# Package: capnet (via r-universe)

November 19, 2026

**Type** Package

**Title** Contribution-Capped Elastic Net Regularization

**Version** 0.4.1

**Date** 2026-10-28

**Copyright** 2026 Harry He

**Description** Implements contribution-capped regularization to directly bound per-feature contributions in predictive models. Includes quasi-Newton optimization for unbounded fits and projected proximal gradient for finite coefficient bounds, as well as cross-validation, walk-forward fitting and prediction, and plotting utilities. References: Andrew and Gao (2007) <doi:10.1145/1273496.1273501>; Friedman et al. (2010) <doi:10.18637/jss.v033.i01>; James et al. (2020) <doi:10.1080/01621459.2019.1609970>; Nocedal (1980) <doi:10.1090/S0025-5718-1980-0572855-7>.

**URL** https://github.com/harryziyuhe/capnet

**BugReports** https://github.com/harryziyuhe/capnet/issues

**Depends** R (>= 4.1)

**Imports** rlang, lbfgs, ggplot2, xts, zoo, Matrix, stats, parallel

**Suggests** knitr, rmarkdown, xtable, glmnet, testthat (>= 3.0.0)

**License** GPL (>= 2)

**VignetteBuilder** knitr

**Encoding** UTF-8

**Config/testthat/edition** 3

**Config/roxygen2/version** 8.1.0

**NeedsCompilation** no

**Author** Harry He [aut, cre, cph]

**Maintainer** Harry He <zih028@ucsd.edu>

**Repository** https://cran.r-universe.dev

**Date/Publication** 2026-11-18 21:00:02 UTC

**RemoteUrl** https://github.com/cran/capnet

**RemoteRef** HEAD

**RemoteSha** ecd8501502ce80f2c1c7eab6a849c604afe853ea

# Contents

capnet . . . . . 2
capnet_violations . . . . . 5
coef.capnet . . . . . 7
coef.walk_capnet . . . . . 8
coef_path . . . . . 9
cv_capnet . . . . . 10
normalize_family . . . . . 13
plot.capnet_path . . . . . 14
plot.cv_capnet . . . . . 15
plot_redistribution . . . . . 16
predict.capnet . . . . . 17
predict.walk_capnet . . . . . 18
print.capnet_violations . . . . . 19
simulate_capnet_data . . . . . 20
validate_family_supported . . . . . 23
walk_capnet . . . . . 24

**Index** **27**

---

`capnet` *Fit a linear model with elastic net and contribution cap regularization*

---

## Description

Fits a penalized linear model by maximum likelihood with an elastic net penalty and an additional penalty that caps per-feature contributions. Unbounded fits use L-BFGS when the L1 penalty is inactive and Orthant-Wise Limited-Memory Quasi-Newton (OWL-QN) when it is active. Fits with finite coefficient bounds use projected proximal gradient, with soft-thresholding for L1 regularization.

## Usage

```
capnet(
  X,
  y,
  L,
  family = "gaussian",
  intercept = TRUE,
  standardize = TRUE,
```

```
  z = NULL,
  multiplier = 1,
  lambda = 0,
  alpha = 0,
  gamma = 0,
  lower.limits = NULL,
  upper.limits = NULL,
  tol = 1e-08,
  maxit = 10000L,
  par = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| `X` | Numeric predictor matrix of shape $n \times p$. Columns are features and rows are observations. |
| `y` | Numeric response vector of length $n$. |
| `L` | Nonnegative numeric scalar or length-$p$ vector giving the contribution ceiling(s). $L_j$ caps the absolute value of feature $j$'s contribution $\lvert z_{ij}\beta_j \rvert$ for each evaluation row $i$. If scalar, the same ceiling is applied to all features. When `gamma = 0`, `L` has no effect on the fit. |
| `family` | Optional character scalar (e.g. `"binomial"`), function (e.g. `stats::binomial`), or family object (e.g. `stats::binomial()`). Supported families: `"gaussian"`, `"binomial"`, `"poisson"`, `"Gamma"` (log link only). |
| `intercept` | Logical; should an intercept be fitted? Default `TRUE`. |
| `standardize` | Logical; if `TRUE`, columns of `X` are standardized (zero mean, unit SD) before fitting and coefficients are returned on the original scale. The cap penalty is always evaluated on the original scale regardless of this setting. Default `TRUE`. |
| `z` | Optional numeric matrix with $p$ columns used to evaluate the contribution cap penalty. The cap is enforced on $\lvert z_{ij}\beta_j \rvert$, not on training rows. If `NULL`, defaults to `X` (caps are enforced on the training data). Pass a held-out evaluation set to enforce caps on out-of-sample rows. Supply `z` on the same scale as `X`. |
| `multiplier` | Optional numeric scalar or length-$m$ vector (where $m = \mathrm{nrow}(z)$) used to scale feature contributions during the capping step. Defaults to 1. Useful when each evaluation row requires a different scaling of its contribution budget. |
| `lambda` | Nonnegative numeric scalar; overall strength of the elastic-net penalty. When `lambda = 0`, no elastic-net penalty is applied. |
| `alpha` | Numeric scalar in $[0, 1]$; elastic net mixing parameter. `alpha = 1` is LASSO (pure L1), `alpha = 0` is Ridge (pure L2). For fits without finite coefficient bounds, OWL-QN is used when `alpha > 0` and `lambda > 0`; otherwise L-BFGS is used. Fits with finite bounds use projected proximal gradient. |
| `gamma` | Nonnegative numeric scalar; strength of the contribution-cap penalty. When `gamma = 0`, no cap penalty is applied and the model reduces to standard elastic net. |

| | |
|---|---|
| `lower.limits` | Optional numeric scalar or length-$p$ vector of lower bounds on coefficients. Initial parameter values must satisfy the bounds. The default zero initialization is valid when zero lies within every interval; otherwise supply a valid `par`. |
| `upper.limits` | Optional numeric scalar or length-$p$ vector of upper bounds on coefficients. Initial parameter values must satisfy the bounds. |
| `tol` | Nonnegative numeric convergence tolerance for fits with finite coefficient bounds. It is the relative step-size threshold for the projected proximal-gradient solver and has no effect on unbounded fits. Default `1e-8`. |
| `maxit` | Integer; maximum number of optimizer iterations. Default `10000`. |
| `par` | Optional numeric vector of length $p + 1$ (intercept first, then slopes) with initial parameter values. If `NULL`, uses zero initialization. |
| `...` | Additional arguments used in fitting. Currently unused. |

## Details

Fits without finite coefficient bounds use OWL-QN to handle the L1 component when `alpha > 0` and `lambda > 0`, and L-BFGS otherwise. If any coefficient bound is finite, a projected proximal-gradient solver is used instead. Its proximal step applies soft-thresholding for the L1 term and then projects coefficients onto their allowed intervals. This permits coefficients to reach zero and change sign when the bounds allow it. `tol` controls the relative step-size convergence criterion for this bounded solver. If `standardize = TRUE`, the model is fit on standardized `X` and coefficients are mapped back to the original scale on return; the cap penalty is always computed on the original scale.

**Soft constraints**: the contribution cap is a *penalty*, not a hard constraint. The returned `feature_contributions` may still exceed `L` when `gamma` is small or the optimizer does not fully converge. Use `capnet_violations()` to check how much each cap is exceeded.

**Standardization with multiplier**: standardization is not recommended when a non-unit `multiplier` is supplied, as the scaling step may distort the intended contribution budget.

## Value

An object of class `"capnet"` with components:

| | |
|---|---|
| `a0` | Best intercept (numeric scalar). |
| `beta` | Numeric vector (length $p$); fitted coefficients. |
| `value` | Numeric; minimized objective value. |
| `feature_contributions` | Numeric matrix of shape $\mathrm{nrow}(z) \times p$ giving per-feature contributions $z_{ij}\hat{\beta}_j$ evaluated on `z`. |
| `z` | The evaluation matrix used for cap enforcement. |
| `convergence` | Integer convergence code from the optimizer: `0` = successful convergence; `-1` = the bounded optimizer reached `maxit`; `-1001` = rounding-error stop (benign near-convergence, common with highly collinear features); `-998` = line search failure (try reducing `gamma` or `lambda`); other negative values indicate optimizer errors. The function does **not** raise an error on convergence failure; always inspect this field after fitting. |

| | |
|---|---|
| message | Character string from the optimizer (may be NULL or uninformative for some failure codes). |
| alpha | alpha value passed in input. |
| lambda | lambda value passed in input. |
| gamma | gamma value passed in input. |
| L | L value passed in input. |
| multiplier | multiplier value passed in input. |
| family | model family passed in input. |
| call | The matched call. |

## See Also

cv_capnet(), walk_capnet(), predict.capnet(), coef.capnet(), capnet_violations()

## Examples

