Package 'cdgd'

May 8, 2026

Title Causal Decomposition of Group Disparities

Version 1.0.1

Description Estimates the causal decompositions of group disparities developed by Yu and Elwert (2025) <doi:10.1214/24-AOAS1990>. For the nuisance functions of the estimators, we provide both parametric and nonparametric options, as well as manual options in case the default models are not satisfying.

License MIT + file LICENSE

URL https://github.com/ang-yu/cdgd

BugReports https://github.com/ang-yu/cdgd/issues

Depends R (>= 4.0.0)

Imports caret (>= 6.0.0)

Suggests gbm (>= 2.1.8), nnet (>= 7.3.0), ranger (>= 0.14.1)

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

NeedsCompilation no

Author Ang Yu [aut, cre, cph] (website: <https://ang-yu.github.io/>, ORCID: <https://orcid.org/0000-0002-1828-0165>)

Maintainer Ang Yu <ang_yu@outlook.com>

Repository CRAN

Date/Publication 2025-11-24 07:10:02 UTC

Contents

`cdgd0_manual` *Perform unconditional decomposition with nuisance functions estimated beforehand*

Description

This function gives the user full control over the estimation of the nuisance functions. For the unconditional decomposition, three nuisance functions (YgivenGX.Pred_D0, YgivenGX.Pred_D1, and DgivenGX.Pred) need to be estimated. The nuisance functions should be estimated using cross-fitting if Donsker class is not assumed.

Usage

```
cdgd0_manual(
  Y,
  D,
  G,
  YgivenGX.Pred_D1,
  YgivenGX.Pred_D0,
  DgivenGX.Pred,
  data,
  alpha = 0.05,
  weight = NULL
)
```

Arguments

`Y`	Outcome. The name of a numeric variable.
`D`	Treatment status. The name of a binary numeric variable taking values of 0 and 1.
`G`	Advantaged group membership. The name of a binary numeric variable taking values of 0 and 1.
`YgivenGX.Pred_D1`	A numeric vector of predicted Y values given X, G, and D=1. Vector length=nrow(data).
`YgivenGX.Pred_D0`	A numeric vector of predicted Y values given X, G, and D=0. Vector length=nrow(data).
`DgivenGX.Pred`	A numeric vector of predicted D values given X and G. Vector length=nrow(data).
`data`	A data frame.
`alpha`	1-alpha confidence interval.
`weight`	Sampling weights. The name of a numeric variable. If unspecified, equal weights are used. Technically, the weight should be a deterministic function of X and G.

Value

A list of estimates.

