Technical Documentation: **clusteval**

An Integrated Clustering Evaluation Framework for Cluster Analysis

Christian Wiwie

June 3, 2015

Contents

1 Introduction

In the next chapter we will discuss briefly how **clusteval** is organized, how it works in principle and what can be achieved through its use.

Chapter 9 explains, how the framework is to be installed and, amongst other things, what prerequisites are necessary to install and use it successfully. For Debian 6 (Squeeze) we provide a detailed step-by-step installation manual. For those of you, who do not want to install it manually but want to try out the framework right away, we offer a VirtualBox hard drive image which contains a Debian 6 installation, together with the framework and all its prerequisites (see 9.1).

Chapter 10 demonstrates the usage of the framework by a simple use case scenario from the very start to the very end. If you want more detailed information about the use cases **clusteval** assists with, please consult [1].

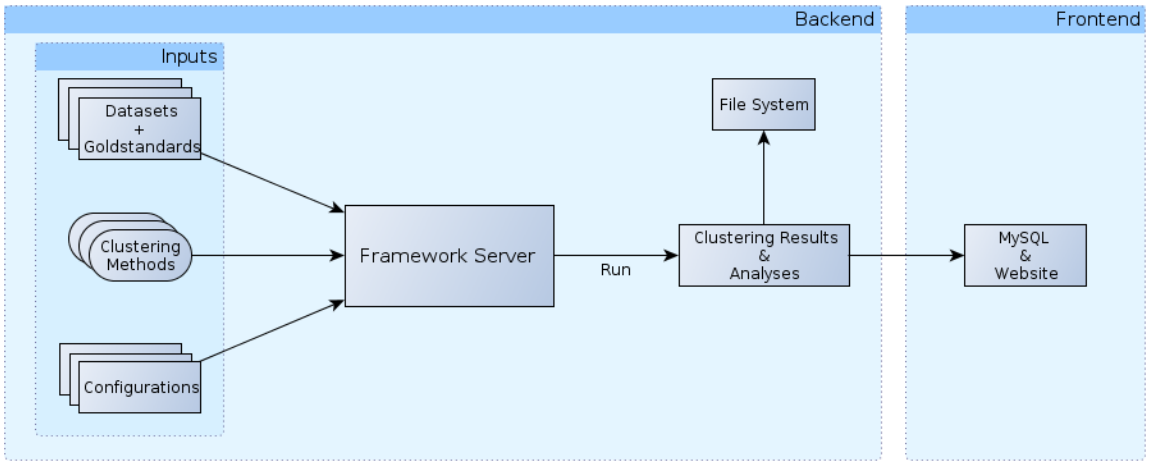
If you are a programmer and you want to extend **clusteval** , you will find more information in section 11.

2 The Framework

Our framework **clusteval** is intended to perform automatized cluster analysis of arbitrary datasets and clustering methods. The goal is, that any clustering method known to the framework can be applied to any known dataset (with certain exceptions, partly inflicted by the clustering methods itself and partly inflicted by the framework constraints).

In general **clusteval** is divided into a backend and a frontend. Figure 1 shows the general structure of the framework. The backend is reponsible for doing all the calculations including clusterings and the frontend has only visualization purposes.

Figure 1: General framework flow

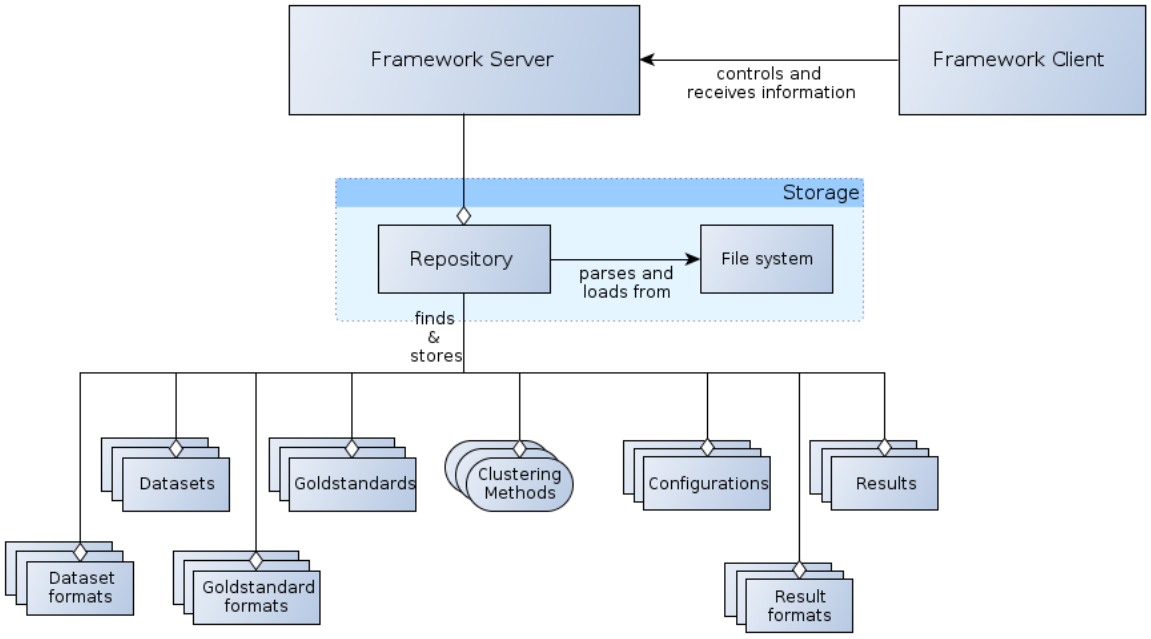


At first we will give an overview of the backend in 4 and then of the frontend in 6.

3 Backend

The backend of the framework is responsible for all calculations. It consists of a server and a client component. The server takes commands from the client and does the calculations internally and in a multithreaded way. The client can query the server to get the current status of a calculation or to give commands to the server which might control current processes. Figure 2 shows the general structure of the backend and the components it contains.

Figure 2: The backend structure



4 Backend Server

In principle the core of all tasks of the backend server is, to apply clustering methods to data (dataset and goldstandard) in an automatized and autonomous way, using only the configurations and inputs the user specified. The produced results have to be parsed in such a way, that the frontend can easily collect certain information and visualize them. Thus the components used by the backend that are not directly included into the framework programatically but need to be provided can be summarized as

- Data (Datasets & Goldstandards)
- Clustering Methods
- Configurations

Additional components that can be extended and might be needed in case the provided standard functionality of the framework is not sufficient for the user

- Data input formats (including parsers)
- Run result formats (including parsers)
- Clustering Quality Measures
- Distance Measures
- Parameter Optimization Methods
- Dataset types
- (Data, Run & Run-Data) Statistics
- Dataset generators

All these components have to be located in the **repository** of the framework (see 4.1 for more details). The repository is a file system hierarchy located at a specified path and contains all components used by and available to the framework. Datasets, clustering methods or configurations outside the repository cannot be used by the framework.

After these components have been made available to the backend, they can be combined almost arbitrarily. In the following we will describe the dependencies of each of these components which need to be fulfilled such that a new component of each type can be recognized and used by the framework.

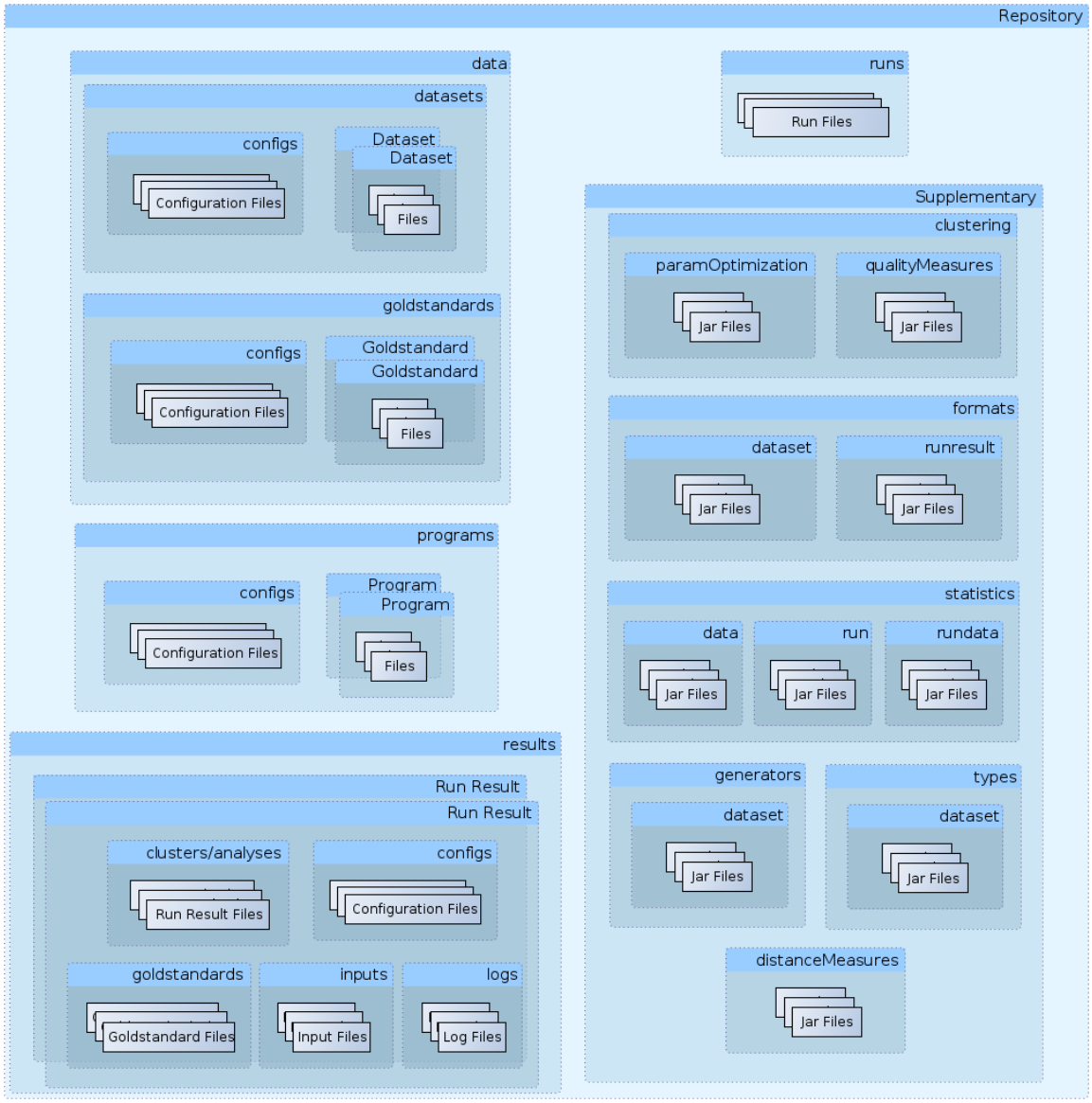
4.1 Repository

The backend server is based on a central repository which concludes all files in its folder structure. You can easily start a framework with a different set of files by simply using another repository.

4.1.1 Repository folder structure

- **data**: Contains all data-related files.
  - **configs** [*.dataconfig]: Contains the data configuration files 4.9.1.
  - **datasets**: Contains all dataset-related files 4.3.
    - **configs** [*.dsconfig]: Contains the dataset configuration files 4.9.3.
    - [**subfolder for every dataset**]: The dataset files themselves.
  - **goldstandards**: Contains all goldstandard-related files 4.4.
    - **configs** [*.gsconfig]: Contains the goldstandard configurations 4.9.5.
    - [**subfolder for every goldstandard**]: Contains the goldstandard files themselves.
- **programs**: Contains all program-related files 4.2.

Figure 3: Repository folder structure



  - **configs** [*.config]: Contains all program configuration files 4.9.7.
  - [**subfolder for every program**]: The program files themselves.
- **results**: The results of run executions 4.11.
  - [**subfolder for every run execution**]: A subfolder contains the results of one run execution.
    - **clusters**: The clustering results, including clustering qualities and graphics.
    - **configs**: Copies of all used configuration files of this run execution to enable exact reproduction.
    - **inputs**: Copies of all used inputs of this run execution to enable exact reproduction.
    - **logs**: All log files corresponding to this run execution.
- **runs**: All run-related files 4.10.
  - [***.run: a file for every run**]: Contains the run-files.
- **supp**: Contains supplementary material.
  - **clustering**: Supplementary material related to clusterings.
    - **paramOptimization**: Contains clustering parameter optimization methods 4.8.
      - [***.jar**]: Each jar-file corresponds to a parameter optimization method and is loaded dynamically by the framework.
    - **qualityMeasures**: Contains clustering quality measures 4.6.
      - [***.jar**]: Each jar-file corresponds to a clustering quality measure and is loaded dynamically by the framework.
  - **formats**: Contains all formats used by the framework.
    - **dataset**: Contains all dataset formats 4.5.
      - [***.jar**]: Each jar-file corresponds to a dataset format and is loaded dynamically by the framework.
    - **runresult**: Contains all runresult formats 4.5.
      - [***.jar**]: Each jar-file corresponds to a runresult format and is loaded dynamically by the framework.

4.2 Clustering Methods

Clustering Methods (technically also called **Programs** throughout this guide) can be executed by the framework, and be applied to data to calculate clusterings. In order to include a new clustering method and use it within the framework,

- the **executable** of the method has to be made available to the framework (an exception are Clustering Methods provided through the R framework, see 4.2.2),
- the method itself has to be specified in a configuration file (called **program configuration**)

- all other components (e.g. **input** and **output formats**) specified in the program configuration need to be available.

The configuration file for a clustering method (short **program configuration**) contains all information required, such that the framework can execute the method in an automatized and autonomous way. These information include for example, among others, the name of the method, its supported input formats, its output format, its parameters (including type and valid range of values). An exact description of how program configurations look like and which options and settings need to be specified can be found in 4.9.7.

clusteval ships with a set of clustering methods including

- Affinity Propagation
- Hierarchical Clustering$^{(Rserve)}$
- K-Means$^{(Rserve)}$
- Markov Clustering
- Spectral Clustering$^{(Rserve)}$
- Transitivity Clustering

For every of these clustering methods a **program configuration** is also provided, such that they are directly usable from the start. Of course these program configurations can be modified and adapted to the user's needs.

4.2.1 Standalone Programs

are programs that come as an executable. Those can be performed by the framework, after they have been specified in a program configuration. The executable needs to be compatible to the server architecture **clusteval** runs on and they need to be executable (+x modifier). How you can add your own standalone programs into the framework can be found here 11.1.1

4.2.2 R Programs$^{(Rserve)}$

are programs, that are implemented within some R package. Arbitrary methods implemented in R can be used as a program, as long as they can be made available within R on the server. This implies that the corresponding R Program is available for your R version and that it can be compiled and installed on your server architecture. How you can add your own R Programs into the framework can be found here 11.1.2.

4.3 Datasets

To add a dataset to the framework and make it usable, such that clustering methods can be applied to it, you have to

- insert the dataset file into the repository of the backend
- insert a header into the dataset which specifies its format, format version and type
- specify the dataset in a configuration file (called **dataset configuration**)

The dataset configuration contains the name and path of the dataset and other details how a possible conversion of the dataset should be handled. An exact description of how dataset configurations look like and which options and settings need to be specified can be found in 4.9.3.

clusteval ships with a set of datasets of different types (PPI, Gene Expression, Protein similarity, Word-Sense disambiguation), for example

- subsets of SCOP Astral95 v1.61,
- Brown et al. protein simliarities,
- leukemia microarray gene expression (Broad Institute),
- word context counts for word-sense disambiguation

4.4 Goldstandards

When assessing the qualities of a resulting clustering for most measures a goldstandard corresponding to the dataset is needed. The comparison of the clustering and goldstandard is then integrated into the calculation of the clustering quality measure (see 4.6).

Goldstandards for a dataset are **in principle optional**. Some operations can be also performed without a goldstandard and also some clustering quality measures do not require a goldstandard (for example Silhouette Value).

Nevertheless, to be able to perform **all operations** on the dataset, a **goldstandard is required**.

