Package ‘DLSSM’

May 22, 2025

Type Package

Title Dynamic Logistic State Space Prediction Model

Version 1.1.1

Maintainer Jiakun Jiang <jiakunj@bnu.edu.cn>

Description Implements the dynamic logistic state space model for binary outcome data proposed by Jiang et al. (2021) <doi:10.1111/biom.13593>.
It provides a computationally efficient way to update the prediction whenever new data becomes available.
It allows for both time-varying and time-invariant coefficients, and use cubic smoothing splines to model varying coefficients.
The smoothing parameters are objectively chosen by maximum likelihood. The model is updated using batch data accumulated at pre-specified time intervals.

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.0

Imports Matrix

Depends R (>= 3.10)

Suggests knitr, rmarkdown, testthat (>= 3.0.0), withr

VignetteBuilder knitr

Config/testthat/edition 3

SystemRequirements Intel MKL (optional for enhanced performance)

NeedsCompilation no

Author Jiakun Jiang [aut, cre],
Wei Yang [aut],
Wensheng Guo [aut]

Repository CRAN

Date/Publication 2025-05-22 05:10:16 UTC

Contents

`Batched` *Combine data into Batched data*

Description

The time domain of observation will first be standardized into [0,1]. Then [0,1] will be divided into S equally spaced intervals as described in Jiang et al.(2021, Biometrics). Then those intervals slice the dataset to S batches of data.

Usage

```
Batched(formula, data, time, S)
```

Arguments

`formula`	An object of class "formula" (or one that can be coerced to that class): a symbolic description of response and covariates in the model.
`data`	Dataset matrix containing the observations (one rows is a sample).
`time`	The time variable in the dataset. The varying coefficient functions are assumed to be smooth functions of this variable.
`S`	Number of batches

Value

`batched`	List of batched data, the element of list is matrix with each row representing a sample
`gap.len`	interval length 1/S

Author(s)

Jiakun Jiang, Wei Yang, Wensheng Guo

`car.insur` *Dataset contains information of full comprehensive Australian automobile insurance policies between years 2004 and 2005 A dataset containing the claim and three attributes of 67,856 policies*

Description

Dataset contains information of full comprehensive Australian automobile insurance policies between years 2004 and 2005 A dataset containing the claim and three attributes of 67,856 policies

Usage

```
car.insur
```

Format

A data frame with 67856 rows and 4 columns

y Binary vaiable with 0 denote a policy with no claim, and 1 denote a claim policy.

gender gender of deriver

age age of deriver

exposure period from the date of insured to the investigation, with a maximum of one year

References

De Jong P et al. (2008). “Generalized linear models for insurance data.” Cambridge Books.

Examples

```
data(car.insur)
```

`DLSSM` *Combine model training and validation in a integrated function*

Description

This combine model training and validation in a integrated automatic function DLSSM().

Usage

```
DLSSM(data.batched, S0, vary.effects, autotune = TRUE, Lambda = NULL, K)
```

Arguments

data.batched	A object generated by function Data.batched()
S0	Number of batches of data to be used as training dataset
vary.effects	The names of variables in the dataset assumed to have a time-varying regression effect on the outcome.
autotune	T/F indicates whether or not the automatic tuning procedure desribed in Jiakun et al. (2021) should be applied. Default is true.
Lambda	Specify smoothing parameters if autotune=F
K	Number of steps for ahead prediction

Value

Lambda:	smoothing parameters
Smooth:	smoothed state vector
Smooth.var:	covariance of smoothed state vector in Smooth.

Author(s)

Jiakun Jiang, Wei Yang and Wensheng Guo

Examples

```
set.seed(321)
n=8000
beta0=function(t)   0.1*t-1
beta1=function(t)  cos(2*pi*t)
beta2=function(t)  sin(2*pi*t)
alph1=alph2=1
x=matrix(runif(n*4,min=-4,max=4),nrow=n,ncol=4)
t=sort(runif(n))
coef=cbind(beta0(t),beta1(t),beta2(t),rep(alph1,n),rep(alph2,n))
covar=cbind(rep(1,n),x)
linear=apply(coef*covar,1,sum)
prob=exp(linear)/(1+exp(linear))
y=as.numeric(runif(n)<prob)
sim.data=cbind(y,x,t)
colnames(sim.data)=c("y","x1","x2","x3","x4","t")
formula = y~x1+x2+x3+x4
# Divide the time domain [0,1] into S=100 equally spaced intervals
S=100
S0=75
data.batched=Batched(formula, data=sim.data, time="t", S)
```

```
# Take first S0=75 batches as training data, remaining S-S0=25 batches of data as validation data.
 fit1=DLSSM(data.batched, S0, vary.effects=c("x1","x2"), autotune=TRUE, Lambda=NULL, K=1)
 DLSSM.plot(fit1)
 fit2=DLSSM(data.batched, S0, vary.effects=c("x1","x2"), autotune=TRUE, Lambda=NULL, K=2)
 DLSSM.plot(fit2)
```

`DLSSM.init` *Initial model fitting*

Description

This function is for tuning smoothing parameters using training data. The likelihood was calculated by Kalman Filter and maximized to estimate the smoothing parameters. For the given smoothing parameters, the model coefficients can be efficiently estimated using a Kalman filtering algorithm.

