Package 'CausalKinetiX'

June 10, 2019

Title Learning Stable Structures in Kinetic Systems

Version 0.2

Description
Implementation of 'CausalKinetiX', a framework for learning stable structures in kinetic systems. Apart from the main functions CausalKinetiX() and CausalKinetiX.modelranking() it includes functions to generate data from three simulations models, which can be used to benchmark structure learning methods for linear ordinary differential equation models. A detailed description of the underlying methods as well as details on the examples are given in Pfister, Bauer and Peters (2018) <arXiv:1810.11776>.

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

Imports fda, cvTools, quadprog, randomForest, deSolve, stats, graphics, pspline, utils, glmnet, nnlasso, sundialr (>= 0.1.3)

NeedsCompilation no

Author Niklas Pfister [aut, cre], Stefan Bauer [aut], Jonas Peters [aut]

Maintainer Niklas Pfister <niklas.pfister@stat.math.ethz.ch>

Repository CRAN

Date/Publication 2019-06-10 17:50:03 UTC

R topics documented:

CausalKinetiX *CausalKinetiX*

Description

Applies CausalKinetiX framework to rank variables and models according to their stability.

Usage

```
CausalKinetiX(D, times, env, target, models = NA, pars = list())
```

Arguments

D	data matrix. Should have dimension n x (L*d), where n is the number of repetitions (over all experiments), L is the number of time points and d is the number of predictor variables.
times	vector of length L specifying the time points at which data was observed.
env	integer vector of length n encoding to which experiment each repetition belongs.
target	integer specifing which variable is the target.
models	list of models. Each model is specified by a list of vectors specifiying the variables included in the interactions of each term. If NA, then models are constructed automatically using the parameters in pars.
pars	list of the following parameters: max.preds (default FALSE) if TRUE also models with lower terms included, expsize (default 2) the expected number of terms, interactions (default FALSE) specifies whether to include interactions in the models, products (default FALSE) specifies whether to include products in the models, include.vars (default NA) specifies variables that should be included in each model, maineffect.models (default FALSE) main-effect models or exhaustive models, screening (default NA) specifies the number of terms remaining after screening - NA implies screening, K (default NA) cutoff parameter used in variable ranking. Additionally all parameters used in CausalKinetiX.modelranking can also be specified here.

Details

For further details see the references.

Value

object of class 'CausalKinetiX' consisting of the following elements

models	list of the individually scored models.
model.scores	vector containing the score for each model.
variable.scores	vector containing the score of each variable.
ranking	vector specifying the ranking of each variable.

Author(s)

Niklas Pfister, Stefan Bauer and Jonas Peters

References

Pfister, N., S. Bauer, J. Peters (2018). Identifying Causal Structure in Large-Scale Kinetic Systems ArXiv e-prints (arXiv:1810.11776).

See Also

The function `CausalKinetiX.modelranking` can be used if the variable ranking is not required.

Examples

```
## Generate data from Maillard reaction
simulation.obj <- generate.data.maillard(target=6,
                                         env=rep(1:3, 5),
                                         L=15,
                                         seed=5,
                                         par.noise=list(noise.sd=1))

D <- simulation.obj$simulated.data
time <- simulation.obj$time
env <- simulation.obj$env
target <- simulation.obj$target

## Fit data using CausalKinetiX
ck.fit <- CausalKinetiX(D, time, env, target,
                        pars=list(expsize=1,
                                  average.reps=TRUE))
# variable ranking (here the true parent is variable 4)
print(ck.fit$ranking)
```

`CausalKinetiX.modelranking` *CausalKinetix.modelranking*

Description

Applies CausalKinetiX framework to rank a list models according to their stability.

