Package: Qindex (via r-universe)

September 15, 2024

Type Package

Title Continuous and Dichotomized Index Predictors Based on Distribution Quantiles

Version 0.1.5

Date 2023-10-17

Author Tingting Zhan [aut, cre, cph] (<https://orcid.org/0000-0001-9971-4844>), Misung Yi [aut, cph] (<https://orcid.org/0000-0002-4007-5408>), Inna Chervoneva [aut, cph] (<https://orcid.org/0000-0002-9104-4505>)

Maintainer Tingting Zhan <tingtingzhan@gmail.com>

Description Select optimal functional regression or dichotomized quantile predictors for survival/logistic/numeric outcome and perform optimistic bias correction for any optimally dichotomized numeric predictor(s), as in Yi, et. al. (2023) <doi:10.1016/j.labinv.2023.100158>.

LazyData true

LazyDataCompression xz

RoxygenNote 7.2.3

Encoding UTF-8

License GPL-2

Depends R (>= 4.2),

Language en-US

Imports grDevices, matrixStats, methods, mgcv, pracma, rpart, stats, survival

Suggests knitr, boot

NeedsCompilation no

Date/Publication 2023-10-17 19:20:02 UTC

Repository https://tingtingzhan.r-universe.dev

RemoteUrl https://github.com/cran/Qindex

RemoteRef HEAD

RemoteSha 0259cdc85b81e58ba347b9d0612fdcd7eea55f53

Contents

Qindex-package . . . 2
BBC_dichotom . . . 3
celldata . . . 6
clusterQp . . . 7
FRindex . . . 8
nlFRindex . . . 12
optimSplit_dichotom . . . 15
rpartD . . . 18
rSplit . . . 20

Index **22**

`Qindex-package` *Continuous and Dichotomized Index Predictors Based on Distribution Quantiles*

Description

Primary functions in this package are

`optimSplit_dichotom()` optimal dichotomizing predictor(s) selection via dichotomizing split sample **Still need? Select optimal functional regression or dichotomized quantile predictors for survival/logistic/numeric outcome**

`BBC_dichotom()` Bootstrap-based optimism correction for dichotomizing selected predictor(s) **Still need? perform optimism correction for any optimal dichotomizing predictor(s)**

`clusterQp()` calculate user-selected sample quantiles in each cluster of observations.

`FRindex()` Functional regression index as a predictor in the functional regression model

References

Selection of optimal quantile protein biomarkers based on cell-level immunohistochemistry data. Misung Yi, Tingting Zhan , Amy P. Peck, Jeffrey A. Hooke, Albert J. Kovatich, Craig D. Shriver, Hai Hu, Yunguang Sun, Hallgeir Rui and Inna Chervoneva. Under revision

Quantile index biomarkers based on single-cell expression data. Misung Yi, Tingting Zhan, Amy P. Peck, Jeffrey A. Hooke, Albert J. Kovatich, Craig D. Shriver, Hai Hu, Yunguang Sun, Hallgeir Rui and Inna Chervoneva. Laboratory Investigation, 2023. doi:10.1016/j.labinv.2023.100158

BBC_dichotom *Bootstrap-based Optimism Correction for Dichotomization*

Description

Functions explained in this documentation are,

`BBC_dichotom()` to obtain a multivariable regression model with bootstrap-based optimism correction on the dichotomized predictors.

`optimism_dichotom()` a helper function to compute the bootstrap-based optimism of the dichotomized predictors.

`coef_dichotom()` a helper function to obtain the estimated multivariable regression coefficients of the dichotomized predictors.

Usage

```
BBC_dichotom(formula, dichotom, data, ...)

optimism_dichotom(formula, X, data, R = 100L, ...)

coef_dichotom(formula, dX, data)
```

Arguments

`formula`	formula, left-hand-side being the response y and right-hand-side being the predictors *in addition to* the predictors to be `dichotomized`. If there is no additional predictor, use `y ~ 1`
`dichotom`	one-sided formula of the set of predictors to be dichotomized. These predictors can be stored in `data` as one or more numeric columns and/or one matrix column
`data`	data.frame, containing the response y and predictors in `formula`, as well as the predictors to be `dichotomized`
`...`	additional parameters, currently not in use
`X`	(for helper function `optimism_dichotom()`) numeric matrix of k columns, a set of k numeric predictors
`R`	positive integer scalar, number of bootstrap replicates R, default `100L`
`dX`	(for helper function `coef_dichotom()`) logical matrix of k columns, a set of k dichotomized predictors

Details

Function `BBC_dichotom()` obtains a multivariable regression model with bootstrap-based optimism correction on the dichotomized predictors. Specifically,

1. Dichotomize the k predictors in the *entire data* (using function `m_rpartD()`). Fit a regression model to the entire data with the k dichotomized predictors as well as the additional predictors, if any (using helper function `coef_dichotom()`). The estimated regression model is referred to as the *apparent performance*.
2. Obtain the bootstrap-based optimism based on R copies of bootstrap samples, using optimism_dichotom. Calculate the median of bootstrap-based optimism, specific to each of the dichotomized predictors. In future, we may expand the options to include the use of trimmed-mean mean.default`(, trim)`, etc. For now, let's refer to the median optimism as the *optimism-correction* of the k dichotomized predictors.

