Package 'crisp'

October 12, 2022

Type Package

Title Fits a Model that Partitions the Covariate Space into Blocks in a Data- Adaptive Way

Version 1.0.0

Author Ashley Petersen

Maintainer Ashley Petersen <ashleyjpete@gmail.com>

Description Implements convex regression with interpretable sharp partitions (CRISP), which considers the problem of predicting an outcome variable on the basis of two covariates, using an interpretable yet non-additive model. CRISP partitions the covariate space into blocks in a data-adaptive way, and fits a mean model within each block. Unlike other partitioning methods, CRISP is fit using a non-greedy approach by solving a convex optimization problem, resulting in low-variance fits. More details are provided in Petersen, A., Simon, N., and Witten, D. (2016). Convex Regression with Interpretable Sharp Partitions. Journal of Machine Learning Research, 17(94): 1-31 <http://jmlr.org/papers/volume17/15-344/15-344.pdf>.

Imports Matrix, MASS, stats, methods, grDevices, graphics

License GPL (>= 2)

LazyData TRUE

RoxygenNote 5.0.1

NeedsCompilation no

Repository CRAN

Date/Publication 2017-01-05 10:39:31

R topics documented:

`crisp-package` *crisp: A package for fitting a model that partitions the covariate space into blocks in a data-adaptive way.*

Description

This package is called crisp for "Convex Regression with Interpretable Sharp Partitions", which considers the problem of predicting an outcome variable on the basis of two covariates, using an interpretable yet non-additive model. CRISP partitions the covariate space into blocks in a data-adaptive way, and fits a mean model within each block. Unlike other partitioning methods, CRISP is fit using a non-greedy approach by solving a convex optimization problem, resulting in low-variance fits. More details are provided in Petersen, A., Simon, N., and Witten, D. (2016). Convex Regression with Interpretable Sharp Partitions. Journal of Machine Learning Research, 17(94): 1-31 <http://jmlr.org/papers/volume17/15-344/15-344.pdf>.

Details

The main functions are: (1)`crisp` and (2)`crispCV`. The first function `crisp` fits CRISP for a sequence of tuning parameters and provides the fits for this entire sequence of tuning parameters. The second function `crispCV` considers a sequence of tuning parameters and provides the fits, but also returns the optimal tuning parameter, as chosen using K-fold cross-validation.

Examples

```
## Not run:
#general example illustrating all functions
#see specific function help pages for details of using each function

#generate data (using a very small 'n' for illustration purposes)
set.seed(1)
data <- sim.data(n = 15, scenario = 2)
#plot the mean model for the scenario from which we generated data
plot(data)

#fit model for a range of tuning parameters, i.e., lambda values
#lambda sequence is chosen automatically if not specified
crisp.out <- crisp(X = data$X, y = data$y)
#or fit model and select lambda using 2-fold cross-validation
#note: use larger 'n.fold' (e.g., 10) in practice
crispCV.out <- crispCV(X = data$X, y = data$y, n.fold = 2)

#summarize all of the fits
summary(crisp.out)
#or just summarize a single fit
#we examine the fit with an index of 25. that is, lambda of
```

```
crisp.out$lambda.seq[25]
summary(crisp.out, lambda.index = 25)
#lastly, we can summarize the fit chosen using cross-validation
summary(crispCV.out)
#and also plot the cross-validation error
plot(summary(crispCV.out))
#the lambda chosen by cross-validation is also available using
crispCV.out$lambda.cv

#plot the estimated relationships between two predictors and outcome
#do this for a specific fit
plot(crisp.out, lambda.index = 25)
#or for the fit chosen using cross-validation
plot(crispCV.out)

#we can make predictions for a covariate matrix with new observations
#new.X with 20 observations
new.data <- sim.data(n = 20, scenario = 2)
new.X <- new.data$X
#these will give the same predictions:
yhat1 <- predict(crisp.out, new.X = new.X, lambda.index = crispCV.out$index.cv)
yhat2 <- predict(crispCV.out, new.X = new.X)

## End(Not run)
```

