To enable reproducibility of our analyses and to abstract from the application domains, we implemented in the R SubtypeDiscovery package our data mining scenario to search for homogeneous subtypes in data by cluster analysis. We present the implementation in this chapter.

5.1 Introduction

We previously introduced a data mining scenario to facilitate and enhance the task of discovering subtypes in data. For different application domains, we also presented some results of our subtyping analyses. In order to enable reproducibility of our analyses, we decided to make our scenario available as an R package. In this chapter, we present its implementation.

The R project for statistical computing is an initiative to provide as public domain software, an integrated suite of software facilities for data manipulation, calculation and graphical display [rla08]. R refers both to the computing environment and to the R language. In itself, R is very alike to the S environment and language. However, R is a public domain software under the GNU General Public Licence and it can be installed on a number of different operating systems such as Windows, MacOS and Unix.

Our data mining scenario consists of five sequential steps: the data preparation, the cluster modeling, the model selection, the characterization of the subtypes, their comparison and their evaluation. In Figure 5.1, we present the implementation of our scenario as the R SubtypeDiscovery package. It involves three classes: the dataset container (cdata), the cluster model (cmodel) and
the set of cluster results (cresult). This last structure stores the outcomes of the SubtypeDiscovery analyses: a dataset cdata that takes as input the data and some settings defining the way the data should be prepared and interpreted, along with the different models (cmodel) calculated while repeating the cluster analysis.

Figure 5.1: We illustrate the main three classes of our package: the data cdata, the cluster model cmodel and the set of cluster results cresult. In particular, the dataset preparation (cdata) uses set_cdata() which takes as input the raw data and the settings that describe the data transformation.

The outline of this chapter is as follows. We start by presenting the design of the implementation: the data preparation methods, the dataset class, the cluster result class, and the methods to characterize, compare and evaluate the cluster results. Next, we show two pieces of code that perform a typical SubtypeDiscovery analysis.

5.2 Design of the scenario implementation

We start by explaining the design of our implementation for preparing data. Then, we describe the constructor methods, their helper-methods, and the generic functions for both the cdata and the cresult classes. Finally, we discuss the methods to characterize, compare and evaluate the subtypes.
5.2.1 Methods for data preparation and data specific settings

As Figure 5.2 illustrates, the data preparation can be described in terms of five sequential steps. The first two steps aim to define a settings configuration file that tells how the data should be prepared; this is the stage of user-interaction. Next, as the cdata constructor is called (set_cdata), a method (transform_cdata) will parse the fun_transform column of the settings file. It enables to call the appropriate methods that will perform the data transformation. In fact, these transformations are defined in terms of two elements: the modeling procedure (e.g. the mean or the standard deviation) and its operating mode on the data (e.g. to subtract or to divide). Therefore, there is a low level method (transform_ALL) that takes as parameter these two elements along with the data and it will process the data accordingly.

In the following, we describe the different methods.

![Diagram showing the flow of operation to prepare the data]

<table>
<thead>
<tr>
<th>group</th>
<th>in_canalysis</th>
<th>fun_transform</th>
<th>visu_groups</th>
<th>visu_ycoord</th>
</tr>
</thead>
<tbody>
<tr>
<td>var.1</td>
<td>&quot;x&quot;</td>
<td>transform_AVG(transform_SIGMA transform_adjust(&quot;var.1-var.x&quot;))</td>
<td>&quot;x&quot;</td>
<td>1</td>
</tr>
<tr>
<td>var.2</td>
<td>&quot;y&quot;</td>
<td>transform_AVG(transform_SIGMA transform_adjust(&quot;var.2-log(var.y)&quot;))</td>
<td>&quot;y&quot;</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>var.n</td>
<td>&quot;z&quot;</td>
<td>transform_AVG(transform_SIGMA)</td>
<td>&quot;z&quot;</td>
<td>10</td>
</tr>
</tbody>
</table>

Figure 5.2: Flow of operation to prepare the data.

First, the user can rely on a helper-method to generate a settings matrix with default values (generate_cdata_settings). That matrix is necessary for a SubtypeDiscovery analysis because it is where the user describes how the data should be prepared, which variable should be involved in the cluster modeling (in_canalysis), how the variables should be represented graphically (e.g. in the parallel coordinate plots and the heatmaps) and characterized statistically (in the
log of the odds). Typically, the user will prepare the settings matrix within a spreadsheet editor. The cdata objects are datasets for SubtypeDiscovery analyses, they embed this configuration file within the data structure in the settings argument.

