MixSim: An R Package for Simulating Data to Study Performance of Clustering Algorithms

Volodymyr Melnykov  
The University of Alabama

Wei-Chen Chen  
Oak Ridge National Laboratory

Ranjan Maitra  
Iowa State University

Abstract

The R package MixSim is a new tool that allows simulating mixtures of Gaussian distributions with different levels of overlap between mixture components. Pairwise overlap, defined as a sum of two misclassification probabilities, measures the degree of interaction between components and can be readily employed to control the clustering complexity of datasets simulated from mixtures. These datasets can then be used for systematic performance investigation of clustering and finite mixture modeling algorithms. Among other capabilities of MixSim, there are computing the exact overlap for Gaussian mixtures, simulating Gaussian and non-Gaussian data, simulating outliers and noise variables, calculating various measures of agreement between two partitionings, and constructing parallel distribution plots for the graphical display of finite mixture models. All features of the package are illustrated in great detail. The utility of the package is highlighted through a small comparison study of several popular clustering algorithms.

Keywords: Gaussian mixture model, data simulation, pairwise overlap, parallel distribution plot, R.

1. Introduction

The main goal of clustering is to form groups of similar observations while also separating dissimilar ones. Many clustering algorithms have been developed, such as the iterative $k$-means (Forgy 1965; MacQueen 1967) and $k$-medoids (Kaufman and Rousseeuw 1990) algorithms, hierarchical (both agglomerative and divisive) algorithms with different merging/splitting criteria called linkages – e.g., Ward’s (Ward 1963), single (Sneath 1957), complete (Sorensen 1948) and other – and the probabilistic model-based clustering algorithms, where the observations are assumed to be sampled from an underlying finite mixture model (Melnykov and Maitra 2010). For a comprehensive review of clustering methods, see Xu and Wunsch (2009).
Clustering is a difficult problem with, consequently, many suggested methodologies, but no uniformly best performer in all situations. Thus, it is of interest to understand the scenarios in which different clustering algorithms perform better, worse, or indifferently. This requires an objective measure that would allow the clustering complexity of a simulated dataset to be controlled and different procedures to be calibrated with regard to this complexity. Performance of clustering algorithms depends on many factors such as cluster representation, orientation, elongation, multimodality, overlap, presence of noise, and others. The majority of procedures aiming to control clustering complexity focus on finding a good measure for the level of separation or overlap among clusters.

There have been many attempts made to define clustering complexity and to evaluate clustering algorithms in different settings. We refer our reader to Steinley and Henson (2005) and Maitra and Melnykov (2010) for a comprehensive review, while focusing our discussion here on a few most recent ones. Dasgupta (1999)’s $c$-separation for two $p$-dimensional Gaussian densities with mean vectors $\mu_i$ and covariance matrices $\Sigma_i$ for $i = 1, 2$ is defined as
\[
c \leq \frac{||\mu_1 - \mu_2||}{\sqrt{\max\{\lambda(p)\Sigma_1, \lambda(p)\Sigma_2\}}},
\]
where $\lambda(p)$ represents the largest eigenvalue in the corresponding covariance matrix $\Sigma_i$. $c$-separation is widely used by researchers but cannot serve as an adequate measure of the interaction between distributions as it does not take into consideration the structure of both covariance matrices relying in its inference just on the largest eigenvalue and the norm of two mean vectors. Another approach to generating clustered data was proposed by Steinley and Henson (2005). The algorithm is implemented in the MATLAB function `OCLUS` and conveniently generates clusters of different shapes from various distributions. However, there are some limitations of this approach as well: clusters are marginally independent by construction and the number of overlapping clusters is restricted. A novel separation index was proposed by Qiu and Joe (2006b) and implemented in their package `clusterGeneration` (Qiu and Joe 2006a) for R (R Development Core Team 2012). In a univariate setting, the developed index is calculated as $(q_l,2 - q_u,1)/(q_u,2 - q_l,1)$, where $q_{l,1}$ and $q_{u,1}$ are the lower and upper quantiles of the first cluster and $q_{l,2}$ and $q_{u,2}$ are the corresponding quantiles of the second cluster. The probability associated with the quantiles is an additional parameter commonly chosen to be equal to 0.05. When two clusters are distant, this index takes values close to 1. It can take negative values but no less than $-1$ for clusters that are poorly separated. The advantage of this index is its simplicity and applicability for clusters of any shapes. At the same time, while being exact in the univariate framework, this index cannot be readily generalized to the case of two and more dimensions. The authors propose finding the best projection onto a one-dimensional space that maximizes the separation of clusters first. Then, their index can be employed as in the univariate case. Unfortunately, the substantial amount of information can be lost while projecting multidimensional clusters; hence, conclusions can be partial or incorrect. Very recently, Maitra and Melnykov (2010) have provided the only known exact measure capable of measuring interaction between two clusters in terms of the pairwise overlap $\omega$, which is defined for Gaussian mixtures as the sum of two misclassification probabilities and can be calculated in a univariate as well as multivariate framework. More details about this measure can be found in Section 2. Maitra and Melnykov (2010) also developed algorithms which can be used to simulate data from a finite mixture model with specified clustering complexity. Further, they developed the open-source CARP package (Melnykov and Maitra 2011) to evaluate clustering algorithms from a command-line interface.

This paper details the use and applicability of MixSim, an R package with kernel writ-
2. Methodological and algorithmic details

This section briefly defines the basics of the overlap measure for the reader and discusses two mixture simulating schemes adopted by \textit{MixSim}. It also summarizes several indices frequently used for comparing two partitionings.

2.1. Pairwise overlap

As mentioned in Section 1, the notion of pairwise overlap was recently introduced by \textit{Maitra and Melnykov} (2010). We refer the reader to their paper for the detailed analysis of the measure and its properties. Here, we briefly explain mechanics behind the measure.

Let $X$ be distributed according to the finite mixture model $g(x) = \sum_{k=1}^{K} \pi_k \phi(x; \mu_k, \Sigma_k)$, where $\phi(x; \mu_k, \Sigma_k)$ is a multivariate Gaussian density of the $k$th component with mean vector $\mu_k$ and covariance matrix $\Sigma_k$. Then, the overlap between $i$th and $j$th components is defined as $\omega_{ij} = \omega_{ij} + \omega_{ji}$, where $\omega_{ij}$ is the misclassification probability that the random variable $X$ originated from the $i$th component but was mistakenly assigned to the $j$th component; $\omega_{ij}$ is defined similarly. Thus, $\omega_{ij}$ is given by

$$\omega_{ij} = \Pr \left[ \pi_i \phi(X; \mu_i, \Sigma_i) < \pi_j \phi(X; \mu_j, \Sigma_j) \mid X \sim N_p(\mu_i, \Sigma_i) \right].$$

