# Package 'personalized'

May 22, 2019

**Type** Package

**Title** Estimation and Validation Methods for Subgroup Identification and Personalized Medicine

**Version** 0.2.4

**Description** Provides functions for fitting and validation of models for subgroup identification and personalized medicine / precision medicine under the general subgroup identification framework of Chen et al. (2017) <doi:10.1111/biom.12676>. This package is intended for use for both randomized controlled trials and observational studies.

**URL** https://jaredhuling.github.io/personalized/, https://arxiv.org/abs/1809.07905

**BugReports** https://github.com/jaredhuling/personalized/issues

**License** GPL-2

**Encoding** UTF-8

**LazyData** true

**Suggests** knitr, rmarkdown, testthat

**Imports** survival, methods, kernlab, foreach

**Depends** glmnet (>= 2.0-13), mgcv, gbm, ggplot2, plotly

**RoxygenNote** 6.1.1

**VignetteBuilder** knitr

**NeedsCompilation** no

**Author** Jared Huling [aut, cre] (<https://orcid.org/0000-0003-0670-4845>), Aaron Potvien [ctb], Alexandros Karatzoglou [cph], Alex Smola [cph]

**Maintainer** Jared Huling <jaredhuling@gmail.com>

**Repository** CRAN

**Date/Publication** 2019-05-22 14:30:03 UTC

# R topics documented:



---

`check.overlap` *Check propensity score overlap*

---

## Description

Results in a plot to check whether the propensity score has adequate overlap between treatment groups

## Usage

```
check.overlap(x, trt, propensity.func, type = c("histogram", "density",
  "both"), bins = 50L, alpha = ifelse(type == "both", 0.35, 0.5))
```

## Arguments

| | |
|---|---|
| `x` | The design matrix (not including intercept term) |
| `trt` | treatment vector with each element equal to a 0 or a 1, with 1 indicating treatment status is active. |
| `propensity.func` | function that inputs the design matrix x and the treatment vector trt and outputs the propensity score, ie Pr(trt = 1 \| X = x). Function should take two arguments 1) x and 2) trt. See example below. For a randomized controlled trial this can simply be a function that returns a constant equal to the proportion of patients assigned to the treatment group, i.e.: `propensity.func = function(x, trt) 0.5`. |
| `type` | Type of plot to create. Options are either a histogram (`type = "histogram"`) for each treatment group, a density (`type = "density"`) for each treatment group, or to plot both a density and histogram (`type = "code"`) |
| `bins` | integer number of bins for histograms when `type = "histogram"` |

alpha — value between 0 and 1 indicating transparency level (1 for solid, 0 for fully transparent)

## Examples

```
library(personalized)

set.seed(123)
n.obs  <- 250
n.vars <- 15
x <- matrix(rnorm(n.obs * n.vars, sd = 3), n.obs, n.vars)


# simulate non-randomized treatment
xbetat   <- 0.25 + 0.5 * x[,11] - 0.5 * x[,12]
trt.prob <- exp(xbetat) / (1 + exp(xbetat))
trt01    <- rbinom(n.obs, 1, prob = trt.prob)

# create function for fitting propensity score model
prop.func <- function(x, trt)
{
    # fit propensity score model
    propens.model <- cv.glmnet(y = trt,
                               x = x, family = "binomial")
    pi.x <- predict(propens.model, s = "lambda.min",
                    newx = x, type = "response")[,1]
    pi.x
}

check.overlap(x = x,
              trt = trt01,
              propensity.func = prop.func)

# now add density plot with histogram
check.overlap(x = x,
              trt = trt01,
              type = "both",
              propensity.func = prop.func)


# simulated non-randomized treatment with multiple levels
xbetat_1   <- 0.15 + 0.5 * x[,9] - 0.25 * x[,12]
xbetat_2   <- 0.15 - 0.5 * x[,11] + 0.25 * x[,15]
trt.1.prob <- exp(xbetat_1) / (1 + exp(xbetat_1) + exp(xbetat_2))
trt.2.prob <- exp(xbetat_2) / (1 + exp(xbetat_1) + exp(xbetat_2))
trt.3.prob <- 1 - (trt.1.prob + trt.2.prob)
prob.mat <- cbind(trt.1.prob, trt.2.prob, trt.3.prob)
trt    <- apply(prob.mat, 1, function(rr) rmultinom(1, 1, prob = rr))
trt    <- apply(trt, 2, function(rr) which(rr == 1))

# use multinomial logistic regression model with lasso penalty for propensity
propensity.multinom.lasso <- function(x, trt)
{
```

```
    if (!is.factor(trt)) trt <- as.factor(trt)
    gfit <- cv.glmnet(y = trt, x = x, family = "multinomial")

    # predict returns a matrix of probabilities:
    # one column for each treatment level
    propens <- drop(predict(gfit, newx = x, type = "response", s = "lambda.min",
                            nfolds = 5, alpha = 0))

    # return the probability corresponding to the
    # treatment that was observed
    probs <- propens[,match(levels(trt), colnames(propens))]

    probs
}

check.overlap(x = x,
              trt = trt,
              type = "histogram",
              propensity.func = propensity.multinom.lasso)
```

---

`fit.subgroup` *Fitting subgroup identification models*

---

## Description

Fits subgroup identification model class of Chen, et al (2017)

## Usage

```
fit.subgroup(x, y, trt, propensity.func = NULL,
  loss = c("sq_loss_lasso", "logistic_loss_lasso", "poisson_loss_lasso",
  "cox_loss_lasso", "owl_logistic_loss_lasso",
  "owl_logistic_flip_loss_lasso", "owl_hinge_loss", "owl_hinge_flip_loss",
  "sq_loss_lasso_gam", "poisson_loss_lasso_gam", "logistic_loss_lasso_gam",
  "sq_loss_gam", "poisson_loss_gam", "logistic_loss_gam",
  "owl_logistic_loss_gam", "owl_logistic_flip_loss_gam",
  "owl_logistic_loss_lasso_gam", "owl_logistic_flip_loss_lasso_gam",
  "sq_loss_gbm", "poisson_loss_gbm", "logistic_loss_gbm", "cox_loss_gbm",
     "custom"), method = c("weighting", "a_learning"), match.id = NULL,
  augment.func = NULL, fit.custom.loss = NULL, cutpoint = 0,
  larger.outcome.better = TRUE, reference.trt = NULL, retcall = TRUE,
  ...)
```

## Arguments

| | |
|---|---|
| x | The design matrix (not including intercept term) |
| y | The response vector |
| trt | treatment vector with each element equal to a 0 or a 1, with 1 indicating treatment status is active. |
| propensity.func | function that inputs the design matrix x and the treatment vector trt and outputs the propensity score, ie Pr(trt = 1 \| X = x). Function should take two arguments 1) x and 2) trt. See example below. For a randomized controlled trial this can simply be a function that returns a constant equal to the proportion of patients assigned to the treatment group, i.e.: `propensity.func = function(x, trt) 0.5`. |
| loss | choice of both the M function from Chen, et al (2017) and potentially the penalty used for variable selection. All `loss` options starting with `sq_loss` use M(y, v) = (v - y) ^ 2, all options starting with `logistic_loss` use the logistic loss: M(y, v) = y * log(1 + exp{-v}), and all options starting with `cox_loss` use the negative partial likelihood loss for the Cox PH model. All options ending with `lasso` have a lasso penalty added to the loss for variable selection. `sq_loss_lasso_gam` and `logistic_loss_lasso_gam` first use the lasso to select variables and then fit a generalized additive model with nonparametric additive terms for each selected variable. `sq_loss_gam` involves a squared error loss with a generalized additive model and no variable selection. `sq_loss_gbm` involves a squared error loss with a gradient-boosted decision trees model for the benefit score; this allows for flexible estimation using machine learning and can be useful when the underlying treatment-covariate interaction is complex. |

