Package 'stremr'

October 3, 2016

Title Streamlined Estimation of Survival for Static, Dynamic and Stochastic Treatment and Monitoring Regimes

Version 0.2

Description Analysis of longitudinal time-to-event or time-to-failure data. Estimates the counterfactual discrete survival curve under static, dynamic and stochastic interventions on treatment (exposure) and monitoring events over time. Estimators (IPW, MSM-IPW, GCOMP, longitudinal TMLE) adjust for measured time-varying confounding and informative right-censoring. Model fitting can be performed either with GLM or H2O-3 machine learning libraries, including the ensemble-based SuperLearner ('h2oEnsemble'). The exposure, monitoring and censoring variables can be coded as either binary, categorical or continuous. Each can be multivariate (e.g., can use more than one column of dummy indicators for different censoring events). The input data needs to be in long format.

URL https://github.com/osofr/stremr

BugReports https://github.com/osofr/stremr/issues

SystemRequirements pandoc (http://pandoc.org) for generating and exporting markdown reports to other formats.

Depends R (>= 3.2.1)

Imports assertthat, data.table, methods, R6, Rcpp, rmarkdown, pander, speedglm, stats, stringr, zoo

LinkingTo Rcpp

Suggests devtools, h2o, h2oEnsemble, knitr, magrittr, RUnit, foreach, doParallel

Additional_repositories https://osofr.github.io/drat/

License GPL (>= 2)

LazyData true

RoxygenNote 5.0.1

NeedsCompilation yes

Author Oleg Sofrygin [aut, cre], Mark J. van der Laan [aut], Romain Neugebauer [aut]

Maintainer Oleg Sofrygin <oleg.sofrygin@gmail.com>

Repository CRAN

Date/Publication 2016-10-03 08:17:49

R topics documented:

`stremr-package` *Estimate the Survival of Intervention on Exposures and MONITORing Process for Right Censored Longitudinal Data.*

Description

The **stremr** R package is a tool for estimation of causal survival curve under various user-specified interventions (e.g., static, dynamic, deterministic, or stochastic). In particular, the interventions may represent exposures to treatment regimens, the occurrence or non-occurrence of right-censoring events, or of clinical monitoring events. **stremr** enables estimation of a selected set of the user-specified causal quantities of interest, such as, treatment-specific survival curves and the average risk difference over time.

Documentation

- To see the package vignette use: `vignette("stremr_vignette", package="stremr")`
- To see all available package documentation use: `help(package = 'stremr')`

Routines

The following routines will be generally invoked by a user, in the same order as presented below.

`stremr` One function for performing estimation

Data structures

The following most common types of output are produced by the package:

- *observed data* - input data.frame in long format (repeated measures over time).

Updates

Check for updates and report bugs at `http://github.com/osofr/stremr`.

`BinaryOutcomeModel` *R6 class for fitting and making predictions for a single binary outcome regression model P(B | PredVars)*

Description

This R6 class can request, store and manage the design matrix Xmat, as well as the binary outcome Bin for the logistic regression P(Bin|Xmat). Can also be used for converting data in wide format to long when requested, e.g., when pooling across binary indicators (fitting one pooled logistic regression model for several indicators) The class has methods that perform queries to data storage R6 class DataStorageClass to get appropriate data columns & row subsets

Usage

```
BinaryOutcomeModel
```

Format

An `R6Class` generator object

Details

- cont.sVar.flag - Is the original outcome variable continuous?
- bw.j - Bin width (interval length) for an outcome that is a bin indicator of a discretized continous outcome.
- GLMpackage - Controls which package will be used for performing model fits (`glm` or `speedglm`).
- binomialModelObj - Pointer to an instance of `binomialModelObj` class that contains the data.

Methods

`new(reg)` Uses `reg` R6 `RegressionClass` object to instantiate a new model for a logistic regression with binary outcome.

`show()` Print information on outcome and predictor names used in this regression model

`fit()` ...

`copy.fit()` ...

`predict()` ...

`copy.predict()` ...

`predictAeqa()` ...

Active Bindings

`getoutvarnm` ...

`getoutvarval` ...

`getsubset` ...

`getprobA1` ...

`getfit` ...

`wipe.alldat` ...

`BinomialGLM` *R6 class for storing the design matrix and the binary outcome for a single GLM (logistic) regression*

Description

This R6 class can request, store and manage the design matrix Xmat, as well as the binary outcome Bin for the logistic regression P(Bin|Xmat). Can also be used for converting data in wide format to long when requested, e.g., when pooling across binary indicators (fitting one pooled logistic regression model for several indicators) The class has methods that perform queries to data storage R6 class DataStorageClass to get appropriate data columns & row subsets

Usage

```
BinomialGLM
```

Format

An `R6Class` generator object

Details

- ID - Vector of observation IDs, `1:n`, used for pooling.
- outvar - Outcome name.
- predvars - Predictor names.
- subset_vars - Defines the subset which would be used for fitting this model (logical, expression or indices).
- subset_idx - Subset `subset_vars` converted to logical vector.

Methods

`new(reg)` Uses `reg` R6 `RegressionClass` object to instantiate a new storage container for a design matrix and binary outcome.

`setdata()` ...

Active Bindings

emptydata ...

emptyY ...

emptySubset_idx ...

getXmat ...

getY ...

CategorModel *R6 class for fitting and predicting joint probability for a univariate categorical summary A[j]*

Description

This R6 class defines and fits a conditional probability model P(A[j]|W,...) for a univariate categorical summary measure A[j]. This class inherits from GenericModel class. Defines the fitting algorithm for a regression model A[j] ~ W + ... Reconstructs the likelihood P(A[j]=a[j]|W,...) afterwards. Categorical A[j] is first redefined into length(levels) bin indicator variables, where levels is a numeric vector of all unique categories in A[j]. The fitting algorithm estimates the binary regressions for hazard for each bin indicator, Bin_A[j][i] ~ W, i.e., the probability that categorical A[j] falls into bin i, Bin_A[j]_i, given that A[j] does not fall in any prior bins Bin_A[j]_1, ..., Bin_A[j]_{i-1}. The dataset of bin indicators (BinA[j]_1,...,BinA[j]_M) is created inside the passed data or newdata object when defining length(levels) bins for A[j].

Usage

```
CategorModel
```

Format

An R6Class generator object

Details

- reg - .
- outvar - .
- levels - .
- nbins - .

Methods

new(reg, DataStorageClass.g0, ...) ...

fit(data) ...

predict(newdata) ...

predictAeqa(newdata) ...

Active Bindings

cats ...

ContinModel — *R6 class for fitting and predicting joint probability for a univariate continuous summary A[j]*

Description

This R6 class defines and fits a conditional probability model P(A[j]|W,...) for a univariate continuous summary measure A[j]. This class inherits from GenericModel class. Defines the fitting algorithm for a regression model A[j] ~ W + ... Reconstructs the likelihood P(A[j]=a[j]|W,...) afterwards. Continuous A[j] is discretized using either of the 3 interval cutoff methods, defined via RegressionClass object reg passed to this class constructor. The fitting algorithm estimates the binary regressions for hazard Bin_A[j][i] ~ W, i.e., the probability that continuous A[j] falls into bin i, Bin_A[j]_i, given that A[j] does not belong to any prior bins Bin_A[j]_1, ..., Bin_A[j]_{i-1}. The dataset of discretized summary measures (BinA[j]_1,...,BinA[j]_M) is created inside the passed data or newdata object while discretizing A[j] into M bins.

Usage

ContinModel

Format

An R6Class generator object

Details

- reg - .
- outvar - .
- intrvls - .
- intrvls.width - .
- bin_weights - .

Methods

new(reg, DataStorageClass.g0, DataStorageClass.gstar, ...) ...

fit(data) ...

predict(newdata) ...

predictAeqa(newdata) ...

Active Bindings

cats ...

DataStorageClass *R6 class for storing, managing, subsetting and manipulating the input data.*

Description

The class DataStorageClass is the only way the package uses to access the input data. The evaluated summary measures from sVar.object are stored as a matrix (private$.mat.sVar). Contains methods for replacing missing values with default in gvars$misXreplace. Also contains method for detecting / setting sVar variable type (binary, categor, contin). Contains methods for combining, subsetting, discretizing & binirizing summary measures (sW,sA). For continous sVar this class provides methods for detecting / setting bin intervals, normalization, disretization and construction of bin indicators. The pointers to this class get passed on to GenericModel functions: $fit(), $predict() and $predictAeqa().

Usage

```
DataStorageClass
```

Format

An R6Class generator object

Details

- YnodeVals
- det.Y

Methods

```
new(Odata, nodes, YnodeVals, det.Y, ...) ...
def.types.sVar(type.sVar = NULL) ...
fixmiss_sVar() ...
set.sVar(name.sVar, new.type) ...
set.sVar.type(name.sVar, new.type) ...
get.sVar(name.sVar, new.sVarVal) ...
replaceOneAnode(AnodeName, newAnodeVal) ...
replaceManyAnodes(Anodes, newAnodesMat) ...
addYnode(YnodeVals, det.Y) ...
evalsubst(subset_exprs, subset_vars) ...
get.dat.sVar(rowsubset = TRUE, covars) ...
get.outvar(rowsubset = TRUE, var) ...
bin.nms.sVar(name.sVar, nbins) ...
```

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

`detect.cat.sVar.levels(name.sVar)` ...

`binirize.sVar(name.sVar, ...)` ...

`get.sVar.bw(name.sVar, intervals)` ...

`get.sVar.bwdiff(name.sVar, intervals)` ...

Active Bindings

`nobs` ...

`ncols.sVar` ...

`names.sVar` ...

`type.sVar` Named list of length `ncol(private$.mat.sVar)` with `sVar` variable types: "binary"/"categor"/"contin".

`dat.sVar` ...

`ord.sVar` Ordinal (categorical) transformation of a continous covariate `sVar`.

`active.bin.sVar` Name of active binarized cont sVar, changes as fit/predict is called (bin indicators are temp. stored in private$.mat.bin.sVar)

`dat.bin.sVar` ...

`emptydat.sVar` ...

`emptydat.bin.sVar` ...

`noNA.Ynodevals` ...

`nodes` ...

`defineIntervedTRT` *Define counterfactual dynamic treatment*

Description

Defines a new column that contains the counterfactual dynamic treatment values. Subject is switched to treatment when the biomarker `I` crosses the threshold `theta` while being monitored `MONITOR(t-1)=1`.

Usage

```
defineIntervedTRT(data, theta, ID, t, I, CENS, TRT, MONITOR, tsinceNis1,
  new.TRT.names = NULL, return.allcolumns = FALSE)
```

Arguments

data	Input data.frame or data.table in long format, see below for the description of the assumed format.
theta	The vector of continuous cutoff values that index each dynamic treatment rule
ID	The name of the unique subject identifier
t	The name of the variable indicating time-period
I	Continuous biomarker variable used for determining the treatmet decision rule
CENS	Binary indicator of being censored at t;
TRT	Binary indicator of the treatment (exposure) at t;
MONITOR	The indicator of having a visit and having measured or observed biomarker I(t+1) (the biomarker value at THE NEXT TIME CYCLE). In other words the value of MONITOR(t-1) (at t-1) being 1 indicates that I(t) at time point t was observed/measured. The very first value of I(t) (at the first time-cycle) is ALWAYS ASSUMED observed/measured.
tsinceNis1	Character vector for the column in data, same meaning as described in convertdata(), must be already defined
new.TRT.names	Vector with names which will be assigned to new columns generated by this routine(must be the same dimension as theta). When not supplied the following convention is adopted for naming these columns: `paste0(TRT,".gstar.","d",theta)`.
return.allcolumns	Set to TRUE to return the original data columns along with new columns that define each rule (can be useful when employing piping/sequencing operators).

Value

A data.table with a separate column for each value in `theta`. Each column consists of indicators of following/not-following each rule indexed by a value form `theta`. In addition, the returned data.table contains `ID` and `t` columns for easy merging with the original data.

Details

* This function takes an input `data.frame` or `data.table` and produces an output data.table with counterfactual treatment assignment based on a rule defined by values of input column `I` and an input scalar `theta`.

* Evaluates which observations should have received (switched to) treatment based on dynamic-decision rule defined by the measured biomarker `I` and pre-defined cutoffs supplied in the vector `theta`.

* Produces a separate column for each value in `theta`:

- (1) By default no one is treated (TRT assigned to 0)
- (2) Subject may switch to treatment the first time `I` goes over the threshold `theta` (while having been monitored, i.e, MONITOR(t-1)==1)
- (3) Once the subject switches to treatment he or she has to continue receiving treatment until the end of the follow-up

* The format (time-ordering) of data is the same as required by the stremr() function: (I(t), CENS(t), TRT(t), MONITOR(t)). MONITOR(t) at time-point t is defined as the indicator of being observed (having an office visit) at time point t+1 (next timepoint after t) It is assumed that MONITOR(t) is always 0 for the very first time-point.

Examples

```
## Not run:
theta <- seq(7,8.5,by=0.5)
FOLLOW.D.DT <- defineIntervedTRT(data = data, theta = theta,
                 ID = "StudyID", t = "X_intnum", I = "X_a1c",
                 TRT = "X_exposure", CENS = "X_censor", MONITOR = "N.t",
                 new.TRT.names = paste0("gstar",theta))

## End(Not run)
#----------------------------------------------------------------------
# EXAMPLE BASED ON SIMULATED DATA
#----------------------------------------------------------------------
require("data.table")
require("magrittr")
data(OdataCatCENS)
OdataDT <- as.data.table(OdataCatCENS, key=c(ID, t))

#----------------------------------------------------------------------
# Define the counterfactual dynamic treatment assignment
#----------------------------------------------------------------------
# Define two dynamic rules: dlow & dhigh
OdataDT <- defineIntervedTRT(OdataDT, theta = c(0,1), ID = "ID", t = "t", I = "highA1c",
                          CENS = "C", TRT = "TI", MONITOR = "N", tsinceNis1 = "lastNat1",
                           new.TRT.names = c("dlow", "dhigh"), return.allcolumns = TRUE)
```

`defineMONITORvars` *Helper routine to define the monitoring indicator and time since last visit*

Description

Helper routine to define the monitoring indicator and time since last visit

Usage

```
defineMONITORvars(data, ID, t, imp.I, MONITOR.name = "N",
  tsinceNis1 = "last.Nt")
```

Arguments

`data`	Input `data.table` or `data.frame`
`ID`	The name of the unique subject identifier (character, numeric or factor).

t	The name of the time/period variable in data.
imp.I	The name of the binary indicator of missingness or imputation for I at time point t. This is used for coding MONITOR(t-1):=1-imp.I(t). When imp.I(t)=1 it means that the patient was not observed (no office visit) at time-point t and hence no biomarker was measured.
MONITOR.name	The name of the new column that represents for each row t the indicator of being MONITORed (having a visit) at time points t+1. This variable is added as a new column to the output dataset. The column MONITOR(t) is set to 1 when the indicator imp.I (imputed biomarker) is 0 at time-point t+1 and vice versa.
tsinceNis1	The name of the future column (created by this routine) that counts number of periods since last monitoring event at t-1. More precisely, it is a function of the past N(t-1), where 0 means that N(t-1)=1; 1 means that N(t-2)=1 and N(t-1)=0; etc.
I	The name of the numeric biomarker value which determines the dynamic treatment rule at each time point t.

Value

A data.table in long format with ordering (I, CENS, TRT, MONITOR)

Details

Convert the input long format data with the time ordering: (I(t), imp.I(t), C(t), A(t)) into the data format required by the stremr input functions: (I(t), C(t), A(t), N(t):=1-imp.I(t+1)). N(t) at time-point t is defined as the indicator of being observed (having an office visit) at time point t+1 (next timepoint after t) The very first value of I(t) (at the first time-cycle) is ALWAYS ASSUMED observed/measured (hence I.imp=0 for each first subject-time observation).

Data Format

The input data.frame data needs to be in long format.

The format of the specified columns needs to be as follows.

The time ordering of the input data at each t is as follows: (I, imp.I, CENS, TRT)

The time ordering of the output data at each t is as follows: (I, CENS, TRT, MONITOR), where MONITOR(t)=1-imp.I(t+1).

In output data.table, MONITOR(t-1)=1 indicates that the biomarker I(t) at t is observed and vice versa.

In addition the output data.table will contain a column 'tsinceNis1', where:

- tsinceNis1(t) = 0 means that the person was monitored at time-point t-1.
- tsinceNis1(t) > 0 is the count of the number of cycles since last monitoring event.

define_single_regression *Define regression models*

Description

Alternative approach to manually define parts of the propsensity score model with formula/strata and model controls

Usage

```
define_single_regression(OData, regforms, stratify = NULL, params = list())
```

Arguments

OData	OData Input data object created by importData function.
regforms	Regression formula, only main terms are allowed.
stratify	Expression(s) for creating strata(s) (model will be fit separately on each strata)
params	Additional modeling controls (for downstream modeling algorithms)

fitIterTMLE *Iterative TMLE wrapper for* fitSeqGcomp

Description

Calls fitSeqGcomp with argument iterTMLE = TRUE.

