Package 'flam'

May 8, 2026

Type Package

Title Fits Piecewise Constant Models with Data-Adaptive Knots

Version 3.2

Date 2018-04-05

Author Ashley Petersen

Maintainer Ashley Petersen <ashleyjpete@gmail.com>

Description Implements the fused lasso additive model as proposed in Petersen, A., Witten, D., and Simon, N. (2016). Fused Lasso Additive Model. Journal of Computational and Graphical Statistics, 25(4): 1005-1025.

License GPL (>= 2)

Imports Rcpp (>= 0.11.6), MASS, graphics, grDevices, stats

LinkingTo Rcpp

RoxygenNote 6.0.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2018-04-06 03:26:45 UTC

Contents

`flam-package` *Fit the Fused Lasso Additive Model*

Description

This package is called flam for "fused lasso additive model", which is proposed in Petersen, A., Witten, D., and Simon, N. (2014). Fused Lasso Additive Model. arXiv preprint arXiv:1409.5391. The fused lasso additive model provides an approach to fit an additive model in which each component is estimated to be piecewise constant with a small number of adaptively-chosen knots.

The main functions are: (1) `flam` and (2) `flamCV`. The first function `flam` fits the fused lasso additive model for a range of tuning parameters and provides the fits for all of these tuning parameters. The second function `flamCV` considers a range of tuning parameters and provides the fits, but also returns the optimal tuning parameters, as chosen using K-fold cross-validation.

Details

Package:	flam
Type:	Package
Version:	3.0
Date:	2015-07-26
License:	GPL (>= 2)

The package includes the following functions: `flam`, `flamCV`, `plot.flam`, `plot.flamCV`, `plot.flamSparsity`, `predict.flam`, `summary.flam`, `summary.flamCV`, `flamDOF`, and `sim.data`.

Author(s)

Ashley Petersen

Maintainer: Ashley Petersen <ajpete@uw.edu>

References

Petersen, A., Witten, D., and Simon, N. (2014). Fused Lasso Additive Model. arXiv preprint arXiv:1409.5391.

Examples

```
#general example illustrating all functions
#see specific function help pages for details of using each function

#generate data
set.seed(1)
data <- sim.data(n = 50, scenario = 1, zerof = 10, noise = 1)

#fit model for a range of tuning parameters, lambda and alpha
```

```
#lambda sequence is chosen automatically if not specified
flam.out <- flam(x = data$x, y = data$y, alpha.seq = c(0.8, 0.9, 1))
#or fit model and select lambda using 2-fold cross-validation
#note: use larger 'n.fold' (e.g., 10) in practice
flamCV.out <- flamCV(x = data$x, y = data$y, alpha = 1, n.fold = 2)

#summarize all of the fits (in particular, the percent sparsity achieved)
#note: percent sparsity is the percentage of features estimated
#to have no relationship with outcome
summary(flam.out)
#the percent sparsity over the range of tuning parameters can also
#be displayed in a plot
plot(summary(flam.out))
#or just summarize a single fit
#we'll examine the fit with an index of 25. that is, lambda and alpha of
flam.out$all.lambda[25]; flam.out$all.alpha[25]
summary(flam.out, index = 25)
#lastly, we can summarize the fit chosen using cross-validation
summary(flamCV.out$flam.out, index = flamCV.out$index.cv)
#the lambda chosen by cross-validation is also available using
flamCV.out$lambda.cv

#plot the estimated relationships between predictors and outcome
#do this for a specific fit
plot(flam.out, index = 25)
#or for the fit chosen using cross-validation
plot(flamCV.out$flam.out, index = flamCV.out$index.cv)
#by default, up to 10 non-sparse features with the largest L2 norms will
#be plotted, see '?plot.flam' for other optional function arguments

#this data is simulated, so with a little more work, we can compare the
#true generating functions to the estimated function fits
#we do this for the truly non-zero functions (i.e., the first four predictors)
#generate data from same model but larger n, just used to plot true functions
temp.data <- sim.data(n = 500, scenario = 1, zerof = 10, noise = 1)
col.vec = c("dodgerblue1","orange","seagreen1","hotpink")
theta.hat = flamCV.out$flam.out$theta.hat.list[[flamCV.out$index.cv]]
par(mfrow=c(2,2))
for (i in 1:4) {
rgb.num = col2rgb(col.vec[i])
col=rgb(rgb.num[1], rgb.num[2], rgb.num[3], 100, max=256)
plot(1,type="n",xlim=c(-2.5,2.5),ylim=c(-2,2),xlab=paste("x",i,sep=""),
ylab=paste("f",i,"(x",i,")",sep=""),main="")
points(sort(temp.data$x[,i]), temp.data$theta[order(temp.data$x[,i]),i],type="l",lwd=3)
points(sort(data$x[,i]), theta.hat[order(data$x[,i]),i],col=col,type="l",lwd=3)
}

#we can make predictions for a covariate matrix with new observations
#choose the alpha and lambda of interest
alpha <- flamCV.out$alpha; lambda <- flamCV.out$lambda.cv
#new.x with 20 observations and the same number of features as flam.out$x
new.data <- sim.data(n = 20, scenario = 1, zerof = 10, noise = 1)
new.x <- new.data$x
```

```
#these will give the same predictions:
yhat1 <- predict(flam.out, new.x = new.x, lambda = lambda, alpha = alpha)
yhat2 <- predict(flamCV.out$flam.out, new.x = new.x, lambda = lambda, alpha = alpha)

#we can summarize the cross-validation function call
summary(flamCV.out)
#and also plot the cross-validation error
plot(flamCV.out)
#or calculate degrees of freedom for the model chosen using cross-validation
flamDOF(object = flamCV.out$flam.out, index = flamCV.out$index.cv)
#or for any fit of a 'flam' object
flamDOF(object = flam.out, index = 25)
```

