Package 'tmleCommunity'

March 7, 2019

Title Targeted Maximum Likelihood Estimation for Hierarchical Data

Version 0.1.0

Author Chi Zhang [aut, cre],
Oleg Sofrygin [aut],
Jennifer Ahern [aut],
Mark J. van der Laan [aut, ths]

Maintainer Chi Zhang <chi.zhang@berkeley.edu>

Description Targeted minimum loss-based estimation (TMLE) of the average causal effect of community-based intervention(s) at a single time point on an individual-based outcome of interest. It provides three approaches to analyze hierarchical data: community-level TMLE, inidividual-level TMLE and stratified TMLE. Implementations of the inverse-probability-of-treatment-weighting (IPTW) and the G-computation formula (GCOMP) are also available for each approach. The package supports multivariate arbitrary interventions (deterministic or stochastic) with a binary or continuous outcome. The tmleCommunity() function calculates the marginal treatment effect among independent community units (or i.i.d individual units if no hierarchical structure) using TMLE. Besides, it allows user-specified data-adaptive machine learning algorithms through SuperLearner and h2oEnsemble packages. The input dataset should be made up of rows of community-specific and individual-specific observations, with each row i (in community j) containing random variables (W_{i,j}, E_j, A_j, Y_{i,j}), where E_j represents a vector of community j's environmental baseline covariates, W_{i,j}represents a vector of individual i's individual-level baseline covariates, A_j is the exposure(s) (can be univariate or multivariate, can be binary, categorical or continuous) assigned or naturally occurred in community j and Y_{i,j} is i's outcome (either binary or continuous). More details can be found in '?tmleCommunity-package' and '?tmleCommunity'.

URL https://github.com/chizhangucb/tmleCommunity

BugReports https://github.com/chizhangucb/tmleCommunity/issues

Depends R (>= 3.3.0)

License GPL-2

LazyData true

Imports assertthat,
data.table,
Matrix,
R6,
stats,
speedglm,
methods,

Hmisc,
plm,
SuperLearner,
h2o,
h2oEnsemble,
sl3

Suggests doParallel,
foreach,
simcausal,
testthat,
knitr,
glmnet,
gam,
arm,
randomForest

RoxygenNote 6.1.1

Collate 'GeneralUtilities.R'
'BinaryOutModelClass.R'
'DatKeepClass.R'
'GenericModelClasses.R'
'MonteCarloSimClass.R'
'hbarDensityModel.R'
'tmleCommunity-package.R'
'tmleCommunity.R'
'zzz.R'

R topics documented:

tmleCommunity-package . . . 3
BinaryOutModel . . . 4
CategorModel . . . 7
comSample.wmT.bA.bY_list . . . 8
ContinModel . . . 10
DatKeepClass . . . 11
fitGenericDensity . . . 14
GenericModel . . . 16
indSample.iid.bA.bY.rareJ1_list . . . 18
indSample.iid.bA.bY.rareJ2_list . . . 19
indSample.iid.cA.cY_list . . . 19
MonteCarloSimClass . . . 20
panelData_Trans . . . 21
print_tmleCom_opts . . . 23
RegressionClass . . . 23
tmleCommunity . . . 25
tmleCom_Options . . . 40

Index **44**

`tmleCommunity-package` *Targeted Maximum Likelihood Estimation for Community-level Data*

Description

Targeted Maximum Likelihood Estimation (TMLE) of the average causal effect of community-based intervention(s) at a single time point on an individual-based outcome of interest. (and can be extended to additive treatment effect). In other words, it estimates the marginal treatment effect of single-time point arbitrary interventions on a continuous or binary outcome in community-independent data, adjusting for both community-level and individual-level baseline covariates. The package also provides Inverse-Probability-of-Treatment-Weighted estimator (IPTW) and parametric G-computation formula estimator (GCOMP). The statistical inference (Standard errors, t statistc, p-value and confidence intervals) of both TMLE and IPTW are based on the corresponding influence curve, respectively. Optional data-adaptive estimation of exposure and outcome mechanisms using the SuperLearner package, the sl3 package (a modern implementation of the Super Learner algorithm) and the h2o package (for a large dataset) is strongly recommended, especially when the outcome mechanism and treatment mechnism are unknown. Besides, it allows for panel data transformation, such as with random effects and fixed effects.

Details

The input dataset should be made up of rows of community-specific and individual-specific observations, for community j, each row i includes random variables $(W_{i,j}, E_j, A_j, Y_{i,j})$, where E_j represents a vector of community j's community-level (environmental) baseline covariates (individuals within the same community share the same values of E_j), $W_{i,j}$ represents a vector of individual i's individual-level baseline covariates, A_j is the exposure(s) (can be univariate or multivariate, can be binary, categorical or continuous) assigned or naturally occurred in community j (individuals within the same community receive the same value of A_j) and $Y_{i,j}$ is i's outcome (either binary or continuous). Each individual's baseline covariates ($W_{i,j}$ depends on the environmental baseline covariates E_j of the community j to which i belongs to. Similarly, each community's exposure A_j depends on its community-level baseline covariates E_j and individual-level baseline covariates of all individuals belonging to community j (all $W_{i,j}$ such that i belongs to j). Besides, each outcome $Y_{i,j}$ could be affected by its baseline community and individual-level covariates $(E_j, W_{i,j})$ and the baseline covariates of other individuals within the same community $(W_{s,j} : s \neq i, s \in j)$, together with its community-based intervention A_j. We note that the input data with no hierarchical structure (i.e., no communities and only individuals) is a special case of the hierarchical data since it simply treats E_j as `NULL`.

There are currently three approaches that can be used in hierarchical data analysis. The first community-level TMLE is developed under a non-parametric causal model that allows for arbitrary interactions between individuals within a community. It estimates the community-level causal effect by aggregating data at a community-level and treating community rather than the individual as the unit of analysis (i.e., both community-level outcome and treatment mechanisms). The second individual-level TMLE is developed under the submodel of the causal model in the first approach, incoporating knowledge of the dependence structure between individual within communities (i.e., both individual-level outcome and treatmnet mechanisms). The third stratified TMLE fits a separate outcome (exposure) mechanism for each community, and then combine those estimates into a (user-specific) average (Default to be community size-weighed). Note that the stratified TMLE naturally controls for the community-level observed covariates and unobserved factors. Namely, there is no E in the regressors for both outcome and treatment mechanisms.

References

1. Balzer L. B., Zheng W., van der Laan M. J., Petersen M. L. and the SEARCH Collaboration (2017). A New Approach to Hierarchical Data Analysis: Targeted Maximum Likelihood Estimation of Cluster-Based Effects Under Interference. ArXiv e-prints. 1706.02675.
2. Muoz, I. D. and van der Laan, M. (2012). Population Intervention Causal Effects Based on Stochastic Interventions. Biometrics, 68(2):541-549.
3. Sofrygin, O. and van der Laan, M. J. (2015). tmlenet: Targeted Maximum Likelihood Estimation for Network Data. R package version 0.1.9. https://github.com/osofr/tmlenet
4. van der Laan, M. (2014). Causal Inference for a Population of Causally Connected Units. Journal of Causal Inference, 2(1)
5. van der Laan, Mark J. and Gruber, Susan (2011). "Targeted Minimum Loss Based Estimation of an Intervention Specific Mean Outcome". U.C. Berkeley Division of Biostatistics Working Paper Series. Working Paper 290. http://biostats.bepress.com/ucbbiostat/paper290
6. van der Laan, Mark J. and Rose, Sherri, "Targeted Learning: Causal Inference for Observational and Experimental Data" New York: Springer, 2011.

Datasets

To learn more about the type of data input required by `tmleCommunity`, see the following example datasets:

- `comSample.wmT.bA.bY_list`
- `indSample.iid.cA.cY_list`
- `indSample.iid.bA.bY.rareJ1_list`
- `indSample.iid.bA.bY.rareJ2_list`

For R code that can simulate more data with different structures, please check

`https://github.com/chizhangucb/tmleCommunity/tree/master/tests/dataGeneration`

Updates

Check for updates and report bugs at `https://github.com/chizhangucb/tmleCommunity`.

`BinaryOutModel` *R6 class for modeling (fitting and predicting) for a single binary regression model P(B | PredVars)*

Description

`BinaryOutModel` can store and manage the (binarize/ discretized) design matrix Xmat and the outcome Bin for the binary regression P(Bin|Xmat). It provides argument `self$estimator` to include different candidate estimators in the fitting and predicting library, such as data-adaptive super learner algorithms and parametric logistic regression. When fitting one pooled regression across multiple bins, it provides method to convert data from wide to long format when requested (to gain computational efficiency).

Usage

```
BinaryOutModel
```

Format

An R6Class generator object

Details

- bin_names - Character vector of names of the bins.
- ID - Integer vector of observation IDs used for pooling. 1:n.
- pooled_bin_name - Original name of the continuous covariate that was discretized into bins and then pooled.
- nbins - Number of bins used for estimation of a continuous outvar, defined in ContinModel$new().
- estimator - Character, one of "speedglm__glm" (default), "glm__glm", "h2o__ensemble", "SuperLearner".
- outvar - Character, outcome name.
- predvars - Character vector of predictor names.
- cont.sVar.flag - Logical. If TRUE, indicate the original outcome variable is continuous.
- bw.j - Bin width of a bin indicator obtained from the discretization of a continous covariate.
- is.fitted - Logical. If TRUE, indicate the BinaryOutModel class object is fitted already.
- pool_cont - Logical. If TRUE, perform pooling of bins.
- outvars_to_pool - Character vector of outcome bin names for pooling.
- ReplMisVal0 - Logical. If TRUE, user-supplied gvars$misXreplace (Default to 0) will be used to replace all gvars$misval among predictors. ReplMisVal0 in RegressionClass will be used when instantiating an new object of BinaryOutModel.
- n - Number of rows in the input data.
- subset_expr - Vector of length n that specifies a subset of data to be used in the fitting process. Either logical, expression or indices.
- subset_idx - Logical version of subset_expr.

Methods

`new(reg)` Use reg (a RegressionClass class object) to instantiate an new object of BinaryOutModel for a single binary regression.

`newdata(newdata, getoutvar = TRUE, ...)` Evaluate subset and perform correct subseting of data to construct X_mat, Yvals & wt_vals.

`define.subset.idx(data)` Create a logical vector which is converted from subset_expr

`fit(overwrite = FALSE, data, predict = FALSE, savespace = TRUE, ...)` fit a binary regression. Note that overwrite is Logical. If FALSE (Default), the previous fitted model cannot be overwritten by new fitting model. savespace is Logical. If TRUE (Default), wipe out all internal data when doing many stacked regressions.

`copy.fit(bin.out.model)` Take fitted BinaryOutModel object as an input and save the fit to itself.

`predict(newdata, savespace = TRUE, ...)` Predict the response P(A = 1|W = w, E = e).

`copy.predict(bin.out.model)` Tke BinaryOutModel object that contains the predictions for P(A=1|w,e) and save to itself

`predictAeqa(newdata, bw.j.sA_diff, savespace = TRUE, wipeProb = TRUE)` Predict the response P(A = a|W = w, E = e) for observed A, W, E. Note that wipeProb is logical argument for self$wipe.alldat. If FALSE, vectors of probA1 & probAeqa will be kept.

`show()` Print regression formula, including outcome and predictor names.

Active Bindings

wipe.alldat(wipeProb = TRUE) ...

getfit ...

getprobA1 ...

getprobAeqa ...

emptydata ...

emptyY ...

emptyWeight ...

emptySubset_idx ...

getXmat ...

getY ...

getWeight ...

See Also

DatKeepClass, RegressionClass, tmleCom_Options

Examples

```
## Not run:
#*********************************************************************************************
# Example 1: Estimate a outcome regression directly through BinaryOutModel
data(indSample.iid.bA.bY.rareJ2_list)
indSample.iid.bA.bY.rareJ2 <- indSample.iid.bA.bY.rareJ2_list$indSample.iid.bA.bY.rareJ2
N <- nrow(indSample.iid.bA.bY.rareJ2)
# speed.glm to fit regressions (it's GLMs to medium-large datasets)
tmleCom_Options(Qestimator = "speedglm__glm", maxNperBin = N)
options(tmleCommunity.verbose = TRUE)  # Print status messages
#*********************************************************************************************

#*********************************************************************************************
# 1.1 Specifying outcome and predictor variables for outcome mechanism
#*********************************************************************************************
# Y depends on all its parent nodes (A, W1, W2, W3, W4)
Qform.all <- Y ~ W1 + W2 + W3 + W4 + A
Q.sVars1 <- tmleCommunity:::define_regform(regform = Qform.all)

# Equivalent way to define Q.sVars: use Anodes.lst (outcomes) & Wnodes.lst (predictors)
# node can only contain one or more of Ynode, Anodes, WEnodes, communityID and Crossnodes
nodes <- list(Ynode = "Y", Anodes = "A", WEnodes = c("W1", "W2", "W3", "W4"))
Q.sVars2 <- tmleCommunity:::define_regform(regform = NULL, Anodes.lst = nodes$Ynode,
                                           Wnodes.lst = nodes[c("Anodes", "WEnodes")])

# Also allows to include interaction terms in regression formula  (Correct Qform)
Qform.interact <- Y ~ W1 + W2*A + W3 + W4
Q.sVars3 <- tmleCommunity:::define_regform(regform = Qform.interact)

# Alternative way to define Qform.interact
Qform.interact2 <- Y ~ W1 + W2 + W3 + W4 + A + W2:A
Q.sVars4 <- tmleCommunity:::define_regform(regform = Qform.interact2)

#*********************************************************************************************
```

```
# 1.2 Fit and predict a regression model for outcome mechanism Qbar(A, W)
#*********************************************************************************
# Create a new object of DatKeepClass that can store and munipulate the input data
OData_R6 <- DatKeepClass$new(Odata = indSample.iid.bA.bY.rareJ2,
                             nodes = nodes, norm.c.sVars = FALSE)
# Add a vector of observation (sampling) weights that encodes knowledge of rare outcome
OData_R6$addObsWeights(obs.wts = indSample.iid.bA.bY.rareJ2_list$obs.wt.J2)

# Create a new object of RegressionClass that defines regression models
# using misspecified Qform (without interaction term)
Qreg <- RegressionClass$new(outvar = Q.sVars1$outvars, predvars = Q.sVars1$predvars,
                            subset_vars = (!rep_len(FALSE, N)))

# Set savespace=FALSE to save all productions during fitting, including models and data
m.Q.init <- BinaryOutModel$new(reg = Qreg)$fit(data = OData_R6, savespace = FALSE)
length(m.Q.init$getY)  # 3000, the outcomes haven't been erased since savespace = FALSE
head(m.Q.init$getXmat)  # the predictor matrix is kept since savespace = FALSE
m.Q.init$getfit$coef  # Provide cofficients from the fitting regression
m.Q.init$is.fitted  # TRUE

# Now fit the same regression model but set savespace to TRUE (only fitted model left)
# Need to set overwrite to TRUE to avoid error when m.Q.init is already fitted
m.Q.init <- m.Q.init$fit(overwrite = TRUE, data = OData_R6, savespace = TRUE)
all(is.null(m.Q.init$getXmat), is.null(m.Q.init$getY))  # TRUE, all wiped out

# Set savespace = TRUE to wipe out any traces of saved data in predict step
m.Q.init$predict(newdata = OData_R6, savespace = TRUE)
is.null(m.Q.init$getXmat)  # TRUE, the covariates matrix has been erased to save RAM space
mean(m.Q.init$getprobA1)  # 0.02175083, bad estimate since misspecified Qform

#*********************************************************************************
# 1.3 Same as above but using Super Learner (data-adaptive algorithms)
#*********************************************************************************
# Specifying the SuperLearner library in tmleCom_Options()
library(SuperLearner)
tmleCom_Options(SL.library = c("SL.glm", "SL.randomForest"), maxNperBin = N)
# Instead of reinitiating a RegressionClass object, change estimator directly in Qreg
# so don't need to redefine Qestimator in tmleCom_Options()
Qreg$estimator <- "SuperLearner"

set.seed(12345)
m.Q.init <- BinaryOutModel$new(reg = Qreg)$fit(data = OData_R6, savespace = TRUE)
m.Q.init$predict(newdata = OData_R6, savespace = TRUE)
mean(m.Q.init$getprobA1)

## End(Not run)
```

`CategorModel` *R6 class for modeling (fitting and predicting) joint probability for a univariate categorical outcome* `A[m]`

Description

`CategorModel` inherits from `GenericModel` class, defining and modeling a conditional density $P(A[m]|W, E...)$ where `A[m]` is univariate and categorical. By calling `self$new()`, `A[m]` will

be redefined into number of bins `length(levels)` (i.e., number of unique categories in `A[m]`). By calling `self$fit()`, it fits hazard regressoin `Bin_A[m][k] ~ W + E` on `data` (a `DatKeepClass` class), which is the hazard probaility of the observation of A[m] belongs to bin `Bin_A[m][t]`, given covariates (W, E) and that observation doesn't belong to any precedent bins `Bin_A[m][1]`, `Bin_A[m][2]`, ..., `Bin_A[m][k-1]`.