- **Continuous Outcomes**
  - `"sq_loss_lasso"` - M(y, v) = (v - y) ^ 2 with linear model and lasso penalty
  - `"owl_logistic_loss_lasso"` - M(y, v) = ylog(1 + exp{-v}) (method of Regularized Outcome Weighted Subgroup Identification)
  - `"owl_logistic_flip_loss_lasso"` - M(y, v) = |y|log(1 + exp{-sign(y)v})
  - `"owl_hinge_loss"` - M(y, v) = ymax(0, 1 - v) (method of Estimating individualized treatment rules using outcome weighted learning)
  - `"owl_hinge_flip_loss"` - M(y, v) = |y|max(0, 1 - sign(y)v)
  - `"sq_loss_lasso_gam"` - M(y, v) = (v - y) ^ 2 with variables selected by lasso penalty and generalized additive model fit on the selected variables
  - `"sq_loss_gam"` - M(y, v) = (v - y) ^ 2 with generalized additive model fit on all variables
  - `"owl_logistic_loss_gam"` - M(y, v) = ylog(1 + exp{-v}) with generalized additive model fit on all variables
  - `"owl_logistic_flip_loss_gam"` - M(y, v) = |y|log(1 + exp{-sign(y)v}) with generalized additive model fit on all variables
  - `"owl_logistic_loss_lasso_gam"` - M(y, v) = ylog(1 + exp{-v}) with variables selected by lasso penalty and generalized additive model fit on the selected variables

  - "owl_logistic_flip_loss_lasso_gam" - M(y, v) = |y|log(1 + exp{-sign(y)v}) with variables selected by lasso penalty and generalized additive model fit on the selected variables
  - "sq_loss_gbm" - M(y, v) = (v - y) ^ 2 with gradient-boosted decision trees model
- **Binary Outcomes**
  - All losses for continuous outcomes can be used plus the following:
  - "logistic_loss_lasso" - M(y, v) = -[yv - log(1 + exp{-v})] with with linear model and lasso penalty
  - "logistic_loss_lasso_gam" - M(y, v) = -[yv - log(1 + exp{-v})] with variables selected by lasso penalty and generalized additive model fit on the selected variables
  - "logistic_loss_gam" - M(y, v) = -[yv - log(1 + exp{-v})] with generalized additive model fit on all variables
  - "logistic_loss_gbm" - M(y, v) = -[yv - log(1 + exp{-v})] with gradient-boosted decision trees model
- **Count Outcomes**
  - All losses for continuous outcomes can be used plus the following:
  - "poisson_loss_lasso" - M(y, v) = -[yv - exp(v)] with with linear model and lasso penalty
  - "poisson_loss_lasso_gam" - M(y, v) = -[yv - exp(v)] with variables selected by lasso penalty and generalized additive model fit on the selected variables
  - "poisson_loss_gam" - M(y, v) = -[yv - exp(v)] with generalized additive model fit on all variables
  - "poisson_loss_gbm" - M(y, v) = -[yv - exp(v)] with gradient-boosted decision trees model
- **Time-to-Event Outcomes**
  - "cox_loss_lasso" - M corresponds to the negative partial likelihood of the cox model with linear model and additionally a lasso penalty
  - "cox_loss_gbm" - M corresponds to the negative partial likelihood of the cox model with gradient-boosted decision trees model

`method` subgroup ID model type. Either the weighting or A-learning method of Chen et al, (2017)

`match.id` a (character, factor, or integer) vector with length equal to the number of observations in `x` indicating using integers or levels of a factor vector which patients are in which matched groups. Defaults to `NULL` and assumes the samples are not from a matched cohort. Matched case-control groups can be created using any method (propensity score matching, optimal matching, etc). If each case is matched with a control or multiple controls, this would indicate which case-control pairs or groups go together. If `match.id` is supplied, then it is unecessary to specify a function via the `propensity.func` argument. A quick usage example: if the first patient is a case and the second and third are controls matched to it, and the fouth patient is a case and the fifth through seventh patients are matched with it, then the user should specify `match.id = c(1,1,1,2,2,2,2)` or `match.id = c(rep("Grp1", 3),rep("Grp2", 4))`

`augment.func` function which inputs the response `y`, the covariates `x`, and `trt` and outputs predicted values (on the link scale) for the response using a model constructed with `x`. `augment.func()` can also be simply a function of `x` and `y`. This function is used for efficiency augmentation. When the form of the augmentation function is correct, it can provide efficient estimation of the subgroups. Some examples of possible augmentation functions are:

Example 1: `augment.func <- function(x, y) {lmod <- lm(y ~ x); return(fitted(lmod))}`

Example 2:

```
augment.func <- function(x, y, trt) {
    data <- data.frame(x, y, trt)
    lmod <- lm(y ~ x * trt)
    ## get predictions when trt = 1
    data$trt <- 1
    preds_1  <- predict(lmod, data)

    ## get predictions when trt = -1
    data$trt <- -1
    preds_n1 <- predict(lmod, data)

    ## return predictions averaged over trt
    return(0.5 * (preds_1 + preds_n1))
}
```

For binary and time-to-event outcomes, make sure that predictions are returned on the scale of the predictors

Example 3:

```
augment.func <- function(x, y) {
        bmod <- glm(y ~ x, family = binomial())
        return(predict(bmod, type = "link"))
    }
```

`fit.custom.loss`

A function which *minimizes* a user-specified custom loss function M(y,v) to be used in model fitting. If provided, `fit.custom.loss` should take the modified design matrix (which includes an intercept term) as an argument and the responses and optimize a custom weighted loss function.

The loss function $M(y, v)$ to be minimized **MUST** meet the following two criteria:

1. $D_M(y, v) = \partial M(y, v)/\partial v$ must be increasing in v for each fixed y. $D_M(y, v)$ is the partial derivative of the loss function M(y, v) with respect to v
2. $D_M(y, 0)$ is monotone in y

An example of a valid loss function is $M(y, v) = (y - v)^2$. In this case $D_M(y, v) = -2(y - v)$. See Chen et al. (2017) for more details on the restrictions on the loss function $M(y, v)$.

The provided function **MUST** return a list with the following elements:

- `predict` a function that inputs a design matrix and a 'type' argument for the type of predictions and outputs a vector of predictions on the scale of the linear predictor. Note that the matrix provided to 'fit.custom.loss' has a column appended to the first column of `x` corresponding to the treatment main effect. Thus, the prediction function should deal with this, e.g. `predict(model, cbind(1, x))`
- `model` a fitted model object returned by the underlying fitting function
- `coefficients` if the underlying fitting function yields a vector of coefficient estimates, they should be provided here

The provided function **MUST** be a function with the following arguments:

1. `x` design matrix
2. `y` vector of responses
3. `weights` vector for observations weights. The underlying loss function **MUST** have samples weighted according to this vector. See below example
4. `...` additional arguments passed via '`...`'. This can be used so that users can specify more arguments to the underlying fitting function if so desired.