`crisp` *Convex Regression with Interpretable Sharp Partitions (CRISP).*

Description

This function implements CRISP, which considers the problem of predicting an outcome variable on the basis of two covariates, using an interpretable yet non-additive model. CRISP partitions the covariate space into blocks in a data-adaptive way, and fits a mean model within each block. Unlike other partitioning methods, CRISP is fit using a non-greedy approach by solving a convex optimization problem, resulting in low-variance fits. More details are provided in Petersen, A., Simon, N., and Witten, D. (2016). Convex Regression with Interpretable Sharp Partitions. Journal of Machine Learning Research, 17(94): 1-31 <http://jmlr.org/papers/volume17/15-344/15-344.pdf>.

Usage

```
crisp(y, X, q = NULL, lambda.min.ratio = 0.01, n.lambda = 50,
  lambda.seq = NULL, rho = 0.1, e_abs = 10^-4, e_rel = 10^-3,
  varyrho = TRUE, double.run = FALSE)
```

Arguments

y	An n-vector containing the response.
X	An n x 2 matrix with each column containing a covariate.

`q` The desired granularity of the CRISP fit, `M.hat`, which will be a `q` by `q` matrix. `M.hat` is a mean matrix whose element `M.hat[i,j]` contains the mean for pairs of covariate values within a quantile range of the observed predictors `X[,1]` and `X[,2]`. For example, `M.hat[1,2]` represents the mean of the observations with the first covariate value less than the `1/q`-quantile of `X[,1]`, and the second covariate value between the `1/q`- and `2/q`-quantiles of `X[,2]`. If left `NULL`, then `q=n` is used when n<100, and `q=100` is used when n>=100. We recommend using `q<=100` as higher values take longer to fit and provide an unneeded amount of granularity.

`lambda.min.ratio` The smallest value for `lambda.seq`, as a fraction of the maximum lambda value, which is the data-derived smallest value for which the fit is a constant value. 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 CRISP 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.

`rho` The penalty parameter for our ADMM algorithm. The default is 0.1.

`e_abs, e_rel` Values used in the stopping criterion for our ADMM algorithm, and discussed in Appendix C.2 of the CRISP paper.

`varyrho` Should `rho` be varied from iteration to iteration? This is discussed in Appendix C.3 of the CRISP paper.

`double.run` The initial complete run of our ADMM algorithm will yield sparsity in z_1i and z_2i, but not necessarily exact equality of the rows and columns of `M.hat`. If `double.run` is `TRUE`, then the algorithm is run a second time to obtain `M.hat` with exact equality of the appropriate rows and columns. This issue is discussed further in Appendix C.4 of the CRISP paper.

Value

An object of class `crisp`, which can be summarized using `summary`, plotted using `plot`, and used to predict outcome values for new covariates using `predict`.

- `M.hat.list`: A list of length `n.lambda` giving `M.hat` for each value of `lambda.seq`.
- `num.blocks`: A vector of length `n.lambda` giving the number of blocks in `M.hat` for each value of `lambda.seq`.
- `obj.vec`: A vector of length `n.lambda` giving the value of the objective of Eqn (4) in the CRISP paper for each value of `lambda.seq`.
- Other elements: As specified by the user.

See Also

`crispCV`, `plot`, `summary`, `predict`

Examples

```
## Not run:
#See ?'crisp-package' for a full example of how to use this package

#generate data (using a very small 'n' for illustration purposes)
set.seed(1)
data <- sim.data(n = 15, scenario = 2)

#fit model for a range of tuning parameters, i.e., lambda values
#lambda sequence is chosen automatically if not specified
crisp.out <- crisp(X = data$X, y = data$y)

## End(Not run)
```

`crispCV` *CRISP with Tuning Parameter Selection via Cross-Validation.*

Description

This function implements CRISP, which considers the problem of predicting an outcome variable on the basis of two covariates, using an interpretable yet non-additive model. CRISP partitions the covariate space into blocks in a data-adaptive way, and fits a mean model within each block. Unlike other partitioning methods, CRISP is fit using a non-greedy approach by solving a convex optimization problem, resulting in low-variance fits. This function differs from the `crisp` function in that the tuning parameter, lambda, is automatically selected using K-fold cross-validation. More details are provided in Petersen, A., Simon, N., and Witten, D. (2016). Convex Regression with Interpretable Sharp Partitions. Journal of Machine Learning Research, 17(94): 1-31 <http://jmlr.org/papers/volume17/15-344/15-344.pdf>.

