Package 'kernDeepStackNet'

May 31, 2017

Version 2.0.2

Title Kernel Deep Stacking Networks

Date 2017-05-31

Author Thomas Welchowski <welchow@imbie.meb.uni-bonn.de> and Matthias Schmid <matthias.schmid@imbie.uni-bonn.de>

Maintainer Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

Description Contains functions for estimation and model selection of kernel deep stacking networks.

Imports methods, Rcpp, glmnet, mvtnorm, lhs, DiceOptim, DiceKriging, parallel, globalOptTests, GenSA, GA, matrixStats, caret

Suggests pROC, mgcv, microbenchmark

License GPL-3

LazyLoad yes

LinkingTo Rcpp, RcppEigen

RoxygenNote 5.0.0

NeedsCompilation yes

Repository CRAN

Date/Publication 2017-05-31 13:38:23 UTC

R topics documented:

`kernDeepStackNet-package`

Kernel deep stacking networks with random Fourier transformation

---

Description

Contains functions for estimation and model selection of kernel deep stacking networks (KDSNs). KDSNs are a supervised learning method, which can be used for continuous or binary responses. The model specification follows the approach of Huang et al. (2013), which is based on a series of kernel ridge regression models to random Fourier transformed input data.

The model selection includes model-based optimization of arbitrary loss functions. All help functions are also available for customized modeling, but it is recommended to use the higher level functions. The main functions are

- `fitKDSN`: Fits kernel deep stacking networks.

- tuneMboLevelCvKDSN: Selection of tuning parameters of kernel deep stacking networks with model-based optimization using cross-valdiation, arbitrary loss functions and pre-specified number of levels. All other tuning parameters are free to vary across the levels.
- tuneMboLevelGcvKDSN: Selection of tuning parameters of kernel deep stacking networks with model-based optimization using generalized cross validation score, arbitrary loss functions and pre-specified number of levels. All other tuning parameters are free to vary across the levels.
- tuneMboSharedCvKDSN: Selection of tuning parameters of kernel deep stacking networks with model-based optimization using cross validation or test set and arbitrary loss functions. The number of levels is included in tuning. All other tuning parameters are shared across the levels.
- tuneMboSharedSubsetKDSN: Selection of tuning parameters of kernel deep stacking networks with model-based optimization using test sets and arbitrary loss functions. The number of levels is included in tuning. All other tuning parameters are shared across the levels. In contrast to function tuneMboSharedCvKDSN an ensemble of (sparse) KDSNs is estimated.

For examples and further information, see the corresponding readme pages of the main functions.

Details

Package:	kernDeepStackNet
Type:	Package
Version:	2.0.2
Date:	2017-05-31
License:	GPL-3

Author(s)

Maintainer: Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

Matthias Schmid <matthias.schmid@imbie.uni-bonn.de>

References

Thomas Welchowski, Matthias Schmid, (2016), *A framework for parameter estimation and model selection in kernel deep stacking networks*, Artificial Intelligence in Medicine, volume 70, pages 31-40

Po-Seng Huang and Li Deng and Mark Hasegawa-Johnson and Xiaodong He, (2013), *Random Features for kernel deep convex network*, Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)

Simon N. Wood, (2006), *Generalized Additive Models: An Introduction with R*, Taylor & Francis Group LLC

R. Brent, (1973), *Algorithms for Minimization without Derivatives*, Englewood Cliffs N.J.: Prentice-Hall

Donald R. Jones and Matthias Schonlau and William J. Welch, (1998), *Efficient Global Optimization of Expensive Black-Box Functions*, Journal of Global Optimization 13: pages 455-492

Krige DG, (1951), *A Statistical Approach to Some Basic Mine Valuation Problems on the Witwatersrand*, Journal of the Chemical, Metallurgical and Mining Society of South Africa, 52(6), 119-139

Olivier Roustant and David Ginsbourger and Yves Deville, (2012), *DiceKriging, DiceOptim: Two R Packages for the Analysis of Computer Experiments by Kriging-Based Metamodeling and Optimization*, Journal of Statistical Software, Volume 51, Issue 1

Michael Stein, (1987), *Large Sample Properties of Simulations Using Latin Hypercube Sampling*, Technometrics. 29, 143-151

Carl Edward Rasmussen and Christopher K. I. Williams, (2006), *Gaussian Processes for Machine Learning*, Massachusetts Institute of Technology

Jerome Friedman and Trevor Hastie and Rob Tibshirani, (2008), *Regularization Paths for Generalized Linear Models via Coordinate Descent*, Department of Statistics, Stanford University

Victor Picheny, David Ginsbourger, Yann Richet, (2012), *Quantile-based optimization of Noisy Computer Experiments with Tunable Precision*, HAL-archives-ouvertes.fr, hal-00578550v3

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky et al., (2014), *Dropout: A simple way to prevent neural networks from overfitting*, Journal of Machine Learning Research, volume 15, pages 1929-1958

David Lopez-Paz and Philipp Hennig and Bernhard Sch\"olkopf, (2013), *The randomized dependence coefficient*, https://arxiv.org/, reference arXiv:1304.7717

Luca Scrucca, (2013), *GA: A Package for Genetic Algorithms in R*, Journal of Statistical Software, issue 4, volume 53, pages 1-37

Constantino Tsallis, Daniel A. Stariolo, (1996), *Generalized simulated annealing*, Elsevier, Physica A, volume 233, pages 395-406

`calcTrA` *Calculates the trace of the hat matrix*

Description

The trace of the hat matrix corresponds to the effective degrees of freedom (edf) of a generalized additive model. The edf is used as an internal measure of model complexity.

Usage

```
calcTrA(W, X, lambda=0)
```

Arguments

`W`	Weight matrix of the pseudo iterated least squares algorithm. See function `calcWdiag`
`X`	Design matrix of the covariates.
`lambda`	Regularization parameter of kernel ridge regression.

Value

Effective degrees of freedom of a generalized additive model with regularization (numeric scalar).

Note

This function is not intended to be called directly by the user. Should only be used by experienced users, who want to customize the model.

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

References

Simon N. Wood, (2006), *Generalized Additive Models: An Introduction with R*, Taylor \& Francis Group LLC

See Also

varMu, calcWdiag, gDerivMu

---

calcTrAFast *Calculates the trace of the hat matrix as C version*

Description

The trace of the hat matrix corresponds to the effective degrees of freedom (edf) of a generalized additive model. The edf is an internal measure of model complexity.

Usage

```
calcTrAFast(X, w, lambda=0)
```

Arguments

X	Design matrix of the covariates.
w	Diagonal weight matrix of the pseudo iterated least squares algorithm. See function calcWdiag.
lambda	Regularization parameter of kernel ridge regression. Default is 0 (numeric scalar).

Details

This function is a more computational efficient version of calcTrA. The general algorithm is simplified, requires less memory and is faster. Therefore it is better suited for data sets above 1000 observations.

Value

Effective degrees of freedom of a generalized additive model with regularization (numeric scalar).

Note

This function is not intended to be called directly by the user. Should only be used by experienced users, who want to customize the model.

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

References

Simon N. Wood, (2006), *Generalized Additive Models: An Introduction with R*, Taylor \& Francis Group LLC

See Also

varMu, gDerivMu

---

calcWdiag *Calculation of weight matrix*

Description

Calculates the weight matrix in the pseudo iterative reweighted least squares step of a generalized linear model. It is based on the variance function and the first derivative of the link function. The values are calculated at the actual expected values. The dimension of the matrix is n x n (n = number of observations). For details see reference.

Usage

```
calcWdiag(varMu, gDerivMu)
```

Arguments

varMu	Value of the variance function of the exponential family on the expected value mu (numeric scalar).
gDerivMu	Value of the first derivative of the link function evaluated at the expected value. (numeric scalar).

Value

Weight matrix (numeric vector).

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

References

Simon N. Wood, (2006), *Generalized Additive Models: An Introduction with R*, Taylor \& Francis Group LLC

See Also

calcTrA, varMu, gDerivMu

---

cancorRed *Calculate first canonical correlation*

Description

Calculates the first canonical correlation between two sets of variables.

Usage

```
cancorRed(x, y, xcenter = TRUE, ycenter = TRUE)
```

Arguments

x	Numeric matrix (n * p1), containing the x coordinates.
y	Numeric matrix (n * p2), containing the y coordinates.
xcenter	Logical or numeric vector of length p1, describing any centering to be done on the x values before the analysis. If the argument is TRUE (default), then all columns are centered. Otherwise not adjustment is made.
ycenter	analogous to xcenter, but for the y values.

Details

The canonical correlation analysis seeks linear combinations of the y variables which are well explained by linear combinations of the x variables. The relationship is symmetric as 'well explained' is measured by correlations.

Value

First canonical correlation (numeric vector).

Note

This function is a reduced version of the original Code in the base package cancor. In comparison to the original, it does only calculate the first canonical correlation, but the runtime is reduced.

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

See Also

cancor, rdcPart, rdcVarOrder

---

crossprodRcpp *Calculates the cross product of a matrix*

Description

This is an computational efficient help function to calculate the GCV loss of the KDSN. Calculates the matrix product A^T A. It is written in C and faster than the regular R version.

Usage

```
crossprodRcpp(A)
```

Arguments

A	Numeric Matrix. Integer matrices are not allowed.

Value

List with the numeric matrix A^T A as element.

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

See Also

crossprod

Examples

```
A <- matrix(seq(-1, 1, length.out=9), nrow=3, ncol=3)
crossprodRcpp(A)[[1]]
all.equal(crossprodRcpp(A)[[1]], crossprod(A))
```

devStandard *Predictive deviance of a linear model*

Description

Calculates the deviance on new data observations. The predictive deviance measures how far the predicted values are apart from the saturated model in the test set.

Usage

```
devStandard(preds, ytest, RMSE=TRUE)
```

Arguments

preds	Predictions of the specified model (numeric vector).
ytest	Data values of the response in the test data.
RMSE	Should the default sum of squares be computed or the RMSE? Default is RMSE.

Details

In the "Gaussian" case it is defined to be the residual sum of squares. `ytest` are the test observations and `preds` are the predicted values of the model on the test data.

Value

Predictive deviance of the linear model, given predictions of test data (numeric scalar).

Note

This function is not intended to be called directly by the user. Should only be used by experienced users, who want to customize the model. It is called in the model selection process of the kernel deep stacking network with cross-validation, e.g. `lossCvKDSN`. The RMSE is used as default, because kriging models may be more stable with smaller variances of the performance criterion

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

References

Simon N. Wood, (2006), *Generalized Additive Models: An Introduction with R*, Taylor \& Francis Group LLC

Examples

```
##################
# Fit Gaussian glm

set.seed(10)
x <- matrix(rnorm(100*20),100,20)
set.seed(100)
y <- rnorm(100)
fit1 <- glm(formula=y ~ ., data=data.frame(x))
preds <- predict(fit1, type="response")
# Performance on training data
all.equal(devStandard(preds=preds, ytest=y, RMSE=FALSE), fit1$deviance)
# Performance on random test data
set.seed(100)
yTest <- simulate(fit1)
devStandard(preds=preds, ytest=yTest)
```

EImod *Expected improvement criterion replacement function*

Description

This function is a replacement of the function EI in package DiceOptim.

Usage

```
EImod (x, model, plugin = NULL, type = "UK", minimization = TRUE,
    envir = NULL)
```

Arguments

x	a vector representing the input for which one wishes to calculate EI,
model	an object of class km
plugin	optional scalar: if provided, it replaces the minimum of the current observations,
type	"SK" or "UK" (by default), depending whether uncertainty related to trend estimation has to be taken into account,
minimization	logical specifying if EI is used in minimiziation or in maximization,
envir	an optional environment specifying where to assign intermediate values for future gradient calculations. Default is NULL.

Value

The expected improvement, defined as

$EI(x) := E[(min(Y(X)) - Y(x))^+ | Y(X) = y(X)]$,

where X is the current design of experiments and Y is the random process assumed to have generated the objective function y. If a plugin is specified, it replaces

$min(Y(X))$

in the previous formula.

Note

For further examples see DiceOptim::EI

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

See Also

EI

`fineTuneCvKDSN` *Fine tuning of random weights of a given KDSN model*

Description

Weight matrices are randomly drawn given a KDSN model structure with prespecified number of levels, dimensions of the random Fourier transformations, precision parameter sigma and regularization parameter lambda. This function supports arbitrary loss functions, which are evaluated on test data. The model with the lowest loss is chosen.

Usage

```
fineTuneCvKDSN(estKDSN, y, X, fineTuneIt=100, info=TRUE,
seedInitW=NULL, seedInitD=NULL, ncpus=1, cvIndex, lossFunc=devStandard)
```

Arguments

`estKDSN`	Model of class "KDSN" `fitKDSN`
`y`	Response of the regression (must be in one column matrix format).
`X`	Design matrix. All factors must be already encoded.
`fineTuneIt`	Number of random generated iterations. Default is 100 (numeric scalar).
`info`	Should additional informations be displayed? Default is TRUE (logical scalar).
`seedInitW`	Gives the seed initialization of the random seeds for weight matrix generation. Each element of the integer vector corresponds to one level. Default is NULL `Random`.
`seedInitD`	Gives the seed initialization of the random seeds for dropout generation. Each element of the integer vector corresponds to one level. Default is NULL `Random`.
`ncpus`	Gives the number of cpus (threads), which should be used. The parallization is based on the parallel package.

cvIndex	Index of cross-validation indices. The indices represent the training data. Must be supplied as list, the required format is identical to the output of the createFolds with argument returnTrain=TRUE.
lossFunc	Specifies how the loss on the test data should be evaluated. Defaults to predictive deviance devStandard.

Details

It is important that the estimated KDSN has a custom set seed (argument "estKDSN") to ensure reproducibility. Otherwise the model is refitted using the default seed values seq(0, (Number of Levels-1), 1) are used for random Fourier transformation.

Value

Model of class "KDSN". For reproduceability the best seeds of the random iterations are stored in the final model.

Note

This procedure is already included in the other tuning algorithms, e. g. tuneMboLevelCvKDSN. It can be used after finding the structure of the KDSN. This may improve predictive accuracy in some cases, but it is not always necessary.

