rmap—v-0.01-07—2015-03-08 22:54:18 PDT

Contents

1 Introduction

The R package rmap (Risk Model Assessment Package) contains tools for validating personal risk models uaing a random sample of censored longitudinal cohort data. We assume that the user has created a risk model that assigns to each subject at the time she enters the study, a risk, which is the probability that she will suffer a specific adverse outcome within a given time period, say in the next 10 years.

Model calibration (also called goodness-of-fit) measures how well the model-specified risks agree with persons' subsequent observed outcomes. For overall calibration, rmap offers two chi-square statistics: **the mean-risk statistic** (Hosmer Lemeshow Chi Square statistic) and the **observed-event-count statistics**. For grouped calibration, rmap offers **grouped attribute diagrams** and **group-specific standardized residuals** together with their corresponding summary goodness-of-fit statistics. For individualized calibration, rmap offers **individualized attribute diagrams**.

Discrimination measures how well a model separates positive and negative outcomes. The ROC and the concordance statistic are the usual measures, which we leave to other authors. rmap offers **case-risk percentiles** and for comparing two risk models **scatterplots of case-risk percentiles**, as well as estimates and confidence intervals for **predictive-power-positive** and **predictive-power-negative**.

See Useful downloads including additonal documentation to help you get started.

TOP

2 Preparing data for R

To run rmap begin with an excel file or an alternative way to create an R data.frame which contains a row for each subject containing specific columns which we describe below. In this example, we assume that you have two risk models: risk model 1 and risk model 2 that you wish to validate.

If you have cumulative outcome hazards (`Lambda_outcome_1` and `Lambda_outcome_2`) but do not have cumulative mortality hazards (`Lambda_mortality_1` and `Lambda_mortality_2`) and you believe that death rates are small, you may still use all of the tools rmap offers. See If you don't have cumulative mortality hazards. If you have NO cumulative hazards (neither outcome nor mortality) you will not be able to obtain **observed-event-count statistics** nor **group-specific standardized residuals** (provided by the functions `score_statistic` and `standardized_residual`), but you can use the other functions in rmap. The sections in this document beginning with the Section riskValidate, attributeDiagram, riskValidateUngrouped, IAD describe the functions that do not require cumulative hazards.

2.1 `e`

The **event** for the subject. `0` denotes censored or alive and free from outcome at time `tStar`, `1` denotes the adverse outcome , and `2` denotes death from other causes. `tStar` is the right end point of the duration of the study. For our example dataset, `tStar = 10`.

2.2 t

The **time** until event. This is the time that event `e` happened. This time must fall in the interval [0, `tStar`).

2.3 r_1

The subject's **assigned risk** under risk model 1. This is the probability of adverse outcome within [0, `tStar`) according to risk model 1, and is a number between `0` and `1`.

2.4 r_2

The subject's **assigned risk** under risk model 2.

2.5 Lambda_outcome_1

The **cumulative hazard of the outcome** until time of event according to risk model 1. This is the hazard rate of outcome integrated from the time the subject enters the study until her event time; `Lambda_outcome_1` for the `n`-th person whose time of event is t_n is `Lambda_outcome_1[n]` = $\int_0^{t_n} \lambda_{n,\text{outcome, risk model 1}}(u)du$.

2.6 Lambda_outcome_2

2.7 Lambda_mortality_1

The **cumulative hazard of death** until event according to risk model 1. This is the hazard rate of death integrated from the time the subject enters the study until her event time; `Lambda_mortality_1` for the `n`-th person whose time of event is t_n is `Lambda_mortality_1` = $\int_0^{t_n} \lambda_{n,\text{mortality, risk model 1}}(u)du$.

2.8 Lambda_mortality_2

2.9 w

Other variables measured on the subject. These are optional. These may be variables that were used to calculate the assigned risk or variables that were missing from the calculation of the assigned risk. My column `w` is a covariate missing from risk model 2. If this variable is important in the prediction of the adverse outcome, risk model 2 will be inferior.

2.10 k1

Grouping variables are integers between 1 and K which can be used to specify into which group each person should be placed. My column k1 specifies quartiles (K = 4) of assigned risk r1. You may specify arbitrary groups using a grouping column like k1. If you wish to explore groups according to quantiles of assigned risk or **other variables** (e.g w), you can let rmap create the groupings automatically, so my k1 is actually unnecessary but included here for illustration.

If you are beginning with an excel file, save your file in Comma Separate Value (CSV) to your working directory. To practice running rmap, you may download my example file here... data_set_score_statistics.csv

TOP

3 Installing rmap

If you have not yet installed R, download the latest version (at least 2.11) for your operating system at:

http://www.r-project.org

Run the R application. To install the rmap package, enter the following lines of code to the R prompt:

```
install.packages("devtools")
library(devtools)
install_github("gailg/rmap")
if("rmap" %in% rownames(installed.packages())){
  print("rmap installed successfully--you are good to go!")
} else {
  print("something went wrong--ask for help")
}
```

If your installation was successful, you should see the message

```
[1] "rmap installed successfully--you are good to go!"
```

TOP

4 Reading in data

If you saved your data from excel as a CSV (comma separated value) file in your working directory, your next step is to read this CSV file into R. To practice reading in my example file "data_set_score_statistics.csv", enter the following lines to R. Your result to `head(x)` should look like my output below.

```
x = read.csv(
    file = "data_set_score_statistics.csv",
    stringsAsFactors = FALSE)
head(x)
```

```
##   e    t    r_1 Lambda_outcome_1 Lambda_mortality_1    r_2
## 1 0 10.0 0.0455           0.0475             0.0420 0.0244
## 2 0  6.4 0.0758           0.0519             0.0271 0.0134
## 3 0 10.0 0.0251           0.0260             0.0420 0.0415
## 4 0 10.0 0.1163           0.1263             0.0420 0.0329
## 5 0 10.0 0.0861           0.0920             0.0420 0.0740
## 6 0  0.4 0.0350           0.0013             0.0016 0.0354
##   Lambda_outcome_2 Lambda_mortality_2      w k_1
## 1           0.0252             0.0420 1.8822   3
## 2           0.0089             0.0271 5.8520   4
## 3           0.0433             0.0420 0.6002   2
## 4           0.0342             0.0420 3.6940   4
## 5           0.0786             0.0420 1.1710   4
## 6           0.0014             0.0016 0.9890   3
```

TOP

5 The rmap functions

I describe the following functions in the coming sections

1. `score_statistic` (mean-risk statistic and observed-event-count statistics)
2. `standardized_residual` (group-specific standardized residuals)
3. `risk_quantile_boxplot` (risk quantile boxplot)
4. `riskValidate` and `attributeDiagram` (mean-risk-statistic and grouped attribute diagrams)
5. `riskValidateUngrouped` and `IAD` (individualized attribute diagram)
6. `caseRiskPercentiles` (case-risk percentiles)
7. `performanceDifference` (predictive-power-positive and predictive-power-negative estimates)

TOP

6 score_statistic and standardized_residuals

The function `score_statistics` calculates observed-event-count statistics that can be used to test the null hypothesis that a risk model is correct. Such a statistic is gotten by embedding the outcome and mortality hazards that are used in the risk model into a larger model by regressing on one or several covariates. The null hypothesis states that the regression coefficients vanish or equivalently the original risk model is correct. Changing the covariate(s) in the regression gives a different observed-event-count statistic. `score_statistic` reports the observed-event-count statistics for a battery of covariates giving a powerful tool for exploring the accuracy of a risk model.

The function `standardized_residuals` calculates the numbers that are used in the attribute-diagram-and-standardized-residuals plot. This plot is a tool for investigating the reasons for poor fit when the null hypothesis is rejected.

See Gong et.al."Assessing the goodness of fit of personal risk models" **Statistics in Medicine** 2014, 33, 3179–3190 and score-statistics-formulas-v01.pdf for more details.

TOP

6.1 Function arguments for score_statistics and standardized_residuals

rmap functions must be called with specific arguments that must be prepared in a precise format. This section describes how to prepare these arguments from the example data.frame `x`. A basic understanding of the R language would be helpful in this section.

6.1.1 e

`e` is a vector containing the event of each subject in the data set. It is the first column in the example dataset `x`. Given my data set, I define `e` like this:

```
e = x$e
```

(Note that `x$e` means "grab the column named `e` from the dataset `x`.)

6.1.2 t

`t` is a vector containing the time until event of each subject in the data set. It is the second column in `x`. Given my data set, I define `t` like this:

```
t = x$t
```

6.1.3 risk_model

A risk_model is a named list that instructs rmap what data to use to explore the accuracy for a given risk model. This list must contain four objects. The first three objects are vectors which must be named r, Lambda_outcome and Lambda_mortality, and are respectively the assigned risk, cumulative outcome hazard, and cumulative mortality hazard for each subject according to the given risk model.

The fourth object in this list is another list named groupings. Each element of the groupings list "describes" a grouping of the data that the user wishes to explore. rmap offers several ways to describe a grouping. Suppose that I have defined a grouping description, called risk_model_1_grouping, that groups the data by assigned risks of risk model 1, and a grouping description missing_covariate_grouping that groups the data by the covariate that is missing in risk model 2.

Given my data set I can define the following risk_model to explore risk model 1.

```
risk_model = list(
  r = x$r_1,
  Lambda_outcome = x$Lambda_outcome_1,
  Lambda_mortality = x$Lambda_mortality_1,
  groupings = list(
    risk = risk_model_1_grouping,
    missing = missing_covariate_grouping))
```

For risk model 1, the assigned risk is x$r_1, the cumulative outcome hazard is x$Lambda_outcome_1, and the cumulative mortality hazard is x$Lambda_mortality_1, so r, Lambda_outcome, and Lambda_mortality are assigned to these respectively. The fourth component grouping is assigned the list that contains risk_model_1_grouping which I conveniently name risk and missing_covariate_grouping which I conveniently name missing. The components of the groupings list must be named, but you can use whatever names you wish to help you document your results.

rmap allows you to define multiple risk_model's. Since my example data set contains variables that will allow me to explore two risk models which I have called risk model 1 and risk model 2, I might define my risk models like this

```
risk_model_1 = list(
  r = x$r_1,
  Lambda_outcome = x$Lambda_outcome_1,
  Lambda_mortality = x$Lambda_mortality_1,
  groupings = list(
    risk = risk_model_1_grouping
    missing = missing_covariate_grouping))
risk_model_2 = list(
  r = x$r_2,
  Lambda_outcome = x$Lambda_outcome_2,
  Lambda_mortality = x$Lambda_mortality_2,
```

```
groupings = list(
  risk = risk_model_2_grouping
  missing = missing_covariate_grouping))
```

I renamed my original `risk_model` to `risk_model_1` and created another `risk_model` called `risk_model_2`. Although the format of `risk_model` arguments is very specific, their names can be chosen for convenience to document the results.

6.1.4 `grouping_description`

A `grouping_description` is a `list` that gives a description of a grouping or partition of the data into `K` subgroups. This list can be defined in various ways to offer flexible specifications for grouping.

1. If you have a column in your data set that contains grouping numbers, you can instruct rmap to use this column. In my example data set, the column `x$k_1` contains numbers `1` through `4`; the number `1` was assigned to the quarter of the subjects with lowest values of assigned risk of risk model 1; the number `4` was assigned to the quarter of the subjects with the highest values of assigned risk. I define the `risk` grouping inside the `groupings` list `risk = list(k = x$k_1))`:

   ```
   risk_model_1 = list(
     r = x$r_1,
     Lambda_outcome = x$Lambda_outcome_1,
     Lambda_mortality = x$Lambda_mortality_1,
     groupings = list(
     risk = list(k = x$k_1)))
   ```

2. If you have not included a grouping column for a grouping that you want to include, you can let rmap create your grouping. In my example data set, I did not include a column that groups subjects according to risk model 2. If I would like to group subjects according to quartiles of assigned risks of risk model 2, I could use 'risk = list(K = 4):

   ```
   risk_model_2 = list(
     r = x$r_2,
     Lambda_outcome = x$Lambda_outcome_2,
     Lambda_mortality = x$Lambda_mortality_2,
     groupings = list(
     risk = list(K = 4)))
   ```

3. rmap can create a grouping based on `cutoffs`. Assume that you want all subjects having an assigned risk (of risk model 1) `r_1` between 0 and 0.33 to be in subgroup 1, all subjects with `r_1` between 0.33 and 0.66 to be in subgroup 2 and all subjects with `r_1` between 0.66 and 1 to be in the final subgroup 3. Define `risk = list(cutoffs = c(0, 0.33, 0.66, 1))`:

```
risk_model_1 = list(
  r = x$r_1,
  Lambda_outcome = x$Lambda_outcome_1,
  Lambda_mortality = x$Lambda_mortality_1,
  groupings = list(
  risk = list(cutoffs = c(0, 0.33, 0.66, 1))))
```

4. rmap can group subjects according to quantiles of a variable that is different from the assigned risk. In my example data set, I have included the column `x$w` which is a variable that is missing from risk model 2. To group according to quantiles of `x$w`, define `missing = list(K = 4, variable = x$w`. (I have named this grouping `missing` for convenience and for documentation of my results. If `x$w` were age of the subject at time of entry, I might name this grouping `age`.)

```
risk_model_1 = list(
  r = x$r_1,
  Lambda_outcome = x$Lambda_outcome_1,
  Lambda_mortality = x$Lambda_mortality_1,
  groupings = list(
  missing = list(K = 4, variable = x$w)))
```

