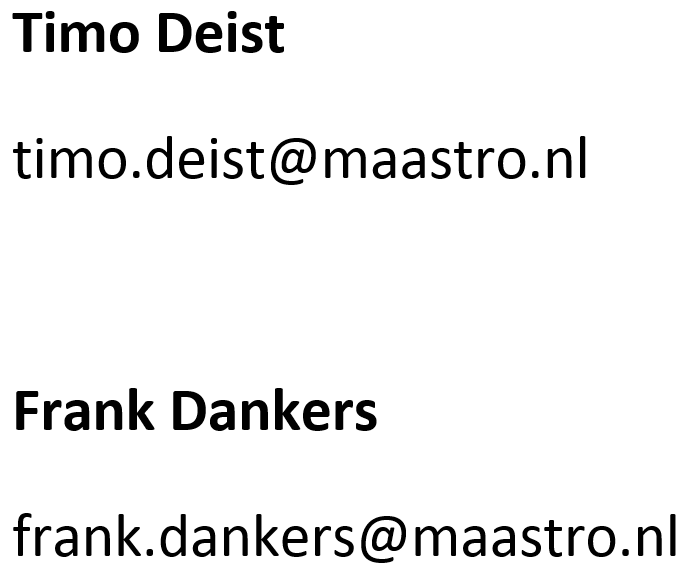
Classifier Review:

Code Base Manual

Code base version 1.0, last updated on November 12, 2017



**MAASTRO clinic**

**Maastricht University Medical Center**

**Department of Radiation Oncology**

www.maastro.nl

P.O. Box 3035

6202 NA Maastricht

The Netherlands

Dr. Tanslaan 12

6229 ET Maastricht

The Netherlands

Tel: 0031 88 44 55 666

Fax: 0031 88 44 55 667

Table of Contents

[1. Introduction 3](#_Toc498276630)

[2. Methodology 3](#_Toc498276631)

[3. Directory structure 4](#_Toc498276632)

[4. Package installation 4](#_Toc498276633)

[5. Perform simulation 5](#_Toc498276634)

[5.1. Datasets 5](#_Toc498276635)

[5.2. Main csv-file 5](#_Toc498276636)

[5.3. Dataset load function 5](#_Toc498276637)

[5.4. Main ini-file 7](#_Toc498276638)

[5.5. Start the run 7](#_Toc498276639)

[6. Output analysis 7](#_Toc498276640)

[References 9](#_Toc498276641)

[Appendix A Classifiers 9](#_Toc498276642)

[Appendix B Required R packages 10](#_Toc498276643)

[Appendix C Scripts 10](#_Toc498276644)

[Appendix D Functions 10](#_Toc498276645)

[Appendix E Code base version management 11](#_Toc498276646)

[Appendix F Poor *svm* performance (*caret* version downgrade) 11](#_Toc498276647)

# Introduction

The purpose of this manual is to explain the workings of the R code base accompanying the manuscript entitled *“Machine learning algorithms for outcome prediction in radiotherapy: advice on selecting a classifier”* [1]. The manuscript provides advice for researchers on selecting machine learning algorithms to perform outcome prediction modelling in the field of radiotherapy. The conclusions drawn in the manuscript are based on numerical simulation of 6 classifiers (Appendix A) implemented in the popular R package *caret* [2], by Kuhn et al. 2016 [3], on a range of multiple datasets covering prediction of survival, and toxicities in lung and head and neck cancer.

The code base on the repository [4] may be helpful for fellow researchers in modelling treatment outcomes. The goal of the code base is to allow comparison of the average discriminating performance of the different classifiers for a given dataset. The code base is not meant for individual model creation.

If you have questions or remarks using the code base please contact one of the authors at the email-addresses listed on the title page of this manual.

All simulations using this code base have been tested on RStudio version 1.0.153 [6] and R version 3.4.1 [6].

