Package ‘FSelectorRcpp’

January 14, 2021

Type Package
Title 'Rcpp' Implementation of 'FSelector' Entropy-Based Feature Selection Algorithms with a Sparse Matrix Support
Version 0.3.8
Depends R (>= 3.4)
License GPL-2
LazyData TRUE
SystemRequirements C++11
Imports Rcpp (>= 0.12.12), foreach, iterators
LinkingTo Rcpp, BH, RcppArmadillo, testthat
Suggests testthat, Matrix, RcppArmadillo, dplyr, RWeka, entropy, FSelector, randomForest, doParallel, rpart, MASS, covr, parallel, htmltools, magrittr, knitr, RTCGA.rnaseq, ggplot2, microbenchmark, pbapply, tibble, rmarkdown, lintr, pkgdown, withr
RoxygenNote 7.1.1
Encoding UTF-8
BugReports https://github.com/mi2-warsaw/FSelectorRcpp/issues
URL https://github.com/mi2-warsaw/FSelectorRcpp
VignetteBuilder knitr
NeedsCompilation yes
Author  Zygmunt Zawadzki [aut, cre],
        Marcin Kosinski [aut],
        Krzysztof Slomczynski [ctb],
        Damian Skrzypiec [ctb],
        Patrick Schratz [ctb] (<https://orcid.org/0000-0003-0748-6624>)
Maintainer  Zygmunt Zawadzki <zygmunt@zstat.pl>
Repository  CRAN
Date/Publication  2021-01-14 15:00:02 UTC

R topics documented:

<table>
<thead>
<tr>
<th>Package</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>cutAttrs</td>
<td>2</td>
</tr>
<tr>
<td>discretize</td>
<td>3</td>
</tr>
<tr>
<td>discretize_transform</td>
<td>5</td>
</tr>
<tr>
<td>feature_search</td>
<td>6</td>
</tr>
<tr>
<td>information_gain</td>
<td>9</td>
</tr>
<tr>
<td>relief</td>
<td>11</td>
</tr>
<tr>
<td>to_formula</td>
<td>12</td>
</tr>
</tbody>
</table>

Index 14

**cut_attrs**

Select Attributes by Score Depending on the Cutoff

Description

Select attributes by their score/rank/weights, depending on the cutoff that may be specified by the percentage of the highest ranked attributes or by the number of the highest ranked attributes.

Usage

`cut_attrs(attrs, k = 0.5)`

Arguments

- **attrs** A *data.frame* with attributes' importance.
- **k** A numeric. For $k \geq 1$ it takes `floor(k)` and then it indicates how many attributes to take with the highest attribute rank (chooses $k$ best attributes). For $0 < k < 1$ it stands for the percent of top attributes to take (chooses best $k \times 100\%$ of attributes).

Author(s)

Damian Skrzypiec <damian.j.skrzypiec@gmail.com> and Zygmunt Zawadzki <zygmunt@zstat.pl>
Examples

```r
x <- information_gain(Species ~ ., iris)
cut_attrs(attrs = x)
to_formula(cut_attrs(attrs = x), "Species")
cut_attrs(attrs = x, k = 1)
```

**discretize**

**Discretization**

Discretize a range of numeric attributes in the dataset into nominal attributes. Minimum Description Length (MDL) method is set as the default control. There is also available equalsizeControl method.

**Usage**

```r
discretize(
  x,
  y,
  control = list(mdlControl(), equalsizeControl()),
  all = TRUE,
  discIntegers = TRUE,
  call = NULL
)
```

- `mdlControl()`: mdlControl
- `equalsizeControl(k = 10)`: equalsizeControl
- `customBreaksControl(breaks)`: customBreaksControl

**Arguments**

- `x`: Explanatory continuous variables to be discretized or a formula.
- `y`: Dependent variable for supervised discretization or a data.frame when x is a formula.
- `control`: discretizationControl object containing the parameters for discretization algorithm. Possible inputs are mdlControl or equalsizeControl, so far. If passed as a list, the first element is used.
- `all`: Logical indicating if a returned data.frame should contain other features that were not discretized. (Example: should Sepal.Width be returned, when you pass iris and discretize Sepal.Length,Petal.Length,Petal.Width.)
- `discIntegers`: logical value. If true (default), then integers are treated as numeric vectors and they are discretized. If false integers are treated as factors and they are left as is.
call Keep as NULL. Inner method parameter for consistency.
k Number of partitions.
breaks custom breaks used for partitioning.