Examples

```
# This example will take a minute to run.

data(exp_data)

Y="outcome"
D="treatment"
G="group_a"
X=c("Q","confounder")
data=exp_data

set.seed(1)

### estimate the nuisance functions with cross-fitting
sample1 <- sample(nrow(data), floor(nrow(data)/2), replace=FALSE)
sample2 <- setdiff(1:nrow(data), sample1)

### outcome regression model

message <- utils::capture.output( YgivenDGX.Model.sample1 <-
   caret::train(stats::as.formula(paste(Y, paste(D,G,paste(X,collapse="+"),sep="+"), sep="~")),
         data=data[sample1,], method="ranger", trControl=caret::trainControl(method="cv"),
         tuneGrid=expand.grid(mtry=c(2,4),splitrule=c("variance"),min.node.size=c(50,100))) )
message <- utils::capture.output( YgivenDGX.Model.sample2 <-
   caret::train(stats::as.formula(paste(Y, paste(D,G,paste(X,collapse="+"),sep="+"), sep="~")),
         data=data[sample2,], method="ranger", trControl=caret::trainControl(method="cv"),
         tuneGrid=expand.grid(mtry=c(2,4),splitrule=c("variance"),min.node.size=c(50,100))) )

### propensity score model
data[,D] <- as.factor(data[,D])
levels(data[,D]) <- c("D0","D1")  # necessary for caret implementation of ranger

message <- utils::capture.output( DgivenGX.Model.sample1 <-
   caret::train(stats::as.formula(paste(D, paste(G,paste(X,collapse="+"),sep="+"), sep="~")),
            data=data[sample1,], method="ranger",
            trControl=caret::trainControl(method="cv", classProbs=TRUE),
         tuneGrid=expand.grid(mtry=c(1,2),splitrule=c("gini"),min.node.size=c(50,100))) )
message <- utils::capture.output( DgivenGX.Model.sample2 <-
   caret::train(stats::as.formula(paste(D, paste(G,paste(X,collapse="+"),sep="+"), sep="~")),
            data=data[sample2,], method="ranger",
            trControl=caret::trainControl(method="cv", classProbs=TRUE),
         tuneGrid=expand.grid(mtry=c(1,2),splitrule=c("gini"),min.node.size=c(50,100))) )

data[,D] <- as.numeric(data[,D])-1

### cross-fitted predictions
YgivenGX.Pred_D0 <- YgivenGX.Pred_D1 <- DgivenGX.Pred <- rep(NA, nrow(data))
```

```
pred_data <- data
pred_data[,D] <- 0
YgivenGX.Pred_D0[sample2] <- stats::predict(YgivenDGX.Model.sample1, newdata = pred_data[sample2,])
YgivenGX.Pred_D0[sample1] <- stats::predict(YgivenDGX.Model.sample2, newdata = pred_data[sample1,])

pred_data <- data
pred_data[,D] <- 1
YgivenGX.Pred_D1[sample2] <- stats::predict(YgivenDGX.Model.sample1, newdata = pred_data[sample2,])
YgivenGX.Pred_D1[sample1] <- stats::predict(YgivenDGX.Model.sample2, newdata = pred_data[sample1,])

pred_data <- data
DgivenGX.Pred[sample2] <- stats::predict(DgivenGX.Model.sample1,
    newdata = pred_data[sample2,], type="prob")[,2]
DgivenGX.Pred[sample1] <- stats::predict(DgivenGX.Model.sample2,
    newdata = pred_data[sample1,], type="prob")[,2]

results <- cdgd0_manual(Y=Y,D=D,G=G,
                        YgivenGX.Pred_D0=YgivenGX.Pred_D0,
                        YgivenGX.Pred_D1=YgivenGX.Pred_D1,
                        DgivenGX.Pred=DgivenGX.Pred,
                        data=data)

results
```

`cdgd0_ml` *Perform unconditional decomposition via machine learning*

Description

Perform unconditional decomposition via machine learning

Usage

```
cdgd0_ml(Y, D, G, X, data, algorithm, alpha = 0.05, trim = 0, weight = NULL)
```

Arguments

`Y`	Outcome. The name of a numeric variable (can be binary and take values of 0 and 1).
`D`	Treatment status. The name of a binary numeric variable taking values of 0 and 1.
`G`	Advantaged group membership. The name of a binary numeric variable taking values of 0 and 1.
`X`	Confounders. A vector of variables names.
`data`	A data frame.
`algorithm`	The ML algorithm for modelling. "nnet" for neural network, "ranger" for random forests, "gbm" for generalized boosted models.

alpha	1-alpha confidence interval.
trim	Threshold for trimming the propensity score. When trim=a, individuals with propensity scores lower than a or higher than 1-a will be dropped.
weight	Sampling weights. The name of a numeric variable. If unspecified, equal weights are used. Technically, the weight should be a deterministic function of X and G.

Value

A list of estimates.

Examples

```
# This example will take a minute to run.

data(exp_data)

set.seed(1)

results <- cdgd0_ml(
Y="outcome",
D="treatment",
G="group_a",
X=c("Q","confounder"),
data=exp_data,
algorithm="gbm")

results[[1]]
```

---

cdgd0_pa *Perform unconditional decomposition via parametric models*

Description

Perform unconditional decomposition via parametric models

Usage

```
cdgd0_pa(Y, D, G, X, data, alpha = 0.05, trim = 0, weight = NULL)
```

Arguments

Y	Outcome. The name of a numeric variable (can be binary and take values of 0 and 1).
D	Treatment status. The name of a binary numeric variable taking values of 0 and 1.
G	Advantaged group membership. The name of a binary numeric variable taking values of 0 and 1.

