Package ‘StratifiedMedicine’

August 3, 2020

Type Package

Title Stratified Medicine

Version 1.0.2

Author Thomas Jemielita [aut, cre]

Maintainer Thomas Jemielita <thomasjemielita@gmail.com>

Description A toolkit for stratified medicine, subgroup identification, and precision medicine. Current tools include (1) filtering models (reduce covariate space), (2) patient-level estimate models (counterfactual patient-level quantities, for example the individual treatment effect), (3) subgroup identification models (find subsets of patients with similar treatment effects), and (4) parameter estimation and inference (for the overall population and discovered subgroups). These tools can directly feed into stratified medicine algorithms including PRISM (patient response identifiers for stratified medicine; Jemielita and Mehrotra 2019 <arXiv:1912.03337>. PRISM is a flexible and general framework which accepts user-created models/functions. This package is in beta and will be continually updated.

License GPL-3

Encoding UTF-8

LazyData true4

Depends R (>= 3.4),

Imports dplyr, partykit, ranger, survival, glmnet, ggplot2, ggparty, mvtnorm

RoxygenNote 7.1.0

URL https://github.com/thomasjemielita/StratifiedMedicine

Suggests knitr, rmarkdown, MASS, BART, pROC, survRM2, TH.data, sandwich, coin, rpart, testthat (>= 2.1.0)

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2020-08-03 05:40:02 UTC

R topics documented:

`filter_train` *filter_train: Identify variables of interest*

Description

Wrapper function to train a filter model to determine variables associated with the outcome and/or treatment.. Options include elastic net (glmnet) and random forest based variable importance (ranger). Used directly in PRISM.

Usage

```
filter_train(
  Y,
  A,
  X,
  family = "gaussian",
  filter = "glmnet",
  hyper = NULL,
  ...
)
```

Arguments

`Y`	The outcome variable. Must be numeric or survival (ex; Surv(time,cens))
`A`	Treatment variable. (Default supports binary treatment, either numeric or factor). "ple_train" accomodates >2 along with binary treatments.
`X`	Covariate space.

`family`	Outcome type. Options include "gaussion" (default), "binomial", and "survival".
`filter`	Filter model to determine variables that are likely associated with the outcome and/or treatment. Outputs a potential reduce list of varia where X.star has potentially less variables than X. Default is "glmnet" (elastic net). Other options include "ranger" (random forest based variable importance with p-values). See `filter_train` for more details. "None" uses no filter.
`hyper`	Hyper-parameters for the filter model (must be list). Default is NULL. See details below.
`...`	Any additional parameters, not currently passed through.

Details

filter_train currently fits elastic net or random forest to find a reduced set of variables which are likely associated with the outcome (Y) and/or treatment (A). Current options include:

1. **glmnet**: Wrapper function for the function "glmnet" from the glmnet package. Here, variables with estimated elastic net coefficients of 0 are filtered. Uses LM/GLM/cox elastic net for family="gaussian","binomial", "survival" respectively. Default is to regress Y~ENET(X) with hyper-parameters:

hyper = list(lambda="lambda.min", family="gaussian",interaction=FALSE))

If interaction=TRUE, then Y~ENET(X,A,X*A), and variables with estimated coefficients of zero in both the main effects (X) and treatment-interactions (X*A) are filtered. This aims to find variables that are prognostic and/or predictive.

2. **ranger**: Wrapper function for the function "ranger" (ranger R package) to calculate random forest based variable importance (VI) p-values. Here, for the test of VI>0, variables are filtered if their one-sided p-value>=0.10. P-values are obtained through subsampling based T-statistics (T=VI_j/SE(VE_j)) for feature j through the delete-d jackknife), as described in Ishwaran and Lu 2017. Used for continuous, binary, or survival outcomes. Default hyper-parameters are:

hyper=list(b=0.66, K=200, DF2=FALSE, FDR=FALSE, pval.thres=0.10)

where b=(% of total data to sample; default=66%), K=# of subsamples, FDR (FDR based multiplicity correction for p-values), pval.thres=0.10 (adjust to change filtering threshold). DF2 fits Y~ranger(X, A, XA) and calculates the VI_2DF = VI_X+VI_XA, which is the variable importance of the main effect + the interaction effect (joint test). Var(VI_2DF) = Var(VI_X)+Var(VI_AX)+2cov(VI_X, VI_AX) where each component is calculated using the subsampling approach described above.

Value

Trained filter model and vector of variable names that pass the filter.

- mod - trained model
- filter.vars - Variables that remain after filtering (could be all)

References

Friedman, J., Hastie, T. and Tibshirani, R. (2008) Regularization Paths for Generalized Linear Models via Coordinate Descent, https://web.stanford.edu/~hastie/Papers/glmnet.pdf Journal of Statistical Software, Vol. 33(1), 1-22 Feb 2010 Vol. 33(1), 1-22 Feb 2010.

Wright, M. N. & Ziegler, A. (2017). ranger: A fast implementation of random forests for high dimensional data in C++ and R. J Stat Softw 77:1-17. https://doi.org/10.18637/jss.v077.i01.

Ishwaran, H. Lu, M. (2017). Standard errors and confidence intervals for variable importance in random forest regression, classification, and survival. Statistics in Medicine 2017.