Subtract the optimism-correction (in Step 2) from the apparent performance estimates (in Step 1), *only for the k dichotomized predictors*. The apparent performance estimates for the additional predictors, if any, are not modified. The variance-covariance (vcov) estimates of the apparent performance is not modified, for now. None of the other regression model diagnostics, such as residuals, logLikelihood, etc., are modified neither, for now. The coefficient-only, partially-modified regression model is referred to as the *optimism-corrected performance*.

Value

Function BBC_dichotom returns a coxph, glm or lm regression model, with attributes,

`attr(,'optimism')` the returned object from optimism_dichotom

`attr(,'apparent_cutoff')` a double vector, cutoff thresholds for the k predictors in the apparent model

Details of Helper Function `optimism_dichotom()`

Function optimism_dichotom computes the bootstrap-based optimism of the dichotomized predictors. First, R bootstrap samples are generated, for which the end-user may specify a Random seed, if needed. Then,

1. From each of the R bootstrap samples, obtain the dichotomizing branches for the k predictors to be dichotomized, using function `m_rpartD()`
2. Dichotomize the k predictors in each *bootstrap sample* using the respective dichotomizing branches from Step 1. The regression coefficients estimate for the k dichotomized predictors (using helper function `coef_dichotom()`) is referred to as the *bootstrap performance estimate*.
3. Dichotomize the k predictors in the *entire data* using each of the bootstrap dichotomizing branches from Step 1. The regression coefficients estimate for the k dichotomized predictors (using helper function `coef_dichotom()`) is referred to as the *test performance estimate*.

The difference between the bootstrap and test performance estimates, based on each of the R bootstrap samples, are referred to as the bootstrap-based *optimism* or optimistic bias.

Details of Helper Function `coef_dichotom()`

Function coef_dichotom obtains the estimated multivariable regression coefficients of the dichotomized predictors. A Cox proportional hazards (coxph) regression for Surv response, a logistic (glm) regression for logical response, or a linear (lm) regression for gaussian response is performed with

- the dichotomous logical predictors, given as the columns of `dX`, and
- the additional predictors specified in `formula`

When `dX` has duplicated columns, the regression model is fitted using the *unique* columns of `dX` and the additional predictors in `formula`. The returned coefficient estimates repeat the corresponding estimates of the unique columns of `dX`.

Returns of Helper Functions

Helper function `optimism_dichotom()` returns an $R \times k$ double matrix of bootstrap-based optimism, with attributes

`attr(,'cutoff')` an $R \times k$ double matrix, the R copies of bootstrap cutoff thresholds for the k predictors. See attribute `'cutoff'` of function `m_rpartD()`

Helper function `coef_dichotom()` returns a double vector of the coefficients of the dichotomized predictors, with attributes

`attr(,'model')` the coxph, glm or lm regression model

References on Helper Function `optimism_dichotom()`

Ewout W. Steyerberg (2009) Clinical Prediction Models. doi:10.1007/9780387772448

Frank E. Harrell Jr., Kerry L. Lee, Daniel B. Mark. (1996) Multivariable prognostic models: issues in developing models, evaluating assumptions and adequacy, and measuring and reducing errors. doi:10.1002/(SICI)10970258(19960229)15:4<361::AIDSIM168>3.0.CO;24

Examples

```
library(survival)
data(flchain, package = 'survival') # see more details from ?survival::flchain
head(flchain2 <- within.data.frame(flchain, expr = {
  mgus = as.logical(mgus)
}))
dim(flchain3 <- subset(flchain2, futime > 0)) # required by ?rpart::rpart
dim(flchain_Circulatory <- subset(flchain3, chapter == 'Circulatory'))

m1 = BBC_dichotom(Surv(futime, death) ~ age + sex + mgus,
 data = flchain_Circulatory, dichotom = ~ kappa + lambda)
summary(m1)
attr(attr(m1, 'optimism'), 'cutoff')
attr(m1, 'apparent_cutoff')
```

`celldata` *Ki67 Data*

Description

Ki67 cell data containing 622 patients

Usage

```
Ki67
```

Format

`PATIENT_ID` factor, unique patient identifier

`tissueID` factor, TMA core identifier

`RECURRENCE` integer, recurrence indicator, 1 = Recurred, 0 = not Recurred

`RECFREESURV_MO` integer, recurrence-free survival time in months

`Marker` double, cell signal intensity of the protein immunofloerscence signal

`inner_x` integer, x-coordinate in the cell centroid in the TMA core

`inner_y` integer, y-coordinate in the cell centroid in the TMA core

`AGE_AT_DX` integer, age at diagnosis

`Tstage` integer, tumor stage

`NodeSt` integer, node stage, -1 = unknown, 0 = Node Negative, 1 = Node Positive

`HRpos` integer, indicator of hormone positive status (ER+ or PR+), 1 = positive, 0 = negative

`HistologicalGrade` integer, histology grade

`Her2_path_qIF` integer, Her2 status, 1 = positive, 0 = negative

`RACE` character, race, White, Black, Asian, Native Hawaiian or Other Pacific Islander, American Indian or Alaska Native, Unknown

`RadjCHEMO` integer, adjuvant chemo treatment, 0 = unknown, 1 = done, 2 = NOT done

`RadjRAD` integer, adjuvant radiation treatment, 0 = unknown, 1 = done, 2 = NOT done

