Package ‘FSelector’

August 22, 2023

Type Package

Title Selecting Attributes

Version 0.34

Date 2023-08-22

Author Piotr Romanski, Lars Kotthoff, Patrick Schratz

Maintainer Lars Kotthoff <larsko@uwyo.edu>

BugReports https://github.com/larskotthoff/fselector/issues

URL https://github.com/larskotthoff/fselector

Description Functions for selecting attributes from a given
dataset. Attribute subset selection is the process of identifying and
removing as much of the irrelevant and redundant information as
possible.

License GPL-2

Imports digest, entropy, randomForest, RWeka

Suggests mlbench, rpart

Encoding UTF-8

LazyLoad yes

NeedsCompilation no

Repository CRAN

Date/Publication 2023-08-22 17:10:02 UTC

R topics documented:

`FSelector-package` *Package for selecting attributes*

Description

Package containing functions for selecting attributes from a given dataset and a destination attribute.

Details

This package contains:

- -Algorithms for filtering attributes: cfs, chi.squared, information.gain, gain.ratio, symmetrical.uncertainty, linear.correlation, rank.correlation, oneR, relief, consistency, random.forest.importance
- -Algorithms for wrapping classifiers and search attribute subset space: best.first.search, backward.search, forward.search, hill.climbing.search
- -Algorithm for choosing a subset of attributes based on attributes' weights: cutoff.k, cutoff.k.percent, cutoff.biggest.diff
- -Algorithm for creating formulas: as.simple.formula

Author(s)

Piotr Romanski
Maintainer: Lars Kotthoff <larsko@uwyo.edu>

`as.simple.formula` *Converting to formulas*

Description

Converts character vector of atrributes' names and destination attribute's name to a simple formula.

Usage

```
as.simple.formula(attributes, class)
```

Arguments

`attributes`	character vector of attributes' names
`class`	name of destination attribute

Value

A simple formula like "class ~ attr1 + attr2"

Author(s)

Piotr Romanski

Examples

```
data(iris)
result <- cfs(Species ~ ., iris)
f <- as.simple.formula(result, "Species")
```

`best.first.search` *Best-first search*

Description

The algorithm for searching atrribute subset space.

Usage

```
best.first.search(attributes, eval.fun, max.backtracks = 5)
```

Arguments

`attributes`	a character vector of all attributes to search in
`eval.fun`	a function taking as first parameter a character vector of all attributes and returning a numeric indicating how important a given subset is
`max.backtracks`	an integer indicating a maximum allowed number of backtracks, default is 5

Details

The algorithm is similar to `forward.search` besides the fact that is chooses the best node from all already evaluated ones and evaluates it. The selection of the best node is repeated approximately `max.backtracks` times in case no better node found.

Value

A character vector of selected attributes.

Author(s)

Piotr Romanski

See Also

forward.search, backward.search, hill.climbing.search, exhaustive.search

Examples

```
library(rpart)
data(iris)

evaluator <- function(subset) {
  #k-fold cross validation
  k <- 5
  splits <- runif(nrow(iris))
  results = sapply(1:k, function(i) {
    test.idx <- (splits >= (i - 1) / k) & (splits < i / k)
    train.idx <- !test.idx
    test <- iris[test.idx, , drop=FALSE]
    train <- iris[train.idx, , drop=FALSE]
    tree <- rpart(as.simple.formula(subset, "Species"), train)
    error.rate = sum(test$Species != predict(tree, test, type="c")) / nrow(test)
    return(1 - error.rate)
  })
  print(subset)
  print(mean(results))
  return(mean(results))
}

subset <- best.first.search(names(iris)[-5], evaluator)
f <- as.simple.formula(subset, "Species")
print(f)
```

---

cfs *CFS filter*

Description

The algorithm finds attribute subset using correlation and entropy measures for continous and discrete data.

Usage

```
cfs(formula, data)
```

Arguments

`formula`	a symbolic description of a model
`data`	data to process

Details

The alorithm makes use of `best.first.search` for searching the attribute subset space.