The provided function can also optionally take the following arguments:

- `match.id` vector of case/control cluster IDs. This is useful if cross validation is used in the underlying fitting function in which case it is advisable to sample whole clusters randomly instead of individual observations.
- `offset` if efficiency augmentation is used, the predictions from the outcome model from `augment.func` will be provided via the `offset` argument, which can be used as an offset in the underlying fitting function as a means of incorporating the efficiency augmentation model's predictions
- `trt` vector of treatment statuses
- `family` family of outcome
- `n.trts` numer of treatment levels. Can be useful if there are more than 2 treatment levels

Example 1: Here we minimize $M(y, v) = (y - v)^2$

```
fit.custom.loss <- function(x, y, weights, ...) {
    df <- data.frame(y = y, x)

    # minimize squared error loss with NO lasso penalty
    lmf <- lm(y ~ x - 1, weights = weights,
              data = df, ...)

    # save coefficients
    cfs = coef(lmf)

    # create prediction function. Notice
    # how a column of 1's is appended
    # to ensure treatment main effects are included
    # in predictions
    prd = function(x, type = "response")
    {
```

```
        dfte <- cbind(1, x)
        colnames(dfte) <- names(cfs)
        predict(lmf, data.frame(dfte))
    }
    # return lost of required components
    list(predict = prd, model = lmf, coefficients = cfs)
}
```

Example 2: $M(y, v) = y \exp(-v)$

```
fit.expo.loss <- function(x, y, weights, ...)
{
    ## define loss function to be minimized
    expo.loss <- function(beta, x, y, weights) {
        sum(weights * y * exp(-drop(tcrossprod(x, t(beta) )))
    }

    # use optim() to minimize loss function
    opt <- optim(rep(0, NCOL(x)), fn = expo.loss, x = x, y = y, weights = weights)

    coefs <- opt$par

    pred <- function(x, type = "response") {
        tcrossprod(cbind(1, x), t(coefs))
    }

    # return list of required components
    list(predict = pred, model = opt, coefficients = coefs)
}
```

`cutpoint` numeric value for patients with benefit scores above which (or below which if `larger.outcome.better = FALSE`) will be recommended to be in the treatment group. Can also set `cutpoint = "median"`, which will use the median value of the benefit scores as the cutpoint or can set specific quantile values via `"quantx"` where `"x"` is a number between 0 and 100 representing the quantile value; e.g. `cutpoint = "quant75"` will use the 75th perent upper quantile of the benefit scores as the quantile.

`larger.outcome.better` boolean value of whether a larger outcome is better/preferable. Set to `TRUE` if a larger outcome is better/preferable and set to `FALSE` if a smaller outcome is better/preferable. Defaults to `TRUE`.

`reference.trt` which treatment should be treated as the reference treatment. Defaults to the first level of `trt` if `trt` is a factor or the first alphabetical or numerically first treatment level. Not used for multiple treatment fitting with OWL-type losses.

`retcall` boolean value. if `TRUE` then the passed arguments will be saved. Do not set to `FALSE` if the `validate.subgroup()` function will later be used for your fitted subgroup model. Only set to `FALSE` if memory is limited as setting to `TRUE` saves the design matrix to the fitted object

`...` options to be passed to underlying fitting function. For all `loss` options with 'lasso', this will be passed to `cv.glmnet`. For all `loss` options with 'gam', this will be passed to `gam` from the **mgcv** package Note that for all `loss` options that use `gam()` from the **mgcv** package, the user cannot supply the `gam` argument `method` because it is also an argument of `fit.subgroup`, so instead, to change the `gam method` argument, supply `method.gam`, ie `method.gam = "REML"`.

For all `loss` options with 'hinge', this will be passed to both `weighted.ksvm` and `ipop` from the **kernlab** package

## Value

An object of class `"subgroup_fitted"`.

`predict` A function that returns predictions of the covariate-conditional treatment effects

`model` An object returned by the underlying fitting function used. For example, if the lasso use used to fit the underlying subgroup identification model, this will be an object returned by `cv.glmnet`.

`coefficients` If the underlying subgroup identification model is parametric, `coefficients` will contain the estimated coefficients of the model.

`call` The call that produced the returned object. If `retcall = TRUE`, this will contain all objects supplied to `fit.subgroup()`

`family` The family corresponding to the outcome provided

`loss` The loss function used

`method` The method used (either weighting or A-learning)

`propensity.func` The propensity score function used

`larger.outcome.better` If larger outcomes are preferred for this model

`cutpoint` Benefit score cutoff value used for determining subgroups

`var.names` The names of all variables used

`n.trts` The number of treatment levels

`comparison.trts` All treatment levels other than the reference level

`reference.trt` The reference level for the treatment. This should usually be the control group/level

`trts` All treatment levels

`trt.received` The vector of treatment assignments

`pi.x` A vector of propensity scores

`y` A vector of outcomes

`benefit.scores` A vector of conditional treatment effects, i.e. benefit scores

`recommended.trts` A vector of treatment recommendations (i.e. for each patient, which treatment results in the best expected potential outcomes)

`subgroup.trt.effects`
: (Biased) estimates of the conditional treatment effects and conditional outcomes. These are essentially just empirical averages within different combinations of treatment assignments and treatment recommendations

`individual.trt.effects`
: estimates of the individual treatment effects as returned by `treat.effects`

## References

Chen, S., Tian, L., Cai, T. and Yu, M. (2017), A general statistical framework for subgroup identification and comparative treatment scoring. Biometrics. doi:10.1111/biom.12676 http://onlinelibrary.wiley.com/doi/10.1111/biom.12676/abstract

Xu, Y., Yu, M., Zhao, Y. Q., Li, Q., Wang, S., & Shao, J. (2015), Regularized outcome weighted subgroup identification for differential treatment effects. Biometrics, 71(3), 645-653. doi: 10.1111/biom.12322 http://onlinelibrary.wiley.com/doi/10.1111/biom.12322/full

Zhao, Y., Zeng, D., Rush, A. J., & Kosorok, M. R. (2012), Estimating individualized treatment rules using outcome weighted learning. Journal of the American Statistical Association, 107(499), 1106-1118. doi: 10.1080/01621459.2012.695674 http://dx.doi.org/10.1080/01621459.2012.695674

## See Also

`validate.subgroup` for function which creates validation results for subgroup identification models, `predict.subgroup_fitted` for a prediction function for fitted models from `fit.subgroup`, `plot.subgroup_fitted` for a function which plots results from fitted models, and `print.subgroup_fitted` for arguments for printing options for `fit.subgroup()`. from `fit.subgroup`.