Usage

```
fitIterTMLE(...)
```

Arguments

...	Arguments that will be passed down to the underlying function fitSeqGcomp

Value

data.table with survival by time for sequential GCOMP and iterative TMLE

See Also

fitSeqGcomp

Examples

```
options(stremr.verbose = TRUE)
require("data.table")
set_all_stremr_options(fit.package = "speedglm", fit.algorithm = "glm")

# ----------------------------------------------------------------------
# Simulated Data
# ----------------------------------------------------------------------
data(OdataNoCENS)
OdataDT <- as.data.table(OdataNoCENS, key=c(ID, t))

# define lagged N, first value is always 1 (always monitored at the first time point):
OdataDT[, ("N.tminus1") := shift(get("N"), n = 1L, type = "lag", fill = 1L), by = ID]
OdataDT[, ("TI.tminus1") := shift(get("TI"), n = 1L, type = "lag", fill = 1L), by = ID]

# ----------------------------------------------------------------------
# Define intervention (always treated):
# ----------------------------------------------------------------------
OdataDT[, ("TI.set1") := 1L]
OdataDT[, ("TI.set0") := 0L]

# ----------------------------------------------------------------------
# Import Data
# ----------------------------------------------------------------------
OData <- importData(OdataDT, ID = "ID", t = "t", covars = c("highA1c", "lastNat1", "N.tminus1"),
                    CENS = "C", TRT = "TI", MONITOR = "N", OUTCOME = "Y.tplus1")

# ----------------------------------------------------------------------
# Model the Propensity Scores
# ----------------------------------------------------------------------
gform_CENS <- "C ~ highA1c + lastNat1"
gform_TRT = "TI ~ CVD + highA1c + N.tminus1"
gform_MONITOR <- "N ~ 1"
stratify_CENS <- list(C=c("t < 16", "t == 16"))

# ----------------------------------------------------------------------
# Fit Propensity Scores
# ----------------------------------------------------------------------
OData <- fitPropensity(OData, gform_CENS = gform_CENS,
                       gform_TRT = gform_TRT,
                       gform_MONITOR = gform_MONITOR,
                       stratify_CENS = stratify_CENS)

# ----------------------------------------------------------------------
# IPW Ajusted KM or Saturated MSM
# ----------------------------------------------------------------------
require("magrittr")
AKME.St.1 <- getIPWeights(OData, intervened_TRT = "TI.set1") %>%
             survNPMSM(OData) %$%
             IPW_estimates
AKME.St.1
```

```
# ------------------------------------------------------------------------
# Bounded IPW
# ------------------------------------------------------------------------
IPW.St.1 <- getIPWeights(OData, intervened_TRT = "TI.set1") %>%
             survDirectIPW(OData)
IPW.St.1[]

# ------------------------------------------------------------------------
# IPW-MSM for hazard
# ------------------------------------------------------------------------
wts.DT.1 <- getIPWeights(OData = OData, intervened_TRT = "TI.set1", rule_name = "TI1")
wts.DT.0 <- getIPWeights(OData = OData, intervened_TRT = "TI.set0", rule_name = "TI0")
survMSM_res <- survMSM(list(wts.DT.1, wts.DT.0), OData, t_breaks = c(1:8,12,16)-1,)
survMSM_res$St

# ------------------------------------------------------------------------
# Sequential G-COMP
# ------------------------------------------------------------------------
t.surv <- c(0:15)
Qforms <- rep.int("Q.kplus1 ~ CVD + highA1c + N + lastNat1 + TI + TI.tminus1", (max(t.surv)+1))
params = list(fit.package = "speedglm", fit.algorithm = "glm")

## Not run:
gcomp_est <- fitSeqGcomp(OData, t_periods = t.surv, intervened_TRT = "TI.set1",
                         Qforms = Qforms, params_Q = params, stratifyQ_by_rule = FALSE)
gcomp_est[]

## End(Not run)
# ------------------------------------------------------------------------
# TMLE
# ------------------------------------------------------------------------
## Not run:
tmle_est <- fitTMLE(OData, t_periods = t.surv, intervened_TRT = "TI.set1",
                    Qforms = Qforms, params_Q = params, stratifyQ_by_rule = TRUE)
tmle_est[]

## End(Not run)

# ------------------------------------------------------------------------
# Running IPW-Adjusted KM with optional user-specified weights:
# ------------------------------------------------------------------------
addedWts_DT <- OdataDT[, c("ID", "t"), with = FALSE]
addedWts_DT[, new.wts := sample.int(10, nrow(OdataDT), replace = TRUE)/10]
survNP_res_addedWts <- survNPMSM(wts.DT.1, OData, weights = addedWts_DT)

# ------------------------------------------------------------------------
# Multivariate Propensity Score Regressions
# ------------------------------------------------------------------------
gform_CENS <- "C + TI + N ~ highA1c + lastNat1"
OData <- fitPropensity(OData, gform_CENS = gform_CENS, gform_TRT = gform_TRT,
                       gform_MONITOR = gform_MONITOR)

# ------------------------------------------------------------------------
```

```
# Fitting Propensity scores with Random Forests:
# ------------------------------------------------------------------------
## Not run:
set_all_stremr_options(fit.package = "h2o", fit.algorithm = "randomForest")
require("h2o")
h2o::h2o.init(nthreads = -1)
gform_CENS <- "C ~ highA1c + lastNat1"
OData <- fitPropensity(OData, gform_CENS = gform_CENS,
                       gform_TRT = gform_TRT,
                       gform_MONITOR = gform_MONITOR,
                       stratify_CENS = stratify_CENS)

# For Gradient Boosting machines:
set_all_stremr_options(fit.package = "h2o", fit.algorithm = "gbm")
# Use `H2O-3` distributed implementation of GLM
set_all_stremr_options(fit.package = "h2o", fit.algorithm = "glm")
# Use Deep Neural Nets:
set_all_stremr_options(fit.package = "h2o", fit.algorithm = "deeplearning")

## End(Not run)

# ------------------------------------------------------------------------
# Fitting different models with different algorithms
# Fine tuning modeling with optional tuning parameters.
# ------------------------------------------------------------------------
## Not run:
params_TRT = list(fit.package = "h2o", fit.algorithm = "gbm", ntrees = 50,
    learn_rate = 0.05, sample_rate = 0.8, col_sample_rate = 0.8,
    balance_classes = TRUE)
params_CENS = list(fit.package = "speedglm", fit.algorithm = "glm")
params_MONITOR = list(fit.package = "speedglm", fit.algorithm = "glm")
OData <- fitPropensity(OData,
          gform_CENS = gform_CENS, stratify_CENS = stratify_CENS, params_CENS = params_CENS,
            gform_TRT = gform_TRT, params_TRT = params_TRT,
            gform_MONITOR = gform_MONITOR, params_MONITOR = params_MONITOR)

## End(Not run)

# ------------------------------------------------------------------------
# Running TMLE based on the previous fit of the propensity scores.
# Also applying Random Forest to estimate the sequential outcome model
# ------------------------------------------------------------------------
## Not run:
t.surv <- c(0:5)
Qforms <- rep.int("Q.kplus1 ~ CVD + highA1c + N + lastNat1 + TI + TI.tminus1", (max(t.surv)+1))
params_Q = list(fit.package = "h2o", fit.algorithm = "randomForest",
                ntrees = 100, learn_rate = 0.05, sample_rate = 0.8,
                col_sample_rate = 0.8, balance_classes = TRUE)
tmle_est <- fitTMLE(OData, t_periods = t.surv, intervened_TRT = "TI.set1",
            Qforms = Qforms, params_Q = params_Q,
            stratifyQ_by_rule = TRUE)

## End(Not run)
```

```
## Not run:
t.surv <- c(0:5)
Qforms <- rep.int("Q.kplus1 ~ CVD + highA1c + N + lastNat1 + TI + TI.tminus1", (max(t.surv)+1))
params_Q = list(fit.package = "h2o", fit.algorithm = "randomForest",
                ntrees = 100, learn_rate = 0.05, sample_rate = 0.8,
                col_sample_rate = 0.8, balance_classes = TRUE)
tmle_est <- fitTMLE(OData, t_periods = t.surv, intervened_TRT = "TI.set1",
            Qforms = Qforms, params_Q = params_Q,
            stratifyQ_by_rule = FALSE)

## End(Not run)

# ----------------------------------------------------------------------
# Ensemble Learning with SuperLearner (using h2oEnsemble R package):
# ----------------------------------------------------------------------
## Not run:
require('h2oEnsemble')
# ----------------------------------------------------------------------
# Define many learners at once via grid search over tuning parameter spaces (hyperparameters)
# ----------------------------------------------------------------------
# 1. Runs h2o.grid in the background for each individual learner and saves cross-validated risks.
# 2. Calls h2o.stack from h2oEnsemble package to evaluate the final SuperLearner.
# ----------------------------------------------------------------------
# Search criteria and grid search space for glm tuning parameters:
GLM_hyper_params <- list(search_criteria = list(strategy = "RandomDiscrete", max_models = 5),
                         alpha = c(0,1,seq(0.1,0.9,0.1)),
                         lambda = c(0,1e-7,1e-5,1e-3,1e-1))

# Search criteria and grid search space for Random Forests:
search_criteria <- list(strategy = "RandomDiscrete", max_models = 5, max_runtime_secs = 60*60)
RF_hyper_params <- list(search_criteria = search_criteria,
                        ntrees = c(100, 200, 300, 500),
                        mtries = 1:4,
                        max_depth = c(5, 10, 15, 20, 25),
                        sample_rate = c(0.7, 0.8, 0.9, 1.0),
                        col_sample_rate_per_tree = c(0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8),
                        balance_classes = c(TRUE, FALSE))

# Search criteria and grid search space for Gradient Boosting Machines (gbm):
GBM_hyper_params <- list(search_criteria = search_criteria,
                         ntrees = c(100, 200, 300, 500),
                         learn_rate = c(0.005, 0.01, 0.03, 0.06),
                         max_depth = c(3, 4, 5, 6, 9),
                         sample_rate = c(0.7, 0.8, 0.9, 1.0),
                         col_sample_rate = c(0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8),
                         balance_classes = c(TRUE, FALSE))

# Specify individual learners to include in the SuperLearner library (in addition to grid models):
h2o.glm.1 <- function(..., alpha = 0.0) h2o.glm.wrapper(..., alpha = alpha)
h2o.glm.2 <- function(..., x = "highA1c", alpha = 0.0) h2o.glm.wrapper(..., x = x, alpha = alpha)
h2o.glm.3 <- function(..., alpha = 1.0) h2o.glm.wrapper(..., alpha = alpha)
```

```
# ----------------------------------------------------------------------
# Put all different algorithms together to define the final model ensemble:
# ----------------------------------------------------------------------
SLparams = list(fit.package = "h2o", fit.algorithm = "SuperLearner",
                grid.algorithm = c("glm", "randomForest", "gbm"),
                learner = c("h2o.glm.1", "h2o.glm.2", "h2o.glm.3"),
                metalearner = "h2o.glm_nn",
                nfolds = 10,
                seed = 23,
                glm = GLM_hyper_params,
                randomForest = RF_hyper_params,
                gbm = GBM_hyper_params)

# ----------------------------------------------------------------------
# Use save.ensemble and ensemble.dir.path arguments to save the final SuperLearner fit
# ----------------------------------------------------------------------
# This will also save the individual learner fits in the same directory.
# Only one directory per single SuperLearner fit is allowed.
# Can be loaded later on to save time and avoid refitting SuperLearner (load.ensemble = TRUE).

params_TRT = c(SLparams, save.ensemble = TRUE, ensemble.dir.path = "./h2o-ensemble-model-TRT")
params_CENS = list(fit.package = "speedglm", fit.algorithm = "glm")
params_MONITOR = list(fit.package = "speedglm", fit.algorithm = "glm")

OData <- fitPropensity(OData,
          gform_CENS = gform_CENS, stratify_CENS = stratify_CENS, params_CENS = params_CENS,
            gform_TRT = gform_TRT, params_TRT = params_TRT,
            gform_MONITOR = gform_MONITOR, params_MONITOR = params_MONITOR)

# ----------------------------------------------------------------------
# Re-using previously saved SuperLearner fit
# ----------------------------------------------------------------------
params_TRT = c(SLparams, load.ensemble = TRUE, ensemble.dir.path = "./h2o-ensemble-model-TRT")
params_CENS = list(fit.package = "speedglm", fit.algorithm = "glm")
params_MONITOR = list(fit.package = "speedglm", fit.algorithm = "glm")

OData <- fitPropensity(OData,
          gform_CENS = gform_CENS, stratify_CENS = stratify_CENS, params_CENS = params_CENS,
            gform_TRT = gform_TRT, params_TRT = params_TRT,
            gform_MONITOR = gform_MONITOR, params_MONITOR = params_MONITOR)

## End(Not run)
```

`fitPropensity` *Define and fit propensity score models.*

Description

Defines and fits regression models for the propensity scores for censoring, treatment and monitoring.

Usage

```
fitPropensity(OData, gform_CENS, gform_TRT, gform_MONITOR,
  stratify_CENS = NULL, stratify_TRT = NULL, stratify_MONITOR = NULL,
  params_CENS = list(), params_TRT = list(), params_MONITOR = list(),
  reg_CENS, reg_TRT, reg_MONITOR, verbose = getOption("stremr.verbose"))
```

Arguments

OData	Input data object created by `importData` function.
gform_CENS	...
gform_TRT	...
gform_MONITOR	...
stratify_CENS	...
stratify_TRT	...
stratify_MONITOR	...
params_CENS	...
params_TRT	...
params_MONITOR	...
reg_CENS	...
reg_TRT	...
reg_MONITOR	...
verbose	Set to `TRUE` to print messages on status and information to the console. Turn this on by default using `options(stremr.verbose=TRUE)`.

Value

...

Examples

```
options(stremr.verbose = TRUE)
require("data.table")
set_all_stremr_options(fit.package = "speedglm", fit.algorithm = "glm")

# ----------------------------------------------------------------------
# Simulated Data
# ----------------------------------------------------------------------
data(OdataNoCENS)
OdataDT <- as.data.table(OdataNoCENS, key=c(ID, t))

# define lagged N, first value is always 1 (always monitored at the first time point):
OdataDT[, ("N.tminus1") := shift(get("N"), n = 1L, type = "lag", fill = 1L), by = ID]
OdataDT[, ("TI.tminus1") := shift(get("TI"), n = 1L, type = "lag", fill = 1L), by = ID]

# ----------------------------------------------------------------------
```

```
# Define intervention (always treated):
# ------------------------------------------------------------------------
OdataDT[, ("TI.set1") := 1L]
OdataDT[, ("TI.set0") := 0L]

# ------------------------------------------------------------------------
# Import Data
# ------------------------------------------------------------------------
OData <- importData(OdataDT, ID = "ID", t = "t", covars = c("highA1c", "lastNat1", "N.tminus1"),
                    CENS = "C", TRT = "TI", MONITOR = "N", OUTCOME = "Y.tplus1")

# ------------------------------------------------------------------------
# Model the Propensity Scores
# ------------------------------------------------------------------------
gform_CENS <- "C ~ highA1c + lastNat1"
gform_TRT = "TI ~ CVD + highA1c + N.tminus1"
gform_MONITOR <- "N ~ 1"
stratify_CENS <- list(C=c("t < 16", "t == 16"))

# ------------------------------------------------------------------------
# Fit Propensity Scores
# ------------------------------------------------------------------------
OData <- fitPropensity(OData, gform_CENS = gform_CENS,
                       gform_TRT = gform_TRT,
                       gform_MONITOR = gform_MONITOR,
                       stratify_CENS = stratify_CENS)

# ------------------------------------------------------------------------
# IPW Ajusted KM or Saturated MSM
# ------------------------------------------------------------------------
require("magrittr")
AKME.St.1 <- getIPWeights(OData, intervened_TRT = "TI.set1") %>%
             survNPMSM(OData) %$%
             IPW_estimates
AKME.St.1

# ------------------------------------------------------------------------
# Bounded IPW
# ------------------------------------------------------------------------
IPW.St.1 <- getIPWeights(OData, intervened_TRT = "TI.set1") %>%
             survDirectIPW(OData)
IPW.St.1[]

# ------------------------------------------------------------------------
# IPW-MSM for hazard
# ------------------------------------------------------------------------
wts.DT.1 <- getIPWeights(OData = OData, intervened_TRT = "TI.set1", rule_name = "TI1")
wts.DT.0 <- getIPWeights(OData = OData, intervened_TRT = "TI.set0", rule_name = "TI0")
survMSM_res <- survMSM(list(wts.DT.1, wts.DT.0), OData, t_breaks = c(1:8,12,16)-1,)
survMSM_res$St

# ------------------------------------------------------------------------
# Sequential G-COMP
```

```
# ------------------------------------------------------------------------
t.surv <- c(0:15)
Qforms <- rep.int("Q.kplus1 ~ CVD + highA1c + N + lastNat1 + TI + TI.tminus1", (max(t.surv)+1))
params = list(fit.package = "speedglm", fit.algorithm = "glm")

## Not run:
gcomp_est <- fitSeqGcomp(OData, t_periods = t.surv, intervened_TRT = "TI.set1",
                         Qforms = Qforms, params_Q = params, stratifyQ_by_rule = FALSE)
gcomp_est[]

## End(Not run)
# ------------------------------------------------------------------------
# TMLE
# ------------------------------------------------------------------------
## Not run:
tmle_est <- fitTMLE(OData, t_periods = t.surv, intervened_TRT = "TI.set1",
                    Qforms = Qforms, params_Q = params, stratifyQ_by_rule = TRUE)
tmle_est[]

## End(Not run)

# ------------------------------------------------------------------------
# Running IPW-Adjusted KM with optional user-specified weights:
# ------------------------------------------------------------------------
addedWts_DT <- OdataDT[, c("ID", "t"), with = FALSE]
addedWts_DT[, new.wts := sample.int(10, nrow(OdataDT), replace = TRUE)/10]
survNP_res_addedWts <- survNPMSM(wts.DT.1, OData, weights = addedWts_DT)

# ------------------------------------------------------------------------
# Multivariate Propensity Score Regressions
# ------------------------------------------------------------------------
gform_CENS <- "C + TI + N ~ highA1c + lastNat1"
OData <- fitPropensity(OData, gform_CENS = gform_CENS, gform_TRT = gform_TRT,
                       gform_MONITOR = gform_MONITOR)

# ------------------------------------------------------------------------
# Fitting Propensity scores with Random Forests:
# ------------------------------------------------------------------------
## Not run:
set_all_stremr_options(fit.package = "h2o", fit.algorithm = "randomForest")
require("h2o")
h2o::h2o.init(nthreads = -1)
gform_CENS <- "C ~ highA1c + lastNat1"
OData <- fitPropensity(OData, gform_CENS = gform_CENS,
                       gform_TRT = gform_TRT,
                       gform_MONITOR = gform_MONITOR,
                       stratify_CENS = stratify_CENS)

# For Gradient Boosting machines:
set_all_stremr_options(fit.package = "h2o", fit.algorithm = "gbm")
# Use `H2O-3` distributed implementation of GLM
set_all_stremr_options(fit.package = "h2o", fit.algorithm = "glm")
# Use Deep Neural Nets:
```

```
set_all_stremr_options(fit.package = "h2o", fit.algorithm = "deeplearning")

## End(Not run)

# ----------------------------------------------------------------------
# Fitting different models with different algorithms
# Fine tuning modeling with optional tuning parameters.
# ----------------------------------------------------------------------
## Not run:
params_TRT = list(fit.package = "h2o", fit.algorithm = "gbm", ntrees = 50,
    learn_rate = 0.05, sample_rate = 0.8, col_sample_rate = 0.8,
    balance_classes = TRUE)
params_CENS = list(fit.package = "speedglm", fit.algorithm = "glm")
params_MONITOR = list(fit.package = "speedglm", fit.algorithm = "glm")
OData <- fitPropensity(OData,
          gform_CENS = gform_CENS, stratify_CENS = stratify_CENS, params_CENS = params_CENS,
            gform_TRT = gform_TRT, params_TRT = params_TRT,
            gform_MONITOR = gform_MONITOR, params_MONITOR = params_MONITOR)

## End(Not run)

# ----------------------------------------------------------------------
# Running TMLE based on the previous fit of the propensity scores.
# Also applying Random Forest to estimate the sequential outcome model
# ----------------------------------------------------------------------
## Not run:
t.surv <- c(0:5)
Qforms <- rep.int("Q.kplus1 ~ CVD + highA1c + N + lastNat1 + TI + TI.tminus1", (max(t.surv)+1))
params_Q = list(fit.package = "h2o", fit.algorithm = "randomForest",
                ntrees = 100, learn_rate = 0.05, sample_rate = 0.8,
                col_sample_rate = 0.8, balance_classes = TRUE)
tmle_est <- fitTMLE(OData, t_periods = t.surv, intervened_TRT = "TI.set1",
            Qforms = Qforms, params_Q = params_Q,
            stratifyQ_by_rule = TRUE)

## End(Not run)

## Not run:
t.surv <- c(0:5)
Qforms <- rep.int("Q.kplus1 ~ CVD + highA1c + N + lastNat1 + TI + TI.tminus1", (max(t.surv)+1))
params_Q = list(fit.package = "h2o", fit.algorithm = "randomForest",
                ntrees = 100, learn_rate = 0.05, sample_rate = 0.8,
                col_sample_rate = 0.8, balance_classes = TRUE)
tmle_est <- fitTMLE(OData, t_periods = t.surv, intervened_TRT = "TI.set1",
            Qforms = Qforms, params_Q = params_Q,
            stratifyQ_by_rule = FALSE)

## End(Not run)

# ----------------------------------------------------------------------
# Ensemble Learning with SuperLearner (using h2oEnsemble R package):
# ----------------------------------------------------------------------
## Not run:
```

```
require('h2oEnsemble')
# ----------------------------------------------------------------------
# Define many learners at once via grid search over tuning parameter spaces (hyperparameters)
# ----------------------------------------------------------------------
# 1. Runs h2o.grid in the background for each individual learner and saves cross-validated risks.
# 2. Calls h2o.stack from h2oEnsemble package to evaluate the final SuperLearner.
# ----------------------------------------------------------------------
# Search criteria and grid search space for glm tuning parameters:
GLM_hyper_params <- list(search_criteria = list(strategy = "RandomDiscrete", max_models = 5),
                         alpha = c(0,1,seq(0.1,0.9,0.1)),
                         lambda = c(0,1e-7,1e-5,1e-3,1e-1))

# Search criteria and grid search space for Random Forests:
search_criteria <- list(strategy = "RandomDiscrete", max_models = 5, max_runtime_secs = 60*60)
RF_hyper_params <- list(search_criteria = search_criteria,
                        ntrees = c(100, 200, 300, 500),
                        mtries = 1:4,
                        max_depth = c(5, 10, 15, 20, 25),
                        sample_rate = c(0.7, 0.8, 0.9, 1.0),
                        col_sample_rate_per_tree = c(0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8),
                        balance_classes = c(TRUE, FALSE))

# Search criteria and grid search space for Gradient Boosting Machines (gbm):
GBM_hyper_params <- list(search_criteria = search_criteria,
                         ntrees = c(100, 200, 300, 500),
                         learn_rate = c(0.005, 0.01, 0.03, 0.06),
                         max_depth = c(3, 4, 5, 6, 9),
                         sample_rate = c(0.7, 0.8, 0.9, 1.0),
                         col_sample_rate = c(0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8),
                         balance_classes = c(TRUE, FALSE))

# Specify individual learners to include in the SuperLearner library (in addition to grid models):
h2o.glm.1 <- function(..., alpha = 0.0) h2o.glm.wrapper(..., alpha = alpha)
h2o.glm.2 <- function(..., x = "highA1c", alpha = 0.0) h2o.glm.wrapper(..., x = x, alpha = alpha)
h2o.glm.3 <- function(..., alpha = 1.0) h2o.glm.wrapper(..., alpha = alpha)

# ----------------------------------------------------------------------
# Put all different algorithms together to define the final model ensemble:
# ----------------------------------------------------------------------
SLparams = list(fit.package = "h2o", fit.algorithm = "SuperLearner",
                grid.algorithm = c("glm", "randomForest", "gbm"),
                learner = c("h2o.glm.1", "h2o.glm.2", "h2o.glm.3"),
                metalearner = "h2o.glm_nn",
                nfolds = 10,
                seed = 23,
                glm = GLM_hyper_params,
                randomForest = RF_hyper_params,
                gbm = GBM_hyper_params)

# ----------------------------------------------------------------------
# Use save.ensemble and ensemble.dir.path arguments to save the final SuperLearner fit
# ----------------------------------------------------------------------
# This will also save the individual learner fits in the same directory.
```

```
# Only one directory per single SuperLearner fit is allowed.
# Can be loaded later on to save time and avoid refitting SuperLearner (load.ensemble = TRUE).

params_TRT = c(SLparams, save.ensemble = TRUE, ensemble.dir.path = "./h2o-ensemble-model-TRT")
params_CENS = list(fit.package = "speedglm", fit.algorithm = "glm")
params_MONITOR = list(fit.package = "speedglm", fit.algorithm = "glm")

OData <- fitPropensity(OData,
          gform_CENS = gform_CENS, stratify_CENS = stratify_CENS, params_CENS = params_CENS,
            gform_TRT = gform_TRT, params_TRT = params_TRT,
            gform_MONITOR = gform_MONITOR, params_MONITOR = params_MONITOR)

# ----------------------------------------------------------------------
# Re-using previously saved SuperLearner fit
# ----------------------------------------------------------------------
params_TRT = c(SLparams, load.ensemble = TRUE, ensemble.dir.path = "./h2o-ensemble-model-TRT")
params_CENS = list(fit.package = "speedglm", fit.algorithm = "glm")
params_MONITOR = list(fit.package = "speedglm", fit.algorithm = "glm")

OData <- fitPropensity(OData,
          gform_CENS = gform_CENS, stratify_CENS = stratify_CENS, params_CENS = params_CENS,
            gform_TRT = gform_TRT, params_TRT = params_TRT,
            gform_MONITOR = gform_MONITOR, params_MONITOR = params_MONITOR)

## End(Not run)
```

`fitSeqGcomp` *Fit sequential GCOMP and TMLE for survival*

Description

Interventions on up to 3 nodes are allowed: `CENS`, `TRT` and `MONITOR`. TMLE adjustment will be based on the inverse of the propensity score fits for the observed likelihood (g0.C, g0.A, g0.N), multiplied by the indicator of not being censored and the probability of each intervention in `intervened_TRT` and `intervened_MONITOR`. Requires column name(s) that specify the counterfactual node values or the counterfactual probabilities of each node being 1 (for stochastic interventions).

