

R topics documented:

\begin{itemize}
\item \texttt{coef.pense_cvfit} \hspace{1cm} 3
\item \texttt{coef.pense_fit} \hspace{1cm} 4
\item \texttt{consistency_const} \hspace{1cm} 6
\item \texttt{deprecatend_en_options} \hspace{1cm} 6
\item \texttt{elnet} \hspace{1cm} 8
\item \texttt{elnet_cv} \hspace{1cm} 10
\item \texttt{enpy} \hspace{1cm} 13
\item \texttt{enpy_initial_estimates} \hspace{1cm} 14
\item \texttt{enpy_options} \hspace{1cm} 15
\item \texttt{en_admm_options} \hspace{1cm} 16
\item \texttt{en_algorithm_options} \hspace{1cm} 17
\item \texttt{en_dal_options} \hspace{1cm} 17
\item \texttt{en_lars_options} \hspace{1cm} 18
\item \texttt{inittest_options} \hspace{1cm} 18
\item \texttt{mloc} \hspace{1cm} 19
\item \texttt{mlocscale} \hspace{1cm} 20
\item \texttt{mm_algorithm_options} \hspace{1cm} 21
\item \texttt{mscale} \hspace{1cm} 21
\item \texttt{mscale_algorithm_options} \hspace{1cm} 22
\item \texttt{mstep_options} \hspace{1cm} 23
\item \texttt{pense} \hspace{1cm} 24
\item \texttt{pensem} \hspace{1cm} 27
\item \texttt{pensem_cv} \hspace{1cm} 28
\item \texttt{pense_cv} \hspace{1cm} 31
\item \texttt{pense_options} \hspace{1cm} 35
\item \texttt{plot.pense_cvfit} \hspace{1cm} 36
\item \texttt{plot.pense_fit} \hspace{1cm} 37
\item \texttt{predict.pense_cvfit} \hspace{1cm} 38
\item \texttt{predict.pense_fit} \hspace{1cm} 40
\item \texttt{prediction_performance} \hspace{1cm} 42
\item \texttt{prinsens} \hspace{1cm} 43
\item \texttt{regmest} \hspace{1cm} 44
\item \texttt{regmest_cv} \hspace{1cm} 46
\item \texttt{residuals.pense_cvfit} \hspace{1cm} 49
\item \texttt{residuals.pense_fit} \hspace{1cm} 51
\item \texttt{rho_function} \hspace{1cm} 53
\item \texttt{starting_point} \hspace{1cm} 53
\item \texttt{summary.pense_cvfit} \hspace{1cm} 55
\item \texttt{tau_size} \hspace{1cm} 55
\end{itemize}

\textbf{Index} \hspace{1cm} 57
coefficients from an adaptive PENSE (or LS-EN) regularization path with hyper-parameters chosen by cross-validation.

Usage

```r
## S3 method for class 'pense_cvfit'
coef(
  object,
  alpha = NULL,
  lambda = "min",
  se_mult = 1,
  sparse = NULL,
  standardized = FALSE,
  exact = deprecated(),
  correction = deprecated(),
  ...
)
```

Arguments

- `object` PENSE with cross-validated hyper-parameters to extract coefficients from.
- `alpha` Either a single number or `NULL` (default). If given, only fits with the given `alpha` value are considered. If `lambda` is a numeric value and `object` was fit with multiple `alpha` values and no value is provided, the first value in `object$alpha` is used with a warning.
- `lambda` either a string specifying which penalty level to use ("min", "se", "{m}-se") or a single numeric value of the penalty parameter. See details.
- `se_mult` If `lambda = "se"`, the multiple of standard errors to tolerate.
- `sparse` should coefficients be returned as sparse or dense vectors? Defaults to the sparsity setting of the given object. Can also be set to `sparse = 'matrix'`, in which case a sparse matrix is returned instead of a sparse vector.
- `standardized` return the standardized coefficients.
- `exact, correction` defunct.
- `...` currently not used.

Value

either a numeric vector or a sparse vector of type `dsparseVector` of size \( p + 1 \), depending on the `sparse` argument. Note: prior to version 2.0.0 sparse coefficients were returned as sparse matrix of type `dgCMatrix`. To get a sparse matrix as in previous versions, use `sparse = 'matrix'`.
Hyper-parameters

If \( \lambda = "(m)-se" \) and object contains fitted estimates for every penalization level in the sequence, use the fit the most parsimonious model with prediction performance statistically indistinguishable from the best model. This is determined to be the model with prediction performance within \( m \times \text{cv.se} \) from the best model. If \( \lambda = "se" \), the multiplier \( m \) is taken from \( \text{se_mult} \).

By default all \( \alpha \) hyper-parameters available in the fitted object are considered. This can be overridden by supplying one or multiple values in parameter \( \alpha \). For example, if \( \lambda = "1-se" \) and \( \alpha \) contains two values, the "1-SE" rule is applied individually for each \( \alpha \) value, and the fit with the better prediction error is considered.

In case \( \lambda \) is a number and object was fit for several \( \alpha \) hyper-parameters, \( \alpha \) must also be given, or the first value in \( \text{object$alpha} \) is used with a warning.

See Also

Other functions for extracting components: \texttt{coef.pense_fit()}, \texttt{predict.pense_cvfit()}, \texttt{predict.pense_fit()}, \texttt{residuals.pense_cvfit()}, \texttt{residuals.pense_fit()}

Examples

# Compute the PENSE regularization path for Freeny's revenue data
# (see ?freeny)
data(freeny)
x <- as.matrix(freeny[, 2:5])
regpath <- pense(x, freeny$y, alpha = 0.5)
plot(regpath)

# Extract the coefficients at a certain penalization level
coef(regpath, lambda = regpath$lambda[[1]][[40]])

# What penalization level leads to good prediction performance?
set.seed(123)
cv_results <- pense_cv(x, freeny$y, alpha = 0.5,
                        cv_repl = 2, cv_k = 4)
plot(cv_results, se_mult = 1)

# Extract the coefficients at the penalization level with
# smallest prediction error ...
coef(cv_results)
# ... or at the penalization level with prediction error
# statistically indistinguishable from the minimum.
coef(cv_results, lambda = '1-se')

---

**coef.pense_fit**

**Extract Coefficient Estimates**

**Description**

Extract coefficients from an adaptive PENSE (or LS-EN) regularization path fitted by \texttt{pense()} or \texttt{einet()}. 
Usage

```r
## S3 method for class 'pense_fit'
coef(
  object,
  lambda,
  alpha = NULL,
  sparse = NULL,
  standardized = FALSE,
  exact = deprecated(),
  correction = deprecated(),
  ...)
```

Arguments

- `object`: PENSE regularization path to extract coefficients from.
- `lambda`: a single number for the penalty level.
- `alpha`: Either a single number or NULL (default). If given, only fits with the given `alpha` value are considered. If `object` was fit with multiple `alpha` values, and no value is provided, the first value in `object$alpha` is used with a warning.
- `sparse`: should coefficients be returned as sparse or dense vectors? Defaults to the sparsity setting in `object`. Can also be set to `sparse = 'matrix'`, in which case a sparse matrix is returned instead of a sparse vector.
- `standardized`: return the standardized coefficients.
- `exact, correction`: defunct.
- `...`: currently not used.

Value
either a numeric vector or a sparse vector of type `dsparseVector` of size \( p + 1 \), depending on the `sparse` argument. Note: prior to version 2.0.0 sparse coefficients were returned as sparse matrix of type `dgCMatrix`. To get a sparse matrix as in previous versions, use `sparse = 'matrix'`.

See Also

- `coef.pense_cvfit()` for extracting coefficients from a PENSE fit with hyper-parameters chosen by cross-validation
- Other functions for extracting components: `coef.pense_cvfit()`, `predict.pense_cvfit()`, `predict.pense_fit()`, `residuals.pense_cvfit()`, `residuals.pense_fit()`

Examples

```r
# Compute the PENSE regularization path for Freeny's revenue data
# (see ?freeny)
data(freeny)
x <- as.matrix(freeny[, 2:5])
regpath <- pense(x, freeny$y, alpha = 0.5)
plot(regpath)

# Extract the coefficients at a certain penalization level
coef(regpath, lambda = regpath$lambda[[1]][[40]])

# What penalization level leads to good prediction performance?
set.seed(123)
cv_results <- pense_cv(x, freeny$y, alpha = 0.5,
            cv_repl = 2, cv_k = 4)
plot(cv_results, se_mult = 1)

# Extract the coefficients at the penalization level with
# smallest prediction error ...
coef(cv_results)
# ... or at the penalization level with prediction error
# statistically indistinguishable from the minimum.
coef(cv_results, lambda = '1-se')

---

consistency_const  Get the Constant for Consistency for the M-Scale

Description

Get the Constant for Consistency for the M-Scale

Usage

consistency_const(delta, rho)

Arguments

delta  desired breakdown point (between 0 and 0.5)
rho    the name of the chosen ρ function.

Value

consistency constant

See Also

Other miscellaneous functions: rho_function()

---

deprecated_en_options  Deprecated

Description

[Deprecated]
Options for computing EN estimates.
Usage

en_options_aug_lars(use_gram = c("auto", "yes", "no"), eps = 1e-12)

en_options_dal(
  maxit = 100,
  eps = 1e-08,
  eta_mult = 2,
  eta_start_numerator = 0.01,
  eta_start,
  preconditioner = c("approx", "none", "diagonal"),
  verbosity = 0
)

Arguments

use_gram         ignored. Should the Gram matrix be pre-computed.
eps              ignored. Numeric tolerance for convergence.
maxit            maximum number of iterations allowed.
eta_mult         multiplier to increase eta at each iteration.
eta_start_numerator
  if eta_start is missing, it is defined by eta_start = eta_start_numerator / lambda.
eta_start        ignored. The start value for eta.
preconditioner   ignored. Preconditioner for the numerical solver. If none, a standard solver will be used, otherwise the faster preconditioned conjugate gradient is used.
verbosity         ignored.

Functions

• en_options_aug_lars: Superseded by en_lars_options()
• en_options_dal: Superseded by en_dal_options()

Warning

Do not use these functions in new code. They may be removed from future versions of the package.

See Also

Other deprecated functions: enpy(), initest_options(), mstep_options(), pense_options(), pensem()
elnet

Compute the Least Squares (Adaptive) Elastic Net Regularization Path

Description

Compute least squares EN estimates for linear regression with optional observation weights and penalty loadings.

Usage

elnet(
  x,  
y,  
alpha,  
nlambda = 100,  
lambda_min_ratio,  
lambda,  
penalty_loadings,  
weights,  
intercept = TRUE,  
en_algorithm_opts,  
sparse = FALSE,  
eps = 1e-06,  
standardize = TRUE,  
correction = deprecated(),  
xtest = deprecated(),  
options = deprecated()
)

Arguments

x  
n by p matrix of numeric predictors.

y  
vector of response values of length n. For binary classification, y should be a factor with 2 levels.

alpha  
elastic net penalty mixing parameter with 0 ≤ α ≤ 1. alpha = 1 is the LASSO penalty, and alpha = 0 the Ridge penalty. Can be a vector of several values, but alpha = 0 cannot be mixed with other values.

nnlambda  
number of penalization levels.

lambda_min_ratio  
Smallest value of the penalization level as a fraction of the largest level (i.e., the smallest value for which all coefficients are zero). The default depends on the sample size relative to the number of variables and alpha. If more observations than variables are available, the default is 1e-3 * alpha, otherwise 1e-2 * alpha.

lambda  
optional user-supplied sequence of penalization levels. If given and not NULL, nlambda and lambda_min_ratio are ignored.

penalty_loadings  
a vector of positive penalty loadings (a.k.a. weights) for different penalization of each coefficient.
weights a vector of positive observation weights.
intercept include an intercept in the model.
en_algorithm_opts options for the EN algorithm. See en_algorithm_options for details.
sparse use sparse coefficient vectors.
eps numerical tolerance.
standardize standardize variables to have unit variance. Coefficients are always returned in original scale.
correction defunct. Correction for EN estimates is not supported anymore.
xtest defunct.
options deprecated. Use en_algorithm_opts instead.

Details
The elastic net estimator for the linear regression model solves the optimization problem

$$\arg\min_{\mu, \beta} \left( \frac{1}{2n} \sum_i w_i (y_i - \mu - x_i^T \beta)^2 + \lambda (\frac{1}{2} - \alpha) \beta_j^2 + \alpha l_j |\beta_j| \right)$$

with observation weights $w_i$ and penalty loadings $l_j$.

Value
a list-like object with the following items

alpha the sequence of alpha parameters.
lambda a list of sequences of penalization levels, one per alpha parameter.
estimates a list of estimates. Each estimate contains the following information:
  intercept intercept estimate.
  beta beta (slope) estimate.
  lambda penalization level at which the estimate is computed.
  alpha alpha hyper-parameter at which the estimate is computed.
  statuscode if > 0 the algorithm experienced issues when computing the estimate.
  status optional status message from the algorithm.
call the original call.

See Also
pense() for an S-estimate of regression with elastic net penalty.
coef.pense_fit() for extracting coefficient estimates.
plot.pense_fit() for plotting the regularization path.
Other functions for computing non-robust estimates: elnet_cv()
Examples