5. rmap can group subjects according to cutoffs of a variable that is different from the assigned risk.
`missing = list(cutoffs = c(0, 1, 2, Inf), variable = x$w)`

```
risk_model_1 = list(
  r = x$r_1,
  Lambda_outcome = x$Lambda_outcome_1,
  Lambda_mortality = x$Lambda_mortality_1,
  groupings = list(
  missing = list(cutoffs = c(0, 1, 2, Inf), variable = x$w)))
```

TOP

6.2 How to call `score_statistics`

The following is the function header of `score_statistic` and shows which arguments need to be specified and in which order. The dot-dot-dot says "any number of named `risk_model` lists".

```
score_statistic(e, t, ...)
```

With our arguments from Function arguments for score_statistics and standardized_residuals in place, we can enter the following code into R to call this function.

```
ss = score_statistics(e, t, risk_model_1 = risk_model_1,
  risk_model_2 = risk_model_2)
```

TOP

6.3 A score_statistics example

This subsection includes code that is part of a complete score_statistics example. This example continues in Sections Adding standardized_residuals to our example, If you don't have cumulative mortality hazards, and Adding risk_model_boxplots to our example.

```
library(rmap)
x = read.csv(
  file = "data_set_score_statistics.csv",
  stringsAsFactors = FALSE)
head(x)
```

```
##   e    t    r_1 Lambda_outcome_1 Lambda_mortality_1    r_2
## 1 0 10.0 0.0455           0.0475             0.0420 0.0244
## 2 0  6.4 0.0758           0.0519             0.0271 0.0134
## 3 0 10.0 0.0251           0.0260             0.0420 0.0415
## 4 0 10.0 0.1163           0.1263             0.0420 0.0329
## 5 0 10.0 0.0861           0.0920             0.0420 0.0740
## 6 0  0.4 0.0350           0.0013             0.0016 0.0354
##   Lambda_outcome_2 Lambda_mortality_2      w k_1
## 1           0.0252             0.0420 1.8822   3
## 2           0.0089             0.0271 5.8520   4
## 3           0.0433             0.0420 0.6002   2
## 4           0.0342             0.0420 3.6940   4
## 5           0.0786             0.0420 1.1710   4
## 6           0.0014             0.0016 0.9890   3
```

```
e = x$e
t = x$t
risk_model_1 = list(
  r = x$r_1,
  Lambda_outcome = x$Lambda_outcome_1,
  Lambda_mortality = x$Lambda_mortality_1,
  groupings = list(
    risk = list(k = x$k_1),
    missing = list(K = 4, variable = x$w)))
risk_model_2 = list(
  r = x$r_2,
  Lambda_outcome = x$Lambda_outcome_2,
  Lambda_mortality = x$Lambda_mortality_2,
  groupings = list(
    risk = list(K = 4),
    missing = list(K = 4, variable = x$w)))
ss = score_statistics(e, t,
  risk_model_1 = risk_model_1,
  risk_model_2 = risk_model_2)
ss
```

```
##                          risk_model_1  risk_model_2
## overall_hosmer_lemeshow   0.57294093 9.1228476e-05
## overall_combined          0.52218795 4.8422259e-05
## overall_outcome           0.69928172 5.6382856e-09
## overall_mortality         0.59305688 5.9305688e-01
## weighted_combined         0.47503131 4.8108544e-05
## weighted_outcome          0.58817961 5.8418030e-09
## weighted_mortality        0.64072384 6.0700599e-01
## risk_hosmer_lemeshow      0.35064394 1.6242948e+01
## risk_combined             0.84978491 4.8010995e-04
## risk_outcome              0.99266832 1.5342834e-07
## risk_mortality            0.84254748 9.2942995e-01
## missing_hosmer_lemeshow   2.20778808 2.5714225e+01
## missing_combined          0.57657610 3.0030638e-06
## missing_outcome           0.72387382 2.0643665e-16
## missing_mortality         0.74717289 7.4717289e-01
```

TOP

6.4 `score_statistics` output description

The output for `score_statistics` consists of a matrix, each column corresponding to a `risk_model` argument in your `score_statistic` call, and each row corresponding to a different observed-event-count statistic resulting from a different vector of covariates that expands the hazard rates for outcome and/or death of the risk model of each column.

The rows of this matrix contain the **p-values** for the following observed-event-count statistics.

1. **overall_hosmer_lemeshow**: The Hosmer-Lemeshow Chi-squared goodness-of-fit statistic. This statistic compares the mean overall outcome probability of outcome to the mean overall assigned risk.

2. **overall_combined**: The score statistic with covariate $z = z_1 = z_2 = 1$ described in *Section 2A* of score-statistics-formulas-v01.pdf.

3. **overall_outcome**: The score statistic with covariate z_1.

4. **overall_mortality**: The score statistic with covariate z_2.

5. **weighted_combined**: The score statistic with covariate for the n-th subject $z_{nj} = \frac{\exp(|r_n - \bar{r}|)}{\exp(\max(r_n) - \bar{r})}$.

6. **weighted_outcome**: The score statistic with covariate for the n-th subject z_{n1} where z_{nj} is described in 5. **weighted_combined**.

7. **weighted_mortality**: The score statistic with covariate for the n-th subject z_{n2} where z_{nj} is described in 5. **weighted_combined**.

For each `grouping_description` in the `groupings` list of the `risk_model` there is a row containing the **p-value** for each of the following statistics.

1. **grouping__hosmer__lemeshow**: The Hosmer-Lemeshow Chi-squared goodness-of-fit statistic with $K =$ the number of subgroups in this grouping.
2. **grouping__combined**: The score statistic with covariate for the n-th subject $z_n = z_{n1} = z_{n2}$ where z_{njk} indicates membership in subgroup k.
3. **grouping__outcome**: The score statistic with covariate for the n-th subject is z_{n1} where z_{nj} is described in 2. **grouping__combined**.
4. **grouping__mortality**: The score statistic with covariate for the n-th subject is z_{n2} where z_{nj} is described in 2. **grouping__combined**.

TOP

6.5 How to call `standardized_residuals`

The following is the function header of `standardized_residuals` and shows that the arguments for this function are exactly the same as those for `score_statistics`.

```
standardized_residuals = function(e, t, ...)
```

With our arguments from Section Function arguments for score_statistics and standardized_residuals in place, we can enter the following code into R to call this function.

```
sr = standardized_residuals(e, t, risk_model_1 = risk_model_1,
  risk_model_2 = risk_model_2)
```

The output from `sr` is a complicated list of list of numbers. You do not need to understand the intricacies of `sr` because you can use it to create some useful plots.

The following is the function header of `plot`.

```
plot(sr, grouping_name)
plot(sr, grouping_name, plot_pars)
```

The arguments that go into `plot` are the following

`sr` is the result from a call to `standardized_residuals`.

`grouping_name` is the name of a `grouping_description` in `groupings` of each `risk_model`.

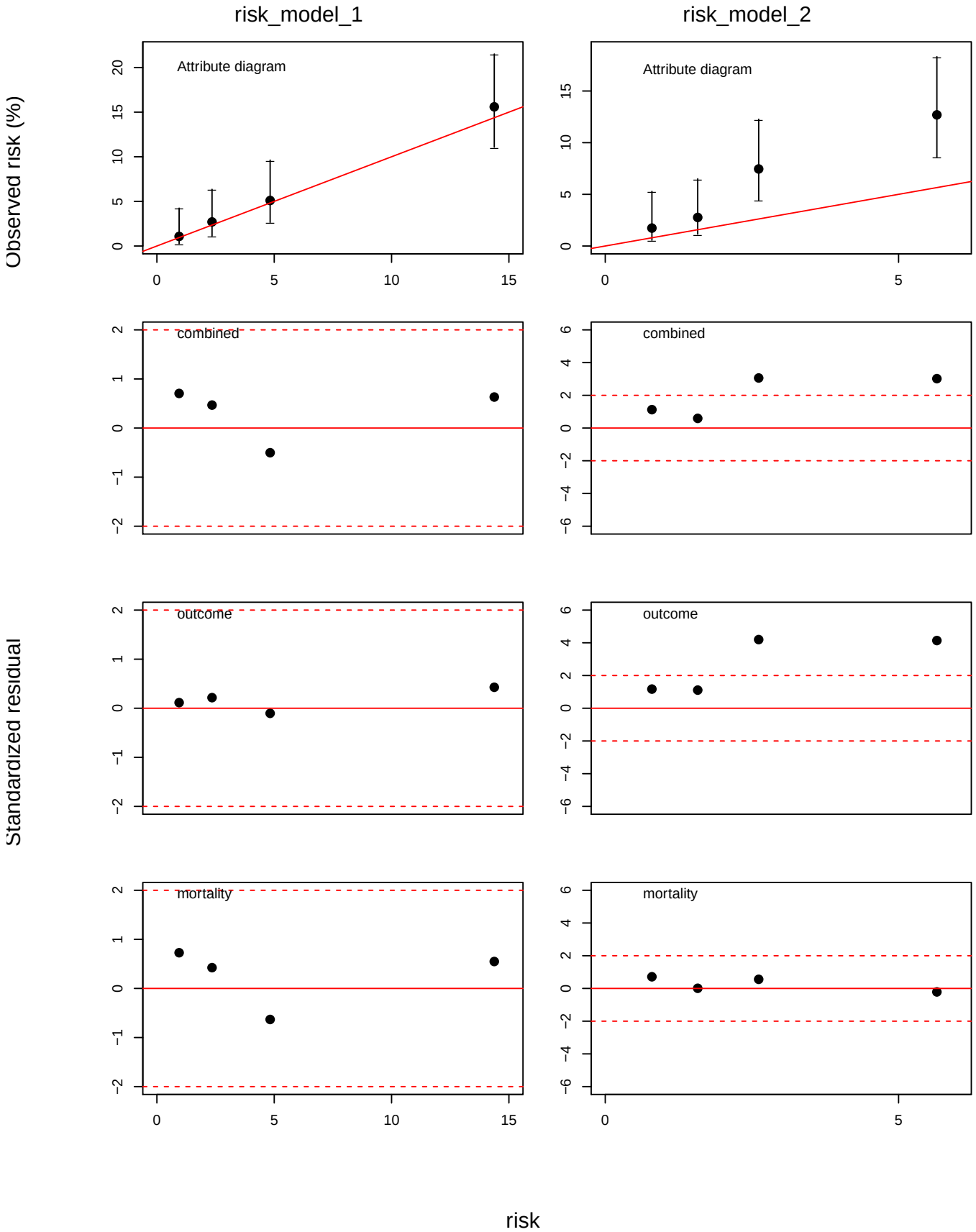
`plot_pars` is an optional list of plot parameters; when included, the x and y axes of the plot are drawn to the same scale for both risk models.

TOP

6.6 Adding `standardized_residuals` to our example

We continue with A score_statistics example

```
sr = standardized_residuals(e, t,
  risk_model_1 = risk_model_1,
  risk_model_2 = risk_model_2)
plot(sr, grouping_name = "risk")
```



The previous command `plot(sr, grouping_name = "risk")` produces a rough picture. To produce a prettier picture, we add `plot_pars` to the `plot` function, using the previous picture to help choose

the numbers in `plot_pars` so that the x-axes and y-axes of both risk models are drawn to the same scale.

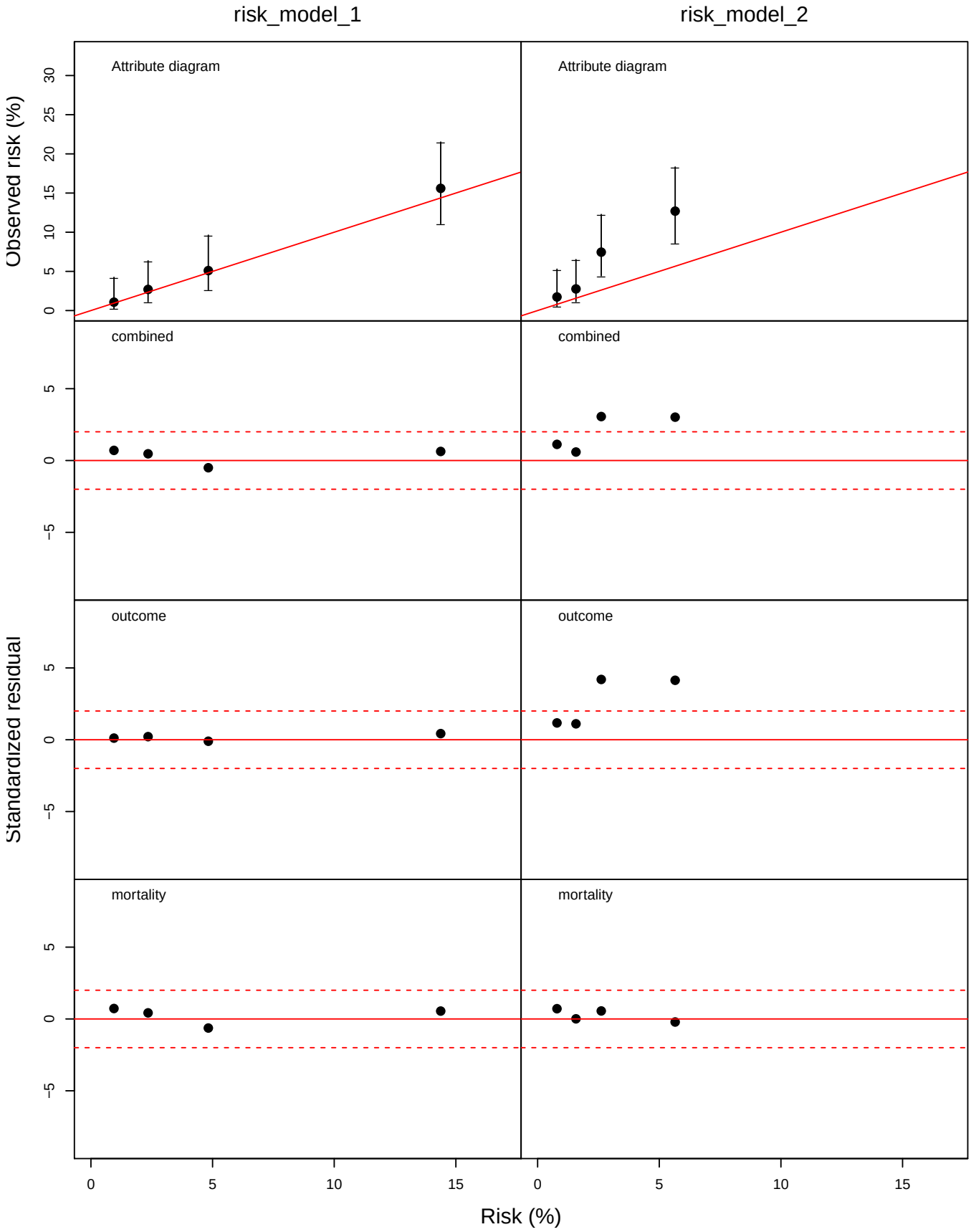
`x_max` specifies the largest risk (in percents) that the x-axis shows.