X	Confounders. A vector of variable names.
data	A data frame.
alpha	1-alpha confidence interval.
trim	Threshold for trimming the propensity score. When trim=a, individuals with propensity scores lower than a or higher than 1-a will be dropped.
weight	Sampling weights. The name of a numeric variable. If unspecified, equal weights are used. Technically, the weight should be a deterministic function of X and G.

Value

A list of estimates.

Examples

```
data(exp_data)

results <- cdgd0_pa(
Y="outcome",
D="treatment",
G="group_a",
X=c("Q","confounder"),
data=exp_data)

results[[1]]
```

`cdgd1_manual` *Perform conditional decomposition with nuisance functions estimated beforehand*

Description

This function gives user full control over the estimation of the nuisance functions. For the unconditional decomposition, ten nuisance functions need to be estimated. The nuisance functions should be estimated using cross-fitting if Donsker class is not assumed.

Usage

```
cdgd1_manual(
  Y,
  D,
  G,
  Q,
  YgivenGXQ.Pred_D0,
  YgivenGXQ.Pred_D1,
  DgivenGXQ.Pred,
  Y0givenQ.Pred_G0,
```

```
  Y0givenQ.Pred_G1,
  Y1givenQ.Pred_G0,
  Y1givenQ.Pred_G1,
  DgivenQ.Pred_G0,
  DgivenQ.Pred_G1,
  GgivenQ.Pred,
  data,
  alpha = 0.05,
  weight = NULL
)
```

Arguments

Y	Outcome. The name of a numeric variable.
D	Treatment status. The name of a binary numeric variable taking values of 0 and 1.
G	Advantaged group membership. The name of a binary numeric variable taking values of 0 and 1.
Q	Conditional set. A vector of variable names.
YgivenGXQ.Pred_D0	A numeric vector of predicted Y values given X, G, and D=0. Vector length=nrow(data).
YgivenGXQ.Pred_D1	A numeric vector of predicted Y values given X, G, and D=1. Vector length=nrow(data).
DgivenGXQ.Pred	A numeric vector of predicted D values given X and G. Vector length=nrow(data).
Y0givenQ.Pred_G0	A numeric vector of predicted Y(0) values given Q and G=0. Vector length=nrow(data). If there are sampling weights, this should be weighted in a specific way (see p.9 of the appendices of Yu and Elwert, 2025).
Y0givenQ.Pred_G1	A numeric vector of predicted Y(0) values given Q and G=0. Vector length=nrow(data). Same as above.
Y1givenQ.Pred_G0	A numeric vector of predicted Y(1) values given Q and G=0. Vector length=nrow(data). Same as above.
Y1givenQ.Pred_G1	A numeric vector of predicted Y(1) values given Q and G=1. Vector length=nrow(data). Same as above.
DgivenQ.Pred_G0	A numeric vector of predicted D values given Q and G=0. Vector length=nrow(data). Same as above.
DgivenQ.Pred_G1	A numeric vector of predicted D values given Q and G=0. Vector length=nrow(data). Same as above.
GgivenQ.Pred	A numeric vector of predicted G values given Q. Vector length=nrow(data).
data	A data frame.
alpha	1-alpha confidence interval.

weight — Sampling weights. The name of a numeric variable. If unspecified, equal weights are used. Technically, the weight should be a deterministic function of X only (note that this is different from the unconditional decomposition).

Value

A dataframe of estimates.