```
set.seed(1)
n <- 40
p <- 8
X <- matrix(rnorm(n * p), n, p)
beta <- c(2, 1.5, rep(0, p - 2))
y <- as.numeric(X %*% beta + rnorm(n))
L <- rep(1.5, p)

# Standard elastic net (gamma = 0, no cap)
fit0 <- capnet(X, y, lambda = 0.1, alpha = 0.5, gamma = 0, L = L)

# With contribution cap penalty
fit1 <- capnet(X, y, lambda = 0.1, alpha = 0.5, gamma = 1, L = L)
fit1$convergence # check optimizer status; 0 = success

# Enforce caps on a held-out evaluation set, not training rows
X_new <- matrix(rnorm(10 * p), 10, p)
fit2 <- capnet(X, y, lambda = 0.1, alpha = 0.5, gamma = 1, L = L, z = X_new)

# Box constraints (non-negative coefficients)
fit3 <- capnet(X, y,
  lambda = 0.1, alpha = 0.5, gamma = 1, L = L,
  lower.limits = 0
)
```

---

capnet_violations *Check contribution-cap violations for a fitted* capnet *model*

---

## Description

Identifies where feature contributions exceed the specified contribution cap. Returns the indices of offending rows and columns along with the excess contribution matrix.

## Usage

```
capnet_violations(object, z = NULL, multiplier = NULL)
```

## Arguments

| | |
|---|---|
| `object` | A fitted `capnet` model returned by `capnet()`. |
| `z` | Optional numeric matrix of new data on which to evaluate contribution caps. If `NULL`, uses the contribution matrix stored in `object$feature_contributions`. |
| `multiplier` | Optional numeric scalar or length-$n$ vector (where $n$ is `nrow(z)`) to override the multiplier used when the model was fit. If `NULL`, defaults to `object$multiplier`. |

## Details

Contributions are computed as $|X_{ij}\beta_j| \times multiplier_i$, compared against the cap $L_j$ stored in the fitted object. The returned `excess` matrix gives $\max(|X_{ij}\beta_j| \times multiplier_i - L_j, 0)$.

## Value

A list with the following components:

| | |
|---|---|
| `columns` | Integer vector of column indices (features) with any violations of the contribution cap. |
| `rows` | Integer vector of row indices (observations) where at least one feature exceeded its cap. |
| `excess` | Numeric matrix of the same dimension as `z` (or `object$feature_contributions`) containing the amount by which each absolute contribution exceeded its cap. Zero entries indicate no violation. |

If no violations are detected (the sum of `excess` is zero), the function prints a message and returns `NULL` invisibly.

## See Also

`capnet()`

## Examples

```
set.seed(1)
n <- 40
p <- 8
X <- matrix(rnorm(n * p), n, p)
beta <- c(2, 1.5, rep(0, p - 2))
y <- X %*% beta + rnorm(n)
fit <- capnet(X, y, lambda = 0.1, alpha = 0.5, gamma = 1, L = 1)
```

```
capnet_violations(fit)
```

---

`coef.capnet` *Extract coefficients from a fitted* `capnet` *model*

---

## Description

Returns the estimated intercept and coefficients from a fitted `capnet` model object.

## Usage