```r
# Compute the LS-EN regularization path for Freeny's revenue data
# (see ?freeny)
data(freeny)
x <- as.matrix(freeny[, 2:5])

regpath <- elnet(x, freeny$y, alpha = c(0.5, 0.75))
plot(regpath)
plot(regpath, alpha = 0.75)

# Extract the coefficients at a certain penalization level
coef(regpath, lambda = regpath$lambda[[1]][[5]],
     alpha = 0.75)

# What penalization level leads to good prediction performance?
set.seed(123)
cv_results <- elnet_cv(x, freeny$y, alpha = c(0.5, 0.75),
                        cv_repl = 10, cv_k = 4,
                        cv_measure = "tau")
plot(cv_results, se_mult = 1.5)
plot(cv_results, se_mult = 1.5, what = "coef.path")

# Extract the coefficients at the penalization level with
# smallest prediction error ...
summary(cv_results)
coef(cv_results)
# ... or at the penalization level with prediction error
# statistically indistinguishable from the minimum.
summary(cv_results, lambda = "1.5-se")
coef(cv_results, lambda = "1.5-se")
```

---

**elnet_cv**  
*Cross-validation for Least-Squares (Adaptive) Elastic Net Estimates*

**Description**

Perform (repeated) K-fold cross-validation for `elnet()`.

**Usage**

```r
elnet_cv(
  x,
  y,
  lambda,
  cv_k,
  cv_repl = 1,
  cv_metric = c("rmspe", "tau_size", "mape", "auroc"),
  fit_all = TRUE,
  cl = NULL,
  ncores = deprecated(),
  ...
)
```
elnet_cv

Arguments

x  
n by p matrix of numeric predictors.
y  
vector of response values of length n. For binary classification, y should be a factor with 2 levels.
lambda  
optional user-supplied sequence of penalization levels. If given and not NULL, nlambda and lambda_min_ratio are ignored.
cv_k  
number of folds per cross-validation.
cv_repl  
number of cross-validation replications.
cv_metric  
either a string specifying the performance metric to use, or a function to evaluate prediction errors in a single CV replication. If a function, the number of arguments define the data the function receives. If the function takes a single argument, it is called with a single numeric vector of prediction errors. If the function takes two or more arguments, it is called with the predicted values as first argument and the true values as second argument. The function must always return a single numeric value quantifying the prediction performance. The order of the given values corresponds to the order in the input data.
fit_all  
If TRUE, fit the model for all penalization levels. Can also be any combination of "min" and "{x}-se", in which case only models at the penalization level with smallest average CV accuracy, or within {x} standard errors, respectively. Setting fit_all to FALSE is equivalent to "min". Applies to all alpha value.
cl  
a parallel cluster. Can only be used in combination with ncores = 1.
ncores  
deprecated and not used anymore.
...  
Arguments passed on to elnet
alpha  
elastic net penalty mixing parameter with 0 ≤ α ≤ 1. alpha = 1 is the LASSO penalty, and alpha = 0 the Ridge penalty. Can be a vector of several values, but alpha = 0 cannot be mixed with other values.
nlambda  
number of penalization levels.
lambda_min_ratio  
Smallest value of the penalization level as a fraction of the largest level (i.e., the smallest value for which all coefficients are zero). The default depends on the sample size relative to the number of variables and alpha. If more observations than variables are available, the default is 1e-3 * alpha, otherwise 1e-2 * alpha.
penalty_loadings  
a vector of positive penalty loadings (a.k.a. weights) for different penalization of each coefficient.
standardize  
standardize variables to have unit variance. Coefficients are always returned in original scale.
weights  
a vector of positive observation weights.
intercept  
include an intercept in the model.
sparse  
use sparse coefficient vectors.
en_algorithm_opts  
options for the EN algorithm. See en_algorithm_options for details.
eps  
numerical tolerance.
xtest  
defunct.
options  
deprecated. Use en_algorithm_opts instead.
correction  
defunct. Correction for EN estimates is not supported anymore.
Details

The built-in CV metrics are

- "tau_size" \( \tau \)-size of the prediction error, computed by \texttt{tau_size()} (default).
- "mape" Median absolute prediction error.
- "rmspe" Root mean squared prediction error.
- "auroc" Area under the receiver operator characteristic curve (actually 1 - AUROC). Only sensible for binary responses.

Value

A list-like object with the same components as returned by \texttt{elnet()}, plus the following:

- cvres data frame of average cross-validated performance.

See Also

- \texttt{elnet()} for computing the LS-EN regularization path without cross-validation.
- \texttt{pense_cv()} for cross-validation of S-estimates of regression with elastic net penalty.
- \texttt{coef.pense_cvfit()} for extracting coefficient estimates.
- \texttt{plot.pense_cvfit()} for plotting the CV performance or the regularization path.

Other functions for computing non-robust estimates: \texttt{elnet()}

Examples

```r
# Compute the LS-EN regularization path for Freeny's revenue data
# (see ?freeny)
data(freeny)
x <- as.matrix(freeny[, 2:5])
regpath <- elnet(x, freeny$y, alpha = c(0.5, 0.75))
plot(regpath)

# Extract the coefficients at a certain penalization level
test = coef(regpath, lambda = regpath$lambda[1][5],
            alpha = 0.75)

# What penalization level leads to good prediction performance?
set.seed(123)
cv_results <- elnet_cv(x, freeny$y, alpha = c(0.5, 0.75),
                        cv_repl = 10, cv_k = 4,
                        cv_measure = "tau")
plot(cv_results, se_mult = 1.5)
plot(cv_results, se_mult = 1.5, what = "coef.path")

# Extract the coefficients at the penalization level with
# smallest prediction error ...
summary(cv_results)
coef(cv_results)
# ... or at the penalization level with prediction error
# statistically indistinguishable from the minimum.
```
enpy

summary(cv_results, lambda = "1.5-se")
coef(cv_results, lambda = "1.5-se")

---

Description

[Deprecated]

Compute initial estimates for EN S-estimates using ENPY. Superseded by `enpy_initial_estimates()`.

Usage

`enpy(x, y, alpha, lambda, delta, cc, options, en_options)`

Arguments

- `x` data matrix with predictors.
- `y` response vector.
- `alpha, lambda` EN penalty parameters (NOT adjusted for the number of observations in `x`).
- `delta` desired breakdown point of the resulting estimator.
- `cc` tuning constant for the S-estimator. Default is chosen based on the breakdown point `delta`. Should never have to be changed.
- `options` `ignored`. Additional options for the initial estimator.
- `en_options` `ignored`. Additional options for the EN algorithm.

Value

- `coeff` A numeric matrix with one initial coefficient per column
- `objf` A vector of values of the objective function for the respective coefficient

Warning

Do not use this function in new code. It may be removed from future versions of the package.

See Also

Other deprecated functions: `deprecated_en_options`, `initest_options()`, `mstep_options()`, `pense_options()`, `pensem()`
enpy_initial_estimates

ENPY Initial Estimates for EN S-Estimators

Description

Compute initial estimates for the EN S-estimator using the EN-PY procedure.

Usage

enpy_initial_estimates(
  x, y, alpha, lambda, bdp = 0.25, cc, intercept = TRUE, penalty_loadings,
  enpy_opts = enpy_options(), mscale_opts = mscale_algorithm_options(),
  eps = 1e-06, sparse = FALSE, ncores = 1L
)

Arguments

x n by p matrix of numeric predictors.
y vector of response values of length n.
alpha elastic net penalty mixing parameter with 0 ≤ α ≤ 1. alpha = 1 is the LASSO penalty, and alpha = 0 the Ridge penalty. Can be a vector of several values, but alpha = 0 cannot be mixed with other values.
lambda a vector of positive values of penalization levels.
bdp desired breakdown point of the estimator, between 0 and 0.5. The actual breakdown point may be slightly larger/smaller to avoid instabilities of the S-loss.
cc cutoff value for the bisquare rho function. By default, chosen to yield a consistent estimate for the Normal distribution.
intercept include an intercept in the model.
penalty_loadings a vector of positive penalty loadings (a.k.a. weights) for different penalization of each coefficient. Only allowed for alpha > 0.
enpy_opts options for the EN-PY algorithm, created with the enpy_options() function.
mscale_opts options for the M-scale estimation. See mscale_algorithm_options() for details.
eps numerical tolerance.
sparse use sparse coefficient vectors.
ncores number of CPU cores to use in parallel. By default, only one CPU core is used. Not supported on all platforms, in which case a warning is given.
Details

If these manually computed initial estimates are intended as starting points for `pense()`, they are by default shared for all penalization levels. To restrict the use of the initial estimates to the penalty level they were computed for, use `as_starting_point(..., specific = TRUE)`. See `as_starting_point()` for details.

References


See Also

Other functions for initial estimates: `prinsens()`, `starting_point()`

---

**enpy_options**  
*Options for the ENPY Algorithm*

Description

Additional control options for the elastic net Peña-Yohai procedure.

Usage