`flam` *Fit the Fused Lasso Additive Model for a Sequence of Tuning Parameters*

Description

Fit an additive model where each component is estimated to piecewise constant with a small number of adaptively-chosen knots. The model is fit for a sequence of tuning parameters. In particular, this function implements the "fused lasso additive model", as proposed in Petersen, A., Witten, D., and Simon, N. (2014). Fused Lasso Additive Model. arXiv preprint arXiv:1409.5391.

Usage

```
flam(x, y, lambda.min.ratio = 0.01, n.lambda = 50, lambda.seq = NULL,
alpha.seq = 1, family = "gaussian", method = "BCD", tolerance = 10e-6)
```

Arguments

`x`	n x p covariate matrix. May have p > n.
`y`	n-vector containing the outcomes for the n observations in `x`.
`lambda.min.ratio`	smallest value for `lambda.seq`, as a fraction of the maximum lambda value, which is the data-derived smallest value for which all estimated functions are zero. The default is 0.01.
`n.lambda`	the number of lambda values to consider - the default is 50.
`lambda.seq`	a user-supplied sequence of positive lambda values to consider. The typical usage is to calculate `lambda.seq` using `lambda.min.ratio` and `n.lambda`, but providing `lambda.seq` overrides this. If provided, `lambda.seq` should be a decreasing sequence of values, since `flam` relies on warm starts for speed. Thus fitting the model for a whole sequence of lambda values is often faster than fitting for a single lambda value. Note that the model is fit for all combinations of `alpha.seq` and `lambda.seq`, so all values of `lambda.seq` provided should be unique.

alpha.seq	the value(s) of alpha to consider - default is 1. Values must be in [0,1] with values near 0 prioritizing sparsity of functions and values near 1 prioritizing limiting the number of knots. Empirical evidence suggests using alpha of 1 when p < n and alpha of 0.75 when p > n. Note that the model is fit for all combinations of alpha.seq and lambda.seq, so all values of alpha.seq provided should be unique.
family	specifies the loss function to use. Currently supports squared error loss (default; family="gaussian") and logistic loss (family="binomial").
method	specifies the optimization algorithm to use. Options are block-coordinate descent (default; method="BCD"), generalized gradient descent (method="GGD"), or generalized gradient descent with backtracking (method="GGD.backtrack"). This argument is ignored if family="binomial".
tolerance	specifies the convergence criterion for the objective (default is 10e-6).