Usage

```
fitSeqGcomp(OData, t_periods, Qforms, Qstratify = NULL,
  intervened_TRT = NULL, intervened_MONITOR = NULL, useonly_t_TRT = NULL,
  useonly_t_MONITOR = NULL, rule_name = paste0(c(intervened_TRT,
  intervened_MONITOR), collapse = ""), stratifyQ_by_rule = FALSE,
  TMLE = FALSE, iterTMLE = FALSE, IPWeights = NULL, stabilize = FALSE,
  trunc_weights = 10^6, params_Q = list(), weights = NULL,
  max_iter = 50, tol.eps = 0.001, parallel = FALSE,
  verbose = getOption("stremr.verbose"))
```

Arguments

OData	Input data object created by importData function.
t_periods	Specify the vector of time-points for which the survival function (and risk) should be estimated
Qforms	Regression formulas, one formula per Q. Only main-terms are allowed.
Qstratify	Placeholder for future user-defined model stratification for all Qs (CURRENTLY NOT FUNCTIONAL, WILL RESULT IN ERROR)
intervened_TRT	Column name in the input data with the probabilities (or indicators) of counterfactual treatment nodes being equal to 1 at each time point. Leave the argument unspecified (NULL) when not intervening on treatment node(s).
intervened_MONITOR	Column name in the input data with probabilities (or indicators) of counterfactual monitoring nodes being equal to 1 at each time point. Leave the argument unspecified (NULL) when not intervening on the monitoring node(s).
useonly_t_TRT	Use for intervening only on some subset of observation and time-specific treatment nodes. Should be a character string with a logical expression that defines the subset of intervention observations. For example, using TRT==0 will intervene only at observations with the value of TRT being equal to zero. The expression can contain any variable name that was defined in the input dataset. Leave as NULL when intervening on all observations/time-points.
useonly_t_MONITOR	Same as useonly_t_TRT, but for monitoring nodes.
rule_name	Optional name for the treatment/monitoring regimen.
stratifyQ_by_rule	Set to TRUE for stratification, fits the outcome model (Q-learning) among rule-followers only. Setting to FALSE will fit the outcome model (Q-learning) across all observations (pooled regression).
TMLE	Set to TRUE to run the usual longitudinal TMLE algorithm (with a separate TMLE update of Q for every sequential regression).
iterTMLE	Set to TRUE to run the iterative univariate TMLE instead of the usual longitudinal TMLE. When set to TRUE this will also provide the standard sequential Gcomp as party of the output. Must set TMLE=FALSE when setting this to TRUE.
IPWeights	(Optional) result of calling function getIPWeights for running TMLE (evaluated automatically when missing)
stabilize	Set to TRUE to use stabilized weights for the TMLE
trunc_weights	Specify the numeric weight truncation value. All final weights exceeding the value in trunc_weights will be truncated.
params_Q	Optional parameters to be passed to the specific fitting algorithm for Q-learning
weights	Optional data.table with additional observation-time-specific weights. Must contain columns ID, t and weight. The column named weight is merged back into the original data according to (ID, t).
max_iter	Maximum number of iterations for iterative TMLE algorithm

tol.eps	Numeric error tolerance for the iterative TMLE update. The iterative TMLE algorithm will stop when the absolute value of the TMLE intercept update is below tol.eps
parallel	Set to TRUE to run the sequential Gcomp or TMLE in parallel (uses foreach with dopar and requires a previously defined parallel back-end cluster)
verbose	...

Value

...

See Also

stremr-package for the general overview of the package,

Examples

```
options(stremr.verbose = TRUE)
require("data.table")
set_all_stremr_options(fit.package = "speedglm", fit.algorithm = "glm")

# ----------------------------------------------------------------------
# Simulated Data
# ----------------------------------------------------------------------
data(OdataNoCENS)
OdataDT <- as.data.table(OdataNoCENS, key=c(ID, t))

# define lagged N, first value is always 1 (always monitored at the first time point):
OdataDT[, ("N.tminus1") := shift(get("N"), n = 1L, type = "lag", fill = 1L), by = ID]
OdataDT[, ("TI.tminus1") := shift(get("TI"), n = 1L, type = "lag", fill = 1L), by = ID]

# ----------------------------------------------------------------------
# Define intervention (always treated):
# ----------------------------------------------------------------------
OdataDT[, ("TI.set1") := 1L]
OdataDT[, ("TI.set0") := 0L]

# ----------------------------------------------------------------------
# Import Data
# ----------------------------------------------------------------------
OData <- importData(OdataDT, ID = "ID", t = "t", covars = c("highA1c", "lastNat1", "N.tminus1"),
                    CENS = "C", TRT = "TI", MONITOR = "N", OUTCOME = "Y.tplus1")

# ----------------------------------------------------------------------
# Model the Propensity Scores
# ----------------------------------------------------------------------
gform_CENS <- "C ~ highA1c + lastNat1"
gform_TRT = "TI ~ CVD + highA1c + N.tminus1"
gform_MONITOR <- "N ~ 1"
stratify_CENS <- list(C=c("t < 16", "t == 16"))
```

```
# ------------------------------------------------------------------------
# Fit Propensity Scores
# ------------------------------------------------------------------------
OData <- fitPropensity(OData, gform_CENS = gform_CENS,
                       gform_TRT = gform_TRT,
                       gform_MONITOR = gform_MONITOR,
                       stratify_CENS = stratify_CENS)

# ------------------------------------------------------------------------
# IPW Ajusted KM or Saturated MSM
# ------------------------------------------------------------------------
require("magrittr")
AKME.St.1 <- getIPWeights(OData, intervened_TRT = "TI.set1") %>%
             survNPMSM(OData) %$%
             IPW_estimates
AKME.St.1

# ------------------------------------------------------------------------
# Bounded IPW
# ------------------------------------------------------------------------
IPW.St.1 <- getIPWeights(OData, intervened_TRT = "TI.set1") %>%
             survDirectIPW(OData)
IPW.St.1[]

# ------------------------------------------------------------------------
# IPW-MSM for hazard
# ------------------------------------------------------------------------
wts.DT.1 <- getIPWeights(OData = OData, intervened_TRT = "TI.set1", rule_name = "TI1")
wts.DT.0 <- getIPWeights(OData = OData, intervened_TRT = "TI.set0", rule_name = "TI0")
survMSM_res <- survMSM(list(wts.DT.1, wts.DT.0), OData, t_breaks = c(1:8,12,16)-1,)
survMSM_res$St

# ------------------------------------------------------------------------
# Sequential G-COMP
# ------------------------------------------------------------------------
t.surv <- c(0:15)
Qforms <- rep.int("Q.kplus1 ~ CVD + highA1c + N + lastNat1 + TI + TI.tminus1", (max(t.surv)+1))
params = list(fit.package = "speedglm", fit.algorithm = "glm")

## Not run:
gcomp_est <- fitSeqGcomp(OData, t_periods = t.surv, intervened_TRT = "TI.set1",
                         Qforms = Qforms, params_Q = params, stratifyQ_by_rule = FALSE)
gcomp_est[]

## End(Not run)
# ------------------------------------------------------------------------
# TMLE
# ------------------------------------------------------------------------
## Not run:
tmle_est <- fitTMLE(OData, t_periods = t.surv, intervened_TRT = "TI.set1",
                    Qforms = Qforms, params_Q = params, stratifyQ_by_rule = TRUE)
tmle_est[]
```

```
## End(Not run)

# ------------------------------------------------------------------------
# Running IPW-Adjusted KM with optional user-specified weights:
# ------------------------------------------------------------------------
addedWts_DT <- OdataDT[, c("ID", "t"), with = FALSE]
addedWts_DT[, new.wts := sample.int(10, nrow(OdataDT), replace = TRUE)/10]
survNP_res_addedWts <- survNPMSM(wts.DT.1, OData, weights = addedWts_DT)

# ------------------------------------------------------------------------
# Multivariate Propensity Score Regressions
# ------------------------------------------------------------------------
gform_CENS <- "C + TI + N ~ highA1c + lastNat1"
OData <- fitPropensity(OData, gform_CENS = gform_CENS, gform_TRT = gform_TRT,
                       gform_MONITOR = gform_MONITOR)

# ------------------------------------------------------------------------
# Fitting Propensity scores with Random Forests:
# ------------------------------------------------------------------------
## Not run:
set_all_stremr_options(fit.package = "h2o", fit.algorithm = "randomForest")
require("h2o")
h2o::h2o.init(nthreads = -1)
gform_CENS <- "C ~ highA1c + lastNat1"
OData <- fitPropensity(OData, gform_CENS = gform_CENS,
                       gform_TRT = gform_TRT,
                       gform_MONITOR = gform_MONITOR,
                       stratify_CENS = stratify_CENS)

# For Gradient Boosting machines:
set_all_stremr_options(fit.package = "h2o", fit.algorithm = "gbm")
# Use `H2O-3` distributed implementation of GLM
set_all_stremr_options(fit.package = "h2o", fit.algorithm = "glm")
# Use Deep Neural Nets:
set_all_stremr_options(fit.package = "h2o", fit.algorithm = "deeplearning")

## End(Not run)

# ------------------------------------------------------------------------
# Fitting different models with different algorithms
# Fine tuning modeling with optional tuning parameters.
# ------------------------------------------------------------------------
## Not run:
params_TRT = list(fit.package = "h2o", fit.algorithm = "gbm", ntrees = 50,
    learn_rate = 0.05, sample_rate = 0.8, col_sample_rate = 0.8,
    balance_classes = TRUE)
params_CENS = list(fit.package = "speedglm", fit.algorithm = "glm")
params_MONITOR = list(fit.package = "speedglm", fit.algorithm = "glm")
OData <- fitPropensity(OData,
          gform_CENS = gform_CENS, stratify_CENS = stratify_CENS, params_CENS = params_CENS,
            gform_TRT = gform_TRT, params_TRT = params_TRT,
            gform_MONITOR = gform_MONITOR, params_MONITOR = params_MONITOR)
```

```
## End(Not run)

# ----------------------------------------------------------------------
# Running TMLE based on the previous fit of the propensity scores.
# Also applying Random Forest to estimate the sequential outcome model
# ----------------------------------------------------------------------
## Not run:
t.surv <- c(0:5)
Qforms <- rep.int("Q.kplus1 ~ CVD + highA1c + N + lastNat1 + TI + TI.tminus1", (max(t.surv)+1))
params_Q = list(fit.package = "h2o", fit.algorithm = "randomForest",
                ntrees = 100, learn_rate = 0.05, sample_rate = 0.8,
                col_sample_rate = 0.8, balance_classes = TRUE)
tmle_est <- fitTMLE(OData, t_periods = t.surv, intervened_TRT = "TI.set1",
            Qforms = Qforms, params_Q = params_Q,
            stratifyQ_by_rule = TRUE)

## End(Not run)

## Not run:
t.surv <- c(0:5)
Qforms <- rep.int("Q.kplus1 ~ CVD + highA1c + N + lastNat1 + TI + TI.tminus1", (max(t.surv)+1))
params_Q = list(fit.package = "h2o", fit.algorithm = "randomForest",
                ntrees = 100, learn_rate = 0.05, sample_rate = 0.8,
                col_sample_rate = 0.8, balance_classes = TRUE)
tmle_est <- fitTMLE(OData, t_periods = t.surv, intervened_TRT = "TI.set1",
            Qforms = Qforms, params_Q = params_Q,
            stratifyQ_by_rule = FALSE)

## End(Not run)

# ----------------------------------------------------------------------
# Ensemble Learning with SuperLearner (using h2oEnsemble R package):
# ----------------------------------------------------------------------
## Not run:
require('h2oEnsemble')
# ----------------------------------------------------------------------
# Define many learners at once via grid search over tuning parameter spaces (hyperparameters)
# ----------------------------------------------------------------------
# 1. Runs h2o.grid in the background for each individual learner and saves cross-validated risks.
# 2. Calls h2o.stack from h2oEnsemble package to evaluate the final SuperLearner.
# ----------------------------------------------------------------------
# Search criteria and grid search space for glm tuning parameters:
GLM_hyper_params <- list(search_criteria = list(strategy = "RandomDiscrete", max_models = 5),
                         alpha = c(0,1,seq(0.1,0.9,0.1)),
                         lambda = c(0,1e-7,1e-5,1e-3,1e-1))

# Search criteria and grid search space for Random Forests:
search_criteria <- list(strategy = "RandomDiscrete", max_models = 5, max_runtime_secs = 60*60)
RF_hyper_params <- list(search_criteria = search_criteria,
                        ntrees = c(100, 200, 300, 500),
                        mtries = 1:4,
                        max_depth = c(5, 10, 15, 20, 25),
                        sample_rate = c(0.7, 0.8, 0.9, 1.0),
```

```
                        col_sample_rate_per_tree = c(0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8),
                        balance_classes = c(TRUE, FALSE))

# Search criteria and grid search space for Gradient Boosting Machines (gbm):
GBM_hyper_params <- list(search_criteria = search_criteria,
                         ntrees = c(100, 200, 300, 500),
                         learn_rate = c(0.005, 0.01, 0.03, 0.06),
                         max_depth = c(3, 4, 5, 6, 9),
                         sample_rate = c(0.7, 0.8, 0.9, 1.0),
                         col_sample_rate = c(0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8),
                         balance_classes = c(TRUE, FALSE))

# Specify individual learners to include in the SuperLearner library (in addition to grid models):
h2o.glm.1 <- function(..., alpha = 0.0) h2o.glm.wrapper(..., alpha = alpha)
h2o.glm.2 <- function(..., x = "highA1c", alpha = 0.0) h2o.glm.wrapper(..., x = x, alpha = alpha)
h2o.glm.3 <- function(..., alpha = 1.0) h2o.glm.wrapper(..., alpha = alpha)

# ------------------------------------------------------------------------
# Put all different algorithms together to define the final model ensemble:
# ------------------------------------------------------------------------
SLparams = list(fit.package = "h2o", fit.algorithm = "SuperLearner",
                grid.algorithm = c("glm", "randomForest", "gbm"),
                learner = c("h2o.glm.1", "h2o.glm.2", "h2o.glm.3"),
                metalearner = "h2o.glm_nn",
                nfolds = 10,
                seed = 23,
                glm = GLM_hyper_params,
                randomForest = RF_hyper_params,
                gbm = GBM_hyper_params)

# ------------------------------------------------------------------------
# Use save.ensemble and ensemble.dir.path arguments to save the final SuperLearner fit
# ------------------------------------------------------------------------
# This will also save the individual learner fits in the same directory.
# Only one directory per single SuperLearner fit is allowed.
# Can be loaded later on to save time and avoid refitting SuperLearner (load.ensemble = TRUE).

params_TRT = c(SLparams, save.ensemble = TRUE, ensemble.dir.path = "./h2o-ensemble-model-TRT")
params_CENS = list(fit.package = "speedglm", fit.algorithm = "glm")
params_MONITOR = list(fit.package = "speedglm", fit.algorithm = "glm")

OData <- fitPropensity(OData,
          gform_CENS = gform_CENS, stratify_CENS = stratify_CENS, params_CENS = params_CENS,
            gform_TRT = gform_TRT, params_TRT = params_TRT,
            gform_MONITOR = gform_MONITOR, params_MONITOR = params_MONITOR)

# ------------------------------------------------------------------------
# Re-using previously saved SuperLearner fit
# ------------------------------------------------------------------------
params_TRT = c(SLparams, load.ensemble = TRUE, ensemble.dir.path = "./h2o-ensemble-model-TRT")
params_CENS = list(fit.package = "speedglm", fit.algorithm = "glm")
params_MONITOR = list(fit.package = "speedglm", fit.algorithm = "glm")
```

```
OData <- fitPropensity(OData,
          gform_CENS = gform_CENS, stratify_CENS = stratify_CENS, params_CENS = params_CENS,
            gform_TRT = gform_TRT, params_TRT = params_TRT,
            gform_MONITOR = gform_MONITOR, params_MONITOR = params_MONITOR)

## End(Not run)
```

`fitTMLE` *TMLE wrapper for* `fitSeqGcomp`

Description

Calls `fitSeqGcomp` with argument `TMLE = TRUE`.

Usage

```
fitTMLE(...)
```

Arguments

`...`	Arguments that will be passed down to the underlying function `fitSeqGcomp`

