```
## Warning:  'latex2' is deprecated.
## Use 'latex' instead.
## See help("Deprecated")
```

Clustering time series gene expression data with TMixClust

Monica Golumbeanu [1] ***and Niko Beerenwinkel*** [1]

[1]Department of Biosystems Science and Engineering, ETH Zuerich, Switzerland and Swiss Institute of Bioinformatics, Basel, Switzerland

October 30, 2017

Abstract

A large number of longitudinal studies measuring gene expression aim to stratify the genes according to their differential temporal behaviors. Genes with similar expression patterns may reflect functional responses of biological relevance. However, these measurements come with intrinsic noise which makes their time series clustering a difficult task. Here, we show how to cluster such data with the package TMixClust. TMixClust is a soft-clustering method which employs mixed-effects models with nonparametric smoothing spline fitting and is able to robustly stratify genes by their complex time series patterns. The package has, besides the main clustering method, a set of functionalities assisting the user to visualise and assess the clustering results, and to choose the optimal clustering solution. In this manual, we present the general workflow of using TMixClust for analysing time series gene expression data and explain the theoretical background of the clustering method used in the package implementation.

Contents

1 Standard workflow applied to simulated and real data 1
- 1.1 Loading data 2
- 1.2 Clustering 4
- 1.3 Stability analysis, choosing optimal clustering solution 5
- 1.4 Assessing and validating clustering consistency with silhouette analysis 7
- 1.5 Generating a results report 9
- 1.6 Application of TMixClust on publicly available time series gene expression data from yeast 9

2 Methodology behind TMixClust 12
- 2.1 Mixed effects model with embedded smoothing splines 13
- 2.2 Estimating model parameters 14
- 2.3 Description of simulated data 14

3 How to get help 15

4 Session info 15

1.1 Loading data

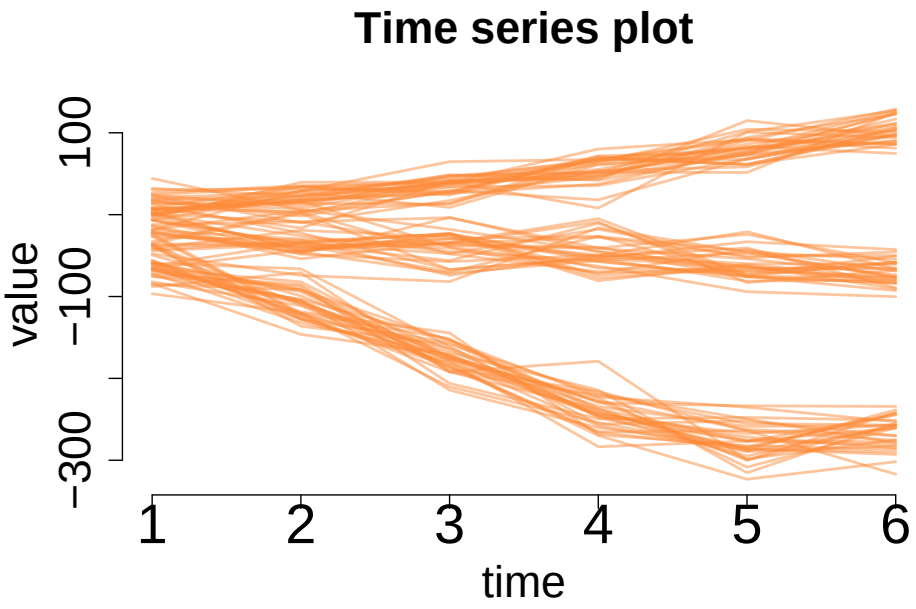
After loading the package TMixClust, we first need to define a data frame which contains the time series gene expression data. By *time series*, we denote a vector of expression values of a gene measured at different, successive time points. $X = \{100, 200, 300, 400\}$ is an example of a time series constituted by measurements at 4 time points. A time series gene expression data set contains an ensemble of time series vectors, each time series being associated to a gene. The input data frame has a number of rows equal to the number of time series vectors and a number of columns equal to the number of time points. Row names correspond to the time series names (i.e., gene names), while column names represent time points names (e.g., "2h", "4h", etc.). The data frame can contain missing values.

Package TMixClust contains a simulated data set, `toy_data_df`, with 91 time series. For a detailed description of how the data set was constructed, see Section 2. We can load the simulated time series data into a data frame, see its first rows and plot its contents as follows:

```
# load the package
library(TMixClust)

# load the simulated time series data set
data(toy_data_df)

# display the first rows of the data frame
print(head(toy_data_df))
```

```
##              time1     time2    time3    time4     time5     time6
## gene_1  14.5515322  7.156850 41.43372 64.17897  80.89528 127.10444
## gene_2   7.0624795 15.192533 33.80188 55.40540  86.15328 110.73804
## gene_3 -18.4186975  4.806256 13.79942 69.30776  61.74546  98.10358
## gene_4  13.8395249 14.386315 42.02976 42.93031 114.87758  95.95243
## gene_5   1.5623627 29.595991 33.93879 35.56273  68.20763  96.32600
## gene_6  -0.8357523 34.048379 38.37403 46.25532  90.50204 128.95565
```

```
# plot the time series of the data set
plot_time_series_df(toy_data_df)
```