Usage

```
CategorModel
```

Format

An `R6Class` generator object

Details

- `reg` - .
- `outvar` - .
- `levels` - Numeric vector of all unique categories in outcome outvar.
- `nbins` - .
- `bin_nms` - .

Methods

`new(reg, DatKeepClass.g0, ...)` Instantiate an new instance of `CategorModel` for a univariate categorical outcome `A[m]`

`fit(data)` ...

`predict(newdata)` ...

`predictAeqa(newdata)` ...

Active Bindings

`cats` ...

See Also

`DatKeepClass`, `RegressionClass`, `GenericModel`, `BinaryOutModel`

`comSample.wmT.bA.bY_list`

An Example of a Hierarchical Data Containing a Cluster-Based Binary Exposure with a Individual-Level Binary Outcome.

Description

Simulated hierarchical dataset containing 1000 independent communities, each (community j) containing n_j (non-fixed) number of individuals where n_j is drawn from a normal with mean 50 and standard deviation 10 and round to the nearest integer. Each row (observation) includes 2 measured community-level baseline covariates (`E1, E2`), 3 dependent individual-level baseline covariates (`W1, W2, W3`), 1 dependent bianry exposure (`A`) and 1 dependent binary outcoem (`Y`), along with one unique community identifier (`id`). The community-level baseline covariates (`E1, E2`) were sampled as i.i.d across all communities, while the individual-level baseline covariates (`W1, W2, W3`) for each individual i within community j was generated conditionally on the values of j's community-level baseline covariates (`E1[j], E2[j]`). Then the community-level exposure (`A`) for each community j was sampled conditionally on the value of j's community-level baseline covariates (`E1[j], E2[j]`), together with all invididuals' baseline covariates (`W1[i], W2[i], W3[i]`) within community j where $i = 1, .., n_j$. Similary, the individual-level binary outcome `Y` for each individual i within community j was sampled conditionally covariates and exposure (`E1[j], E2[j], A[j]`), as well as the value of individual i's baseline covariates on the value of community j's baseline (`W1[i], W2[i], W3[i]`). The following section provides more details regarding individual variables in simulated data.

Usage

```
data(comSample.wmT.bA.bY_list)
```

Format

A data frame with 1000 independent communities, each containing around 50 individuals (in total 50,457 observations (rows)), and 8 variables (columns):

id integer (unique) community identifier from 1 to 1000, identical within the same community

E1 continuous uniform community-level baseline covariate with `min=0` and `max=1` (independent and identical across all individuals in the same community)

E2 discrete uniform community-level baseline covariate with 5 elements (0, 0.2, 0.4, 0.8, 1) (independent and identical across all individuals in the same community)

W1 binary individual-level baseline covariate that depends on the values of community-level baseline covaries (`E1,E2`)

W2 continuous individual-level baseline covariate, together with `W3`, are drawn from a bivariate normal distribution with correlation 0.6, depending on the values of community's baseline covaries (`E1, E2`)

W3 continuous normal individual-level baseline covariate, correlated with `W2`, see details in above

A binary exposure that depends on community's baseline covariate values in (`E1, E2`), and the mean of all individuals' baseline covariates `W1` within the same community

Y binary outcome that depends on community's baseline covariate and exposure values in (`E1, E2, A`), and all individuals' baseline covariate values in (`W2, W3`)

Source

https://github.com/chizhangucb/tmleCommunity/blob/master/tests/dataGeneration/get.cluster.dat.Abin.R

Examples

```
data(comSample.wmT.bA.bY_list)
comSample.wmT.bA.bY <- comSample.wmT.bA.bY_list$comSample.wmT.bA.bY
head(comSample.wmT.bA.bY)
comSample.wmT.bA.bY_list$psi0.Y  # 0.103716, True ATE
# summarize the number of individuals within each community
head(table(comSample.wmT.bA.bY$id))
```

`ContinModel` *R6 class for modeling (fitting and predicting) joint probability for a univariate continuous outcome* `A[m]`

Description

`ContinModel` inherits from `GenericModel` class, defining and modeling a joint conditional density $P(A[m]|W, E, ...)$ where `A[m]` is univariate and continuous. By calling `self$new()`, `A[m]` will be discretized into `nbins` bins via one of the 3 bin cutoff approaches (See Details for `tmleCommunity`). By calling `self$fit()`, it fits hazard regressoin `Bin_A[m][k] ~ W + E` on `data` (a `DatKeepClass` class), which is the hazard probaility of the the observation of `A[m]` belongs to bin `Bin_A[m][k]`, given covariates (W, E) and that observation doesn't belong to any precedent bins `Bin_A[m][1]`, `Bin_A[m][2]`, ..., `Bin_A[m][k-1]`.

Usage

```
ContinModel
```

Format

An `R6Class` generator object

Details

- `reg` - .
- `outvar` - .
- `nbins` -
- `bin_nms` - Character vector of column names of bin indicators.
- `intrvls` -
- `intrvls.width` -
- `bin_weights` - .

Methods

`new(reg, DataStorageClass.g0, DataStorageClass.gstar, ...)` Instantiate an new instance of `ContinModel` for a univariate continuous outcome `A[m]`

`fit(data, savespace = TRUE)` ...

`predict(newdata, savespace = TRUE)` ...

`predictAeqa(newdata, savespace = TRUE, wipeProb = TRUE)` ...

Active Bindings

cats ...

See Also

DatKeepClass, RegressionClass, GenericModel, BinaryOutModel

DatKeepClass *R6 class for Storing, Managing, Subsetting and Manipulating the Input Data.*

Description

DatKeepClass allows user to access the input data. The processed covariates from sVar.object are stored as a matrix in (private$.mat.sVar). This class could subset, combine, normalize, discretize and binarize covariates in (A, W, E). For disretization of continous and categorical variables, it can automatically detect / set covariates type (binary, categor, contin), detect / set bin intervals, and construct bin indicators. Besides, it provides methods for generating new exposures under user-specific arbitrary intervention g^* through self$make.dat.sVar, and allows user to replace missing values with user-specific gvars$misXreplace (Default to 0). Its pointers will be passed on to GenericModel functions: using in $fit(), $predict() and $predictAeqa().

Usage

DatKeepClass

Format

An R6Class generator object

Details

- norm.c.sVars - flag if TRUE normalize continuous covariates.
- mat.bin.sVar - Matrix of the binary indicators created from discretization of continuous covariate active.bin.sVar.
- ord.sVar - Ordinal (categorical) transformation of a continous covariate sVar.
- obs.wts - Vectopr of observation (sampling) weights (of length ndat.sVar). If NULL, assumed to be all 1.
- YnodeVals - Vector of outcome values (Ynode) in observed data
- det.Y - Logical vector, where YnodeVals[det.Y==TRUE] are deterministic and set to NA.
- p - Number of Monte-Carlo simulations performed.
- ndat.sVar - Number of observations in the observed data frame.

Methods

`new(Odata, nodes, YnodeVals, det.Y, norm.c.sVars = FALSE, ..)` Instantiate an new instance of `DatKeepClass` that is used for storing and manipulating the input data.

`addYnode(YnodeVals, det.Y)` Add protected Y node to private field and set to NA all deterministc Y values for public field YnodeVals.

`addObsWeights(obs.wts)` Add observation weights to public field.

`evalsubst(subset_vars, subset_exprs = NULL)` ...

`get.dat.sVar(rowsubset = TRUE, covars)` Subset covariate design matrix for `BinaryOutModel`.

`get.outvar(rowsubset = TRUE, var)` Subset a vector of outcome variable for `BinaryOutModel`.

`get.obsweights(rowsubset = TRUE)` Subset a vector of observation weights for `BinaryOutModel`.

`def.types.sVar(type.sVar = NULL)` Define each variable' class in input data: bin, cat or cont.

`set.sVar.type(name.sVar, new.type)` Assign a new class type to one variable that belongs to the input data.

`get.sVar.type(name.sVar)` Return the class type of a variable.

`is.sVar.cont(name.sVar)` Check if the variable is continuous.

`is.sVar.cat(name.sVar)` Check if the variable is categorical.

`is.sVar.bin(name.sVar)` Check if the variable is binary.

`get.sVar(name.sVar)` Return a vector of the variable values.

`set.sVar(name.sVar, new.sVarVal)` Assign a vector of new values to the specific variable.

`bin.nms.sVar(name.sVar, nbins)` Define names of bin indicators for `sVar`.

`detect.sVar.intrvls(name.sVar, nbins, bin_bymass, bin_bydhist, max_nperbin)` ...

`detect.cat.sVar.levels(name.sVar)` Detect the unique categories in categorical sVar, returning in increasing order.

`get.sVar.bw(name.sVar, intervals)` Get the bin widths vector for the discretized cont `sVar`.

`get.sVar.bwdiff(name.sVar, intervals)` Get the bin widths differences vector for the discretized continuous `sVar`.

`binirize.sVar(name.sVar, ...)` Create a matrix of bin indicators for categorical/cont `sVar`.

`norm.cont.sVars()` Normalize continuous `sVar` (Note that this process is memory-costly).

`fixmiss_sVar()` Replace all missing (NA) values with a default integer (Default to 0).

`make.dat.sVar(p = 1, f.g_fun = NULL, regform = NULL)` Generate new exposures under user-specific arbitrary intervention `f.g_fun` and construct a data.frames that combines all covariates, replacing the old exposures with the new ones.

Active Bindings

`names.sVar` Return variable names of the input data.

`names.c.sVar` Return continuous variable names of the input data.

`ncols.sVar` Return the number of columns of the input data.

`nobs` Return the number of observations of the input data.

`dat.sVar` Return a data frame object that stores the entire dataset (including all `sVar`.).

`dat.bin.sVar` Return a stored matrix for bin indicators on currently binarized continous `sVar`.

`active.bin.sVar` Return name(s) of active binarized continous sVar(s), changing when `fit` or `predict` is called.

`emptydat.sVar` Wipe out `dat.sVar`.

`emptydat.bin.sVar` Wipe out `dat.bin.sVar`.

`noNA.Ynodevals` Return the observed Y without any missing values.

`nodes` ...

`type.sVar` ...

See Also

`tmleCom_Options`, `tmleCommunity`

Examples

```
## Not run:
#******************************************************************************************
# Example 1: storing, managing, subsetting and manipulating a data with continuous A
data(indSample.iid.cA.cY_list)
indSample.iid.cA.cY <- indSample.iid.cA.cY_list$indSample.iid.cA.cY
psi0.Y <- indSample.iid.cA.cY_list$psi0.Y  # 0.333676
# Assume that W2 has no effect on neither A nor Y, so no need to put into nodes
nodes <- list(Ynode = "Y", Anodes = "A", WEnodes = c("W1", "W3", "W4"))
tmleCom_Options(nbins = 10, maxNperBin = nrow(indSample.iid.cA.cY))
#******************************************************************************************

#******************************************************************************************
# 1.1 Specifying the stochastic intervention of interest gstar
#******************************************************************************************
# Interested in the effect of a shift of delta(W1, W3, W4) of the current treatment
define_f.gstar <- function(data, ...) {
  shift.mu <- 0.3 * data[,"W1"] + 0.6 * data[,"W3"] - 0.14 * data[,"W4"]
  shift.val <- rnorm(n = NROW(data), mean = shift.mu, sd = 0.5)
  shifted.new.A <- data[, "A"] - shift.val
  return(shifted.new.A)
}

#******************************************************************************************
# 1.2 Creating an R6 object of DatKeepClass (to store the input data)
#******************************************************************************************
# Don't normalize continous covariates by setting norm.c.sVars = FALSE
OData_R6 <- DatKeepClass$new(Odata = subset(indSample.iid.cA.cY, select=-Y),
                             nodes = nodes[c("Anodes", "WEnodes")], norm.c.sVars = FALSE)
OData_R6$nodes <- nodes
# names of all variables that are in input data and specified in nodes
OData_R6$names.sVar  # "A"  "W1" "W3" "W4"
# names of all continuous variables that are in input data and specified in nodes
OData_R6$names.c.sVar  # "A" "W3" "W4"
# a sub dataframe of the input data, including all variables in nodes
head(OData_R6$dat.sVar)
# the number of observations of the input data
OData_R6$nobs  # 10000
OData_R6$get.sVar.type("A")  # "contin"
OData_R6$get.sVar.type()  # Provide a list of types of all variables

#******************************************************************************************
# 1.3 Manipulating the input data by adding observed outcomes and observation weights
#******************************************************************************************
```

```
# Bound observed outcome into [0, 1]
obsYvals <- indSample.iid.cA.cY[, nodes$Ynode]
ab <- range(obsYvals, na.rm=TRUE)
indSample.iid.cA.cY[, nodes$Ynode] <- (obsYvals-ab[1]) / diff(ab)

# Add YnodeVals (a vector of outcomes) to both public and private field
OData_R6$addYnode(YnodeVals = indSample.iid.cA.cY[, nodes$Ynode], det.Y = FALSE)
# set YnodeVals[det.Y=TRUE] to NA in public field (with NAs)
head(OData_R6$YnodeVals)
# protect YnodeVals from being set to NA in private field (without NAs)
head(OData_R6$noNA.Ynodevals)

# Add a vector of observation (sampling) weights
OData_R6$addObsWeights(obs.wts = rep(c(1,2), 5000))
# Assume all weights to be 1 (i.e., equally weighted)
OData_R6$addObsWeights(obs.wts = 1)

#*********************************************************************************
# 1.4 Creating an new R6 object of DatKeepClass under stochastic intervention g.star
# Generate new exposures under user-specific intervention f.g_fun
#*********************************************************************************
OData.gstar_R6 <- DatKeepClass$new(Odata = indSample.iid.cA.cY, nodes = nodes)
# Create 1 new Odata and replace A under g0 in Odata with A* under g.star
set.seed(12345)
OData.gstar_R6$make.dat.sVar(p = 1, f.g_fun = define_f.gstar)
dim(OData.gstar_R6$dat.sVar)  # 10000     4
# Create 3 new Odatas and repalce A with A*
OData.gstar_R6$make.dat.sVar(p = 3, f.g_fun = define_f.gstar)
dim(OData.gstar_R6$dat.sVar)  # 30000     4
# Since A* is stochastically generated, each p may produce different values of A*
head(OData.gstar_R6$dat.sVar[1:10000, ])
head(OData.gstar_R6$dat.sVar[10001:20000, ])

## End(Not run)
```

`fitGenericDensity` *Define and fit the multivariate conditional density under the user-specified arbitrary intervention function.*

Description

Defines and fits regression models for the conditional density `P(A=a|W=w)` where `a` is generated under the user- specified arbitrary (can be static, dynamic or stochastic) intervention function `f_gstar`. Note that `A` can be multivariate `(A[1], ..., A[j])` and each of the compoenents A[i] can be either binary, categorical or continuous. See detailed description in `RegressionClass`.