`y_max` specifies the largest positive standardized residual that the y-axis of each standardized residual plot shows.

`y_max_ad` specifies the largest risk (in percents) that the y-axis of the attribute diagram shows.

We choose the above three numbers to be slightly bigger than the biggest number shown in the rough plot (and choose `xlab` to show a prettier label for the x-axis).

```
plot_pars = list(x_max = 0.17,
                 y_max = 9,
                 y_max_ad = 0.33,
                 xlab = "Risk (%)")
plot(sr, grouping_name = "risk", plot_pars)
```


risk_model_1
risk_model_2
Attribute diagram
Attribute diagram
Observed risk (%)
combined
combined
outcome
outcome
mortality
mortality
Standardized residual
Risk (%)

TOP

6.7 `plot` description

The plot consists of a grid containing 4 rows and a column for each `risk_model` object in your `standardized_residuals` call.

The first row contains an attribute diagram for each risk model. Each attribute diagram contains a dot for each subgroup in the grouping. The x-value of the dot is the mean assigned risk of the subgroup and the y-value is of observed risk of the subgroup. Points close to the diagonal line show good model fit. The attribute diagram also shows a 95 percent confidence interval for the observe

The second through fourth rows show standardized residual plots for `combined`, `outcome`, and `mortality` events respectively. The standardized residuals are the δ_{jk} in *2C Example 3* of score-statistics-formulas-v01.pdf, and deviations above or below the dotted red lines show poor model fit.

TOP

6.8 If you don't have cumulative mortality hazards

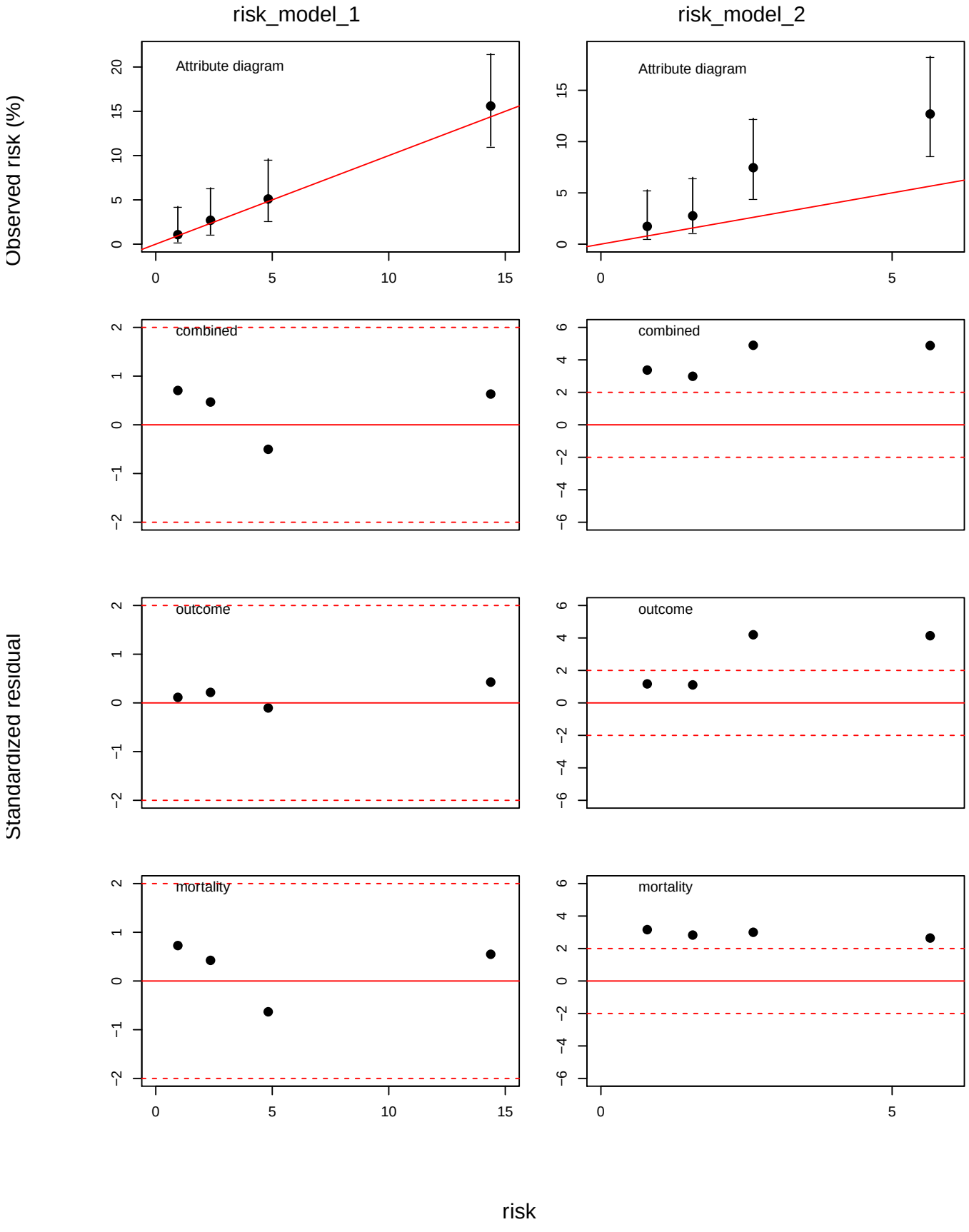
Suppose risk_model_2 has no cumulative mortality hazards. set `lambda_mortality = NULL` as I have done in the example below.

```
risk_model_2 = list(
  r = x$r_2,
  Lambda_outcome = x$Lambda_outcome_2,
  Lambda_mortality = NULL,
  groupings = list(
    risk = list(K = 4),
    missing = list(K = 4, variable = x$w)))
ss_no_mortality = score_statistics(e, t,
  risk_model_1 = risk_model_1,
  risk_model_2 = risk_model_2)
ss_no_mortality
```

```
##                           risk_model_1  risk_model_2
## overall_hosmer_lemeshow   0.57294093 9.1228476e-05
## overall_combined          0.52218795 3.0173375e-16
## overall_outcome           0.69928172 5.6382856e-09
## overall_mortality         0.59305688 5.5112073e-09
## weighted_combined         0.47503131 2.9404453e-16
## weighted_outcome          0.58817961 5.8418030e-09
## weighted_mortality        0.64072384 5.5168611e-09
## risk_hosmer_lemeshow      0.35064394 1.6242948e+01
## risk_combined             0.84978491 5.7187265e-14
## risk_outcome              0.99266832 1.5342834e-07
## risk_mortality            0.84254748 7.4518879e-07
## missing_hosmer_lemeshow   2.20778808 2.5714225e+01
## missing_combined          0.57657610 1.3794545e-17
```

```
## missing_outcome           0.72387382 2.0643665e-16
## missing_mortality         0.74717289 7.4518879e-07
```

```
sr_no_mortality = standardized_residuals(e, t,
  risk_model_1 = risk_model_1,
  risk_model_2 = risk_model_2)
plot(sr_no_mortality, grouping_name = "risk")
```


risk_model_1
risk_model_2
Attribute diagram
Attribute diagram
Observed risk (%)
combined
combined
outcome
outcome
mortality
mortality
Standardized residual
risk

TOP

7 risk_model_boxplots

The function `risk_model_boxplots` creates boxplots of risk quantiles. Such boxplots can be useful for judging information loss from grouping the data into quantiles.

7.1 Function arguments for `risk_model_boxplots`

7.1.1 `list_of_risk_models`

A `list_of_risk_models` is as the name indicates a list of risk models. Each element in this list is a named vector of assigned risks according to a risk model. Given my example data set, I can define the following `list_of_risk_models`

```
list_of_risk_models = list(
  risk_model_1 = x$r_1,
  risk_model_2 = x$r_2)
```

7.1.2 `K`

`K` is a positive integer specifying the number of quantiles. `K = 4` specifies quartiles.

7.1.3 `risk_max`

`risk_max` is a number between `0` and `1` and specifies the largest risk that is displayed on the x-axis of the boxplots. This number defaults to `1`.

7.1.4 `text_x`

`text_x` is a number between `0` and `1` and specifies the x-axis of the subheading of each boxplot. If you leave it unspecified or specify `text_x = NULL`, rmap will place the subheading in the center of the range of x.

7.1.5 `text_y`

`text_y` is a number between `0` and `K` and specifies the y-axis of the subheading of each boxplot. If you leave it unspecified or specify `text_y = NULL`, rmap will place the subheading close to the top range of boxplot.

TOP

7.2 How to call `risk_model_boxplots`

The following is the function header of `risk_model_boxplots` and shows which arguments need to be specified and in which order.

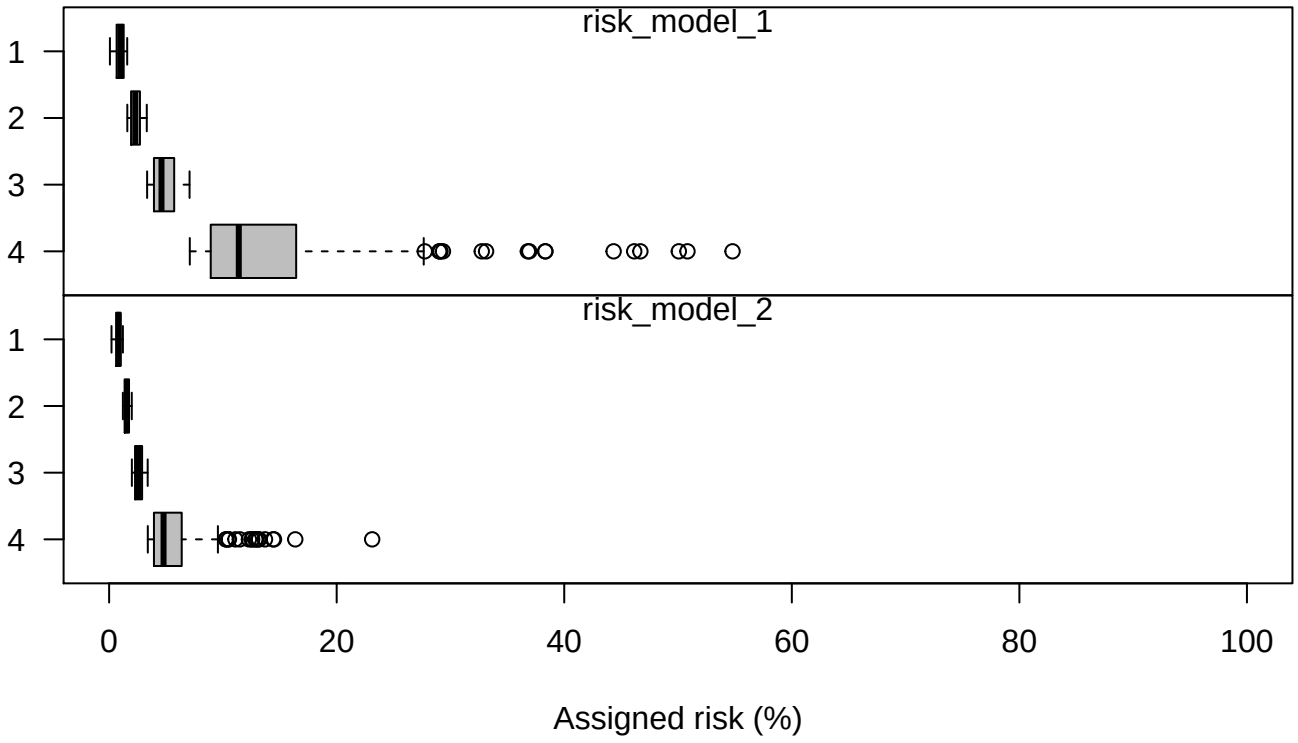
```
risk_quantile_boxplots(
  list_of_risk_models,
  K,
  risk_max = 1,
  text_x = NULL,
  text_y = NULL)
```