The user can also load the data from a tab-delimited text file, using the function `get_time_series_df` in package TMixClust. For example, the previously-used time series data has been stored in the `toy_time_series.txt`, also available with the TMixClust package. We can retrieve the contents of the file and store it in a data frame as follows:

```
# retrieve the time series data frame from a text file
toy_data_file = system.file("extdata", "toy_time_series.txt",
                            package = "TMixClust")
toy_data = get_time_series_df(toy_data_file)

# display the first rows of the data frame
print(head(toy_data))
```

```
##              time1     time2    time3    time4     time5     time6
## gene_1  14.5515322  7.156850 41.43372 64.17897  80.89528 127.10444
## gene_2   7.0624795 15.192533 33.80188 55.40540  86.15328 110.73804
## gene_3 -18.4186975  4.806256 13.79942 69.30776  61.74546  98.10358
## gene_4  13.8395249 14.386315 42.02976 42.93031 114.87758  95.95243
## gene_5   1.5623627 29.595991 33.93879 35.56273  68.20763  96.32600
## gene_6  -0.8357523 34.048379 38.37403 46.25532  90.50204 128.95565
```

Finally, it is possible to load the time series data directly from a Biobase ExpressionSet object with function `get_time_series_df_bio`. Here is an example using the SOS data from the Bioconductor SPEM package:

```
# Load the SOS pathway data from Bioconductor package SPEM
library(SPEM)
```

```
## Loading required package:  Rsolnp
## Loading required package:  Biobase
## Loading required package:  BiocGenerics
## Loading required package:  parallel
```

```
##
## Attaching package:  'BiocGenerics'

## The following objects are masked from 'package:parallel':
##
##    clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##    clusterExport, clusterMap, parApply, parCapply, parLapply,
##    parLapplyLB, parRapply, parSapply, parSapplyLB

## The following objects are masked from 'package:stats':
##
##    IQR, mad, sd, var, xtabs

## The following objects are masked from 'package:base':
##
##    Filter, Find, Map, Position, Reduce, anyDuplicated, append,
##    as.data.frame, cbind, colMeans, colSums, colnames, do.call,
##    duplicated, eval, evalq, get, grep, grepl, intersect,
##    is.unsorted, lapply, lengths, mapply, match, mget, order, paste,
##    pmax, pmax.int, pmin, pmin.int, rank, rbind, rowMeans, rowSums,
##    rownames, sapply, setdiff, sort, table, tapply, union, unique,
##    unsplit, which, which.max, which.min

## Welcome to Bioconductor
##
##    Vignettes contain introductory material; view with
##    'browseVignettes()'.  To cite Bioconductor, see
##    'citation("Biobase")', and for packages 'citation("pkgname")'.

data(sos)
sos_data = get_time_series_df_bio(sos)

# Print the first lines of the retrieved time series data frame
print(head(sos_data))
```

1.2 Clustering

Once the time series data is loaded, we can perform clustering using the function `TMixClust`. The only required argument of this function is the data frame containing the time series data. By default, the function clusters the time series in 2 groups. For a different number of desired groups, the user must specify the value of argument `nb_clusters`. Other optional arguments of the `TMixClust` function are described in the package reference manual.

The function `TMixClust` creates a TMixClust object which is a list containing a set of attributes such as clustering membership, estimated model parameters, posterior probabilities of the time series, mixing coefficients or model likelihood (cf. package reference manual).

To cluster once the simulated time series data in 3 groups, we can proceed as follows:

```
# cluster the time series in 3 groups
cluster_obj = TMixClust(toy_data_df, nb_clusters = 3)

## Initializing ...
```

```
## Performing EM, this operation might take a few minutes ...
## log likelihood is increasing
## Clustering done, results stored in a TMixClust object.
```

Note that, depending on the input data, the clustering result may be different than the optimal solution. This behavior can be observed if the clustering operation is repeated several times. Technically, this arises due to local optima in the inference procedure of `TMixClust` and can only be avoided by repeating clustering several times and investigating the likelihood of the data in each case, as described in the next section.

1.3 Stability analysis, choosing optimal clustering solution

TMixClust is based on a statistical model where inference is made through the Expectation-Maximization (EM) technique (see Section 2 for details). When running the clustering algorithm, the EM procedure might get stuck in a local optimum. The local optimum solution has a lower likelihood and is suboptimal. It is therefore highly recommended to perform a stability analysis of the clustering in order to see how often the algorithm gets stuck in local optima and how different are these local optima from the best clustering solution. Finally, we highly recommend to run several times the `TMixClust` function, in order to ensure that the global optimum solution is reached.

