EBBC Package User Manual

v1.0

M. Castelluzzo
A. Perinelli
M. A. Denti
L. Ricci

January 2020

Contents

1. Overview & License

EBBC (Expression-Based Bayesian Classifier) is an R package that provides a set of functions to carry out training and classification analyses on datasets containing multiplets of values (expressions) of a given measured feature. The package consists of a set of R functions, each contained in a dedicated source file located in `/R/`.

DISCLAIMER

The software in this package is for general research purposes only and is thus provided WITHOUT ANY WARRANTY. It is not intended to form the basis of any diagnostic decision. USE AT YOUR OWN RISK!

Reference

The analysis algorithm implemented in this package was originally published in

L. Ricci, V. Del Vescovo, C. Cantaloni, M. Grasso, M. Barbareschi and M. A. Denti, *Statistical analysis of a Bayesian classifier based on the expression of miRNAs*, BMC Bioinformatics **16**:287 (2015).
DOI: 10.1186/s12859-015-0715-9

In this paper, MicroRNA expression were used as measured feature. The paper is henceforth referred to as "**main reference**". Citing it in manuscripts describing works that rely on this package is appreciated.

Licensing

Package authors

Michele Castelluzzo

Department of Physics, University of Trento, 38123 Trento, Italy
michele.castelluzzo@unitn.it

Alessio Perinelli

Department of Physics, University of Trento, 38123 Trento, Italy
alessio.perinelli@unitn.it

Michela Alessandra Denti

Department of Cellular, Computational and Integrative Biology (CIBIO), University of Trento, 38123 Trento, Italy
michela.denti@unitn.it

Leonardo Ricci

Department of Physics, University of Trento, 38123 Trento, Italy
CIMeC, Center for Mind/Brain Sciences, University of Trento, 38068, Rovereto, Italy
leonardo.ricci@unitn.it
Nonlinear Systems & Electronics Lab website: nse.physics.unitn.it

NSE LAB
Nonlinear
Systems &
Electronics

2. Download & setup

The EBBC package requires R version 3.2 or later. The package depends on the packages `stats`, `utils`, `tools`, `pROC`, and `ggplot2` (the latter for plotting purposes).

Download

The EBBC package can be downloaded at https://github.com/LeonardoRicci/EBBC. Setup instructions can be found within the `/setup/` directory.

Manual setup

After downloading the package, setup can be carried out as follows. Launch an R console within `/setup/` and type

```
install.packages("EBBC_1.0.tar.gz", type="source")
```

Alternatively, the whole path of the archive file can be specified. Please note that permissions to write on the target library directory are required.
Another way to install the package is to open a terminal in the folder where the archive file is located and type

```
R CMD INSTALL EBBC_1.0.tar.gz
```

If the package cannot be installed from the `.tar.gz` archive, the latter can be rebuilt from source. This procedure requires the DEVTOOLS package to be installed in order to produce the function documentation. Open R from within the package root directory and type

```
library(devtools)
document()
build()
```

A `.tar.gz` file will be produced in the parent directory. Then, the package can be installed through the `install.packages()` function as described above.

For further information on how to install packages please refer to the official CRAN manual page.

3. Package structure

3.1 Function list

The ebbc package is entirely written in R . All source files (`*.R`) are located within `/R/`. Each source file defines a single function. A list of the functions is provided in the following table; clicking on a link leads to the corresponding documentation in Part 4, where arguments, options and returned values are reported. Functions devoted to the generation of graphical output (and internally called by some of the functions listed in the table) are not exported to the package namespace.

`ebbc_preprocess`	Preprocesses a dataset
`ebbc_assessCriticalSigma`	Assesses, for the purpose of outlier identification, critical standard deviation (sigma) values
`ebbc_removeOutliers`	Removes outliers from a dataset according to a set of critical standard deviation (sigma) values
`ebbc_loadCriticalSigma`	Loads from file the outcomes of `ebbc_assessCriticalSigma`
`ebbc_bayes`	Trains the Bayesian classifier on a training dataset; alternatively, carries out a statistical analysis of features (means and variances, normality, cross-correlation)
`ebbc_classifyDataset`	Classifies a datasets according to a trained classifier
`ebbc_loadThresholds`	Loads from file the outcomes of `ebbc_bayes`

3.2 Preprocessing and dataset cleaning

The functions `ebbc_preprocess`, `ebbc_assessCriticalSigma` and `ebbc_removeOutliers`, as well as the load utility `ebbc_loadCriticalSigma`, provide the necessary tools to preprocess and clean datasets. The workflow concerning this first analysis stage is schematically shown in Fig. 3.1 and is summarized as follows.