See Also

PRISM

Examples

```
library(StratifiedMedicine)
## Continuous ##
dat_ctns = generate_subgrp_data(family="gaussian")
Y = dat_ctns$Y
X = dat_ctns$X
A = dat_ctns$A

# Fit ple_ranger directly (treatment-specific ranger models) #
mod1 = filter_train(Y, A, X, filter="filter_glmnet")
mod1$filter.vars

mod2 = filter_train(Y, A, X, filter="filter_glmnet", hyper=list(interaction=TRUE))
mod2$filter.vars

mod3 = filter_train(Y, A, X, filter="filter_ranger")
mod3$filter.vars
```

---

generate_subgrp_data *Generate Subgroup Data-sets*

Description

Simulation/real data-sets; useful for testing new models and PRISM configurations.

Usage

```
generate_subgrp_data(n = 800, seed = 513413, family, null = FALSE, ...)
```

Arguments

`n`	sample size (default=800)
`seed`	seed number (default=513413)
`family`	Outcome type ("gaussian", "binomial", "survival")
`null`	Simulate null hypothesis of no treatment effect and no subgruops. Default is FALSE.
`...`	Any additional parameters, not currently passed through.

Value

Simulation data set (Y=outcome, A=treatment, X=covariates)

`param_combine` *Overall Population Estimate: Aggregating Subgroup-Specific Parameter Estimates*

Description

Function that combines subgroup-specific estimates to obtain an overall population estimate. Options including sample size weighting and adaptive weighting (default; as described in Marceau-West and Mehrotra (to appear)).

Usage

```
param_combine(param.dat, combine = "SS", alpha_ovrl = 0.05, ...)
```

Arguments

`param.dat`	Parameter data-set with subgroup-specific point estimates, SEs, and sample sizes.
`combine`	Method to combine subgroup-specific estimates. Default is "adaptive". combine="SS" uses sample size weighting.
`alpha_ovrl`	Two-sided alpha level for overall population. Default=0.05
`...`	Any additional parameters, not currently passed through.

Value

Data-frame with overall population point estimate, SE, and CI `param_est` *Parameter Estimation: Across Subgroups*

Description

For each identified subgroup, obtain point-estimates and variability metrics (est, SE, CI). fit separate linear regression models. Point-estimates and variability metrics in the overall population are obtained by aggregating subgroup specific results (adaptive weighting or sample size weighting).

Usage

```
param_est(
  Y,
  A,
  X,
  param,
  mu_hat = NULL,
  Subgrps,
  alpha_ovrl = 0.05,
  alpha_s = 0.05,
  combine = "SS",
  ...
)
```

Arguments

`Y`	The outcome variable. Must be numeric or survival (ex; Surv(time,cens))
`A`	Treatment variable. (Default supports binary treatment, either numeric or factor). "ple_train" accomodates >2 along with binary treatments.
`X`	Covariate space.
`param`	Parameter estimation and inference function. Based on the discovered subgroups, estimate parameter estimates and correspond variability metrics. Options include "lm" (unadjusted linear regression), "dr" (doubly-robust estimator), "ple" (G-computation, average the patient-level estimates), "cox" (cox regression), and "rmst" (RMST based estimates as in survRMST package). Default for "gaussian", "binomial" is "dr", while default for "survival" is "cox". Currently only available for binary treatments or A=NULL.
`mu_hat`	Patient-level estimates (see `ple_train`)
`Subgrps`	Identified subgroups. Can be pre-specified, or determined adaptively (see `submod_train`).
`alpha_ovrl`	Two-sided alpha level for overall population
`alpha_s`	Two-sided alpha level at subgroup
`combine`	Given identified subgroups and correspond point-estimates/SEs/sample sizes, combine="SS" will use sample size weighting for estimates at the overall level. Not applicable for param="dr","ple".
`...`	Any additional parameters, not currently passed through.

Value

Data-set with parameter estimates and corresponding variability metrics, for overall and subgroups. Subgrps="ovrl" corresponds to the overall population by default.

- param.dat - Parameter estimates and variability metrics (est, SE, LCL/UCL = lower/upper confidence limits, pval = p-value).

References

Funk et al. Doubly Robust Estimation of Causal Effects. Am J Epidemiol 2011. 173(7): 761-767.

Andersen, P. and Gill, R. (1982). Cox's regression model for counting processes, a large sample study. Annals of Statistics 10, 1100-1120.

Uno et al. Moving beyond the hazard ratio in quantifying the between-group difference in survival analysis. Journal of clinical Oncology 2014, 32, 2380-2385.

See Also

param_combine

Examples

```
library(StratifiedMedicine)

## Continuous ##
dat_ctns = generate_subgrp_data(family="gaussian")
Y = dat_ctns$Y
X = dat_ctns$X
A = dat_ctns$A

## Identify Subgroups: MOB (lmtree) ##
res_lmtree = submod_train(Y, A, X, submod="lmtree")

## Parameter-estimation ##
param.dat = param_est(Y, A, X, param="lm", Subgrps = res_lmtree$Subgrps.train)
param.dat
```

---

ple_train *Patient-level Estimates: Train Model*

Description

Wrapper function to train a patient-level estimate (ple) model. Used directly in PRISM and can be used to directly fit a ple model by name.