`HORM_4cat` integer, hormone treatment, 0 = unknown, 1 = not indicated, 2 = done, 3 = recommended, but not done

`MSI` double, mean signal intensity (mean over all cells in the TMA core)

clusterQp *Cluster-Specific Sample Quantiles*

Description

Obtain vectors of sample quantiles in each cluster of observations

Usage

```
clusterQp(
  formula,
  data,
  exclude,
  from = 0.01,
  to = 0.99,
  by = 0.01,
  type = 7,
  ...
)
```

Arguments

formula	formula passed to aggregate.formula. To calculate the cluster-specific statistics for response y, the user may use `y ~ id` to retain only the cluster `id` in the returned value `y ~ id + x1 + x2` to retain the cluster `id` and cluster-specific variables x_1 and x_2 in the returned value `y ~ .` to retain all (supposedly cluster-specific) variables from `data` in the returned value
data	data.frame
exclude	(optional) formula or character vector, (supposedly non-cluster-specific) variables to be excluded from aggregation. To remove variables z_1 and z_2, the user may use either • `exclude = c('z1', 'z2')`; or • `exclude = . ~ . - z1 - z2`
from, to, by	double scalars, the starting, end, and increment values to specify a sequence of probabilities $p = (p_1, \cdots, p_N)'$ for the sample quantiles $q = (q_1, \cdots, q_N)'$
type	integer scalar, type of quantile algorithm
...	additional parameters, currently not in use

Details

Function `clusterQp()` calculates N sample quantiles in each aggregated cluster of observations. The aggregation is specified by parameters `formula` and `exclude`.

Value

Function `clusterQp()` returns an aggregated data.frame. A double matrix of N columns is created to store the sample quantiles q of each aggregated cluster. The column names of this quantile matrix are the probabilities p.

Examples

```
Ki67q = clusterQp(Marker ~ ., data = Ki67, exclude = c('tissueID','inner_x','inner_y'))
tmp = clusterQp(Marker ~ ., data = Ki67, exclude = . ~ . - tissueID - inner_x - inner_y)
# stopifnot(identical(Ki67q, tmp))
# stopifnot(!anyDuplicated.default(Ki67q$subjID))
head(Ki67q)
sapply(Ki67q, FUN = class)
```

FRindex *Functional Regression Indices & Weights*

Description

Functions explained in this documentation are,

`FRindex()` to compute the functional regression indices and weights based on the functional predictors.

`predict.FRindex()` to compute the predicted values based on functional regression indices and weights model.

`FR_gam()` a helper function to fit a functional regression model using generalized additive models with integrated smoothness estimation (gam).

Usage

```
FRindex(formula, data, sign_prob = 0.5, ...)

FR_gam(
  formula,
  data,
  xarg = as.double(colnames(X)),
  family,
  knot_pct = 0.4,
  knot.value = ceiling(length(xarg) * knot_pct),
  ...
)

## S3 method for class 'FRindex'
predict(
  object,
  newdata = object@data,
```

```
  newX = newdata[[object@formula[[3L]]]],
  new_xarg = as.double(colnames(newX)),
  ...
)
```

Arguments

`formula`	a two-sided formula. **Left-hand-side** is the name of the response y. Supported types of responses are double, logical and Surv. **Right-hand-side** is the name of the tabulated double matrix X of functional predictor values. Each row of X represents the tabulated values for a subject. All rows/subjects are tabulated on a common grid `xarg`. Each column of X represents the tabulated values at a point on the common grid for each subject.
`data`	data.frame, with the response y and the tabulated functional predictor values X specified in `formula`. If the functional predictor is the quantile function, then `data` is preferably the returned object of `clusterQp()`.
`sign_prob`	double scalar between 0 and 1, probability corresponding to the selected nearest-even quantile in `xarg`, which is used to define the sign of the functional regression weights. Default is `.5`, i.e., the nearest-even median of `xarg`
`...`	for function `predict.FRindex()` and helper function `FR_gam()`, these are currently not in use. For function `FRindex()`, see a detailed explanation in section **Using `...` in `FRindex()`**
`xarg`	strictly increasing double vector, the common grid on which the functional predictor values X are tabulated
`family`	family object, the distribution and link function to be used in gam. Default family for Surv response is `mgcv::cox.ph()`, for logical response is `binomial(link = 'logit')`, for double response is `gaussian(link = 'identity')`.
`knot_pct`	positive double scalar, percentage of the number of columns of X, to be used as `knot.value`. Default is 40%. If `knot.value` is provided by the end-user, then `knot_pct` is ignored.
`knot.value`	positive integer scalar, number of knots (i.e., parameter `k` in the spline smooth function s) used in gam. Default is the ceiling of `knot_pct` of the column dimension of X
`object`	an FRindex object for the predict method, the returned object from function `FRindex()`
`newdata`	data.frame, with at least the tabulated functional predictor values X^{new} based on `object@formula`
`newX`	double matrix, functional predictor values X^{new} for a set of new subjects. Each row of X^{new} represents the tabulated values for a new subject. All rows/subjects are tabulated on a common grid `new_xarg`. Each column of X^{new} represents the tabulated values at a point on the common grid for each new subject.
`new_xarg`	strictly increasing double vector, the common grid on which the functional predictor values X^{new} are tabulated. The length of `new_xarg` does not need to be the same as the length of `object@xarg`, but they must share the same range.