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

References

Simon N. Wood, (2006), *Generalized Additive Models: An Introduction with R*, Taylor \& Francis Group LLC

See Also

tuneMboLevelCvKDSN, tuneMboSharedCvKDSN, tuneMboLevelGcvKDSN

Examples

```
####################################
# Example with binary outcome

# Generate covariate matrix
sampleSize <- 100
X <- matrix(0, nrow=100, ncol=5)
for(j in 1:5) {
  set.seed (j)
  X [, j] <- rnorm(sampleSize)
}

# Generate bernoulli response
rowSumsX <- rowSums(X)
```

```
logistVal <- exp(rowSumsX) / (1 + exp(rowSumsX))
set.seed(-1)
y <- sapply(1:100, function(x) rbinom(n=1, size=1, prob=logistVal[x]))

# Generate test indices
library(caret)
set.seed(123)
cvFoldIndices <- createFolds(y=y, k=2, returnTrain=TRUE)

# Fit kernel deep stacking network with three levels
# Initial seed should be supplied in fitted model!
fitKDSN1 <- fitKDSN(y=y, X=X, levels=3, Dim=c(20, 10, 5),
             sigma=c(0.5, 1, 2), lambdaRel=c(1, 0.1, 0.01),
             alpha=rep(0, 3), info=TRUE, seedW=c(0, 1:2))

# Apply additional fine tuning based on predictive deviance
fitKDSN2 <- fineTuneCvKDSN(estKDSN=fitKDSN1, y=y, X=X,
fineTuneIt=25, info=TRUE, cvIndex=cvFoldIndices)

# Generate new test data
sampleSize <- 100
Xtest <- matrix(0, nrow=100, ncol=5)
for(j in 1:5) {
  set.seed (j+50)
  Xtest [, j] <- rnorm(sampleSize)
}
rowSumsXtest <- rowSums(Xtest)
logistVal <- exp(rowSumsXtest) / (1 + exp(rowSumsXtest))
set.seed(-1)
ytest <- sapply(1:100, function(x) rbinom(n=1, size=1, prob=logistVal[x]))

# Evaluate on test data with auc
library(pROC)
preds <- predict(fitKDSN1, Xtest)
auc1 <- auc(response=ytest, predictor=c(preds))
preds <- predict(fitKDSN2, Xtest)
auc2 <- auc(response=ytest, predictor=c(preds))
auc1 < auc2 # TRUE
# The fine tuning improved the test auc
```

`fitEnsembleKDSN` *Fit an ensemble of KDSN (experimental)*

Description

Given a KDSN structure, either tuned `tuneMboLevelCvKDSN`, `tuneMboSharedCvKDSN` or estimated `fitKDSN`, bootstrap samples are choosen of the original data. Then new random Fourier weights and random Dropout (if specified) are generated. A prespecified number of ensembles is fitted.

Usage

```
fitEnsembleKDSN (estKDSN, y, X, ensembleSize=100,
                 bagging=rep(FALSE, ensembleSize), seedBag=NULL,
                 randSubsets=rep(FALSE, ensembleSize),
                 seedRandSubSize=NULL, seedRandSubDraw=NULL,
                 seedW1=NULL, seedW2=NULL,
                 seedDrop1=NULL, seedDrop2=NULL,
                 info=TRUE, nCores=1, saveOnDisk=FALSE,
                 dirNameDisk=paste(tempdir(), "/ensembleModel", sep=""))
```

Arguments

estKDSN	Model of class "KDSN" fitKDSN.
y	Response of the regression (must be in one column matrix format).
X	Design matrix. All factors must be already encoded.
ensembleSize	Number of ensembles. Default is 100 (integer scalar).
bagging	Should bagging be used in ensemble? Default is FALSE. Each entry corresponds to usage in one ensemble (logical vector).
seedBag	Gives the seed initialization of the bootstrap samples. Each element of the integer vector corresponds to one ensemble. Default is NULL Random.
randSubsets	Should random subsets be used in ensemble? Default is FALSE. Each entry corresponds to usage in one ensemble (logical vector).
seedRandSubSize	Seed for size of random subsets selection. Each entry corresponds to the seed of one ensemble (integer vector).
seedRandSubDraw	Seed for random subsets selection given size. Each entry corresponds to the seed of one ensemble (integer vector).
seedW1	Gives the seed initialization of the bootstrap samples. The random number generation is split into two parts: This seed represents drawing random integer numbers. Each element of the integer vector corresponds to one ensemble. Default is NULL Random.
seedW2	Gives the seed initialization of the bootstrap samples. The random number generation is split into two parts: This seed represents drawing random signs. Each element of the integer vector corresponds to one ensemble. Default is NULL Random.
seedDrop1	Gives the seed initialization of the dropout applied in the random Fourier transformation. The random number generation is split into two parts: This seed represents drawing random integer numbers. Each element of the integer vector corresponds to one ensemble. Default is NULL Random.
seedDrop2	Gives the seed initialization of the dropout applied in the random Fourier transformation. The random number generation is split into to parts: This seed represents drawing random signs. Each element of the integer vector corresponds to one ensemble. Default is NULL Random.
info	Should additional informations be displayed? Default is TRUE (logical scalar).

nCores	Gives the number of threads, which will be executed by the parallel-package package (integer scalar). Defaults to no parallel processing.
saveOnDisk	Should all models be saved on hard disk instead of in the workspace? (Logical scalar) If the data sets are big and the KDSN is deep, then one model will occupy lots of workspace. Sometimes it is prohibitive to store the complete ensemble into workspace. In this approach every ensemble model is saved in an temporary Folder and the names of the files are saved. Later in prediction only one model is stored in workspace and stored until predictions are made. Only supported for single thread execution.
dirNameDisk	Only relevant if saveOnDisk=TRUE. Gives the path, where the fitted models will be saved.

Details

Note that this function is still experimental. It is important that the estimated KDSN has a custom set seed (argument "estKDSN") to ensure reproducibility. Otherwise the model is refitted using the default seed values seq(0, (Number of Levels-1), 1) are used for random Fourier transformation.

The disadvantage on using argument saveOnDisk=TRUE is, that saving and compressing the objects takes more time instead of using them together into workspace.

Value

Model of class "KDSNensemble" or "KDSNensembleDisk" if saveOnDisk=TRUE. For reproduceability all seeds of the random iterations are stored as attributes.

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

Examples

```
####################################
# Example with binary outcome

# Generate covariate matrix
sampleSize <- 100
X <- matrix(0, nrow=100, ncol=5)
for(j in 1:5) {
  set.seed (j)
  X [, j] <- rnorm(sampleSize)
}

# Generate bernoulli response
rowSumsX <- rowSums(X)
logistVal <- exp(rowSumsX) / (1 + exp(rowSumsX))
set.seed(-1)
y <- sapply(1:100, function(x) rbinom(n=1, size=1, prob=logistVal[x]))

# Fit kernel deep stacking network with three levels
# Initial seed should be supplied in fitted model!
```

```
fitKDSN1 <- fitKDSN(y=y, X=X, levels=3, Dim=c(20, 10, 5),
             sigma=c(0.5, 1, 2), lambdaRel=c(1, 0.1, 0.01),
             alpha=rep(0, 3), info=TRUE, seedW=c(0, 1:2))

# Fit 10 ensembles
fitKDSNens <- fitEnsembleKDSN (estKDSN=fitKDSN1, y=y, X=X, ensembleSize=10,
                 seedBag=1:10, seedW1=101:110, seedW2=-(101:110),
                 seedDrop1=3489:(3489+9), seedDrop2=-(3489:(3489+9)), info=TRUE)

# Generate new test data
sampleSize <- 100
Xtest <- matrix(0, nrow=100, ncol=5)
for(j in 1:5) {
  set.seed (j+50)
  Xtest [, j] <- rnorm(sampleSize)
}
rowSumsXtest <- rowSums(Xtest)
logistVal <- exp(rowSumsXtest) / (1 + exp(rowSumsXtest))
set.seed(-1)
ytest <- sapply(1:100, function(x) rbinom(n=1, size=1, prob=logistVal[x]))

# Evaluate on test data with auc
library(pROC)
preds <- predict(fitKDSN1, Xtest)
auc1 <- auc(response=ytest, predictor=c(preds))
predsMat <- predict(fitKDSNens, Xtest)
preds <- rowMeans(predsMat)
auc2 <- auc(response=ytest, predictor=c(preds))
auc1 < auc2 # TRUE
# The ensemble predictions give a better test auc than the original model
```

`fitKDSN` *Fit kernel deep stacking network with random Fourier transformations*

Description

Estimates the kernel deep stacking network with identity link function (Gaussian) for the response in each level. Sparsity may be included with the addition of dropout (experimental), variable pre- and internal selection (experimental) as well as L1 penalty of the cofficients.

Usage

```
fitKDSN(y, X, levels=1, Dim=rep(round(sqrt(dim(X) [1]) / 2), levels),
             sigma=rep(1, levels), lambdaRel=rep(0.5, levels),
          alpha=rep(0, levels), info=FALSE, seedW=NULL, standX=TRUE, standY=FALSE,
          varSelect=rep(FALSE, levels), varRanking=NULL, varExFreq=rep(NA, levels),
             dropHidden=rep(FALSE, levels),
             dropHiddenProb=rep(NA, levels),
             seedDrop=NULL, baggingInd=NULL, randSubsetInd=NULL,
             maxItOpt=10000)
```

Arguments

y	Response variable as numeric vector.
X	Design matrix. All factors must be encoded, e.g. dummy coding.
levels	Number of levels of the kernel deep stacking network (integer scalar).
Dim	Dimension of the random Fourier transformations in each level (integer vector). The first entry corresponds to the first level, the second entry to the second level and so on.
sigma	Variance of the Gaussian kernel of each level. The higher the variance, on average the more different transformed observations will be generated.
lambdaRel	Relative ridge regularization parameter of each level. The relative ridge regularization parameter is limited to the interval [0, 1] and 0 corresponds to no shrinkage and 1 stands for maximum penalty to shrink all coefficients except the intercept to zero. Default is set in the middle between those extremes.
alpha	Weight parameter between lasso and ridge penalty (numeric vector) of each level. Default=0 corresponds to ridge penalty and 1 equals lasso.
info	Determines if additional infos of the level computation are displayed during training (logical value). Default is FALSE.
seedW	Random seed for drawing Fourier transformation weights of the multivariate normal distribution (integer vector). Each entry in the vector corresponds to one level.
standX	Should the design matrix be standardized by median and median absolute deviation? Default is TRUE.
standY	Should the response be standardized by median and median absolute deviation? Default is FALSE.
varSelect	Should unimportant variables be excluded in each level? Default is that all available variables are used. (logical scalar)
varRanking	Defines a variable ranking in increasing order. The first variable is least important and the last is the most important. (integer vector)
varExFreq	Gives the relative frequency of variables to drop within the interval [0, 1]. Latent variables are always included in the model.
dropHidden	Should dropout be applied on the random Fourier transformation? Each entry corresponds to the one level. Default is without dropout in all levels (logical vector).
dropHiddenProb	Probability of inclusion of cells in random Fourier transformation. Each entry corresponds to the probability of one level.
seedDrop	Specifies the seed of the random dropouts in the calculation of random Fourier transformation per level. Default is random (integer vector).
baggingInd	Gives the indices of the bootstrap samples in list format. Each entry of the list (integer vector) corresponds to one level.
randSubsetInd	Gives the indices of the random variable subsets in list format. Each entry of the list (integer vector) corresponds to one level.
maxItOpt	Gives the maximum number of iterations in the optimization procedure of the elastic net regression. For details see glmnet.

Details

Note that the implementation of drop-out and variable selection is still experimental. \ \ A kernel deep stacking network is an approximation of artificial neural networks with multiple number of levels. In each level there is a random Fourier transformation of the data, based on Gaussian kernels, which can be represented as an infinite series. The random Fourier transform converges exponentially fast to the Gaussian kernel matrix by increasing dimension. Then a kernel ridge regression is applied given the transformed data. The predictions of the all last levels are included as covariates in the next level.

Value

- Output: A list with objects:
  - rftX: List of random Fourier transformation matrices for each level
  - linWeights: List of estimated parameter vectors of the ridge regression for each level
- Input: A list with all specified input parameter, e. g. number of levels, dimension of each level, ridge penalty, etc.

Note

It is recommended to standardize the design matrix before training. Otherwise the resulting values of the Fourier transform can be either all very small or very high, which results in poor prediction performance.