If you use this code base in your research or a paper, please refer to the accompanying manuscript as follows [1]: [REF TO PAPER]

# Methodology

The experimental design is depicted in Figure 1. Each dataset is split into 5 stratified folds (step 1). For each of the folds, the data is pre-processed (imputation, dummy coding, deleting zero variance features, rescaling) (step 2). The tuned hyperparameters are determined in the training set via a 5-fold inner CV (steps 3-5). Based on the selected hyperparameters, a model is learned on the training set (step 6) and applied on the test set (step 7). Performance metrics are calculated on the test set (step 8) and stored for all folds. This process is repeated for all the classifiers. Randomization seeds are stable across classifiers within a repetition to allow pairwise comparison of classifier performance. The entire process is repeated 100 times.

The number of repetitions, outer folds and inner folds can be set through the main ini-file (see 5.4 Main ini-file). For additional information regarding the performance metrics, please see the manuscript [1].

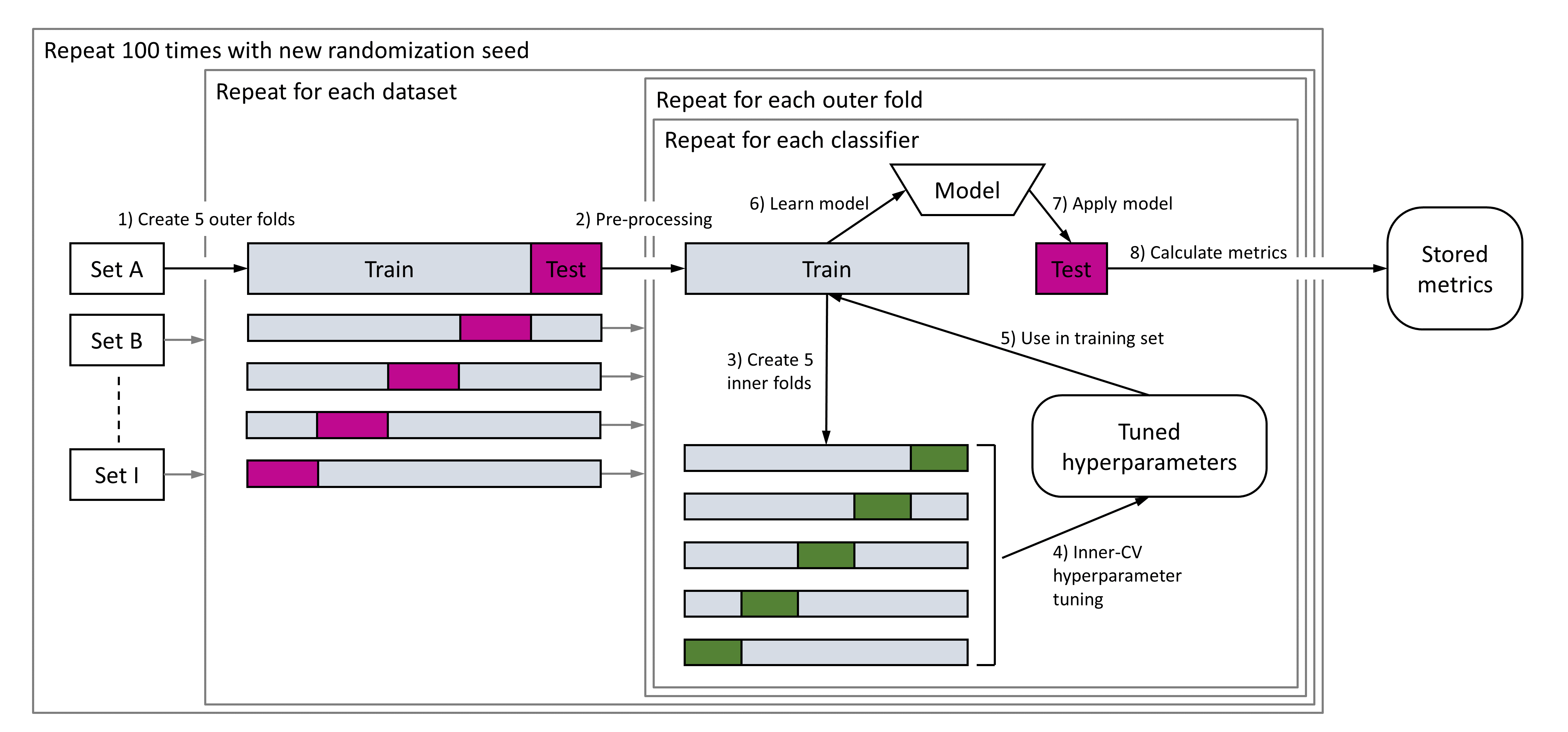


Figure : Schematic overview of the experimental design of the code base.

# Directory structure

The code base requires the presence of three directories listed in the following table. The dot in the directory represents the installation/current directory.

Table : The required directories prior to running the code base are Code, Data and Output.

|  |  |
| --- | --- |
| Directory | Explanation |
| “.\Code\” | Contains the code base downloaded from the repository. |
| “.\Data\” | Contains the dataset(s) that you want to run through this code base.  Each dataset should preferable be placed in a unique subdirectory. An alternative data directory can be set through the main ini-file (see 5.4 Main ini-file). |
| “.\Output\” | Contains the output of a simulation run and the output analysis. Additionally, it will contain copies of the main ini- and csv-files that were used for the simulation.  An alternative output directory can be set through the main ini-file (see 5.4 Main ini-file). |

# Package installation

The code base requires the installation of several packages. Needless to say this includes the installation of the *caret* package, version 6.0-73 [2-3]. Note: if you see deteriorated performance for classifier *svm* please see Appendix F. Additionally, there are packages required for several classifiers, for data imputation, plotting functionality etc. These packages can be installed by running the script named “main\_install\_all\_packages.R”. This script needs to be run only once. A list of required packages is given in Appendix B.