Usage

```
CausalKinetiX.modelranking(D, times, env, target, models, pars = list())
```

Arguments

D	data matrix. Should have dimension n x (L*d), where n is the number of repetitions (over all experiments), L is the number of time points and d is the number of predictor variables.
times	vector of length L specifying the time points at which data was observed.
env	integer vector of length n encoding to which experiment each repetition belongs.
target	integer specifing which variable is the target.
models	list of models. Each model is specified by a list of vectors specifiying the variables included in the interactions of each term.
pars	list of the following parameters: pen.degree (default 2) specifies the penalization degree in the smoothing spline, num.folds (default 2) number of folds used in cross-validation of smoothing spline, include.vars (default NA) specifies variables that should be included in each model, include.intercept (default FALSE) specifies whether to include a intercept in models, average.reps (default FALSE) specifies whether to average repetitions in each environment, smooth.X (default FALSE) specifies whether to smooth predictor observations before fitting, smooth.Y (default FALSE) specifies whether to smooth target observations before fitting, regression.class (default OLS) other options are signed.OLS, optim, random.forest, sample.splitting (default "loo") either leave-one-out (loo) or no splitting (none), score.type (default "mean_absolute") specifies the type of score funtion to use (note that "mean" and "max" are proportional scores which should be used if the noise variance is expected to change across experiments, if this is not the case "mean_absolute" and "max_absolute" are prefered as they also work for perfect unconstrained spline fits), integrated.model (default TRUE) specifies whether to fit the integrated or the derived model, splitting.env (default NA) an additonal environment vector used for scoring, weight.vec (default rep(1, length(env)) a weight vector used when scoring.type=="weighted.mean", set.initial (default FALSE) specifies whether to fix the initial value, silent (default TRUE) turn of additional output, show.plot (default FALSE) show diagnostic plots.

Details

This function scores a specified list of models and does not include a variable ranking.

Value

returns a list with the entries "scores" and "parameter"

Author(s)

Niklas Pfister, Stefan Bauer and Jonas Peters

References

Pfister, N., S. Bauer, J. Peters (2018). Identifying Causal Structure in Large-Scale Kinetic Systems ArXiv e-prints (arXiv:1810.11776).

See Also

The function CausalKinetiX is a wrapper for this function that also computes the variable ranking and generates sensible classes of models.

Examples

```
## Generate data from Maillard reaction
simulation.obj <- generate.data.maillard(target=1,
                                         env=rep(1:5, 3),
                                         L=20,
                                         par.noise=list(noise.sd=1))
D <- simulation.obj$simulated.data
time <- simulation.obj$time
env <- simulation.obj$env
target <- simulation.obj$target

## Fit data to the following two models using CausalKinetiX:
## 1: dy = theta_1*x_1 + theta_2*x_2 + theta_3*x_1*x_10 (true model)
## 2: dy = theta_1*x_2 + theta_2*x_4 + theta_3*x_3*x_10 (wrong model)
ck.fit <- CausalKinetiX.modelranking(D, time, env, target,
                                     list(list(1, 2, c(1, 10)), list(2, 4, c(3, 10))))
print(ck.fit$scores)
```

`constrained.smoothspline` *constrained.smoothspline*

Description

Fit a smoothing spline with constraints on derivatives

Usage

```
constrained.smoothspline(y, times, pen.degree, constraint = "fixed",
  derivative.values = NA, initial.value = NA, times.new,
  num.folds = "leave-one-out", lambda = "optim")
```

Arguments

`y`	a vector of response variables.
`times`	a vector of time points at which y was measured, same length as y.
`pen.degree`	desired degree of the derivative in smoothing penalty.
`constraint`	one of 'none', 'fixed' or 'bounded' depending on whether the derivatives should not be constrained, fixed to a constant or bounded in an interval, respectively.
`derivative.values`	either a vector with the same length as y if contraint=='fixed' or a matrix with 2 columns conatining the lower and upper bounds on the derivatives if constraint=='bounded'.

initial.value	optional paramter that specifies an initial value of the spline incase it should be fixed.
times.new	optional vector of new time points at which the spline should be evaluated.
num.folds	either a numeric value of the number of folds or the string "leave-one-out" for a leave one out type cross-validation in determining the penalty parameter.
lambda	either a numeric value if the penalty parameter is fixed explicitely or one of the values 'optim' or 'grid.search' depending on the desired optimization procedure.