TOP

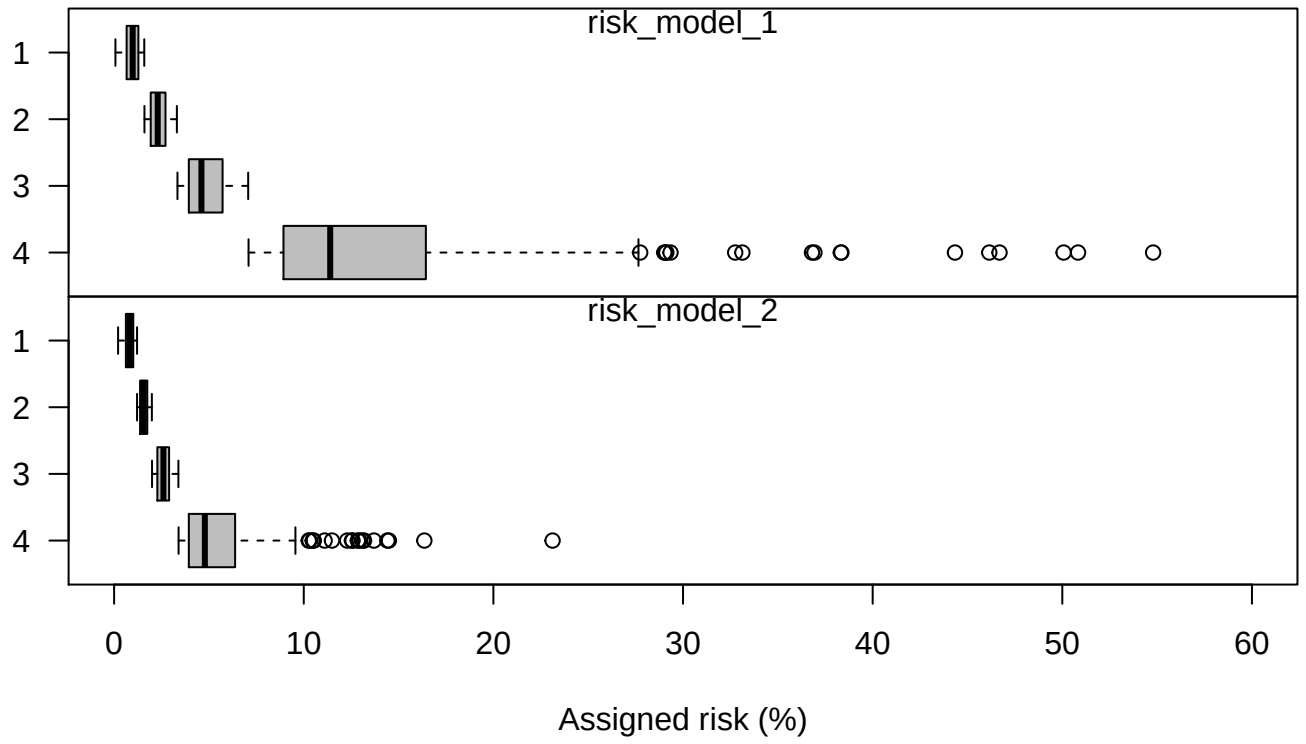
7.3 Adding risk_model_boxplots to our example

We continue with A score_statistics example

```
list_of_risk_models = list(
  risk_model_1 = x$r_1,
  risk_model_2 = x$r_2)
K = 4
risk_quantile_boxplots(list_of_risk_models, K)
```



```
risk_quantile_boxplots(list_of_risk_models, K, risk_max = 0.6)
```



TOP

7.4 `risk_model_boxplots` output description

The plot consists of a grid containing 1 column and a row for each vector in `list_of_risk_models`. Each row contains a boxplot display of the assigned risks of a risk model broken down into `K` quantiles.

TOP

8 riskValidate, attributeDiagram, riskValidateUngrouped, IAD

Beginning with this section I discuss rmap functions that do not require cumulative hazards.

The function riskValidate validates a personal risk model using a grouped analysis. It compares assigned risks to subsequent outcomes by calculating for each risk group a piHat, the estimated probability of disease occuring in the duration fo the study. riskValidate reports a summary of statistics that can be used to describe the validity of one's model. It can also produce an attribute diagram, a graphic that compares piHat to assigned risk in each risk group.

Similarly to `riskValidate`, the function `riskValidateUngrouped` validates a personal risk model, but instead of estimating the probability of disease at each of several risk groups, whose definition might be somewhat arbitrary, `riskValidateUngrouped` calculates piHat at each distinct value

of assigned risk. `riskValidateUngrouped` also calculates an AUC estimate, and if desired, an individualized attribute diagram, which is a graphic similar to the attribute diagram but for ungrouped data.

See Quante et.al. "Breast cancer risk assessment across the risk continuum: genetic and nongentic risk factors contributing to differential model performance" **Breast Cancer Research** 2012, 14, R144 and The mathematical formula behind the functions used in rmap for details.

TOP

8.1 Function arguments for 'riskValidate and riskValidateUngrouped

8.1.1 `e`

`e` is a vector containing the event of each subject in the data set. It is the first column in the example dataset `x`. To define `my_e`, issue the command:

```
my_e = x$e
```

(Note that `x$e` means "grab the column named 'e' from the dataset 'x' ".)

8.1.2 `t`

`t` is a vector containing the time until event of each subject in the data set. It is the second column in `x`. To define the object `my_t`, issue the command:

```
my_t = x$t
```

8.1.3 `r`

`r` is a vector containing the risk assigned to each subject by a given risk model. It is the fourth column of the dataset `x`. To define `my_r` issue the command:

```
my_r = x$r
```

8.1.4 `design`

`design` describes the sampling design used in the dataset. The sampling design can be a random sample or a two-stage sample.

1. If the data were obtained by a random sample, define `my_design` as follows:

   ```
   my_design = "randomSample"
   ```

2. If the data were obtained by two-stage sampling, we must provide more information to describe the details of the sampling design. In the first stage, a random sample of subjects were screened and placed into two or more categories. Suppose that in our example, we have two categories, and 472 subjects fell into the first category and 528 subjects fell into the second category. We name the categories `A` and `B` and define `my_N` as follows:

   ```
   my_N = c(A = 472, B = 528)
   ```

   Note that it is important that we include labels `A` and `B` with these counts, so don't forget the "`A =`" and "`B =`" parts of the line above. In R parlance, such labels are called names and we will use the word name instead of label throughout the remainder of this page.

   In the second stage of two-stage sampling, subjects were resampled with different probabilities depending on their categories. The subjects resampled are the ones recorded in our tabular dataset such as `x`.

   We want to prepare a vector `my_c`, which describes the sampling category of each subject in the dataset. The data set `x` was obtained by random sampling; if it had been obtained by two-stage sampling, and if the column `x$c` recorded which two-stage category each subject fell into, we would define `my_c` like this:

   ```
   my_c = x$c
   ```

   Finally we bundle the two objects, first-stage counts (`my_N`) and each subject's sampling category (`my_c`), into one R object called a "list". Just as the elements in `my_N` required names, the elements in this list also require names:

   ```
   my_design = list(N = my_N, c = my_c)
   ```

8.1.5 `riskGroup`

The argument `riskGroup` describes the way in which subjects are divided into risk groups. There are four ways that `riskGroup` can specify risk groups:

1. If you have specified risk groups in a column of your dataset, `riskGroup` can be described by this column. In the case of our sample dataset `x`, the sixth column `k` holds risk group designation. If we wish to use this column to define our risk groups, we can define `my_riskGroup` as follows:

   ```
   my_riskGroup = list(k = x$k)
   ```

   ```
   Note that this object is a list containing a vector (`x$k`)
   named `k` to distinguish this way of describing risk groups
   from the other ways.
   ```

2. If you have not specified risk groups in a column of your dataset, rmap can do the job for you. One way is to tell rmap the number of risk groups to use. rmap will then divide the subjects into risk groups automatically, putting approximately equal number of subjects into each risk group, taking into account two-stage sampling if applicable. The first risk group will hold the subjects with the smallest assigned risk `r`. The last risk group will hold the subjects with the highest assigned risk. To tell rmap to use 4 risk groups, define `my_riskGroup` as follows:

```
my_riskGroup = list(K = 4)

Again, this object must be a named list, but the name of the
element is `K` to distinguish this way of defining the
risk groups.
```

3. rmap can also specify risk groups using risk "cutoffs". Assume that we want all subjects having an assigned risk `r` between 0 and 0.33 to be in risk group 1, all subjects with `r` between 0.33 and 0.66 to be in risk group 2 and all subjects with `r` between 0.66 and 1 to be in the final risk group, 3. We can define `my_riskGroup` as follows:

```
my_riskGroup = list(cutoffs = c(0, 0.33, 0.66, 1))

Again, this object must be a named list, but the name of the
element is `cutoffs` to distinguish this way of
defining the risk groups.  The first value in `cutoffs` must
be 0, and the last value must be 1.
```

4. Instead of estimating the outcome probability at each of a few risk groups and producing an attribute diagram, `riskValidateUngrouped` estimates the outcome probability at each distinct assigned risk by using an epsilon kernel neighborhood and producing an "individualized attribute diagram" (IAD). The quantity epsilon is a number in (0,1) and determines the proportion of the data set to include in the neighborhood. Theoretical calculations suggest epsilon should grow as `NTotal^(-1/3)` where `NTotal` is the number of observations in the data set if the data were obtained by random sampling or the number of observations in the first stage if the data were obtained by two-stage sampling. rmap can also calculate an ungrouped AUC estimate using "case risk percentiles" (CRPs) which require `tStar`, the right end point of the duration of study. To specify an ungrouped analysis, define `my_riskGroup` as follows:

```
my_riskGroup = list(ungrouped = list(epsilon = epsilon, tStar = tStar))
```

8.1.6 `rSummary`

The argument `rSummary` is a summary statistic for the assigned risks for all subjects in each risk group. `rSummary` can be provided by the user or can be left for rmap to calculate. There are four options in specifying `rSummary`:

1. rmap can compute `rSummary` as the "mean" value (adjusted for two-stage sampling weights if applicable) of the assigned risks for the subjects in each risk group. To use this option, define `my_rSummary` as follows:

```
my_summary = "mean"
```

2. rmap can compute `rSummary` as the "median" value (adjusted for two-stage sampling weights if applicable) of the assigned risks for the subjects in each risk group. To use this option, define `my_rSummary` as follows:

   ```
   my_summary = "median"
   ```

3. If `cutoffs` were supplied for the `riskGroup` argument, then this option can be used. `rSummary` can be calculated as the "midpoint" of each interval defined in the `riskGroup`'s `cutoffs`. If the values for the cutoffs in the `riskGroup`'s argument were 0, 0.33, 0.66, and 1, then rmap would automatically compute the `rSummary` values to be 0.165, 0.495, and 0.83. To use this option, define `my_rSummary` as follows:

   ```
   my_summary = "midpoint"
   ```

4. To bypass the above options and specify your own rSummary values define `rSummary` directly, using for example,

   ```
   my_summary = c(0.3, 0.5, 0.7)
   ```

8.1.7 bootstrap

Some user functions can provide bootstrap confidence intervals for various parameters. To turn on bootstrapping, `my_bootstrap` can be defined accordingly:

```
my_bootstrap = 1000
```

The above example will use 1000 bootstrap samples to compute confidence intervals. Using 1000 or more bootstrap replications is recommended to achieve stable bootstrap estimates.

You can also turn off bootstrapping. This will speed up code, but will report less information. To turn off bootstrapping, issue the command:

```
my_bootstrap = FALSE
```

8.1.8 rvpar

The argument `rvpar` controls graphical parameters. The default value for this argument in `riskValidate` and `riskValidateUngrouped` directs rmap to use prespecified colors, light colors, the maximum values that the x- and y-axes can show, comments, whether or not to annotate the graphic, the type and size of plot characters, whether to label the plot with percents or fractions, where to draw tick marks, and the labels for the axes and the plot. The easiest way to specify `rvpar` is to let `rmap` use the default graphical parameters. To learn how to change these graphical parameters, visit the function help page by issuing the following to R:

```
help(rvparFn)
```

`rvpar()` is modeled after `par()` in traditional graphics and `gpar()` in grid graphics.

8.1.9 `multicore`

For a large data set, `riskValidateUngrouped` with bootstrapping turned on can require huge computing times. If you have multiple processors available, you can spread the calculations of the bootstrap among your processors by defining

```
my_multicore = TRUE
```

To perform all bootstrap calculations on a single processor, define

```
my_multicore = FALSE
```

8.1.10 `verbose`

Even with bootstrapping turned off, `riskValidateUngrouped` can take some time. Define

```
my_verbose = TRUE
```

to instruct rmap to give a little progress report at intermittent steps of the calculation.

