Package 'LZeroSpikeInference'

October 18, 2017

Type Package

Title Exact Spike Train Inference via L0 Optimization

Version 1.0.3

Description
An implementation of algorithms described in Jewell and Witten (2017) <arXiv:1703.08644>.

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 6.0.1

Suggests testthat

Imports graphics, stats

NeedsCompilation no

Author Sean Jewell [aut, cre]

Maintainer Sean Jewell <swjewell@uw.edu>

Repository CRAN

Date/Publication 2017-10-18 03:33:29 UTC

R topics documented:

`cv.estimateSpikes` *Cross-validate and optimize model parameters*

Description

Cross-validate and optimize model parameters

Usage

```
cv.estimateSpikes(dat, type = "ar1", gam = NULL, lambdas = NULL,
  nLambdas = 10, hardThreshold = TRUE)
```

Arguments

`dat`	fluorescence trace (a vector)
`type`	type of model, must be one of AR(1) 'ar1', or AR(1) with intercept 'intercept'
`gam`	a scalar value for the AR(1)/AR(1) + intercept decay parameter
`lambdas`	vector of tuning parameters to use in cross-validation
`nLambdas`	number of tuning parameters to estimate the model (grid of values is automatically produced)
`hardThreshold`	boolean specifying whether the calcium concentration must be non-negative (in the AR-1 problem)

Details

We perform cross-validation over a one-dimensional grid of λ values. For each value of λ in this grid, we solve the corresponding optimization problem, that is, one of

AR(1)-model: minimize_c1,...,cT 0.5 sum_t=1^T (y_t - c_t)^2 + lambda sum_t=2^T 1_c_t neq gamma c_t-1 for the global optimum, where y_t is the observed fluorescence at the tth timepoint.

If hardThreshold = T then the additional constraint c_t >= 0 is added to the optimzation problem above.

AR(1) with intercept: minimize_c1,...,cT,b1,...,bT 0.5 sum_t=1^T (y_t - c_t - b_t)^2 + lambda sum_t=2^T 1_c_t neq gamma c_t-1, b_t neq b_t-1 where the indicator variable 1_(A,B) equals 1 if the event A cup B holds, and equals zero otherwise.

on a training set using a candidate value for γ. Given the resulting set of changepoints, we solve a constrained optimization problem for γ. We then refit the optimization problem with the optimized value of γ, and then evaluate the mean squared error (MSE) on a hold-out set. Note that in the final output of the algorithm, we take the square root of the optimal value of γ in order to address the fact that the cross-validation scheme makes use of training and test sets consisting of alternately-spaced timesteps.

If there is a tuning parameter lambdaT in the path [lambdaMin, lambdaMax] that produces a fit with less than 1 spike per 10,000 timesteps the path is truncated to [lambdaMin, lambdaT] and a warning is produced.

See Algorithm 3 of Jewell and Witten (2017) <arXiv:1703.08644>

Value

A list of values corresponding to the 2-fold cross-validation:

`cvError` the MSE for each tuning parameter

`cvSE` the SE for the MSE for each tuning parameter

`lambdas` tuning parameters

`optimalGam` matrix of (optimized) parameters, rows correspond to tuning parameters, columns correspond to optimized parameter

`lambdaMin` tuning parameter that gives the smallest MSE

`lambda1SE` 1 SE tuning parameter

`indexMin` the index corresponding to lambdaMin

`index1SE` the index corresponding to lambda1SE

See Also

Estimate spikes: `estimateSpikes`, `print.estimatedSpikes`, `plot.estimatedSpikes`.

Cross validation: `cv.estimateSpikes`, `print.cvSpike`, `plot.cvSpike`.

Simulation: `simulateAR1`, `plot.simdata`.