```
## S3 method for class 'capnet'
coef(object, ...)
```

## Arguments

| | |
|---|---|
| `object` | A fitted object of class `"capnet"`. |
| `...` | Further arguments passed to or from other methods. |

## Value

A $(1+p)\times 1$ numeric matrix. The first row is the intercept (named `"(Intercept)"`); the remaining $p$ rows are the fitted slopes, named from `colnames(X)` when available.

## See Also

`capnet()`, `predict.capnet()`

## Examples

```
set.seed(1)
n <- 40
p <- 5
X <- matrix(rnorm(n * p), n, p)
y <- as.numeric(X %*% c(1, -1, 0.5, 0, 0) + rnorm(n))
fit <- capnet(X, y, lambda = 0.1, alpha = 0.5, gamma = 1, L = 1)
coef(fit) # (1 + p) x 1 matrix
coef(fit)[-1, ] # slopes only
```

`coef.walk_capnet` *Extract coefficient paths from a walk-forward* `capnet` *fit*

## Description

Extract coefficient paths from a walk-forward `capnet` fit

## Usage

```
## S3 method for class 'walk_capnet'
coef(object, index = NULL, ...)
```

## Arguments

| | |
|---|---|
| `object` | A fitted object of class `"walk_capnet"` returned by `walk_capnet()`. |
| `index` | Optional integer vector of step indices (1 to `nrow(z)`) to subset. If `NULL`, all steps are returned. |
| `...` | Further arguments passed to or from other methods. |

## Value

A $\mathrm{nrow}(z) \times (1 + p)$ matrix (or `xts` if the input `z` was `xts`). The first column is the intercept path (named `"intercept"`); the remaining $p$ columns are coefficient paths named from `colnames(X)`. Rows within the same walk step share identical values. `NA` rows indicate steps that failed to converge. If `index` is supplied, only those rows are returned.

## See Also

`walk_capnet()`, `predict.walk_capnet()`

## Examples

```
set.seed(1)
n <- 50
p <- 5
n_new <- 8
X <- matrix(rnorm(n * p), n, p)
z <- matrix(rnorm(n_new * p), n_new, p)
y <- as.numeric(X %*% c(1, -1, 0.5, 0, 0) + rnorm(n))
fit <- walk_capnet(X, y, L = 1, z = z, lambda = 0.1, alpha = 0.5, gamma = 1)
coef(fit) # nrow(z) x (1 + p) path
coef(fit, index = 1:3) # first three steps only
```

coef_path *Calculate coefficients along a penalty path*

## Description

Computes fitted coefficients from `capnet()` along a path of `lambda` values (with `alpha` held fixed). This is useful for inspecting coefficient shrinkage and entry/exit as regularization increases.

## Usage

```
coef_path(
  X,
  y,
  L,
  lambda = exp(seq(1, -5, length.out = 50)),
  alpha = 0.5,
  gamma = 1,
  standardize = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `X` | Numeric predictor matrix of shape $n \times p$. |
| `y` | Numeric response vector of length $n$. |
| `L` | Nonnegative numeric scalar or length-$p$ vector giving the contribution ceiling(s). If scalar, the same ceiling is applied to all coefficients. |
| `lambda` | Numeric vector (default `exp(seq(1, -5, length.out = 50))`) of nonnegative elastic-net penalties to trace. |
| `alpha` | Numeric scalar in $[0, 1]$ (default `0.5`); the elastic-net mixing parameter. `alpha = 1` is Lasso, `alpha=0` is Ridge. |
| `gamma` | Nonnegative numeric scalar; strength of the contribution-cap penalty (default `1`). |
| `standardize` | Logical; if `TRUE`, columns of `X` and `y` are standardized for fitting at each point on the path; coefficients are still returned on the original scale. Default `FALSE`, so that coefficients along the path are computed directly on the original scale without a per-fit standardization step. |
| `...` | Additional arguments forwarded to `capnet()`, e.g., `z`, `par`, `multiplier`, `intercept`, `lower.limits`, `upper.limits`, `tol`, `maxit`, `check.finite`, `verbose`. |

## Details

For each point on the path, `capnet()` is fit with the remaining two hyperparameters held fixed, and the resulting coefficients are collected. Lambda and gamma paths use `log()` of the parameter value as the first column (suitable for log-scale x-axis); the alpha path uses the raw value. Pass the result to `plot()` to visualize the coefficient paths.

## Value

A `data.frame` of class `c("capnet_path", "data.frame")` with one row per point on the path. The first column is the path parameter (`"lambda (log)"`, `"gamma (log)"`, or `"alpha"`); the remaining $p$ columns are fitted coefficients, named from `colnames(X)` when available. Exactly one of `lambda`, `gamma`, `alpha` may be a vector; the other two must be scalars.

## See Also

`capnet()`, `plot.capnet_path()`

## Examples

```
set.seed(1)
n <- 50
p <- 8
X <- matrix(rnorm(n * p), n, p)
colnames(X) <- paste0("x", seq_len(p))
beta <- c(1.5, 0.8, 0.2, rep(0, p - 3))
y <- as.numeric(X %*% beta + rnorm(n))
path <- coef_path(X, y,
  L = 0.5, alpha = 0.5,
  lambda = exp(seq(1, -5, length.out = 50)), gamma = 1
)
plot(path)
```

---

# cv_capnet *Perform cross-validation for* capnet

---

## Description

Runs a grid search over `alpha` and `lambda` with K-fold cross-validation for the `capnet()` model. Returns the searched grid, fold-wise errors, mean errors, and the best hyperparameters.