Value

`data.table` with TMLE survival by time

See Also

`fitSeqGcomp`

Examples

```
options(stremr.verbose = TRUE)
require("data.table")
set_all_stremr_options(fit.package = "speedglm", fit.algorithm = "glm")

# ----------------------------------------------------------------------
# Simulated Data
# ----------------------------------------------------------------------
data(OdataNoCENS)
OdataDT <- as.data.table(OdataNoCENS, key=c(ID, t))

# define lagged N, first value is always 1 (always monitored at the first time point):
OdataDT[, ("N.tminus1") := shift(get("N"), n = 1L, type = "lag", fill = 1L), by = ID]
OdataDT[, ("TI.tminus1") := shift(get("TI"), n = 1L, type = "lag", fill = 1L), by = ID]

# ----------------------------------------------------------------------
# Define intervention (always treated):
# ----------------------------------------------------------------------
OdataDT[, ("TI.set1") := 1L]
```

```
OdataDT[, ("TI.set0") := 0L]

# ----------------------------------------------------------------------
# Import Data
# ----------------------------------------------------------------------
OData <- importData(OdataDT, ID = "ID", t = "t", covars = c("highA1c", "lastNat1", "N.tminus1"),
                    CENS = "C", TRT = "TI", MONITOR = "N", OUTCOME = "Y.tplus1")

# ----------------------------------------------------------------------
# Model the Propensity Scores
# ----------------------------------------------------------------------
gform_CENS <- "C ~ highA1c + lastNat1"
gform_TRT = "TI ~ CVD + highA1c + N.tminus1"
gform_MONITOR <- "N ~ 1"
stratify_CENS <- list(C=c("t < 16", "t == 16"))

# ----------------------------------------------------------------------
# Fit Propensity Scores
# ----------------------------------------------------------------------
OData <- fitPropensity(OData, gform_CENS = gform_CENS,
                       gform_TRT = gform_TRT,
                       gform_MONITOR = gform_MONITOR,
                       stratify_CENS = stratify_CENS)

# ----------------------------------------------------------------------
# IPW Ajusted KM or Saturated MSM
# ----------------------------------------------------------------------
require("magrittr")
AKME.St.1 <- getIPWeights(OData, intervened_TRT = "TI.set1") %>%
             survNPMSM(OData) %$%
             IPW_estimates
AKME.St.1

# ----------------------------------------------------------------------
# Bounded IPW
# ----------------------------------------------------------------------
IPW.St.1 <- getIPWeights(OData, intervened_TRT = "TI.set1") %>%
             survDirectIPW(OData)
IPW.St.1[]

# ----------------------------------------------------------------------
# IPW-MSM for hazard
# ----------------------------------------------------------------------
wts.DT.1 <- getIPWeights(OData = OData, intervened_TRT = "TI.set1", rule_name = "TI1")
wts.DT.0 <- getIPWeights(OData = OData, intervened_TRT = "TI.set0", rule_name = "TI0")
survMSM_res <- survMSM(list(wts.DT.1, wts.DT.0), OData, t_breaks = c(1:8,12,16)-1,)
survMSM_res$St

# ----------------------------------------------------------------------
# Sequential G-COMP
# ----------------------------------------------------------------------
t.surv <- c(0:15)
Qforms <- rep.int("Q.kplus1 ~ CVD + highA1c + N + lastNat1 + TI + TI.tminus1", (max(t.surv)+1))
```

```
params = list(fit.package = "speedglm", fit.algorithm = "glm")

## Not run:
gcomp_est <- fitSeqGcomp(OData, t_periods = t.surv, intervened_TRT = "TI.set1",
                         Qforms = Qforms, params_Q = params, stratifyQ_by_rule = FALSE)
gcomp_est[]

## End(Not run)
# ----------------------------------------------------------------------
# TMLE
# ----------------------------------------------------------------------
## Not run:
tmle_est <- fitTMLE(OData, t_periods = t.surv, intervened_TRT = "TI.set1",
                    Qforms = Qforms, params_Q = params, stratifyQ_by_rule = TRUE)
tmle_est[]

## End(Not run)

# ----------------------------------------------------------------------
# Running IPW-Adjusted KM with optional user-specified weights:
# ----------------------------------------------------------------------
addedWts_DT <- OdataDT[, c("ID", "t"), with = FALSE]
addedWts_DT[, new.wts := sample.int(10, nrow(OdataDT), replace = TRUE)/10]
survNP_res_addedWts <- survNPMSM(wts.DT.1, OData, weights = addedWts_DT)

# ----------------------------------------------------------------------
# Multivariate Propensity Score Regressions
# ----------------------------------------------------------------------
gform_CENS <- "C + TI + N ~ highA1c + lastNat1"
OData <- fitPropensity(OData, gform_CENS = gform_CENS, gform_TRT = gform_TRT,
                       gform_MONITOR = gform_MONITOR)

# ----------------------------------------------------------------------
# Fitting Propensity scores with Random Forests:
# ----------------------------------------------------------------------
## Not run:
set_all_stremr_options(fit.package = "h2o", fit.algorithm = "randomForest")
require("h2o")
h2o::h2o.init(nthreads = -1)
gform_CENS <- "C ~ highA1c + lastNat1"
OData <- fitPropensity(OData, gform_CENS = gform_CENS,
                       gform_TRT = gform_TRT,
                       gform_MONITOR = gform_MONITOR,
                       stratify_CENS = stratify_CENS)

# For Gradient Boosting machines:
set_all_stremr_options(fit.package = "h2o", fit.algorithm = "gbm")
# Use `H2O-3` distributed implementation of GLM
set_all_stremr_options(fit.package = "h2o", fit.algorithm = "glm")
# Use Deep Neural Nets:
set_all_stremr_options(fit.package = "h2o", fit.algorithm = "deeplearning")

## End(Not run)
```

```
# ------------------------------------------------------------------------
# Fitting different models with different algorithms
# Fine tuning modeling with optional tuning parameters.
# ------------------------------------------------------------------------
## Not run:
params_TRT = list(fit.package = "h2o", fit.algorithm = "gbm", ntrees = 50,
    learn_rate = 0.05, sample_rate = 0.8, col_sample_rate = 0.8,
    balance_classes = TRUE)
params_CENS = list(fit.package = "speedglm", fit.algorithm = "glm")
params_MONITOR = list(fit.package = "speedglm", fit.algorithm = "glm")
OData <- fitPropensity(OData,
          gform_CENS = gform_CENS, stratify_CENS = stratify_CENS, params_CENS = params_CENS,
            gform_TRT = gform_TRT, params_TRT = params_TRT,
            gform_MONITOR = gform_MONITOR, params_MONITOR = params_MONITOR)

## End(Not run)

# ------------------------------------------------------------------------
# Running TMLE based on the previous fit of the propensity scores.
# Also applying Random Forest to estimate the sequential outcome model
# ------------------------------------------------------------------------
## Not run:
t.surv <- c(0:5)
Qforms <- rep.int("Q.kplus1 ~ CVD + highA1c + N + lastNat1 + TI + TI.tminus1", (max(t.surv)+1))
params_Q = list(fit.package = "h2o", fit.algorithm = "randomForest",
                ntrees = 100, learn_rate = 0.05, sample_rate = 0.8,
                col_sample_rate = 0.8, balance_classes = TRUE)
tmle_est <- fitTMLE(OData, t_periods = t.surv, intervened_TRT = "TI.set1",
            Qforms = Qforms, params_Q = params_Q,
            stratifyQ_by_rule = TRUE)

## End(Not run)

## Not run:
t.surv <- c(0:5)
Qforms <- rep.int("Q.kplus1 ~ CVD + highA1c + N + lastNat1 + TI + TI.tminus1", (max(t.surv)+1))
params_Q = list(fit.package = "h2o", fit.algorithm = "randomForest",
                ntrees = 100, learn_rate = 0.05, sample_rate = 0.8,
                col_sample_rate = 0.8, balance_classes = TRUE)
tmle_est <- fitTMLE(OData, t_periods = t.surv, intervened_TRT = "TI.set1",
            Qforms = Qforms, params_Q = params_Q,
            stratifyQ_by_rule = FALSE)

## End(Not run)

# ------------------------------------------------------------------------
# Ensemble Learning with SuperLearner (using h2oEnsemble R package):
# ------------------------------------------------------------------------
## Not run:
require('h2oEnsemble')
# ------------------------------------------------------------------------
# Define many learners at once via grid search over tuning parameter spaces (hyperparameters)
```

```
# ----------------------------------------------------------------------
# 1. Runs h2o.grid in the background for each individual learner and saves cross-validated risks.
# 2. Calls h2o.stack from h2oEnsemble package to evaluate the final SuperLearner.
# ----------------------------------------------------------------------
# Search criteria and grid search space for glm tuning parameters:
GLM_hyper_params <- list(search_criteria = list(strategy = "RandomDiscrete", max_models = 5),
                         alpha = c(0,1,seq(0.1,0.9,0.1)),
                         lambda = c(0,1e-7,1e-5,1e-3,1e-1))

# Search criteria and grid search space for Random Forests:
search_criteria <- list(strategy = "RandomDiscrete", max_models = 5, max_runtime_secs = 60*60)
RF_hyper_params <- list(search_criteria = search_criteria,
                        ntrees = c(100, 200, 300, 500),
                        mtries = 1:4,
                        max_depth = c(5, 10, 15, 20, 25),
                        sample_rate = c(0.7, 0.8, 0.9, 1.0),
                        col_sample_rate_per_tree = c(0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8),
                        balance_classes = c(TRUE, FALSE))

# Search criteria and grid search space for Gradient Boosting Machines (gbm):
GBM_hyper_params <- list(search_criteria = search_criteria,
                         ntrees = c(100, 200, 300, 500),
                         learn_rate = c(0.005, 0.01, 0.03, 0.06),
                         max_depth = c(3, 4, 5, 6, 9),
                         sample_rate = c(0.7, 0.8, 0.9, 1.0),
                         col_sample_rate = c(0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8),
                         balance_classes = c(TRUE, FALSE))

# Specify individual learners to include in the SuperLearner library (in addition to grid models):
h2o.glm.1 <- function(..., alpha = 0.0) h2o.glm.wrapper(..., alpha = alpha)
h2o.glm.2 <- function(..., x = "highA1c", alpha = 0.0) h2o.glm.wrapper(..., x = x, alpha = alpha)
h2o.glm.3 <- function(..., alpha = 1.0) h2o.glm.wrapper(..., alpha = alpha)

# ----------------------------------------------------------------------
# Put all different algorithms together to define the final model ensemble:
# ----------------------------------------------------------------------
SLparams = list(fit.package = "h2o", fit.algorithm = "SuperLearner",
                grid.algorithm = c("glm", "randomForest", "gbm"),
                learner = c("h2o.glm.1", "h2o.glm.2", "h2o.glm.3"),
                metalearner = "h2o.glm_nn",
                nfolds = 10,
                seed = 23,
                glm = GLM_hyper_params,
                randomForest = RF_hyper_params,
                gbm = GBM_hyper_params)

# ----------------------------------------------------------------------
# Use save.ensemble and ensemble.dir.path arguments to save the final SuperLearner fit
# ----------------------------------------------------------------------
# This will also save the individual learner fits in the same directory.
# Only one directory per single SuperLearner fit is allowed.
# Can be loaded later on to save time and avoid refitting SuperLearner (load.ensemble = TRUE).
```

```
params_TRT = c(SLparams, save.ensemble = TRUE, ensemble.dir.path = "./h2o-ensemble-model-TRT")
params_CENS = list(fit.package = "speedglm", fit.algorithm = "glm")
params_MONITOR = list(fit.package = "speedglm", fit.algorithm = "glm")

OData <- fitPropensity(OData,
          gform_CENS = gform_CENS, stratify_CENS = stratify_CENS, params_CENS = params_CENS,
            gform_TRT = gform_TRT, params_TRT = params_TRT,
            gform_MONITOR = gform_MONITOR, params_MONITOR = params_MONITOR)

# ----------------------------------------------------------------------
# Re-using previously saved SuperLearner fit
# ----------------------------------------------------------------------
params_TRT = c(SLparams, load.ensemble = TRUE, ensemble.dir.path = "./h2o-ensemble-model-TRT")
params_CENS = list(fit.package = "speedglm", fit.algorithm = "glm")
params_MONITOR = list(fit.package = "speedglm", fit.algorithm = "glm")

OData <- fitPropensity(OData,
          gform_CENS = gform_CENS, stratify_CENS = stratify_CENS, params_CENS = params_CENS,
            gform_TRT = gform_TRT, params_TRT = params_TRT,
            gform_MONITOR = gform_MONITOR, params_MONITOR = params_MONITOR)

## End(Not run)
```

`GenericModel` *Generic R6 class for modeling (fitting and predicting) P(A=a|W=w) where A can be a multivariate (A[1], ..., A[k]) and each A[i] can be binary, categorical or continous*

Description

This R6 class Class for defining, fitting and predicting the probability model `P(A|W)` under `g_star` or under `g_0` for variables (`A`,`W`). Defines and manages the factorization of the multivariate conditional probability model `P(A=a|...)` into univariate regression models `A[j] ~ A[j-1] + ... + A[1] + W`. The class `self$new` method automatically figures out the correct joint probability factorization into univariate conditional probabilities based on name ordering provided by (`A_nms`, `W_nms`). When the outcome variable `A[j]` is binary, this class will automatically call a new instance of `BinaryOutcomeModel` class. Provide `self$fit()` function argument `data` as a `DataStorageClass` class object. This data will be used for fitting the model `P(A|W)`. Provide `self$fit()` function argument `newdata` (also as `DataStorageClass` class) for predictions of the type `P(A=1|W=w)`, where `w` values are coming from `newdata` object. Finally, provide `self$predictAeqa` function `newdata` argument (also `DataStorageClass` class object) for getting the likelihood predictions `P(A=sa|W=w)`, where both, `sa` and `sw` values are coming from `newdata` object.

Usage

```
GenericModel
```

Format

An `R6Class` generator object

Details

- n_regs - .

Methods

new(reg, ...) ...

length ...

getPsAsW.models ...

getcumprodAeqa ...

copy.fit(GenericModel) ...

fit(data) ...

predict(newdata) ...

predictAeqa(newdata, ...) ...

Active Bindings

wipe.alldat ...

---

getIPWeights *Inverse Probability Weights.*

Description

Evaluate the inverse probability weights for up to 3 intervention nodes: CENS, TRT and MONITOR. This is based on the inverse of the propensity score fits for the observed likelihood (g0.C, g0.A, g0.N), multiplied by the indicator of not being censored and the probability of each intervention in intervened_TRT and intervened_MONITOR. Requires column name(s) that specify the counterfactual node values or the counterfactual probabilities of each node being 1 (for stochastic interventions). The output is person-specific data with evaluated weights, wts.DT, only observation-times with non-zero weight are kept Can be one regimen per single run of this block, which are then combined into a list of output datasets with lapply. Alternative is to allow input with several rules/regimens, which are automatically combined into a list of output datasets.

Usage

```
getIPWeights(OData, intervened_TRT = NULL, intervened_MONITOR = NULL,
  useonly_t_TRT = NULL, useonly_t_MONITOR = NULL,
  rule_name = paste0(c(intervened_TRT, intervened_MONITOR), collapse = ""))
```

Arguments

OData	Input data object created by importData function.
intervened_TRT	Column name in the input data with the probabilities (or indicators) of counterfactual treatment nodes being equal to 1 at each time point. Leave the argument unspecified (NULL) when not intervening on treatment node(s).
intervened_MONITOR	Column name in the input data with probabilities (or indicators) of counterfactual monitoring nodes being equal to 1 at each time point. Leave the argument unspecified (NULL) when not intervening on the monitoring node(s).
useonly_t_TRT	Use for intervening only on some subset of observation and time-specific treatment nodes. Should be a character string with a logical expression that defines the subset of intervention observations. For example, using "TRT == 0" will intervene only at observations with the value of TRT being equal to zero. The expression can contain any variable name that was defined in the input dataset. Leave as NULL when intervening on all observations/time-points.
useonly_t_MONITOR	Same as useonly_t_TRT, but for monitoring nodes.
rule_name	Optional name for the treatment/monitoring regimen.

Value

...

Examples

```
options(stremr.verbose = TRUE)
require("data.table")
set_all_stremr_options(fit.package = "speedglm", fit.algorithm = "glm")

# ----------------------------------------------------------------------
# Simulated Data
# ----------------------------------------------------------------------
data(OdataNoCENS)
OdataDT <- as.data.table(OdataNoCENS, key=c(ID, t))

# define lagged N, first value is always 1 (always monitored at the first time point):
OdataDT[, ("N.tminus1") := shift(get("N"), n = 1L, type = "lag", fill = 1L), by = ID]
OdataDT[, ("TI.tminus1") := shift(get("TI"), n = 1L, type = "lag", fill = 1L), by = ID]

# ----------------------------------------------------------------------
# Define intervention (always treated):
# ----------------------------------------------------------------------
OdataDT[, ("TI.set1") := 1L]
OdataDT[, ("TI.set0") := 0L]

# ----------------------------------------------------------------------
# Import Data
# ----------------------------------------------------------------------
OData <- importData(OdataDT, ID = "ID", t = "t", covars = c("highA1c", "lastNat1", "N.tminus1"),
```

```
                    CENS = "C", TRT = "TI", MONITOR = "N", OUTCOME = "Y.tplus1")

# ----------------------------------------------------------------------
# Model the Propensity Scores
# ----------------------------------------------------------------------
gform_CENS <- "C ~ highA1c + lastNat1"
gform_TRT = "TI ~ CVD + highA1c + N.tminus1"
gform_MONITOR <- "N ~ 1"
stratify_CENS <- list(C=c("t < 16", "t == 16"))

# ----------------------------------------------------------------------
# Fit Propensity Scores
# ----------------------------------------------------------------------
OData <- fitPropensity(OData, gform_CENS = gform_CENS,
                       gform_TRT = gform_TRT,
                       gform_MONITOR = gform_MONITOR,
                       stratify_CENS = stratify_CENS)

# ----------------------------------------------------------------------
# IPW Ajusted KM or Saturated MSM
# ----------------------------------------------------------------------
require("magrittr")
AKME.St.1 <- getIPWeights(OData, intervened_TRT = "TI.set1") %>%
             survNPMSM(OData) %$%
             IPW_estimates
AKME.St.1

# ----------------------------------------------------------------------
# Bounded IPW
# ----------------------------------------------------------------------
IPW.St.1 <- getIPWeights(OData, intervened_TRT = "TI.set1") %>%
             survDirectIPW(OData)
IPW.St.1[]

# ----------------------------------------------------------------------
# IPW-MSM for hazard
# ----------------------------------------------------------------------
wts.DT.1 <- getIPWeights(OData = OData, intervened_TRT = "TI.set1", rule_name = "TI1")
wts.DT.0 <- getIPWeights(OData = OData, intervened_TRT = "TI.set0", rule_name = "TI0")
survMSM_res <- survMSM(list(wts.DT.1, wts.DT.0), OData, t_breaks = c(1:8,12,16)-1,)
survMSM_res$St

# ----------------------------------------------------------------------
# Sequential G-COMP
# ----------------------------------------------------------------------
t.surv <- c(0:15)
Qforms <- rep.int("Q.kplus1 ~ CVD + highA1c + N + lastNat1 + TI + TI.tminus1", (max(t.surv)+1))
params = list(fit.package = "speedglm", fit.algorithm = "glm")

## Not run:
gcomp_est <- fitSeqGcomp(OData, t_periods = t.surv, intervened_TRT = "TI.set1",
                         Qforms = Qforms, params_Q = params, stratifyQ_by_rule = FALSE)
gcomp_est[]
```

```
## End(Not run)
# ------------------------------------------------------------------------
# TMLE
# ------------------------------------------------------------------------
## Not run:
tmle_est <- fitTMLE(OData, t_periods = t.surv, intervened_TRT = "TI.set1",
                    Qforms = Qforms, params_Q = params, stratifyQ_by_rule = TRUE)
tmle_est[]

## End(Not run)

# ------------------------------------------------------------------------
# Running IPW-Adjusted KM with optional user-specified weights:
# ------------------------------------------------------------------------
addedWts_DT <- OdataDT[, c("ID", "t"), with = FALSE]
addedWts_DT[, new.wts := sample.int(10, nrow(OdataDT), replace = TRUE)/10]
survNP_res_addedWts <- survNPMSM(wts.DT.1, OData, weights = addedWts_DT)

# ------------------------------------------------------------------------
# Multivariate Propensity Score Regressions
# ------------------------------------------------------------------------
gform_CENS <- "C + TI + N ~ highA1c + lastNat1"
OData <- fitPropensity(OData, gform_CENS = gform_CENS, gform_TRT = gform_TRT,
                       gform_MONITOR = gform_MONITOR)

# ------------------------------------------------------------------------
# Fitting Propensity scores with Random Forests:
# ------------------------------------------------------------------------
## Not run:
set_all_stremr_options(fit.package = "h2o", fit.algorithm = "randomForest")
require("h2o")
h2o::h2o.init(nthreads = -1)
gform_CENS <- "C ~ highA1c + lastNat1"
OData <- fitPropensity(OData, gform_CENS = gform_CENS,
                       gform_TRT = gform_TRT,
                       gform_MONITOR = gform_MONITOR,
                       stratify_CENS = stratify_CENS)

# For Gradient Boosting machines:
set_all_stremr_options(fit.package = "h2o", fit.algorithm = "gbm")
# Use `H2O-3` distributed implementation of GLM
set_all_stremr_options(fit.package = "h2o", fit.algorithm = "glm")
# Use Deep Neural Nets:
set_all_stremr_options(fit.package = "h2o", fit.algorithm = "deeplearning")

## End(Not run)

# ------------------------------------------------------------------------
# Fitting different models with different algorithms
# Fine tuning modeling with optional tuning parameters.
# ------------------------------------------------------------------------
## Not run:
```

```
params_TRT = list(fit.package = "h2o", fit.algorithm = "gbm", ntrees = 50,
    learn_rate = 0.05, sample_rate = 0.8, col_sample_rate = 0.8,
    balance_classes = TRUE)
params_CENS = list(fit.package = "speedglm", fit.algorithm = "glm")
params_MONITOR = list(fit.package = "speedglm", fit.algorithm = "glm")
OData <- fitPropensity(OData,
          gform_CENS = gform_CENS, stratify_CENS = stratify_CENS, params_CENS = params_CENS,
            gform_TRT = gform_TRT, params_TRT = params_TRT,
            gform_MONITOR = gform_MONITOR, params_MONITOR = params_MONITOR)

## End(Not run)

# ----------------------------------------------------------------------
# Running TMLE based on the previous fit of the propensity scores.
# Also applying Random Forest to estimate the sequential outcome model
# ----------------------------------------------------------------------
## Not run:
t.surv <- c(0:5)
Qforms <- rep.int("Q.kplus1 ~ CVD + highA1c + N + lastNat1 + TI + TI.tminus1", (max(t.surv)+1))
params_Q = list(fit.package = "h2o", fit.algorithm = "randomForest",
                ntrees = 100, learn_rate = 0.05, sample_rate = 0.8,
                col_sample_rate = 0.8, balance_classes = TRUE)
tmle_est <- fitTMLE(OData, t_periods = t.surv, intervened_TRT = "TI.set1",
            Qforms = Qforms, params_Q = params_Q,
            stratifyQ_by_rule = TRUE)

## End(Not run)

## Not run:
t.surv <- c(0:5)
Qforms <- rep.int("Q.kplus1 ~ CVD + highA1c + N + lastNat1 + TI + TI.tminus1", (max(t.surv)+1))
params_Q = list(fit.package = "h2o", fit.algorithm = "randomForest",
                ntrees = 100, learn_rate = 0.05, sample_rate = 0.8,
                col_sample_rate = 0.8, balance_classes = TRUE)
tmle_est <- fitTMLE(OData, t_periods = t.surv, intervened_TRT = "TI.set1",
            Qforms = Qforms, params_Q = params_Q,
            stratifyQ_by_rule = FALSE)

## End(Not run)

# ----------------------------------------------------------------------
# Ensemble Learning with SuperLearner (using h2oEnsemble R package):
# ----------------------------------------------------------------------
## Not run:
require('h2oEnsemble')
# ----------------------------------------------------------------------
# Define many learners at once via grid search over tuning parameter spaces (hyperparameters)
# ----------------------------------------------------------------------
# 1. Runs h2o.grid in the background for each individual learner and saves cross-validated risks.
# 2. Calls h2o.stack from h2oEnsemble package to evaluate the final SuperLearner.
# ----------------------------------------------------------------------
# Search criteria and grid search space for glm tuning parameters:
GLM_hyper_params <- list(search_criteria = list(strategy = "RandomDiscrete", max_models = 5),
```

```
                        alpha = c(0,1,seq(0.1,0.9,0.1)),
                        lambda = c(0,1e-7,1e-5,1e-3,1e-1))

# Search criteria and grid search space for Random Forests:
search_criteria <- list(strategy = "RandomDiscrete", max_models = 5, max_runtime_secs = 60*60)
RF_hyper_params <- list(search_criteria = search_criteria,
                        ntrees = c(100, 200, 300, 500),
                        mtries = 1:4,
                        max_depth = c(5, 10, 15, 20, 25),
                        sample_rate = c(0.7, 0.8, 0.9, 1.0),
                        col_sample_rate_per_tree = c(0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8),
                        balance_classes = c(TRUE, FALSE))

# Search criteria and grid search space for Gradient Boosting Machines (gbm):
GBM_hyper_params <- list(search_criteria = search_criteria,
                         ntrees = c(100, 200, 300, 500),
                         learn_rate = c(0.005, 0.01, 0.03, 0.06),
                         max_depth = c(3, 4, 5, 6, 9),
                         sample_rate = c(0.7, 0.8, 0.9, 1.0),
                         col_sample_rate = c(0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8),
                         balance_classes = c(TRUE, FALSE))

# Specify individual learners to include in the SuperLearner library (in addition to grid models):
h2o.glm.1 <- function(..., alpha = 0.0) h2o.glm.wrapper(..., alpha = alpha)
h2o.glm.2 <- function(..., x = "highA1c", alpha = 0.0) h2o.glm.wrapper(..., x = x, alpha = alpha)
h2o.glm.3 <- function(..., alpha = 1.0) h2o.glm.wrapper(..., alpha = alpha)

# ------------------------------------------------------------------------
# Put all different algorithms together to define the final model ensemble:
# ------------------------------------------------------------------------
SLparams = list(fit.package = "h2o", fit.algorithm = "SuperLearner",
                grid.algorithm = c("glm", "randomForest", "gbm"),
                learner = c("h2o.glm.1", "h2o.glm.2", "h2o.glm.3"),
                metalearner = "h2o.glm_nn",
                nfolds = 10,
                seed = 23,
                glm = GLM_hyper_params,
                randomForest = RF_hyper_params,
                gbm = GBM_hyper_params)

# ------------------------------------------------------------------------
# Use save.ensemble and ensemble.dir.path arguments to save the final SuperLearner fit
# ------------------------------------------------------------------------
# This will also save the individual learner fits in the same directory.
# Only one directory per single SuperLearner fit is allowed.
# Can be loaded later on to save time and avoid refitting SuperLearner (load.ensemble = TRUE).

params_TRT = c(SLparams, save.ensemble = TRUE, ensemble.dir.path = "./h2o-ensemble-model-TRT")
params_CENS = list(fit.package = "speedglm", fit.algorithm = "glm")
params_MONITOR = list(fit.package = "speedglm", fit.algorithm = "glm")

OData <- fitPropensity(OData,
          gform_CENS = gform_CENS, stratify_CENS = stratify_CENS, params_CENS = params_CENS,
```

```
            gform_TRT = gform_TRT, params_TRT = params_TRT,
            gform_MONITOR = gform_MONITOR, params_MONITOR = params_MONITOR)

# ----------------------------------------------------------------------
# Re-using previously saved SuperLearner fit
# ----------------------------------------------------------------------
params_TRT = c(SLparams, load.ensemble = TRUE, ensemble.dir.path = "./h2o-ensemble-model-TRT")
params_CENS = list(fit.package = "speedglm", fit.algorithm = "glm")
params_MONITOR = list(fit.package = "speedglm", fit.algorithm = "glm")

OData <- fitPropensity(OData,
          gform_CENS = gform_CENS, stratify_CENS = stratify_CENS, params_CENS = params_CENS,
            gform_TRT = gform_TRT, params_TRT = params_TRT,
            gform_MONITOR = gform_MONITOR, params_MONITOR = params_MONITOR)

## End(Not run)
```

`get_MSM_RDs` *Risk Difference Estimates and SEs for IPW-MSM*

Description

Produces table(s) with pair-wise risk differences for all regimens that were used for fitting IPW-MSM. The corresponding SEs are evaluated based on the estimated influence curves (IC).