1. Input datasets have to be preprocessed before training and classification. The `ebbc_preprocess` function transforms a "raw" dataset into a format that is suited to be used by any other function of the package. Besides properly checking and formatting the data, the function computes the mean and standard deviation of every multiplet in the dataset.

2. In order to properly remove outliers, critical standard deviation (sigma) values have to be assessed. The `ebbc_assessCriticalSigma` function carries out this task. Alternatively, a set of critical sigma values that were previously assessed on another dataset can be loaded from a file through the `ebbc_loadCriticalSigma` function.

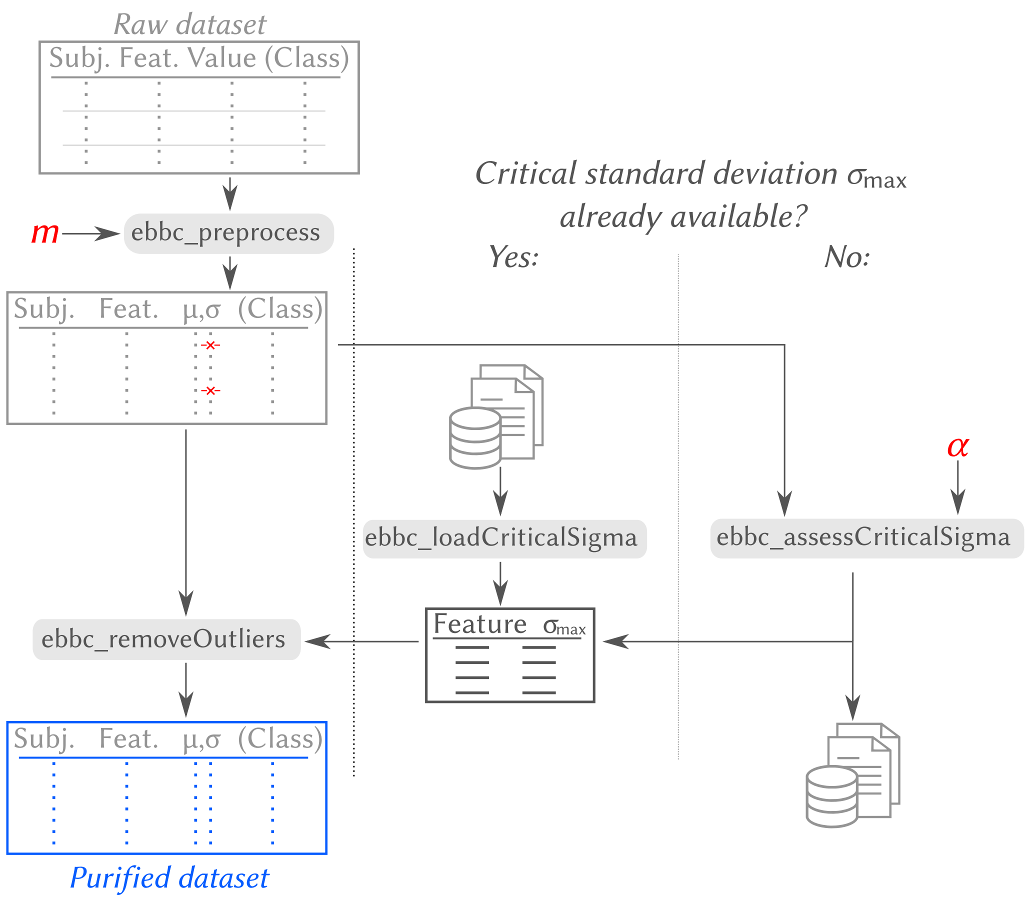


Figure 3.1: Workflow of the preprocessing and outlier removal functions.

3. Given a preprocessed dataset and a set of critical sigma values, the ebbc_removeOutliers function removes the outliers from the dataset and produces a purified dataset to be used in the subsequent analysis stages.

3.3 Training and classification

The functions ebbc_bayes and ebbc_classifyDataset, as well as the load utility ebbc_loadThresholds, provide the necessary tools to train a Bayesian classifier and use it for dataset classification. In addition, the ebbc_bayes function also allows to assess statistics of features. The workflow concerning this analysis stage is schematically shown in Fig. 3.2 and is summarized as follows. Notice that both training and classification require a preprocessed and purified dataset as described in Sec. 3.2.

1. A Bayesian classifier relies on a *sample score*, i.e. a linear combination of feature values, and a threshold that discriminates sample scores into two possible classification outcomes, henceforth referred to as "*Target*" and "*Versus*". The optimal threshold value has to be assessed by training the Bayesian classifier on a dataset whose entries have been already classified by using an alternative method. Training can be performed on a dataset by means of the ebbc_bayes function, which returns threshold values to be used for classification. Methodological details on the Bayesian classifier can be found in the *main reference* . Alternatively, a set of threshold values that were previously determined on another dataset can be loaded from a file through the ebbc_loadThresholds function.

