ClustEval

Jan 27, 2020

Contents

ClustEval is an automated framework for cluster analysis written in Java which comes together with a Ruby on Rails website. This includes most common subtasks of every cluster analysis, such as

- *data preprocessing & data normalization*
- *data format conversion*
- *clustering method execution*
- *parameter training and*
- *visualizations*

CHAPTER 1

Getting Started

1.1 Getting started

1.1.1 Using Docker

Follow the instructions *here* to install ClustEval using docker. Then browse to the administration interface at the machine on which you started ClustEval, e.g. http://localhost:8080.

Use the administration web interface to:

- upload data sets
- upload clustering methods, R programs as well as standalone programs
- create runs
- upload ClustEval components, such as distance measures
- execute / stop runs
- supervise the progress of run executions
- inspect clustering and analysis results

1.1.2 Manual installation (outdated)

Warning: This documentation is outdated. We highly recommend using docker and docker-compose instead.

Follow the instructions *here* to manually install ClustEval.

Note: In this scenario the administration interface is not available.

The backend server and client are distributed as two Java executables which can be simply started with a single command in the command line. Both can be invoked with different command line parameters, which are explained *here*. For the following scenarios we assume, that you have followed the instructions under *Download & Installation*.

Here we assume, that you have cloned the

- **clusteval** GitHub repository to /home/clusteval/clusteval
- **clustevalWebsite** GitHub repository to /home/clusteval/clustevalWebsite

Check out a Minimal Repository

We provide a ready to use repository on GitHub. You can check it out using the git command line tool:

```
git clone https://github.com/wiwie/clustevalDockerRepository /home/clusteval/
↪clustevalDockerRepository
```

Do you need the Website?

Using the website is completely optional. If not needed, edit the repository configuration file under:

```
/home/clusteval/clustevalDockerRepository/repository.config
```

and simply remove the postgresql/mysql sections. ClustEval will now not connect to a database.

Start the Backend Server

As mentioned above, the backend server is a Java executable (jar) and has several parameters. The parameters are described *here*. We use the repository from the previous step:

```
java -jar /home/clusteval/clusteval/clusteval/packages/clustevalBackendServer.jar --
↪absRepoPath /home/clusteval/clustevalDockerRepository
```

If you are using a database at this point, you will be prompted for the credentials.

ClustEval will now start parsing the contents of the repository and will show you status messages in the terminal. Once it is finished, it will inform you that the server is now listening for clients commands.

Start the Backend Client

The client is a Java executable as well and can be started as follows:

```
java -jar /home/clusteval/clusteval/clusteval/packages/clustevalBackendClient.jar
```

It will now connect to the backend server and present you with a CLI to enter commands which will then be sent to the server. For example, you can get a list of all available runs in our repository:

```
getRuns
```

Afterwards you can perform one of those runs by entering:

```
performRun bonemarrow
```

ClustEval will now cluster the bone_marrow data set with DBSCAN, Fanny, KMedoids, K-means, Transitivity Clustering, Hierarchical Clustering, Affinity Propagation, Markov Clustering, Self Organizing Maps, Spectral Clustering, clusterdp, clusterONE and MCODE and 1000 parameter sets each. For each of the resulting clusterings it will evaluate the clustering quality measures which are specified in the run file (bonemarrow.run).

Getting the Status of the Run Execution

To track the progress of your run, you can either follow the status messages on the server terminal or use the client:

```
getRunStatus <RESULTID>
```

Here <RESULTID> is the unique identifier that your run execution was assigned. By typing getRunStatus into the client CLI, pressing tab will show you all available such ids. Which currently, should only be one for the bonemarrow run.

Since we are executing a run of type parameter optimization, we can also use the following command of the client to get a more detailed status:

```
getOptRunStatus <RESULTID>
```

Terminating a Run

If the run takes too long or you want to decrease the number of parameter sets (iterations) per clustering method, you first want to stop the running job, before editing the files and starting it again:

```
terminateRun <RESULTID>
```

Inspecting the Results

The results of a run can be either inspected using the client, the result files itself or the website (if the database and website have been set up and the repository.config contained a configuration for the database).

Using the Client

We can get a summary of the performances of each tool using the following command:

```
getOptRunStatus <RESULTID>
```

On the File-System

Each run execution is assigned a unique result id which you have already been using in the steps above. The results of a run are also stored under this result id on the file-system. If we want to inspect the results of our run, we can navigate to:

```
/home/clusteval/clustevalDockerRepository/results/<RESULTID>/clusters
```

This folder contains all the clustering results procuced by any of the clustering methods. Each *.complete file summarizes all iterations of one particular clustering method and makes it easier to identify parameter sets which performed better.

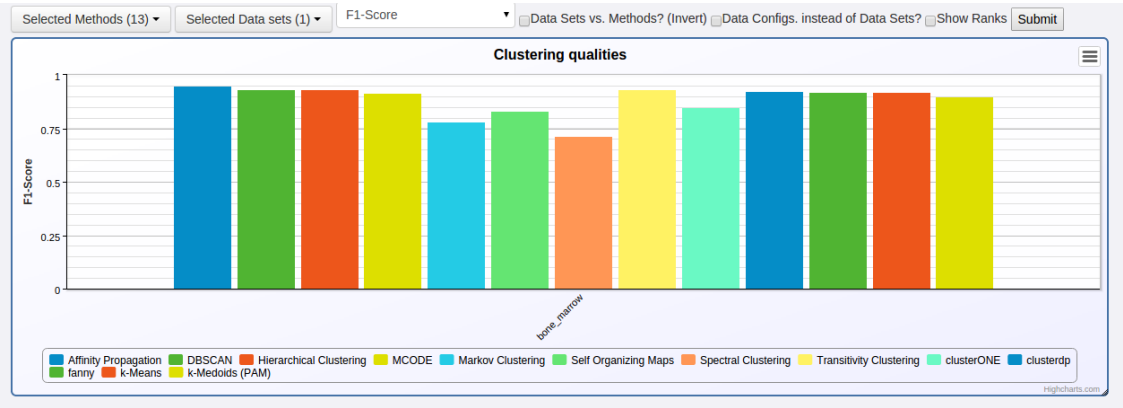
On the Website

If you have configured the server to run with a database connection, it will automatically parse the clustering results into the database once a run terminates.