Usage

```
ple_train(
  Y,
  A,
  X,
  Xtest = NULL,
  family = "gaussian",
  propensity = FALSE,
  ple = "ranger",
  meta = "X-learner",
  hyper = NULL,
  tau = NULL,
  ...
)
```

Arguments

Y	The outcome variable. Must be numeric or survival (ex; Surv(time,cens))
A	Treatment variable. (Default supports binary treatment, either numeric or factor). "ple_train" accomodates >2 along with binary treatments.
X	Covariate space.
Xtest	Test set. Default is NULL which uses X (training set). Variable types should match X.
family	Outcome type. Options include "gaussion" (default), "binomial", and "survival".
propensity	Propensity score estimation, P(A=alX). Default=FALSE which use the marginal estimates, P(A=a) (applicable for RCT data). If TRUE, will use "ple" base learner.
ple	Base-learner used to estimate patient-level equantities, such as the individual treatment effect. Default is random based based through "ranger". "None" uses no ple. See below for details on estimating the treatment contrasts.
meta	Using the ple model as a base learner, meta-learners can be used for estimating patient-level treatment differences. Options include "T-learner" (treatment specific models), "S-learner" (single model), and "X-learner". For family="gaussian" & "binomial", the default is "X-learner", which uses a two-stage regression approach (See Kunzel et al 2019). For "survival", the default is "T-learner".
hyper	Hyper-parameters for the ple model (must be list). Default is NULL.
tau	Maximum follow-up time for RMST based estimates (family="survival"). Default=NULL, which takes min(max(time[a])), for a=1,..,A.
...	Any additional parameters, not currently passed through.

Details

ple_train uses base-learners along with a meta-learner to obtain patient-level estimates under different treatment exposures (see Kunzel et al). For family="gaussian" or "binomial", output estimates of $mu(a,x) = E(Y|x,a)$ and treatment differences (average treatment effect or risk difference).

For survival, either logHR based estimates or RMST based estimates can be obtained. Current base-learner ("ple") options include:

1. **linear**: Uses either linear regression (family="gaussian"), logistic regression (family="binomial"), or cox regression (family="survival"). No hyper-parameters.

2. **ranger**: Uses random forest ("ranger" R package). The default hyper-parameters are: hyper = list(mtry=NULL, min.node.pct=0.10)

where mtry is number of randomly selected variables (default=NULL; sqrt(dim(X))) and min.node.pct is the minimum node size as a function of the total data size (ex: min.node.pct=10% requires at least 10

3. **glmnet**: Uses elastic net ("glmnet" R package). The default hyper-parameters are: hyper = list(lambda="lambda.min")

where lambda controls the penalty parameter for predictions. lambda="lambda.1se" will likely result in a less complex model.

4. **bart**: Uses bayesian additive regression trees (Chipman et al 2010; BART R package). Default hyper-parameters are:

hyper = list(sparse=FALSE)

where sparse controls whether to perform variable selection based on a sparse Dirichlet prior rather than simply uniform.

Value

Trained ple models and patient-level estimates for train/test sets.

- mod - trained model(s)
- mu_train - Patient-level estimates (training set)
- mu_test - Patient-level estimates (test set)

References

Wright, M. N. & Ziegler, A. (2017). ranger: A fast implementation of random forests for high dimensional data in C++ and R. J Stat Softw 77:1-17. https://doi.org/10.18637/jss.v077.i01.

Friedman, J., Hastie, T. and Tibshirani, R. (2008) Regularization Paths for Generalized Linear Models via Coordinate Descent, https://web.stanford.edu/~hastie/Papers/glmnet.pdf Journal of Statistical Software, Vol. 33(1), 1-22 Feb 2010 Vol. 33(1), 1-22 Feb 2010.

Chipman, H., George, E., and McCulloch R. (2010) Bayesian Additive Regression Trees. The Annals of Applied Statistics, 4,1, 266-298

Kunzel S, Sekhon JS, Bickel PJ, Yu B. Meta-learners for Estimating Hetergeneous Treatment Effects using Machine Learning. 2019.

See Also

PRISM

Examples

```
library(StratifiedMedicine)
## Continuous ##
dat_ctns = generate_subgrp_data(family="gaussian")
Y = dat_ctns$Y
X = dat_ctns$X
A = dat_ctns$A


# X-Learner (With ranger based learners)
mod1 = ple_train(Y=Y, A=A, X=X, Xtest=X, ple="ranger", method="X-learner")
summary(mod1$mu_train)

# T-Learner (Treatment specific)
mod2 = ple_train(Y=Y, A=A, X=X, Xtest=X, ple="ranger", method="T-learner")
summary(mod2$mu_train)


mod3 = ple_train(Y=Y, A=A, X=X, Xtest=X, ple="bart", method="X-learner")
summary(mod3$mu_train)
```

`plot.PRISM` *plot.PRISM*

Description

Plots PRISM results. Options include "tree", "forest", "resample", and "PLE:waterfall".

Usage

```
## S3 method for class 'PRISM'
plot(
  x,
  type = "tree",
  target = NULL,
  grid.data = NULL,
  grid.thres = ">0",
  tree.thres = NULL,
  est.resamp = TRUE,
  tree.plots = "outcome",
  nudge_out = 0.1,
  width_out = 0.5,
  nudge_dens = ifelse(tree.plots == "both", 0.3, 0.1),
  width_dens = 0.5,
  ...
)
```