2. In addition to training capability, the ebbc_bayes function allows to analyze statistical properties of features such as the separation between *Target* and *Versus* distributions, the outcomes of normality test, and the cross correlation between each pair of features.

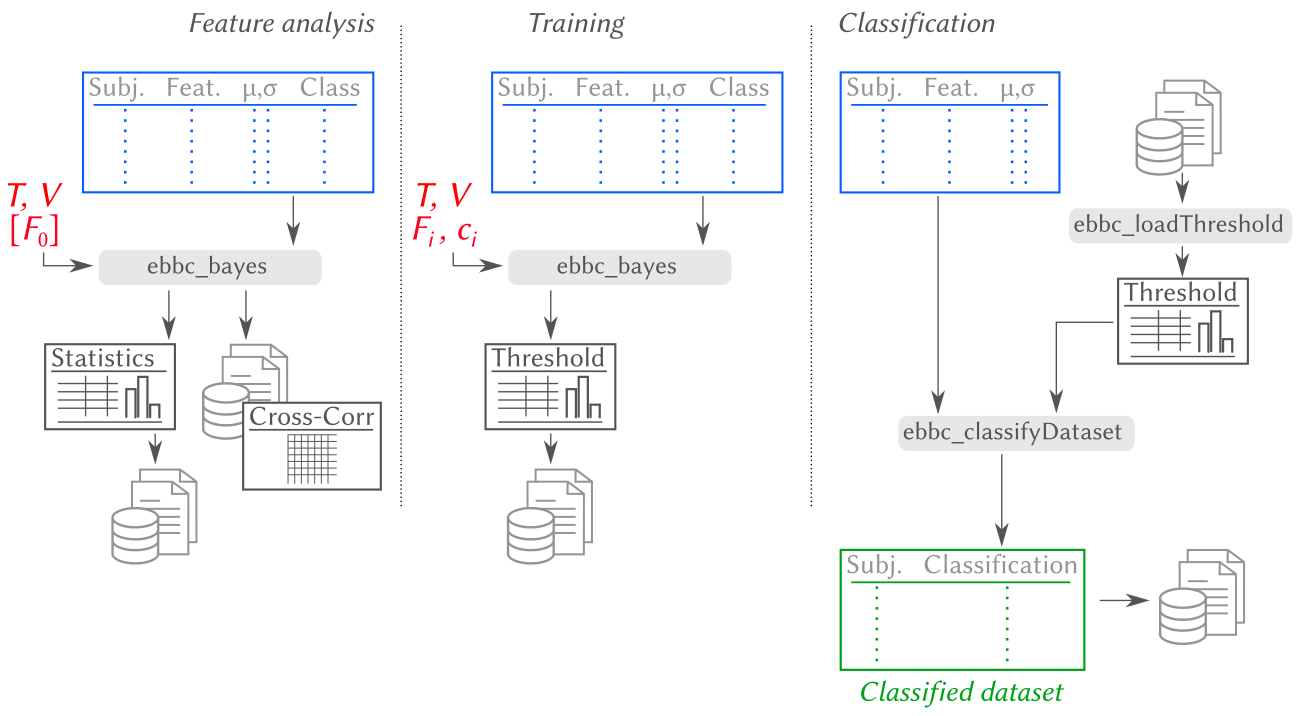


Figure 3.2: Workflow of the analysis, training and classification stage.

3. Given a preprocessed and purified dataset and given a set of threshold values corresponding to a trained classifier, the `ebbc_classifyDataset` carries out the classification of the dataset entries by assigning each of them either to the *Target* or the *Versus* set.

4. Detailed functions documentation

A brief documentation about each function can be evoked from within the R environment by typing

```
        help(<function name>)
        # e.g.
        help(ebbc_preprocess)
```

In the following, a more detailed documentation is reported. Functions are grouped as in Part 3. The mathematical aspects underlying each function are discussed in details in the *main reference* .

4.1 Preprocessing and dataset cleaning functions

Preprocess – *Preprocesses a dataset.*

```
preprocDataset <- ebbc_preprocess(inputDataset, multipletSize)
```

Arguments:

`inputDataset`	**Data frame** containing the input dataset. Three columns, labelled with `‘Feature’`, `‘Subject’`, `‘Value’`, must be present: if any of these is missing, the function exits. For each feature and subject, an n-tuplet of values can be present. The column labelled with `‘Class’`, if present, is also echoed in the output frame. Any other column is discarded.
`multipletSize`	**Integer** number corresponding to the size of multiplets to be used. If missing or invalid (e.g. $\leqslant 0$), the function exits. Multiplets having different size are discarded.

