Package ‘OptHoldoutSize’

December 20, 2021

Title Estimation of Optimal Size for a Holdout Set for Updating a Predictive Score

Version 0.0.0.9000

Author James Liley, Sam Emerson, Sami Haidar-Wehbe, and Louis Aslett

Maintainer James Liley <james.liley@durham.ac.uk>

Description Predictive scores must be updated with care, because actions taken on the basis of existing risk scores causes bias in risk estimates from the updated score. A holdout set is a straightforward way to manage this problem: a proportion of the population is 'held-out' from computation of the previous risk score. This package provides tools to estimate a size for this holdout set and associated errors. Comprehensive vignettes are included.

License GPL (>=3)

Encoding UTF-8

LazyData true

Depends R (>= 3.5.0), matrixStats, mnormt, mvtnorm, ranger, mle.tools

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.2

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

R topics documented:

add_aspre_interactions 2
aspre 3
aspre_emulation 4
aspre_k2 4
aspre_parametric 5
ci_cover_a_yn 5
ci_cover_e_yn 6
ci_ohs 6
cov_fn 9
data_example_simulation 10
data_nextpoint_em 10
data_nextpoint_par 11
error_ohs_emulation 11

exp_imp_fn . . . 13
gen_base_coefs . . . 15
gen_preds . . . 16
gen_resp . . . 16
grad_nstar_powerlaw . . . 17
logistic . . . 18
logit . . . 18
model_predict . . . 19
model_train . . . 20
mu_fn . . . 21
next_n . . . 23
ohs_array . . . 25
ohs_resample . . . 25
optimal_holdout_size . . . 26
optimal_holdout_size_emulation . . . 27
oracle_pred . . . 28
params_aspre . . . 29
plot.optholdoutsize . . . 29
plot.optholdoutsize_emul . . . 30
powersolve . . . 31
powersolve_general . . . 32
powersolve_se . . . 32
psi_fn . . . 34
sens10 . . . 35
sim_random_aspre . . . 36
split_data . . . 37

Index 38

`add_aspre_interactions` *Add interaction terms corresponding to ASPRE model*

Description

Add various interaction terms to X. Interaction terms correspond to those in ASPRE.

Usage

```
add_aspre_interactions(X)
```

Arguments

`X`	data frame

Value

New data frame containing interaction terms.

Examples

```
# Load ASPRE related data
data(params_aspre)

X=sim_random_aspre(1000,params_aspre)
Xnew=add_aspre_interactions(X)

print(colnames(X))
print(colnames(Xnew))
```

`aspre` *Computes ASPRE score*

Description

Computes ASPRE model prediction on a matrix `X` of covariates

Full ASPRE model from https://www.nejm.org/doi/suppl/10.1056/NEJMoa1704559/suppl_file/nejmoa1704559_append

Model is to predict gestational age at PE; that is, a higher score indicates a lower PE risk, so coefficients are negated for model to predict PE risk.

Usage

```
aspre(X)
```

Arguments

`X`	matrix, assumed to be output of sim_random_aspre with parameter params=params_aspre and transformed using add_aspre_interactions

Value

vector of scores.

Examples

```
# Load ASPRE related data
data(params_aspre)

X=sim_random_aspre(1000,params_aspre)
Xnew=add_aspre_interactions(X)

aspre_score=aspre(Xnew)

plot(density(aspre_score))
```

`aspre_emulation` *Emulation-based OHS estimation for ASPRE*

Description

This object contains data relating to emulation-based OHS estimation for the ASPRE model. For generation, see hidden code in vignette, or in pipeline at https://github.com/jamesliley/OptHoldoutSize_pipelines

Usage

```
aspre_emulation
```

Format

An object of class `list` of length 4.

`aspre_k2` *Cost estimating function in ASPRE simulation*

Description

Estimate cost at a given holdout set size in ASPRE model

Usage

```
aspre_k2(
  n,
  X,
  PRE,
  seed = NULL,
  pi_PRE = 1426/58974,
  pi_intervention = 0.1,
  alpha = 0.37
)
```

Arguments

`n`	Holdout set size at which to estimate k_2 (cost)
`X`	Matrix of predictors
`PRE`	Vector indicating PRE incidence
`seed`	Random seed; set before starting or set to NULL
`pi_PRE`	Population prevalence of PRE if not prophylactically treated. Defaults to empirical value 1426/58974
`pi_intervention`	Proportion of the population on which an intervention will be made. Defaults to 0.1
`alpha`	Reduction in PRE risk with intervention. Defaults to empirical value 0.37

Value

Estimated cost

Examples

```
# Simulate
set.seed(32142)

N=1000; p=15
X=matrix(rnorm(N*p),N,p); PRE=rbinom(N,1,prob=logit(X%*% rnorm(p)))
aspre_k2(1000,X,PRE)
```

`aspre_parametric` *Parametric-based OHS estimation for ASPRE*

Description

This object contains data relating to parametric-based OHS estimation for the ASPRE model. For generation, see hidden code in vignette, or in pipeline at https://github.com/jamesliley/OptHoldoutSize_pipelines

Usage

```
aspre_parametric
```

Format

An object of class `list` of length 4.

`ci_cover_a_yn` *Data for example on asymptotic confidence interval for CI.*

Description

Data for example for asymptotic confidence interval for CI. For generation, see example.

Usage

```
ci_cover_a_yn
```

Format

An object of class `matrix` (inherits from `array`) with 11 rows and 5000 columns.

`ci_cover_e_yn` *Data for example on empirical confidence interval for CI.*

Description

Data for example for empirical confidence interval for CI. For generation, see example.

Usage

```
ci_cover_e_yn
```

Format

An object of class `matrix` (inherits from `array`) with 11 rows and 5000 columns.

`ci_ohs` *Confidence interval for optimal holdout size, when estimated using parametric method*

Description

Compute confidence interval for optimal holdout size given either a standard error covariance matrix or a set of n_e estimates of parameters.

This can be done either asymptotically, using a method analogous to the Fisher information matrix, or empirically (using bootstrap resampling)

If sigma (covariance matrix) is specified and method='bootstrap', a confidence interval is generated assuming a Gaussian distribution of (N,k1,theta). To estimate a confidence interval assuming a non-Gaussian distribution, simulate values under the requisite distribution and use then as parameters N,k1, theta, with sigma set to NULL.

Usage

```
ci_ohs(
  N,
  k1,
  theta,
  alpha = 0.05,
  k2 = powerlaw,
  grad_nstar = NULL,
  sigma = NULL,
  n_boot = 1000,
  seed = NULL,
  mode = "empirical",
  ...
)
```

Arguments

`N`	Vector of estimates of total number of samples on which the predictive score will be used/fitted, or single estimate
`k1`	Vector of estimates of cost value in the absence of a predictive score, or single number
`theta`	Matrix of estimates of parameters for function k2(n) governing expected cost to an individual sample given a predictive score fitted to n samples. Can be a matrix of dimension n x n_par, where n_par is the number of parameters of k2.
`alpha`	Construct 1-alpha confidence interval. Defaults to 0.05
`k2`	Function governing expected cost to an individual sample given a predictive score fitted to n samples. Must take two arguments: n (number of samples) and theta (parameters). Defaults to a power-law form k2(n,c(a,b,c))=a n^(-b) + c.
`grad_nstar`	Function giving partial derivatives of optimal holdout set, taking three arguments: N, k1, and theta. Only used for asymptotic confidence intervals. F NULL, estimated empirically
`sigma`	Standard error covariance matrix for (N,k1,theta), in that order. If NULL, will derive as sample covariance matrix of parameters. Must be of the correct size and positive definite.
`n_boot`	Number of bootstrap resamples for empirical estimate.
`seed`	Random seed for bootstrap resamples. Defaults to NULL.
`mode`	One of 'asymptotic' or 'empirical'. Defaults to 'empirical'
`...`	Passed to function `optimize`

Value

A vector of length two containing lower and upper limits of confidence interval.

