Package ‘viking’

October 12, 2022

Title State-Space Models Inference by Kalman or Viking

Version 1.0.0

Maintainer Joseph de Vilmarest <joseph.de-vilmarest@vikingconseil.fr>

Description Inference methods for state-space models, relying on the Kalman Filter or on Viking (Variational Bayesian VarIance tracKING). See J. de Vilmarest (2022) <https://tel.archives-ouvertes.fr/tel-03716104/>.

License LGPL-3

Suggests RColorBrewer, mvtnorm

Encoding UTF-8

RoxygenNote 7.2.1

NeedsCompilation no

Author Joseph de Vilmarest [aut, cre]
(<https://orcid.org/0000-0002-0634-8484>)

Repository CRAN

Date/Publication 2022-10-07 15:00:02 UTC

R topics documented:

`expectation_maximization` *Expectation-Maximization*

Description

`expectation_maximization` is a method to choose hyper-parameters of the linear Gaussian State-Space Model with time-invariant variances relying on the Expectation-Maximization algorithm.

Usage

```
expectation_maximization(
  X,
  y,
  n_iter,
  Q_init,
  sig_init = 1,
  verbose = 1000,
  lambda = 10^-9,
  mode_diag = FALSE,
  p1 = 0
)
```

Arguments

`X`	explanatory variables
`y`	time series
`n_iter`	number of iterations of the EM algorithm
`Q_init`	initial covariance matrix on the state noise
`sig_init`	(optional, default `1`) initial value of the standard deviation of the observation noise
`verbose`	(optional, default `1000`) frequency for prints
`lambda`	(optional, default `10^-9`) regularization parameter to avoid singularity
`mode_diag`	(optional, default `FALSE`) if `TRUE` then we restrict the search to diagonal matrices for `Q`
`p1`	(optional, default `0`) deterministic value of `P1 = p1 I`

Details

E-step is realized through recursive Kalman formulae (filtering then smoothing).
M-step is the maximization of the expected complete likelihood with respect to the hyper-parameters.
We only have the guarantee of convergence to a LOCAL optimum. We fix P1 = p1 I (by default p1 = 0). We optimize theta1, sig, Q.

Value

a list containing values for `P,theta,sig,Q`, and two vectors `DIFF`, `LOGLIK` assessing the convergence of the algorithm.

Examples

```
set.seed(1)
### Simulate data
n <- 100
d <- 5
Q <- diag(c(0,0,0.25,0.25,0.25))
sig <- 1

X <- cbind(matrix(rnorm((d-1)*n,sd=1),n,d-1),1)
theta <- matrix(rnorm(d), d, 1)
theta_arr <- matrix(0, n, d)
for (t in 1:n) {
  theta_arr[t,] <- theta
  theta <- theta + matrix(mvtnorm::rmvnorm(1,matrix(0,d,1),Q),d,1)
}
y <- rowSums(X * theta_arr) + rnorm(n, sd=sig)

l <- viking::expectation_maximization(X, y, 50, diag(d), verbose=10)
print(l$Q)
print(l$sig)
```

`iterative_grid_search` *Iterative Grid Search*

Description

`iterative_grid_search` is an iterative method to choose hyper-parameters of the linear Gaussian State-Space Model with time-invariant variances.

Usage

```
iterative_grid_search(
  X,
  y,
  q_list,
  Q_init = NULL,
  max_iter = 0,
  delay = 1,
  use = NULL,
  restrict = NULL,
  mode = "gaussian",
  p1 = 0,
  ncores = 1,
```

```
  train_theta1 = NULL,
  train_Q = NULL,
  verbose = TRUE
)
```

Arguments

`X`	the explanatory variables
`y`	the observations
`q_list`	the possible values of `diag(Q) / sig^2`
`Q_init`	(default `NULL`) initial value of `Q / sig^2`, if `NULL` it is set to 0
`max_iter`	(optional 0) maximal number of iterations. If 0 then optimization is done as long as we can improve the log-likelihood
`delay`	(optional, default 1) to predict `y[t]` we have access to `y[t-delay]`
`use`	(optional, default `NULL`) the availability variable
`restrict`	(optional, default `NULL`) if not `NULL` it allows to specify the indices of the diagonal coefficient to optimize
`mode`	(optional, default `gaussian`)
`p1`	(optional, default `0`) coefficient for `P1/sig^2` = `p1 I`
`ncores`	(optional, default `1`) number of available cores for computation
`train_theta1`	(optional, default `NULL`) training set for estimation of `theta1`
`train_Q`	(optional, default `NULL`) time steps on which the log-likelihood is computed
`verbose`	(optional, default `TRUE`) whether to print intermediate progress

Details

We restrict ourselves to a diagonal matrix `Q` and we optimize `Q / sig^2` on a grid. Each diagonal coefficient is assumed to belong to a pre-defined `q_list`.
We maximize the log-likelihood on that region of search in an iterative fashion. At each step, we change the diagonal coefficient improving the most the log-likelihood. We stop when there is no possible improvement. This doesn't guarantee an optimal point on the grid, but the computational time is much lower.

