Package 'BKPC'

June 22, 2026

Title Bayesian Kernel Projection Classifier

Version 1.0.2

URL https://github.com/domijan/BKPC

BugReports https://github.com/domijan/BKPC/issues

Description Bayesian kernel projection classifier (Domijan and Wilson,2011) <doi:10.1007/s11222-009-9161-8> is a nonlinear multicategory classifier which performs the classification of the projections of the data to the principal axes of the feature space. A Gibbs sampler is implemented to find the posterior distributions of the parameters.

LazyData true

License MIT + file LICENSE

Depends R (>= 3.5.0)

Imports kernlab

NeedsCompilation yes

Author Katarina Domijan [aut, cre] (ORCID: <https://orcid.org/0000-0002-4268-2236>)

Maintainer Katarina Domijan <domijank@tcd.ie>

Repository CRAN

Date/Publication 2026-06-22 13:10:16 UTC

Contents

BKPC-package *Bayesian Kernel Projection Classifier*

Description

Bayesian kernel projection classifier is a nonlinear multicategory classifier which performs the classification of the projections of the data to the principal axes of the feature space. A Gibbs sampler is implemented to find the posterior distributions of the parameters. The main function for end users is bkpc.

Details

Package:	BKPC
Type:	Package
Version:	1.0
Date:	2016-02-27
License:	GPL-2

Author(s)

K. Domijan <domijank@tcd.ie>

References

Domijan K. and Wilson S. P.: Bayesian kernel projections for classification of high dimensional data. Statistics and Computing, 2011, Volume 21, Issue 2, pp 203-216

See Also

kernlab

bkpc *Bayesian Kernel Projection Classifier*

Description

Function bkpc is used to train a Bayesian kernel projection classifier. This is a nonlinear multicategory classifier which performs the classification of the projections of the data to the principal axes of the feature space. The Gibbs sampler is implemented to find the posterior distributions of the parameters, so probability distributions of prediction can be obtained for for new observations.

Usage

```
## Default S3 method:
bkpc(x, y, theta = NULL, n.kpc = NULL, thin = 100, n.iter = 1e+05, std = 10,
g1 = 0.001, g2 = 0.001, g3 = 1, g4 = 1, initSigmasq = NULL, initBeta = NULL,
initTau = NULL, intercept = TRUE, rotate = TRUE, ...)


## S3 method for class 'kern'
bkpc(x, y, n.kpc = NULL, thin = 100, n.iter = 1e+05, std = 10,
g1 = 0.001, g2 = 0.001, g3 = 1, g4 = 1, initSigmasq = NULL, initBeta = NULL,
initTau = NULL, intercept = TRUE, rotate = TRUE, ...)


## S3 method for class 'kernelMatrix'
bkpc(x, y, n.kpc = NULL, thin = 100, n.iter = 1e+05, std = 10,
g1 = 0.001, g2 = 0.001, g3 = 1, g4 = 1, initSigmasq = NULL, initBeta = NULL,
initTau = NULL, intercept = TRUE, rotate = TRUE, ...)
```

Arguments

x	either: a data matrix, a kernel matrix of class "kernelMatrix" or a kernel matrix of class "kern".
y	a response vector with one label for each row of x. Should be a factor.
theta	the inverse kernel bandwidth parameter.
n.kpc	number of kernel principal components to use.
n.iter	number of iterations for the MCMC algorithm.
thin	thinning interval.
std	standard deviation parameter for the random walk proposal.
g1	γ_1 hyper-parameter of the prior inverse gamma distribution for the σ^2 parameter in the BKPC model.
g2	γ_2 hyper-parameter of the prior inverse gamma distribution for the σ^2 parameter of the BKPC model.
g3	γ_3 hyper-parameter of the prior gamma distribution for the τ parameter in the BKPC model.
g4	γ_4 hyper-parameter of the prior gamma distribution for the τ parameter in the BKPC model.
initSigmasq	optional specification of initial value for the σ^2 parameter in the BKPC model.
initBeta	optional specification of initial values for the β parameters in the BKPC model.
initTau	optional specification of initial values for the τ parameters in the BKPC model.
intercept	if intercept=TRUE (the default) then include the intercept in the model.
rotate	if rotate=TRUE (the default) then run the BKPC model. Else run the BKMC model.
...	Currently not used.