Examples

```
## We will assume that our observations of N, k1, and theta=(a,b,c) are distributed with mean mu_par and variance
mu_par=c(N=10000,k1=0.35,A=3,B=1.5,C=0.1)
sigma_par=cbind(
  c(100^2,       1,      0,       0,       0),
  c(    1,  0.07^2,      0,       0,       0),
  c(    0,       0,  0.5^2,    0.05,  -0.001),
  c(    0,       0,   0.05,   0.4^2,  -0.002),
  c(    0,       0, -0.001,  -0.002,  0.02^2)
)

# Firstly, we make 500 observations
par_obs=rmnorm(500,mean=mu_par,varcov=sigma_par)

# Optimal holdout size and asymptotic and empirical confidence intervals
ohs=optimal_holdout_size(N=mean(par_obs[,1]),k1=mean(par_obs[,2]),theta=colMeans(par_obs[,3:5]))$size
ci_a=ci_ohs(N=par_obs[,1],k1=par_obs[,2],theta=par_obs[,3:5],alpha=0.05,seed=12345,mode="asymptotic")
ci_e=ci_ohs(N=par_obs[,1],k1=par_obs[,2],theta=par_obs[,3:5],alpha=0.05,seed=12345,mode="empirical")


# Assess cover at various n_e
n_e_values=c(20,30,50,100,150,200,300,500,750,1000,1500)
ntrial=5000
```

```
alpha_trial=0.1 # use 90% confidence intervals
nstar_true=optimal_holdout_size(N=mu_par[1],k1=mu_par[2],theta=mu_par[3:5])$size

## The matrices indicating cover take are included in this package but take around 30 minutes to generate. They a
data(ci_cover_a_yn)
data(ci_cover_e_yn)

if (!exists("ci_cover_a_yn")) {
 ci_cover_a_yn=matrix(NA,length(n_e_values),ntrial) # Entry [i,j] is 1 if ith asymptotic CI for jth value of n_
 ci_cover_e_yn=matrix(NA,length(n_e_values),ntrial) # Entry [i,j] is 1 if ith empirical CI for jth value of n_e

  for (i in 1:length(n_e_values)) {
    n_e=n_e_values[i]
    for (j in 1:ntrial) {
      # Set seed
      set.seed(j*ntrial + i + 12345)

      # Make n_e observations
      par_obs=rmnorm(n_e,mean=mu_par,varcov=sigma_par)
    ci_a=ci_ohs(N=par_obs[,1],k1=par_obs[,2],theta=par_obs[,3:5],alpha=alpha_trial,mode="asymptotic")
    ci_e=ci_ohs(N=par_obs[,1],k1=par_obs[,2],theta=par_obs[,3:5],alpha=alpha_trial,mode="empirical",n_boot=

    if (nstar_true>ci_a[1] & nstar_true<ci_a[2]) ci_cover_a_yn[i,j]=1 else ci_cover_a_yn[i,j]=0
    if (nstar_true>ci_e[1] & nstar_true<ci_e[2]) ci_cover_e_yn[i,j]=1 else ci_cover_e_yn[i,j]=0
    }
    print(paste0("Completed for n_e = ",n_e))
  }

}

# Cover at each n_e value and standard error
cover_a=rowMeans(ci_cover_a_yn)
cover_e=rowMeans(ci_cover_e_yn)
zse_a=2*sqrt(cover_a*(1-cover_a)/ntrial)
zse_e=2*sqrt(cover_e*(1-cover_e)/ntrial)


# Draw plot. Convergence to 1-alpha cover is evident. Cover is not far from alpha even at small n_e.

plot(0,type="n",xlim=range(n_e_values),ylim=c(0.7,1),xlab=expression("n"[e]),ylab="Cover")

# Asymptotic cover and 2*SE pointwise envelope
polygon(c(n_e_values,rev(n_e_values)),c(cover_a+zse_a,rev(cover_a-zse_a)),
  col=rgb(1,1,1,alpha=0.3),border=NA)
lines(n_e_values,cover_a,col="black")

# Empirical cover and 2*SE pointwiseenvelope
polygon(c(n_e_values,rev(n_e_values)),c(cover_e+zse_e,rev(cover_e-zse_e)),
  col=rgb(0,0,1,alpha=0.3),border=NA)
lines(n_e_values,cover_e,col="blue")

abline(h=1-alpha_trial,col="red")
legend("bottomright",c("Asym.","Emp.",expression(paste("1-",alpha))),lty=1,col=c("black","blue","red"))
```

`cov_fn` *Covariance function for Gaussian process*

Description

Radial kernel covariance function for Gaussian process.

Used for a Gaussian process `GP(m,k(.,.))`, an instance X of which has covariance k(n,n') between X(n) and X(n').

Covariance function is parametrised by `var_u` (general variance) and `k_width` (kernel width)

Usage

```
cov_fn(n, ndash, var_u, k_width)
```

Arguments

`n`	Argument 1: kernel is a function of `ndash-n`
`ndash`	Argument 2: kernel is a function of `ndash-n`
`var_u`	Global variance
`k_width`	Kernel width

Value

Covariance value

Examples

```
##' # We will sample from Gaussian processes GP(0,k(.,.)=cov_fn(.,.;var_u,theta)) at these values of n
nvals=seq(1,300,length=100)

# We will consider two theta values
kw1=10; kw2=30

# We will consider two var_u values
var1=1; var2=10

# Covariance matrices
cov11=outer(nvals,nvals,function(n,ndash) cov_fn(n,ndash,var_u=var1,k_width=kw1))
cov12=outer(nvals,nvals,function(n,ndash) cov_fn(n,ndash,var_u=var1,k_width=kw2))
cov21=outer(nvals,nvals,function(n,ndash) cov_fn(n,ndash,var_u=var2,k_width=kw1))
cov22=outer(nvals,nvals,function(n,ndash) cov_fn(n,ndash,var_u=var2,k_width=kw2))

# Dampen slightly to ensure positive definiteness
damp=1e-5
cov11=(1-damp)*(1-diag(length(nvals)))*cov11 + diag(length(nvals))*cov11
cov12=(1-damp)*(1-diag(length(nvals)))*cov12 + diag(length(nvals))*cov12
cov21=(1-damp)*(1-diag(length(nvals)))*cov21 + diag(length(nvals))*cov21
cov22=(1-damp)*(1-diag(length(nvals)))*cov22 + diag(length(nvals))*cov22

# Sample
set.seed(35243)
y11=rmnorm(1,mean=0,varcov=cov11)
```

```
y12=rmnorm(1,mean=0,varcov=cov12)
y21=rmnorm(1,mean=0,varcov=cov21)
y22=rmnorm(1,mean=0,varcov=cov22)

# Plot
rr=max(abs(c(y11,y12,y21,y22)))
plot(0,xlim=range(nvals),ylim=c(-rr,rr+10),xlab="n",ylab=expression("GP(0,cov_fn(.,.;var_u,theta))"))
lines(nvals,y11,lty=1,col="black")
lines(nvals,y12,lty=2,col="black")
lines(nvals,y21,lty=1,col="red")
lines(nvals,y22,lty=2,col="red")
legend("topright",c("k_width=10, var_u=1", "k_width=30, var_u=1", "k_width=10, var_u=10","k_width=30, var_u=1
  lty=c(1,2,1,2),col=c("black","black","red","red"))
```

data_example_simulation *Data for vignette showing general example*

Description

Data for general vignette. For generation, see hidden code in vignette, or in pipeline at https://github.com/jamesliley/OptH

