

# Package ‘DRR’

January 6, 2018

**Title** Dimensionality Reduction via Regression

**Version** 0.0.3

**Description** An Implementation of Dimensionality Reduction via Regression using Kernel Ridge Regression.

**License** GPL-3 | file LICENSE

**URL** <https://github.com/gdkrmr/DRR>

**BugReports** <https://github.com/gdkrmr/DRR/issues>

**Imports** stats, methods

**Suggests** knitr, rmarkdown

**VignetteBuilder** knitr

**LazyData** true

**Depends** kernlab, CVST, Matrix

**RoxygenNote** 6.0.1

**NeedsCompilation** no

**Author** Guido Kraemer [aut, cre]

**Maintainer** Guido Kraemer <gkraemer@bgc-jena.mpg.de>

**Repository** CRAN

**Date/Publication** 2018-01-06 22:34:19 UTC

## R topics documented:

DRR-package . . . . .	2
constructFastKRRLearner . . . . .	2
drr . . . . .	4
<b>Index</b>	<b>7</b>

---

DRR-package

*Dimensionality Reduction via Regression.*

---

### Description

DRR implements the Dimensionality Reduction via Regression using Kernel Ridge Regression. It also adds a faster implementation of Kernel Ridge regression that can be used with the CVST package.

### Details

Funding provided by the Department for Biogeochemical Integration, Empirical Inference of the Earth System Group, at the Max Plack Institute for Biogeochemistry, Jena.

### Author(s)

**Maintainer:** Guido Kraemer <gkraemer@bgc-jena.mpg.de>

### References

Laparra, V., Malo, J., Camps-Valls, G., 2015. Dimensionality Reduction via Regression in Hyperspectral Imagery. IEEE Journal of Selected Topics in Signal Processing 9, 1026-1036. doi:10.1109/JSTSP.2015.2417833  
Zhang, Y., Duchi, J.C., Wainwright, M.J., 2013. Divide and Conquer Kernel Ridge Regression: A Distributed Algorithm with Minimax Optimal Rates. arXiv:1305.5029 [cs, math, stat].

### See Also

Useful links:

- <https://github.com/gdkrmr/DRR>
- Report bugs at <https://github.com/gdkrmr/DRR/issues>

---

constructFastKRRLearner

*Fast implementation for Kernel Ridge Regression.*

---

### Description

Constructs a learner for the divide and conquer version of KRR.

### Usage

```
constructFastKRRLearner()
```

## Details

This function is to be used with the CVST package as a drop in replacement for [constructKRRLearner](#). The implementation approximates the inversion of the kernel Matrix using the divide and conquer scheme, lowering computational and memory complexity from  $O(n^3)$  and  $O(n^2)$  to  $O(n^3/m^2)$  and  $O(n^2/m^2)$  respectively, where  $m$  are the number of blocks to be used (parameter nblocks). Theoretically safe values for  $m$  are  $< n^{1/3}$ , but practically  $m$  may be a little bit larger. The function will issue a warning, if the value for  $m$  is too large.

## Value

Returns a learner similar to [constructKRRLearner](#) suitable for the use with [CV](#) and [fastCV](#).

## References

Zhang, Y., Duchi, J.C., Wainwright, M.J., 2013. Divide and Conquer Kernel Ridge Regression: A Distributed Algorithm with Minimax Optimal Rates. arXiv:1305.5029 [cs, math, stat].

## See Also

[constructLearner](#)

## Examples

```
ns <- noisySinc(1000)
nsTest <- noisySinc(1000)

fast.krr <- constructFastKRRLearner()
fast.p <- list(kernel="rbfdot", sigma=100, lambda=.1/getN(ns), nblocks = 4)
system.time(fast.m <- fast.krr$learn(ns, fast.p))
fast.pred <- fast.krr$predict(fast.m, nsTest)
sum((fast.pred - nsTest$y)^2) / getN(nsTest)

## Not run:
krr <- CVST::constructKRRLearner()
p <- list(kernel="rbfdot", sigma=100, lambda=.1/getN(ns))
system.time(m <- krr$learn(ns, p))
pred <- krr$predict(m, nsTest)
sum((pred - nsTest$y)^2) / getN(nsTest)

plot(ns, col = '#00000030', pch = 19)
lines(sort(nsTest$x), fast.pred[order(nsTest$x)], col = '#00C000', lty = 2)
lines(sort(nsTest$x), pred[order(nsTest$x)], col = '#0000C0', lty = 2)
legend('topleft', legend = c('fast KRR', 'KRR'),
      col = c('#00C000', '#0000C0'), lty = 2)

## End(Not run)
```

**Description**

drr Implements Dimensionality Reduction via Regression using Kernel Ridge Regression.