Overlap has nice closed form expressions in some special cases. For example, when $\pi_i = \pi_j$ as well as $\Sigma_i = \Sigma_j \equiv \Sigma$, we obtain

$$\omega_{ij} = 2 \Phi \left( -\frac{1}{2} \sqrt{(\mu_j - \mu_i)' \Sigma^{-1} (\mu_j - \mu_i)} \right),$$

where $\Phi$ is the standard normal cumulative density function. For spherical clusters, the above reduces to $\omega_{ij} = 2 \Phi \left( -\frac{1}{2\sqrt{p}} \|\mu_i - \mu_j\| \right)$. In general, misclassification probabilities are given by

$$\omega_{ij} = \Pr_{N_p(\mu_i, \Sigma_i)} \left[ \sum_{l=1}^{p} \left( \lambda_l - 1 \right) U_l + 2 \sum_{l=1}^{p} \frac{\delta_l W_l}{\lambda_l} \leq \sum_{l=1}^{p} \frac{\lambda_l \delta_l^2}{\lambda_l - 1} - \sum_{l=1}^{p} \frac{\delta_l^2}{\lambda_l} + \log \frac{\pi_j^2}{\pi_i^2} \right],$$

where $\lambda_1, \lambda_2, \ldots, \lambda_p$ are eigenvalues of the matrix $\Sigma_i^{-\frac{1}{2}} \Sigma_j^{-1} \Sigma_i^{-\frac{1}{2}}$ and $\gamma_1, \gamma_2, \ldots, \gamma_p$ are the corresponding eigenvectors, $U_l$’s are independent noncentral $\chi^2$ random variables with one degree of freedom and noncentrality parameter given by $\lambda_l \delta_l^2 / (\lambda_l - 1)^2$ with $\delta_l = \gamma_l' \Sigma_i^{-\frac{1}{2}} (\mu_i - \mu_j)$, independent of $W_l$’s, which are independent $N(0, 1)$ random variables. This provides an efficient way of calculating $\omega_{ij}$ and $\omega_{ji}$ due to the algorithm AS155 (Davies 1980) that computes probabilities for linear combinations of noncentral $\chi^2$ random variables.
Note that if the covariance matrices are multiplied by some positive constant $c$, it causes inflation ($c > 1$) or deflation ($c < 1$) of the components. Thus, we can manipulate the value of $c$ in order to reach the pre-specified level of overlap $\omega_{ij}(c)$ between the components. According to Maitra and Melnykov (2010), the function $\omega_{ij}(c)$ does not have to be monotone increasing; however, $\omega_{ij}(c)$ enjoys monotonicity in the overwhelming number of simulated mixtures. In those rare cases when monotonicity is violated, the simplest solution is to drop the current mixture and simulate a new one.

If $c \to \infty$ and clusters are heterogeneous, the above expression reduces to

$$\omega_{j|i}^\infty = \Pr_{N_p(\mu, \Sigma)} \left[ \sum_{l=1}^{p} (\lambda_l - 1)U_l \leq \log \frac{\pi_j^2 | \Sigma_i |}{\pi_i^2 | \Sigma_j |} \right],$$

where $U_l$’s are independent central $\chi^2$ random variables with one degree of freedom.

Maitra and Melnykov (2010) do not discuss the case of homogenenous clusters but it can be also addressed as follows. If all clusters are homogeneous, the expression for $\omega_{j|i}$ reduces to

$$\omega_{j|i} = \Phi \left( -\frac{1}{2} \sum_{l=1}^{p} \delta_l^2 + \frac{\log \pi_j / \pi_i}{\sqrt{\sum_{l=1}^{p} \delta_l^2}} \right).$$

When $c \to \infty$, $\delta_l^2 \to 0$ for $l = 1, 2, \ldots, p$. This yields $\omega_{j|i}^\infty = 0$ for $\pi_j < \pi_i$, $\omega_{j|i}^\infty = \frac{1}{2}$ for $\pi_j = \pi_i$, and $\omega_{j|i}^\infty = 1$ for $\pi_j > \pi_i$. This indicates that the value of asymptotic overlap for homogeneous clusters is $\omega_{i|i}^\infty = \omega_{j|i}^\infty + \omega_{i|j}^\infty = 1$ for any mixing proportions $\pi_i$ and $\pi_j$.

2.2. Mixture model and data generation

Having detailed the pairwise overlap and its implementation in Section 2.1, the overlap can now be employed to control the degree of interaction among mixture components. Before we proceed to the next section describing algorithms for generating mixtures with a pre-specified overlap characteristic, we discuss how mixture model parameters are simulated.

Mean vectors of Gaussian mixture components $\mu_k$ are obtained as $K$ independent realizations from a uniform $p$-variate hypercube with bounds specified by the user. Covariance matrices $\Sigma_k$ are taken as draws from the Wishart distribution with parameter $p$ and $p + 1$ degrees of freedom. The low number of degrees of freedom provides us with random covariance matrices of different orientation and elongation. Finally, mixing proportions $\pi_k$ are generated on the $[0, 1]$ interval subject to the restriction $\sum_{k=1}^{K} \pi_k = 1$ with the lower bound pre-specified by the user. To simulate a dataset from a generated mixture, first, cluster sizes are obtained as a draw from a multinomial distribution based on mixing proportions. Then, the corresponding number of realizations are obtained from each multivariate normal component.

2.3. Algorithms

Maitra and Melnykov (2010) provided two algorithms that simulate Gaussian mixtures according to pre-specified values of average ($\bar{\omega}$) or/and maximum ($\hat{\omega}$) overlap. The algorithms are briefly summarized below.
Generate mixture model controlling the average or maximum overlap

The first algorithm generates a mixture model with respect to the level of average or maximum overlap. The algorithm consists of the following three steps.

1. **Generating initial parameters.** Generate $K$ mixing proportions, mean vectors and covariance matrices as discussed in Section 2.2. Compute limiting average ($\bar{\omega}^{\infty}$) (maximum ($\hat{\omega}^{\infty}$)) overlap. If $\bar{\omega} > \bar{\omega}^{\infty}$ ($\hat{\omega} > \hat{\omega}^{\infty}$), discard the realization and start Step 1 again.

2. **Calculating pairwise overlaps.** Compute all pairwise overlaps. Calculate the current estimate of $\hat{\bar{\omega}}$ ($\hat{\hat{\omega}}$). If the difference between $\hat{\bar{\omega}}$ and $\bar{\omega}$ ($\hat{\hat{\omega}}$ and $\hat{\omega}$) is negligible, stop the algorithm and provide the current parameters.

3. **Scaling clusters.** Use root-finding techniques to find a covariance matrix multiplier $c$ such that the difference between $\hat{\bar{\omega}}(c)$ and $\bar{\omega}$ ($\hat{\hat{\omega}}(c)$ and $\hat{\omega}$) is negligible.

For finding roots, MixSim obtains bounds of an interval that contains the root by considering positive or negative powers of 2, and then applies the approach of Forsythe, Malcolm, and Moler (1980) to find the root.

Generate mixture model controlling the average and maximum overlap

The second algorithm deals with both characteristics $\bar{\omega}$ and $\hat{\omega}$ simultaneously. It can be preferred over the first algorithm to better control the overlap between simulated components.

1. **Scaling clusters to reach $\hat{\omega}$.** Use the first algorithm to obtain the set of parameters that satisfies $\hat{\omega}$ and fix two components that produced the highest overlap; their covariance matrices will not be involved in inflation/deflation process.

2. **Finding $c_{\hat{\omega}}$.** Find the largest value of $c$ (say $c_{\hat{\omega}}$) such that none of pairwise overlaps $\omega_{ij}(c_{\hat{\omega}})$ exceeds $\hat{\omega}$. If $\hat{\omega}(c_{\hat{\omega}}) < \bar{\omega}$, discard the realization and return to Step 1.

3. **Limited scaling.** While keeping the two fixed components unchanged, apply Step 3 of the first algorithm to the rest of components to reach the desired $\bar{\omega}$. If the obtained parameters satisfy $\bar{\omega}$ and $\hat{\omega}$, report them. Otherwise, start with Step 1 again.

It can be noticed that not every combination of $\bar{\omega}$ and $\hat{\omega}$ can be obtained. Immediate restrictions are $\bar{\omega} \leq \hat{\omega}$ and $\hat{\omega} \leq \bar{\omega} K (K - 1)/2$, where $K$ is the number of mixture components. Also, some combinations of $\bar{\omega}$ and $\hat{\omega}$ can be more difficult to reach than the others. In this case, a higher number of mixture resimulations may be needed to find a targeted mixture.