Usage

```
fitGenericDensity(data, Anodes, Wnodes, gform = NULL, f_gstar,
  h.gstar_GenericModel = NULL, lbound = 0.01, n_MCsims = 1,
  obs.wts = NULL, rndseed = NULL, verbose = TRUE)
```

Arguments

data	data.frame with named columns, containing Wnodes, Anode and Ynode.
Anodes	Column names or indices in data of outcome variables; exposures can be either binary, categorical or continuous.
Wnodes	Column names or indices in data of baseline covariates. Factors are not currently allowed.
gform	Character vector of regression formula for estimating the conditional density of P(A | W)
f_gstar	Either a function or a vector or a matrix/ data frame of counterfactual exposures. See details in function argument f_gstar1 in tmleCommunity.
h.gstar_GenericModel	...
lbound	lower bounds on estimated cumulative probabilities for P(A=a|W=w), default to 0.01
n_MCsims	...
obs.wts	...
rndseed	...
verbose	...

Value

A named list with 3 items containing the estimation results for:

- h_gstar - A vector of likelihood prediction for P(A=a|W=w) where a is generated under the user-specified intervention.
- OData.gstar - A DatKeepClassclass object,where outcomes are generated under intervention f_gstar.
- genericmodels.gstar - A GenericModel class object that defines and models P(A=a|W=w).

See Also

tmleCom_Options, DatKeepClass, RegressionClass, GenericModel, ContinModel, CategorModel, tmleCommunity

Examples

```
## Not run:
data(indSample.iid.cA.cY_list)
indSample.iid.cA.cY <- indSample.iid.cA.cY_list$indSample.iid.cA.cY
tmleCom_Options(gestimator = "speedglm__glm", maxNperBin = nrow(indSample.iid.cA.cY),
                bin.method = "dhist", nbins = 8)
options(tmleCommunity.verbose = TRUE)  # Print status messages

# Define a stochastic intervention
define_f.gstar <- function(shift.rate, ...) {
  eval(shift.rate)
  f.gstar <- function(data, ...) {
    print(paste0("rate of shift: ", shift.rate))
    shifted.new.A <- data[, "A"] - mean(data[, "A"]) * shift.rate
    return(shifted.new.A)
  }
```

```
  return(f.gstar)
}
f.gstar <- define_f.gstar(shift.rate = 0.5)

# Under current treatment mechanism g0
h_gN <- fitGenericDensity(data = indSample.iid.cA.cY, Anodes = "A",
                          Wnodes = c("W1", "W2", "W3", "W4"),
                          f_gstar = NULL, lbound = 0)$h_gstar
# Under stochastic intervention gstar
h_gstar <- fitGenericDensity(data = indSample.iid.cA.cY, Anodes = "A",
                             Wnodes = c("W1", "W2", "W3", "W4"),
                             f_gstar = f.gstar, lbound = 0)$h_gstar

## End(Not run)
```

`GenericModel` *Generic R6 class for modeling (fitting and predicting)* $P(A|W,E)$ *where* A *can be multivariate* $(A[1], ..., A[M])$

Description

`GenericModel` defines and models the conditional density $P(A = a|W = w, E = e)$, where a are generated under g_star or g_0. By calling `self$new(reg)`, it utilizes estimation options defined in `RegressionClass` class, and automatically factorizes the multivariate conditional probability model $P(A|W,E)$ into M univariate conditional probability models (can be binary, categorical or continuous) and finally an entire tree of binary regression models (see details in `tmleCommunity`), where a new instance of `BinaryOutModel` class will be initialized for each binary regression (If one outcome variable $A[m]$ is already binary, then immediately call a new instance of `BinaryOutModel`). By calling `self$fit(data)` and `self$predict(newdata)`, where `data` and `newdata` are `DatKeepClass` class objects, it fits $P(A|W,E)$ and predicts $P(A = 1|W = w, E = e)$, where values of (w, e) are from `newdata`. Moreover, it predicts likelihood function $P(A = a|W = w, E = e)$ through `self$predictAeqa(newdata)`, where (a, w, e) are from `newdata` (also a `DatKeepClass` class).

Usage

```
GenericModel
```

Format

An `R6Class` generator object

Details

- `n_regs` - Total number of regression models.
- `parfit_allowed` - Logical. If TRUE, allow parallel computing to fit multivariate outvar.

Methods

`new(reg, ...)` Use `reg` (a `RegressionClass` object) to instantiate an new object of `GenericModel`

`length` Get the number of regression models (i.e., the number of exposure viarables)

`getPsAsW.models` Get all model objects (one model object per outcome var `A[m]`)

`getcumprodAeqa` Get joint prob as a vector of the cumulative prod over j for `P(A[m]=a[m]|W,E)`

`fit(data, savespace = TRUE)` ...

`copy.fit(Generic.Model)` ...

`predict(newdata, savespace = TRUE)` ...

`predictAeqa(newdata, savespace = TRUE, wipeProb = TRUE, ...)` ...

Active Bindings

`wipe.alldat(wipeProb = TRUE)` ...

See Also

DatKeepClass, RegressionClass, ContinModel, CategorModel, BinaryOutModel

Examples

```
## Not run:
#**********************************************************************************
# Example 1: Defining and modeling P(A | W) with continuous A
data(indSample.iid.cA.cY_list)
indSample.iid.cA.cY <- indSample.iid.cA.cY_list$indSample.iid.cA.cY
nodes <- list(Ynode = "Y", Anodes = "A", WEnodes = c("W1", "W2", "W3", "W4"))
tmleCom_Options(gestimator = "speedglm__glm", maxNperBin = nrow(indSample.iid.cA.cY),
                bin.method = "equal.mass", nbins = 10)
options(tmleCommunity.verbose = FALSE)  # Don't print status messages
#**********************************************************************************

#**********************************************************************************
# 1.1 Defining new R6 objects of DatKeepClass and RegressionClass and GenericModel
#**********************************************************************************
OData.g0 <- DatKeepClass$new(Odata = indSample.iid.cA.cY, nodes = nodes)
h.g0.sVars <- tmleCommunity:::define_regform(NULL, Anodes.lst = nodes$Anodes,
                                             Wnodes.lst = nodes$WEnodes)
subsets_expr <- lapply(h.g0.sVars$outvars, function(var) {var})
regclass.g0 <- RegressionClass$new(outvar = h.g0.sVars$outvars,
                                   predvars = h.g0.sVars$predvars,
                                   subset_vars = subsets_expr,
                                  outvar.class = OData.g0$type.sVar[h.g0.sVars$outvars])
genericmodels.g0 <- GenericModel$new(reg = regclass.g0, DatKeepClass.g0 = OData.g0)

#**********************************************************************************
# 1.2 Details regarding the GenericModel of the first exposure variable
#**********************************************************************************
genericmodels.g0.A1 <- genericmodels.g0$getPsAsW.models()$`P(A|W).1`
genericmodels.g0$getPsAsW.models()$`P(A|W).2`  # NULL as only one A in the input data
# Defining the number and positions of the bins via arguments in tmleCom_Options()
genericmodels.g0.A1$intrvls
# Creating a matrix of dummy bin indicators for continuous A
OData.g0$binirize.sVar(name.sVar = genericmodels.g0.A1$outvar,
                       intervals = genericmodels.g0.A1$intrvls,
                       nbins = genericmodels.g0.A1$reg$nbins,
                       bin.nms = genericmodels.g0.A1$bin_nms)
bin.ind.mat <- OData.g0$dat.bin.sVar
colSums(bin.ind.mat, na.rm = TRUE)   # Each bin has 1000 obs as "equal.mass" with 10 bins
```

```
#*****************************************************************************************
# 1.3 Fitting regression models for the first exposure variable
#*****************************************************************************************
genericmodels.g0.A1$fit(data = OData.g0)
genericmodels.g0.A1.B2 <- genericmodels.g0.A1$getPsAsW.models()$`P(A|W).2`  # 2nd bin
genericmodels.g0.A1.B2$getfit$coef

## End(Not run)
```

`indSample.iid.bA.bY.rareJ1_list`

An Example of a Non-Hierarchical Data Containing a Binary Exposure with a Rare Outcome (Independent Case-Control J = 1)

Description

Simulated (non-hierarchical) dataset containing 2,000 i.i.d. observations, with each row `i` consisting of 4 measured baseline covariates (`W1`, `W2`, `W3` and `W4`), 1 binary exposure (`A`) and 1 binary outcome (`Y`) that defines case or control status. The baseline covariates `W1`, `W2`, `W3` and `W4` were sampled as i.i.d., while the exposure `A` for each observation `i` depends on `i`'s four baseline covariates. Similarly, the outcome `Y` for each observation depends on `i`'s baseline covariates and exposure values. Moreover, we can also describe the case-control design as first sampling 1 case $(W_1^1, W_2^1, W_3^1, W_4^1, A^1)$ from the conditional distribution of (W_1, W_2, W_3, W_4, A), given Y = 1. One then samples J controls $(W_1^{0,j}, W_2^{0,j}, W_3^{0,j}, W_4^{0,j}, A^{0,j})$ from (W_1, W_2, W_3, W_4, A), given Y = 0, $j = 1, ..., J$. Thus, the cluster containing one case and `J` controls is considered the experimental unit. Finally one gets nC cases and nCo controls with $J = nC/nCo$, where J can be used effectively in observation weights. The following section provides more details regarding individual variables in simulated data.

Usage

```
data(indSample.iid.bA.bY.rareJ1_list)
```

Format

A data frame with 2,000 independent observations (rows), containing 1000 cases and 1000 controls, and 6 variables:

W1 continuous uniform baseline covariate with `min=0` and `max=1`

W2 continuous normal baseline covariate with μ = 0 and σ = 0.3

W3 binary baseline covariate with $P(W2 = 1) = 0.5$

W4 binary baseline covariate with $P(W2 = 1) = 0.5$

A binary exposure that depends on baseline covariate values in `(W1, W2, W3, W4)`

Y binary outcome that depends on baseline covariate and exposure values in `(W1, W2, W3, W4, A)`

Source

https://github.com/chizhangucb/tmleCommunity/blob/master/tests/dataGeneration/get.iid.dat.Acont.R

Examples

```
data(indSample.iid.bA.bY.rareJ1_list)
indSample.iid.bA.bY.rareJ1 <- indSample.iid.bA.bY.rareJ1_list$indSample.iid.bA.bY.rareJ1
head(indSample.iid.bA.bY.rareJ1_list$obs.wt.J1)  # Assigned weights to each observations
indSample.iid.bA.bY.rareJ1_list$q0  # 0.013579 True prevalence probability
indSample.iid.bA.bY.rareJ1_list$psi0.Y  # 0.012662 True ATE
indSample.iid.bA.bY.rareJ1_list$J  # 1 The ratio of number of controls to cases
```

---

indSample.iid.bA.bY.rareJ2_list

An Example of a Non-Hierarchical Data Containing a Binary Exposure with a Rare Outcome (Independent Case-Control J = 2)

Description

Simulated (non-hierarchical) dataset containing 3,000 i.i.d. observations. The data structure of `indSample.iid.bA.bY.rareJ2` is identical to this of `indSample.iid.bA.bY.rareJ1`, except that now the ratio of the number of controls to the number of case J is 2.

Usage

```
data(indSample.iid.bA.bY.rareJ2_list)
```

Format

A data frame with 3,000 independent observations (rows), containing 1000 cases and 2000 controls, and 6 variables

---

indSample.iid.cA.cY_list

An Example of a Non-Hierarchical Data Containing a Continuous Exposure with a Continuous Outcome.

Description

Simulated (non-hierarchical) dataset containing 10,000 i.i.d. observations, with each row `i` consisting of measured baseline covariates (`W1`, `W2`, `W3` and `W4`), continuous exposure (`A`) and continous outcome (`Y`). The baseline covariates `W1`, `W2`, `W3` and `W4` were sampled as i.i.d., while the value of exposure `A` for each observation `i` was drawn conditionally on the value of `i`'s four baseline covariates. Besides, the continuous outcome `Y` for each observation depends on `i`'s baseline covariates and exposure values in (`W1[i]`,`W2[i]`, `W3[i]`, `W4[i]`, `A[i]`). The following section provides more details regarding individual variables in simulated data.

Usage

```
data(indSample.iid.cA.cY_list)
```

Format

A data frame with 10,000 independent observations (rows) and 6 variables:

W1 binary baseline covariate with $P(W1 = 1) = 0.5$

W2 binary baseline covariate with $P(W2 = 1) = 0.3$

W3 continuous normal baseline covariate with $\mu = 0$ and $\sigma = 0.25$

W4 continuous uniform baseline covariate with `min=0` and `max=1`

A continuous normal exposure where its mean depends on individual's baseline covariate values in `(W1, W2, W3, W4)`

Y continuous normal outcome where its mean depends on individual's baseline covariate and exposure values in (`W1`, `W2`, `W3`, `W4`, `A`)

Source

https://github.com/chizhangucb/tmleCommunity/blob/master/tests/dataGeneration/get.iid.dat.Acont.R

Examples

```
data(indSample.iid.cA.cY_list)
indSample.iid.cA.cY <- indSample.iid.cA.cY_list$indSample.iid.cA.cY
# True mean of outcome under intervention g0
psi0.Y <- indSample.iid.cA.cY_list$psi0.Y
# True mean of outcoem under stochastic intervention gstar
psi0.Ygstar <- indSample.iid.cA.cY_list$psi0.Ygstar
# truncated bound used in sampling A* under gstar (in data generating mechanism)
indSample.iid.cA.cY_list$truncBD
# shift value used in sampling A* under gstar
indSample.iid.cA.cY_list$shift.val
```

`MonteCarloSimClass` *R6 class for evaluating different plug-in estimators via Monte-Carlo resampling where new exposures are generated under the user-specified arbitrary intervention function g*.*

Description

`MonteCarloSimClass` only resamples `A` under the intervention function `g_star`, never `W` or `E`. For each MC simulation, it firstly treats `model.Q.init` as the fitted models for `E[Y|A,W,E]`, then estimate `psi_n` using Monte-Carlo integration. i.e., average of `n` predicted `E(Y|A=a, W=w,E=e)` where `a` is a vector of `n` new exposures randomly drawn under `g_star`. Take as many iterations as needed until convergence of ψ_n^I (or ψ_n^{II}) occurs.