Details

For further details see the references.

Value

a list consisting of the following elements

smooth.vals	predicted values at points times
residual	residuals
smooth.vals.new	predicted values at time points times.new
smooth.deriv	predicted derivative values at points times
pen.par	penality parameter used for smoothing

Author(s)

Niklas Pfister, Stefan Bauer and Jonas Peters

References

Pfister, N., S. Bauer, J. Peters (2018). Identifying Causal Structure in Large-Scale Kinetic Systems ArXiv e-prints (arXiv:1810.11776).

Examples

```
## Example
x <- seq(0,4,length.out=200)
x.long <- seq(0,4,length.out=200)
y <- x^2+rnorm(200,0,2)
dy <- 2*x
dybdd <- cbind(dy-0.5,dy+0.5)
plot(x,y)

ptm <- proc.time()
fit <- constrained.smoothspline(y=y,
                                times=x,
                                pen.degree=2,
                                constraint="none",
                                derivative.values=NA,
                                times.new=x.long,
```

```
                                num.folds=5,
                                lambda="optim")
print(proc.time()-ptm)
fit2 <- smooth.spline(x,y)
lines(x.long,fit[[3]],col="blue")
lines(fit2, col="green")
lines(x.long, x.long^2, col="black")
```

`generate.data.hidden` *Hidden variable model*

Description

Generate sample data from the hidden variable model.

Usage

```
generate.data.hidden(env = rep(1, 10), L = 15,
  par.noise = list(noise.sd = 0.01, only.target.noise = TRUE, relativ =
  FALSE), intervention = "initial_blockreactions",
  intervention.par = 0.1, hidden = TRUE, ode.solver = "lsoda",
  seed = NA, silent = FALSE)
```

Arguments

`env`	integer vector of length n encoding to which experiment each repetition belongs.
`L`	number of time points for evaluation.
`par.noise`	list of parameters that specify the added noise. `noise.sd` specifies the standard deviation of noise, `only.target.noise` specifies whether to only add noise to target and `relative` specifies if the size of the noise should be relative to size of variable (if TRUE standard deviation is given by par.noise$noise.sd*(x(t)-x(t-1))).
`intervention`	string specifying type of intervention. Currently three type of interventions are implemented "initial" (only intervene on intial values), "blockreactions" (intervene by blocking random reactions) or "intial_blockreactions" (intervene on both initial values and blockreactions").
`intervention.par`	if intervention is either "blockreactions" or "initial_blockreactions", the rate constant k_7 is set to a noisy version of k_4. This parameter specifies size of this noise, more precisely k_7 is set to k_4 + `runif(1,-intervention.par, intervention.par)`.
`hidden`	boolean whether the variables H1 and H2 should be removed from output.
`ode.solver`	string specifying which ODE solver to use when solving ODE. Should be one of the methods from the `deSolve` package ("lsoda", "lsode", "lsodes", "lsodar", "vode", "daspk", "euler", "rk4", "ode23", "ode45", "radau", "bdf", "bdf_d", "adams", "impAdams", "impAdams_d", "iteration").
`seed`	random seed. Does not work if a "Detected blow-up" warning shows up.
`silent`	set to TRUE if no status output should be produced.

Details

For further details see the references.

Value

list consisting of the following elements

`simulated.data`	D-matrix of noisy data.
`time`	vector containing time points
`env`	vector specifying the experimental environment.
`simulated.model`	object returned by ODE solver.
`true.model`	vector specifying the target equation model.
`target`	target variable.

Author(s)

Niklas Pfister, Stefan Bauer and Jonas Peters

References

Pfister, N., S. Bauer, J. Peters (2018). Identifying Causal Structure in Large-Scale Kinetic Systems ArXiv e-prints (arXiv:1810.11776).