You can then get a quick overview on the Overview page:

```
Start page -> Overview
```

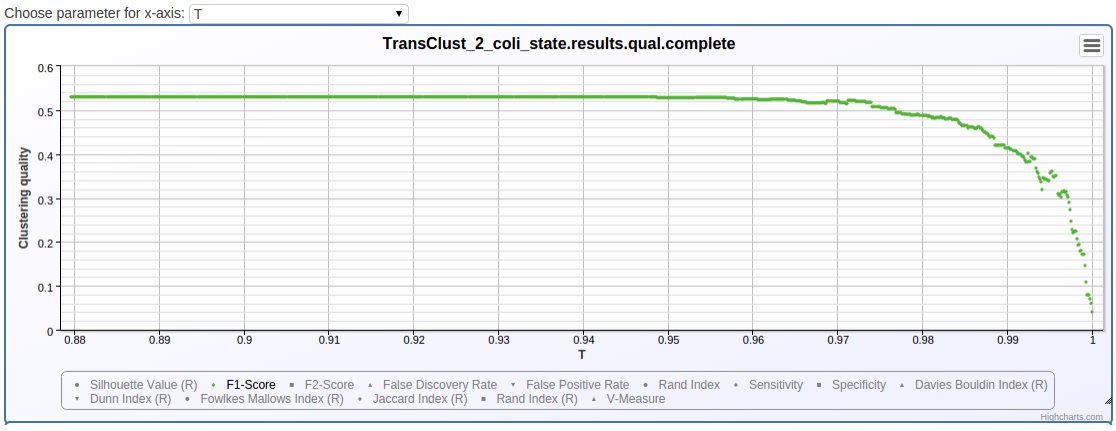
Select the clustering quality measure on the top and you see which clustering method performed how well on the bone_marrow data set.

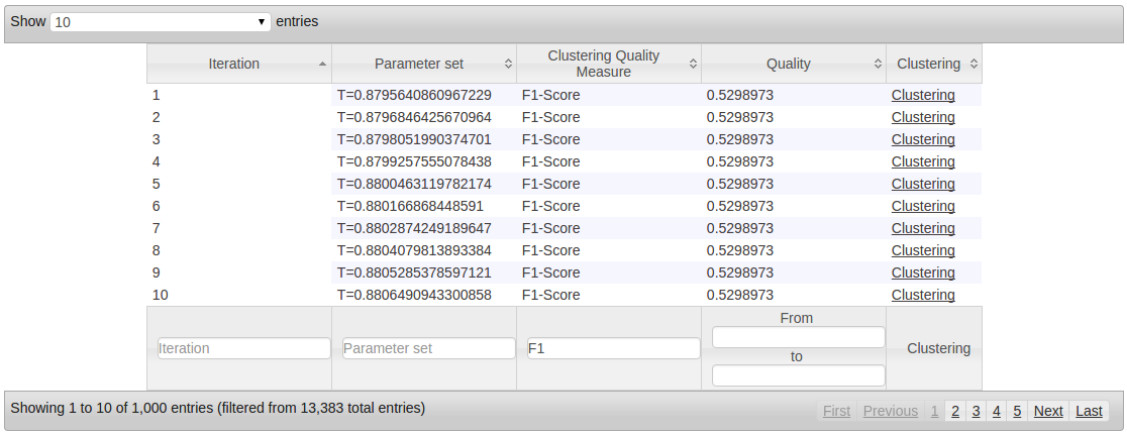


To get more detailed information about the run, navigate to:

```
Start page -> Advanced -> Run results -> Click on result with <RESULTID>
```

The page shows you a section for each pair of program configuration and data configuration. The iterations of the parameter optimization are shown in a table and a chart.




Show 10 entries

Iteration	Parameter set	Clustering Quality Measure	Quality	Clustering
1	T=0.8795640860967229	F1-Score	0.5298973	Clustering
2	T=0.8796846425670964	F1-Score	0.5298973	Clustering
3	T=0.8798051990374701	F1-Score	0.5298973	Clustering
4	T=0.8799257555078438	F1-Score	0.5298973	Clustering
5	T=0.8800463119782174	F1-Score	0.5298973	Clustering
6	T=0.880166868448591	F1-Score	0.5298973	Clustering
7	T=0.8802874249189647	F1-Score	0.5298973	Clustering
8	T=0.8804079813893384	F1-Score	0.5298973	Clustering
9	T=0.8805285378597121	F1-Score	0.5298973	Clustering
10	T=0.8806490943300858	F1-Score	0.5298973	Clustering
Iteration	Parameter set	F1	From / to	Clustering

Showing 1 to 10 of 1,000 entries (filtered from 13,383 total entries) First Previous 1 2 3 4 5 Next Last

1.2 Download & Installation

1.2.1 Docker & Docker Compose (OS X and Linux only)

ClustEval can be installed using Docker and Docker Compose on OS X and Linux.

1. Install Docker & Docker-Compose
2. Check out our git repository with all dependencies (using the –recursive switch): “git clone -b 1.8.1 –recursive https://gitlab.com/clusteval/clusteval-docker-compose”.
3. Execute “docker-compose up” in the folder containing the docker-compose.yml from the checked out repository.

1.2.2 VirtualBox Image (Windows, OS X, Linux, Solaris) (outdated)

Warning: This documentation is outdated. We highly recommend using docker and docker-compose instead.

Download & Set-Up

You can also install ClustEval by downloading one of our VirtualBox images. All VirtualBox images contain an installation of Ubuntu 16.04.1 and have the user **clusteval** preinstalled with password **clusteval**. The user has administrator rights.

We provide two VirtualBox images:

Link	ClustEval Version	Date	Size
ClustEval NetInstall	Latest	17.09.2015	1.6 GB
ClustEval Complete	1.6	•	4.7 GB

ClustEval Complete This image has version 1.6 of ClustEval already installed and you can just start up backend, frontend and are ready to go.

ClustEval NetInstall This image comes only with an installation script which downloads the newest versions of all necessary components from the web. It is thus smaller, but after download of the ClustEval VirtualBox an internet connection is required.

After downloading one of the above Virtualbox images, you need to create a Linux/Ubuntu VirtualBox host and add the image as hard disk. Make sure to allow sufficient RAM to the box. We recommend at least 2GB.