Usage

```
MonteCarloSimClass
```

Format

An `R6Class` generator object

Details

- OData.ObsP0 - A DatKeepClass class object, where exposures are generated under observed exposure mechanism g0.
- OData.gstar - A DatKeepClass class object, where new exposures are generated under user-specified intervention g^*.
- model.Q.init - A fitted model for E[Y|A,W,E].
- model.Q.star.cov - A targeting model for covariate-based unweighted TMLE.
- model.Q.star.int - A targeting model for intercept-based TMLE.
- QY.init - Estimates of G-COMP.
- QY.star.cov - Estimates of covariate-based unweighted TMLE.
- QY.star.int - Estimates of intercept-based TMLE.
- nobs - Number of observations in the observed data frame.
- p - Number of Monte-Carlo simulations performed.

Methods

new(OData.ObsP0, OData.gstar, ...) Instantiate an new instance of MonteCarloSimClass.

get.gcomp(m.Q.init) Predict QY.init = $E[Y_{g^*}]$ based on the initial model fit model.Q.init.

get.tmleCov(model.Q.star.cov, model.h.fit) Update QY.init based on the targeting model model.Q.star.cov and the model for clever covriate h model.h.fit.

get.tmleCov(model.Q.star.cov, model.h.fit) Update QY.init based on the targeting model model.Q.star.cov and the model for clever covriate h model.h.fit.

get.tmleInt(model.Q.star.int) Update QY.init based on the targeting model model.Q.star.int.

get.fiW() Get an estimate of fiW (hold ALL W's fixed).

See Also

tmleCom_Options, DatKeepClass, RegressionClass, tmleCommunity

---

panelData_Trans *Panel Dataset Transformation, Using Individual (and Time) Indexes*

Description

panelData_Trans provides a wide variety of ways of data transformation for panel datasets, such as fixid effect and pooling model. It allows users to only apply transformation on regressors of interests, instead of on the entire dataset. See details in https://CRAN.R-project.org/package=plm.

Usage

```
panelData_Trans(data, yvar, xvar, effect = "individual",
  model = "within", index = NULL, transY = TRUE)
```

Arguments

data	A data frame (will be automatically transferred to panel data frame) or a panel data frame
yvar	Column name in data of outcome variable (Currently only support univariate).
xvar	Column names in data of explanatory variables (Including (A, W, E)).
effect	The effects introduced in the model, one of "individual", "time", "twoways" and "nested". Default to "individual".
model	Model of estimation, one of "pooling" (pooled OLS), "within" (fixed effect), "between" (group mean), "random"(random effect), "fd" (first differences) and "ht" (Hausman-Taylor estimator). Default to "within". Notice that when model = "random", "swar" is chosen as the method of estimation for the variance components
index	A vector of two character strings which contains the names of the individual and of the time indices, sequentially. If only individual index is given, treat each observation within a unit as a time point. If no index is given, the first two columns will be automatically treated as individual and time indices, sequentially.
transY	Logical. If TRUE, indicate the outcome variable yvar will also be tranformed. Default to TRUE.

Value

newdata	Transformed panel data

Examples

```
## Not run:
#*******************************************************************************************
# Examples of Panel Data Transformation
data(Grunfeld, package = "plm")
data(comSample.wmT.bA.bY_list)
pData <- comSample.wmT.bA.bY_list$comSample.wmT.bA.bY
#*******************************************************************************************

# 1. Fixed effect transformation ("within") where individual effect is introduced
pData.FE <- panelData_Trans(data = pData, xvar = c("E1", "E2", "W1", "W2", "W3", "A"),
                            yvar = "Y", index = "id", effect = "individual",
                            model = "within", transY = TRUE)
# "E1", E2" and "A" are removed since they are constant in community level
names(pData.FE)  # "Y" "W1" "W2" "W3"
head(pData.FE)

# 2. Same as example 1 but not transforming the outcome variable
pData.FE.fixY <- panelData_Trans(data = pData, xvar = c("E1", "E2", "W1", "W2", "W3", "A"),
                                 yvar = "Y", index = "id", effect = "individual",
                                 model = "within", transY = FALSE)
all.equal(pData.FE.fixY$Y, pData$Y)  # TRUE

# 3. Same as example 2 but different yvar and xvar
pData.FE.fixY.2 <- panelData_Trans(data = pData, xvar = c("E1", "E2", "W1", "W2", "W3"),
                                   yvar = "A", index = "id", transY = FALSE)

# 4. Pooled OLS transformation ("pooling") where individual effect is introduced
pData.pool <- panelData_Trans(data = pData, xvar = c("E1", "E2", "W1", "W2", "W3", "A"),
```

```
                                yvar = "Y", index = "id", effect = "individual",
                                model = "pooling", transY = TRUE)
names(pData.pool)  # Y" "(Intercept)" "E1" "E2" "W1" "W2" "W3" "A"

# 5. Random effect transformation ("random") where time effect is introduced
Grunfeld.RE <- panelData_Trans(yvar = "inv", xvar = c("value", "capital"), data = Grunfeld,
                               effect = "time", model = "random", index = "year")

## End(Not run)
```

`print_tmleCom_opts` *Print Current Option Settings for* `tmleCommunity`

Description

Print Current Option Settings for `tmleCommunity`

Usage

```
print_tmleCom_opts()
```

Value

Invisibly returns a list of `tmleCommunity` options.

See Also

`tmleCom_Options`

`RegressionClass` *R6 class for defining regression models that evaluate multivariate joint conditional density* $P(A|W,E)$ *(or* $P(A|W)$ *if non-hierarchical structure)*

Description

`RegressionClass` provides multiple options used when estimating a conditional density $P(A|W,E)$. `A` can be multivariate, if so, hazard specification will factorize `P(A|W,E)` = `P(A[1],...,A[M]|W,E)` as a sequence `P(A[1]|W,E)` * `P(A[2]|W, E, A[1])` * ... * `P(A[M]|W, E, A[1],...,A[M-1])`, where each of the compoenents `A[m]` can be either binary, categorical or continuous, and each of the conditional densities `P(A[m]|W, E, A[1],...,A[m-1])` will be controlled by a new instance of `GenericModel` class. If `A[m]` binary, `P(A[m]|W, E, A[1],...,A[m-1])` will be esimtated by a user-specific library of candidate algorithms, including parametric estimators such as logistic model with only main terms, and data-adaptive estimator such as super-learner algorithms. If `A[m]` continuous (or categorical), `P(A[m]|W, E, A[1],...,A[m-1])` will then be controlled by a new instance of `ContinModel` class (or `CategorModel` class). Note that each `GenericModel`, `ContinModel` and `CategorModel` class will accompany with an adjunctive clone of a parent `RegressionClass` class. The automatically recursive process of defining new instances of `GenericModel` and cloning `RegressionClass` classes won't stop until the entire sequence of binary regressions that represents `P(A|W,E)` is constructed.

Usage

```
RegressionClass
```

Format

An R6Class generator object

Details

- outvar.class - Character vector of outcome variable classes (of length(outvar)): one of bin, cont, cat.
- outvar - Character vector of regression outcome variable names.
- predvars - Character vector of regression-specific predictor names or a pool of all available predictor names.
- reg_hazard - Logical, hazard fitting method. If TRUE, factorize P(outvar | predvars) into \prod_jP(outvar[j] | predvars) for each j.
- subset_vars - Named list for subset variables/ expression (later converted to logical vector).
- ReplMisVal0 - Logical. If TRUE, user-supplied gvars$misXreplace (Default to 0) will be used to replace all gvars$misval among predictors (Default to TRUE).
- nbins - Number of bins used for estimation of a continuous outvar, defined in ContinModel$new().
- estimator - Character, one of "speedglm__glm" (default), "glm__glm", "h2o__ensemble", "SuperLearner". The estimator for which to fit regression model. For "h2o__ensemble" and "SuperLearner", users can specify the data-adaptive algorithms through tmleCom_Options.
- parfit - Logical. If TRUE, use parallel computing on binary regressions. See foreach::foreach.
- pool_cont - Logical. If TRUE, pool over bins of a continuous outvar and fit one regression, along with bin_ID as an extra variable.
- outvars_to_pool - Character vector of bin names of a continuous outvars, should be identical to bin_nms.
- intrvls - Numeric vector defining the number and positions of the bins or a named list of numeric vectors if 2 or more outvars. If not specified and outvar continuous, intervals will be determined in ContinModel through DatKeepClass$detect.sVar.intrvls.
- intrvls.width - Named numeric vector of bin widths for each bin in self$intrvls. If not specified, default to 1 if outvar binary, default to diff(self$intrvls) if outvar continuous,

Methods

`new(outvar.class = gvars$sVartypes$bin, outvar, predvars, subset_vars, intrvls, ReplMisVal0 = TR`
Instantiate an new instance of RegressionClass

`ChangeManyToOneRegresssion(k_i, reg)` Clone the parent RegressionClass (reg) that include length(self$outvar) regressions, and reset self to a single univariate k_i regression for outcome self$outvar[[k_i]].

`resetS3class()` Reset the object class to "RegressionClass" and "R6".

Active Bindings

`S3class(newclass)` ...

`get.reg` ...

See Also

DatKeepClass, GenericModel, ContinModel, CategorModel

Examples

```
data(indSample.iid.cA.cY_list)
indSample.iid.cA.cY <- indSample.iid.cA.cY_list$indSample.iid.cA.cY
nodes <- list(Ynode = "Y", Anodes = "A", WEnodes = c("W1", "W2", "W3", "W4"))
tmleCom_Options(maxNperBin = nrow(indSample.iid.cA.cY))

OData.g0 <- DatKeepClass$new(Odata = indSample.iid.cA.cY, nodes = nodes)
h.g0.sVars <- tmleCommunity:::define_regform(A ~ W1 + W2 + W3 + W4)
A.nms.g0 <- h.g0.sVars$outvars
regclass.g0 <- RegressionClass$new(outvar = A.nms.g0, predvars = h.g0.sVars$predvars,
                                   subset_vars = lapply(A.nms.g0, function(var) {var}),
                                   outvar.class = OData.g0$type.sVar[A.nms.g0])
regclass.g0$estimator  # "speedglm__glm"
regclass.g0$pool_cont  # FALSE (Don't pool across all bins of a continuous exposure)
regclass.g0$parfit     # FALSE (Don't preform parallel computing in estimation)

# Clone the parent regclass.g0 and reset to a single univariate k_i regression
# for outcome regclass.g0$outvar[[k_i]]
k_i <- 1
reg_i <- regclass.g0$clone()
reg_i$ChangeManyToOneRegresssion(k_i, regclass.g0)
genericmodels.g0.A1 <- ContinModel$new(reg = reg_i, DatKeepClass.g0 = OData.g0)
```

`tmleCommunity` *Estimate Marginal Treatment Effects For Arbitrary (Stochastic) Interventions in Hierarchical Data*

Description

Estimate the marginal treatment effect among independent communities (or i.i.d units if no hierarchical structure) using **TMLE** (targeted maximum likelihood estimation). It supports two different TMLEs that are based on community-level and individual-level analysis, respectively. The individual-level TMLE cooperates with additional working assumptions and has potential efficiency gain. It also provide corresponding **IPTW** (the inverse-probability-of-treatment or Horvitz-Thompson) and **GCOMP** (parametric G-computation).

Usage

```
tmleCommunity(data, Ynode, Anodes, WEnodes, YnodeDet = NULL,
  obs.wts = c("equal.within.pop", "equal.within.community"),
  community.step = c("NoCommunity", "community_level",
  "individual_level", "perCommunity"), communityID = NULL,
  community.wts = c("size.community", "equal.community"),
  pooled.Q = FALSE, f_g0 = NULL, f_gstar1, f_gstar2 = NULL,
  Qform = NULL, Qbounds = NULL, alpha = 0.995,
  fluctuation = "logistic", hform.g0 = NULL, hform.gstar = NULL,
  lbound = 0.005, h.g0_GenericModel = NULL,
  h.gstar_GenericModel = NULL, TMLE.targetStep = c("tmle.intercept",
  "tmle.covariate"), n_MCsims = 1, CI_alpha = 0.05, rndseed = NULL,
  verbose = getOption("tmleCommunity.verbose"))
```

Arguments

data	Observed data, data.frame with named columns, containing WEnodes, Anode, Ynode and possibly communityID, YnodeDet. See "Details".
Ynode	Column name or index in data of outcome variable. Outcome can be either binary or continuous (could be beyond 0 and 1). If Ynode undefined, the left-side of the regression formula in argument Qform will be treated as Ynode.
Anodes	Column names or indices in data of exposure (treatment) variables
WEnodes	Column names or indices in data of individual-level (and possibly community-level) baseline covariates. Factors are not allowed.
YnodeDet	Optional column name or index in data of indicators of deterministic values of outcome Ynode, coded as (TRUE / FALSE) or (1 / 0). If TRUE or 1, value of Ynode is given deterministically / constant.
obs.wts	Optional choice to provide/ construct a vector of individual-level observation (sampling) weights (of length nrow(data)). Currently supports a non-negative numeric vector, "equal.within.pop" (Default) and equal.within.community. If "equal.within.pop", weigh individuals in the entire dataset equally (weigh to be all 1); If "equal.within.community", weigh individuals within the same community equally (i.e., 1 / (number of individuals in each community)).
community.step	Methods to deal with hierarchical data, one of "NoCommunity" (Default), "community_level", "individual_level" and "PerCommunity". If "NoCommunity", claim that no hirerachical structure in data; If "community_level", use community-level TMLE; If "individual_level", use individual-level TMLE cooperating with the assumption of no covariate interference. If "perCommunity", use stratified TMLE. If communityID = NULL, then automatically pool over all communities (i.e., treated it as "NoCommunity"). See "Details".
communityID	Optional column name or index in data of community identifier variable. If known, it can support the three options within community.step: "community_level", "individual_level" and "PerCommunity" (See details for community.step).
community.wts	Optional choice to provide/ construct a matrix of community-level observation weights (where dimension = $J \times 2$, where J = the number of communities). The first column contains the identifiers / names of communities (ie., data[, communityID]) and the second column contains the corresponding non-negative weights. Currently only support a numeric matrix with 2 columns, "size.community" (Default) and "equal.community". If setting community.wts = "size.community", treat the number of individuals within each community as its weight, respectively. If community.wts = "equal.community", assumed weights to be all 1;
pooled.Q	Logical for incorporating hierarchical data to estimate the outcome mechanism. If TRUE, use a pooled individual-level regression for initial estimation of the mean outcome (i.e., outcome mechanism). Default to be FASLE. See "Details".
f_g0	Optional function used to specify model knowledge about value of Anodes. It estimates P(A | W, E) under g0 by sampling a large vector/ data frame of Anode (of length nrow(data)*n_MCsims or number of rows if a data frame) from f_g0 function.
f_gstar1	Either a function or a vector or a matrix/ data frame of counterfactual exposures, dependin on the number of exposure variables. If a matrix/ data frame, its number of rows must be either nrow(data) or 1 (constant exposure assigned to all observations), and its number of columns must be length(Anodes). Note that the column names should match with the names in Anodes. If a vector, it must

be of length nrow(data) or 1. If a function, it must return a vector or a data frame of counterfactual exposures sampled based on Anodes, WEnodes (and possibly communityID) passed as a named argument "data". Thus, the function must include "data" as one of its argument names. The interventions defined by f_gstar1 can be static, dynamic or stochastic. See Exmaples below.

f_gstar2 Either a function or a vector or a matrix/ data frame of counterfactual exposures, dependin on the number of exposure variables. It has the same components and requirements as f_gstar1 has.