Details

Functional regression indices & weights model:

Function `FRindex()` defines and calculates the functional regression indices and weights in the following steps.

1. Fit a functional regression model to the response y using the functional predictor X, with tabulated tabulated on a same grid `xarg` for all subjects, using helper function `FR_gam()`
2. Select one point in the tabulating grid `xarg`. For one-dimensional domain, we select the nearest-even quantile of the tabulating grid `xarg`, corresponding to the user-specified probability `sign_prob`. Default `sign_prob = .5` indicates the median of `xarg`.
3. Obtain the fitted coefficient function $\hat{\beta}(x)$, tabulated on the grid `xarg`, using internal helper function `gam2beta()`
4. Calculate the integral of the product of the fitted coefficient function $\hat{\beta}(x)$ (from Step 3) and the functional predictor values X, using the trapzoid rule
5. Obtain the sign of the correlation between
   - the subject-specific functional predictor *values*, at the selected quantile of `xarg` (from Step 2), and
   - the subject-specific integrals from Step 4

Functional regression weights (slot `@weight`) are the tabulated weight function on the grid `xarg`. These weights are defined as the product of `sign` (from Step 5) and $\hat{\beta}(x)$ (from Step 3).

Functional regression indices (slot `@index`) are defined as the product of `sign` (from Step 5) and `intg` (from Step 4). Multiplication by `sign` is required to ensure that the resulting functional regression indices are positively associated with the functional predictor values at the selected quantile of `xarg` (from Step 2).

Predict method for functional regression indices & weights:

Function `predict.FRindex()` computes functional regression indices and weights based on the tabulated functional predictors X^{new} in a new sets of subjects. It's important that the new tabulation grid `new_xarg` must have the same range as the model tabulation grid `object@xarg`. Then,

1. Obtain the fitted coefficient function $\hat{\beta}(x^{new})$ of the existing generalized additive model `object@gam`, but tabulated on the new grid `new_xarg`, using internal helper function `gam2beta()`
2. Calculate the integral of the product of the fitted coefficient function $\hat{\beta}(x^{new})$ (from Step 1) and the new functional predictor values X^{new}, using the trapzoid rule

Predicted functional regression weights are the tabulated weight function on the new grid `new_xarg`. These weights are defined as the product of `object@sign` and $\hat{\beta}(x^{new})$ (from Step 1).

Predicted functional regression indices are defined as the product of `object@sign` and `intg` (from Step 2). Multiplication by `object@sign` is required to ensure that the resulting functional regression indices are positively associated with the functional predictor values at the selected quantile of `object@xarg`.

Value

Functional regression indices & weights model:

Function `FRindex()` returns an S4 FRindex object. The slots of S4 class FRindex are described in section **Slots**.

Predict method for functional regression indices & weights:

Function `predict.FRindex()` returns a double vector, which is the predicted functional regression indices. The returned object contains an attributes

`attr(,'weight')` double vector, the predicted functional regression weights

Slots

`formula,data,xarg` see explanations in section **Arguments**

`gam` gam object, the returned object of helper function `FR_gam()`

`sign` double scalar of either 1 or -1, see Step 5 in section **Details** on function `FRindex()`

`index,weight` double vectors, functional regression indices and functional regression weights, respectively. See section **Details** on function `FRindex()`

Using `...` in `FRindex()`

Function `FRindex()` passes the parameters `xarg`, `family`, `knot_pct` and `knot.value` into helper function `FR_gam()` through three dots `....`

The most important parameter among them is `xarg`. The default argument of the parameter `xarg` comes from the column names of the matrix of tabulated functional predictor values X. This is particularly convenient when the functional predictor is the quantile function, and `data` is the returned object of function `clusterQp()`.

Both `FRindex()` and helper function `FR_gam()` accept user-provided `xarg`. In such case, the provided values will be checked such that

1. `xarg` is a numeric vector without missingness
2. length of `xarg` is the same as the number of columns of matrix X
3. `xarg` must be strictly sorted (see is.unsorted)

Otherwise, an error message will be returned.

Details of Helper Function

Helper function `FR_gam()` uses gam to estimate the functional coefficient by fitting functional regression model.

Returns of Helper Functions

Helper function `FR_gam()` returns a gam object, with additional attributes

`attr(,'X')` double matrix of tabulated functional predictor values X

`attr(,'xarg')` double vector, see explanation of parameter `xarg`

References

Cui, E., Crainiceanu, C. M., & Leroux, A. (2021). Additive Functional Cox Model. Journal of Computational and Graphical Statistics. doi:10.1080/10618600.2020.1853550

Gellar, J. E., Colantuoni, E., Needham, D. M., & Crainiceanu, C. M. (2015). Cox regression models with functional covariates for survival data. Statistical Modelling. doi:10.1177/1471082X14565526