Value

An object with S3 class "flam".

all.alpha	vector of alpha values considered. This will be m times longer than the user-specified alpha.seq where m is the length of the user-specified lambda.seq.
all.lambda	vector of lambda values considered. This will be q times longer than the user-specified lambda.seq where q is the length of the user-specified alpha.seq.
theta.hat.list	list of estimated theta matrices of dimension n x p. Note that the predicted values y.hat.mat[i,] = g(beta0.hat.vec[i] + rowSums(theta.hat.list[[i]])) where g is the link function (identity if family="gaussian" and expit if family="binomial").
f.hat.list	list of estimated function matrices of dimension n x p. Note that f.hat.list[[i]] is theta.hat.list[[i]] with the elements of each column ordered in terms of increasing x[,i].
beta0.hat.vec	vector of estimated intercepts with beta0.hat.vec[i] being the intercept for the model with tuning parameters all.alpha[i] and all.lambda[i].
y.hat.mat	matrix with y.hat.mat[i,] containing fitted y values for the tuning parameters all.alpha[i] and all.lambda[i].
non.sparse.list	list with non.sparse.list[[i]] containing the indices for the predictors with non-sparse fits for the tuning parameters all.alpha[i] and all.lambda[i].
num.non.sparse	vector with num.non.sparse[i] indicating the number of non-sparse predictor fits for the tuning parameters all.alpha[i] and all.lambda[i].
y	as specified by user.
x	as specified by user.
family	as specified by user (or default).
method	as specified by user (or default).
tolerance	as specified by user (or default).
call	the matched call.

Author(s)

Ashley Petersen

References

Petersen, A., Witten, D., and Simon, N. (2014). Fused Lasso Additive Model. arXiv preprint arXiv:1409.5391.

See Also

predict.flam, plot.flam, summary.flam

Examples

```
#See ?'flam-package' for a full example of how to use this package

#generate data
set.seed(1)
data <- sim.data(n = 50, scenario = 1, zerof = 10, noise = 1)

#fit model for a range of lambda chosen by default and alpha's of 0.75 and 1
flam.out <- flam(x = data$x, y = data$y, alpha.seq = c(0.75, 1))
#or specify desired lambda sequence (often equally spaced on log scale)
#should be a decreasing sequence of several values for computational speed
user.lambda.seq <- exp(seq(log(50), log(1), len=40))
flam.out2 <- flam(x = data$x, y = data$y, lambda.seq = user.lambda.seq)

## Not run:
#alternatively, generate data for logistic FLAM model
data2 <- sim.data(n = 50, scenario = 1, zerof = 10, family = "binomial")
#fit the FLAM model using logistic loss
flam.logistic.out <- flam(x = data2$x, y = data2$y, family = "binomial")

## End(Not run)

#'flam' returns an object of the class 'flam'
#see ?'flam-package' for an example using S3 methods for 'flam' objects
```

---

flamCV *Fit the Fused Lasso Additive Model and Do Tuning Parameter Selection using K-Fold Cross-Validation*

Description

Fit an additive model where each component is estimated to piecewise constant with a small number of adaptively-chosen knots. Tuning parameter selection is done using K-fold cross-validation. In particular, this function implements the "fused lasso additive model", as proposed in Petersen, A., Witten, D., and Simon, N. (2014). Fused Lasso Additive Model. arXiv preprint arXiv:1409.5391.

Usage

```
flamCV(x, y, lambda.min.ratio = 0.01, n.lambda = 50, lambda.seq = NULL,
alpha = 1, family = "gaussian", method = "BCD", fold = NULL,
n.fold = NULL, seed = NULL, within1SE = T, tolerance = 10e-6)
```

