Binarize

Tamara J. Blätte, Florian Schmid, Stefan Mundus, Ludwig Lausser, Martin Hopfensitz, Christoph Müssel and Hans A. Kestler

July 2, 2015

Contents

1 Introduction

`Binarize` is an R package that provides three different methods for the binarization of one-dimensional data. The package also offers an assessment of the quality of the computed binarization as well as two different visualization functions for the results.

Binarization is the process of dividing data into two groups and assigning one out of two values to all the members of the same group. This is usually accomplished by defining a threshold t and assigning the value 0 to all the data points below the threshold and 1 to those above it. Binarization can thus be formulated as a threshold function $f : \mathbb{R} \to \mathbb{B}$ with

$$f(u) = \begin{cases} 0 & u \leq t \\ 1 & u > t \end{cases}$$

Accordingly, the binarized vector that corresponds to a real valued vector $\mathbf{u} = (u_1, \ldots, u_N) \in \mathbb{R}^N$ is $\mathbf{b} = (f(u_1), \ldots, f(u_N))$. It is the aim of any binarization method that uses such a threshold t to find the one that most appropriately splits the data in two. A correct binarization is especially important considering that binarization is often part of the initial preprocessing of a dataset. This data preprocessing is the basis for all the methods and analyses that follow.

One important application of binarization is the reconstruction of Boolean networks from data. These networks model gene regulation by representing each gene as a binary variable to illustrate the two states "active/transcribed" and "inactive/not transcribed" [3, 4]. For gene expression data to be used in the reconstruction of such a network, the data has to be binarized beforehand [5, 6, 8, 9]. However, a binarization that does not correctly represent the distribution of the original data will have misleading effects on the resulting network. The `Binarize` package addresses this problem by offering a quality assessment of the computed binarization. Based on this assessment, genes that are only poorly binarizeable can be excluded from the network reconstruction. In this way, the quality of the reconstruction can be improved.

The following sections describe three different binarization algorithms that each take a different approach to this common objective of finding an appropriate threshold and assessing the quality of the respective binarization.

2 Methods in the package

The package comprises three binarization methods that are associated with statistical testing procedures for quality control. These methods are described in the following.

2.1 Binarization by k-means clustering

Binarization can be seen as a clustering problem: The aim is to sort the measurements into two groups in an unsupervised manner. A common clustering algorithm is the k-means algorithm [7] (with $k = 2$) which is also used by the `Binarize` package. In general, this algorithm starts by randomly picking k initial

cluster centers from the original data and then assigning the remaining points to the closest cluster center. In several consecutive cycles the algorithm then updates this clustering by calculating the mean of each cluster's data, redefining the cluster centers to these mean values and reassigning all data points to the closest updated cluster center.

In this way the algorithm tries to minimize the sum over the squared distance of the data points to their respective cluster center. This sum is given by

$$\sum_{j=1}^{k} \sum_{u_i \in C_j} (u_i - \mu_j)^2,$$

where C_j is the j-th Cluster, u_i is one value of the input vector, and μ_j is the center of the j-th cluster. For binarization purposes the number of clusters k is set to 2.

2.2 Binarization Across multiple SCales (BASC)

The BASC algorithms [2] are binarization techniques that aim at determining a robust binarization by analyzing the data at multiple scales. This packages implements two BASC variants called BASC A and BASC B. The BASC algorithms consider the sorted input values $\left(u_{(1)}, \ldots, u_{(N)}\right)$ as a step function and determine step functions with fewer steps that approximate this original function.

A step function on a sorted input vector $\mathbf{u}$ can be generally defined as a function $s : (0, N] \to \left[u_{(1)}, u_{(N)}\right]$ with (sorted) discontinuity positions $d_{(i)} \in \mathcal{D} \subseteq \{1, \ldots, N-1\}$

$$s(x) = \sum_{i=1}^{|\mathcal{D}|+1} u_{(i)} \cdot \mathbb{I}_{\left[d_{(i-1)} < x \leq d_{(i)}\right]},$$

where $d_{(0)} = 0$ and $d_{(|\mathcal{D}|+1)} = N$.

Both algorithms can be divided into three steps:

1. *Compute a series of step functions:* From an initial step function that corresponds to the sorted measurements, step functions with fewer discontinuities are calculated. BASC A determines step functions that minimize the Euclidean distance to this initial step function. BASC B obtains step functions by smoothening the input function in a scale-space manner.

2. *Find the strongest discontinuity in each step function:* The strongest discontinuity of each step function corresponds to a potential binarization threshold and is determined by a high jump size (derivative) in combination with a low approximation error.