Usage

```
data_example_simulation
```

Format

An object of class `list` of length 4.

data_nextpoint_em *Data for 'next point' demonstration vignette on algorithm comparison using emulation algorithm*

Description

Data containing 'next point selected' information for emulation algorithm in vignette comparing emulation and parametric algorithms. For generation, see hidden code in vignette, or in pipeline at https://github.com/jamesliley/OptHoldoutSize_pipelines

Usage

```
data_nextpoint_em
```

Format

An object of class `list` of length 6.

`data_nextpoint_par` *Data for 'next point' demonstration vignette on algorithm comparison using parametric algorithm*

Description

Data containing 'next point selected' information for parametric algorithm in vignette comparing emulation and parametric algorithms. For generation, see hidden code in vignette, or in pipeline at https://github.com/jamesliley/OptHoldoutSize_pipelines

Usage

```
data_nextpoint_par
```

Format

An object of class `list` of length 6.

`error_ohs_emulation` *Measure of error for emulation-based OHS emulation*

Description

Measure of error for semiparametric (emulation) based estimation of optimal holdout set sizes.

Returns a set of values of n for which a `1-alpha` credible interval for cost at includes a lower value than the cost at the estimated optimal holdout size.

This is not a confidence interval, credible interval or credible set for the OHS, and is prone to misinterpretation.

Usage

```
error_ohs_emulation(
  nset,
  d,
  var_w,
  N,
  k1,
  alpha = 0.1,
  var_u = 1e+07,
  k_width = 5000,
  mean_fn = powerlaw,
  theta = powersolve(nset, d, y_var = var_w)$par,
  npoll = 1000
)
```

Arguments

nset	Training set sizes for which a loss has been evaluated
d	Loss at training set sizes nset
var_w	Variance of error in loss estimate at each training set size.
N	Total number of samples on which the model will be fitted/used
k1	Mean loss per sample with no predictive score in place
alpha	Use 1-alpha credible interval. Defaults to 0.1.
var_u	Marginal variance for Gaussian process kernel. Defaults to 1e7
k_width	Kernel width for Gaussian process kernel. Defaults to 5000
mean_fn	Functional form governing expected loss per sample given sample size. Should take two parameters: n (sample size) and theta (parameters). Defaults to function powerlaw.
theta	Current estimates of parameter values for mean_fn. Defaults to the MLE power-law solution corresponding to n,d, and var_w.
npoll	Check npoll equally spaced values between 1 and N for minimum. If NULL, check all values (this can be slow). Defaults to 1000

Value

Vector of values n for which 1-alpha credible interval for cost l(n) at n contains mean posterior loss at estimated optimal holdout size.

Examples

```
 # Set seed
set.seed(57365)

# Parameters
N=100000;
k1=0.3
A=8000; B=1.5; C=0.15; theta=c(A,B,C)

# True mean function
k2_true=function(n) powerlaw(n,theta)

# True OHS
nx=1:N
ohs_true=nx[which.min(k1*nx + k2_true(nx)*(N-nx))]

# Values of n for which cost has been estimated
np=50 # this many points
nset=round(runif(np,1,N))
var_w=runif(np,0.001,0.0015)
d=rnorm(np,mean=k2_true(nset),sd=sqrt(var_w))

# Compute OHS
res1=optimal_holdout_size_emulation(nset,d,var_w,N,k1)

# Error estimates
ex=error_ohs_emulation(nset,d,var_w,N,k1)

# Plot
```

```
plot(res1)

# Add error
abline(v=ohs_true)
abline(v=ex,col=rgb(1,0,0,alpha=0.2))

# Show justification for error
n=seq(1,N,length=1000)
mu=mu_fn(n,nset,d,var_w,N,k1); psi=pmax(0,psi_fn(n, nset, var_w, N)); Z=-qnorm(0.1/2)
lines(n,mu - Z*sqrt(psi),lty=2,lwd=2)
legend("topright",
    c("Err. region",expression(paste(mu(n)- "z"[alpha/2]*sqrt(psi(n))))),
    pch=c(16,NA),lty=c(NA,2),lwd=c(NA,2),col=c("pink","black"),bty="n")
```

`exp_imp_fn` *Expected improvement*

Description

Expected improvement

Essentially chooses the next point to add to `n`, called n*, in order to minimise the expectation of loss(n*).

Usage

```
exp_imp_fn(
  n,
  nset,
  d,
  var_w,
  N,
  k1,
  var_u = 1e+07,
  k_width = 5000,
  mean_fn = powerlaw,
  theta = powersolve(nset, d, y_var = var_w)$par
)
```

Arguments

`n`	Set of training set sizes to evaluate
`nset`	Training set sizes for which a loss has been evaluated
`d`	Loss at training set sizes `nset`
`var_w`	Variance of error in loss estimate at each training set size.
`N`	Total number of samples on which the model will be fitted/used
`k1`	Mean loss per sample with no predictive score in place
`var_u`	Marginal variance for Gaussian process kernel. Defaults to 1e7
`k_width`	Kernel width for Gaussian process kernel. Defaults to 5000

mean_fn Functional form governing expected loss per sample given sample size. Should take two parameters: n (sample size) and theta (parameters). Defaults to function powerlaw.

theta Current estimates of parameter values for mean_fn. Defaults to the MLE power-law solution corresponding to n,d, and var_w.