Qform Character vector of regression formula for Ynode. If not specified (i.e., NULL), the outcome variable is regressed on all covariates included in Anodes and WEnodes (i.e., Ynode ~ Anodes + WEnodes). See "Details".

Qbounds Vector of upper and lower bounds on Y and predicted value for initial Q. Default to the range of Y, widened by 10% of the min and max values. See "Details".

alpha Used to keep predicted values for initial Q bounded away from (0,1) for logistic fluctuation (set Qbounds to (1 - alpha), alpha).

fluctuation Default to "logistic", it could also be "linear" (for targeting step).

hform.g0 Character vector of regression formula for estimating the conditional density of P(A | W, E) under observed treatment mechanism g0. If not specified, its form will be Anodes ~ WEnodes. If there are more than one expsosure, it fits a joint probability. See section "Modeling P(A | W, E) for covariates (A, W, E)".

hform.gstar Character vector of regression formula for estimating the conditional density P(A | W, E) under user-supplied interventions f_gstar1 or f_gstar2. If not specified, it use the same regression formula as used in hform.g0.

lbound Value between (0,1) for truncation of predicted P(A | W, E). Default to 0.005

h.g0_GenericModel
An object of GenericModel **R6** class containing the previously fitted models for P(A | W, E) under observed treatment mechanism g0, one of the returns of tmleCommunity function. If known, predictions for P(A=a | W=w, E=e) under g0 are based on the fitted models in h.g0_GenericModel.

h.gstar_GenericModel
An object of GenericModel **R6** class containing the previously fitted models for P(A^* | W, E) under intervention gstar, one of the returns of tmleCommunity function. If known, predictions for P(A=a | W=w, E=e) under gstar are based on the fitted models in h.gstar_GenericModel.

TMLE.targetStep
TMLE targeting step method, either "tmle.intercept" (Default) or "tmle.covariate". See "Details".

n_MCsims Number of simulations for Monte-Carlo analysis. Each simulation generates new exposures under f_gstar1 or f_gstar2 (if specified) or f_g0 (if specified), with a sample size of nrow(data). Then these generated expsosures are used when fitting the conditional densities P(A | W, E). and estimating for **IPTW** and **GCOMP** under intervention f_gstar1 or f_gstar2. Note that deterministic intervention only needs one simulation and stochastic intervention could use more simulation times such as 10 (Default to 1).

CI_alpha Significance level (alpha) used in constructing a confidence interval. Default to 0.05.

rndseed Random seed for controlling sampling A under f_gstar1 or f_gstar2 (for reproducibility of Monte-Carlo simulations)

verbose Flag. If TRUE, print status messages. Default to getOption("tmleCommunity.verbose") (Default to FALSE). It can be turned on by setting options(tmleCommunity.verbose = TRUE).

Details

The estimates returned by `tmleCommunity` are of a treatment-specific mean, $E[Y_{g^*}]$, the expected community-level outcome if all communities in the target population received the exposures generated under the user-supplied (deterministic or stochastic) intervention g^*.

`data` must be a data frame (no matrix accepted) and does not support factor values (considering removing or recording such columns as strings), `data` includes the following (optional) columns:

- community-level and individual-level baseline covariate columns (`WEnodes`): can be any numeric data. Notice that W represents individual-level covariates and E represent community-level covariates.
- exposure columns (`Anodes`): can have more than one exposure and each exposure could be can be either binary, categorical or continuous.
- outcome column (`Ynode`): can be any numeric data. If `Ynode` values are continuous, they may be automatically scaled. See details for `Qbounds` below.
- deterministic `Ynode` indicator column (`YnodeDet`): (optional) column that must be logical or binary. Only non-determinstic `Ynode` values will be used in the final estimation step (e.g., `IPTW[!determ.Q] <- Y[!determ.Q] * h_wts[!determ.Q]`).
- community identifier variable column (`communityID`): (optional) column that stores community identifier for hierarchical data (or subject identifier if multiple observations for the same individual). Integer or character recommended (No factor allowed). `NULL` means no hierarchical structure and all distinct individuals.

`community.step` specifies how to read the structure of the input data (as hierarchical or non-hierarchical) and how to analyze the data. community_level TMLE (`"community_level"`) is exactly analogous to the TMLE of the treatment specific individual-level mean outcome (`"NoCommunity"`) with the trivial modification that the community rather than the individual serves as the unit of analysis. `communityID` is needed when using `"community_level"`, `"individual_level"` and `"perCommunity"`. Lack of `communityID` forces the algorithm to automatically pool data over all communities and treat it as non-hierarchical dataset (so force `community.step = "NoCommunity"`). If `community.step = "individual_level"`, it incorporates with working models that assume that each individual's outcome is known not to be affected by the covariates of other individuals in the same community (i.e., "no covariate interference"). This strong assumption can be relaxed by integrating knowledge of the dependence structure among individuals within communities (i.e., "weak covariate interference"). But currrently only the "no covariate interference" assumption is implemented. If `community.step = "perCommunity"`), run a single TMLE on each community and calculate a (weighted) mean outcome for the J communities.

`pooled.Q` is in regard to incorporate the working model of "no covariate inference" in community-level `TMLE` (and the corresponding `IPTW` and `GCOMP`) although the working model is not assumed to hold. In other words, when `community.step = "community_level"`, if `pooled.Q = TRUE`, add pooled individual-level regressions as candidates in the Super Learner library for initial estimation of the outcome mechanism. If `pooled.Q = FALSE`, both outcome and treatment mechanisms are estimated on the community-level (no use of individual-level information).

`Qform` should be `NULL`, in which cases all parent nodes of Y node will be used as regressors, or a character vector that can be coerced to class `"formula"`. If `Qestimator` (an argument in `tmleCom_Options`) is `"speedglm__glm"` (or `"glm__glm"`), then `speedglm` (or `glm`) will be called using the components of `Qform`. If `Qestimator` is `"SuperLearner"`, then `SuperLearner` will be called after a data frame is created using `Qform`, based on the specified algorithms in `SL.library` (an argument in `tmleCom_Options`); If `Qestimator` is `"h2o__ensemble"`, then `h2o` and `h2oEnsemble` will be called after a H2OFrame dataset is creating using `Qform`, based on specified algorithms in `h2olearner` and `h2ometalearner`. See "Arguments" in `tmleCom_Options`.

hform.g0 and hform.gstar should also be NULL, in which cases all parent nodes of A node(s) will be used as regressors, or a character vector that can be coerced to class "formula". It follows the same rules applied to Qform except it's now based on gestimator (an argument in tmleCom_Options). Besides, gestimator can be "sl3_pipelines". If gestimator is "sl3_pipelines", then learner$train will be called after a sl3 learner has been created by using make_learner. See "Arguments" in tmleCom_Options.

Qbounds can be used to bound continuous outcomes Y. If Qbounds not specified (NULL), it will be set to the range of Y, widened by 10% of the minimum and maximum. That is, [0.9*min(Y), 1.1*max(Y)]. If specified, then Y will be truncated at the min max values of Qbounds, and then scaled to be in [0, 1] by (Y - min(Qbound))/(diff(Qbound)). Statistical inferences and for the transformed outcome can be directly translated back into statistical inference for the unscaled outcome. Once Qbounds finish bounding the observed outcomes, it will be set to (1 - alpha, alpha) and used to bound the predicted values for the initial outcome mechanism. Thus, alpha needs to be between (0, 1), otherwise reset to 0.995. Besides, lbound can be used to truncate the weights h_gstar/h_gN, that is, [0, 1/lbound].

TMLE.targetStep specifies how to use weights h_gstar/h_gN in the **TMLE** targeting step. If tmle.intercept (default), it performs the weighted intercept-based TMLE that runs a intercept-only weighted logistic regression using offsets logit(Qstar) and weights h_gstar/h_gN and so no covariate. If tmle.covariate, it performs the unweighted covariate-based TMLE that run a unweighted logistic regression using offsets logit(Qstar) and a clever covariate h_gstar/h_gN.

Value

tmleCommunity returns an object of class "tmleCommunity", which is a named list containing the estimation results stored by the following 3 elements:

- EY_gstar1 - a list with estimates of the mean counterfactual outcome under (deterministic or stochastic) intervention function f_gstar1 ($E[Y_{g_1^*}]$).
- EY_gstar2 - a list with estimates of the mean counterfactual outcome under (deterministic or stochastic) intervention function f_gstar2 ($E[Y_{g_2^*}]$); or NULL if f_gstar2 not specified.
- ATE - a list with estimates of additive treatment effect ($E[Y_{g_1^*}]$ - $E[Y_{g_2^*}]$) under two interventions functions f_gstar1 VS f_gstar2; or NULL if f_gstar2 not specified.

Each element in the returned tmleCommunity object is itself a list containing the following 8 items:

- estimates - matrix, 3×1, storing 3 algorithm estimates of the target parameter (population community-level counterfactual mean under (deterministic or stochastic) intervention), including TMLE, IPTW and GCOMP.
- vars - matrix, 3×1, storing 3 influence-curve based asymptotic variance estimates for TMLE, IPTW and GCOMP. Notice, all IC-based statistical inference for **GCOMP** is not accurate (Just for reference). See explanation in IC.
- CIs - matrix, 3×2, storing 3 confidence interval estimates at CI_alpha level, for TMLE, IPTW and GCOMP. The first column contains the lower bounds and the second column contains the upper bounds.
- tstat - matrix, 3×1, storing 3 test statistics.
- pval - matrix, 3×1, storing 3 p-values. It's based on the Student's T distribution if the number of communities (or the number of individuals if no hierarchical structure) is less than 41, otherwise based on the Z normal distribution.
- IC - data frame, nobs×3, the first column contains the influence curves (ICs) for TMLE estimate, the second column contains the ICs for IPTW estimate, and the third column contains the ICs for GCOMP estimate (not accurate since it is based on ICs for TMLE estimate without updating step).

- `h.g0_GenericModel` - An object of `GenericModel` **R6** class, storing the model fits for P(A | W, E) under observed exposure mechanism `g0`. This can be used in `tmleCommunity` (See **Arguments**).
- `h.gstar_GenericModel` - An object of `GenericModel` **R6** class, storing the model fits for P(A | W, E) under intervention `f_gstar1` or `f_gstar2`. This can be used in `tmleCommunity` (See **Arguments**).

Estimations are based on either community-level or individual-level analysis. Each analysis currently implements 3 estimators:

- `tmle` - Either weighted intercept-based TMLE based on weights `h_gstar/h_gN` (default choice) or unweighted covariate-based TMLE based on a covariate `h_gstar/h_gN`.
- `iptw` - IPTW (Horvitz-Thompson) estimator based on the `TMLE` weights h_gstar/h_gN.
- `gcomp` - Maximum likelihood based G-computation substitution estimator.

IPTW estimator

The IPTW estimator is based on the TMLE weights $h_{g^*}(A^*, W, E)/h_g(A, W, E)$, which is equivalent to $P_{g^*}(A^*|W, E)/P_g(A|W, E)$ and is defined as the the weighted average of the observed outcomes Y. The following algorithm shows a general template of the community-level `IPTW`:

- As described in the following section ("Modeling `P(A | W, E)` for covariates `(A, W, E)`"), the first step is to construct an estimator $P_{\hat{g}^c}(A|W, E)$ of the density for the common (in j) conditional distribution of A given W, E, that is $P_{g_0^c}(A|W, E)$ for common (in j) community-level covariates (A, W, E).
- Implementing the same modeling & fitting algorithm to construct an estimator $P_{\hat{g}^{c*}}(A^*|W, E)$ of the density for the common (in j) conditional distribution of A^* given (W, E), that is $P_{g_0^{c*}}(A^*|W, E)$ for common (in j) community-level covariates (A^*, W, E) where A^* is determined by the user-supplied stochastic intervention `f_gstar1` or `f_gstar2`, given the observed baseline covariates (W, E).
- Given observed J independent communities $\mathbf{O}_j = (E_j, \mathbf{W}_j, A_j, Y_j^c : j = 1, ..., J)$, the IPTW estimator is given by:

$$\psi^I_{IPTW,n} = \frac{1}{J}\sum_{j=1}^{J} Y_j^c \frac{P_{\hat{g}^{c*}}(A_j^*|\mathbf{W}_j, E_j)}{P_{\hat{g}^{c*}}(A_j|\mathbf{W}_j, E_j)}$$

For individual-level IPTW, it reads the input data as $O_{i,j} = (E_j, W_{i,j}, A_j, Y_{i,j} : j = 1, ..., J; i = 1, ..., n_j)$ and incorporates working model that assumes no covariate interference, weighing each individual within one community by $\alpha_{i,j}$, where the IPTW estimator is given by:

$$\psi^{II}_{IPTW,n} = \frac{1}{J}\sum_{j=1}^{J}\sum_{i=1}^{n_j} \alpha_{i,j} Y_{i,j} \frac{P_{g^*}(A_j^*|W_{i,j}, E_j)}{P_{g^*}(A_j|W_{i,j}, E_j)}$$

TMLE estimator

The TMLE estimator is based on the updated model prediction $\bar{Q}^*(A, W, E)$ and is defined by the G-formula. The following algorithm shows a general template of the community-level `TMLE`:

- The first step is exactly the same as `IPTW`: construct two density estimators and use the ratio of them as the weights $P_{g^*}(A^*|W, E)/P_g(A|W, E)$ in the targeting step.

- Construct an initial estimator $\hat{\bar{Q}}^c(A|W,E)$ of the common (in j) conditional distribution of Y^c given (A,W,E) and update $\hat{\bar{Q}}^{c*}(A|W,E)$ for $\hat{\bar{Q}}^c(A|W,E)$ by weights calculated in the first step.
- The TMLE estimator is defined as the following substitution estimator:

$$\psi^I_{TMLE,n} = \frac{1}{J}\sum_{j=1}^{J}\int_a \hat{\bar{Q}}^{c*}(a,\mathbf{W}_j,E_j)g^{c*}(a|\mathbf{W}_j,E_j)d\mu(a)$$

For individual-level TMLE, its estimator is obtained as:

$$\psi^{II}_{TMLE,n} = \frac{1}{J}\sum_{j=1}^{J}\sum_{i=1}^{n_j}\alpha_{i,j}\int_a \hat{\bar{Q}}^{*}(a,W_{i,j},E_j)g^{*}(a|W_{i,j},E_j)d\mu(a)$$

GCOMP estimator

The GCOMP estimator is similar to the the TMLE estimator except it uses the untargeted (initial) model $\hat{\bar{Q}}^c(A|W,E)$ instead of its targeted version $\hat{\bar{Q}}^{c*}(A|W,E)$, for example the community-level GCOMP estimator is given by:

$$\psi^I_{GCOMP,n} = \frac{1}{J}\sum_{j=1}^{J}\int_a \hat{\bar{Q}}^{c}(a,\mathbf{W}_j,E_j)g^{c*}(a|\mathbf{W}_j,E_j)d\mu(a)$$

Modeling $P(A|W,E)$ for covariates (A,W,E)

For simplicity (and without loss of generality), we now suppose that there is no hierarchical structure in data and are interested in finding an non-parametric estimator of the common (in i) **individual-level** exposure mechanism $g_0(A|W)$, or the commom multivariate joint conditional probability model $P_{g_0}(A|W)$, where the exposures and baseline covariates ($(A,W)=(A_i,W_i:i=1,...,n)$) denote the random variables drawn jointly from distribution $H_0(A,W)$ with denisty $h_0(a,w) \equiv g_0(a|w)q_{W,0}(w)$ and $q_{W,0}(W)$ denotes the marginal density of the baseline covariates W specified by the regression formula `hform.g0` (Notice that an non-parametric estimator of the model $P_{g^*_0}(A|W)$) is similar, except that now the exposures and baseline covariates ($(A^*,W)=(A^*_i,W_i:i=1,...,n)$) are randomly drawn from $H^*_0(A,W)$ with density $h^*_0(a,w) \equiv g^*_0(a|w)q_{W,0}(w)$, where A^* is determined by the user-supplied (stochastic) intervention `f_gstar1` or `f_gstar2` and $q_{W,0}(W)$ denotes the marginal density of the baseline covariates W specified by the regression formula `hform.gstar`. Thus, the fitting algorithm for $P_{g^*_0}(A|W)$) is equivalent for $P_{g_0}(A|W)$).