## Usage

```
cv_capnet(
  X,
  y,
  gamma,
  L,
  family = "gaussian",
  lambda = exp(seq(1, -5, length.out = 50)),
  alpha = seq(0, 1, length.out = 5),
  z = NULL,
  multiplier = 1,
  intercept = TRUE,
  standardize = TRUE,
  splits = NULL,
```

```
  K = 5,
  metric = c("mse", "rsq", "logloss", "brier", "deviance"),
  verbose = 0,
  parallel = FALSE,
  workers = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| X | Numeric predictor matrix of shape $n \times p$. Columns are features and rows are observations. |
| y | Numeric response vector of length $n$. |
| gamma | Nonnegative numeric scalar; strength of the contribution-cap penalty. Unlike alpha and lambda, gamma is held fixed during the CV grid search. To compare different gamma values, call cv_capnet() separately for each. |
| L | Nonnegative numeric scalar or length-$p$ vector giving the contribution ceiling(s). If scalar, the same ceiling is applied to all features. |
| family | Optional character scalar (e.g. "binomial"), function (e.g. stats::binomial), or family object (e.g. stats::binomial()). |
| lambda | Numeric vector (default exp(seq(1, -5, length.out = 50))) of nonnegative elastic-net penalty strengths to search. |
| alpha | Numeric vector (default seq(0, 1, length.out = 5)) with values in $[0, 1]$ to search (elastic-net mixing parameters; alpha = 1 is Lasso, alpha = 0 is Ridge). |
| z | Optional numeric matrix with $p$ columns used to evaluate and apply contribution caps. If NULL, defaults to X. |
| multiplier | Optional numeric scalar or length-$n$ vector used to scale feature contributions during the capping step; defaults to 1. |
| intercept | Logical; should an intercept be fitted? Default TRUE. |
| standardize | Logical; if TRUE, columns of X and y are standardized for fitting; coefficients are returned on the original scale. Default TRUE. |
| splits | Optional integer vector of length $n$ giving the fold assignment for each row. Each unique integer value defines one fold; rows with splits == k are held out when fold k is the test fold. Folds need not be contiguous or equal-sized. If NULL, folds are assigned randomly using K. For time-ordered data, supply splits explicitly to avoid leakage — for example: splits <- rep(1:5, each = n / 5) assigns the first fifth of rows to fold 1, the second fifth to fold 2, and so on. |
| K | Integer $\geq 2$; number of folds used when splits is NULL. Default 5. If splits is provided, K is set to length(unique(splits)). |
| metric | Character string; CV scoring metric. When omitted, the default is chosen by family: "mse" for Gaussian (lower is better), "logloss" for binomial (lower is better), "deviance" for Poisson and Gamma (lower is better). Available options: "mse", "rsq" (higher is better), "logloss", "brier", "deviance". |
| verbose | Integer; 0 for silent, 1 to print progress. |

| | |
|---|---|
| parallel | Logical; if TRUE, the (fold, alpha) grid is dispatched across a parallel::makeCluster() PSOCK cluster via parallel::parLapply(). The lambda path within each (fold, alpha) pair is always fit sequentially, warm-starting from the previous lambda. Default FALSE. |
| workers | Optional integer; number of parallel workers to use when parallel = TRUE. Defaults to min(n_tasks, detectCores() - 1,90): capped at the number of (fold x alpha) tasks (idle workers add no value) and at 90 to avoid exhausting R's 128-connection limit. |
| ... | Additional arguments forwarded to capnet(), e.g., lower.limits, upper.limits, tol, maxit. |

## Details

If splits is NULL, folds are generated as sample(rep(1:K, length.out = n)). Set a seed beforehand to reproduce the random fold allocation. If splits is supplied as an integer vector of fold IDs, K is inferred as length(unique(splits)).

For each (alpha, lambda) pair, capnet() is fit on the training portion of each fold and evaluated on the held-out rows using metric.

**Time-series data**: the default random fold creation assumes i.i.d. rows. For time-ordered data supply splits explicitly so that each test fold lies strictly after its training fold:

```
# Walk-forward folds: fold k tests the k-th block of rows
splits <- rep(seq_len(K), each = ceiling(n / K))[seq_len(n)]
```

## Value

An object of class "cv_capnet" with components:

| | |
|---|---|
| alpha | Numeric vector of alpha values searched. |
| lambda | Numeric vector of lambda values searched. |
| gamma | gamma value passed in input. |
| L | L value passed in input. |
| metric | Metric used to evaluate performance. |
| splits | Cross-validation splits. |
| cv_errors | Numeric array of shape $A \times L \times K$; cv_errors[a, l, k] is the fold-k error for alpha[a], lambda[l]. Units match metric. |
| mean_errors | Numeric matrix of shape $A \times L$ with mean CV error across folds for each parameter pair. |
| best_alpha | Numeric; the selected alpha. |
| best_lambda | Numeric; the selected lambda. |
| best_error | Numeric; mean CV score at (best_alpha, best_lambda). For "mse", "logloss", "deviance", and "brier" this is the minimum; for "rsq" it is the maximum. |

## See Also

capnet(), walk_capnet(), plot.cv_capnet()

## Examples

```
set.seed(1)
n <- 80
p <- 10
X <- matrix(rnorm(n * p), n, p)
beta <- c(1.2, 0.7, 0.5, rep(0, p - 3))
y <- as.numeric(X %*% beta + rnorm(n))

# Basic cross-validation
cv <- cv_capnet(X, y, gamma = 1, L = 1.5)
cv$best_alpha
cv$best_lambda
cv$best_error

# Visualize the CV error surface
plot(cv)
plot(cv, alpha = cv$best_alpha)

# Time-ordered data: use sequential folds to avoid leakage
K <- 5
splits <- rep(seq_len(K), each = ceiling(n / K))[seq_len(n)]
cv_ts <- cv_capnet(X, y, gamma = 1, L = 1.5, splits = splits)
```

---

# normalize_family *Normalize user input into a stats family object*

---

## Description

Normalize user input into a stats family object

## Usage

```
normalize_family(family = NULL)
```

## Arguments

| | |
|---|---|
| family | optional character scalar (e.g. "binomial"), a function (e.g. `stats::binomial`), or a family object (e.g. `stats::binomial()`). |

## Value

A stats family object

`plot.capnet_path` *Plot coefficient paths along a single hyperparameter*

## Description

Plot coefficient paths along a single hyperparameter

## Usage

```
## S3 method for class 'capnet_path'
plot(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | An object of class `"capnet_path"` returned by `coef_path()`, or any `data.frame` with the same structure: first column is the path parameter, remaining columns are coefficients. |
| `...` | Further arguments passed to or from other methods. |

## Details

Produces a line plot with the path parameter on the x-axis (log scale for `lambda` and `gamma` paths, linear for `alpha`) and coefficient values on the y-axis. Each feature is drawn as a separate colored line, making it easy to see which coefficients enter, shrink, or change sign along the path.