Examples

```
# This example will take a minute to run.

data(exp_data)

Y="outcome"
D="treatment"
G="group_a"
X="confounder"
Q="Q"
data=exp_data

set.seed(1)

### estimate the nuisance functions with cross-fitting
sample1 <- sample(nrow(data), floor(nrow(data)/2), replace=FALSE)
sample2 <- setdiff(1:nrow(data), sample1)

### outcome regression model

message <- utils::capture.output( YgivenDGXQ.Model.sample1 <-
 caret::train(stats::as.formula(paste(Y, paste(D,G,Q,paste(X,collapse="+"),sep="+"), sep="~")),
              data=data[sample1,], method="ranger",
              trControl=caret::trainControl(method="cv"),
         tuneGrid=expand.grid(mtry=c(2,4),splitrule=c("variance"),min.node.size=c(50,100))) )
message <- utils::capture.output( YgivenDGXQ.Model.sample2 <-
 caret::train(stats::as.formula(paste(Y, paste(D,G,Q,paste(X,collapse="+"),sep="+"), sep="~")),
              data=data[sample2,], method="ranger",
              trControl=caret::trainControl(method="cv"),
         tuneGrid=expand.grid(mtry=c(2,4),splitrule=c("variance"),min.node.size=c(50,100))) )

### propensity score model
data[,D] <- as.factor(data[,D])
levels(data[,D]) <- c("D0","D1")  # necessary for caret implementation of ranger

message <- utils::capture.output( DgivenGXQ.Model.sample1 <-
   caret::train(stats::as.formula(paste(D, paste(G,Q,paste(X,collapse="+"),sep="+"), sep="~")),
                data=data[sample1,], method="ranger",
                trControl=caret::trainControl(method="cv", classProbs=TRUE),
          tuneGrid=expand.grid(mtry=c(1,2),splitrule=c("gini"),min.node.size=c(50,100))) )
message <- utils::capture.output( DgivenGXQ.Model.sample2 <-
   caret::train(stats::as.formula(paste(D, paste(G,Q,paste(X,collapse="+"),sep="+"), sep="~")),
                data=data[sample2,], method="ranger",
```

```
                trControl=caret::trainControl(method="cv", classProbs=TRUE),
            tuneGrid=expand.grid(mtry=c(1,2),splitrule=c("gini"),min.node.size=c(50,100))) )

    data[,D] <- as.numeric(data[,D])-1

    ### cross-fitted predictions
    YgivenGXQ.Pred_D0 <- YgivenGXQ.Pred_D1 <- DgivenGXQ.Pred <- rep(NA, nrow(data))

    pred_data <- data
    pred_data[,D] <- 0
    YgivenGXQ.Pred_D0[sample2] <- stats::predict(YgivenDGXQ.Model.sample1,
        newdata = pred_data[sample2,])
    YgivenGXQ.Pred_D0[sample1] <- stats::predict(YgivenDGXQ.Model.sample2,
        newdata = pred_data[sample1,])

    pred_data <- data
    pred_data[,D] <- 1
    YgivenGXQ.Pred_D1[sample2] <- stats::predict(YgivenDGXQ.Model.sample1,
        newdata = pred_data[sample2,])
    YgivenGXQ.Pred_D1[sample1] <- stats::predict(YgivenDGXQ.Model.sample2,
        newdata = pred_data[sample1,])

    pred_data <- data
    DgivenGXQ.Pred[sample2] <- stats::predict(DgivenGXQ.Model.sample1,
        newdata = pred_data[sample2,], type="prob")[,2]
    DgivenGXQ.Pred[sample1] <- stats::predict(DgivenGXQ.Model.sample2,
        newdata = pred_data[sample1,], type="prob")[,2]

    ### Estimate E(Y_d | Q,g)
    YgivenGXQ.Pred_D1_ncf <- YgivenGXQ.Pred_D0_ncf <- DgivenGXQ.Pred_ncf <- rep(NA, nrow(data))
    # ncf stands for non-cross-fitted

    pred_data <- data
    pred_data[,D] <- 1
    YgivenGXQ.Pred_D1_ncf[sample1] <- stats::predict(YgivenDGXQ.Model.sample1,
        newdata = pred_data[sample1,])
    YgivenGXQ.Pred_D1_ncf[sample2] <- stats::predict(YgivenDGXQ.Model.sample2,
        newdata = pred_data[sample2,])

    pred_data <- data