```r
enpy_options(
  max_it = 10,
  keep_psc_proportion = 0.5,
  en_algorithm_opts,
  keep_residuals_measure = c("threshold", "proportion"),
  keep_residuals_proportion = 0.5,
  keep_residuals_threshold = 2,
  retain_best_factor = 2,
  retain_max = 500
)
```

Arguments

- `max_it`  
  maximum number of EN-PY iterations.
- `keep_psc_proportion`  
  how many observations should to keep based on the Principal Sensitivity Components.
- `en_algorithm_opts`  
  options for the LS-EN algorithm. See `en_algorithm_options` for details.
- `keep_residuals_measure`  
  how to determine what observations to keep, based on their residuals. If proportion, a fixed number of observations is kept. If threshold, only observations with residuals below the threshold are kept.
- `keep_residuals_proportion`  
  proportion of observations to kept based on their residuals.
**en_admm_options**

keep_residuals_threshold
only observations with (standardized) residuals less than this threshold are kept.

retain_best_factor
only keep candidates that are within this factor of the best candidate. If <= 1, only keep candidates from the last iteration.

retain_max
maximum number of candidates, i.e., only the best retain_max candidates are retained.

Details
The EN-PY procedure for computing initial estimates iteratively cleans the data of observations with possibly outlying residual or high leverage. Least-squares elastic net (LS-EN) estimates are computed on the possibly clean subsets. At each iteration, the Principal Sensitivity Components are computed to remove observations with potentially high leverage. Among all the LS-EN estimates, the estimate with smallest M-scale of the residuals is selected. Observations with largest residual for the selected estimate are removed and the next iteration is started.

Value
options for the ENPY algorithm.

---

**en_admm_options** Use the ADMM Elastic Net Algorithm

Description
Use the ADMM Elastic Net Algorithm

Usage
en_admm_options(max_it = 1000, step_size, acceleration = 1)

Arguments
max_it maximum number of iterations.
step_size step size for the algorithm.
acceleration acceleration factor for linearized ADMM.

Value
options for the ADMM EN algorithm.

See Also
Other EN algorithms: en_dal_options(), en_lars_options()
Control the Algorithm to Compute (Weighted) Least-Squares Elastic Net Estimates

Description

The package supports different algorithms to compute the EN estimate for weighted LS loss functions. Each algorithm has certain characteristics that make it useful for some problems. To select a specific algorithm and adjust the options, use any of the en_***_options functions.

Details

- \texttt{en_lars_options()}: Use the tuning-free LARS algorithm. This computes exact (up to numerical errors) solutions to the EN-LS problem. It is not iterative and therefore can not benefit from approximate solutions, but in turn guarantees that a solution will be found.
- \texttt{en_admm_options()}: Use an iterative ADMM-type algorithm which needs $O(np)$ operations per iteration and converges sub-linearly.
- \texttt{en_dal_options()}: Use the iterative Dual Augmented Lagrangian (DAL) method. DAL needs $O(n^3p^2)$ operations per iteration, but converges exponentially.

Use the DAL Elastic Net Algorithm

Usage

\begin{verbatim}
en_dal_options(
  max_it = 100,
  max_inner_it = 100,
  eta_multiplier = 2,
  eta_start_conservative = 0.01,
  eta_start_aggressive = 1,
  lambda_relchange_aggressive = 0.25
)
\end{verbatim}

Arguments

- \texttt{max_it} maximum number of (outer) iterations.
- \texttt{max_inner_it} maximum number of (inner) iterations in each outer iteration.
- \texttt{eta_multiplier} multiplier for the barrier parameter. In each iteration, the barrier must be more restrictive (i.e., the multiplier must be > 1).
- \texttt{eta_start_conservative} conservative initial barrier parameter. This is used if the previous penalty is undefined or too far away.
eta_start_aggressive
aggressive initial barrier parameter. This is used if the previous penalty is close.
lambda_relchange_aggressive
how close must the lambda parameter from the previous penalty term be to use
an aggressive initial barrier parameter (i.e., what constitutes "too far").

Value
options for the DAL EN algorithm.

See Also
Other EN algorithms: en_admm_options(), en_lars_options()

en_lars_options
Use the LARS Elastic Net Algorithm

Description
Use the LARS Elastic Net Algorithm

Usage
en_lars_options()

See Also
Other EN algorithms: en_admm_options(), en_dal_options()

initest_options
Deprecated

Description
[Deprecated]
Options for computing initial estimates via ENPY. Superseded by enpy_options().

Usage
initest_options(
    keep_solutions = 5,
    psc_method = c("exact", "rr"),
    maxit = 10,
    maxit_pense_refinement = 5,
    eps = 1e-06,
    psc_keep = 0.5,
    resid_keep_method = c("proportion", "threshold"),
    resid_keep_prop = 0.6,
    resid_keep_thresh = 2,
    mscale_eps = 1e-08,
    mscale_maxit = 200
)
mloc

**Arguments**

- **keep_solutions**: how many initial estimates should be kept to perform full PENSE iterations?
- **psc_method**: The method to use for computing the principal sensitivity components. See details for the possible choices.
- **maxit**: maximum number of refinement iterations.
- **maxit_pense_refinement**: ignored. Maximum number of PENSE iterations to refine initial estimator.
- **eps**: ignored. Numeric tolerance for convergence.
- **psc_keep**: proportion of observations to keep based on the PSC scores.
- **resid_keep_method**: How to clean the data based on large residuals. If "proportion", observations with the smallest resid_keep_prop residuals will be retained. If "threshold", all observations with scaled residuals smaller than the threshold resid_keep_thresh will be retained.
- **resid_keep_prop**, **resid_keep_thresh**: proportion or threshold for observations to keep based on their residual.
- **mscale_eps**, **mscale_maxit**: ignored. Maximum number of iterations and numeric tolerance for the M-scale.

**Warning**

Do not use this function in new code. It may be removed from future versions of the package.

**See Also**

Other deprecated functions: `deprecated_en_options`, `enpy()`, `mstep_options()`, `pense_options()`, `pensem()`

---

**mloc**

*Compute the M-estimate of Location*

**Description**

Compute the M-estimate of location using an auxiliary estimate of the scale.

**Usage**

```r
mloc(x, scale, rho, cc, opts = mscale_algorithm_options())
```

**Arguments**

- **x**: numeric values. Missing values are verbosely ignored.
- **scale**: scale of the x values. If omitted, uses the `mad()`.
- **rho**: the ρ function to use. See `rho_function()` for available functions.
- **cc**: value of the tuning constant for the chosen ρ function. By default, chosen to achieve 95% efficiency under the Normal distribution.
- **opts**: a list of options for the M-estimating algorithm, see `mscale_algorithm_options()` for details.
Value

a single numeric value, the M-estimate of location.

See Also

Other functions to compute robust estimates of location and scale: mlocscale(), mscale(), tau_size()

---

mlocscale  Compute the M-estimate of Location and Scale

Description

Simultaneous estimation of the location and scale by means of M-estimates.

Usage

mlocscale(
  x,
  bdp = 0.25,
  scale_cc = consistency_const(bdp, "bisquare"),
  location_rho,
  location_cc,
  opts = mscale_algorithm_options()
)

Arguments

x  numeric values. Missing values are verbosely ignored.
bdp  desired breakdown point (between 0 and 0.5).
scale_cc  cutoff value for the bisquare ρ function for computing the scale estimate. By
default, chosen to yield a consistent estimate for normally distributed values.
location_rho, location_cc  ρ function and cutoff value for computing the location estimate. See rho_function()
for a list of available ρ functions.
opts  a list of options for the M-estimating equation, see mscale_algorithm_options() for details.

Value

a vector with 2 elements, the M-estimate of location and the M-scale estimate.

See Also

Other functions to compute robust estimates of location and scale: mloc(), mscale(), tau_size()
**mm_algorithm_options**

**MM-Algorithm to Compute Penalized Elastic Net S- and M-Estimates**

**Description**
Additional options for the MM algorithm to compute EN S- and M-estimates.

**Usage**

```r
mm_algorithm_options(
  max_it = 500,
  tightening = c("adaptive", "exponential", "none"),
  tightening_steps = 2,
  en_algorithm_opts
)
```

**Arguments**
- `max_it`: maximum number of iterations.
- `tightening`: how to make inner iterations more precise as the algorithm approaches a local minimum.
- `tightening_steps`: for adaptive tightening strategy, how often to tighten until the desired tolerance is attained.

**Value**
options for the MM algorithm.

---

**mscale**

**Compute the M-Scale of Centered Values**

**Description**
Compute the M-scale without centering the values.

**Usage**

```r
mscale(
  x,
  bdp = 0.25,
  cc = consistency_const(bdp, "bisquare"),
  opts = mscale_algorithm_options(),
  delta = deprecated(),
  rho = deprecated(),
  eps = deprecated(),
  maxit = deprecated()
)
```
mscale_algorithm_options

Options for the M-scale Estimation Algorithm

Description

Options for the M-scale Estimation Algorithm

Usage

mscale_algorithm_options(max_it = 200, eps = 1e-08)

Arguments

max_it maximum number of iterations.
eps numerical tolerance to check for convergence.

Value

options for the M-scale estimation algorithm.

See Also

Other functions to compute robust estimates of location and scale: mlocscale(), mloc(), tau_size()
Description

[Deprecated]

Additional options for computing penalized EN MM-estimates. Superseded by `mm_algorithm_options()` and options supplied directly to `pensem_cv()`.

Usage

