Package ‘concrete’

September 10, 2023

Type Package
Title Continuous-Time Competing Risks Estimation using Targeted Minimum Loss-Based Estimation (TMLE)
Version 1.0.5
Date 2023-09-07
Author David Chen [aut, cre] (<https://orcid.org/0000-0002-9413-8152>)
Maintainer David Chen <david.chen49@berkeley.edu>
Description One-step continuous-time Targeted Minimum Loss-Based Estimation (TMLE) for outcome-specific absolute risk estimands in right-censored survival settings with or without competing risks, implementing the methodology described in Rytgaard et al. (2023) <doi:10.1111/biom.13856> and Rytgaard and van der Laan (2023) <doi:10.1007/s10985-022-09576-2>. Currently 'concrete' can be used to estimate the effects of static or dynamic interventions on binary treatments given at baseline, cross-validated initial estimation of treatment propensity is done using the 'SuperLearner' package, and initial estimation of conditional hazards is done using ensembles of Cox regressions from the 'survival' package or Coxnet from the 'glmnet' package.
License GPL (>= 3)
Imports data.table, survival, zoo, origami, SuperLearner, nleqslv, MASS, Rcpp (>= 1.0.11)
Depends R (>= 3.5.0)
LinkingTo Rcpp, RcppArmadillo
Encoding UTF-8
RoxygenNote 7.2.3
Suggests nnls, xgboost, glmnet, ranger, ggplot2, testthat (>= 3.0.0), knitr, rmarkdown
Config/testthat/edition 3
BugReports https://github.com/imbroglio-dc/concrete/issues
VignetteBuilder knitr
NeedsCompilation yes
One-step continuous-time Targeted Minimum Loss-Based Estimator (TMLE) for outcome-specific absolute risk estimands in right-censored survival settings with or without competing risks

Implements the methodology described in Rytgaard et al. (2023) <doi:10.1111/biom.13856> and Rytgaard and van der Laan (2023) <doi:10.1007/s10985-022-09576-2>. Currently can be used to estimate the effects of static or dynamic interventions on binary treatments given at baseline, cross-validated initial estimation of treatment propensity is done using the 'SuperLearner' package, and initial estimation of conditional hazards is done using ensembles of Cox regressions from the 'survival' package or Coxnet from the 'glmnet' package.

Details

formatArguments() many check...(), format...() functions getInitialEstimates() getPropScores() getHazEstimates() getEIC() getIC() doTMLEUpdate() getOutput()

Author(s)

David Chen, <david.chen49@berkeley.edu> Maintainer: David Chen <david.chen49@berkeley.edu>

References


See Also

SuperLearner coxph glmnet
Examples

library(concrete)
library(data.table)
set.seed(12345)

data <- as.data.table(survival::pbc)
data <- data[!is.na(trt), ][, trt := trt - 1]
data <- data[, c("time", "status", "trt", "age", "sex", "albumin")]

ConcreteArgs <- formatArguments(DataTable = data,
                               EventTime = "time",
                               EventType = "status",
                               Treatment = "trt",
                               Intervention = 0:1,
                               TargetTime = 1500,
                               TargetEvent = 1:2,
                               MaxUpdateIter = 250,
                               CVArg = list(V = 10),
                               Verbose = FALSE)

ConcreteEst <- doConcrete(ConcreteArgs)
ConcreteOut <- getOutput(ConcreteEst)

## Joint Intervention

data <- data[, trt2 := sample(0:1, .N, replace = TRUE, prob = c(0.3, .7))]
Intervention <- makeITT("A1" = data.frame(trt = rep_len(1, nrow(data)),
                                          trt2 = rep_len(1, nrow(data))),
                           "A0" = data.frame(trt = rep_len(0, nrow(data)),
                                           trt2 = rep_len(0, nrow(data))))

ConcreteArgs <- formatArguments(DataTable = data,
                               EventTime = "time",
                               EventType = "status",
                               Treatment = c("trt", "trt2"),
                               Intervention = Intervention,
                               TargetTime = 2000,
                               TargetEvent = 1:2,
                               MaxUpdateIter = 250,
                               CVArg = list(V = 10),
                               Verbose = FALSE)

ConcreteEst <- doConcrete(ConcreteArgs)
ConcreteOut <- getOutput(ConcreteEst)

Description

doConcrete
doConcrete

Usage

doConcrete(ConcreteArgs)

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

## S3 method for class 'ConcreteEst'
plot(x, convergence = FALSE, gweights = TRUE, ask = FALSE, ...)

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

Arguments

ConcreteArgs "ConcreteArgs" object: output of formatArguments()
x a ConcreteOut object
... additional arguments to be passed into print methods
convergence logical: plot the PnEIC norms for each TMLE small update step
gweights logical: plot the densities of the intervention-related nuisance weights for each intervention
ask logical: whether or not to prompt for user input before displaying plots