To add a goldstandard to the framework, you have to

- insert the goldstandard file into the repository of the backend
- specify the goldstandard in a configuration file (called **goldstandard configuration**)

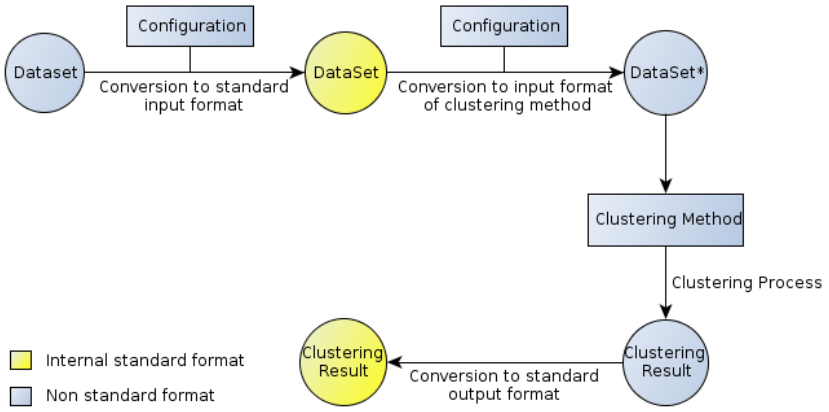
The goldstandard configuration contains the name and path to the goldstandard file. Since the framework does only support one goldstandard format, this does not need to be provided. An exact description of how goldstandard configurations look like and which options and settings need to be specified can be found in 4.9.5.

4.5 Input & Output formats

As already mentioned, datasets have their own formats and clustering methods can require different input and output formats. The general process how these formats link together can be visualized as seen in figure 4.

A dataset of a certain format known to the framework is converted into the standard input format of the framework using the parser belonging to the format of the dataset. Then the dataset in the standard format is converted to any of the supported input formats of the clustering method using the parser belonging to the chosen supported input format of the clustering method. Now the clustering method is applied to the dataset in the supported format. A clustering result is produced in the format of the clustering method. This result is then converted to the standard output format of the framework using the parser belonging to the format of the result. The framework then has access to the clustering results of the clustering method applied to the dataset in a format it understands, such that diverse operations can be performed on the result.

Figure 4: Format conversion processes



If a new clustering method is added to the framework, its input and output formats need to be known to the framework.

clusteval ships with a set of supported input and output formats. The input formats are

- APRowSimDataSetFormat
- BLASTDataSetFormat
- MatrixDataSetFormat
- RowSimDataSetFormat
- SimMatrixDataSetFormat
- TransClustSimMatrixDataSetFormat

and the output formats are

- APRunResultFormat
- MCLRunResultFormat
- TabSeparatedRunResultFormat
- TransClustRunResultFormat

If the new clustering method requires another input format not in the list, you will have to make it available to the framework by writing

- a wrapper class for this dataset format and
- a parser, which converts the standardized input format of this framework to this input format.

If the new clustering method has a so far unknown output format, you will have to provide

- a wrapper class for this output format as well as
- a parser that converts that format to the standardized output format.

When this clustering method is applied to a dataset, the resulting clustering in the new format is converted to a standardized output format using your parser, such that further analyses can be performed regardless of the used clustering method.

For more information on how the framework can be extended by new input or output formats see 4.5 respectively.

4.5.1 Standard input format

The standard input format is a SimMatrixDataSetFormat which is described under 4.5.6.

4.5.2 APRowSimDataSetFormat

This input format is used by Affinity Propagation. It is similar to the RowSimDataSet-Format, except that it accepts only numbers as ids and that it leaves out lines where id1=id2 such that it looks like this:

```
1 1       2       0.2
2 1       3       0.6
3 2       1       0.2
4 2       3       0.5
5 3       1       0.6
6 3       2       0.5
```

4.5.3 BLASTDataSetFormat

The BLAST dataset format needs a all-vs-all BLAST file and a corresponding FASTA file.

Please note, that the FASTA file has to be named exactly like the BLAST file plus the additional .fasta extension. If the BLAST file is named like MyBlastFile.blast, then the FASTA file only be found by the framework, when it lies in the same directory and is named MyBlastFile.blast.fasta.

```
1  d1dlwa_ d1dlwa_ 77.59   116     0       0       1       ↩
     ↪ 116     1       116     4.4e-46 172.6
2  d1dlwa_ d1idra_ 40.30   67      40      0       19      85↩
     ↪       31      97      3.2e-12 60.08
3  d1dlwa_ d1dlya_ 32.00   100     62      1       19      ↩
     ↪ 116     19      118     1.6e-11 57.77
4  d1dlwa_ d2gdm__ 45.45   22      12      0       60      81↩
     ↪       89      110     0.435   23.10
5  d1dlwa_ d1cqxa1 37.50   24      15      0       62      85↩
     ↪       79      102     0.969   21.94
6  d1dlwa_ d1cqxa1 40.00   5       3       0       71      75↩
     ↪       134     138     4975.0  9.62
7  d1dlwa_ d1kr7a_ 31.25   32      22      0       41      72↩
     ↪       45      76      4.8     19.63
8  d1dlwa_ d1kr7a_ 50.00   8       4       0       50      57↩
     ↪       91      98      202.0   14.24
9  d1dlwa_ d1kr7a_ 66.67   3       1       0       50      52↩
     ↪       86      88      6497.5  9.23
10 d1dlwa_ d1edg__ 45.00   20      11      0       47      66↩
     ↪       262     281     6.3     19.25           ↩
     ↪
```

```
1 >d1dlwa_
2 slfeqlggqaavqavtaqfyaniqadatvatffngidmpnqtnktaaflc
3 >d1dlya_
4 slfaklggreaveaavdkfynkivadptvstyfsntdmkvqrskqfafla
5 >d1idra_
6 gllsrlrkrepisiydkiggheaievvvedfyvrvladdqlsaffsgtnm
7 >d1kr7a_
8 mvnwaavvddfyqelfkahpeyqnkfgfkgvalgslkgnaayktqagktv
```

4.5.4 MatrixDataSetFormat

This input format is a absolute dataset format, that means it contains samples together with their absolute coordinates:

```
1 ALL_19769_B.cell          759     169     54      36
2 ALL_19769_B.cell_2        1062    88      235     38
3 ALL_28373_B.cell          822     196     150     120
4 ALL_28373_B.cell_2        1068    146     221     16
5 ALL_9692_B.cell           1455    53      217     169
```

4.5.5 RowSimDataSetFormat

This input format consists of tab-separated rows looking as follows:

```
1 id1     id1     1.0
2 id1     id2     0.2
3 id1     id3     0.6
4 id2     id1     0.2
5 id2     id2     1.0
6 id2     id3     0.5
7 id3     id1     0.6
8 id3     id2     0.5
9 id3     id3     1.0
```

4.5.6 SimMatrixDataSetFormat

The SimMatrixDataSetFormat is a complete quadratic tab-separated similarity matrix with header row and column (containing ids). A dataset file with the SimMatrixDataSetFormat could look as follows:

```
1         id1     id2     id3
2 id1     1.0     0.2     0.6
3 id2     0.2     1.0     0.5
4 id3     0.6     0.5     1.0
```

There are no spaces in this file, only tabs. It does not necessarily need to be symmetric, it depends on the clustering method, whether it supports asymmetric similarity data.

4.5.7 TransClustSimMatrixDataSetFormat

This format can be used by Transitivity Clustering and it looks as follows:

```
1 3
2 id1
3 id2
4 id3
5 0.2     0.6
6 0.5
```

The number in the first line is the number of ids. It is followed by all ids in the next lines. Then it follows the tab-separated upper half of the similarity matrix. This format expects that the similarity matrix is symmetric.

4.5.8 Standard output format

The standard output format contains one clustering generated for parameter values $p_1 = v1$,...,$p_K = vK$ in one line with clusters $c1, ..., cK$, cluster sizes $size(ci) = si$. Every cluster ci contains elements $e\_i\_1, ..., e\_i\_si$ with fuzzy coefficients $f\_i\_1, ..., f\_i\_si$. The format for this looks as follows:

```
p1,...,pK Clustering
v1,...,vK e_1_1:f_1_1,...,e_1_s1:f_1_s1;...;e_K_1:f_K_1,...,e_K_sK:f_K_sK
```

The parameter names and values on the left have to be separated by a TAB from the string "Clustering" and the clustering on the right. If the fuzzy coefficients are missing, the framework will not be able to parse the result file.

4.5.9 APRunResultFormat

This is the result format of Affinity Propagation.

4.5.10 MCLRunResultFormat

This is the result format of Markov Clustering.

4.5.11 TransClustRunResultFormat

This is the result format of Transitivity Clustering.

4.6 Clustering Quality Measures

The clustering quality measures are used by the framework to assess the quality of calculated clusterings. **clusteval** ships with a standard set of clustering quality measures

- F1 & F2-Score
- False Discovery Rate (FDR)
- False Positive Rate (FPR)
- Rand Index
- Sensitivity
- Specificity

- Silhouette Value (Java & R implementation)

Please check 4.6 for more information on how to extend the framework by new clustering quality measures.

4.7 Distance Measures

The distance measures are used when converting absolute datasets (containing absolute coordinates) to relative datasets (pairwise similarities). The distance measures define how to assess the similarity between a pair of objects given their absolute coordinates. **clusteval** ships with the following distance measures

- Euclidian
- Hoeffding D Statistic$^{(Rserve)}$
- Pearson Correlation Coefficient$^{(Rserve)}$
- Spearman Correlation Coefficient$^{(Rserve)}$

Please check 4.7 for more information on how to extend the framework by new distance measures.

4.8 Parameter Optimization Methods

The backend can perform automatized and autonomous optimization of parameters of clustering methods. This is an iterative procedure where the backend assesses qualities of clustering results of the last iteration and adapts the parameter for the next iteration in order to find optimal parameters for the method on the given data. The parameter optimization method determines the following aspects:

1. the number of iterations of the optimization process
2. the parameter sets evaluated
3. the handling of diverging iterations
4. the storage of the iteration results in RAM

clusteval ships with the following set of parameter optimization methods

- Affinity Propagation Divisive Parameter Optimization Method
- Affinity Propagation Parameter Optimization Method
- Divisive Parameter Optimization Method
- Gap Statistic Parameter Optimization Method

- Layered Divisive Parameter Optimization Method
- Transitivity Clustering Parameter Optimization Method
- Transitivity Clustering Quantile Parameter Optimization Method

Detailed information about each of these methods can be found in [1]. Please check 4.8 for more information on how to extend the framework by new parameter optimization methods.

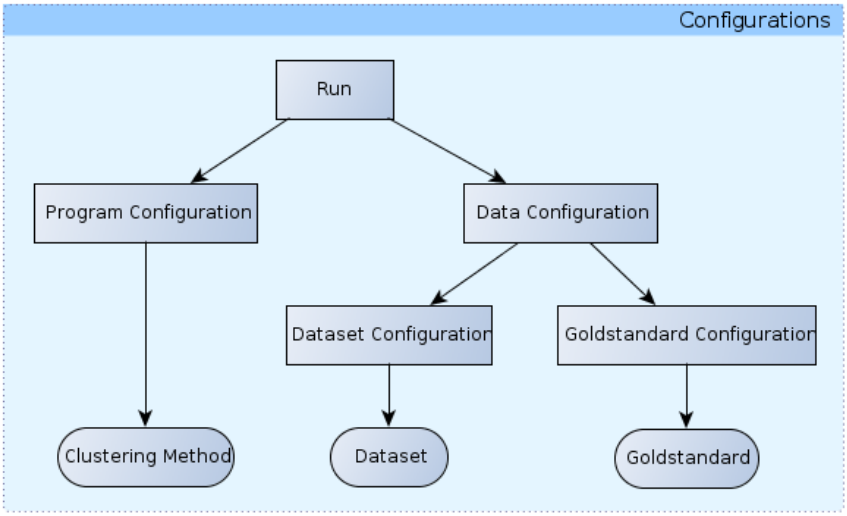
4.9 Configuration Files

Above we already discussed that the framework needs **program configurations**, **dataset configurations** and **goldstandard configurations**. Those configuration files directly reference corresponding files (dataset, goldstandard, ...) on the filesystem. Internally the framework has some abstraction layers to store all the configurations. Figure 5 shows the overall abstractional structure of the configuration files used in the backend. One can see that dataset- and goldstandard configuration are linked together in a **data configuration**.

A run is an abstract entity that can be performed by the backend. Its execution involves (in most cases) application of clustering methods to several datasets, and afterwards clustering qualities are assessed using the goldstandards corresponding to each dataset. A run corresponds to a **run configuration** file, which then again references the **program-** and **data configurations** that should be pairwise combined.

When a run is performed by the backend, the clustering methods wrapped by all referenced program configurations are applied to all datasets indirectly referenced through the data configurations.

Figure 5: Backend Configurations



4.9.1 Data Configurations

A data configuration is a file, that combines two other configurations together: A dataset configuration (see 4.9.3) and a goldstandard configuration (see 4.9.5). Later on when you create a run (and its configuration) and in this run you want to apply two clustering methods to three datasets (together with their goldstandards) you will do so by telling the run configuration the names of the three corresponding data configurations. **Please note:** The data configuration file has to have the file extension .dataconfig, otherwise it will not be recognized by the framework.

datasetConfig: This option has to be set to the name of the dataset configuration (see 4.9.3), *not* the name of the file.

goldstandardConfig: This option has to be set to the name of the goldstandard configuration (see 4.9.5), *not* the name of the file.

4.9.2 Example Data Configuration

A data configuration could like as follows:

```
1  datasetConfig = astral_1
2  goldstandardConfig = astral_1_161
```

4.9.3 Dataset Configuration

A dataset configuration tells the framework meta information about the corresponding dataset. That is: The internal name of the dataset, its filename and its format.

datasetName: This name is used to find and access datasets within the framework. The name of the dataset *has* to be identical to the subfolder of the corresponding dataset.

datasetFile: This option has to be set to the filename of the dataset file residing within the subfolder of the dataset.

datasetFormat: This option tells the framework, which format the dataset is in. **clusteval** ships with a set of supported dataset formats. Please note, that the entries in this list *have* to be identical with the simple names of the corresponding dataset format classes.

[**distanceMeasureAbsoluteToRelative:**] This indicates, which distance measure should be used, when this dataset is converted to another format. Defaults to EuclidianDistanceMeasure.

[**preprocessorAfterDistance:**]A comma seperated list of data preprocessors to apply, before the data is converted to pairwise similarities (the standard input format)

[**preprocessorAfterDistance:**] A comma seperated list of data preprocessors to apply, after the data is converted to pairwise similarities (the standard input format)

4.9.4 Example Dataset Configuration

```
1  datasetName = astral_1_161
2  datasetFile = blastResults.txt
3  datasetFormat = BLASTDataSetFormat
4  distanceMeasureAbsoluteToRelative = EuclidianDistanceMeasure
```

4.9.5 GoldStandard Configuration

A goldstandard configuration tells the framework meta information about the corresponding goldstandard. That is: The internal name of the goldstandard and its filename.

goldstandardName: This name is used to find and access goldstandards within the framework. The name of the goldstandard *has* to be identical to the subfolder of the corresponding goldstandard.

goldstandardFile: This option has to be set to the filename of the goldstandard file residing within the subfolder of the goldstandard.

4.9.6 Example Goldstandard Configuration

```
1  goldstandardName = astral_1_161
2  goldstandardFile = astral_161.goldstandard_3.txt
```

4.9.7 Program Configurations

For every clustering method there can be several configuration files. All program configurations have to be located in `<REPOSITORY_ROOT>/programs/configs`. A program configuration tells the framework, what parameters the program expects, how to invoke the executable, with what parameter values to invoke it and several other information. Possible entries in a program configuration follow.