Arguments

`x`	PRISM object
`type`	Type of plot (default="tree", `ggparty` based plot with parameter estimates, along with options for including outcome or probability based plots). Other options include "forest" (forest plot for overall and subgroups),"PLE:waterfall" (waterfall plot of PLEs), "PLE:density" (density plot of PLEs), "resample" (resampling distribution of parameter estimates for overall and subgroups), and "heatmap" (heatmap of ple estimates/probabilities). For "tree" and "forest", CIs are based on the observed data unless resampling is used. For bootstrap resampling, if calibrate=TRUE, then calibrated CIs along are shown, otherse CIs based on the percentile method are shown.
`target`	For "resample" plot only, must be specify which estimand to visualize. Default=NULL.
`grid.data`	Input grid of values for 2-3 covariates (if 3, last variable cannot be continuous). This is required for type="heatmap". Default=NULL.
`grid.thres`	Threshold for PLE, ex: I(PLE>thres). Used to estimate P(PLE>thres) for type="heatmap". Default is ">0". Direction can be reversed and can include equality sign (ex: "<=").
`tree.thres`	Probability threshold, ex: P(Mean(A=1 vs A=0)>c. Default=NULL, which defaults to using ">0", unless param="cox", which "P(HR(A=1 vs A=0))<1". If a density plot is included, setting tree.thres=">c" will use green colors for values above c, and red colors for values below c. If tree.thres="<c", the reverse color scheme is used.
`est.resamp`	Should plot present resampling based estimates? Default=TRUE if bootstrap or CV based resampling is used. Only applicable for type="submod". If bootstrap calibration is used, calibrated CIs are presented. If no calibration, then percentile Cis are presented with the smoothed bootstrap point-estimates.
`tree.plots`	Type of plots to include in the "tree" plot. Default="outcome". For non-survival data, this includes boxplots of treatment-specific outcomes (param="lm"), model-based estimates (param="ple"), or double-robust pseudo outcomes (param="lm"). For survival data, kaplan-meier plots are shown. For "density", the estimated probability density of the treatment effects is shown (normal approximation, unless resampling is used). "both" combines both plots.
`nudge_out`	Nudge tree outcome plot (see ggparty for details)
`width_out`	Width of tree outcome plot (see ggparty for details)
`nudge_dens`	Nudge tree density plot
`width_dens`	Width of density tree outcome plot
`...`	Additional arguments (currently ignored).

Value

Plot (ggplot2) object

See Also

PRISM

plot_dependence *Partial dependence plots: Single Variable (marginal effect) or heat map (2 to 3 variables).*

Description

Partial dependence plots: Single Variable (marginal effect) or heat map (2 to 3 variables).

Usage

```
plot_dependence(object, X = NULL, target = NULL, vars, grid.data = NULL, ...)
```

Arguments

`object`	Fitted `ple_train` or `PRISM` object
`X`	input covariate space. Default=NULL.
`target`	Which patient-level estimate to target for PDP based plots. Default=NULL, which uses the estimated treatment difference.
`vars`	Variables to visualize (ex: c("var1", "var2", "var3)). If no grid.data provided, defaults to using seq(min(var), max(var)) for each continuous variables. For categorical, uses all categories.
`grid.data`	Input grid of values for 2-3 covariates (if 3, last variable cannot be continuous). This is required for type="heatmap". Default=NULL.
`...`	Additional arguments (currently ignored).

Value

Plot (ggplot2) object

References

- Friedman, J. Greedy function approximation: A gradient boosting machine. Annals of statistics (2001): 1189-1232
- Zhao, Qingyuan, and Trevor Hastie. Causal interpretations of black-box models. Journal of Business & Economic Statistics, to appear. (2017).

Examples

```
library(StratifiedMedicine)
## Continuous ##
dat_ctns = generate_subgrp_data(family="gaussian")
Y = dat_ctns$Y
X = dat_ctns$X
A = dat_ctns$A
```

```
# Fit through ple_train wrapper #
mod = ple_train(Y=Y, A=A, X=X, Xtest=X, ple="ranger", meta="X-learner")
plot_dependence(mod, X=X, vars="X1")
```

`plot_importance` *Importance Plot: Visualize relative importance of variables*

Description

Importance is currently based on the PRISM filter model. For elastic net (filter_glmnet). variables with non-zero coefficients are shown. For random forest variable importance (filter_ranger), variables are sorted by their p-values, and "top_n" will show only the "top_n" most importance variables (based on p-values).

Usage

```
plot_importance(object, top_n = NULL, ...)
```

Arguments

`object`	PRISM object
`top_n`	Show top_n variables only, default=NULL (show all)
`...`	Additional arguments (currently ignored).

Value

Plot (ggplot2) object

Examples

```
library(StratifiedMedicine)
## Continuous ##
dat_ctns = generate_subgrp_data(family="gaussian")
Y = dat_ctns$Y
X = dat_ctns$X
A = dat_ctns$A

mod1 = filter_train(Y=Y, A=A, X=X)
plot_importance(mod1)
```

`plot_ple` *Patient-Level Estimate Plot (plot_ple): Visualize distribution of estimates*

Description

Plots based on Patient-level estimate (see `ple_train`) model results. Options include "waterfall" and "density". Target controls which column of "mu_train" (from ple_train object) is shown on the plot.

Usage

```
plot_ple(object, target = NULL, type = "waterfall", ...)
```

Arguments

`object`	`ple_train` object
`target`	Which patient-level estimate to visualize. Default=NULL, which uses the estimated treatment difference.
`type`	TYpe of plot. Default="waterfall"; type="density" shows density plot.
`...`	Additional arguments (currently ignored).

Value

Plot (ggplot2) object

Examples

```
library(StratifiedMedicine)
## Continuous ##
dat_ctns = generate_subgrp_data(family="gaussian")
Y = dat_ctns$Y
X = dat_ctns$X
A = dat_ctns$A


mod1 = ple_train(Y=Y, A=A, X=X, Xtest=X, ple="ranger", meta="X-learner")
plot_ple(mod1)
```

predict.ple_train *Patient-level Estimates Model: Prediction*

Description

Prediction function for the trained patient-level estimate (ple) model.

Usage

```
## S3 method for class 'ple_train'
predict(object, newdata = NULL, ...)
```

Arguments

object	Trained ple model.
newdata	Data-set to make predictions at (Default=NULL, predictions correspond to training data).
...	Any additional parameters, not currently passed through.

Value

Data-frame with predictions (depends on trained ple model).