Returned value:

`preprocDataset`	**Data frame** containing the preprocessed dataset. The data frame contains the columns `‘Subject’`, `‘Feature’`, `‘Mean’`, `‘StdDev’`, `‘SampleSize’` and, if present in the input data frame, `‘Class’`.

Assess Critical Sigma – *Assesses, for the purpose of outlier identification, critical standard deviation (sigma) values.*

```
criticalSigmaValues <- ebbc_assessCriticalSigma(inputDataset,
                                                significanceLevel=0.05,
                                                outputFileName="",
                                                sep="\t")
```

Arguments:

`inputDataset`	**Data frame** containing the preprocessed dataset upon which the critical sigma values have to be assessed. Five columns, labelled with '`Subject`', '`Feature`', '`Mean`', '`StdDev`', '`SampleSize`', must be present: if any of these is missing, the function exits. Any other column, inclusively, if any, '`Class`', is ignored.
`significanceLevel`	**Floating-point** number corresponding to the significance level to be used. Default is 0.05, i.e. 5%.
`outputFileName`	**String** specifying the name (and path) of the file where critical sigma values have to be saved. If not assigned, a suitable filename is automatically generated.
`sep`	**Character** specifying the column separator for the output file. The default is tabulator (`\t`).

Returned value:

`criticalSigmaValues`	**Data frame** containing two columns: the first one, labelled with '`Feature`', for the feature names; the second one, labelled with '`SigmaMax`', for the related critical sigma values.

The function produces the following data file (plain text):

- A file containing the same data as the data frame returned by the function.

Load Critical Sigma – *Loads from file the outcomes of* `ebbc_assessCriticalSigma`.

```
criticalSigmaValues <- ebbc_loadCriticalSigma(inputFileName, sep="")
```

Arguments:

`inputFileName`	**String** specifying the name (and path) of the file. Two columns, labelled with '`Feature`' and '`SigmaMax`', must be present in the file: if any of these is missing, the function exits. Any other column is ignored.
`sep`	**Character** specifying the column separator. If none or empty (`""`) is specified, any space or tabulator is assumed to be a separator.

Returned value:

`criticalSigmaValues`	**Data frame** containing two columns: the first one, labelled with '`Feature`', for the feature names; the second one, labelled with '`SigmaMax`', for the related critical sigma values.

Remove Outliers – *Removes outliers from a dataset according to a set of critical sigma values.*

```
cleanedDataset <- ebbc_removeOutliers(inputDataset, criticalSigmaValues)
```

Arguments:

`inputDataset`	**Data frame** containing the preprocessed dataset that has to be purified from outliers. Five columns, labelled with ‘`Subject`’, ‘`Feature`’, ‘`Mean`’, ‘`StdDev`’, ‘`SampleSize`’, must be present: if any of these is missing, the function exits. The column labelled with ‘`Class`’, if present, is also echoed in the output frame. Any other column is discarded.
`criticalSigmaValues`	**Data frame** containing the assessed critical sigma values to be used for outlier removal. Two columns, labelled with ‘`Feature`’ and ‘`SigmaMax`’, must be present: if any of these is missing, the function exits. Any other column is ignored.

Returned value:

`cleanedDataset`	**Data frame** containing the purified dataset. The data frame contains the columns ‘`Subject`’, ‘`Feature`’, ‘`Mean`’, ‘`StdDev`’, ‘`SampleSize`’ and, if present in the input data frame, ‘`Class`’.

4.2 Training and classification functions

Bayes Training / Feature Analysis – *Trains the Bayesian classifier on a training dataset; alternatively, carries out a statistical analysis of features (means and variances, normality, cross-correlation).*

```
outputFrame <- ebbc_bayes(inputDataset, inputTargetList, inputVersusList="",
                          inputFeaturesList="", coeffList="",
                          outputFileBasename="", sep="\t",
                          plotFormat="pdf")
```

This function carries out two different tasks and thus can run in two different modes, henceforth referred to as "*Analysis mode*" and "*Training mode*".

- *Analysis mode*: carries out a statistical analysis of features (means and variances, normality, cross-correlation), either directly or upon subtraction of the expression of a "normalizer".
- *Training mode*: trains a Bayesian classifier by assessing the corresponding optimal threshold value and its uncertainty.

In order to select between *Analysis mode* and *Training mode*, the input parameters `inputFeaturesList` and `coeffList` have to comply with the requirements described below.

- *Analysis mode*: `coeffList` has zero length (default setting); `inputFeaturesList` can either have zero length (default setting) or unitary length; in the latter case, the single entry of `inputFeaturesList` is assumed to be the normalizer.
- *Training mode*: `inputFeaturesList` and `coeffList` have the same nonzero length.

In the case of any other setting of `inputFeaturesList` and `coeffList`, the function exits.