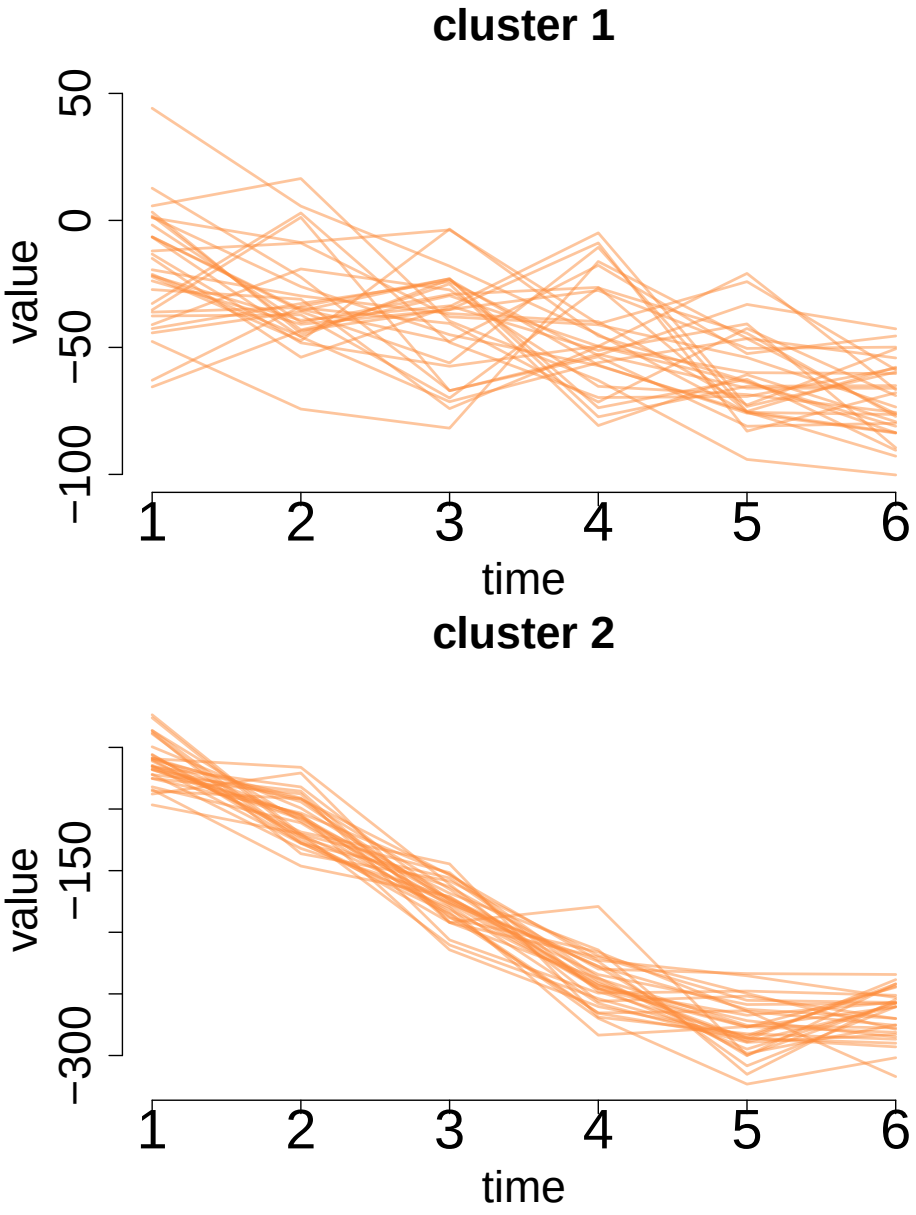
Package TMixClust provides the function `analyse_stability` which runs several times the clustering algorithm, selects the clustering solution with the highest likelihood (assumed to be the global optimum) and returns it. The function also computes and plots the distribution of the Rand index [1] between each clustering solution and the global optimum solution. The Rand index quantifies the agreement probability between two clustering runs, also showing clustering stability.

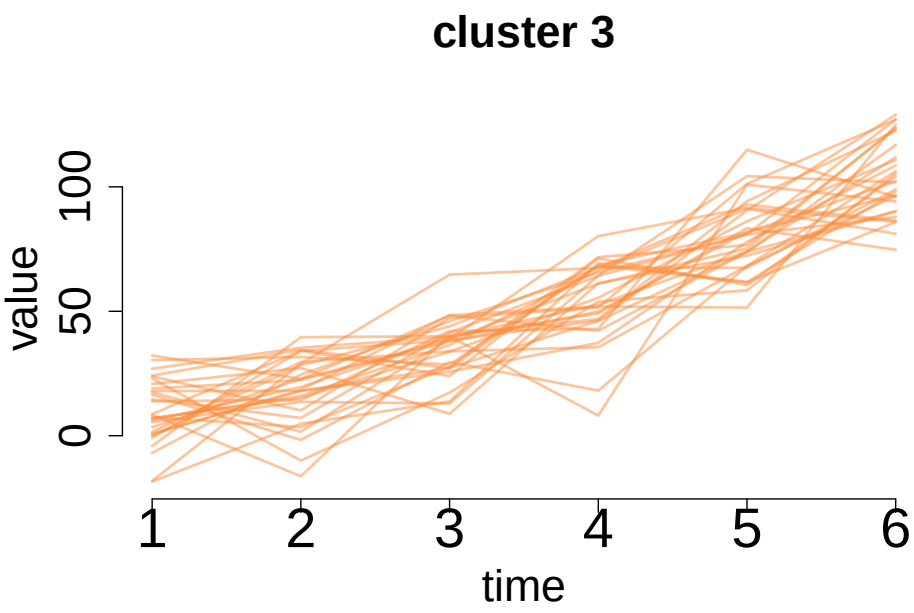
The user can define the number of clustering runs (i.e, the number of times TMixClust algorithm is run on the same data, initial conditions and clustering configuration) and has the possibility to parallelize the runs by defining a number of computing cores. By default, the function uses 2 cores.

