Package: CCNR (via r-universe)

November 5, 2026

Type Package

Title Classifier Chain Network for Multi-Label Classification

Version 0.2.0

Encoding UTF-8

Description Fits a Classifier Chain Network (CCN) for multi-label binary classification. The model jointly learns label dependencies through a chain structure and optimizes an Lq norm aggregated cross-entropy loss. Provides methods for prediction, model summary, and k-fold cross-validation with grid search over hyperparameters. The methodology is described in Touw and Van de Velden (2025) <doi:10.1016/j.eswa.2025.128048>.

License GPL-3

Depends R (>= 4.4.0)

URL https://github.com/djwtouw/CCNR

BugReports https://github.com/djwtouw/CCNR/issues

Imports Rcpp (>= 1.0.9), MASS, stats, utils

Suggests testthat (>= 3.0.0)

LazyData true

Config/testthat/edition 3

Config/build/clean-inst-doc FALSE

LinkingTo Rcpp, RcppEigen

Config/roxygen2/version 8.1.0

NeedsCompilation yes

Author Daniel J.W. Touw [aut, cre] (ORCID: <https://orcid.org/0000-0003-3074-5401>), Michel Van de Velden [ths] (ORCID: <https://orcid.org/0000-0002-9807-9057>, PhD advisor of Daniel J.W. Touw)

Maintainer Daniel J.W. Touw <touw@ese.eur.nl>

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

Date/Publication 2026-11-04 20:50:02 UTC

RemoteUrl https://github.com/cran/CCNR

RemoteRef HEAD

RemoteSha 9f665b7348a7f0b514f4739d6840bc3dc837c61c

Contents

CCNR-package . . . 2
amphibians . . . 3
ccn . . . 4
ccn_cv . . . 6
ccn_exact_match . . . 8
ccn_hamming_accuracy . . . 10
ccn_log_likelihood . . . 11
ccn_macro_f1 . . . 12
ccn_mean_auc . . . 13
ccn_micro_f1 . . . 14
coef.ccn . . . 15
coef.ccn_cv . . . 16
conditional_entropy_matrix . . . 18
entropy_label_order . . . 20
fitted.ccn . . . 22
fitted.ccn_cv . . . 23
generate_dataset . . . 25
multilabel_stratified_folds . . . 26
predict.ccn . . . 27
predict.ccn_cv . . . 28
print.ccn . . . 29
print.ccn_cv . . . 31
residuals.ccn . . . 32
summary.ccn . . . 33
summary.ccn_cv . . . 34

Index **36**

`CCNR-package` *CCNR: Classifier Chain Network for Multi-Label Classification*

Description

Fits a Classifier Chain Network (CCN) for multi-label binary classification. The model jointly learns label dependencies through a chain structure and optimizes an Lq norm aggregated cross-entropy loss. Provides methods for prediction, model summary, and k-fold cross-validation with grid search over hyperparameters. The methodology is described in Touw and Van de Velden (2025) doi:10.1016/j.eswa.2025.128048.

Author(s)

Maintainer: Daniel J.W. Touw <touw@ese.eur.nl> (ORCID)

Authors:

- Daniel J.W. Touw <touw@ese.eur.nl> (ORCID)

Other contributors:

- Michel Van de Velden (ORCID) (PhD advisor of Daniel J.W. Touw) [thesis advisor]

See Also

Useful links:

- https://github.com/djwtouw/CCNR
- Report bugs at https://github.com/djwtouw/CCNR/issues

`amphibians` *Presence of amphibian species at Polish water reservoirs*

Description

A multilabel data set describing 189 water reservoirs in Poland. The explanatory variables summarize the site: the size and type of the reservoir, the surrounding land use, and the proximity of roads and buildings. The seven binary labels record whether a given amphibian species is present. Species prevalence differs considerably, from 78\ frog down to 11\ realistic illustration of multilabel problems with rare labels.