```r
mstep_options(
  cc = 3.44,
  maxit = 1000,
  eps = 1e-06,
  adjust_bdp = FALSE,
  verbosity = 0,
  en_correction = TRUE
)
```

Arguments

- **cc**: ignored. Tuning constant for the M-estimator.
- **maxit**: maximum number of iterations allowed.
- **eps**: ignored. Numeric tolerance for convergence.
- **adjust_bdp**: ignored. Should the breakdown point be adjusted based on the effective degrees of freedom?
- **verbosity**: ignored. Verbosity of the algorithm.
- **en_correction**: ignored. Should the corrected EN estimator be used to choose the optimal lambda with CV. If TRUE, as by default, the estimator is "bias corrected".

Warning

Do not use this function in new code. It may be removed from future versions of the package.

See Also

Other deprecated functions: `deprecated_en_options.enpy()`, `initest_options()`, `pense_options()`, `pensem()`
Compute (Adaptive) Elastic Net S-Estimates of Regression

Description

Compute elastic net S-estimates (PENSE estimates) along a grid of penalization levels with optional penalty loadings for adaptive elastic net.

Usage

pense(
  x,
  y,
  alpha,
  nlambda = 50,
  nlambda_enpy = 10,
  lambda,
  lambda_min_ratio,
  enpy_lambda,
  penalty_loadings,
  intercept = TRUE,
  bdp = 0.25,
  cc,
  add_zero_based = TRUE,
  enpy_specific = FALSE,
  other_starts,
  eps = 1e-06,
  explore_solutions = 10,
  explore_tol = 0.1,
  explore_it = 20,
  max_solutions = 10,
  comparison_tol = sqrt(eps),
  sparse = FALSE,
  ncores = 1,
  standardize = TRUE,
  algorithm_opts = mm_algorithm_options(),
  mscale_opts = mscale_algorithm_options(),
  enpy_opts = enpy_options(),
  cv_k = deprecated(),
  cv_objective = deprecated(),
  ...
)

Arguments

x          n by p matrix of numeric predictors.
y          vector of response values of length n. For binary classification, y should be a factor with 2 levels.
alpha      elastic net penalty mixing parameter with 0 ≤ α ≤ 1. α = 1 is the LASSO penalty, and α = 0 the Ridge penalty. Can be a vector of several values, but α = 0 cannot be mixed with other values.
**pense**

- **nlambda**
  number of penalization levels.

- **nlambda_enpy**
  number of penalization levels where the EN-PY initial estimate is computed.

- **lambda**
  optional user-supplied sequence of penalization levels. If given and not NULL, nlambda and lambda_min_ratio are ignored.

- **lambda_min_ratio**
  Smallest value of the penalization level as a fraction of the largest level (i.e., the smallest value for which all coefficients are zero). The default depends on the sample size relative to the number of variables and alpha. If more observations than variables are available, the default is 1e-3 * alpha, otherwise 1e-2 * alpha.

- **enpy_lambda**
  optional user-supplied sequence of penalization levels at which EN-PY initial estimates are computed. If given and not NULL, nlambda_enpy is ignored.

- **penalty_loadings**
  a vector of positive penalty loadings (a.k.a. weights) for different penalization of each coefficient. Only allowed for alpha > 0.

- **intercept**
  include an intercept in the model.

- **bdp**
  desired breakdown point of the estimator, between 0 and 0.5. The actual breakdown point may be slightly larger/smaller to avoid instabilities of the S-loss.

- **cc**
  tuning constant for the S-estimator. Default is to chosen based on the breakdown point bdp. Does not affect the estimated coefficients, only the estimated scale of the residuals.

- **add_zero_based**
  also consider the 0-based regularization path. See details for a description.

- **enpy_specific**
  use the EN-PY initial estimates only at the penalization level they are computed for. See details for a description.

- **other_starts**
  a list of other staring points, created by starting_point(). If the output of enpy_initial_estimates() is given, the starting points will be shared among all penalization levels. Note that if a the starting point is specific to a penalization level, this penalization level is added to the grid of penalization levels (either the manually specified grid in lambda or the automatically generated grid of size nlambda). If standardize = TRUE, the starting points are also scaled.

- **eps**
  numerical tolerance.

- **explore_solutions**
  number of solutions to compute up to the desired precision eps.

- **explore_tol, explore_it**
  numerical tolerance and maximum number of iterations for exploring possible solutions. The tolerance should be (much) looser than eps to be useful, and the number of iterations should also be much smaller than the maximum number of iterations given via algorithm_OPTS.

- **max_solutions**
  only retain up to max_solutions unique solutions per penalization level.

- **comparison_tol**
  numeric tolerance to determine if two solutions are equal. The comparison is first done on the absolute difference in the value of the objective function at the solution If this is less than comparison_tol, two solutions are deemed equal if the squared difference of the intercepts is less than comparison_tol and the squared $L_2$ norm of the difference vector is less than comparison_tol.

- **sparse**
  use sparse coefficient vectors.

- **ncores**
  number of CPU cores to use in parallel. By default, only one CPU core is used. Not supported on all platforms, in which case a warning is given.
standardize logical flag to standardize the x variables prior to fitting the PENSE estimates. Coefficients are always returned on the original scale. This can fail for variables with a large proportion of a single value (e.g., zero-inflated data). In this case, either compute with standardize = FALSE or standardize the data manually.

algorithm_opts options for the MM algorithm to compute the estimates. See \texttt{mm_algorithm_options()} for details.

mscale_opts options for the M-scale estimation. See \texttt{mscale_algorithm_options()} for details.

enpy_opts options for the ENPY initial estimates, created with the \texttt{enpy_options()} function. See \texttt{enpy_initial_estimates()} for details.

cv_k, cv_objective deprecated and ignored. See \texttt{pense_cv()} for estimating prediction performance via cross-validation.

Value a list-like object with the following items

alpha the sequence of alpha parameters.

lambda a list of sequences of penalization levels, one per alpha parameter.

estimates a list of estimates. Each estimate contains the following information:

intercept intercept estimate.

beta beta (slope) estimate.

lambda penalization level at which the estimate is computed.

alpha alpha hyper-parameter at which the estimate is computed.

bdp chosen breakdown-point.

objf_value value of the objective function at the solution.

statuscode if > 0 the algorithm experienced issues when computing the estimate.

status optional status message from the algorithm.

bdp the actual breakdown point used.

call the original call.

Strategies for Using Starting Points

The function supports several different strategies to compute, and use the provided starting points for optimizing the PENSE objective function.

Starting points are computed internally but can also be supplied via other_starts. By default, starting points are computed internally by the EN-PY procedure for penalization levels supplied in enpy_lambda (or the automatically generated grid of length nlambda_enpy). By default, starting points computed by the EN-PY procedure are shared for all penalization levels in lambda (or the automatically generated grid of length nlambda). If the starting points should be specific to the penalization level the starting points’ penalization level, set the enpy_specific argument to TRUE.

In addition to EN-PY initial estimates, the algorithm can also use the "0-based" strategy if add_zero_based = TRUE (by default). Here, the 0-vector is used to start the optimization at the largest penalization level in lambda. At subsequent penalization levels, the solution at the previous penalization level is also used as starting point.

At every penalization level, all starting points are explored using the loose numerical tolerance explore_tol. Only the best explore_solutions are computed to the stringent numerical tolerance eps. Finally, only the best max_solutions are retained and carried forward as starting points for the subsequent penalization level.
**Deprecated Arguments**

Starting with version 2.0.0, cross-validation is performed by separate function `pense_cv()`. Arguments related cross-validation cause an error when supplied to `pense()`. Furthermore, the following arguments are deprecated as of version 2.0.0: `initial`, `warm_reset`, `cl`, `options`, `init_options`, `en_options`. If `pense()` is called with any of these arguments, warnings detail how to replace them.

**See Also**

- `pense_cv()` for selecting hyper-parameters via cross-validation.
- `coef.pense_fit()` for extracting coefficient estimates.
- `plot.pense_fit()` for plotting the regularization path.

Other functions to compute robust estimates: `regmest()`

**Examples**

```r
# Compute the PENSE regularization path for Freeny's revenue data
# (see ?freeny)
data(freeny)
x <- as.matrix(freeny[, 2:5])
regpath <- pense(x, freeny$y, alpha = 0.5)
plot(regpath)

# Extract the coefficients at a certain penalization level
coef(regpath, lambda = regpath$lambda[[1]][[40]])

# What penalization level leads to good prediction performance?
set.seed(123)
cv_results <- pense_cv(x, freeny$y, alpha = 0.5,
                        cv_repl = 2, cv_k = 4)
plot(cv_results, se_mult = 1)

# Extract the coefficients at the penalization level with
# smallest prediction error ...
coef(cv_results)
# ... or at the penalization level with prediction error
# statistically indistinguishable from the minimum.
coef(cv_results, lambda = '1-se')
```

---

**Description**

`pensem()` is a deprecated alias for `pense_cv()`.

**Usage**

`pensem(x, ...)`
Arguments

x  either a numeric matrix of predictor values, or a cross-validated PENSE fit from `pense_cv()`.

... ignored. See the section on deprecated parameters below.

See Also

Other deprecated functions: `deprecated_en_options`, `enpy()`, `initest_options()`, `mstep_options()`, `pense_options()`

pensem_cv

Compute Penalized Elastic Net M-Estimates from PENSE

Description

This is a convenience wrapper around `pense_cv()` and `regmest_cv()`, for the common use-case of computing a highly-robust S-estimate followed by a more efficient M-estimate using the scale of the residuals from the S-estimate.

Usage

```r
pensem_cv(x, ...)
```

## Default S3 method:
pensem_cv(
x, y,
alpha = 0.5,
nlambda = 50,
lambda_min_ratio,
lambda_m,
lambda_s,
standardize = TRUE,
penalty_loadings,
intercept = TRUE,
bdp = 0.25,
ncores = 1,
sparse = FALSE,
eps = 1e-06,
cc = 4.7,
cc_k = 5,
cc_repl = 1,
c1 = NULL,
cc_metric = c("tau_size", "mape", "rmspe"),
add_zero_based = TRUE,
explore_solutions = 10,
explore_tol = 0.1,
max_solutions = 10,
fit_all = TRUE,
comparison_tol = sqrt(eps),
```
algorithm_opts = mm_algorithm_options(),
mscale_opts = mscale_algorithm_options(),
nlambda_enpy = 10,
enpy_opts = enpy_options(),
...)

## S3 method for class 'pense_cvfit'
pense_cvfit(pensem_cv(
  x,
scale,
alpha,
nlambda = 50,
lambda_min_ratio,
lambda_m,
standardize = TRUE,
penalty_loadings,
intercept = TRUE,
bdp = 0.25,
ncores = 1,
sparse = FALSE,
eps = 1e-06,
cc = 4.7,
cv_k = 5,
cv_repl = 1,
c1 = NULL,
cv_metric = c("tau_size", "mape", "rmspe"),
add_zero_based = TRUE,
explore_solutions = 10,
explore_tol = 0.1,
max_solutions = 10,
fit_all = TRUE,
comparison_tol = sqrt(eps),
algorithm_opts = mm_algorithm_options(),
mscale_opts = mscale_algorithm_options(),
x_train,
y_train,
...)

Arguments

x either a numeric matrix of predictor values, or a cross-validated PENSE fit from pense_cv().

... ignored. See the section on deprecated parameters below.

y vector of response values of length n. For binary classification, y should be a factor with 2 levels.

alpha elastic net penalty mixing parameter with 0 ≤ α ≤ 1. alpha = 1 is the LASSO penalty, and alpha = 0 the Ridge penalty. Can be a vector of several values, but alpha = 0 cannot be mixed with other values.

nlambda number of penalization levels.
lambda_min_ratio

Smallest value of the penalization level as a fraction of the largest level (i.e., the smallest value for which all coefficients are zero). The default depends on the sample size relative to the number of variables and alpha. If more observations than variables are available, the default is $1e^{-3} \times \alpha$, otherwise $1e^{-2} \times \alpha$.

lambda_m, lambda_s

optional user-supplied sequence of penalization levels for the S- and M-estimates. If given and not NULL, nlambda and lambda_min_ratio are ignored for the respective estimate (S and/or M).

standardize

logical flag to standardize the $x$ variables prior to fitting the PENSE estimates. Coefficients are always returned on the original scale. This can fail for variables with a large proportion of a single value (e.g., zero-inflated data). In this case, either compute with standardize = FALSE or standardize the data manually.

penalty_loadings

a vector of positive penalty loadings (a.k.a. weights) for different penalization of each coefficient. Only allowed for $\alpha > 0$.

intercept

include an intercept in the model.

bdp

desired breakdown point of the estimator, between 0 and 0.5. The actual breakdown point may be slightly larger/smaller to avoid instabilities of the S-loss.

ncores

number of CPU cores to use in parallel. By default, only one CPU core is used. Not supported on all platforms, in which case a warning is given.

sparse

use sparse coefficient vectors.

eps

numerical tolerance.

cc

cutoff constant for Tukey’s bisquare $\rho$ function in the M-estimation objective function.

cv_k

number of folds per cross-validation.

cv_repl

number of cross-validation replications.

cl

a parallel cluster. Can only be used in combination with ncores = 1.

cv_metric

either a string specifying the performance metric to use, or a function to evaluate prediction errors in a single CV replication. If a function, the number of arguments define the data the function receives. If the function takes a single argument, it is called with a single numeric vector of prediction errors. If the function takes two or more arguments, it is called with the predicted values as first argument and the true values as second argument. The function must always return a single numeric value quantifying the prediction performance. The order of the given values corresponds to the order in the input data.

add_zero_based

also consider the 0-based regularization path. See details for a description.

explore_solutions

number of solutions to compute up to the desired precision eps.

explore_tol

numerical tolerance and maximum number of iterations for exploring possible solutions. The tolerance should be (much) looser than eps to be useful, and the number of iterations should also be much smaller than the maximum number of iterations given via algorithm_opts.

max_solutions

only retain up to max_solutions unique solutions per penalization level.

fit_all

If TRUE, fit the model for all penalization levels. Can also be any combination of "min" and "{x}-se", in which case only models at the penalization level with smallest average CV accuracy, or within {x} standard errors, respectively. Setting fit_all to FALSE is equivalent to "min". Applies to all alpha value.
comparison_tol numeric tolerance to determine if two solutions are equal. The comparison is first done on the absolute difference in the value of the objective function at the solution. If this is less than comparison_tol, two solutions are deemed equal if the squared difference of the intercepts is less than comparison_tol and the squared $L_2$ norm of the difference vector is less than comparison_tol.

algorithm_opts options for the MM algorithm to compute the estimates. See `mm_algorithm_options()` for details.

mscale_opts options for the M-scale estimation. See `mscale_algorithm_options()` for details.

nlambda_enpy number of penalization levels where the EN-PY initial estimate is computed.

enpy_opts options for the ENPY initial estimates, created with the `enpy_options()` function. See `enpy_initial_estimates()` for details.

scale initial scale estimate to use in the M-estimation. By default the S-scale from the PENSE fit is used.

x_train, y_train override arguments x and y as provided in the call to `pense_cv()`. This is useful if the arguments in the `pense_cv()` call are not available in the current environment.