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

References

Po-Seng Huang and Li Deng and Mark Hasegawa-Johnson and Xiaodong He, (2013), *Random Features for kernel deep convex network*, Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)

D.S. Broomhead and David Lowe, (1988), *Radial Basis Functions, Multi-Variable Functional Interpolation and Adaptive Networks*, Controller HMSO, London

Jerome Friedman and Trevor Hastie and Rob Tibshirani, (2008), *Regularization Paths for Generalized Linear Models via Coordinate Descent*, Department of Statistics, Stanford University

See Also

predict.KDSN, randomFourierTrans, robustStandard, glmnet

Examples

```
###################################
# Example with binary outcome

# Generate covariate matrix
sampleSize <- 100
```

```
X <- matrix(0, nrow=100, ncol=5)
for(j in 1:5) {
  set.seed (j)
  X [, j] <- rnorm(sampleSize)
}

# Generate Bernoulli response
rowSumsX <- rowSums(X)
logistVal <- exp(rowSumsX) / (1 + exp(rowSumsX))
set.seed(-1)
y <- sapply(1:100, function(x) rbinom(n=1, size=1, prob=logistVal[x]))

# Fit kernel deep stacking network with three levels
fitKDSN1 <- fitKDSN(y=y, X=X, levels=3, Dim=c(20, 10, 5),
             sigma=c(6, 8, 10), lambdaRel=c(1, 0.1, 0.01),
             alpha=rep(0, 3), info=TRUE,
             seedW=c(1882335773, -1640543072, -931660653))

# Generate new test data
sampleSize <- 100
Xtest <- matrix(0, nrow=100, ncol=5)
for(j in 1:5) {
  set.seed (j+50)
  Xtest [, j] <- rnorm(sampleSize)
}
rowSumsXtest <- rowSums(Xtest)
logistVal <- exp(rowSumsXtest) / (1 + exp(rowSumsXtest))
set.seed(-1)
ytest <- sapply(1:100, function(x) rbinom(n=1, size=1, prob=logistVal[x]))

# Evaluate on test data with auc
library(pROC)
preds <- predict(fitKDSN1, Xtest)
auc(response=ytest, predictor=c(preds))

####################################
# Example with continuous outcome

# Generate covariate matrix
sampleSize <- 100
X <- matrix(0, nrow=100, ncol=5)
for(j in 1:5) {
  set.seed (j)
  X [, j] <- rnorm(sampleSize)
}

# Generate Gaussian random variable conditional on the covariates
linPred <- 1*X[,1] + 2*X[,2] + 0.5*X[,3] + 0.5
set.seed(-1)
y <- sapply(1:100, function(x) rnorm(n=1, mean=linPred[x], sd=2))

# Fit kernel deep stacking network with five levels
fitKDSN1 <- fitKDSN(y=y, X=X, levels=5, Dim=c(40, 20, 10, 7, 5),
```

```
              sigma=c(0.125, 0.25, 0.5, 1, 2), lambdaRel=c(1, 0.5, 0.1, 0.01, 0.001),
              alpha=rep(0, 5), info=TRUE,
              seedW=c(-584973296, -251589341, -35931358, 313178052, -1322344272))

# Generate new data
sampleSize <- 100
Xtest <- matrix(0, nrow=100, ncol=5)
for(j in 1:5) {
  set.seed (j+10)
  Xtest [, j] <- rnorm(sampleSize)
}
linPred <- 1*Xtest[,1] + 2*Xtest[,2] + 0.5*Xtest[,3] + 0.5
set.seed(-10)
ytest <- sapply(1:100, function(x) rnorm(n=1, mean=linPred[x], sd=2))

# Predictions on new data and compute root mean squared error
preds <- predict(obj=fitKDSN1, newx=Xtest)
RMSE <- sqrt(mean((ytest-c(preds))^2))
RMSE
```

`fourierTransPredict` *Prediction based on random Fourier transformation*

Description

Based on prior given weights, the estimated Fourier transformation is applied to new data.

Usage

```
fourierTransPredict(newx, rW)
```

Arguments

`newx`	New data design matrix.
`rW`	Prior drawn random weight matrix.

Value

Numeric transformed data matrix with dimension `2*Dim x n`.

Note

This function is not intended to be called directly by the user. Should only be used by experienced users, who want to customize the model. It is called in the estimation process of the kernel deep stacking network, e.g. `fitKDSN`.

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

References

Po-Seng Huang and Li Deng and Mark Hasegawa-Johnson and Xiaodong He, (2013), *Random Features for kernel deep convex network*, Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)

See Also

randomFourierTrans

Examples

```
# Generate data matrix
set.seed(50)
X <- matrix(rnorm(100*3), ncol=3)

# Apply a random Fourier transformation of higher dimension
rft <- randomFourierTrans(X=X, Dim=3, sigma=1, seedW=0)

# New data matrix
set.seed(100)
Xnew <- matrix(rnorm(100*3), ncol=3)

# Apply same Fourier transformation on new data
newZ <- fourierTransPredict(newx=Xnew, rW=rft$rW)
head(newZ)
```

---

gDerivMu *Derivative of the link function evaluated at the expected values*

Description

Evaluates the first derivative of the link function, given an exponential family distribution, at the expected values. The expected values are estimated with a generalized linear model assuming a Gaussian distribution.

Usage

```
gDerivMu(mu)
```

Arguments

mu	Fitted values of the model (numeric vector)

Value

Numeric scalar with gives the generalized cross-validation score. The kernel deep stacking network used to calculate the score is available as attribute.

Note

This function is not intended to be called directly by the user. Should only be used by experienced users, who want to customize the model.

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

References

Simon N. Wood, (2006), *Generalized Additive Models: An Introduction with R*, Taylor \& Francis Group LLC

See Also

calcTrA, calcWdiag, varMu

---

getEigenValuesRcpp *Calculates the eigenvalues of a matrix*

Description

This is an computational efficient help function to calculate the GCV loss of the KDSN. It is written in C and is faster than the regular R version.

Usage

```
getEigenValuesRcpp(M)
```

Arguments

M	Numeric Matrix.

Value

Numeric vector eigenvalues in increasing order.

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

See Also

eigen

Examples

```
# Define simple binary matrix
M <- cbind(c(1,-1), c(-1,1))

# Calculate eigen values with standard function
eigenVal <- eigen(M, only.values=TRUE)$values

# Calculate eigen values with Rcpp version
eigenValRcpp <- rev(getEigenValuesRcpp(M))

# Check equality
all.equal(eigenVal, eigenValRcpp) # TRUE

# Analyse run time of both variants
if(tryCatch(require("microbenchmark", quietly=TRUE, warn.conflicts=FALSE,
character.only = TRUE), warning=function(w) FALSE, error=function(e) FALSE)) {
  microbenchmark(Standard=eigen(M, only.values=TRUE), Rcpp=getEigenValuesRcpp(M))
}
# -> Rcpp function is on average about 30 times faster than the standard R function
```

`kernDeepStackNet_crossprodRcpp`

Calculates the cross product of a matrix

Description

This is an computational efficient help function to calculate the GCV loss of the KDSN. Calculates the matrix product A^T A. It is written in C and faster than the regular R version.

Usage

```
kernDeepStackNet_crossprodRcpp(A)
```

Arguments

`A`	Numeric Matrix. Integer matrices are not allowed.

Value

List with the numeric matrix A^T A as element.

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

See Also

`crossprod`

Examples

```
A <- matrix(seq(-1, 1, length.out=9), nrow=3, ncol=3)
crossprodRcpp(A)[[1]]
all.equal(crossprodRcpp(A)[[1]], crossprod(A))
```

`kernDeepStackNet_getEigenValuesRcpp`
Calculates the eigenvalues of a matrix

Description

This is an computational efficient help function to calculate the GCV loss of the KDSN. It is written in C and is faster than the regular R version.

Usage

```
kernDeepStackNet_getEigenValuesRcpp(M)
```

Arguments

`M`	Numeric Matrix.

Value

Numeric vector eigenvalues in increasing order.

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

See Also

`eigen`

Examples

```
# Define simple binary matrix
M <- cbind(c(1,-1), c(-1,1))

# Calculate eigen values with standard function
eigenVal <- eigen(M, only.values=TRUE)$values

# Calculate eigen values with Rcpp version
eigenValRcpp <- rev(getEigenValuesRcpp(M))

# Check equality
all.equal(eigenVal, eigenValRcpp) # TRUE

# Analyse run time of both variants
```

```
if(tryCatch(require("microbenchmark", quietly=TRUE, warn.conflicts=FALSE,
character.only = TRUE), warning=function(w) FALSE, error=function(e) FALSE)) {
  microbenchmark(Standard=eigen(M, only.values=TRUE), Rcpp=getEigenValuesRcpp(M))
}
# -> Rcpp function is on average about 30 times faster than the standard R function
```

`lossApprox` *Kernel deep stacking network loss function*

Description

Computes the generalized cross-validation (gcv) score of a tuning parameter configuration given the data. The tuning parameters consists of dimension of the random Fourier transform (=D), the variance of the Gaussian kernels (=sigma) and the regularization parameter of the kernel ridge regression (=lambda). It is a wrapper around the function `lossGCV` and therefore only supports the generalized cross-validation score. First a kernel deep stacking network is fitted. Then the inputs are prepared to calculate the gcv score.

Usage

```
lossApprox(parOpt, y, X, levels, seedW=NULL,
           varSelect=rep(FALSE, levels), varRanking=NULL,
           alpha=rep(0, levels), dropHidden=rep(FALSE, levels),
           seedDrop=NULL, standX=TRUE, standY=FALSE,
           namesParOpt=rep(c("dim", "sigma", "lambdaRel"), levels),
           gammaPar=1)
```

Arguments

`parOpt`	The numeric parameter vector of the tuning parameters. The length of the vector is required to be divisible with three without rest. The order of the parameters is (D, sigma, lambda) in one level. Each additional level is added as new elements, e.g. with two levels (D_1, sigma_1, lambda_1, D_2, sigma_2, lambda_2), etc.
`y`	Numeric response vector of the outcome. Should be formated as a one column matrix.
`X`	Numeric design matrix of the covariates. All factors have to be prior encoded.
`levels`	Number of levels of the kernel deep stacking network (integer scalar).
`seedW`	Seeds for the random Fourier transformation (integer vector). Each value corresponds to the seed of one level. The length must equal the number of levels of the kernel deep stacking network.
`varSelect`	Should unimportant variables be excluded in each level? Default is that all available variables are used. (logical scalar)
`varRanking`	Defines a variable ranking in increasing order. The first variable is least important and the last is the most important. (integer vector)
`alpha`	Weight parameter between lasso and ridge penalty (numeric vector) of each level. Default=0 corresponds to ridge penalty and 1 equals lasso.

dropHidden	Should dropout be applied on the random Fourier transformation? Each entry corresponds to the one level. Default is without dropout (logical vector).
seedDrop	Specifies the seed of the random dropouts in the calculation of random Fourier transformation per level. Default is random (integer vector).
standX	Should the design matrix be standardized by median and median absolute deviation? Default is TRUE.
standY	Should the response be standardized by median and median absolute deviation? Default is FALSE.
namesParOpt	Gives the names of the argument parOpt (character vector). It is used to encode the parameters into the correct structure suitable in fitting the kernel deep stacking network.
gammaPar	Weighting parameter (numeric scalar), which specifies how the generalized cross-validation score is penalized by the effective degrees of freedom. Default value is 1.

Value

Numeric scalar with gives the generalized cross-validation score. The kernel deep stacking network used to calculate the score is available in the attributes.

Note

This function is not intended to be called directly by the user. Should only be used by experienced users, who want to customize the model.

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

References

Simon N. Wood, (2006), *Generalized Additive Models: An Introduction with R*, Taylor \& Francis Group LLC

See Also

lossCvKDSN, lossGCV, lossSharedCvKDSN, lossSharedTestKDSN

Examples

```
####################################
# Example with simple binary outcome

# Generate covariate matrix
sampleSize <- 100
X <- matrix(0, nrow=100, ncol=10)
for(j in 1:10) {
  set.seed (j)
  X [, j] <- rnorm(sampleSize)
```

```
}

# Generate response of binary problem with sum(X) > 0 -> 1 and 0 elsewhere
# with Gaussian noise
set.seed(-1)
error <- rnorm (100)
y <- ifelse((rowSums(X) + error) > 0, 1, 0)

# Calculate loss function with parameters (D=10, sigma=1, lambda=0)
# in one layer
calcLoss <- lossApprox (parOpt=c(10, 1, 0), y=y, X=X,
levels=1, seedW=0)
str(calcLoss)

# Calculate loss function with parameters
# (D=10, sigma=1, lambda=0.1, D=5, sigma=2, lambda=0.01) in two layers
calcLoss <- lossApprox (parOpt=c(10, 1, 0.1, 5, 2, 0.01),
y=y, X=X, levels=1, seedW=0)
str(calcLoss)
```

`lossCvKDSN` *Kernel deep stacking network loss function based on cross-validation*

Description

Computes the average root mean squared error by cross-validation. The model is fitted `length(cvIndex)` times and predicts the left out test cases. This procedure may be repeated to decrease the variance of the average predictive deviance. The repetitions are assumed to be included in the list `cvIndex`. The format can be conviniently created by using supplied functions in the package `caret`, e. g. `createFolds`.

Usage

```
lossCvKDSN(parOpt, y, X, levels, cvIndex, seedW=NULL, lossFunc=devStandard,
varSelect=rep(FALSE, levels), varRanking=NULL, alpha=rep(0, levels),
dropHidden=rep(FALSE, levels), seedDrop=NULL, standX=TRUE, standY=FALSE,
namesParOpt=rep(c("dim", "sigma", "lambdaRel"), levels))
```

Arguments

`parOpt`	The numeric parameter vector of the tuning parameters. The length of the vector is required to be divisible with three without rest. The order of the parameters is (D, sigma, lambda) in one level. Each additional level is added as new elements, e.g. with two levels (D_1, sigma_1, lambda_1, D_2, sigma_2, lambda_2), etc.
`y`	Numeric response vector of the outcome. Should be formated as a one column matrix.
`X`	Numeric design matrix of the covariates. All factors have to be prior encoded.
`levels`	Number of levels of the kernel deep stacking network (integer scalar).

cvIndex	List of training indices. Each list gives one drawn training set indices. See for example function createMultiFolds.
seedW	Seeds for the random Fourier transformation (integer vector). Each value corresponds to the seed of one level. The length must equal the number of levels of the kernel deep stacking network.
lossFunc	Specifies the used loss function for evaluation on the test observations. It must have arguments "preds" (predictions) and "ytest" (test observations). Default=devStandard is predictive deviance.
varSelect	Should unimportant variables be excluded in each level? Default is that all available variables are used. (logical scalar)
varRanking	Defines a variable ranking in increasing order. The first variable is least important and the last is the most important. (integer vector)
alpha	Weight parameter between lasso and ridge penalty (numeric vector) of each level. Default=0 corresponds to ridge penalty and 1 equals lasso.
dropHidden	Should dropout be applied on the random Fourier transformation? Each entry corresponds to the one level. Default is without dropout (logical vector).
seedDrop	Specifies the seed of the random dropouts in the calculation of random Fourier transformation per level. Default is random (integer vector).
standX	Should the design matrix be standardized by median and median absolute deviation? Default is TRUE.
standY	Should the response be standardized by median and median absolute deviation? Default is FALSE.
namesParOpt	Gives the names of the argument parOpt (character vector). It is used to encode the parameters into the correct structure suitable in fitting the kernel deep stacking network.

Details

namesParOpt: \ "select" corresponds to the relative frequency of dropping unimportant variables according to the randomized dependence coefficient. \ "dim" corresponds to a dimension parameter of the random Fourier transformation. \ "sigma" corresponds to the precision parameter of the gaussian distribution to generate random weights. \ "dropProb" corresponds to probability of dropping out cells in the calculated random Fourier transformation matrix. \ "lambdaRel" corresponds to the regularization parameter of kernel ridge regression.