Arguments:

`inputDataset`	**Data frame** containing the preprocessed and purified dataset to analyze or to train on. Six columns, labelled with '`Subject`', '`Feature`', '`Mean`', '`StdDev`', '`SampleSize`' and '`Class`', must be present: if any of these is missing, the function exits. Any other column is ignored.
`inputTargetList`	**List** of classes, specified by means of, for example, `c("A", "B", ...)`, to be used as *Target* classification set.
`inputVersusList`	**List** of classes, specified by means of, for example, `c("C", "D", ...)`, to be used as *Versus* classification set. If no list is entered (default setting), the *Versus* classification set is taken as the complement of the *Target* classification set within `inputDataset`.
`inputFeaturesList`	**List** of features, specified by means of, for example, `c("F1", "F2", ...)`, to be used by the classifier in *Training mode*. Subjects for which any of the chosen features is absent are discarded. In *Analysis mode* two possibilities are given: if no normalizer is used, the `inputFeaturesList` argument is omitted; alternatively, the `inputFeaturesList` argument has to correspond to the label of the feature used as a normalizer.
`coeffList`	**List** of coefficients, specified by means of, for example, `c(1.0, 0.5, ...)`, to be used by the classifier in *Training mode*. The number and the order of the coefficients have to be the same as the number and order of features. In *Analysis mode*, the `coeffList` argument is omitted.

`outputFileBasename`	**String** specifying the name and path, without extension, of the files where the results are stored. If not assigned, a filename is automatically generated.
`sep`	**Character** specifying the column separator for the output file. The default is tabulator (`\t`).
`plotFormat`	**String** specifying the format of graphic files. Possible choices are "`pdf`" (default) or "`png`".

Returned value:

`outputFrame`	In *Training mode*: **Data frame** containing the following columns: '`Threshold`', with the threshold value assessed by the training procedure; '`DeltaThreshold`', with the corresponding uncertainty; '`DPrime`', with the normalized difference between the *Target* and *Versus* sample means, namely $(\bar{x}_T - \bar{x}_V)/\sqrt{(s_T^2 + s_V^2)/2}$; '`Pc`', with the probability of correct classification; '`ChiUp`' and '`DChiUp`', with the threshold and the corresponding uncertainty, respectively, for the odds 90:10; '`ChiDown`', '`DChiDown`', with the threshold and the corresponding uncertainty, respectively, for the odds 10:90.
`outputFrame`	In *Analysis mode*: **Data frame** containing the statistics assessed by analyzing each feature. The data frame contains the columns '`Feature`', '`Classification`', '`NumberOfSubjects`', '`Mean`', '`StdDev`', '`NormalityTest`' (p-value of Shapiro-Wilk test of normality), '`t-test`' (p-value of Student's t-test to check the null hypothesis that the *Target* and *Versus* sets have the same population mean).

In *Training mode* the function produces the following files:

- **Subjects' sample score and classification** file (`<outputFileBasename>.dat`) containing, for each subject, the sample score and the *a priori* classification.
- **Output parameters** file (`<outputFileBasename>.txt`) containing the same data as the `outputFrame` and, in addition, the area under the ROC curve (see below) and the boundary values of the related confidence interval.
- **Histograms** (`<outputFileBasename>_histogram.<plotFormat>`) of the classifier scores both in the case of the *Target* and the *Versus* set.
- **Score diagram** (`<outputFileBasename>_score.<plotFormat>`) of the sample scores highlighting the *Target* and *Versus* classification.
- **ROC curve** (Receiver Operating Characteristic) plot (`<outputFileBasename>_ROC.<plotFormat>`). The area under the curve is reported in the **Output parameters** file.

In *Analysis mode* the function produces the following files:

- **Output parameters** file (`<outputFileBasename>.txt`) containing the same data as the `outputFrame` and, in addition, the correlation matrix between features and the related p value matrix computed on three different sets of subjects: subjects whose classification belongs to the *Target* set, subjects whose classification belongs to the *Versus* set, subjects whose classification belongs to the union of the *Target* and the *Versus* set. In this last case, a matrix of coefficients that, according to the standard deviation and correlation analysis, maximize the classifier's performance (see Eq. (9) of the *main reference*) is also reported.
- **Histograms** (`<outputFileBasename>_[<feature>_]histogram.<plotFormat>`), for each analyzed feature, of the feature values both in the case of the *Target* and the *Versus* set.

Load Threshold Values – *Loads from file the outcomes of* `ebbc_bayes`.

```
thresholdFrame <- ebbc_loadThresholds(inputFileName, sep="")
```

Arguments:

`inputFileName`	**String** specifying the name (and path) of the file. Six columns, labelled with `‘Threshold’`, `‘DeltaThreshold’`, `‘ChiUp’`, `‘DChiUp’`, `‘ChiDown’`, `‘DChiDown’`, must be present in the file: if any of these is missing, the function exits. Any other column is ignored.
`sep`	**Character** specifying the column separator. If none or empty (`""`) is specified, any space or tabulator is assumed to be a separator.