Details

The built-in CV metrics are

"tau_size" $\tau$-size of the prediction error, computed by `tau_size()` (default).

"mape" Median absolute prediction error.

"rmspe" Root mean squared prediction error.

"auroc" Area under the receiver operator characteristic curve (actually 1 - AUROC). Only sensible for binary responses.

Value

an object of cross-validated regularized M-estimates as returned from `regmest_cv()`.

See Also

`pense_cv()` to compute the starting S-estimate.

Other functions to compute robust estimates with CV: `pense_cv()`, `regmest_cv()`
Usage

```r
pense_cv(
  x,
  y,
  standardize = TRUE,
  lambda,
  cv_k,
  cv_repl = 1,
  cv_metric = c("tau_size", "mape", "rmspe", "auroc"),
  fit_all = TRUE,
  cl = NULL,
  ...
)

adapense_cv(x, y, alpha, alpha_preliminary = 0, exponent = 1, ...)
```

Arguments

- **x**: `n` by `p` matrix of numeric predictors.
- **y**: vector of response values of length `n`. For binary classification, `y` should be a factor with 2 levels.
- **standardize**: whether to standardize the `x` variables prior to fitting the PENSE estimates. Can also be set to "cv_only", in which case the input data is not standardized, but the training data in the CV folds is scaled to match the scaling of the input data. Coefficients are always returned on the original scale. This can fail for variables with a large proportion of a single value (e.g., zero-inflated data). In this case, either compute with `standardize = FALSE` or standardize the data manually.
- **lambda**: optional user-supplied sequence of penalization levels. If given and not `NULL`, `nlambda` and `lambda_min_ratio` are ignored.
- **cv_k**: number of folds per cross-validation.
- **cv_repl**: number of cross-validation replications.
- **cv_metric**: either a string specifying the performance metric to use, or a function to evaluate prediction errors in a single CV replication. If a function, the number of arguments define the data the function receives. If the function takes a single argument, it is called with a single numeric vector of prediction errors. If the function takes two or more arguments, it is called with the predicted values as first argument and the true values as second argument. The function must always return a single numeric value quantifying the prediction performance. The order of the given values corresponds to the order in the input data.
- **fit_all**: If `TRUE`, fit the model for all penalization levels. Can also be any combination of "min" and "{x}-se", in which case only models at the penalization level with smallest average CV accuracy, or within `{x}` standard errors, respectively. Setting `fit_all` to `FALSE` is equivalent to "min". Applies to all `alpha` value.
- **cl**: a parallel cluster. Can only be used in combination with `ncores = 1`.
- **...**: Arguments passed on to `pense`
alpha. If more observations than variables are available, the default is $10^{-3} \times \alpha$, otherwise $10^{-2} \times \alpha$.

nlambda_enpy  number of penalization levels where the EN-PY initial estimate is computed.

penalty_loadings  a vector of positive penalty loadings (a.k.a. weights) for different penalization of each coefficient. Only allowed for $\alpha > 0$.

enpy_lambda  optional user-supplied sequence of penalization levels at which EN-PY initial estimates are computed. If given and not NULL, nlambda_enpy is ignored.

other_starts  a list of other staring points, created by starting_point(). If the output of enpy_initial_estimates() is given, the starting points will be shared among all penalization levels. Note that if a the starting point is specific to a penalization level, this penalization level is added to the grid of penalization levels (either the manually specified grid in lambda or the automatically generated grid of size nlambda). If standardize = TRUE, the starting points are also scaled.

intercept  include an intercept in the model.

bdp  desired breakdown point of the estimator, between 0 and 0.5. The actual breakdown point may be slightly larger/smaller to avoid instabilities of the S-loss.

cc  tuning constant for the S-estimator. Default is to chosen based on the breakdown point bdp. Does not affect the estimated coefficients, only the estimated scale of the residuals.

eps  numerical tolerance.

explore_solutions  number of solutions to compute up to the desired precision eps.

explore_tol  numerical tolerance and maximum number of iterations for exploring possible solutions. The tolerance should be (much) looser than eps to be useful, and the number of iterations should also be much smaller than the maximum number of iterations given via algorithm_opts.

explore_it  numerical tolerance and maximum number of iterations for exploring possible solutions. The tolerance should be (much) looser than eps to be useful, and the number of iterations should also be much smaller than the maximum number of iterations given via algorithm_opts.

max_solutions  only retain up to max_solutions unique solutions per penalization level.

comparison_tol  numeric tolerance to determine if two solutions are equal. The comparison is first done on the absolute difference in the value of the objective function at the solution If this is less than comparison_tol, two solutions are deemed equal if the squared difference of the intercepts is less than comparison_tol and the squared $L_2$ norm of the difference vector is less than comparison_tol.

add_zero_based  also consider the 0-based regularization path. See details for a description.

enpy_specific  use the EN-PY initial estimates only at the penalization level they are computed for. See details for a description.

sparse  use sparse coefficient vectors.

ncores  number of CPU cores to use in parallel. By default, only one CPU core is used. Not supported on all platforms, in which case a warning is given.
algorithm_opts options for the MM algorithm to compute the estimates. See `mm_algorithm_options()` for details.

mscale_opts options for the M-scale estimation. See `mscale_algorithm_options()` for details.

enpy_opts options for the ENPY initial estimates, created with the `enpy_options()` function. See `enpy_initial_estimates()` for details.

cv_objective deprecated and ignored. See `pense_cv()` for estimating prediction performance via cross-validation.

alpha elastic net penalty mixing parameter with $0 \leq \alpha \leq 1$. $\alpha = 1$ is the LASSO penalty, and $\alpha = 0$ the Ridge penalty. Can be a vector of several values, but $\alpha = 0$ cannot be mixed with other values.

alpha_preliminary alpha parameter for the preliminary estimate.

exponent the exponent for computing the penalty loadings based on the preliminary estimate.

Details

The built-in CV metrics are

"tau_size" $\tau$-size of the prediction error, computed by `tau_size()` (default).

"mape" Median absolute prediction error.

"rmspe" Root mean squared prediction error.

"auroc" Area under the receiver operator characteristic curve (actually 1 - AUROC). Only sensible for binary responses.

`adapense_cv()` is a convenience wrapper which performs 3 steps:

1. compute preliminary estimates via `pense_cv(..., alpha = alpha_preliminary)`,
2. computes the penalty loadings from the estimate beta with best prediction performance by
   `adapense_loadings = 1 / abs(beta)^exponent`, and
3. compute the adaptive PENSE estimates via `pense_cv(..., penalty_loadings = adapense_loadings)`.

Value

a list-like object with the same components as returned by `pense()`, plus the following:

cvres data frame of average cross-validated performance.

a list-like object as returned by `pense_cv()` plus the following

preliminary the CV results for the preliminary estimate.

exponent exponent used to compute the penalty loadings.

penalty_loadings the penalty loadings used for the adaptive PENSE estimate.

See Also

`pense()` for computing regularized S-estimates without cross-validation.

`coef.pense_cvfit()` for extracting coefficient estimates.

`plot.pense_cvfit()` for plotting the CV performance or the regularization path.

Other functions to compute robust estimates with CV: `pensem_cv()`, `regmest_cv()`

Other functions to compute robust estimates with CV: `pensem_cv()`, `regmest_cv()`

Examples

```r
# Compute the adaptive PENSE regularization path for Freeny's
# revenue data (see ?freeny)
data(freeny)
x <- as.matrix(freeny[, 2:5])

## Either use the convenience function directly ...
set.seed(123)
ada_convenience <- adapense_cv(x, freeny$y, alpha = 0.5,
                               cv_repl = 2, cv_k = 4)

## ... or compute the steps manually:
# Step 1: Compute preliminary estimates with CV
set.seed(123)
preliminary_estimate <- pense_cv(x, freeny$y, alpha = 0,
                                 cv_repl = 2, cv_k = 4)
plot(preliminary_estimate, se_mult = 1)

# Step 2: Use the coefficients with best prediction performance
# to define the penalty loadings:
prelim_coefs <- coef(preliminary_estimate, lambda = 'min')
pen_loadings <- 1 / abs(prelim_coefs[-1])

# Step 3: Compute the adaptive PENSE estimates and estimate
# their prediction performance.
set.seed(123)
ada_manual <- pense_cv(x, freeny$y, alpha = 0.5,
                       cv_repl = 2, cv_k = 4,
                       penalty_loadings = pen_loadings)

# Visualize the prediction performance and coefficient path of
# the adaptive PENSE estimates (manual vs. automatic)
def.par <- par(no.readonly = TRUE)
layout(matrix(1:4, ncol = 2, byrow = TRUE))
plot(ada_convenience$preliminary)
plot(preliminary_estimate)
plot(ada_convenience)
plot(ada_manual)
par(def.par)
```

pense_options

```

pense_options

 Deprecated

Description

[Deprecated]

Additional options for computing penalized EN S-estimates. Superseded by `mm_algorithm_options()` and options supplied directly to `pense()`.

Usage

pense_options(
  delta = 0.25,
)
Arguments

- **delta**: desired breakdown point of the resulting estimator.
- **maxit**: maximum number of iterations allowed.
- **eps**: numeric tolerance for convergence.
- **mscale_eps, mscale_maxit**: maximum number of iterations and numeric tolerance for the M-scale.
- **verbosity**: ignored. Verbosity of the algorithm.
- **cc**: ignored. Tuning constant for the S-estimator. Default is chosen based on the breakdown point delta. Should never have to be changed.
- **en_correction**: ignored. Should the corrected EN estimator be used to choose the optimal lambda with CV? If TRUE, as by default, the estimator is "bias corrected".

Warning

Do not use this function in new code. It may be removed from future versions of the package.

See Also

Other deprecated functions: `deprecated_en_options`, `enpy()`, `initest_options()`, `mstep_options()`, `pensem()`

---

**plot.pense_cvfit**  
Plot Method for Penalized Estimates With Cross-Validation

Description

Plot the cross-validation performance or the coefficient path for fitted penalized elastic net S- or LS-estimates of regression.

Usage

```r
## S3 method for class 'pense_cvfit'
plot(x, what = c("cv", "coef.path"), alpha = NULL, se_mult = 1, ...)```
plot.pense_fit

Arguments

- **x**: fitted estimates with cross-validation information.
- **what**: plot either the CV performance or the coefficient path.
- **alpha**: If `what = "cv"`, only CV performance for fits with matching `alpha` are plotted. In case `alpha` is missing or `NULL`, all fits in `x` are plotted. If `what = "coef.path"`, plot the coefficient path for the fit with the given hyper-parameter value or, in case `alpha` is missing, for the first value in `x$alpha`.
- **se_mult**: if plotting CV performance, multiplier of the estimated SE.
- **...**: currently ignored.