Value

Root average predictive deviance of the model (numeric scalar). The fit is available as attribute.

Note

Should only be used by experienced users, who want to customize the model. It is called in the model selection process of the kernel deep stacking network, e. g. tuneMboLevelCvKDSN

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

References

Po-Seng Huang and Li Deng and Mark Hasegawa-Johnson and Xiaodong He, (2013), *Random Features for kernel deep convex network*, Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)

Simon N. Wood, (2006), *Generalized Additive Models: An Introduction with R*, Taylor \& Francis Group LLC

See Also

lossApprox, lossGCV, lossSharedCvKDSN, lossSharedTestKDSN

Examples

```
###################################
# Example with simple binary outcome

# Generate covariate matrix
sampleSize <- 100
X <- matrix(0, nrow=100, ncol=10)
for(j in 1:10) {
  set.seed (j)
  X [, j] <- rnorm(sampleSize)
}

# Generate response of binary problem with sum(X) > 0 -> 1 and 0 elsewhere
# with Gaussian noise
set.seed(-1)
error <- rnorm (100)
y <- ifelse((rowSums(X) + error) > 0, 1, 0)

# Draw random test sets
library(caret)
Indices <- createMultiFolds(y=y, k = 10, times = 5)

# Calculate loss function with parameters (D=10, sigma=1, lambda=0)
# in one level
calcLoss <- lossCvKDSN(parOpt=c(10, 1, 0), y=y, X=X,
cvIndex=Indices, seedW=0, levels=1)
c(calcLoss)

# Calculate loss function with parameters
# (D=10, sigma=1, lambda=0.1, D=5, sigma=2, lambda=0.01) in two levels
calcLoss <- lossCvKDSN(parOpt=c(10, 1, 0.1, 5, 2, 0.01),
y=y, X=X, cvIndex=Indices, seedW=1:2, levels=2)
c(calcLoss)
```

lossGCV *Generalized cross-validation loss*

Description

Calculates the generalized cross-validation score given number of observations, model deviance, effective degrees of freedom and gamma.

Usage

```
lossGCV(n, Dev, trA, gammaPar=1)
```

Arguments

n	Number of observations (integer scalar).
Dev	Deviance of the kernel ridge regression (numeric scalar).
trA	Effective degrees of freedom (numeric scalar).
gammaPar	Weighting parameter (numeric scalar), which specifies how the generalized cross-validation score is penalized by the effective degrees of freedom. Default value is 1.

Value

Generalized cross-validation loss (numeric scalar).

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

References

Simon N. Wood, (2006), *Generalized Additive Models: An Introduction with R*, Taylor \& Francis Group LLC

See Also

lossApprox, lossCvKDSN, lossSharedCvKDSN, lossSharedTestKDSN

Examples

```
# Simple example based on help pages of mgcv::gam
# GCV Score is the same as used in function mgcv::mgcv
library(mgcv)
dat <- gamSim(1, n=100, dist="normal", scale=2)
gamMod <- gam(y~s(x0)+s(x1)+s(x2)+s(x3),data=dat)
calcGCV <- lossGCV(n=100, Dev=deviance(gamMod), trA=sum(gamMod$edf), gammaPar=1)
all.equal(calcGCV, gamMod$gcv.ubre) # TRUE
```

`lossSharedCvKDSN` *Kernel deep stacking network loss function based on cross-validation and shared hyperparameters*

Description

Computes the average root mean squared error (RMSE) by cross-validation. The model is fitted `length(cvIndex)` times and predicts the left out test cases. This procedure may be repeated to decrease the variance of the RMSE. The repetitions are assumed to be included in the cvIndex list. The number of tuning parameters is independent of the specified level.

Usage

```
lossSharedCvKDSN(parOpt, y, X, cvIndex, seedW=NULL, lossFunc=devStandard,
varSelectShared=FALSE, varRanking=NULL, alphaShared=0, dropHiddenShared=FALSE,
seedDrop=NULL, standX=TRUE, standY=FALSE,
namesParOpt=c("levels", "dim", "sigma", "lambdaRel"))
```

Arguments

`parOpt`	The numeric parameter vector of the tuning parameters. The length of the vector is required to be divisible with three without rest. The order of the parameters is (D, sigma, lambda) in one level. Each additional level is added as new elements, e.g. with two levels (D_1, sigma_1, lambda_1, D_2, sigma_2, lambda_2), etc.
`y`	Numeric response vector of the outcome. Should be formated as a one column matrix.
`X`	Numeric design matrix of the covariates. All factors have to be prior encoded.
`cvIndex`	List of training indices. Each list gives one drawn training set indices. See for example function `createMultiFolds`.
`seedW`	Seeds for the random Fourier transformation (integer vector). Each value corresponds to the seed of one level. The length must equal the number of levels of the kernel deep stacking network.
`lossFunc`	Specifies the used loss function for evaluation on the test observations. It must have arguments "preds" (predictions) and "ytest" (test observations). Default=devStandard is predictive deviance.
`varSelectShared`	Should unimportant variables be excluded? Default is that all available variables are used. (logical scalar). This setting is equal over all levels.
`varRanking`	Defines a variable ranking in increasing order. The first variable is least important and the last is the most important. (integer vector)
`alphaShared`	Weight parameter between lasso and ridge penalty (numeric vector). Default=0 corresponds to ridge penalty and 1 equals lasso. This setting is equal across all levels.

`dropHiddenShared`
: Should dropout be applied on the random Fourier transformation? This setting is equal for all levels. Default is without dropout (logical vector).

`seedDrop`
: Specifies the seed of the random dropouts in the calculation of random Fourier transformation per level. Default is random (integer vector).

`standX`
: Should the design matrix be standardized by median and median absolute deviation? Default is TRUE.

`standY`
: Should the response be standardized by median and median absolute deviation? Default is FALSE.

`namesParOpt`
: Gives the names of the argument parOpt (character vector). It is used to encode the parameters into the correct structure suitable in fitting the kernel deep stacking network.

Details

`namesParOpt`: \ "level" corresponds to the number of levels of KDSN. \ "select" corresponds to the relative frequency of dropping unimportant variables according to the randomized dependence coefficient. \ "dim" corresponds to a dimension parameter of the random Fourier transformation. \ "sigma" corresponds to the precision parameter of the gaussian distribution to generate random weights. \ "dropProb" corresponds to probability of dropping out cells in the calculated random Fourier transformation matrix. \ "lambdaRel" corresponds to the regularization parameter of kernel ridge regression.

Value

Root average predictive deviance of the model (numeric scalar). The fit is available as attribute.

Note

Should only be used by experienced users, who want to customize the model. It is called in the model selection process of the kernel deep stacking network, e. g. `tuneMboLevelCvKDSN`

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

References

Po-Seng Huang and Li Deng and Mark Hasegawa-Johnson and Xiaodong He, (2013), *Random Features for kernel deep convex network*, Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)

Simon N. Wood, (2006), *Generalized Additive Models: An Introduction with R*, Taylor \& Francis Group LLC

See Also

`lossApprox`, `lossCvKDSN`, `lossGCV`, `lossSharedTestKDSN`

Examples

```
###################################
# Example with simple binary outcome

# Generate covariate matrix
sampleSize <- 100
X <- matrix(0, nrow=100, ncol=10)
for(j in 1:10) {
  set.seed (j)
  X [, j] <- rnorm(sampleSize)
}

# Generate response of binary problem with sum(X) > 0 -> 1 and 0 elsewhere
# with Gaussian noise
set.seed(-1)
error <- rnorm (100)
y <- ifelse((rowSums(X) + error) > 0, 1, 0)

# Draw random test sets
library(caret)
Indices <- createMultiFolds(y=y, k = 10, times = 5)

# Calculate loss function with parameters (D=10, sigma=1, lambda=0)
# in one level
calcLoss <- lossSharedCvKDSN(parOpt=c(1, 10, 1, 0), y=y, X=X,
cvIndex=Indices, seedW=0)
c(calcLoss)
```

`lossSharedTestKDSN` *Kernel deep stacking network loss function with test set and shared hyperparameters*

Description

Computes the RMSE on a specified test set. The model is fitted one time and predicts the response in the test set. Tuning parameters are independent of the level.

Usage

```
lossSharedTestKDSN(parOpt, y, X, yTest, Xtest, seedW=NULL, lossFunc=devStandard,
varSelectShared=FALSE, varRanking=NULL, alphaShared=0, dropHiddenShared=FALSE,
seedDrop=NULL, standX=TRUE, standY=FALSE,
namesParOpt=c("levels", "dim", "sigma", "lambdaRel"))
```

Arguments

`parOpt`	The numeric parameter vector of the tuning parameters. The length of the vector is required to be divisible with three without rest. The order of the parameters is (D, sigma, lambda) in one level. Each additional level is added as new elements, e.g. with two levels (D_1, sigma_1, lambda_1, D_2, sigma_2, lambda_2), etc.

y	Numeric response vector of the outcome. Should be formated as a one column matrix.
X	Numeric design matrix of the covariates. All factors have to be prior encoded.
yTest	Numeric response vector of the test set. Should be formated as a one column matrix.
Xtest	Numeric design matrix of the covariates of the test set. All factors have to be properly encoded, e. g. dummy encoding.
seedW	Seeds for the random Fourier transformation (integer vector). Each value corresponds to the seed of one level. The length must equal the number of levels of the kernel deep stacking network.
lossFunc	Specifies the used loss function for evaluation on the test observations. It must have arguments "preds" (predictions) and "ytest" (test observations). Default=devStandard is predictive deviance.
varSelectShared	Should unimportant variables be excluded? Default is that all available variables are used. (logical scalar). This setting is equal over all levels.
varRanking	Defines a variable ranking in increasing order. The first variable is least important and the last is the most important. (integer vector)
alphaShared	Weight parameter between lasso and ridge penalty (numeric vector). Default=0 corresponds to ridge penalty and 1 equals lasso. This setting is equal across all levels.
dropHiddenShared	Should dropout be applied on the random Fourier transformation? This setting is equal for all levels. Default is without dropout (logical vector).
seedDrop	Specifies the seed of the random dropouts in the calculation of random Fourier transformation per level. Default is random (integer vector).
standX	Should the design matrix be standardized by median and median absolute deviation? Default is TRUE.
standY	Should the response be standardized by median and median absolute deviation? Default is FALSE.
namesParOpt	Gives the names of the argument parOpt (character vector). It is used to encode the parameters into the correct structure suitable in fitting the kernel deep stacking network.

Details

namesParOpt: \ "level" corresponds to the number of levels of KDSN. \ "select" corresponds to the relative frequency of dropping unimportant variables according to the randomized dependence coefficient. \ "dim" corresponds to a dimension parameter of the random Fourier transformation. \ "sigma" corresponds to the precision parameter of the gaussian distribution to generate random weights. \ "dropProb" corresponds to probability of dropping out cells in the calculated random Fourier transformation matrix. \ "lambdaRel" corresponds to the regularization parameter of kernel ridge regression.

Value

Root mean squared error (numeric scalar). The fit is available as attribute.

Note

Should only be used by experienced users, who want to customize the model. It is called in the model selection process of the kernel deep stacking network, e. g. tuneMboLevelCvKDSN. The loss function can be adapted to specific data situations.

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

References

Po-Seng Huang and Li Deng and Mark Hasegawa-Johnson and Xiaodong He, (2013), *Random Features for kernel deep convex network*, Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)

Simon N. Wood, (2006), *Generalized Additive Models: An Introduction with R*, Taylor \& Francis Group LLC

See Also

lossApprox, lossCvKDSN, lossSharedCvKDSN, lossGCV

Examples

```
####################################
# Example with simple binary outcome

# Generate covariate matrix
sampleSize <- 100
X <- matrix(0, nrow=100, ncol=10)
for(j in 1:10) {
  set.seed (j)
  X [, j] <- rnorm(sampleSize)
}

# Generate response of binary problem with sum(X) > 0 -> 1 and 0 elsewhere
# with Gaussian noise
set.seed(-1)
error <- rnorm (100)
y <- ifelse((rowSums(X) + error) > 0, 1, 0)

# Draw a subset as test data
library(caret)
Indices <- createDataPartition(y=y, times = 1, p=0.8) [[1]]

# Calculate loss function with parameters (D=10, sigma=1, lambda=0)
# in one level
calcLoss <- lossSharedTestKDSN(parOpt=c(1, 10, 1, 0), y=y, X=X,
```

```
yTest=y[-Indices], Xtest=X[-Indices, , drop=FALSE], seedW=0)
c(calcLoss)
```

---

mbo1d *Efficient global optimization with iterative point proposals*

Description

Implements the efficient global optimization algorithm based on Kriging. New design points are generated by the expected improvement criterion or expected quantile improvement criterion. The optimizer of the performance criterion can be choosen as generalized simulated annealing or conditional one dimensional optimization.

Usage

```
mbo1d(model, fun, nsteps, lower, upper, parinit, isoInput,
maxRunsMult=1, repMult=1, tol_input=.Machine$double.eps^0.25, addInfo=TRUE,
envir=parent.frame(), EIopt="1Dmulti", GenSAmaxCall=100, timeAlloc="constant",
EItype="EI")
```

Arguments

model	Fitted Kriging model of the experimental design and evaluation measures. See km
fun	Loss function to be minimized.
nsteps	Number of points the efficient global optimizer should compute.
lower	Vector of lower bounds of the tuning parameters. First element is the lower bound of the first tuning parameter, second element the lower bound of the next tuning parameter etc.
upper	Vector of upper bounds of the tuning parameters. First element is the upper bound of the first tuning parameter, second element the upper bound of the next tuning parameter etc.
parinit	Starting value of the one dimensional optimization algorithm. See optimize1dMulti.
isoInput	Force the covariance structure of the km to have only one range parameter. For details see km.
maxRunsMult	Multiplies the base number of iterations of the one dimensional algorithm. Default number of iterations are the number of hyperparameters. Only applied, if the argument EIopt is set to "1Dmulti".
repMult	Multiplies the base number of random starting values of the one dimensional algorithm. Default number of iterations are the number of hyperparameters. Only applied, if the argument EIopt is set to "1Dmulti".
tol_input	Accuracy threshold of the one dimensional optimization algorithm. See optimize1dMulti.
addInfo	Should additional information be displayed during optimization? (logical value). Default is FALSE.

envir	Internal variable to store environments. Default is to look up the one higher level environment. Modification is unnecessary.
EIopt	Specifies which algorithm is used to optimize the expected improvement criterion. Two alternatives are available "1Dmulti" and "GenSA". The former uses the conditional 1D algorithm and the latter generalized, simulated annealing.
GenSAmaxCall	Maximum number of function calls per parameter to estimate in generalized, simulated annealing. Higher values result in more accurate estimates, but the optimization process is slowed.
timeAlloc	Specifies how the new noise variance is influenced by iteration progress. Default is to use "constant"" allocation. The other available option is to specify "zero", which corresponds to the original expected improvement criterion.
EItype	Defines the type of the improvement criterion. The default `EI` corresponds to the expected improvement. As an alternative `EQI` the expected quantile improvement is also possible.