Please note: The program configuration file has to have the file extension `.config`, otherwise it will not be recognized by the framework.

type: Indicates, whether the program described by this program configuration is a standalone or an R program. This option can be set to either *standalone* or the simple name of the R program class.

program: This is the full name of the clustering method, this configuration references. When a clustering method is located in `<REPOSITORY_ROOT>/programs/some/program`, then the full name of the program is "some/program"

alias: This option is only interpreted for standalone programs. It tells the framework a alias of the corresponding program, which is used whenever the program needs to be represented in a readable format, e.g. on the website.

parameters: A comma-separated list of parameters the program uses and which can be used when the program is invoked. These parameters need to be set to valid values before the program is actually applied to a dataset. Parameter values can be specificely defined either in a program or run configuration or, in case of parameter optimization runs, they are autonomously determined by the framework. If no value is defined for a program parameter at all, it will be set to its default value given in the program configuration.

optimizationParameters: This option is only used, when a run is performed in parameter optimization mode. Then this list of parameters (that needs to be a subset of the list given in option "parameters") is used to determine, which parameters can be in principle optimized when this program is used in parameter optimization mode.

compatibleDataSetFormats: This list tells the framework, which input formats this program supports. Please note, that the entries in this list *have* to be identical to the simple names of the classes of the corresponding dataset formats.

outputFormat: This option tells the framework, what the output format of this program is. Please note, that the naming convention of this list follows the same rules as those of "compatibleDataSetFormats", as the value of this option has to be named exactly after the simple name of the corresponding class.

expectsNormalizedDataSet: This option can be set to "true" or "false". By default, it is set to false. If you set it to true, the input similarities (only) for this program are normalized to values between 0 and 1. This may help you, if a clustering method does not support negative values or values outside certain ranges.

[**invocationFormat**] This is the section containing a set of invocation formats which tell the framework in string format, how to invoke this program. The invocation formats may contain program parameters or certain predefined variables, which are replaced by the framework during runtime. Such variable names are enclosed by % signs. All program parameters defined in the parameters section of the program configuration can be used in the invocation format string. Additionally the following variables are hardcoded for every program and cannot be used for other parameters:

%e% will be replaced by the absolute path of the executable

%i% will be replaced by the absolute path of the input

%o% will be replaced by the absolute path of the output

%gs% will be replaced by the absolute path of the goldstandard

For example, if the option is set like this:

```
invocationFormat = %e% %i% %preference% %o% maxits=%maxits%
convits=%convits%
```

"preference", "maxits" and "convits" have to be defined in the "parameters" entry of the same program configuration.

invocationFormat: This is the option which tells the framework, how to invoke this program in case we have a goldstandard and no parameter optimization run. All words enclosed with % will be replaced by the framework at runtime. All other variables in the invocation line have to be parameters defined in this program configuration.

invocationFormatWithoutGoldStandard: This is how to invoke this program in case we have no goldstandard and no parameter optimization run.

invocationFormatParameterOptimization: This is how to invoke this program in case we have a goldstandard and a parameter optimization run.

invocationFormatParameterOptimizationWithoutGoldStandard: This is how to invoke this program in case we have no goldstandard and a parameter optimization run.

[<**parameterName**>] For every parameter defined in the list of entry "parameters", there needs to be an additional section in the program configuration, which tells the framework several information about the parameter:

desc: A description of the parameter

type: One of the types FLOAT ("2"), INTEGER ("1") or STRING ("0").

def: A default value for the parameter.

minValue: The minimal value for the parameter.

maxValue: The maximal value for the parameter.

4.9.8 Example program configuration

```
1  program = APcluster
2  parameters = preference,maxits,convits,dampfact
3  optimizationParameters = preference,maxits,convits,dampfact
4  executable = apcluster
5  compatibleDataSetFormats = APRowSimDataSetFormat
6  outputFormat = APRunResultFormat
7
8  [invocationFormat]
```

```
9   invocationFormat = %e %i %preference %o maxits=%maxits convits=%convits
10  dampfact=%dampfact
11
12  [maxits]
13  desc = Max iterations
14  type = 1
15  def = 2000
16  minValue = 2000
17  maxValue = 5000
18
19  [convits]
20  desc = Cluster Center duration
21  type = 1
22  def = 200
23  minValue = 200
24  maxValue = 500
25
26  [dampfact]
27  type = 2
28  def = 0.9
29  minValue = 0.7
30  maxValue = 0.99
31
32  [preference]
33  desc = Preference
34  type = 2
35  def = 0.5
36  minValue = 0.0
37  maxValue = 1.0
```

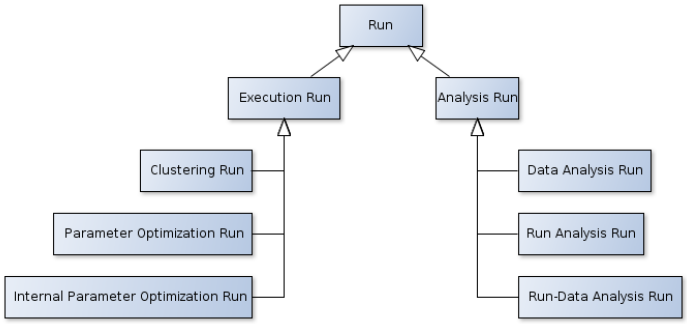
4.10 Runs

Runs are entities that can be performed by the backend server. A run is defined by a file in the folder `<REPOSITORY_ROOT>/runs`. The name of that file (without extension) also defines the name of the run. Depending on the type of the run this file contains several other components which configure the process when the run is performed. Figure 6 shows the different types of runs and how they relate to each other.

4.10.1 Run Files

Every run is defined in a run-file in the corresponding folder of the repository. Depending on the type of the run, different options are available that can be specified in the

Figure 6: Inheritance of different run types



run-file. Common to all types of runs are the following options:

mode: This entry can be set to "clustering", "parameter_optimization", "dataAnalysis", "runAnalysis" or "runDataAnalysis". These types can be found in the aforementioned figure and are described in the following paragraphs.

4.10.2 Execution Runs

Execution runs calculate clusterings during their execution and assess qualities for every of those clusterings. Clusterings are calculated by applying clustering methods to datasets using a certain parameter set. That is why execution runs have sets of both, program and data configurations. During execution time every program configuration is applied to every data configuration in a pairwise manner. For every calculated clustering a set of clustering quality measures are assessed.

In general the options of such a combination of data and program configuration will be taken from these configurations respectively, but can be overridden by the options in the run configuration, That means parameter values defined in the program as well as in the run configuration will be taken from the latter.

4.10.3 Execution Run Files

For execution runs, additionally to the options defined for all runs (see Run Files), the following options for the run-file are defined:

programConfig: This entry has to be set to a single name or a comma-separated list of names of program configurations. When this run is performed, these program configurations will be pairwise combined with the data configurations given in the option "dataConfig".

dataConfig: This entry has to be set to a single name or a comma-separated list of names of data configurations. When this run is performed, these data configurations will be pairwise combined with the program configurations given in the option "programConfig".

qualityMeasures: This option determines, which quality measures will be assessed for every clustering calculated during the run process. When this run is a clustering run (see option "mode"), then for every pair of data and program configurations there will be only one clustering as a result for which quality measures will be evaluated. When the mode is set to "parameter_optimization", for every iteration during the parameter optimization process these quality measures will be evaluated.

[<**programConfigName**> :] If a dedicated section is found in this run file that is called like one of the program configurations given in option "programConfig", several parameters can be overridden individually only for this program configuration which are

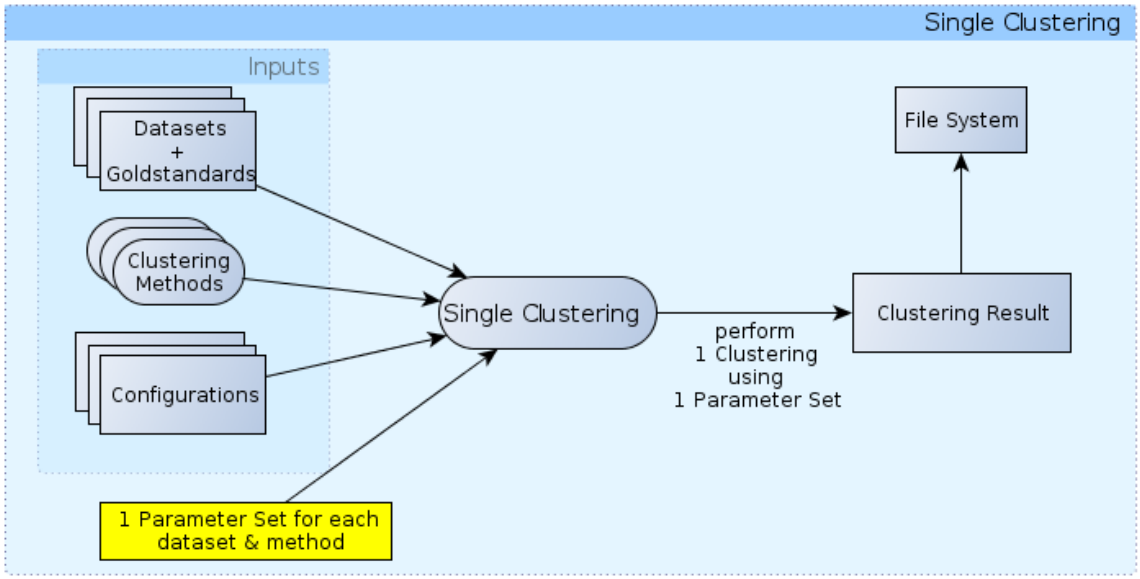
<**parameterName**>**:** The program parameter with the given name which needs to be defined in the program configuration can be fixed to a certain value.

4.10.4 Clustering Runs

Clustering runs are a type of execution run, that means they calculate clusterings by applying every program configuration to every data configuration. Afterwards they assess the qualities of those clusterings in terms of several clustering quality measures.

In the case of clustering runs for every pair of program and data configuration exactly one clustering is calculated and assessed. Clustering runs are visualized in figure 7.

Figure 7: Clustering Run Process



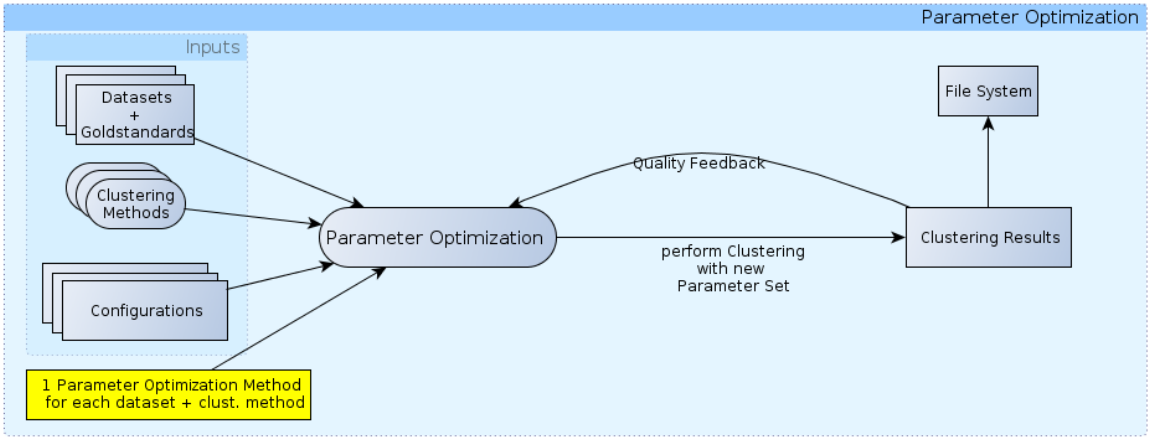
Clustering Run File For clustering runs, the options are the same as for all execution runs (see Execution Run Files).

4.10.5 Parameter Optimization Runs

Parameter optimization runs are a type of execution run, that means they calculate clusterings by applying every program configuration to every data configuration. Afterwards they assess the qualities of those clusterings in terms of several clustering quality measures.

In contrast to clustering runs, parameter optimization runs calculate several clusterings for every pair of data and program configuration in a pairwise manner. Every clustering corresponds to a certain parameter set and the parameter sets to evaluate are determined by a parameter optimization method (see 4.8 for more information). Parameter optimization runs are visualized in figure 8.

Figure 8: Parameter Optimization Run Process



Parameter Optimization Run File For parameter optimization runs, additionally to the options defined for all execution runs (see Execution Run Files), the following options for the run-file are defined:

optimizationMethod: The parameter optimization method to use when this run is performed.

optimizationCriterion: The clustering quality measure which should be used as optimization criterion. This criterion is used to determine the optimal parameter set during the optimization process. Therefore it can influence the cause of the optimization process, if the chosen parameter optimization method integrates the qualities of previous iterations into future iterations.

optimizationIterations: The number of total optimization iterations that should be performed for every pair of program and data configuration. **Hint:** This number might not be the number, the optimization process performs in the end, since it gives only a desirable number that might not be accurately realizable for a specific optimization method.

[<**programConfigName**> :] If a dedicated section is found in this run file that is called like one of the program configurations given in option "programConfig", several parameters can be overridden individually only for this program configuration which are

optimizationParameters: A comma separated list of the parameters that should be optimized for this program configuration

optimizationMethod: The parameter optimization method to use for this program configuration

4.10.6 Analysis Runs

Analysis runs assess certain properties of objects of interest. An analysis run has a set of target objects and a set of statistics, that should be assessed for each of the target objects. That means, during execution time for every target object every statistic is assessed in a pairwise manner.

4.10.7 Data Analysis Runs

In case of data analysis runs the target objects to analyze are data configurations (indirectly datasets) and the statistics are data statistics, that is properties of datasets. Data analysis runs are visualized in figure 9.

4.10.8 Data Analysis Run File

For data analysis runs the following options for the run-file are defined:

dataStatistics: A comma separated list of data statistics to assess for the given data configurations.

dataConfig: A comma separated list of data configurations to analyse.

4.10.9 Run Analysis Runs

In case of run analysis runs the target objects to analyze are clusterings (results of execution runs) and the statistics are run statistics, that is properties of execution run results. Run analysis runs are visualized in figure 10.