See Also

Other functions for plotting and printing: `plot.pense_fit()`, `prediction_performance()`, `summary.pense_cvfit()`

Examples

```r
# Compute the PENSE regularization path for Freeny's revenue data
# (see ?freeny)
data(freeny)
x <- as.matrix(freeny[, 2:5])
regpath <- pense(x, freeny$y, alpha = 0.5)
plot(regpath)

# Extract the coefficients at a certain penalization level
coef(regpath, lambda = regpath$lambda[[1]][[40]])

# What penalization level leads to good prediction performance?
set.seed(123)
cv_results <- pense_cv(x, freeny$y, alpha = 0.5,
cv_repl = 2, cv_k = 4)
plot(cv_results, se_mult = 1)

# Extract the coefficients at the penalization level with
# smallest prediction error ...
coef(cv_results)
# ... or at the penalization level with prediction error
# statistically indistinguishable from the minimum.
coef(cv_results, lambda = '1-se')
```

---

**plot.pense_fit**

**Plot Method for Penalized Estimates**

Description

Plot the coefficient path for fitted penalized elastic net S- or LS-estimates of regression.

Usage

```r
## S3 method for class 'pense_fit'
plot(x, alpha, ...)
```
predict.pense_cvfit

Description

Predict response values using a PENSE (or LS-EN) regularization path with hyper-parameters chosen by cross-validation.

Usage

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

lambda = "min",
se_mult = 1,
exact = deprecated(),
correction = deprecated(),
...  
)

Arguments

object  
PENSE with cross-validated hyper-parameters to extract coefficients from.

newdata  
an optional matrix of new predictor values. If missing, the fitted values are computed.

alpha  
Either a single number or NULL (default). If given, only fits with the given alpha value are considered. If lambda is a numeric value and object was fit with multiple alpha values and no value is provided, the first value in object$alpha is used with a warning.

lambda  
either a string specifying which penalty level to use ("min", "se", "{m}-se") or a single numeric value of the penalty parameter. See details.

se_mult  
If lambda = "se", the multiple of standard errors to tolerate.

exact  
deprecated. Always gives a warning if lambda is not part of the fitted sequence and coefficients are interpolated.

correction  
defunct.

Value

a numeric vector of residuals for the given penalization level.

Hyper-parameters

If lambda = "{m}-se" and object contains fitted estimates for every penalization level in the sequence, use the fit the most parsimonious model with prediction performance statistically indistinguishable from the best model. This is determined to be the model with prediction performance within m * cv_se from the best model. If lambda = "se", the multiplier m is taken from se_mult.

By default all alpha hyper-parameters available in the fitted object are considered. This can be overridden by supplying one or multiple values in parameter alpha. For example, if lambda = "1-se" and alpha contains two values, the "1-SE" rule is applied individually for each alpha value, and the fit with the better prediction error is considered.

In case lambda is a number and object was fit for several alpha hyper-parameters, alpha must also be given, or the first value in object$alpha is used with a warning.

See Also

Other functions for extracting components: coef.pense_cvfit(), coef.pense_fit(), predict.pense_fit(), residuals.pense_cvfit(), residuals.pense_fit()
Examples

# Compute the LS-EN regularization path for Freeny's revenue data
# (see ?freeny)
data(freeny)
x <- as.matrix(freeny[, 2:5])
regpath <- elnet(x, freeny$y, alpha = 0.75)

# Predict the response using a specific penalization level
predict(regpath, newdata = freeny[1:5, 2:5],
lambda = regpath$lambda[1][[10]])

# Extract the residuals at a certain penalization level
residuals(regpath, lambda = regpath$lambda[1][[5]])

# Select penalization level via cross-validation
set.seed(123)
cv_results <- elnet_cv(x, freeny$y, alpha = 0.5,
cv_repl = 10, cv_k = 4)

# Predict the response using the "best" penalization level
predict(cv_results, newdata = freeny[1:5, 2:5])

# Extract the residuals at the "best" penalization level
cv_results
residuals(cv_results)
# Extract the residuals at a more parsimonious penalization level
residuals(cv_results, lambda = "1.5-se")

predict.pense_fit  Predict Method for PENSE Fits

Description

Predict response values using a PENSE (or LS-EN) regularization path fitted by `pense()`, `regmest()` or `elnet()`.

Usage

```r
## S3 method for class 'pense_fit'
predict(
  object, 
  newdata, 
  alpha = NULL, 
  lambda, 
  exact = deprecated(), 
  correction = deprecated(), 
  ... 
)
```

Arguments

- `object`  PENSE regularization path to extract residuals from.
predict.pense_fit

newdata an optional matrix of new predictor values. If missing, the fitted values are computed.

alpha Either a single number or NULL (default). If given, only fits with the given alpha value are considered. If object was fit with multiple alpha values, and no value is provided, the first value in object$alpha is used with a warning.

lambda a single number for the penalty level.

exact defunct Always gives a warning if lambda is not part of the fitted sequence and coefficients need to be interpolated.

correction defunct.

... currently not used.

Value

a numeric vector of residuals for the given penalization level.

See Also

Other functions for extracting components: coef.pense_cvfit(), coef.pense_fit(), predict.pense_cvfit(), residuals.pense_cvfit(), residuals.pense_fit()

Examples

# Compute the LS-EN regularization path for Freeny's revenue data
# (see ?freeny)
data(freeny)
x <- as.matrix(freeny[, 2:5])
regpath <- elnet(x, freeny$y, alpha = 0.75)

# Predict the response using a specific penalization level
predict(regpath, newdata = freeny[1:5, 2:5],
        lambda = regpath$lambda[[1]][[10]])

# Extract the residuals at a certain penalization level
residuals(regpath, lambda = regpath$lambda[[1]][[5]])

# Select penalization level via cross-validation
set.seed(123)
cv_results <- elnet_cv(x, freeny$y, alpha = 0.5,
                        cv_repl = 10, cv_k = 4)

# Predict the response using the "best" penalization level
predict(cv_results, newdata = freeny[1:5, 2:5])

# Extract the residuals at the "best" penalization level
residuals(cv_results)
# Extract the residuals at a more parsimonious penalization level
residuals(cv_results, lambda = "1.5-se")
prediction_performance

Prediction Performance of Adaptive PENSE Fits

Description

Extract the prediction performance of one or more (adaptive) PENSE fits.

Usage

prediction_performance(..., alpha = NULL, lambda = "min", se_mult = 1)

## S3 method for class 'pense_pred_perf'
predict(x, ...)  

Arguments

... one or more (adaptive) PENSE fits with cross-validation information.

alpha Either a numeric vector or NULL (default). If given, only fits with the given
          alpha value are considered. If lambda is a numeric value and object was fit
          with multiple alpha values, the parameter alpha must not be missing.

lambda either a string specifying which penalty level to use ("min", "se", "{x}-se") or
          a single numeric value of the penalty parameter. See details.

se_mult If lambda = "se", the multiple of standard errors to tolerate.

x an object with information on prediction performance created with prediction_performance().

Details

If lambda = "se" and the cross-validation was performed with multiple replications, use the penalty
level with prediction performance within se_mult of the best prediction performance.

Value

a data frame with details about the prediction performance of the given PENSE fits. The data frame
has a custom print method summarizing the prediction performances.

See Also

summary.pense_cvfit() for a summary of the fitted model.

Other functions for plotting and printing: plot.pense_cvfit(), plot.pense_fit(), summary.pense_cvfit()
**Description**

Compute Principal Sensitivity Components for Elastic Net Regression

**Usage**

```r
prinsens(
  x,
  y,
  alpha,
  lambda,
  intercept = TRUE,
  penalty_loadings,
  en_algorithm_opts,
  eps = 1e-06,
  sparse = FALSE,
  ncores = 1L,
  method = deprecated()
)
```

**Arguments**

- `x`: n by p matrix of numeric predictors.
- `y`: vector of response values of length n.
- `alpha`: elastic net penalty mixing parameter with $0 \leq \alpha \leq 1$. $\alpha = 1$ is the LASSO penalty, and $\alpha = 0$ the Ridge penalty. Can be a vector of several values, but $\alpha = 0$ cannot be mixed with other values.
- `lambda`: optional user-supplied sequence of penalization levels. If given and not NULL, nlambda and lambda_min_ratio are ignored.
- `intercept`: include an intercept in the model.
- `penalty_loadings`: a vector of positive penalty loadings (a.k.a. weights) for different penalization of each coefficient. Only allowed for $\alpha > 0$.
- `eps`: numerical tolerance.
- `sparse`: use sparse coefficient vectors.
- `ncores`: number of CPU cores to use in parallel. By default, only one CPU core is used. Not supported on all platforms, in which case a warning is given.
- `method`: defunct. PSCs are always computed for EN estimates. For the PY procedure for unpenalized estimation use package pyinit.

**Value**

a list of principal sensitivity components, one per element in lambda. Each PSC is itself a list with items lambda, alpha, and pscs.
regmest

Compute (Adaptive) Elastic Net M-Estimates of Regression

Description

Compute elastic net M-estimates along a grid of penalization levels with optional penalty loadings for adaptive elastic net.

Usage

regmest(
  x,
  y,
  alpha,
  nlambda = 50,
  lambda,
  lambda_min_ratio,
  scale,
  starting_points,
  penalty_loadings,
  intercept = TRUE,
  cc = 4.7,
  eps = 1e-06,
  explore_solutions = 10,
  explore_tol = 0.1,
  max_solutions = 10,
  comparison_tol = sqrt(eps),
  sparse = FALSE,
  ncores = 1,
  standardize = TRUE,
  algorithm_opts = mm_algorithm_options(),
  add_zero_based = TRUE,
  mscale_bdp = 0.25,
  mscale_opts = mscale_algorithm_options()
)

References


See Also

Other functions for initial estimates: *enpy_initial_estimates()*, *starting_point()*
Arguments

**x**
$n \times p$ matrix of numeric predictors.

**y**
vector of response values of length $n$. For binary classification, $y$ should be a factor with 2 levels.

**alpha**
elastic net penalty mixing parameter with $0 \leq \alpha \leq 1$. $alpha = 1$ is the LASSO penalty, and $alpha = 0$ the Ridge penalty.

**nlambda**
number of penalization levels.

**lambda**
optional user-supplied sequence of penalization levels. If given and not NULL, nlambda and lambda_min_ratio are ignored.

**lambda_min_ratio**
Smallest value of the penalization level as a fraction of the largest level (i.e., the smallest value for which all coefficients are zero). The default depends on the sample size relative to the number of variables and alpha. If more observations than variables are available, the default is $1e^{-3} \times alpha$, otherwise $1e^{-2} \times alpha$.

**scale**
fixed scale of the residuals.

**starting_points**
a list of starting points, created by `starting_point()`. The starting points are shared among all penalization levels.

**penalty_loadings**
a vector of positive penalty loadings (a.k.a. weights) for different penalization of each coefficient. Only allowed for $alpha > 0$.

**intercept**
include an intercept in the model.

**cc**
cutoff constant for Tukey’s bisquare $\rho$ function.

**eps**
umerical tolerance.

**explore_solutions**
number of solutions to compute up to the desired precision $eps$.

**explore_tol**
umerical tolerance for exploring possible solutions. Should be (much) looser than $eps$ to be useful.

**max_solutions**
only retain up to $max_solutions$ unique solutions per penalization level.

**comparison_tol**
numeric tolerance to determine if two solutions are equal. The comparison is first done on the absolute difference in the value of the objective function at the solution. If this is less than $comparison_tol$, two solutions are deemed equal if the squared difference of the intercepts is less than $comparison_tol$ and the squared $L_2$ norm of the difference vector is less than $comparison_tol$.