**Usage**

```
drr(X, ndim = ncol(X), lambda = c(0, 10^(-3:2)), kernel = "rbfdot",
    kernel.pars = list(sigma = 10^(-3:4)), pca = TRUE, pca.center = TRUE,
    pca.scale = FALSE, fastcv = FALSE, cv.folds = 5, fastcv.test = NULL,
    fastkrr.nblocks = 4, verbose = TRUE)
```

**Arguments**

X	input data, a matrix.
ndim	the number of output dimensions and regression functions to be estimated, see details for inversion.
lambda	the penalty term for the Kernel Ridge Regression.
kernel	a kernel function or string, see <a href="#">kernel-class</a> for details.
kernel.pars	a list with parameters for the kernel. each parameter can be a vector, crossvalidation will choose the best combination.
pca	logical, do a preprocessing using pca.
pca.center	logical, center data before applying pca.
pca.scale	logical, scale data before applying pca.
fastcv	if TRUE uses <a href="#">fastCV</a> , if FALSE uses <a href="#">CV</a> for crossvalidation.
cv.folds	if using normal crossvalidation, the number of folds to be used.
fastcv.test	an optional separate test data set to be used for <a href="#">fastCV</a> , handed over as option test to <a href="#">fastCV</a> .
fastkrr.nblocks	the number of blocks used for fast KRR, higher numbers are faster to compute but may introduce numerical inaccuracies, see <a href="#">constructFastKRRLearner</a> for details.
verbose	logical, should the crossvalidation report back.

**Details**

Parameter combination will be formed and cross-validation used to select the best combination. Cross-validation uses [CV](#) or [fastCV](#).

Pre-treatment of the data using a PCA and scaling is made  $\alpha = Vx$ . the representation in reduced dimensions is

$$y_i = \alpha - f_i(\alpha_1, \dots, \alpha_{i-1})$$

then the final DRR representation is:

$$r = (\alpha_1, y_2, y_3, \dots, y_d)$$

DRR is invertible by

$$\alpha_i = y_i + f_i(\alpha_1, \alpha_2, \dots, \alpha_{i-1})$$

If less dimensions are estimated, there will be less inverse functions and calculating the inverse will be inaccurate.

### Value

A list the following items:

- "fitted.data" The data in reduced dimensions.
- "pca.means" The means used to center the original data.
- "pca.scale" The standard deviations used to scale the original data.
- "pca.rotation" The rotation matrix of the PCA.
- "models" A list of models used to estimate each dimension.
- "apply" A function to fit new data to the estimated model.
- "inverse" A function to untransform data.

### References

Laparra, V., Malo, J., Camps-Valls, G., 2015. Dimensionality Reduction via Regression in Hyperspectral Imagery. IEEE Journal of Selected Topics in Signal Processing 9, 1026-1036. doi:10.1109/JSTSP.2015.2417833

### Examples

```
tt <- seq(0,4*pi, length.out = 200)
helix <- cbind(
  x = 3 * cos(tt) + rnorm(length(tt), sd = seq(0.1, 1.4, length.out = length(tt))),
  y = 3 * sin(tt) + rnorm(length(tt), sd = seq(0.1, 1.4, length.out = length(tt))),
  z = 2 * tt      + rnorm(length(tt), sd = seq(0.1, 1.4, length.out = length(tt)))
)
helix <- helix[sample(nrow(helix)),] # shuffling data is important!!
system.time(
drr.fit <- drr(helix, ndim = 3, cv.folds = 4,
               lambda = 10^(-2:1),
               kernel.pars = list(sigma = 10^(0:3)),
               fastkrr.nblocks = 2, verbose = TRUE,
               fastcv = FALSE)
)
```

```
## Not run:
library(rgl)
plot3d(helix)
points3d(drr.fit$inverse(drr.fit$fitted.data[,1,drop = FALSE]), col = 'blue')
points3d(drr.fit$inverse(drr.fit$fitted.data[,1:2]), col = 'red')

plot3d(drr.fit$fitted.data)
pad <- -3
fd <- drr.fit$fitted.data
xx <- seq(min(fd[,1]), max(fd[,1]), length.out = 25)
yy <- seq(min(fd[,2]) - pad, max(fd[,2]) + pad, length.out = 5)
zz <- seq(min(fd[,3]) - pad, max(fd[,3]) + pad, length.out = 5)

dd <- as.matrix(expand.grid(xx, yy, zz))
plot3d(helix)
for(y in yy) for(x in xx)
  rgl.linestrips(drr.fit$inverse(cbind(x, y, zz)), col = 'blue')
for(y in yy) for(z in zz)
  rgl.linestrips(drr.fit$inverse(cbind(xx, y, z)), col = 'blue')
for(x in xx) for(z in zz)
  rgl.linestrips(drr.fit$inverse(cbind(x, yy, z)), col = 'blue')

## End(Not run)
```

# Index

`constructFastKRRLearner`, [2](#), [4](#)

`constructKRRLearner`, [3](#)

`constructLearner`, [3](#)

`CV`, [3](#), [4](#)

DRR (DRR-package), [2](#)

`drr`, [4](#)

DRR-package, [2](#)

`fastCV`, [3](#), [4](#)