3. *Estimate location and variation of the strongest discontinuities:* Based on these estimates, a statistical testing procedure can reject input vectors that yield an unreliable binarization.

Computing a series of step functions

Starting with the initial step function f given by the sorted input vector $\mathbf{u} \in \mathbb{R}^N$, the BASC algorithms determine step functions with fewer discontinuities.

BASC A computes a set of $N-2$ successive step functions with a decreasing number of discontinuities. For each number of discontinuities n, it chooses the step function $s_n^* \in \mathcal{S}_n$ that minimizes the approximation error to the the initial step function f:

$$s_n^* = \arg\min_{s_n \in \mathcal{S}_n} \sqrt{\sum_{x=1}^{N} (f(x) - s_n(x))^2}.$$

Here, $\mathcal{S}_n$ denotes the set of all possible step functions of n discontinuities based on the input vector $\mathbf{u}$. For the identification of the optimal step functions s_n^*, BASC A uses a dynamic programming strategy.

BASC B differs from BASC A in the way the step functions with a lower number of steps are calculated from the initial step function. First, this algorithm calculates the slopes of the original step function with $\triangle(x) = f'(x) = f(x+1) - f(x)$, $x \in \mathbb{N}$. After that, a set of smoothed versions is calculated by convolving $f'(x)$ with a kernel $T_\sigma(n)$ based on I_n, the modified Bessel function of order n:

$$\begin{aligned} \triangle_\sigma(x) &= \sum_{k=-\infty}^{\infty} \triangle(x) \cdot T_\sigma(x-k), \text{ with} \\ T_\sigma(n) &= e^{-2\sigma} \cdot I_n(2\sigma), \end{aligned}$$

Here, $\sigma \in \mathbb{R}$ denotes the smoothing parameter. The smoothed version of the step function itself can be constructed from its smoothed derivative as follows:

$$f_\sigma(x) = u_{(1)} + \sum_{n=1}^{\lceil x \rceil - 1} \triangle_\sigma(n)$$

BASC B utilizes the maxima M_σ of a smoothed slope function as discontinuity positions for constructing a step function s_σ

$$M_\sigma = \{x \mid \triangle_\sigma(x-1) < \triangle_\sigma(x) \wedge \triangle_\sigma(x+1) < \triangle_\sigma(x)\}.$$

A variety of step functions is generated by utilizing different smoothing parameters $\sigma_1, \ldots, \sigma_S$. Possible duplicates are eliminated.

Finding the strongest discontinuity in each step function

Once the series of step functions is calculated, both BASC algorithms continue in the same way. Every discontinuity $d_{(i)} \in \mathcal{D}$ of an optimal step function is scored according to two criteria. The first criterion is the jumping height $h(d_{(i)})$ at the discontinuity

$$h(d_{(i)}) = \frac{1}{|\mathcal{N}_{i+1}|} \sum_{n \in \mathcal{N}_{i+1}} f(n) - \frac{1}{|\mathcal{N}_i|} \sum_{n \in \mathcal{N}_i} f(n)$$

where

$$\mathcal{N}_i = \left\{ n \in \mathbb{N} : d_{(i-1)} < n \leq d_{(i)} \right\}$$

are the indices of the points from the original vector $\mathbf{u}$ that are located between the $(i-1)$-th and the i-th discontinuity. In this context, $f(d_{(i)})$ denotes the original step function in case of BASC A and the smoothed version of the original step function $f_\sigma(d_{(i)})$ for the current σ in case of BASC B.

The second criterion is the approximation error $e(d_{(i)})$ of a possible binarization threshold to the initial step function

$$e(d_{(i)}) = \sum_{n=1}^{N} \left(f(n) - z(d_{(i)}) \right)^2 \quad \text{with} \quad z(d_{(i)}) = \frac{f(d_{(i)}) + f(d_{(i)} + 1)}{2}.$$

A high score is reached by a large jumping height combined with a small approximation error. For each optimal step function, the position with the highest score

$$v = \underset{d_i \in \mathcal{D}}{\operatorname{argmax}} \frac{h(d_{(i)})}{e(d_{(i)})}$$

is determined. This yields a vector $\mathbf{v}$ of possible binarization threshold locations for all step functions.