See Also

PRISM

Examples

```
library(StratifiedMedicine)
## Continuous ##
dat_ctns = generate_subgrp_data(family="gaussian")
Y = dat_ctns$Y
X = dat_ctns$X
A = dat_ctns$A


mod1 = ple_train(Y=Y, A=A, X=X, Xtest=X, ple="ranger", meta="X-learner")
summary(mod1$mu_train)

res1 = predict(mod1, newdata=X)
summary(res1)
```

predict.PRISM	*PRISM: Patient Response Identifier for Stratified Medicine (Predictions)*

Description

Predictions for PRISM algorithm. Given the training set (Y,A,X) or new test set (Xtest), output ple predictions and identified subgroups with correspond parameter estimates.

Usage

```
## S3 method for class 'PRISM'
predict(object, newdata = NULL, type = "all", ...)
```

Arguments

`object`	Trained PRISM model.
`newdata`	Data-set to make predictions at (Default=NULL, predictions correspond to training data).
`type`	Type of prediction. Default is "all" (ple, submod, and param predictions). Other options include "ple" (ple predictions), "submod" (submod predictions with associated parameter estimates).
`...`	Any additional parameters, not currently passed through.

Value

Data-frame with predictions (ple, submod, or both).

Examples

```
## Load library ##
library(StratifiedMedicine)

##### Examples: Continuous Outcome ###########

dat_ctns = generate_subgrp_data(family="gaussian")
Y = dat_ctns$Y
X = dat_ctns$X
A = dat_ctns$A

# Run Default: filter_glmnet, ple_ranger, lmtree, param_ple #
res0 = PRISM(Y=Y, A=A, X=X)
summary(predict(res0, X)) # all #
summary(predict(res0, X, type="ple"))
summary(predict(res0, X, type="submod"))
```

predict.submod_train *Subgroup Identification: Train Model (Predictions)*

Description

Prediction function for the trained subgroup identification model (submod).

Usage

```
## S3 method for class 'submod_train'
predict(object, newdata = NULL, ...)
```

Arguments

object	Trained submod model.
newdata	Data-set to make predictions at (Default=NULL, predictions correspond to training data).
...	Any additional parameters, not currently passed through.

Value

Identified subgroups with subgroup-specific predictions (depends on subgroup model)

- Subgrps - Identified subgroups
- pred - Predictions, depends on subgroup model

Examples

```
library(StratifiedMedicine)
## Continuous ##
dat_ctns = generate_subgrp_data(family="gaussian")
Y = dat_ctns$Y
X = dat_ctns$X
A = dat_ctns$A

# Fit through submod_train wrapper #
mod1 = submod_train(Y=Y, A=A, X=X, Xtest=X, submod="submod_lmtree")
out1 = predict(mod1)
table(mod1$Subgrps.train)
table(out1$Subgrps)
```

PRISM *PRISM: Patient Response Identifier for Stratified Medicine*

Description

PRISM algorithm. Given a data-set of (Y, A, X) (Outcome, treatment, covariates), the PRISM identifies potential subgroups along with point-estimate and variability metrics; with and without resampling (bootstrap or cross-validation based). This four step procedure (filter, ple, submod, param) is flexible and accepts user-inputs at each step.

Usage

```
PRISM(
  Y,
  A = NULL,
  X,
  Xtest = NULL,
  family = "gaussian",
  filter = "glmnet",
  ple = "ranger",
  submod = NULL,
  param = NULL,
  meta = "X-learner",
  pool = "no",
  delta = ">0",
  propensity = FALSE,
  alpha_ovrl = 0.05,
  alpha_s = 0.05,
  filter.hyper = NULL,
  ple.hyper = NULL,
  submod.hyper = NULL,
  param.hyper = NULL,
  bayes = NULL,
  prefilter_resamp = FALSE,
  resample = NULL,
  stratify = TRUE,
  R = NULL,
  calibrate = FALSE,
  alpha.mat = NULL,
  filter.resamp = NULL,
  ple.resamp = NULL,
  submod.resamp = NULL,
  verbose = TRUE,
  verbose.resamp = FALSE,
  seed = 777
)
```

Arguments

`Y`	The outcome variable. Must be numeric or survival (ex; Surv(time,cens))
`A`	Treatment variable. (Default supports binary treatment, either numeric or factor). "ple_train" accomodates >2 along with binary treatments.
`X`	Covariate space.
`Xtest`	Test set. Default is NULL which uses X (training set). Variable types should match X.
`family`	Outcome type. Options include "gaussion" (default), "binomial", and "survival".
`filter`	Filter model to determine variables that are likely associated with the outcome and/or treatment. Outputs a potential reduce list of varia where X.star has potentially less variables than X. Default is "glmnet" (elastic net). Other options include "ranger" (random forest based variable importance with p-values). See `filter_train` for more details. "None" uses no filter.
`ple`	Base-learner used to estimate patient-level equantities, such as the individual treatment effect. Default is random based based through "ranger". "None" uses no ple. See below for details on estimating the treatment contrasts.
`submod`	Subgroup identification model function. Maps the observed data and/or PLEs to subgroups. Default for family="gaussian" is "lmtree" (MOB with OLS loss). For "binomial" the default is "glmtree" (MOB with binomial loss). Default for "survival" is "mob_weib" (MOB with weibull loss). "None" uses no submod. Currently only available for binary treatments or A=NULL.
`param`	Parameter estimation and inference function. Based on the discovered subgroups, estimate parameter estimates and correspond variability metrics. Options include "lm" (unadjusted linear regression), "dr" (doubly-robust estimator), "ple" (G-computation, average the patient-level estimates), "cox" (cox regression), and "rmst" (RMST based estimates as in survRMST package). Default for "gaussian", "binomial" is "dr", while default for "survival" is "cox". Currently only available for binary treatments or A=NULL.
`meta`	Using the ple model as a base learner, meta-learners can be used for estimating patient-level treatment differences. Options include "T-learner" (treatment specific models), "S-learner" (single model), and "X-learner". For family="gaussian" & "binomial", the default is "X-learner", which uses a two-stage regression approach (See Kunzel et al 2019). For "survival", the default is "T-learner".
`pool`	Whether to pool discovered subgroups. Default is "no" (no pooling). Other options include "otr:logistic", which uses an optimal treatment regime approach, where a weighted logistic regression is fit with I(mu_1-mu_0>delta) as the outcome, the candidate subgroups as covariates, and weights=abs(PLE). Lastly, the youden index is used to assign optimal treatments across the discovered subgroups.
`delta`	Threshold for defining benefit vs non-benefitting patients. Only applicable for pool="otr:logistic" or "otr:rf"; Default=">0".
`propensity`	Propensity score estimation, P(A=a|X). Default=FALSE which use the marginal estimates, P(A=a) (applicable for RCT data). If TRUE, will use "ple" base learner.