Start-Up

Both VirtualBox images come with Ubuntu and have an active user clusteval with password clusteval.

After creating a new VirtualBox host and adding one of the images as disk, to VirtualBox booting them up logging in The "ClustEval Complete" image has a shortcut on the desktop

1.2.3 Manual Download & Installation (outdated)

Warning: This documentation is outdated. We highly recommend using docker and docker-compose instead.

To manually install ClustEval you first need to install the following dependencies. We provide all components of ClustEval in GitHub repositories.

Backend Dependencies

The backend is a Java application which communicates with R using Rserve. Thus, you need to install Java, R and Rserve.

Java

A working installation of Java 7 to execute server and client jar-executables. We recommend you to use Oracle Java 7 (over Open-JDK), as we observed performance degenerations with Open-JDK and unexpected behaviour.

To install Oracle Java in Ubuntu execute the following commands in a terminal:

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java7-installer
```

R >= 2.15 & Rserve

ClustEval uses some existing implementations of statistical calculations in R instead of implementing them from scratch in Java again. This is used in many components of ClustEval, such as distance measures, data statistics or clustering quality measures. To make R available in Java applications (like ClustEval), Rserve is required. 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 programs written in other programming languages, for example Java.

In principle the clustering processes of ClustEval can be started without R or a running Rserve instance, but a lot of functionality will then not be available.

Hint: Rserve requires at least version 2.15 of R.

To install R in Ubuntu execute the following commands in a terminal:

```
sudo apt-get update
sudo apt-get install r-base
```

To then install the Rserve package and start up the Rserve daemon open an R terminal by typing R in the terminal and then execute the following commands:

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

Frontend Dependencies

PostgreSQL >= 9.3 / MySQL

The website requires a database. At the moment PostgreSQL >= 9.3 and MySQL are supported.

In Ubuntu you can install postgreSQL as follows:

```
sudo apt-get install postgresql
```

You can then connect to the server and create a database and a user:

```
CREATE DATABASE clusteval;
CREATE USER clusteval LOGIN UNENCRYPTED PASSWORD '<YOUR_PASSWORD>';
GRANT ALL PRIVILEGES ON DATABASE clusteval to clusteval;
```

Alternatively, you can install MySQL with the following command:

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

During the installation procedure you will be asked for a root password, and the credentials of a non-root user. Here we assume your non-root user to be **clusteval**.

You then need to create a database and to grant read and write permissions on that database to the **clusteval** user:

```
CREATE DATABASE clusteval;
GRANT ALL ON clusteval.* TO clusteval;
```

Ruby >= 2.2 & Rails >= 4.2

The website is written in Ruby on Rails and thus requires the programming language Ruby as well as the web-framework Rails to be installed.

The easiest way to get an up-to-date installation of both is using rvm (Ruby Version Manager).

Thus, we first install rvm:

```
sudo apt-get update
sudo apt-get install curl
\curl -L https://get.rvm.io | bash -s stable
source ~/.rvm/scripts/rvm
rvm requirements
```

We can now use rvm to install the current version of ruby and activate it:

```
rvm install ruby
rvm use ruby --default
```

We furthermore need the rubygems package which provides us with some tools, such as the gem tool which we can use to install Ruby on Rails gems:

```
rvm rubygems current
```

And then we install the bundler gem, which is a gem that automatizes the installation of our ClustEval website:

```
gem install bundler
```

Download Backend (Client & Server)

To use the backend it is sufficient to download the server and client jar executables. Alternatively, you can also check out the whole GitHub repository to have the Java sources as well (consumes much more space).

Link	Version
Backend Server Executable Only	Latest
Backend Client Executable Only	Latest
Backend GitHub repository	Latest

Set-Up a ClustEval Repository

Here you have again two choices:

1. Clone our GitHub repository containing all dynamic components (e.g. formats, distance measures). This GitHub repository contains the folder "clustevalPackages/repository" which is a minimal ClustEval repository without any data sets, standalone programs, runs or configurations.
2. We provide a slightly larger example repository containing some data sets, gold standards, clustering methods and configurations. This repository is also used by our docker scripts and in the VirtualBox images. Hint: The dynamic components in this repository are not necessarily up-to-date.

Link	Version
Minimal Repository on GitHub	Latest
Minimal Repository as zip-file	Latest
Larger Example Repository on GitHub	Latest
Larger Example Repository as zip-file	Latest

Download Frontend (Website & SQL DB Structure)

Link	Version
Website on GitHub	Latest
Website as zip-file	Latest

Install Website

Now you can change to the root directory of the ClustEval website (the directory which contains the Gemfile - let's call it <WEBROOT>) within the directory into which you cloned our clustevalWebsite GitHub repository earlier and then execute:

```
bundle install
```

This will now install all dependencies of our app and make it ready to be executed. This command will take some time.

Now we need to initialize the database. To that end you need to create a valid database configuration in <WEBROOT>/config/database.yml. You can use the template file <WEBROOT>/config/database.yml.example and adapt it.

If you have installed postgreSQL on your local machine, it is running on port 5432 and you have created a user called **clusteval** with a password **clusteval**, you should set the file as follows:

```
development:
  adapter: postgresql
  database: clusteval
  pool: 5
  timeout: 5000
  username: clusteval
  password: clusteval
  host: localhost
  port: 5432

production:
  adapter: postgresql
  database: clusteval
  pool: 5
  timeout: 5000
  username: clusteval
  password: clusteval
  host: localhost
  port: 5432
```

Now we can initialize the database by executing:

```
rake db:migrate && rake db:seed
```

and when it is finished you can start the ClustEval website with:

```
rails server
```

This will make the website available at http://localhost:3000

CHAPTER 2

The Framework

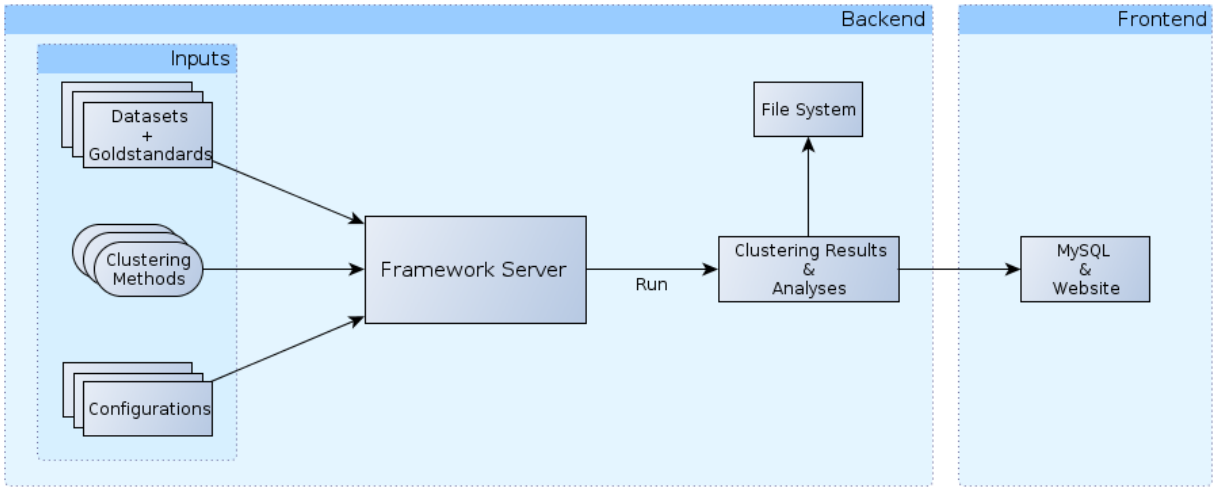


Fig. 1: A general process diagram of ClustEval

Our framework clusteval is intended to perform automatized cluster analysis of arbitrary data sets and clustering methods. The goal is, that any clustering method known to the framework can be applied to any known data set (with certain exceptions, partly imposed by the employed clustering methods and partly by ClustEval).

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.

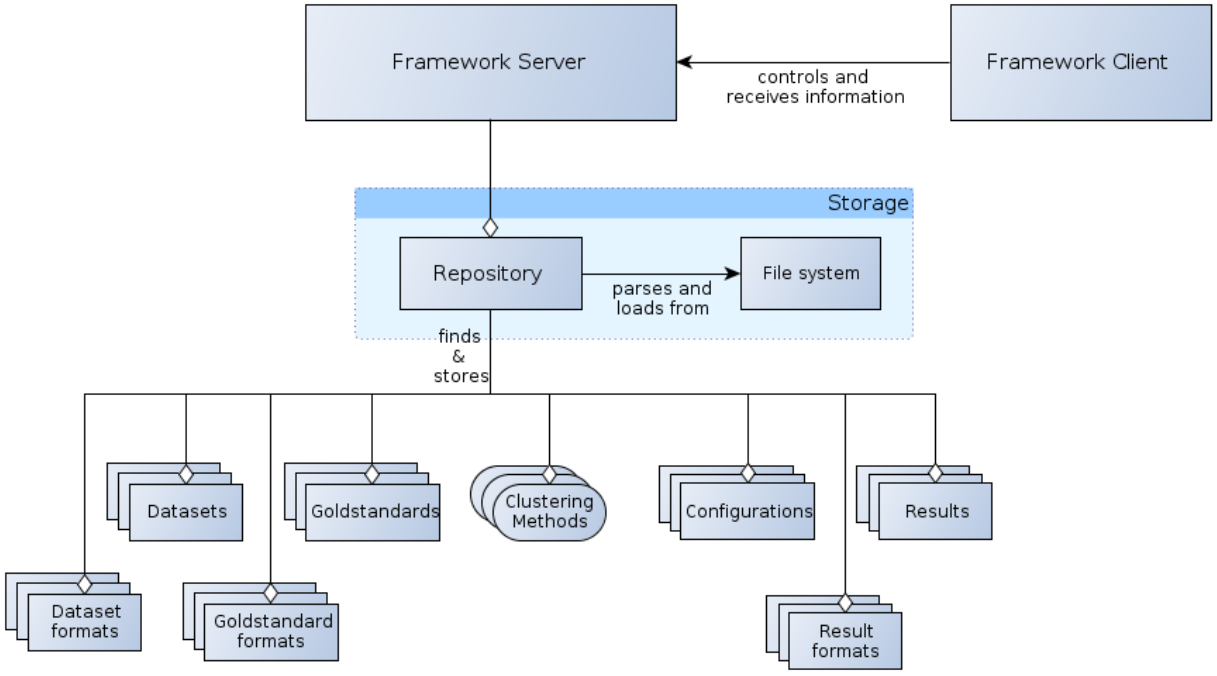


Fig. 2: A schema of the ClustEval backend

2.1 ClustEval Server

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.

In principle the core of all tasks of the backend server is, to apply clustering methods to data (data set 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

2.1.1 Data Sets

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

2.1.2 Gold Standards

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.

2.1.3 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),
- 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.

Standalone

Standalone programs come as an executable that 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).

R Programs (Rserve)

ClustEval supports the execution of R implementations. 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.

Available Clustering Methods

ClustEval ships with a set of clustering methods:

Affinity Propagation

Affinity propagation performs the clustering by identifying exemplars (also called prototypes) among the available data points and reports their neighborhoods as clusters. It considers all data points as possible exemplars. The final set of exemplars is determined by sending messages between the data points voting for the best set of exemplars for the given similarity function in combination with the preference.

DBSCAN

DBSCAN regards clusters of objects as dense regions that are separated by regions of low density. A certain minimum number of objects from each cluster is required to be within a certain distance. The number and the distance are user-given. Singletons are considered noise.

clusterONE

Clustering with Overlapping Neighborhood Expansion (ClusterONE) is a recently developed clustering method designed for detecting overlapping protein complexes in a given protein-protein interaction network. The primary concept behind the algorithm is the so-called cohesiveness measure, which is a combination of the number of reliable interactions within a cluster and the separation of the cluster from the rest of the network. We may apply ClusterONE to arbitrary data sets by transforming a given similarity matrix into a weighted similarity graph.

clusterdp

Clusterdp is a recently published tool based on the idea that centroids are characterized by a higher local density (rho) than their neighbors and by a comparably high distance from objects with higher density. Rodriguez and Laio claim that the number of clusters arises intuitively and that clusters are recognized regardless of their shape and dimensionality of the space in which they are embedded.

Fanny

Fanny is a fuzzy clustering method requiring a distance matrix and the desired number of clusters as input. Fanny aims to find for each object the best membership vector describing the degree of membership of each object to each cluster by minimizing the sum of the average within-cluster distances weighted by the degree of the membership. This objective function is minimized numerically and the nearest crisp-clustering is reported as result.

Hierarchical Clustering

Hierarchical clustering creates a tree from a given set of objects. Each node in the tree represents a cluster. Agglomerative algorithms initially assign all objects to singleton clusters, which are merged based on their similarity (bottom-up). Divisive algorithms start with one big cluster, which is decomposed into smaller clusters along the tree (top-down). A linkage function determines the distance between two clusters. As linkage function, we use the Unweighted Pair Group Method using arithmetic Averages (known as UPGMA or average linking), which is probably the most popular algorithm for hierarchical clustering in computational biology.

K-means

K-means might be the most well-known clustering algorithm. Using the number of clusters k as parameter, k-means aims to iteratively minimize the within-cluster-sum-of-squares by assigning each object to the closest centroid (an artificial object) followed by a subsequent update of these centroids. The assignment after the convergence of this process is reported as clusters.

MCODE

Molecular COmplex DEtection (MCODE) is a graph based clustering method. In a first step the 'cliquishness'-score of all nodes is measured by taking into account the number of vertices and edges in the direct neighborhood. In a second step, clusters are built by starting at the vertex with the highest score followed by a recursive growing of these clusters by adding all neighboring nodes exceeding a certain score. MCODE may become extremely slow on large connected components in the similarity graph.

Markov Clustering

In Markov Clustering the input similarities are interpret as a graph on which densely connected areas are identified by simulating random walks. The clustering processes itself is performed by repeatedly alternating so-called expansion steps and inflation steps. The expansion steps allow simulated walks of long distances while the inflation steps sharpen the emerging local cluster structure.

k-Medoids (PAM)

Partitioning Around Medoids (PAM) is one of the most common realizations of the k-medoid clustering approach, which uses exemplars as cluster centers. The algorithm works identical to k-means except that the exemplars are updated by testing each object in each cluster as exemplar instead of calculating artificial mean objects (centroids).

Self Organizing Maps

Self Organizing Maps (SOMs) involve training a neural network, where each neuron represents one centroid. The number of centroids (or the grid size) is a user-given parameter reflecting the number of clusters. The SOM aims at finding a set of neurons/centroids and to assign each object to the neuron that approximates that object best while iteratively imposing a topographic ordering on the neurons by influencing neurons that are close by. The output is a set of neurons that implicate clusters (objects are assigned to the closest neuron/centroid).

Spectral Clustering

Spectral clustering computes the second largest eigenvalue of the Laplacian of the similarity matrix to decide where to partition the similarity matrix. The resulting clusters may then be re-partitioned in order to generate a hierarchy. Efficient numeric algorithms solve the underlying linear algebra problems to facilitate spectral clustering on large datasets.

Transitivity Clustering

Transitivity Clustering is based on the weighted transitive graph projection problem. A given similarity matrix is interpret as weighted graph and transformed into a so-called cost graph by removing edges weighted below a user-given threshold. Such a potentially intransitive graph is transformed into a transitive graph afterwards by adding

and removing a minimal number of edges. Edge weights are taken into account by using a similarity-dependent cost function (the distance of the edge weight to the cutoff), which is to be minimized. A combination of exact and heuristic algorithms tackle this NP-hard problem. The number of clusters is influenced by the user threshold. The method guarantees that the mean similarity of objects within the same clusters is above the threshold, while the mean similarity between object from different clusters is below.

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.

Extending ClustEval

How you can extend ClustEval with your own clustering methods is described under *Clustering Methods*.

2.1.4 Configuration Files - Handling ClustEval

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 filesys- tem. Internally the framework has some abstraction layers to store all the configura- tions. 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 after- wards 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 ref- erenced program configurations are applied to all datasets indirectly referenced through the data configurations.

Data Configurations

A data configuration is a file, that combines two other configurations together: A dataset configuration and a goldstandard configuration. When you create a run 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.

See `DataConfigParser.parseFromFile(File)` for options that are respected when the file is parsed.

Example

A data configuration could look as follows:

```
datasetConfig = astral_1
goldstandardConfig = astral_1_161
```

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.

See `DataSetConfigParser.parseFromFile(File)` for options that are respected when the file is parsed.

Example

A dataset configuration could look as follows:

```
datasetName = astral_1_161
datasetFile = blastResults.txt
datasetFormat = BLASTDataSetFormat
distanceMeasureAbsoluteToRelative = EuclidianDistanceMeasure
```

GoldStandard Configuration

A goldstandard configuration tells the framework meta information about the corre- sponding goldstandard. That is: The internal name of the goldstandard and its file- name.

See `GoldStandardConfigParser.parseFromFile(File)` for options that are respected when the file is parsed.

Example

A goldstandard configuration could look as follows:

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

Program Configuration

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.

See `ProgramConfigParser.parseFromFile(File)` for options that are respected when the file is parsed.

Example

A program configuration could look as follows:

```
program = APcluster
parameters = preference,maxits,convits,dampfact
optimizationParameters = preference,maxits,convits,dampfact
executable = apcluster
compatibleDataSetFormats = APRowSimDataSetFormat
outputFormat = APRunResultFormat

[invocationFormat]
invocationFormat = %e %i %preference %o maxits=%maxits convits=%convits
dampfact=%dampfact

[maxits]
```

(continues on next page)

(continued from previous page)

```
desc = Max iterations
type = 1
def = 2000
minValue = 2000
maxValue = 5000

[convits]
desc = Cluster Center duration
type = 1
def = 200
minValue = 200
maxValue = 500

[dampfact]
type = 2
def = 0.9
minValue = 0.7
maxValue = 0.99

[preference]
desc = Preference
type = 2
def = 0.5
minValue = 0.0
maxValue = 1.0
```

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.

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

See `RunParser.parseFromFile(File)` for options that are respected when the file is parsed.

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.

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

See `ExecutionRunParser.parseFromFile(File)` for options that are respected when the file is parsed.

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.

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

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.

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:

See `ParameterOptimizationRunParser.parseFromFile(File)` for options that are respected when the file is parsed.

Example

A parameter optimization run could look as follows:

```
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
```

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.

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.

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

See `DataAnalysisRunParser.parseFromFile(File)` for options that are respected when the file is parsed.

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.

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

See `RunAnalysisRunParser.parseFromFile(File)` for options that are respected when the file is parsed.

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.

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

See `RunDataAnalysisRunParser.parseFromFile(File)` for options that are respected when the file is parsed.

Robustness Analysis Run

A robustness analysis run can be used to measure the effect of changes in clustering performances of methods on data sets. The run first parses the best performances and corresponding parameter sets of clustering methods on data sets in a set of run results. Next, these original data configurations are distorted and clustering methods are executed on these with the same parameter sets that originally lead to the best performances.

The randomizers are parameterized, i.e. they have parameters that will lead to different distorted data configurations. One robustness run will generate a certain number of distorted data configurations for each randomizer parameter set. The robustness analysis run file provides the following options:

See `RobustnessAnalysisRunParser.parseFromFile(File)` for options that are respected when the file is parsed.

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

2.1.5 Formats - Input & Output

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

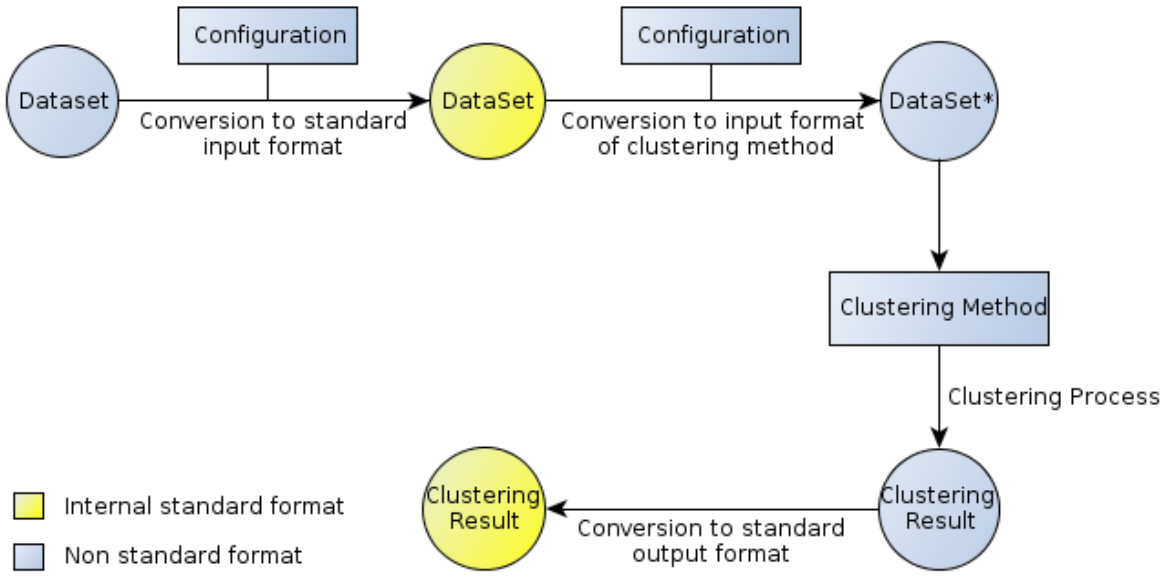


Fig. 3: A schema of the format conversion processes in ClustEval

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.

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

Available Formats

For lists of all available input and result formats see here and here respectively.

Standard Formats

The standard input format is the SimMatrixDataSetFormat. The standard result format is the TabSeparatedRunResultFormat.

Providing New Formats

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 extend ClustEval by new formats see *Formats*.

2.1.6 Clustering Quality Measures - Is It a Good Clustering?

Clustering quality measures assess the quality of calculated clusterings.

ClustEval ships with a standard set of clustering quality measures. For a list of available clustering quality measures see here

Check *Writing Clustering Quality Measures* for more information on how to extend ClustEval by new clustering quality measures.

2.1.7 Distance Measures - Converting Absolute Coordinates to Pairwise Similarities

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 a standard set of distance measures. For a list of available clustering quality measures see here

Check *Writing Distance Measures* for more information on how to extend the framework by new distance measures.

2.1.8 Parameter Optimization Methods - Finding Good Clustering Parameters

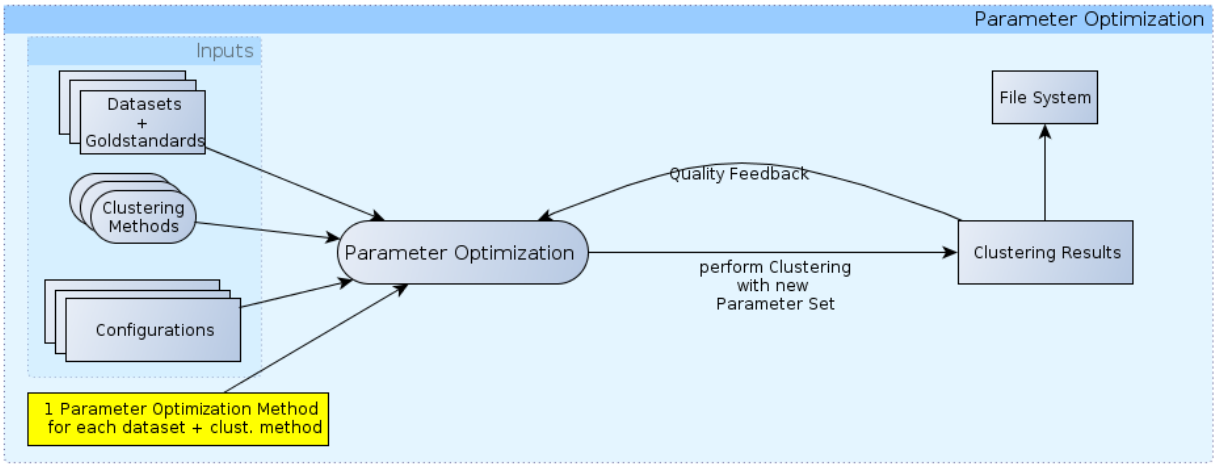


Fig. 4: A schema of parameter optimization in ClustEval

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

Available Parameter Optimization Methods

ClustEval ships with a standard set of parameter optimization methods. You can find a list here

Extending ClustEval

Check *Writing Parameter Optimization Methods* for more information on how to extend the framework by new parameter optimization methods.

2.1.9 Data Set Types - Classifying Data

Each data set is required to have a type. This type is used by ClustEval to group data sets of similar types when representing results.

For a list of all data set types head over here.

Check *Writing Data Set Types* for more information on how to extend ClustEval by types.

2.1.10 Statistics - Analyzing Data & Clusterings

Analysis runs are analyzing certain aspects, called statistics, of the data set or clustering of interest.

Depending on what is being analyzed ClustEval distinguishes between different sub types of analysis runs and statistics:

- **Data analysis run / Data statistics** are being used to analyze a data set
- **Run analysis run / Run statistics** are being used to analyze results of a run, for example clustering performances of methods.
- **Run-data analysis run / Run-data statistics** are being used to analyze results of a data analysis run together with results of a run analysis run. An example is, to train a linear regression models on data statistics and clustering method performances of a data set.

2.1.11 Data Set Generators - Validation On Artificial Data

ClustEval can generate new data sets using archetypes of data sets, so called data set generators.

Each generator exposes a certain set of parameters that can be set by the user and that affect how the instance of the data set is generated in detail. For instance, each data set generator has to have a parameter for the number of objects that the resulting data set should contain. Another parameter, that is contained in some generators is the dimensionality of the data set.

Depending on the generator implementation, generators can also generate gold standards corresponding to a generated data set.

Furthermore, a generator should create appropriate configuration files for the generated data set and optionally the gold standard.

For a list of all available generators head over here.

Check *Writing Data Set Generators* for more information on how to extend ClustEval by generators.

2.1.12 Data Preprocessors - Cleaning Data and Removing Noise

Data preprocessing describes the process in which a noisy data set is processed into a cleaned data set by transforming its data features. There are different kinds of noise, two prominent examples are biological noise and technical noise. Both affect how precisely a data set reflects the true information. Biological noise describes the fact that biological systems exhibit variation which may be interpreted falsely as signal. Technical noise is introduced into the data set during its measurement process by technical confounding factors.

ClustEval provides means to clean data sets from noise. For a list of all available data preprocessors head over here.

Check *Writing Data Preprocessors* for more information on how to extend ClustEval by new data preprocessors.

2.1.13 Data Randomizer - Distorting Existing Data Sets

ClustEval provides means to distort data sets. For a list of all available data randomizers head over here.

Check *Writing Data Randomizers* for more information on how to extend ClustEval by new data randomizer.

2.1.14 Repository

All these components have to be located in the repository of the framework (see repository 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. Data sets, clustering methods or configurations outside the repository cannot be used by the framework. When 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.

More information about the above components can be found in the following sections:

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:

- <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 files, one for the complete run and one for every iteration performed during the run.

Execution Run Results

Execution run result folders additionally contain the following subfolders:

- <REPOSITORY ROOT>/results/<runIdentifier>/clusters

Analysis Run Results