Value

a character vector containing chosen attributes

Author(s)

Piotr Romanski

See Also

`best.first.search`

Examples

```
data(iris)

subset <- cfs(Species~., iris)
f <- as.simple.formula(subset, "Species")
print(f)
```

`chi.squared` *Chi-squared filter*

Description

The algorithm finds weights of discrete attributes basing on a chi-squared test.

Usage

```
chi.squared(formula, data)
```

Arguments

`formula`	a symbolic description of a model
`data`	a symbolic description of a model

Details

The result is equal to Cramer's V coefficient between source attributes and destination attribute.

Value

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

Author(s)

Piotr Romanski

Examples

```
library(mlbench)
data(HouseVotes84)

weights <- chi.squared(Class~., HouseVotes84)
print(weights)
subset <- cutoff.k(weights, 5)
f <- as.simple.formula(subset, "Class")
print(f)
```

---

consistency *Consistency-based filter*

Description

The algorithm finds attribute subset using consistency measure for continous and discrete data.

Usage

```
consistency(formula, data)
```

Arguments

`formula`	a symbolic description of a model
`data`	data to process

Details

The alorithm makes use of `best.first.search` for searching the attribute subset space.

Value

a character vector containing chosen attributes

Author(s)

Piotr Romanski

See Also

`best.first.search`

Examples

```
## Not run:
  library(mlbench)
  data(HouseVotes84)

  subset <- consistency(Class~., HouseVotes84)
  f <- as.simple.formula(subset, "Class")
  print(f)

## End(Not run)
```

---

correlation — *Correlation filter*

Description

The algorithm finds weights of continous attributes basing on their correlation with continous class attribute.

Usage

```
linear.correlation(formula, data)
rank.correlation(formula, data)
```

Arguments

`formula`	a symbolic description of a model
`data`	data to process

Details

`linear.correlation` uses Pearson's correlation

`rank.correlation` uses Spearman's correlation

Rows with `NA` values are not taken into consideration.

Value

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

Author(s)

Piotr Romanski

Examples

```
library(mlbench)
data(BostonHousing)
d=BostonHousing[-4] # only numeric variables

weights <- linear.correlation(medv~., d)
print(weights)
subset <- cutoff.k(weights, 3)
f <- as.simple.formula(subset, "medv")
print(f)

weights <- rank.correlation(medv~., d)
print(weights)
subset <- cutoff.k(weights, 3)
f <- as.simple.formula(subset, "medv")
print(f)
```

---

cutoff *Cutoffs*

Description

The algorithms select a subset from a ranked attributes.

Usage

```
cutoff.k(attrs, k)
cutoff.k.percent(attrs, k)
cutoff.biggest.diff(attrs)
```

Arguments

`attrs`	a data.frame containing ranks for attributes in the first column and their names as row names
`k`	a positive integer in case of `cutoff.k` and a numeric between 0 and 1 in case of `cutoff.k.percent`

Details

`cutoff.k` chooses k best attributes

`cutoff.k.percent` chooses best k * 100% of attributes

`cutoff.biggest.diff` chooses a subset of attributes which are significantly better than other.

Value

A character vector containing selected attributes.

Author(s)

Piotr Romanski

Examples

```
data(iris)

weights <- information.gain(Species~., iris)
print(weights)

subset <- cutoff.k(weights, 1)
f <- as.simple.formula(subset, "Species")
print(f)

subset <- cutoff.k.percent(weights, 0.75)
f <- as.simple.formula(subset, "Species")
print(f)

subset <- cutoff.biggest.diff(weights)
f <- as.simple.formula(subset, "Species")
print(f)
```

`entropy.based` *Entropy-based filters*

Description

The algorithms find weights of discrete attributes basing on their correlation with continous class attribute.

Usage

```
information.gain(formula, data, unit)
gain.ratio(formula, data, unit)
symmetrical.uncertainty(formula, data, unit)
```

Arguments

`formula`	A symbolic description of a model.
`data`	Data to process.
`unit`	Unit for computing entropy (passed to `entropy`. Default is "log".