For example, we can repeat clustering on the previously presented simulated data for 10 times, for a number K=3 of clusters and using 3 cores, then plot the best clustering solution. For execution time reasons, we have stored the result of this analysis (commented `analyse_stability` command in the code below) in a pre-computed object available with the package TMixClust. We can interrogate this object as follows:

```
# command used for running clustering 10 times with K=3
# and obtaining the result stored in best_clust_toy_obj
# best_clust_toy_obj = analyse_stability(toy_data_df, nb_clusters = 3,
#                                        nb_clustering_runs = 10,
#                                        nb_cores = 3)

# load the optimal clustering result following stability analysis
data("best_clust_toy_obj")

# display the likelihood of the best clustering solution
print(paste("Likelihood of the best solution:",
```

```
                best_clust_toy_obj$em_ll[length(best_clust_toy_obj$em_ll)]))

## [1] "Likelihood of the best solution: -2432.46527298599"

# plot the time series in each cluster
for (i in 1:3) {
    # extract the time series in the current cluster and plot them
    c_df=toy_data_df[which(best_clust_toy_obj$em_cluster_assignment==i),]
    plot_time_series_df(c_df, plot_title = paste("cluster",i))
}
```



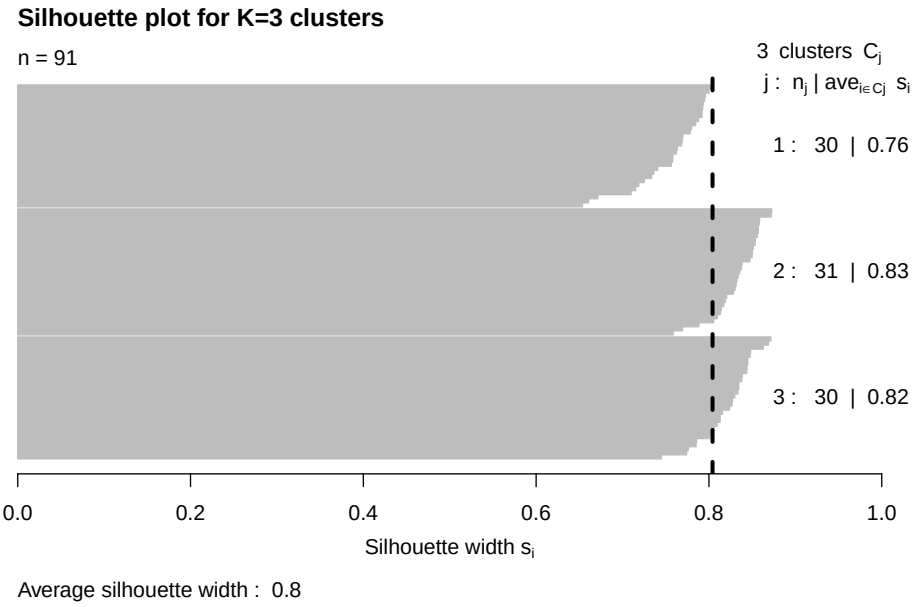


The function `analyse_stability` produces also a histogram of the Rand indexes corresponding to each clustering solution. For our example and straightforward simulated data, we have performed only 10 clustering runs. Depending on the size and complexity of the data, 10 runs might not be enough for attaining the global optimum. We therefore recommend the user to explore with larger numbers of runs, especially if the obtained clustering solutions are not satisfactory.

1.4 Assessing and validating clustering consistency with silhouette analysis

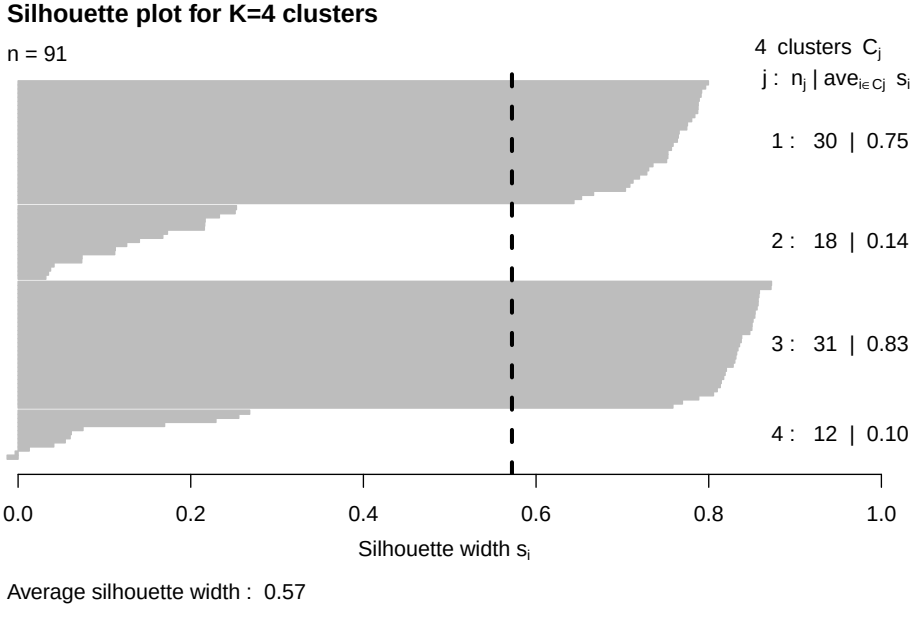
To assist the user in performing a qualitative analysis of different clustering configurations and choosing an adequate number of clusters, package TMixClust provides a tool based on the silhouette technique [2]. The silhouette coefficient, also called silhouette width, is a measure of how similar a data point is to the other points from the same cluster as opposed to the rest of the clusters. Therefore, a high average silhouette width indicates a good clustering cohesion. The most straightforward way to investigate silhouette widths for the data points in a clustering is through visualisation of a silhouette plot. This plot displays the distribution of silhouette coefficients calculated for each data point (in our case each time series) from every cluster. The user can generate a silhouette plot using the `plot_silhouette` function. For our simulated data, we can generate a silhouette plot for the previously obtained global optimum clustering solution for K=3:

```
# silhouette plot of the best clustering solution for K=3
plot_silhouette(best_clust_toy_obj)
```



and compare it to the silhouette plot for a clustering solution with K=4:

```
# cluster the data in 4 groups and generate a silhouette plot
cluster_obj_2 = TMixClust(toy_data_df, nb_clusters = 4)

## Initializing ...

## Performing EM, this operation might take a few minutes ...

## log likelihood is increasing

## Clustering done, results stored in a TMixClust object.

plot_silhouette(cluster_obj_2)
```