Estimating location and variation of the strongest discontinuities

Each strongest step v_i in $\mathbf{v}$ an optimal step function is a possible location for the binarization threshold t. The final binarization threshold is the median value of $\mathbf{v}$. BASC uses a bootstrap test to evaluate the robustness of this threshold. The null hypothesis of this test is that the expected normalized deviation of strongest discontinuities in step functions from the median is greater than or equal to a prespecified value $\tau \in (0, 1]$ that balances the α error versus the β error of the test. A significant p-value resulting from this test indicates that the number of different strongest step positions is low and that their values are within a small range, i.e. the computed binarization is of good quality. Details on the testing procedure can be found in [2].

2.3 Trinarization Across multiple SCales (TASC)

`Binarize` includes extensions of the BASC algorithms that allow for the trinarization of a signal $f : \mathbb{R} \to \{0, 1, 2\}$. This type of data transformation discretizes a real-valued number u into one of three levels according to two adapted thresholds t_1 and t_2,

$$f(u) = \begin{cases} 0 & u \leq t_1 \\ 1 & t_1 < u \leq t_2 \\ 2 & t_2 < u \end{cases}.$$

The corresponding extensions are called TASC A and TASC B respectively. The search for an optimal pair of thresholds leads to modifications in the analysis of the optimal step functions. The TASC algorithms have to identify the strongest pair of discontinuities in each optimal step function. Afterwards the location and variation of the strongest pair of discontinuities have to be determined.

Finding the strongest pair of discontinuities in each step function

BASCs evaluation criteria for a single optimal discontinuity must be replaced by a score for a pair of discontinuities at break points i, k with $i < k$,

$$v = \underset{d_i, d_k \in \mathcal{D}}{\operatorname{argmax}} \frac{h(d_{(i)}) + h(d_{(k)})}{2e^{\text{tri}}(d_{(i)}, d_{(k)})},\ i < k$$

In this context, the error function has to be replaced by

$$\begin{aligned} e^{\text{tri}}(d_{(i)}, d_{(k)}) = &\sum_{n=1}^{d_{(i)}} \big(f(n) - z(d_{(i)})\big)^2 + \sum_{n=d_{(k)}}^{N} \big(f(n) - z(d_{(k)})\big)^2 \\ &+ \frac{1}{2} \sum_{n=d_{(i)}}^{d_{(k)}} \big(f(n) - z(d_{(i)})\big)^2 + \frac{1}{2} \sum_{n=d_{(i)}}^{d_{(k)}} \big(f(n) - z(d_{(k)})\big)^2 \end{aligned}$$

Estimating location and variation of the strongest discontinuities

For testing the robustness of a TASC result we use a bootstrap test with a test statistic similar to the one used for BASC. The difference is that here the normalized deviation of both thresholds is used for the calculation.

3 Application

In the following, the usage of the binarization methods and plot functions is illustrated using code examples. To execute these examples, the `Binarize` package must be properly installed in the R environment. This can be done by typing

```
> install.packages("Binarize")
```

in an R console or by the corresponding menu entries in the R GUI. In a second step, the `Binarize` package is loaded via

```
> library("Binarize")
```

The package must be installed only once, but it has to be loaded at the beginning of every R session. Only then are the methods and classes of the `Binarize` package available to the R workspace and ready to use.

The binarization methods require at least an input vector of measurements $\mathbf{u} = (u_1, \ldots, u_N) \in \mathbb{R}^N$ with $N \geq 3$. Additional arguments depend on the particular method used and will be discussed in greater detail in the appropriate subsection below. The method output of each function call is encapsulated into an S4 object whose class is either of type `BinarizationResult` or of a derived type with additional attributes. The class `BinarizationResult` contains the following attributes:

- **originalMeasurements:** The original input vector **u**
- **binarizedMeasurements:** The binarized vector **b**

- **method:** The name of the method used for the binarization
- **threshold:** The threshold t used for the binarization
- **p.value:** A parameter that rates how well the method was able to binarize the original data

In the following subsections, an artifical dataset is used to illustrate the application of each algorithm. This example dataset is a matrix with ten rows that each represent a feature vector to be binarized. Each feature vector consists of five random measurements sampled from one normal distribution $\mathcal{N}(0, 1)$ and five measurements sampled from a second normal distribution. The mean of this second normal distribution varies between the different feature vectors in the dataset. It starts at 10 in the first feature vector (row of the dataset) and is decremented by one in each consecutive feature vector. In the last vector, the mean of the second normal distribution is therefore 1. To load the matrix and make the variable `binarizationExample` available in the R workspace, type

```
> data(binarizationExample)
```

One would expect the binarization threshold for each feature vector in this dataset to be in-between the means of the two normal distributions from which the data points were sampled (see Figure 1).