Arguments

x	n x p covariate matrix. May have p > n.
y	n-vector containing the outcomes for the n observations in x.
lambda.min.ratio	smallest value for lambda.seq, as a fraction of the maximum lambda value, which is the data-derived smallest value for which all estimated functions are zero. The default is 0.01.
n.lambda	the number of lambda values to consider - the default is 50.
lambda.seq	a user-supplied sequence of positive lambda values to consider. The typical usage is to calculate lambda.seq using lambda.min.ratio and n.lambda, but providing lambda.seq overrides this. If provided, lambda.seq should be a decreasing sequence of values, since flamCV relies on warm starts for speed. Thus fitting the model for a whole sequence of lambda values is often faster than fitting for a single lambda value.
alpha	the value of the tuning parameter alpha to consider - default is 1. Value must be in [0,1] with values near 0 prioritizing sparsity of functions and values near 1 prioritizing limiting the number of knots. Empirical evidence suggests using alpha of 1 when p < n and alpha of 0.75 when p > n.
family	specifies the loss function to use. Currently supports squared error loss (default; family="gaussian") and logistic loss (family="binomial").
method	specifies the optimization algorithm to use. Options are block-coordinate descent (default; method="BCD"), generalized gradient descent (method="GGD"), or generalized gradient descent with backtracking (method="GGD.backtrack"). This argument is ignored if family="binomial".
fold	user-supplied fold numbers for cross-validation. If supplied, fold should be an n-vector with entries in 1,...,K when doing K-fold cross-validation. The default is to choose fold using n.fold.
n.fold	the number of folds, K, to use for the K-fold cross-validation selection of tuning parameters. The default is 10 - specification of fold overrides use of n.fold.
seed	an optional number used with set.seed() at the beginning of the function. This is only relevant if fold is not specified by the user.
within1SE	logical (TRUE or FALSE) for how cross-validated tuning parameters should be chosen. If within1SE=TRUE, lambda is chosen to be the value corresponding to the most sparse model with cross-validation error within one standard error of the minimum cross-validation error. If within1SE=FALSE, lambda is chosen to be the value corresponding to the minimum cross-validation error.
tolerance	specifies the convergence criterion for the objective (default is 10e-6).

Details

Note that `flamCV` does not cross-validate over `alpha` - just a single value should be provided. However, if the user would like to cross-validate over `alpha`, then `flamCV` should be called multiple times for different values of `alpha` and the same `seed`. This ensures that the cross-validation folds (`fold`) remain the same for the different values of `alpha`. See the example below for details.

Value

An object with S3 class "flamCV".

`mean.cv.error`	m-vector containing cross-validation error where m is the length of `lambda.seq`. Note that `mean.cv.error[i]` contains the cross-validation error for tuning parameters `alpha` and `flam.out$all.lambda[i]`.
`se.cv.error`	m-vector containing cross-validation standard error where m is the length of `lambda.seq`. Note that `se.cv.error[i]` contains the standard error of the cross-validation error for tuning parameters `alpha` and `flam.out$all.lambda[i]`.
`lambda.cv`	optimal lambda value chosen by cross-validation.
`alpha`	as specified by user (or default).
`index.cv`	index of the model corresponding to 'lambda.cv'.
`flam.out`	object of class 'flam' returned by `flam`.
`fold`	as specified by user (or default).
`n.folds`	as specified by user (or default).
`within1SE`	as specified by user (or default).
`tolerance`	as specified by user (or default).
`call`	matched call.

Author(s)

Ashley Petersen

References

Petersen, A., Witten, D., and Simon, N. (2014). Fused Lasso Additive Model. arXiv preprint arXiv:1409.5391.

See Also

`flam`, `plot.flamCV`, `summary.flamCV`

Examples

```
#See ?'flam-package' for a full example of how to use this package

#generate data
set.seed(1)
data <- sim.data(n = 50, scenario = 1, zerof = 10, noise = 1)
```

```
#fit model for a range of lambda chosen by default
#pick lambda using 2-fold cross-validation
#note: use larger 'n.fold' (e.g., 10) in practice
flamCV.out <- flamCV(x = data$x, y = data$y, alpha = 0.75, n.fold = 2)

## Not run:
#note that cross-validation is only done to choose lambda for specified alpha
#to cross-validate over alpha also, call 'flamCV' for several alpha and set seed
#note: use larger 'n.fold' (e.g., 10) in practice
flamCV.out1 <- flamCV(x = data$x, y = data$y, alpha = 0.65, seed = 100,
within1SE = FALSE, n.fold = 2)
flamCV.out2 <- flamCV(x = data$x, y = data$y, alpha = 0.75, seed = 100,
within1SE = FALSE, n.fold = 2)
flamCV.out3 <- flamCV(x = data$x, y = data$y, alpha = 0.85, seed = 100,
within1SE = FALSE, n.fold = 2)
#this ensures that the folds used are the same
flamCV.out1$fold; flamCV.out2$fold; flamCV.out3$fold
#compare the CV error for the optimum lambda of each alpha to choose alpha
CVerrors <- c(flamCV.out1$mean.cv.error[flamCV.out1$index.cv],
flamCV.out2$mean.cv.error[flamCV.out2$index.cv],
flamCV.out3$mean.cv.error[flamCV.out3$index.cv])
best.alpha <- c(flamCV.out1$alpha, flamCV.out2$alpha,
flamCV.out3$alpha)[which(CVerrors==min(CVerrors))]

#also can generate data for logistic FLAM model
data2 <- sim.data(n = 50, scenario = 1, zerof = 10, family = "binomial")
#fit the FLAM model with cross-validation using logistic loss
#note: use larger 'n.fold' (e.g., 10) in practice
flamCV.logistic.out <- flamCV(x = data2$x, y = data2$y, family = "binomial",
n.fold = 2)

## End(Not run)

#'flamCV' returns an object of the class 'flamCV' that includes an object
#of class 'flam' (flam.out); see ?'flam-package' for an example using S3
#methods for the classes of 'flam' and 'flamCV'
```