Generally, the larger the silhouette widths and the more data points with silhouette width above average, the better a clustering is defined. By comparing the silhouette plots, if we look at the average silhouette width (black dotted line) for K=4, we can clearly see how both the silhouette width and the proportion of data points above average width are less than for K=3, meaning that the clustering with K=4 is starting to overfit the data. The solution with K=3 is better. The user can in this way use the silhouette plot to choose the best number of clusters corresponding to the data.

1.5 Generating a results report

Besides directly accessing the clustering results through a `TMixClust` object, the user has the possibility of generating a more detailed clustering report. The report consists of a comprehensive ensemble of figures and files. Time series plots for each cluster, likelihood convergence plot, lists with time-series from each cluster, and the silhouette plot are part of the generated report.

The report can be generated with the function `generate_TMixClust_report`. The user must supply a `TMixClust` object and optionally a location path for creating the report folder with all the generated files. If a location path is not provided, a folder `TMixClust_report/` will be created in the current working directory.

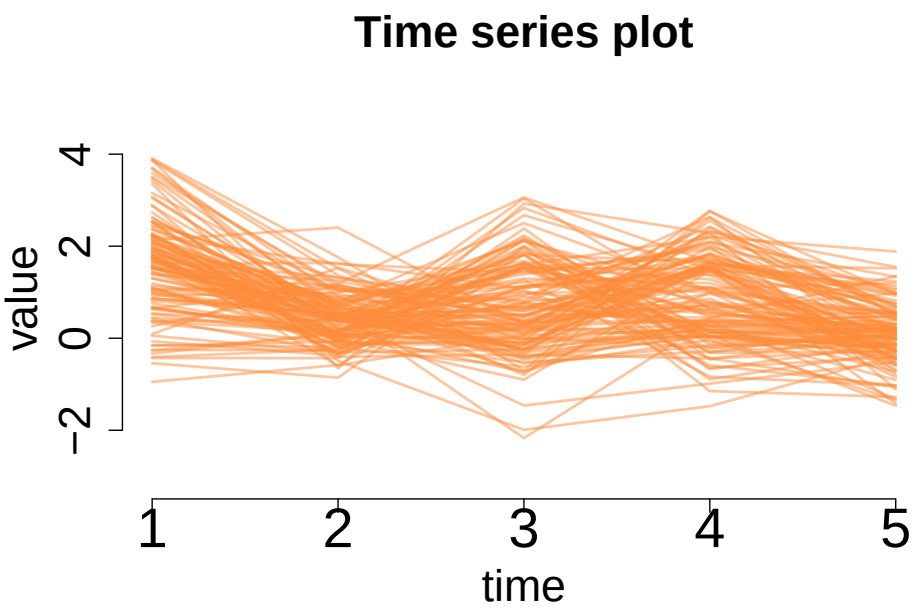
```
# generate a TMixClust report with detailed clustering results
# (not executed here)
# generate_TMixClust_report(cluster_obj)
```