Figure 9: Data Analysis Run Process

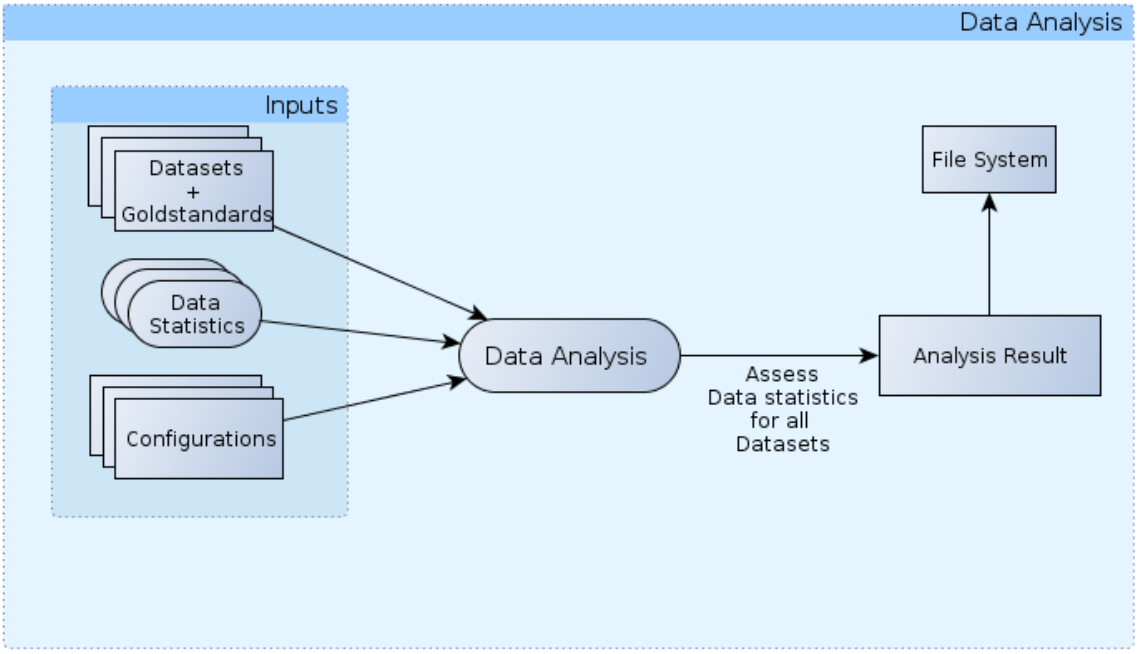
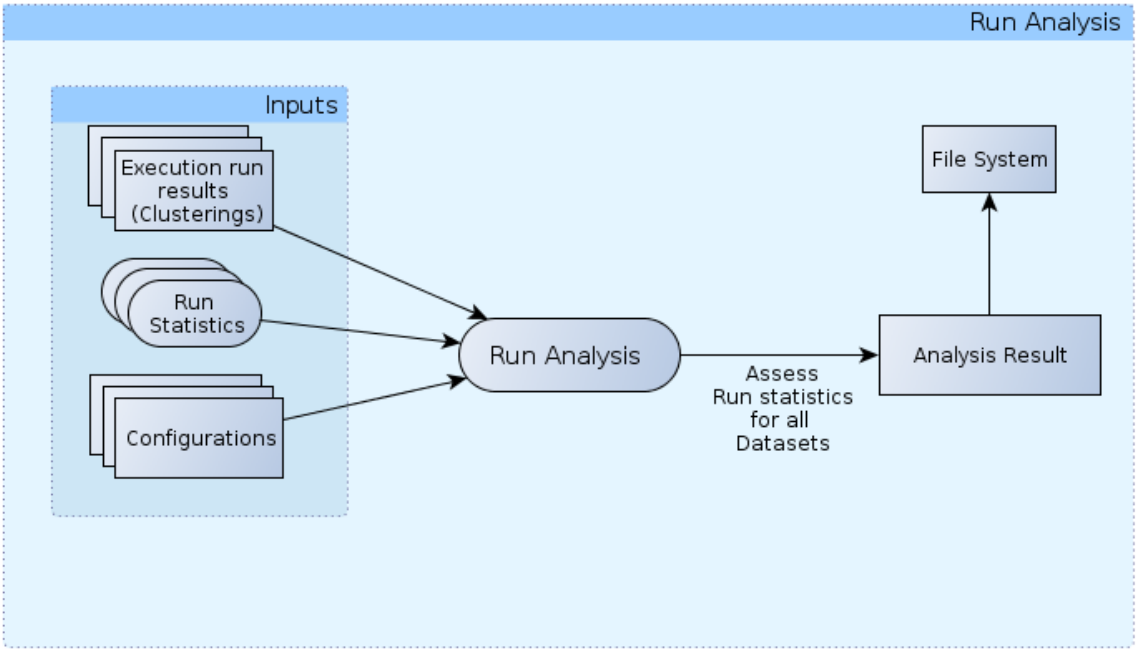


Figure 10: Run Analysis Run Process



4.10.10 Run Analysis Run File

For run analysis runs the following options for the run-file are defined:

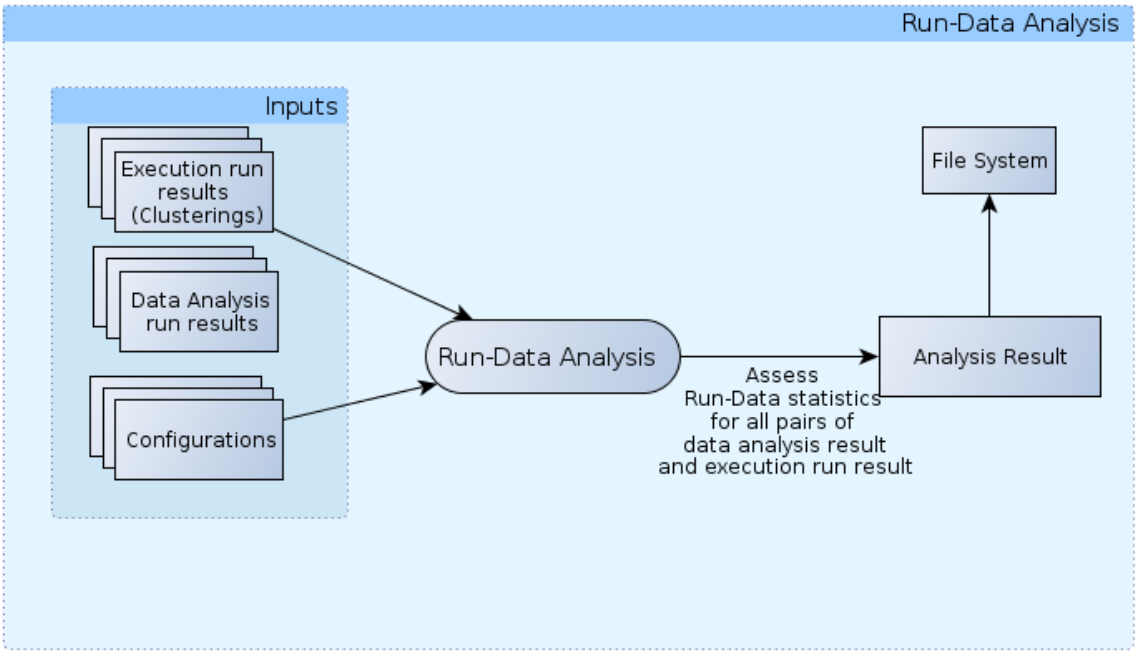
runStatistics: A comma separated list of run statistics to assess for the given execution run results.

uniqeRunIdentifiers: A comma separated list of identifiers of execution run results. See 4.11 for an explanation on run result identifiers.

4.10.11 Run-Data Analysis Runs

In case of run-data analysis runs the target objects to analyze are pairs of data configurations and clusterings (results of execution runs) and the statistics are run-data statistics, that is relationships between execution run results and properties of data configurations. Run-Data analysis runs are visualized in figure 11.

Figure 11: Run-Data Analysis Run Process



4.10.12 Run-Data Analysis Run File

For run-data analysis runs the following options for the run-file are defined:

runDataStatistics: A comma separated list of run-data statistics to assess for every pair of given execution run result and data analysis result.

uniqeRunIdentifiers: A comma separated list of identifiers of execution run results. See 4.11 for an explanation on run result identifiers.

uniqeDataIdentifiers: A comma separated list of identifiers of data analysis run results. See 4.11 for an explanation on run result identifiers.

4.10.13 Examples of a Run Configurations

See 4.10.13 and 4.10.13 for examples of run configuration files.

Figure 12: Example run configuration "all_vs_astral_1_171.runconfig" for Parameter Optimization Mode

```
programConfig = APcluster_1,TransClust_2,MCL_1
dataConfig = astral_1_171
qualityMeasures = TransClustF2ClusteringQualityMeasure,↩
  ↪ SilhouetteValueRClusteringQualityMeasure
mode = parameter_optimization
optimizationMethod = DivisiveParameterOptimizationMethod
optimizationCriterion = ↩
  ↪ SilhouetteValueRClusteringQualityMeasure
optimizationIterations = 1001

[TransClust_2]
optimizationParameters = T

[MCL_1]
optimizationParameters = I

[APcluster_1]
optimizationParameters = preference,dampfact,maxits,↩
  ↪ convits
optimizationMethod = APDivisiveParameterOptimizationMethod
```

Figure 13: Example run configuration "mcl_1.runconfig" for Clustering mode

```
programConfig = MCL_1
dataConfig = DS1,test50

[MCL_1]
I = 2.0
```

4.11 Run Results

When a run is performed, a unique run identifier is determined which includes the start-time and date and the name of the run. If the run `exampleRun` is performed at the 5th of July 2012 at 12:58:38, its unique run identifier is `06_05_2012-12_58_38_exampleRun` which is also used as the subfolder to store its results in
`<REPOSITORY_ROOT>/results/06_05_2012-12_58_38_exampleRun`
Every such folder contains some subfolders. Common to all run types are the following subfolders:

4.11.1 All run result

folders contain the following subfolders:

`<REPOSITORY_ROOT>/results/<runIdentifier>/configs` : Contains the configuration files that are used in this run, which includes all data-, dataset-, goldstandard- and program configurations as well as the run file.

`<REPOSITORY_ROOT>/results/<runIdentifier>/inputs` : Contains backups of all the input files used in this run, which includes all datasets referenced by the data configurations.

`<REPOSITORY_ROOT>/results/<runIdentifier>/goldstandards` : Contains backups of all the goldstandard files used in this run.

`<REPOSITORY_ROOT>/results/<runIdentifier>/logs` : Contains different log filesm, one for the complete run and one for every iteration performed during the run.

Depending on the run type there are additional subfolders:

4.11.2 Execution run result

folders additionally contain the following subfolders:

- `<REPOSITORY_ROOT>/results/<runIdentifier>/clusters`

4.11.3 Analysis run result

folders additionally contain the following subfolders:

- `<REPOSITORY_ROOT>/results/<runIdentifier>/analyses`

5 Backend Client

The backend client is a small command-line tool which gives commands to the backend server, for example to execute a certain run or terminate a run that is currently executing. It offers tab completion and communicates with the server using the Rserve package via TCP/IP.

6 Frontend

The frontend of the framework includes a website and a mysql database, which stores status and results of the backend only for the website. The website in general provides lists of all data and clustering methods available to the framework, as well as tabular and graphical representations of the results.

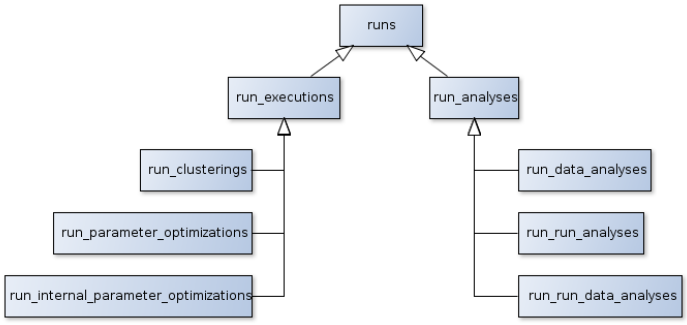
7 Frontend Database

The MySQL database of the frontend stores a subset of the data contained in the repository of the backend. The stored information can then be retrieved and visualized by the website.

7.1 Tables

In the following we will give a short description of every table of the mysql database. Table names in square brackets correspond to rather technical tables, that are just required by the models of the ruby on rails website and do not have a strong meaning with regard to contents.

Figure 14: Inheritance of different run types (database hierarchy)



Hint 1 : All tables that correspond to and are responsible for storing repository objects have a foreign key to the repository table. Thus for every repository object it is known, to which repository it belongs. This is not mentioned in the following descriptions.

Hint 2 : The abbreviation FK means Foreign Key.

Hint 3 : Column names are denoted in italic.

Hint 4 : When a run is performed, certain files are copied into a new result folder. This includes datasets, goldstandards and all configuration files. These files in the result folder are mapped to their original files they correspond to in the original repository. These relationships are stored in the database in a separate column which is named after the table name plus the postfix "_id". For example datasets store this relationship in the column "dataset_id".

7.1.1 [aboutus]

A technical table containing the information regarding the 'About us' section of the website.

7.1.2 [admins]

A technical table containing the information regarding the 'Admin' section of the website.

7.1.3 cluster_objects

Every clustering contains clusters, which again contain cluster objects. This table stores all cluster objects with their *name* and knows, to which cluster they belong (*cluster_id*, FK).

7.1.4 clustering_quality_measure_optimums

For visualization and interpretation of results the website needs to know, whether a certain clustering quality measure is optimal when min- or maximized. This table maps every measure to 'Minimum' or 'Maximum' (*name*).

7.1.5 clustering_quality_measures

This table keeps track of all the clustering quality measures available in the framework. For every measure it stores its *name*, minimal and maximal value (*minValue* and *maxValue*) and whether the measure requires a goldstandard (*requiresGoldStandard*). On the website every clustering quality measure has a readable alias. This alias is stored in the column (*alias*).

7.1.6 clusterings

This table holds all the clusterings that were calculated and are stored in the repository. Every parsed clustering corresponds to a file in the repository, for which we store its absolute path (*absPath*).

7.1.7 clusters

Every clustering has clusters. This table holds all clusters and maps them to their corresponding clustering. Every cluster has a *name*.

7.1.8 data_configs

A table holding all data configurations. Every data configuration has an absolute path (*absPath*), a *name*, a corresponding dataset configuration (*dataset_config_id*, FK) and a goldstandard configuration (*goldstandard_config_id*, FK).

7.1.9 dataset_configs

A table holding all dataset configurations. Every dataset configuration has an absolute path (*absPath*), a *name* and a corresponding dataset (*dataset_id*, FK).

7.1.10 dataset_formats

A table holding all dataset formats. Every format has a *name* and an *alias*.

7.1.11 dataset_types

This table holds all types of datasets. For every dataset type it stores a *name*. On the website every dataset type has a readable alias. For every dataset type this table stores the *alias*.

7.1.12 datasets

This table holds all datasets. For every dataset its absolute path (*absPath*), *checksum*, format (*dataset_format_id*, FK) and type (*dataset_type_id*, FK) is stored.

7.1.13 goldstandard_configs

This table holds all goldstandard configurations. For every goldstandard configuration its absolute path (*absPath*), *name* and corresponding goldstandard (*goldstandard_id*, FK) is stored.

7.1.14 goldstandards

This table holds all goldstandards. For every goldstandard its absolute path (*absPath*), and the corresponding goldstandard (*goldstandard_id*, FK) is stored.

7.1.15 [helps]

A technical table containing the information regarding the 'Help' section of the website.

7.1.16 [mains]

A technical table containing the information regarding the startpage of the website.

7.1.17 optimizable_program_parameters

This table stores the program parameters (*program_parameter_id*, FK) of a program configuration (*program_config_id*, FK), that can be optimized.

7.1.18 parameter_optimization_methods

This table stores all available parameter optimization methods (*name*) registered in a repository (*repository_id*, FK).

7.1.19 program_configs

All program configurations registered in a repository are stored in this table. Every program configuration has a *name*, an absolute path (*absPath*), different invocation formats for different scenarios (*invocationFormat*, *invocationFormatWithoutGoldStandard*, *invocationFormatParameterOptimization*, *invocationFormatParameterOptimizationWithoutGoldStandard*) and a boolean whether the program expects input with normalized similiarites (*expectsNormalizedDataSet*). For every program configuration we store the corresponding repository (*repository_id*, FK), the program (*program_id*, FK) this configuration belongs to, the run result format (*run_result_format_id*, FK) of the program using this configuration.

7.1.20 program_configs_compatible_dataset_formats

Every program configuration (*program_config_id*, FK) has a set of compatible dataset formats (*dataset_format_id*, FK), which the program will understand when it is executed using this configuration.

7.1.21 [program_descriptions]

This table stores descriptions of clustering methods, for when they are shown on the website.

7.1.22 [program_images]

When this table contains an image for a clustering method, it will be shown on the website.

7.1.23 program_parameter_types

This table contains the names (*name*) of the different possible program parameter types (see 4.9.7 for more information on the types of parameters supported by **clusteval**).

7.1.24 program_parameters

This table stores the program parameters defined in a program configuration (*program_config_id*, FK). Every program parameter has a type (*program_parameter_type_id*, FK), a *name*, an (optional) *description*, a *minValue*, a *maxValue* and a default value (*def*).

7.1.25 [program_publications]

When this table contains publication information for a clustering method, it will be shown on the website.

7.1.26 programs

This tables stores all clustering methods together with their *absPath* and an *alias*, which is used to represent this clustering method on the website.

7.1.27 repositories

The repositories that are using this database to store their results. Every repository has a absolute base directory (*basePath*) and a type (*repository_type_id*, FK).

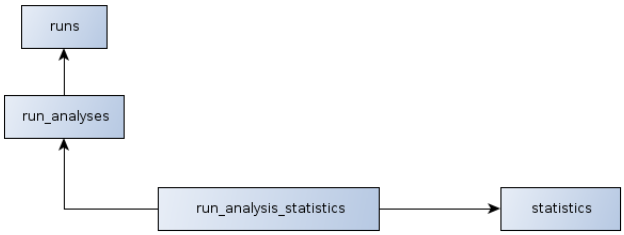
7.1.28 repository_types

The types that repositories can have. Every type has a *name*. Check out 4.1 for more information on which repository types exist.