Upon construction of a dataset (cdata) for a SubtypeDiscovery analysis, the method transform_cdata is called. It parses the column fun_transform of the settings matrix sequentially from left to right and from top to bottom. Lower level methods are then called to process the data (e.g. transform_AVG, transform_SIGMA, etc.). The parsing uses commas and spaces first, and second, the parenthesis because the method transform_adjust can take as parameter a linear model formula in parenthesis. When data-preparing, we store in tdata structures the computed models and estimates. This enables to transform additional data given previous models and estimates.

To remove for each variable the variability explained by a given factor, e.g. the time, the method transform_adjust is used. It outputs a transformed vector and a tdata storing the regression model and the estimates. With a similar output, the methods transform_ABSMAX, transform_L1, transform_L2, transform_MAX, transform_SIGMA (respectively transform_AVG, transform_MEDIAN, transform_MIN) can normalize (resp. center) the values of the variables.

At the lowest level, the processing of the data is done by transform_ALL. Given two operations, it will either estimate the relevant statistics (e.g. the mean or the median) and use these to transform the data (e.g. by substracting or dividing), or use a previously computed estimate (in tdata) and apply it to the data. The method returns a transformed vector and a tdata structure where the estimators and the models are stored.

### 5.2.2 The dataset class (cdata) and its generic methods

In Figure 5.4, we illustrate the construction via the method set_cdata of a cdata structure that will contain the dataset of a SubtypeDiscovery analysis. First, the constructor copies the original dataset into data_o, second, it applies the initialization and filtering methods and next, it processes the data by the method transform_cdata given the settings matrix. In the following, we describe the default initialization procedure and the generic plotting method of a cdata structure.

In the initialization, the method init_data_cc is called. Via the column in_canal_analysis of the settings matrix, it limits the SubtypeDiscovery analysis to the observations showing complete records on the variables selected for the cluster modeling. This is because model based clustering can not process datasets having missing values. However, by subsetting on in_canal_analysis, we do not drop unnecessarily observations having missing values on variables not in the cluster analysis.
5.2. Design of the scenario implementation

Figure 5.3: A cdata stores information about the dataset for a subtype discovery analysis. This diagram illustrates the different methods to construct and access a cdata structure.

Finally, the method plot.cdata reports boxplots and histograms for each of the variables of the dataset. Besides the histograms, additional information is reported as text over the estimators of the transformations (e.g. the mean or the standard deviation).

5.2.3 The cluster result class (cresult) and its generic methods

In Figure 5.1, we show a cresult data structure that can be constructed by the method set.cresult. It contains all the information of a SubtypeDiscovery analysis, i.e. the data (cdata), the experimental settings, all the models of the repeated cluster analyses (cmodel) and statistics such as the set of Bayesian Information Criterion (BIC) scores (bicanalysis). In the following paragraphs, we first describe the parameters of the constructor and then we discuss, together with their helper-methods, the generic methods associated to the cresult data structure.

First, the constructor takes as parameter a dataset (cdata), a cluster modeling method (cfun) that can be parameterized via (cfun.params), a set of methods to characterize and evaluate both graphically (fun.plot) and statistically (fun.stats) the subtypes, a number that indicates how many top-ranking
models we consider for the cross-comparison (\texttt{nbr\_top\_models}) and some methods (e.g. the mean or some other quantile statistics) to summarize the subtypes (\texttt{fun\_pattern}) or the BIC scores (\texttt{fun\_bic\_pattern}).

![Diagram of the \texttt{cresult} structure]

**Figure 5.4:** A \texttt{cresult} stores all the data computed in the course of a Subtype Discovery analysis: the dataset \texttt{cdata}, the set of cluster models \texttt{cmodel}, the analysis settings, and some additional operating system/computing environment information \texttt{rinfo}. This diagram illustrates the different methods to construct and access a \texttt{cresult} structure.

Next, we implemented a generic plotting method (\texttt{plot.cresult}) to report the visual characteristics of a \texttt{cresult} object. A first parameter (\texttt{device}) enables to define whether the graphical output should be redirected to a postscript file or a series of png pictures. A second parameter (\texttt{query}) can limit the plotting to one particular model among those listed by the command \texttt{names(a\_cresult)}.

We also implemented the method \texttt{print.cresult} that can describe a \texttt{cresult} by a series of tables aggregating the BIC scores in a number of ways, e.g. by various rankings and some summary statistics such as the mean or a quantile statistics. It also reports tables where the top-ranking models are cross-compared.