Value

Value of expected improvement at values n

Examples

```
# Set seed.
set.seed(24015)

# Kernel width and Gaussian process variance
kw0=5000
vu0=1e7

# Include legend on plots or not; inclusion can obscure plot elements on small figures
inc_legend=FALSE

# Suppose we have population size and cost-per-sample without a risk score as follows
N=100000
k1=0.4

# Suppose that true values of a,b,c are given by
theta_true=c(10000,1.2,0.2)
theta_lower=c(1,0.5,0.1) # lower bounds for estimating theta
theta_upper=c(20000,2,0.5) # upper bounds for estimating theta



# We start with five random holdout set sizes (nset0),
#  with corresponding cost-per-individual estimates d0 derived
#  with various errors var_w0
nstart=4
vwmin=0.001; vwmax=0.005
nset0=round(runif(nstart,1000,N/2))
var_w0=runif(nstart,vwmin,vwmax)
d0=rnorm(nstart,mean=powerlaw(nset0,theta_true),sd=sqrt(var_w0))

# We estimate theta from these three points
theta0=powersolve(nset0,d0,y_var=var_w0,lower=theta_lower,upper=theta_upper,init=theta_true)$par

# We will estimate the posterior at these values of n
n=seq(1000,N,length=1000)

# Mean and variance
p_mu=mu_fn(n,nset=nset0,d=d0,var_w = var_w0, N=N,k1=k1,theta=theta0,k_width=kw0,var_u=vu0)
p_var=psi_fn(n,nset=nset0,N=N,var_w = var_w0,k_width=kw0,var_u=vu0)

# Plot
yrange=c(-30000,100000)
plot(0,xlim=range(n),ylim=yrange,type="n",
  xlab="Training/holdout set size",
```

```
  ylab="Total cost (= num. cases)")
lines(n,p_mu,col="blue")
lines(n,p_mu - 3*sqrt(p_var),col="red")
lines(n,p_mu + 3*sqrt(p_var),col="red")
points(nset0,k1*nset0 + d0*(N-nset0),pch=16,col="purple")
lines(n,k1*n + powerlaw(n,theta0)*(N-n),lty=2)
lines(n,k1*n + powerlaw(n,theta_true)*(N-n),lty=3,lwd=3)
if (inc_legend) {
  legend("topright",
    c(expression(mu(n)),
      expression(mu(n) %+-% 3*sqrt(psi(n))),
      "prior(n)",
      "True",
      "d"),
    lty=c(1,1,2,3,NA),lwd=c(1,1,1,3,NA),pch=c(NA,NA,NA,NA,16),pt.cex=c(NA,NA,NA,NA,1),
    col=c("blue","red","black","purple"),bg="white")
}

## Add line corresponding to recommended new point
exp_imp_em <- exp_imp_fn(n,nset=nset0,d=d0,var_w = var_w0, N=N,k1=k1,theta=theta0,k_width=kw0,var_u=vu0)
abline(v=n[which.max(exp_imp_em)])
```

`gen_base_coefs` *Coefficients for imperfect risk score*

Description

Generate coefficients corresponding to an imperfect risk score, interpretable as 'baseline' behaviour in the absence of a risk score

Usage

```
gen_base_coefs(coefs, noise = TRUE, num_vars = 2, max_base_powers = 1)
```

Arguments

`coefs`	Original coefficients
`noise`	Set to TRUE to add Gaussian noise to coefficients
`num_vars`	Number of variables at hand for baseline calculation
`max_base_powers`	If >1, return a matrix of coefficients, with successively more noise

Value

Vector of coefficients

Examples

```
# See examples for model_predict
```

`gen_preds` *Generate matrix of random observations*

Description

Generate matrix of random observations. Observations are unit Gaussian-distributed.

Usage

```
gen_preds(nobs, npreds, ninters = 0)
```

Arguments

`nobs`	Number of observations (samples)
`npreds`	Number of predictors
`ninters`	Number of interaction terms, default 0. Up to npreds*(npreds-1)/2

Value

Data frame of observations

Examples

```
# See examples for model_predict
```

`gen_resp` *Generate response*

Description

Generate random outcome (response) according to a ground-truth logistic model

Usage

```
gen_resp(X, coefs = NA, coefs_sd = 1, retprobs = FALSE)
```

Arguments

`X`	Matrix of observations
`coefs`	Vector of coefficients for logistic model. If NA, random coefficients are generated. Defaults to NA
`coefs_sd`	If random coefficients are generated, use this SD (mean 0)
`retprobs`	If TRUE, return class probability; otherwise, return classes. Defaults to FALSE

Value

Vector of length as first dimension of dim(X) with outcome classes (if retprobs==FALSE) or outcome probabilities (if retprobs==TRUE)

Examples

```
# See examples for model_predict
```

grad_nstar_powerlaw *Gradient of optimal holdout size (power law)*

Description

Compute gradient of optimal holdout size assuming a power-law form of k2

Assumes cost function is l(n;k1,N,theta) = k1 n + k2(n;theta) (N-n) with k2(n;theta)=k2(n;a,b,c)= a n^(-b) + c

Usage

```
grad_nstar_powerlaw(N, k1, theta)
```

Arguments

N	Total number of samples on which the predictive score will be used/fitted. Can be a vector.
k1	Cost value in the absence of a predictive score. Can be a vector.
theta	Parameters for function k2(n) governing expected cost to an individual sample given a predictive score fitted to n samples. Can be a matrix of dimension n x n_par, where n_par is the number of parameters of k2.

Value

List/data frame of dimension (number of evaluations) x 5 containing partial derivatives of nstar (optimal holdout size) with respect to N, k1, a, b, c respectively.

Examples

```
# Evaluate optimal holdout set size for a range of values of k1, and compute derivative
N=10000;
k1=seq(0.1,0.5,length=20)
A=3; B=1.5; C=0.15; theta=c(A,B,C)

nstar=optimal_holdout_size(N,k1,theta)
grad_nstar=grad_nstar_powerlaw(N,k1,theta)

plot(0,type="n",ylim=c(-2000,500),xlim=range(k1),xlab=expression("k"[1]),ylab="Optimal holdout set size")
lines(nstar$k1,nstar$size,col="black")
lines(nstar$k1,grad_nstar[,2],col="red")
legend("bottomright",c(expression("n"["*"]),expression(paste(partialdiff[k1],"n"["*"]))),
    col=c("black","red"),lty=1)
```

logistic — *Logistic*

Description

Logistic function: -log((1/x)-1)

Usage

```
logistic(x)
```

Arguments

x	argument

Value

value of logit(x); na if x is outside (0,1)

Examples

```
# Plot
x=seq(0,1,length=100)
plot(x,logistic(x),type="l")

# Logit and logistic are inverses
x=seq(-5,5,length=1000)
plot(x,logistic(logit(x)),type="l")
```

logit — *Logit*

Description

Logit function: 1/(1+exp(-x))

Usage

```
logit(x)
```

Arguments

x	argument

Value

value of logit(x)

Examples

```
# Plot
x=seq(-5,5,length=1000)
plot(x,logit(x),type="l")
```

model_predict *Make predictions*

Description

Make predictions according to a given model

Usage

```
model_predict(
  data_test,
  trained_model,
  return_type,
  threshold = NULL,
  model_family = NULL,
  ...
)
```

Arguments

data_test	Data for which predictions are to be computed
trained_model	Model for which predictions are to be made
return_type	??
threshold	??
model_family	??
...	Passed to function predict.glm() or predict.ranger()