Usage

```
DLSSM.init(data.batched, S0, vary.effects, autotune = TRUE, Lambda = NULL)
```

Arguments

`data.batched`	A object generated by function Data.batched()
`S0`	Number of batches of data to be used as training dataset
`vary.effects`	The names of variables in the dataset assumed to have a time-varying regression effect on the outcome.
`autotune`	T/F indicates whether or not the automatic tuning procedure described in Jiang et al. (2021) should be applied. Default is true.
`Lambda`	Specify smoothing parameters if autotune=F

Value

`Lambda:`	smoothing parameters
`Smooth:`	smoothed state vector
`Smooth.var:`	covariance of smoothed state vector in Smooth.

Author(s)

Jiakun Jiang, Wei Yang and Wensheng Guo

Examples

```
set.seed(321)
n=8000
beta0=function(t)   0.1*t-1
beta1=function(t)  cos(2*pi*t)
beta2=function(t)  sin(2*pi*t)
alph1=alph2=1
x=matrix(runif(n*4,min=-4,max=4),nrow=n,ncol=4)
t=sort(runif(n))
coef=cbind(beta0(t),beta1(t),beta2(t),rep(alph1,n),rep(alph2,n))
covar=cbind(rep(1,n),x)
linear=apply(coef*covar,1,sum)
prob=exp(linear)/(1+exp(linear))
y=as.numeric(runif(n)<prob)
sim.data=cbind(y,x,t)
colnames(sim.data)=c("y","x1","x2","x3","x4","t")
formula = y~x1+x2+x3+x4
# Divide the time domain [0,1] into S=100 equally spaced intervals
S=100
S0=75
data.batched=Batched(formula, data=sim.data, time="t", S)

# using first 75 batches as training dataset to tune smoothing parameters
fit0=DLSSM.init(data.batched, S0, vary.effects=c("x1","x2"))
fit0$Lambda
DLSSM.plot(fit0)
```

`DLSSM.plot` *Plot coefficients*

Description

Plot smoothed coefficients in the training part and predicted coefficients in validation part, the two parts are divided by vertical dash line.

Usage

```
DLSSM.plot(fit)
```

Arguments

`fit`	fitted object

Details

If argument "fit" is an initial fitted model then only smoothed coefficients part are plotted.

Value

Figures

Author(s)

Jiakun Jiang, Wei Yang and Wensheng Guo

`DLSSM.valid` *Dynamical prediction on validation dataset*

Description

After we have fitted initial model, we can do validation. It is iteratively doing K-steps ahead prediction and model updating (filtering) when a new batch of data becomes available. The validation include K-steps ahead prediction of state vector and probabilities on validation interval.

Usage

```
DLSSM.valid(fit0, data.batched, K)
```

Arguments

`fit0`	Initial fitted model
`data.batched`	Batched dataset generated by function Batched()
`K`	Number of steps for ahead prediction

Details

The argument fit could be object of DLSSM or DLSSM.init.

Value

`pred.K:`	K-steps ahead predicted coefficients
`pred.var.K:`	covariance of K-steps ahead predicted coefficients
`pred.prob.K:`	K-steps ahead predicted probabilities

Author(s)

Jiakun Jiang, Wei Yang and Wensheng Guo

Examples

```
set.seed(321)
n=8000
beta0=function(t)   0.1*t-1
beta1=function(t)  cos(2*pi*t)
beta2=function(t)  sin(2*pi*t)
alph1=alph2=1
x=matrix(runif(n*4,min=-4,max=4),nrow=n,ncol=4)
t=sort(runif(n))
coef=cbind(beta0(t),beta1(t),beta2(t),rep(alph1,n),rep(alph2,n))
covar=cbind(rep(1,n),x)
linear=apply(coef*covar,1,sum)
prob=exp(linear)/(1+exp(linear))
y=as.numeric(runif(n)<prob)
sim.data=cbind(y,x,t)
colnames(sim.data)=c("y","x1","x2","x3","x4","t")
formula = y~x1+x2+x3+x4
# Divide the time domain [0,1] into S=100 equally spaced intervals
S=100
S0=75
data.batched=Batched(formula, data=sim.data, time="t", S)

# using first 75 batches as training dataset to tune smoothing parameters
fit0=DLSSM.init(data.batched, S0, vary.effects=c("x1","x2"))
fit0$Lambda

#After initial model fitting on training data, we move to dynamic prediction
 fit=DLSSM.valid(fit0, data.batched, K=1)
 DLSSM.plot(fit)
```

Index