## Examples

```
library(personalized)

set.seed(123)
n.obs  <- 500
n.vars <- 15
x <- matrix(rnorm(n.obs * n.vars, sd = 3), n.obs, n.vars)


# simulate non-randomized treatment
xbetat   <- 0.5 + 0.5 * x[,7] - 0.5 * x[,9]
trt.prob <- exp(xbetat) / (1 + exp(xbetat))
trt01    <- rbinom(n.obs, 1, prob = trt.prob)

trt      <- 2 * trt01 - 1

# simulate response
# delta below drives treatment effect heterogeneity
delta <- 2 * (0.5 + x[,2] - x[,3] - x[,11] + x[,1] * x[,12] )
xbeta <- x[,1] + x[,11] - 2 * x[,12]^2 + x[,13] + 0.5 * x[,15] ^ 2
xbeta <- xbeta + delta * trt

# continuous outcomes
```

```
y <- drop(xbeta) + rnorm(n.obs, sd = 2)

# binary outcomes
y.binary <- 1 * (xbeta + rnorm(n.obs, sd = 2) > 0 )

# count outcomes
y.count <- round(abs(xbeta + rnorm(n.obs, sd = 2)))

# time-to-event outcomes
surv.time <- exp(-20 - xbeta + rnorm(n.obs, sd = 1))
cens.time <- exp(rnorm(n.obs, sd = 3))
y.time.to.event  <- pmin(surv.time, cens.time)
status           <- 1 * (surv.time <= cens.time)

# create function for fitting propensity score model
prop.func <- function(x, trt)
{
    # fit propensity score model
    propens.model <- cv.glmnet(y = trt,
                               x = x, family = "binomial")
    pi.x <- predict(propens.model, s = "lambda.min",
                    newx = x, type = "response")[,1]
    pi.x
}


####################  Continuous outcomes ################################


subgrp.model <- fit.subgroup(x = x, y = y,
                           trt = trt01,
                           propensity.func = prop.func,
                           loss   = "sq_loss_lasso",
                           nfolds = 10)              # option for cv.glmnet

summary(subgrp.model)

# estimates of the individual-specific
# treatment effect estimates:
subgrp.model$individual.trt.effects

# fit lasso + gam model with REML option for gam


subgrp.modelg <- fit.subgroup(x = x, y = y,
                            trt = trt01,
                            propensity.func = prop.func,
                            loss   = "sq_loss_lasso_gam",
                            method.gam = "REML",     # option for gam
                            nfolds = 5)              # option for cv.glmnet

subgrp.modelg
```

```
#####################  Using an augmentation function ######################
## augmentation funcions involve modeling the conditional mean E[Y|T, X]
## and returning predictions that are averaged over the treatment values
## return <- 1/2 * (hat{E}[Y|T=1, X] + hat{E}[Y|T=-1, X])
############################################################################

augment.func <- function(x, y, trt) {
    data <- data.frame(x, y, trt)
    xm <- model.matrix(y~trt*x-1, data = data)

    lmod <- cv.glmnet(y = y, x = xm)
    ## get predictions when trt = 1
    data$trt <- 1
    xm <- model.matrix(y~trt*x-1, data = data)
    preds_1  <- predict(lmod, xm, s = "lambda.min")

    ## get predictions when trt = -1
    data$trt <- -1
    xm <- model.matrix(y~trt*x-1, data = data)
    preds_n1  <- predict(lmod, xm, s = "lambda.min")

    ## return predictions averaged over trt
    return(0.5 * (preds_1 + preds_n1))
}


subgrp.model.aug <- fit.subgroup(x = x, y = y,
                           trt = trt01,
                           propensity.func = prop.func,
                           augment.func    = augment.func,
                           loss   = "sq_loss_lasso",
                           nfolds = 10)              # option for cv.glmnet

summary(subgrp.model.aug)


#####################  Binary outcomes ####################################

# use logistic loss for binary outcomes
subgrp.model.bin <- fit.subgroup(x = x, y = y.binary,
                           trt = trt01,
                           propensity.func = prop.func,
                           loss   = "logistic_loss_lasso",
                           type.measure = "auc",    # option for cv.glmnet
                           nfolds = 5)              # option for cv.glmnet

subgrp.model.bin


#####################  Count outcomes #####################################

# use poisson loss for count/poisson outcomes
```

```
subgrp.model.poisson <- fit.subgroup(x = x, y = y.count,
                           trt = trt01,
                           propensity.func = prop.func,
                           loss   = "poisson_loss_lasso",
                           type.measure = "mse",    # option for cv.glmnet
                           nfolds = 5)              # option for cv.glmnet

subgrp.model.poisson


####################  Time-to-event outcomes ##############################

library(survival)

subgrp.model.cox <- fit.subgroup(x = x, y = Surv(y.time.to.event, status),
                           trt = trt01,
                           propensity.func = prop.func,
                           loss   = "cox_loss_lasso",
                           nfolds = 5)              # option for cv.glmnet

subgrp.model.cox



####################  Using custom loss functions ########################

## Use custom loss function for binary outcomes

fit.custom.loss.bin <- function(x, y, weights, offset, ...) {
    df <- data.frame(y = y, x)

    # minimize logistic loss with NO lasso penalty
    # with allowance for efficiency augmentation
    glmf <- glm(y ~ x - 1, weights = weights,
                offset = offset, # offset term allows for efficiency augmentation
                family = binomial(), ...)

    # save coefficients
    cfs = coef(glmf)

    # create prediction function.
    prd = function(x, type = "response") {
         dfte <- cbind(1, x)
         colnames(dfte) <- names(cfs)
         ## predictions must be returned on the scale
         ## of the linear predictor
         predict(glmf, data.frame(dfte), type = "link")
    }
    # return lost of required components
    list(predict = prd, model = glmf, coefficients = cfs)
}
```

```
subgrp.model.bin.cust <- fit.subgroup(x = x, y = y.binary,
                                      trt = trt01,
                                      propensity.func = prop.func,
                                      fit.custom.loss = fit.custom.loss.bin)

subgrp.model.bin.cust



## try exponential loss for
## positive outcomes

fit.expo.loss <- function(x, y, weights, ...)
{
    expo.loss <- function(beta, x, y, weights) {
        sum(weights * y * exp(-drop(x %*% beta)))
    }

    # use optim() to minimize loss function
    opt <- optim(rep(0, NCOL(x)), fn = expo.loss, x = x, y = y, weights = weights)

    coefs <- opt$par

    pred <- function(x, type = "response") {
        tcrossprod(cbind(1, x), t(coefs))
    }

    # return list of required components
    list(predict = pred, model = opt, coefficients = coefs)
}


# use exponential loss for positive outcomes
subgrp.model.expo <- fit.subgroup(x = x, y = y.count,
                                  trt = trt01,
                                  propensity.func = prop.func,
                                  fit.custom.loss = fit.expo.loss)

subgrp.model.expo
```

---

LaLonde *National Supported Work Study Data*

---

## Description

The LaLonde dataset comes from the National Supported Work Study, which sought to evaluate the effectiveness of an employment trainining program on wage increases.

## Usage

```
LaLonde
```

## Format

A data frame with 722 observations and 12 variables:

**outcome** whether earnings in 1978 are larger than in 1975; 1 for yes, 0 for no

**treat** whether the individual received the treatment; "Yes" or "No"

**age** age in years

**educ** education in years

**black** black or not; factor with levels "Yes" or "No"

**hisp** hispanic or not; factor with levels "Yes" or "No"

**white** white or not; factor with levels "Yes" or "No"

**marr** married or not; factor with levels "Yes" or "No"

**nodegr** No high school degree; factor with levels "Yes" (for no HS degree) or "No"

**log.re75** log of earnings in 1975

**u75** unemployed in 1975; factor with levels "Yes" or "No"

**wts.extrap** extrapolation weights to the 1978 Panel Study for Income Dynamics dataset

## Source

The National Supported Work Study.

## References

LaLonde, R.J. 1986. "Evaluating the econometric evaulations of training programs with experimental data." American Economic Review, Vol.76, No.4, pp. 604-620.

Egami N, Ratkovic M, Imai K (2017). "**FindIt**: Finding Heterogeneous Treatment Effects." R package version 1.1.2, https://CRAN.R-project.org/package=FindIt.

## Examples

```
data(LaLonde)
y <- LaLonde$outcome

trt <- LaLonde$treat

x.varnames <- c("age", "educ", "black", "hisp", "white",
                "marr", "nodegr", "log.re75", "u75")

# covariates
data.x <- LaLonde[, x.varnames]

# construct design matrix (with no intercept)
x <- model.matrix(~ -1 + ., data = data.x)
```

```
const.propens <- function(x, trt)
{
    mean.trt <- mean(trt == "Trt")
    rep(mean.trt, length(trt))
}

subgrp_fit_w <- fit.subgroup(x = x, y = y, trt = trt,
    loss = "logistic_loss_lasso",
    propensity.func = const.propens,
    cutpoint = 0,
    type.measure = "auc",
    nfolds = 10)

summary(subgrp_fit_w)
```

---

`plot.subgroup_fitted` *Plotting results for fitted subgroup identification models*

---

## Description

Plots results for estimated subgroup treatment effects

Plots validation results for estimated subgroup treatment effects

## Usage

```
## S3 method for class 'subgroup_fitted'
plot(x, type = c("boxplot", "density",
  "interaction", "conditional"), avg.line = TRUE, ...)

## S3 method for class 'subgroup_validated'
plot(x, type = c("boxplot", "density",
  "interaction", "conditional", "stability"), avg.line = TRUE, ...)
```

## Arguments

| | |
|---|---|
| `x` | fitted object returned by `validate.subgroup()` or `fit.subgroup()` function |
| `type` | type of plot. `"density"` results in a density plot for the results across all observations (if `x` is from `fit.subgroup()`) or if `x` is from `validate.subgroup()` across iterations of either the bootstrap or training/test re-fitting. For the latter case the test results will be plotted. `"boxplot"` results in boxplots across all observations/iterations of either the bootstrap or training/test re-fitting. For the latter case the test results will be plotted. `"interaction"` creates an interaction plot for the different subgroups (crossing lines here means a meaningful subgroup). For the interaction plot, the intervals around each point represent +1 one SE `"conditional"` For subgroup_fitted objects, plots smoothed (via a GAM smoother) means of the outcomes as a function of the estimated benefit score separately for the treated and untreated groups. For subgroup_validated objects, boxplots of summary statistics within subgroups will be plotted as subgroups are |

defined by different cutoffs of the benefit scores. These cutoffs can be specified via the `benefit.score.quantiles` argument of `validate.subgroup`.