Value

Vector of predictions

Examples

```
## Set seed for reproducibility
seed=1234
set.seed(seed)

# Initialisation of patient data
n_iter <- 500           # Number of point estimates to be calculated
nobs <- 5000            # Number of observations, i.e patients
npreds <- 7             # Number of predictors

# Model family
family="log_reg"

# Baseline behaviour is an oracle Bayes-optimal predictor on only one variable
max_base_powers <- 1
base_vars=1

# Check the following holdout size fractions
frac_ho = 0.1
```

```
# Set ground truth coefficients, and the accuracy at baseline
coefs_general <- rnorm(npreds,sd=1/sqrt(npreds))
coefs_base <- gen_base_coefs(coefs_general, max_base_powers = max_base_powers)

# Generate dataset
X <- gen_preds(nobs, npreds)

# Generate labels
newdata <- gen_resp(X, coefs = coefs_general)
Y <- newdata$classes

# Combined dataset
pat_data <- cbind(X, Y)
pat_data$Y = factor(pat_data$Y)

# For each holdout size, split data into intervention and holdout set
mask <- split_data(pat_data, frac_ho)
data_interv <- pat_data[!mask,]
data_hold <- pat_data[mask,]

# Train model
trained_model <- model_train(data_hold, model_family = family)
thresh <- 0.5

# Make predictions
class_pred <- model_predict(data_interv, trained_model,
                            return_type = "class",
                            threshold = 0.5, model_family = family)


# Simulate baseline predictions
base_pred <- oracle_pred(data_hold,coefs_base[base_vars, ], num_vars = base_vars)


# Contingency table for model-based predictor (on intervention set)
print(table(class_pred,data_interv$Y))

# Contingency table for model-based predictor (on holdout set)
print(table(base_pred,data_hold$Y))
```

`model_train` *Train model (wrapper)*

Description

Train model using either a GLM or a random forest

Usage

```
model_train(train_data, model_family = "log_reg", ...)
```

Arguments

`train_data`	Data to use for training; assumed to have one binary column called `Y`
`model_family`	Either 'log_reg' for logistic regression or 'rand_forest' for random forest
`...`	Passed to function `glm()` or `ranger()`

Value

Fitted model of type GLM or Ranger

Examples

```
# See examples for model_predict
```

`mu_fn` *Power law function*

Description

Power law function for modelling learning curve (taken to mean change in expected loss per sample with training set size)

Recommended in review of learning curve forms

If `theta=c(a,b,c)` then models as a n^(-b) + c. Note `b` is negated.

Note that `powerlaw(n,c(a,b,c))` has limit `c` as `n` tends to infinity, if a,b > 0

Posterior mean for emulator given points `n`.

Usage

```
powerlaw(n, theta)

mu_fn(
  n,
  nset,
  d,
  var_w,
  N,
  k1,
  var_u = 1e+07,
  k_width = 5000,
  mean_fn = powerlaw,
  theta = powersolve(nset, d, y_var = var_w)$par
)
```

Arguments

`n`	Set of training set sizes to evaluate
`theta`	Current estimates of parameter values for mean_fn. Defaults to the MLE power-law solution corresponding to n,d, and var_w.
`nset`	Training set sizes for which a loss has been evaluated

d	Loss at training set sizes nset
var_w	Variance of error in loss estimate at each training set size.
N	Total number of samples on which the model will be fitted/used
k1	Mean loss per sample with no predictive score in place
var_u	Marginal variance for Gaussian process kernel. Defaults to 1e7
k_width	Kernel width for Gaussian process kernel. Defaults to 5000
mean_fn	Functional form governing expected loss per sample given sample size. Should take two parameters: n (sample size) and theta (parameters). Defaults to function powerlaw.

Value

Vector of values of same length as n

Vector Mu of same length of n where Mu_i=mean(posterior(cost(n_i)))

Examples

```
ncheck=seq(1000,10000)
plot(ncheck, powerlaw(ncheck, c(5e3,1.2,0.3)),type="l",xlab="n",ylab="powerlaw(n)")


# Suppose we have population size and cost-per-sample without a risk score as follows
N=100000
k1=0.4

# Kernel width and variance for GP
k_width=5000
var_u=8000000

# Suppose we begin with loss estimates at n-values
nset=c(10000,20000,30000)

# with cost-per-individual estimates
k2=c(0.35,0.26,0.28)

# and associated error on those estimates
var_w=c(0.02^2,0.01^2,0.03^2)

# We estimate theta from these three points
theta=powersolve(nset,k2,y_var=var_w)$par

# We will estimate the posterior at these values of n
n=seq(1000,50000,length=1000)

# Mean and variance
p_mu=mu_fn(n,nset=nset,d=k2,var_w = var_w, N=N,k1=k1,theta=theta,
           k_width=k_width,var_u=var_u)
p_var=psi_fn(n,nset=nset,N=N,var_w = var_w,k_width=k_width,var_u=var_u)

# Plot
plot(0,xlim=range(n),ylim=c(20000,60000),type="n",
     xlab="Training/holdout set size",
     ylab="Total cost (= num. cases)")
lines(n,p_mu,col="blue")
```

```
lines(n,p_mu - 3*sqrt(p_var),col="red")
lines(n,p_mu + 3*sqrt(p_var),col="red")
points(nset,k1*nset + k2*(N-nset),pch=16,col="purple")
lines(n,k1*n + powerlaw(n,theta)*(N-n),lty=2)
segments(nset,k1*nset + (k2 - 3*sqrt(var_w))*(N-nset),
         nset,k1*nset + (k2 + 3*sqrt(var_w))*(N-nset))
legend("topright",
       c(expression(mu(n)),
         expression(mu(n) %+-% 3*sqrt(psi(n))),
         "prior(n)",
         "d",
         "3SD(d|n)"),
       lty=c(1,1,2,NA,NA),lwd=c(1,1,1,NA,NA),pch=c(NA,NA,NA,16,124),
       pt.cex=c(NA,NA,NA,1,1),
       col=c("blue","red","black","purple","black"),bg="white")
```

`next_n` *Finds best value of n to sample next*

Description

Recommends a value of n at which to next evaluate individual cost in order to most accurately estimate optimal holdout size. Currently only for use with a power-law parametrisation of k2.

Approximately finds a set of n points which, given estimates of cost, minimise width of 95% confidence interval around OHS. Uses a greedy algorithm, so various parameters can be learned along the way.

Given existing training set size/cost estimates `nset` and `d`, with `var_w[i]=variance(d[i])`, finds, for each candidate point `n[i]`, the median width of the 90% confidence interval for OHS if

`nset` <-`c(nset,n[i])` `var_w` <-`c(var_w,mean(var_w))` `d` <-`c(d,rnorm(powerlaw(n[i],theta),variance=mean(v`

Usage

```
next_n(
  n,
  nset,
  d,
  N,
  k1,
  nmed = 100,
  var_w = rep(1, length(nset)),
  mode = "asymptotic",
  ...
)
```

Arguments

`n`	Set of training set sizes to evaluate
`nset`	Training set sizes for which a loss has been evaluated
`d`	Loss at training set sizes `nset`
`N`	Total number of samples on which the model will be fitted/used

k1	Mean loss per sample with no predictive score in place
nmed	number of times to re-evaluate d and confidence interval width.
var_w	Variance of error in loss estimate at each training set size.
mode	Mode for calculating OHS CI (passed to ci_ohs): 'asymptotic' or 'empirical'
...	Passed to powersolve and powersolve_se

Value

Vector out of same length as n, where out[i] is the expected width of the 95% confidence interval for OHS should n be added to nset.

