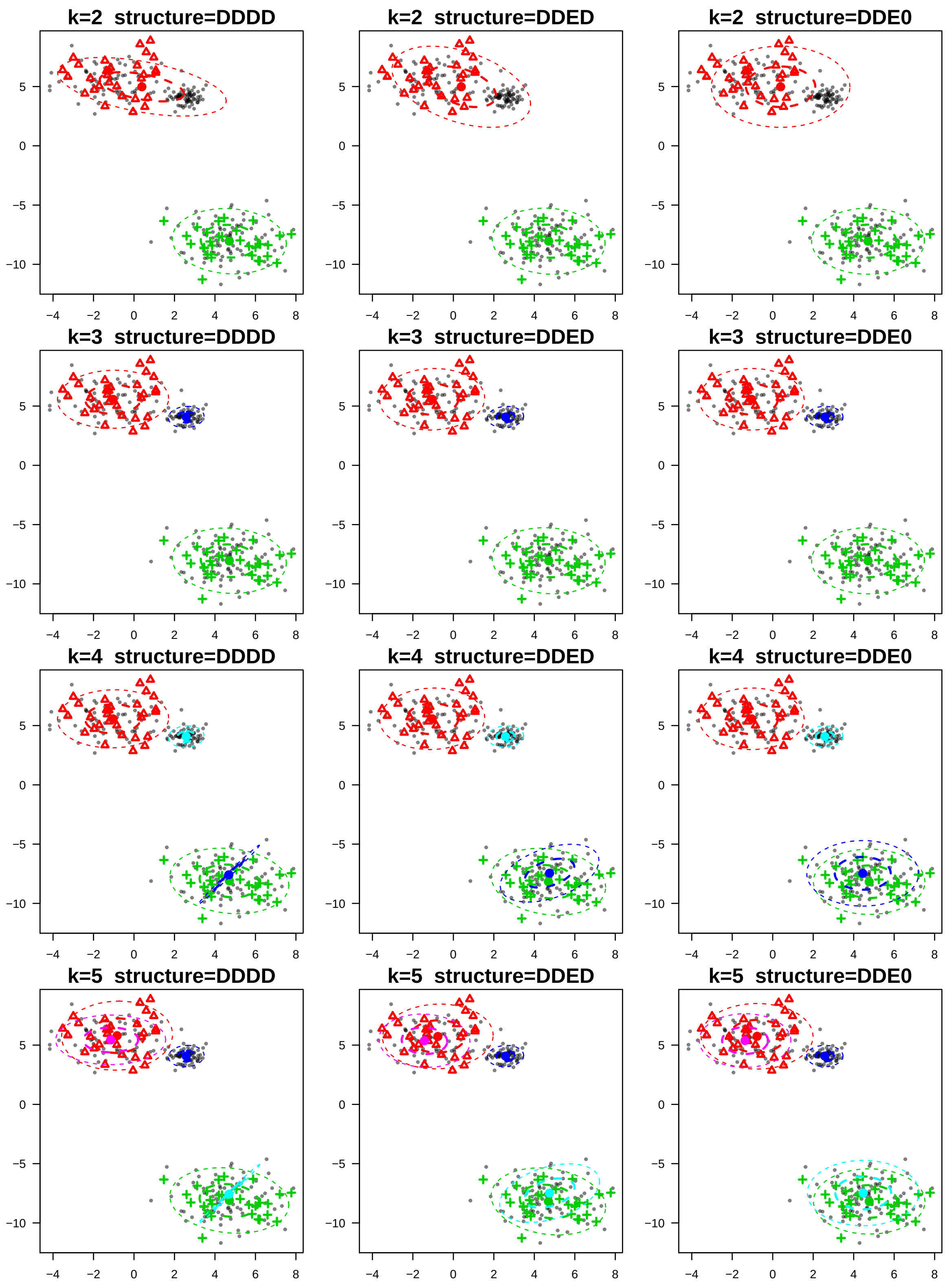

Figure 9: Models with a range of component numbers from two to five, and with one of the `"DDDD"`, `"DDED"`, and `"DDE0"` structures, fitted to the `simulated` dataset. Plot designation as in Figure 5. The true number of components is three, and the true structure is `"DDE0"`.