2.4. Classification indices

One application of MixSim is the systematic investigation of the properties of clustering algorithms. In order to assess the level of similarity between partitioning vectors, some measure has to be used. There are multiple indices that have been developed for this purpose – see Meilä (2006) for a detailed review. Here, we summarize those indices that are implemented in MixSim. Meilä (2006) brackets all indices into one of three groups. The first group of indices compares clusterings counting the pairs of points that are assigned to the same or
different clusters under both partitionings. The Rand (1971) index falls into this category and is defined as
\[ R(c_1, c_2) = \frac{N_{11} + N_{00}}{\binom{n}{2}}, \]
where \( c_1 \) and \( c_2 \) are the first and second partitioning vectors respectively, \( N_{11} \) is the number of pairs of points in the same cluster under \( c_1 \) and \( c_2 \), and \( N_{00} \) is the number of pairs in different clusters under \( c_1 \) and \( c_2 \); \( n \) represents the number of points in partitioning vectors. The more commonly used modification of \( R(c_1, c_2) \) involves adjustment with its \( E(R(c_1, c_2)) \), providing the adjusted Rand index (Hubert and Arabie 1985):
\[ AR(c_1, c_2) = \frac{R(c_1, c_2) - E(R(c_1, c_2))}{1 - E(R(c_1, c_2))}. \]
Another, albeit less popular, adjustment of the Rand index was proposed by Mirkin (1996):
\[ M(c_1, c_2) = n(n - 1)(1 - R(c_1, c_2)). \]
An interesting index proposed by Fowlkes and Mallows (1983) combines two asymmetric criteria of Wallace (1983), \( W_1(c_1, c_2) \) and \( W_2(c_1, c_2) \):
\[ F(c_1, c_2) = \sqrt{W_1(c_1, c_2)W_2(c_1, c_2)}, \]
where
\[ W_i(c_1, c_2) = \frac{2N_{11}}{\sum_{k=1}^{K^{(i)}} n_k^{(i)} (n_k^{(i)} - 1)} \]
for \( i = 1, 2 \) with \( n_k^{(i)} \) representing the size of the \( k \)th cluster according to the partitioning \( c_i \) and \( K^{(i)} \) representing the number of clusters in \( c_i \). Indices \( R, AR \), and \( F \) have upper bounds equal to 1 which can be achieved only in the case of the same partitioning, i.e., \( c_1 = c_2 \). On the contrary, the Mirkin index reaches 0 for identical partitioning vectors; otherwise, it takes positive integer values. These four indices are implemented in MixSim’s function RandIndex().
A second group of indices compares partitionings by set matching. The most well-known index here is the proportion of observations that agree on classification for both partitioning vectors. It may be noted that label-switching plays an important role here as the result is label-dependent. MixSim’s function ClassProp() calculates this proportion considering all possible permutations of labels and choosing the permutation yielding the highest proportion of agreement between two partitionings. Of course, this approach becomes restrictive for a high number of classes. In this case, heuristic approaches for matching classes, such as provided by the function matchClasses() from the package e1071 (Meyer, Dimitriadou, Hornik, Weingessel, and Leisch 2012), can be applied.
The last category of indices is based on the analysis of the variation of information in two partitioning vectors. Meil˘a (2006) developed an index defined as
\[ VI(c_1, c_2) = H(c_1) + H(c_2) - 2I(c_1, c_2), \]
where \( H(c_i) \) is the entropy associated with \( c_i \) defined as
\[ H(c_i) = -\sum_{k=1}^{K^{(i)}} \frac{n_k^{(i)}}{n} \log \frac{n_k^{(i)}}{n}. \]
for $i = 1, 2$. $I(c_1, c_2)$ represents the mutual information between two partitionings and is defined as

$$I(c_1, c_2) = \sum_{k=1}^{K^{(1)}} \sum_{r=1}^{K^{(2)}} \frac{n_{kr}}{n} \log \frac{n_{kr} n}{n^{(1)} n^{(2)}},$$

where $n_{kr}$ is the number of observations being assigned simultaneously to the $k$th and $r$th clusters under partitionings $c_1$ and $c_2$ respectively. When $c_1 = c_2$, $VI(c_1, c_2) = 0$. The upper bound for $VI$ is equal to $\log n$. The $\text{MixSim}$ function responsible for calculating $VI$ is called $\text{VarInf()}$. It is worth pointing out that all the indices listed above and implemented in $\text{MixSim}$ are symmetric, which means that for any index, $\text{Index}(c_1, c_2) = \text{Index}(c_2, c_1)$.

### 3. Package description and illustrative examples

In this section we provide a detailed description of $\text{MixSim}$’s capabilities, along with illustrations. First, we briefly summarize $\text{MixSim}$’s functionality which includes the following features:

- simulating Gaussian mixture models with homogeneous or heterogeneous, spherical or ellipsoidal covariance matrices according to the pre-specified level of average or/and maximum overlap;
- simulating datasets from Gaussian mixtures;
- calculating pairwise overlap for Gaussian mixture components;
- simulating outliers, noise and inverse Box-Cox transformed variables;
- constructing parallel distribution plots to display multivariate Gaussian mixtures;
- calculating various indices for measuring the classification agreement between two examined partitionings.

The complete list of functions included in the package along with their brief descriptions can be found in Table 1. More details and illustrative examples for each function are provided in the following sections. Our illustrative examples can be run by the function $\text{demo()}$ in all considered cases. For instance, the code of the first example in Section 3.1 can be reproduced using the command $\text{demo("sec3.1_ex1", package = "MixSim")}$. If there are several

| Function       | Description                                                                 |
|----------------|------------------------------------------------------------------------------|
| MixSim()       | Simulates a mixture with pre-specified level of $\omega$ or/and $\hat{\omega}$ |
| overlap()      | Calculates the exact overlap given the parameters of a mixture              |
| simdataset()   | Simulates datasets given the parameters of a mixture                        |
| pdplot()       | Constructs a parallel distribution plot                                      |
| RandIndex()    | Calculates Rand, adjusted Rand, Fowlkes-Mallows, and Merkin indices         |
| ClassProp()    | Calculates the agreement proportion between two classification vectors       |
| VarInf()       | Calculates the variation of information for two classification vectors       |
| perms()        | Returns all permutations given the number of elements                         |

Table 1: Summary of functions implemented in $\text{MixSim}$. 
Section Demo names
Section 3.1 "sec3.1_ex1", "sec3.1_ex2", "sec3.1_ex3"
Section 3.2 "sec3.2_ex1"
Section 3.3 "sec3.3_ex1a", "sec3.3_ex1b", "sec3.3_ex2c", "sec3.3_ex2d"
"sec3.3_ex3a", "sec3.3_ex3b", "sec3.3_ex4c", "sec3.3_ex4d"
Section 3.4 "sec3.4_ex1", "sec3.4_ex2b", "sec3.4_ex2c"
Section 4 "sec4_ex1"

Table 2: Summary of demos included in MixSim.

plots produced in the example, the plot name is also included in the demo name, for example: demo("sec3.3_ex1a", package = "MixSim"). The names of all demo programs are provided in Table 2.

3.1. Simulating mixtures with the function MixSim()

MixSim() is the main function of the package and is responsible for finding a Gaussian mixture model satisfying the user-specified level of average or/and maximum overlap. The command has the following syntax:

MixSim(BarOmega = NULL, MaxOmega = NULL, K, p, sph = FALSE, hom = FALSE, ecc = 0.90, PiLow = 1.0, int = c(0.0, 1.0), resN = 100, eps = 1e-06, lim = 1e06)

The parameters of the function are listed in Table 3.