Usage

```
get_MSM_RDs(MSM, t.periods.RDs, getSEs = TRUE)
```

Arguments

`MSM`	Object returned by `survMSM`.
`t.periods.RDs`	Vector of time-points for evaluation of pairwise risk differences.
`getSEs`	Evaluate the influence curve based RD estimates of standard errors (SEs) along with point estimates?

Value

A list with RD tables. One table for each time-point in `t.periods.RDs`.

See Also

`survMSM` for estimation with MSM.

get_TMLE_RDs *Risk Difference Estimates and SEs for a list of TMLE outputs*

Description

Produces table(s) with pair-wise risk differences for all regimens that were used for fitting TMLE. The corresponding SEs are evaluated based on the estimated influence curves (IC).

Usage

```
get_TMLE_RDs(TMLE_list, t.periods.RDs)
```

Arguments

TMLE_list	A list of objects returned by fitIterTMLE, fitTMLE or fitSeqGcomp.
t.periods.RDs	Vector of time-points for evaluation of pairwise risk differences.

Value

A list with RD tables. One table for each time-point in t.periods.RDs.

See Also

survMSM for estimation with MSM.

get_wtsummary *IP-Weights Summary Tables*

Description

Produces various table summaries of IP-Weights.

Usage

```
get_wtsummary(wts_data, cutoffs = c(0, 0.5, 1, 10, 20, 30, 40, 50, 100, 150),
  varname = "Stabilized IPAW", by.rule = FALSE)
```

Arguments

wts_data	Either a list of data.table containing weights (one for each separate regimen/intervention) or a single data.table with weights for one regimen / intervention.
cutoffs	Weight cut off points for summary tables.
varname	Character string describing the type of the weights
by.rule	Can optionally evaluate the same summary tables separately for each regimen / rule.

Value

A list with various IP-weights summary tables.

See Also

`getIPWeights` for evaluation of IP-weights.

`h2o.glm_nn` *h2o glm wrapper for non-negative least squares*

Description

This wrapper is used as a default metalearner for SuperLearner / h2oEnsemble.

Usage

```
h2o.glm_nn(..., non_negative = TRUE)
```

Arguments

`...`	All arguments which will be passed down to `h2oEnsemble::h2o.glm.wrapper(...)`
`non_negative`	Flag enabling non-negative least squares

Value

...

`importData` *Import data, define various nodes, define dummies for factor columns and define OData R6 object*

Description

Import data, define various nodes, define dummies for factor columns and define OData R6 object

Usage

```
importData(data, ID = "Subject_ID", t_name = "time_period", covars,
  CENS = "C", TRT = "A", MONITOR = "N", OUTCOME = "Y", noCENScat = 0L,
  verbose = getOption("stremr.verbose"))
```

Arguments

data	Input data in long format. Can be a data.frame or a data.table with named columns, containing the time-varying covariates (covars), the right-censoring event indicator(s) (CENS), the exposure variable(s) (TRT), the monitoring process variable(s) (MONITOR) and the survival OUTCOME variable (OUTCOME).
ID	Unique subject identifier column name in data.
t_name	The name of the time/period variable in data.
covars	Vector of names with time varying and baseline covariates in data. This argument does not need to be specified, by default all variables that are not in ID, t, CENS, TRT, MONITOR and OUTCOME will be considered as covariates.
CENS	Column name of the censoring variable(s) in data. Each separate variable specified in CENS can be either binary (0/1 valued integer) or categorical (integer). For binary indicators of CENSoring, the value of 1 indicates the CENSoring or end of follow-up event (this cannot be changed). For categorical CENSoring variables, by default the value of 0 indicates no CENSoring / continuation of follow-up and other values indicate different reasons for CENSoring. Use the argument noCENScat to change the reference (continuation of follow-up) category from default 0 to any other value. (NOTE: Changing noCENScat has zero effect on coding of the binary CENSoring variables, those have to always use 1 to code the CENSoring event). Note that factors are not allowed in CENS.
TRT	A column name in data for the exposure/treatment variable(s).
MONITOR	A column name in data for the indicator(s) of monitoring events.
OUTCOME	A column name in data for the survival OUTCOME variable name, code as 1 for the outcome event.
noCENScat	The level (integer) that indicates CONTINUATION OF FOLLOW-UP for ALL censoring variables. Defaults is 0. Use this to modify the default reference category (no CENSoring / continuation of follow-up) for variables specifed in CENS.
verbose	Set to TRUE to print messages on status and information to the console. Turn this on by default using options(stremr.verbose=TRUE).

Value

...

Examples

```
options(stremr.verbose = TRUE)
require("data.table")
set_all_stremr_options(fit.package = "speedglm", fit.algorithm = "glm")

# ----------------------------------------------------------------------
# Simulated Data
# ----------------------------------------------------------------------
data(OdataNoCENS)
OdataDT <- as.data.table(OdataNoCENS, key=c(ID, t))
```

```
# define lagged N, first value is always 1 (always monitored at the first time point):
OdataDT[, ("N.tminus1") := shift(get("N"), n = 1L, type = "lag", fill = 1L), by = ID]
OdataDT[, ("TI.tminus1") := shift(get("TI"), n = 1L, type = "lag", fill = 1L), by = ID]

# ----------------------------------------------------------------------
# Define intervention (always treated):
# ----------------------------------------------------------------------
OdataDT[, ("TI.set1") := 1L]
OdataDT[, ("TI.set0") := 0L]

# ----------------------------------------------------------------------
# Import Data
# ----------------------------------------------------------------------
OData <- importData(OdataDT, ID = "ID", t = "t", covars = c("highA1c", "lastNat1", "N.tminus1"),
                    CENS = "C", TRT = "TI", MONITOR = "N", OUTCOME = "Y.tplus1")

# ----------------------------------------------------------------------
# Model the Propensity Scores
# ----------------------------------------------------------------------
gform_CENS <- "C ~ highA1c + lastNat1"
gform_TRT = "TI ~ CVD + highA1c + N.tminus1"
gform_MONITOR <- "N ~ 1"
stratify_CENS <- list(C=c("t < 16", "t == 16"))

# ----------------------------------------------------------------------
# Fit Propensity Scores
# ----------------------------------------------------------------------
OData <- fitPropensity(OData, gform_CENS = gform_CENS,
                       gform_TRT = gform_TRT,
                       gform_MONITOR = gform_MONITOR,
                       stratify_CENS = stratify_CENS)

# ----------------------------------------------------------------------
# IPW Ajusted KM or Saturated MSM
# ----------------------------------------------------------------------
require("magrittr")
AKME.St.1 <- getIPWeights(OData, intervened_TRT = "TI.set1") %>%
             survNPMSM(OData) %$%
             IPW_estimates
AKME.St.1

# ----------------------------------------------------------------------
# Bounded IPW
# ----------------------------------------------------------------------
IPW.St.1 <- getIPWeights(OData, intervened_TRT = "TI.set1") %>%
             survDirectIPW(OData)
IPW.St.1[]

# ----------------------------------------------------------------------
# IPW-MSM for hazard
# ----------------------------------------------------------------------
wts.DT.1 <- getIPWeights(OData = OData, intervened_TRT = "TI.set1", rule_name = "TI1")
```

```
wts.DT.0 <- getIPWeights(OData = OData, intervened_TRT = "TI.set0", rule_name = "TI0")
survMSM_res <- survMSM(list(wts.DT.1, wts.DT.0), OData, t_breaks = c(1:8,12,16)-1,)
survMSM_res$St

# ----------------------------------------------------------------------
# Sequential G-COMP
# ----------------------------------------------------------------------
t.surv <- c(0:15)
Qforms <- rep.int("Q.kplus1 ~ CVD + highA1c + N + lastNat1 + TI + TI.tminus1", (max(t.surv)+1))
params = list(fit.package = "speedglm", fit.algorithm = "glm")

## Not run:
gcomp_est <- fitSeqGcomp(OData, t_periods = t.surv, intervened_TRT = "TI.set1",
                         Qforms = Qforms, params_Q = params, stratifyQ_by_rule = FALSE)
gcomp_est[]

## End(Not run)
# ----------------------------------------------------------------------
# TMLE
# ----------------------------------------------------------------------
## Not run:
tmle_est <- fitTMLE(OData, t_periods = t.surv, intervened_TRT = "TI.set1",
                    Qforms = Qforms, params_Q = params, stratifyQ_by_rule = TRUE)
tmle_est[]

## End(Not run)

# ----------------------------------------------------------------------
# Running IPW-Adjusted KM with optional user-specified weights:
# ----------------------------------------------------------------------
addedWts_DT <- OdataDT[, c("ID", "t"), with = FALSE]
addedWts_DT[, new.wts := sample.int(10, nrow(OdataDT), replace = TRUE)/10]
survNP_res_addedWts <- survNPMSM(wts.DT.1, OData, weights = addedWts_DT)

# ----------------------------------------------------------------------
# Multivariate Propensity Score Regressions
# ----------------------------------------------------------------------
gform_CENS <- "C + TI + N ~ highA1c + lastNat1"
OData <- fitPropensity(OData, gform_CENS = gform_CENS, gform_TRT = gform_TRT,
                       gform_MONITOR = gform_MONITOR)

# ----------------------------------------------------------------------
# Fitting Propensity scores with Random Forests:
# ----------------------------------------------------------------------
## Not run:
set_all_stremr_options(fit.package = "h2o", fit.algorithm = "randomForest")
require("h2o")
h2o::h2o.init(nthreads = -1)
gform_CENS <- "C ~ highA1c + lastNat1"
OData <- fitPropensity(OData, gform_CENS = gform_CENS,
                       gform_TRT = gform_TRT,
                       gform_MONITOR = gform_MONITOR,
                       stratify_CENS = stratify_CENS)
```

```
# For Gradient Boosting machines:
set_all_stremr_options(fit.package = "h2o", fit.algorithm = "gbm")
# Use `H2O-3` distributed implementation of GLM
set_all_stremr_options(fit.package = "h2o", fit.algorithm = "glm")
# Use Deep Neural Nets:
set_all_stremr_options(fit.package = "h2o", fit.algorithm = "deeplearning")

## End(Not run)

# ----------------------------------------------------------------------
# Fitting different models with different algorithms
# Fine tuning modeling with optional tuning parameters.
# ----------------------------------------------------------------------
## Not run:
params_TRT = list(fit.package = "h2o", fit.algorithm = "gbm", ntrees = 50,
    learn_rate = 0.05, sample_rate = 0.8, col_sample_rate = 0.8,
    balance_classes = TRUE)
params_CENS = list(fit.package = "speedglm", fit.algorithm = "glm")
params_MONITOR = list(fit.package = "speedglm", fit.algorithm = "glm")
OData <- fitPropensity(OData,
         gform_CENS = gform_CENS, stratify_CENS = stratify_CENS, params_CENS = params_CENS,
           gform_TRT = gform_TRT, params_TRT = params_TRT,
           gform_MONITOR = gform_MONITOR, params_MONITOR = params_MONITOR)

## End(Not run)

# ----------------------------------------------------------------------
# Running TMLE based on the previous fit of the propensity scores.
# Also applying Random Forest to estimate the sequential outcome model
# ----------------------------------------------------------------------
## Not run:
t.surv <- c(0:5)
Qforms <- rep.int("Q.kplus1 ~ CVD + highA1c + N + lastNat1 + TI + TI.tminus1", (max(t.surv)+1))
params_Q = list(fit.package = "h2o", fit.algorithm = "randomForest",
                ntrees = 100, learn_rate = 0.05, sample_rate = 0.8,
                col_sample_rate = 0.8, balance_classes = TRUE)
tmle_est <- fitTMLE(OData, t_periods = t.surv, intervened_TRT = "TI.set1",
            Qforms = Qforms, params_Q = params_Q,
            stratifyQ_by_rule = TRUE)

## End(Not run)

## Not run:
t.surv <- c(0:5)
Qforms <- rep.int("Q.kplus1 ~ CVD + highA1c + N + lastNat1 + TI + TI.tminus1", (max(t.surv)+1))
params_Q = list(fit.package = "h2o", fit.algorithm = "randomForest",
                ntrees = 100, learn_rate = 0.05, sample_rate = 0.8,
                col_sample_rate = 0.8, balance_classes = TRUE)
tmle_est <- fitTMLE(OData, t_periods = t.surv, intervened_TRT = "TI.set1",
            Qforms = Qforms, params_Q = params_Q,
            stratifyQ_by_rule = FALSE)
```

```
## End(Not run)

# ----------------------------------------------------------------------
# Ensemble Learning with SuperLearner (using h2oEnsemble R package):
# ----------------------------------------------------------------------
## Not run:
require('h2oEnsemble')
# ----------------------------------------------------------------------
# Define many learners at once via grid search over tuning parameter spaces (hyperparameters)
# ----------------------------------------------------------------------
# 1. Runs h2o.grid in the background for each individual learner and saves cross-validated risks.
# 2. Calls h2o.stack from h2oEnsemble package to evaluate the final SuperLearner.
# ----------------------------------------------------------------------
# Search criteria and grid search space for glm tuning parameters:
GLM_hyper_params <- list(search_criteria = list(strategy = "RandomDiscrete", max_models = 5),
                         alpha = c(0,1,seq(0.1,0.9,0.1)),
                         lambda = c(0,1e-7,1e-5,1e-3,1e-1))

# Search criteria and grid search space for Random Forests:
search_criteria <- list(strategy = "RandomDiscrete", max_models = 5, max_runtime_secs = 60*60)
RF_hyper_params <- list(search_criteria = search_criteria,
                        ntrees = c(100, 200, 300, 500),
                        mtries = 1:4,
                        max_depth = c(5, 10, 15, 20, 25),
                        sample_rate = c(0.7, 0.8, 0.9, 1.0),
                        col_sample_rate_per_tree = c(0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8),
                        balance_classes = c(TRUE, FALSE))

# Search criteria and grid search space for Gradient Boosting Machines (gbm):
GBM_hyper_params <- list(search_criteria = search_criteria,
                         ntrees = c(100, 200, 300, 500),
                         learn_rate = c(0.005, 0.01, 0.03, 0.06),
                         max_depth = c(3, 4, 5, 6, 9),
                         sample_rate = c(0.7, 0.8, 0.9, 1.0),
                         col_sample_rate = c(0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8),
                         balance_classes = c(TRUE, FALSE))

# Specify individual learners to include in the SuperLearner library (in addition to grid models):
h2o.glm.1 <- function(..., alpha = 0.0) h2o.glm.wrapper(..., alpha = alpha)
h2o.glm.2 <- function(..., x = "highA1c", alpha = 0.0) h2o.glm.wrapper(..., x = x, alpha = alpha)
h2o.glm.3 <- function(..., alpha = 1.0) h2o.glm.wrapper(..., alpha = alpha)

# ----------------------------------------------------------------------
# Put all different algorithms together to define the final model ensemble:
# ----------------------------------------------------------------------
SLparams = list(fit.package = "h2o", fit.algorithm = "SuperLearner",
                grid.algorithm = c("glm", "randomForest", "gbm"),
                learner = c("h2o.glm.1", "h2o.glm.2", "h2o.glm.3"),
                metalearner = "h2o.glm_nn",
                nfolds = 10,
                seed = 23,
                glm = GLM_hyper_params,
                randomForest = RF_hyper_params,
```

```
                gbm = GBM_hyper_params)

# ------------------------------------------------------------------------
# Use save.ensemble and ensemble.dir.path arguments to save the final SuperLearner fit
# ------------------------------------------------------------------------
# This will also save the individual learner fits in the same directory.
# Only one directory per single SuperLearner fit is allowed.
# Can be loaded later on to save time and avoid refitting SuperLearner (load.ensemble = TRUE).

params_TRT = c(SLparams, save.ensemble = TRUE, ensemble.dir.path = "./h2o-ensemble-model-TRT")
params_CENS = list(fit.package = "speedglm", fit.algorithm = "glm")
params_MONITOR = list(fit.package = "speedglm", fit.algorithm = "glm")

OData <- fitPropensity(OData,
          gform_CENS = gform_CENS, stratify_CENS = stratify_CENS, params_CENS = params_CENS,
            gform_TRT = gform_TRT, params_TRT = params_TRT,
            gform_MONITOR = gform_MONITOR, params_MONITOR = params_MONITOR)

# ------------------------------------------------------------------------
# Re-using previously saved SuperLearner fit
# ------------------------------------------------------------------------
params_TRT = c(SLparams, load.ensemble = TRUE, ensemble.dir.path = "./h2o-ensemble-model-TRT")
params_CENS = list(fit.package = "speedglm", fit.algorithm = "glm")
params_MONITOR = list(fit.package = "speedglm", fit.algorithm = "glm")

OData <- fitPropensity(OData,
          gform_CENS = gform_CENS, stratify_CENS = stratify_CENS, params_CENS = params_CENS,
            gform_TRT = gform_TRT, params_TRT = params_TRT,
            gform_MONITOR = gform_MONITOR, params_MONITOR = params_MONITOR)

## End(Not run)
```

`make_report_rmd` *Generate report(s) with modeling stats and survival estimates using pandoc.*

Description

Generate report(s) with modeling stats and survival estimates using pandoc.

Usage

```
make_report_rmd(OData, MSM, NPMSM, TMLE, GCOMP, wts_data, SurvByRegimen,
  WTtables = NULL, AddFUPtables = FALSE, MSM.RDtables, TMLE.RDtables,
  format = c("html", "pdf", "word"), skip.modelfits = FALSE,
  file.name = getOption("stremr.file.name"),
  file.path = getOption("stremr.file.path"), openFile = TRUE,
  keep_md = FALSE, keep_tex = FALSE, ...)
```

Arguments

OData	Input data object returned by the function importData.
MSM	The MSM object fits returned by the function survMSM.
NPMSM	Optional list of a resulting calls to survNPMSM or a result of a single call to survNPMSM.
TMLE	Optional list of a resulting calls to fitTMLE or a result of a single call to fitTMLE.
GCOMP	Optional list of a resulting calls to fitSeqGcomp or a result of a single call to fitSeqGcomp.
wts_data	Optional list of data.tables or a single data.table with weights by regimen.
SurvByRegimen	... Not implemented ...
WTtables	Table(s) with distribution(s) of the IPTW weights, a result of calling the function get_wtsummary
AddFUPtables	Logical, set to TRUE to print tables describing the distribution of the maximum follow-up times by rule (monitoring and treatment).
MSM.RDtables	List of tables with risk differences returned by the function get_MSM_RDs.
TMLE.RDtables	List of tables with risk differences returned by the function get_TMLE_RDs.
format	Choose the Pandoc output format for the report file (html, pdf or word). Note that the html report file is always produced in addition to any other selected format.
skip.modelfits	Do not report any of the modeling stats.
file.name	File name for the report file without extension. Default file name is assigned based on the current date.
file.path	Directory path where the report file(s) should be written. Default is to use the system temporary directory.
openFile	Open the report file with OS default viewer?
keep_md	Keep the source .md files?
keep_tex	Keep the source .tex files for pdf output?
...	Additional arguments may specify the report title (author), author (title). Specifying the logical flag only.coefs=TRUE disables printing of all h2o-specific model summaries. Additional set of arguments control the survival plotting, these are passed on to the function f_plot_survest: t_int_sel, y_lab, x_lab, miny, x_legend, y_legend.