`flamDOF` *Calculate Degrees of Freedom for Fused Lasso Additive Model*

Description

This function calculates the degrees of freedom for a fused lasso additive model fit using `flam`.

Usage

```
flamDOF(object, index)
```

Arguments

object	an object of the class "flam".
index	the index for the model of interest. Note that index of i corresponds to the model with tuning parameters object$all.alpha[i] and object$all.lambda[i].

Details

The degrees of freedom for FLAM were derived in Section 4.1 of Petersen, A., Witten, D., and Simon, N. (2014). Fused Lasso Additive Model. arXiv preprint arXiv:1409.5391.

Value

The degrees of freedom for the specified model.

Author(s)

Ashley Petersen

References

Petersen, A., Witten, D., and Simon, N. (2014). Fused Lasso Additive Model. arXiv preprint arXiv:1409.5391.

Examples

```
#See ?'flam-package' for a full example of how to use this package

#generate data
#note: use larger 'n' for more reasonable results
set.seed(1)
data <- sim.data(n = 20, scenario = 1, zerof = 10, noise = 1)

#fit model for a range of tuning parameters
flam.out <- flam(x = data$x, y = data$y)
#or fit model and select tuning parameters using 2-fold cross-validation
#note: use larger 'n.fold' (e.g., 10) in practice
flamCV.out <- flamCV(x = data$x, y = data$y, n.fold = 2)

#calculate degrees of freedom for the model chosen using cross-validation
flamDOF(object = flamCV.out$flam.out, index = flamCV.out$index.cv)
#or for any fit from a 'flam' object
flamDOF(object = flam.out, index = 25)
flamDOF(object = flamCV.out$flam.out, index = 25)
#which corresponds to lambda and alpha of
flam.out$all.lambda[25]; flam.out$all.alpha[25]
```

plot.flam *Plots Function Estimates for Fit of Class "flam"*

Description

This function plots the estimated functions from a model estimated using flam. The user specifies the model of interest (i.e., the tuning parameters) and a plot is made for the estimated association between all non-sparse (or a chosen subset of) features and the outcome.

Usage

```
## S3 method for class 'flam'
plot(x, index, n.plot = 10, predictor.indicators = NULL,
predictor.labels = NULL, outcome.label = "outcome", ticks = F,
col = "dodgerblue", n.panel.width = NULL, n.panel.height = NULL, ...)
```

Arguments

x	an object of class "flam".
index	the index for the model of interest to be plotted. Note that index of i corresponds to the model with tuning parameters x$all.alpha[i] and x$all.lambda[i].
n.plot	the number of predictors to be plotted (default of 10). Note that only non-sparse predictors are plotted, however, this argument is ignored if predictor.indicators is specified.
predictor.indicators	a vector indicating which predictor function estimates to plot. The vector should contain the column numbers of x$x whose function estimates are to be plotted. By default, the n.plot predictors with the largest L2 norm are plotted.
predictor.labels	a vector containing the predictor labels of the same length as predictor.indicators if specified, otherwise of length ncol(x$x). By default, "Predictor 1", "Predictor 2", ... are used.
outcome.label	the name of the outcome used in the y-axis label. By default, the label is "outcome".
ticks	a logical (TRUE or FALSE) for whether tick marks indicating the distribution of the predictor should be included at the bottom of each plot.
col	a vector of colors used to plot the function estimates. If col has length 1, then the same color is used for all predictors. Otherwise, col should indicate the color for each predictor and have the same length as predictor.indicators if specified, otherwise the number of columns of x$x.
n.panel.width	the plots will be plotted with par(mfrow=c(n.panel.height,n.panel.width)). If specified, n.panel.height must also be specified. By default, an appropriate grid of plots is used.