Value

object with s3 class "ConcreteEst"

Functions

• print(ConcreteEst): print.ConcreteEst print method for "ConcreteEst" class
• plot(ConcreteEst): plot.ConcreteEst plot method for "ConcreteEst" class
• print(ConcreteOut): print.ConcreteOut print method for "ConcreteOut" class

Examples

library(data.table)
library(concrete)

data <- as.data.table(survival::pbc)
data <- data[1:200, .SD, .SDcols = c("id", "time", "status", "trt", "age", "sex")]
data[, trt := sample(0:1, nrow(data), TRUE)]

# formatArguments() returns correctly formatted arguments for doConcrete()

concrete.args <- formatArguments(DataTable = data,
                                  EventTime = "time",
                                  EventType = "status",
                                  Treatment = "trt",
                                  ID = "id",
                                  ...)

doTmleUpdate

TargetTime = 2500,
TargetEvent = c(1, 2),
Intervention = makeITT(),
CVArg = list(V = 2))

# doConcrete() returns tmle (and g-formula plug-in) estimates of targeted risks
crude.est <- doConcrete(concrete.args)

doTmleUpdate

Title

Description

Title

Usage

doTmleUpdate(
    Estimates,
    SummEIC,
    Data,
    TargetEvent,
    TargetTime,
    MaxUpdateIter,
    OneStepEps,
    NormPnEIC,
    Verbose
)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estimates</td>
<td>list</td>
</tr>
<tr>
<td>SummEIC</td>
<td>data.table</td>
</tr>
<tr>
<td>Data</td>
<td>data.table</td>
</tr>
<tr>
<td>TargetEvent</td>
<td>numeric vector</td>
</tr>
<tr>
<td>TargetTime</td>
<td>numeric vector</td>
</tr>
<tr>
<td>MaxUpdateIter</td>
<td>numeric</td>
</tr>
<tr>
<td>OneStepEps</td>
<td>numeric</td>
</tr>
<tr>
<td>NormPnEIC</td>
<td>numeric</td>
</tr>
<tr>
<td>Verbose</td>
<td>boolean</td>
</tr>
</tbody>
</table>
Description

formatArguments() checks and reformats inputs into a form that can be interpreted by doConcrete(). makeITT() returns an Intervention list for a single, binary, point-treatment variable

Usage

formatArguments(
  DataTable,
  EventTime,
  EventType,
  Treatment,
  ID = NULL,
  TargetTime = NULL,
  TargetEvent = NULL,
  Intervention,
  CVArg = NULL,
  Model = NULL,
  MaxUpdateIter = 500,
  OneStepEps = 0.1,
  MinNuisance = 5/sqrt(nrow(DataTable))/log(nrow(DataTable)),
  Verbose = TRUE,
  GComp = TRUE,
  ReturnModels = TRUE,
  ConcreteArgs = NULL,
  RenameCovs = TRUE,
  ...
)

makeITT(...) ## S3 method for class 'ConcreteArgs'
print(x, ...)
• "Treatment": numeric; the observed treatment value. Binary treatments must be coded as 0, 1
• "Treatment": numeric; the observed treatment

May include
• "ID": factor, character, or numeric; unique subject id. If ID column is missing, row numbers will be used as ID. For longitudinal data, ID must be provided
• "Baseline Covariates": factor, character, or numeric;

EventTime character: the column name of the observed event or censoring time
EventType character: the column name of the observed event type. (0 indicating censoring)
Treatment character: the column name of the observed treatment assignment
ID character (default: NULL): the column name of the observed subject id longitudinal data structures
TargetTime numeric: vector of target times. If NULL, the last observed non-censoring event time will be targeted.
TargetEvent numeric: vector of target events - some subset of unique EventTypes. If NULL, all non-censoring observed event types will be targeted.
Intervention list: a list of desired interventions on the treatment variable. Each intervention must be a list containing two named functions: 'intervention' = function(treatment vector, covariate data) and 'gstar' = function(treatment vector, covariate data) concrete::makeITT() can be used to specify an intent-to-treat analysis for a binary intervention variable
CVArg list: arguments to be passed into do.call(origami::make_folds). If NULL, the default is list(n = nrow(DataTable), fold_fun = folds_vfold, cluster_ids = NULL, strata_ids = NULL)
Model list (default: NULL): named list of models, one for each failure or censoring event and one for the 'Treatment' variable. If Model = NULL, then a template will be generated for the user to amend.
MaxUpdateIter numeric (default: 500): the number of one-step update steps
OneStepEps numeric (default: 1): the one-step tmle step size
MinNuisance numeric (default: 5/log(n)/sqrt(n)): value between (0, 1) for truncating the g-related denominator of the clever covariate
Verbose boolean
GComp boolean (default: TRUE): return g-computation formula plug-in estimates
ReturnModels boolean (default: TRUE): return fitted models from the initial estimation stage
ConcreteArgs list (default: NULL, not yet ready): Use to recheck amended output from previous formatArguments() calls. A non-NULL input will cause all other arguments to be ignored.
RenameCovs boolean (default: TRUE): whether or not to rename covariates
...
additional arguments to be passed into print methods
x a ConcreteArgs object
Value