Examples

```
# Set seed.
set.seed(24015)

# Kernel width and Gaussian process variance
kw0=5000
vu0=1e7

# Include legend on plots or not; inclusion can obscure plot elements on small figures
inc_legend=FALSE

# Suppose we have population size and cost-per-sample without a risk score as follows
N=100000
k1=0.4

# Suppose that true values of a,b,c are given by
theta_true=c(10000,1.2,0.2)
theta_lower=c(1,0.5,0.1) # lower bounds for estimating theta
theta_upper=c(20000,2,0.5) # upper bounds for estimating theta



# We start with five random holdout set sizes (nset0),
#  with corresponding cost-per-individual estimates d0 derived
#  with various errors var_w0
nstart=10
vwmin=0.001; vwmax=0.005
nset0=round(runif(nstart,1000,N/2))
var_w0=runif(nstart,vwmin,vwmax)
d0=rnorm(nstart,mean=powerlaw(nset0,theta_true),sd=sqrt(var_w0))

# We estimate theta from these three points
theta0=powersolve(nset0,d0,y_var=var_w0,lower=theta_lower,upper=theta_upper,init=theta_true)$par

# We will estimate the posterior at these values of n
n=seq(1000,N,length=1000)

# Mean and variance
p_mu=mu_fn(n,nset=nset0,d=d0,var_w = var_w0, N=N,k1=k1,theta=theta0,k_width=kw0,var_u=vu0)
p_var=psi_fn(n,nset=nset0,N=N,var_w = var_w0,k_width=kw0,var_u=vu0)

# Plot
yrange=c(-30000,100000)
```

```
plot(0,xlim=range(n),ylim=yrange,type="n",
  xlab="Training/holdout set size",
  ylab="Total cost (= num. cases)")
lines(n,p_mu,col="blue")
lines(n,p_mu - 3*sqrt(p_var),col="red")
lines(n,p_mu + 3*sqrt(p_var),col="red")
points(nset0,k1*nset0 + d0*(N-nset0),pch=16,col="purple")
lines(n,k1*n + powerlaw(n,theta0)*(N-n),lty=2)
lines(n,k1*n + powerlaw(n,theta_true)*(N-n),lty=3,lwd=3)
if (inc_legend) {
  legend("topright",
    c(expression(mu(n)),
      expression(mu(n) %+-% 3*sqrt(psi(n))),
      "prior(n)",
      "True",
      "d"),
    lty=c(1,1,2,3,NA),lwd=c(1,1,1,3,NA),pch=c(NA,NA,NA,NA,16),pt.cex=c(NA,NA,NA,NA,1),
    col=c("blue","red","black","purple"),bg="white")
}

## Add line corresponding to recommended new point. This is slow.
nn=seq(1000,N,length=20)
exp_imp <- next_n(nn,nset=nset0,d=d0,var_w = var_w0, N=N,k1=k1,nmed=10,
                     lower=theta_lower,upper=theta_upper)
abline(v=nn[which.min(exp_imp)])
```

`ohs_array` *Data for vignette on algorithm comparison*

Description

This object contains data relating to the vignette comparing emulation and parametric algorithms. For generation, see hidden code in vignette, or in pipeline at https://github.com/jamesliley/OptHoldoutSize_pipelines

Usage

```
ohs_array
```

Format

An object of class `array` of dimension 200 x 200 x 2 x 2 x 2.

`ohs_resample` *Data for vignette on algorithm comparison*

Description

This object contains data relating to the first plot in the vignette comparing emulation and parametric algorithms. For generation, see hidden code in vignette, or in pipeline at https://github.com/jamesliley/OptHoldoutSize_p

Usage

```
ohs_resample
```

Format

An object of class `matrix` (inherits from `array`) with 1000 rows and 4 columns.

`optimal_holdout_size` *Estimate optimal holdout size under parametric assumptions*

Description

Compute optimal holdout size for updating a predictive score given appropriate parameters of cost function

Evaluates empirical minimisation of cost function l(n;k1,N,theta) = k1 n + k2(n;theta) (N-n).

The function will return `Inf` if no minimum exists. It does not check if the minimum is unique, but this can be guaranteed using the assumptions for theorem 1 in the manuscript.

This calls the function `optimize` from package `stats`.

Usage

```
optimal_holdout_size(N, k1, theta, k2 = powerlaw, round_result = FALSE, ...)
```

Arguments

`N`	Total number of samples on which the predictive score will be used/fitted. Can be a vector.
`k1`	Cost value in the absence of a predictive score. Can be a vector.
`theta`	Parameters for function k2(n) governing expected cost to an individual sample given a predictive score fitted to n samples. Can be a matrix of dimension n x n_par, where n_par is the number of parameters of k2.
`k2`	Function governing expected cost to an individual sample given a predictive score fitted to n samples. Must take two arguments: n (number of samples) and theta (parameters). Defaults to a power-law form powerlaw(n,c(a,b,c))=a n^(-b) + c.
`round_result`	Set to TRUE to solve over integral sizes
`...`	Passed to function `optimize`

Value

List/data frame of dimension (number of evaluations) x (4 + n_par) containing input data and results. Columns `size` and `cost` are optimal holdout size and cost at this size respectively. Parameters N, k1, theta.1, theta.2,...,theta.n_par are input data.

Examples

```
# Evaluate optimal holdout set size for a range of values of k1 and two values of N, some of which lead to infinit
N1=10000; N2=12000
k1=seq(0.1,0.5,length=20)
A=3; B=1.5; C=0.15; theta=c(A,B,C)

res1=optimal_holdout_size(N1,k1,theta)
res2=optimal_holdout_size(N2,k1,theta)

par(mfrow=c(1,2))
plot(0,type="n",ylim=c(0,500),xlim=range(res1$k1),xlab=expression("k"[1]),ylab="Optimal holdout set size")
  lines(res1$k1,res1$size,col="black")
  lines(res2$k1,res2$size,col="red")
  legend("topright",as.character(c(N1,N2)),title="N:",col=c("black","red"),lty=1)
plot(0,type="n",ylim=c(1500,1600),xlim=range(res1$k1),xlab=expression("k"[1]),ylab="Minimum cost")
  lines(res1$k1,res1$cost,col="black")
  lines(res2$k1,res2$cost,col="red")
  legend("topleft",as.character(c(N1,N2)),title="N:",col=c("black","red"),lty=1)
```

---

optimal_holdout_size_emulation

Estimate optimal holdout size under semi-parametric assumptions

---

Description

Compute optimal holdout size for updating a predictive score given a set of training set sizes and estimates of mean cost per sample at those training set sizes.

This is essentially a wrapper for function `mu_fn()`.