`n.panel.height`	the plots will be plotted with `par(mfrow=c(n.panel.height,n.panel.width))`. If specified, `n.panel.width` must also be specified. By default, an appropriate grid of plots is used.
`...`	additional arguments to be passed. These are ignored in this function.

Details

The estimated function fits are drawn by connecting the predictions for all of the observations in `x$x`. This may result in fits that appear not to be piecewise consant if only a single observation is observed for a range of x-values.

Author(s)

Ashley Petersen

References

Petersen, A., Witten, D., and Simon, N. (2014). Fused Lasso Additive Model. arXiv preprint arXiv:1409.5391.

See Also

`flam`

Examples

```
#See ?'flam-package' for a full example of how to use this package

#generate data
set.seed(1)
data <- sim.data(n = 50, scenario = 1, zerof = 10, noise = 1)
#fit model for a range of tuning parameters
flam.out <- flam(x = data$x, y = data$y, alpha.seq = c(0.8, 0.9, 1))

#we plot the predictor fits for a specific index, e.g. 25
#that is, lambda and alpha of
flam.out$all.lambda[25]; flam.out$all.alpha[25]
plot(flam.out, index = 25)
#the fit only has 5 non-sparse features

#by default, up to 10 non-sparse features with the largest L2 norms are
#plotted, but we can plot a different number of features if desired
plot(flam.out, index = 40, n.plot = 12)
#or we can plot specific predictors of interest
plot(flam.out, index = 40, predictor.indicators = c(1:4, 6, 8, 11, 12))
```

plot.flamCV *Plots Cross-Validation Curve for Object of Class "flamCV"*

Description

This function plots the cross-validation curve for a series of models fit using flamCV. The cross-validation error with +/-1 standard error is plotted for each value of lambda considered in the call to flamCV with a dotted vertical line indicating the chosen lambda.

Usage

```
## S3 method for class 'flamCV'
plot(x, showSE = T, ...)
```

Arguments

x	an object of class "flamCV".
showSE	a logical (TRUE or FALSE) for whether the standard errors of the curve should be plotted.
...	additional arguments to be passed. These are ignored in this function.

Author(s)

Ashley Petersen

References

Petersen, A., Witten, D., and Simon, N. (2014). Fused Lasso Additive Model. arXiv preprint arXiv:1409.5391.

See Also

flamCV

Examples

```
#See ?'flam-package' for a full example of how to use this package

#generate data
set.seed(1)
data <- sim.data(n = 50, scenario = 1, zerof = 0, noise = 1)

#fit model and select tuning parameters using 2-fold cross-validation
#note: use larger 'n.fold' (e.g., 10) in practice
flamCV.out <- flamCV(x = data$x, y = data$y, within1SE = TRUE, n.fold = 2)

#lambdas chosen is
flamCV.out$lambda.cv
```

```
#we can now plot the cross-validation error curve with standard errors
#vertical dotted line at lambda chosen by cross-validation
plot(flamCV.out)
#or without standard errors
plot(flamCV.out, showSE = FALSE)

## Not run:
#can choose lambda to be value with minimum CV error
#instead of lambda with CV error within 1 standard error of the minimum
flamCV.out2 <- flamCV(x = data$x, y = data$y, within1SE = FALSE, n.fold = 2)

#contrast to chosen lambda for minimum cross-validation error
#it's a less-regularized model (i.e., lambda is smaller)
plot(flamCV.out2)

## End(Not run)
```

`predict.flam` *Predicts Observations for a New Covariate Matrix and Fit from* `flam`

Description

This function makes predictions from a specified covariate matrix for a fit of the class "flam" with user-specified tuning parameters.