## Value

The `ggplot` object, returned invisibly. As a side effect the plot is printed to the current graphics device.

## Examples

```
set.seed(1)
n <- 50
p <- 6
X <- matrix(rnorm(n * p), n, p)
colnames(X) <- paste0("x", seq_len(p))
y <- as.numeric(X %*% c(1.5, -1, 0.5, 0, 0, 0) + rnorm(n))
path <- coef_path(X, y,
  L = 0.5, alpha = 0.5,
  lambda = exp(seq(1, -5, length.out = 30)), gamma = 1
)
plot(path)
```

`plot.cv_capnet` *Plot cross-validation results for* `cv_capnet`

## Description

Plot cross-validation results for `cv_capnet`

## Usage

```
## S3 method for class 'cv_capnet'
plot(x, alpha = NULL, lambda = NULL, ...)
```

## Arguments

| | |
|---|---|
| `x` | An object of class `"cv_capnet"` returned by `cv_capnet()`. |
| `alpha` | Optional numeric; if provided, show CV errors vs `lambda` at this alpha. |
| `lambda` | Optional numeric; if provided, show CV errors vs `alpha` at this lambda. |
| `...` | Further arguments passed to or from other methods. |

## Details

Three display modes depending on which arguments are supplied:

**Neither** `alpha` **nor** `lambda` Heatmap of ranked mean CV errors across the full $\alpha \times \lambda$ grid. Lighter fill = better rank. Useful for identifying the most promising region before inspecting slices.

`alpha` **specified** Line plot of mean CV error (± 1 SE across folds) vs $\log \lambda$ at the given $\alpha$. A dashed vertical line marks the best $\lambda$.

`lambda` **specified** Line plot of mean CV error (± 1 SE) vs $\alpha$ at the given $\lambda$.

Supply at most one of `alpha` and `lambda`; both must be values that appear in the searched grid.

## Value

The `ggplot` object, returned invisibly. As a side effect the plot is printed to the current graphics device.

## Examples

```
set.seed(1)
n <- 80
p <- 10
X <- matrix(rnorm(n * p), n, p)
y <- as.numeric(X %*% c(rep(1, 3), rep(0, p - 3)) + rnorm(n))
cv <- cv_capnet(X, y, gamma = 1, L = 1.5)
plot(cv) # full heatmap
plot(cv, alpha = cv$best_alpha) # error vs lambda slice
plot(cv, lambda = cv$best_lambda) # error vs alpha slice
```

`plot_redistribution` *Plot redistribution of coefficients and contributions*

## Description

Compares regression coefficients and per-feature contributions between an uncapped and a capped `capnet` fit. Produces either side-by-side ("raw") bar charts of levels or ("delta") bar charts of changes.

## Usage

```
plot_redistribution(
  uncapped,
  capped,
  X = NULL,
  multiplier = NULL,
  top_n = Inf,
  plot = c("raw", "delta")
)
```

## Arguments

| | |
|---|---|
| `uncapped` | A `"capnet"` object fit without an active contribution cap (typically `gamma = 0`), serving as the baseline. |
| `capped` | A `"capnet"` object fit to the same `X` and `y` with contribution caps applied (`gamma > 0` and finite `L`). Both models should use the same evaluation matrix `z` so that contributions are comparable. |
| `X` | Optional numeric matrix of features used to compute contributions. If `NULL`, the function uses `capped$z`. |
| `multiplier` | Optional numeric scalar or length-$n$ vector of multipliers to scale contributions. If `NULL`, uses `capped$multiplier` when available; otherwise defaults to 1. |
| `top_n` | Integer ($\geq 1$) or `Inf`. Number of features to display. Features are ordered by input column order; set `top_n = Inf` to include all. |
| `plot` | Character string, one of `"raw"` or `"delta"`. If `"raw"`, shows side-by-side bar charts of coefficients and contributions for each fit. If `"delta"`, shows bar charts of the capped minus uncapped changes. |

## Details

For each model, per-row contributions are computed as $contrib_{ij} = X_{ij}\hat{\beta}_j \times multiplier_i$; the displayed contribution for feature $j$ is the mean over rows: $\bar{contrib}_j = \text{mean}_i(X_{ij}\hat{\beta}_j \times multiplier_i)$. The `multiplier` from the `capped` model is used; if the two models were fit with different multipliers the comparison may be misleading.

## Value

A list of two ggplot objects, which are also printed::

p1 If plot = "raw": side-by-side bar chart of coefficients for uncapped vs capped models. If plot = "delta": bar chart of changes in coefficients (capped - uncapped).

p2 If plot = "raw": side-by-side bar chart of contributions for uncapped vs capped models. If plot = "delta": bar chart of changes in contributions (capped - uncapped).

The function prints both plots and invisibly returns the list list("beta" = p1, "contribution" = p2)

## See Also

capnet(), capnet_violations()

## Examples

```
set.seed(1)
n <- 40
p <- 6
X <- matrix(rnorm(n * p), n, p)
beta <- c(1.5, 0.8, 0.2, rep(0, p - 3))
y <- as.numeric(X %*% beta + rnorm(n))
fit_uncap <- capnet(X, y, lambda = 0.05, alpha = 0.5, gamma = 0, L = 1)
fit_cap <- capnet(X, y, lambda = 0.05, alpha = 0.5, gamma = 1, L = 1)
plot_redistribution(fit_uncap, fit_cap, plot = "delta")
```

---

# predict.capnet *Predict from a fitted* capnet *model*

---

## Description

Predict from a fitted capnet model

## Usage

```
## S3 method for class 'capnet'
predict(object, newdata = NULL, type = c("link", "response"), ...)
```

## Arguments

| | |
|---|---|
| object | A fitted object of class "capnet". |
| newdata | Optional numeric matrix with $p$ columns for prediction. If NULL, uses object$z (the evaluation matrix stored at fit time). |
| type | Character; "link" returns the linear predictor $\hat{\eta} = \hat{\beta}_0 + X\hat{\beta}$; "response" applies the inverse link function and returns the fitted mean $\hat{\mu}$. For Gaussian models the two are identical. Default "link". |
| ... | Further arguments passed to or from other methods. |

## Details

When `newdata = NULL`, predictions are made on `object$z`, which is the evaluation matrix used when the model was fit. This is convenient for inspecting fitted contributions without re-specifying the data. When `newdata` is supplied it must have exactly $p$ columns (matching `length(object$beta)`) but may have any number of rows.