    pred_data[,D] <- 0
    YgivenGXQ.Pred_D0_ncf[sample1] <- stats::predict(YgivenDGXQ.Model.sample1,
        newdata = pred_data[sample1,])
    YgivenGXQ.Pred_D0_ncf[sample2] <- stats::predict(YgivenDGXQ.Model.sample2,
        newdata = pred_data[sample2,])

    DgivenGXQ.Pred_ncf[sample1] <- stats::predict(DgivenGXQ.Model.sample1,
        newdata = pred_data[sample1,], type="prob")[,2]
    DgivenGXQ.Pred_ncf[sample2] <- stats::predict(DgivenGXQ.Model.sample2,
        newdata = pred_data[sample2,], type="prob")[,2]

    # IPOs for modelling E(Y_d | Q,g)
    IPO_D0_ncf <- (1-data[,D])/(1-DgivenGXQ.Pred_ncf)/mean((1-data[,D])/(1-DgivenGXQ.Pred_ncf))*
```

```
    (data[,Y]-YgivenGXQ.Pred_D0_ncf) + YgivenGXQ.Pred_D0_ncf
IPO_D1_ncf <- data[,D]/DgivenGXQ.Pred_ncf/mean(data[,D]/DgivenGXQ.Pred_ncf)*
    (data[,Y]-YgivenGXQ.Pred_D1_ncf) + YgivenGXQ.Pred_D1_ncf

data_temp <- data[,c(G,Q)]
data_temp$IPO_D0_ncf <- IPO_D0_ncf
data_temp$IPO_D1_ncf <- IPO_D1_ncf

message <- utils::capture.output( Y0givenGQ.Model.sample1 <-
    caret::train(stats::as.formula(paste("IPO_D0_ncf", paste(G,Q,sep="+"), sep="~")),
                 data=data_temp[sample1,], method="ranger",
                 trControl=caret::trainControl(method="cv"),
           tuneGrid=expand.grid(mtry=1,splitrule=c("variance"),min.node.size=c(25,50))) )
message <- utils::capture.output( Y0givenGQ.Model.sample2 <-
    caret::train(stats::as.formula(paste("IPO_D0_ncf", paste(G,Q,sep="+"), sep="~")),
                 data=data_temp[sample2,], method="ranger",
                 trControl=caret::trainControl(method="cv"),
           tuneGrid=expand.grid(mtry=1,splitrule=c("variance"),min.node.size=c(25,50))) )
message <- utils::capture.output( Y1givenGQ.Model.sample1 <-
    caret::train(stats::as.formula(paste("IPO_D1_ncf", paste(G,Q,sep="+"), sep="~")),
                 data=data_temp[sample1,], method="ranger",
                 trControl=caret::trainControl(method="cv"),
           tuneGrid=expand.grid(mtry=1,splitrule=c("variance"),min.node.size=c(25,50))) )
message <- utils::capture.output( Y1givenGQ.Model.sample2 <-
    caret::train(stats::as.formula(paste("IPO_D1_ncf", paste(G,Q,sep="+"), sep="~")),
                 data=data_temp[sample2,], method="ranger",
                 trControl=caret::trainControl(method="cv"),
           tuneGrid=expand.grid(mtry=1,splitrule=c("variance"),min.node.size=c(25,50))) )

Y0givenQ.Pred_G0 <- Y0givenQ.Pred_G1 <- Y1givenQ.Pred_G0 <- Y1givenQ.Pred_G1 <- rep(NA, nrow(data))

pred_data <- data
pred_data[,G] <- 1
# cross-fitting is used
Y0givenQ.Pred_G1[sample2] <- stats::predict(Y0givenGQ.Model.sample1, newdata = pred_data[sample2,])
Y0givenQ.Pred_G1[sample1] <- stats::predict(Y0givenGQ.Model.sample2, newdata = pred_data[sample1,])
Y1givenQ.Pred_G1[sample2] <- stats::predict(Y1givenGQ.Model.sample1, newdata = pred_data[sample2,])
Y1givenQ.Pred_G1[sample1] <- stats::predict(Y1givenGQ.Model.sample2, newdata = pred_data[sample1,])

pred_data <- data
pred_data[,G] <- 0
# cross-fitting is used
Y0givenQ.Pred_G0[sample2] <- stats::predict(Y0givenGQ.Model.sample1, newdata = pred_data[sample2,])
Y0givenQ.Pred_G0[sample1] <- stats::predict(Y0givenGQ.Model.sample2, newdata = pred_data[sample1,])
Y1givenQ.Pred_G0[sample2] <- stats::predict(Y1givenGQ.Model.sample1, newdata = pred_data[sample2,])
Y1givenQ.Pred_G0[sample1] <- stats::predict(Y1givenGQ.Model.sample2, newdata = pred_data[sample1,])

### Estimate E(D | Q,g')
data[,D] <- as.factor(data[,D])