Value

a list containing values for `P,theta,sig,Q`, as well as `LOGLIK`, the evolution of the log-likelihood during the search.

Examples

```
set.seed(1)
### Simulate data
n <- 100
d <- 5
Q <- diag(c(0,0,0.25,0.25,0.25))
sig <- 1
```

```
X <- cbind(matrix(rnorm((d-1)*n,sd=1),n,d-1),1)
theta <- matrix(rnorm(d), d, 1)
theta_arr <- matrix(0, n, d)
for (t in 1:n) {
  theta_arr[t,] <- theta
  theta <- theta + matrix(mvtnorm::rmvnorm(1,matrix(0,d,1),Q),d,1)
}
y <- rowSums(X * theta_arr) + rnorm(n, sd=sig)

l <- viking::iterative_grid_search(X, y, seq(0,1,0.25))
print(l$Q)
print(l$sig)
```

`kalman_filtering` *Kalman Filtering*

Description

Compute the filtered estimation of the parameters `theta` and `P`.

Usage

```
kalman_filtering(X, y, theta1, P1, Q = 0, sig = 1)
```

Arguments

`X`	the explanatory variables
`y`	the time series
`theta1`	initial `theta`
`P1`	initial `P`
`Q`	(optional, default `0`) covariance matrix of the state noise
`sig`	(optional, default `1`) variance of the spate noise

Value

a list containing `theta_arr` and `P_arr`, the filtered estimation of the parameters `theta` and `P`.

`kalman_smoothing` *Kalman Smoothing*

Description

Compute the smoothed estimation of the parameters `theta` and `P`.

Usage

```
kalman_smoothing(X, y, theta1, P1, Q = 0, sig = 1)
```

Arguments

`X`	the explanatory variables
`y`	the time series
`theta1`	initial `theta`
`P1`	initial `P`
`Q`	(optional, default `0`) covariance matrix of the state noise
`sig`	(optional, default `1`) variance of the spate noise

Value

a list containing `theta_arr` and `P_arr`, the smoothed estimation of the parameters `theta` and `P`.

`loglik` *Log-likelihood*

Description

`loglik` computes the log-likelihood of a state-space model of specified `Q/sig^2`, `P1/sig^2`, `theta1`.

Usage

```
loglik(X, y, Qstar, use, p1, train_theta1, train_Q, mode = "gaussian")
```

Arguments

`X`	explanatory variables
`y`	time series
`Qstar`	the ratio `Q/sig^2`
`use`	the availability variable
`p1`	coefficient for `P1/sig^2` = `p1 I`
`train_theta1`	training set for estimation of `theta1`
`train_Q`	time steps on which the log-likelihood is computed
`mode`	(optional, default `gaussian`)

Value

a numeric value for the log-likelihood.

---

plot.statespace *Plot a statespace object*

Description

`plot.statespace` displays different graphs expressing the behavior of the state-space model:
1. Evolution of the Bias: rolling version of the error of the model.
2. Evolution of the RMSE: root-mean-square-error computed on a rolling window.
3. State Evolution: time-varying state coefficients, subtracted of the initial state vector.
4. Normal Q-Q Plot: we check if the observation follows the Gaussian distribution of estimated mean and variance. To that end, we display a Q-Q plot of the residual divided by the estimated standard deviation, against the standard normal distribution.

Usage

```
## S3 method for class 'statespace'
plot(x, pause = FALSE, window_size = 7, date = NULL, sel = NULL, ...)
```

Arguments

`x`	the statespace object.
`pause`	(default `FALSE`) if set to `FALSE` then the plots are displayed on a single page, otherwise a new page is created for each plot.
`window_size`	(default `7`) the window size of the rolling mean computed on the error to display the bias, and on the mean squared error to display a rolling RMSE.
`date`	(default `NULL`) defines the values for the x-axis.
`sel`	(default `NULL`) defines a subset of the data on which we zoom. For instance one can display the evolution of the SSM on a test set and not the whole data set.
`...`	additional parameters

Value

No return value, called to display plots.

predict.statespace *Predict using a statespace object*

Description

predict.statespace makes a prediction for a statespace object, in the offline or online setting.