Details

In each step iteration the performance criterion is minimized regarding the design input parameters. The loss function is evaluated at the proposed design point. The Kriging model is updated with the new point and refitted with nugget effect to ensure convergence. The starting value for the next step iteration is set to the parameter values with lowest function value.

Value

Retains the parameter values with the lowest function value. The function value (`funcVal`) and the MBO tuning process history (`MBOprogress`) are available as attributes.

Note

Function is supplied for model customization and intended for the experienced user. The more user friendly function uses this code as intermediate step.

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

References

Olivier Roustant and David Ginsbourger and Yves Deville, (2012), *DiceKriging, DiceOptim: Two R Packages for the Analysis of Computer Experiments by Kriging-Based Metamodeling and Optimization*, Journal of Statistical Software, Volume 51, Issue 1

Donald R. Jones and Matthias Schonlau and William J. Welch, (1998), *Efficient Global Optimization of Expensive Black-Box Functions*, Journal of Global Optimization 13: pages 455-492

Constantino Tsallis and Daniel A. Stariolo, (1996), Generalized Simulated Annealing, Elsevier Physica A: Statistical Mechanics and its Applications, Volume 233, Issues 1-2, 15, Pages 395-406

Victor Picheny, David Ginsbourger, Yann Richet, (2012), *Quantile-based optimization of Noisy Computer Experiments with Tunable Precision*, HAL-archives-ouvertes.fr, hal-00578550v3

See Also

optimize1dMulti, km, EI, mboAll

---

mboAll *Efficient global optimization inclusive meta model validation*

Description

Implements the efficient global optimization algorithm based on Kriging. It is a wrapper around mbo1d and additionally includes latin hypercube design generation and meta model validation of the Kriging model.

Usage

```
mboAll(loss_func, n_steps, initDesign, lower_bounds,
upper_bounds, x_start, isoInput=FALSE, addInfo=TRUE,
maxRunsMult=1, repMult=1, tol_input=.Machine$double.eps^0.25,
envir=parent.frame(), metaModelSelect=TRUE,
EIopt="1Dmulti", GenSAmaxCall=100, timeAlloc="constant",
EItype="EI")
```

Arguments

loss_func	Loss function to be minimized.
n_steps	Number of steps of the EGO algorithm.
initDesign	Number of initial design points to be evaluated. The higher the number, the more function evaluations of the loss function are required, but the approximation with Kriging will be more accurate.
lower_bounds	Vector of lower bounds of the tuning parameters. First element is the lower bound of the first tuning parameter, second element the lower bound of the next tuning parameter etc.
upper_bounds	Vector of upper bounds of the tuning parameters. First element is the upper bound of the first tuning parameter, second element the upper bound of the next tuning parameter etc.
x_start	Starting value of the one dimensional optimization algorithm. See optimize1dMulti.
isoInput	Force the covariance structure of the km to have only one range parameter. For details see km
addInfo	Should additional information be displayed during optimization? (logical value). Default is TRUE.
maxRunsMult	Multiplies the base number of iterations in the conditional optimization. Default is to use the number of hyperparameters. See optimize1dMulti.
repMult	Multiplies the base number of random starting values for helping to avoid local optima. Default is the number of hyperparameters. See optimize1dMulti.

`tol_input` Convergence criteria of each one dimensional sub-optimization. Higher values will be more accurate, but require much more function evaluations. Default is the fourth root of the machine double accuracy. See `optimize1dMulti`.

`envir` Internal variable to store environments. Default is to look up in a one level higher environment. Modification is unnecessary.

`metaModelSelect` Should the covariance kernel of the Kriging model be automatically selected? (logical scalar) Default is TRUE and corresponds to the 5/2 materin covariance structure.

`EIopt` Specifies which algorithm is used to optimize the expected improvement criterion. Two alternatives are available "1Dmulti" and "GenSA". The former uses the conditional 1D algorithm and the latter generalized, simulated annealing.

`GenSAmaxCall` Maximum number of function calls per parameter to estimate in generalized, simulated annealing. Higher values result in more accurate estimates, but the optimization process is slowed.

`timeAlloc` Specifies how the new noise variance is influenced by iteration progress. Default is to use "constant"" allocation. The other available option is to specify "zero", which corresponds to the original expected improvement criterion.

`EItype` Defines the type of the improvement criterion. The default `EI` corresponds to the expected improvement. As an alternative `EQI` the expected quantile improvement is also possible.

Details

In addition to the function `mbo1d` a latin hypercube design will be generated. The design is generated by maximizing the minimum distance between the design points (maximin criteria). For reference see `maximinLHS`. Then the complete initial design is evaluated with the loss function. The model validation of Kriging searches for the best covariance kernel structure between five alternative specifications (see `covTensorProduct-class`). The performance is evaluated with a Gaussian likelihood with leave one out estimated expectations and variances of the Kriging model.

Value

List with following components:

- par: Best found parameter values
- value: Function value at the best found parameters
- MBOprogress: MBO tuning process history

Note

Function is supplied for model customization and intended for the experienced user. The more user friendly function `tuneMboLevelCvKDSN` uses this code as intermediate step.

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

References

Michael Stein, (1987), *Large Sample Properties of Simulations Using Latin Hypercube Sampling*, Technometrics. 29, 143-151

Carl Edward Rasmussen and Christopher K. I. Williams, (2006), *Gaussian Processes for Machine Learning*, Massachusetts Institute of Technology

See Also

km, leaveOneOut.km, maximinLHS, tuneMboLevelCvKDSN, mbo1d

Examples

```
# Example with Branin function
library(globalOptTests)
tryBranin <- mboAll (loss_func=function (x) goTest(par=x, fnName="Branin",
checkDim = FALSE), n_steps=5, initDesign=15, lower_bounds=c(-5, 0),
upper_bounds=c(10, 15), x_start=c(5, -5))
abs(tryBranin$value-getGlobalOpt("Branin"))
```

---

optimize1dMulti *One dimensional optimization of multivariate loss functions*

Description

Applies a stepwise one dimensional algorithm conditional on the previous best parameter values. The one dimensional algorithm is a combination of golden search and linear interpolation.

Usage

```
optimize1dMulti(f_input, lower, upper, maxRuns=3, repetitions=5,
tol_input=.Machine$double.eps^0.25, x_0=NULL, addInfo=TRUE,
nCores=1, envir=parent.frame(), directUse=TRUE, OptTypePar="")
```

Arguments

f_input	General loss function with multiple inputs and one numeric outcome value. The function must have a x as first argument, e.g. f(x).
lower	Gives the lower bounds of the input parameters (numeric vector). Each variable corresponds to one element of the vector.
upper	Gives the upper bounds of the input parameters (numeric vector). Each variable corresponds to one element of the vector.
maxRuns	Maximal number of iterations in the conditional optimization. Default is three.
repetitions	Maximal number of random starting values to avoid local optima. Default is five.

tol_input	Convergence criteria of each one dimensional sub-optimization. Higher values will be more accurate, but require much more function evaluations. Default is the fourth root of the machine double accuracy.
x_0	Intitial parameter values. If not supplied, a random number is drawn between the given bounds.
addInfo	Should the optimization print additional information? (logical value) Default is TRUE. The information consists of printing, which parameters have been optimized, which iteration and repetition the optimization process just finished.
nCores	Specifies how many cores are used in tuning (integer scalar). Default=1 is serial processing.
envir	Internal variable to store environments. Default is to look up the next higher level environment. Modification is unnecessary.
directUse	Specifies if the optimization is carried out inside a higher level function (see) or directly (logical scalar). Only relevant in parallel mode initialization.
OptTypePar	Internal variable to store, which higher level function was called previously. Modification has no effect.

Details

First the supplied initional value x_0 is used. If not present a random intitialisation is used. Then each parameter is optimized, conditional on the other starting values. An iteration is finished, if all parameters have been optimized once. The procedure is repeated until convergence or maxRuns is reached. If repetitions is higher than 1, new starting values are generated and the process starts new. In the end the best parameter values according the lowest function value are returned.

Value

- List with following components:
  - minimum: Best parameter values
  - objective: Minimum of function evaluated at the best parameter values

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

References

R. Brent, (1973), *Algorithms for Minimization without Derivatives*, Englewood Cliffs N.J.: Prentice-Hall

See Also

tuneMboLevelCvKDSN, tuneMboLevelGcvKDSN, tuneMboSharedCvKDSN

Examples

```
library(globalOptTests)
# Two dimensional task
getDefaultBounds("AluffiPentini")
oneDres <- optimize1dMulti (f_input=function (x)
goTest (x, fnName="AluffiPentini", checkDim=TRUE),
maxRuns=3, repetitions=5, lower=rep(-12, 2), upper=rep(10, 2),
tol_input=.Machine$double.eps^0.25)
abs(oneDres$objective-getGlobalOpt("AluffiPentini"))

# Four dimensional optimization task
getDefaultBounds("CosMix4")
oneDres <- optimize1dMulti (f_input=function (x)
goTest (x, fnName="CosMix4", checkDim=TRUE),
maxRuns=3, repetitions=5, lower=rep(-2, 4), upper=rep(1, 4),
tol_input=.Machine$double.eps^0.25)
abs(oneDres$objective-getGlobalOpt("CosMix4"))

# Ten dimensional optimization task
getDefaultBounds("Rastrigin")
oneDres <- optimize1dMulti (f_input=function (x)
goTest (x, fnName="Rastrigin", checkDim=TRUE),
maxRuns=3, repetitions=5, lower=rep(-525, 10), upper=rep(512, 10),
tol_input=.Machine$double.eps^0.25)
abs(oneDres$objective-getGlobalOpt("Rastrigin"))

# Ten dimensional optimization task with higher accuracy
getDefaultBounds("Rastrigin")
oneDres <- optimize1dMulti (f_input=function (x)
goTest (x, fnName="Rastrigin", checkDim=TRUE),
maxRuns=3, repetitions=5, lower=rep(-525, 10), upper=rep(512, 10),
tol_input=.Machine$double.eps^0.5)
abs(oneDres$objective-getGlobalOpt("Rastrigin"))
```

`predict.KDSN` *Predict kernel deep stacking networks*

Description

Predicts new data with a given kernel deep stacking network. All levels are applied successively with fixed weights to reproduce results.

Usage

```
## S3 method for class 'KDSN'
predict(object, newx, ...)
```

Arguments

object	Object of class KDSN. This object is generated with the function fitKDSN.
newx	New data design matrix, for which predictions are needed.
...	Further arguments to predict function.

Details

The data is put through all specified layers of the kernel deep stacking network. The weights are not random, but fixed at the values generated by the fitting process. Examples are given in the help page of fitKDSN.

Value

Numeric vector of predicted values of each observation.

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

See Also

fitKDSN, fitEnsembleKDSN

---

predict.KDSNensemble *Predict kernel deep stacking networks ensembles (experimental)*

Description

Predicts new data with a given kernel deep stacking network ensemble. All levels are applied successively with fixed weights to reproduce results. Note that this function is still experimental.

Usage

```
## S3 method for class 'KDSNensemble'
predict(object, newx, ...)
```

Arguments

object	Object of class KDSNensemble. This object is generated with the function fitEnsembleKDSN.
newx	New data design matrix, for which predictions are needed. Variables must be in the same order, as the original training data.
...	Further arguments to predict function.

Details

The data is put through all specified layers of the kernel deep stacking network. The weights are not random, but fixed at the values generated by the fitting process. Examples are given in the help page of fitEnsembleKDSN.

Value

A prediction matrix will be returned. Each row corresponds to one observation and each column is another KDSN ensemble.

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

See Also

fitKDSN, fitEnsembleKDSN

---

predict.KDSNensembleDisk

Predict kernel deep stacking networks ensembles (experimental)

Description

Predicts new data with a given kernel deep stacking network ensembles. The model is not stored in workspace, but on disk in temporary folder, where it has been created. The temporary file folder should be available in the working directory. Note that this function is still experimental.

Usage

```
## S3 method for class 'KDSNensembleDisk'
predict(object, newx, ...)
```

Arguments

object	Object of class KDSNensemble. This object is generated with the function fitEnsembleKDSN.
newx	New data design matrix, for which predictions are needed. Variables must be in the same order, as the original training data.
...	Further arguments to predict function.

Details

The data is put through all specified layers of the kernel deep stacking network. The weights are not random, but fixed at the values generated by the fitting process. Examples are given in the help page of fitEnsembleKDSN.

Value

A prediction matrix will be returned. Each row corresponds to one observation and each column is another KDSN ensemble.

Note

Do not rename the temporary directory. Otherwise the model files will not be found.

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

See Also

fitKDSN, fitEnsembleKDSN

---

predLogProb *Predictive logarithmic probability of Kriging model*

Description

Calculates the predictive logarithmic probability of Kriging model given the model parameters (likelihood function). It is used to choose better meta models in model-based optimization using Kriging. The observation used for prediction is left out in the training data and then estimated.