When both parameters BarOmega and MaxOmega are specified, the second algorithm from Section 2.3 is run. If only one of the above parameters is provided, the first algorithm from Section 2.3 is employed. The smallest allowed number of components K is 2; in this case, BarOmega ≡ MaxOmega. MixSim() allows the simulation of mixtures with spherical or general covariance structure as well as with homogenous or nonhomogeneous components. In order to better control the shape of produced components, the parameter ecc specifying the maximum eccentricity can be used. Maitra and Melnykov (2010) defined the multidimensional eccentricity by extending its definition for two-dimensional ellipses: $e = \sqrt{1 - \lambda_{\text{min}}/\lambda_{\text{max}}}$ with $\lambda_{\text{min}}$ and $\lambda_{\text{max}}$ being correspondingly the smallest and largest eigenvalues of the (covariance) matrix. If some simulated dispersion matrices have $e > e_{\text{max}}$ specified by ecc, all eigenvalues will be scaled in order to have $e_{\text{new}} = e_{\text{max}}$. PiLow controls the minimum of the mixing proportions, with PiLow = 1 indicating equality of all mixing proportions. Option int specifies a side of a hypercube on which mean vectors are generated. If int is not provided, the default interval is (0, 1). The last three options resN, eps, and lim described in Table 3 specify technical parameters used in MixSim(). resN sets the maximum number of resimulations allowed when the desired overlap cannot be reached easily. This happens, for example, when asymptotic maximum or average overlap is lower than the corresponding value specified by the user. eps represents the error bound used in our root-finding procedure and Davies (1980)’s algorithm approximating the distribution of a linear combination of $\chi^2$ random variables based on the numerical inversion of the characteristic function. Finally, lim specifies the maximum number of terms allowed in numerical integration involved in Davies (1980)’s procedure. We next illustrate three sample usages of MixSim().
Arguments Description

BarOmega Average pairwise overlap $\bar{\omega}$
MaxOmega Maximum pairwise overlap $\bar{\omega}$
K Number of mixture components
p Number of dimensions
sph Nonspherical (FALSE) or spherical (TRUE) mixture components
hom Nonhomogeneous (FALSE) or homogeneous (TRUE) mixture components
ecc Maximum eccentricity
PiLow Smallest mixing proportion
int Side of a hypercube for simulating mean vectors
resN Maximum number of mixture resimulations
eps Error bound
lim Maximum number of integration terms according to Davies (1980)

Values Description

$\Pi$ Vector of mixing proportions
$\Mu$ Mean vectors
$\Sigma$ Covariance matrices
$\OmegaMap$ Map of misclassification probabilities
$\BarOmega$ Average pairwise overlap $\bar{\omega}$
$\MaxOmega$ Maximum pairwise overlap $\bar{\omega}$
$\rcMax$ Index of the pair of clusters producing maximum overlap
$\fail$ Flag of successful completion

Table 3: Summary of available arguments and values returned by the function `MixSim()`.

Simulating a mixture with non-homogeneous mixture components, equal mixing proportions and pre-specified values of maximum and average pairwise overlap

Here, we illustrate the use of `MixSim()` in simulating a 5-dimensional mixture with 4 components, and average and maximum overlaps equal to 0.05 and 0.15 correspondingly. The following example ("sec3.1_ex1") beginning with `set.seed(1234)` for reproducibility illustrates the use of the function. Since the parameter `PiLow` is not specified, the mixture model has equal mixing proportions. As options `sph` and `hom` are not provided, nonhomogeneous and general (ellipsoidal) covariance matrices are simulated.

```r
R> set.seed(1234)
R> (ex.1 <- MixSim(BarOmega = 0.05, MaxOmega = 0.15, K = 4, p = 5))
K = 4, p = 5, BarOmega = 0.04999942, MaxOmega = 0.1499996, success = TRUE.
Pi:
[1] 0.25 0.25 0.25 0.25
Mu:
           p.1     p.2       p.3      p.4      p.5
K.1 0.8446347 0.0295568 0.59976935 0.26841977 0.12060890
K.2 0.1007055 0.74816114 0.01596063 0.04946115 0.74762379
```

R> set.seed(1234)
R> (ex.1 <- MixSim(BarOmega = 0.05, MaxOmega = 0.15, K = 4, p = 5))
MixSim: Simulating Data for Clustering Algorithms in R

K.3 0.3572377 0.75895820 0.37595634 0.79946271 0.02569277
K.4 0.5063586 0.82122865 0.54475658 0.26668445 0.34463732

S: ... too long and skipped. Use operator $ to access.

R> summary(ex.1)

OmegaMap:

|   | k.1       | k.2       | k.3       | k.4       |
|---|-----------|-----------|-----------|-----------|
| k.1| 1.0000000000  0.0004152306 0.002332968 0.01901556 |
| k.2| 0.0009103698 1.0000000000 0.014849260 0.08575316 |
| k.3| 0.0031164483 0.0079691289 1.000000000 0.02517269 |
| k.4| 0.0355134505 0.0642464525 0.040701777 1.00000000 |

rcMax: 2 4

From the output, we can see that mixture components with numbers 2 and 4 provide the largest overlap (see vector ex.1$rcMax). The corresponding probabilities of misclassification can be found in the matrix ex.1$OmegaMap: they are 0.0642 and 0.0858. The map of misclassification probabilities as well as the numbers of components producing the largest pairwise overlap can be conveniently accessed by function summary(). Both desired values, $\bar{\omega}$ and $\bar{\bar{\omega}}$, have been reached within the error bound as we can see from ex.1$BarOmega and ex.1$MaxOmega correspondingly.

Simulating a mixture with non-homogeneous spherical mixture components, unequal mixing proportions and pre-specified value of maximum pairwise overlap

The following example ("sec3.1_ex2") simulates a bivariate mixture with three spherical components, mixing proportions no less than 0.1 and maximum pairwise overlap 0.1. Parameter hom is not provided, thus nonhomogeneous components are simulated.

R> set.seed(1234)
R> (ex.2 <- MixSim(MaxOmega = 0.1, K = 3, p = 2, sph = TRUE, PiLow = 0.1))

K = 3, p = 2, BarOmega = 0.03672386, MaxOmega = 0.09999973, success = TRUE.

Pi:

[1] 0.1048634 0.4386534 0.4564832

Mu:

|   | p.1       | p.2       |
|---|-----------|-----------|
| k.1| 0.2325505 0.6660838 |
| k.2| 0.5142511 0.6935913 |
| k.3| 0.5449748 0.2827336 |

S: ... too long and skipped. Use operator $ to access.

R> summary(ex.2)
OmegaMap:

```
 k.1    k.2    k.3
k.1   1.0000000e+00 0.075664063 4.520316e-05
k.2   2.433567e-02 1.000000000 6.629426e-03
k.3   9.497808e-06 0.003487719 1.0000000e+00
```

rcMax: 1 2

As we can see from the above output, the desired maximum overlap has been reached. It is produced by the components 1 and 2 with misclassification probabilities 0.0243 and 0.0757 respectively.

Simulating a mixture with homogeneous spherical components, equal mixing proportions and pre-specified value of average pairwise overlap

The last illustration ("sec3.1_ex3") of the function MixSim() deals with a 4-dimensional mixture with 2 components. The average overlap is specified at the level 0.05. To increase accuracy, we let `eps` be equal to 1e-10. Coordinates of mean vectors are simulated between 0 and 10. Since arguments `sph` and `hom` are specified as `TRUE`, clusters should be spherical and homogeneous.

```
R> set.seed(1234)
R> (ex.3 <- MixSim(BarOmega = 0.05, K = 2, p = 4, sph = TRUE, hom = TRUE,
+     int = c(0, 10), eps = 1e-10))
```

```
K = 2, p = 4, BarOmega = 0.05, MaxOmega = 0.05, success = TRUE.
Pi: [1] 0.5 0.5
Mu:
  p.1  p.2  p.3  p.4
K.1   1.137034 6.222994 6.09274733 6.233794
K.2   8.609154 6.403106 0.09495756 2.325505
S: ... too long and skipped. Use operator $ to access.
R> summary(ex.3)
OmegaMap:

```
 k.1    k.2
k.1   1.000 0.025
k.2   0.025 1.000
```

rcMax: 1 2

As we can see from the obtained output, the average and maximum overlaps provided in `ex.3$BarOmega` and `ex.3$MaxOmega` respectively are equal to each other. It happens because
MixSim: Simulating Data for Clustering Algorithms in R

Table 4: Summary of available arguments and values returned by the function overlap().

| Arguments | Description |
|-----------|-------------|
| Pi        | Vector of mixing proportions |
| Mu        | Mean vectors |
| S         | Covariance matrices |
| eps       | Error bound |
| lim       | Maximum number of integration terms according to Davies (1980) |

| Values     | Description |
|-----------|-------------|
| $\OmegaMap$ | Map of misclassification probabilities |
| $\bar{\Omega}$ | Average pairwise overlap $\bar{\omega}$ |
| $\max\Omega$ | Maximum pairwise overlap $\tilde{\omega}$ |
| $rcMax$   | Index of the pair of clusters producing maximum overlap |

we have only two mixture components. Misclassification probabilities provided in the matrix ex.3$\OmegaMap$ are both equal to 0.025 because the components are homogeneous and mixing proportions are equal to each other. From the output, we can also see the effect of the increased accuracy.

3.2. Calculating exact overlap with the function overlap()

In this section we discuss the capability of MixSim to calculate the exact overlap when given the parameters of a Gaussian mixture model. This feature is useful if it is desired to estimate the level of clustering complexity for an existing classification dataset. This is another important application of MixSim. There exist numerous classification datasets widely used for testing clustering algorithms. However, the knowledge about these datasets is usually very limited: the investigator typically knows only which clusters are problematic having no information about the level of interaction among the clusters. Function overlap() provides the user with the map of misclassification probabilities. The command has the following syntax:

\[
\text{overlap}(Pi, Mu, S, eps = 1e-06, lim = 1e06)
\]

The parameters accepted by overlap as well as values returned by the function are provided in Table 4. All five arguments – Pi, Mu, S, eps, and lim – have the same meaning as in the function MixSim() discussed in Section 3.1. The returned values are also discussed in the same section.

Finding exact overlap for the Iris dataset

Here, we analyze the celebrated Iris dataset of Anderson (1935) and Fisher (1936). The data consist of 150 4-dimensional points measuring the width and length of petals and sepals; there are 50 observations from each of three different species of Iris: Setosa, Versicolor, and Virginica. It is well-known that Virginica and Versicolor are difficult to separate while Setosa creates a very distinct cluster. The following code ("sec3.2_ex1") estimates the parameters of a Gaussian mixture model with three components for Iris, provided the true classification vector. Then, the exact overlap is calculated using the function overlap().

\[
\text{R} \text{> data("iris", package = "datasets")}
\text{R} \text{> p <- ncol(iris) - 1}
\]
R> id <- as.integer(iris[, 5])
R> K <- max(id)
R> Pi <- prop.table(tabulate(id))
R> Mu <- t(sapply(1:K, function(k){ colMeans(iris[id == k, -5]) }))
R> S <- sapply(1:K, function(k){ var(iris[id == k, -5]) })
R> dim(S) <- c(p, p, K)
R> overlap(Pi = Pi, Mu = Mu, S = S)

$\Omega_{Map}
\begin{bmatrix}
[,1] & [,2] & [,3] \\
[1,] & 1.000000e+00 & 7.201413e-08 & 0.00000000 \\
[2,] & 1.158418e-07 & 1.000000e+00 & 0.02302315 \\
[3,] & 0.000000e+00 & 2.629446e-02 & 1.00000000 \\
\end{bmatrix}$

$\bar{\Omega}$
[1] 0.01643926

$\max\Omega$
[1] 0.0493176

$rcMax$
[1] 2 3

As we can see from the output, the substantial maximum overlap of 0.0493 is produced by the second and third components: Virginica and Versicolor. At the same time, these two clusters almost do not overlap with Setosa. This explains why the majority of clustering methods prefer two-cluster solutions combining Virginica and Versicolor together.

3.3. Simulating datasets with the function simdataset()

In this section, we discuss some capabilities of MixSim for simulating datasets, along with outliers, from a given (perhaps simulated) mixture model. The function responsible for data generation is called simdataset() and has the following form:

\[
\text{simdataset}(n, \Pi, \mu, S, n\text{-}noise = 0, n\text{-}out = 0, \alpha = 0.001, \\
max\text{-}out = 100000, \text{int} = \text{NULL}, \lambda = \text{NULL})
\]

The arguments and returned values are listed in Table 5. Parameters \(\Pi\), \(\mu\), and \(S\) have the same meaning as before. The size of a generated dataset is defined as \(n + n\text{-}out\), where \(n\text{-}out\) specifies the number of outliers needed. If the parameter \(n\text{-}out\) is not specified, no outliers are produced by simdataset. Parameter max.out specifies the maximum number of outlier resimulations with the default value of 1e05. \(\alpha\) specifies ellipsoidal contours beyond which outliers have to be simulated. The number of dimensions for the dataset is defined as \(\text{dim}(\mu)[2] + n\text{-}noise\). By default, \(n\text{-}noise = 0\). The interval int defines a side of a hypercube for outlier simulation. It is also used for simulating noise variables if n.noise is greater than 0. Both outliers and noise variables are simulated from a uniform hypercube. When int is not provided, the interval bounds are chosen to be equal to the smallest and
MixSim: Simulating Data for Clustering Algorithms in R

### Arguments Description

| Arguments | Description |
|-----------|-------------|
| \text{n}  | Sample size |
| \text{Pi} | Vector of mixing proportions |
| \text{Mu} | Mean vectors |
| \text{S}  | Covariance matrices |
| \text{n.noise} | Number of noise variables |
| \text{n.out} | Number of outliers |
| \text{alpha} | Level for 1 - alpha contour for simulating outliers |
| \text{max.out} | Maximum number of trials to simulate outliers |
| \text{int} | Interval to simulate outliers or/and noise variables |
| \text{lambda} | Vector of coefficients for an inverse Box-Cox transformation |

### Values Description

| Values | Description |
|--------|-------------|
| \$X   | Produced dataset |
| \$id  | Classification vector for the produced dataset |

Table 5: Summary of available arguments and values returned by the function \text{simdataset}().

The largest coordinates in mean vectors correspondingly. The last parameter, \text{lambda}, specifies a vector of size \text{dim(Mu)[2]} + \text{n.noise} that performs an inverse Box-Cox transformation for every coordinate. By default, \text{simdataset()} generates datasets from Gaussian mixture models. If the user wants to produce data that are not normally distributed, the argument \text{lambda} can be helpful. The procedure uses the multivariate Box-Cox transformation given by \( x^* = (x^\lambda - 1)/\lambda \), where \( x \) and \( x^* \) represent the original and transformed observations, respectively. \( x^* \) is approximately normally distributed for some value \( \lambda \). We use an idea related to inverting the Box-Cox transformation to provide non-normally distributed data. The transformation we propose is given by \( x = (\lambda x^* + 1)^{\frac{1}{\lambda}} - 1 \). When \( \lambda = 1 \), the identity transformation is employed. Thus, the parameter \text{lambda} specifies desired transformations for each coordinate.

The following examples illustrate the capabilities of the function.

#### Simulating datasets from Gaussian mixtures

The following code illustrates how \text{simdataset()} can be used to simulate data from the mixture obtained by MixSim. Here, we obtain a two-dimensional mixture with 5 components, average overlap 0.05 and maximum overlap 0.20. Then, 500 observations are simulated from the mixture. Two samples obtained this way are provided in Figure 1a,b. The following commands ("sec3.3_ex1a") construct the Figure 1a:

```R
R> set.seed(1234)
R> Q <- MixSim(MaxOmega = 0.20, BarOmega = 0.05, K = 5, p = 2)
R> A <- simdataset(n = 500, Pi = Q$Pi, Mu = Q$Mu, S = Q$S)
R> colors <- c("red", "green", "blue", "brown", "magenta")
R> par(mar = c(0.1, 0.1, 0.1, 0.1))
R> plot(A$X, col = colors[A$id], pch = 19, cex = 0.8, + xlab = "", ylab = ", axes = FALSE)
R> box()
```
Figure 1: Simulated datasets obtained (a–b) from Gaussian mixture models and (c–d) by an inverse Box-Cox transformation.

The code for the Figure 1b is identical with the only difference in the seed value. The user needs to run `set.seed(1235)`.

**Simulating datasets with inverse Box-Cox transformation**

The following short example ("sec3.3_ex1c") provides an illustration to an inverse Box-Cox transformation. First, we simulate a bivariate mixture with three components and maximum
MixSim: Simulating Data for Clustering Algorithms in R

overlap 0.1; mean vectors are simulated from a hypercube with a side (0, 1). Then, a dataset with 300 observations is simulated. The dataset is transformed using coefficients provided by option lambda. Two illustrative plots are included in Figure 1c,d. In both cases, we can see that obtained clusters do not follow patterns typical for normal distributions. The plots can be constructed running the following commands:

```r
R> set.seed(1238)
R> Q <- MixSim(MaxOmega = 0.1, K = 3, p = 2, int = c(0.2, 1))
R> A <- simdataset(n = 300, Pi = Q$Pi, Mu = Q$Mu, S = Q$S,
+   lambda = c(0.1, 10))
R> colors <- c("red", "green", "blue")
R> par(mar = c(0.1, 0.1, 0.1, 0.1))
R> plot(A$X, col = colors[A$id], pch = 19, cex = 0.8,
+   xlab = "", ylab = "", axes = FALSE)
R> box()
```

To obtain the Figure 1d, use `lambda = c(10, 10)` instead of `lambda = c(0.1, 10)` used for the construction of the Figure 1c. This is the only change needed in the above code.

**Simulating datasets with outliers**

If it is desired to include outliers into a simulated dataset to increase clustering complexity or check the performance of clustering algorithms in the presence of scatter, the corresponding option `n.out` has to be specified. The following example demonstrates how `simdataset()` can be employed for simulating datasets with outlying observations. First, bivariate normal mixtures with 4 components and average overlap 0.01 are simulated similar to the previous examples. Then, the function `simdataset()` is employed to generate a dataset of 500 observations and introduce 10 outliers. Figure 2a,b demonstrates two datasets obtained this way. Red color points represent outlying observations. `id` is equal to 0 for such points. To obtain the plots, the user needs to run the following commands ("sec3.3_ex3a").

```r
R> set.seed(1234)
R> Q <- MixSim_BarOmega = 0.01, K = 4, p = 2)
R> A <- simdataset(n = 500, Pi = Q$Pi, Mu = Q$Mu, S = Q$S, n.out = 10)
R> colors <- c("red", "green", "blue", "brown", "magenta")
R> par(mar = c(0.1, 0.1, 0.1, 0.1))
R> plot(A$X, col = colors[A$id+1], pch = 19, cex = 0.8,
+   xlab = "", ylab = "", axes = FALSE)
R> box()
```

In order to obtain the Figure 2b, the seed 1237 has to be used: `set.seed(1237)`.

**Simulating datasets with noise variables**

To increase the complexity of clustering or test procedures reducing dimensionality, it can be needed to simulate datasets with noise variables. Here, we illustrate the use of `simdataset()` for simulating data from a one-dimensional mixture with 4 components and maximum overlap 0.1. One noise variable is added after that. As we can see from Figure 2c,d, it is obvious
that the second variable introduces substantial clustering complexity. The Figure 2c can be obtained by running the following code ("sec3.3_ex4c").

```
R> set.seed(1235)
R> Q <- MixSim(MaxOmega = 0.1, K = 4, p = 1)
R> A <- simdataset(n = 300, Pi = Q$Pi, Mu = Q$Mu, S = Q$S, n.noise = 1)
R> colors <- c("red", "green", "blue", "brown")
R> par(mar = c(0.1, 0.1, 0.1, 0.1))
```
MixSim: Simulating Data for Clustering Algorithms in R

R> plot(A$X, col = colors[A$id], pch = 19, cex = 0.8,
+       xlab = "", ylab = "", axes = FALSE)
R> box()

For the Figure 2d, the seed has to be changed to 1236.

3.4. Constructing a parallel distribution plot with the function `pdplot()`

The next function illustrated here is `pdplot()` – parallel distribution plot – described in detail by Maitra and Melnykov (2010). This plot is a convenient tool for visualizing multidimensional mixture models and components’ interaction. The plot displays the principal components of a Gaussian mixture model. The first several components can serve as a good indicator of the major source of variability in a mixture. Table 6 lists the arguments acceptable in `pdplot()`. \( \text{Pi}, \text{Mu}, \text{and} \ S \) are the usual mixture parameters. \text{file} is the name of the pdf-file the plot has to be written to. Parameters \( \text{Nx} \) and \( \text{Ny} \) provide the numbers of horizontal and vertical smoothing intervals respectively. \( \text{MaxInt} \) sets the level of maximum color intensity while \text{marg} specifies plot margins. The function call has the following form:

\[
pdplot(\text{Pi}, \text{Mu}, \text{S}, \text{file} = \text{NULL}, \text{Nx} = 5, \text{Ny} = 5, \text{MaxInt} = 1, \text{marg} = c(2, 1, 1, 1))
\]

We illustrate the use of the function with several examples.

Parallel distribution plot for the Iris dataset

For the Iris dataset, we first estimate mixture parameters using the true classification vector. Then, we employ `pdplot()` function to construct a parallel distribution plot and save it into the file “Iris.pdf”. The obtained plot is presented in Figure 3a. It can be clearly seen that the blue and green components have substantial overlap being close to each other in every principal component while the red one is well separated. It agrees well with our findings in Section 3.2. To construct the plot, the user has to estimate the mixture parameters using the code from the example in Section 3.2 first, and then to run the following command:

\[
R> pdplot(\text{Pi} = \text{Pi}, \text{Mu} = \text{Mu}, \text{S} = \text{S})
\]

Parallel distribution plot for simulated mixtures

| Arguments | Description                      |
|-----------|----------------------------------|
| Pi        | Vector of mixing proportions     |
| Mu        | Mean vectors                     |
| S         | Covariance matrices              |
| file      | Name of the output pdf-file      |
| Nx        | Number of horizontal smoothing regions |
| Ny        | Number of vertical smoothing regions |
| MaxInt    | Maximum color intensity          |
| marg      | Plot margins                     |

Table 6: Summary of available arguments in the function `pdplot()`.
This example illustrates the difference between parallel distribution plots for well and poorly separated mixtures. We simulate two 4-dimensional mixtures with 6 components and average overlap 0.001 and 0.05 correspondingly. Figure 3b,c provides their parallel distribution plots. As can be seen, there is substantial difference between patterns in pictures. In the plot b, between-cluster variability dominates over within-cluster variability. This is an indication of well-separated clusters. The plot c suggests that in-cluster variability dominates over between-cluster variability implying larger overlap. The two plots can be easily reproduced with the following code ("sec3.4_ex1b" and "sec3.4_ex1c").