7.1.29 run_analyses

This table holds all analysis runs. Every analysis run is also a run (*run_id*, FK).

Figure 15: Foreign keys of analysis runs



7.1.30 run_analysis_statistics

Every analysis run (*run_analysis_id*, FK) evaluates certain statistics (*statistic_id*, FK).

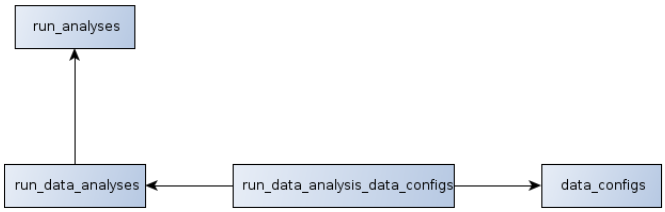
7.1.31 run_clusterings

This table holds all clustering runs. Every clustering run is also a execution run (*run_execution_id*, FK).

7.1.32 run_data_analyses

This table holds all data analysis runs. Every data analysis run is also an analysis run (*run_analysis_id*, FK).

Figure 16: Foreign keys of data analysis runs



7.1.33 run_data_analysis_data_configs

Every data analysis run analysis a set of data configurations wrapping datasets. This table holds the data configurations (*data_config_id*, FK) that a certain data analysis run (run_data_analysis_id, FK) analyses.

7.1.34 run_execution_data_configs

An execution run applies program configurations to data configurations. This table stores the data configurations (*data_config_id*, FK) belonging to execution runs (*run_execution_id*, FK).

7.1.35 run_execution_parameter_values

An execution run can specify values for program parameters. This table stores for every execution run (*run_execution_id*, FK), program configuration (*program_config_id*, FK) and program parameter (*program_parameter_id*, FK) the specified *value*.

7.1.36 run_execution_program_configs

An execution run applies program configurations to data configurations. This table stores the program configurations (*program_config_id*, FK) belonging to execution runs (*run_execution_id*, FK).

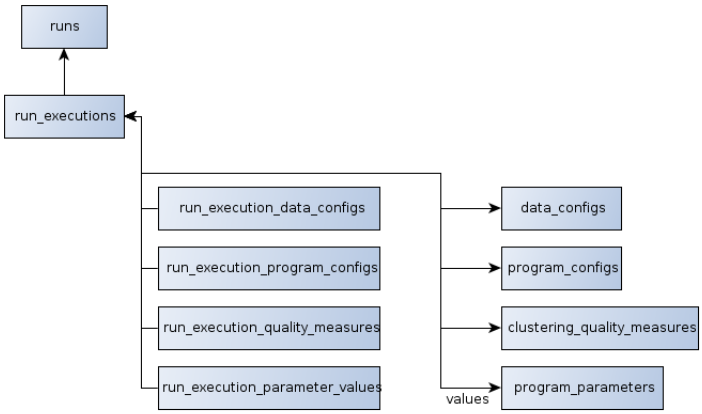
7.1.37 run_execution_quality_measures

An execution run applies clustering methods to datasets and assesses clustering quality measures. This table stores the execution run (run_execution_id, FK) together with the clustering quality measures (*clustering_quality_measure_id*, FK) to assess.

7.1.38 run_executions

This table holds all execution runs. Every execution run is also a run (*run_id*, FK).

Figure 17: Foreign keys of execution runs



7.1.39 run_internal_parameter_optimizations

This table holds all internal parameter optimization runs. Every such run is also an execution run (*run_execution_id*, FK).

7.1.40 run_parameter_optimization_methods

A parameter optimization run uses parameter optimization methods to optimize parameters. For a certain parameter optimization run (*run_parameter_optimization_id*, FK) for every program configuration (*program_config_id*, FK) a different parameter optimization method (*parameter_optimization_method_id*, FK) and a clustering quality measure to optimize can be specified.

7.1.41 run_parameter_optimization_parameters

A parameter optimization run optimizes parameters of clustering methods. This table stores for a certain run (*run_parameter_optimization_id*, FK) for every program configuration contained (*program_config_id*, FK) the parameters (*program_parameter_id*, FK) to optimize.

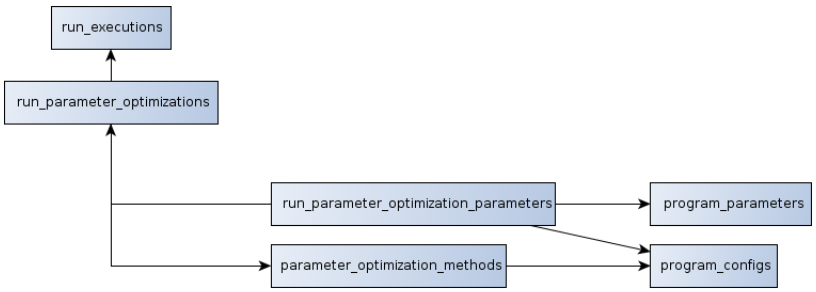
7.1.42 run_parameter_optimization_quality_measures

A parameter optimization run optimizes parameters by maximizing or minimizing clustering quality measures. This table stores all clustering quality measures (*clustering_quality_measure_id*, FK) to assess for the calculated clusterings.

7.1.43 run_parameter_optimizations

This table holds all parameter optimization runs. Every parameter optimization run is also an execution run (*run_execution_id*, FK).

Figure 18: Foreign keys of parameter optimization runs



7.1.44 run_result_data_analysis_data_configs_statistics

A data analysis run assesses statistics for certain data configurations. This table stores the assessed statistics (*statistic_id*, FK) for every run result (*run_result_id*, FK) generated by a

7.1.45 run_result_formats

This table holds all run result formats. Every run result format has a *name*.

7.1.46 run_results

When a run is executed, it produces a unique folder in the results directory of the repository. These folders are stored in this table together with the type of the corresponding run (*run_id*, FK) that created this result (*run_type_id*, FK), an absolute path to the results folder (*absPath*), the *uniqueRunIdentifier* of this run result (which corresponds to the name of the folder) and the *date* the run result was created.

7.1.47 run_results_analyses

When an analysis run is executed, it produces a unique folder in the results directory of the repository. Every analysis run result is also a run result (*run_result_id*, FK).

7.1.48 run_results_clusterings

When a clustering run is executed, it produces a unique folder in the results directory of the repository. Every clustering run result is also an execution run result (*run_results_execution_id*, FK).

7.1.49 run_results_data_analyses

When a data analysis run is executed, it produces a unique folder in the results directory of the repository. Every data analysis run result is also an analysis run result (*run_results_analysis_id*, FK).

7.1.50 run_results_executions

When an execution run is executed, it produces a unique folder in the results directory of the repository. Every execution run result is also a run result (*run_result_id*, FK).

7.1.51 run_results_internal_parameter_optimizations

When an internal parameter optimization run is executed, it produces a unique folder in the results directory of the repository. Every internal parameter optimization run result is also an execution run result (*run_results_execution_id*, FK).

7.1.52 run_results_parameter_optimizations

When a parameter optimization run is executed, it produces a set of iterative run results, that are all summarized in .complete-files for every pair of program and data configuration (*program_config_id*, FK) and (*data_config_id*, FK). Every parameter optimization run result is also an execution run result (*run_results_execution_id*, FK).

7.1.53 run_results_parameter_optimizations_parameter_set_iterations

Every parameter optimization produces clustering results for a set of iterations. In each iteration a different parameter set (*run_results_parameter_optimizations_parameter_set_id*, FK) is evaluated. This table holds the number of the *iteration*, together with the parameter set, the produced clustering (*clustering_id*, FK) and the parameter set in a string representation (*paramSetAsString*).

7.1.54 run_results_parameter_optimizations_parameter_set_parameters

This table holds the program parameters (*program_parameter_id*, FK) that belong to parameter sets (*run_results_parameter_optimizations_parameter_set_id*, FK) contained in a parameter optimization run result (*run_results_parameter_optimization_id*, FK), evaluated by the framework.

7.1.55 run_results_parameter_optimizations_parameter_sets

This table holds the parameter sets evaluated during a parameter optimization run and contained in a parameter optimization run result (*run_results_parameter_optimization_id*, FK).

7.1.56 run_results_parameter_optimizations_parameter_values

This table contains the *value* of a certain program parameter (*run_results_parameter_optimizations_param* FK) evaluated in a certain parameter optimization iteration (*run_results_parameter_optimizations_parame* FK).

7.1.57 run_results_parameter_optimizations_qualities

In every iteration of a parameter optimization a parameter set is evaluated and clustering qualities are assessed. This table holds the clustering quality measure (*cluster_quality_measure_id*, FK) together with the assessed *quality*.

7.1.58 run_results_run_analyses

When a run analysis run is executed, it produces a unique folder in the results directory of the repository. Every run analysis run result is also an analysis run result (*run_results_analysis_id*, FK).

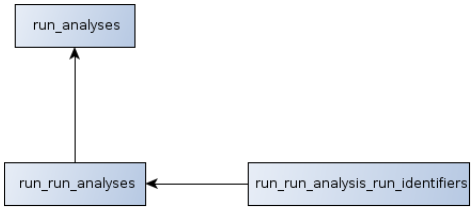
7.1.59 run_results_run_data_analyses

When a run-data analysis run is executed, it produces a unique folder in the results directory of the repository. Every run-data analysis run result is also an analysis run result (*run_results_analysis_id*, FK).

7.1.60 run_run_analyses

This table holds all run analysis runs. Every run analysis run is also an analysis run (*run_analysis_id*, FK).

Figure 19: Foreign keys of run analysis runs



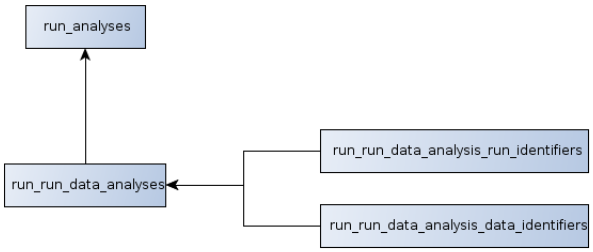
7.1.61 run_run_analysis_run_identifiers

Every run analysis run analyses (*run_run_analysis_id*, FK) a set of run results with certain identifiers (*runIdentifier*).

7.1.62 run_run_data_analyses

This table holds all run-data analysis runs. Every run-data analysis run is also an analysis run (*run_analysis_id*, FK).

Figure 20: Foreign keys of run-data analysis runs



7.1.63 run_run_data_analysis_data_identifiers

Every run-data analysis run analyzes (*run_run_analysis_id*, FK) a set of data analysis run results with certain identifiers (*dataIdentifier*).

7.1.64 run_run_data_analysis_run_identifiers

Every run-data analysis run analyzes (*run_run_analysis_id*, FK) a set of execution run results with certain identifiers (*runIdentifier*).

7.1.65 run_types

This table holds all types of runs. Every type has a *name*. Check out 6 for more information on which run types exist.

7.1.66 runs

This table holds all runs. Every run has a type (*run_type_id*, FK), an absolute path (*absPath*), a *name* and a *status*.

7.1.67 [schema_migrations]

This table holds all the migrations of the ruby on rails website.

7.1.68 statistics

This table holds all the statistics registered in a repository. Every statistic has a *name* and an *alias*. The alias is used on the website as a readable name.

7.1.69 statistics_datas

This table holds all the data statistics. Every data statistic is also a statistic (*statistic_id*, FK).

7.1.70 statistics_runs

This table holds all the run statistics. Every run statistic is also a statistic (*statistic_id*, FK).

7.1.71 statistics_run_datas

This table holds all the run-data statistics. Every run-data statistic is also a statistic (*statistic_id*, FK).

7.1.72 [submit_datasets]

This table corresponds to the section "Submit Dataset" of the website.

7.1.73 [submit_methods]

This table corresponds to the section "Submit Clustering Method" of the website.

7.1.74 [submits]

This table corresponds to the "Submit" section of the website.

7.1.75 [users]

This table holds all the users, that have registered on the website.

8 Frontend Website

The frontend website is designed to give a visual representation of the results calculated by the backend and also of the datasets, goldstandards, configurations available in the repository. The website is divided into several sections which will be discussed in 8.1. Here it is described, which information you can find in which section. Afterwards it follows a short explanation for every task you can realize using the website, including how to compare available clustering methods or datasets and how to interprete visualized run results.

8.1 Navigation

The website is divided into the following sections:

8.1.1 Welcome

This page welcomes you to the website. It provides you a list of available repositories, from which you can choose. This influences what contents are shown in all other sections. After choosing a repository, only data and results are shown from this repository.

8.1.2 Overview

This section presents you with a comparison matrix, containing the best achieved clustering qualities for every pair of clustering method and dataset. Above the matrix you can select from a list, which clustering quality measure you want to compare. Values in **bold** indicate the optimal clustering quality in the respective row. By hitting the **Invert matrix** button, you can interchange datasets with clustering methods, rows with columns.

8.1.3 Clustering Methods

On the left hand side you have different subsections to navigate to. Now you are at the "Overview" page.

Overview shows you a list of all available clustering methods. For the clustering methods that ship with **clusteval** there will be shown a short description and a literature reference for more detailed information.

Clicking on "Details" of a specific clustering method, will lead you to information only about that method.

General provides you with general information and a download link of the executable.

Performance shows a comparison table, how this clustering method performs on different datasets measured by different clustering quality measures.

Best Clusterings is comparable to the "Performance" subsection, but also shows the parameter sets, that lead to the optimal clusterings.

Comparison will show you a table containing the optimal clustering qualities a chosen clustering method achieved on different datasets together with the parameter sets used.

8.1.4 Datasets

On the left hand side you have different subsections to navigate to. Now you are at the "Overview" page.

Overview shows you a list of all available datasets. The datasets are grouped together, based on their type.

By clicking on a dataset, you will be lead to information only about that dataset.

General shows general information about the dataset including the first 10 lines of the raw file and a download link.

Statistics shows data statistics assessed for this dataset.

Comparison shows a comparison table, how different clustering methods perform on this dataset, measured by different clustering quality measures.

Best Clusterings is comparable to the "Comparison" subsection, but also shows the parameter sets, that lead to the optimal clusterings.

Clusterings shows all qualities of clusterings measured with a certain clustering quality measure that were calculated for this dataset together with the corresponding parameter sets.

Comparison will show you a table containing the optimal clustering qualities clustering methods achieved on a chosen dataset together with the parameter sets used.

8.1.5 Measures

This section shows you the available clustering quality measures.

By clicking on a specific clustering quality measure you will get more information about it, including a formal definition.

8.1.6 Submit

This section allows you to contribute your own clustering methods or datasets to **clusteval** .

General shows a short text, what you can do within this section.

Submit dataset presents you an email form to send an email to the **clusteval** server to request an addition of your new dataset.

Submit method presents you an email form to send and email to the **clusteval** server to request an addition of your new clustering method.

8.1.7 Admin

The admin section is only available after logging in into the website. This section shows more detailed information about the single runs that were performed, which run results were produced and all other data that is stored within the repository including configuration files.

Clustering Method Configurations shows a table with all program configurations. By clicking on a specific program configuration, you will see detailed information only about that configuration including e.g. the contents of the file or defined program parameters.

Additionally you see a second table containing the clusterings that were calculated for this program configuration.

Data Configurations shows a table with all data configurations. By clicking on a specific data configuration, you will see detailed information only about that configuration including e.g. the contents of the file or defined program parameters.

Additionally you see a second table containing the clusterings that were calculated for this data configuration.

Dataset Configurations shows a table with all dataset configurations. By clicking on a specific dataset configuration, you will see detailed information only about that configuration.

Goldstandard Configurations shows a table with all goldstandard configurations. By clicking on a specific goldstandard configuration, you will see detailed information only about that configuration.

Runs shows a table with all runs. By clicking on a specific run, you will see detailed information. Depending on the run type these information will vary.

For **execution runs** the used program and data configurations as well as the assessed clustering quality measures will be shown, for **analysis runs** the statistics to assess and the target objects (either data configurations, execution run results or both data analysis run results and execution run results together). For parameter optimization runs the used parameter optimization methods and optimization parameters will be listed.