8.2 How to call `riskValidate`

The following is the function header of `riskValidate` and shows which arguments need to be specified and in which order.

```
riskValidate(
  e, t, r, design = "randomSample",
  riskGroup, rSummary,
  bootstrap = FALSE, rvpar = rvparFn())
```

With our arguments from Function arguments for riskValidate and riskValidateUngrouped in place, we can enter the following code into R to call this function.

```
rv = riskValidate(
  e = my_e, t = my_t, r = my_r, design = my_design,
  riskGroup = my_riskGroup, rSummary = my_rSummary,
  bootstrap = my_bootstrap, rvpar = rvparFn())
```

As we saw in Function arguments for riskValidate and riskValidateUngrouped, the last argument `rvpar` specifies graphical parameters. Setting

```
rvpar = rvparFn()
```

instructs `riskValidate` to use default graphical parameters. To instruct `riskValidate` to omit the attribute diagram, use

```
rvpar = FALSE
```

TOP

8.3 A `riskValidate` example

Below is an example of a complete risk validation grouped analysis of the sample dataset `x`.

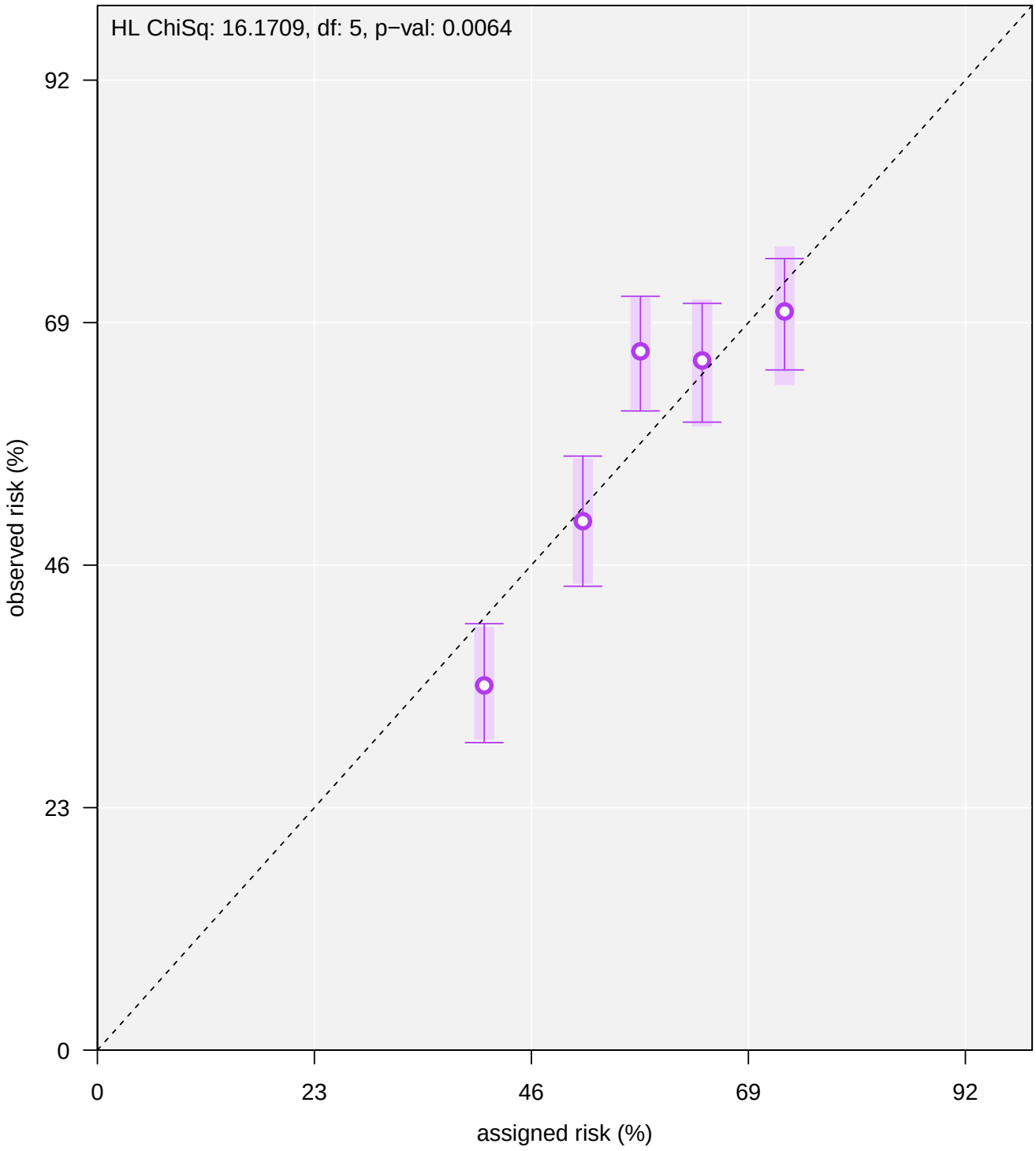
```
library(rmap)
x = read.csv(
  file = "datafRandomSample.csv",
  stringsAsFactors = FALSE)
head(x)
```

```
##   e           t          w          r c k
## 1 2 2.561438270 0.12866006 0.50549600 A 2
## 2 0 0.095180473 0.17789840 0.60040155 A 3
## 3 1 5.345122849 0.11833314 0.48092134 A 2
## 4 2 2.384192874 0.31289498 0.74560713 A 5
## 5 1 1.243401695 0.18858680 0.61701527 A 4
## 6 2 1.377850831 0.11905290 0.48269588 A 2
```

```
my_e = x$e
my_t = x$t
my_r = x$r
my_design = "randomSample"
my_riskGroup = list(k = x$k)
my_rSummary = "mean"
my_bootstrap = 30

rv = riskValidate(
  e = my_e, t = my_t, r = my_r, design = my_design,
  riskGroup = my_riskGroup, rSummary = my_rSummary,
  bootstrap = my_bootstrap, rvpar = rvparFn())
```

```
## [1] "Loading 'grid' package"
## [1] "Done loading 'grid' package"
```



2015−03−08 22:54:49
riskValidate(e = my_e, t = my_t, r = my_r, design = my_design,

```
options(width = 120)
options(digits = 3)
rv
```

```
## $gammaHat
## [1] 0.2 0.2 0.2 0.2 0.2
##
## $piHat
##    k1    k2    k3    k4    k5
## 0.346 0.502 0.663 0.654 0.701
##
## $Sigma
##        [,1]  [,2]  [,3]  [,4] [,5] [,6] [,7] [,8] [,9]
##  [1,]  0.16 -0.04 -0.04 -0.04 0.00    0 0.00 0.00 0.00
##  [2,] -0.04  0.16 -0.04 -0.04 0.00    0 0.00 0.00 0.00
##  [3,] -0.04 -0.04  0.16 -0.04 0.00    0 0.00 0.00 0.00
##  [4,] -0.04 -0.04 -0.04  0.16 0.00    0 0.00 0.00 0.00
##  [5,]  0.00  0.00  0.00  0.00 1.67    0 0.00 0.00 0.00
##  [6,]  0.00  0.00  0.00  0.00 0.00    2 0.00 0.00 0.00
##  [7,]  0.00  0.00  0.00  0.00 0.00    0 1.55 0.00 0.00
##  [8,]  0.00  0.00  0.00  0.00 0.00    0 0.00 1.67 0.00
##  [9,]  0.00  0.00  0.00  0.00 0.00    0 0.00 0.00 1.46
##
## $piHatSummary
##        r piHat  sigma lower upper inCI sigmaBoot lowerBoot upperBoot inBootCI
## k1 0.410 0.346 0.0289 0.292 0.404   no    0.0271     0.295     0.401       no
## k2 0.515 0.502 0.0317 0.440 0.563  yes    0.0301     0.443     0.560      yes
## k3 0.576 0.663 0.0278 0.606 0.715   no    0.0268     0.609     0.713       no
## k4 0.641 0.654 0.0289 0.596 0.708  yes    0.0306     0.592     0.711      yes
## k5 0.728 0.701 0.0270 0.645 0.751  yes    0.0333     0.631     0.762      yes
##
## $ChiSq
## HosmerLemeshow        HL_pval
##       16.17087        0.00637
##
## attr(,"class")
## [1] "rv"   "list"
```

TOP

8.4 Explaining `riskValidate` output

The output for `riskValidate` consists of the following items:

1. `gammaHat` - The estimated proportion of subjects in each risk group.
2. `piHat` - Estimated value for pi for each risk group. pi is the probability of getting disease in (0, `tStar`], the duration of the study.
3. `Sigma` - Define K as the number of risk groups. Sigma is the covariance matrix for the vector (`gammaHat[1], ... gammaHat[K-1]`, `piHat[1], ..., piHat[K]`). `gammaHat` is from item

(1) above (the values of `gammaHat` add up to 1, so the last value can be inferred). In other words, `Sigma` is a covariance matrix for the vector containing the first K-1 gammaHat, and all piHats.

4. `piHatSummary` - A data.frame with columns:

   - `r`: The central measure of risk in each risk group, taking two-stage sampling into account if necessary.
   - `piHat`: The estimates `piHat` from item (2) above.
   - `sigma`: The standard deviations of `piHat`.
   - `lower`: The lower bound of a 95% confidence interval for pi.
   - `upper`: The upper bound of a 95% confidence interval for pi.
   - `inCI`: Is `r` in the 95% confidence interval?

5. `ChiSq` - The Hosmer-Lemeshow Chi-squared goodness-of-fit statistic, and corresponding p-value.

6. The attribute diagram that accompanies `riskValidate`. By default, `riskValidate` accompanies its results with an attribute diagram. (To tell `riskValidate` to omit the attribute diagram, set the `rvpar` argument equal to `FALSE`.)

   The theoretical confidence intervals for `piHat` are represented by solid vertical lines together with short horizontal lines, in the shape of an upper case "I", and bootstrap confidence intervals are represented by lighter-colored rectangles.

   The lower box contains possibly three lines of text to help identify the call that produced the attribute diagram. The first line (left blank here) allows the user to add a comment. The second line shows the date and time when the graphic was drawn. The third line shows the first part of the function call that produced this graphic.

TOP

8.5 How to call `attributeDiagram` to show multiple risk models on the same plot

The `attributeDiagram` function is useful for drawing attribute diagrams for multiple risk models on the same plot using different colors.

Below, we define the results of two calls to `riskValidate`. The only difference in the two calls is the number of risk groups in the `riskGroup` argument. We then draw an attribute diagram for both models (`r1` and `r2`). We also demonstrate some of the `rvpar` options. For more information about graphing options, see `help(rvparFn)`.

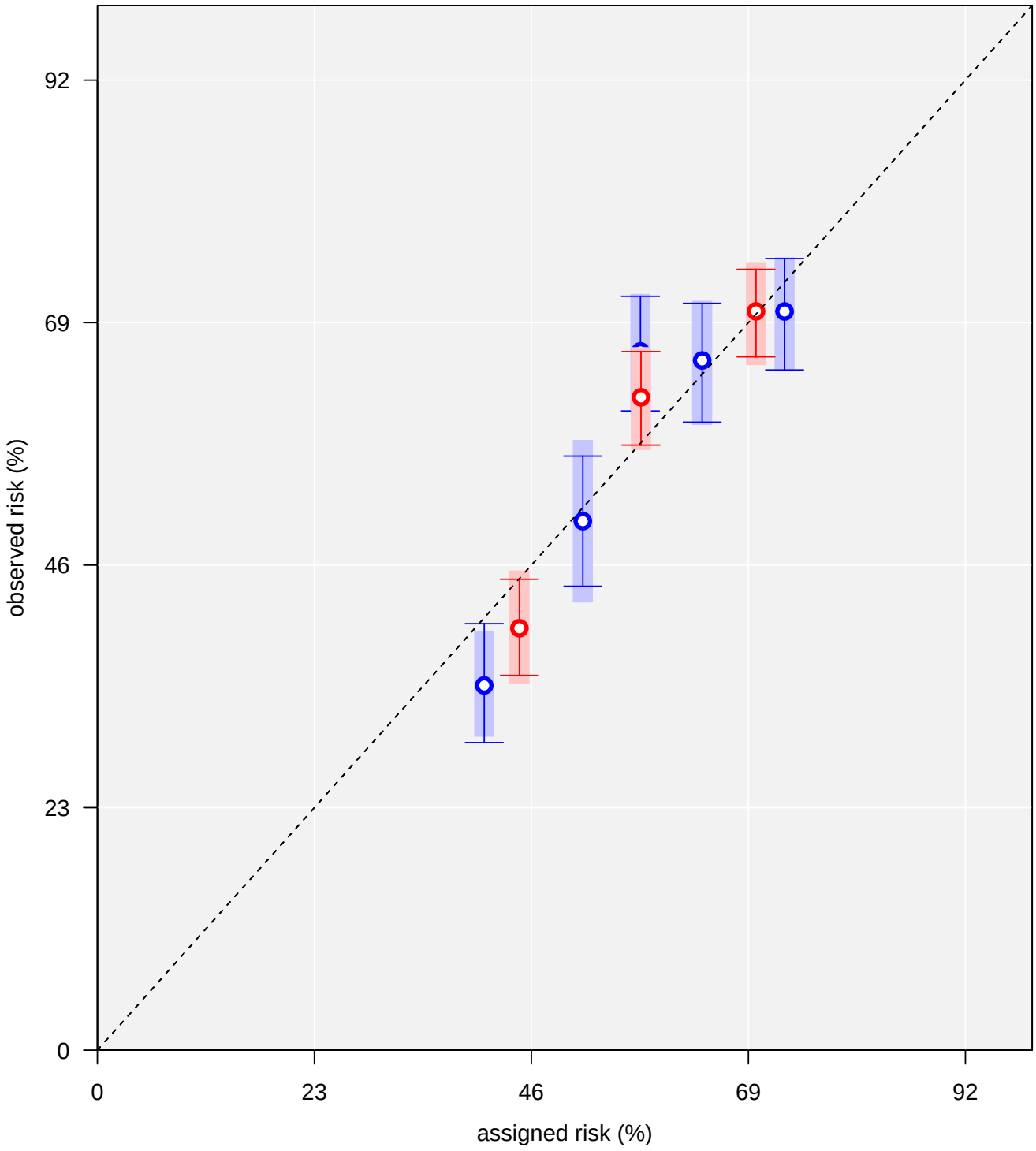
```
rv1 = riskValidate(
  e = my_e, t = my_t, r = my_r, design = my_design,
  riskGroup = list(K = 5), rSummary = my_rSummary,
  bootstrap = my_bootstrap, rvpar = FALSE)
```

```
## Note: No plot produced.  If an attribute diagram is desired,
## rvparFn() should be used to set the argument 'rvpar'
```

```
rv2 = riskValidate(
  e = my_e, t = my_t, r = my_r, design = my_design,
  riskGroup = list(K = 3), rSummary = my_rSummary,
  bootstrap = my_bootstrap, rvpar = FALSE)
```

```
## Note: No plot produced.  If an attribute diagram is desired,
## rvparFn() should be used to set the argument 'rvpar'
```

```
attributeDiagram(
  rvs = list(rv1, rv2),
  rvpar = rvparFn(col = c("blue", "red"),
  comment = "rv1 and rv2"))
```



rv1 and rv2
2015−03−08 22:55:06
attributeDiagram(rvs = list(rv1, rv2), rvpar = rvparFn(col = c("blue",