Details

Initial values for a BKPC model can be supplied, otherwise they are generated using `runif` function.

The data can be passed to the `bkpc` function in a matrix and the Gaussian kernel computed using the `gaussKern` function is then used in training the algorithm and predicting. The bandwidth parameter `theta` can be supplied to the `gaussKern` function, else a default value is used.

In addition, `bkpc` also supports input in the form of a kernel matrix of class `"kern"` or `"kernelMatrix"`.The latter allows for a range of kernel functions as well as user specified ones.

If `rotate=TRUE` (the default) then the BKPC is trained. This algorithm performs the classification of the projections of the data to the principal axes of the feature space. Else the Bayesian kernel multicategory classifier (BKMC) is trained, where the data is mapped to the feature space via the kernel matrix, but not projected (rotated) to the principal axes. The hierarchichal prior structure for the two models is the same, but BKMC model is not sparse.

Value

An object of class `"bkpc"` including:

`beta`	realizations of the β parameters from the joint posterior distribution in the BKPC model.
`tau`	realizations of the τ parameters from the joint posterior distribution in the BKPC model.
`z`	realizations of the latent variables z from the joint posterior distribution in the BKPC model.
`sigmasq`	realizations of the σ^2 parameter from the joint posterior distribution in the BKPC model.
`n.class`	number of independent classes of the response variable i.e. number of classes - 1.
`n.kpc`	number of kernel principal components used.
`n.iter`	number of iterations of the MCMC algorithm.
`thin`	thinning interval.
`intercept`	if true, intercept was included in the model.
`rotate`	if true, the sparse BKPC model was fitted, else BKMC model.
`kPCA`	if `rotate=TRUE` an object of class `"kPCA"`, else `NULL`.
`x`	the supplied data matrix or kernel matrix.
`theta`	if data was supplied, as opposed to the kernel, this is the inverse kernel bandwidth parameter used in obtaining the Gaussian kernel, else `NULL`.

Note

If supplied, data are not scaled internally. If `rotate=TRUE` the mapping is centered internally by the `kPCA` function.

Author(s)

K. Domijan

References

Domijan K. and Wilson S. P.: Bayesian kernel projections for classification of high dimensional data. Statistics and Computing, 2011, Volume 21, Issue 2, pp 203-216

See Also

kPCA gaussKern predict.bkpc plot.bkpc summary.bkpc kernelMatrix (in package **kernlab**)