Results shows the run results calculated by the backend. Depending on the type of the run that corresponds to these run results, the representation varies.

Clustering Runs: Here only the program and data configurations together with the clustering quality measures to assess are shown.

Parameter Optimization Runs: Additionally to the details of clustering runs here also the used parameter optimization methods are listed together with the optimization parameters for each program configuration.

Data Analysis Runs: The data configurations to assess and the data statistics are shown.

Run Analysis Runs: The unique execution run result identifier are listed together with the run statistics to assess.

Run-Data Analysis Runs: The unique run result identifier of the execution run results and of the data analysis run results are shown together with the Run-Data Statistics to assess for pairs of these run results.

8.1.8 Help

Gives you some hints and guides how to use the frontend website.

8.1.9 About us

Contains information about the developer of **clusteval** .

8.2 Clustering Run Results

A table is shown, with every pair of data and program configuration together with the evaluated parameter set and achieved clustering quality.

8.3 Parameter Optimization Run Results

For every pair of program and data configuration a graph and a table is shown. The graph contains the clustering qualities achieved in every iteration. It is good for visual inspection of the general optimization course. The table contains the same information, but in a textual format. It contains for every performed iteration of the parameter optimization process the evaluated parameter set, the assessed clustering quality measures together with the qualities of the resulting clusterings.

8.4 Data Analysis Run Results

For every data configuration you see a table containing rows for every assessed data statistic. On the left the name of the data statistic is shown and on the right you see either a plot or a string representation of the calculated data statistic.

8.5 Run Analysis Run Results

For every execution run result you see a table containing rows for every assessed run statistic. On the left the name of the run statistic is shown and on the right you see either a plot or a string representation of the calculated run statistic.

8.6 Run-Data Analysis Run Results

For every pair of execution run result and data analysis run result you see a table containing rows for every assessed run-data statistic. On the left the name of the run-data statistic is shown and on the right you see either a plot or a string representation of the calculated run-data statistic.

9 Download & Installation

There are basically two options how to get a working installation of **clusteval** . The easy way is to download our VirtualBox Image and start the virtual machine. The image contains both, the website and the backend server- and client executables. This procedure is explained in 9.1. The second option is to install the framework manually step-by-step. The requirements of the backend and its installation are explained in 9.3 and 9.4. respectively. The requirements of the frontend and its installation are detailed in 9.5 and 9.6 .

9.1 VirtualBox Image

The first option how to get a running **clusteval** is to download the VirtualBox from our server, start the machine and have everything installed right from the start. You can then access the frontend website in the webbrowser, start the backend server and client and produce your first clustering results. You do not have to install any unmet requirements or do any configurations. Here is how:

1. Install VirtualBox
2. Download the VirtualBox disk image from our server and start the machine
3. The backend server can be found under

   `/home/clusteval/clustevalBackendServer.jar`
4. The backend client is located under

   `/home/clusteval/clustevalBackendClient.jar`
5. To access the frontend website, open up the preinstalled webbrowser and open the address

   `http://localhost/`

9.2 Manual Installation

To get **clusteval** running on your own server, you have to install it manually. The following sections will show you, what requirements have to be met for each component of **clusteval** (backend server, client and frontend website, database) and how these components can be installed.

9.3 Backend Requirements

The backend (including server and client component) uses several java libraries that already ship within the corresponding binaries (jar-files). They have to be available within the class path of **clusteval** when it is started.

Additionally many functions of the backend server require a working R installation (version ≥ 2.15) together with an installed Rserve library within R. In principle the clustering processes of **clusteval** can be started **without** a running **Rserve**, but you will not be able to use any functions depending on R. Throughout this manual, functionalities which require R will be labeled with $^{(Rserve)}$.

9.3.1 Java $\geq$ 1.6

clusteval is completely written in Java and requires a JRE ≥ 1.6 on the target system.

9.3.2 Java libraries

The following Java libraries have to be accessible within the classpath when **clusteval** is started. The versions in brackets are the versions, that are by default shipped with the framework. **clusteval** has been tested together with these versions, so we cannot guarantee functionality, when different versions are used.

Apache Commons CLI (1.2) is a library for parsing command line arguments and printing CLI usage

Apache Commons Configuration (1.8) provides classes and parsers for configuration files

Apache Commons IO (2.3) provides file operations like copying and moving

Apache Commons Lang (2.6 & 3.1) Base library of apache commons

Apache Commons Logging (1.1.1) is a library used by other apache commons libraries for logging

JAnsi (1.9) provides ANSI codes on the console output

JLine (2.8 Snapshot) a library for command line tab completion

JUnit provides classes for testing and validation

JUnit Addons (1.4) an extension library of JUnit

log4j (1.2.16) an logging API for Java

Logback (1.0.9) a logging library based on log4j

Parallel Colt (0.9.4) a library for fast and memory efficient methods and data structures is used by **clusteval** to store data matrices of datasets

MySQL Connector Java (5.1.21) enables Java to communicate with MySQL databases

Rserve (1.7) libraries providing possibilities to use the R framework from within Java classes

TransClust (1.0) a clustering method, is used as a library to parse and convert FASTA and BLAST files

Wiutils (1.0) a java library providing several general-purpose functionalities and classes, mostly inspired by R convenience functions

9.3.3 [R installation ($\geq$ 2.15)]

R-depending components introduced into **clusteval** can make use of functions of arbitrary R packages. R is in principle optional, but then those functions will not be available. Only subclasses of the following classes are by design intended to make use of R:

- `de.clusteval.cluster.quality.ClusteringQualityMeasure`
- `de.clusteval.data.distance.DistanceMeasure`
- `de.clusteval.data.generator.DataSetGenerator`
- `de.clusteval.program.r.RProgram`
- `de.clusteval.cluster.paramOptimization.ParameterOptimizationMethod`
- `de.clusteval.data.statistics.DataStatistic`
- `de.clusteval.run.statistics.RunStatistic`
- `de.clusteval.run.statistics.RunDataStatistic`

Subclasses of these classes need to tell the framework which R packages they require. The corresponding class is only loaded, if these R packages are available.

R is available for Unix, MacOS and Windows systems and can be downloaded from `http://cran.r-project.org/`. The Rserve package requires a R version $\geq$ 2.15. Some systems (e.g. Debian 6) do not have these R versions in their default repositories. In these cases you can download the binaries from the official website and install them manually. If you are on a Debian system, you can add the CRAN repositories to your

aptitude package system and install it via console. For Debian 6 this can be achieved by doing the following steps:

1. Append the aptitude repository to your sources.list

   e.g. 'deb http://cran.cnr.berkeley.edu/bin/linux/debian squeeze-cran/'

2. Update your local aptitude package information

   `> apt-get update`

   - If the update fails with an error, that you have not authenticated the new server, you have to import the public key of the server with the following command

     `> gpg --keyserver subkeys.pgp.net --recv-keys <serverId>`

     In case of the above example CRAN repository server:

     `<serverId> = 06F90DE5381BA480`

   - Then you will have to add the imported public key of the server to the authorized server keys of aptitude

     `> gpg -a --export <serverId> | sudo apt-key add -`

   - Rerun `> apt get update`

3. Install the R framework by executing

   `> apt-get install r-base`

If you are not familiar with R but are interested in some more information about it, you can also find good documentations on the aforementioned website.

Rserve package Rserve acts as a distribution server to make R available via TCP/IP. Different interfaces are provided for Rserve, such that R can be used from within other programs written in different programming languages, for example Java. This functionality is exploited by **clusteval** in that statistical calculations are outsourced to existing R implementations instead of implementing them from scratch in Java.

This is used in several components of **clusteval** , which are listed below. In principle the clustering processes of **clusteval** can be started without R or a running Rserve instance, but you will not be able to use any functions depending on R.

Installation To install the Rserve package in your R installation, open your R console. On Unix systems you type R in your console and a R terminal will open up:

`> R`

Now you can install the Rserve package using the following command:

`> install.packages("Rserve")`

After you have installed the Rserve package in R, you have to start an instance of Rserve, which will listen on a specific port for incoming calls.

`> Rserve()`

In our case these calls will origin from the backend server, which connects to Rserve. This happens without further intervention by the user.

R libraries Several components that ship with **clusteval** require certain R packages. If you want to use the depending components, you will have to install these packages using the `> install.packages("<packageName>")` command.

"cluster" is required by

- GapStatisticParameterOptimizationMethod
- SilhouetteValueRClusteringQualityMeasure

"fields" is required by

- HopkinsDataStatistic

"Hmisc" is required by

- HoeffdingDRDistanceMeasure
- PearsonCorrelationRDistanceMeasure
- SpearmanCorrelationRDistanceMeasure

"igraph" is required by

- GraphMinCutRDataStatistic
- GraphDiversityAverageRDataStatistic
- GraphDensityRDataStatistic
- GraphCohesionRDataStatistic
- GraphAdhesionRDataStatistic
- ClusteringCoefficientRDataStatistic
- ClusteringCoefficientRDataStatistic

"kernlab" is required by

- SpectralClusteringRProgram

"lattice" is required by

- CooccurrenceBestRunStatistic
- CooccurrenceRunStatistic

"MASS" is required by

- LinearModelRidgeRunDataStatistic

"mlbench" is required by

- CassiniDataSetGenerator
- CircleDataSetGenerator
- CuboidDataSetGenerator
- Gaussian2DDataSetGenerator
- HyperCubeCornersDataSetGenerator
- RingnormDataSetGenerator
- SimplexCornersDataSetGenerator
- SpiralsDataSetGenerator

"stats" is required by

- HierarchicalClusteringRProgram
- KMeansClusteringRProgram

9.4 Backend Installation

In the following we provide a step by step installation manual for Debian 6 servers. We expect the aforementioned requirements to be installed and fulfilled including a Java and R installation.

- Download server and client executables from here and here.

- Execute backend server by typing

  ```
  > java -jar clustevalServer.jar
  ```

  This will start the backend server with default options, listening on port 1099 for clients and using the folder repository in the current directory as repository root. The repository root is recreated, if it does not exist. If it does exist, everything will be loaded and parsed only from this directory (see 4.1 for more information on Repositories).

  For help on available command line parameters you can check out 10.1.1 or invoke the backend server with the help parameter:

  ```
  > java -jar clustevalServer.jar -help
  ```

9.5 Frontend Requirements

The frontend requires a MySQL installation together with a database, that has all necessary tables in it. The backend server has to be configured to use this database and fill it with all relevant data contained in the backend repository. The website itself is based on Ruby on Rails and therefore requires an installation of Ruby on Rails.

Throughout development and for testing as well as validation we used MySQL 5.1 and RoR 1.9.1. For other versions we cannot guarantee the website behaving correctly.

9.5.1 MySQL (5.1)

If the **clusteval** frontend website is to be used, the MySQL database is required to store and retrieve all the data visualized on the website. This data in turn is inserted into the database by the backend server. To install MySQL under Debian 6, you can type

```
> sudo apt-get install mysql-server
```

During the installation procedure this will ask you for a root password, and the credentials of a non-root user. We assume these to be `clustEvalRead`

We can only ensure full compatibility of **clusteval** with the version of MySQL that is indicated in brackets.

Database & User setup The website will require a database and a user with read-rights for that database. It is **highly discouraged** to use the **root user** or any other user with write-rights for this purpose. In case the website contains potential security holes, this can lead to misuse and damage to the MySQL database. If the website uses a user with read-rights, this cannot happen. To create a mysql database, open up a MySQL terminal by typing

```
> mysql -u <YOUR_USERNAME> -p
```

this will ask you for your password. To create a database, perform the following command

```
mysql> CREATE DATABASE <DB_NAME>;
```

Now we create the user which will be used by the website to access the database.

```
mysql> CREATE USER 'clustEvalRead'@'localhost' IDENTIFIED BY
'<YOUR_PASSWORD>';
```

We grant our user the right, to select from the database and to show views.

```
mysql> GRANT SELECT ON <DB_NAME>.* TO clustEvalRead;
mysql> GRANT SHOW VIEW ON <DB_NAME>.* TO clustEvalRead;
```

9.5.2 Ruby on Rails (1.9.1)

The frontend website is developed in Ruby on Rails. Therefore the website requires several components on the target machine:

- Ruby, the programming language
- a webserver compatible to Ruby on Rails

In this guide we will be using Apache with the passenger extension as webserver and guide you through the installation process on a Debian 6 server. The passenger extension adds Ruby on Rails support to Apache webservers.

We can only ensure full compatibility of **clusteval** with the version in brackets. In the following steps we will denote the version of ruby as `<RUBY_VERSION>`.

1. To install Ruby, use the command:

   ```
   > apt-get install ruby<RUBY_VERSION>-full
   ```

   This will install ruby, all of its dependencies and developmental libraries of the chosen ruby version. We create some symlinks such that the ruby executables are available without providing the version number. If you do not create these symlinks, this can cause problems during the installation process.

   ```
   > ln -s /usr/bin/ruby<RUBY_VERSION> /usr/bin/ruby
   ```

   ```
   > ln -s /usr/bin/gem<RUBY_VERSION> /usr/bin/gem
   ```

   ```
   > ln -s /var/lib/gems/<RUBY_VERSION>/bin/bundle /usr/bin/bundle
   ```

   You can verify, that the creation was successful, by using the `which` command and the `-v` command line switch of the corresponding commands.

   We install the bundler gem, which will be needed later on to install gem dependencies of our website.

   ```
   > gem install bundler
   ```

2. To install Apache:

   ```
   > apt-get install apache2
   ```

   This will install the apache webserver with a default setup. We will modify this later on by adding a virtual host.

3. Now we can install the passenger ruby gem, which will provide us with a script that installs the apache passenger module.

   `> gem install passenger`

   We assume the passenger gem has been installed to

   `/var/lib/gems/<RUBY_VERSION>/gems/passenger-<PASSENGER_VERSION>/`

   such that in the subfolder `bin` we can find and execute the script

   `> passenger-install-apache2-module`

   The last command will give you some hints, about missing packages. For every missing package the install command is provided. Please follow these hints and install the missing packages. Also the passenger installer will tell you, how you have to modify your `/etc/apache2/apache2.conf` in order for your apache webserver to load the passenger module.

9.6 Frontend

At this point Ruby and Apache together with the passenger module are assumed to be installed. Now you can install the frontend website, by downloading a zip-file from our server containing the **clusteval** website. After extracting the archive and placing the website at your favored location, you will use some ruby commands, to install all remaining package dependencies (gems) required by the website.

1. Download and extract the latest version of the frontend from our website. Place the extracted contents in your favored directory. We will denote this path by `<WEBSITE_ROOT>` in the following steps.

2. In your terminal navigate to `<WEBSITE_ROOT>` and execute the command

   `> bundle install --no-deployment`

   `> bundle install --deployment`

   This will install all gems required by the website. For this process an internet connection is required.

3. Create an Apache virtual host by creating the file '/etc/apache2/sites-available/clusteval' with the following content and replace the placeholders `<YOUR_DOMAIN>` and `<WEBSITE_ROOT>` with your values.

```
<VirtualHost *:80>
      ServerName <YOUR_DOMAIN>
      DocumentRoot <WEBSITE_ROOT>/public
      <Directory <WEBSITE_ROOT>/public>
         # This relaxes Apache security settings.
         AllowOverride all
```

```
        # MultiViews must be turned off.
        Options -MultiViews
    </Directory>
</VirtualHost>
```

4. You can use the apache2 virtual host configuration command to enable your new clusteval virtual host

   `> a2ensite clusteval`

   Afterwards you can restart your Apache Webserver, for the new settings to take effect.

   `> apache2ctl restart`

5. Now we tell the website which database to use by adapting the database name and user details in `<WEBSITE_ROOT>/config/database.yml`.

6. To initialize the database with a default state, perform

   `> rake db:reset`

10 Basic Usage

After you downloaded and installed **clusteval** , you need to know how to use it. The backend server and client are distributed as two executables which can be simply started with a single command in the command line. Both can be invoked with different command line parameters, which will be explained in the next sections.

For the following scenarios we assume, that the website is ready to use.