TOP

8.6 How to call `riskValidateUngrouped`

The following shows the function header of `riskValidateUngrouped`.

```
riskValidateUngrouped(
       e, t, r, design = "randomSample",
       riskGroup,
       bootstrap = FALSE, rvpar = rvparFn(),
       multicore = FALSE, verbose = FALSE)
```

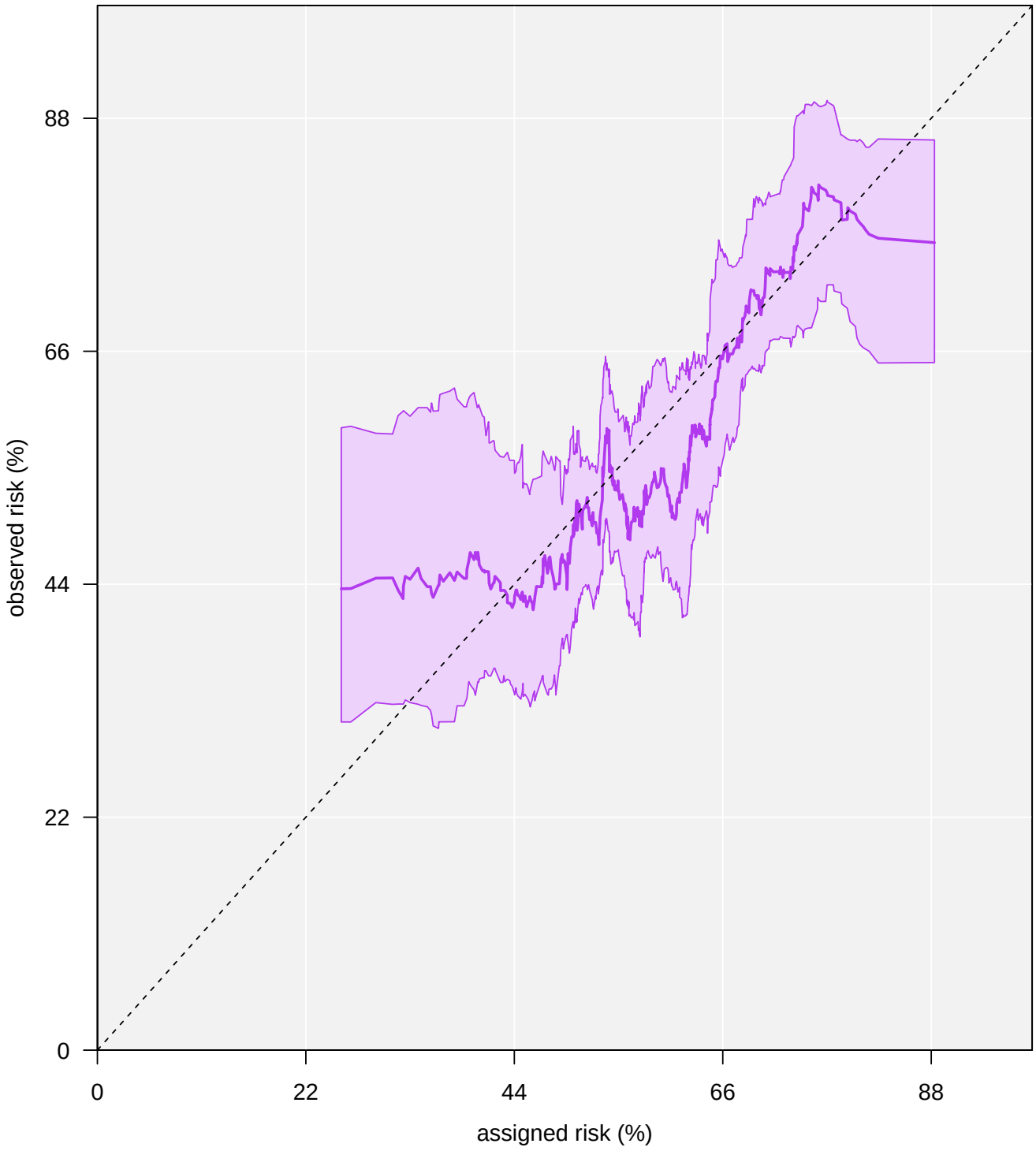
TOP

8.7 A complete `riskValidateUngrouped` example

Below is an example of a complete risk validation ungrouped analysis using some simulated data.

```
set.seed(1)
sampleData = df_randomSample_r1_r2(NTotal = 500)
epsilon = nrow(sampleData)^(-1/3)
tStar = 10

rvu = riskValidateUngrouped(
  e = sampleData$e, t = sampleData$t, r = sampleData$r1,
  design = "randomSample",
  riskGroup = list(
    ungrouped = list(epsilon = epsilon, tStar = tStar)),
  bootstrap = 20, rvpar = rvparFn(),
  multicore = FALSE, verbose = FALSE)
```



2015−03−08 22:55:57
riskValidateUngrouped(e = sampleData$e, t = sampleData$t, r = sampleData$r1,

```
rvu
```

```
##
```

```
## Head of CRP's:
## [1] 0.215 0.221 0.761 0.429 0.975 0.724
##
## Head of Nearest Neighbor piHat estimates:
##        rho piHatNN
## [1,] 0.257   0.436
## [2,] 0.267   0.436
## [3,] 0.294   0.446
## [4,] 0.312   0.446
## [5,] 0.317   0.434
## [6,] 0.323   0.426
##
## Head of Nearest Neighbor piHat bootstrap 95% confidence band:
##        2.5% 97.5%
## rho_1 0.310 0.588
## rho_2 0.310 0.589
## rho_3 0.328 0.583
## rho_4 0.327 0.582
## rho_5 0.327 0.599
## rho_6 0.327 0.603
```

TOP

8.8 Explaining `riskValidateUngrouped` output

The pretty output for `riskValidateUngrouped` shows the following items.

1. `Head of CRP's` - The first few elements of the case risk percentiles. There is one CRP at each sorted distinct value of the assigned risks.

2. `Head of Nearest Neighbor piHat estimates` - The first few rows of an array with columns `rho` and `piHat`. `rho` is the vector of sorted and distinct values of assigned risks, and `piHat` is the nearest neighbor estimate of probability of disease at each value of `rho`.

3. `Head of Nearest Neighbor piHat bootstrap 95% confidence band` - The first few rows of an array with columns `2.5%` and `97.5%` which contain the lower and upper bounds of bootstrap 95% confidence intervals, one interval for each value in `rho`.

   Only the heads are displayed because the entire rvu data structure is very large.

4. The IAD that accompanies `riskValidateUngrouped` - By default, `riskValidateUngrouped` accompanies its results with an individualized attribute diagram. The darker colored line shows `piHat`, and the lighter colored region shows a 95% (nonsimultaneous) confidence band for `pi`.

TOP

8.9 How to call IAD to show multiple risk models on the same plot

The IAD function is useful for drawing IADs (Individualized Attribute Diagrams) for multiple risk models on the same plot using different colors.

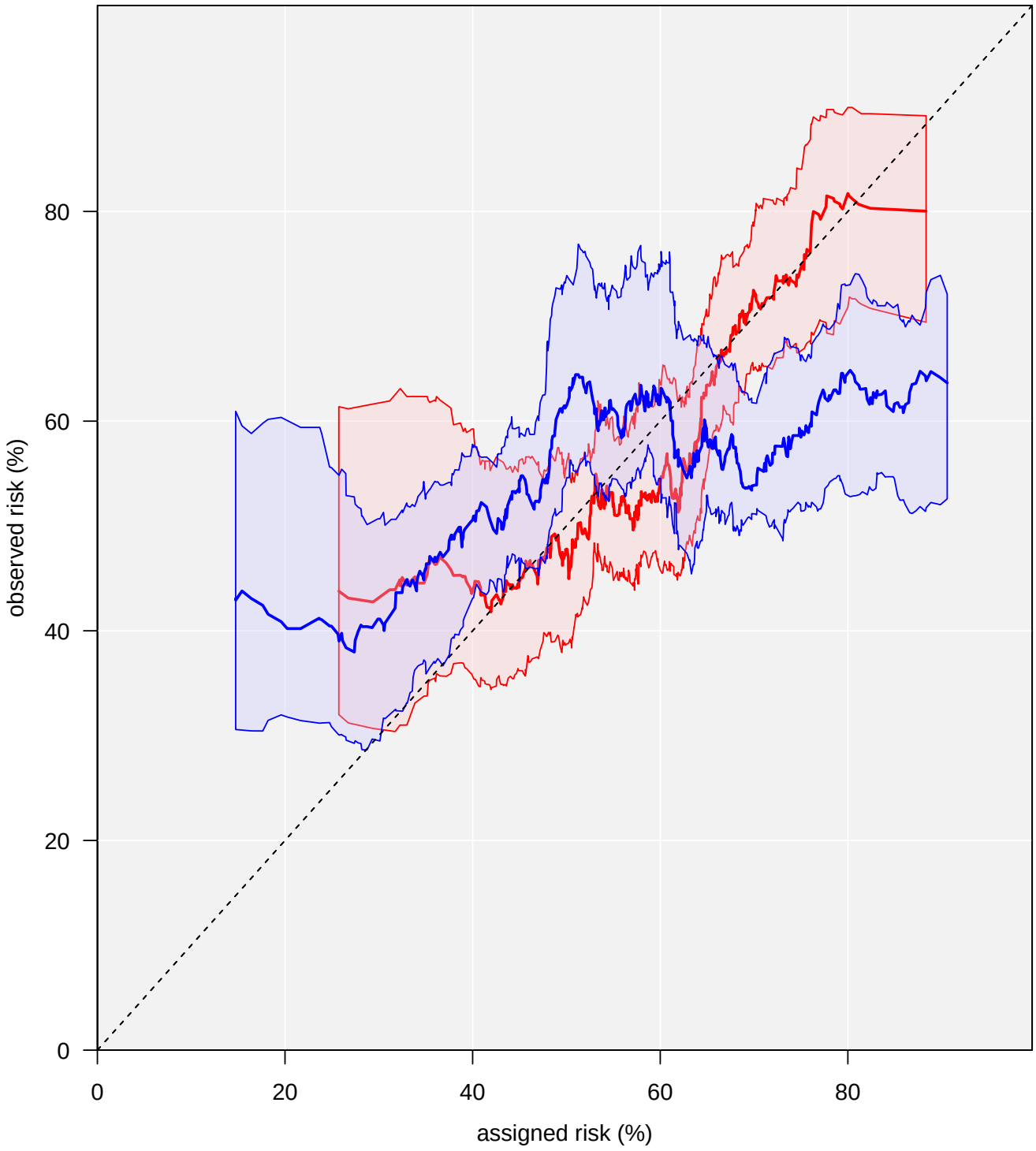
Below, we define the results of two calls to riskValidateUngrouped. The only difference in the two calls is one uses r = sampleData$r1 and the other uses r = sampleData$r2.

```
set.seed(1)
sampleData = df_randomSample_r1_r2(NTotal = 500)
riskGroup = list(
  ungrouped = list(epsilon = 0.15, tStar = 10))

rvu1b = riskValidateUngrouped(
  e = sampleData$e, t = sampleData$t, r = sampleData$r1,
  design = "randomSample", riskGroup = riskGroup, bootstrap = 20,
  rvpar = FALSE, multicore = FALSE, verbose = FALSE)

rvu2b = riskValidateUngrouped(
  e = sampleData$e, t = sampleData$t, r = sampleData$r2,
  design = "randomSample", riskGroup = riskGroup, bootstrap = 20,
  rvpar = FALSE, multicore = FALSE, verbose = FALSE)

IAD(
  list(rvu1b, rvu2b),
  rvpar = rvparFn(
    col = c("red", "blue"),
    atX = seq(0, 100, 20), atY = seq(0, 100, 20),
    comment = "20 bootstraps for confidence bands"))
```



20 bootstraps for confidence bands
2015−03−08 22:57:55
IAD(rvus = list(rvu1b, rvu2b), rvpar = rvparFn(col = c("red",

TOP

9 caseRiskPercentiles

This function produces case risk percentiles and for comparing two risk models scatterplots of case risk percentiles. This function assumes random samples (not two-stage samples).