# Perform simulation

## Datasets

Each dataset that you want to include in the simulation should consist of a single dataset csv-file and be stored in a unique subdirectory in the “.\Data\” directory. Patients are given across rows, features and outcome(s) are given across columns. It is advisable to do any data processing via the dataset load function (see 5.3 Dataset load function) and not directly on the dataset csv-file, so that a processing trail remains traceable.

## Main csv-file

Before running the simulation on a dataset it is important to correctly set and verify parameters in the main csv-file (and also the main ini-file, see 5.4 Main ini-file). The name of the main csv-file is “main\_simulation\_datasetDefinitions.csv” and it contains information about the dataset(s) that you want to include in the simulation. You can edit the main csv-file through a text-editor or with Excel. The separator of the csv-file should be a semicolon (;). The repository version of this file contains example entries for publicly available datasets. The first line of the main csv-file always has to read:

datasetName;datasetFoldername;datasetFilename;datasetLoadfunction

There should be subsequent lines per dataset that you want to include. Each line should contain four dataset parameters according to Table 2. An example of a dataset line in the main csv-file could be:

OberijeEtAl;Oberije\_et\_al\_(2015);Stage3\_anonymized.csv;loadDataOberijeEtAl

Table : Overview of entries to fill in inside the main csv-file and per dataset.

|  |  |
| --- | --- |
| Parameter | Explanation |
| datasetName | Name of the dataset, used in output analysis (e.g., plots etc.) |
| datasetFoldername | Name of the dataset subdirectory (located in “.\Data\”) |
| datasetFilename | Name of the dataset csv-file (located in the dataset subdirectory) with .csv-extension |
| datasetLoadfunction | Name of the dataset load function (located in “.\Code\”) without .R-extension |

## Dataset load function

The purpose of the load function is to load in the dataset (a csv-file itself) based on the information that is entered in the main dataset definitions csv-file (see 5.1 Datasets). Each dataset that you want to include in the simulation needs to have its own load function that is to be located in the “.\Code\” directory. You need to generate a load function for your dataset yourself. Example load functions for publicly available datasets are present in the code base on the repository (e.g. “loadDataOberijeEtAl.R”). The name of the load function for a given dataset should correspond with the name that has been set in the main csv-file (see 5.2 Main csv-file) and it should match with the function name inside the file on the first line (e.g. loadDataOberijeEtAl = function(pathFromDataFolderToCsv,pathToDataFolder)).