Examples

```
set.seed(-88106935)

data(microarray)

# consider only four tumour classes (NOTE: "NORM" is not a class of tumour)
y <- microarray[, 2309]
train <- as.matrix(microarray[y != "NORM", -2309])
wtr <- factor(microarray[y != "NORM", 2309], levels = c("BL" ,  "EWS" , "NB" ,"RMS" ))

n.kpc <- 6
n.class <- length(levels(wtr)) - 1

K <- gaussKern(train)$K

# supply starting values for the parameters
# use Gaussian kernel as input

result <- bkpc(K, y = wtr, n.iter = 1000,  thin = 10, n.kpc = n.kpc,
initSigmasq = 0.001, initBeta = matrix(10, n.kpc *n.class, 1),
initTau =matrix(10, n.kpc * n.class, 1), intercept = FALSE, rotate = TRUE)

# predict

out <- predict(result, n.burnin = 10)

table(out$class, as.numeric(wtr))

# plot the data projection on the kernel principal components


pairs(result$kPCA$KPCs[, 1 : n.kpc], col = as.numeric(wtr),
main =  paste("symbol = predicted class", "\n", "color = true class" ),
pch = as.numeric(out$class), upper.panel = NULL)
par(xpd=TRUE)
legend('topright', levels(wtr), pch = unique(out$class),
col = as.numeric(unique(wtr)), bty = "n")



# Another example: Iris data

data(iris)
```

```
testset <- sample(1:150,50)

train <- as.matrix(iris[-testset,-5])
test <- as.matrix(iris[testset,-5])

wtr <- iris[-testset, 5]
wte <- iris[testset, 5]

# use default starting values for paramteres in the model.

result <- bkpc(train, y = wtr,  n.iter = 1000,  thin = 10, n.kpc = 2,
intercept = FALSE, rotate = TRUE)

# predict
out <- predict(result, test, n.burnin = 10)

# classification rate
sum(out$class == wte)/dim(test)[1]

table(out$class, wte)

## Not run:
# Another example: synthetic data from MASS library

library(MASS)

train<- as.matrix(synth.tr[, -3])
test<- as.matrix(synth.te[, -3])

wtr <- as.factor(synth.tr[, 3])
wte <- as.factor(synth.te[, 3])


#  make training set kernel using kernelMatrix from kernlab library

library(kernlab)

kfunc <- laplacedot(sigma = 1)
Ktrain <- kernelMatrix(kfunc, train)

#  make testing set kernel using kernelMatrix {kernlab}

Ktest <- kernelMatrix(kfunc, test, train)

result <- bkpc(Ktrain, y = wtr, n.iter = 1000,  thin = 10,  n.kpc = 3,
intercept = FALSE, rotate = TRUE)

# predict

out <- predict(result, Ktest, n.burnin = 10)

# classification rate
```

```
sum(out$class == wte)/dim(test)[1]
table(out$class, wte)


# embed data from the testing set on the new space:

KPCtest <- predict(result$kPCA, Ktest)

# new data is linearly separable in the new feature space where classification takes place
library(rgl)
plot3d(KPCtest[ , 1 :  3], col = as.numeric(wte))
pairs(KPCtest[ , 1 :  3], col = as.numeric(wte))

# another model:  do not project the data to the principal axes of the feature space.
# NOTE: Slow
# use Gaussian kernel with the default bandwidth parameter

Ktrain <- gaussKern(train)$K

Ktest <- gaussKern(train, test, theta = gaussKern(train)$theta)$K

resultBKMC <- bkpc(Ktrain, y = wtr, n.iter = 1000,  thin = 10,
intercept = FALSE, rotate = FALSE)

# predict
outBKMC <- predict(resultBKMC, Ktest, n.burnin = 10)

# to compare with previous model
table(outBKMC$class, wte)


# another example: wine data from gclus library

library(gclus)
data(wine)

testset <- sample(1 : 178, 90)
train <- as.matrix(wine[-testset, -1])
test <- as.matrix(wine[testset, -1])

wtr <- as.factor(wine[-testset, 1])
wte <- as.factor(wine[testset, 1])

#  make training set kernel using kernelMatrix from kernlab library

kfunc <- anovadot(sigma = 1, degree = 1)
Ktrain <- kernelMatrix(kfunc, train)

#  make testing set kernel using kernelMatrix {kernlab}
Ktest <- kernelMatrix(kfunc, test, train)

result <- bkpc(Ktrain, y = wtr, n.iter = 1000,  thin = 10,  n.kpc = 3,
intercept = FALSE, rotate = TRUE)
```

```
out <- predict(result, Ktest, n.burnin = 10)

# classification rate in the test set
sum(out$class == wte)/dim(test)[1]


# embed data from the testing set on the new space:
KPCtest <- predict(result$kPCA, Ktest)

# new data is linearly separable in the new feature space where classification takes place


pairs(KPCtest[ , 1 :  3], col = as.numeric(wte),
main =  paste("symbol = predicted class", "\n", "color = true class" ),
pch = as.numeric(out$class), upper.panel = NULL)

par(xpd=TRUE)

legend('topright', levels(wte), pch = unique(out$class),
col = as.numeric(unique(wte)), bty = "n")



## End(Not run)
```

---

gaussKern *Gaussian kernel*

Description

Calculates Gaussian kernel: $k(x, x') = \exp(-\theta\|x - x'\|^2)$

Usage

```
gaussKern(x, newdata = x, theta = NULL)
```

Arguments

x	a data matrix.
newdata	optional second data matrix.
theta	the inverse kernel bandwidth parameter. If NULL a default value is used $\theta = 1/max(\|x - x'\|^2)$.