## Value

Numeric vector of length `nrow(newdata)` (or `nrow(z)` if `newdata` is `NULL`).

## See Also

`capnet()`, `coef.capnet()`

## Examples

```
set.seed(1)
n <- 40
p <- 5
X <- matrix(rnorm(n * p), n, p)
y <- as.numeric(X %*% c(1, -1, 0.5, 0, 0) + rnorm(n))
fit <- capnet(X, y, lambda = 0.1, alpha = 0.5, gamma = 1, L = 1)
predict(fit, type = "response") # fitted values on training z
predict(fit, newdata = X, type = "link") # linear predictor on new data
```

---

# `predict.walk_capnet` *Predict from a walk-forward* `capnet` *fit*

---

## Description

Predict from a walk-forward `capnet` fit

## Usage

```
## S3 method for class 'walk_capnet'
predict(object, ...)
```

## Arguments

| | |
|---|---|
| `object` | A fitted object of class `"walk_capnet"`. |
| `...` | Currently unused. |

## Value

A $\mathrm{nrow}(z) \times 1$ matrix (or `xts` if the input `z` was `xts`) of predictions on the response scale, as computed during the walk-forward evaluation. `NA` entries indicate steps where the optimizer failed to converge. To extract the full coefficient path, use `coef(object)`.

## See Also

walk_capnet(), coef.walk_capnet()

## Examples

```
set.seed(1)
n <- 50
p <- 5
n_new <- 8
X <- matrix(rnorm(n * p), n, p)
z <- matrix(rnorm(n_new * p), n_new, p)
y <- as.numeric(X %*% c(1, -1, 0.5, 0, 0) + rnorm(n))
fit <- walk_capnet(X, y, L = 1, z = z, lambda = 0.1, alpha = 0.5, gamma = 1)
predict(fit)
```

---

# print.capnet_violations

*Print the violation matrix from a* capnet_violations *object*

---

## Description

Print the violation matrix from a capnet_violations object

## Usage

```
## S3 method for class 'capnet_violations'
print(x, ...)
```

## Arguments

| | |
|---|---|
| x | An object of class "capnet_violations" returned by capnet_violations(). |
| ... | Optional arguments forwarded to Matrix::printSpMatrix(), e.g. col.names, digits, align. |

## Details

Displays the excess-contribution matrix as a sparse matrix: only non-zero entries (i.e., actual cap violations) are printed. Each non-zero entry gives the amount by which $|z_{ij}\hat{\beta}_j|$ exceeded $L_j$ for that row-feature combination.

## Value

x, invisibly.

## See Also

capnet_violations(), capnet()

## Examples

```
set.seed(1)
n <- 40
p <- 5
X <- matrix(rnorm(n * p), n, p)
y <- as.numeric(X %*% c(2, -1.5, 0, 0, 0) + rnorm(n))
fit <- capnet(X, y, lambda = 0.05, alpha = 0.5, gamma = 0.1, L = 0.5)
v <- capnet_violations(fit)
if (!is.null(v)) print(v) # sparse display of excess contributions
if (!is.null(v)) v$excess[1:5, ] # raw access
```

---

# simulate_capnet_data *Generate synthetic data for testing* capnet

---

## Description

Creates synthetic regression data for two scenarios with a unified return format. In both cases the function returns `X`, `y`, the ground-truth coefficients `beta_true`, and suggested caps `L`. When `m > 0`, an evaluation matrix `Z` of `m` rows is also returned, generated from the same data-generating process and ready to pass as the `z` argument of `capnet()` or `walk_capnet()`.

## Usage

```
simulate_capnet_data(
  n = 100,
  p = 20,
  m = 0,
  mode = c("basic", "proxy"),
  sparsity = 0.7,
  beta_range = c(-2, 2),
  sigma = 1,
  seed = NULL,
  xstar_sd = 1,
  a_sd = 0.3,
  unstable_idx = NULL,
  unstable_frac = 0.2,
  sigma_low = 1,
  sigma_high = 8,
  high_prob = 0.1,
  family = c("gaussian", "binomial", "poisson", "gamma"),
  poisson_mean = 10,
  gamma_mean = 10,
  gamma_shape = 2,
  binomial_prob = 0.5
)
```

## Arguments

| | |
|---|---|
| n | Integer; number of training observations. Default 100. |
| p | Integer; number of predictors. Default 20. |
| m | Integer; number of evaluation rows to generate. When m > 0 the function returns a Z matrix of m rows drawn from the same DGP as X. Default 0 (no evaluation set returned). |
| mode | Character; one of "basic" or "proxy". Default "basic". |
| sparsity | Numeric in $[0, 1]$; fraction of nonzero coefficients in beta_true. Default 0.7 |
| beta_range | Numeric length-2; uniform range for nonzero beta_true entries. Default c(-2, 2). |
| sigma | Positive numeric; noise sd for y. Only used when family = "gaussian". Default 1. |
| seed | Optional integer for reproducibility. Default NULL. |
| xstar_sd | Positive numeric; sd of latent true signal $X^*$. Default 1. |
| a_sd | Nonnegative numeric; sd of log-scales a_j = exp(N(0,a_sd^2)). Default 0.3. |
| unstable_idx | Optional integer vector of columns of X with unstable variance. If NULL, round(unstable_frac * p) columns are sampled at random. |
| unstable_frac | Numeric in $[0, 1]$; fraction of features to label unstable when unstable_idx is NULL. Default 0.25. |
| sigma_low | Positive numeric; base noise sd for X. Default 1. |
| sigma_high | Positive numeric; intermittent high noise sd for unstable features. Default 8. |
| high_prob | Numeric in $[0, 1]$; per-row probability an unstable feature uses sigma_high. Default 0.10. |
| family | Character; response family, one of "gaussian", "binomial", "poisson", or "gamma". Default "gaussian". Only supported in mode = "basic"; mode = "proxy" always generates a Gaussian response, since that mode's measurement-error structure is orthogonal to response family. For every non-Gaussian family, the slope part of the linear predictor, X %*% beta_true, is rescaled to unit standard deviation and beta_true is updated to match, then a data-generating intercept (not returned, analogous to the a0 that capnet() fits internally) shifts the linear predictor so the response's mean matches poisson_mean/gamma_mean/binomial_prob as applicable. This keeps the response in a numerically sane range regardless of p or beta_range. |
| poisson_mean | Positive numeric; target average response when family = "poisson". Default 10. |
| gamma_mean | Positive numeric; target average response when family = "gamma". Default 10. |
| gamma_shape | Positive numeric; fixed Gamma shape parameter used when family = "gamma" (rate is set to gamma_shape / mu so the mean is mu). Default 2. |
| binomial_prob | Numeric in $(0, 1)$; target average success probability when family = "binomial". Default 0.5. |