1.6 Application of TMixClust on publicly available time series gene expression data from yeast

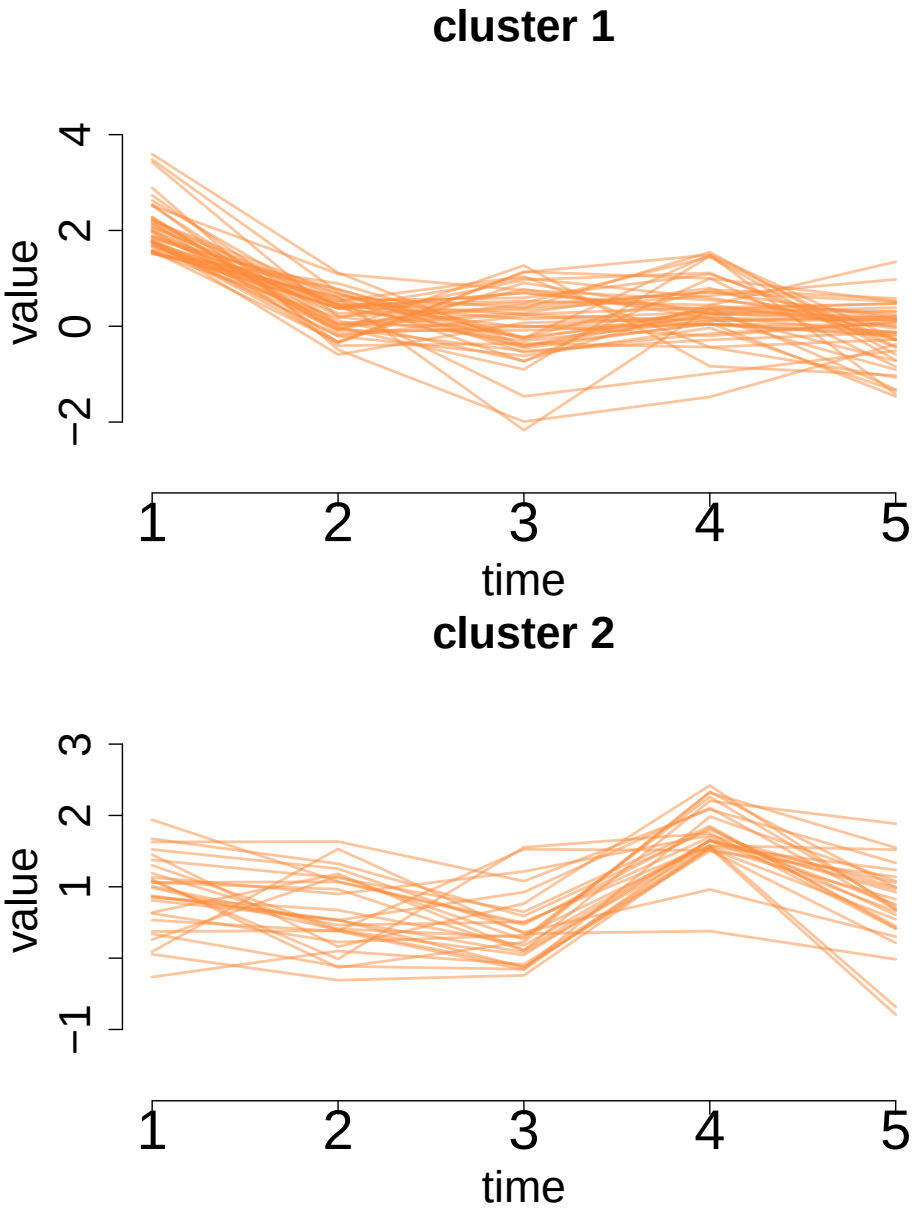
As a final example, we apply TMixClust to a real gene expression time series data set which records transcriptional changes during budding yeast cell-division cycle [3]. For our example, we use a subset of 125 time series measured at five different time points included in the package file `yeast_time_series.txt`.

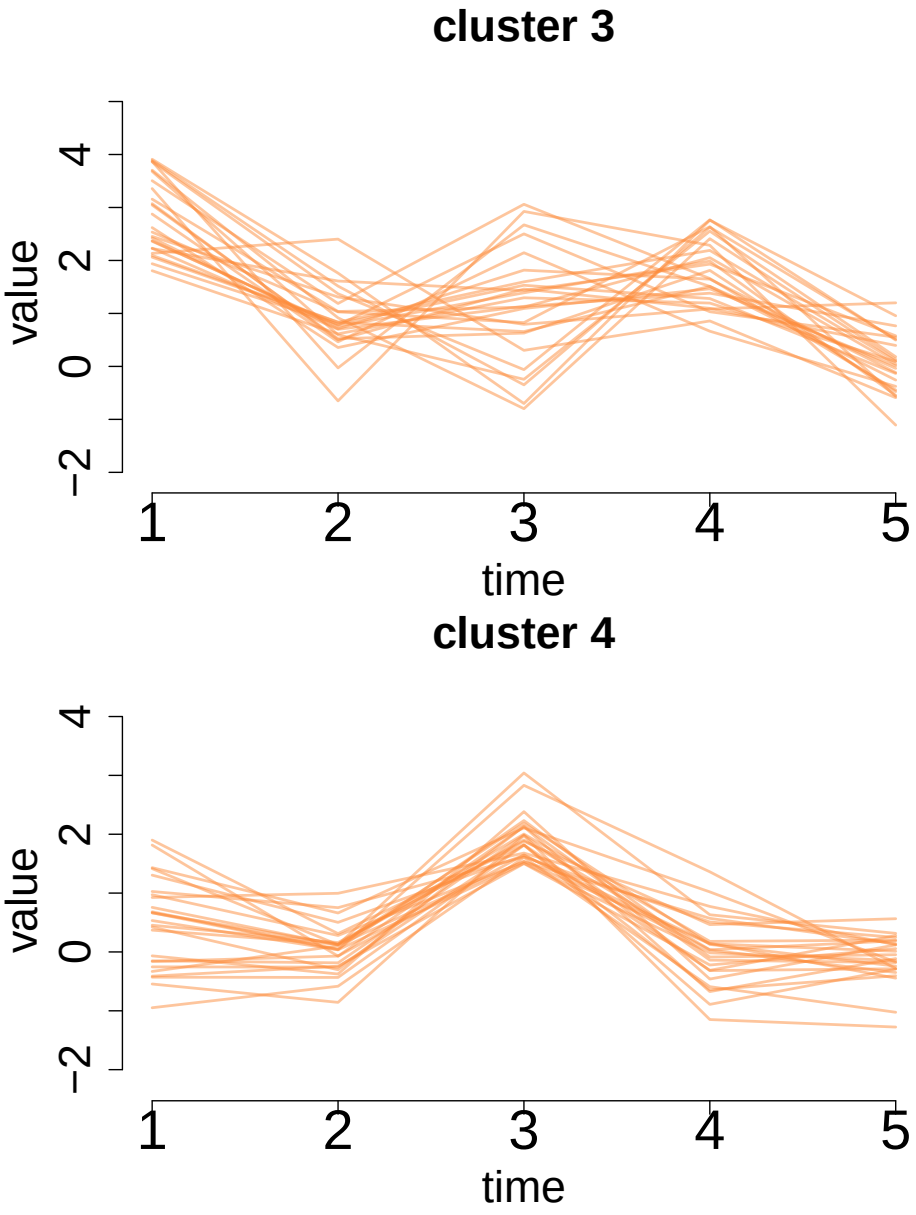
After running TMixClust with different numbers of clusters, investigating the silhouette plots and stability as presented in the previous section, we concluded that the main patterns of gene expression were best described by K=4 clusters. We have stored the TMixClust object containing the optimal clustering solution in the `best_clust_yeast_obj` object, available with the package. We can load the data, plot its time series, load the optimal clustering solution and plot the 4 identified clusters as following:

```
# retrieve the yeast time series data frame from a text file
yeast_data_file = system.file("extdata", "yeast_time_series.txt",
                              package = "TMixClust")
yeast_data = get_time_series_df(yeast_data_file)

# plot the time series of the data set
plot_time_series_df(yeast_data)
```




```
# command used for running clustering 10 times with K=4
# and obtaining the result stored in best_clust_yeast_obj
# best_clust_yeast_obj = analyse_stability(yeast_data,
                                    # time_points = c(0,24,48,63,87),
                                    # nb_clusters = 4,
                                    # nb_clustering_runs = 10,
                                    # nb_cores = 3)

# load the optimal clustering object for the yeast dataset
data("best_clust_yeast_obj")

# plot the identified 4 time series clusters:
for (i in 1:4) {
    # extract the time series in the current cluster and plot them
    c_df=yeast_data[which(best_clust_yeast_obj$em_cluster_assignment==i),]
    plot_time_series_df(c_df, plot_title = paste("cluster",i))
}
```