Value

String specifying the path to the main report file.

OdataCatCENS *An example of a dataset in long format with categorical censoring variable.*

Description

Simulated dataset containing 1,000 i.i.d. observations organized in long format as person-time rows. The binary exposure is `TI` and binary outcome is `Y.tplus1`. See /tests/ for R code that generated this data.

Usage

```
data(OdataCatCENS)
```

Format

A data frame with 1,000 observations and variables:

ID Unique subject identifier

CVD Baseline confounder (time invariant)

t Interger for current time period, range 0-16

lastNat1 Time since last monitoring event, set to 0 when N[t-1]=0 and then added one for each new period where N[t] is 0.

highA1c Time-varying confounder

CatC Categorical censoring variable, range 0-2. The value of 0 indicates no censoring 1 or 2 indicates censoring (possibly for different reasons)

C Binary censoring indicator derived from CatC. 0 if CatC is 0 and 1 if CatC is 1 or 2.

TI Binary exposure variable

N The indicator of being monitored (having a visit)

Y.tplus1 Binary outcome at t

OdataNoCENS *An example of a dataset in long format with random monitoring and no right censoring.*

Description

Simulated dataset containing 1,000 i.i.d. observations organized in long format as person-time rows. The binary exposure is `TI` and binary outcome is `Y.tplus1`. See /tests/ for R code that generated this data.

Usage

```
data(OdataNoCENS)
```

Format

A data frame with 1,000 observations and variables:

ID Unique subject identifier

CVD Baseline confounder (time invariant)

t Interger for current time period, range 0-16

lastNat1 Time since last monitoring event, set to 0 when N[t-1]=0 and then added one for each new period where N[t] is 0.

highA1c Time-varying confounder

TI Binary exposure variable

C Administrative censoring indicator, always set to 0 unless the end of study is reached (t==16)

N The indicator of being monitored (having a visit)

Y.tplus1 Binary outcome at t

OdatDT_10K *An example of a dataset in long format with random monitoring process and no right censoring.*

Description

Simulated dataset containing 10,000 i.i.d. observations organized in long format as person-time rows. The binary exposure is `TI` and binary outcome is `Y.tplus1`. See /tests/ for R code that generated this data as well as R code that uses stremr to analyze this data.

Usage

```
data(OdatDT_10K)
```

Format

A data frame with 10,000 observations and variables:

ID Unique subject identifier

t Interger for current time period, range 0-16

CVD Baseline confounder (time invariant)

lastNat1 Time since last monitoring event, set to 0 when N[t-1]=0 and then added one for each new period where N[t] is 0.

highA1c Time-varying confounder

TI Binary exposure variable

C Administrative censoring indicator, always set to 0 unless the end of study is reached (t==16)

N The random indicator of being monitored (having a visit), simulated as a Bernoulli RV with P(N(t)=1)=0.5

Y.tplus1 Indicator of the survival event at t

gTI.dlow Counterfactual exposure under static intervention - always treat

gTI.dhigh Counterfactual exposure under dynamic intervention - treat only when highA1c is above 1 and the subject is being monitored

gPois3.yrly Poisson probability of counterfactual monitoring indicator being equal to 1

gPois3.biyrly Poisson probability of counterfactual monitoring indicator being equal to 1

gp05 Bernoulli probability of counterfactual monitoring indicator being equal to 1

`openFileInOS` *Open file*

Description

Tries to open a file with operating system's default program.

Usage

```
openFileInOS(f)
```

Arguments

`f`	file (with full path)

References

This function is a fork of David Hajage's `convert` function: https://github.com/eusebe/ascii/blob/master/R/export.r

---

`pander.H2OBinomialMetrics` *Pander method for H2OBinomialMetrics class*

Description

Prints a H2OBinomialMetrics object in Pandoc's markdown.

Usage

```
## S3 method for class 'H2OBinomialMetrics'
pander(H2OBinomialMetricsObject)
```

Arguments

`H2OBinomialMetricsObject`	H2OBinomialMetrics object

Value

By default this function outputs (see: `?cat`) the result. If you would want to catch the result instead, then call `pander_return` instead.

`pander.H2ORegressionMetrics` *Pander method for H2ORegressionMetrics class*

Description

Prints a H2ORegressionMetrics object in Pandoc's markdown.

Usage

```
## S3 method for class 'H2ORegressionMetrics'
pander(H2ORegressionMetricsObject)
```

Arguments

`H2ORegressionMetricsObject`	H2ORegressionMetrics object

Value

By default this function outputs (see: `?cat`) the result. If you would want to catch the result instead, then call `pander_return` instead.

`print.GLMmodel` *S3 methods for printing model fit summary for glmfit class object*

Description

Prints the modeling summary for the glm fit (`stats::glm.fit` or `speedglm::speedglm.wfit`)

Usage

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

Arguments

`x`	The model fit object produced by functions stremr:::fit.glm or stremr:::fit.speedglm
`...`	Additional options passed on to `summary.GLMmodel`.

Value

The output is printed with `cat`. To capture the markdown-formated model summary use `summary.GLMmodel`.

print.H2Oensemblemodel *S3 methods for printing model fit summary for H2Omodel class object*

Description

Prints the modeling summary for the h2o model fit (see h2o R package).

Usage

```
## S3 method for class 'H2Oensemblemodel'
print(x, only.coefs = FALSE, ...)
```

Arguments

x	The model fit object produced by any stremr S3 function starting with stremr:::H2Omodel.
only.coefs	Skip all of the h2o-specific model stats, only print the coefficients table (when running fit.algorithm = "GLM").
...	Additional options passed on to summary.H2Omodel.

Value

The output is printed with cat. To capture the markdown-formated model summary use summary.H2Omodel.

print.H2Omodel *S3 methods for printing model fit summary for H2Omodel class object*

Description

Prints the modeling summary for the h2o model fit (see h2o R package).

Usage

```
## S3 method for class 'H2Omodel'
print(x, only.coefs = FALSE, ...)
```

Arguments

x	The model fit object produced by any stremr S3 function starting with stremr:::H2Omodel.
only.coefs	Skip all of the h2o-specific model stats, only print the coefficients table (when running fit.algorithm = "GLM").
...	Additional options passed on to summary.H2Omodel.

Value

The output is printed with cat. To capture the markdown-formated model summary use summary.H2Omodel.

print_stremr_opts *Print Current Option Settings for* stremr

Description

Print Current Option Settings for stremr

Usage

```
print_stremr_opts()
```

Value

Invisibly returns a list of stremr options.

See Also

set_all_stremr_options

QlearnModel *R6 Class for Q-Learning*

Description

R6 class for controlling the internal implementation of Q-learning functionality. Supports sequential (recursive) G-computation and longitudinal TMLE. Inherits from BinaryOutcomeModel R6 Class.

Usage

```
QlearnModel
```

Format

An R6Class generator object

Details

- regimen_names - character. Note used. For future pooling across regimens.
- classify - ... logical
- TMLE - ... logical
- nIDs - ... integer
- stratifyQ_by_rule - ... logical
- lower_bound_zero_Q - ... logical
- skip_update_zero_Q - ... logical
- Qreg_counter - ... integer
- t_period - ... integer
- idx_used_to_fit_initQ - ... integer

Methods

new(reg, ...) ...

define.subset.idx(data, subset_vars, subset_exprs) ...

define_idx_to_fit_initQ(data) ...

define_idx_to_predictQ(data) ...

fit(overwrite = FALSE, data, ...) ...

Propagate_TMLE_fit(overwrite = TRUE, data, new.TMLE.fit, ...) ...

predict(newdata, subset_idx, ...) ...

predictStatic(data, g0, gstar, subset_idx) ...

predictStochastic(data, g0, gstar, subset_idx, stoch_indicator) ...

predictAeqa(newdata, ...) ...

get.fits() ...

Active Bindings

wipe.alldat ...

getfit ...

getTMLEfit ...

RegressionClass *R6 class that defines regression models evaluating P(sA|sW), for summary measures (sW,sA)*

Description

This R6 class defines fields and methods that controls all the parameters for non-parametric modeling and estimation of multivariate joint conditional probability model P(sA|sW) for summary measures (sA,sW). Note that sA can be multivariate and any component of sA[j] can be either binary, categorical or continuous. The joint probability for P(sA|sA) = P(sA[1],...,sA[k]|sA) is first factorized as P(sA[1]|sA) * P(sA[2]|sA, sA[1]) * ... * P(sA[k]|sA, sA[1],...,sA[k-1]), where each of these conditional probability models is defined by a new instance of a GenericModel class (and a corresponding instance of the RegressionClass class). If sA[j] is binary, the conditional probability P(sA[j]|sW,sA[1],...,sA[j-1]) is evaluated via logistic regression model. When sA[j] is continuous (or categorical), its estimation will be controlled by a new instance of the ContinModel class (or the CategorModel class), as well as the accompanying new instance of the RegressionClass class. The range of continuous sA[j] will be fist partitioned into K bins and the corresponding K bin indicators (B_1,...,B_K), with K new instances of GenericModel class, each instance defining a single logistic regression model for one binary bin indicator outcome B_j and predictors (sW, sA[1],...,sA[k-1]). Thus, the first instance of RegressionClass and GenericModel classes will automatically spawn recursive calls to new instances of these classes until the entire tree of binary logistic regressions that defines the joint probability P(sA|sW) is build.

Usage

```
RegressionClass
```

Format

An R6Class generator object

Details

- sep_predvars_sets - Logical indicating the type of regression to run, if TRUE fit the joint P(outvar|predvars) (default), More specifically, if FALSE (default), use the same predictors in predvars (vector of names) for all nodes in outvar; when TRUE uses separate sets in predvars (must be a named list of character vectors) for fitting each node in outvar.
- outvar.class - Character vector indicating a class of each outcome var: bin / cont / cat.
- outvar - Character vector of regression outcome variable names.
- predvars - Either a pooled character vector of all predictors (sW) or a vector of regression-specific predictor names. When sep_predvars_sets=TRUE, this must be a named list of predictor names, the list names corresponding to each node name in outvar, and each list item being a vector specifying the regression predictors for a specific outcome in outvar.
- reg_hazard - Logical, if TRUE, the joint probability model P(outvar | predvars) is factorized as \prod_jP(outvar[j] | predvars) for each j outvar (for fitting hazard).
- subset_vars - Subset variables (later evaluated to logical vector based on non-missing (!is.na()) values of these variables).
- subset_exprs - Subset expressions (later evaluated to logical vector in the envir of the data).
- ReplMisVal0 - Logical, if TRUE all gvars$misval among predicators are replaced with with gvars$misXreplace (0).
- nbins - Integer number of bins used for a continuous outvar, the intervals are defined inside ContinModel$new() and then saved in this field.
- bin_nms - Character vector of column names for bin indicators.
- useglm - Logical, if TRUE then fit the logistic regression model using glm.fit, if FALSE use speedglm.wfit.. regressions (requires registering back-end cluster prior to calling the fit/predict functions)..
- bin_bymass - Logical, for continuous outvar, create bin cutoffs based on equal mass distribution.
- bin_bydhist - Logical, if TRUE, use dhist approach for bin definitions. See Denby and Mallows "Variations on the Histogram" (2009)) for more..
- max_nperbin - Integer, maximum number of observations allowed per one bin.
- pool_cont - Logical, pool binned continuous outvar observations across bins and only fit only regression model across all bins (adding bin_ID as an extra covaraite)..
- outvars_to_pool - Character vector of names of the binned continuous outvars, should match bin_nms.

- intrvls.width - Named numeric vector of bin-widths (bw_j : j=1,...,M) for each each bin in self$intrvls. When sA is not continuous, intrvls.width IS SET TO 1. When sA is continuous and this variable intrvls.width is not here, the intervals are determined inside ContinModel$new() and are assigned to this variable as a list, with names(intrvls.width) <- reg$bin_nms. Can be queried by BinaryOutcomeModel$predictAeqa() as: intrvls.width[outvar].
- intrvls - Numeric vector of cutoffs defining the bins or a named list of numeric intervals for length(self$outvar) > 1.
- cat.levels - Numeric vector of all unique values in categorical outcome variable. Set by CategorModel constructor.

Methods

new(sep_predvars_sets = FALSE, outvar.class = gvars$sVartypes$bin, outvar, predvars, subset_vars, s
: Uses the arguments to instantiate an object of R6 class and define the future regression model.

ChangeManyToOneRegresssion(k_i, reg) Take a clone of a parent RegressionClass (reg) for length(self$outvar) regressions and set self to a single univariate k_i regression for outcome self$outvar[[k_i]].

ChangeOneToManyRegresssions(regs_list) Take the clone of a parent RegressionClass for univariate (continuous outvar) regression and set self to length(regs_list) bin indicator outcome regressions.

resetS3class() ...

Active Bindings

S3class ...

get.reg ...

---

set_all_stremr_options

Setting stremr *Options*

Description

Options that control stremr package. **Will reset all unspecified options (omitted arguments) to their default values**. The preferred way to set options for stremr is to use stremrOptions, which allows specifying individual options without having to reset all other options. To reset all options to their defaults simply run set_all_stremr_options() without any parameters/arguments.

Usage

```
set_all_stremr_options(fit.package = c("speedglm", "glm", "h2o"),
  fit.algorithm = c("glm", "gbm", "randomForest", "deeplearning",
  "SuperLearner"), bin.method = c("equal.mass", "equal.len", "dhist"),
  nbins = 10, maxncats = 20, maxNperBin = 500,
  lower_bound_zero_Q = TRUE, skip_update_zero_Q = TRUE)
```

Arguments

fit.package	Specify the default package for performing model fitting: c("speedglm", "glm", "h2o")
fit.algorithm	Specify the default fitting algorithm: c("glm", "gbm", "randomForest", "deeplearning", "SuperLearner")
bin.method	The method for choosing bins when discretizing and fitting the conditional continuous summary exposure variable sA. The default method is "equal.len", which partitions the range of sA into equal length nbins intervals. Method "equal.mass" results in a data-adaptive selection of the bins based on equal mass (equal number of observations), i.e., each bin is defined so that it contains an approximately the same number of observations across all bins. The maximum number of observations in each bin is controlled by parameter maxNperBin. Method "dhist" uses a mix of the above two approaches, see Denby and Mallows "Variations on the Histogram" (2009) for more detail.
nbins	Set the default number of bins when discretizing a continous outcome variable under setting bin.method = "equal.len". If left as NA the total number of equal intervals (bins) is determined by the nearest integer of nobs/maxNperBin, where nobs is the total number of observations in the input data.
maxncats	Max number of unique categories a categorical variable sA[j] can have. If sA[j] has more it is automatically considered continuous.
maxNperBin	Max number of observations per 1 bin for a continuous outcome (applies directly when bin.method="equal.mass" and indirectly when bin.method="equal.len", but nbins = NA).
lower_bound_zero_Q	Set to TRUE to bound the observation-specific Qs during the TMLE update step away from zero (with minimum value set at 10^-4). Can help numerically stabilize the TMLE intercept estimates in some small-sample cases. Has no effect when TMLE = FALSE.
skip_update_zero_Q	Set to FALSE to perform TMLE update with glm even when all of the Q's are zero. When set to TRUE the TMLE update step is skipped if the predicted Q's are either all 0 or near 0, with TMLE intercept being set to 0.

Value

Invisibly returns a list with old option settings.

See Also

stremrOptions, print_stremr_opts

`StratifiedModel` *R6 class for fitting and predicting with several stratified models for a single outcome variable (conditional on some covariate values)*

Description

This R6 class defines and fits a conditional probability model `P(A[j]|W,...)` for a summary `A[j]`. This class inherits from `GenericModel` class. The stratification criteria is determined by the `R` expression in the field `subset_exprs`.

Usage

```
StratifiedModel
```

Format

An `R6Class` generator object

Details

- `reg` - .
- `outvar` - .
- `subset_exprs` - .

Methods

`new(reg, DataStorageClass.g0, ...)` ...

`fit(data)` ...

`predict(newdata)` ...

`predictAeqa(newdata)` ...

Active Bindings

`cats` ...

stremr *Estimate Survival with Interventions on Exposure and MONITORing Process in Right Censored Longitudinal Data.*

Description

Estimate the causal survival curve for a particular stochastic, dynamic or static intervention on the treatment/exposure and monitoring process. Implements the **IPW** (Inverse Probability-Weighted or Horvitz-Thompson) estimator of the discrete survival hazard function which is mapped into survival function.

Usage

```
stremr(data, ID = "Subj_ID", t.name = "time_period", covars, CENS = "C",
  TRT = "A", MONITOR = "N", OUTCOME = "Y", gform_CENS, gform_TRT,
  gform_MONITOR, stratify_CENS = NULL, stratify_TRT = NULL,
  stratify_MONITOR = NULL, intervened_TRT = NULL,
  intervened_MONITOR = NULL, noCENScat = 0L,
  verbose = getOption("stremr.verbose"), optPars = list())
```

Arguments

data	Input data in long format. Can be a data.frame or a data.table with named columns, containing the time-varying covariates (covars), the right-censoring event indicator(s) (CENS), the exposure variable(s) (TRT), the monitoring process variable(s) (MONITOR) and the survival OUTCOME variable (OUTCOME).
ID	Unique subject identifier column name in data.
t.name	The name of the time/period variable in data.
covars	Vector of names with time varying and baseline covariates in data. This argument does not need to be specified, by default all variables that are not in ID, t, CENS, TRT, MONITOR and OUTCOME will be considered as covariates.
CENS	Column name of the censoring variable(s) in data. Each separate variable specified in CENS can be either binary (0/1 valued integer) or categorical (integer). For binary indicators of CENSoring, the value of 1 indicates the CENSoring or end of follow-up event (this cannot be changed). For categorical CENSoring variables, by default the value of 0 indicates no CENSoring / continuation of follow-up and other values indicate different reasons for CENSoring. Use the argument noCENScat to change the reference (continuation of follow-up) category from default 0 to any other value. (NOTE: Changing noCENScat has zero effect on coding of the binary CENSoring variables, those have to always use 1 to code the CENSoring event). Note that factors are not allowed in CENS.
TRT	A column name in data for the exposure/treatment variable(s).
MONITOR	A column name in data for the indicator(s) of monitoring events.
OUTCOME	A column name in data for the survival OUTCOME variable name, code as 1 for the outcome event.

`gform_CENS` Regression formula(s) for estimating the propensity score for the censoring mechanism: P(C(t) | W). See Details.

`gform_TRT` Regression formula(s) for propensity score for the exposure/treatment(s): P(A(t) | W). See Details.

`gform_MONITOR` Regression formula(s) for estimating the propensity score for the MONITORing process: P(N(t) | W). See Details. (the observed exposure mechanism), When omitted the regression is defined by `sA~sW`, where `sA`

`stratify_CENS` A named list with one item per variable in `CENS`. Each list item is a character vector of stratification subsets for the corresponding variable in `CENS`.

`stratify_TRT` A named list with one item per variable in `TRT`. Each list item is a character vector of stratification subsets for the corresponding variable in `TRT`.

`stratify_MONITOR` A named list with one item per variable in `MONITOR`. Each list item is a character vector of stratification subsets for the corresponding variable in `MONITOR`.

`intervened_TRT` Column name in `data` containing the counterfactual probabilities of following a specific treatment regimen.

`intervened_MONITOR` Column name in `data` containing the counterfactual probabilities of following a specific monitoring regimen.

`noCENScat` The level (integer) that indicates CONTINUATION OF FOLLOW-UP for ALL censoring variables. Defaults is 0. Use this to modify the default reference category (no CENSoring / continuation of follow-up) for variables specifed in `CENS`.

`verbose` Set to `TRUE` to print messages on status and information to the console. Turn this on by default using `options(stremr.verbose=TRUE)`.

`optPars` A named list of additional optional parameters to be passed to `stremr`, such as `alpha`, `lbound`, `family`, `YnodeDET`, `h_g0_GenericModel` and `h_gstar_GenericModel`. See Details below for the description of each parameter.

Value

...

Details

The regression formalas in `Qform`, `hform.g0` and `hform.gstar` can include any summary measures names defined in `sW` and `sA`, referenced by their individual variable names or by their aggregate summary measure names. For example, `hform.g0 = "netA ~ netW"` is equivalent to `hform.g0 = "A + A_netF1 + A_netF2 ~ W + W_netF1 + W_netF2"` for `sW,sA` summary measures defined by `def_sW(netW=W[[0:2]])` and `def_sA(netA=A[[0:2]])`.

Additional parameters

Some of the parameters that control the estimation in `stremr` can be set by calling the function `set_all_stremr_options`.