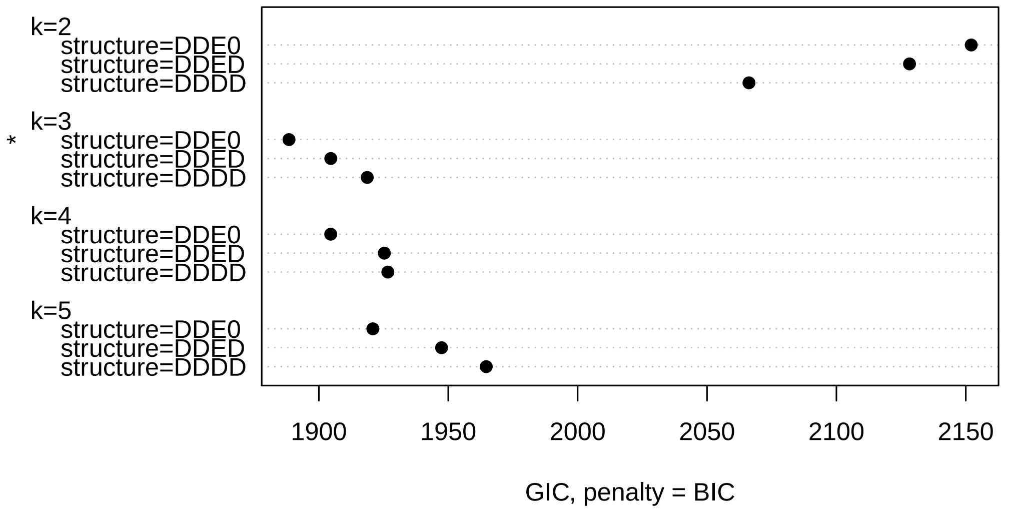

Figure 10: The GIC scores computed for models with a range of component numbers from two to five, and with one of the `"DDDD"`, `"DDED"`, and `"DDE0"` structures, fitted to the `simulated` dataset (the models are visualized in Figure 9). Plot designation as in Figure 6. The BIC scores are the lowest for the model with the correct number of three components and the correct structure `"DDE0"` (marked by a star).

Apart from the basic model-fitting functionality, **bgmm** allows for specifying constraints on the model structure, which can significantly reduce the number of fitted parameters. Goodness-of-fit of the models is evaluated using the GIC scores. The scores serve as model selection criteria in cases, when the structure or the number of model components is not known *a priori*. To this end, **bgmm** fits a set of models with different structures or component numbers, and selects the model which according to GIC fits best to the data. Finally, the model of choice can be applied to predict classes or clusters of a given set of observations.

Importantly, the implementation of the presented functionality was designed for maximal flexibility. Multiple input settings are possible for the model-fitting functions. **bgmm** provides a generic function for fitting a range of models with different component numbers or structures, using modeling variants of choice. The universal interface of **bgmm** is fully described in the package manual.

Côme *et al.* (2009) show on benchmark datasets that the partially supervised modeling implemented by the soft-label method is advantageous over unsupervised approaches. Our previous publication (Szczurek *et al.* 2010) proves both the belief-based and soft-label methods are successful in application to differential gene expression analysis. The **bgmm** package offers a practical support for Gaussian mixture modeling of one-dimensional expression data. The standard data are ratios of expression measurements in an experiment comparing some treatment with a control. This functionality of the **bgmm**, not covered here, provides calculation of differential expression probabilities for the genes measured under a given experiment. We hope that in the future more methodological advances can be made on the basis of partially supervised mixture modeling implemented in **bgmm**.

In summary, **bgmm** is practically the only tool applicable for partially labeled data, and proves to be a convenient, flexible software for mixture modeling.

Acknowledgments

The authors thank the associate editor and two anonymous referees for their remarks and suggestions that contributed to a better presentation of the paper. This work was partly supported by the Polish Ministry of Science and Education (grant NN301-065236) and the Deutsche Forschungsgemeinschaft (DFG, grant SFB618).

The first two authors (PB and ES) contributed equally to this work.

References

Akaike H (1974). "A New Look at the Statistical Model Identification." *IEEE Transactions on Automatic Control*, **19**(6), 716–723.

Ambroise C, Denœux T, Govaert G, Smets P (2001). "Learning From an Imprecise Teacher: Probabilistic and Evidential Approaches." In *Proceedings of ASMDA 01*, pp. 100–105. Compiégne, France.

Ambroise C, Govaert G (2000). "EM Algorithm for Partially Known Labels." In *Data Analysis, Classification, and Related Methods, Proceedings of the 7th Conference of the International Federation of Classication Societies (IFCS-2000)*, pp. 161–166. Springer-Verlag.

Bair E, Tibshirani R (2004). "Semi-Supervised Methods to Predict Patient Survival from Gene Expression Data." *PLoS Biology*, **2**(4), e108+. doi:10.1371/journal.pbio.0020108.

Bergé L, Bouveyron C, Girard S (2012). "**HDclassif**: An R Package for Model-Based Clustering and Discriminant Analysis of High-Dimensional Data." *Journal of Statistical Software*, **46**(6), 1–29. URL http://www.jstatsoft.org/v46/i06/.

Bouveyron C, Girard S (2009). "Robust Supervised Classification with Mixture Models: Learning from Data with Uncertain Labels." *Pattern Recognition*, **42**(11), 2649–2658.