Analysis run result folders additionally contain the following subfolders:

- <REPOSITORY ROOT>/results/<runIdentifier>/analyses

2.1.15 CLI (Command line interface)

The server provides the following command line parameters:

–absRepoPath <ABSOLUTEFOLDERPATH> The absolute path to the repository

–help Print help and usage information

–version Print the version of the server

–port <INTEGER> The port this server should listen on for clients

–logLevel <INTEGER> The verbosity this server should use during the logging process. 0=ALL, 1=TRACE, 2=DEBUG, 3=INFO, 4=WARN, 5=ERROR, 6=OFF

–numberOfThreads <INTEGER> The maximal number of threads that should be created in parallel when executing runs.

–checkForRunResults <BOOLEAN> Indicates, whether this server should check for run results in its repository.

–noDatabase <BOOLEAN> Indicates, whether this server should connect to a database.

–rServeHost <IPADDRESS|STRING> The address on which Rserve is listening.

–rServePort <INTEGER> The port on which Rserve is listening.

2.2 ClustEval Client

The backend client is a small command-line tool which gives commands to the back- end 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.

2.2.1 CLI (Command line interface)

The client provides the following command line parameters:

–help Print help and usage information

–logLevel <INTEGER> The verbosity this server should use during the logging process. 0=ALL, 1=TRACE, 2=DEBUG, 3=INFO, 4=WARN, 5=ERROR, 6=OFF

–version Print the version of the server

–ip <IPADDRESS> The ip address of the backend server to connect to.

–port <INTEGER> The port number of the backend server to connect to.

–clientId <INTEGER> The client id for identification purposes of this client with the server. Each connecting client has its unique id and can only retrieve results that have been executed by the same client id.

–performRun <RUNNAME> Performs a certain run (if not already running)

–resumeRun <RESULTID> Resumes a certain run that was started and terminated earlier

–terminateRun <RESULTID> Terminates a certain run that was started earlier.

–getDataSets Queries the available datasets from the server.

–getPrograms Queries the available programs from the server.

–getRuns Queries the available runs from the server.

–getRunResumes Queries the available run resumes from the server.

–getRunStatus <RESULTID> Queries the status of a certain run

–getOptRunStatus <RESULTID> Queries the optimization status of a certain run

–getQueue Gets the enqueued runs and run resumes of the backend server

–getActiveThreads Gets the currently active threads and the corresponding iterations which they perform

–setThreadNumber <INTEGER> Sets the maximal number of parallel threads.

–generateDataSet <GENERATORNAME> Generates a dataset using the generator with the given name. Usage help will be shown after this command has been executed.

–randomizeDataConfig <RANDOMIZERNAME> Randomizes a dataconfig using the randomizer with the given name. Usage help will be shown after this command has been executed.

–shutdown Shut down the ClustEval server.

–waitForRuns The client will wait until all runs that this client started are finished.

2.3 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 the following. Here we describe, 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.

2.3.1 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.

2.3.2 Data Sets

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.

2.3.3 Clustering Methods

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

- The 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.

2.3.4 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.

2.3.5 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.

2.3.6 Advanced

The advanced 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.

2.3.7 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.

2.3.8 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.

2.3.9 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.

2.3.10 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.

2.3.11 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.

2.4 Database

The SQL 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.

2.4.1 Supported SQL derivates

Currently, the ClustEval backend supports MySQL and postgreSQL. The website uses postgreSQL by default because of its support for materialized views. See repository_config for an explanation how to configure a database for a repository.

2.4.2 Tables

In the following we will give a short description of every table of the database.

- 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".

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).

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).

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).

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).

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

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).

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).

dataset_descriptions Holds descriptions of each data set. Is seeded by the seeds.db file of the Rails app.

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

dataset_images Contains the name of the image correspond to a data set, if it has one.

dataset_publications Contains the references of data sets, if available.

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.

dataset_visibilities Determines whether a data set is visible on the website (by default).

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.

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.

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

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.

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

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.

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.

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).

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).

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.

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).

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.

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

run_analysis_statistics Every analysis run (run analysis id, FK) evaluates certain statistics (statistic id, FK).

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

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

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.

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).

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.

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).

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.

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

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

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.

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.

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.

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

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

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.

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).

run_results_clustering_qualities Holds the qualities for each program config, data config, parameter set of a particular run.

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).

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).

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).

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).

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).