```
R> set.seed(1234)
R> Q <- MixSim(BarOmega = 0.001, K = 6, p = 4)
R> pdplot(Pi = Q$Pi, Mu = Q$Mu, S = Q$S)
R> set.seed(1232)
R> Q <- MixSim(BarOmega = 0.05, K = 6, p = 4)
R> pdplot(Pi = Q$Pi, Mu = Q$Mu, S = Q$S)
```

### 3.5. Calculating indices with functions `RandIndex()`, `ClassProp()`, `VarInf()`

`MixSim` implements all indices described in Section 2.4. Three functions responsible for them are `RandIndex()`, `ClassProp()`, and `VarInf()`. Table 7 provides their brief description. The first function returns values of four indices based on counting the pairs of points in clusterings, `ClassProp()` calculates the agreement proportion for two classifications, while the latter one, `VarInf()`, computes the variation in information for two clusterings. The functions have the following syntax:

```
RandIndex(id1, id2)
ClassProp(id1, id2)
VarInf(id1, id2)
```

The following examples are illustrations of the use of all the three functions.

---

**Figure 3:** Parallel distribution plots for (a) *Iris* dataset, (b) 6-component mixture with well-separated clusters ($\bar{\omega} = 0.001$), and (c) 6-component mixture with substantial overlap ($\bar{\omega} = 0.05$).
MixSim: Simulating Data for Clustering Algorithms in R

| Function          | Values | Description                  |
|-------------------|--------|------------------------------|
| RandIndex()       | $R$    | Rand index                   |
|                   | $AR$   | Adjusted Rand index          |
|                   | $F$    | Fowlkes and Mallows index    |
|                   | $M$    | Mirkin index                 |
| ClassProp()       |        | Agreement proportion in classification vectors |
| VarInf()          |        | Variation in information of classification vectors |

Table 7: Summary of values returned by functions RandIndex(), ClassProp(), and VarInf(). All three functions take two arguments id1 and id2 with the first and second classification vector, respectively.

R> id1 <- c(1, 1, 1, 1, 2, 2, 2, 3, 3, 3)
R> id2 <- c(1, 1, 1, 2, 2, 2, 3, 2, 3, 3)
R> RandIndex(id1, id2)

$R
[1] 0.6888889

$AR
[1] 0.2045455

$F
[1] 0.4166667

$M
[1] 28

R> ClassProp(id1, id2)

[1] 0.7

R> VarInf(id1, id2)

[1] 1.213685

3.6. Other auxiliary capabilities

The last function discussed in this section is perms(), which returns all possible permutations given the number of elements in a vector. Although this function is not directly related to other capabilities of MixSim, it is needed in the construction of ClassProp() since it is a faster implementation of the capability also provided by permn() available from package combinat (Chasalow 2012). The function has the following syntax:

perms(n)

A small example below illustrates the use of the function.
R> perms(3)

[,1] [,2] [,3]
[1,] 1 2 3
[2,] 1 3 2
[3,] 2 1 3
[4,] 2 3 1
[5,] 3 1 2
[6,] 3 2 1

4. Illustrative use of the package

This section provides a brief illustration of the utility of MixSim. Here, we investigate and compare performances of four clustering methods realized in R. The first method is model-based clustering by means of the EM algorithm, the second and third ones are $k$-means and partitioning around medoids algorithms respectively, while the last method is hierarchical clustering with Ward’s linkage. The algorithms are tested on datasets simulated from 4-dimensional mixtures with 6 heterogeneous non-spherical components and equal mixing proportions. The value of average overlap $\bar{\omega}$ varies from extreme (0.4) to very low (0.001). For each overlap, 100 mixtures are simulated with MixSim(), then a dataset of size 200 is generated from each mixture using function simdataset(). It is worth mentioning that we obtain exactly one dataset from each simulated mixture. Indeed, one can simulate more than one set from a mixture and even multiple datasets from the only mixture model. In this experiment, however, we are interested in examining as widely varying datasets as possible, subject to the only condition that simulated mixture models satisfy the pre-specified degree of average overlap. Variation in data patterns is substantially higher when we use different mixtures. Hence, we simulated just one set from each mixture, although in some other problems, a different strategy can be preferred.

The performance of methods is evaluated based on the adjusted Rand index, proportion of correct classifications, and variation in information for two partitionings. We use the original true and estimated classifications obtained from each clustering method to calculate the value of each index. Functions RandIndex(), ClassProp(), and VarInf() were employed. We calculate sample mean $\mu$ and standard deviation $s$ for each characteristic over 100 simulated datasets for every level of $\bar{\omega}$. The results are provided in Table 8. Function Mclust() from the package mclust (Fraley and Raftery 2006) was employed for the EM algorithm. Functions kmeans() and hclust() were used for $k$-means and hierarchal clustering algorithms. The partitioning around medoids algorithm was employed by means of the function PAM() from the package cluster (Mächler, Rousseeuw, Struyf, Hubert, and Hornik 2012). The experiment can be reproduced by running the demo "sec4_ex1" or by using the code provided in the supplementary materials.

The obtained results are summarized in Table 8. As we can see, there is no uniform winner among the considered four clustering methods. The results suggest that $k$-means algorithm should be preferred over the rest of the field for substantial and moderate overlaps. The model-based clustering, however, is the clear winner for well-separated clusters ($\bar{\omega} \leq 0.01$). Hierarchical clustering with Ward’s linkage slightly outperforms $k$-means for well-separated
| Model-based | \( \bar{\omega} \) | 0.400 | 0.300 | 0.250 | 0.200 | 0.150 | 0.100 | 0.050 | 0.010 | 0.005 | 0.001 |
|-------------|----------------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
|             | \( \mu \)     | 0.057  | 0.106  | 0.143  | 0.200  | 0.272  | 0.409  | 0.608  | 0.900  | 0.938  | 0.986  |
|             | \( s \)       | 0.024  | 0.043  | 0.054  | 0.061  | 0.064  | 0.086  | 0.081  | 0.055  | 0.044  | 0.021  |
|             | \( P \)       | 0.321  | 0.761  | 1.185  | 0.454  | 0.520  | 0.621  | 0.768  | 0.951  | 0.970  | 0.993  |
|             | \( s \)       | 0.032  | 3.863  | 5.506  | 0.504  | 0.555  | 0.78   | 0.665  | 0.038  | 0.032  | 0.017  |
|             | \( VI \)      | 2.972  | 2.737  | 2.579  | 2.382  | 2.086  | 1.640  | 1.064  | 0.278  | 0.181  | 0.044  |
|             | \( s \)       | 0.125  | 0.178  | 0.200  | 0.193  | 0.200  | 0.247  | 0.186  | 0.118  | 0.099  | 0.049  |
| k-means     | \( AR \)      | 0.082  | 0.141  | 0.188  | 0.240  | 0.309  | 0.423  | 0.611  | 0.865  | 0.919  | 0.974  |
|             | \( s \)       | 0.026  | 0.033  | 0.042  | 0.047  | 0.045  | 0.070  | 0.066  | 0.059  | 0.043  | 0.038  |
|             | \( P \)       | 0.349  | 0.406  | 0.443  | 0.493  | 0.552  | 0.634  | 0.772  | 0.929  | 0.959  | 0.986  |
|             | \( s \)       | 0.035  | 0.042  | 0.048  | 0.048  | 0.048  | 0.071  | 0.060  | 0.052  | 0.037  | 0.034  |
|             | \( VI \)      | 2.938  | 2.681  | 2.480  | 2.280  | 2.002  | 1.635  | 1.077  | 0.362  | 0.232  | 0.077  |
|             | \( s \)       | 0.117  | 0.129  | 0.155  | 0.157  | 0.133  | 0.196  | 0.161  | 0.119  | 0.091  | 0.074  |