10.1 Backend server

The backend server is available as a jar-executable `clustevalBackendServer.jar`. It can be executed from the command line by performing

`java -jar clustevalBackendServer.jar`

There are several command line options available, which are listed in 10.1.1. This command executes the server, which will parse the contents of the repository and wait for commands from clients. Log messages of the server will be printed to the console and to a log file in the invocation directory.

10.1.1 Command line

The backend server provides several command line options:

-absRepoPath <absRepositoryPath> is the absolute path to the repository which should be used by the backend server to store results and parse any data from.

-help prints help and usage information.

-logLevel <**level**> defines the log level of the server. This determines, how verbose the server will be during its execution. Default is 3=INFO. All available logging levels are: 0=ALL, 1=TRACE, 2=DEBUG, 3=INFO, 4=WARN, 5=ERROR, 6=OFF

-serverPort <**port**> is the port the backend server should listen for clients

10.2 Backend client

The backend client is also available as a jar-executable `clustevalBackendClient.jar` and can be executed from the command line by performing

```
java -jar clustevalBackendClient.jar
```

Invoking the client without any parameters besides "clientId", "hostport", "hostip", "logLevel", "help" or "waitForRuns" will open up a tab-completed shell, in which commands can be committed. If any other parameters are specified, the client will directly perform them and terminate immediately afterwards. The available command line parameters are shown in 10.2.1.

10.2.1 Command line

Supported commands of the backend client are:

generateDataSet <**generatorName**> : This command can be used to generate synthetic datasets using the specified generator.

-getDataSets : This tells the client to get and print the available datasets from the server.

-getPrograms : This tells the client to get and print the available programs from the server.

-getQueue : Gets the enqueued runs and run resumes from the backend server.

-getRuns : This tells the client to get and print the available runs from the server.

-getRunResumes : This tells the client to get and print all the run result directories contained in the repository of this server. Those run result directories can be resumed, if they were terminated before.

-getRunResults : This tells the client to get and print all the run result directories contained in the repository of this server, that contain a clusters subfolder and at least one *.complete file containing results (can be slow if many run result folders are present).

-getRunStatus : This tells the client to get the status and percentage (if) of a certain run.

-help prints this help and usage information

-hostip <ip> specifies the ip address of the backend server to connect to. Defaults to the localhost.

-hostport <port> specifies the port number of the backend server to connect to. Defaults to port 1099.

-logLevel <level> defines the log level of the client. This determines, how verbose the client should be during its execution. Available logging levels are: 0=ALL, 1=TRACE, 2=DEBUG, 3=INFO, 4=WARN, 5=ERROR, 6=OFF

-performRun <runName> : This tells the client to perform a run with a certain name.

-resumeRun <runResultIdentifier> : This tells the client to resume a run previously performed identified by its run result identifier.

-shutdown : Orders the backend server to terminate gracefully.

-terminateRun <runName> : This tells the client to terminate the execution of a run with a certain name. shutdown: This tells the client to shutdown the framework.

-waitForRuns : This option can be used together with getRunStatus, in order to cause the client to wait until the run has finished its execution.

10.3 Creating and Executing a Parameter Optimization Run

Goal: You want to create a new parameter optimization run `newParamOptRun`, that applies two clustering methods

- `awesomeCluster` and
- `groupTest`

to two datasets together with their goldstandards

- `myGeneExpressionData_3.txt` (`geneExpr_3_goldstandard.txt`) and
- `methAbs.txt` (`methylation_gs.txt`).

The run should assess three clustering quality measures for every clustering

- `SilhouetteValueRClusteringQualityMeasure`

- TransClustF2ClusteringQualityMeasure
- SensitivityClusteringQualityMeasure

You want to optimize the TransClustF2ClusteringQualityMeasure and use the LayeredDivisiveParameterOptimizationMethod to determine the parameter sets that should be evaluated during the optimization process.

Prerequisites: You downloaded **clusteval** backend server and client, have put the executables into the desired directory. We assume this directory to be <clustEvalDirectory>, such that the executables are located at

- <clustEvalDirectory>/clustevalBackendServer.jar and
- <clustEvalDirectory>/clustevalBackendClient.jar

When the backend server is started without an absolute repository path, it will create (if it does not exist yet) a new repository in the folder <clustEvalDirectory>/repository. We assume the repository path to be <REPOSITORY_ROOT>. The repository is empty, such that we need to create all configuration files and put all input files into the corresponding folders of the repository.

Input files: First you need to put the two dataset and goldstandard files into the repository. For this purpose you have to think of a meaningful foldername containing the dataset/goldstandard. In this case we choose *expression* for the first and *methylation* for the second dataset. Thus we copy the files into the corresponding folders:

- myGeneExpressionData_3.txt to <REPOSITORY_ROOT>/data/datasets/expression/myGeneExpressionData_3.txt
- geneExpr_3_goldstandard.txt to <REPOSITORY_ROOT>/data/goldstandards/expression/geneExpr_3_goldstandard.txt

and

- methAbs.txt to <REPOSITORY_ROOT>/data/datasets/methylation/methAbs.txt
- methylation_gs.txt to <REPOSITORY_ROOT>/data/goldstandards/methylation/methylation_gs.txt

You need to make sure, that the dataset files have a valid header in them. In our example the dataset myGeneExpressionData_3.txt has the format RowSimDataSetFormat (version 1) and methAbs.txt has the format MatrixDataSetFormat (version 1). As the names indicate, the first dataset has the type GeneExpressionDataSetType

and the second one has `MethylationDataSetType`. Since the latter is not available in **clusteval** by default, you need to add it yourself (see 11.2).

Now the headers of your two dataset files look as follows:

```
1 // dataSetFormat = RowSimDataSetFormat
2 // dataSetType = GeneExpressionDataSetType
3 // dataSetFormatVersion = 1
```

and

```
1 // dataSetFormat = MatrixDataSetFormat
2 // dataSetType = MethylationDataSetType
3 // dataSetFormatVersion = 1
```

These headers have to be the first lines in your dataset.

Dataset, Goldstandard & Data Configurations: Now you need to create configuration files for the newly inserted data files.

We create two dataset configuration files under

- `<REPOSITORY_ROOT>/data/datasets/configs/geneExpr.dsconfig`

```
1 datasetName = expression
2 datasetFile = geneExpr_3_goldstandard.txt
```

- `<REPOSITORY_ROOT>/data/datasets/configs/methyl.dsconfig`

```
1 datasetName = methylation
2 datasetFile = methylation_gs.txt
```

We create two goldstandard configuration files under

- `<REPOSITORY_ROOT>/data/goldstandards/configs/geneExpr.gsconfig`

```
1 goldstandardName = expression
2 goldstandardFile = ↩
    ↪ Zachary_karate_club_gold_standard.txt
```

- `<REPOSITORY_ROOT>/data/goldstandards/configs/methyl.gsconfig`

```
1 datasetName = methylation
2 datasetFile = methAbs.txt
```

And finally we create two data configurations which combine the dataset and goldstandard configurations:

- `<REPOSITORY_ROOT>/data/configs/geneExpr.dataconfig`

```
datasetConfig = geneExpr
goldstandardConfig = geneExpr
```

- <REPOSITORY_ROOT>/data/configs/methyl.dataconfig

```
datasetConfig = methyl
goldstandardConfig = methyl
```

Clustering Methods: To add your two clustering methods we again need to think of a foldername and put the executables into the repository:

- <REPOSITORY_ROOT>/programs/awesomeCluster/awesomeCluster
- <REPOSITORY_ROOT>/programs/groupTest/groupTest

Program Configurations: Now you need to create configuration files for each of these clustering methods and put them into the folder <REPOSITORY_ROOT>/programs/configs . They might be located at

- <REPOSITORY_ROOT>/programs/configs/awesomeCluster.config
- <REPOSITORY_ROOT>/programs/configs/groupTest.config

and the contents might look like this:

```
program = awesomeCluster/awesomeCluster
parameters = a,w
optimizationParameters = a
compatibleDataSetFormats = RowSimDataSetFormat
outputFormat = AwesomeRunResultFormat
alias = Awesome Cluster

[invocationFormat]
invocationFormat = %e% --input %i% --output %o% -a %a%↩
   ↪  -b %b% -c %c%

[a]
desc = The a parameter
type = 2
def = 1002
minValue = 900
maxValue = 2000

[w]
desc = The w parameter
type = 2
```

```
def = 0.001
minValue = 0.0
maxValue = 0.02
```

and

```
program = groupTest/groupTest
parameters = g
optimizationParameters = g
compatibleDataSetFormats = RowSimDataSetFormat,↩
    ↪ SimMatrixDataSetFormat
outputFormat = GroupRunResultFormat
alias = Group-Test Clustering

[invocationFormat]
invocationFormat = %e% -g %g% %i% %o% -silent

[g]
desc = The grouping coefficient
type = 2
def = 1
minValue = 0
maxValue = 15.3
```

In this case we have to follow the explanations under 11.4 to add the two runresult formats of the clusterin methods to our framework.

Run: Now we create the parameter optimization run file under

<REPOSITORY_ROOT>/runs/myNewRun.run

with the details specified above:

```
programConfig = awesomeCluster,groupTest
dataConfig = methyl,geneExpr
qualityMeasures = TransClustF2ClusteringQualityMeasure↩
    ↪ ,SilhouetteValueRClusteringQualityMeasure,↩
    ↪ SensitivityClusteringQualityMeasure
mode = parameter_optimization
optimizationMethod = ↩
    ↪ LayeredDivisiveParameterOptimizationMethod
optimizationCriterion = ↩
    ↪ TransClustF2ClusteringQualityMeasure
optimizationIterations = 100

[awesomeCluster]
```

```
10 optimizationParameters = a
11
12 [groupTest]
13 optimizationParameters = g
```

Executing: To execute our new run, we start the backend server

```
> java -jar <clustEvalDirectory>/clustevalBackendServer.jar
```

we wait until everything contained in the repository is parsed. Then we start the client

```
> java -jar <clustEvalDirectory>/clustevalBackendClient.jar
```

This will open up a shell and we can start our new run by typing

```
localhost:1099> performRun myNewRun
```

11 Extending the Framework

clusteval can be extended in different ways. The following sections will show you, which functionality you can add to the framework and how.

11.1 Clustering Methods

As explained in 4.2 **clusteval** supports two different kinds of clustering methods: Standalone programs and R programs. Standalone programs are those, for which you have to provide an executable file which then will be executed by the framework. R programs are methods implemented in R, which will be invoked by **clusteval** by using the Rserve interface.

11.1.1 Standalone Programs

Standalone programs can be added to **clusteval** by

1. putting the executable file (together with all required shared libraries) into a respective folder in the repository programs directory

   <REPOSITORY_ROOT>/programs/<programFolder>/<executable>

2. putting a program configuration file (see 4.9.7) into the repository program configuration directory

   <REPOSITORY_ROOT>/programs/configs

3. if the program requires a new input format, follow the instructions under 11.3 for the new input format

4. if the program has an unknown output format, follow the instructions under 11.4 for the new output format

11.1.2 R Programs

R programs can be added to **clusteval** by

1. extending the class `de.clusteval.program.r.RProgram` with your own class `MyRProgram`. You have to provide your own implementations for the following methods, otherwise the framework will not be able to load your class.

   (a) `public MyRProgram(Repository)`: The constructor of your class taking a repository parameter. This constructor has to be implemented and public, otherwise the framework will not be able to load your class.

   (b) `public MyRProgram(MyRProgram)`: The copy constructor of your class taking another instance of your class. This constructor has to be implemented and public.

   (c) `public String getAlias()`: This alias is used whenever this program is visually represented and a readable name is needed. This is used to represent your program on the website for example.

   (d) `public String getInvocationFormat()`: This is the invocation of the R method including potential parameters, that have to be defined in the program configuration.

   (e) `public Set getRequiredLibraries()`: This method returns the set of strings, with names of all required R libraries this program uses.

   (f) `public Process exec(DataConfig,ProgramConfig,String[],Map,Map)`: In this method the actual execution of the R Program happens. In here you have to implement the invocation of the R method via Rserve and any pre- and postcalculations necessary. The communications with R can be visualized by the following code snippet:

```
try {
    // precalculations
    double[] input = ...;
    ...
    MyRengine rEngine = new MyRengine("");
    try {
        rEngine.assign("input",input);
        rEngine.eval("result <- yourMethodInvocation()");
        REXP result = rEngine.eval("result@.Data");

        // postcalculations
        ...
    } catch (RserveException e) {
        e.printStackTrace();
    } finally {
```

```
            rEngine.close();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    // for return type compatibility reasons
    return null;
```

2. Creating a jar file named `MyRProgram.jar` containing the `MyRProgram.class` compiled on your machine in the correct folder structure corresponding to the packages:

   `de/clusteval/program/r/MyRProgram.class`

3. Putting the `MyRProgram.jar` into the programs folder of the repository:

   `<REPOSITORY_ROOT>/programs`

   The backend server will recognize and try to load the new program automatically the next time, the `RProgramFinderThread` checks the filesystem.

11.2 Dataset Types

Dataset types can be added to **clusteval** by

1. extending the class `de.clusteval.data.dataset.type.DataSetType` with your own class `MyDataSetType`. You have to provide your own implementations for the following methods, otherwise the framework will not be able to load your class.

   (a) `public MyDataSetType(Repository, boolean,long, File)`: The constructor of your class. This constructor has to be implemented and public, otherwise the framework will not be able to load your class.

   (b) `public MyDataSetType(MyDataSetType)`: The copy constructor of your class taking another instance of your class. This constructor has to be implemented and public.

   (c) `public String getAlias()`: This alias is used whenever this program is visually represented and a readable name is needed. This is used to represent your program on the website for example.

2. Creating a jar file named `MyDataSetType.jar` containing the `MyDataSetType.class` compiled on your machine in the correct folder structure corresponding to the packages:

   `de/clusteval/data/dataset/type/MyDataSetType.class`

3. Putting the MyDataSetType.jar into the dataset types folder of the repository:

   <REPOSITORY_ROOT>/supp/types/dataset

   The backend server will recognize and try to load the new dataset type automatically the next time, the DataSetTypeFinderThread checks the filesystem.

11.3 Dataset Formats

A dataset format MyDataSetFormat can be added to **clusteval** by

1. extending the class de.clusteval.data.dataset.format.DataSetFormat with your own class MyDataSetFormat. You have to provide your own implementations for the following methods, otherwise the framework will not be able to load your dataset format.

   (a) public MyDataSetFormat(Repository, boolean, long, File, int): The constructor of your dataset format class. This constructor has to be implemented and public, otherwise the framework will not be able to load your dataset format.

   (b) public MyDataSetFormat(MyDataSetFormat): The copy constructor of your class taking another instance of your class. This constructor has to be implemented and public.

2. extending the class de.clusteval.data.dataset.format.DataSetFormatParser with your own class MyDataSetFormatParser. You have to provide your own implementations for the following methods, otherwise the framework will not be able to load your class.

   (a) public DataSet convertToStandardFormat(DataSet, ConversionInputToStandardConfiguration): This method converts the given dataset to the standard input format of the framework using the given conversion configuration. This assumes, that the passed dataset has this format.

   (b) public DataSet convertToThisFormat(DataSet, DataSetFormat, ConversionConfiguration): This method converts the given dataset to the given input format using the conversion configuration.

   (c) public Object parse(DataSet): This method parses the given dataset and returns an object, wrapping the contents of the dataset (e.g. an instance of SimilarityMatrix or DataMatrix).

3. Creating a jar file named MyDataSetFormat.jar containing the MyDataSetFormat.class and MyDataSetFormatParser.class compiled on your machine in the correct folder structure corresponding to the packages:

`de/clusteval/data/dataset/format/MyDataSetFormat.class`

`de/clusteval/data/dataset/format/MyDataSetFormatParser.class`

4. Putting the `MyDataSetFormat.jar` into the dataset formats folder of the repository:

   `<REPOSITORY_ROOT>/supp/formats/dataset`

   The backend server will recognize and try to load the new dataset format automatically the next time, the `DataSetFormatFinderThread` checks the filesystem.