Usage

```
## S3 method for class 'statespace'
predict(
  object,
  newX,
  newy = NULL,
  online = TRUE,
  compute_smooth = FALSE,
  type = c("mean", "proba", "model"),
  ...
)
```

Arguments

object	the statespace object
newX	the design matrix in the prediction set
newy	(default NULL) the variable of interest in the prediction set. If specified it allows to use the state-space model in the online setting. Otherwise the prediction is offline.
online	(default TRUE) specifies if the prediction is made online, that is if the observation at time t-1 is used to update the model before predicting at time t.
compute_smooth	(default FALSE) specifies if Kalman Smoothing is also computed.
type	type of prediction. Can be either **mean** return the mean forecast. **proba** return a probabilistic forecast (list containing estimation of the mean and standard deviation). **model** return the updated statespace object (containing also the forecasts).
...	additional parameters

Value

Depending on the type specified, the result is
- a vector of mean forecast if type='mean' - a list of two vectors, mean forecast and standard deviations if type='proba' - a statespace object if type='model'

`select_Kalman_variances` *Select time-invariant variances of a State-Space Model*

Description

`select_Kalman_variances` is a function to choose hyper-parameters of the linear Gaussian State-Space Model with time-invariant variances. It relies on the functions `iterative_grid_search` and `expectation_maximization`.

Usage

```
select_Kalman_variances(ssm, X, y, method = "igd", ...)
```

Arguments

`ssm`	the statespace object
`X`	explanatory variables
`y`	time series
`method`	(optional, default `'igd'`) it can be either `'igd'` `iterative_grid_search` is called `'em'` `expectation_maximization` is called
`...`	additional parameters

Value

a new statespace object with new values in `kalman_params`

`statespace` *Design a State-Space Model*

Description

The function `statespace` builds a state-space model, with known or unknown variances. By default, this function builds a state-space model in the static setting, with a constant state (zero state noise covariance matrix) and constant observation noise variance.

Usage

```
statespace(X, y, kalman_params = NULL, viking_params = NULL, ...)
```

Arguments

`X`	design matrix.
`y`	variable of interest.
`kalman_params`	(default `NULL`) list containing initial values for `theta,P` as well as the variances (`Q,sig`). If it is not specified, the state-space model is constructed in the static setting (`theta=0, P=I, Q=0, sig=1`).
`viking_params`	(default `NULL`) list of parameters for the Viking algorithm.
`...`	additional parameters

Value

a statespace object.

Examples

```
set.seed(1)
### Simulate data
n <- 1000
d <- 5
Q <- diag(c(0,0,0.25,0.25,0.25))
sig <- 1

X <- cbind(matrix(rnorm((d-1)*n,sd=1),n,d-1),1)
theta <- matrix(rnorm(d), d, 1)
theta_arr <- matrix(0, n, d)
for (t in 1:n) {
  theta_arr[t,] <- theta
  theta <- theta + matrix(mvtnorm::rmvnorm(1,matrix(0,d,1),Q),d,1)
}
y <- rowSums(X * theta_arr) + rnorm(n, sd=sig)

####################
### Kalman Filter
# Default Static Setting
ssm <- viking::statespace(X, y)
plot(ssm)

# Known variances
ssm <- viking::statespace(X, y, kalman_params = list(Q=Q, sig=sig))
plot(ssm)
```

`viking` *Viking: Variational bayesIan variance tracKING*

Description

`viking` is the state-space estimation realized by Viking, generalizing the Kalman Filter to variance uncertainty.

Usage

```
viking(
  X,
  y,
  theta0,
  P0,
  hata0,
  s0,
  hatb0,
  Sigma0,
  n_iter = 2,
  mc = 10,
  rho_a = 0,
  rho_b = 0,
  learn_sigma = TRUE,
  learn_Q = TRUE,
  K = NULL,
  mode = "diagonal",
  thresh = TRUE,
  phi = logt,
  phi1 = logt1,
  phi2 = logt2
)
```

Arguments

`X`	the explanatory variables
`y`	the time series
`theta0`	initial `theta`
`P0`	initial `P`
`hata0`	initial `hata`
`s0`	initial `s`
`hatb0`	initial `hatb`
`Sigma0`	initial `Sigma`
`n_iter`	(optional, default `2`) number of alternate steps
`mc`	(optional, default `10`) number of Monte-Carlo samples
`rho_a`	(optional, default `0`) learning rate of `a`
`rho_b`	(optional, default `0`) learning rate of `b`
`learn_sigma`	(optional, default `TRUE`) asserts the estimation of `a`
`learn_Q`	(optional, default `TRUE`) asserts the estimation of `b`
`K`	(optional, default `NULL`) if not `NULL` then it is a multiplicative factor of the state in the state update
`mode`	(optional, default `'diagonal'`)

thresh	(optional, default TRUE)
phi	(optional, default logt)
phi1	(optional, default logt1)
phi2	(optional, default logt2)

Value

a list composed of the evolving value of all the parameters: theta_arr, P_arr, q_arr, hata_arr, s_arr, hatb_arr, Sigma_arr

References

J. de Vilmarest, O. Wintenberger (2021), Viking: Variational Bayesian Variance Tracking. <arXiv:2104.10777>

Index