## Details

In **basic** mode, nonzeros in `beta_true` are placed uniformly at random and drawn from `runif(beta_range[1], beta_range[2])`. When `family = "gaussian"`, `y = X %*% beta_true + N(0, sigma^2)`. For the other families (see the rescaling description under the `family` argument above), `y ~ Poisson(exp(eta))` for `"poisson"`, `y ~ Gamma(shape = gamma_shape, rate = gamma_shape / exp(eta))` for `"gamma"`, and `y ~ Bernoulli(plogis(eta))` for `"binomial"`. In all cases, the suggested caps are `L_j = pmax(|beta_true_j|, 0.1)` as a simple starting point. When `m > 0`, `Z` is an $m \times p$ matrix of i.i.d. standard normal draws from the same pool as `X`.

In **proxy** mode, `y = X^* %*% beta_true + N(0, sigma^2)` and `X = diag(a) X^* + varepsilon`. For columns in `unstable_idx`, each row independently uses `sigma_high` with probability `high_prob`. When `m > 0`, `Z` is generated by applying the same noisy proxy process to `m` fresh latent rows, giving the observed evaluation matrix a practitioner would actually have available at evaluation time.

## Value

A list with the following components:

| | |
|---|---|
| `X` | $(n \times p)$ training design matrix. |
| `y` | Numeric response vector of length $n$. |
| `beta_true` | Numeric length-$p$ vector of ground-truth coefficients. |
| `L` | Length-$p$ vector of suggested contribution caps (nonnegative). |
| `Z` | $(m \times p)$ evaluation matrix, returned only when `m > 0`. Generated from the same DGP as `X` and ready to pass as `z` in `capnet()`. |

For `mode = "proxy"`, additional fields are included:

| | |
|---|---|
| `X_latent` | $(n \times p)$ latent true-signal matrix $X^*$ for the training rows. |
| `a` | Feature scales. |
| `unstable_idx` | Integer vector of unstable feature indices. |
| `stable_idx` | Integer vector of stable feature indices. |

## Proxy/unstable-feature controls (used when `mode = "proxy"`)

These arguments control the generation of unstable proxy features when simulating data under `mode = "proxy"`.

## Examples

```
set.seed(42)
d_basic <- simulate_capnet_data(n = 100, p = 10, m = 20, sparsity = 0.5)
names(d_basic) # includes Z

d_pois <- simulate_capnet_data(
  n = 100, p = 10, m = 20, sparsity = 0.5,
  family = "poisson", poisson_mean = 15
)
range(d_pois$y)
```

```
d_binom <- simulate_capnet_data(
  n = 100, p = 10, m = 20, sparsity = 0.5,
  family = "binomial", binomial_prob = 0.3
)
table(d_binom$y)

d_gamma <- simulate_capnet_data(
  n = 100, p = 10, m = 20, sparsity = 0.5,
  family = "gamma", gamma_mean = 5
)
range(d_gamma$y)

d_proxy <- simulate_capnet_data(
  mode = "proxy", n = 300, p = 8, m = 50,
  sparsity = 0.6, unstable_frac = 0.25,
  sigma_low = 1, sigma_high = 6, high_prob = 0.2
)
names(d_proxy) # includes Z and X_latent
```

---

`validate_family_supported` *Validate the family/link combination is supported*

---

## Description

Validate the family/link combination is supported

## Usage

```
validate_family_supported(fam)
```

## Arguments

| | |
|---|---|
| `fam` | stats family object |

## Value

invisibly `TRUE`, errors otherwise

walk_capnet *Perform walk-forward model fitting with contribution caps*

## Description

Fits a linear elastic net model with an additional contribution-cap penalty once on the full training data X, y, then walks the contribution-cap penalty forward across the evaluation matrix z, re-solving the objective at each step with the penalty applied only to the next walk row(s) of z (rather than to all of z at once, as a single call to capnet would do). The training data is not expanded or refit incrementally as z is walked; this mirrors a deployment setting where X, y are a data snapshot (e.g. a rolling training window) already prepared upstream, and repeating that preprocessing for every new evaluation row would be prohibitively expensive. To simulate a fully expanding training window instead, call capnet directly in a loop with an updated X, y at each step.