levels(data[,D]) <- c("D0","D1")  # necessary for caret implementation of ranger

message <- utils::capture.output( DgivenGQ.Model.sample1 <-
    caret::train(stats::as.formula(paste(D, paste(G,Q,sep="+"), sep="~")),
```

```
                  data=data[sample1,], method="ranger",
                  trControl=caret::trainControl(method="cv", classProbs=TRUE),
               tuneGrid=expand.grid(mtry=1,splitrule=c("gini"),min.node.size=c(25,50))) )
    message <- utils::capture.output( DgivenGQ.Model.sample2 <-
        caret::train(stats::as.formula(paste(D, paste(G,Q,sep="+"), sep="~")),
                  data=data[sample2,], method="ranger",
                  trControl=caret::trainControl(method="cv", classProbs=TRUE),
               tuneGrid=expand.grid(mtry=1,splitrule=c("gini"),min.node.size=c(25,50))) )

    data[,D] <- as.numeric(data[,D])-1

    DgivenQ.Pred_G0 <- DgivenQ.Pred_G1 <- rep(NA, nrow(data))

    pred_data <- data
    pred_data[,G] <- 0
    DgivenQ.Pred_G0[sample2] <- stats::predict(DgivenGQ.Model.sample1,
        newdata = pred_data[sample2,], type="prob")[,2]
    DgivenQ.Pred_G0[sample1] <- stats::predict(DgivenGQ.Model.sample2,
        newdata = pred_data[sample1,], type="prob")[,2]

    pred_data <- data
    pred_data[,G] <- 1
    DgivenQ.Pred_G1[sample2] <- stats::predict(DgivenGQ.Model.sample1,
       newdata = pred_data[sample2,], type="prob")[,2]
    DgivenQ.Pred_G1[sample1] <- stats::predict(DgivenGQ.Model.sample2,
       newdata = pred_data[sample1,], type="prob")[,2]

    ### Estimate p_g(Q)=Pr(G=g | Q)
    data[,G] <- as.factor(data[,G])
    levels(data[,G]) <- c("G0","G1")  # necessary for caret implementation of ranger

    message <- utils::capture.output( GgivenQ.Model.sample1 <-
        caret::train(stats::as.formula(paste(G, paste(Q,sep="+"), sep="~")),
                  data=data[sample1,], method="ranger",
                  trControl=caret::trainControl(method="cv", classProbs=TRUE),
               tuneGrid=expand.grid(mtry=1,splitrule=c("gini"),min.node.size=c(25,50))) )
    message <- utils::capture.output( GgivenQ.Model.sample2 <-
        caret::train(stats::as.formula(paste(G, paste(Q,sep="+"), sep="~")),
                  data=data[sample2,], method="ranger",
                  trControl=caret::trainControl(method="cv", classProbs=TRUE),
               tuneGrid=expand.grid(mtry=1,splitrule=c("gini"),min.node.size=c(25,50))) )

    data[,G] <- as.numeric(data[,G])-1

    GgivenQ.Pred <- rep(NA, nrow(data))
    GgivenQ.Pred[sample2] <- stats::predict(GgivenQ.Model.sample1,
        newdata = data[sample2,], type="prob")[,2]
    GgivenQ.Pred[sample1] <- stats::predict(GgivenQ.Model.sample2,
        newdata = data[sample1,], type="prob")[,2]


    results <- cdgd1_manual(Y=Y,D=D,G=G,Q=Q,
                            YgivenGXQ.Pred_D0=YgivenGXQ.Pred_D0,
```

```
                        YgivenGXQ.Pred_D1=YgivenGXQ.Pred_D1,
                        DgivenGXQ.Pred=DgivenGXQ.Pred,
                        Y0givenQ.Pred_G0=Y0givenQ.Pred_G0,
                        Y0givenQ.Pred_G1=Y0givenQ.Pred_G1,
                        Y1givenQ.Pred_G0=Y1givenQ.Pred_G0,
                        Y1givenQ.Pred_G1=Y1givenQ.Pred_G1,
                        DgivenQ.Pred_G0=DgivenQ.Pred_G0,
                        DgivenQ.Pred_G1=DgivenQ.Pred_G1,
                        GgivenQ.Pred=GgivenQ.Pred,
                        data,alpha=0.05)

results
```