Returned value:

`thresholdFrame`	**Data frame** containing six columns: `‘Threshold’`, `‘DeltaThreshold’`, `‘ChiUp’`, `‘DChiUp’`, `‘ChiDown’`, `‘DChiDown’`.

Classify Dataset – *Classifies a datasets according to a trained classifier.*

```
classifiedDataset <- ebbc_classifyDataset(inputDataset, inputFeaturesList,
                                           coeffList, inputThreshold,
                                           outputFileBasename="", sep="\t",
                                           plotFormat="pdf")
```

Arguments:

`inputDataset`	**Data frame** containing the preprocessed and purified dataset to be classified. Five columns, labelled with '`Subject`', '`Feature`', '`Mean`', '`StdDev`', '`SampleSize`', must be present: if any of these is missing, the function exits. Any other column is ignored.
`inputFeaturesList`	**List** of features, specified by means of, for example, `c("F1", "F2", ...)`, to be used by the classifier. Subjects for which any of the chosen features is absent are discarded.
`coeffList`	**List** of coefficients, specified by means of, for example, `c(1.0, 0.5, ...)`, to be used by the classifier. The number and the order of the coefficients have to be the same as the number and order of features.
`inputThreshold`	**Data frame** containing the results of a classifier training, e.g. the `outputFrame` returned by `ebbc_bayes` in *Training mode* or loaded by the function `ebbc_loadThresholds`. The column '`Threshold`' has to be present.
`outputFileBasename`	**String** specifying the name and path, without extension, of the files where the results are stored. If not assigned, a filename is automatically generated.
`sep`	**Character** specifying the column separator for the output file. The default is tabulator (`\t`).
`fileFormat`	**String** specifying the format of graphic files. Possible choices are "`pdf`" (default) or "`png`".

Returned value:

`classifiedDataset`	**Data frame** containing the classified dataset. The data frame contains the columns '`Subject`', '`Classification`', '`Score`'.

The function produces the following files:

- **Subjects' sample score and classification** file (`<outputFileBasename>.dat`) containing, for each subject, the sample score and the (*a posteriori*) classification.
- **Score diagram** (`<outputFileBasename>_score.<plotFormat>`) of the sample scores highlighting the *Target* and *Versus* classification.

5. Examples

The following examples show how to preprocess and analyze datasets by means of the EBBC functions. The datasets used in the examples, as well as a script containing all instructions reported here, can be found in /examples/.

Data format, loading and preprocessing

The folder /examples/ contains the raw dataset dataset_alpha.dat. Opening the file with a text editor reveals the following data lines:

```
Subject Feature Value   Class
1       FX      17.9494 A
1       FX      18.1404 A
1       FX      17.8583 A
1       FY      14.1306 A
...
```

The dataset contains 120 subjects (numbered 1-120) and three features (FX, FY, FZ). In this case the Class column is present: the dataset is thus suited to be used as a training one. The EBBC package does not provide any function to load raw datasets; a simple read.table does the job:

```
datasetAlpha <- read.table(file="dataset_alpha.dat", header=TRUE)
```

In datasetAlpha, the size of multiplets is 3. The ebbc_preprocess function is called as

```
preprocessedDatasetAlpha <- ebbc_preprocess(datasetAlpha, multipletSize=3)
```

The output data frame processedDatasetAlpha looks like this:

```
Subject Feature Mean    StdDev  SampleSize      Class
1       FX      17.9827 0.1440  3               A
1       FY      14.0644 0.0993  3               A
1       FZ      11.8195 0.1243  3               A
10      FX      7.6036  0.3606  3               B
...
```

The preprocessed data frame processedDatasetAlpha does not contain an entry for subject 120, which was discarded due to its features being reported in doublets, instead of triplets.

Outlier removal

In order to remove outliers from a dataset, the critical standard deviation (sigma) values that discriminate outliers have to be assessed. Upon setting, for example, the significance level to 5%, the critical sigma is assessed by means of the following function call:

```
criticalSigmaValuesAlpha <- ebbc_assessCriticalSigma(preprocessedDatasetAlpha,
    significanceLevel=0.05, outputFileName="criticalSigma_datasetAlpha.dat")
```

The resulting criticalSigmaValues data frame contains a list of features and the related critical sigma values:

```
Feature SigmaMax
FX      0.51
FY      0.5
FZ      0.53
```

The ebbc_assessCriticalSigma function also saves a copy of criticalSigmaValuesAlpha on the file criticalSigma_datasetAlpha.dat, within the current directory /examples/. The file will be used in later examples.