## Usage

```
walk_capnet(
  X,
  y,
  L,
  z,
  family = "gaussian",
  intercept = TRUE,
  standardize = TRUE,
  multiplier = 1,
  walk = 1,
  lambda = 0,
  alpha = 0,
  gamma = 0,
  max_gamma_tries = 6,
  parallel = FALSE,
  workers = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| X | Numeric predictor matrix of shape $n \times p$. Columns are features and rows are observations. |
| y | Numeric response vector of length $n$. |
| L | Nonnegative numeric scalar or length-$p$ vector giving the contribution ceiling(s). If scalar, the same ceiling is applied to all coefficients |
| z | Numeric matrix with $p$ columns (required). Each row of z is one evaluation point; the walk steps through these rows in order, enforcing the cap on walk rows at a time. Must be on the same scale as X. If z is an xts object, the output matrices will also be xts with the same index. |

| | |
|---|---|
| `family` | Optional character scalar (e.g. "binomial"), function (e.g. `stats::binomial`), or family object (e.g. `stats::binomial()`). |
| `intercept` | Logical; should an intercept be fitted? Default `TRUE`. |
| `standardize` | Logical; if `TRUE`, columns of `X` and `y` are standardized for fitting; coefficients are returned on the original scale. Default `TRUE`. |
| `multiplier` | Optional numeric scalar or length-$n$ vector used to scale feature contributions during the capping step. Defaults to 1. |
| `walk` | Positive integer; number of consecutive rows of `z` predicted as a batch at each step. At step $s$, rows $(s-1)\cdot \text{walk}+1$ through $s\cdot \text{walk}$ of `z` are used to enforce the cap and generate predictions together. If `nrow(z)` is not evenly divisible by `walk`, the last step covers the remaining rows. Default 1 (one row per step). |
| `lambda` | Nonnegative numeric scalar; overall strength of the elastic net penalty. |
| `alpha` | Numeric scalar in $[0, 1]$; elastic net mixing parameter. `alpha = 1` is Lasso, `alpha=0` is Ridge. |
| `gamma` | Nonnegative numeric scalar; strength of the contribution-cap penalty. |
| `max_gamma_tries` | Positive integer; maximum number of optimizer retries per step with a reduced `gamma`. If the optimizer fails to converge (convergence code $< 0$, excluding the benign `-1001` rounding error), `gamma` is divided by 10 and the step is retried up to `max_gamma_tries` times. If all retries fail, `NA` values are stored for that step. Default `6`. |
| `parallel` | Logical; if `TRUE`, walk-forward steps are dispatched across a `parallel::makeCluster()` PSOCK cluster via `parallel::parLapply()`. Each step fits independently, so this does not change results. Default `FALSE`. |
| `workers` | Optional integer; number of parallel workers to use when `parallel = TRUE`. Defaults to `min(n_steps, detectCores() - 1, 90)`: capped at the number of walk steps (idle workers add no value) and at 90 to avoid exhausting R's 128-connection limit. |
| `...` | Additional arguments used in fitting. See `capnet()` for more details. |

## Details

Given the evaluation matrix `z` of size $S \times p$, `X` and `y` are standardized (if requested) once, up front, and held fixed across all steps. At step $s = 1, \ldots, S$, `capnet()` is called internally on this fixed training fit, with the contribution-cap penalty restricted to rows $(s-1) \times \text{walk} : s \times \text{walk}$ of `z`; because the cap penalty (and its gradient) depend on the evaluation slice, coefficients are genuinely re-optimized at each step, not merely re-evaluated. Penalizing a smaller, local slice of `z` instead of the whole evaluation matrix at once (as plain `capnet()` would) avoids the over-penalization that comes from forcing a single fit to satisfy contribution caps uniformly across every evaluation row simultaneously.

## Value

An object of class `"walk_capnet"` with components (each of length / size $\text{nrow}(z)$, or an `xts` with the same index if `z` is `xts`):

`intercepts` $\mathrm{nrow}(z) \times 1$ matrix of fitted intercepts; row $i$ gives the intercept used to predict row $i$ of `z`. `NA` if the step failed to converge.

`betas` $\mathrm{nrow}(z) \times p$ matrix of fitted slopes; rows within the same step share identical values. `NA` rows indicate convergence failure.

`feature_contributions` $\mathrm{nrow}(z) \times p$ matrix; element $(i, j)$ is $z_{ij}\hat{\beta}_j$ at step $s(i)$.

`predictions` $\mathrm{nrow}(z) \times 1$ matrix of predictions on the response scale (`family\$linkinv(eta)`). `NA` where the corresponding step failed to converge.

`gammas` $\mathrm{nrow}(z) \times 1$ matrix; the effective `gamma` used at each step (may be smaller than the supplied `gamma` if retries reduced it). `NA` on failure.

## See Also

`capnet()`, `cv_capnet()`, `coef.walk_capnet()`, `predict.walk_capnet()`

## Examples

```
set.seed(1)
n <- 60
p <- 6
n_new <- 12
X <- matrix(rnorm(n * p), n, p)
z <- matrix(rnorm(n_new * p), n_new, p)
beta <- c(2.5, 1.5, 0.8, rep(0, p - 3))
y <- as.numeric(X %*% beta + rnorm(n))

# One prediction per step
out1 <- walk_capnet(X, y,
  L = 0.5, z = z, lambda = 0.1, alpha = 0.5,
  gamma = 1, walk = 1
)
predict(out1) # nrow(z) x 1 predictions
coef(out1) # nrow(z) x (1 + p) coefficient path

# Batch 3 rows per step (useful for weekly or monthly evaluation)
out3 <- walk_capnet(X, y,
  L = 0.5, z = z, lambda = 0.1, alpha = 0.5,
  gamma = 1, walk = 3
)
any(is.na(predict(out3))) # check for convergence failures
```

# Index

capnet, 2, *24*
capnet(), *6, 7*, *10*, *12*, *17–19*, *25, 26*
capnet_violations, 5
capnet_violations(), *5*, *17*, *19*
coef.capnet, 7
coef.capnet(), *5*, *18*
coef.walk_capnet, 8
coef.walk_capnet(), *19*, *26*
coef_path, 9
cv_capnet, 10
cv_capnet(), *5*, *26*

normalize_family, 13

plot.capnet_path, 14
plot.capnet_path(), *10*
plot.cv_capnet, 15
plot.cv_capnet(), *12*
plot_redistribution, 16
predict.capnet, 17
predict.capnet(), *5*, *7*
predict.walk_capnet, 18
predict.walk_capnet(), *8*, *26*
print.capnet_violations, 19

simulate_capnet_data, 20

validate_family_supported, 23

walk_capnet, 24
walk_capnet(), *5*, *8*, *12*, *19*