`cdgd1_ml` *Perform conditional decomposition via machine learning*

Description

Perform conditional decomposition via machine learning

Usage

```
cdgd1_ml(
  Y,
  D,
  G,
  X,
  Q,
  data,
  algorithm,
  alpha = 0.05,
  trim1 = 0,
  trim2 = 0,
  weight = NULL
)
```

Arguments

`Y`	Outcome. The name of a numeric variable (can be binary and take values of 0 and 1).
`D`	Treatment status. The name of a binary numeric variable taking values of 0 and 1.
`G`	Advantaged group membership. The name of a binary numeric variable taking values of 0 and 1.
`X`	Confounders. A vector of variables names.
`Q`	Conditional set. A vector of names of numeric variables.
`data`	A data frame.

algorithm	The ML algorithm for modelling. "nnet" for neural network, "ranger" for random forests, "gbm" for generalized boosted models.
alpha	1-alpha confidence interval.
trim1	Threshold for trimming the propensity score. When trim1=a, individuals with propensity scores lower than a or higher than 1-a will be dropped.
trim2	Threshold for trimming the G given Q predictions. When trim2=a, individuals with G given Q predictions lower than a or higher than 1-a will be dropped.
weight	Sampling weights. The name of a numeric variable. If unspecified, equal weights are used. Technically, the weight should be a deterministic function of X only (note that this is different from the unconditional decomposition).

Value

A dataframe of estimates.

Examples

```
# This example will take a minute to run.

data(exp_data)

set.seed(1)

results <- cdgd1_ml(
Y="outcome",
D="treatment",
G="group_a",
X="confounder",
Q="Q",
data=exp_data,
algorithm="gbm")

results
```

---

cdgd1_pa *Perform conditional decomposition via parametric models*

Description

Perform conditional decomposition via parametric models

Usage

```
cdgd1_pa(
  Y,
  D,
  G,
```

```
  X,
  Q,
  data,
  alpha = 0.05,
  trim1 = 0,
  trim2 = 0,
  weight = NULL
)
```

Arguments

Y	Outcome. The name of a numeric variable (can be binary and take values of 0 and 1).
D	Treatment status. The name of a binary numeric variable taking values of 0 and 1.
G	Advantaged group membership. The name of a binary numeric variable taking values of 0 and 1.
X	Confounders. A vector of variable names.
Q	Conditional set. A vector of variable names.
data	A data frame.
alpha	1-alpha confidence interval.
trim1	Threshold for trimming the propensity score. When trim1=a, individuals with propensity scores lower than a or higher than 1-a will be dropped.
trim2	Threshold for trimming the G given Q predictions. When trim2=a, individuals with G given Q predictions lower than a or higher than 1-a will be dropped.
weight	Sampling weights. The name of a numeric variable. If unspecified, equal weights are used. Technically, the weight should be a deterministic function of X only (note that this is different from the unconditional decomposition).

Value

A dataframe of estimates.

Examples

```
data(exp_data)

results <- cdgd1_pa(
Y="outcome",
D="treatment",
G="group_a",
X="confounder",
Q="Q",
data=exp_data)

results
```

exp_data *Simulated example data*

Description

Simulated example data

Usage

```
data(exp_data)
```

Format

An object of class `data.frame` with 1000 rows and 5 columns.

Index