Examples

```
library(survival)

pt = unique(Ki67$PATIENT_ID)
length(pt) # 622
# set.seed if necessary
train_pt = sample(pt, size = 500L)
Ki67q = clusterQp(Marker ~ ., data = Ki67, exclude = c('tissueID','inner_x','inner_y'))
train_q = subset(Ki67q, PATIENT_ID %in% train_pt)
test_q = subset(Ki67q, !(PATIENT_ID %in% train_pt))
train_q$Marker = log1p(train_q$Marker)
test_q$Marker = log1p(test_q$Marker)

FRi = FRindex(Surv(RECFREESURV_MO, RECURRENCE) ~ Marker, data = train_q)
FRi@index # functional regression index
FRi@weight # functional regression weights
head(show(FRi)) # append `FRi` to the data

(FRi_test = predict(FRi, newdata = test_q))

FRi_train = predict(FRi)
# stopifnot(identical(FRi@index, c(FRi_train)),
#  identical(FRi@weight, attr(FRi_train, 'weight')))

# set.seed if necessary
Ki67bbc_v2 = BBC_dichotom(Surv(RECFREESURV_MO, RECURRENCE) ~ NodeSt + Tstage,
  data = data.frame(train_q, FRi_std = std_IQR(FRi_train)),
  dichotom = ~ FRi_std)
summary(Ki67bbc_v2)

Ki67q = clusterQp(Marker ~ ., data = Ki67, exclude = c('tissueID','inner_x','inner_y'))
Ki67q$Marker = log1p(Ki67q$Marker)

library(survival)
FR_gam(Surv(RECFREESURV_MO, RECURRENCE) ~ Marker, data = Ki67q)
```

`nlFRindex` *Nonlinear Functional Regression Indices*

Description

Functions explained in this documentation are,

`nlFRindex()` to compute the non-linear functional regression indices based on the functional predictors.

`predict.FRindex()` to compute the predicted values based on functional regression indices model.

Usage

```
nlFRindex(
  formula,
  data,
  xarg = as.double(colnames(X)),
  family,
  fit = TRUE,
  ...
)

## S3 method for class 'nlFRindex'
predict(object, newdata, ...)
```

Arguments

formula	a two-sided formula. **Left-hand-side** is the name of the response y. Supported types of responses are double, logical and Surv. **Right-hand-side** is the name of the tabulated double matrix X of functional predictor values. Each row of X represents the tabulated values for a subject. All rows/subjects are tabulated on a common grid `xarg`. Each column of X represents the tabulated values at a point on the common grid for each subject.
data	data.frame, with the response y and the tabulated functional predictor values X specified in `formula`. If the functional predictor is the quantile function, then `data` is preferably the returned object of `clusterQp()`.
xarg	numeric vector. The default argument comes from the column names of the matrix of tabulated functional predictor values X. This is particularly convenient when the functional predictor is the quantile function, and `data` is the returned object of function `clusterQp()`. The user-provided `xarg` will be checked such that 1. `xarg` is a numeric vector without missingness 2. length of `xarg` is the same as the number of columns of matrix X 3. `xarg` must be strictly sorted (see is.unsorted) Otherwise, an error message will be returned.
family	..
fit	logical scalar, see gam
...	additional parameters, currently not in use
object	an nlFRindex object for the predict method, the returned object from function `nlFRindex()`
newdata	data.frame, with at least the tabulated functional predictor values X^{new} based on `object@formula`

Details

Functional regression indices & weights model:

Function `nlFRindex()` fits a non-linear functional regression model to the response y using the functional predictor X, with values tabulated on a same grid `xarg` for all subjects (Cui et al, 2021).

Predict method for non-linear functional regression indices:

Function `predict.nlFRindex()` computes non-linear functional regression indices based on the tabulated functional predictors X^{new} in a new sets of subjects. It's important that the new tabulation grid must be exactly the same as the model tabulation grid `object@xarg`.

Value

Functional regression indices & weights model:

Function `nlFRindex()` returns an S4 nlFRindex object. The slots of S4 class nlFRindex are described in section **Slots**.

Predict method for non-linear functional regression indices:

Function `predict.nlFRindex()` returns a double vector, which is the predicted non-linear functional regression indices.

Slots

`formula,data,xarg` see explanations in section **Arguments**

`gam` gam object

`p.value` numeric scalar, p-value for the test of significance of the functional predictor

`index` double vector, functional regression indices.

References

Cui, E., Crainiceanu, C. M., & Leroux, A. (2021). Additive Functional Cox Model. Journal of Computational and Graphical Statistics. doi:10.1080/10618600.2020.1853550

Examples

```
pt = unique(Ki67$PATIENT_ID)
length(pt) # 622
# set.seed if necessary
train_pt = sample(pt, size = 500L)
Ki67q = clusterQp(Marker ~ ., data = Ki67, exclude = c('tissueID','inner_x','inner_y'))
train_q = subset(Ki67q, PATIENT_ID %in% train_pt)
test_q = subset(Ki67q, !(PATIENT_ID %in% train_pt))
train_q$Marker = log1p(train_q$Marker)
test_q$Marker = log1p(test_q$Marker)

# using Cox model
m = nlFRindex(Surv(RECFREESURV_MO, RECURRENCE) ~ Marker, data = train_q)
m@p.value # test significance of `Marker` as a functional predictor
train_index = predict(m, newdata = train_q) # non-linear FR index of training data
# stopifnot(identical(train_index, m@index))
```

```
predict(m, newdata = test_q) # non-linear FR index of test data

# using logistic regression model
nlFRindex(RECURRENCE ~ Marker, data = train_q)

# using Gaussian model
nlFRindex(RECFREESURV_MO ~ Marker, data = train_q)
```

---

optimSplit_dichotom *Optimal Dichotomizing Predictors via Repeated Sample Splits*

Description

Functions explained in this documentation are,

`optimSplit_dichotom()` to identify the optimal dichotomizing predictors using repeated sample splits.