Chen WC (2011). *Overlapping Codon Model, Phylogenetic Clustering, and Alternative Partial Expectation Conditional Maximization Algorithm*. Ph.D. thesis, Iowa State University. URL http://thirteen-01.stat.iastate.edu/snoweye/phyclust/.

Côme E, Oukhellou L, Denœux T, Aknin P (2009). "Learning from Partially Supervised Data Using Mixture Models and Belief Functions." *Pattern Recognition*, **42**(3), 334–348.

Culp M (2011). "**spa**: A Semi-Supervised R Package for Semi-Parametric Graph-Based Estimation." *Journal of Statistical Software*, **40**(10), 1–29. URL http://www.jstatsoft.org/v40/i10/.

Fonseca J (2008). "The Application of Mixture Modelind and Information Criteria for Discovering Patterns of Coronary Heart Disease." *Journal of Applied Quantitative Methods*, **3**(4), 292–202.

Fraley C, Raftery AE (2006). "**mclust** Version 3 for R: Normal Mixture Modeling and Model-Based Clustering." *Technical Report 504*, University of Washington, Department of Statistics. URL http://www.stat.washington.edu/fraley/mclust/tr504.pdf.

Hüllermeier E, Beringer J (2006). "Learning from Ambiguously Labeled Examples." *Intelligent Data Analysis*, **10**, 419–439.

Konishi S, Kitagawa G (1996). "Generalised Information Criteria in Model Selection." *Biometrika*, **83**(4), 875–890.

Lawrence N, Schölkopf B (2001). "Estimating a Kernel Fisher Discriminant in the Presence of Label Noise." *In Proceedings of 18th International Conference on Machine Learning*, pp. 306–313.

Leisch F, Grün B (2012). "CRAN Task View: Cluster Analysis & Finite Mixture Models." Version 2012-03-22, URL http://CRAN.R-project.org/view=Cluster.

Ohnishi Y, Tanaka T, Ozaki K, Yamada R, Suzuki H, Nakamura Y (2001). "A High-Throughput SNP Typing System for Genome-Wide Association Studies." *Journal of Human Genetics*, **46**, 471–477.

R Development Core Team (2012). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Schwarz G (1978). "Estimating the Dimension of a Model." *The Annals of Statistics*, **6**(2), 461–464.

Steele R, Raftery AE (2009). "Performance of Bayesian Model Selection Criteria for Gaussian Mixture Models." *Technical Report 559*, University of Washington, Department of Statistics. URL http://www.stat.washington.edu/research/reports/2009/tr559.pdf.

Szczurek E, Biecek P, Tiuryn J, Vingron M (2010). "Introducing Knowledge into Differential Expression Analysis." *Journal of Computational Biology*, **17**(8), 953–967.

Takitoh S, Fujii S, Mase Y, Takasaki J, Yamazaki T, Ohnishi Y, Yanagisawa M, Nakamura Y, Kamatani N (2007). "Accurate Automated Clustering of Two-Dimensional Data for Single-Nucleotide Polymorphism Genotyping by a Combination of Clustering Methods: Evaluation by Large-scale Real Data." *Bioinformatics*, **23**, 408–413.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S*. 4th edition. Springer-Verlag, New York.

Weston J, Elisseeff A, Bakir G, Sinz F (2010). "**The Spider**, Machine Learning Toolbox for MATLAB." URL http://people.kyb.tuebingen.mpg.de/spider/.

Weston J, Leslie C, Ie E, Zhou D, Elisseeff A, Noble WS (2005). "Semi-Supervised Protein Classification Using Cluster Kernels." *Bioinformatics*, **21**(15), 3241–3247.

Zhu X (2005). "Semi-Supervised Learning Literature Survey." *Technical report*, Computer Sciences, University of Wisconsin-Madison. URL http://www.cs.wisc.edu/~jerryzhu/pub/ssl_survey.pdf.

Zhu X, Goldberg AB (2009). *Introduction to Semi-Supervised Learning*. Morgan & Claypool Publishers.

Affiliation:

Przemysław Biecek
Institute of Applied Mathematics and Mechanics
University of Warsaw
Banach 2
02-097 Warszawa, Poland
E-mail: P.Biecek@mimuw.edu.pl

Ewa Szczurek
Max Planck Institute for Molecular Genetics
Ihnestrasse 63–73,
14195 Berlin, Germany
and
Institute of Informatics
University of Warsaw
Banacha 2
02-097 Warszawa, Poland
E-mail: szczurek@molgen.mpg.de
URL: http://bgmm.molgen.mpg.de/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 47, Issue 3 *Submitted:* 2011-02-15
April 2012 *Accepted:* 2012-01-16