Note that A can be multivariate (i.e., $(A(1),...,A(M))$) and each of its components $A(m)$ can be either binary, categorical or continuous. The joint probability model for $P(A|W)=P(A(1),...,A(M)|W)$ can be factorized as a sequence $P(A(1)|W)\times P(A(2)|W,A(1))\times ...\times P(A(M)|W,A(1),...,A(M-1))$, where each of these conditional probability models $P(A(m)|W,A(1),...,A(m-1))$ is fit separately, depending on the type of the m-specific outcome variable $A(m)$. For binary $A(m)$, the conditional probability $P(A(m)|W,A(1),...,A(m-1))$ will be esimtated by a user-specific library of candidate algorithms, including parametric estimators such as logistic model with only main terms, and data-adaptive estimator such as super-learner algorithms. For continuous (or categorical) $A(m)$, consider a sequence of values $\delta_1,\delta_2,...,\delta_{K+1}$ that span the range of A and define K bins and the corresponding K bin indicators $(B_1,...,B_K)$, in which case each bin indicator $B_k \equiv [\delta_k,\delta_{k+1})$ is used as an binary outcome in a seperate user-specific library of candidate algorithms, with predictors given by $(W,A(1),...,A(m-1))$. That is how the joint probability $P(A|W)$ is factorized into such an entire tree of binary regression models.

For simplicity (and without loss of generality), we now suppose A is univariate (i.e., M=1) and continuous and a general template of an fitting algorithm for $P_{g_0}(A|W)$ is summrized below:

1. Consider the usual setting in which we observe n independently and identically distributed copies $o_i = (w_i, a_i, y_i : i = 1, .., n)$ of the random variable $O = (W, A, Y)$, where the observed $(a_i : i = 1..., n)$ are continuous.
2. As described above, consider a sequence of $K + 1$ values that span the support of A values into K bin intervals $\Delta = (\delta_1, \delta_2, ..., \delta_{K+1})$ so that any observed data point a_i belongs to one interval within R, in other words, for each possible value $a \in A$ (even if it is not in the observed $(a_i : i)$), there always exists a $k \in 1, ..., K$ such that $\delta_k \leq a < \delta_{k+1}$, and the length (bandwidth) of the interval can be defined as $bw_k = \delta_{k+1} - \delta_k$. Then let the mapping $S(a) \in \{1, 2, .., K\}$ denote a unique index of the indicator in Λ that a falls in, where $S(a) = k$ if $a \in [\delta_k, \delta_{k+1})$, namely, $\delta_{S(a)} \leq a < \delta_{S(a)+1}$. Moreover, we use b_k to denote a binary indicator of whether the observed a belongs to bin k (i.e., $b_k \equiv I(S(a) = k)$ for all $k \leq S(a)$; $b_k \equiv$ NA for all $k > S(a)$). This is similar to methods for censored longitudinal data, which code observations as NA (censored or missing) once the indicator b_k jumps from 0 to 1. Since a is a realization of the random variable A for one individual, the corresponding random binary indicators of whether A belongs to bin k can be denoted by $B_k : k = 1, .., = K$ where $B_k \equiv I(S(A) = k)$ for all $k \leq S(A)$; $B_k \equiv$NA for all $k > S(A)$.
3. Then for each k = 1,...,K, a binary nonparametric regression is used to estimate the conditional probability $P(B_k = 1|B_{k-1} = 0, W)$, which corresponds to the probability of B_k jumping from 0 to 1, given $B_{k-1} = 0$ and tbe baseline covariates W. Note tha for each k, the corresponding nonparametric regression model is fit only among observations that are uncensored (i.e., still at risk of getting $B_k = 1$ with $B_{k-1} = 0$). Note the above conditional probability $P(B_k = 1|B_{k-1} = 0, W)$ is equivalent to $P(A \in [\delta_k, \delta_{k+1})|A \geq \delta_k, W)$, which is the probability of A belongs to the interval $[\delta_k, \delta_{k+1})$, conditional on A does not belong to any intervals before $[\delta_k, \delta_{k+1})$ and W. Then the discrete conditional hazard function for each k is defined as a normalization of the conditional probability using the corresponding interval bandwidth bw_k: $\lambda_k(A, W) = \frac{P(B_k=1|B_{k-1}=0,W)}{bw_k} = \frac{P(A\in[\delta_k,\delta_{k+1})|A\geq\delta_{k+1},W)}{bw_k}$
4. Finally, for any given observation (a,w), we first find out the interval index k to which a belongs (i.e., $k = S(a) \in 1, ..., K$). Then the discretized conditional density of $P(A = a|W = w)$ can be evaluated by $\lambda_k(A, W) \times [\prod_{j=1}^{k-1}(1 - \lambda_j(A, W))]$, which corresponds to the conditional probability of a belongs to the interval $[\delta_k, \delta_{k+1})$ and does not belong to any intervals before, given W.

Three methods for choosing bin (interval) locations for a univariate and continuous variable A

Note that the choice of the values $\delta_k(k = 1, .., K)$ implies defining the number and positions of the bins. First, a cross- validation selector can be applied to data-adaptively select the candidate number of bins, which minimizes variance and maximizes precision (Do not recommend too many bins due to easily violating the positivity assumption). Then, we need to choose the most convenient locations (cuttoffs) for the bins (for fixed K). There are 3 alternative methods that use the histogram as a graphical descriptive tool to define the bin cutoffs $\Delta = (\delta_1, ..., \delta_K, \delta_{K+1})$ for a continuous variable A. In **tmleCommunity** package, the choice of methods bin.method together with the other discretization arguments in function tmleCom_Options() such as nbins (total number of bins) and maxNperBin (the maximum number of observations in each bin), can be used to define the values of bin cutoffs. See help(tmleCom_Options) for more argument details.

- equal.mass: The default discretization method, equal mass (aka equal area) interval method, set by passing an argument bin.method="equal.mass" to tmleCom_Options() prior to calling the main function tmleCommunity(). The interval are defined by spanning the support of A into non-equal length of bins, each containing (approximately) the same number of observations. It is data-adaptive since it tends to be wide where the population density is small, and narrow where the density is large. If nbins is NA (or is smaller than n/maxNperBin), nbins will be (re)set to the interger value of n/maxNperBin where n is the total number of

observations in A, and the default setting of maxNperBin is 500 observations per interval. This method could identify spikes in the density, but oversmooths in the tails and so could not discover outliers.

- equal.len: equal length interval method, set by passing an argument bin.method="equal.len" to tmleCom_Options() prior to calling tmleCommunity(). The intervals are defined by spanning the support of A into nbins number of equal length of bins. This method describes the tails of the density and identifies outliers well, but oversmooths in regions of high density and so is poor at identifying sharp peaks.
- dhist: (named for diagonally cut histogram) Combination of equal-area equal length and equal mass method, set by passing an argument bin.method="dhist" to tmleCom_Options() prior to calling tmleCommunity(). For consistency, We choose the slope a = $5 \times \text{IQR}(A)$ as suggested by Denby and Mallows ("Variations on the Histogram" (2009)). For more details, please also see this paper.

See Also

tmleCommunity-package for the general description of the package,

tmleCom_Options for additional parameters that control the estimation in tmleCommunity,

DatKeepClass for details about storing, managing, subsetting and manipulating the input data,

indSample.iid.cA.cY_list for an example of a continuous exposure and its estimation,