`avg.line` boolean value of whether or not to plot a line for the average value in addition to the density (only valid for `type = "density"`)

`...` not used

## See Also

`fit.subgroup` for function which fits subgroup identification models.

`validate.subgroup` for function which creates validation results and `fit.subgroup` for function which fits subgroup identification models.

## Examples

```
library(personalized)

set.seed(123)
n.obs  <- 250
n.vars <- 15
x <- matrix(rnorm(n.obs * n.vars, sd = 3), n.obs, n.vars)


# simulate non-randomized treatment
xbetat   <- 0.5 + 0.5 * x[,11] - 0.5 * x[,13]
trt.prob <- exp(xbetat) / (1 + exp(xbetat))
trt01    <- rbinom(n.obs, 1, prob = trt.prob)

trt      <- 2 * trt01 - 1

# simulate response
delta <- 2 * (0.5 + x[,2] - x[,3] - x[,11] + x[,1] * x[,12])
xbeta <- x[,1] + x[,11] - 2 * x[,12]^2 + x[,13]
xbeta <- xbeta + delta * trt

# continuous outcomes
y <- drop(xbeta) + rnorm(n.obs, sd = 2)

# create function for fitting propensity score model
prop.func <- function(x, trt)
{
    # fit propensity score model
    propens.model <- cv.glmnet(y = trt,
                               x = x, family = "binomial")
    pi.x <- predict(propens.model, s = "lambda.min",
                    newx = x, type = "response")[,1]
    pi.x
}

subgrp.model <- fit.subgroup(x = x, y = y,
                             trt = trt01,
                             propensity.func = prop.func,
```

```
                          loss   = "sq_loss_lasso",
                          nfolds = 5)              # option for cv.glmnet

subgrp.model$subgroup.trt.effects

plot(subgrp.model)

plot(subgrp.model, type = "boxplot")

plot(subgrp.model, type = "interaction")

plot(subgrp.model, type = "conditional")

valmod <- validate.subgroup(subgrp.model, B = 3,
                            method = "training_test",
                            benefit.score.quantiles = c(0.25, 0.5, 0.75),
                            train.fraction = 0.75)

plot(valmod)


plot(valmod, type = "interaction")

# see how summary statistics of subgroups change
# when the subgroups are defined based on different cutoffs
# (25th quantile of bene score, 50th, and 75th)
plot(valmod, type = "conditional")

# visualize the frequency of particular variables
# of being selected across the resampling iterations with
# 'type = "stability"'
# not run:
# plot(valmod, type = "stability")
```

---

`plotCompare` *Plot a comparison results for fitted or validated subgroup identification models*

---

## Description

Plots comparison of results for estimated subgroup treatment effects

## Usage

```
plotCompare(..., type = c("boxplot", "density", "interaction",
  "conditional"), avg.line = TRUE)
```

## Arguments

| | |
|---|---|
| ... | the fitted (model or validation) objects to be plotted. Must be either objects returned from `fit.subgroup()` or `validate.subgroup()` |
| type | type of plot. `"density"` results in a density plot for the results across all observations (if `x` is from `fit.subgroup()`) or if `x` is from `validate.subgroup()` across iterations of either the bootstrap or training/test re-fitting. For the latter case the test results will be plotted. `"boxplot"` results in boxplots across all observations/iterations of either the bootstrap or training/test re-fitting. For the latter case the test results will be plotted. `"interaction"` creates an interaction plot for the different subgroups (crossing lines here means a meaningful subgroup). `"conditional"` plots smoothed (via a GAM smoother) means of the outcomes as a function of the estimated benefit score separately for the treated and untreated groups. |
| avg.line | boolean value of whether or not to plot a line for the average value in addition to the density (only valid for `type = "density"`) |

## See Also

`fit.subgroup` for function which fits subgroup identification models and `validate.subgroup` for function which creates validation results.

## Examples

```
library(personalized)

set.seed(123)
n.obs  <- 100
n.vars <- 15
x <- matrix(rnorm(n.obs * n.vars, sd = 3), n.obs, n.vars)


# simulate non-randomized treatment
xbetat   <- 0.5 + 0.5 * x[,1] - 0.5 * x[,4]
trt.prob <- exp(xbetat) / (1 + exp(xbetat))
trt01    <- rbinom(n.obs, 1, prob = trt.prob)

trt      <- 2 * trt01 - 1

# simulate response
delta <- 2 * (0.5 + x[,2] - x[,3] - x[,11] + x[,1] * x[,12])
xbeta <- x[,1] + x[,11] - 2 * x[,12]^2 + x[,13]
xbeta <- xbeta + delta * trt

# continuous outcomes
y <- drop(xbeta) + rnorm(n.obs, sd = 2)

# create function for fitting propensity score model
prop.func <- function(x, trt)
{
    # fit propensity score model
```

```
    propens.model <- cv.glmnet(y = trt,
                               x = x, family = "binomial")
    pi.x <- predict(propens.model, s = "lambda.min",
                    newx = x, type = "response")[,1]
    pi.x
}

subgrp.model <- fit.subgroup(x = x, y = y,
                             trt = trt01,
                             propensity.func = prop.func,
                             loss   = "sq_loss_lasso",
                             nfolds = 5)              # option for cv.glmnet


subgrp.model.o <- fit.subgroup(x = x, y = y,
                             trt = trt01,
                             propensity.func = prop.func,
                             loss   = "owl_logistic_flip_loss_lasso",
                             nfolds = 5)

plotCompare(subgrp.model, subgrp.model.o)
```

---

`predict.subgroup_fitted` *Function to predict either benefit scores or treatment recommendations*

---

## Description

Predicts benefit scores or treatment recommendations based on a fitted subgroup identification model

Function to obtain predictions for weighted ksvm objects

## Usage

```
## S3 method for class 'subgroup_fitted'
predict(object, newx, type = c("benefit.score",
  "trt.group"), cutpoint = 0, ...)

## S3 method for class 'wksvm'
predict(object, newx, type = c("class",
  "linear.predictor"), ...)
```

## Arguments

| | |
|---|---|
| `object` | fitted object returned by `validate.subgrp()` function.<br>For `predict.wksvm()`, this should be a fitted `wksvm` object from the `weighted.ksvm()` function |

| | |
|---|---|
| newx | new design matrix for which predictions will be made |
| type | type of prediction. type = "benefit.score" results in predicted benefit scores and type = "trt.group" results in prediction of recommended treatment group.<br>For predict.wksvm(), type = 'class' yields predicted class and type = 'linear.predictor' yields estimated function (the sign of which is the estimated class) |
| cutpoint | numeric value for patients with benefit scores above which (or below which if larger.outcome.better = FALSE) will be recommended to be in the treatment group. Can also set cutpoint = "median", which will use the median value of the benefit scores as the cutpoint or can set specific quantile values via "quantx" where "x" is a number between 0 and 100 representing the quantile value; e.g. cutpoint = "quant75" will use the 75th perent upper quantile of the benefit scores as the quantile. |
| ... | not used |