Usage

```
crispCV(y, X, q = NULL, lambda.min.ratio = 0.01, n.lambda = 50,
  lambda.seq = NULL, fold = NULL, n.fold = NULL, seed = NULL,
  within1SE = FALSE, rho = 0.1, e_abs = 10^-4, e_rel = 10^-3,
  varyrho = TRUE, double.run = FALSE)
```

Arguments

`y`	An n-vector containing the response.
`X`	An n x 2 matrix with each column containing a covariate.
`q`	The desired granularity of the CRISP fit, `M.hat`, which will be a `q` by `q` matrix. `M.hat` is a mean matrix whose element `M.hat[i,j]` contains the mean for pairs of covariate values within a quantile range of the observed predictors `X[,1]` and `X[,2]`. For example, `M.hat[1,2]` represents the mean of the observations with the first covariate value less than the `1/q`-quantile of `X[,1]`, and the second covariate value between the `1/q`- and `2/q`-quantiles of `X[,2]`. If left `NULL`, then `q=n` is used when n<100, and `q=100` is used when n>=100. We recommend using

q<=100 as higher values take longer to fit and provide an unneeded amount of granularity.

lambda.min.ratio
: The smallest value for lambda.seq, as a fraction of the maximum lambda value, which is the data-derived smallest value for which the fit is a constant value. 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 CRISP 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.

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 the tuning parameter, lambda. 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 value indicating 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.

rho
: The penalty parameter for our ADMM algorithm. The default is 0.1.

e_abs, e_rel
: Values used in the stopping criterion for our ADMM algorithm, and discussed in Appendix C.2 of the CRISP paper.

varyrho
: Should rho be varied from iteration to iteration? This is discussed in Appendix C.3 of the CRISP paper.

double.run
: The initial complete run of our ADMM algorithm will yield sparsity in z_1i and z_2i, but not necessarily exact equality of the rows and columns of M.hat. If double.run is TRUE, then the algorithm is run a second time to obtain M.hat with exact equality of the appropriate rows and columns. This issue is discussed further in Appendix C.4 of the CRISP paper.

Value

An object of class crispCV, which can be summarized using summary, plotted using plot, and used to predict outcome values for new covariates using predict.

- lambda.cv: Optimal lambda value chosen by K-fold cross-validation.
- index.cv: The index of the model corresponding to the chosen tuning parameter, lambda.cv. That is, lambda.cv=crisp.out$lambda.seq[index.cv].

- `crisp.out`: An object of class `crisp` returned by `crisp`.
- `mean.cv.error`: An 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 the tuning parameter `crisp.out$lambda.seq[i]`.
- `se.cv.error`: An m-vector containing cross-validation standard error where m is the length of `lambda.seq`. Note that `se.cv.error[i]` contains the standard error of the cross-validation error for the tuning parameter `crisp.out$lambda.seq[i]`.
- Other elements: As specified by the user.

See Also

`crisp`, `plot`, `summary`, `predict`, `plot.cvError`

Examples

```
## Not run:
#See ?'crisp-package' for a full example of how to use this package

#generate data (using a very small 'n' for illustration purposes)
set.seed(1)
data <- sim.data(n = 15, scenario = 2)

#fit model and select lambda using 2-fold cross-validation
#note: use larger 'n.fold' (e.g., 10) in practice
crispCV.out <- crispCV(X = data$X, y = data$y, n.fold = 2)

## End(Not run)
```

---

plot *Plots Fit from* `crisp` *or* `crispCV`.

Description

This function plots fit of the class `crispCV`, or class `crisp` with a user-specified tuning parameter.