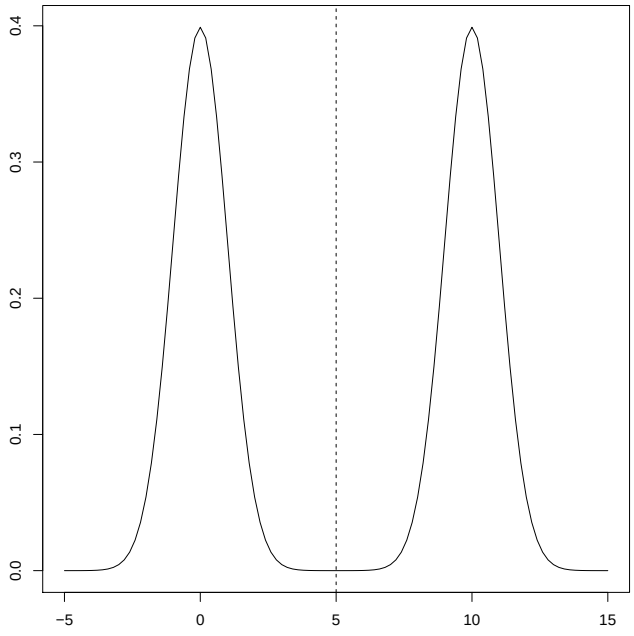

Figure 1: The density function of the first feature vector of the `binarizationExample` dataset. The dashed line marks the mean of this row. The density functions of the other feature vectors in `binarizationExample` only differ in the distance between the two peaks of the normal distributions sampled from.

3.1 k-means

To binarize the first feature vector of the `binarizationExample` dataset using the k-means algorithm, type:

```
> bin <- binarize.kMeans(binarizationExample[1,])
> print(bin)

Method: k-Means

Threshold: 4.98118

Binarized vector: [ 0 0 0 0 0 1 1 1 1 1 ]

p value: 0.0002593634
```

In addition to the required input vector, other arguments may be passed. The most important one is `dip.test` which defaults to `TRUE`. If set to `TRUE`, Hartigan's dip test for unimodality [1] is performed on the input vector. If the test is significant, the assumption that the distribution of the data is unimodal is rejected. Thus, the corresponding p-value indicates how well the data is binarizeable.

The function call results in an object of type `BinarizationResult` which contains the attributes listed above. These can be accessed using the `@` operator. E.g., to access the vector of binarized measurements, type:

```
> print(bin@binarizedMeasurements)

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

A binarization can be visualized using the generic `plot()` method of the package. This method visualizes the computed binarization result using a one- or two-dimensional plot and can be applied to all binarization methods available in the package. The only required argument is an object of type `BinarizationResult` or `BASCResult`. To visualize the results of the binarization in a one-dimensional plot, type:

```
> plot(bin)
```

The resulting plot is shown in the left panel of Figure 2. By setting the argument `twoDimensional` to `TRUE`, a two-dimensional plot can be created:

```
> plot(bin, twoDimensional=TRUE)
```

In this two-dimensional plot, the values of the input vector are plotted on the y-axis and their positions within the original input vector are aligned on the x-axis (see the right panel of Figure 2).

`plot()` also accepts the arguments of the standard plotting methods like `col` and `pch`. This can be used to visualize two different groupings, such as a binarization and a class assignment, in the same plot. In this way, it is possible to see how much these two groupings coincide.

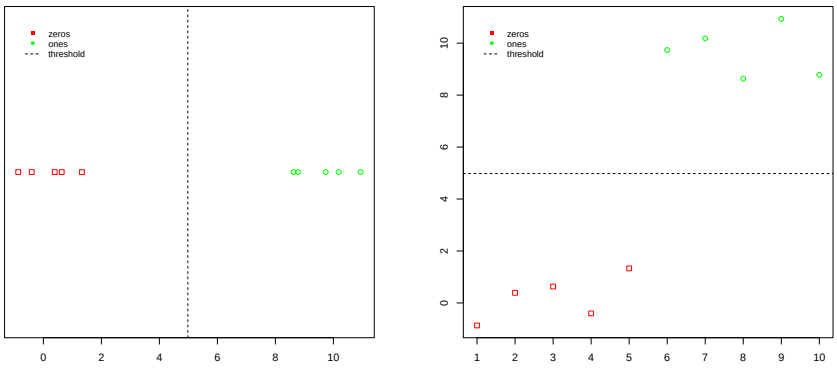

Figure 2: Two plots that were created with `plot()` and illustrate the result of the binarization of `binarizationExample[1,]` using the k-means algorithm. The left picture shows a plot with the parameter `twoDimensional` set to `FALSE`, whereas `twoDimensional=TRUE` in the right plot.

For the next example, we assume that the class labels correspond to the distributions from which the data points in the `binarizationExample` dataset were sampled. Instead of the first feature vector, we take the last feature vector here, as the two classes cannot be clearly separated here:

```
> label <- c(rep(0,5), rep(1,5))
> bin <- binarize.kMeans(binarizationExample[10,])
```

To display the class label simultaneously with a binarization result, we assign different colors and shapes to the corresponding data points:

```
> plot(bin, twoDimensional=TRUE,
+      col=label+1, pch=label,
+      showLegend=FALSE)
```

The plot is shown in Figure 3.

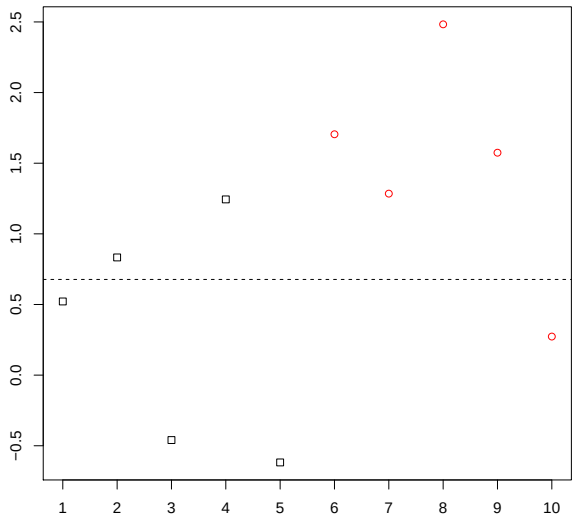

Figure 3: A plot created with the `plot()` function where the threshold illustrates the binarization of the data and the color and shape of the data points correspond to the class label, i.e. the distribution the points were originally sampled from.

So far, we have only binarized a single feature vector. To binarize all the feature vectors of a matrix at once, the `binarizeMatrix` method can be used. This method must be supplied with a matrix of measurements (one vector per row) as well as the binarization algorithm that should be used. To binarize `binarizationExample` using the k-Means algorithm, type:

```
> binMatrix <- binarizeMatrix(binarizationExample,
+                             method="kMeans")
```

The resulting matrix contains the binarized measurements, the binarization thresholds and the p-values of the different features' binarization in the respective rows. As an additional argument, a method to correct the p-values for multiple testing may be passed via `adjustment`. All values that are accepted by `p.adjust` may be passed. Per default, no adjustment is applied. To binarize `binarizationExample` and adjust the resulting p-values according to the false discovery rate (FDR), type:

```
> binMatrixFDR <- binarizeMatrix(binarizationExample,
+                                method="kMeans",
+                                adjustment="fdr")
```

3.2 BASC

The second binarization function in the `Binarize` package provides implementations of the two BASC variants. The following examples illustrate the use of the BASC algorithms. To binarize the first feature of `binarizationExample` using BASC A, type:

```
> bin <- binarize.BASC(binarizationExample[1,], method="A")
> print(bin)

Method: BASC A

Threshold: 4.98118

Binarized vector: [ 0 0 0 0 0 1 1 1 1 1 ]

p value: 0.001
```

The required argument for `binarize.BASC()` is the input vector. The most important arguments that may be passed in addition are `method`, `tau` and `sigma`. `method` specifies the BASC algorithm to be used with `"A"` for BASC A and `"B"` for BASC B. If no method is specified, BASC A is used by default. An optional parameter `tau` adjusts the sensitivity and specificity of the statistical test that rates the quality of binarization. For BASC B, the parameter `sigma` specifies a vector of different σ values for the convolutions of the Bessel functions.

The function call returns an object of the type `BASCResult`. In addition to the attributes contained by objects of the type `BinarizationResult`, this type also contains the following attributes:

- **intermediateSteps:** A matrix specifying the optimal step functions from which the binarization was calculated. The number of rows corresponds to the number of step functions and the number of columns is determined by the length of the input vector minus 2. From the first to the last row the number of steps per function increases. Non-zero entries represent the step locations, while rows with fewer steps than the input step function have entries in the matrix set to zero.
- **intermediateHeights:** A matrix specifying the jump heights of the steps supplied in `intermediateSteps`
- **intermediateStrongestSteps:** A vector with one entry for each step function, i.e. row in `intermediateSteps`. The entries specifiy the location of the strongest step of each function.

E.g., one can view the locations of the strongest discontinuities in the reduced step functions by typing