Author(s)
Zygmunt Zawadzki <zygmunt@zstat.pl>

References

Examples

```r
# vectors
discretize(x = iris[[1]], y = iris[[5]])

# list and vector
head(discretize(x = list(iris[[1]], iris$Sepal.Width), y = iris$Species))

# formula input
head(discretize(x = Species ~ ., y = iris))
head(discretize(Species ~ ., iris))

# use different methods for specific columns
ir1 <- discretize(Species ~ Sepal.Length, iris)
ir2 <- discretize(Species ~ Sepal.Width, ir1, control = equalsizeControl(3))
ir3 <- discretize(Species ~ Petal.Length, ir2, control = equalsizeControl(5))
head(ir3)

# custom breaks
ir <- discretize(Species ~ Sepal.Length, iris,
                 control = customBreaksControl(breaks = c(0, 2, 5, 7.5, 10)))
head(ir)
```

## Not run:
# Same results
library(RWeka)
RWeka_disc_out <- RWeka::Discretize(Species ~ Sepal.Length, iris)[, 1]
FSelectorRcpp_disc_out <- FSelectorRcpp::discretize(Species ~ Sepal.Length, iris)[, 1]
table(RWeka_disc_out, FSelectorRcpp_disc_out)
# But faster method
library(microbenchmark)
microbenchmark(FSelectorRcpp::discretize(Species ~ Sepal.Length, iris),
               RWeka::Discretize(Species ~ Sepal.Length, iris))
```

## End(Not run)
discretize_transform

Transform a data.frame using split points returned by discretize function.

Description
Transform a data.frame using split points returned by discretize function.

Usage

```r
discretize_transform(disc, data, dropColumns = NA)

extract_discretize_transformer(disc)
```

Arguments

- `disc`: a result of the `discretize` function.
- `data`: a data.frame to transform using cutpoints from disc.
- `dropColumns`: determine

Value

A new data.frame with discretized columns using cutpoints from the result of discretize function.

Examples

```r
set.seed(123)
idx <- sort(sample.int(150, 100))
iris1 <- iris[idx, ]
iris2 <- iris[-idx, ]
disc <- discretize(Species ~ ., iris)
head(discretize_transform(disc, iris2))

# Chain discretization:
ir1 <- discretize(Species ~ Sepal.Length, iris1)
ir2 <- discretize(Species ~ Sepal.Width, ir1, control = equalsizeControl(3))
ir3 <- discretize(Species ~ Petal.Length, ir2, control = equalsizeControl(5))

## note that Petal.Width is untouched:
head(discretize_transform(ir3, iris2))

## extract_discretize_transformer
discObj <- extract_discretize_transformer(ir3)
head(discretize_transform(discObj, iris2))
```
feature_search

Description
A convenience wrapper for greedy and exhaustive feature selection algorithms that extract valuable attributes depending on the evaluation method (called evaluator). This function is a reimplementation of FSelector's exhaustive.search and greedy.search.

Usage
feature_search(
  attributes,
  fun,
  data,
  mode = c("greedy", "exhaustive"),
  type = c("forward", "backward"),
  sizes = 1:length(attributes),
  parallel = TRUE,
  ...
)

Arguments
attributes A character vector with attributes' names to be used to extract the most valuable features.
fun A function (evaluator) to be used to score features' sets at each iteration of the algorithm passed via mode. See Examples.
data A data set for fun function (evaluator).
mode A character that determines which search algorithm to perform. Default is "greedy".
type Used when mode = "greedy" - whether to use the backward or the forward multiple-way search. Default is "forward".
sizes Used when mode = "exhaustive" - a vector of sizes of attributes subsets.
parallel Allow parallelization.
...
Other arguments passed to foreach function.

Details
The evaluator function passed with fun is used to determine the importance score of current features' subset. The score is used in a multiple-way (backward or forward) greedy algorithm as a stopping moment or as a selection criterion in the exhaustive search that checks all possible attributes' subset combinations (of sizes passed in sizes).
**Value**

A list with following components

- best - a `data.frame` with the best subset and it's score (1 - feature used, 0 - feature not used),
- all - a `data.frame` with all checked features' subsets and their score (1 - feature used, 0 - feature not used),
- data - the data used in the feature selection,
- fun - the evaluator used to compute the score of importance for features' subsets,
- call - an origin call of the `feature_search`,
- mode - the mode used in the call.

**Note**

Note that score depends on the evaluator you provide in the `fun` parameter.

**Author(s)**

Zygmunt Zawadzki <zygmunt@zstat.pl>

Krzysztof Slomczynski <krzysztofslomczynski@gmail.com>

**Examples**