The critical values stored in criticalSigmaValues can now be used to purify a dataset from outliers by calling

```
purifiedDatasetAlpha <- ebbc_removeOutliers(preprocessedDatasetAlpha,
    criticalSigmaValues)
```

As a consequence of the outliers that have been discarded, the purifiedDatasetAlpha data frame (345 entries) turns out to be smaller than preprocessedDatasetAlpha (357 entries).

Features analysis

The purifiedDatasetAlpha data frame is ready to be used to train a classifier. The Target class is "A", while the Versus classes are "B" and "C":

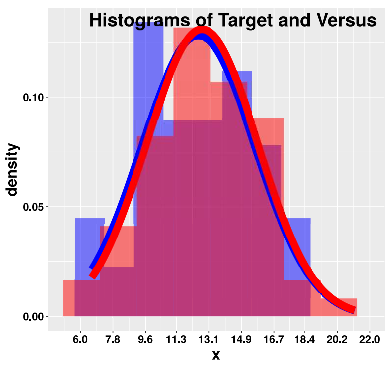
```
Target <- c("A")
Versus <- c("B", "C")
```

In this case the coefficients of the linear combination of features to be used as classifier are *a priori* unknown. In order to determine them, the ebbc_bayes function is first used in *"Analysis mode"*. By analyzing each feature separately, the function provides insight on which features are better suited to discriminate between Target and Versus.

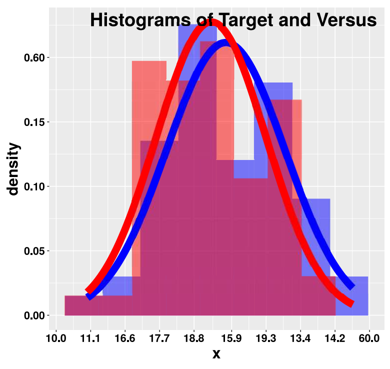
First, ebbc_bayes is called by omitting feature and coefficient lists (see function documentation):

```
statisticsAlpha <- ebbc_bayes(purifiedDatasetAlpha, inputTargetList=Target,
    inputVersusList=Versus, outputFileBasename="featureAnalysis_datasetAlpha")
```

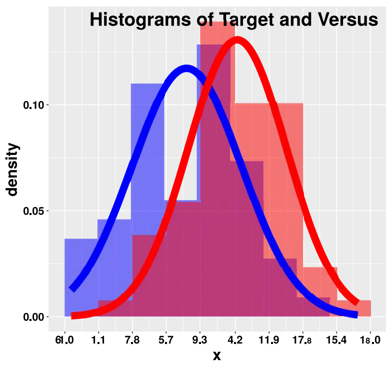
A list of the generated files is displayed by on-screen log messages that automatically follow the function call. In the present example, the list includes three graphic files, one for each feature (FX, FY, FZ). Each graphic file contains two histograms. The content of the three files is shown in the following figure: Target and Versus data are respectively shown in blue and red.



(FX)



(FY)

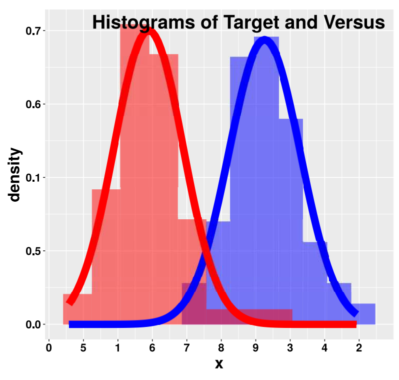


(FZ)

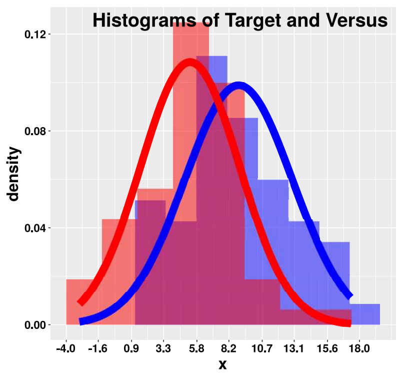
As shown above no clear separation between Target and Versus sets is provided by any of the three features. The function returns more detailed statistical information, including cross-correlations between features: a discussion on how to use this information to infer an improved linear combination of the features can be found in the *main reference*. However, for the sake of this example, let us repeat the last analysis by using a normalizer, i.e. by selecting one feature to be subtracted from the others. To carry out this operation, ebbc_bayes is called by listing a single feature in its arguments, without any coefficient (see function documentation):

```
statisticsAlphaNorm <- ebbc_bayes(purifiedDatasetAlpha, inputTargetList=Target,
    inputVersusList=Versus, inputFeaturesList=c("FZ"), outputFileBasename="
    featureAnalysisNorm_datasetAlpha")
```

In this case only two graphic files are generated, one for ΔFX = FX−FZ and one for ΔFY = FY−FZ, as shown below:



(FX - FZ)



(FY - FZ)

The combination FX−FZ appears to discriminates the two sets. For the sake of this example, one could then use as classifier the linear combination 1·FX+(−1)·FZ.