Details

`information.gain` is

$$H(Class) + H(Attribute) - H(Class, Attribute)$$

.

`gain.ratio` is

$$\frac{H(Class) + H(Attribute) - H(Class, Attribute)}{H(Attribute)}$$

`symmetrical.uncertainty` is

$$2\frac{H(Class) + H(Attribute) - H(Class, Attribute)}{H(Attribute) + H(Class)}$$

Value

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

Author(s)

Piotr Romanski, Lars Kotthoff

Examples

```
data(iris)

weights <- information.gain(Species~., iris)
print(weights)
subset <- cutoff.k(weights, 2)
f <- as.simple.formula(subset, "Species")
print(f)

weights <- information.gain(Species~., iris, unit = "log2")
print(weights)

weights <- gain.ratio(Species~., iris)
print(weights)
subset <- cutoff.k(weights, 2)
f <- as.simple.formula(subset, "Species")
print(f)

weights <- symmetrical.uncertainty(Species~., iris)
print(weights)
subset <- cutoff.biggest.diff(weights)
f <- as.simple.formula(subset, "Species")
print(f)
```

`exhaustive.search` *Exhaustive search*

Description

The algorithm for searching atrribute subset space.

Usage

```
exhaustive.search(attributes, eval.fun)
```

Arguments

`attributes`	a character vector of all attributes to search in
`eval.fun`	a function taking as first parameter a character vector of all attributes and returning a numeric indicating how important a given subset is

Details

The algorithm searches the whole attribute subset space in breadth-first order.

Value

A character vector of selected attributes.

Author(s)

Piotr Romanski

See Also

`forward.search`, `backward.search`, `best.first.search`, `hill.climbing.search`

Examples

```
library(rpart)
data(iris)

evaluator <- function(subset) {
  #k-fold cross validation
  k <- 5
  splits <- runif(nrow(iris))
  results = sapply(1:k, function(i) {
    test.idx <- (splits >= (i - 1) / k) & (splits < i / k)
    train.idx <- !test.idx
    test <- iris[test.idx, , drop=FALSE]
    train <- iris[train.idx, , drop=FALSE]
    tree <- rpart(as.simple.formula(subset, "Species"), train)
    error.rate = sum(test$Species != predict(tree, test, type="c")) / nrow(test)
```

```
    return(1 - error.rate)
  })
  print(subset)
  print(mean(results))
  return(mean(results))
}

subset <- exhaustive.search(names(iris)[-5], evaluator)
f <- as.simple.formula(subset, "Species")
print(f)
```

`greedy.search` *Greedy search*

Description

The algorithms for searching atrribute subset space.

Usage

```
backward.search(attributes, eval.fun)
forward.search(attributes, eval.fun)
```

Arguments

`attributes`	a character vector of all attributes to search in
`eval.fun`	a function taking as first parameter a character vector of all attributes and returning a numeric indicating how important a given subset is

Details

These algorithms implement greedy search. At first, the algorithms expand starting node, evaluate its children and choose the best one which becomes a new starting node. This process goes only in one direction. `forward.search` starts from an empty and `backward.search` from a full set of attributes.

Value

A character vector of selected attributes.

Author(s)

Piotr Romanski

See Also

`best.first.search`, `hill.climbing.search`, `exhaustive.search`

Examples

```
library(rpart)
data(iris)

evaluator <- function(subset) {
  #k-fold cross validation
  k <- 5
  splits <- runif(nrow(iris))
  results = sapply(1:k, function(i) {
    test.idx <- (splits >= (i - 1) / k) & (splits < i / k)
    train.idx <- !test.idx
    test <- iris[test.idx, , drop=FALSE]
    train <- iris[train.idx, , drop=FALSE]
    tree <- rpart(as.simple.formula(subset, "Species"), train)
    error.rate = sum(test$Species != predict(tree, test, type="c")) / nrow(test)
    return(1 - error.rate)
  })
  print(subset)
  print(mean(results))
  return(mean(results))
}

subset <- forward.search(names(iris)[-5], evaluator)
f <- as.simple.formula(subset, "Species")
print(f)
```

`hill.climbing.search` *Hill climbing search*

Description

The algorithm for searching atrribute subset space.