Examples

```
# Not run
# sim <- simulateAR1(n = 500, gam = 0.998, poisMean = 0.009, sd = 0.05, seed = 1)
# plot(sim)

# AR(1) model
# outAR1 <- cv.estimateSpikes(sim$fl, type = "ar1")
# plot(outAR1)
# print(outAR1)
# fit <- estimateSpikes(sim$fl, gam = outAR1$optimalGam[outAR1$index1SE, 1],
# lambda = outAR1$lambda1SE, type = "ar1")
# plot(fit)
# print(fit)

# AR(1) + intercept model
# outAR1Intercept <- cv.estimateSpikes(sim$fl, type = "intercept",
# lambdas = seq(0.1, 5, length.out = 10))
# plot(outAR1Intercept)
# print(outAR1Intercept)
# fit <- estimateSpikes(sim$fl, gam = outAR1Intercept$optimalGam[outAR1Intercept$index1SE, 1],
# lambda = outAR1Intercept$lambda1SE, type = "intercept")
# plot(fit)
# print(fit)
```

`estimateSpikes` *Estimate spike train, underlying calcium concentration, and change-points based on fluorescence trace.*

Description

Estimate spike train, underlying calcium concentration, and changepoints based on fluorescence trace.

Usage

```
estimateSpikes(dat, gam, lambda, type = "ar1", calcFittedValues = TRUE,
  hardThreshold = FALSE)
```

Arguments

`dat`	fluorescence data
`gam`	a scalar value for the AR(1)/AR(1) + intercept decay parameter.
`lambda`	tuning parameter lambda
`type`	type of model, must be one of AR(1) 'ar1', AR(1) + intercept 'intercept'.
`calcFittedValues`	TRUE to calculate fitted values.
`hardThreshold`	boolean specifying whether the calcium concentration must be non-negative (in the AR-1 problem)

Details

This algorithm solves the optimization problems

AR(1)-model: minimize_c1,...,cT 0.5 sum_t=1^T (y_t - c_t)^2 + lambda sum_t=2^T 1_c_t neq gamma c_t-1 for the global optimum, where y_t is the observed fluorescence at the tth timepoint.

If hardThreshold = T then the additional constraint c_t >= 0 is added to the optimzation problem above.

AR(1) with intercept: minimize_c1,...,cT,b1,...,bT 0.5 sum_t=1^T (y_t - c_t - b_t)^2 + lambda sum_t=2^T 1_c_t neq gamma c_t-1, b_t neq b_t-1 where the indicator variable 1_(A,B) equals 1 if the event A cup B holds, and equals zero otherwise.

See Jewell and Witten (2017) <arXiv:1703.08644>, section 2 and 5.

Note that "changePts" and "spikes" differ by one index due to a quirk of the conventions used in the changepoint literature and the definition of a spike.

Value

Returns a list with elements:

`changePts` the set of changepoints

`spikes` the set of spikes

`fittedValues` estimated calcium concentration

See Also

Estimate spikes: estimateSpikes, print.estimatedSpikes, plot.estimatedSpikes.

Cross validation: cv.estimateSpikes, print.cvSpike, plot.cvSpike.

Simulation: simulateAR1, plot.simdata.

Examples

```
sim <- simulateAR1(n = 500, gam = 0.998, poisMean = 0.009, sd = 0.05, seed = 1)
plot(sim)

# AR(1) model

fit <- estimateSpikes(sim$fl, gam = 0.998, lambda = 1, type = "ar1")
plot(fit)
print(fit)

# AR(1) + intercept model
fit <- estimateSpikes(sim$fl, gam = 0.998, lambda = 1, type = "intercept")
plot(fit)
print(fit)
```

LZeroSpikeInference *LZeroSpikeInference: LZeroSpikeInference: A package for estimating spike times from calcium imaging data using an L0 penalty*

Description

This package implements an algorithm for deconvolving calcium imaging data for a single neuron in order to estimate the times at which the neuron spikes.

Details

This package implements an algorithm for deconvolving calcium imaging data for a single neuron in order to estimate the times at which the neuron spikes. This algorithm solves the optimization problems

AR(1)-model: minimize_c1,...,cT 0.5 sum_t=1^T (y_t - c_t)^2 + lambda sum_t=2^T 1_c_t neq gamma c_t-1 for the global optimum, where y_t is the observed fluorescence at the tth timepoint.

If hardThreshold = T then the additional constraint c_t >= 0 is added to the optimzation problem above.

AR(1) with intercept: minimize_c1,...,cT,b1,...,bT 0.5 sum_t=1^T (y_t - c_t - b_t)^2 + lambda sum_t=2^T 1_c_t neq gamma c_t-1, b_t neq b_t-1 where the indicator variable 1_(A,B) equals 1 if the event A cup B holds, and equals zero otherwise.