Finally, the method \texttt{get\_plot\_fun} returns a function that, upon execution with a \texttt{cdata} argument will return another \texttt{unexecuted} plotting function. This new function takes a \texttt{cmodel} as parameter. In fact, by storing the unevalu-
ated plotting functions within the `cresult`, we can redraw independently the plots. Yet, in the course of a regular `SubtypeDiscovery` analysis via the method `analysis`, all cluster models are by default graphically characterized.

### 5.2.4 Statistical methods to characterize, compare and evaluate subtypes

In the following, we present procedures that report statistical measures and summaries of cluster models. Some of these procedures are unevaluated function calls taking as argument a cluster model (`cmodel`); they are retrieved by the helper-method (`get_fun_stats`) when the constructor `set_cresult` is called. Then, we present additional methods that can calculate statistical patterns on the subtypes (`fun_pattern`) or the BIC scores (`fun_bic_pattern`). Last, we present the method that enables to cross-compare cluster results.

**Statistical characterization of subtypes**  We first implemented `stats_logodds` that can calculate summary statistics of the subtypes based on an odd-ratio statistics. This method enables to identify the main characteristics of the subtypes on a number of factors; the factors are used to calculate sum-scores on groups of variables (`group`) defined by the user in the `settings` matrix. In practice, the odd ratios are calculated by comparing the distribution of the sum-scores in the cluster with the one in the whole dataset.

We also implemented a method (`stats_auuc`) to summarize the average level of uncertainty to cluster the observations for the current model. We refer to this average as the area under the curve of the clustering uncertainty (`auuc`).

Finally, in order to assess the reproducibility of the clustering result, we prepared a method (`stats_generalization`) to evaluate the classification accuracy of different machine learning algorithms trained on the clustered data. This procedure repeats the training of classifiers a number of times (by default 10) given a random training-test split (stratified) with 70% of the observations in the training set and 30% in the test set. At this moment, our package features three classification algorithms the naive Bayes, the \(k\) nearest neighbors classifier and the linear Support Vector Machines.

**Statistical evaluation**  Because our research on OA involves a cohort study made of sibling pairs, we implemented two statistical tests that can assess the level of familial aggregation of the cluster models. One test quantifies the risk increases of the second sibling given the characteristics of the proband; it is referred to as the \(\lambda_{sibs}\) risk ratio (`stats_lambdasibs`). The other test counts the pairs of siblings in each cluster and it compares them to the the expected counts if observations were affected randomly to the clusters; the statistic is the one of a \(\chi^2\)-test of goodness of fit.

In drug discovery research, we chose to report the joint distribution between the subtypes and the bioactivity classes in terms of cell counts. Second, in order
to illustrate how unequal the marginals of this distribution are, we report the $\chi^2$ values (i.e. the deviation to the counts expected at random). We are especially interested in the cells that exhibit a high $\chi^2$ value.

**Statistical patterns** To show the characteristic patterns of each subtype, we rely on summary statistics like the 2.5% quantile, the maximum, the mean, the median, the standard deviation and the 97.5% quantile of a numeric vector. These statistics are estimated by methods that discard by default the missing values (patternLowquant, patternMax, patternMean, patternMedian, patternSd, patternUpquant). When the constructor set_cresult is called, there are two parameters that define the different patterns to calculate: first, on each data subtype (fun_pattern) and second, on the set of BIC scores (fun_bic_pattern).

**Cross-comparison of subtypes** In the course of a SubtypeDiscovery analysis, we use the compare_cresult method to draw comparisons between the top-ranking cluster results. However, this method can also be called independently if two cluster models to compare are provided as argument. The function returns a list of tables where the models are cross-compared; there are two kinds of tables, those with the original values and those designed to be visualized (e.g. on a HTML report). The method can also store the original tables into comma separated value files.

### 5.2.5 Other methods

The method analysis defines a sample workflow for subtype discovery. It generates graphics on the data preparation, it performs the cluster analysis, it computes the subtype’s characteristics and evaluate them, and it cross-compares the top-ranking results.

The function fun_mbc_em does a model based clustering on the data (a data matrix), given the model (modelName) and the number of clusters (G). The initialization is particular in that it draws at random a cluster membership probability matrix ($z$) of dimension $(N \times G)$, with $N$ the number of observations. Then, from that vector, it estimates the corresponding model by an M-step which is further used to start EM given the model.

### 5.3 Sample analyses

The purpose of this section is to show two fragments of code to give the reader an idea about the code needed for an analysis. For this purpose, we use a public chemoinformatics dataset that we embed in our package.

The outline of this section is as follows. First, we show the piece of code to conduct a typical SubtypeDiscovery analysis. Yet, as molecular descriptors tend
to correlate highly, we show a second analysis performed on the scores from the principal components; we select the dimensions that explain 95% of the variability.