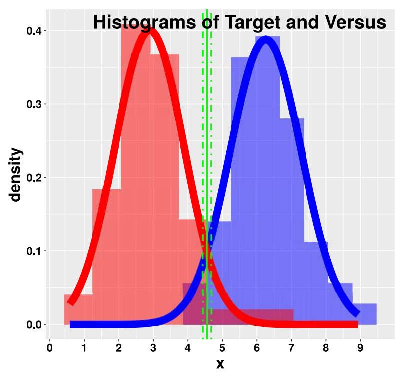
```
featuresToUse <- c("FX", "FZ")
coefficientsToUse <- c(1.0, -1.0)
```

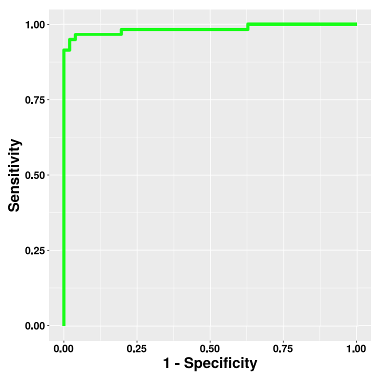
Training of a Bayesian classifier

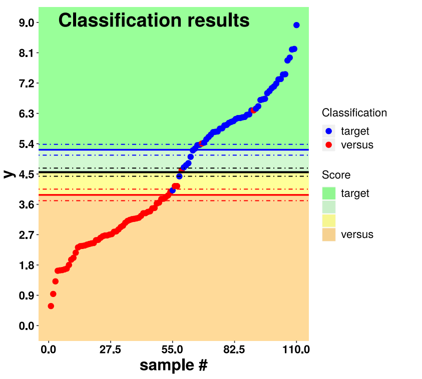
In order to train a Bayesian classifier, the ebbc_bayes function has to be used in *"Training mode"*. The required parameters – namely the Target and Versus sets, and the lists of features and coefficients – have to be entered upon the analysis carried out in *"Analysis mode"*. The ebbc_bayes function is thus called as follows:

```
thresholdValues <- ebbc_bayes(purifiedDatasetAlpha, inputTargetList=Target,
    inputVersusList=Versus, inputFeaturesList=featuresToUse, coeffList=
    coefficientsToUse, outputFileBasename="threshold_datasetAlpha")
```

The function saves a file containing the threshold values, as well as the three figures shown below and corresponding to the Target-Versus histograms (left), the ROC curve (center), and a plot of all scores (right).







Classification of a test dataset

Once trained, the classifier can be used to classify other datasets. As an example, this section shows how to use the previously-trained classifier to classify the second dataset available in /examples/, namely dataset_-

beta.dat, which includes 200 subjects. In this case, no Class column is present. Loading and preprocessing are identical to the case of the alpha dataset:

```
datasetBeta <- read.table(file="dataset_beta.dat", header=TRUE)
preprocessedDatasetBeta <- ebbc_preprocess(datasetBeta, multipletSize=3)
```

Outliers are then removed from the dataset by using the critical sigma values estimated in the case of the alpha dataset and loaded from the related file:

```
criticalSigmaValuesAlpha <- ebbc_assessCriticalSigma(preprocessedDatasetAlpha,
    significanceLevel=0.05, outputFileName="criticalSigma_datasetAlpha.dat")
purifiedDatasetBeta <- ebbc_removeOutliers(preprocessedDatasetBeta,
    criticalSigmaValues)
```

Similarly, the threshold values previously obtained by training the classifier are loaded from the related file:

```
thresholdValues <- ebbc_loadThresholds("threshold_datasetAlpha.txt")
```

The dataset is classified by calling ebbc_classifyDataset. The lists of features and coefficients are the same as the ones used in the training step:

```
featuresToUse <- c("FX", "FZ")
coefficientsToUse <- c(1.0, -1.0)
classifiedBeta <- ebbc_classifyDataset(purifiedDatasetBeta, inputFeaturesList=
    featuresToUse, coeffList=coefficientsToUse, inputThreshold=thresholdValues,
    outputFileBasename="classified_datasetBeta")
```

Finally, the data file classified_datasetBeta.dat contains the results of the classification:

```
Subject Classification  Score
52      versus          0.4560
49      versus          0.8917
167     versus          1.0935
57      versus          1.1289
...
189     target          8.2310
192     target          8.2479
72      target          8.7586
```