a list of class "ConcreteArgs"

- Data: data.table containing EventTime, EventType, Treatment, and potentially ID and baseline covariates. Has the following attributes
  - EventTime: the column name of the observed event or censoring time
  - EventType: the column name of the observed event type. (0 indicating censoring)
  - Treatment: the column name of the observed treatment assignment
  - ID: the column name of the observed subject id
  - RenameCovs: boolean whether or not covariates are renamed
- TargetTime: numeric vector of target times to evaluate risk/survival
- TargetEvent: numeric vector of target events
- Regime: named list of desired regimes, each tagged with a 'g.star' attribute function
  - Regime[[i]]: a vector of desired treatment assignments
  - attr(Regime[[i]], 'g.star'): function of Treatment and Covariates, outputting a vector of desired treatment assignment probabilities
- CVFolds: list of cross-validation fold assignments in the structure as output by origami::make_folds()
- Model: named list of model specifications, one for each unique 'EventType' and one for the 'Treatment' variable.
- MaxUpdateIter: the number of one-step update steps
- OneStepEps: list of cross-validation fold assignments in the structure as output by origami::make_folds()
- MinNuisance: numeric lower bound for the propensity score denominator in the efficient influence function
- Verbose: boolean to print additional information
- GComp: boolean to return g-computation formula plug-in estimates
- ReturnModels: boolean to return fitted models from the initial estimation stage

Functions

- makeITT(): makeITT ...
- print(ConcreteArgs): print.ConcreteArgs print method for "ConcreteArgs" class

Examples

library(data.table)
library(concrete)

data <- as.data.table(survival::pbc)
data <- data[1:200, .SD, .SDcols = c("id", "time", "status", "trt", "age", "sex")]
data[, trt := sample(0:1, nrow(data), TRUE)]

# makeITT() creates a list of functions to specify intent-to-treat
# regimes for a binary, single, point treatment variable
intervention <- makeITT()
# formatArguments() returns correctly formatted arguments for doConcrete()
# If no input is provided for the Model argument, a default will be generated
concrete.args <- formatArguments(DataTable = data,
   EventTime = "time",
   EventType = "status",
   Treatment = "trt",
   ID = "id",
   TargetTime = 2500,
   TargetEvent = c(1, 2),
   Intervention = intervention,
   CVArg = list(V = 2))

# Alternatively, estimation algorithms can be provided as a named list
model <- list("trt" = c("SL.glm", "SL.glmnet"),
   "0" = list(Surv(time, status == 0) ~ .),
   "1" = list(Surv(time, status == 1) ~ .),
   "2" = list(Surv(time, status == 2) ~ .))
concrete.args <- formatArguments(DataTable = data,
   EventTime = "time",
   EventType = "status",
   Treatment = "trt",
   ID = "id",
   TargetTime = 2500,
   TargetEvent = c(1, 2),
   Intervention = intervention,
   CVArg = list(V = 2),
   Model = model)

# 'ConcreteArgs' output can be modified and passed back through formatArguments()
# examples of modifying the censoring and failure event candidate regressions
concrete.args[["Model"]][["0"]]<- list(Surv(time, status == 0) ~ trt:sex + age)
concrete.args[["Model"]][["1"]]<- list("mod1" = Surv(time, status == 1) ~ trt,
   "mod2" = Surv(time, status == 1) ~ .)
formatArguments(concrete.args)

---

getEIC

get EICs

description

get EICs

usage

getEIC(  
   Estimates,
getHazFit

Arguments

Estimates list
Data data.table
Regime list
TargetEvent numeric vector
TargetTime numeric vector
MinNuisance numeric
GComp boolean

getHazFit Title

Description

Title

Usage

gethazFit(Data, Model, CVFolds, Hazards, ReturnModels)

Arguments

Data data.table
Model list
CVFolds list
Hazards list
ReturnModels boolean
getInitialEstimate