`alpha_ovrl` Two-sided alpha level for overall population. Default=0.05

`alpha_s` Two-sided alpha level at subgroup level. Default=0.05

`filter.hyper` Hyper-parameters for the filter function (must be list). Default is NULL.

`ple.hyper` Hyper-parameters for the PLE function (must be list). Default is NULL.

`submod.hyper` Hyper-parameters for the submod function (must be list). Default is NULL.

`param.hyper` Hyper-parameters for the param function (must be list). Default is NULL.

`bayes` Based on input point estimates/SEs, this uses a bayesian based approach to obtain ests, SEs, CIs, and posterior probabilities. Currently includes "norm_norm" (normal prior at overall estimate with large uninformative variance; normal posterior). Default=NULL.

`prefilter_resamp` Option to filter the covariate space (based on filter model) prior to resampling. Default=FALSE.

`resample` Resampling method for resample-based estimates and variability metrics. Options include "Bootstrap", "Permutation", and "CV" (cross-validation). Default=NULL (No resampling).

`stratify` Stratified resampling (Default=TRUE)

`R` Number of resamples (default=NULL; R=100 for Permutation/Bootstrap and R=5 for CV)

`calibrate` Bootstrap calibration for nominal alpha (Loh et al 2016). Default=FALSE. For TRUE, outputs the calibrated alpha level and calibrated CIs for the overall population and subgroups. Not applicable for permutation/CV resampling.

`alpha.mat` Grid of alpha values for calibration. Default=NULL, which uses seq(alpha/1000,alpha,by=0.005) for alpha_ovrl/alpha_s.

`filter.resamp` Filter function during resampling, default=NULL (use filter)

`ple.resamp` PLE function during resampling, default=NULL (use ple)

`submod.resamp` submod function for resampling, default=NULL (use submod)

`verbose` Detail progress of PRISM? Default=TRUE

`verbose.resamp` Output iterations during resampling? Default=FALSE

`seed` Seed for PRISM run (Default=777)

Details

PRISM is a general framework with five key steps:

0. Estimand: Determine the question of interest (ex: mean treatment difference)

1. Filter (filter): Reduce covariate space by removing noise covariates. Options include elastic net ("glmnet") and random forest variable importance ("ranger").

2. Patient-Level Estimates (ple): Estimate counterfactual patient-level quantities, for example, the individual treatment effect, E(Y|A=1)-E(Y|A=0). This calls the "ple_train" function, and follows the framework of Kunzel et al 2019. Base-learners include random forest ("ranger"), BART ("bart"), elastic net ("glmnet"), and linear models (LM, GLM, or Cox regression). Meta-learners include the "S-Learner" (single model), "T-learner" (treatment specific models), and "X-learner" (2-stage approach).

3. Subgroup Model (submod): Currently uses tree-based methods to identify predictive and/or prognostic subgroups. Options include MOB OLS ("lmtree"), MOB GLM ("glmtree"), optimal treatment regimes ("otr") conditional inference trees ("ctree"), and recursive partitioning and regression trees ("rpart").

4. Parameter Estimation (param): For the overall population and the discovered subgroups (if any), obtain point-estimates and variability metrics. Options include: cox regression ("cox"), double robust estimator ("dr"), linear regression ("lm"), average of patient-level estimates ("ple"), and restricted mean survival time ("rmst").

Steps 1-4 also support user-specific models. If treatment is provided (A!=NULL), the default settings are as follows:

Y is continuous (family="gaussian"): Elastic Net Filter ==> X-learner with random forest ==> MOB (OLS) ==> Double Robust estimator

Y is binary (family="binomial"): Elastic Net Filter ==> X-learner with random forest ==> MOB (GLM) ==> Double Robust estimator

Y is right-censored (family="survival"): Elastic Net Filter ==> T-learner with random forest ==> MOB (Weibull) ==> Cox regression

If treatment is not provided (A=NULL), the default settings are as follows:

Y is continuous (family="gaussian"): Elastic Net Filter ==> Random Forest ==> ctree ==> linear regression

Y is binary (family="binomial"): Elastic Net Filter ==> Random Forest ==> ctree ==> linear regression

Y is right-censored (family="survival"): Elastic Net Filter ==> Survival Random Forest ==> ctree ==> RMST

Value

Trained PRISM object. Includes filter, ple, submod, and param outputs.