Usage

```
## S3 method for class 'crisp'
plot(x, lambda.index, title = NULL, x1lab = NULL,
  x2lab = NULL, min = NULL, max = NULL, cex.axis = 1, cex.lab = 1,
  color1 = "seagreen1", color2 = "steelblue1", color3 = "darkorchid4",
  ...)

## S3 method for class 'crispCV'
plot(x, title = NULL, x1lab = NULL, x2lab = NULL,
  min = NULL, max = NULL, cex.axis = 1, cex.lab = 1,
  color1 = "seagreen1", color2 = "steelblue1", color3 = "darkorchid4",
  ...)
```

Arguments

x	An object of class crisp or crispCV, which result from running the crisp or crispCV functions, respectively.
lambda.index	The index for the desired value of lambda, i.e., x$lambda.seq[lambda.index].
title	The title of the plot. By default, the value of lambda is noted.
x1lab	The axis label for the first covariate. By default, it is "X1".
x2lab	The axis label for the second covariate. By default, it is "X2".
min, max	The minimum and maximum y-values, respectively, to use when plotting the fit. By default, they are chosen to be the minimum and maximum of all of the fits, i.e., the minimum and maximum of unlist(x$M.hat.list).
cex.axis	The magnification to be used for axis annotation relative to the current setting of cex.
cex.lab	The magnification to be used for x and y labels relative to the current setting of cex.
color1, color2, color3	The colors to use to create the color gradient for plotting the response values. At least the first two must be specified, or the defaults of "seagreen1", "steelblue1", and "darkorchid4" will be used.
...	Additional arguments to be passed, which are ignored in this function.

Value

None.

Examples

```
## Not run:
#See ?'crisp-package' for a full example of how to use this package

#generate data (using a very small 'n' for illustration purposes)
set.seed(1)
data <- sim.data(n = 15, scenario = 2)

#fit model for a range of tuning parameters, i.e., lambda values
#lambda sequence is chosen automatically if not specified
crisp.out <- crisp(X = data$X, y = data$y)
#or fit model and select lambda using 2-fold cross-validation
#note: use larger 'n.fold' (e.g., 10) in practice
crispCV.out <- crispCV(X = data$X, y = data$y, n.fold = 2)

#plot the estimated relationships between two predictors and outcome
#do this for a specific fit
plot(crisp.out, lambda.index = 25)
#or for the fit chosen using cross-validation
plot(crispCV.out)

## End(Not run)
```

plot.cvError *Plots Cross-Validation Curve for* crispCV.

Description

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

Usage

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

Arguments

x	An object of class cvError, which results from calling summary on an object of class crispCV.
showSE	A logical value indicating whether the standard error of the curve should be plotted.
...	Additional arguments to be passed, which are ignored in this function.

Value

None.

Examples

```
## Not run:
#See ?'crisp-package' for a full example of how to use this package

#generate data (using a very small 'n' for illustration purposes)
set.seed(1)
data <- sim.data(n = 15, scenario = 2)

#fit model and select lambda using 2-fold cross-validation
#note: use larger 'n.fold' (e.g., 10) in practice
crispCV.out <- crispCV(X = data$X, y = data$y, n.fold = 2)

#plot the cross-validation error
plot(summary(crispCV.out))

## End(Not run)
```

plot.sim.data *Plot Mean Model for Data.*

Description

This function plots the mean model for the scenario from which data was generated using sim.data.

Usage

```
## S3 method for class 'sim.data'
plot(x, ...)
```

Arguments

x	An object of class sim.data, which results from running the sim.data function.
...	Additional arguments to be passed, which are ignored in this function.

Value

None.

See Also

sim.data

Examples

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

#generate data (using a very small 'n' for illustration purposes)
set.seed(1)
data <- sim.data(n = 15, scenario = 2)

#plot the mean model for the scenario from which we generated data
plot(data)
```

predict *Predicts Observations for a New Covariate Matrix using Fit from* crisp *or* crispCV.

Description

This function makes predictions for a specified covariate matrix for a fit of the class crispCV, or class crisp with a user-specified tuning parameter.

Usage

```
## S3 method for class 'crisp'
predict(object, new.X, lambda.index, ...)