Additional parameters can be also specified as a named list `optPars` argument of the `stremr` function. The items that can be specified in `optPars` are:

- `alpha` - alpha-level for CI calculation (0.05 for 95
- `lbound` - One value for symmetrical bounds on P(sW | sW).
- `family` - Family specification for regression models, defaults to binomial (CURRENTLY ONLY BINOMIAL FAMILY IS IMPLEMENTED).

Specifying the counterfactual intervention function (`f_gstar1` and `optPars$f_gstar2`)

The functions `f_gstar1` and `f_gstar2` can only depend on variables specified by the combined matrix of summary measures (`sW`,`sA`), which is passed using the argument `data`. The functions should return a vector of length `nrow(data)` of counterfactual treatments for observations in the input data.

IPTW estimator

****

- As described in the following section, the first step is to construct an estimator $P_{g_N}(A(t)|L(t))$ for the probability of exposure $P_{g_0}(A(t)|W(t))$.
- Based on the user specified stochastic intervention, we can also obtain $P_{g^*_N}(A^*(t)|L(t)$
- Combining the two probabilities forms the basis of the IPTW estimator, which is evaluated at the observed N data points $O_i = ((L_i(t), A_i(t) : t = 0, ..., K), Y_i), i = 1, ..., N$ and is given by

$$\psi_n^{IPTW} = \sum_{i=1,...,N} Y_i \frac{P_{g^*_N}(A^*(t) = A_i(t)|L(t) = L_i(t))}{P_{g_N}(A(t) = A_i(t)|L(t) = L_i(t))}.$$

See Also

`stremr-package` for the general overview of the package,

Examples

```
#----------------------------------------------------------------------
# EXAMPLE WITH CATEGORICAL CENSORING (3 levels)
#----------------------------------------------------------------------
require("data.table")
require("magrittr")
data(OdataCatCENS)
OdataDT <- as.data.table(OdataCatCENS, key=c(ID, t))
# Indicator that the person has never been treated in the past:
OdataDT[, "barTIm1eq0" := as.integer(c(0, cumsum(TI)[-.N]) %in% 0), by = ID]
# Define lagged N, first value is always 1 (always monitored at the first time point):
OdataDT[, ("N.tminus1") := shift(get("N"), n = 1L, type = "lag", fill = 1L), by = ID]

#----------------------------------------------------------------------
# Regressions for modeling the exposure (TRT)
#----------------------------------------------------------------------
gform_TRT <- "TI ~ CVD + highA1c + N.tminus1"
# Fit a separate model for TRT (stratify) for each of the following subsets:
stratify_TRT <- list(
```

```
  TI=c(
       # MODEL TI AT t=0
       "t == 0L",
       # MODEL TRT INITATION WHEN MONITORED
       "(t > 0L) & (N.tminus1 == 1L) & (barTIm1eq0 == 1L)",
       # MODEL TRT INITATION WHEN NOT MONITORED
       "(t > 0L) & (N.tminus1 == 0L) & (barTIm1eq0 == 1L)",
       # MODEL TRT CONTINUATION (BOTH MONITORED AND NOT MONITORED)
       "(t > 0L) & (barTIm1eq0 == 1L)"
      ))

#-----------------------------------------------------------------------
# Regressions for modeling the categorical censoring (CENS)
#-----------------------------------------------------------------------
gform_CENS <- c("CatC ~ highA1c")
# stratify by time-points (separate model for all t<16 and t=16)
stratify_CENS <- list(CatC=c("t < 16", "t == 16"))

#-----------------------------------------------------------------------
# Regressions for modeling the monitoring regimen (MONITOR)
#-----------------------------------------------------------------------
# Intercept only model, pooling across all time points t
gform_MONITOR <- "N ~ 1"

#-----------------------------------------------------------------------
# Define the counterfactual monitoring regimen of interest
#-----------------------------------------------------------------------
# probability of being monitored at each t is 0.1
OdataDT[, "gstar.N" := 0.1]

# Define two dynamic rules: dlow & dhigh
OdataDT <- defineIntervedTRT(OdataDT, theta = c(0,1), ID = "ID", t = "t", I = "highA1c",
                          CENS = "C", TRT = "TI", MONITOR = "N", tsinceNis1 = "lastNat1",
                           new.TRT.names = c("dlow", "dhigh"), return.allcolumns = TRUE)

# Estimate IPW-based hazard and survival (KM) for a rule "dhigh":
IPW_KM_res <- stremr(OdataDT, intervened_TRT = "dhigh", intervened_MONITOR = "gstar.N",
              ID = "ID", t = "t", covars = c("highA1c", "lastNat1"),
              CENS = "CatC", gform_CENS = gform_CENS, stratify_CENS = stratify_CENS,
              TRT = "TI", gform_TRT = gform_TRT, stratify_TRT = stratify_TRT,
              MONITOR = "N", gform_MONITOR = gform_MONITOR, OUTCOME = "Y.tplus1")

# Survival estimates by time:
IPW_KM_res$IPW_estimates
# Input data:
IPW_KM_res$dataDT
```

---

stremrOptions *Querying/setting a single* stremr *option*

Description

To list all `stremr` options, just run this function without any parameters provided. To query only one value, pass the first parameter. To set that, use the `value` parameter too.

Usage

```
stremrOptions(o, value)
```

Arguments

`o`	Option name (string). See `set_all_stremr_options`.
`value`	Value to assign (optional)

Details

The arguments of `set_all_stremr_options` list all available `stremr` options.

See Also

`set_all_stremr_options`

Examples

```
## Not run:
stremrOptions()
stremrOptions('fit.package')
stremrOptions('fit.package', 'h2o')

## End(Not run)
```

`summary.GLMmodel` *S3 methods for getting model fit summary for glmfit class object*

Description

Prints the modeling summary for the GLM fit (`stats::glm.fit` or `speedglm::speedglm.wfit`)

Usage

```
## S3 method for class 'GLMmodel'
summary(object, format_table = TRUE, ...)
```

Arguments

`object`	The model fit object produced by functions stremr:::glmfit.glm or stremr:::glmfit.speedglm
`format_table`	Format the coefficients into a data.frame table?
`...`	Additional options (not used)

Value

The markdown-formated model summary returned by `pander::pander_return`.

`summary.H2Oensemblemodel` *S3 methods for getting model fit summary for H2Oensemblemodel class object*

Description

Prints the modeling summary for the h2o model fit (see `h2o` R package).

Usage

```
## S3 method for class 'H2Oensemblemodel'
summary(object, only.coefs = FALSE,
  format_table = TRUE, ...)
```

Arguments

`object`	The model fit object produced by any stremr S3 function based on h2o
`only.coefs`	Skip all of the h2o-specific model stats, only print the coefficients table (when running `fit.algorithm = "glm"`).
`format_table`	Format the coefficients into a data.frame table (when running `fit.algorithm = "glm"`)?
`...`	Additional options (not used)

Value

The markdown-formated model summary returned by `pander::pander_return`.

`summary.H2Omodel` *S3 methods for getting model fit summary for H2Omodel class object*

Description

Prints the modeling summary for the h2o model fit (see `h2o` R package).

Usage

```
## S3 method for class 'H2Omodel'
summary(object, only.coefs = FALSE, format_table = TRUE,
  ...)
```

Arguments

object	The model fit object produced by any stremr S3 function based on h2o
only.coefs	Skip all of the h2o-specific model stats, only print the coefficients table (when running fit.algorithm = "glm").
format_table	Format the coefficients into a data.frame table (when running fit.algorithm = "glm")?
...	Additional options (not used)

Value

The markdown-formated model summary returned by pander::pander_return.

---

survDirectIPW *Direct (bounded) IPW estimator of discrete survival function.*

Description

Direct (bounded) IPW estimator of discrete survival function.

Usage

```
survDirectIPW(wts_data, OData, weights, trunc_weights)
```

Arguments

wts_data	data.table returned by a single call to getIPWeights. Must contain the treatment/monitoring estimated IPTW weights for a SINGLE rule.
OData	The object returned by function fitPropensity. Contains the input data and the previously fitted propensity score models for the exposure, monitoring and right-censoring.
weights	(NOT IMPLEMENTED) Optional data.table with additional observation-time-specific weights. Must contain columns ID, t and weight. The column named weight is merged back into the original data according to (ID, t).
trunc_weights	(NOT IMPLEMENTED) Specify the numeric weight truncation value. All final weights exceeding the value in trunc_weights will be truncated.

Value

A data.table with bounded IPW estimates of risk and survival by time.

Examples

```
options(stremr.verbose = TRUE)
require("data.table")
set_all_stremr_options(fit.package = "speedglm", fit.algorithm = "glm")

# ----------------------------------------------------------------------
# Simulated Data
# ----------------------------------------------------------------------
data(OdataNoCENS)
OdataDT <- as.data.table(OdataNoCENS, key=c(ID, t))

# define lagged N, first value is always 1 (always monitored at the first time point):
OdataDT[, ("N.tminus1") := shift(get("N"), n = 1L, type = "lag", fill = 1L), by = ID]
OdataDT[, ("TI.tminus1") := shift(get("TI"), n = 1L, type = "lag", fill = 1L), by = ID]

# ----------------------------------------------------------------------
# Define intervention (always treated):
# ----------------------------------------------------------------------
OdataDT[, ("TI.set1") := 1L]
OdataDT[, ("TI.set0") := 0L]

# ----------------------------------------------------------------------
# Import Data
# ----------------------------------------------------------------------
OData <- importData(OdataDT, ID = "ID", t = "t", covars = c("highA1c", "lastNat1", "N.tminus1"),
                    CENS = "C", TRT = "TI", MONITOR = "N", OUTCOME = "Y.tplus1")

# ----------------------------------------------------------------------
# Model the Propensity Scores
# ----------------------------------------------------------------------
gform_CENS <- "C ~ highA1c + lastNat1"
gform_TRT = "TI ~ CVD + highA1c + N.tminus1"
gform_MONITOR <- "N ~ 1"
stratify_CENS <- list(C=c("t < 16", "t == 16"))

# ----------------------------------------------------------------------
# Fit Propensity Scores
# ----------------------------------------------------------------------
OData <- fitPropensity(OData, gform_CENS = gform_CENS,
                       gform_TRT = gform_TRT,
                       gform_MONITOR = gform_MONITOR,
                       stratify_CENS = stratify_CENS)

# ----------------------------------------------------------------------
# IPW Ajusted KM or Saturated MSM
# ----------------------------------------------------------------------
require("magrittr")
AKME.St.1 <- getIPWeights(OData, intervened_TRT = "TI.set1") %>%
             survNPMSM(OData) %$%
             IPW_estimates
AKME.St.1
```

```
# ------------------------------------------------------------------------
# Bounded IPW
# ------------------------------------------------------------------------
IPW.St.1 <- getIPWeights(OData, intervened_TRT = "TI.set1") %>%
             survDirectIPW(OData)
IPW.St.1[]

# ------------------------------------------------------------------------
# IPW-MSM for hazard
# ------------------------------------------------------------------------
wts.DT.1 <- getIPWeights(OData = OData, intervened_TRT = "TI.set1", rule_name = "TI1")
wts.DT.0 <- getIPWeights(OData = OData, intervened_TRT = "TI.set0", rule_name = "TI0")
survMSM_res <- survMSM(list(wts.DT.1, wts.DT.0), OData, t_breaks = c(1:8,12,16)-1,)
survMSM_res$St

# ------------------------------------------------------------------------
# Sequential G-COMP
# ------------------------------------------------------------------------
t.surv <- c(0:15)
Qforms <- rep.int("Q.kplus1 ~ CVD + highA1c + N + lastNat1 + TI + TI.tminus1", (max(t.surv)+1))
params = list(fit.package = "speedglm", fit.algorithm = "glm")

## Not run:
gcomp_est <- fitSeqGcomp(OData, t_periods = t.surv, intervened_TRT = "TI.set1",
                         Qforms = Qforms, params_Q = params, stratifyQ_by_rule = FALSE)
gcomp_est[]

## End(Not run)
# ------------------------------------------------------------------------
# TMLE
# ------------------------------------------------------------------------
## Not run:
tmle_est <- fitTMLE(OData, t_periods = t.surv, intervened_TRT = "TI.set1",
                    Qforms = Qforms, params_Q = params, stratifyQ_by_rule = TRUE)
tmle_est[]

## End(Not run)

# ------------------------------------------------------------------------
# Running IPW-Adjusted KM with optional user-specified weights:
# ------------------------------------------------------------------------
addedWts_DT <- OdataDT[, c("ID", "t"), with = FALSE]
addedWts_DT[, new.wts := sample.int(10, nrow(OdataDT), replace = TRUE)/10]
survNP_res_addedWts <- survNPMSM(wts.DT.1, OData, weights = addedWts_DT)

# ------------------------------------------------------------------------
# Multivariate Propensity Score Regressions
# ------------------------------------------------------------------------
gform_CENS <- "C + TI + N ~ highA1c + lastNat1"
OData <- fitPropensity(OData, gform_CENS = gform_CENS, gform_TRT = gform_TRT,
                       gform_MONITOR = gform_MONITOR)

# ------------------------------------------------------------------------
```

```
# Fitting Propensity scores with Random Forests:
# ------------------------------------------------------------------------
## Not run:
set_all_stremr_options(fit.package = "h2o", fit.algorithm = "randomForest")
require("h2o")
h2o::h2o.init(nthreads = -1)
gform_CENS <- "C ~ highA1c + lastNat1"
OData <- fitPropensity(OData, gform_CENS = gform_CENS,
                       gform_TRT = gform_TRT,
                       gform_MONITOR = gform_MONITOR,
                       stratify_CENS = stratify_CENS)

# For Gradient Boosting machines:
set_all_stremr_options(fit.package = "h2o", fit.algorithm = "gbm")
# Use `H2O-3` distributed implementation of GLM
set_all_stremr_options(fit.package = "h2o", fit.algorithm = "glm")
# Use Deep Neural Nets:
set_all_stremr_options(fit.package = "h2o", fit.algorithm = "deeplearning")

## End(Not run)

# ------------------------------------------------------------------------
# Fitting different models with different algorithms
# Fine tuning modeling with optional tuning parameters.
# ------------------------------------------------------------------------
## Not run:
params_TRT = list(fit.package = "h2o", fit.algorithm = "gbm", ntrees = 50,
    learn_rate = 0.05, sample_rate = 0.8, col_sample_rate = 0.8,
    balance_classes = TRUE)
params_CENS = list(fit.package = "speedglm", fit.algorithm = "glm")
params_MONITOR = list(fit.package = "speedglm", fit.algorithm = "glm")
OData <- fitPropensity(OData,
          gform_CENS = gform_CENS, stratify_CENS = stratify_CENS, params_CENS = params_CENS,
            gform_TRT = gform_TRT, params_TRT = params_TRT,
            gform_MONITOR = gform_MONITOR, params_MONITOR = params_MONITOR)

## End(Not run)

# ------------------------------------------------------------------------
# Running TMLE based on the previous fit of the propensity scores.
# Also applying Random Forest to estimate the sequential outcome model
# ------------------------------------------------------------------------
## Not run:
t.surv <- c(0:5)
Qforms <- rep.int("Q.kplus1 ~ CVD + highA1c + N + lastNat1 + TI + TI.tminus1", (max(t.surv)+1))
params_Q = list(fit.package = "h2o", fit.algorithm = "randomForest",
                ntrees = 100, learn_rate = 0.05, sample_rate = 0.8,
                col_sample_rate = 0.8, balance_classes = TRUE)
tmle_est <- fitTMLE(OData, t_periods = t.surv, intervened_TRT = "TI.set1",
            Qforms = Qforms, params_Q = params_Q,
            stratifyQ_by_rule = TRUE)

## End(Not run)
```

```
## Not run:
t.surv <- c(0:5)
Qforms <- rep.int("Q.kplus1 ~ CVD + highA1c + N + lastNat1 + TI + TI.tminus1", (max(t.surv)+1))
params_Q = list(fit.package = "h2o", fit.algorithm = "randomForest",
                ntrees = 100, learn_rate = 0.05, sample_rate = 0.8,
                col_sample_rate = 0.8, balance_classes = TRUE)
tmle_est <- fitTMLE(OData, t_periods = t.surv, intervened_TRT = "TI.set1",
            Qforms = Qforms, params_Q = params_Q,
            stratifyQ_by_rule = FALSE)

## End(Not run)

# ----------------------------------------------------------------------
# Ensemble Learning with SuperLearner (using h2oEnsemble R package):
# ----------------------------------------------------------------------
## Not run:
require('h2oEnsemble')
# ----------------------------------------------------------------------
# Define many learners at once via grid search over tuning parameter spaces (hyperparameters)
# ----------------------------------------------------------------------
# 1. Runs h2o.grid in the background for each individual learner and saves cross-validated risks.
# 2. Calls h2o.stack from h2oEnsemble package to evaluate the final SuperLearner.
# ----------------------------------------------------------------------
# Search criteria and grid search space for glm tuning parameters:
GLM_hyper_params <- list(search_criteria = list(strategy = "RandomDiscrete", max_models = 5),
                         alpha = c(0,1,seq(0.1,0.9,0.1)),
                         lambda = c(0,1e-7,1e-5,1e-3,1e-1))

# Search criteria and grid search space for Random Forests:
search_criteria <- list(strategy = "RandomDiscrete", max_models = 5, max_runtime_secs = 60*60)
RF_hyper_params <- list(search_criteria = search_criteria,
                        ntrees = c(100, 200, 300, 500),
                        mtries = 1:4,
                        max_depth = c(5, 10, 15, 20, 25),
                        sample_rate = c(0.7, 0.8, 0.9, 1.0),
                        col_sample_rate_per_tree = c(0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8),
                        balance_classes = c(TRUE, FALSE))

# Search criteria and grid search space for Gradient Boosting Machines (gbm):
GBM_hyper_params <- list(search_criteria = search_criteria,
                         ntrees = c(100, 200, 300, 500),
                         learn_rate = c(0.005, 0.01, 0.03, 0.06),
                         max_depth = c(3, 4, 5, 6, 9),
                         sample_rate = c(0.7, 0.8, 0.9, 1.0),
                         col_sample_rate = c(0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8),
                         balance_classes = c(TRUE, FALSE))

# Specify individual learners to include in the SuperLearner library (in addition to grid models):
h2o.glm.1 <- function(..., alpha = 0.0) h2o.glm.wrapper(..., alpha = alpha)
h2o.glm.2 <- function(..., x = "highA1c", alpha = 0.0) h2o.glm.wrapper(..., x = x, alpha = alpha)
h2o.glm.3 <- function(..., alpha = 1.0) h2o.glm.wrapper(..., alpha = alpha)
```

```
# ------------------------------------------------------------------------
# Put all different algorithms together to define the final model ensemble:
# ------------------------------------------------------------------------
SLparams = list(fit.package = "h2o", fit.algorithm = "SuperLearner",
                grid.algorithm = c("glm", "randomForest", "gbm"),
                learner = c("h2o.glm.1", "h2o.glm.2", "h2o.glm.3"),
                metalearner = "h2o.glm_nn",
                nfolds = 10,
                seed = 23,
                glm = GLM_hyper_params,
                randomForest = RF_hyper_params,
                gbm = GBM_hyper_params)

# ------------------------------------------------------------------------
# Use save.ensemble and ensemble.dir.path arguments to save the final SuperLearner fit
# ------------------------------------------------------------------------
# This will also save the individual learner fits in the same directory.
# Only one directory per single SuperLearner fit is allowed.
# Can be loaded later on to save time and avoid refitting SuperLearner (load.ensemble = TRUE).

params_TRT = c(SLparams, save.ensemble = TRUE, ensemble.dir.path = "./h2o-ensemble-model-TRT")
params_CENS = list(fit.package = "speedglm", fit.algorithm = "glm")
params_MONITOR = list(fit.package = "speedglm", fit.algorithm = "glm")

OData <- fitPropensity(OData,
          gform_CENS = gform_CENS, stratify_CENS = stratify_CENS, params_CENS = params_CENS,
            gform_TRT = gform_TRT, params_TRT = params_TRT,
            gform_MONITOR = gform_MONITOR, params_MONITOR = params_MONITOR)

# ------------------------------------------------------------------------
# Re-using previously saved SuperLearner fit
# ------------------------------------------------------------------------
params_TRT = c(SLparams, load.ensemble = TRUE, ensemble.dir.path = "./h2o-ensemble-model-TRT")
params_CENS = list(fit.package = "speedglm", fit.algorithm = "glm")
params_MONITOR = list(fit.package = "speedglm", fit.algorithm = "glm")

OData <- fitPropensity(OData,
          gform_CENS = gform_CENS, stratify_CENS = stratify_CENS, params_CENS = params_CENS,
            gform_TRT = gform_TRT, params_TRT = params_TRT,
            gform_MONITOR = gform_MONITOR, params_MONITOR = params_MONITOR)

## End(Not run)
```

---

survMSM *Estimate Survival with a particular MSM for the survival-hazard function using previously fitted weights.*

Description

Estimate the causal survival curve for a particular stochastic, dynamic or static intervention on the treatment/exposure and monitoring processes based on the user-specified Marginal Structural

Model (MSM) for the counterfactual survival function.