See Jewell and Witten (2017) <arXiv:1703.08644>

See Also

Estimate spikes: estimateSpikes, print.estimatedSpikes, plot.estimatedSpikes.

Cross validation: cv.estimateSpikes, print.cvSpike, plot.cvSpike.

Simulation: simulateAR1, plot.simdata.

Examples

```
# To reproduce Figure 1 of Jewell and Witten (2017) <arXiv:1703.08644>

sampleData <- simulateAR1(n = 500, gam = 0.998, poisMean = 0.009, sd = 0.15, seed = 8)
fit <- estimateSpikes(sampleData$fl, gam = 0.998, lambda = 8, type = "ar1")
plot(fit)
```

---

plot.cvSpike *Plot mean squared error vs. tuning parameter from the cross-validation output*

Description

Plot mean squared error vs. tuning parameter from the cross-validation output

Usage

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

Arguments

x	output from cross validation procedure
...	arguments to be passed to methods

---

plot.estimatedSpikes *Plot the solution to an L0 segmentation problem*

Description

Plot the solution to an L0 segmentation problem

Usage

```
## S3 method for class 'estimatedSpikes'
plot(x, xlims = NULL, ...)
```

Arguments

x	output from running estimatedSpikes
xlims	optional parameter to specify the x-axis limits
...	arguments to be passed to methods

See Also

Estimate spikes: estimateSpikes, print.estimatedSpikes, plot.estimatedSpikes.

Cross validation: cv.estimateSpikes, print.cvSpike, plot.cvSpike.

Simulation: simulateAR1, plot.simdata.

---

plot.simdata *Plot simulated data*

Description

Plot simulated data

Usage

```
## S3 method for class 'simdata'
plot(x, xlims = NULL, ...)
```

Arguments

x	output data from simulateAR1
xlims	optional parameter to specify the x-axis limits
...	arguments to be passed to methods

Value

Plot with simulated fluorescence (dark grey circles), calcium concentration (dark green line) and spikes (dark green tick marks on x-axis)

See Also

Estimate spikes: estimateSpikes, print.estimatedSpikes, plot.estimatedSpikes.

Cross validation: cv.estimateSpikes, print.cvSpike, plot.cvSpike.

Simulation: simulateAR1, plot.simdata.

Examples

```
sim <- simulateAR1(n = 500, gam = 0.998, poisMean = 0.009, sd = 0.05, seed = 1)
plot(sim)
```

print.cvSpike *Print CV results*

Description

Print CV results

Usage

```
## S3 method for class 'cvSpike'
print(x, ...)
```

Arguments

x	output from CV
...	arguments to be passed to methods

print.estimatedSpikes *Print estimated spikes*

Description

Print estimated spikes

Usage

```
## S3 method for class 'estimatedSpikes'
print(x, ...)
```

Arguments

x	estimated spikes
...	arguments to be passed to methods

print.simdata *Print simulated data*

Description

Print simulated data

Usage

```
## S3 method for class 'simdata'
print(x, ...)
```

Arguments

x	simulated data
...	arguments to be passed to methods

simulateAR1 *Simulate fluorescence trace based on simple AR(1) generative model*

Description

Simulate fluorescence trace based on simple AR(1) generative model

Usage

```
simulateAR1(n, gam, poisMean, sd, seed)
```

Arguments

n	number of timesteps
gam	AR(1) decay rate
poisMean	mean for Poisson distributed spikes
sd	standard deviation
seed	random seed

Details

Simulate fluorescence trace based on simple AR(1) generative model

y_t = c_t + eps, eps ~ N(0, sd)

c_t = gam * c_t-1 + s_t

s_t ~ Pois(poisMean)

Value

spikes, fluorescence, and calcium concentration

See Also

Estimate spikes: `estimateSpikes`, `print.estimatedSpikes`, `plot.estimatedSpikes`.

Cross validation: `cv.estimateSpikes`, `print.cvSpike`, `plot.cvSpike`.

Simulation: `simulateAR1`, `plot.simdata`.

Examples

```
sim <- simulateAR1(n = 500, gam = 0.998, poisMean = 0.009, sd = 0.05, seed = 1)
plot(sim)
```

Index