cluster 1
value
4
2
0
−2
1
2
3
4
5
time
cluster 2
value
3
2
1
−1
1
2
3
4
5
time



2 Methodology behind TMixClust

TMixClust uses the concepts described by [4] for clustering gene expression time series data within the Gaussian mixed-effects models framework with nonparametric smoothing spline estimation [5]. In the following, we provide a short description of these concepts.

2.1 Mixed effects model with embedded smoothing splines

Let $\mathcal{X} = \{\boldsymbol{X_i}\}_{1 \leq i \leq N}$ be a set of N gene expression observations, where each observation $\boldsymbol{X_i}$ is a gene expression time series with T time-points: $\boldsymbol{X_i} = \{x_{i,1}, ..., x_{i,T}\}$. The task is then to cluster the N observations into K groups based on their time series behavior.

Each element of the time series $\boldsymbol{X_i}$ is modeled as a linear combination of a fixed effect $\xi_k(t_j)$ associated to the cluster k, a random effect β_i, and an error term $\epsilon_{i,j}$:

$$x_{i,j} = \xi_k(t_j) + \beta_i + \epsilon_{i,j} \quad (1)$$

where $\beta_i \sim \mathcal{N}(0, \theta_k)$ and $\epsilon_{i,j} \sim \mathcal{N}(0, \theta)$. The fixed effect ξ_k corresponds to the general time series trend or baseline associated to cluster k, the random effect β_i captures any gene-specific systematic shift from the general trend, and $\epsilon_{i,j}$ corresponds to the measurement error. Consequently, $\boldsymbol{X_i}$ follows a multivariate normal distribution $\mathcal{N}(\boldsymbol{\xi_k}, \Sigma_k)$. The covariance matrix Σ_k is defined as follows:

$$\Sigma_k = \theta_k I_T + \theta J_T = \begin{pmatrix} \theta_k + \theta & \theta & ... & \theta \\ \theta & \theta_k + \theta & ... & \theta \\ ... & ... & ... & ... \\ \theta & \theta & ... & \theta_k + \theta \end{pmatrix} \quad (2)$$

where I_T is the unity matrix of dimension T, while J_T is a squared matrix of dimension T with all elements equal to 1.

Our clustering problem is transposed into a mixture model, where each cluster can be described with a Gaussian distribution whose paremeters were defined before, $\mathcal{N}(\boldsymbol{\xi_k}, \Sigma_k)$:

$$\boldsymbol{X_i} \sim \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{\xi_k}, \Sigma_k) \quad (3)$$

and π_k are the mixing coefficients of the mixture model.

Instead of choosing a parametric form for the baseline $\boldsymbol{\xi_k}$, a less restrictive, nonparametric approach using smoothing splines can be used, based on the assumption of existence of smoothness in gene expression. Gu *et* al. have shown that when fitting smoothing splines to a set of Gaussian random variables, the typical residual sum of squares (RSS) minimization problem has an equivalent maximum-likelihood formulation [5]. Precisely, when we try to fit a cubic spline ξ_k to a set of data points, we try to find the ξ_k that minimizes the following score:

$$\sum_{j=1}^{T} (x_{i,j} - \xi_k(j))^2 + \lambda_k \int (\xi_k''(t))^2 dt \quad (4)$$

where the first term quantifies the deviation of the observed values from the curve $\boldsymbol{\xi_k}$, and the second term penalizes the roughness of the curve. If the variables $x_{i,j}$ are normally distributed, then the first term of the score becomes proportional to their minus log likelihood, leading to the following penalized likelihood score [5]:

$$-l(\boldsymbol{X_i}) + \lambda_k \int (\xi_k''(t))^2 dt. \quad (5)$$

Here $l(\boldsymbol{X_i})$ stands for the log likelihood of the data. Therefore, the problem can be formulated as a special case of maximum-likelihood estimation and can be solved using the Expectation-Maximization (EM) method [4].