## S3 method for class 'crispCV'
predict(object, new.X, ...)
```

Arguments

object	An object of class crisp or crispCV, which result from running the crisp or crispCV functions, respectively.
new.X	The covariate matrix for which to make predictions.
lambda.index	The index for the desired value of lambda, i.e., object$lambda.seq[lambda.index].
...	Additional arguments to be passed, which are ignored in this function.

Details

The ith prediction is made to be the value of object$M.hat.list[[lambda.index]] corresponding to the pair of covariates closest (in Euclidean distance) to new.X[i,].

Value

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

Examples

```
## Not run:
#See ?'crisp-package' for a full example of how to use this package

#generate data (using a very small 'n' for illustration purposes)
set.seed(1)
data <- sim.data(n = 15, scenario = 2)

#fit model for a range of tuning parameters, i.e., lambda values
#lambda sequence is chosen automatically if not specified
crisp.out <- crisp(X = data$X, y = data$y)
#or fit model and select lambda using 2-fold cross-validation
#note: use larger 'n.fold' (e.g., 10) in practice
crispCV.out <- crispCV(X = data$X, y = data$y, n.fold = 2)

#we can make predictions for a covariate matrix with new observations
#new.X with 20 observations
new.data <- sim.data(n = 20, scenario = 2)
new.X <- new.data$X
#these will give the same predictions:
yhat1 <- predict(crisp.out, new.X = new.X, lambda.index = crispCV.out$index.cv)
yhat2 <- predict(crispCV.out, new.X = new.X)

## End(Not run)
```

sim.data *Simulate Data to Use with* crisp.

Description

This function generates data according to the simulation scenarios considered in Section 3 of the CRISP paper (and plotted in Figure 2 of the paper).

Usage

```
sim.data(n, scenario, noise = 1, X = NULL)
```

Arguments

n	The number of observations.
scenario	The simulation scenario to use. Options are 1 (additive model), 2 (interaction model), 3 ('tetris' model), or 4 (smooth model), which correspond to the simulation scenarios of Section 3 of the CRISP paper. Each scenario has two covariates.
noise	The standard deviation of the normally-distributed noise that is added to the signal.
X	The n x 2 covariate matrix, which is automatically generated if not specified.

Value

A list containing:

- X: An n x 2 covariate matrix.
- y: An n-vector containing the response values.
- Other elements: As specified by the user.

See Also

crisp, crispCV

Examples

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

#generate data (using a very small 'n' for illustration purposes)
set.seed(1)
data <- sim.data(n = 15, scenario = 2)

#plot the mean model for the scenario from which we generated data
plot(data)
```

summary *Summarizes Fit from* crisp *or* crispCV.

Description

This function summarizes fit of the class crispCV or crisp.

Usage

```
## S3 method for class 'crisp'
summary(object, lambda.index = NULL, ...)

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

Arguments

object	An object of class crisp or crispCV, which result from running the crisp or crispCV functions, respectively.
lambda.index	The index for the desired value of lambda, i.e., object$lambda.seq[lambda.index]. By default, fits for all values of lambda are summarized.
...	Additional arguments to be passed, which are ignored in this function.

Value

None.

Examples

```
## Not run:
#See ?'crisp-package' for a full example of how to use this package
#generate data (using a very small 'n' for illustration purposes)
set.seed(1)
data <- sim.data(n = 15, scenario = 2)

#fit model for a range of tuning parameters, i.e., lambda values
#lambda sequence is chosen automatically if not specified
crisp.out <- crisp(X = data$X, y = data$y)
#or fit model and select lambda using 2-fold cross-validation
#note: use larger 'n.fold' (e.g., 10) in practice
crispCV.out <- crispCV(X = data$X, y = data$y, n.fold = 2)

#summarize all of the fits
summary(crisp.out)
#or just summarize a single fit
#we examine the fit with an index of 25. that is, lambda of
crisp.out$lambda.seq[25]
summary(crisp.out, lambda.index = 25)
```

```
#lastly, we can summarize the fit chosen using cross-validation
summary(crispCV.out)

## End(Not run)
```

Index