The easiest way to make sure your load function is working correctly is to enter it via the debugging mode. Make sure to examine the result of your load function to check if the dataset loading has been performed correctly. Some help is provided below.

At the start of the load function it should read the dataset csv-file and define the separator and decimal characters as well as definition(s) for missing values.

# read csv file

pathToFile = file.path(pathToDataFolder,pathFromDataFolderToCsv)

data = read.csv(pathToFile,sep = ";", dec = ".", strip.white=TRUE, na.strings = c("UNK","NA","na"))

Columns to remove can be indicated via standard R code for data frames (e.g. patient ID’s or outcomes you don’t want to model (prevent data leakage!)).

# columns to drop, feature/outcome

drop = c('ID','localcontrol,'5yearsurvival')

data = data[ , !(names(data) %in% drop)]

Alternatively, you can define the columns to keep.

# columns to keep

keep = c('Sex', 'Age', 'T\_stage', 'meandose')

data = data[, keep]

While inside the debug-mode you can expand the data frame in the upper right workspace to see if the feature types (integers, logicals, numericals or factors (i.e. categoricals)) were correctly assigned by the read csv function. If necessary, manually convert features to factors or numericals (make sure the outcome feature is a factor). Note: if forcing into numericals is necessary in your case this might be indicative of a faulty decimal indicator in the earlier *read.csv* function call.

# convert to numericals

toNumericals = c('PTVmindose','PTVmaxdose')

data[,toNumericals] <- sapply(data[, toNumericals], as.numeric)

# convert to factors

toFactors = c('Arm','zhs','siteprimary','histology')

data[toFactors] = lapply(data[toFactors], factor)

The output parameter should be extracted and assigned to a separate variable and removed from the data frame containing the input features. If the output is not yet in binary form, then it should be dichotomized.

# outcome variable

data\_class = data$survival # place outcome in separate variable

data$survival = NULL # remove outcome from data

data\_class = revalue(data\_class, c('0' = 'nonEvent', '1' = 'event')) # label as (non)event

## Main ini-file

In the main ini-file (“main\_simulation\_parameters.ini”) the simulation parameters are set. These include the number of repetitions, the number of outer and inner folds, the classifiers to use and the datasets to include in the simulation. Additionally, you can anonymize the output data (you can also anonymize at the output analysis stage, see 6 Output analysis), set the randomization seed and turn saving of output on or off. The directories for the datasets and the output can be set, and the main csv-filename is defined in the ini-file. If you enter backslashes in the paths for the dataset and output directories (Windows convention) these will be automatically converted to forward slashes (R convention). The last line of the ini-file should be empty to prevent warnings when the ini-file is read during the run.

Finally, you can turn default tuning of classifier hyperparameters on or off. In the accompanying manuscript [1] all simulations are performed using *caret*’s default tuning. Some custom tuning parameters are set for certain classifiers via their respective fit-functions (listed in Appendix A), but we highly recommend to explore these settings yourself if you plan on performing custom tuning.

## Start the run

When all of the information given in paragraphs 5.1 through 5.4 has been considered it is time to start a simulation by running “main\_simulation.R”. At the top of the script, it is important to correctly point the *this.dir* setting to the “.\Code\” directory since this directory is not set via the ini-file.

During the simulation run progress information will be displayed in RStudio’s console window during the run. It will first display the parameters that were read from the main ini-file. Then it will show the results of loading the dataset(s), reporting the number of features, patients and prevalence. Then the actual simulation will start by looping in order of repetitions, datasets, outer folds and classifiers. A timer will be displayed after each repetition to indicate the approximate time to finish the simulation.