| PAM         | \( AR \)      | 0.075  | 0.128  | 0.179  | 0.229  | 0.295  | 0.412  | 0.593  | 0.864  | 0.918  | 0.974  |
|             | \( s \)       | 0.023  | 0.031  | 0.043  | 0.052  | 0.051  | 0.069  | 0.074  | 0.051  | 0.039  | 0.020  |
|             | \( P \)       | 0.340  | 0.390  | 0.436  | 0.483  | 0.540  | 0.623  | 0.758  | 0.933  | 0.962  | 0.989  |
|             | \( s \)       | 0.034  | 0.038  | 0.048  | 0.052  | 0.057  | 0.069  | 0.068  | 0.038  | 0.025  | 0.009  |
|             | \( VI \)      | 2.995  | 2.740  | 2.533  | 2.322  | 2.065  | 1.679  | 1.126  | 0.373  | 0.245  | 0.085  |
|             | \( s \)       | 0.114  | 0.121  | 0.153  | 0.177  | 0.157  | 0.194  | 0.184  | 0.112  | 0.100  | 0.059  |
| Ward        | \( AR \)      | 0.070  | 0.125  | 0.168  | 0.219  | 0.286  | 0.403  | 0.586  | 0.866  | 0.922  | 0.980  |
|             | \( s \)       | 0.025  | 0.035  | 0.042  | 0.051  | 0.053  | 0.070  | 0.070  | 0.062  | 0.045  | 0.022  |
|             | \( P \)       | 0.335  | 0.389  | 0.425  | 0.476  | 0.533  | 0.619  | 0.753  | 0.935  | 0.963  | 0.991  |
|             | \( s \)       | 0.033  | 0.038  | 0.047  | 0.047  | 0.052  | 0.064  | 0.065  | 0.037  | 0.028  | 0.011  |
|             | \( VI \)      | 2.971  | 2.716  | 2.536  | 2.330  | 2.060  | 1.665  | 1.108  | 0.360  | 0.222  | 0.064  |
|             | \( s \)       | 0.128  | 0.139  | 0.154  | 0.177  | 0.168  | 0.190  | 0.166  | 0.142  | 0.102  | 0.061  |

Table 8: Performance of the model-based, k-means, partitioning around medoids, and hierarchical clustering with Ward’s linkage algorithms in clustering 4-dimensional datasets of size 200 with 6 groups. \( \bar{\omega} \) and \( s \) represent the sample mean and standard deviation for the adjusted Rand index (\( AR \)), proportion of correct classifications (\( P \)), and variation in information (\( VI \)) obtained over 100 replications.

clusters but loses to it in the cases of substantial and moderate overlap. Overall, k-means performs slightly better than the partitioning around medoids algorithm. All three indices considered agree on the suggested conclusions. Of course, this is a small example demonstrating the potential of MixSim, but it illustrates the utility of the package well.

### 5. Summary

This paper describes the R package MixSim which provides a convenient and friendly tool for simulating Gaussian mixture models and datasets according to the pre-specified level of clustering complexity expressed in terms of the average and maximum pairwise overlap among mixture components. MixSim’s functions are described and carefully illustrated on multiple examples. A small study illustrating the utility of the package is provided. The package is meant to be of interest to a broad audience working in the area of machine learning, clustering and classification.
Acknowledgments

The authors acknowledge partial support by the National Science Foundation CAREER Grant # DMS-0437555. We also thank the associate editor and reviewers whose comments and suggestions substantially improved the paper.

References

Anderson E (1935). “The Irises of the Gaspe Peninsula.” Bulletin of the American Iris Society, 59, 2–5.

Chasalow S (2012). combinat: Combinatorics Utilities. R package version 0.0-8, URL http://CRAN.R-project.org/package=combinat.

Dasgupta S (1999). “Learning Mixtures of Gaussians.” In Proceedings of the IEEE Symposium on Foundations of Computer Science, pp. 633–644. New York.

Davies R (1980). “The Distribution of a Linear Combination of χ² Random Variables.” Applied Statistics, 29, 323–333.

Fisher RA (1936). “The Use of Multiple Measurements in Taxonomic Poblems.” The Annals of Eugenics, 7, 179–188.

Forgy E (1965). “Cluster Analysis of Multivariate Data: Efficiency vs. Interpretability of Classifications.” Biometrics, 21, 768–780.

Forsythe G, Malcolm M, Moler C (1980). Computer Methods for Mathematical Computations. Mir, Moscow.

Fowlkes E, Mallows C (1983). “A Method for Comparing Two Hierarchical Clusterings.” Journal of American Statistical Association, 78, 553–569.

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, Seattle, WA.

Hubert L, Arabie P (1985). “Comparing Partitions.” Journal of Classification, 2, 193–218.

Kaufman L, Rousseuw PJ (1990). Finding Groups in Data. John Wiley & Sons, New York.

Mächler M, Rousseuw P, Struyf A, Hubert M, Hornik K (2012). cluster: Cluster Analysis Basics and Extensions. R package version 1.14.3, URL http://CRAN.R-project.org/package=cluster.

MacQueen J (1967). “Some Methods for Classification and Analysis of Multivariate Observations.” Proceedings of the Fifth Berkeley Symposium, 1, 281–297.

Maitra R, Melnykov V (2010). “Simulating Data to Study Performance of Finite Mixture Modeling and Clustering Algorithms.” Journal of Computational and Graphical Statistics, 19(2), 354–376.
MixSim: Simulating Data for Clustering Algorithms in R

Meilă M (2006). “Comparing Clusters – An Information Based Distance.” *Journal of Multivariate Analysis*, 98, 873–895.

Melnykov V, Maitra R (2010). “Finite Mixture Models and Model-Based Clustering.” *Statistics Surveys*, 4, 80–116.

Melnykov V, Maitra R (2011). “CARP: Software for Fishing Out Good Clustering Algorithms.” *Journal of Machine Learning Research*, 12, 69 – 73.

Meyer D, Dimitriadou E, Hornik K, Weingessel A, Leisch F (2012). *e1071: Misc Functions of the Department of Statistics (e1071)*, TU Wien. R package version 1.6-1, URL http://CRAN.R-project.org/package=e1071.

Mirkin B (1996). *Mathematical Classification and Clustering*. Kluwer Academic Press, Dordrecht.

Qiu W, Joe H (2006a). “Generation of Random Clusters with Specified Degree of Separation.” *Journal of Classification*, 23, 315–334.

Qiu W, Joe H (2006b). “Separation Index and Partial Membership for Clustering.” *Computational Statistics & Data Analysis*, 50, 585–603.

Rand WM (1971). “Objective Criteria for the Evaluation of Clustering Methods.” *Journal of the American Statistical Association*, 66, 846–850.

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/.

Sneath P (1957). “The Application of Computers to Taxonomy.” *Journal of General Microbiology*, 17, 201–226.

Sorensen T (1948). “A Method of Establishing Groups of Equal Amplitude in Plant Sociology Based on Similarity of Species Content and Its Application to Analyses of the Vegetation on Danish Commons.” *Biologiske Skrifter*, 5, 1–34.

Steinley D, Henson R (2005). “OCLUS: An Analytic Method for Generating Clusters with Known Overlap.” *Journal of Classification*, 22, 221–250.

Wallace D (1983). “Comment on “A Method for Comparing Two Hierarchical Clusterings”.” *Journal of American Statistical Association*, 78, 569–576.

Ward JH (1963). “Hierarchical Grouping to Optimize an Objective Function.” *Journal of the American Statistical Association*, 58, 236–244.

Xu R, Wunsch DC (2009). *Clustering*. John Wiley & Sons, Hoboken.
Affiliation:
Volodymyr Melnykov
Department of Information Systems, Statistics, and Management Science
The University of Alabama
Tuscaloosa, AL 35487, United States of America
E-mail: vmelnykov@ua.edu
URL: http://cba.ua.edu/personnel/vmelnykov

Wei-Chen Chen
Computer Science and Mathematics Division
Oak Ridge National Laboratory
Oak Ridge, TN 37831, United States of America
E-mail: wccsnow@gmail.com
URL: http://thirteen-01.stat.iastate.edu/snoweye

Ranjan Maitra
Department of Statistics and Statistical Laboratory
Iowa State University
Ames, IA 50011, United States of America
E-mail: maitra@iastate.edu
URL: http://maitra.public.iastate.edu