**sparse**
use sparse coefficient vectors.

**ncores**
number of CPU cores to use in parallel. By default, only one CPU core is used. Not supported on all platforms, in which case a warning is given.

**standardize**
logical flag to standardize the $x$ variables prior to fitting the M-estimates. Coefficients are always returned on the original scale. This can fail for variables with a large proportion of a single value (e.g., zero-inflated data). In this case, either compute with `standardize = FALSE` or standardize the data manually.

**algorithm_opts**
options for the MM algorithm to compute estimates. See `mm_algorithm_options()` for details.

**add_zero_based**
also consider the 0-based regularization path in addition to the given starting points.

**mscale_bdp, mscale_opts**
options for the M-scale estimate used to standardize the predictors (if `standardize = TRUE`).
Value

a list-like object with the following items

- **alpha**: the sequence of alpha parameters.
- **lambda**: a list of sequences of penalization levels, one per alpha parameter.
- **scale**: the used scale of the residuals.
- **estimates**: a list of estimates. Each estimate contains the following information:
  - **intercept**: intercept estimate.
  - **beta**: beta (slope) estimate.
  - **lambda**: penalization level at which the estimate is computed.
  - **alpha**: alpha hyper-parameter at which the estimate is computed.
  - **objf_value**: value of the objective function at the solution.
  - **statuscode**: if > 0 the algorithm experienced issues when computing the estimate.
  - **status**: optional status message from the algorithm.
- **call**: the original call.

See Also

- `regmest_cv()` for selecting hyper-parameters via cross-validation.
- `coef.pense_fit()` for extracting coefficient estimates.
- `plot.pense_fit()` for plotting the regularization path.

Other functions to compute robust estimates: `pense()`

---

**Description**

Perform (repeated) K-fold cross-validation for *regmest()*.

*adamest_cv()* is a convenience wrapper to compute adaptive elastic-net M-estimates.

**Usage**

```r
regmest_cv(
  x,
  y,
  standardize = TRUE,
  lambda,
  cv_k,
  cv_repl = 1,
  cv_metric = c("tau_size", "mape", "rmspe", "auroc"),
  fit_all = TRUE,
  cl = NULL,
  ...
)
```

```r
adamest_cv(x, y, alpha, alpha_preliminary = 0, exponent = 1, ...)
```
Arguments

x  
n by p matrix of numeric predictors.
y  
vector of response values of length n. For binary classification, y should be a factor with 2 levels.

standardize  
whether to standardize the x variables prior to fitting the PENSE estimates. Can also be set to "cv_only", in which case the input data is not standardized, but the training data in the CV folds is scaled to match the scaling of the input data. Coefficients are always returned on the original scale. This can fail for variables with a large proportion of a single value (e.g., zero-inflated data). In this case, either compute with standardize = FALSE or standardize the data manually.

lambda  
optional user-supplied sequence of penalization levels. If given and not NULL, nlambda and lambda_min_ratio are ignored.

cv_k  
number of folds per cross-validation.

cv_repl  
number of cross-validation replications.

cv_metric  
either a string specifying the performance metric to use, or a function to evaluate prediction errors in a single CV replication. If a function, the number of arguments define the data the function receives. If the function takes a single argument, it is called with a single numeric vector of prediction errors. If the function takes two or more arguments, it is called with the predicted values as first argument and the true values as second argument. The function must always return a single numeric value quantifying the prediction performance. The order of the given values corresponds to the order in the input data.

fit_all  
If TRUE, fit the model for all penalization levels. Can also be any combination of "min" and "{x}-se", in which case only models at the penalization level with smallest average CV accuracy, or within {x} standard errors, respectively. Setting fit_all to FALSE is equivalent to "min". Applies to all alpha value.

cl  
a parallel cluster. Can only be used in combination with ncores = 1.

... Arguments passed on to regmest

scale  
fixed scale of the residuals.
nlambda  
number of penalization levels.

lambda_min_ratio  
Smallest value of the penalization level as a fraction of the largest level (i.e., the smallest value for which all coefficients are zero). The default depends on the sample size relative to the number of variables and alpha. If more observations than variables are available, the default is 1e-3 * alpha, otherwise 1e-2 * alpha.

penalty_loadings  
a vector of positive penalty loadings (a.k.a. weights) for different penalization of each coefficient. Only allowed for alpha > 0.

starting_points  
a list of staring points, created by starting_point(). The starting points are shared among all penalization levels.

intercept  
include an intercept in the model.

add_zero_based  
also consider the 0-based regularization path in addition to the given starting points.

c  
cutoff constant for Tukey’s bisquare ρ function.

eps  
numerical tolerance.

explore_solutions  
number of solutions to compute up to the desired precision eps.

explore_tol  
numerical tolerance for exploring possible solutions. Should be (much) looser than eps to be useful.
max_solutions only retain up to max_solutions unique solutions per penal-
ization level.
comparison_tol numeric tolerance to determine if two solutions are equal.
The comparison is first done on the absolute difference in the value of the
objective function at the solution. If this is less than comparison_tol, two
solutions are deemed equal if the squared difference of the intercepts is less
than comparison_tol and the squared $L_2$ norm of the difference vector is
less than comparison_tol.
sparse use sparse coefficient vectors.
ncores number of CPU cores to use in parallel. By default, only one CPU core
is used. Not supported on all platforms, in which case a warning is given.
algorithm_opts options for the MM algorithm to compute estimates. See
mm_algorithm_options() for details.
mscale_bdp options for the M-scale estimate used to standardize the predictors
(if standardize = TRUE).
mscale_opts options for the M-scale estimate used to standardize the predic-
tors (if standardize = TRUE).
alpha elastic net penalty mixing parameter with $0 \leq \alpha \leq 1$. alpha = 1 is the LASSO
penalty, and alpha = 0 the Ridge penalty.
alpha_preliminary alpha parameter for the preliminary estimate.
exponent the exponent for computing the penalty loadings based on the preliminary esti-
mate.

Details
The built-in CV metrics are
"tau_size" $\tau$-size of the prediction error, computed by tau_size() (default).
"mape" Median absolute prediction error.
"rmspe" Root mean squared prediction error.
"auroc" Area under the receiver operator characteristic curve (actually 1 - AUROC). Only sensible
for binary responses.
adamest_cv() is a convenience wrapper which performs 3 steps:
1. compute preliminary estimates via regmest_cv(...,alpha = alpha_preliminary).
2. computes the penalty loadings from the estimate beta with best prediction performance by
adamest_loadings = 1 / abs(beta)^exponent, and
3. compute the adaptive PENSE estimates via regmest_cv(...,penalty_loadings = adamest_loadings).

Value
a list-like object as returned by regmest(), plus the following components:
cvres data frame of average cross-validated performance.
a list-like object as returned by adamest_cv() plus the following components:
exponent value of the exponent.
preliminary CV results for the preliminary estimate.
penalty_loadings penalty loadings used for the adaptive elastic net M-estimate.
See Also

regmest() for computing regularized S-estimates without cross-validation.

coef.pense_cvfit() for extracting coefficient estimates.

plot.pense_cvfit() for plotting the CV performance or the regularization path.

Other functions to compute robust estimates with CV: pense_cv(), pensem_cv()
Other functions to compute robust estimates with CV: pense_cv(), pensem_cv()

Examples

# Compute the adaptive PENSE regularization path for Freeny's # revenue data (see ?freeny)
data(freeny)
x <- as.matrix(freeny[ , 2:5])

## Either use the convenience function directly ...
set.seed(123)
ada_convenience <- adapense_cv(x, freeny$y, alpha = 0.5,
cv_repl = 2, cv_k = 4)

## ... or compute the steps manually:
# Step 1: Compute preliminary estimates with CV
set.seed(123)
preliminary_estimate <- pense_cv(x, freeny$y, alpha = 0,
cv_repl = 2, cv_k = 4)
plot(preliminary_estimate, se_mult = 1)

# Step 2: Use the coefficients with best prediction performance # to define the penalty loadings:
prelim_coefs <- coef(preliminary_estimate, lambda = 'min')
pen_loadings <- 1 / abs(prelim_coefs[-1])

# Step 3: Compute the adaptive PENSE estimates and estimate # their prediction performance.
set.seed(123)
ada_manual <- pense_cv(x, freeny$y, alpha = 0.5,
cv_repl = 2, cv_k = 4,
penalty_loadings = pen_loadings)

# Visualize the prediction performance and coefficient path of # the adaptive PENSE estimates (manual vs. automatic)
def.par <- par(no.readonly = TRUE)
layout(matrix(1:4, ncol = 2, byrow = TRUE))
plot(ada_convenience$preliminary)
plot(ada_convenience)
plot(ada_manual)
par(def.par)
Description

Extract residuals from a PENSE (or LS-EN) regularization path with hyper-parameters chosen by cross-validation.

Usage

```r
## S3 method for class 'pense_cvfit'
residuals(
  object,
  alpha = NULL,
  lambda = "min",
  se_mult = 1,
  exact = deprecated(),
  correction = deprecated(),
  ...
)
```

Arguments

- **object**: PENSE with cross-validated hyper-parameters to extract coefficients from.
- **alpha**: Either a single number or NULL (default). If given, only fits with the given alpha value are considered. If lambda is a numeric value and object was fit with multiple alpha values and no value is provided, the first value in object$alpha is used with a warning.
- **lambda**: either a string specifying which penalty level to use ("min", "se", "{m}-se") or a single numeric value of the penalty parameter. See details.
- **se_mult**: If lambda = "se", the multiple of standard errors to tolerate.
- **exact**: deprecated. Always gives a warning if lambda is not part of the fitted sequence and coefficients are interpolated.
- **correction**: defunct.
- **...**: currently not used.

Value

a numeric vector of residuals for the given penalization level.

Hyper-parameters

If lambda = "+(m)-se" and object contains fitted estimates for every penalization level in the sequence, use the fit the most parsimonious model with prediction performance statistically indistinguishable from the best model. This is determined to be the model with prediction performance within \( m \times \text{cv.se} \) from the best model. If lambda = "se", the multiplier \( m \) is taken from se_mult.

By default all alpha hyper-parameters available in the fitted object are considered. This can be overridden by supplying one or multiple values in parameter alpha. For example, if lambda = "1-se" and alpha contains two values, the "1-SE" rule is applied individually for each alpha value, and the fit with the better prediction error is considered.

In case lambda is a number and object was fit for several alpha hyper-parameters, alpha must also be given, or the first value in object$alpha is used with a warning.
See Also

Other functions for extracting components: `coef.pense_cvfit()`, `coef.pense_fit()`, `predict.pense_cvfit()`, `predict.pense_fit()`, `residuals.pense_fit()`

Examples

# Compute the LS-EN regularization path for Freeny's revenue data  
# (see ?freeny)

data(freeny)

x <- as.matrix(freeny[, 2:5])

regpath <- elnet(x, freeny$y, alpha = 0.75)

# Predict the response using a specific penalization level

predict(regpath, newdata = freeny[1:5, 2:5],
        lambda = regpath$lambda[[1]][[10]])

# Extract the residuals at a certain penalization level

residuals(regpath, lambda = regpath$lambda[[1]][[5]])

# Select penalization level via cross-validation

set.seed(123)

cv_results <- elnet_cv(x, freeny$y, alpha = 0.5,
                       cv_repl = 10, cv_k = 4)

# Predict the response using the "best" penalization level

predict(cv_results, newdata = freeny[1:5, 2:5])

# Extract the residuals at the "best" penalization level

residuals(cv_results)

# Extract the residuals at a more parsimonious penalization level

residuals(cv_results, lambda = "1.5-se")

residuals.pense_fit  

Extract Residuals

Description

Extract residuals from a PENSE (or LS-EN) regularization path fitted by `pense()`, `regmest()` or `elnet()`.