```r
# Enable parallelization in examples
## Not run:
library(doParallel)
c1 <- makeCluster(2)
registerDoParallel(c1)
## End(Not run)
# Close at the end
# stopCluster(cl) #nolint
# registerDoSEQ() #nolint
# 1) Evaluator from FSelector package.
evaluator <- function(subset, data, dependent = names(iris)[5]) {
  library(rpart)
k <- 5
splits <- runif(nrow(data))
results <- sapply(1:k, function(i) {
  test.idx <- (splits >= (i - 1) / k) & (splits < i / k)
  train.idx <- !test.idx
  test <- data[test.idx, , drop = FALSE]
  train <- data[train.idx, , drop = FALSE]
  tree <- rpart(to_formula(subset, dependent), train)
  error.rate <- sum(test[[dependent]] != predict(tree, test, type = "c")) /
               nrow(test)
  return(1 - error.rate)
})
```
return(mean(results))
}

set.seed(123)
# Default greedy search.
system.time(
    feature_search(attributes = names(iris)[-5],
                   fun = evaluator,
                   data = iris)
)
system.time(
    feature_search(attributes = names(iris)[-5],
                   fun = evaluator,
                   data = iris,
                   parallel = FALSE)
)

# Optional exhaustive search.
system.time(
    feature_search(attributes = names(iris)[-5],
                   fun = evaluator,
                   data = iris,
                   mode = "exhaustive")
)
system.time(
    feature_search(attributes = names(iris)[-5],
                   fun = evaluator,
                   data = iris,
                   mode = "exhaustive",
                   parallel = FALSE)
)

# 2) Maximize R^2 statistics in the linear regression model/problem.
evaluator_R2_lm <- function(attributes, data, dependent = names(iris)[1]) {
    summary(
        lm(to_formula(attributes, dependent), data = data)
    )$r.squared
}

feature_search(attributes = names(iris)[-1],
                fun = evaluator_R2_lm, data = iris,
                mode = "exhaustive")

# 3) Optimize BIC criterion in generalized linear model.
# Aim of Bayesian approach it to identify the model with the highest
# probability of being the true model. - Kuha 2004

utils::data(anorexia, package = "MASS")
evaluator_BIC_glm <- function(attributes, data, dependent = "Postwt") {
    extractAIC(
        fit = glm(to_formula(attributes, dependent), family = gaussian,
                    data = data),
    )
}

feature_search(attributes = names(iris)[-1],
                fun = evaluator_BIC_glm, data = iris,
                mode = "exhaustive")
data = data),
k = log(nrow(data))
)[2]
}

feature_search(attributes = c("Prewt", "Treat", "offset(Prewt)"),
fun = evaluator_BIC_glm,
data = anorexia,
mode = "exhaustive")

# Close parallelization
## Not run:
stopCluster(cl)
registerDoSEQ()

## End(Not run)

---

**information_gain**  
**Entropy-based Filters**

**Description**

Algorithms that find ranks of importance of discrete attributes, basing on their entropy with a continuous class attribute. This function is a reimplementation of FSelector’s information.gain, gain.ratio and symmetrical.uncertainty.

**Usage**