Usage

```
optimal_holdout_size_emulation(
  nset,
  d,
  var_w,
  N,
  k1,
  var_u = 1e+07,
  k_width = 5000,
  mean_fn = powerlaw,
  theta = powersolve(nset, d, y_var = var_w)$par,
  npoll = 1000,
  ...
)
```

Arguments

`nset`	Training set sizes for which a loss has been evaluated
`d`	Loss at training set sizes `nset`
`var_w`	Variance of error in loss estimate at each training set size.
`N`	Total number of samples on which the model will be fitted/used

k1	Mean loss per sample with no predictive score in place
var_u	Marginal variance for Gaussian process kernel. Defaults to 1e7
k_width	Kernel width for Gaussian process kernel. Defaults to 5000
mean_fn	Functional form governing expected loss per sample given sample size. Should take two parameters: n (sample size) and theta (parameters). Defaults to function powerlaw.
theta	Current estimates of parameter values for mean_fn. Defaults to the MLE powerlaw solution corresponding to n,d, and var_w.
npoll	Check npoll equally spaced values between 1 and N for minimum. If NULL, check all values (this can be slow). Defaults to 1000
...	Passed to function optimise()

Value

Object of class 'optholdoutsize_emul' with elements "cost" (minimum cost),"size" (OHS),"nset","d","var_w","N","k1"," (parameters)

Examples

```
# See examples for mu_fn()
```

`oracle_pred` *Generate responses*

Description

Probably for deprecation

Usage

```
oracle_pred(X, coefs, num_vars = 3, noise = TRUE)
```

Arguments

X	Matrix of observations
coefs	Vector of coefficients for logistic model.
num_vars	If noise==FALSE, computes using only first num_vars predictors
noise	If TRUE, uses all predictors

Value

Vector of predictions

Examples

```
# See examples for model_predict
```

params_aspre — *Parameters of reported ASPRE dataset*

Description

Distribution of covariates for ASPRE dataset; see Rolnik, 2017, NEJM

Usage

```
params_aspre
```

Format

An object of class `list` of length 16.

plot.optholdoutsize — *Plot estimated cost function*

Description

Plot estimated cost function, when parametric method is used for estimation.

Draws cost function as a line and indicates minimum. Assumes a power-law form of k2 unless parameter k2 is set otherwise.

Usage

```
## S3 method for class 'optholdoutsize'
plot(x, ..., k2 = powerlaw)
```

Arguments

`x`	Object of type `optholdoutsize`
`...`	Other arguments passed to `plot()` and `lines()`
`k2`	Function governing expected cost to an individual sample given a predictive score fitted to n samples. Must take two arguments: n (number of samples) and theta (parameters). Defaults to a power-law form powerlaw(n,c(a,b,c))=a n^(-b) + c.

Examples

```
# Simple example

N=100000;
k1=0.3
A=8000; B=1.5; C=0.15; theta=c(A,B,C)

res1=optimal_holdout_size(N,k1,theta)

plot(res1)
```

plot.optholdoutsize_emul

Plot estimated cost function using emulation (semiparametric)

Description

Plot estimated cost function, when semiparametric (emulation) method is used for estimation.

Draws posterior mean of cost function as a line and indicates minimum. Also draws mean +/- 3 SE.

Assumes a power-law form of k2 unless parameter k2 is set otherwise.

Usage

```
## S3 method for class 'optholdoutsize_emul'
plot(x, ..., k2 = powerlaw)
```

Arguments

x	Object of type `optholdoutsize_emul`
...	Other arguments passed to `plot()`
k2	Function governing expected cost to an individual sample given a predictive score fitted to n samples. Must take two arguments: n (number of samples) and theta (parameters). Defaults to a power-law form powerlaw(n,c(a,b,c))=a n^(-b) + c.

Examples

```
# Simple example

# Parameters
N=100000;
k1=0.3
A=8000; B=1.5; C=0.15; theta=c(A,B,C)

# True mean function
k2_true=function(n) powerlaw(n,theta)

# Values of n for which cost has been estimated
np=50 # this many points
nset=round(runif(np,1,N))
var_w=runif(np,0.001,0.002)
d=rnorm(np,mean=k2_true(nset),sd=sqrt(var_w))

# Compute OHS
res1=optimal_holdout_size_emulation(nset,d,var_w,N,k1)

# Plot
plot(res1)
```

powersolve *Fit power law curve*

Description

Find least-squares solution: MLE of (a,b,c) under model y_i = a x_i^-b + c + e_i; `e_i~N(0,y_var_i)`

Usage

```
powersolve(
  x,
  y,
  init = c(20000, 2, 0.1),
  y_var = rep(1, length(y)),
  estimate_s = FALSE,
  ...
)
```

Arguments

`x`	X values
`y`	Y values
`init`	Initial values of (a,b,c) to start. Default c(20000,2,0.1)
`y_var`	Optional parameter giving sampling variance of each y value. Defaults to 1.
`estimate_s`	Parameter specifying whether to also estimate s (as above). Defaults to FALSE (no).
`...`	further parameters passed to optim. We suggest specifying lower and upper bounds for (a,b,c); e.g. lower=c(1,0,0),upper=c(10000,3,1)

Value

List (output from `optim`) containing MLE values of (a,b,c)

Examples

```
# Retrieval of original values
A_true=2000; B_true=1.5; C_true=0.3; sigma=0.002

X=1000*abs(rnorm(10000,mean=4))
Y=A_true*(X^(-B_true)) + C_true + rnorm(length(X),sd=sigma)

c(A_true,B_true,C_true)
powersolve(X[1:10],Y[1:10])$par
powersolve(X[1:100],Y[1:100])$par
powersolve(X[1:1000],Y[1:1000])$par
powersolve(X[1:10000],Y[1:10000])$par
```

powersolve_general *General solver for power law curve*

Description

Find least-squares solution: MLE of (a,b,c) under model y_i = a x_i^-b + c + e_i; e_i~N(0,y_var_i)

Try a range of starting values and refine estimate.

Slower than a single call to powersolve()

Usage

```
powersolve_general(x, y, y_var = rep(1, length(x)), ...)
```

Arguments

x	X values
y	Y values
y_var	Optional parameter giving sampling variance of each y value. Defaults to 1.
...	further parameters passed to optim. We suggest specifying lower and upper bounds for (a,b,c); e.g. lower=c(1,0,0),upper=c(10000,3,1)

Value

List (output from optim) containing MLE values of (a,b,c)

Examples

```
# Retrieval of original values
A_true=2000; B_true=1.5; C_true=0.3; sigma=0.002

X=1000*abs(rnorm(10000,mean=4))
Y=A_true*(X^(-B_true)) + C_true + rnorm(length(X),sd=sigma)

c(A_true,B_true,C_true)
powersolve_general(X[1:10],Y[1:10])$par
powersolve_general(X[1:100],Y[1:100])$par
powersolve_general(X[1:1000],Y[1:1000])$par
powersolve_general(X[1:10000],Y[1:10000])$par
```

powersolve_se *Standard error matrix for learning curve parameters (power law)*

Description

Find approximate standard error matrix for (a,b,c) under power law model for learning curve.

Assumes that
y_i= a x_i^-b + c + e, e~N(0,s^2 y_var_i^2)

Standard error can be computed either asymptotically using Fisher information (`method='fisher'`) or boostrapped (`method='bootstrap'`)

These estimate different quantities: the asymptotic method estimates

`Var[MLE(a,b,c)|X,y_var]`

and the boostrap method estimates

`Var[MLE(a,b,c)]`.

