Package 'RECA'

November 12, 2016

Type Package

Title Relevant Component Analysis for Supervised Distance Metric Learning

Version 1.3

Date 2016-11-12

Author Nan Xiao <me@nanx.me>

Maintainer Nan Xiao <me@nanx.me>

Description Relevant Component Analysis (RCA) tries to find a linear transformation of the feature space such that the effect of irrelevant variability is reduced in the transformed space.

License GPL (>= 2)

URL https://github.com/road2stat/RECA

BugReports https://github.com/road2stat/RECA/issues

Suggests MASS

RoxygenNote 5.0.1

NeedsCompilation no

Repository CRAN

Date/Publication 2016-11-12 19:54:49

R topics documented:

RECA-package *Relevant Component Analysis*

Description

The RECA package offers an R implementation for Relevant Component Analysis (RCA).

Details

Package:	RECA
Type:	Package
Version:	1.3
License:	GPL (>= 2)

Note

Bug reports and feature requests could be sent to https://github.com/road2stat/RECA/issues.

rca *Relevant Component Analysis*

Description

`rca` performs relevant component analysis (RCA) for the given data. It takes a data set and a set of positive constraints as arguments and returns a linear transformation of the data space into better representation, alternatively, a Mahalanobis metric over the data space.

Usage

```
rca(x, chunks, useD = NULL)
```

Arguments

`x`	`n * d` matrix or data frame of original data.
`chunks`	a vector of size `N` describing the chunklets: `-1` in the `i`-th place says that point `i` does not belong to any chunklet; integer `j` in place `i` says that point `i` belongs to chunklet `j`; The chunklets indexes should be `1:number-of-chunklets`.
`useD`	optional. When not given, RCA is done in the original dimension and `B` is full rank. When `useD` is given, RCA is preceded by constraints based LDA which reduces the dimension to `useD`. `B` in this case is of rank `useD`.

Details

The new representation is known to be optimal in an information theoretic sense under a constraint of keeping equivalent data points close to each other.

The three returned objects are just different forms of the same output. If one is interested in a Mahalanobis metric over the original data space, the first argument is all she/he needs. If a transformation into another space (where one can use the Euclidean metric) is preferred, the second returned argument is sufficient. Using `A` and `B` are equivalent in the following sense:

if `y1 = A * x1`, `y2 = A * y2` then

`(x2 - x1)^T * B * (x2 - x1) = (y2 - y1)^T * (y2 - y1)`

Value

A list of the RCA results:

- `B`: The RCA suggested Mahalanobis matrix. Distances between data points `x1`, `x2` should be computed by `(x2 - x1)^T * B * (x2 - x1)`
- `RCA`: The RCA suggested transformation of the data. The data should be transformed by `RCA * data`
- `newX`: The data after the RCA transformation. `newX = data * RCA`

Note

Note that any different sets of instances (chunklets), e.g. `{1, 3, 7}` and `{4, 6}`, might belong to the same class and might belong to different classes.

Author(s)

Nan Xiao <http://nanx.me>

References

Aharon Bar-Hillel, Tomer Hertz, Noam Shental, and Daphna Weinshall (2003). Learning Distance Functions using Equivalence Relations. *Proceedings of 20th International Conference on Machine Learning (ICML2003)*

Examples

```
library("MASS")  # generate synthetic MVN data
set.seed(42)
k = 100L         # sample size of each class
n = 3L           # specify how many classes
N = k * n        # total sample size
x1 = mvrnorm(k, mu = c(-8, 6), matrix(c(15, 1, 2, 10),   ncol = 2))
x2 = mvrnorm(k, mu = c(0, 0),  matrix(c(15, 0.5, 2, 10), ncol = 2))
x3 = mvrnorm(k, mu = c(8, -6), matrix(c(15, 1, 2, 10),   ncol = 2))
x = as.data.frame(rbind(x1, x2, x3))  # predictor
y = gl(n, k)  # response

# The fully labeled data set with 3 classes
```

```
plot(x[, 1L], x[, 2L], bg = c("#E41A1C", "#377EB8", "#4DAF4A")[y],
     pch = rep(c(22, 21, 25), each = k))

# Same data unlabeled; clearly the class structure is less evident
plot(x[, 1L], x[, 2L])

# Manually generating synthetic chunklets
chunk1 = sample(1L:100L, 3L)
chunk2 = sample(1L:100L, 3L)
chunk3 = sample(1L:100L, 3L)
chunk4 = sample(1L:100L, 3L)
chunk5 = sample(1L:100L, 3L)
chunk6 = sample(1L:100L, 3L)
chunk7 = sample(101L:200L, 3L)
chunk8 = sample(101L:200L, 3L)
chunk9 = sample(101L:200L, 3L)
chunk10 = sample(101L:200L, 3L)
chunk11 = sample(101L:200L, 3L)
chunk12 = sample(101L:200L, 3L)
chunk13 = sample(101L:200L, 3L)
chunk14 = sample(101L:200L, 3L)
chunk15 = sample(201L:300L, 3L)
chunk16 = sample(201L:300L, 3L)
chunk17 = sample(201L:300L, 3L)
chunk18 = sample(201L:300L, 3L)
chunk19 = sample(201L:300L, 3L)
chunk20 = sample(201L:300L, 3L)
chks = x[c(chunk1, chunk2, chunk3, chunk4, chunk5,
           chunk6, chunk7, chunk8, chunk9, chunk10,
           chunk11, chunk12, chunk13, chunk14, chunk15,
           chunk16, chunk17, chunk18, chunk19, chunk20), ]
chunks = list(chunk1, chunk2, chunk3, chunk4, chunk5,
              chunk6, chunk7, chunk8, chunk9, chunk10,
              chunk11, chunk12, chunk13, chunk14, chunk15,
              chunk16, chunk17, chunk18, chunk19, chunk20)

# Make "chunklet" vector to feed the chunks argument
chunksvec = rep(-1L, nrow(x))
for ( i in 1L:length(chunks) ) {
  for ( j in 1L:length(chunks[[i]]) ) {
    chunksvec[chunks[[i]][j]] = i
  }
}

# Chunklets for the RCA algorithm
plot(chks[, 1L], chks[, 2L], col = rep(1L:20L, each = 3L),
     pch = rep(0L:19L, each = 3L))

# RCA suggested transformation of the data
rca(x, chunksvec)$RCA

# RCA suggested Mahalanobis matrix
rca(x, chunksvec)$B
```

```
# Whitening transformation applied to the  chunklets
chkTransformed = as.matrix(chks) %*% rca(x, chunksvec)$RCA

plot(chkTransformed[, 1L], chkTransformed[, 2L],
     col = rep(1L:20L, each = 3L),
     pch = rep(0L:19L, each = 3L))

# Origin data after applying RCA transformation
xnew = rca(x, chunksvec)$newX
plot(xnew[, 1L], xnew[, 2L],
     bg = c("#E41A1C", "#377EB8", "#4DAF4A")[gl(n, k)],
     pch = c(rep(22, k), rep(21, k), rep(25, k)))
```

Index