Usage

```
predLogProb(predMean, predSigma, y, X)
```

Arguments

predMean	Predicted leave one out mean of the Kriging model of all observations (numeric vector).
predSigma	Predicted leave one out variance of the Kriging model of all observations.
y	Numeric response vector of the outcome. Should be formated as a one column matrix.
X	Numeric design matrix of the covariates. All factors have to be prior encoded.

Details

Gaussian processes are conditionaly normal distributed and therefore the normal likelihood is used. The leave out parameters can be efficiently computed without performing leave one out cross-validation.

Value

Numeric Value of log-likelihood with leave-one-out parameters.

Note

This function is not intended to be called directly by the user. Should only be used by experienced users, who want to customize the model. It is called in the model selection process of the kernel deep stacking network, e.g. tuneMboLevelCvKDSN.

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

References

Carl Edward Rasmussen and Christopher K. I. Williams, (2006), *Gaussian Processes for Machine Learning* Massachusetts Institute of Technology

Examples

```
library(DiceKriging)
# Generate design of experiments
design.fact <- expand.grid(x1=seq(0,1,length=4), x2=seq(0,1,length=4))
y <- apply(design.fact, 1, branin)

# Estimate Kriging model
km_fit <- km(design=design.fact, response=y, control=list(trace=FALSE),
nugget.estim=TRUE, iso=FALSE)

# Calculate leave one out parameters and performance measure
loo <- leaveOneOut.km(model=km_fit, type="UK", trend.reestim=TRUE)
predLogProbs <- predLogProb(predMean=loo$mean, predSigma=loo$sd^2, y=y, X=X)
```

---

randomFourierTrans *Random Fourier transformation*

Description

Calculates the random Fourier transformation using a Gaussian kernel, given the original data.

Usage

```
randomFourierTrans(X, Dim, sigma, seedW=NULL)
```

Arguments

X	Original data design matrix. All factors have to be encoded, e.g. dummy coding.
Dim	Specifies the dimension of the random Fourier transformation (integer scalar).
sigma	Variance of the Gaussian kernel (positive numeric scalar).
seedW	Random seed for drawing from the multivariate normal distribution (integer scalar).

Details

First a random weight matrix is drawn from the multivariate normal distribution. Then the Data is linear transformed. The linear transformed data is mapped nonlinear by applying cosinus and sinus functions. The matrix multiplaction t(Z) Z approximates the Gaussian radial basis function kernel matrix. The dimension of t(Z) Z is always n x n. The higher the dimension argument Dim, the more accurate the results.

Value

Numeric transformed data matrix with dimension 2*Dim x n.

Note

This function is not intended to be called directly by the user. Should only be used by experienced users, who want to customize the model. It is called in the estimation process of the kernel deep stacking network, e. g. `fitKDSN`.

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

References

Po-Seng Huang and Li Deng and Mark Hasegawa-Johnson and Xiaodong He, (2013), *Random Features for kernel deep convex network*, Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)

See Also

`fourierTransPredict`

Examples

```
# Generate data matrix
X <- data.frame(rnorm(100), rnorm(100), rnorm(100), rnorm(100), rnorm(100),
factor(sample(c("a", "b", "c", "d", "e"), 100, replace=TRUE)))
X <- model.matrix(object=~., data=X)
# Exclude intercept
X <- X[, -1]

# Apply a random Fourier transformation of lower dimension
rft <- randomFourierTrans(X=X, Dim=2, sigma=1, seedW=0)

# Transformed data
rft$Z

# Used weight matrix
rft$rW
```

rdcPart *Randomized dependence coefficient partial calculation*

Description

Calculates the randomized dependence coefficient based on interim results. It is a generalized dependence measure based on maximum correlation of random non-linear projections.

Usage

```
rdcPart(subsetX, xTrans, yTrans, s=1/6, f=sin, randX)
```

Arguments

subsetX	Subset of the covariate matrix as indices (integer vector).
xTrans	Transformed matrix to the [0, 1] scale (numeric matrix).
yTrans	Random, non-linear projection of the response (numeric vector).
s	Variance of the random weights. Default is 1/6.
f	Non-linear transformation function. Default is sin.
randX	Random weights (numeric vector).

Details

This function allows for more efficient calculation than the complete calculation by excluding repetitive calculations.

Value

Value of randomized dependence coefficient (numeric scalar).

Note

This function is a help function within variable selection. It is given for experts for model customization. It is recommended to use instead function rdcVarOrder.

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

References

David Lopez-Paz et. al, (2013), The randomized dependence coefficient, Proceedings of Advances in Neural Information Processing Systems 26 (NIPS)

See Also

rdcVarOrder, cancorRed

rdcSubset *Randomized dependence coefficients score on given subset*

Description

Variable pre selection scoring for KDSN. Estimates the RDC score for a subset of variables.

Usage

```
rdcSubset(binCode, x, y, k=20, s=1/6, f=sin, seedX=NULL, seedY=NULL,
rdcRep=1, trans0to1=TRUE)
```

Arguments

binCode	Specifies which set of variables of the covariates is used to explain the responses (binary vector). One to assiged inclusion and zero excludes variables.
x	Covariates data (numeric matrix).
y	Responses (numeric matrix).
k	Number of random features (integer scalar).
s	Variance of the random weights. Default is 1/6.
f	Non-linear transformation function. Default is sin.
seedX	Random number seed of normal distributed weights for covariates (integer scalar). Default is to randomly draw weights.
seedY	Random number seed of normal distributed weights for responses (integer scalar). Default is to randomly draw weights.
rdcRep	Gives the number of rdc repetitions. All repetitions are averaged per variable, to give more robust estimates. Default is to use one repetition.
trans0to1	Should the design matrix and response be transformed to the interval [0, 1]? (Logical). If the data is available in this for form, it can be evaluated much faster.

Details

Covariates are ranked according to their dependence with the response variable.

Value

RDC score (numeric scalar).

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

References

David Lopez-Paz and Philipp Hennig and Bernhard Schoelkopf, (2013), *The Randomized dependence coefficient*, Proceedings of Neural Information Processing Systems (NIPS) 26, Stateline Nevada USA, C.J.C. Burges and L. Bottou and M. Welling and Z. Ghahramani and K.Q. Weinberger (eds.)

See Also

rdcPart, cancorRed, rdcVarOrder, rdcVarSelSubset

Examples

```
############################
# Cubic noisy association

# Generate 10 covariates
library(mvtnorm)
set.seed(3489)
X <- rmvnorm(n=200, mean=rep(0, 10))

# Generate responses based on some covariates
set.seed(-239247)
y <- 0.5*X[, 1]^3 - 2*X[, 2]^2 + X[, 3] - 1 + rnorm(200)

# Score of true subset
scoreTrue <- rdcSubset(binCode=c(rep(1, 3), rep(0, 7)),
x=X, y=y, seedX=1:10, seedY=-(1:10), rdcRep=10)
scoreTrue

# Only unneccessary variables
scoreFalse <- rdcSubset(binCode=c(rep(0, 3), rep(1, 7)),
x=X, y=y, seedX=1:10, seedY=-(1:10), rdcRep=10)
scoreFalse

# One important two important variables and some non causal variables
scoreMix <- rdcSubset(binCode=c(1, 0, 1, rep(0, 3), rep(1, 4)),
x=X, y=y, seedX=1:10, seedY=-(1:10), rdcRep=10)
scoreMix
```

---

rdcVarOrder *Variable ordering using randomized dependence coefficients (experimental)*

Description

Variable selection for KDSN. All variables are univariately compared to the responses with the randomized dependence coefficient (RDC). The indices of the variables are ordered accordingly. Variables with low RDC may be discarded.

Usage

```
rdcVarOrder(x, y, k=20, s=1/6, f=sin, seedX=NULL,
          seedY=NULL, nCores=1, info=FALSE, cutoff=0, rdcRep=1)
```

Arguments

x	Covariates data (numeric matrix).
y	Responses (numeric matrix).
k	Number of random features (integer scalar).
s	Variance of the random weights. Default is 1/6.
f	Non-linear transformation function. Default is sin.
seedX	Random number seed of normal distributed weights for covariates (integer scalar). Default is to randomly draw weights.
seedY	Random number seed of normal distributed weights for responses (integer scalar). Default is to randomly draw weights.
nCores	Number of threads used. If greater than one, parallel processing using the parallel package is used.
info	Should additional infos on the progress of the function be given? (logical scalar) Default is not to give additional information.
cutoff	Gives the empirical cut-off value (numeric scalar). Variables below this threshold will be discarded. Default is to include all variables.
rdcRep	Gives the number of rdc repetitions. All repetitions are averaged per variable, to give more robust estimates. Default is to use one repetition.

Details

Covariates are ranked according to their dependence with the response variable. Note that this function is still experimental.

Value

Ordered indices of original covariates in increasing order of importance. The first variable is the least important.

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

References

David Lopez-Paz and Philipp Hennig and Bernhard Schoelkopf, (2013), *The Randomized dependence coefficient*, Proceedings of Neural Information Processing Systems (NIPS) 26, Stateline Nevada USA, C.J.C. Burges and L. Bottou and M. Welling and Z. Ghahramani and K.Q. Weinberger (eds.)

See Also

rdcPart, cancorRed

Examples

```
##############################
# Cubic noisy association

# Generate 10 covariates
library(mvtnorm)
set.seed(3489)
X <- rmvnorm(n=200, mean=rep(0, 10))

# Generate responses based on some covariates
set.seed(-239247)
y <- 0.5*X[, 1]^3 - 2*X[, 2]^2 + X[, 3] - 1 + rnorm(200)

# Variable selection with RDC
selectedInd <- rdcVarOrder(x=X, y=y, seedX=1, seedY=2, cutoff=0.7)
selectedInd
# -> If the numbers of variables should be reduced from 10 to 3,
# then all important variables are selected.

# With more repetitions and different random transformations
selectedInd <- rdcVarOrder(x=X, y=y, seedX=1:25, seedY=-(1:25), cutoff=0.7, rdcRep=25)
selectedInd
# -> Gives identical result as one repetition
```

---

rdcVarSelSubset *Variable selection based on RDC with genetic algorithm (experimental)*

Description

Selects important variables, which have high RDC scores. A genetic algorithm is used to search the discrete space. Note that this function is still experimental.

Usage

```
rdcVarSelSubset(x, y, k=20, s=1/6, f=sin, seedX=1:10, seedY=-c(1:10),
rdcRep=10, popSize=100, maxiter=100, nCores=1, addInfo=TRUE)
```

Arguments

x	Covariates data (numeric matrix).
y	Responses (numeric matrix).
k	Number of random features (integer scalar).
s	Variance of the random weights. Default is 1/6.

f	Non-linear transformation function. Default is sin.
seedX	Random number seed of normal distributed weights for covariates (integer scalar). Default is to randomly draw weights.
seedY	Random number seed of normal distributed weights for responses (integer scalar). Default is to randomly draw weights.
rdcRep	Gives the number of rdc repetitions. All repetitions are averaged per variable, to give more robust estimates. Default is to use one repetition.
popSize	Size of population of the genetic algorithm.
maxiter	Maximum number of generations to generate.
nCores	Number of threads used in parallelisation in ga. Default is no parallelisation.
addInfo	Should details of the optimization be printed? (logical scalar) Default TRUE enables default monitoring, see ga for further details.

Value

Indices of selected variables

Author(s)

Thomas Welchowski

References

David Lopez-Paz and Philipp Hennig and Bernhard Schoelkopf, (2013), *The Randomized dependence coefficient*, Proceedings of Neural Information Processing Systems (NIPS) 26, Stateline Nevada USA, C.J.C. Burges and L. Bottou and M. Welling and Z. Ghahramani and K.Q. Weinberger (eds.)

M. Wahde, (2008), Biological inspired methods: An introduction, WIT Press

See Also

rdcPart, cancorRed, rdcSubset, rdcVarOrder

Examples

```
# Generate 10 covariates
library(mvtnorm)
set.seed(3489)
X <- rmvnorm(n=200, mean=rep(0, 10))

# Generate responses based on some covariates
set.seed(-239247)
y <- 0.5*X[, 1]^3 - 2*X[, 2]^2 + X[, 3] - 1 + rnorm(200)

# Running variable selection
foundVar <- rdcVarSelSubset(x=X, y=y, seedX=1, seedY=-(1), rdcRep=1,
popSize=80, maxiter=5)
foundVar
```

robustStandard *Robust standardization*

Description

Scales the data matrix with the median and median absolute deviation.

Usage

```
robustStandard(X)
```

Arguments

X	Data design matrix.

Value

Numeric transformed data matrix with same dimension as original data.

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

References

Ricardo A. Maronna and R. Douglas Martin and Victor J. Yohai, (2006), *Robust Statistics: Theory and Methods*, John Wiley \& Sons, Ltd.

Examples

```
# Generate data matrix
set.seed(150)
X <- matrix(rnorm(100*3), ncol=3)

# Robust standardization
scaledX <- robustStandard(X=X)

# Median equals 0
all.equal(sapply(1:3, function(x) median(scaledX[,x])), rep(0, 3))

# MAD equals 1
all.equal(sapply(1:3, function(x) mad(scaledX[,x], constant=1)), rep(1, 3))
```

tuneMboLevelCvKDSN *Tuning of KDSN with efficient global optimization given level by cross-validation*

Description

Implements the efficient global optimization algorithm based on Kriging for kernel deep stacking networks (KDSN). This function uses cross-validation or a test set to calculate the desing grid and is more computationally expensive, but uses more accurate generalization error estimates. Additionally preselection of variable are available based on the randomized dependence coefficient (RDC).