Details

Also known as the radial basis kernel function, see rbfdot (in package **kernlab**)

Value

Returns a list containing the following components:

K	a Gaussian kernel matrix of class "kern".
theta	the inverse kernel bandwidth parameter.

Author(s)

K. Domijan

See Also

kPCA bkpc kernelMatrix (in package **kernlab**)

Examples

```
data(iris)

testset <- sample(1:150,20)
train <- as.matrix(iris[-testset ,-5])
test <- as.matrix(iris[testset ,-5])


# make training set kernel
gk <- gaussKern(train)
Ktrain <- gk$K

image(Ktrain)

# make testing set kernel
gk2 <- gaussKern(train, test, gk$theta)
Kest <- gk2$K
```

---

kPCA *Kernel Principal Components Analysis*

Description

Kernel PCA is a nonlinear generalization of principal component analysis.

Usage

```
## Default S3 method:
kPCA(x, theta = NULL, ...)

## S3 method for class 'kern'
kPCA(x, ...)
```

```
## S3 method for class 'kernelMatrix'
kPCA(x, ...)
```

Arguments

x	either: a data matrix, a kernel matrix of class "kernelMatrix" or a kernel matrix of class "kern".
theta	the inverse kernel bandwidth parameter for the Gaussian kernel.
...	Currently not used.

Details

The data can be passed to the kPCA function in a matrix and the Gaussian kernel (via the gaussKern function) is used to map the data to the high-dimensional feature space where the principal components are computed. The bandwidth parameter theta can be supplied to the gaussKern function, else a default value is used. Alternatively, the Gaussian kernel matrix of class "kern" can be supplied to the kPCA function directly. In addition, kPCA also supports input in the form of a kernel matrix of class "kernelMatrix" (in package **kernlab**) thus allowing for other kernel functions.

Value

An object of class "kPCA" including:

KPCs	The original data projected on the principal components.
Es	the corresponding eigenvalues.
Vecs	a matrix containing principal component vectors (column wise).
K	a kernel matrix of class "kernelMatrix" or of class "kern".
theta	if Gaussian kernel was calculated, this is the bandwidth parameter used in its calculation.
x	if supplied, the original data matrix.

Note

The predict function can be used to embed new data on the new space

Author(s)

K. Domijan

References

Schoelkopf B., A. Smola, K.-R. Mueller : Nonlinear component analysis as a kernel eigenvalue problem. Neural Computation 10, 1299-1319.

See Also

gaussKern kpca (in package **kernlab**) kernelMatrix (in package **kernlab**)

Examples

```
data(iris)
testset <- sample(1:150,20)

train <- as.matrix(iris[-testset,-5])
test <- as.matrix(iris[testset,-5])


# make training set kernel

gk <- gaussKern(train)
Ktrain <- gk$K
image(Ktrain)

# make testing set kernel

gk2 <- gaussKern(train, test, gk$theta)
Ktest <- gk2$K


#  make training set kernel using kernelMatrix from library kernlab.

library(kernlab)

kfunc <- laplacedot(sigma = 1)
Ktrain2 <- kernelMatrix(kfunc, train)
image(Ktrain2)

# make testing set kernel using kernelMatrix {kernlab}

Ktest2 <- kernelMatrix(kfunc, test, train)



# Do KPCA:

kpcData <- kPCA(train)
kpcKern <- kPCA(Ktrain)
kpcKern2 <- kPCA(Ktrain2)


# plot the data projection on the principal components

pairs(kpcData$KPCs[ , 1 : 3], col = iris[-testset, 5])

# proportion of variance explained by each PC

plot(kpcData$Es/sum(kpcData$Es), xlab = "PC", ylab = "Proportion of variance")


# embed data from the testing set on the new space:
```

```
KPCpred1 <- predict(kpcData, test)

KPCpred2 <- predict(kpcKern, Ktest)

KPCpred3 <- predict(kpcKern2, Ktest2)

#plot the test data projection on the principal components

pairs(KPCpred3[ , 1 : 3], col = iris[testset, 5])
```

`marginalRelevance` *Feature Marginal Relevance*

Description

Calculates Marginal Relevance of each feature (Fisher criterion) useful for class (group) separation. The marginal relevance score is a ratio of the between-group to within-group sum of squares.

