Package ‘FSelector’

May 16, 2018

Type Package
Title Selecting Attributes
Version 0.31
Date 2018-05-16
Author Piotr Romanski, Lars Kotthoff
Maintainer Lars Kotthoff <larsko@uwyo.edu>
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 randomForest, RWeka, digest, entropy
Suggests mlbench, rpart
LazyLoad yes
NeedsCompilation no
Repository CRAN
Date/Publication 2018-05-16 21:38:09 UTC

R topics documented:

  FSelector-package .................................................. 2
  as.simple.formula .................................................. 2
  best.first.search ................................................... 3
  cfs ................................................................. 4
  chi.squared .......................................................... 5
  consistency ........................................................... 6
  correlation ............................................................. 7
  cutoff ................................................................. 8
  entropy.based ........................................................ 9
  exhaustive.search .................................................. 11
  greedy.search ....................................................... 12
  hill.climbing.search ................................................ 13
FSelector-package  

Package for selecting attributes

Description

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

Details

Package: FSelector
Type: Package
Version: 0.31
Date: 2018-05-16
License: GPL-2
LazyLoad: yes

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 attributes’ names and destination attribute’s name to a simple formula.
**Usage**

```r
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**

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

---

**best.first.search**

*Best-first search*

**Description**

The algorithm for searching attribute subset space.

**Usage**

```r
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 it 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**

```r
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.
chi.squared

Usage

chi.squared(formula, data)

Arguments

formula a symbolic description of a model
data data to process

Details

The algorithm 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 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)

<table>
<thead>
<tr>
<th>consistency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consistency-based filter</td>
</tr>
</tbody>
</table>

Description
The algorithm finds attribute subset using consistency measure for continuous 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
correlation

See Also

best.first.search

Examples

```r
## 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 continuous attributes basing on their correlation with continuous class attribute.

Usage

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

Arguments

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

Details

```r
linear.correlation uses Pearson's correlation
rank.correlation uses Spearman's correlation
Rows with NA values are not taken into consideration.
```

Value

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

Author(s)

Piotr Romanski
Examples

```r
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)
```

<table>
<thead>
<tr>
<th>cutoff</th>
<th>Cutoffs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 continuous 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

\[ H(\text{Class}) + H(\text{Attribute}) - H(\text{Class}, \text{Attribute}) \]

\[ \text{gain.ratio is} \]

\[ \frac{H(\text{Class}) + H(\text{Attribute}) - H(\text{Class}, \text{Attribute})}{H(\text{Attribute})} \]

\[ \text{symmetrical.uncertainty is} \]

\[ \frac{2(H(\text{Class}) + H(\text{Attribute}) - H(\text{Class}, \text{Attribute}))}{H(\text{Attribute}) + H(\text{Class})} \]

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 attribute 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)
  })
  # Choose the best subset
  best_subset = names(which.max(results))
  return(best_subset)
}

# Evaluate the evaluator function
errors = evaluator(iris[, attributes]

evaluation <- function(subset) {
  error.rate = evaluator(subset)
  return(error.rate)
}

# Find the best subset
best_subset = exhaustive.search(attributes, evaluation)

# Display the best subset
best_subset

greedy.search

Greedy search

Description
The algorithms for searching attribute 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

```r
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 attribute subset space.

**Usage**

```r
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.
**random.forest.importance**

Usage

```r
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

```r
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

```r
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 continuous 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

Examples
```r
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

*Topic package
  FSelector-package, 2

as.simple.formula, 2

backward.search, 4, 11, 14
backward.search (greedy.search), 12
best.first.search, 3, 5–7, 11, 12, 14

cfs, 4
chi.squared, 5
consistency, 6
correlation, 7
cutoff, 8

dency, 9
dency.based, 9
exhaustive.search, 4, 11, 12, 14

forward.search, 3, 4, 11, 14
forward.search (greedy.search), 12
FSelector (FSelector-package), 2
FSelector-package, 2

gain.ratio (entropy.based), 9
greedy.search, 12

hill.climbing.search, 4, 11, 12, 13

importance, 16
information.gain (entropy.based), 9

linear.correlation (correlation), 7

oneR, 14

random.forest.importance, 15
rank.correlation (correlation), 7
relief, 16

symmetrical.uncertainty
  (entropy.based), 9