## See Also

fit.subgroup for function which fits subgroup identification models.

weighted.ksvm for fitting weighted.ksvm objects

## Examples

```
library(personalized)

set.seed(123)
n.obs  <- 1000
n.vars <- 50
x <- matrix(rnorm(n.obs * n.vars, sd = 3), n.obs, n.vars)


# simulate non-randomized treatment
xbetat   <- 0.5 + 0.5 * x[,21] - 0.5 * x[,41]
trt.prob <- exp(xbetat) / (1 + exp(xbetat))
trt01    <- rbinom(n.obs, 1, prob = trt.prob)

trt      <- 2 * trt01 - 1

# simulate response
delta <- 2 * (0.5 + x[,2] - x[,3] - x[,11] + x[,1] * x[,12])
xbeta <- x[,1] + x[,11] - 2 * x[,12]^2 + x[,13]
xbeta <- xbeta + delta * trt

# continuous outcomes
y <- drop(xbeta) + rnorm(n.obs, sd = 2)

# create function for fitting propensity score model
prop.func <- function(x, trt)
{
    # fit propensity score model
    propens.model <- cv.glmnet(y = trt,
```

```
                                x = x, family = "binomial")
    pi.x <- predict(propens.model, s = "lambda.min",
                    newx = x, type = "response")[,1]
    pi.x
}

subgrp.model <- fit.subgroup(x = x, y = y,
                             trt = trt01,
                             propensity.func = prop.func,
                             loss   = "sq_loss_lasso",
                             nfolds = 5)              # option for cv.glmnet

subgrp.model$subgroup.trt.effects
benefit.scores <- predict(subgrp.model, newx = x, type = "benefit.score")

rec.trt.grp <- predict(subgrp.model, newx = x, type = "trt.group")
```

---

`print.individual_treatment_effects` *Printing individualized treatment effects*

---

## Description

Prints results for estimated subgroup treatment effects

## Usage

```
## S3 method for class 'individual_treatment_effects'
print(x,
  digits = max(getOption("digits") - 3, 3), ...)
```

## Arguments

| | |
|---|---|
| `x` | a fitted object from either `treat.effects` or `treatment.effects` |
| `digits` | minimal number of significant digits to print. |
| `...` | further arguments passed to or from `print.default`. |

---

`print.subgroup_fitted` *Printing results for fitted subgroup identification models*

---

## Description

Prints results for estimated subgroup treatment effects

Prints summary results for estimated subgroup treatment effects

## Usage

```
## S3 method for class 'subgroup_fitted'
print(x, digits = max(getOption("digits") - 3,
  3), ...)

## S3 method for class 'subgroup_validated'
print(x, digits = max(getOption("digits") -
  3, 3), sample.pct = FALSE, which.quant = NULL, ...)

## S3 method for class 'subgroup_summary'
print(x, p.value = 0.001,
  digits = max(getOption("digits") - 3, 3), ...)
```

## Arguments

| | |
|---|---|
| x | a fitted object from either `fit.subgroup`, `validate.subgroup`, or `summarize.subgroups()` |
| digits | minimal number of significant digits to print. |
| ... | further arguments passed to or from `print.default`. |
| sample.pct | boolean variable of whether to print the percent of the test sample within each subgroup. If false the sample size itself, not the percent is printed. This may not be informative if the test sample size is much different from the total sample size |
| which.quant | when `validate.subgroup()` is called with a vector of quantile values specified for `benefit.score.quantiles`, i.e. `benefit.score.quantiles = c(0.25, 0.5, 0.75)`, the argument `which.quant` can be a vector of indexes specifying which quantile cutoff value validation results to display, i.e. `which.quant = c(1,3)` in the above example results in the display of validation results for subgroups defined by cutoff values of the benefit score defined by the 25th abnd 75th quantiles of the benefit score |
| p.value | a p-value threshold for mean differences below which covariates will be displayed. P-values are adjusted for multiple comparisons by the Hommel approach. For example, setting `p.value = 0.05` will display all covariates that have a significant difference between subgroups with p-value less than 0.05. Defaults to 0.001. |

## See Also

`validate.subgroup` for function which creates validation results and `fit.subgroup` for function which fits subgroup identification models.

`summarize.subgroups` for function which summarizes subgroup covariate values

subgroup.effects *Computes treatment effects within various subgroups*

## Description

Computes treatment effects within various subgroups to estimate subgroup treatment effects

## Usage

```
subgroup.effects(benefit.scores, y, trt, pi.x, cutpoint = 0,
  larger.outcome.better = TRUE, reference.trt = NULL)
```

## Arguments

| | |
|---|---|
| benefit.scores | vector of estimated benefit scores |
| y | The response vector |
| trt | treatment vector with each element equal to a 0 or a 1, with 1 indicating treatment status is active. |
| pi.x | The propensity score for each observation |
| cutpoint | numeric value for patients with benefit scores above which (or below which if larger.outcome.better = FALSE) will be recommended to be in the treatment group. Can also set cutpoint = "median", which will use the median value of the benefit scores as the cutpoint or can set specific quantile values via "quantx" where "x" is a number between 0 and 100 representing the quantile value; e.g. cutpoint = "quant75" will use the 75th perent upper quantile of the benefit scores as the quantile. |
| larger.outcome.better | boolean value of whether a larger outcome is better. Set to TRUE if a larger outcome is better and set to FALSE if a smaller outcome is better. Defaults to TRUE. |
| reference.trt | index of which treatment is the reference (in the case of multiple treatments). This should be known already, as for a trt with K-levels, there will be K-1 benefit scores (1 per column) of benefit.scores, where each column is a comparison of each K-1 treatments with the reference treatment. The default is the last level of trt if it is a factor. |

## See Also

fit.subgroup for function which fits subgroup identification models which generate benefit scores.

summarize.subgroups *Summarizing covariates within estimated subgroups*

## Description

Summarizes covariate values within the estimated subgroups

## Usage

```
summarize.subgroups(x, ...)

## Default S3 method:
summarize.subgroups(x, subgroup, ...)

## S3 method for class 'subgroup_fitted'
summarize.subgroups(x, ...)
```

## Arguments

| | |
|---|---|
| x | a fitted object from `fit.subgroup()` or a matrix of covariate values |
| ... | optional arguments to `summarize.subgroups` methods |
| subgroup | vector of indicators of same length as the number of rows in x if x is a matrix. A value of 1 in the ith position of `subgroup` indicates patient i is in the subgroup of patients recommended the treatment and a value of 0 in the ith position of `subgroup` indicates patient i is in the subgroup of patients recommended the control. If x is a fitted object returned by `fit.subgroup()`, `subgroup` is not needed. |

## Details

The p-values shown are raw p-values and are not adjusted for multiple comparisons.

## See Also

`fit.subgroup` for function which fits subgroup identification models and `print.subgroup_summary` for arguments for printing options for `summarize.subgroups()`.