`split_dichotom()` a helper function to perform a univariable regression model on the test set with a dichotomized predictor, using a dichotomizing rule determined by a recursive partitioning of the training set.

`quantile_split_dichotom()` a helper function to locate a quantile of multiple split_dichotom objects, based on the estimated univariable regression coefficient.

Usage

```
optimSplit_dichotom(formula, data, include, top = 1L, nsplit, ...)

split_dichotom(y, x, index, ...)

quantile_split_dichotom(y, x, indices = rSplit(y, ...), probs = 0.5, ...)
```

Arguments

`formula`	formula. Left-hand-side is the name of a Surv, logical, or double response y. Right-hand-side is the candidate numeric predictors in `data`, given either as the name of a numeric matrix column (e.g., `y ~ X`), or as the names of several numeric vector columns (e.g., `y ~ x1 + x2 + x3`)
`data`	data.frame, containing the response and predictors in `formula`
`include`	language object, inclusion criteria for the optimal dichotomizing predictors. A suggested choice is `(highX>.15 & highX<.85)` to guarantee a user-desired range of proportions in `highX`. See explanation of `highX` in helper function `split_dichotom()`.
`top`	positive integer scalar, number of optimal dichotomizing predictors, default `1L`
`nsplit, ...`	additional parameters for function `rSplit()`

y	(for helper functions) a Surv object, a logical vector, or a double vector, the response y
x	(for helper functions) numeric vector, a single predictor x
index	(for helper function split_dichotom()) logical vector, indices of training and test set. TRUE elements indicate training subjects and FALSE elements indicate test subjects.
indices	(optional, for helper function quantile_split_dichotom()) a list of logical vectors, the indices of multiple training-test sample splits. Default value is provided by function rSplit().
probs	(for helper function quantile_split_dichotom()) double scalar, see quantile

Details

Function optimSplit_dichotom() selects the optimal dichotomizing predictors via repeated sample splits. Specifically,

1. Generate multiple training-test sample splits using function rSplit()
2. For each candidate predictor, find the median split_dichotom (using helper function quantile_split_dichotom()) of the multiple sample splits from Step 1.
3. (Optional) limit the selection in a subset of the candidate predictors. Typically, we would prefer to guarantee a user-desired range of highX (see explanations on highX in section **Returns of Helper Functions**). A suggested choice is (highX>.15 & highX<.85).
4. Rank the candidate predictors, from either Step 2 or Step 3, by the decreasing order of the absolute values of the estimated univariable regression coefficients of the corresponding split_dichotom objects.

The *optimal dichotomizing predictors* are the ones with the largest absolute values of the estimated univariable regression coefficients of the corresponding split_dichotom objects.

Value

Function optimSplit_dichotom() returns a data.frame, which contains the response, and only the optimal dichotomizing predictors out of all candidate predictors. Other variables in data, which are not specified in formula, are retained. In addition, the dichotomized values of the optimal dichotomizing predictors, according to their respective dichotomizing rules, are also included. The returned value has attributes,

attr(,'id_top') positive integer scalar or vector, the indices of the optimal dichotomizing predictors out of all candidate predictors.

attr(,'top') a diagnostic data.frame of the median split_dichotoms of each of the optimal dichotomizing predictors, with columns

- $cutoff the cutoff threshold, identified in the training set
- $highX proportion of the dichotomizing predictors greater-than or greater-than-or-equal-to the cutoff threshold, in the test set
- $coef the estimated univariable regression coefficient of the dichotomized predictor, in the test set

Details on Helper Functions

Univariable regression model with a dichotomized predictor:

Helper function `split_dichotom()` performs a univariable regression model on the test set with a dichotomized predictor, using a dichotomizing rule determined by a recursive partitioning of the training set. Currently the Cox proportional hazards (coxph) regression for Surv response, logistic (glm) regression for logical response and linear (lm) regression for gaussian response are supported. Specifically, given a training-test sample split,

1. find the dichotomizing rule of the response y given the predictor x, using function `rpartD()`, in the training set
2. dichotomize the predictor x using the rule identified in Step 1, in the test set.
3. run a univariable regression model on the response y on the dichotomized predictor from Step 2, in the test set.

Quantile of split_dichotom objects:

Helper function `quantile_split_dichotom()` finds the quantile of the univariable regression coefficient (i.e., effect size) of a dichotomized predictor, based on multiple given training-test sample splits. Specifically,

1. for each training-test sample split, fit the univariable regression model based on the dichotomized predictor, using helper function `split_dichotom()`
2. finds the nearest-even (`type = 3`) quantile of the estimated univariable regression coefficients obtained in Step 1, based on the user-specified probability `prob`

The split_dichotom object from Step 1, whose estimated univariable regression coefficient equals to the specified quantile identified in Step 2, is referred to as the quantile of split_dichotom objects based on the multiple given training-test sample splits.