Usage

```
tuneMboLevelCvKDSN(y, X, levels=1, alpha=rep(0, levels),
fineTuneIt=100, nStepMult=20, designMult=10,
dimMax=round(sqrt(dim(X)[1])/2), addInfo=TRUE,
maxRunsMult=1, repMult=1, tol_input=.Machine$double.eps^0.25,
cvIndex, lossFunc=devStandard, EIopt="1Dmulti", GenSAmaxCall=100,
varSelect=rep(FALSE, levels), rdcRep=1, dropHidden=rep(FALSE, levels),
standX=TRUE, standY=FALSE, timeAlloc="constant",
varPreSelect=FALSE, varPreSelpopSize=100, varPreSelMaxiter=100,
EItype="EQI")
```

Arguments

y	Response matrix with one column.
X	Design matrix. All factors must be already encoded.
levels	Maximum number of levels for the kernel deep stacking network (integer scalar).
alpha	Weight parameter between lasso and ridge penalty (numeric vector) of each level. Default=0 corresponds to ridge penalty and 1 equals lasso.
fineTuneIt	Number of drawn random weight matrices in fine tuning (integer scalar). If set to zero, no fine tuning is done.
nStepMult	Multiplier, which affects how many steps the EGO algorithm is run, depending on the number of parameters to estimate.
designMult	Multiplier, which affects how many initial design points are evaluated in the loss function, depending on the number of parameters to estimate.
dimMax	Maximal dimension of the random Fourier transformation. The effective number of parameters is dimMax*2. The default heuristic depends on the sample size.
addInfo	Should additional information be printed during estimation? Default is TRUE.
maxRunsMult	Multiplies the base number of iterations in the conditional one dimensional optimization. Default is to use the number of hyperparameters. See optimize1dMulti.
repMult	Multiplies the base number of random starting values in the conditional one dimensional optimization to avoid local optima. Default is to use the number of hyperparameters. See optimize1dMulti.

tol_input	Convergence criteria of each one dimensional sub-optimization. Higher values will be more accurate, but require much more function evaluations. Default is the fourth root of the machine double accuracy. See optimize1dMulti.
cvIndex	Index of cross-validation indices. The indices represent the training data. Must be supplied as list, the required format is identical to the output of the createFolds with argument returnTrain=TRUE.
lossFunc	Specifies how the loss on the test data should be evaluated. Defaults to predictive deviance devStandard.
EIopt	Specifies which algorithm is used to optimize the expected improvement criterion. Two alternatives are available "1Dmulti" and "GenSA". The former uses the conditional 1D algorithm and the latter generalized, simulated annealing.
GenSAmaxCall	Maximum number of function calls per parameter to estimate in generalized, simulated annealing. Higher values result in more accurate estimates, but the optimization process is slowed.
varSelect	Specifies, if variables should be preselected by using randomized, dependence coefficient. Default is no variable selection in all levels.
rdcRep	Number of repetitions for the randomized dependence coefficient rdcVarOrder.
dropHidden	Should dropout be applied on the random Fourier transformation? Each entry corresponds to the one level. Default is without dropout (logical vector).
standX	Should the design matrix be standardized by median and median absolute deviation? Default is TRUE.
standY	Should the response be standardized by median and median absolute deviation? Default is FALSE.
timeAlloc	Specifies how the new noise variance is influenced by iteration progress. Default is to use "constant"" allocation. The other available option is to specify "zero", which sets the future noise variance always to zero.
varPreSelect	Should variables be pre-selected using RDC and genetic algorithm? Default is no. May consume a lot of start up time.
varPreSelpopSize	Population size of the genetic algorithm (integer scalar).
varPreSelMaxiter	Maximum number of generations of the genetic algorithm (integer scalar).
EItype	Defines the type of the improvement criterion. The default EQI corresponds to the expected quantile improvement. As an alternative EI expected improvement is also possible.

Value

Gives the best tuned kernel deep stacking network of class k_DSN_rft given a specific level (see fitKDSN).

Note

This function is supplied to optimize units KDSN with many levels. Very complex data generating mechanisms may be more efficiently represented with large number of levels. The computation time

increases progressive the more levels are added. Higher values of tol_input, fineTuneIt, maxRuns, repetitions may increase performance. The computation time of the tuning algorithm increases mostly with higher values of dimMax. \\ The variable selection `varSelect=TRUE` calculates the rdc of all variables. Then the variables above a pre-specified threshold are choosen as important and all others are discarded. In the estimation process, in each level the least important variable according to rdc, is removed from the design matrix. Only observed variables may be removed. Latent variables are always used in the model.

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

References

David Lopez-Paz and Philipp Hennig and Bernhard Schoelkopf, (2013), *The Randomized Dependence Coefficient*, Max Planck Institute for Intelligent Systems, Germany

Victor Picheny, David Ginsbourger, Yann Richet, (2012), *Quantile-based optimization of Noisy Computer Experiments with Tunable Precision*, HAL-archives-ouvertes.fr, hal-00578550v3

See Also

`km`, `leaveOneOut.km`, `maximinLHS`, `mboAll`, `mbo1d`

Examples

```
# Generate small sample of 20 observations of a binary classification task
# Due to keeping the example as fast as possible, the parameters of the tuning
# algorithm are set for low accuracy. Higher values of tol_input, fineTuneIt,
# maxRuns, repetitions will increase performance considerably.
library(pROC)

# Generate design matrix
sampleSize <- 20
X <- matrix(0, nrow=sampleSize, ncol=5)
for(j in 1:5) {
  set.seed (j)
  X [, j] <- rnorm(sampleSize)
}

# Generate response of binary problem with sum(X) > 0 -> 1 and 0 elsewhere
set.seed(-1)
error <- rnorm (sampleSize)
y <- ifelse((rowSums(X) + error) > 0, 1, 0)

# Generate test data
Xtest <- matrix(, nrow=sampleSize, ncol=5)
for(j in 1:5) {
  set.seed (j*2+1)
  Xtest [, j] <- rnorm(sampleSize)
}
```

```
# Generate test response
set.seed(-10)
error <- rnorm (sampleSize)
ytest <- ifelse((rowSums(Xtest) + error) > 0, 1, 0)

# Draw cv training indices
library(caret)
cvTrainInd <- createFolds(y=y, k = 2, list = TRUE, returnTrain = TRUE)

# Define loss function
defLossFunc <- function(preds, ytest) {-c(auc(response=ytest, predictor=c(preds)))}

# Tune kernel deep stacking network by auc on test data
## Not run:
tuned_KDSN_EGO_level <- tuneMboLevelCvKDSN (y=y, X=X,
levels=2, fineTuneIt=10, nStepMult=2, designMult=5,
cvIndex=cvTrainInd, lossFunc=defLossFunc)
preds <- predict(tuned_KDSN_EGO_level, newx=Xtest)
library(pROC)
auc(response=ytest, predictor=c(preds))

## End(Not run)
```

---

tuneMboLevelGcvKDSN *Tuning of KDSN with efficient global optimization given level by cross-validation*

Description

Implements the efficient global optimization algorithm based on Kriging for kernel deep stacking networks (KDSN). This function uses generalized cross validation approximation of out of sample error without using external data. Preselection of variables is based on the randomized dependence coefficient (RDC).

Usage

```
tuneMboLevelGcvKDSN(y, X, levels=1, alpha=rep(0, levels), fineTuneIt=100,
                    nStepMult=20, designMult=10,
                    dimMax=round(sqrt(dim(X)[1])/2), addInfo=TRUE,
                    maxRunsMult=1, repMult=1,
                    tol_input=.Machine$double.eps^0.25,
                    EIopt="1Dmulti", GenSAmaxCall=100,
                    varSelect=rep(FALSE, levels), rdcRep=1,
                    dropHidden=rep(FALSE, levels),
                    standX=TRUE, standY=FALSE, timeAlloc="constant",
                  varPreSelect=FALSE, varPreSelpopSize=100, varPreSelMaxiter=100,
                    EItype="EQI")
```

Arguments

y	Response matrix with one column.
X	Design matrix. All factors must be already encoded.
levels	Maximum number of levels for the kernel deep stacking network (integer scalar).
alpha	Weight parameter between lasso and ridge penalty (numeric vector) of each level. Default=0 corresponds to ridge penalty and 1 equals lasso.
fineTuneIt	Number of drawn random weight matrices in fine tuning (integer scalar). If set to zero, no fine tuning is done.
nStepMult	Multiplier, which affects how many steps the EGO algorithm is run, depending on the number of parameters to estimate.
designMult	Multiplier, which affects how many initial design points are evaluated in the loss function, depending on the number of parameters to estimate.
dimMax	Maximal dimension of the random Fourier transformation. The effective number of parameters is dimMax*2. The default heuristic depends on the sample size.
addInfo	Should additional information be printed during estimation? Default is TRUE.
maxRunsMult	Multiplies the base number of iterations in the conditional one dimensional optimization. Default is to use the number of hyperparameters. See optimize1dMulti.
repMult	Multiplies the base number of random starting values in the conditional one dimensional optimization to avoid local optima. Default is to use the number of hyperparameters. See optimize1dMulti.
tol_input	Convergence criteria of each one dimensional sub-optimization. Higher values will be more accurate, but require much more function evaluations. Default is the fourth root of the machine double accuracy. See optimize1dMulti.
EIopt	Specifies which algorithm is used to optimize the expected improvement criterion. Two alternatives are available "1Dmulti" and "GenSA". The former uses the conditional 1D algorithm and the latter generalized, simulated annealing.
GenSAmaxCall	Maximum number of function calls per parameter to estimate in generalized, simulated annealing. Higher values result in more accurate estimates, but the optimization process is slowed.
varSelect	Specifies, if variables should be preselected by using randomized, dependence coefficient. Default is no variable selection in all levels.
rdcRep	Number of repetitions for the randomized dependence coefficient rdcVarOrder.
dropHidden	Should dropout be applied on the random Fourier transformation? Each entry corresponds to the one level. Default is without dropout (logical vector).
standX	Should the design matrix be standardized by median and median absolute deviation? Default is TRUE.
standY	Should the response be standardized by median and median absolute deviation? Default is FALSE.
timeAlloc	Specifies how the new noise variance is influenced by iteration progress. Default is to use "constant"" allocation. The other available option is to specify "zero", which sets the future noise variance always to zero.

varPreSelect Should variables be pre-selected using RDC and genetic algorithm? Default is no. May consume a lot of start up time.

varPreSelpopSize
Population size of the genetic algorithm (integer scalar).

varPreSelMaxiter
Maximum number of generations of the genetic algorithm (integer scalar).

EItype Defines the type of the improvement criterion. The default EQI corresponds to the expected quantile improvement. As an alternative EI expected improvement is also possible.

Value

Gives the best tuned kernel deep stacking network of class k_DSN_rft given a specific level (see fitKDSN).

Note

This function is supplied to optimize units KDSN with many levels. Complex data generating mechanisms may be more efficiently represented with large number of levels. The computation time increases progressive the more levels are added. Higher values of tol_input, fineTuneIt, maxRuns, repetitions may increase performance. The computation time of the tuning algorithm increases mostly with higher values of dimMax. \\ The variable selection varSelect=TRUE calculates the rdc of all variables. Then the variables above a pre-specified threshold are choosen as important and all others are discarded. In the estimation process, in each level the least important variable according to rdc, is removed from the design matrix. Only observed variables may be removed. Latent variables are always used in the model.

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

References

David Lopez-Paz and Philipp Hennig and Bernhard Schoelkopf, (2013), *The Randomized Dependence Coefficient*, Max Planck Institute for Intelligent Systems, Germany

Victor Picheny, David Ginsbourger, Yann Richet, (2012), *Quantile-based optimization of Noisy Computer Experiments with Tunable Precision*, HAL-archives-ouvertes.fr, hal-00578550v3

See Also

km, leaveOneOut.km, maximinLHS, mboAll, mbo1d

Examples

```
# Generate small sample of 20 observations of a binary classification task
# Due to keeping the example as fast as possible, the parameters of the tuning
# algorithm are set for low accuracy. Higher values of tol_input, fineTuneIt,
# maxRuns, repetitions will increase performance considerably.
library(pROC)
```

```
# Generate design matrix
sampleSize <- 20
X <- matrix(0, nrow=sampleSize, ncol=5)
for(j in 1:5) {
  set.seed (j)
  X [, j] <- rnorm(sampleSize)
}

# Generate response of binary problem with sum(X) > 0 -> 1 and 0 elsewhere
set.seed(-1)
error <- rnorm (sampleSize)
y <- ifelse((rowSums(X) + error) > 0, 1, 0)

# Generate test data
Xtest <- matrix(, nrow=sampleSize, ncol=5)
for(j in 1:5) {
  set.seed (j*2+1)
  Xtest [, j] <- rnorm(sampleSize)
}

# Generate test response
set.seed(-10)
error <- rnorm (sampleSize)
ytest <- ifelse((rowSums(Xtest) + error) > 0, 1, 0)

# Tune kernel deep stacking network by auc on test data
## Not run:
tuned_KDSN_EGO_level <- tuneMboLevelGcvKDSN (y=y, X=X,
levels=2, fineTuneIt=10, nStepMult=2, designMult=5)
preds <- predict(tuned_KDSN_EGO_level, newx=Xtest)
library(pROC)
auc(response=ytest, predictor=c(preds))

## End(Not run)
```

`tuneMboSharedCvKDSN` *Tuning of KDSN with efficient global optimization given level by cross-validation and shared hyperparameters*

Description

Implements the efficient global optimization algorithm based on Kriging for kernel deep stacking networks (KDSN). This function uses cross-validation or a test set to calculate the desing grid and is more computationally expensive, but uses more accurate generalization error estimates. In contrast `tuneMboLevelCvKDSN` the number of tuning parameters is independent of the number of levels. In this function the number of levels is included in the tuning process. Additionally preselection of variables (still experimental) is available based on the randomized dependence coefficient (RDC).