TOP

9.1 How to call caseRiskPercentiles

The following is the function header of `caseRiskPercentiles`

```
caseRiskPercentiles(cutoff, e, t, tStar, r, ...)
caseRiskPercentiles(cutoff, e, t, tStar, r, rAnother, ...)
```

`cutoff` is a number between 0 and 1. `e`, `t`, `tStar` and the risks `r` and `rAnother` are described in Function-arguments-for-riskValidate-and-riskValidateUngrouped.

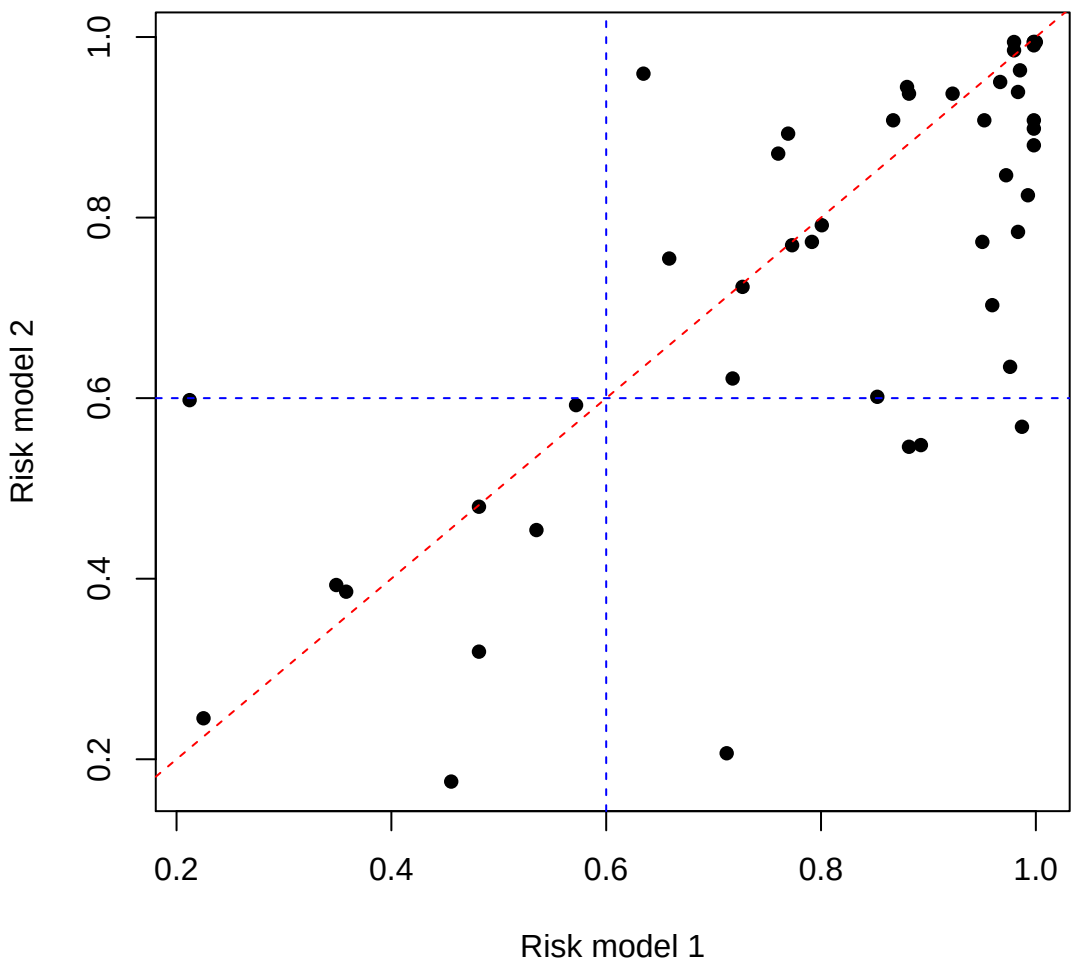
TOP

9.2 A caseRiskPercentiles example

```
data(data_set_score_statistics)
xxx = data_set_score_statistics
tail(xxx)
```

```
##      e    t    r_1 Lambda_outcome_1 Lambda_mortality_1    r_2 Lambda_outcome_2 Lambda_mo
## 995  0  9.8 0.0182           0.0185             0.0413 0.0118           0.0119
## 996  0 10.0 0.0239           0.0247             0.0420 0.0280           0.0290
## 997  0 10.0 0.1572           0.1748             0.0420 0.0576           0.0607
## 998  0 10.0 0.0379           0.0394             0.0420 0.0193           0.0199
## 999  0 10.0 0.0335           0.0348             0.0420 0.0328           0.0341
## 1000 0 10.0 0.0391           0.0407             0.0420 0.0437           0.0457
```

```
cutoff = 0.6
tStar = 10
e = xxx$e
t = xxx$t
r = xxx$r_1
rAnother = xxx$r_2
crp = caseRiskPercentiles(
  cutoff, e, t, tStar, r, rAnother,
  main = "Case risk percentiles",
  xlab = "Risk model 1", ylab = "Risk model 2",
  cex = 1, pch = 16)
```

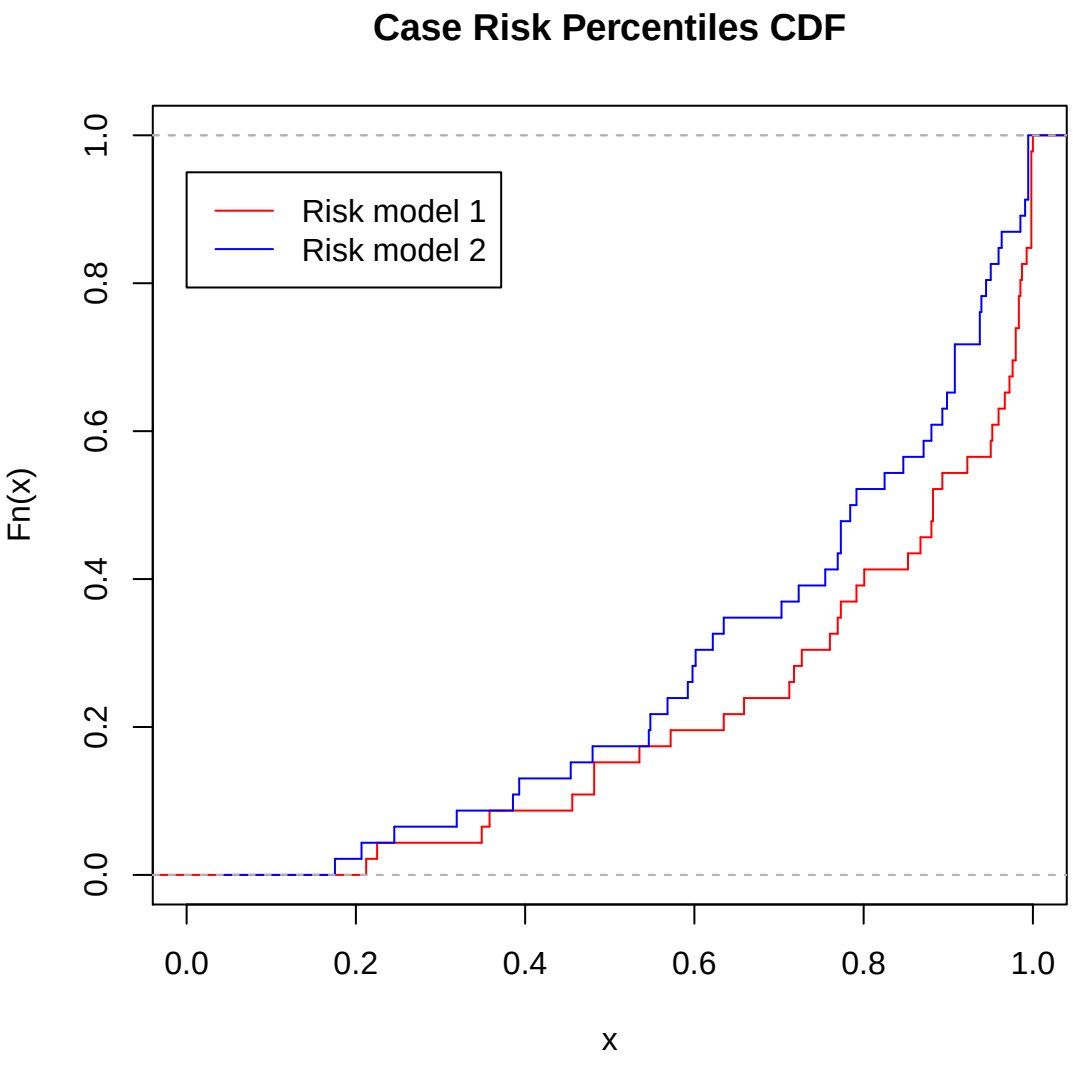



```
crp
```

```
## $N_cases
## [1] 46
##
## $the_proportion_of_points_below_diagonal
## [1] 0.674
##
## $the_proportion_of_crps_above_cutoff_for_model_r
## [1] 0.804
##
## $the_proportion_of_crps_above_cutoff_for_model_rAnother
## [1] 0.717
##
##
## Head of Model 1 caseRiskPercentiles:
## [1] 0.983 0.987 0.482 0.998 0.852 0.923
##
## Head of Model 2 caseRiskPercentiles:
```

```
## [1] 0.939 0.568 0.319 0.899 0.601 0.937
```

```
F1 = ecdf(crp$crp1)
F2 = ecdf(crp$crp2)
plot(F1, verticals = TRUE, do.points = FALSE, col = "red",
     xlim = c(0, 1),
     main = "Case Risk Percentiles CDF")
plot(F2, verticals = TRUE, do.points = FALSE, col = "blue", add = TRUE)
legend(x = 0, y = 0.95, legend = c("Risk model 1", "Risk model 2"),
       col = c("red", "blue"),
       lty = c(1, 1))
```




```
crp = caseRiskPercentiles(cutoff, e, t, tStar, r)
crp
```

```
## $N_cases
## [1] 46
##
```

```
## $the_proportion_of_crps_above_cutoff
## [1] 0.804
##
##
## Head ofcaseRiskPercentiles:
## [1] 0.983 0.987 0.482 0.998 0.852 0.923
```

TOP

9.3 Explaining `caseRiskPercentiles` output

If `caseRiskPercentiles` is called with two risk models,

```
crp = caseRiskPercentiles(cutoff, e, t, tStar, r, rAnother)
```

it returns a list containing the following

1. A scatterplot containing the points (x, y) = (the n-th case's risk percentile assigned by `r` among the controls, the n-th cases's risk percentile assigned by `rAnother` among the controls). A dotted red line is drawn on the diagonal (x = y), and dotted blue lines are drawn to indicate the `cutoff` (x = `cutoff` and y = `cutoff`).
2. `N_cases` the number of cases found. Here, cases are defined as outcome positive persons (`e = 1`).
3. `the_proportion_of_points_below_diagonal`. This number exceeding `0.5` indicates that the risk model `r` discriminates better than `rAnother`.
4. `the_proportion_of_crps_above_cutoff_for_model_r` or `the_proportion_of_crps_above_cutoff`. This is the proportion of the crps of model r that exeed `cutoff`.
5. `the_proportion_of_crps_above_cutoff_for_model_rAnother`. This is the proportion of the crps of model rAnother that exceed `cutoff`.
6. crp1. This is a vector of length N_cases containing the crps of model r.
7. crp2. This is a vector of length N_cases containing the crps of mode rAnother.

TOP

10 `performanceDifference`

This function predictive-power-positive and predictive-power-negative statistics between two models. Both estimates are for Model 2 - Model 1.