11.4 Runresult Formats

A runresult format `MyRunResultFormat` can be added to **clusteval** by

1. extending the class `de.clusteval.run.result.format.RunResultFormat` with your own class `MyRunResultFormat`. You have to provide your own implementations for the following methods, otherwise the framework will not be able to load your runresult format.

   (a) `public MyRunResultFormat(Repository, boolean, long, File)`: The constructor of your runresult format class. This constructor has to be implemented and public, otherwise the framework will not be able to load your runresult format.

   (b) `public MyRunResultFormat(MyRunResultFormat)`: The copy constructor of your class taking another instance of your class. This constructor has to be implemented and public.

2. extending the class `de.clusteval.run.result.format.RunResultFormatParser` with your own class `MyRunResultFormatParser`. You have to provide your own implementations for the following methods, otherwise the framework will not be able to load your class.

   (a) `public void convertToStandardFormat()`: This method converts the given runresult to the standard runresult format of the framework. The converted runresult has to be named exactly as the input file postfixed with the extension ".conv". The original runresult

   `<REPOSITORY_ROOT>/results/<runIdentifier>/clusters/TransClust_sfld.1.result`

   has to be converted to

   `<REPOSITORY_ROOT>/results/<runIdentifier>/clusters/TransClust_sfld.1.result.conv`

   by this method. A wrapper object for the converted runresult has to be stored in the result attribute.

3. Creating a jar file named `MyRunResultFormat.jar` containing the `MyRunResultFormat.class` and `MyRunResultFormatParser.class` compiled on your machine in the correct folder structure corresponding to the packages:

   `de/clusteval/run/result/format/MyRunResultFormat.class`

   `de/clusteval/run/result/format/MyRunResultFormatParser.class`

4. Putting the `MyRunResultFormat.jar` into the runresult formats folder of the repository:

   `<REPOSITORY_ROOT>/supp/formats/runresult`

   The backend server will recognize and try to load the new runresult format automatically the next time, the `RunResultFormatFinderThread` checks the filesystem.

11.5 Parameter Optimization Methods

A parameter optimization method `MyParameterOptimizationMethod` can be added to **clusteval** by

1. extending the class `de.clusteval.cluster.paramOptimization.ParameterOptimizationMethod` with your own class `MyParameterOptimizationMethod`. You have to provide your own implementations for the following methods, otherwise the framework will not be able to load your parameter optimization method.

   (a) `public MyParameterOptimizationMethod(Repository, boolean, long, File, ParameterOptimizationRun, ProgramConfig, DataConfig, List, ClusteringQualityMeasure, int[], boolean)`: The constructor for your parameter optimization method. This constructor has to be implemented and public, otherwise the framework will not be able to load your parameter optimization method.

   (b) `public MyParameterOptimizationMethod( MyParameterOptimizationMethod)`: The copy constructor for your parameter optimization method. This constructor has to be implemented and public, otherwise the framework will not be able to load your parameter optimization method.

   (c) `public List getCompatibleDataSetFormatBaseClasses()`: A list of dataset formats, this parameter optimization method can be used for. If the list is empty, all dataset formats are assumed to be compatible.

   (d) `public List getCompatibleProgramNames()`: A list of names of all programs that are compatible to this parameter optimization method. If the list is empty, all programs are assumed to be compatible.

(e) `public boolean hasNext()` : This method indicates, whether their is another parameter set to evaluate. This method **must not change** the current parameter set, as it may be invoked several times before next() is invoked.

(f) `protected ParameterSet getNextParameterSet(ParameterSet)` : Returns the next parameter set to evaluate. This method may change the internal status of the parameter optimization method, in that it stores the newly determined and returned parameter set as the current parameter set.

(g) `public Set getRequiredRlibraries()` : Returns a set of names of all R libraries, this parameter optimization method requires.

(h) `public int getTotalIterationCount()` : This is the total iteration count this parameter optimization method will perform. The returned value might not correspond to the expected value, when the method is instantiated. Therefore always use the return value of this method, when trying to determine the finished percentage of the parameter optimization process.

2. Creating a jar file named `MyParameterOptimizationMethod.jar` containing the `MyParameterOptimizationMethod.class` compiled on your machine in the correct folder structure corresponding to the packages:

   ```
   de/clusteval/cluster/paramOptimization/MyParameterOptimizationMethod.class
   ```

3. Putting the `MyParameterOptimizationMethod.jar` into the parameter optimization methods folder of the repository:

   `<REPOSITORY_ROOT>/supp/clustering/paramOptimization`

   The backend server will recognize and try to load the new parameter optimization method automatically the next time, the `ParameterOptimizationMethodFinderThread` checks the filesystem.

11.6 Distance Measures

A distance measure `MyDistanceMeasure` can be added to **clusteval** by

1. extending the class `de.clusteval.data.distance.DistanceMeasure` with your own class `MyDistanceMeasure` . You have to provide your own implementations for the following methods, otherwise the framework will not be able to load your distance measure.

   (a) `public MyDistanceMeasure(Repository, boolean, long, File)` : The constructor for your distance measure. This constructor has to be implemented and public, otherwise the framework will not be able to load your distance measure.

(b) `public MyDistanceMeasure(MyDistanceMeasure)` : The copy constructor for your distance measure. This constructor has to be implemented and public, otherwise the framework will not be able to load your distance measure.

(c) `public double getDistance(double[],double[])` : This method is the core of your distance measure. It returns the distance of the two points specified by the absolute coordinates in the two double arrays.

(d) `public boolean supportsMatrix()` : This method indicates, whether your distance measure can calculate distances of a whole set of point-pairs, i.e. your distance measure implements the method getDistances(double[][]).

(e) `public double[][] getDistances(double[][])` : The absolute coordinates of the points are stored row-wise in the given matrix and distances are calculated between every pair of rows. Position [i][j] of the returned double[][] matrix contains the distance between the i-th and j-th row of the input matrix.

(f) `public Set getRequiredRlibraries()` : Returns a set of names of all R libraries, this distance measure requires.

2. Creating a jar file named `MyDistanceMeasure.jar` containing the `MyDistanceMeasure.class` compiled on your machine in the correct folder structure corresponding to the packages:

   ```
   de/clusteval/data/distance/MyDistanceMeasure.class
   ```

3. Putting the `MyDistanceMeasure.jar` into the distance measure folder of the repository:

   `<REPOSITORY_ROOT>/supp/distanceMeasures`

   The backend server will recognize and try to load the new distance measure automatically the next time, the `DistanceMeasureFinderThread` checks the filesystem.

11.7 Clustering Quality Measures

A clustering quality measure `MyClusteringQualityMeasure` can be added to **clusteval** by

1. extending the class `de.clusteval.cluster.quality.ClusteringQualityMeasure` with your own class `MyClusteringQualityMeasure`. You have to provide your own implementations for the following methods, otherwise the framework will not be able to load your clustering quality measure.

   (a) `public MyClusteringQualityMeasure(Repository, boolean, long, File)` : The constructor for your distance measure. This constructor has to be implemented and public, otherwise the framework will not be able to load your clustering quality measure.

(b) `public MyClusteringQualityMeasure(MyClusteringQualityMeasure)` : The copy constructor for your distance measure. This constructor has to be implemented and public, otherwise the framework will not be able to load your clustering quality measure.

(c) `public String getAlias()` : This method returns a readable alias for this clustering quality measure which is used e.g. on the website.

(d) `public double getMinimum()` : Returns the minimal value this measure can calculate.

(e) `public double getMaximum()` : Returns the maximal value this measure can calculate.

(f) `public boolean requiresGoldStandard()` : Indicates, whether this clustering quality measure requires a goldstandard to assess the quality of a given clustering.

(g) `public ClusteringQualityMeasureValue getQualityOfClustering(Clustering)` : This method is the core of your clustering quality measure. It assesses and returns the quality of the given clustering.

(h) `public boolean isBetterThanHelper(ClusteringQualityMeasureValue` : This method is used by sorting algorithms of the framework to compare clustering quality measure results and find the optimal parameter sets.

(i) `public Set getRequiredRlibraries()` : Returns a set of names of all R libraries, this clustering quality measure requires.

2. Creating a jar file named `MyClusteringQualityMeasure.jar` containing the `MyClusteringQualityM` compiled on your machine in the correct folder structure corresponding to the packages:

   `de/clusteval/cluster/quality/MyClusteringQualityMeasure.class`

3. Putting the `MyClusteringQualityMeasure.jar` into the clustering quality measure folder of the repository:

   `<REPOSITORY_ROOT>/supp/clustering/qualityMeasures`

   The backend server will recognize and try to load the new clustering quality measure automatically the next time, the `ClusteringQualityMeasureFinderThread` checks the filesystem.

11.8 Data Statistics

A data statistic `MyDataStatistic` can be added to **clusteval** by

1. extending the class `de.clusteval.data.statistic.DataStatistic` with your own class `MyDataStatistic` . You have to provide your own implementations for the following methods, otherwise the framework will not be able to load your class.

(a) public MyDataStatistic(Repository, boolean, long, File) : The constructor for your data statistic. This constructor has to be implemented and public, otherwise the framework will not be able to load your class.

(b) public MyDataStatistic(MyDataStatistic) : The copy constructor for your data statistic. This constructor has to be implemented and public, otherwise the framework will not be able to load your class.

(c) public boolean requiresGoldStandard() : Indicates, whether this data statistic requires a goldstandard to assess the property of a given dataset.

(d) public String getAlias() : This method returns a readable alias for this data statistic which is used e.g. on the website.

(e) public void parseFromString(String) : This method interprets the string and fills this statistic object with its parsed contents.

(f) public Set getRequiredRlibraries() : Returns a set of names of all R libraries, this data statistic requires.

2. extending the class de.clusteval.data.statistic.DataStatisticCalculator with your own class MyDataStatisticCalculator . You have to provide your own implementations for the following methods, otherwise the framework will not be able to load your class.

   (a) public MyDataStatisticCalculator(Repository, long, File, DataConfig) : The constructor for your data statistic calculator. This constructor has to be implemented and public, otherwise the framework will not be able to load your class.

   (b) public MyDataStatisticCalculator(MyDataStatisticCalculator) : The copy constructor for your data statistic calculator. This constructor has to be implemented and public, otherwise the framework will not be able to load your class.

   (c) protected MyDataStatistic calculateResult() : This method is the core of your data statistic calculator. It analysis the given data configuration and returns a wrapper object for the results.

   (d) public void writeOutputTo(File) : After calculateResult() has been invoked, this method writes the assessed results into the given file.

3. Creating a jar file named MyDataStatisticCalculator.jar containing the MyDataStatistic.class and MyDataStatisticCalculator.class compiled on your machine in the correct folder structure corresponding to the packages:

   `de/clusteval/data/statistics/MyDataStatistic.class`

   `de/clusteval/data/statistics/MyDataStatisticCalculator.class`

4. Putting the `MyDataStatistic.jar` into the data statistics folder of the repository:

   `<REPOSITORY_ROOT>/supp/statistics/data`

   The backend server will recognize and try to load the new data statistics automatically the next time, the `DataStatisticFinderThread` checks the filesystem.

11.9 Run Statistics

A run statistic `MyRunStatistic` can be added to **clusteval** by

1. extending the class `de.clusteval.run.statistic.RunStatistic` with your own class `MyRunStatistic`. You have to provide your own implementations for the following methods, otherwise the framework will not be able to load your class.
   (a) `public MyRunStatistic(Repository, boolean, long, File)` : The constructor for your run statistic. This constructor has to be implemented and public, otherwise the framework will not be able to load your class.
   (b) `public MyRunStatistic(MyRunStatistic)` : The copy constructor for your run statistic. This constructor has to be implemented and public, otherwise the framework will not be able to load your class.
   (c) `public String getAlias()` : This method returns a readable alias for this run statistic which is used e.g. on the website.
   (d) `public void parseFromString(String)` : This method interprets the string and fills this statistic object with its parsed contents.
   (e) `public Set getRequiredRlibraries()` : Returns a set of names of all R libraries, this run statistic requires.
2. extending the class `de.clusteval.run.statistic.RunStatisticCalculator` with your own class `MyRunStatisticCalculator`. You have to provide your own implementations for the following methods, otherwise the framework will not be able to load your class.
   (a) `public MyRunStatisticCalculator(Repository, long, File, DataConfig)` : The constructor for your run statistic calculator. This constructor has to be implemented and public, otherwise the framework will not be able to load your class.
   (b) `public MyRunStatisticCalculator(MyRunStatisticCalculator)` : The copy constructor for your run statistic calculator. This constructor has to be implemented and public, otherwise the framework will not be able to load your class.
   (c) `protected MyRunStatistic calculateResult()` : This method is the core of your run statistic calculator. It analysis the given runresults and returns a wrapper object for the results.

(d) `public void writeOutputTo(File)` : After calculateResult() has been invoked, this method writes the assessed results into the given file.

3. Creating a jar file named `MyRunStatisticCalculator.jar` containing the `MyRunStatistic.class` and `MyRunStatisticCalculator.class` compiled on your machine in the correct folder structure corresponding to the packages:

   `de/clusteval/run/statistics/MyRunStatistic.class`

   `de/clusteval/run/statistics/MyRunStatisticCalculator.class`

4. Putting the `MyRunStatistic.jar` into the run statistics folder of the repository:

   `<REPOSITORY_ROOT>/supp/statistics/run`

   The backend server will recognize and try to load the new run statistics automatically the next time, the `RunStatisticFinderThread` checks the filesystem.

11.10 Run-Data Statistics

A run-data statistic `MyRunDataStatistic` can be added to **clusteval** by

1. extending the class `de.clusteval.run.statistic.RunDataStatistic` with your own class `MyRunDataStatistic`. You have to provide your own implementations for the following methods, otherwise the framework will not be able to load your class.

   (a) `public MyRunDataStatistic(Repository, boolean, long, File)` : The constructor for your run-data statistic. This constructor has to be implemented and public, otherwise the framework will not be able to load your class.

   (b) `public MyRunDataStatistic(MyRunDataStatistic)` : The copy constructor for your run-data statistic. This constructor has to be implemented and public, otherwise the framework will not be able to load your class.

   (c) `public String getAlias()` : This method returns a readable alias for this run-data statistic which is used e.g. on the website.

   (d) `public void parseFromString(String)` : This method interprets the string and fills this statistic object with its parsed contents.

   (e) `public Set getRequiredRlibraries()` : Returns a set of names of all R libraries, this run-data statistic requires.

2. extending the class `de.clusteval.run.statistic.RunDataStatisticCalculator` with your own class `MyRunDataStatisticCalculator`. You have to provide your own implementations for the following methods, otherwise the framework will not be able to load your class.

(a) `public MyRunDataStatisticCalculator(Repository, long, File, DataConfig)` : The constructor for your run-data statistic calculator. This constructor has to be implemented and public, otherwise the framework will not be able to load your class.

(b) `public MyRunDataStatisticCalculator(MyRunDataStatisticCalculator)` : The copy constructor for your run-data statistic calculator. This constructor has to be implemented and public, otherwise the framework will not be able to load your class.

(c) `protected MyRunDataStatistic calculateResult()` : This method is the core of your run-data statistic calculator. It analysis the given runresults (data analysis and clustering) and returns a wrapper object for the results.

(d) `public void writeOutputTo(File)` : After calculateResult() has been invoked, this method writes the assessed results into the given file.

3. Creating a jar file named `MyRunDataStatisticCalculator.jar` containing the `MyRunDataStatistic.` and `MyRunDataStatisticCalculator.class` compiled on your machine in the correct folder structure corresponding to the packages:

   `de/clusteval/run/statistics/MyRunDataStatistic.class`

   `de/clusteval/run/statistics/MyRunDataStatisticCalculator.class`

4. Putting the `MyRunDataStatistic.jar` into the run-data statistics folder of the repository:

   `<REPOSITORY_ROOT>/supp/statistics/rundata`

   The backend server will recognize and try to load the new run-data statistics automatically the next time, the `RunDataStatisticFinderThread` checks the filesystem.

References

[1] Christian Wiwie. Development of an integrated clustering framework for cluster analysis. Master's thesis, Max-Planck Institute for Computer Science, 2013.