See Also

The functions `generate.data.maillard` and `generate.data.targetmodel` allow to simulate ODE data from two additional models.

Examples

```
simulation.obj <- generate.data.hidden(env=rep(1:5, 3),
                                       L=15,
                                       par.noise=list(noise.sd=0.02,
                                                      only.target.noise=FALSE,
                                                      relativ=TRUE),
                                       intervention="initial_blockreactions",
                                       intervention.par=0.1)

D <- simulation.obj$simulated.data
fulldata <- simulation.obj$simulated.model
time <- simulation.obj$time
plot(fulldata[[1]][,1], fulldata[[1]][,2], type= "l", lty=2,
     xlab="time", ylab="concentration")
points(time, D[1,1:length(time)], col="red", pch=19)
legend("topright", c("true trajectory", "observations"),
       col=c("black", "red"), lty=c(2, NA), pch=c(NA, 19))
```

`generate.data.maillard` *Maillard reaction*

Description

Generate sample data from the Maillard reaction as specified by Bio52 in the BioModel data base.

Usage

```
generate.data.maillard(target, env = rep(1, 10), L = 15,
  par.noise = list(noise.sd = 0.01, only.target.noise = FALSE, relativ =
  FALSE), intervention = "initial_blockreactions",
  ode.solver = "lsoda", seed = NA, silent = FALSE)
```

Arguments

`target`	specifies which species is used as a target, needs to be an integer between 1 and 11.
`env`	integer vector of length n encoding to which experiment each repetition belongs.
`L`	number of time points for evaluation.
`par.noise`	list of parameters that specify the added noise. `noise.sd` specifies the standard deviation of noise, `only.target.noise` specifies whether to only add noise to target and `relative` specifies if the size of the noise should be relative to size of variable (if TRUE standard deviation is given by par.noise$noise.sd*(x(t)-x(t-1))).
`intervention`	string specifying type of intervention. Currently three type of interventions are implemented "initial" (only intervene on intial values), "blockreactions" (intervene by blocking random reactions) or "intial_blockreactions" (intervene on both initial values and blockreactions").
`ode.solver`	specifies which ODE solver to use when solving ODE. Should be one of the methods from the `deSolve` package ("lsoda", "lsode", "lsodes", "lsodar", "vode", "daspk", "euler", "rk4", "ode23", "ode45", "radau", "bdf", "bdf_d", "adams", "impAdams", "impAdams_d", "iteration").
`seed`	random seed. Does not work if a "Detected blow-up" warning shows up.
`silent`	set to TRUE if no status output should be produced.

Details

For further details see the references.

Value

list consisting of the following elements

`simulated.data`	D-matrix of noisy data.
`time`	vector containing time points
`env`	vector specifying the experimental environment.
`simulated.model`	object returned by ODE solver.
`true.model`	vector specifying the target equation model.
`target`	target variable.

Author(s)

Niklas Pfister, Stefan Bauer and Jonas Peters

References

Pfister, N., S. Bauer, J. Peters (2018). Identifying Causal Structure in Large-Scale Kinetic Systems ArXiv e-prints (arXiv:1810.11776).

Brands C. and van Boekel M. (2002). Kinetic modeling of reactions in heated monosaccharide-casein systems. Journal of agricultural and food chemistry, 50(23):6725–6739.

See Also

The functions `generate.data.hidden` and `generate.data.targetmodel` allow to simulate ODE data from two additional models.

Examples

```
simulation.obj <- generate.data.maillard(target=1,
                                         env=rep(1:5, 3),
                                         L=15)

D <- simulation.obj$simulated.data
fulldata <- simulation.obj$simulated.model
time <- simulation.obj$time
plot(fulldata[[1]][,1], fulldata[[1]][,2], type= "l", lty=2,
     xlab="time", ylab="concentration")
points(time, D[1,1:length(time)], col="red", pch=19)
legend("topright", c("true trajectory", "observations"),
       col=c("black", "red"), lty=c(2, NA), pch=c(NA, 19))
```

`generate.data.targetmodel` *Target model based on predictor trajectories*

Description

Generate sample data from the target model based on predictor trajectories.