- filter.mod - Filter model
- filter.vars - Variables remaining after filtering
- ple.fit - Fitted ple model (model fit, other fit outputs)
- mu_train - Patient-level estimates (train)
- mu_test - Patient-level estimates (test)
- submod.fit - Fitted submod model (model fit, other fit outputs)
- out.train - Training data-set with identified subgroups
- out.test - Test data-set with identified subgroups
- Rules - Subgroup rules / definitions
- param.dat - Parameter estimates and variablity metrics (depends on param)
- resamp.dist - Resampling distributions (NULL if no resampling is done)
- bayes.fun - Function to simulate posterior distribution (NULL if no bayes)

References

Friedman, J., Hastie, T. and Tibshirani, R. (2008) Regularization Paths for Generalized Linear Models via Coordinate Descent, https://web.stanford.edu/~hastie/Papers/glmnet.pdf Journal of Statistical Software, Vol. 33(1), 1-22 Feb 2010 Vol. 33(1), 1-22 Feb 2010.

Jemielita T, Mehrotra D. PRISM: Patient Response Identifiers for Stratified Medicine. https://arxiv.org/abs/1912.03337

Hothorn T, Hornik K, Zeileis A (2006). Unbiased Recursive Partitioning: A Conditional Inference Framework. Journal of Computational and Graphical Statistics, 15(3), 651–674.

Wright, M. N. & Ziegler, A. (2017). ranger: A fast implementation of random forests for high dimensional data in C++ and R. J Stat Softw 77:1-17. https://doi.org/10.18637/jss.v077.i01.

Zeileis A, Hothorn T, Hornik K (2008). Model-Based Recursive Partitioning. Journal of Computational and Graphical Statistics, 17(2), 492–514.

Examples

```
## Load library ##
library(StratifiedMedicine)

## Examples: Continuous Outcome ##

dat_ctns = generate_subgrp_data(family="gaussian")
Y = dat_ctns$Y
X = dat_ctns$X
A = dat_ctns$A

# Run Default: filter_glmnet, ple_ranger, lmtree, param_ple #
res0 = PRISM(Y=Y, A=A, X=X)

summary(res0)
plot(res0)

# Without filtering #

res1 = PRISM(Y=Y, A=A, X=X, filter="None")
summary(res1)
plot(res1)


# Search for Prognostic Only (omit A from function) #

res3 = PRISM(Y=Y, X=X)
summary(res3)
plot(res3)


## With bootstrap (No filtering) ##

library(ggplot2)
  res_boot = PRISM(Y=Y, A=A, X=X, resample = "Bootstrap", R=50, verbose.resamp = TRUE)
```

```
  # Plot of distributions and P(est>0) #
  plot(res_boot, type="resample", estimand = "E(Y|A=1)-E(Y|A=0)")+
  geom_vline(xintercept = 0)
  aggregate(I(est>0)~Subgrps, data=res_boot$resamp.dist, FUN="mean")


## Examples: Binary Outcome ##

dat_bin = generate_subgrp_data(family="binomial")
Y = dat_bin$Y
X = dat_bin$X
A = dat_bin$A

# Run Default: glmnet, ranger, glmtree, dr #
res0 = PRISM(Y=Y, A=A, X=X)

plot(res0)


# Survival Data ##

  library(survival)
  library(ggplot2)
  require(TH.data); require(coin)
  data("GBSG2", package = "TH.data")
  surv.dat = GBSG2
  # Design Matrices ###
  Y = with(surv.dat, Surv(time, cens))
  X = surv.dat[,!(colnames(surv.dat) %in% c("time", "cens")) ]
  set.seed(513)
  A = rbinom( n = dim(X)[1], size=1, prob=0.5  )

  # PRISM: glmnet ==> Random Forest to estimate Treatment-Specific RMST
  # ==> MOB (Weibull) ==> Cox for HRs#
  res_weib = PRISM(Y=Y, A=A, X=X)
  plot(res_weib, type="PLE:waterfall")
  plot(res_weib)

  # PRISM: glmnet ==> Random Forest to estimate Treatment-Specific RMST
  # ==> OTR (CTREE, uses RMST estimates as input) ==> Cox for HRs #
  res_otr = PRISM(Y=Y, A=A, X=X)
  plot(res_otr)

  # PRISM: ENET ==> CTREE ==> Cox; with bootstrap #
  res_ctree1 = PRISM(Y=Y, A=A, X=X, ple="None", submod = "ctree",
                   resample="Bootstrap", R=50, verbose.resamp = TRUE)
  plot(res_ctree1)
 plot(res_ctree1, type="resample", estimand="HR(A=1 vs A=0)")+geom_vline(xintercept = 1)
  aggregate(I(est<0)~Subgrps, data=res_ctree1$resamp.dist, FUN="mean")
```

submod_train — *Subgroup Identification: Train Model*

Description

Wrapper function to train a subgroup model (submod). Outputs subgroup assignments and fitted model.