Usage

```
powersolve_se(
  x,
  y,
  method = "fisher",
  init = c(20000, 2, 0.1),
  y_var = rep(1, length(y)),
  n_boot = 1000,
  seed = NULL,
  ...
)
```

Arguments

`x`	X values (typically training set sizes)
`y`	Y values (typically observed cost per individual/sample)
`method`	One of 'fisher' (for asymptotic variance via Fisher Information) or 'bootstrap' (for Bootstrap)
`init`	Initial values of (a,b,c) to start when computing MLE. Default c(20000,2,0.1)
`y_var`	Optional parameter giving sampling variance of each y value. Defaults to 1.
`n_boot`	Number of bootstrap resamples. Only used if method='bootstrap'. Defaults to 1000
`seed`	Random seed for bootstrap resamples. Defaults to NULL.
`...`	further parameters passed to optim. We suggest specifying lower and upper bounds; since optim is called on (a*1000^-b,b,c), bounds should be relative to this; for instance, lower=c(0,0,0),upper=c(100,3,1)

Value

Standard error matrix; approximate covariance matrix of MLE(a,b,c)

Examples

```
A_true=10; B_true=1.5; C_true=0.3; sigma=0.1

set.seed(31525)
```

```
X=1+3*rchisq(10000,df=5)
Y=A_true*(X^(-B_true)) + C_true + rnorm(length(X),sd=sigma)

# 'Observations' - 100 samples
obs=sample(length(X),100,rep=FALSE)
Xobs=X[obs]; Yobs=Y[obs]

# True covariance matrix of MLE of a,b,c on these x values
ntest=1000
abc_mat_xfix=matrix(0,ntest,3)
abc_mat_xvar=matrix(0,ntest,3)
E1=A_true*(Xobs^(-B_true)) + C_true
for (i in 1:ntest) {
  Y1=E1 + rnorm(length(Xobs),sd=sigma)
  abc_mat_xfix[i,]=powersolve(Xobs,Y1)$par # Estimate (a,b,c) with same X

  X2=1+3*rchisq(length(Xobs),df=5)
  Y2=A_true*(X2^(-B_true)) + C_true + rnorm(length(Xobs),sd=sigma)
  abc_mat_xvar[i,]=powersolve(X2,Y2)$par # Estimate (a,b,c) with variable X
}

Ve1=var(abc_mat_xfix) # empirical variance of MLE(a,b,c)|X
Vf=powersolve_se(Xobs,Yobs,method='fisher') # estimated SE matrix, asymptotic

Ve2=var(abc_mat_xvar) # empirical variance of MLE(a,b,c)
Vb=powersolve_se(Xobs,Yobs,method='bootstrap') # estimated SE matrix, bootstrap

cat("Empirical variance of MLE(a,b,c)|X\n")
print(Ve1)
cat("\n")
cat("Asymptotic variance of MLE(a,b,c)|X\n")
print(Vf)
cat("\n\n")
cat("Empirical variance of MLE(a,b,c)\n")
print(Ve2)
cat("\n")
cat("Bootstrap-estimated variance of MLE(a,b,c)\n")
print(Vb)
cat("\n\n")
```

---

psi_fn *Updating function for variance.*

Description

Posterior variance for emulator given points n.

Usage

```
psi_fn(n, nset, var_w, N, var_u = 1e+07, k_width = 5000)
```

Arguments

`n`	Set of training set sizes to evaluate at
`nset`	Training set sizes for which a loss has been evaluated
`var_w`	Variance of error in loss estimate at each training set size.
`N`	Total number of samples on which the model will be fitted/used. Only used to rescale var_w
`var_u`	Marginal variance for Gaussian process kernel. Defaults to 1e7
`k_width`	Kernel width for Gaussian process kernel. Defaults to 5000

Value

Vector Psi of same length of n where Psi_i=var(posterior(cost(n_i)))

Examples

```
# See examples for `mu_fn`
```

`sens10` *Sensitivity at theshold quantile 10%*

Description

Computes sensitivity of a risk score at a threshold at which 10% of samples (or some proportion pi_int) are above the threshold.

Usage

```
sens10(Y, Ypred, pi_int = 0.1)
```

Arguments

`Y`	True labels (1 or 0)
`Ypred`	Predictions (univariate; real numbers)
`pi_int`	Compute sensitivity when a proportion pi_int of samples exceed threshold, default 0.1

Value

Sensitivity at this threshold

Examples

```
# Simulate
set.seed(32142)

N=1000
X=rnorm(N); Y=rbinom(N,1,prob=logit(X/2))

pi_int=0.1
q10=quantile(X,1-pi_int) # 10% of X values are above this threshold

print(length(which(Y==1 & X>q10))/length(which(X>q10)))
print(sens10(Y,X,pi_int))
```

`sim_random_aspre` *Simulate random dataset similar to ASPRE training data*

Description

Generate random population of individuals (e.g., newly pregnant women) with given population parameters

Assumes independence of parameter variation. This is not a realistic assumption, but is satisfactory for our purposes.

Usage

```
sim_random_aspre(n, params = NULL)
```

Arguments

n	size of population
params	list of parameters

Value

Matrix of samples

Examples

```
# Load ASPRE related data
data(params_aspre)

X=sim_random_aspre(1000,params_aspre)

print(c(median(X$age),params_aspre$age$median))

print(rbind(table(X$parity)/1000,params_aspre$parity$freq))
```

split_data — *Split data*

Description

Split data into holdout and intervention sets

Usage

```
split_data(X, frac)
```

Arguments

X	Matrix of observations
frac	Fraction of observations to use for the training set

Value

Vector of TRUE/FALSE values (randomised) with proportion `frac` as TRUE

Examples

```
# See examples for model_predict
```

Index

∗ **aspre**
add_aspre_interactions, 2
aspre, 3
aspre_k2, 4
sens10, 35
sim_random_aspre, 36
∗ **data,aspre**
aspre_emulation, 4
aspre_parametric, 5
ci_cover_a_yn, 5
ci_cover_e_yn, 6
data_example_simulation, 10
data_nextpoint_em, 10
data_nextpoint_par, 11
ohs_array, 25
ohs_resample, 25
params_aspre, 29
∗ **emulation**
cov_fn, 9
∗ **estimation,aspre**
powersolve, 31
powersolve_general, 32
powersolve_se, 32
∗ **estimation,emulation**
error_ohs_emulation, 11
optimal_holdout_size_emulation, 27
∗ **estimation**
ci_ohs, 6
grad_nstar_powerlaw, 17
optimal_holdout_size, 26
plot.optholdoutsize, 29
plot.optholdoutsize_emul, 30
∗ **simulation**
gen_base_coefs, 15
gen_preds, 16
gen_resp, 16
model_predict, 19
model_train, 20
oracle_pred, 28
split_data, 37

add_aspre_interactions, 2
aspre, 3
aspre_emulation, 4
aspre_k2, 4
aspre_parametric, 5

ci_cover_a_yn, 5
ci_cover_e_yn, 6
ci_ohs, 6
cov_fn, 9

data_example_simulation, 10
data_nextpoint_em, 10
data_nextpoint_par, 11

error_ohs_emulation, 11
exp_imp_fn, 13

gen_base_coefs, 15
gen_preds, 16
gen_resp, 16
grad_nstar_powerlaw, 17

logistic, 18
logit, 18

model_predict, 19
model_train, 20
mu_fn, 21

next_n, 23

ohs_array, 25
ohs_resample, 25
optimal_holdout_size, 26
optimal_holdout_size_emulation, 27
oracle_pred, 28

params_aspre, 29
plot.optholdoutsize, 29
plot.optholdoutsize_emul, 30
powerlaw *(mu_fn)*, 21
powersolve, 31
powersolve_general, 32
powersolve_se, 32
psi_fn, 34

sens10, 35
sim_random_aspre, 36
split_data, 37