BinaryOutModel, RegressionClass, GenericModel, MonteCarloSimClass

Examples

```
## Not run:
#******************************************************************************************
# Example 1: Hierarchical example, with one binary A and bianry Y
# True ATE of the community-based treatment is approximately 0.103716
data(comSample.wmT.bA.bY_list)  # load the sample data
comSample.wmT.bA.bY <- comSample.wmT.bA.bY_list$comSample.wmT.bA.bY
N <- NROW(comSample.wmT.bA.bY)
Qform.corr <- "Y ~ E1 + E2 + W2 + W3 + A" # correct Q form
gform.corr <- "A ~ E1 + E2 + W1"  # correct g
#******************************************************************************************

#******************************************************************************************
# 1.1 Estimating the additive treatment effect (ATE) for two deterministic interventions
# (f_gstar1 = 1 vs f_gstar2 = 0) via community-level / individual-level analysis.
# speed.glm using correctly specified Qform, hform.g0 and hform.gstar;
#******************************************************************************************
# Setting global options that may be used in tmleCommunity(), e.g., using speed.glm
tmleCom_Options(Qestimator = "speedglm__glm", gestimator = "speedglm__glm", maxNperBin = N)

# Community-level analysis without a pooled individual-level regression on outcome
tmleCom_wmT.bA.bY.1a_sglm <-
  tmleCommunity(data = comSample.wmT.bA.bY, Ynode = "Y", Anodes = "A",
                WEnodes = c("E1", "E2", "W1", "W2", "W3"), f_gstar1 = 1L, f_gstar2 = 0L,
                community.step = "community_level", communityID = "id", pooled.Q = FALSE,
                Qform = Qform.corr, hform.g0 = gform.corr, hform.gstar = gform.corr)

# Examples of estimates under f_gstar1 = 1:
tmleCom_wmT.bA.bY.1a_sglm$EY_gstar1$estimates
tmleCom_wmT.bA.bY.1a_sglm$EY_gstar1$vars
tmleCom_wmT.bA.bY.1a_sglm$EY_gstar1$CIs
```

```
# Examples of estimates under f_gstar0 = 0:
tmleCom_wmT.bA.bY.1a_sglm$EY_gstar2$estimates
tmleCom_wmT.bA.bY.1a_sglm$EY_gstar2$vars
tmleCom_wmT.bA.bY.1a_sglm$EY_gstar2$CIs

# Examples of estimates for ATE under f_gstar1 - f_gstar0:
tmleCom_wmT.bA.bY.1a_sglm$ATE$estimates
tmleCom_wmT.bA.bY.1a_sglm$ATE$vars
tmleCom_wmT.bA.bY.1a_sglm$ATE$CIs
head(tmleCom_wmT.bA.bY.1a_sglm$ATE$IC)

# Community-level analysis with a pooled individual-level regression on outcome
tmleCom_wmT.bA.bY.1b_sglm <-
  tmleCommunity(data = comSample.wmT.bA.bY, Ynode = "Y", Anodes = "A",
                WEnodes = c("E1", "E2", "W1", "W2", "W3"), f_gstar1 = 1L, f_gstar2 = 0L,
                community.step = "community_level", communityID = "id", pooled.Q = TRUE,
                Qform = Qform.corr, hform.g0 = gform.corr, hform.gstar = gform.corr)
tmleCom_wmT.bA.bY.1b_sglm$ATE$estimates

# Individual-level analysis with both individual-level outcome and treatment mechanisms
tmleCom_wmT.bA.bY.2_sglm <-
  tmleCommunity(data = comSample.wmT.bA.bY, Ynode = "Y", Anodes = "A",
                WEnodes = c("E1", "E2", "W1", "W2", "W3"), f_gstar1 = 1L, f_gstar2 = 0L,
                community.step = "individual_level", communityID = "id",
                Qform = Qform.corr, hform.g0 = gform.corr, hform.gstar = gform.corr)
tmleCom_wmT.bA.bY.2_sglm$ATE$estimates

# Failing to provide communityID will automatically set community.step to "NoCommunity"
tmleCom_wmT.bA.bY.NoC_sglm <-
  tmleCommunity(data = comSample.wmT.bA.bY, Ynode = "Y", Anodes = "A",
                WEnodes = c("E1", "E2", "W1", "W2", "W3"), f_gstar1 = 1L, f_gstar2 = 0L,
                community.step = "individual_level", communityID = NULL,
                Qform = Qform.corr, hform.g0 = gform.corr, hform.gstar = gform.corr)
tmleCom_wmT.bA.bY.NoC_sglm$ATE$estimates

# Stratification analysis that run separate outcome (exposure) mechanism for each community
# use glm since only around 50 observations per community, speed.glm easily fails
# takes longer time than the tests above since doing 1000 TMLEs (one TMLE per community)
# so set verbose to TRUE to track running progress
tmleCom_Options(Qestimator = "glm__glm", gestimator = "glm__glm", maxNperBin = N)
tmleCom_wmT.bA.bY.str_sglm <-
  tmleCommunity(data = comSample.wmT.bA.bY, Ynode = "Y", Anodes = "A",
                WEnodes = c("E1", "E2", "W1", "W2", "W3"), f_gstar1 = 1L, f_gstar2 = 0L,
                community.step = "perCommunity", communityID = "id", verbose = TRUE,
                Qform = Qform.corr, hform.g0 = gform.corr, hform.gstar = gform.corr)
tmleCom_wmT.bA.bY.str_sglm$ATE$estimates

#*******************************************************************************
# 1.2 Same as above but for different Qestimator and gestimator through tmleCom_Options()
# via community-level analysis with a pooled individual-level regression on outcome.
# (See more details in examples in tmleCom_Options())
#*******************************************************************************
# SuperLearner for both outcome and treatment (clever covariate) regressions
# using all parent nodes (of Y and A) as regressors (respectively)
require("SuperLearner")
tmleCom_Options(Qestimator = "SuperLearner", gestimator = "SuperLearner",
```

```
                maxNperBin = N, SL.library = c("SL.glm", "SL.step", "SL.bayesglm"))
tmleCom_wmT.bA.bY.2_SL <-
  tmleCommunity(data = comSample.wmT.bA.bY, Ynode = "Y", Anodes = "A",
                WEnodes = c("E1", "E2", "W1", "W2", "W3"), f_gstar1 = 1L, f_gstar2 = 0L,
                community.step = "community_level", communityID = "id", pooled.Q = TRUE,
                Qform = NULL, hform.g0 = NULL, hform.gstar = NULL)
tmleCom_wmT.bA.bY.2_SL$ATE$estimates

# SuperLearner for outcome regressions and glm for treatment regressions
# using all regressors in the correctly specified Qform and
# all regressors in the misspecified hform.g0 and hform.gstar
tmleCom_Options(Qestimator = "SuperLearner", gestimator = "glm__glm",
                maxNperBin = N, SL.library = c("SL.mean", "SL.stepAIC", "SL.bayesglm"))
tmleCom_wmT.bA.bY.2_SL.glm <-
  tmleCommunity(data = comSample.wmT.bA.bY, Ynode = "Y", Anodes = "A",
                WEnodes = c("E1", "E2", "W1", "W2", "W3"), f_gstar1 = 1L, f_gstar2 = 0L,
                community.step = "community_level", communityID = "id", pooled.Q = TRUE,
                Qform = NULL, hform.g0 = "A ~ W1", hform.gstar = "A ~ E1 + W2")
tmleCom_wmT.bA.bY.2_SL.glm$ATE$estimates

# Speedglm for outcome regressions and sl3 for treatment regressions
# using all regressors in the correctly specified Qform and
# all regressors in the misspecified hform.g0 and hform.gstar
tmleCom_Options(Qestimator = "speedglm__glm", gestimator = "sl3_pipelines", maxNperBin = N,
                sl3_learner = list(glm_fast = sl3::make_learner(sl3::Lrnr_glm_fast)),
                sl3_metalearner = sl3::make_learner(
                  sl3::Lrnr_optim, loss_function = sl3::loss_loglik_binomial,
                  learner_function = sl3::metalearner_logistic_binomial))
tmleCom_wmT.bA.bY.2_speedglm.sl3 <-
  tmleCommunity(data = comSample.wmT.bA.bY, Ynode = "Y", Anodes = "A",
                WEnodes = c("E1", "E2", "W1", "W2", "W3"), f_gstar1 = 1L, f_gstar2 = 0L,
                community.step = "community_level", communityID = "id", pooled.Q = TRUE,
                Qform = NULL, hform.g0 = "A ~ W1", hform.gstar = "A ~ E1 + W2")
tmleCom_wmT.bA.bY.2_speedglm.sl3$ATE$estimates

#*****************************************************************************************
# 1.3 Evaluating mean population outcome under static intervention A = 0
# with different community-level and individual-level weight choices
#*****************************************************************************************
tmleCom_Options(Qestimator = "speedglm__glm", gestimator = "speedglm__glm", maxNperBin = N)
# weigh individuals in data equally & weigh community by its number of individuals
tmleCom_wmT.bA.bY.1a_w1 <-
  tmleCommunity(data = comSample.wmT.bA.bY, Ynode = "Y", Anodes = "A",
                WEnodes = c("E1", "E2", "W1", "W2", "W3"), f_gstar1 = 1L,
                obs.wts = "equal.within.pop", community.wts = "size.community",
                community.step = "community_level", communityID = "id")
tmleCom_wmT.bA.bY.1a_w1$EY_gstar1$estimates

# same as above but weigh individuals within the same community equally
tmleCom_wmT.bA.bY.1a_w2 <-
  tmleCommunity(data = comSample.wmT.bA.bY, Ynode = "Y", Anodes = "A",
                WEnodes = c("E1", "E2", "W1", "W2", "W3"), f_gstar1 = 1L,
                obs.wts = "equal.within.community", community.wts = "size.community",
                community.step = "community_level", communityID = "id")
tmleCom_wmT.bA.bY.1a_w2$EY_gstar1$estimates

# weigh individuals within the same community equally & weigh community equally
```

```
tmleCom_wmT.bA.bY.1a_w3 <-
  tmleCommunity(data = comSample.wmT.bA.bY, Ynode = "Y", Anodes = "A",
                WEnodes = c("E1", "E2", "W1", "W2", "W3"), f_gstar1 = 1L,
                obs.wts = "equal.within.community", community.wts = "equal.community",
                community.step = "community_level", communityID = "id")
tmleCom_wmT.bA.bY.1a_w3$EY_gstar1$estimates

#*********************************************************************************************
# 1.4 Specifying user-supplied stochastic or deterministic intervention function
#*********************************************************************************************
# Intervention function that will sample A with probability P(A=1) = prob.val
define_f.gstar <- function(prob.val, rndseed = NULL) {
  eval(prob.val)
  f.gstar <- function(data, ...) {
    print(paste0("probability of selection: ", prob.val))
    rbinom(n = NROW(data), size = 1, prob = prob.val)
  }
  return(f.gstar)
}
# Stochastically set 50% of the population to A=1
f.gstar_stoch.0.5 <- define_f.gstar(prob.val = 0.5)
# Deterministically set 100% of the population to A=1
f.gstar_determ.1 <- define_f.gstar(prob.val = 1)
# Deterministically set 100% of the population to A=0
f.gstar_determ.0 <- define_f.gstar(prob.val = 0)

#*********************************************************************************************
# 1.5 Equivalent ways of specifying user-supplied (static) intervention (f_gstar1 = 1)
#*********************************************************************************************
# Alternative 1: via intervention functoin that sets every invidual's A to constant 1
tmleCom_wmT.bA.bY.1a_fgtar1 <-
  tmleCommunity(data = comSample.wmT.bA.bY, Ynode = "Y", Anodes = "A",
                WEnodes = c("E1", "E2", "W1", "W2", "W3"), f_gstar1 = f.gstar_determ.1,
                community.step = "community_level", communityID = "id")
tmleCom_wmT.bA.bY.1a_fgtar1$EY_gstar1$estimates

# Alternative 2: by simply setting f_gstar1 to 1
tmleCom_wmT.bA.bY.1a_fgtar2 <-
  tmleCommunity(data = comSample.wmT.bA.bY, Ynode = "Y", Anodes = "A",
                WEnodes = c("E1", "E2", "W1", "W2", "W3"), f_gstar1 = 1L,
                community.step = "community_level", communityID = "id")
tmleCom_wmT.bA.bY.1a_fgtar2$EY_gstar1$estimates

# Alternative 3: by setting f_gstar1 to a vector of 1's of length NROW(data)
tmleCom_wmT.bA.bY.1a_fgtar3 <-
  tmleCommunity(data = comSample.wmT.bA.bY, Ynode = "Y", Anodes = "A",
                WEnodes = c("E1", "E2", "W1", "W2", "W3"),
                f_gstar1 = rep(1L, NROW(comSample.wmT.bA.bY)),
                community.step = "community_level", communityID = "id")
tmleCom_wmT.bA.bY.1a_fgtar1$EY_gstar1$estimates

#*********************************************************************************************
# 1.6 Running exactly the same estimator as 1.1 but using h_gstar/h_gN as a coviariate
# in the targeting step (default to use weighted intercept-based TMLE)
#*********************************************************************************************
# unweighted covariate-based TMLE
tmleCom_wmT.bA.bY.1a_covTMlE <-
```

```
  tmleCommunity(data = comSample.wmT.bA.bY, Ynode = "Y", Anodes = "A",
                WEnodes = c("E1", "E2", "W1", "W2", "W3"), f_gstar1 = 1L, f_gstar2 = 0L,
               community.step = "community_level", communityID = "id", pooled.Q = FALSE,
                TMLE.targetStep = "tmle.covariate",  # default as "tmle.intercept"
                Qform = Qform.corr, hform.g0 = gform.corr, hform.gstar = gform.corr)
tmleCom_wmT.bA.bY.1a_covTMlE$ATE$estimates

#*********************************************************************************************
# 1.7 Equivalent ways of specifying the regression formulae
# (if Ynode is specified as "Y" and WEnodes = c("E1", "E2", "W1", "W2", "W3"))
#*********************************************************************************************
# For outcome regression, the left side of Qform will be ignored if Ynode is specified,
# with dependent variable being set to Ynode.
Qform1 <- "Y ~ E1 + E2 + W2 + W3 + A"
Qform2 <- "AnythingIsFine ~ E1 + E2 + W2 + W3 + A"
Qform3 <- NULL  # since all parent nodes of Y will be used as regressors

# For treatment regressions, if hform.gstar unspecified, it uses the same regression
# formula as hform.g0 does.
# Alternative 1: specify hform.g0 and hform.gstar respectively
hform.g0 <- "A ~ E1 + E2 + W1"
hform.gstar <- "A ~ E1 + E2 + W1"

# Alternative 2: specify hform.g0 only
hform.g0 <- "A ~ E1 + E2 + W1"
hform.gstar <- NULL

#*********************************************************************************************
# 1.8 Equivalent ways of allowing printing status message
#*********************************************************************************************
# Controlling the global setting
options(tmleCommunity.verbose = TRUE)
tmleCom_wmT.bA.bY.print1 <-
  tmleCommunity(data = comSample.wmT.bA.bY, Ynode = "Y", Anodes = "A",
                WEnodes = c("E1", "E2", "W1", "W2", "W3"), f_gstar1 = 1L, f_gstar2 = 0L,
                community.step = "community_level", communityID = "id")

# Alternative: using the verbose argument in tmleCommunity()
tmleCom_wmT.bA.bY.print2 <-
  tmleCommunity(data = comSample.wmT.bA.bY, Ynode = "Y", Anodes = "A",
                WEnodes = c("E1", "E2", "W1", "W2", "W3"), f_gstar1 = 1L, f_gstar2 = 0L,
                community.step = "community_level", communityID = "id", verbose = TRUE)

#*********************************************************************************************
# Example 2: Non-hierarchical example, with one continuous A and continuous Y
# True mean population outcome under stochastic intervention (specified below)
# is approximately 3.50856
data(indSample.iid.cA.cY_list)  # load the sample data
indSample.iid.cA.cY <- indSample.iid.cA.cY_list$indSample.iid.cA.cY
true.shift <- indSample.iid.cA.cY_list$shift.val  # 2
true.truncBD <- indSample.iid.cA.cY_list$truncBD  # 10
N <- NROW(indSample.iid.cA.cY)
Qform.corr <- "Y ~ W1 + W2 + W3 + W4 + A" # correct Q form
gform.corr <- "A ~ W1 + W2 + W3 + W4"  # correct g
#*********************************************************************************************

#*********************************************************************************************
```

```
# 2.1 Specifying stochastic intervention function that could represent the true
# shifted version of the current treatment mechanism
#*********************************************************************************
define_f.gstar <- function(shift.val, truncBD, rndseed = NULL) {
  shift.const <- shift.val
  trunc.const <- truncBD
  f.gstar <- function(data, ...) {
    print(paste0("shift.const: ", shift.const))
    set.seed(rndseed)
   A.mu <- 0.86 * data[,"W1"] + 0.41 * data[,"W2"] - 0.34 * data[,"W3"] + 0.93 * data[,"W4"]
    untrunc.A <- rnorm(n = nrow(data), mean = A.mu + shift.const, sd = 1)
    r.new.A <- exp(0.8 * shift.const * (untrunc.A - A.mu - shift.const / 3))
    trunc.A <- ifelse(r.new.A > trunc.const, untrunc.A - shift.const, untrunc.A)
    return(trunc.A)
  }
  return(f.gstar)
}
# correctly specified stochastic intervention with true shift value and truncated bound
f.gstar.corr <- define_f.gstar(shift = true.shift, truncBD = true.truncBD)
# Misspecified specified stochastic intervention
f.gstar.mis <- define_f.gstar(shift = 5, truncBD = 8)

#*********************************************************************************
# 2.2 Estimating mean population outcome under different stochastic interventions
# speed.glm using correctly specified Qform, hform.g0 and hform.gstar
#*********************************************************************************
tmleCom_Options(Qestimator = "speedglm__glm", gestimator = "speedglm__glm", maxNperBin = N)
# correctly specified stochastic intervention
tmleind_iid.cA.cY_true.fgstar <-
  tmleCommunity(data = indSample.iid.cA.cY, Ynode = "Y", Anodes = "A",
                WEnodes = c("W1", "W2", "W3", "W4"), f_gstar1 = f.gstar.corr,
                Qform = Qform.corr, hform.g0 = gform.corr, hform.gstar = gform.corr)
tmleind_iid.cA.cY_true.fgstar$EY_gstar1$estimates

# misspecified stochastic intervention
tmleind_iid.cA.cY_mis.fgstar <-
  tmleCommunity(data = indSample.iid.cA.cY, Ynode = "Y", Anodes = "A",
                WEnodes = c("W1", "W2", "W3", "W4"), f_gstar1 = f.gstar.mis,
                Qform = Qform.corr, hform.g0 = gform.corr, hform.gstar = gform.corr)
tmleind_iid.cA.cY_mis.fgstar$EY_gstar1$estimates

#*********************************************************************************
# 2.3 Same as above but using larger number of Monte-Carlo simulations
# using all parent nodes (of Y and A) as regressors (respectively)
#*********************************************************************************
# A will be sampled 10 times (for a total sample size of NROW(data)*10 under f_gstar1)
tmleind_iid.cA.cY_10MC <-
  tmleCommunity(data = indSample.iid.cA.cY, Ynode = "Y", Anodes = "A",
              WEnodes = c("W1", "W2", "W3", "W4"), f_gstar1 = f.gstar.corr, n_MCsims = 10)
tmleind_iid.cA.cY_10MC$EY_gstar1$estimates

#*********************************************************************************
# 2.4 Running exactly the same estimator as 2.1 but defining different values of bin cutoffs
#*********************************************************************************
# using equal-length method with 10 bins
tmleCom_Options(bin.method = "equal.len", nbins = 10, maxNperBin = N)
tmleind_iid.cA.cY_len <-
```

```
  tmleCommunity(data = indSample.iid.cA.cY, Ynode = "Y", Anodes = "A",
                WEnodes = c("W1", "W2", "W3", "W4"), f_gstar1 = f.gstar.corr,
                Qform = Qform.corr, hform.g0 = gform.corr, hform.gstar = gform.corr)
tmleind_iid.cA.cY_len$EY_gstar1$estimates

# using combination of equal-length and equal-mass method with 20 bins
tmleCom_Options(bin.method = "dhist", nbins = 20, maxNperBin = N)
tmleind_iid.cA.cY_dhist <-
  tmleCommunity(data = indSample.iid.cA.cY, Ynode = "Y", Anodes = "A",
                WEnodes = c("W1", "W2", "W3", "W4"), f_gstar1 = f.gstar.corr,
                Qform = Qform.corr, hform.g0 = gform.corr, hform.gstar = gform.corr)
tmleind_iid.cA.cY_dhist$EY_gstar1$estimates

#*****************************************************************************************
# 2.5 Estimating the additive treatment effect (ATE) for two stochastic interventions
#*****************************************************************************************
# Intervention function that will shift A by constant rate (shift.rate)
# (A special case of stochastic intervention with constant shift)
define_f.gstar <- function(shift.rate) {
  eval(shift.rate)
  f.gstar <- function(data, ...) {
    print(paste0("rate of shift: ", shift.rate))
    data[, "A"] * shift.rate
  }
  return(f.gstar)
}
f.gstar_shift0.8 <- define_f.gstar(shift.rate = 0.8)
f.gstar_shift0.5 <- define_f.gstar(shift.rate = 0.6)

tmleCom_Options(bin.method = "equal.mass", nbins = 5, maxNperBin = N)
tmleind_iid.cA.cY_ATE <-
  tmleCommunity(data = indSample.iid.cA.cY, Ynode = "Y", Anodes = "A",
                WEnodes = c("W1", "W2", "W3", "W4"),
                f_gstar1 = f.gstar_shift0.8, f_gstar2 = f.gstar_shift0.5,
                Qform = Qform.corr, hform.g0 = gform.corr, hform.gstar = gform.corr)

# ATE estimates for f_gstar1-f_gstar2:
tmleind_iid.cA.cY_ATE$ATE$estimates
tmleind_iid.cA.cY_ATE$ATE$vars
tmleind_iid.cA.cY_ATE$ATE$CIs

#*****************************************************************************************
# Example 3: Non-Hierarchical example, with one binary A and one rare bianry Y
# (Independent case-control)  True ATE is approximately 0.012662
data(indSample.iid.bA.bY.rareJ1_list)
indSample.iid.bA.bY.rareJ1 <- indSample.iid.bA.bY.rareJ1_list$indSample.iid.bA.bY.rareJ1
obs.wt.J1 <- indSample.iid.bA.bY.rareJ1_list$obs.wt.J1
Qform.corr <- "Y ~ W1 + W2*A + W3 + W4" # correct Q form
gform.corr <- "A ~ W1 + W2 + W3 + W4"  # correct g
tmleCom_Options(maxNperBin = NROW(indSample.iid.bA.bY.rareJ1))
#*****************************************************************************************

#*****************************************************************************************
# 3.1 Estimating ATE for f_gstar1 = 1 vs f_gstar2 = 0
# using correct observation weights and correctly specified Qform & gform
#*****************************************************************************************
tmleind_iid.bA.bY_corrWT <-
```

```
  tmleCommunity(data = indSample.iid.bA.bY.rareJ1, Ynode = "Y", Anodes = "A",
                WEnodes = c("W1", "W2", "W3", "W4"), f_gstar1 = 1, f_gstar2 = 0,
                Qform = Qform.corr, hform.g0 = gform.corr, hform.gstar = gform.corr,
                obs.wts = obs.wt.J1, verbose = TRUE)
tmleind_iid.bA.bY_corrWT$ATE$estimates["tmle", ]  # 0.01220298, good estimate

#*********************************************************************************
# 3.2 Same as above but not specifying the observation weights
# obs.wts = NULL is equivalent to obs.wts = "equal.within.pop"
#*********************************************************************************
tmleind_iid.bA.bY_misWT <-
  tmleCommunity(data = indSample.iid.bA.bY.rareJ1, Ynode = "Y", Anodes = "A",
                WEnodes = c("W1", "W2", "W3", "W4"), f_gstar1 = 1, f_gstar2 = 0,
                Qform = Qform.corr, hform.g0 = gform.corr, hform.gstar = gform.corr,
                obs.wts = NULL, verbose = TRUE)
tmleind_iid.bA.bY_misWT$ATE$estimates["tmle", ]  # 0.2466575, bad estimate

## End(Not run)
```