Usage

```
## S3 method for class 'flam'
predict(object, new.x, lambda, alpha, ...)
```

Arguments

`object`	an object of the class "flam".
`new.x`	the covariate matrix for which to make predictions - the number of columns should match that of `object$x`.
`lambda`	the desired value for the tuning parameter lambda. This does not need to be a value in `object$all.lambda`.
`alpha`	the desired value for the tuning parameter alpha. This does not need to be a value in `object$all.alpha`.
`...`	additional arguments to be passed. These are ignored in this function.

Details

It is likely that `new.x[,i]` contains values not contained in `object$x[,i]`. Predictions for that particular case are taken to be a linear interpolation of the nearest neighboring values in `object$x[,i]`, i.e., the closest smaller value and the closest larger value.

Value

A vector containing the fitted y values for new.x.

Author(s)

Ashley Petersen

References

Petersen, A., Witten, D., and Simon, N. (2014). Fused Lasso Additive Model. arXiv preprint arXiv:1409.5391.

See Also

flam

Examples

```
#See ?'flam-package' for a full example of how to use this package

#generate data
set.seed(1)
data <- sim.data(n = 100, scenario = 1, zerof = 0, noise = 1)

#fit model for a range of tuning parameters
flam.out <- flam(x = data$x, y = data$y)

#we can make predictions for a covariate matrix with new observations
#choose desired alpha and lambda
alpha <- flam.out$all.alpha[15]; lambda <- flam.out$all.lambda[15]
#new.x with 20 observations and the same number of features as flam.out$x
new.data <- sim.data(n = 20, scenario = 1, zerof = 0, noise = 1)
new.x <- new.data$x
#make predictions
y.hat <- predict(flam.out, new.x = new.x, lambda = lambda, alpha = alpha)
#which can be compared to the true y
plot(new.data$y, y.hat, xlab="y", ylab=expression(hat(y)))
abline(0,1,lty=2)

#can also make predictions for any alpha and lambda:
predict(flam.out, new.x = new.x, lambda = 2, alpha = 0.9)
```

---

sim.data *Simulate Data from a Variety of Functional Scenarios*

Description

This function generates data according to the simulation scenarios considered in Section 5 and plotted in Figure 2 of Petersen, A., Witten, D., and Simon, N. (2014). Fused Lasso Additive Model. arXiv preprint arXiv:1409.5391. Each scenario has four covariates that have some non-linear association with the outcome. There is the option to also generate a user-specified number of covariates that have no association with the outcome.

Usage

```
sim.data(n, scenario, zerof, noise = 1, family = "gaussian")
```

Arguments

`n`	number of observations.
`scenario`	simulation scenario to use. Options are 1, 2, 3, or 4, which correspond to the simulation scenarios of Section 5 in Petersen, A., Witten, D., and Simon, N. (2014). Fused Lasso Additive Model. arXiv preprint arXiv:1409.5391. Each scenario has four covariates.
`zerof`	number of noise covariates (those that have no relationship to the outcome) to include. This can be used to replicate the high-dimensional scenarios of Section 5 in Petersen, A., Witten, D., and Simon, N. (2014). Fused Lasso Additive Model. arXiv preprint arXiv:1409.5391. The total number of covariates will be `4 + zerof`.
`noise`	the variance of the errors. If `family = "gaussian"`, the errors of observations are generated using MVN(0, `noise`I) with default of 1, otherwise `noise` is not used if `family="binomial"`.
`family`	the error distribution of observations (must be `family="gaussian"` or `family="binomial"`). If `family = "gaussian"`, the errors of observations are generated using MVN(0, `noise`I), otherwise observations are Bernoulli if `family="binomial"`.

Value

`x`	n x p covariate matrix.
`y`	n-vector containing the outcomes for the n observations in `x`.
`theta`	n x p mean matrix used to generate `y`.

Author(s)

Ashley Petersen

References

Petersen, A., Witten, D., and Simon, N. (2014). Fused Lasso Additive Model. arXiv preprint arXiv:1409.5391.