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).

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.

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).

run_results_parameter_optimizations_parameter_values This table contains the value of a certain program parameter (run results parameter optimizations parame FK) evaluated in a certain parameter optimization iteration (run results parameter optimizations parame FK).

run_results_parameter_optimizations_qualities In every iteration of a parameter optimization a parameter set is evaluated and clus- tering qualities are assessed. This table holds the clustering quality measure (cluster quality measure id, FK) together with the assessed quality.

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).

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).

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

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

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).

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).

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).

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.

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

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.

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

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

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

Technical Tables

The following tables 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.

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

[aboutus_impressums] A technical table containing the information regarding the 'Impressum' section of the website.

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

[help_installations] Models of the help installation section of the website.

[help_publications] Models of the help publications section of the website.

[help_source_codes] Models of the help source code section of the website.

[help_technical_documentations] Models of the help technical documentations section of the website.

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

[impressions] A technical table containing the impressions of the website.

[mains] A technical table containing the information regarding the startpage of the website.

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

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

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

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

[small_rankings] ClustEval makes use of reusable modules in its website; one of them is the small ranking cell, which shows highcharts diagrams.

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

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

[submits] This table corresponds to the "Submit" section of the website.

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

2.4.3 Materialized Views

dataset_statistics Lists all statistics for a particular data set

dataset_recent_statistics Lists all statistics for a particular data set, but only the most recent version of each statistic.

parameter_optimization_data_configs_iterations Lists all parameter optimization iterations that have ever been calculated per data configuration.

parameter_optimization_iterations Lists all parameter optimization iterations.

parameter_optimization_iterations_exts Additionally joins the above information with the corresponding data set and clustering method.

parameter_optimization_iteration_exts_configs Joins the information of *parameter_optimization_iterations* with the corresponding original data configuration and clustering method.

parameter_optimization_iterations_woparam Holds the same information as *parameter_optimization_iterations* but without the parameter set.

parameter_optimization_max_qual_rows Lists those parameter optimization iterations (including all information such as used parameter set) which achieved the highest qualities for a particular quality measure, clustering method and data set.

parameter_optimization_max_quals Lists the highest achieved qualities for a particular quality measure, clustering method and data set (this is useful for quality measures that are best if maximized).

parameter_optimization_min_quals Lists the lowest achieved quality values for a particular quality measure, clustering method and data set (this is useful for quality measures that are best if minimized).

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 an analysis run.

run_results_data_configs_rankings Holds the best performances of all clustering methods and clustering quality measures for one particular data configuration.

run_results_program_configs_rankings Holds the best performances of one particular program configuration for all clustering quality measures and data configurations.

2.5 Web Administration Interface

CHAPTER 3

Extending ClustEval

ClustEval can be extended in many ways. Please head over to *Extending ClustEval* to get more information.

3.1 Extending ClustEval

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

3.1.1 Clustering Methods

As explained in *Clustering Methods* clusteval supports two different kinds of clustering methods: Stan- dalone 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.

Adding Standalone Programs

See the API documentation of `de.clusteval.program.StandaloneProgram` on information about how to extend ClustEval.

Writing R Programs

See the API documentation of `de.clusteval.program.r.RProgram` on information about how to extend ClustEval.

3.1.2 Writing Data Set Types

See the API documentation of `de.clusteval.data.dataset.type.DataSetType` on information about how to extend ClustEval.

3.1.3 Formats

Writing Data Set Formats

See the API documentation of `de.clusteval.data.dataset.format.DataSetFormat` on information about how to extend ClustEval.

Writing Run Result Formats

See the API documentation of `de.clusteval.run.runresult.format.RunResultFormat` on information about how to extend ClustEval.

3.1.4 Writing Parameter Optimization Methods

See the API documentation of `de.clusteval.cluster.paramOptimization.ParameterOptimizationMethod` on information about how to extend ClustEval.

3.1.5 Writing Distance Measures

See the API documentation of `de.clusteval.data.distance.DistanceMeasure` on information about how to extend ClustEval.

3.1.6 Writing Clustering Quality Measures

See the API documentation of `de.clusteval.cluster.quality.ClusteringQualityMeasure` on information about how to extend ClustEval.

3.1.7 Writing Statistics

ClustEval can analyze properties of clusterings, data sets and relationship between the two. We call such properties run, data and run-data statistics respectively.

For more information about how to extend ClustEval with your own statistics have a look at

- the API documentation of `de.clusteval.run.statistics.RunStatistic`,
- the API documentation of `de.clusteval.data.statistics.DataStatistic`,
- the API documentation of `de.clusteval.run.statistics.RunDataStatistic`.

3.1.8 Writing Data Preprocessors

See the API documentation of `de.clusteval.data.preprocessing.DataPreprocessor` on information about how to extend ClustEval.

3.1.9 Writing Data Randomizers

See the API documentation of `de.clusteval.data.randomizer.DataRandomizer` on information about how to extend ClustEval.

3.1.10 Writing Data Set Generators

See the API documentation of `de.clusteval.data.dataset.generator.DataSetGenerator` on information about how to extend ClustEval.