**Description**

getInitialEstimate

**Usage**

```r
getInitialEstimate(
  Data,
  Model,
  CVFolds,
  MinNuisance,
  TargetEvent,
  TargetTime,
  Regime,
  ReturnModels
)
```

**Arguments**

- **Data**: data.table
- **Model**: list
- **CVFolds**: : list
- **MinNuisance**: numeric
- **TargetEvent**: numeric vector
- **TargetTime**: numeric vector
- **Regime**: list
- **ReturnModels**: boolean

---

**getOutput**

**Description**

getOutput
getOutput

Usage

getOutput(
    ConcreteEst,
    Estimand = c("Risk"),
    Intervention = seq_along(ConcreteEst),
    GComp = NULL,
    Simultaneous = TRUE,
    Signif = 0.05
)

## S3 method for class 'ConcreteOut'
plot(x, NullLine = TRUE, ask = TRUE, ...)

Arguments

ConcreteEst "ConcreteEst" object
Estimand character: "RR" for Relative Risks, "RD" for Risk Differences, and "Risk" for absolute risks
Intervention numeric (default = seq_along(ConcreteEst)): the ConcreteEst list element corresponding to the target intervention. For comparison estimands such as RD and RR, Intervention should be a numeric vector with length 2, the first term designating "treatment" ConcreteEst list element and the second designating the "control".
GComp logical: return g-formula point estimates based on initial nuisance parameter estimation
Simultaneous logical: return simultaneous confidence intervals
Signif numeric (default = 0.05): alpha for 2-tailed hypothesis testing
x a ConcreteOut object
NullLine logical: to plot a red line at y=1 for RR plots and at y=0 for RD plots
ask logical: to prompt for user input before each plot
... additional arguments to be passed into plot methods

Value
data.table of point estimates and standard deviations

Functions

• plot(ConcreteOut): plot.ConcreteOut plot method for "ConcreteOut" class

Examples

library(data.table)
library(concrete)
data <- as.data.table(survival::pbc)
getPropScore

```r
data <- data[1:200, .SD, .SDcols = c("id", "time", "status", "trt", "age", "sex")]
data[, trt := sample(0:1, nrow(data), TRUE)]

# formatArguments() returns correctly formatted arguments for doConcrete()
concrete.args <- formatArguments(DataTable = data,
EventTime = "time",
EventType = "status",
Treatment = "trt",
ID = "id",
TargetTime = 2500,
TargetEvent = c(1, 2),
Intervention = makeITT(),
CVArg = list(V = 2))

# doConcrete() returns tmle (and g-formula plug-in) estimates of targeted risks
concrete.est <- doConcrete(concrete.args)

# getOutput returns risk difference, relative risk, and treatment-specific risks
# GComp=TRUE returns g-formula plug-in estimates
# Simultaneous=TRUE computes simultaneous CI for all output TMLE estimates
concrete.out <- getOutput(concrete.est, Estimand = c("RR", "RD", "Risk"),
GComp = TRUE, Simultaneous = TRUE)

print(concrete.out)
plot(concrete.out, ask = FALSE)
```

---

getPropScore

### Description

getPropScore

### Usage

```r
getPropScore(
  TrtVal,
  CovDT,
  TrtModel,
  MinNuisance,
  Regime,
  CVFolds,
  TrtLoss = NULL,
  ReturnModels
)
```
getPropScore

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>TrtVal</td>
<td>numeric vector</td>
</tr>
<tr>
<td>CovDT</td>
<td>data.table</td>
</tr>
<tr>
<td>TrtModel</td>
<td>list or fitted object</td>
</tr>
<tr>
<td>MinNuisance</td>
<td>numeric</td>
</tr>
<tr>
<td>Regime</td>
<td>list</td>
</tr>
<tr>
<td>CVFolds</td>
<td>list</td>
</tr>
<tr>
<td>TrtLoss</td>
<td>character or function</td>
</tr>
<tr>
<td>ReturnModels</td>
<td>boolean</td>
</tr>
</tbody>
</table>
Index

* package
  concrete-package, 2

concrete (concrete-package), 2
concrete-package, 2
coxph, 2
doConcrete, 3
doTmleUpdate, 5

formatArguments, 6

getEIC, 9
getHazFit, 10
getInitialEstimate, 11
getOutput, 11
getPropScore, 13
glmnet, 2

makeITT (formatArguments), 6

plot.ConcreteEst (doConcrete), 3
plot.ConcreteOut (getOutput), 11
print.ConcreteArgs (formatArguments), 6
print.ConcreteEst (doConcrete), 3
print.ConcreteOut (doConcrete), 3

SuperLearner, 2