## Examples

```
library(personalized)

set.seed(123)
n.obs  <- 1000
n.vars <- 50
x <- matrix(rnorm(n.obs * n.vars, sd = 3), n.obs, n.vars)


# simulate non-randomized treatment
```

```
xbetat   <- 0.5 + 0.5 * x[,21] - 0.5 * x[,41]
trt.prob <- exp(xbetat) / (1 + exp(xbetat))
trt01    <- rbinom(n.obs, 1, prob = trt.prob)

trt      <- 2 * trt01 - 1

# simulate response
delta <- 2 * (0.5 + x[,2] - x[,3] - x[,11] + x[,1] * x[,12])
xbeta <- x[,1] + x[,11] - 2 * x[,12]^2 + x[,13]
xbeta <- xbeta + delta * trt

# continuous outcomes
y <- drop(xbeta) + rnorm(n.obs, sd = 2)

# create function for fitting propensity score model
prop.func <- function(x, trt)
{
    # fit propensity score model
    propens.model <- cv.glmnet(y = trt,
                               x = x, family = "binomial")
    pi.x <- predict(propens.model, s = "lambda.min",
                    newx = x, type = "response")[,1]
    pi.x
}

subgrp.model <- fit.subgroup(x = x, y = y,
                             trt = trt01,
                             propensity.func = prop.func,
                             loss   = "sq_loss_lasso",
                             nfolds = 5)    # option for cv.glmnet

comp <- summarize.subgroups(subgrp.model)
print(comp, p.value = 0.01)

# or we can simply supply the matrix x and the subgroups
comp2 <- summarize.subgroups(x, subgroup = 1 * (subgrp.model$benefit.scores > 0))

print(comp2, p.value = 0.01)
```

---

`summary.subgroup_fitted` *Summary of results for fitted subgroup identification models*

---

## Description

Prints summary of results for estimated subgroup treatment effects

Prints summary of results for estimated weighted ksvm

## Usage

```
## S3 method for class 'subgroup_fitted'
summary(object,
  digits = max(getOption("digits") - 3, 3), ...)

## S3 method for class 'wksvm'
summary(object, digits = max(getOption("digits") - 3, 3),
  ...)
```

## Arguments

| | |
|---|---|
| object | a fitted object from either `fit.subgroup` or `validate.subgroup` |
| digits | minimal number of significant digits to print. |
| ... | further arguments passed to or from `print.default`. |

## See Also

`validate.subgroup` for function which creates validation results and `fit.subgroup` for function which fits subgroup identification models.

---

# treatment.effects *Calculation of covariate-conditional treatment effects*

---

## Description

Calculates covariate conditional treatment effects using estimated benefit scores

## Usage

```
treatment.effects(x, ...)

## Default S3 method:
treatment.effects(x, ...)

treat.effects(benefit.scores, loss = c("sq_loss_lasso",
  "logistic_loss_lasso", "poisson_loss_lasso", "cox_loss_lasso",
  "owl_logistic_loss_lasso", "owl_logistic_flip_loss_lasso",
  "owl_hinge_loss", "owl_hinge_flip_loss", "sq_loss_lasso_gam",
  "poisson_loss_lasso_gam", "logistic_loss_lasso_gam", "sq_loss_gam",
  "poisson_loss_gam", "logistic_loss_gam", "owl_logistic_loss_gam",
  "owl_logistic_flip_loss_gam", "owl_logistic_loss_lasso_gam",
  "owl_logistic_flip_loss_lasso_gam", "sq_loss_gbm", "poisson_loss_gbm",
  "logistic_loss_gbm", "cox_loss_gbm",      "custom"),
  method = c("weighting", "a_learning"), pi.x = NULL, ...)

## S3 method for class 'subgroup_fitted'
treatment.effects(x, ...)
```

## Arguments

| | |
|---|---|
| x | a fitted object from fit.subgroup() |
| ... | not used |
| benefit.scores | vector of estimated benefit scores |
| loss | loss choice USED TO CALCULATE benefit.scores of both the M function from Chen, et al (2017) and potentially the penalty used for variable selection. See fit.subgroup for more details. |
| method | method choice USED TO CALCULATE benefit.scores. Either the "weighting" method or "a_learning" method. See fit.subgroup for more details |
| pi.x | The propensity score for each observation |

## Value

A List with elements delta (if the treatment effects are a difference/contrast, i.e. $E[Y|T=1,X]-E[Y|T=-1,X]$) and gamma (if the treatment effects are a ratio, i.e. $E[Y|T=1,X]/E[Y|T=-1,X]$)

## See Also

fit.subgroup for function which fits subgroup identification models.

print.individual_treatment_effects for printing of objects returned by treat.effects or treatment.effects

## Examples

```
library(personalized)

set.seed(123)
n.obs  <- 1000
n.vars <- 50
x <- matrix(rnorm(n.obs * n.vars, sd = 3), n.obs, n.vars)


# simulate non-randomized treatment
xbetat   <- 0.5 + 0.5 * x[,21] - 0.5 * x[,41]
trt.prob <- exp(xbetat) / (1 + exp(xbetat))
trt01    <- rbinom(n.obs, 1, prob = trt.prob)

trt      <- 2 * trt01 - 1

# simulate response
delta <- 2 * (0.5 + x[,2] - x[,3] - x[,11] + x[,1] * x[,12])
xbeta <- x[,1] + x[,11] - 2 * x[,12]^2 + x[,13]
xbeta <- xbeta + delta * trt

# continuous outcomes
y <- drop(xbeta) + rnorm(n.obs, sd = 2)

# time-to-event outcomes
```

```
surv.time <- exp(-20 - xbeta + rnorm(n.obs, sd = 1))
cens.time <- exp(rnorm(n.obs, sd = 3))
y.time.to.event  <- pmin(surv.time, cens.time)
status           <- 1 * (surv.time <= cens.time)

# create function for fitting propensity score model
prop.func <- function(x, trt)
{
    # fit propensity score model
    propens.model <- cv.glmnet(y = trt,
                               x = x, family = "binomial")
    pi.x <- predict(propens.model, s = "lambda.min",
                    newx = x, type = "response")[,1]
    pi.x
}

subgrp.model <- fit.subgroup(x = x, y = y,
                             trt = trt01,
                             propensity.func = prop.func,
                             loss   = "sq_loss_lasso",
                             nfolds = 5)    # option for cv.glmnet

trt_eff <- treatment.effects(subgrp.model)
str(trt_eff)

trt_eff


library(survival)
subgrp.model.cox <- fit.subgroup(x = x, y = Surv(y.time.to.event, status),
                            trt = trt01,
                            propensity.func = prop.func,
                            loss   = "cox_loss_lasso",
                            nfolds = 5)              # option for cv.glmnet

trt_eff_c <- treatment.effects(subgrp.model.cox)
str(trt_eff_c)

trt_eff_c
```

---

`validate.subgroup` *Validating fitted subgroup identification models*

---

## Description

Validates subgroup treatment effects for fitted subgroup identification model class of Chen, et al (2017)

## Usage

```
validate.subgroup(model, B = 50L,
  method = c("training_test_replication", "boot_bias_correction"),
  train.fraction = 0.75, benefit.score.quantiles = c(0.1666667,
  0.3333333, 0.5, 0.6666667, 0.8333333), parallel = FALSE)
```

## Arguments

| | |
|---|---|
| `model` | fitted model object returned by `fit.subgroup()` function |
| `B` | integer. number of bootstrap replications or refitting replications. |
| `method` | validation method. `"boot_bias_correction"` for the bootstrap bias correction method of Harrell, et al (1996) or `"training_test_replication"` for repeated training and test splitting of the data (`train.fraction` should be specified for this option) |
| `train.fraction` | fraction (between 0 and 1) of samples to be used for training in training/test replication. Only used for `method = "training_test_replication"` |
| `benefit.score.quantiles` | a vector of quantiles (between 0 and 1) of the benefit score values for which to return bootstrapped information about the subgroups. ie if one of the quantile values is 0.5, the median value of the benefit scores will be used as a cutoff to determine subgroups and summary statistics will be returned about these subgroups |
| `parallel` | Should the loop over replications be parallelized? If `FALSE`, then no, if `TRUE`, then yes. If user sets `parallel = TRUE` and the fitted `fit.subgroup()` object uses the parallel version of an internal model, say for `cv.glmnet()`, then the internal parallelization will be overridden so as not to create a conflict of parallelism. |

## Details

Estimates of various quantities conditional on subgroups and treatment statuses are provided and displayed via the `print.subgroup_validated` function:

1. "Conditional expected outcomes" The first results shown when printing a `subgroup_validated` object are estimates of the expected outcomes conditional on the estimated subgroups (i.e. which subgroup is 'recommended' by the model) and conditional on treatment/intervention status. If there are two total treatment options, this results in a 2x2 table of expected conditional outcomes.
2. "Treatment effects conditional on subgroups" The second results shown when printing a `subgroup_validated` object are estimates of the expected outcomes conditional on the estimated subgroups. If the treatment takes levels $j \in \{1, \ldots, K\}$, a total of $K$ conditional treatment effects will be shown. For example, of the outcome is continuous, the $j$th conditional treatment effect is defined as $E(Y|Trt = j, Subgroup = j) - E(Y|Trt = j, Subgroup = / = j)$, where $Subgroup = j$ if treatment $j$ is recommended, i.e. treatment $j$ results in the largest/best expected potential outcomes given the fitted model.