5.3.1 Analysis on the original scores

In the following, we describe and then report a piece of code to perform a subtyping analysis on the `wada2008` dataset.

First, it is necessary to load the `SubtypeDiscovery` package via the method `library`. Then, we load the `wada2008` dataset and its predefined settings `wada2008_settings` which our package embeds. The next step is to prepare the dataset for a subtyping analysis (cdata). For this purpose, we use the constructor `set_cdata` that takes the dataset, the settings and a short name describing the data. In a similar way, we will prepare a `cresult` using the constructor `set_cresult`; many parameters are set by default. Finally, the subtyping analysis is carried out using the method `analysis`. It will repeat the cluster modeling, perform an analysis on the BIC scores, characterize graphically and statistically the cluster results and save the computed models.

```r
library(SubtypeDiscovery)
# LOAD dataset
data(wada2008)
data(wada2008_settings)
# PREPARE CDATA
cdata1 <- set_cdata(data=wada2008,
  prefix="WADA2008_Sample_Analysis",
  settings=wada2008_settings)
# PREPARE THE SET OF RESULTS FOR CLUSTER modeling
x <- set_cresult(cdata=cdata1,
  fun_stats=list(oddratios=get_fun_stats(
    fun_name="oddratios", fun_midthreshold=mean)),
  nbr_top_models=5,
  cfun=fun_mbc_em,
  cfun_settings=list(
    modelName=c("EII", "VII","EEI","VEI","VVI"),
    G=3:6,
    rseed=6013:6063))
# PROCEED TO THE ANALYSIS:
cresult_set <- analysis(x)
```

5.3.2 Analysis on the principal components

Here, we present a second subtyping analysis on the `wada2008` dataset. The difference with previous analysis is in the preparation of the data because here we decide to repeat the cluster modeling on the principal component dimensions. For
this purpose, we rely on the method `get_cdata_princomp` that will estimate the principal components of the dataset. This method will update the dataset structure `cdata1` such that the subtyping analysis is performed on the principal components of the data. The remainder of the code is the same than in previous analysis.

```r
library(SubtypeDiscovery)
# LOAD dataset
data(wada2008)
data(wada2008_settings)
# PREPARE CDATA
cdata1 <- set_cdata(data=wada2008,
                   prefix="WADA2008_Sample_Analysis",
                   settings=wada2008_settings)
# PREPARE NEW CDATA FOR CANALYSIS ON PRINCOMP
cdata2 <- get_cdata_princomp(cdata1)
# PREPARE THE SET OF RESULTS FOR CLUSTER modeling
x <- set_cresult(cdata=cdata2,
                   fun_stats=list(oddratios=get_fun_stats(
                       fun_name="oddratios", fun_midthreshold=mean)),
                   nbr_top_models=5,
                   cfun=fun_mbc_em,
                   cfun_settings=list(
                       modelName=c("EII", "VII","EEI","VEI","VVI"),
                       G=3:6,
                       rseed=6013:6063),
                   fun_pattern=list(mean=patternMean))
# PROCEED TO THE ANALYSIS:
x <- analysis(x)
```

### 5.4 Concluding remarks

We initially prototyped our subtyping methodology for OA in collaboration with the MOLecular EPIdemiology department (MOLEPI) of the Leiden University Medical Center. Then, following the interest of the Neurology department (LUMC) working on PD, we prepared it as the R SubtypeDiscovery package because this would enable reproducibility and reliability of our analyses. More recently, we collaborated with the Pharma-IT platform of the Leiden University to apply subtyping to the field of drug discovery. As our primary users are from biology, we simplified the scenario’s design until it relied solely on a spreadsheet-like description of the data. This particular effort is to match the user’s demand who, in the end, is expected to carry out his / her own analyses.

Yet, usability of our package could be further improved by constructing a basic graphical front-end. It would certainly reduce the amount of time needed to define
the dataset settings: by limiting the possibility for spelling mistakes, by making
the software features more accessible, by removing the need to exchange between
R and an editor, and by removing the manual initialization of the analysis via a
R script file.

Besides, the software design could be more robust if we used the MLInterfaces
package of BioConductor to determine generalization estimates of the classification
algorithms on the cluster models. Similarly, using the Sweave package would
enable us to separate the report-making process from the data generation. At
this moment we still use our own Machine Learning and report-making methods.

Finally, a more extensive use of R object-oriented programming in our software
design would further increase both the usability, robustness and reliability.

Our R package can be found at:
- http://www.grano-salis.net/SubtypeDiscovery/.