Simulation results will be saved in a subdirectory of the “.\Output\” directory when the run is finished. The naming convention of the subdirectory is: “[date]\_[time]\_[computer name]\_[number of repetitions]reps\_[number of dataset]datasets”. After a run it should contain an RData-file with the simulation results. This file is the input of the output analysis script (see 6 Output analysis).

Additionally, the copies of the main-csv and main-ini files stored in the output directory as well, so that the initialization parameters of the run can always be deduced.

# Output analysis

Once a simulation has finished the results can be analyzed through the script “main\_output\_analysis.R”. At the top of the script, it is important to correctly point the *this.dir* setting to the “.\Code\” directory.

An ini-file named “main\_output\_analysis.ini” allows the setting of saving the analysis output, excluding datasets, anonymization and classifier/dataset ordering. Note: that changing classifier or dataset order can lead to slightly different numbers for AUC *rank* and AUC *rank* derived metrics, since tied AUCs are ranked randomly (to not benefit classifiers based on the specified order).

Running the main script will pop-up a file selection dialog window asking for the Rdata-file with the simulation result. Selecting this file will continue the evaluation via a multitude of subfunctions (Appendix D). These functions will perform aggregation of folds, datasets and repetitions, and calculate a variety of performance metrics. A typical analysis takes in the order of 10s of seconds.

The results of the output analysis are presented in tables and figures, which are saved in the same output subdirectory as the RData-file that has been analyzed. Figures include a combined AUC *rank* box and scatterplot of classifiers, heatmaps of AUC and AUC *rank* performance of classifiers per dataset and pairwise comparisons of classifiers. A table is generated with classifier performance by metric, and there are tables related to the classifier selection simulation experiment as described in the manuscript [1].

For a more comprehensive discussion regarding the analysis of the simulation results please refer to the manuscript.

References

1. T.M. Deist, F.J.W.M. Dankers, G. Valdes, R. Wijsman, I. Hsuc, C. Oberije, T. Lustberg, J. Soest, J. Belderbos, M. Kwint, T. Solberg, R. Monshouwer, J. Bussink, A. Dekker, P. Lambin, “Machine learning algorithms for outcome prediction in radiotherapy: advice on selecting a classifier”, Journal., vol. x, no. x, pp. xxx-xxx, xxx. 2017.
2. Link to R package *caret*: <http://topepo.github.io/caret/index.html>
3. M. Kuhn et al., “caret: Classification and Regression Training,” 2016.
4. Link to code base repository: <https://github.com/timodeist/classifier_selection_code.git>
5. Link to R: <https://cran.r-project.org/bin/windows/base/>
6. Link to RStudio: <https://www.rstudio.com/products/rstudio/download/>
7. J. Friedman, T. Hastie, and R. Tibshirani, “Regularization Paths for Generalized Linear Models via Coordinate Descent,” J. Stat. Softw., vol. 33, no. 1, pp. 1–22, 2010.
8. A. Liaw and M. Wiener, “Classification and Regression by randomForest,” R News, vol. 2, no. 3, pp. 18–22, 2002.
9. W. N. Venables and B. D. Ripley, Modern Applied Statistics with S, Fourth. New York: Springer, 2002.
10. A. Karatzoglou, A. Smola, K. Hornik, and A. Zeileis, “kernlab – An S4 Package for Kernel Methods in R,” J. Stat. Softw., vol. 11, no. 9, pp. 1–20, 2004.
11. J. Tuszynski, caTools: Tools: moving window statistics, GIF, Base64, ROC AUC, etc. 2014.
12. T. Therneau, B. Atkinson, and B. Ripley, rpart: Recursive Partitioning and Regression Trees. 2017.