Usage

```
tuneMboSharedCvKDSN(y, X, alphaShared=0, fineTuneIt=100, nStepMult=20, designMult=10,
dimMax=round(sqrt(dim(X)[1])/2), addInfo=TRUE, maxRunsMult=1, repMult=1,
tol_input=.Machine$double.eps^0.25, cvIndex=NULL, lossFunc=devStandard, EIopt="1Dmulti",
GenSAmaxCall=100, varSelectShared=FALSE, rdcRep=1, dropHiddenShared=FALSE,
standX=TRUE, standY=FALSE, timeAlloc="constant", varPreSelect=FALSE,
varPreSelpopSize=100, varPreSelMaxiter=100, maxLevels=10,
useCV=TRUE, yTest=NULL, Xtest=NULL, EItype="EQI")
```

Arguments

y	Response matrix with one column.
X	Design matrix. All factors must be already encoded.
alphaShared	Weight parameter between lasso and ridge penalty (numeric vector) of each level. Default=0 corresponds to ridge penalty and 1 equals lasso.
fineTuneIt	Number of drawn random weight matrices in fine tuning (integer scalar). If set to zero, no fine tuning is done.
nStepMult	Multiplier, which affects how many steps the EGO algorithm is run, depending on the number of parameters to estimate.
designMult	Multiplier, which affects how many initial design points are evaluated in the loss function, depending on the number of parameters to estimate.
dimMax	Maximal dimension of the random Fourier transformation. The effective number of parameters is dimMax*2. The default heuristic depends on the sample size.
addInfo	Should additional information be printed during estimation? Default is TRUE.
maxRunsMult	Multiplies the base number of iterations in the conditional one dimensional optimization. Default is to use the number of hyperparameters. See optimize1dMulti.
repMult	Multiplies the base number of random starting values in the conditional one dimensional optimization to avoid local optima. Default is to use the number of hyperparameters. See optimize1dMulti.
tol_input	Convergence criteria of each one dimensional sub-optimization. Higher values will be more accurate, but require much more function evaluations. Default is the fourth root of the machine double accuracy. See optimize1dMulti.
cvIndex	Index of cross-validation indices. The indices represent the training data. Must be supplied as list, the required format is identical to the output of the createFolds with argument returnTrain=TRUE.
lossFunc	Specifies how the loss on the test data should be evaluated. Defaults to predictive deviance devStandard.
EIopt	Specifies which algorithm is used to optimize the expected improvement criterion. Two alternatives are available "1Dmulti" and "GenSA". The former uses the conditional 1D algorithm and the latter generalized, simulated annealing.
GenSAmaxCall	Maximum number of function calls per parameter to estimate in generalized, simulated annealing. Higher values result in more accurate estimates, but the optimization process is slowed.

varSelectShared
: Specifies, if variables should be preselected by using randomized, dependence coefficient. Default is no variable selection. This setting is shared across all levels.

rdcRep
: Number of repetitions for the randomized dependence coefficient rdcVarOrder.

dropHiddenShared
: Should dropout be applied on the random Fourier transformation? Each entry corresponds to the one level. Default is without dropout (logical vector).

standX
: Should the design matrix be standardized by median and median absolute deviation? Default is TRUE.

standY
: Should the response be standardized by median and median absolute deviation? Default is FALSE.

timeAlloc
: Specifies how the new noise variance is influenced by iteration progress. Default is to use "constant"" allocation. The other available option is to specify "zero", which sets the future noise variance always to zero.

varPreSelect
: Should variables be pre-selected using RDC and genetic algorithm? Default is no. May consume a lot of start up time.

varPreSelpopSize
: Population size of the genetic algorithm (integer scalar).

varPreSelMaxiter
: Maximum number of generations of the genetic algorithm (integer scalar).

maxLevels
: Maximum number of levels possible to tune. Lower number speeds up tuning, but is less flexible. Default is to use 10 levels.

useCV
: Should the loss be calculated from cross validation samples cvIndex or by using one subset? (logical) Default uses cross validation.

yTest
: External response of the test data. Only used, if useCV is TRUE.

Xtest
: External covariates of the test data. Only used, if useCV is TRUE.

EItype
: Defines the type of the improvement criterion. The default EQI corresponds to the expected quantile improvement. As an alternative EI expected improvement is also possible.

Details

The tuning parameters are specified the same across all levels. This leads to more parsimonious models and faster tuning. For additional flexibility the number of levels is not given in advance and also considered in tuning. Note that this function is still experimental.

Value

Gives the best tuned kernel deep stacking network of class k_DSN_rft given a specific level (see fitKDSN).

Note

This function is supplied to optimize units KDSN with many levels. Very complex data generating mechanisms may be more efficiently represented with large number of levels. The computation time increases progressive the more levels are added. Higher values of tol_input, fineTuneIt, maxRuns, repetitions may increase performance. The computation time of the tuning algorithm increases mostly with higher values of dimMax.

The variable selection `varSelect=TRUE` calculates the rdc of all variables (experimental). Then the variables above a pre-specified threshold are choosen as important and all others are discarded. In the estimation process, in each level the least important variable according to rdc, is removed from the design matrix. Only observed variables may be removed. Latent variables are always used in the model.

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

References

David Lopez-Paz and Philipp Hennig and Bernhard Schoelkopf, (2013), *The Randomized Dependence Coefficient*, Max Planck Institute for Intelligent Systems, Germany

Victor Picheny, David Ginsbourger, Yann Richet, (2012), *Quantile-based optimization of Noisy Computer Experiments with Tunable Precision*, HAL-archives-ouvertes.fr, hal-00578550v3

See Also

`km`, `leaveOneOut.km`, `maximinLHS`, `mboAll`, `mbo1d`

Examples

```
# Generate small sample of 20 observations of a binary classification task
# Due to keeping the example as fast as possible, the parameters of the tuning
# algorithm are set for low accuracy. Higher values of tol_input, fineTuneIt,
# maxRuns, repetitions will increase performance considerably.
library(pROC)

# Generate design matrix
sampleSize <- 20
X <- matrix(0, nrow=sampleSize, ncol=5)
for(j in 1:5) {
  set.seed (j)
  X [, j] <- rnorm(sampleSize)
}

# Generate response of binary problem with sum(X) > 0 -> 1 and 0 elsewhere
set.seed(-1)
error <- rnorm (sampleSize)
y <- ifelse((rowSums(X) + error) > 0, 1, 0)

# Generate test data
Xtest <- matrix(, nrow=sampleSize, ncol=5)
```

```
for(j in 1:5) {
  set.seed (j*2+1)
  Xtest [, j] <- rnorm(sampleSize)
}

# Generate test response
set.seed(-10)
error <- rnorm (sampleSize)
ytest <- ifelse((rowSums(Xtest) + error) > 0, 1, 0)

# Draw cv training indices
library(caret)
cvTrainInd <- createFolds(y=y, k = 2, list = TRUE, returnTrain = TRUE)

# Define loss function
defLossFunc <- function(preds, ytest) {-c(auc(response=ytest, predictor=c(preds)))}

# Tune kernel deep stacking network by auc on test data
## Not run:
tuned_KDSN_EGO_level <- tuneMboSharedCvKDSN (y=y, X=X,
fineTuneIt=10, nStepMult=2, designMult=3,
cvIndex=cvTrainInd, lossFunc=defLossFunc)
preds <- predict(tuned_KDSN_EGO_level, newx=Xtest)
library(pROC)
auc(response=ytest, predictor=c(preds))

## End(Not run)
```

---

tuneMboSharedSubsetKDSN *Tuning subsets of KDSN with efficient global optimization and shared hyperparameters (experimental)*

Description

Estimates ensembles of (sparse) KDSNs with shared tuning parameters (still experimental). The number of levels is incorporated into the tuning process. Each part of the ensemble is tuned using model-based optimization as implemented in function tuneMboSharedCvKDSN.

Usage

```
tuneMboSharedSubsetKDSN (noSubsets=2, noSubsetRep=1, subSeed=NULL, y, X, alphaShared=1,
                         nStepMult=20, designMult=10,
                         lossFunc=devStandard, GenSAmaxCall=100,
                         varSelectShared=TRUE, dropHiddenShared=TRUE,
                         standX=TRUE, standY=FALSE, timeAlloc="constant",
                         varPreSelect=TRUE, varPreSelpopSize=100,
                         varPreSelMaxiter=100, maxLevels=10, nCores=1,
                         addInfo=1, saveOnDisk=FALSE,
```

```
                    dirNameDisk=paste(tempdir(), "/ensembleModel", sep=""),
                     useAllSub=TRUE, trainPartition=0.5, noPartion=1,
                     EItype="EQI")
```

Arguments

noSubsets	Number of training data splits (integer scalar). Training data will be randomly split into disjoint subsets. Each subset will be used as training set and the remaining data as test set.
noSubsetRep	Number of independent random subset draws (integer scalar).
subSeed	Random seed of subset generation Random. Default is NULL.
y	Response matrix with one column.
X	Design matrix. All factors must be already encoded.
alphaShared	Weight parameter between lasso and ridge penalty (numeric vector) of each level. Default=0 corresponds to ridge penalty and 1 equals lasso.
nStepMult	Multiplier, which affects how many steps the EGO algorithm is run, depending on the number of parameters to estimate.
designMult	Multiplier, which affects how many initial design points are evaluated in the loss function, depending on the number of parameters to estimate.
lossFunc	Specifies how the loss on the test data should be evaluated. Defaults to predictive deviance devStandard.
GenSAmaxCall	Maximum number of function calls per parameter to estimate in generalized, simulated annealing. Higher values result in more accurate estimates, but the optimization process is slowed.
varSelectShared	Specifies, if variables should be preselected by using randomized, dependence coefficient. Default is no variable selection. This setting is shared across all levels.
dropHiddenShared	Should dropout be applied on the random Fourier transformation? Each entry corresponds to the one level. Default is without dropout (logical vector).
standX	Should the design matrix be standardized by median and median absolute deviation? Default is TRUE.
standY	Should the response be standardized by median and median absolute deviation? Default is FALSE.
timeAlloc	Specifies how the new noise variance is influenced by iteration progress. Default is to use "constant"" allocation. The other available option is to specify "zero", which sets the future noise variance always to zero.
varPreSelect	Should variables be pre-selected using RDC and genetic algorithm? Default is no. May consume a lot of start up time.
varPreSelpopSize	Population size of the genetic algorithm (integer scalar).
varPreSelMaxiter	Maximum number of generations of the genetic algorithm (integer scalar).

maxLevels	Maximum number of levels possible to tune. Lower number speeds up tuning, but is less flexible. Default is to use 10 levels. (integer scalar)
nCores	Number of threads to use in implizit calculation based on parallel-package package. Default is serial processing. (integer scalar)
addInfo	Should the progress during tuning be printed? (integer scalar) Value zero means no additional printing. The next step is addInfo=1 overall progress will be printed. addInfo=2 additionally prints progress of inner tuning procedures. Default is printing of overall progress.
saveOnDisk	Should the estimated models be saved on disk instead of memory? (logical scalar). If the data is high dimensional, the workspace memory may not be sufficient to store all of them at once.
dirNameDisk	Gives the directory and file name of the tuned SKDSN models. The Number at the end of the filename represents the subset. Default is the temporary R-directory.
useAllSub	The data is split into noSubsets equal parts. If useAllSub==TRUE the tuning process is applied to all training parts. Otherwise the complete data set is split into one training and one validation set. The proportion is given by the argument trainPartition.
trainPartition	Gives the proportion of the complete data set used in training. Only applied, if useAllSub==FALSE.
noPartion	Gives the number of randomly drawn partitions of the complete data set. Only applied, if useAllSub==FALSE.
EItype	Defines the type of the improvement criterion. The default EQI corresponds to the expected quantile improvement. As an alternative EI expected improvement is also possible.

Details

The tuning parameters fixed across all levels. This leads to more parsimonious models and faster tuning. For additional flexibility the number of levels is not given in advance and also considered in tuning.

Value

Gives the best tuned kernel deep stacking network of class KDSNensemble or KDSNensembleDisk. For further details see predict.KDSNensemble, predict.KDSNensembleDisk and fitKDSN.

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

References

David Lopez-Paz and Philipp Hennig and Bernhard Schoelkopf, (2013), *The Randomized Dependence Coefficient*, Max Planck Institute for Intelligent Systems, Germany

Victor Picheny, David Ginsbourger, Yann Richet, (2012), *Quantile-based optimization of Noisy Computer Experiments with Tunable Precision*, HAL-archives-ouvertes.fr, hal-00578550v3

See Also

fitKDSN, tuneMboSharedCvKDSN, km, leaveOneOut.km, maximinLHS, mboAll, mbo1d

Examples

```
# Generate small sample of 20 observations of a binary classification task
# Due to keeping the example as fast as possible, the parameters of the tuning
# algorithm are set for low accuracy. Higher values of tol_input, fineTuneIt,
# maxRuns, repetitions will increase performance considerably.
library(pROC)

# Generate design matrix
sampleSize <- 20
X <- matrix(0, nrow=sampleSize, ncol=5)
for(j in 1:5) {
  set.seed (j)
  X [, j] <- rnorm(sampleSize)
}

# Generate response of binary problem with sum(X) > 0 -> 1 and 0 elsewhere
set.seed(-1)
error <- rnorm (sampleSize)
y <- ifelse((rowSums(X) + error) > 0, 1, 0)

# Generate test data
Xtest <- matrix(, nrow=sampleSize, ncol=5)
for(j in 1:5) {
  set.seed (j*2+1)
  Xtest [, j] <- rnorm(sampleSize)
}

# Generate test response
set.seed(-10)
error <- rnorm (sampleSize)
ytest <- ifelse((rowSums(Xtest) + error) > 0, 1, 0)

# Draw cv training indices
library(caret)
cvTrainInd <- createFolds(y=y, k = 2, list = TRUE, returnTrain = TRUE)

# Define loss function
defLossFunc <- function(preds, ytest) {-c(auc(response=ytest, predictor=c(preds)))}

# Tune kernel deep stacking network by auc on test data
## Not run:
tuned_KDSN_EGO_level <- tuneMboSharedCvKDSN (y=y, X=X,
nStepMult=2, designMult=3, lossFunc=defLossFunc,
GenSAmaxCall=10, varPreSelpopSize=10, varPreSelMaxiter=10)
preds <- predict(tuned_KDSN_EGO_level, newx=Xtest)
library(pROC)
auc(response=ytest, predictor=c(preds))
```

```
## End(Not run)
```

---

varMu *Variance function evaluated at expected value*

Description

Evaluates the variance function at the expected value. Only implemented for Gaussian distribution.

Usage

```
varMu(mu)
```

Arguments

mu	Expected values conditional on the design matrix. These are the fitted values of the model (numeric vector).

Value

Variance function of exponential family evaluated at the fitted values (numeric vector).

Note

This function is not intended to be called directly by the user. Should only be used by experienced users, who want to customize the model.

Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

References

Simon N. Wood, (2006), *Generalized Additive Models: An Introduction with R*, Taylor \& Francis Group LLC

See Also

calcTrA, calcWdiag, gDerivMu

Index