Usage

```
generate.data.targetmodel(env = rep(1, 10), noise.sd = 0.01, L = 15,
  d = 7, seed = NA)
```

Arguments

`env`	integer vector of length n encoding to which experiment each repetition belongs.
`noise.sd`	numerical value specifying the standard deviation of the noise.
`L`	number of time points for evaluation.
`d`	number of total variables (d-1 preditor variables).
`seed`	random seed. Does not work if a "Detected blow-up" warning shows up.

Details

For further details see the references.

Value

list consisting of the following elements

`simulated.data`	D-matrix of noisy data.
`time`	vector containing time points
`env`	vector specifying the experimental environment.
`true.model`	vector specifying the target equation model.
`target`	target variable.

Author(s)

Niklas Pfister, Stefan Bauer and Jonas Peters

References

Pfister, N., S. Bauer, J. Peters (2018). Identifying Causal Structure in Large-Scale Kinetic Systems ArXiv e-prints (arXiv:1810.11776).

See Also

The functions `generate.data.maillard` and `generate.data.hidden` allow to simulate ODE data from two additional models.

Examples

```
simulation.obj <- generate.data.targetmodel(env=rep(1:5, 3),
                                            L=15,
                                            d=5)

D <- simulation.obj$simulated.data
fulldata <- simulation.obj$simulated.model
time <- simulation.obj$time
plot(time, D[1,1:length(time)], col="red", pch=19)
legend("topright", c("observations"),
       col=c("red"), pch=c(19))
```

`ode.solver` *ode.solver*

Description

Solves a mass-action ODE for a target variable Y by using smooth approximations of the predictor variables X.

Usage

```
ode.solver(time_vec, initial_value, times, X, model, target, coefs,
  smooth.type = "smoothing.spline", reltol = 10^(-10),
  abstol = 10^(-16))
```

Arguments

`time_vec`	numeric vector. Specifices the points at which to evaluate the target trajectory.
`initial_value`	numeric value. Specifies the value of the target at initial time point.
`times`	numeric vector of length L. Specifices the time points at which the predictors where observed.
`X`	predictor matrix of dimension L x d. Each column corresponds to a different predictor observed at the time points `times`.
`model`	list of mass-action terms. Each element in the list consists of a vector of predictor variables which are multiplied to a single term in the mass-action equation.
`target`	integer specifing which variable is the target.
`coefs`	numeric vector. Specifies the parameter values for each term in `model`.
`smooth.type`	string. Specifies which type of smoothing to use. The following options exist: "smoothing.spline", "loess", "linear", "constant".
`reltol`	numeric. Relative tolarance used in CVODE.
`abstol`	numeric. Absolute tolarance used in CVODE.

Details

For further details see the references.

Value

object returned by CVODE.

Author(s)

Niklas Pfister, Stefan Bauer and Jonas Peters

References

Pfister, N., S. Bauer, J. Peters (2018). Identifying Causal Structure in Large-Scale Kinetic Systems ArXiv e-prints (arXiv:1810.11776).

Examples

```
## Generate data from Maillard reaction
simulation.obj <- generate.data.maillard(target=11,
                                         env=rep(1:5, each=5),
                                         L=20,
                                         par.noise=list(noise.sd=0.1,
                                                        only.target.noise=FALSE,
                                                        relativ=TRUE))
D <- simulation.obj$simulated.data
time <- simulation.obj$time

## Solve for Melanoidin
X <- do.call(cbind, split(as.vector(t(D[1:5,])), rep(1:11, each=length(unique(time)))))
times <- rep(unique(time), 5)
odefit <- ode.solver(time, 0, times, X, list(c(8)), 11, 0.12514)
plot(odefit[,1], odefit[,2], type="l")
points(times, X[,11])
```

Index