Classifiers

Table : An overview of classifiers that are implemented in the code base. All classifiers have built-in feature selection.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Classifier | *caret* label | Manuscript label | *R* package | Dummy coding | Hyper-parameters |
| Logistic elastic net regression | *glmnet* | *glmnet* | *glmnet* [7] | Yes | , |
| Random forest | *rf* | *rf* | *randomForest* [8] | No |  |
| Single-hidden-layer neural network | *nnet* | *nnet* | *nnet* [9] | No | , |
| Support vector machine with radial basis function kernel | *svmRadial* | *svm* | *kernlab* [10] | Yes | , |
| LogitBoost | *LogitBoost* | *logitboost* | *caTools* [11] | Yes |  |
| Decision tree | *rpart* | *rpart* | *rpart* [12] | No |  |

Required R packages

Table : A list of required R packages prior to running the code base. These packages can be installed by running the script named “main\_install\_all\_packages.R”.

|  |  |
| --- | --- |
| Name | Purpose |
| *caret* | Functions for streamlining different steps of predictive modelling |
| caTools | Classifier LogitBoost |
| ggplot2 | Generating plots, e.g., boxplots, heatmaps |
| glmnet | Classifier generalized logistic elastic net regression |
| imputeMissing | Functions for handling of missing values |
| kernlab | Classifier SVM radial |
| klaR | Model classification and visualization |
| nnet | Classifier nnet |
| plyr | Functions for variable manipulation, e.g., revalue() |
| randomForest | Classifier random forest |
| resourceSelection | Functions for Hosmer-Lemeshow test |
| rpart | Classifier rpart (decision tree) |
| svDialogs | Functions for user interface dialog windows (e.g. file selection) |

Scripts

Table : An overview of the scripts and their purpose.

|  |  |
| --- | --- |
| Names | Purpose |
| main\_install\_all\_packages.R | Main script for package installation (see 4 Package installation). |
| main\_simulation.R | Main script for performing a simulation (see 5.5 Start the run). |
| main\_output\_analysis.R | Main script for simulation output analysis (see 6 Output analysis). |
| script\_check\_rpart\_failures.R | Legacy script for checking the amount of failures using the decision tree classifier. |

Functions

Table : An overview of the functions, grouped by purpose.

|  |  |
| --- | --- |
| Name | Purpose |
| loadData[dataset name] | Custom functions per dataset to perform loading of the dataset csv-files (see 5.3 Dataset load function). |
| runCvForClassifiers.R  preprocess\_dataset.R  preprocess\_imputeDataset.R  preprocess\_removeZeroVarianceColumns.R | Perform outer-fold cross-validation and the dataset pre-processing before simulation run (see 2 Methodology). |
| runClassifier.R  fit[classifier name].R | Performs the fit of current training/test slices for a classifier using *caret* functionality (includes the inner folds required for hyperparameter tuning). |
| generate[x].R  aggregate[x].R  add[x].R | Functions used for analysis of simulation run output. These functions add metrics, aggregate over folds and repetitions and generate data frames for plotting or saving of tables. |

Code base version management

Table : An overview of version history of the repositories’ code base.

|  |  |  |
| --- | --- | --- |
| Version | Date | Change |
| 1.0 | November 12, 2017 | Initial code base version and repository manual (this document) |
|  |  |  |

Poor *svm* performance (*caret* version downgrade)

The manuscript results were generated employing various open-source *R* packages interfaced with the *R* package *caret*, version 6.0-73 (Kuhn et al. 2016 [3]). Deteriorated performance (i.e. low AUC values) for classifier *svm* was seen using a newer version of *caret*, 6.0-77. If you notice this behavior you can downgrade *caret* to the older version running the following commands in succession in the RStudio console window:

remove.packages("caret")

install.packages("rtools")

install.packages("devtools")

require(devtools)

install\_version("caret", version = "6.0-73", repos = "http://cran.us.r-project.org")

Check the installed *caret* package version with these commands in the RStudio console window:

ip <- as.data.frame(installed.packages()[,c(1,3:4)])

rownames(ip) <- NULL

ip <- ip[is.na(ip$Priority),1:2,drop=FALSE]

print(ip, row.names=FALSE)