```
> print(bin@intermediateStrongestSteps)

[1] 5 5 5 5 5 5 5 5
```

It can be seen that the potential threshold locations of all step functions are the same for this feature. As there is no variation, the statistical test is also highly significant (see above).

Apart from the generic `plot()` function demonstrated for the k-means binarization, a specialized plot for the BASC methods is supplied: The `plotStepFunctions()` method plots all the optimal step functions calculated by the BASC algorithms. The only required argument is an object of type `BASCResult`. To plot the step functions of such an object, type for example:

```
> plotStepFunctions(bin)
```

The corresponding plot can be seen in Figure 4. Figure 5 shows the reduced number of step functions that BASC B creates for the same data. It can be seen from the figure that the strongest steps (dashed lines) are the same for all step functions, which means that the binarization is very reliable. This is also indicated by the p-value of 0.001 in the result object.

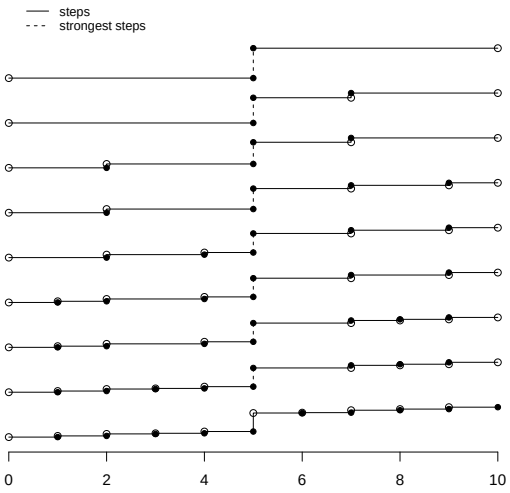

Figure 4: The step functions calculated by BASC A and the original step function of the `binarizationExample[1,]` data with an increasing number of steps from top to bottom.

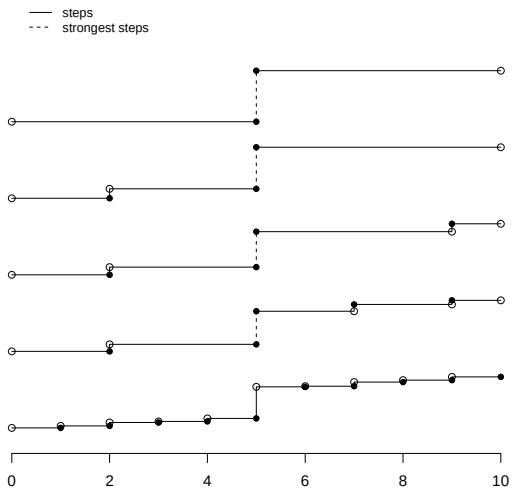

Figure 5: The computed step functions calculated by BASC B and the original step function of the `binarizationExample[1,]` data with an increasing number of steps from top to bottom.

3.3 TASC

The application of the TASC function is similar to the application of BASC. The method provides an implementation of TASC A and B, with calculation of step functions equal to BASC A and B respectively.

The package includes an artificial dataset for illustrating the application of the trinarization algorithms. The dataset is (as the binarization dataset) a matrix. It consists of one hundred rows that each represent a feature vector to be trinarized. Each row consists of five random measurements sampled from a normal distribution $\mathcal{N}(0, 1)$, five measurements sampled from a second normal distribution and 5 additional samples from a third normal distribution. The means of the second and third normal distribution vary between the different feature vectors in the dataset. The second starts at 10 in the first feature vector (row of the dataset) and is decremented by one in each consecutive feature vector until it is 1. The mean of the third distribution is decremented in steps of 2 with a maximum of 20 and a minimum of 2. This is repeated for each value of the second distribution. In the last vector, the mean of the second normal distribution is 1 and the mean of the third one is 2. To load the matrix and make the variable `trinarizationExample` available in the R workspace, type

```
> data(trinarizationExample)
```

The trinarization of the first row can be done by typing:

```
> tri <- TASC(trinarizationExample[1,], method="A")
> print(tri)