Usage

```r
## S3 method for class 'pense_fit'
residuals(
  object,
  alpha = NULL,
  lambda,
  exact = deprecated(),
  correction = deprecated(),
  ...
)
```
Arguments

object  
  PENSE regularization path to extract residuals from.

alpha  
  Either a single number or NULL (default). If given, only fits with the given alpha value are considered. If object was fit with multiple alpha values, and no value is provided, the first value in object$alpha is used with a warning.

lambda  
  a single number for the penalty level.

exact  
  defunct Always gives a warning if lambda is not part of the fitted sequence and coefficients need to be interpolated.

correction  
  defunct.

...  
  currently not used.

Value

a numeric vector of residuals for the given penalization level.

See Also

Other functions for extracting components: coef.pense_cvfit(), coef.pense_fit(), predict.pense_cvfit(), predict.pense_fit(), residuals.pense_cvfit()

Examples

# Compute the LS-EN regularization path for Freeny's revenue data  
# (see ?freeny)
data(freeny)
x <- as.matrix(freeny[, 2:5])
regpath <- elnet(x, freeny$y, alpha = 0.75)

# Predict the response using a specific penalization level
predict(regpath, newdata = freeny[1:5, 2:5],
  lambda = regpath$lambda[1][[10]])

# Extract the residuals at a certain penalization level
residuals(regpath, lambda = regpath$lambda[1][[5]])

# Select penalization level via cross-validation
set.seed(123)
cv_results <- elnet_cv(x, freeny$y, alpha = 0.5,
  cv_repl = 10, cv_k = 4)

# Predict the response using the "best" penalization level
predict(cv_results, newdata = freeny[1:5, 2:5])

# Extract the residuals at the "best" penalization level
residuals(cv_results)
# Extract the residuals at a more parsimonious penalization level
residuals(cv_results, lambda = "1.5-se")
rho_function

List Available Rho Functions

Description
List Available Rho Functions

Usage
rho_function(rho)

Arguments
rho the name of the $\rho$ function to check for existence.

Value
if rho is missing returns a vector of supported $\rho$ function names, otherwise the internal integer representation of the $\rho$ function.

See Also
Other miscellaneous functions: `consistency_const()`

starting_point
Create Starting Points for the PENSE Algorithm

Description
Create a starting point for starting the PENSE algorithm in `pense()`. Multiple starting points can be created by combining starting points via `c(starting_point_1, starting_point_2, ...)`. 

Usage
starting_point(beta, intercept, lambda, alpha)

as_starting_point(object, specific = FALSE, ...)

## S3 method for class 'empy_starting_points'
as_starting_point(object, specific = FALSE, ...)

## S3 method for class 'pense_fit'
as_starting_point(object, specific = FALSE, alpha, lambda, ...)

## S3 method for class 'pense_cvfit'
as_starting_point(
  object,
  specific = FALSE,
  alpha,
  lambda = c("min", "se"),
)
starting_point

se_mult = 1,
...
)

Arguments

beta  beta coefficients at the starting point. Can be a numeric vector, a sparse vector of class dsparseVector, or a sparse matrix of class dgCMatrix with a single column.
intercept intercept coefficient at the starting point.
lambda optionally either a string specifying which penalty level to use ("min" or "se") or a numeric vector of the penalty levels to extract from object. Penalization levels not present in object are ignored with a warning. If NULL, all estimates in object are extracted. If a numeric vector, alpha must be given and a single number.
alpha optional value for the alpha hyper-parameter. If given, only estimates with matching alpha values are extracted. Values not present in object are ignored with a warning.
object an object with estimates to use as starting points.
specific whether the estimates should be used as starting points only at the penalization level they are computed for. Defaults to using the estimates as starting points for all penalization levels.
... further arguments passed to or from other methods.
se_mult If lambda = "se", the multiple of standard errors to tolerate.

Details

A starting points can either be shared, i.e., used for every penalization level PENSE estimates are computed for, or specific to one penalization level. To create a specific starting point, provide the penalization parameters lambda and alpha. If lambda or alpha are missing, a shared starting point is created. Shared and specific starting points can all be combined into a single list of starting points, with pense() handling them correctly. Note that specific starting points will lead to the lambda value being added to the grid of penalization levels. See pense() for details.

Starting points computed via enpy_initial_estimates() are by default shared starting points but can be transformed to specific starting points via as_starting_point(...,specific = TRUE).

When creating starting points from cross-validated fits, it is possible to extract only the estimate with best CV performance (lambda = "min"), or the estimate with CV performance statistically indistinguishable from the best performance (lambda = "se"). This is determined to be the estimate with prediction performance within se_mult * cv_se from the best model.

Value

an object of type starting_points to be used as starting point for pense().

See Also

Other functions for initial estimates: enpy_initial_estimates(), prinsens()
**summary.pense_cvfit**  

*Summarize Cross-Validated PENSE Fit*

**Description**

If \( \lambda = "se" \) and object contains fitted estimates for every penalization level in the sequence, extract the coefficients of the most parsimonious model with prediction performance statistically indistinguishable from the best model. This is determined to be the model with prediction performance within \( se_{mult} \times cv_{se} \) from the best model.

**Usage**

```r
## S3 method for class 'pense_cvfit'
summary(object, alpha, lambda = "min", se_mult = 1, ...)
## S3 method for class 'pense_cvfit'
print(x, alpha, lambda = "min", se_mult = 1, ...)
```

**Arguments**

- `object, x` an (adaptive) PENSE fit with cross-validation information.
- `alpha` Either a single number or missing. If given, only fits with the given \( \alpha \) value are considered. If \( \lambda \) is a numeric value and object was fit with multiple \( \alpha \) values, the parameter \( \alpha \) must not be missing.
- `lambda` either a string specifying which penalty level to use ("min", "se", "{x}-se") or a single numeric value of the penalty parameter. See details.
- `se_mult` If \( \lambda = "se" \), the multiple of standard errors to tolerate.
- `...` ignored.

**See Also**

- `prediction_performance()` for information about the estimated prediction performance.
- `coef.pense_cvfit()` for extracting only the estimated coefficients.

Other functions for plotting and printing: `plot.pense_cvfit()`, `plot.pense_fit()`, `prediction_performance()`

---

**tau_size**  

*Compute the Tau-Scale of Centered Values*

**Description**

Compute the \( \tau \)-scale without centering the values.

**Usage**

`tau_size(x)`

**Arguments**

- `x` numeric values. Missing values are verbosely ignored.
Value

the $\tau$ estimate of scale of centered values.

See Also

Other functions to compute robust estimates of location and scale: `mlocscale()`, `mloc()`, `mscale()`
Index

* EN algorithms
  en_admm_options, 16
  en_dal_options, 17
  en_lars_options, 18

* deprecated functions
  deprecated_en_options, 6
  enpy, 13
  mstep_options, 23
  pense_options, 35
  pensem, 27

* functions for computing non-robust estimates
  elnet, 8
  elnet_cv, 10

* functions for extracting components
  coef.pense_cvfit, 3
  coef.pense_fit, 4
  predict.pense_cvfit, 36
  predict.pense_fit, 37
  residuals.pense_cvfit, 49
  residuals.pense_fit, 51

* functions for initial estimates
  enpy_initial_estimates, 14
  prinsens, 43
  starting_point, 53

* functions for plotting and printing
  plot.pense_cvfit, 36
  plot.pense_fit, 37
  prediction_performance, 42
  summary.pense_cvfit, 55

* functions to compute robust estimates of location and scale
  mloc, 19
  mlocscale, 20
  mscale, 21
  tau_size, 55

* functions to compute robust estimates with CV
  pense_cv, 31
  pensem_cv, 28
  regmest_cv, 46

* functions to compute robust estimates
  pense, 24
  regmest, 44

* miscellaneous functions
  consistency_const, 6
  rho_function, 53

  adaelnet (elnet), 8
  adaeen (elnet), 8
  adamest_cv (regmest_cv), 46
  adamest_cv(), 48
  adapense (pense), 24
  adapense_cv (pense_cv), 31
  as_starting_point (starting_point), 53
  as_starting_point(), 15

  coef.pense_cvfit, 3, 5, 39, 41, 51, 52
  coef.pense_cvfit(), 3, 5, 12, 34, 49, 55
  coef.pense_fit, 3, 39, 41, 51, 52
  coef.pense_fit(), 9, 27, 46
  consistency_const, 6, 53

  deprecated_en_options, 6, 13, 19, 23, 28, 36
  dgCMatrix, 54
dsparseVector, 3, 5, 54

  elnet, 8, 11, 12
  elnet(), 3, 10, 12, 40, 51
  elnet_cv, 9, 10
  en_admm_options, 16, 18
  en_admm_options(), 17
  en_algorithm_options, 9, 11, 15, 17, 21, 43
  en_dal_options, 16, 17, 18
  en_dal_options(), 7, 17
  en_lars_options, 16, 18, 18
  en_lars_options(), 7, 17
  en_options_aug_lars
    (deprecated_en_options), 6
  en_options_dal (deprecated_en_options), 6
  enpy, 7, 13, 19, 23, 28, 36
  enpy_initial_estimates, 14, 44, 54
  enpy_initial_estimates(), 13, 25, 26, 31,
    33, 34, 54
  enpy_options, 15
enpy_options(), 14, 18, 26, 31, 34
initest_options, 7, 13, 18, 23, 28, 36
mad(), 19
mloc, 19, 20, 22, 56
mlocscale, 20, 20, 22, 56
mm_algorithm_options, 21
mm_algorithm_options(), 23, 26, 31, 34, 35, 45, 48
mscale, 20, 21, 56
mscale_algorithm_options, 22
mscale_algorithm_options(), 14, 19, 20, 22, 26, 31, 34
mstep_options, 7, 13, 19, 23, 28, 36
parallel, 11, 30, 32, 47
pense, 24, 32, 46
pense(), 4, 9, 15, 31, 34, 35, 40, 51, 53, 54
pense_cv, 31, 31, 49
pense_cv(), 12, 26–29, 31, 34
pense_options, 7, 13, 19, 23, 28, 35
pensem, 7, 13, 19, 23, 27, 36
pensem_cv, 28, 34, 49
pensem_cv(), 23, 27
plot.pense_cvfit, 36, 38, 42, 55
plot.pense_cvfit(), 12, 34, 49
plot.pense_fit, 37, 37, 42, 55
plot.pense_fit(), 9, 27, 46
predict.pense_cvfit, 4, 5, 38, 41, 51, 52
predict.pense_fit, 4, 5, 39, 40, 51, 52
prediction_performance, 37, 38, 42, 55
prediction_performance(), 55
prinsens, 15, 43, 54
print.pense_cvfit
  (summary.pense_cvfit), 55
print.pense_pred_perf
  (prediction_performance), 42
regmest, 27, 44, 47
regmest(), 40, 46, 48, 49, 51
regmest_cv, 31, 34, 46
regmest_cv(), 28, 31, 46
residuals.pense_cvfit, 4, 5, 39, 41, 49, 52
residuals.pense_fit, 4, 5, 39, 41, 51, 51
rho_function, 6, 53
rho_function(), 19, 20
starting_point, 15, 44, 53
starting_point(), 25, 33, 45, 47
summary.pense_cvfit, 37, 38, 42, 55
summary.pense_cvfit(), 42
tau_size, 20, 22, 55
tau_size(), 12, 31, 34, 48