Usage

```
survMSM(wts_data, OData, t_breaks, use_weights = TRUE, stabilize = TRUE,
  trunc_weights = 10^6, weights = NULL, getSEs = TRUE, est_name = "IPW",
  verbose = getOption("stremr.verbose"))
```

Arguments

wts_data	A list of data.tables, each data set is a result of calling the function getIPWeights. Must contain the treatment/monitoring rule-specific estimated IPTW weights. This argument can be also a single data.table obtained with data.table::rbindlist(wts_data).
OData	The object returned by function fitPropensity. Contains the input dat and the previously fitted propensity score models for the exposure, monitoring and right-censoring.
t_breaks	The vector of integer (or numeric) breaks that defines the dummy indicators of the follow-up in the observed data. Used for fitting the parametric (or saturated) MSM for the survival hazard. See Details.
use_weights	Logical value. Set to FALSE to ignore the weights in wts_data and fit a "crude" MSM that does not adjust for the possible confounding due to non-random assignment of the exposure/censoring and monitoring indicators.
stabilize	Set to TRUE for weight stabilization
trunc_weights	Specify the numeric weight truncation value. All final weights exceeding the value in trunc_weights will be truncated.
weights	Optional data.table with additional observation-time-specific weights. Must contain columns ID, t and weight. The column named weight is merged back into the original data according to (ID, t).
getSEs	A logical indicator. Set to TRUE to evaluate the standard errors for the estimated survival by using the MSM influence curve.
est_name	A string naming the current MSM estimator. Ignored by the current routine but is used when generating reports with make_report_rmd.
verbose	Set to TRUE to print messages on status and information to the console. Turn this on by default using options(stremr.verbose=TRUE).

Details

This routine will run the weighted logistic regression using the (possibly-weighted) outcomes from many regimens, with dummy indicators for each treatment/monitoring regimen available in wts_data and each follow-up time interval specified in t_breaks. When use_weights = TRUE, the logistic regression for the survival hazard is weighted by the **IPW** (Inverse Probability-Weighted or Horvitz-Thompson) estimated weights in wts_data. These IPW weights are based on the previously fitted propensity scores (function fitPropensity), allowing adjustment for confounding by possibly non-random assignment to exposure and monitoring and possibly informative right-censoring.

Value

A named list with items containing the MSM estimation results:

- est_name - .
- St - .
- ht - .
- MSM.fit - .
- MSM.intervals - .
- IC.Var.S.d - .
- nID - .
- wts_data - .
- use_weights - .
- trunc_weights - .

Details

****

t_breaks is used for defining the time-intervals of the MSM coefficients for estimation of the survival hazard function. The first value in t_breaks defines a dummy variable (indicator) for a fully closed interval, with each subsequent value in t_breaks defining a single right-closed time-interval. For example, t_breaks = c(0,1) will define the MSM dummy indicators: I(min(t) <= t <=0), I(0 < t <= 1) and I(1 < t <= max(t)). On the other hand t_breaks = c(1) will define the following (more parametric) MSM dummy indicators: I(min(t) <= t <=1) and I(1 < t <= max(t)). If omitted, the default is to define a saturated (non-parametric) MSM with a separate dummy variable for every unique period in the observed data.

MSM for the hazard

****

See Also

stremr-package for the general overview of the package,

Examples

```
#----------------------------------------------------------------------
# Simulated data with informative right-censoring
#----------------------------------------------------------------------
require("data.table")
require("magrittr")
data(OdataCatCENS)
OdataDT <- as.data.table(OdataCatCENS, key=c(ID, t))
# Indicator that the person has never been treated in the past:
OdataDT[, "barTIm1eq0" := as.integer(c(0, cumsum(TI)[-.N]) %in% 0), by = ID]
OdataDT[, ("N.tminus1") := shift(get("N"), n = 1L, type = "lag", fill = 1L), by = ID]
```

```
#------------------------------------------------------------------
# Regressions for modeling the exposure (TRT)
#------------------------------------------------------------------
gform_TRT <- "TI ~ CVD + highA1c + N.tminus1"
# Fit a separate model for TRT (stratify) for each of the following subsets:
stratify_TRT <- list(
  TI=c(
       # MODEL TI AT t=0
       "t == 0L",
       # MODEL TRT INITATION WHEN MONITORED
       "(t > 0L) & (N.tminus1 == 1L) & (barTIm1eq0 == 1L)",
       # MODEL TRT INITATION WHEN NOT MONITORED
       "(t > 0L) & (N.tminus1 == 0L) & (barTIm1eq0 == 1L)",
       # MODEL TRT CONTINUATION (BOTH MONITORED AND NOT MONITORED)
       "(t > 0L) & (barTIm1eq0 == 1L)"
      ))

#------------------------------------------------------------------
# Regressions for modeling the categorical censoring (CENS)
#------------------------------------------------------------------
gform_CENS <- c("CatC ~ highA1c")
# stratify by time-points (separate model for all t<16 and t=16)
stratify_CENS <- list(CatC=c("t < 16", "t == 16"))

#------------------------------------------------------------------
# Regressions for modeling the monitoring regimen (MONITOR)
#------------------------------------------------------------------
# Intercept only model, pooling across all time points t
gform_MONITOR <- "N ~ 1"

#------------------------------------------------------------------
# Define the counterfactual monitoring regimen of interest
#------------------------------------------------------------------
# probability of being monitored at each t is 0.1
OdataDT[, "gstar.N" := 0.1]

# Define two dynamic rules: dlow & dhigh
OdataDT <- defineIntervedTRT(OdataDT, theta = c(0,1), ID = "ID", t = "t", I = "highA1c",
                          CENS = "C", TRT = "TI", MONITOR = "N", tsinceNis1 = "lastNat1",
                           new.TRT.names = c("dlow", "dhigh"), return.allcolumns = TRUE)

#------------------------------------------------------------------
# Import data into stremr object
#------------------------------------------------------------------
OData <- importData(OdataDT, ID = "ID", t = "t", covars = c("highA1c", "lastNat1"),
                    CENS = "CatC", TRT = "TI", MONITOR = "N", OUTCOME = "Y.tplus1")

#------------------------------------------------------------------
# Estimate Propensity scores
#------------------------------------------------------------------
OData <- fitPropensity(OData, gform_CENS = gform_CENS, gform_TRT = gform_TRT,
                       stratify_TRT = stratify_TRT, gform_MONITOR = gform_MONITOR)
```

```
#-----------------------------------------------------------------
# Defining weights for two dynamic regimens "dlow" and "dhigh"
#-----------------------------------------------------------------
wts.St.dlow <- getIPWeights(OData, intervened_TRT = "dlow")
wts.St.dhigh <- getIPWeights(OData, intervened_TRT = "dhigh")

# -----------------------------------------------------------------
# Estimate survival with IPW-adjusted MSM for the hazard (logistic model)
# 1. Saturated model for time points 0 to 7
# 2. Smoothing with one hazard coefficient over time-points 8 to 11
# 3. Smoothing with one hazard coefficient over time-points 12 to 15
# -----------------------------------------------------------------
IPW_MSM_res <- survMSM(OData, wts_data = list(dlow = wts.St.dlow, dhigh = wts.St.dhigh),
                       t_breaks = c(1:8,12,16)-1,
                       est_name = "IPAW", getSEs = TRUE)
names(IPW_MSM_res)
# Survival estimates over time
IPW_MSM_res$St
# SE estimates for each time point:
IPW_MSM_res$IC.Var.S.d$dhigh$se.S
IPW_MSM_res$IC.Var.S.d$dlow$se.S
# MSM coefficient fits
IPW_MSM_res$MSM.fit

# -----------------------------------------------------------------
# Generate automatic html report with results of the analysis
# This assumes that pandoc is already installed
# For more information, go to: http://pandoc.org/installing.html
# -----------------------------------------------------------------
## Not run:
make_report_rmd(OData, MSM = IPW_MSM_res,
  AddFUPtables = TRUE,
  RDtables = get_MSM_RDs(IPW_MSM_res, t.periods.RDs = c(12, 15), getSEs = FALSE),
  WTtables = get_wtsummary(IPW_MSM_res$wts_data,
    cutoffs = c(0, 0.5, 1, 10, 20, 30, 40, 50, 100, 150), by.rule = TRUE),
  file.name = "sim.data.example.fup",
  title = "Custom Report Title",
  author = "Jane Doe",
  y_legend = 0.95)

## End(Not run)
```

`survNPMSM` *Non-parametric (saturated) MSM for survival based on previously evaluated IP weights.*

Description

Non-parametric (saturated) MSM for survival based on previously evaluated IP weights.

Usage

```
survNPMSM(wts_data, OData, weights = NULL, trunc_weights = 10^6)
```

Arguments

wts_data	data.table returned by a single call to getIPWeights. Must contain the treatment/monitoring estimated IPTW weights for a SINGLE rule.
OData	The object returned by function fitPropensity. Contains the input data and the previously fitted propensity score models for the exposure, monitoring and right-censoring.
weights	Optional data.table with additional observation-time-specific weights. Must contain columns ID, t and weight. The column named weight is merged back into the original data according to (ID, t).
trunc_weights	Specify the numeric weight truncation value. All final weights exceeding the value in trunc_weights will be truncated.

Value

A data.table with hazard and survival function estimates by time. Also include the unadjusted Kaplan-Maier estimates.

Examples

```
options(stremr.verbose = TRUE)
require("data.table")
set_all_stremr_options(fit.package = "speedglm", fit.algorithm = "glm")

# ----------------------------------------------------------------------
# Simulated Data
# ----------------------------------------------------------------------
data(OdataNoCENS)
OdataDT <- as.data.table(OdataNoCENS, key=c(ID, t))

# define lagged N, first value is always 1 (always monitored at the first time point):
OdataDT[, ("N.tminus1") := shift(get("N"), n = 1L, type = "lag", fill = 1L), by = ID]
OdataDT[, ("TI.tminus1") := shift(get("TI"), n = 1L, type = "lag", fill = 1L), by = ID]

# ----------------------------------------------------------------------
# Define intervention (always treated):
# ----------------------------------------------------------------------
OdataDT[, ("TI.set1") := 1L]
OdataDT[, ("TI.set0") := 0L]

# ----------------------------------------------------------------------
# Import Data
# ----------------------------------------------------------------------
OData <- importData(OdataDT, ID = "ID", t = "t", covars = c("highA1c", "lastNat1", "N.tminus1"),
                    CENS = "C", TRT = "TI", MONITOR = "N", OUTCOME = "Y.tplus1")

# ----------------------------------------------------------------------
```

```
# Model the Propensity Scores
# ------------------------------------------------------------------------
gform_CENS <- "C ~ highA1c + lastNat1"
gform_TRT = "TI ~ CVD + highA1c + N.tminus1"
gform_MONITOR <- "N ~ 1"
stratify_CENS <- list(C=c("t < 16", "t == 16"))

# ------------------------------------------------------------------------
# Fit Propensity Scores
# ------------------------------------------------------------------------
OData <- fitPropensity(OData, gform_CENS = gform_CENS,
                       gform_TRT = gform_TRT,
                       gform_MONITOR = gform_MONITOR,
                       stratify_CENS = stratify_CENS)

# ------------------------------------------------------------------------
# IPW Ajusted KM or Saturated MSM
# ------------------------------------------------------------------------
require("magrittr")
AKME.St.1 <- getIPWeights(OData, intervened_TRT = "TI.set1") %>%
             survNPMSM(OData) %$%
             IPW_estimates
AKME.St.1

# ------------------------------------------------------------------------
# Bounded IPW
# ------------------------------------------------------------------------
IPW.St.1 <- getIPWeights(OData, intervened_TRT = "TI.set1") %>%
             survDirectIPW(OData)
IPW.St.1[]

# ------------------------------------------------------------------------
# IPW-MSM for hazard
# ------------------------------------------------------------------------
wts.DT.1 <- getIPWeights(OData = OData, intervened_TRT = "TI.set1", rule_name = "TI1")
wts.DT.0 <- getIPWeights(OData = OData, intervened_TRT = "TI.set0", rule_name = "TI0")
survMSM_res <- survMSM(list(wts.DT.1, wts.DT.0), OData, t_breaks = c(1:8,12,16)-1,)
survMSM_res$St

# ------------------------------------------------------------------------
# Sequential G-COMP
# ------------------------------------------------------------------------
t.surv <- c(0:15)
Qforms <- rep.int("Q.kplus1 ~ CVD + highA1c + N + lastNat1 + TI + TI.tminus1", (max(t.surv)+1))
params = list(fit.package = "speedglm", fit.algorithm = "glm")

## Not run:
gcomp_est <- fitSeqGcomp(OData, t_periods = t.surv, intervened_TRT = "TI.set1",
                         Qforms = Qforms, params_Q = params, stratifyQ_by_rule = FALSE)
gcomp_est[]

## End(Not run)
# ------------------------------------------------------------------------
```

```
# TMLE
# ------------------------------------------------------------------------
## Not run:
tmle_est <- fitTMLE(OData, t_periods = t.surv, intervened_TRT = "TI.set1",
                    Qforms = Qforms, params_Q = params, stratifyQ_by_rule = TRUE)
tmle_est[]

## End(Not run)

# ------------------------------------------------------------------------
# Running IPW-Adjusted KM with optional user-specified weights:
# ------------------------------------------------------------------------
addedWts_DT <- OdataDT[, c("ID", "t"), with = FALSE]
addedWts_DT[, new.wts := sample.int(10, nrow(OdataDT), replace = TRUE)/10]
survNP_res_addedWts <- survNPMSM(wts.DT.1, OData, weights = addedWts_DT)

# ------------------------------------------------------------------------
# Multivariate Propensity Score Regressions
# ------------------------------------------------------------------------
gform_CENS <- "C + TI + N ~ highA1c + lastNat1"
OData <- fitPropensity(OData, gform_CENS = gform_CENS, gform_TRT = gform_TRT,
                       gform_MONITOR = gform_MONITOR)

# ------------------------------------------------------------------------
# Fitting Propensity scores with Random Forests:
# ------------------------------------------------------------------------
## Not run:
set_all_stremr_options(fit.package = "h2o", fit.algorithm = "randomForest")
require("h2o")
h2o::h2o.init(nthreads = -1)
gform_CENS <- "C ~ highA1c + lastNat1"
OData <- fitPropensity(OData, gform_CENS = gform_CENS,
                       gform_TRT = gform_TRT,
                       gform_MONITOR = gform_MONITOR,
                       stratify_CENS = stratify_CENS)

# For Gradient Boosting machines:
set_all_stremr_options(fit.package = "h2o", fit.algorithm = "gbm")
# Use `H2O-3` distributed implementation of GLM
set_all_stremr_options(fit.package = "h2o", fit.algorithm = "glm")
# Use Deep Neural Nets:
set_all_stremr_options(fit.package = "h2o", fit.algorithm = "deeplearning")

## End(Not run)

# ------------------------------------------------------------------------
# Fitting different models with different algorithms
# Fine tuning modeling with optional tuning parameters.
# ------------------------------------------------------------------------
## Not run:
params_TRT = list(fit.package = "h2o", fit.algorithm = "gbm", ntrees = 50,
    learn_rate = 0.05, sample_rate = 0.8, col_sample_rate = 0.8,
    balance_classes = TRUE)
```

```
params_CENS = list(fit.package = "speedglm", fit.algorithm = "glm")
params_MONITOR = list(fit.package = "speedglm", fit.algorithm = "glm")
OData <- fitPropensity(OData,
          gform_CENS = gform_CENS, stratify_CENS = stratify_CENS, params_CENS = params_CENS,
            gform_TRT = gform_TRT, params_TRT = params_TRT,
            gform_MONITOR = gform_MONITOR, params_MONITOR = params_MONITOR)

## End(Not run)

# ----------------------------------------------------------------------
# Running TMLE based on the previous fit of the propensity scores.
# Also applying Random Forest to estimate the sequential outcome model
# ----------------------------------------------------------------------
## Not run:
t.surv <- c(0:5)
Qforms <- rep.int("Q.kplus1 ~ CVD + highA1c + N + lastNat1 + TI + TI.tminus1", (max(t.surv)+1))
params_Q = list(fit.package = "h2o", fit.algorithm = "randomForest",
                ntrees = 100, learn_rate = 0.05, sample_rate = 0.8,
                col_sample_rate = 0.8, balance_classes = TRUE)
tmle_est <- fitTMLE(OData, t_periods = t.surv, intervened_TRT = "TI.set1",
            Qforms = Qforms, params_Q = params_Q,
            stratifyQ_by_rule = TRUE)

## End(Not run)

## Not run:
t.surv <- c(0:5)
Qforms <- rep.int("Q.kplus1 ~ CVD + highA1c + N + lastNat1 + TI + TI.tminus1", (max(t.surv)+1))
params_Q = list(fit.package = "h2o", fit.algorithm = "randomForest",
                ntrees = 100, learn_rate = 0.05, sample_rate = 0.8,
                col_sample_rate = 0.8, balance_classes = TRUE)
tmle_est <- fitTMLE(OData, t_periods = t.surv, intervened_TRT = "TI.set1",
            Qforms = Qforms, params_Q = params_Q,
            stratifyQ_by_rule = FALSE)

## End(Not run)

# ----------------------------------------------------------------------
# Ensemble Learning with SuperLearner (using h2oEnsemble R package):
# ----------------------------------------------------------------------
## Not run:
require('h2oEnsemble')
# ----------------------------------------------------------------------
# Define many learners at once via grid search over tuning parameter spaces (hyperparameters)
# ----------------------------------------------------------------------
# 1. Runs h2o.grid in the background for each individual learner and saves cross-validated risks.
# 2. Calls h2o.stack from h2oEnsemble package to evaluate the final SuperLearner.
# ----------------------------------------------------------------------
# Search criteria and grid search space for glm tuning parameters:
GLM_hyper_params <- list(search_criteria = list(strategy = "RandomDiscrete", max_models = 5),
                         alpha = c(0,1,seq(0.1,0.9,0.1)),
                         lambda = c(0,1e-7,1e-5,1e-3,1e-1))
```

```
# Search criteria and grid search space for Random Forests:
search_criteria <- list(strategy = "RandomDiscrete", max_models = 5, max_runtime_secs = 60*60)
RF_hyper_params <- list(search_criteria = search_criteria,
                        ntrees = c(100, 200, 300, 500),
                        mtries = 1:4,
                        max_depth = c(5, 10, 15, 20, 25),
                        sample_rate = c(0.7, 0.8, 0.9, 1.0),
                        col_sample_rate_per_tree = c(0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8),
                        balance_classes = c(TRUE, FALSE))

# Search criteria and grid search space for Gradient Boosting Machines (gbm):
GBM_hyper_params <- list(search_criteria = search_criteria,
                         ntrees = c(100, 200, 300, 500),
                         learn_rate = c(0.005, 0.01, 0.03, 0.06),
                         max_depth = c(3, 4, 5, 6, 9),
                         sample_rate = c(0.7, 0.8, 0.9, 1.0),
                         col_sample_rate = c(0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8),
                         balance_classes = c(TRUE, FALSE))

# Specify individual learners to include in the SuperLearner library (in addition to grid models):
h2o.glm.1 <- function(..., alpha = 0.0) h2o.glm.wrapper(..., alpha = alpha)
h2o.glm.2 <- function(..., x = "highA1c", alpha = 0.0) h2o.glm.wrapper(..., x = x, alpha = alpha)
h2o.glm.3 <- function(..., alpha = 1.0) h2o.glm.wrapper(..., alpha = alpha)

# ------------------------------------------------------------------------
# Put all different algorithms together to define the final model ensemble:
# ------------------------------------------------------------------------
SLparams = list(fit.package = "h2o", fit.algorithm = "SuperLearner",
                grid.algorithm = c("glm", "randomForest", "gbm"),
                learner = c("h2o.glm.1", "h2o.glm.2", "h2o.glm.3"),
                metalearner = "h2o.glm_nn",
                nfolds = 10,
                seed = 23,
                glm = GLM_hyper_params,
                randomForest = RF_hyper_params,
                gbm = GBM_hyper_params)

# ------------------------------------------------------------------------
# Use save.ensemble and ensemble.dir.path arguments to save the final SuperLearner fit
# ------------------------------------------------------------------------
# This will also save the individual learner fits in the same directory.
# Only one directory per single SuperLearner fit is allowed.
# Can be loaded later on to save time and avoid refitting SuperLearner (load.ensemble = TRUE).

params_TRT = c(SLparams, save.ensemble = TRUE, ensemble.dir.path = "./h2o-ensemble-model-TRT")
params_CENS = list(fit.package = "speedglm", fit.algorithm = "glm")
params_MONITOR = list(fit.package = "speedglm", fit.algorithm = "glm")

OData <- fitPropensity(OData,
          gform_CENS = gform_CENS, stratify_CENS = stratify_CENS, params_CENS = params_CENS,
            gform_TRT = gform_TRT, params_TRT = params_TRT,
            gform_MONITOR = gform_MONITOR, params_MONITOR = params_MONITOR)
```

```
# ------------------------------------------------------------------------
# Re-using previously saved SuperLearner fit
# ------------------------------------------------------------------------
params_TRT = c(SLparams, load.ensemble = TRUE, ensemble.dir.path = "./h2o-ensemble-model-TRT")
params_CENS = list(fit.package = "speedglm", fit.algorithm = "glm")
params_MONITOR = list(fit.package = "speedglm", fit.algorithm = "glm")

OData <- fitPropensity(OData,
          gform_CENS = gform_CENS, stratify_CENS = stratify_CENS, params_CENS = params_CENS,
            gform_TRT = gform_TRT, params_TRT = params_TRT,
            gform_MONITOR = gform_MONITOR, params_MONITOR = params_MONITOR)

## End(Not run)
```

Index