Method: TASC A

Threshold1: 4.445755

Threshold2: 14.93978

Trinarized vector: [ 0 0 0 0 0 1 1 1 1 1 2 2 2 2 2 ]

p value: 0.001
```

The arguments of the TASC function are equal to `binarize.BASC`. They are `method` to switch between TASC A and TASC B, `tau` for the sensitivity of the statistical test and `sigma` for the Bessel functions if TASC B is applied.
The returned result is of type `TASCResult` which is a subclass of `TrinarizationResult`. A `TrinarizationResult` consists of the following elements:

- **originalMeasurements:** The original input vector **u**
- **trinarizedMeasurements:** The trinarized vector **b**
- **method:** The name of the method used for the trinarization
- **threshold1:** The threshold t_1 used for the trinarization
- **threshold2:** The threshold t_2 used for the trinarization
- **p.value:** A parameter that rates how well the method was able to trinarize the original data

The `TASCResult` additionally contains:

- **intermediateSteps:** A matrix specifying the optimal step functions from which the binarization was calculated. The number of rows corresponds to the number of step functions and the number of columns is determined by the length of the input vector minus 2. From the first to the last row the number of steps per function increases. Non-zero entries represent the step locations, while rows with fewer steps than the input step function have entries in the matrix set to zero.
- **intermediateHeights1:** A matrix specifying the jump heights of the steps supplied in `intermediateSteps` for the first threshold
- **intermediateHeights2:** A matrix specifying the jump heights of the steps supplied in `intermediateSteps` for the second threshold
- **intermediateStrongestSteps:** A matrix with two columns and one row for each step function, i.e. row in `intermediateSteps`. The entries specify the location of the pair of strongest steps of each function.

The location of the strongest steps can be viewed by typing

```
> print(tri@intermediateStrongestSteps)
```

```
      [,1] [,2]
 [1,]    5   10
 [2,]    5   10
 [3,]    5   10
 [4,]    5   10
 [5,]    5   10
 [6,]    5   10
 [7,]    5   10
 [8,]    5   10
 [9,]    5   10
[10,]    5   10
[11,]    5   10
[12,]    5   10
```

It can be seen that the potential threshold locations of all step functions are the same as in the binarization example. As there is no variation, the statistical test is highly significant (see above).

Analogously to the visualizations of binarization results there are functions for trinarization results included in the package:

```
> plotStepFunctions(tri)
> plot(tri, twoDimensional = TRUE)
```

The corresponding plots can be seen in Figure 6. It shows the step functions that TASC A creates for the data. On the right the original data, trinarized into three groups is shown in two-dimensional space with the determined thresholds drawn as dashed lines.

4 Quality assessment using statistical tests

As already mentioned, all binarization and trinarization routines in the package provide a statistical testing procedure that evaluates the reliability of the results. While the dip test employed with k-means is based on the distribution of the original data, the bootstrap test of BASC/TASC assesses the variability of potential thresholds.

The statistical tests can be utilized to select a subset of reliable feature vectors from a set of features and to reject those features which do not show a clear separability into two groups.

The following example binarizes all 10 feature vectors of the `binarizationExample` dataset using k-means and determines the number of features with significant p-values:

```
> binMatrix <- binarizeMatrix(binarizationExample,
+                             method="kMeans",
+                             adjustment="fdr")
```

```
> significantRows <- sum(binMatrix[,12] < 0.05)
> print(significantRows)

[1] 4
```

As we performed 10 tests here, we have to correct the p-values for multiple testing. This is done according to the false discovery rate using the `adjustment` argument.

For the BASC algorithms, the code is analogous and yields the following numbers of significant features:

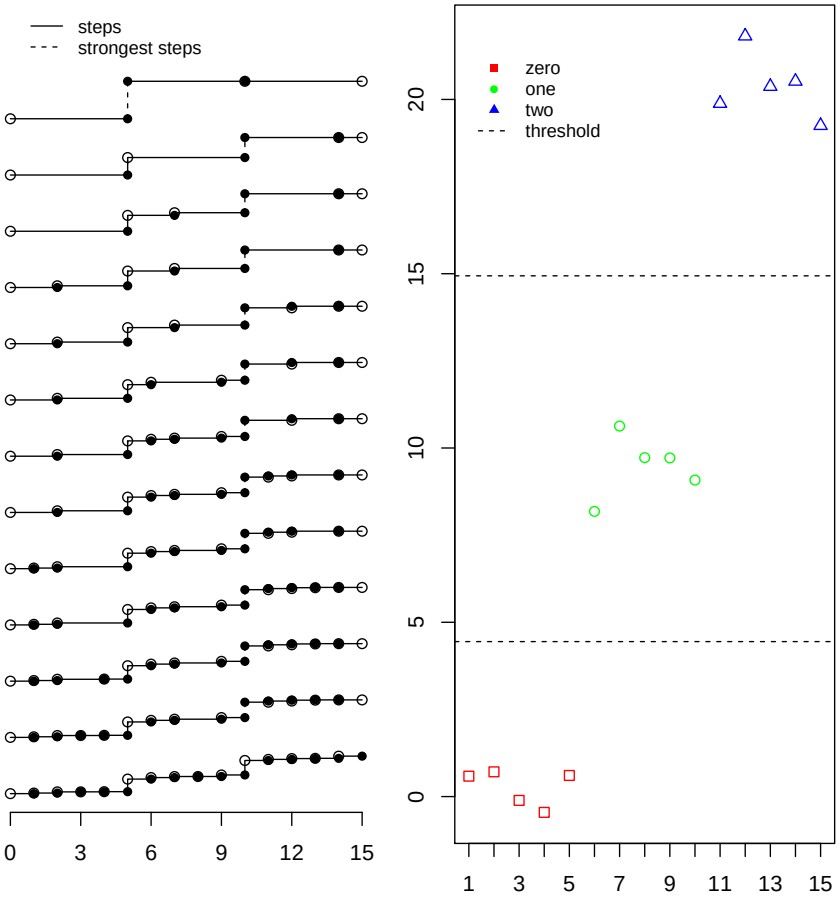

Figure 6: Left: The step functions calculated by TASC A and the original step function of the `trinarizationExample[1,]` data with an increasing number of steps from top to bottom. Right: The trinarized data in a two dimensional plot. The grouping is shown by the different colors. The thresholds are drawn as dashed lines.

BASC A:

```
> print(significantRows)

[1] 8
```

BASC B:

```
> print(significantRows)

[1] 6
```

It can be observed that the dip test employed with k-means is more restrictive than the bootstrap test in the BASC variants. The sensitivity and specificity of the bootstrap test can be adjusted by changing the `tau` parameter.
The following example demonstrates this:

```
> tauValues <- seq(0,0.25, 0.05)
> print(tauValues)

[1] 0.00 0.05 0.10 0.15 0.20 0.25
```

The vector above stores different `tau` values. For each of these values, `binarizationExample` is now binarized using BASC B. The resulting number of features in `binarizationExample` with a p-value < 0.05 is stored in `significantFeatures` and printed out.

```
> significantFeatures <- sapply(tauValues, function(tau)
+ {
+   binMatrix <- binarizeMatrix(binarizationExample,
+                               method="BASCB",
+                               adjustment="fdr",
+                               tau=tau)
+
+   significantRows <- sum(binMatrix[,12] < 0.05)
+   return(significantRows)})
> names(significantFeatures) <- tauValues
> print(significantFeatures)

   0 0.05  0.1 0.15  0.2 0.25
   0    6    7    8    9   10
```

It can be seen that different `tau` values result in a different number of significant features. The binarization thresholds themselves are independent of `tau` and remain unchanged when `tau` is changed.

References

[1] J. A. Hartigan and P. M. Hartigan. The dip test of unimodality. *The Annals of Statistics*, 13(1):1–433, 1985.

[2] M. Hopfensitz, C. Müssel, C. Wawra, M. Maucher, M. Kühl, H. Neumann, and H. A. Kestler. Multiscale binarization of gene expression data for reconstructing Boolean networks. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 9(2):487–498, 2012.

[3] S. A. Kauffman. Metabolic stability and epigensis in randomly constructed genetic nets. *Journal of Theoretical Biology*, 22(3):437–467, 1969.

[4] S. A. Kauffman. *The origins of order: Self-organization and selection in evolution*. Oxford University Press, 1993.

[5] H. Lähdesmäki, I. Shmulevich, and O. Yli-Harja. On learning gene regulatory networks under the Boolean network model. *Machine Learning*, 52(1-2):147–167, 2003.

[6] S. Liang, S. Fuhrman, R. Somogyi, et al. REVEAL, a general reverse engineering algorithm for inference of genetic network architectures. In *Pacific symposium on biocomputing*, volume 3, page 2, 1998.

[7] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In J. Neyman and L. L. Cam, editors, *5th Berkeley Symposium on Math, Statistics and Probability*, volume 1, pages 281–297, Berkeley, 1967. University of California Press.

[8] M. Maucher, B. Kracher, M. Kühl, and H. A. Kestler. Inferring Boolean network structure via correlation. *Bioinformatics*, 27(11):1529–1536, 2011.

[9] C. Müssel, M. Hopfensitz, and H. A. Kestler. BoolNet – an R package for generation, reconstruction and analysis of Boolean networks. *Bioinformatics*, 26(10):1378–1380, 2010.