Returns of Helper Functions

Helper function `split_dichotom()`, as well as helper function `quantile_split_dichotom()`, returns a Cox proportional hazards (coxph), or a logistic (glm), or a linear (lm) regression model, with additional attributes

`attr(,'rule')` function, the dichotomizing rule based on the training set

`attr(,'cutoff')` numeric scalar, the cutoff threshold based on the training set

`attr(,'highX')` double scalar, proportion of numeric predictor x, in the test set, which is greater-than or greater-than-or-equal-to the cutoff threshold `attr(, 'cutoff')`

`attr(,'coef')` double scalar, the estimated univariable regression coefficient of the dichotomized predictor in the test set

Examples

```
library(survival)
data(pbc, package = 'survival') # see more details from ?survival::pbc
head(pbc2 <- within.data.frame(subset(pbc, status != 1L), expr = {
  death = (status == 2L)
  trt = structure(trt, levels = c('D-penicillmain', 'placebo'), class = 'factor')
  trt = relevel(trt, ref = 'placebo')
```

```
}))

# set.seed if needed
m1 = optimSplit_dichotom(
 Surv(time, death) ~ bili + chol + albumin + copper + alk.phos + ast + trig + platelet + protime,
  data = pbc2, nsplit = 20L, include = (highX > .15 & highX < .85), top = 2L)
head(m1, n = 10L)
attr(m1, 'top')
```

`rpartD` *Dichotomize via Recursive Partitioning*

Description

Dichotomize one or more predictors of a Surv, a logical, or a double response, using recursive partitioning and regression tree rpart.

Usage

```
rpartD(
  y,
  x,
  check_degeneracy = TRUE,
  cp = .Machine$double.eps,
  maxdepth = 2L,
  ...
)

m_rpartD(y, X, check_degeneracy = TRUE, ...)
```

Arguments

`y`	a Surv object, a logical vector, or a double vector, the response y
`x`	numeric vector, one predictor x
`check_degeneracy`	logical scalar, whether to allow the dichotomized value to be all-`FALSE` or all-`TRUE` (i.e., degenerate) for any one of the predictors. Default `TRUE` to produce a warning message for degeneracy.
`cp`	double scalar, complexity parameter, see rpart.control. Default `.Machine$double.eps`, so that a split is enforced no matter how small improvement in overall R^2 is
`maxdepth`	positive integer scalar, maximum depth of any node, see rpart.control. Default `2L`, because only the first node is needed
`...`	additional parameters of rpart and/or rpart.control
`X`	numeric matrix, a set of predictors. Each column of X is one predictor.

Details

Dichotomize Single Predictor:

Function `rpartD()` dichotomizes one predictor in the following steps,

1. Recursive partitioning and regression tree rpart analysis is performed for the response y and the predictor x.
2. The labels.rpart of the first node of the rpart tree is considered as the dichotomizing rule of the double predictor x. The term *dichotomizing rule* indicates the combination of an inequality sign (>, >=, < and <=) and a double cutoff threshold a
3. The dichotomizing rule from Step 2 is further processed, such that
   - $< a$ is regarded as $\geq a$
   - $\leq a$ is regarded as $> a$
   - $> a$ and $\geq a$ are regarded as is.

   This step is necessary for a narrative of *greater than* or *greater than or equal to* the threshold a.
4. A warning message is produced, if the dichotomizing rule, applied to a new double predictor `newx`, creates an all-`TRUE` or all-`FALSE` result. We do not make the algorithm stop, as most regression models in R are capable of handling an all-`TRUE` or all-`FALSE` predictor, by returning a `NA_real_` regression coefficient estimate.

Dichotomize Multiple Predictors:

Function `m_rpartD()` dichotomizes each predictor `X[,i]` based on the response y using function `rpartD()`. Applying the multiple dichotomizing rules to a new set of predictors `newX`,

- A warning message is produced, if at least one of the dichotomized predictors is all-`TRUE` or all-`FALSE`.
- We do not check if more than one of the dichotomized predictors are identical to each other. We take care of this situation in helper function `coef_dichotom()`

Value

Dichotomize Single Predictor:

Function `rpartD()` returns a function, with a double vector parameter `newx`. The returned value of `rpartD(y,x)(newx)` is a logical vector with attributes

`attr(,'cutoff')` double scalar, the cutoff value for `newx`

Dichotomize Multiple Predictors:

Function `m_rpartD()` returns a function, with a double matrix parameter `newX`. The argument for `newX` must have the same number of columns and the same column names as the input matrix X. The returned value of `m_rpartD(y,X)(newX)` is a logical matrix with attributes

`attr(,'cutoff')` named double vector, the cutoff values for each predictor in `newX`

Note

In future integer and factor predictors will be supported.

Examples

```
## Dichotomize Single Predictor
data(cu.summary, package = 'rpart') # see more details from ?rpart::cu.summary
with(cu.summary, rpartD(y = Price, x = Mileage, check_degeneracy = FALSE))
(foo = with(cu.summary, rpartD(y = Price, x = Mileage)))
foo(rnorm(10, mean = 24.5))

## Dichotomize Multiple Predictors
library(survival)
data(stagec, package = 'rpart') # see more details from ?rpart::stagec
nrow(stagec) # 146
(foo = with(stagec[1:100,], m_rpartD(y = Surv(pgtime, pgstat), X = cbind(age, g2, gleason))))
foo(as.matrix(stagec[-(1:100), c('age', 'g2', 'gleason')]))
```

`rSplit` *Random Split Sampling with Stratification*

Description

Random split sampling, stratified based on the type of the response.