Examples

```
#See ?'flam-package' for a full example of how to use this package

#generate data to fit FLAM model with squared-error loss
set.seed(1)
data <- sim.data(n = 50, scenario = 1, zerof = 10, noise = 1)
flam.out <- flam(x = data$x, y = data$y, family = "gaussian")

#alternatively, generate data for logistic FLAM model
#note: 'noise' argument no longer needed
data2 <- sim.data(n = 50, scenario = 1, zerof = 0, family = "binomial")
flam.logistic.out <- flam(x = data2$x, y = data2$y, family = "binomial")


#vary generating functions
#choose large n because we want to plot generating functions
data1 <- sim.data(n = 500, scenario = 1, zerof = 0)
data2 <- sim.data(n = 500, scenario = 2, zerof = 0)
data3 <- sim.data(n = 500, scenario = 3, zerof = 0)
data4 <- sim.data(n = 500, scenario = 4, zerof = 0)
#and plot to see functional forms
par(mfrow=c(2,2))
col.vec = c("dodgerblue1","orange","seagreen1","hotpink")
for (i in 1:4) {
if (i==1) data = data1 else if (i==2) data = data2
else if (i==3) data = data3 else data = data4
plot(1,type="n",xlim=c(-2.5,2.5),ylim=c(-3,3),xlab=expression(x[j]),
ylab=expression(f[j](x[j])),main=paste("Scenario ",i,sep=""))
sapply(1:4, function(j) points(sort(data$x[,j]),
data$theta[order(data$x[,j]),j],col=col.vec[j],type="l",lwd=3))
}

#include large number of predictors that have no relationship to outcome
data <- sim.data(n = 50, scenario = 1, zerof = 100, noise = 1)
```

---

summary.flam *Summarizes a Call to* flam

Description

This function summarizes a call to flam, as well as the sparsity pattern of the resulting feature estimates for a single or all fits.

Usage

```
## S3 method for class 'flam'
summary(object, index = NULL, ...)
## S3 method for class 'flamSparsity'
plot(x, ...)
```

Arguments

object	an object of class "flam".
index	the index for the fit of interest to be summarized. Note that index of i corresponds to the model with tuning parameters object$all.alpha[i] and object$all.lambda[i]. If index is not specified, information summarizing all fits is given.
x	an object of class 'flamSparsity', which is silently returned by summary.flam.
...	additional arguments to be passed. These are ignored in this function.

Value

If index is not specified, summary.flam silently returns the sparsity matrix and tuning parameters in an object of class 'flamSparsity'. This is used when plot(summary(object)) is called.

Author(s)

Ashley Petersen

References

Petersen, A., Witten, D., and Simon, N. (2014). Fused Lasso Additive Model. arXiv preprint arXiv:1409.5391.

See Also

flam

Examples

```
#See ?'flam-package' for a full example of how to use this package

#generate data
set.seed(1)
data <- sim.data(n = 50, scenario = 1, zerof = 10, noise = 1)
#fit model for a range of tuning parameters
flam.out <- flam(x = data$x, y = data$y, alpha.seq = c(0.8, 0.9, 1))

#summarize all of the fits (in particular, the percent sparsity achieved)
#note: percent sparsity is the percentage of features estimated to have
#no relationship with outcome
summary(flam.out)
#the percent sparsity over the range of tuning parameters can also
#be displayed in a plot
plot(summary(flam.out))

#we can also summarize the fit with a specific index, e.g. 25
#that is, lambda and alpha of
flam.out$all.lambda[25]; flam.out$all.alpha[25]
summary(flam.out, index = 25)
```

summary.flamCV *Summarizes a Call to* flamCV

Description

This function summarizes a call to codeflamCV and identifies the tuning parameter chosen by cross-validation.

Usage

```
## S3 method for class 'flamCV'
summary(object, ...)
```

Arguments

object	an object of class "flamCV".
...	additional arguments to be passed. These are ignored in this function.

Author(s)

Ashley Petersen

References

Petersen, A., Witten, D., and Simon, N. (2014). Fused Lasso Additive Model. arXiv preprint arXiv:1409.5391.

See Also

flamCV

Examples

```
#See ?'flam-package' for a full example of how to use this package

#generate data
set.seed(1)
data <- sim.data(n = 50, scenario = 1, zerof = 0, noise = 1)
#fit model and select tuning parameters using 2-fold cross-validation
#note: use larger 'n.fold' (e.g., 10) in practice
flamCV.out <- flamCV(x = data$x, y = data$y, n.fold = 2)

#we can summarize the cross-validation function call
summary(flamCV.out)
#lambda chosen by cross-validation is also available from
flamCV.out$lambda.cv
```

Index