Usage

```
marginalRelevance(x, y)
```

Arguments

`x`	a data matrix.
`y`	a response vector. Should be a factor.

Value

An object of class `"marginalRelevance"` including:

`score`	Marginal relevance score of each feature.

Author(s)

K. Domijan

References

Dudoit S., J. Fridlyand, T. P. Speed: Comparison of discrimination methods for the classification of tumors using gene expression data. Journal of the American Statistical Association, 2002, Volume 97 No 457, pp 77-87.

See Also

microarray

Examples

```
## Not run:
data(microarray)

profiles <- as.matrix(microarray[, -2309])
tumourType <-  microarray[, 2309]

margRelv <- marginalRelevance(profiles, tumourType)

#plot 30 gene profiles with highest marginal


bestVars <- match(1:ncol(profiles), rank(margRelv$score, ties.method = "random"))


library(gclus)

 cparcoord(profiles[,bestVars[2308:2280]],  col = tumourType)


cpairs(profiles[,bestVars[2308:2300]], col = tumourType)


# another example: wine data from gclus
library(gclus)
data(wine)
dt <- as.matrix(wine[, -1])
colnames(dt) <- names(wine[, -1])

label <- as.factor(wine[, 1])


margRelv <- marginalRelevance(dt, label)


## End(Not run)
```

`microarray` *Microarray gene expression data*

Description

Microarray gene expression data published by Khan et al. (2001). There are 2308 gene expression profiles recorded over 88 arrays.

Usage

```
data("microarray")
```

Format

A data frame with 88 observations on the following 2309 variables.

The first 2308 variables are the gene expression values for 88 arrays. The first 63 arrays correspond to the training set and the remaining 25 are from the testing set of the filtered data made available in the supplementary files. The last variable is the tumour class, a factor with levels `BL`, `EWS`, `NB`, `NORM`, `RMS`.

References

Khan, J., Wei, J.S., Ringner, M., Saal, L.H., Ladanyi, M., Westermann, F., Berthold, F., Schwab, M., Antonescu, C.R., Peterson, C. et al.: Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks. 2001. Nat. Med., 7, 673-679.

---

plot.bkpc *Plot bkpc Objects*

Description

Plots realizations of the parameters from the joint posterior distribution in the BKPC model. The default plots show: medians, 10th and 90th percentiles. The "tracePlot" and "boxPlot" show the traceplots and boxplots of the samples.

Usage

```
## S3 method for class 'bkpc'
plot(x, type = "default", n.burnin = 0, ...)
```

Arguments

`x`	a bkpc object.
`type`	"tracePlot", "boxPlot" or default.
`n.burnin`	number of burn-in iterations from the thinned sample to discard.
`...`	options directly passed to the plot function.

Author(s)

K. Domijan

See Also

`bkpc` `summary.bkpc`

Examples

```
set.seed(1)

data(microarray)

# consider only four tumour classes (NOTE: "NORM" is not a class of tumour)
y <- microarray[, 2309]
train <- as.matrix(microarray[y != "NORM", -2309])
wtr <- factor(microarray[y != "NORM", 2309], levels = c("BL" ,  "EWS" , "NB" ,"RMS" ))

n.kpc <- 6
n.class <- length(levels(wtr)) - 1

K <- gaussKern(train)$K

# supply starting values for the parameters
# use Gaussian kernel as input

result <- bkpc(K, y = wtr, n.iter = 10000,  thin = 100, n.kpc = n.kpc,
initSigmasq = 0.001, initBeta = matrix(10, n.kpc *n.class, 1),
initTau =matrix(10, n.kpc * n.class, 1), intercept = FALSE, rotate = TRUE)



plot(result, type = "tracePlot")
plot(result, type = "boxPlot", n.burnin = 20)
plot(result, n.burnin = 20)
```

`predict.bkpc` *Predict Method for Bayesian Kernel Projection Classifier*

Description

This function predicts values based upon a model trained by bkpc for new input data. BKPC employs a Gibbs sampler to sample from the posterior distributions of the parameters, so sample probability distributions of prediction can be obtained for for new data points.