Usage

```
rSplit(y, nsplit, stratified = TRUE, trainFrac = 0.8, ...)
```

Arguments

`y`	a double vector, a logical vector, a factor, or a Surv object, response y
`nsplit`	positive integer scalar, replicates of random splits to be performed
`stratified`	logical scalar, whether stratification based on response y needs to be implemented, default `TRUE`
`trainFrac`	double scalar between 0 and 1, fraction of the training set, default `.8`
`...`	additional parameters, currently not in use

Details

Function `rSplit()` performs random split sampling, with or without stratification. Specifically,

- If `stratified` = `FALSE`, or if we have a double response y, then split the sample into a training and a test set by ratio `trainFrac`, without stratification.
- Otherwise, split a Surv response y, stratified by its censoring status. Specifically, split subjects with observed event into a training and a test set with training set fraction `trainFrac`, and split the censored subjects into a training and a test set with training set fraction `trainFrac`. Then combine the training sets from subjects with observed events and censored subjects, and combine the test sets from subjects with observed events and censored subjects.

- Otherwise, split a logical response y, stratified by itself. Specifically, split the subjects with TRUE response into a training and a test set with training set fraction trainFrac, and split the subjects with FALSE response into a training and a test set with training set fraction trainFrac. Then combine the training sets, and the test sets, in a similar fashion as described above.
- Otherwise, split a factor response y, stratified by its levels. Specifically, split the subjects in each level of y into a training and a test set by ratio trainFrac. Then combine the training sets, and the test sets, from all levels of y.

Value

Function rSplit() returns a length-nsplit list of logical vectors. In each logical vector, the TRUE elements indicate training subjects and the FALSE elements indicate test subjects.

Note

caTools::sample.split() is not what we need.

See Also

split

Examples

```
rSplit(y = rep(c(TRUE, FALSE), times = c(20, 30)), nsplit = 3L)
```

Index

∗ **datasets**
celldata, 6
∗ **package**
Qindex-package, 2
<, *19*
<=, *19*
>, *19*
>=, *19*

abs, *16*
aggregate, *7, 8*
aggregate.formula, *7*
attributes, *4, 5*, *11*, *16, 17*, *19*

BBC_dichotom, 3, *4*
BBC_dichotom(), *2, 3*

ceiling, *9*
celldata, 6
character, *6, 7*
clusterQp, 7
clusterQp(), *2*, *7–9*, *11*, *13*
coef_dichotom, *4*
coef_dichotom *(*BBC_dichotom*)*, 3
coef_dichotom(), *3–5*, *19*
cor, *10*
coxph, *4, 5*, *17*

data.frame, *3*, *7–9*, *13*, *15, 16*
double, *4–9*, *11*, *13–20*

factor, *6*, *19–21*
family, *9*
formula, *3*, *7*, *9*, *13*, *15*
FR_gam *(*FRindex*)*, 8
FR_gam(), *8–11*
FRindex, 8, *9, 10*
FRindex(), *2*, *8–11*
FRindex-class *(*FRindex*)*, 8
function, *17*, *19*

gam, *8, 9*, *11*, *13, 14*
gam2beta(), *10*
gaussian, *4*, *17*
glm, *4, 5*, *17*

identical, *19*
integer, *3*, *6, 7*, *9*, *15, 16*, *18–20*
is.unsorted, *11*, *13*

Ki67 *(*celldata*)*, 6

labels.rpart, *19*
language, *15*
length, *11*, *13*
levels, *21*
list, *16*, *21*
lm, *4, 5*, *17*
logical, *3–5*, *9*, *13*, *15–21*
logLik, *4*

m_rpartD *(*rpartD*)*, 18
m_rpartD(), *4, 5*, *19*
matrix, *3*, *5*, *8, 9*, *11*, *13*, *15*, *18, 19*
mean.default, *4*
median, *4*, *9, 10*

name, *9*, *13*, *15*
nlFRindex, 12, *13, 14*
nlFRindex(), *12–14*
nlFRindex-class *(*nlFRindex*)*, 12
numeric, *3*, *11*, *13–18*

optimism_dichotom, *4*
optimism_dichotom *(*BBC_dichotom*)*, 3
optimism_dichotom(), *3*, *5*
optimSplit_dichotom, 15
optimSplit_dichotom(), *2*, *16*

predict, *9*, *13*
predict.FRindex *(*FRindex*)*, 8
predict.FRindex(), *8–12*

predict.nlFRindex *(*nlFRindex*)*, 12
predict.nlFRindex(), *14*

Qindex-package, 2
quantile, *7–11*, *13*, *16, 17*
quantile_split_dichotom
*(*optimSplit_dichotom*)*, 15
quantile_split_dichotom(), *16, 17*

Random, *4*
range, *10*
replicate, *20*
resid, *4*
rpart, *18, 19*
rpart.control, *18*
rpartD, 18
rpartD(), *17*, *19*
rSplit, 20
rSplit(), *15, 16*, *20, 21*

s, *9*
S4, *10*, *14*
seq, *7*
sign, *9, 10*
split, *21*
split_dichotom, *15–17*
split_dichotom *(*optimSplit_dichotom*)*, 15
split_dichotom(), *15–17*
stop, *19*
Surv, *4*, *9*, *13*, *15–18*, *20*

trapz, *10*

vcov, *4*
vector, *4, 5*, *7*, *9*, *11*, *13–16*, *18–21*

warning, *18, 19*