2.2 Estimating model parameters

As described in [4], the log-likelihood function in the context of the above-defined mixture of Gaussian mixed effects models is:

$$l(\mathcal{X}) = logP(\mathcal{X} \mid \Xi) = log \prod_{i=1}^{N} P(\boldsymbol{X_i} \mid \Xi) \\ = \sum_{i=1}^{N} log \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{X_i} \mid \xi_k, \Sigma_k) \quad (6)$$

where Ξ represents the complete set of model parameters.

The R package `gss` [5], available on CRAN, performs non-parametric smoothing spline fitting for Gaussian random variables. The method finds the cubic smoothing spline that minimizes the penalized likelihood score described in the previous section and estimates the parameters of the associated multivariate Gaussian distribution, namely the mean vector $\boldsymbol{\xi_k}$ and covariance matrix Σ_k.

Within TMixClust, we use package `gss` in the following implemented EM learning scheme:

- 1. initialize clusters (e.g. with a K-means solution for speeding up convergence)
- 2. calculate data likelihood and repeat until convergence:

  E-step:

  - compute posterior probabilities
  - assign genes to clusters based on their posterior probabilities
  - compute mixing coefficients

  M-step:

  - maximize penalised likelihood score with package `gss`
  - update model parameters
- 3. when convergence is reached, return maximum likelihood solution

2.3 Description of simulated data

For this manual, we built a simulated time series data-set, `toy_data_df`, available with package TMixClust. The simulation procedure relies on the assumption that each gene expression time series vector $\boldsymbol{X_i}$ is described by a mixed effects model described by equation 1.

Using equation 1, we generate data from three different clusters whose general patterns $\boldsymbol{\xi_k}$ correspond to the following three polynomials:

- $\boldsymbol{\xi_1(t)} = 3t^2 - 3t + 1$
- $\boldsymbol{\xi_2(t)} = -10t - 5$
- $\boldsymbol{\xi_3(t)} = 4t^3 - 34t^2 + 25t - 47$

and the corresponding gene shifts vectors are $\boldsymbol{\beta_1} \sim N(14, 15)$ and $\boldsymbol{\beta_2} = \boldsymbol{\beta_3} \sim N(-5, 20)$.

3 How to get help

In addition to the package reference manual and current vignette, a detailed description of each package function along with a corresponding example of use can be obtained through one of the following commands:

```
?TMixClust
?generate_TMixClust_report
?get_time_series_df
?plot_time_series_df
?plot_silhouette
?analyse_stability
```

To post any questions related to the TMixClust package, please use the Bioconductor support site https://support.bioconductor.org.

4 Session info

- R version 3.4.2 (2017-09-28), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=en_US.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`
- Running under: `Ubuntu 16.04.3 LTS`
- Matrix products: default
- BLAS: `/home/biocbuild/bbs-3.6-bioc/R/lib/libRblas.so`
- LAPACK: `/home/biocbuild/bbs-3.6-bioc/R/lib/libRlapack.so`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, utils
- Other packages: Biobase 2.38.0, BiocGenerics 0.24.0, Rsolnp 1.16, SPEM 1.18.0, TMixClust 1.0.0
- Loaded via a namespace (and not attached): BiocParallel 1.12.0, BiocStyle 2.6.0, Rcpp 0.12.13, backports 1.1.1, cluster 2.0.6, compiler 3.4.2, digest 0.6.12, evaluate 0.10.1, flexclust 1.3-4, grid 3.4.2, gss 2.1-7, highr 0.6, htmltools 0.3.6, knitr 1.17, lattice 0.20-35, magrittr 1.5, modeltools 0.2-21, mvtnorm 1.0-6, rmarkdown 1.6, rprojroot 1.2, stats4 3.4.2, stringi 1.1.5, stringr 1.2.0, tools 3.4.2, truncnorm 1.0-7, yaml 2.1.14, zoo 1.8-0

5 Citing this package

If you use this package for published research, please cite the package:

```
citation('TMixClust')
```

as well as [6].

References

[1] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, 1985. URL: http://dx.doi.org/10.1007/BF01908075, doi:10.1007/BF01908075.

[2] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53 – 65, 1987. URL: http://www.sciencedirect.com/science/article/pii/0377042787901257, doi:http://dx.doi.org/10.1016/0377-0427(87)90125-7.

[3] Daniel F. Simola, Chantal Francis, Paul D. Sniegowski, and Junhyong Kim. Heterochronic evolution reveals modular timing changes in budding yeast transcriptomes. *Genome Biology*, 11(10):R105, 2010. URL: http://dx.doi.org/10.1186/gb-2010-11-10-r105, doi:10.1186/gb-2010-11-10-r105.

[4] Ping Ma, Cristian I. Castillo-Davis, Wenxuan Zhong, and Jun S. Liu. A data-driven clustering method for time course gene expression data. *Nucleic Acids Res*, 34(4):1261–1269, Mar 2006. 16510852[pmid]. URL: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1388097/, doi:10.1093/nar/gkl013.

[5] Chong Gu. Smoothing spline anova models: R package gss. *Journal of Statistical Software*, 58(1):1–25, 2014. URL: https://www.jstatsoft.org/index.php/jss/article/view/v058i05, doi:10.18637/jss.v058.i05.

[6] Monica Golumbeanu, Sebastien Desfarges, Celine Hernandez, Manfredo Quadroni, Sylvie Rato, Pejman Mohammadi, Amalio Telenti, Niko Beerenwinkel, and Angela Ciuffi. Dynamics of proteo-transcriptomic response to hiv-1 infection. *in preparation*, 2017.