TOP

10.1 How to call performanceDifference

Both models need to be using the same data set. That is, both models should share vectors e and t, and if two-stage sampling, c. Whereas riskValidate requires one assigned risk vector and one risk group assignment, specified by the arguments r and riskGroup respectively, performanceDifference requires two assigned risk vectors and two risk group assignments, and these are specified by the arguments rs and riskGroups respectively.

rs must be a list with two named elements: r1 and r2.

riskGroups must be a list with two named elements. The first element can be named K1, k1 or cutoffs1. The second element can be named K2, k2 or cutoffs2. The value in each of the elements of riskGroups is defined analogously to the argument riskGroup from Section Function arguments for riskValidate and riskValidateUngrouped

The following is the function header for performanceDifference.

```
performanceDifference = function(
    e, t, rs, design = "randomSample",
    riskGroups, bootstrap = 100)
```

TOP

10.2 A performanceDifference example

Below we generate a simulated dataset with two risk assignment columns using the function df_twoStage_r1_r2. This simulated dataset is two-stage data. The function df_twoStage_r1_r2 returns a list with three elements: d, the rectangular dataset itself, N, the number of people in the initial sampling population, and n, the number of people who were sampled for the second stage. Simulation functions for details about this simulation function and others like it.) We then compute performanceDifference.

```
library(rmap)
set.seed(1)
twoMod = df_twoStage_r1_r2()
twoMod$N
```

```
##   A   B
## 419 581
```

```
head(twoMod$d)
```

```
##     e     t c    r1    r2
## 446 1 1.960 A 0.600 0.413
## 975 1 3.544 A 0.746 0.677
## 232 0 0.553 B 0.617 0.526
## 799 1 2.949 A 0.483 0.478
## 361 2 1.932 B 0.635 0.282
## 963 0 1.977 B 0.662 0.735
```

```
pd = performanceDifference(
  e = twoMod$d$e,
  t = twoMod$d$t,
  rs = list(r1 = twoMod$d$r1, r2 = twoMod$d$r2),
  design = list(c = twoMod$d$c, N = twoMod$N),
  riskGroup = list(K1 = 2, K2 = 2),
  bootstrap = 1000)
pd
```

```
## $PPP_diff
## [1] -0.0333
##
## $ci_PPP_diff
##   lower   upper
## -0.0883  0.0218
##
## $PPN_diff
## [1] -0.0254
##
## $ci_PPN_diff
##   lower   upper
## -0.0669  0.0161
```

It may take a few minutes for 1000 bootstrap replications to complete - if R seems to have frozen, it is probably still busy computing. The line `set.seed(1)` ensures that the `twoMod` simulated dataset is reproducible (if the seed is reset to 1 before redefining `twoMod`).

TOP

10.3 Explaining `performanceDifference` output

Below is an explanation of the `performanceDifference` output:

1. `PPP_diff` - PPP of model 2 - PPP of model 1 (where PPP is 'predictive power positive')
2. `ci_PPP_diff` - A bootstrap 95% confidence interval for `PPP\_diff`
3. `PPN_diff` - PPN of model 2 - PPN of model 1 (where PPN is 'predictive power negative')
4. `ci_PPN_diff` - A bootstrap 95% confidence interval for `PPN\_diff`

TOP

11 Composite plots

rmap has tools for creating composite plots. Any of the above graphics can be included on a single page containing a grid of these graphics. We include the example below to show the kinds of things rmap can display.

```
set.seed(1)
sampleData = df_randomSample_r1_r2(NTotal = 200)

rv1 = riskValidate(
  e = sampleData$e, t = sampleData$t, r = sampleData$r1,
  design = "randomSample",
  riskGroup = list(K = 5), rSummary = "mean",
  bootstrap = 20, rvpar = FALSE)
```

```
## Note: No plot produced.  If an attribute diagram is desired,
## rvparFn() should be used to set the argument 'rvpar'
```

```
rv2 = riskValidate(
  e = sampleData$e, t = sampleData$t, r = sampleData$r2,
  design = "randomSample",
  riskGroup = list(K = 5), rSummary = "mean",
  bootstrap = 20, rvpar = FALSE)
```

```
## Note: No plot produced.  If an attribute diagram is desired,
## rvparFn() should be used to set the argument 'rvpar'
```

```
rvu1 = riskValidateUngrouped(
  e = sampleData$e, t = sampleData$t, r = sampleData$r1,
  design = "randomSample",
  riskGroup = list(
    ungrouped = list(epsilon = 0.15, tStar = 10)),
  bootstrap = 20, rvpar = FALSE,
  multicore = FALSE, verbose = FALSE)

rvu2 = riskValidateUngrouped(
  e = sampleData$e, t = sampleData$t, r = sampleData$r2,
  design = "randomSample",
  riskGroup = list(
    ungrouped = list(epsilon = 0.15, tStar = 10)),
  bootstrap = 20, rvpar = FALSE,
  multicore = FALSE, verbose = FALSE)

setUpTrellisFn(2, 2,
               main = "Attribute and Individualized Attribute Diagrams")
```

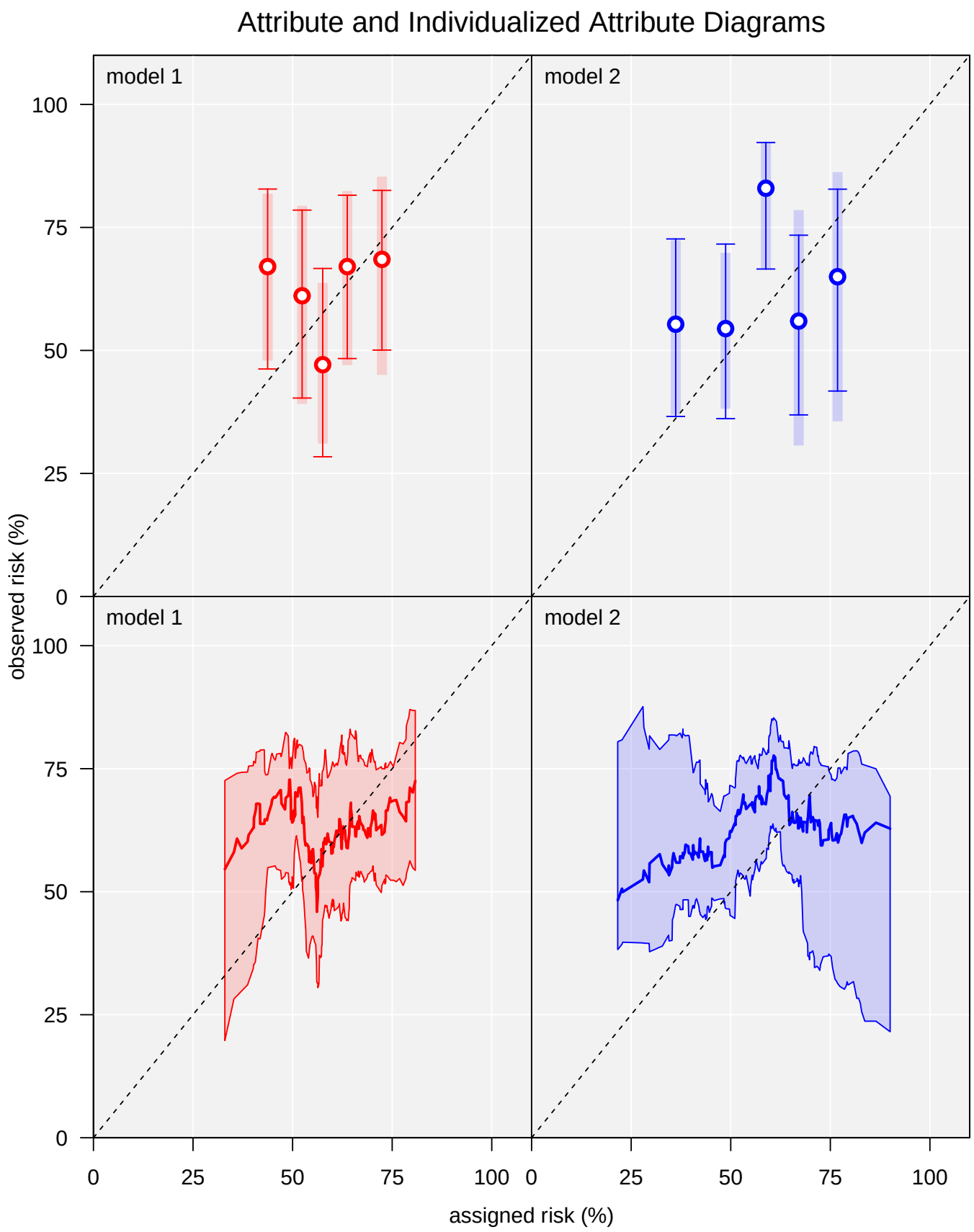
```
## $nrow
## [1] 2
##
## $ncol
## [1] 2
##
## $main
## [1] "Attribute and Individualized Attribute Diagrams"
##
## $ylab
## [1] "observed risk (%)"
##
## $xlab
## [1] "assigned risk (%)"
##
## $xmax
## [1] 1
##
## $ymax
## [1] 1
##
## $inflate
## [1] 1.1
##
## $xmaxActual
## [1] 1.1
##
## $ymaxActual
## [1] 1.1
```

```
attributeDiagramRawFn(pos = c(1, 1), rv1,
                      col = "#FF0000", lightCol = "#FF666640")
addTextToTrellisFn(pos = c(1, 1), "model 1")

attributeDiagramRawFn(pos = c(1, 2), rv2,
                      col = "#0000FF", lightCol = "#6666FF40")
addTextToTrellisFn(pos = c(1, 2), "model 2")

IAD_RawFn(pos = c(2, 1), rvu1,
          col = "#FF0000", lightCol = "#FF666640")
addTextToTrellisFn(pos = c(2, 1), "model 1")

IAD_RawFn(pos = c(2, 2), rvu2,
          col = "#0000FF", lightCol = "#6666FF40")
addTextToTrellisFn(pos = c(2, 2), "model 2")
```


Attribute and Individualized Attribute Diagrams
model 1
model 2
model 1
model 2
observed risk (%)
assigned risk (%)
0
25
50
75
100

TOP

12 Simulation functions

There are four functions that are included in the package to generate simulated personal risk model datasets. (See rmap-simulatedData-v01.pdf for a detailed explanation of the models used in these functions.) Each function creates a different type of dataset. For default usage, each function can be called with no arguments (and the default values will be used instead). You may wish to specify the number of subjects to be included in a dataset. You can do that using the `NTotal` argument in all four functions.

1. `df_randomSample`:

   This function creates a randomly sampled dataset. There is no two-stage sampling. `df_randomSample` generates a data frame with columns `e`, `t`, `w`, `r`, `c`, and `k`. The column `r` is the probability of disease according to the model that was used to generate the data and therefore is the true risk.

2. `df_randomSample_r1_r2`:

   This function generates a randomly sampled dataset with two assigned risks, `r1` and `r2` (for two models). Data with two assigned risk columns is suitable for the function `performanceDifference`. The output dataframe has columns `e`, `t`, `c`, `r1`, and `r2`. The column `r1` is the true risk, and the column `r2` is a noisy version of `r1`.

3. `df_twoStage`:

   This function generates a dataset from a two-stage sampling design, where category A contains people with disease events `e = 1` and category B contains everyone else. The default probabilities of resampling are `ppp = c(A = 1, B = 0.5)`. This function outputs a list of three elements:

   - `N`, which holds the counts of the initial number of people in categories A and B (before sampling)
   - `n`, which holds the second-stage sampling counts for categories A and B.
   - `d`, which holds the data frame that we are accustomed to seeing. This data frame has columns `e`, `t`, `w`, `r`, `c`, and `k`. The column `r` is the true risk.

4. `df_twoStage_r1_r2`:

   This function generates a dataset from a two-stage sampling design with two assigned risks `r1` and `r2` (for two models). Data with two assigned risk columns is suitable for the function `performanceDifference`. Just as `df_twoStage`, this function `df_twoStage_r1_r2` outputs a list of three elements; the only difference is the third element, the data frame `d`, contains columns `e`, `t`, `c`, `r1`, and `r2`. The column `r1` is the true risk, and the column `r2` is a noisy version of `r1`.

```
set.seed(1)
data1 = df_randomSample()
head(data1)
```

```
##     e     t     w     r c k
## 446 0 6.009 0.129 0.505 A 2
## 975 1 1.960 0.178 0.600 A 3
## 656 0 1.669 0.118 0.481 A 2
## 232 1 3.544 0.313 0.746 A 5
## 799 0 0.553 0.189 0.617 A 4
## 361 1 2.949 0.119 0.483 A 2
```

```
data2 = df_randomSample_r1_r2()
head(data2)
```

```
##     e    t c    r1    r2
## 292 2 3.41 A 0.411 0.551
## 141 1 1.33 A 0.689 0.697
## 474 1 2.67 A 0.657 0.706
## 821 0 4.26 A 0.476 0.456
## 489 1 1.16 A 0.506 0.610
## 947 0 1.06 A 0.525 0.522
```

```
data3 = df_twoStage()
data3$N
```

```
##   A   B
## 465 535
```

```
head(data3$d)
```

```
##     e     t     w     r c k
## 17  1 5.592 0.215 0.653 A 4
## 512 1 5.680 0.183 0.609 A 4
## 262 1 2.240 0.638 0.864 A 5
## 124 1 0.477 0.194 0.625 A 4
## 398 1 3.733 0.368 0.779 A 5
## 219 2 0.736 0.385 0.787 B 5
```

```
data4 = df_twoStage_r1_r2()
data4$N
```

```
##   A   B
## 418 582
```

```
head(data4$d)
```

```
##     e     t c    r1    r2
## 441 1 2.680 A 0.809 0.847
## 510 1 2.886 A 0.614 0.781
## 787 1 1.165 A 0.722 0.498
## 450 1 0.321 A 0.662 0.738
## 151 1 1.230 A 0.649 0.527
## 945 1 1.042 A 0.706 0.800
```

TOP

13 Useful downloads including additonal documentation

data_set_score_statistics.csv

datafRandomSample.csv

RStudio file containing all the examples in this document

R file containing all the examples in this document

A pdf version of this html page

The mathematical formula behind the functions used in rmap

An in-depth examination of some of the more involved data structures in this package including B2, V2Stage, Sigma, etc.

An explanation of the models used in the functions that simulate data for this package

score-statistics-formulas-v01.pdf

TOP