Usage

```
hill.climbing.search(attributes, eval.fun)
```

Arguments

`attributes`	a character vector of all attributes to search in
`eval.fun`	a function taking as first parameter a character vector of all attributes and returning a numeric indicating how important a given subset is

Details

The algorithm starts with a random attribute set. Then it evaluates all its neighbours and chooses the best one. It might be susceptible to local maximum.

Value

A character vector of selected attributes.

Author(s)

Piotr Romanski

See Also

forward.search, backward.search, best.first.search, exhaustive.search

Examples

```
library(rpart)
data(iris)

evaluator <- function(subset) {
  #k-fold cross validation
  k <- 5
  splits <- runif(nrow(iris))
  results = sapply(1:k, function(i) {
    test.idx <- (splits >= (i - 1) / k) & (splits < i / k)
    train.idx <- !test.idx
    test <- iris[test.idx, , drop=FALSE]
    train <- iris[train.idx, , drop=FALSE]
    tree <- rpart(as.simple.formula(subset, "Species"), train)
    error.rate = sum(test$Species != predict(tree, test, type="c")) / nrow(test)
    return(1 - error.rate)
  })
  print(subset)
  print(mean(results))
  return(mean(results))
}

subset <- hill.climbing.search(names(iris)[-5], evaluator)
f <- as.simple.formula(subset, "Species")
print(f)
```

---

oneR *OneR algorithm*

Description

The algorithms find weights of discrete attributes basing on very simple association rules involving only one attribute in condition part.

Usage

```
oneR(formula, data)
```

Arguments

`formula`	a symbolic description of a model
`data`	data to process

Details

The algorithm uses OneR classifier to find out the attributes' weights. For each attribute it creates a simple rule based only on that attribute and then calculates its error rate.

Value

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

Author(s)

Piotr Romanski

Examples

```
library(mlbench)
data(HouseVotes84)

weights <- oneR(Class~., HouseVotes84)
print(weights)
subset <- cutoff.k(weights, 5)
f <- as.simple.formula(subset, "Class")
print(f)
```

---

random.forest.importance

RandomForest filter

Description

The algorithm finds weights of attributes using RandomForest algorithm.

Usage

```
random.forest.importance(formula, data, importance.type = 1)
```

Arguments

`formula` a symbolic description of a model

`data` data to process

`importance.type`
either 1 or 2, specifying the type of importance measure (1=mean decrease in accuracy, 2=mean decrease in node impurity)

Details

This is a wrapper for `importance`.

Value

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

Author(s)

Piotr Romanski

Examples

```
library(mlbench)
data(HouseVotes84)

weights <- random.forest.importance(Class~., HouseVotes84, importance.type = 1)
print(weights)
subset <- cutoff.k(weights, 5)
f <- as.simple.formula(subset, "Class")
print(f)
```

`relief` *RReliefF filter*

Description

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

Usage

```
relief(formula, data, neighbours.count = 5, sample.size = 10)
```

Arguments

`formula` a symbolic description of a model

`data` data to process

`neighbours.count`
number of neighbours to find for every sampled instance

`sample.size` number of instances to sample

Details

The algorithm samples instances and finds their nearest hits and misses. Considering that result, it evaluates weights of attributes.

Value

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

Author(s)

Piotr Romanski

References

- -Igor Kononenko: Estimating Attributes: Analysis and Extensions of RELIEF. In: European Conference on Machine Learning, 171-182, 1994.
- -Marko Robnik-Sikonja, Igor Kononenko: An adaptation of Relief for attribute estimation in regression. In: Fourteenth International Conference on Machine Learning, 296-304, 1997.

Examples

```
data(iris)

weights <- relief(Species~., iris, neighbours.count = 5, sample.size = 20)
print(weights)
subset <- cutoff.k(weights, 2)
f <- as.simple.formula(subset, "Species")
print(f)
```

Index