`tmleCom_Options` *Setting all possible options for* `tmleCommunity`

Description

Additional options that control the estimation algorithm in `tmleCommunity` package

Usage

```
tmleCom_Options(Qestimator = c("speedglm__glm", "glm__glm",
  "h2o__ensemble", "SuperLearner"), gestimator = c("speedglm__glm",
  "glm__glm", "h2o__ensemble", "SuperLearner", "sl3_pipelines"),
  bin.method = c("equal.mass", "equal.len", "dhist"), nbins = 5,
  maxncats = 10, maxNperBin = 500, parfit = FALSE,
  poolContinVar = FALSE, savetime.fit.hbars = TRUE,
  h2ometalearner = "h2o.glm.wrapper", h2olearner = "h2o.glm.wrapper",
  sl3_metalearner = sl3::make_learner(sl3::Lrnr_optim, loss_function =
  sl3::loss_loglik_binomial, learner_function =
  sl3::metalearner_logistic_binomial), sl3_learners = list(glm_fast =
  sl3::make_learner(sl3::Lrnr_glm_fast)), CVfolds = 5,
  SL.library = c("SL.glm", "SL.step", "SL.glm.interaction"))
```

Arguments

`Qestimator`	A string specifying default estimator for outcome mechanism model fitting. The default estimator is `"speedglm__glm"`, which estimates regressions with `speedglm.wfit`; Estimator `"glm__glm"` uses `glm.fit`; Estimator `"h2o__ensemble"` implements the super learner ensemble (stacking) algorithm using the H2O R interface; Estimator `"SuperLearner"` implements the super learner prediction methods. alongside a framework for general-purpose machine learning with pipelines. Note that if `"h2o__ensemble"` fails, it falls back on "SuperLearner". If `"SuperLearner"` fails, it falls back on "speedglm__glm". If `"speedglm__glm"` fails, it falls back on "glm__glm".

`gestimator` A string specifying default estimator for exposure mechanism fitting. It has the same options as `Qestimator` except that `gestimator` can also be `"sl3_pipelines"`, which is a modern implementation of the Super Learner algorithm for ensemble learning and model stacking. In such case, if `"h2o__ensemble"` fails, it falls back on "SuperLearner". If `"sl3_pipelines"` fails, it falls back on "SuperLearner", and so on.

`bin.method` Specify the method for choosing bins when discretizing the conditional continuous exposure variable `A`. The default method is `"equal.mass"`, which provides a data-adaptive selection of the bins based on equal mass/ area, i.e., each bin will contain approximately the same number of observations as otheres. Method `"equal.len"` partitions the range of `A` into equal length `nbins` intervals. Method `"dhist"` uses a combination of the above two approaches. Please see Denby and Mallows "Variations on the Histogram" (2009) for more details. Note that argument `maxNperBin` controls the maximum number of observations in each bin.

`nbins` When `bin.method = "equal.len"`, set to the user-supplied number of bins when discretizing a continous variable/ If not specified, then default to 5; If setting to as `NA`, then set to the nearest integer of `nobs/ maxNperBin`, where `nobs` is the total number of observations in the input data. When method is `"equal.mass"`, `nbins` will be set as the maximum of the default `nbins` and the nearest integer of `nobs/ maxNperBin`.

`maxncats` Integer that specifies the maximum number of unique categories a categorical variable `A[j]` can have. If `A[j]` has more unique categories, it is automatically considered a continuous variable. Default to 10.

`maxNperBin` Integer that specifies the maximum number of observations in each bin when discretizing a continuous variable `A[j]` (applies directly when `bin.method = "equal.mass"` and indirectly when `bin.method = "equal.len"`, but `nbins = NA`).

`parfit` Logical. If `TRUE`, perform parallel regression fits and predictions for discretized continuous variables by functions `foreach` and `dopar` in `foreach` package. Default to `FALSE`. Note that it requires registering a parallel backend prior to running `tmleCommunity` function, e.g., using `doParallel` R package and running `registerDoParallel(cores = ncores)` for `ncores` parallel jobs.

`poolContinVar` Logical. If `TRUE`, when fitting a model for binirized continuous variable, pool bin indicators across all bins and fit one pooled regression. Default to `FALSE`.

`savetime.fit.hbars` Logical. If `TRUE`, skip estimation and prediction of exposure mechanism P(A|W,E) under g0 & gstar when `f.gstar1 = NULL` and `TMLE.targetStep = "tmle.intercept"`, and then directly set `h_gstar_h_gN = 1` for each observation. Default to `TRUE`.

`h2ometalearner` A string to pass to `h2o.ensemble`, specifying the prediction algorithm used to learn the optimal combination of the base learners. Supports both h2o and SuperLearner wrapper functions. Default to "h2o.glm.wrapper".

`h2olearner` A string or character vector to pass to `h2o.ensemble`, naming the prediction algorithm(s) used to train the base models for the ensemble. The functions must have the same format as the h2o wrapper functions. Default to "h2o.glm.wrapper".

`CVfolds` Set the number of splits for the V-fold cross-validation step to pass to `SuperLearner` and `h2o.ensemble`. Default to 5.

`SL.library` A string or character vector of prediction algorithms to pass to `SuperLearner`. Default to c("SL.glm", "SL.step", "SL.glm.interaction"). For more available algorithms see `SuperLearner::listWrappers()`. Additional wrapper functions are available at https://github.com/ecpolley/SuperLearnerExtra.

Value

Invisibly returns a list with old option settings.

See Also

print_tmleCom_opts

Examples

```
## Not run:
#*********************************************************************************************
# Example 1: using different estimators in estimation of Q and g mechanisms
#*********************************************************************************************
# 1.1 using speed.glm (and glm)
tmleCom_Options(Qestimator = "speedglm__glm", gestimator = "speedglm__glm")
tmleCom_Options(Qestimator = "speedglm__glm", gestimator = "glm__glm")

# 1.2 using SuperLearner
library(SuperLearner)
# library including "SL.glm", "SL.glmnet", "SL.ridge", and "SL.stepAIC"
tmleCom_Options(Qestimator = "SuperLearner", gestimator = "SuperLearner", CVfolds = 5,
                SL.library = c("SL.glm", "SL.glmnet", "SL.ridge", "SL.stepAIC"))

# library including "SL.bayesglm", "SL.gam", and "SL.randomForest", and split to 10 CV folds
# require("gam"); require("randomForest")
tmleCom_Options(Qestimator = "SuperLearner", gestimator = "SuperLearner", CVfolds = 10,
                SL.library = c("SL.bayesglm", "SL.gam", "SL.randomForest"))

# Create glmnet wrappers with different alphas (the default value of alpha in SL.glmnet is 1)
create.SL.glmnet <- function(alpha = c(0.25, 0.50, 0.75)) {
  for(mm in seq(length(alpha))){
    eval(parse(text = paste('SL.glmnet.', alpha[mm], '<- function(..., alpha = ',
                            alpha[mm], ') SL.glmnet(..., alpha = alpha)', sep = '')),
         envir = .GlobalEnv)
  }
  invisible(TRUE)
}
create.SL.glmnet(seq(0, 1, length.out=3))  # 3 glmnet wrappers with alpha = 0, 0.5, 1
# Create custom randomForest learners (set ntree to 100 rather than the default of 500)
create.SL.rf <- create.Learner("SL.randomForest", list(ntree = 100))
# Create a sequence of 3 customized KNN learners
# set the number of nearest neighbors as 8 and 12 rather than the default of 10
create.SL.Knn <- create.Learner("SL.kernelKnn", detailed_names=TRUE, tune=list(k=c(8, 12)))
SL.library <- c(grep("SL.glmnet.", as.vector(lsf.str()), value=TRUE),
                create.SL.rf$names, create.SL.Knn$names)
tmleCom_Options(Qestimator = "SuperLearner", gestimator = "SuperLearner",
                SL.library = SL.library, CVfolds = 5)

# 1.3 using h2o.ensemble
library("h2o"); library("h2oEnsemble")
# h2olearner including "h2o.glm.wrapper" and "h2o.randomForest.wrapper"
tmleCom_Options(Qestimator = "h2o__ensemble", gestimator = "h2o__ensemble",
                CVfolds = 10, h2ometalearner = "h2o.glm.wrapper",
                h2olearner = c("h2o.glm.wrapper", "h2o.randomForest.wrapper"))

# Create a sequence of customized h2o glm, randomForest and deeplearning wrappers
```

```
h2o.glm.1 <- function(..., alpha = 1, prior = NULL) {
  h2o.glm.wrapper(..., alpha = alpha, , prior=prior)
}
h2o.glm.0.5 <- function(..., alpha = 0.5, prior = NULL) {
  h2o.glm.wrapper(..., alpha = alpha, , prior=prior)
}
h2o.randomForest.1 <- function(..., ntrees = 200, nbins = 50, seed = 1) {
  h2o.randomForest.wrapper(..., ntrees = ntrees, nbins = nbins, seed = seed)
}
h2o.deeplearning.1 <- function(..., hidden = c(500, 500), activation = "Rectifier", seed = 1) {
  h2o.deeplearning.wrapper(..., hidden = hidden, activation = activation, seed = seed)
}
h2olearner <- c("h2o.glm.1", "h2o.glm.0.5", "h2o.randomForest.1",
                "h2o.deeplearning.1", "h2o.gbm.wrapper")
tmleCom_Options(Qestimator = "h2o__ensemble", gestimator = "h2o__ensemble",
            SL.library = c("SL.glm", "SL.glmnet", "SL.ridge", "SL.stepAIC"), CVfolds = 5,
                h2ometalearner = "h2o.deeplearning.wrapper", h2olearner = h2olearner)

# using "h2o.deeplearning.wrapper" for h2ometalearner
tmleCom_Options(Qestimator = "h2o__ensemble", gestimator = "h2o__ensemble",
            SL.library = c("SL.glm", "SL.glmnet", "SL.ridge", "SL.stepAIC"), CVfolds = 5,
                h2ometalearner = "h2o.deeplearning.wrapper", h2olearner = h2olearner)

#*********************************************************************************
# Example 2: Define the values of bin cutoffs for continuous outcome in different ways
# through three arguments - bin.method, nbins, maxNperBin
#*********************************************************************************
# 2.1 using equal-length method
# discretize a continuous outcome variable into 10 bins, no more than 1000 obs in each bin
tmleCom_Options(bin.method = "equal.len", nbins = 10, maxNperBin = 1000)

# 2.2 find a compromise between equal-mass and equal-length method
# discretize into 5 bins (default), and no more than 5000 obs in each bin
tmleCom_Options(bin.method = "dhist", nbins = 10, maxNperBin = 5000)

# 2.3 Default to use equal-mass method with 5 bins, no more than 500 obs in each bin
tmleCom_Options()

## End(Not run)
```

Index

∗Topic **R6**
BinaryOutModel, 4
CategorModel, 7
ContinModel, 10
DatKeepClass, 11
GenericModel, 16
MonteCarloSimClass, 20
RegressionClass, 23
∗Topic **class**
BinaryOutModel, 4
CategorModel, 7
ContinModel, 10
DatKeepClass, 11
GenericModel, 16
MonteCarloSimClass, 20
RegressionClass, 23
∗Topic **datasets**
comSample.wmT.bA.bY_list, 8
indSample.iid.bA.bY.rareJ1_list, 18
indSample.iid.bA.bY.rareJ2_list, 19
indSample.iid.cA.cY_list, 19

BinaryOutModel, 4, *8*, *11, 12*, *16, 17*, *33*

CategorModel, 7, *15*, *17*, *23*, *25*
comSample.wmT.bA.bY_list, *4*, 8
ContinModel, 10, *15*, *17*, *23*, *25*

DatKeepClass, *6*, *8*, *10, 11*, 11, *15–17*, *21*, *25*, *33*

fitGenericDensity, 14

GenericModel, *7, 8*, *10, 11*, *15*, 16, *23*, *25*, *27*, *30*, *33*
glm.fit, *40*

h2o.ensemble, *41*

indSample.iid.bA.bY.rareJ1_list, *4*, 18
indSample.iid.bA.bY.rareJ2_list, *4*, 19
indSample.iid.cA.cY_list, *4*, 19, *33*

MonteCarloSimClass, 20, *33*

panelData_Trans, 21
print_tmleCom_opts, 23, *42*

R6Class, *5*, *8*, *10, 11*, *16*, *20*, *24*
RegressionClass, *5, 6*, *8*, *11*, *14–17*, *21*, 23, *33*

speedglm.wfit, *40*
SuperLearner, *41*

tmleCom_Options, *6*, *13*, *15*, *21*, *23*, *33*, 40
tmleCommunity, *4*, *10*, *13*, *15, 16*, *21*, 25, *30*
tmleCommunity-package, 3