```
information_gain(
  formula,
  data,
  x,
  y,
  type = c("infogain", "gainratio", "symuncert"),
  equal = FALSE,
  discIntegers = TRUE,
  nbins = 5,
  threads = 1
)
```

**Arguments**

- **formula**: An object of class formula with model description.
- **data**: A data.frame accompanying formula.
- **x**: A data.frame or sparse matrix with attributes.
- **y**: A vector with response variable.
- **type**: Method name.
equal  A logical. Whether to discretize dependent variable with the equal frequency binning discretization or not.

discIntegers logical value. If true (default), then integers are treated as numeric vectors and they are discretized. If false integers are treated as factors and they are left as is.

nbins Number of bins used for discretization. Only used if 'equal = TRUE' and the response is numeric.

threads defunct. Number of threads for parallel backend - now turned off because of safety reasons.

Details
type = "infogain" is
\[ H(\text{Class}) + H(\text{Attribute}) - H(\text{Class}, \text{Attribute}) \]

type = "gainratio" is
\[ \frac{H(\text{Class}) + H(\text{Attribute}) - H(\text{Class}, \text{Attribute})}{H(\text{Attribute})} \]

type = "symuncert" is
\[ 2 \frac{H(\text{Class}) + H(\text{Attribute}) - H(\text{Class}, \text{Attribute})}{H(\text{Attribute}) + H(\text{Class})} \]

where \( H(X) \) is Shannon’s Entropy for a variable \( X \) and \( H(X, Y) \) is a joint Shannon’s Entropy for a variable \( X \) with a condition to \( Y \).

Value
data.frame with the following columns:

• attributes - variables names.
• importance - worth of the attributes.

Author(s)
Zygmunt Zawadzki <zygmunt@zstat.pl>

Examples

```r
irisX <- iris[-5]
y <- iris$Species

## data.frame interface
information_gain(x = irisX, y = y)

# formula interface
information_gain(formula = Species ~ ., data = iris)
```
relief

relief(formula = Species ~ ., data = iris, type = "gainratio")
relief(formula = Species ~ ., data = iris, type = "symuncert")

# sparse matrix interface
library(Matrix)
i <- c(1, 3:8); j <- c(2, 9, 6:10); x <- 7 * (1:7)
x <- sparseMatrix(i, j, x = x)
y <- c(1, 1, 1, 1, 2, 2, 2, 2)

relief(x = x, y = y)
relief(x = x, y = y, type = "gainratio")
relief(x = x, y = y, type = "symuncert")

relief

RReliefF filter

Description
The algorithm finds weights of continuous and discrete attributes basing on a distance between instances.

Usage
relief(formula, data, x, y, neighboursCount = 5, sampleSize = 10)

Arguments

formula An object of class formula with model description.
data A data.frame accompanying formula.
x A data.frame with attributes.
y A vector with response variable.
neighboursCount number of neighbours to find for every sampled instance
sampleSize number of instances to sample

Details

Value
a data.frame containing the worth of attributes in the first column and their names as row names
to_formula

Create a formula Object

description

Utility function to create a formula object. Note that it may be very useful when you use pipes.

Usage

to_formula(attrs, class)

Arguments

attrs Character vector with names of independent variables.
class Single string with a dependent variable’s name.

Examples

# evaluator from FSelector package
evaluator <- function(subset, data, dependent = names(iris)[5]) {
  library(rpart)
  k <- 5
  splits <- runif(nrow(data))
  results <- sapply(1:k, function(i) {
    test.idx <- (splits >= (i - 1) / k) & (splits < i / k)
    train.idx <- !test.idx
    test <- data[test.idx, , drop = FALSE]
    train <- data[train.idx, , drop = FALSE]
    tree <- rpart(to_formula(subset, dependent), train)
    error.rate <- sum(test[[dependent]] != predict(tree, test, type = "c")) /
nrow(test)
return(1 - error.rate)
}
return(mean(results))
}

set.seed(123)
fit <- feature_search(attributes = names(iris)[-5], fun = evaluator, data = iris,
mode = "exhaustive", parallel = FALSE)
fit$best
names(fit$best)[fit$best == 1]
# with to_formula
to_formula(names(fit$best)[fit$best == 1], "Species")
Index

customBreaksControl (discretize), 3
cut_attrs, 2

data.frame, 2, 3, 7, 9, 11
discretize, 3, 5
discretize_transform, 5
equalsizeControl (discretize), 3
exhaustive.search, 6
extract_discretize_transformer
discretize_transform, 5

feature_search, 6
foreach, 6
formula, 3, 9, 11, 12

gain.ratio, 9
greedy.search, 6

information.gain, 9
information_gain, 9

mdlControl (discretize), 3
relief, 11
symmetrical.uncertainty, 9
to_formula, 12