Usage

```
submod_train(
  Y,
  A,
  X,
  Xtest = NULL,
  mu_train = NULL,
  family = "gaussian",
  submod,
  hyper = NULL,
  pool = "no",
  delta = ">0",
  ...
)
```

Arguments

`Y`	The outcome variable. Must be numeric or survival (ex; Surv(time,cens))
`A`	Treatment variable. (Default supports binary treatment, either numeric or factor). "ple_train" accomodates >2 along with binary treatments.
`X`	Covariate space.
`Xtest`	Test set. Default is NULL which uses X (training set). Variable types should match X.
`mu_train`	Patient-level estimates in training set (see `ple_train`). Default=NULL
`family`	Outcome type. Options include "gaussion" (default), "binomial", and "survival".
`submod`	Subgroup identification model function. Maps the observed data and/or PLEs to subgroups. Default for family="gaussian" is "lmtree" (MOB with OLS loss). For "binomial" the default is "glmtree" (MOB with binomial loss). Default for "survival" is "mob_weib" (MOB with weibull loss). "None" uses no submod. Currently only available for binary treatments or A=NULL.
`hyper`	Hyper-parameters for submod (must be list). Default is NULL.
`pool`	Whether to pool discovered subgroups. Default is "no" (no pooling). Other options include "otr:logistic", which uses an optimal treatment regime approach, where a weighted logistic regression is fit with I(mu_1-mu_0>delta) as the outcome, the candidate subgroups as covariates, and weights=abs(PLE). Lastly, the youden index is used to assign optimal treatments across the discovered subgroups.

delta Threshold for defining benefit vs non-benefitting patients. Only applicable for pool="otr:logistic" or "otr:rf"; Default=">0".

... Any additional parameters, not currently passed through.

Details

submod_train currently fits a number of tree-based subgroup models, most of which aim to find subgroups with varying treatment effects (i.e. predictive variables). Current options include:

1. lmtree: Wrapper function for the function "lmtree" from the partykit package. Here, model-based partitioning (MOB) with an OLS loss function, Y~MOB_LM(A,X), is used to identify prognostic and/or predictive variables.

Default hyper-parameters are: hyper = list(alpha=0.05, maxdepth=4, parm=NULL, minsize=floor(dim(X)[1]*0.10)).

2. glmtree: Wrapper function for the function "glmtree" from the partykit package. Here, model-based partitioning (MOB) with GLM binomial + identity link loss function, (Y~MOB_GLM(A,X)), is used to identify prognostic and/or predictive variables.

Default hyper-parameters are: hyper = list(link="identity", alpha=0.05, maxdepth=4, parm=NULL, minsize=floor(dim(X)[1]*0.10)).

3. ctree: Wrapper function for the function "ctree" from the partykit package. Here, conditional inference trees are used to identify either prognostic, Y~CTREE(X), or predictive variables, PLE~CTREE(X) (outcome_PLE=TRUE; requires mu_train data).

Default hyper-parameters are: hyper=list(alpha=0.10, minbucket = floor(dim(X)[1]*0.10), maxdepth = 4, outcome_PLE=FALSE).

4. otr: Optimal treatment regime approach using "ctree". Based on patient-level treatment effect estimates, fit PLE~CTREE(X) with weights=abs(PLE).

Default hyper-parameters are: hyper=list(alpha=0.10, minbucket = floor(dim(X)[1]*0.10), maxdepth = 4, thres=">0").

4. mob_weib: Wrapper function for the function "mob" with weibull loss function using the partykit package. Here, model-based partitioning (MOB) with weibull loss (survival), (Y~MOB_WEIB(A,X)), is used to identify prognostic and/or predictive variables.

Default hyper-parameters are: hyper = list(alpha=0.10, maxdepth=4, parm=NULL, minsize=floor(dim(X)[1]*0.10)).

5. rpart: Recursive partitioning through the "rpart" R package. Here, recursive partitioning and regression trees are used to identify either prognostic, Y~rpart(X), or predictive variables, PLE~rpart(X) (outcome_PLE=TRUE; requires mu_train data).

Value

Trained subgroup model and subgroup predictions/estimates for train/test sets.

- mod - trained subgroup model
- Subgrps.train - Identified subgroups (training set)
- Subgrps.test - Identified subgroups (test set)
- pred.train - Predictions (training set)
- pred.test - Predictions (test set)
- Rules - Definitions for subgroups, if provided in fitted submod output.

References

- Zeileis A, Hothorn T, Hornik K (2008). Model-Based Recursive Partitioning. Journal of Computational and Graphical Statistics, 17(2), 492–514.
- Seibold H, Zeileis A, Hothorn T. Model-based recursive partitioning for subgroup analyses. Int J Biostat, 12 (2016), pp. 45-63
- Hothorn T, Hornik K, Zeileis A (2006). Unbiased Recursive Partitioning: A Conditional Inference Framework. Journal of Computational and Graphical Statistics, 15(3), 651–674.
- Zhao et al. (2012) Estimated individualized treatment rules using outcome weighted learning. Journal of the American Statistical Association, 107(409): 1106-1118.
- Breiman L, Friedman JH, Olshen RA, and Stone CJ. (1984) Classification and Regression Trees. Wadsworth

See Also

PRISM

Examples

```
library(StratifiedMedicine)
## Continuous ##
dat_ctns = generate_subgrp_data(family="gaussian")
Y = dat_ctns$Y
X = dat_ctns$X
A = dat_ctns$A

# Fit through submod_train wrapper #
mod1 = submod_train(Y=Y, A=A, X=X, Xtest=X, submod="submod_lmtree")
table(mod1$Subgrps.train)
plot(mod1$fit$mod)
```

---

summary.PRISM *PRISM: Patient Response Identifier for Stratified Medicine (Summary)*

Description

Predictions for PRISM algorithm. Given the training set (Y,A,X) or new test set (Xtest), output ple predictions and identified subgroups with correspond parameter estimates.

Usage

```
## S3 method for class 'PRISM'
summary(object, ...)
```

Arguments

`object`	Trained PRISM model.
`...`	Any additional parameters, not currently passed through.

Value

List of key PRISM outputs: (1) Configuration, (2) Variables that pass filter (if filter is used), (3) Number of Identified Subgroups, and (4) Parameter Estimates, SEs, and CIs for each subgroup/estimand

Index