3. "Overall treatment effect conditional on subgroups " The third quantity displayed shows the overall improvement in outcomes resulting from all treatment recommendations. This is essentially an average over all of the conditional treatment effects weighted by the proportion of the population recommended each respective treatment level.

## Value

An object of class `"subgroup_validated"`

`avg.results` Estimates of average conditional treatment effects when subgroups are determined based on the provided cutoff value for the benefit score. For example, if `cutoff = 0` and there is a treatment and control only, then the treatment is recommended if the benefit score is greater than 0.

`se.results` Standard errors of the estimates from `avg.estimates`

`boot.results` Contains the individual results for each replication. `avg.results` is comprised of averages of the values from `boot.results`

`avg.quantile.results`
Estimates of average conditional treatment effects when subgroups are determined based on different quntile cutoff values for the benefit score. For example, if `benefit.score.quantiles = 0.75` and there is a treatment and control only, then the treatment is recommended if the benefit score is greater than the 75th upper quantile of all benefit scores. If multiple quantile values are provided, e.g. `benefit.score.quantiles = c(0.15, 0.5, 0.85)`, then results will be provided for all quantile levels.

`se.quantile.results`
Standard errors corresponding to `avg.quantile.results`

`boot.results.quantiles`
Contains the individual results for each replication. `avg.quantile.results` is comprised of averages of the values from `boot.results.quantiles`

`family` Family of the outcome. For example, `"gaussian"` for continuous outcomes

`method` Method used for subgroup identification model. Weighting or A-learning

`n.trts` The number of treatment levels

`comparison.trts`
All treatment levels other than the reference level

`reference.trt` The reference level for the treatment. This should usually be the control group/level

`larger.outcome.better`
If larger outcomes are preferred for this model

`cutpoint` Benefit score cutoff value used for determining subgroups

`val.method` Method used for validation

`iterations` Number of replications used in the validation process

`nobs` Number of observations in `x` provided to `fit.subgroup`

`nvars` Number of variables in `x` provided to `fit.subgroup`

## References

Chen, S., Tian, L., Cai, T. and Yu, M. (2017), A general statistical framework for subgroup identification and comparative treatment scoring. Biometrics. doi:10.1111/biom.12676

Harrell, F. E., Lee, K. L., and Mark, D. B. (1996). Tutorial in biostatistics multivariable prognostic models: issues in developing models, evaluating assumptions and adequacy, and measuring and reducing errors. Statistics in medicine, 15, 361-387. doi:10.1002/(SICI)1097-0258(19960229)15:4<361::AID-SIM168>3.0.CO;2-4

## See Also

`fit.subgroup` for function which fits subgroup identification models, `plot.subgroup_validated` for plotting of validation results, and `print.subgroup_validated` for arguments for printing options for `validate.subgroup()`.

## Examples

```
library(personalized)

set.seed(123)
n.obs  <- 500
n.vars <- 20
x <- matrix(rnorm(n.obs * n.vars, sd = 3), n.obs, n.vars)


# simulate non-randomized treatment
xbetat   <- 0.5 + 0.5 * x[,11] - 0.5 * x[,13]
trt.prob <- exp(xbetat) / (1 + exp(xbetat))
trt01    <- rbinom(n.obs, 1, prob = trt.prob)

trt      <- 2 * trt01 - 1

# simulate response
delta <- 2 * (0.5 + x[,2] - x[,3] - x[,11] + x[,1] * x[,12])
xbeta <- x[,1] + x[,11] - 2 * x[,12]^2 + x[,13]
xbeta <- xbeta + delta * trt

# continuous outcomes
y <- drop(xbeta) + rnorm(n.obs, sd = 2)

# create function for fitting propensity score model
prop.func <- function(x, trt)
{
    # fit propensity score model
    propens.model <- cv.glmnet(y = trt,
                               x = x, family = "binomial")
    pi.x <- predict(propens.model, s = "lambda.min",
                    newx = x, type = "response")[,1]
    pi.x
}

subgrp.model <- fit.subgroup(x = x, y = y,
```

```
                              trt = trt01,
                              propensity.func = prop.func,
                              loss   = "sq_loss_lasso",
                              nfolds = 5)    # option for cv.glmnet


x.test <- matrix(rnorm(10 * n.obs * n.vars, sd = 3), 10 * n.obs, n.vars)


# simulate non-randomized treatment
xbetat.test   <- 0.5 + 0.5 * x.test[,11] - 0.5 * x.test[,13]
trt.prob.test <- exp(xbetat.test) / (1 + exp(xbetat.test))
trt01.test    <- rbinom(10 * n.obs, 1, prob = trt.prob.test)

trt.test      <- 2 * trt01.test - 1

# simulate response
delta.test <- 2 * (0.5 + x.test[,2] - x.test[,3] - x.test[,11] + x.test[,1] * x.test[,12])
xbeta.test <- x.test[,1] + x.test[,11] - 2 * x.test[,12]^2 + x.test[,13]
xbeta.test <- xbeta.test + delta.test * trt.test

y.test <- drop(xbeta.test) + rnorm(10 * n.obs, sd = 2)

valmod <- validate.subgroup(subgrp.model, B = 3,
                            method = "training_test",
                            train.fraction = 0.75)
valmod

print(valmod, which.quant = c(4, 5))
```

---

`weighted.ksvm` *Fit weighted kernel svm model.*

---

## Description

Fits weighted kernel SVM. To be used for OWL with hinge loss (but can be used more generally)

## Usage

```
weighted.ksvm(y, x, weights, C = c(0.1, 0.5, 1, 2, 10),
  kernel = "rbfdot", kpar = "automatic", nfolds = 10,
  foldid = NULL, ...)
```

## Arguments

| | |
|---|---|
| y | The response vector (either a character vector, factor vector, or numeric vector with values in -1, 1) |
| x | The design matrix (not including intercept term) |

| | |
|---|---|
| weights | vector of sample weights for weighted SVM |
| C | cost of constraints violation, see ksvm |
| kernel | kernel function used for training and prediction. See ksvm and kernels |
| kpar | list of hyperparameters for the kernel function. See ksvm |
| nfolds | number of cross validation folds for selecting value of C |
| foldid | optional vector of values between 1 and nfolds specifying which fold each observation is in. If specified, it will override the nfolds argument. |
| ... | extra arguments to be passed to ipop from the kernlab package |

## See Also

predict.wksvm for predicting from fitted weighted.ksvm objects

## Examples

```
library(kernlab)

x <- matrix(rnorm(200 * 2), ncol = 2)

y <- 2 * (sin(x[,2]) ^ 2 * exp(-x[,2]) - 0.2 > rnorm(200, sd = 0.1)) - 1

weights <- runif(100, max = 1.5, min = 0.5)

wk <- weighted.ksvm(x = x[1:100,], y = y[1:100], C = c(0.1, 0.5, 1, 2, 10),
                    weights = weights[1:100])

pr <- predict(wk, newx = x[101:200,])

mean(pr == y[101:200])
```

# Index