Usage

```
## S3 method for class 'bkpc'
predict(object, newdata  = NULL, n.burnin = 0, ...)
```

Arguments

`object`	a bkpc object.
`newdata`	a matrix containing the new input data
`n.burnin`	number of burn-in iterations to discard.
`...`	Currently not used.

Details

If newdata is omitted the predictions are based on the data used for the fit.

Value

A list with the following components:

class	estimated class for each observation in the input data.
map	maximum a posteriori probability estimate for belonging to each class for all observations in the input data.
p	a matrix of samples of estimated probabilities for belonging to each class for each observation in the input data.

Author(s)

K. Domijan

References

Domijan K. and Wilson S. P.: Bayesian kernel projections for classification of high dimensional data. Statistics and Computing, 2011, Volume 21, Issue 2, pp 203-216

See Also

bkpc

Examples

```
set.seed(-88106935)

data(iris)
testset <- sample(1:150,30)

train <- as.matrix(iris[-testset,-5])
test <- as.matrix(iris[testset,-5])

wtr <- iris[-testset, 5]
wte <- iris[testset, 5]

result <- bkpc(train, y = wtr,  n.iter = 1000,  thin = 10, n.kpc = 2,
intercept = FALSE, rotate = TRUE)

# predict
out <- predict(result, test, n.burnin = 20)

# classification rate for the test set

sum(out$class == as.numeric(wte))/dim(test)[1]

table(out$class, as.numeric(wte))
```

```
# consider just misclassified observations:

missclassified <- out$class != as.numeric(wte)


tab <- cbind(out$map[missclassified, ], out$class[missclassified], as.numeric(wte)[missclassified])
colnames(tab) = c("P(k = 1)", "P(k = 2)", "P(k = 3)", "predicted class", "true class")
tab


# consider, say, 28th observation in the test set:
# sample probability distributions of belonging to each of the three classes:


ProbClass2samples <- out$p[28, ]
ProbClass3samples <- out$p[28 + dim(test)[1], ]
ProbClass1samples <- 1 - (ProbClass2samples + ProbClass3samples)
hist(ProbClass1samples)
hist(ProbClass2samples)
hist(ProbClass3samples)
```

---

summary.bkpc *Summary statistics for Markov Chain Monte Carlo chain from Bayesian Kernel Projection Classifier*

Description

summary.bkpc produces two sets of summary statistics for each variable: mean and standard deviation (ignoring autocorrelation of the chain) of the sample distribution and quantiles of the sample distribution using the quantiles argument.

Usage

```
## S3 method for class 'bkpc'
summary(object, quantiles = c(0.025, 0.25, 0.5, 0.75, 0.975), n.burnin = 0, ...)
```

Arguments

object	an object of class "bkpc".
quantiles	a vector of quantiles to evaluate for each variable.
n.burnin	number of burn-in iterations to discard from the thinned sample.
...	Currently not used.

Author(s)

K. Domijan

See Also

bkpc plot.bkpc

Examples

```
set.seed(-88106935)

data(iris)
testset <- sample(1:150,50)

train <- as.matrix(iris[-testset,-5])
test <- as.matrix(iris[testset,-5])

wtr <- iris[-testset, 5]
wte <- iris[testset, 5]

result <- bkpc(train, y = wtr,  n.iter = 1000,  thin = 10, n.kpc = 2,
intercept = FALSE, rotate = TRUE)


summary(result, n.burnin = 0)
```

Index