Usage

```
amphibians
```

Format

A list with two elements:

X Numeric matrix of principal components (189×25), with columns `PC1` to `PC25`.

Y Binary integer matrix of species presence (189×7), with columns `GreenFrog` (green frogs), `BrownFrog` (brown frogs), `Toad` (common toad), `FireToad` (fire-bellied toad), `TreeFrog` (tree frog), `Newt` (common newt), and `CrestNewt` (great crested newt).

Details

The raw data hold 15 site characteristics, 10 of which are categorical. After one-hot encoding those, 56 variables remain. To keep the number of explanatory variables manageable, principal component analysis is applied to the centered and scaled variables, and the smallest number of components explaining at least 80\ components at 82\ so that printed output stays readable.

The script that produced the stored object from the raw file, together with the raw file itself, is available under `data-raw` in the package sources at https://github.com/djwtouw/CCNR.

Source

UCI Machine Learning Repository, Amphibians data set, doi:10.24432/C5H90N. Described in Blachnik, M., Soltysiak, M., and Dabrowska, D. (2019) "Predicting Presence of Amphibian Species Using Features Obtained from GIS and Satellite Images", ISPRS International Journal of Geo-Information, 8(3), 123, doi:10.3390/ijgi8030123.

Examples

```
# Load the data
data(amphibians)

X <- amphibians$X
Y <- amphibians$Y

# Principal components (189 x 25) and species presence (189 x 7)
dim(X)
dim(Y)

# Prevalence of each species
round(colMeans(Y), 3)

# Fit a classifier chain network
model <- ccn(X, Y, q = 1.5, lambda = 0.1)
model
```

---

ccn *Fit a Classifier Chain Network*

Description

Fit a Classifier Chain Network

Usage

```
ccn(
  x,
  y,
  q,
  lambda,
  label_order = NULL,
  conv_tol = 1e-06,
  n_starts = 1L,
  restart_scale = 0.1
)
```

Arguments

x	A numeric matrix of predictors (n x m).
y	A numeric matrix of binary responses (n x L).
q	Numeric; exponent of the per-observation Lq norm used to aggregate label losses, must be ≥ 1. Given per-label losses h_{il}, the per-observation loss is $(\sum_l h_{il}^q)^{1/q}$. When q = 1 all labels contribute equally. As q increases, larger label losses receive disproportionately more weight. As q $\to \infty$ this approaches the infinity norm, where only the worst-predicted label drives the loss.
lambda	Non-negative numeric; regularization penalty.
label_order	Integer vector of length L specifying the order in which labels are chained. Entry label_order[l] gives the index of the original column of y that occupies position l in the chain. Earlier labels in the chain are used as features when predicting later labels. If NULL (default), the natural column order 1:L is used. Outputs are always returned in the original column order of y. A function is also accepted, and is called as label_order(x, y) on the data supplied to ccn before fitting; it must return a permutation of 1:L. This defers the choice of order to fit time, so that under cross-validation every fold derives its order from its own training observations. See entropy_label_order for ready made rules and ccn_cv for comparing several of them.
conv_tol	Convergence tolerance for the optimizer. Default 1e-6.
n_starts	Number of random restarts; the fit with the lowest training loss is returned. The first restart always uses a deterministic initialization; subsequent restarts draw from $N(0, \sigma^2)$ where σ is restart_scale. Default 1.
restart_scale	Standard deviation of the normal distribution used to generate random starting values for restarts 2 through n_starts. Ignored when n_starts = 1. Default 0.1.

Value

An object of class "ccn".

See Also

predict.ccn, summary.ccn, coef.ccn, fitted.ccn, residuals.ccn, ccn_cv for hyperparameter selection via cross-validation.

Examples

```
# Generate sample data
set.seed(2)
data <- generate_dataset(n = 200)

# Fit a classifier chain network
fit <- ccn(x = data$X, y = data$Y, q = 1, lambda = 0.01)
fit

# Inspect the estimated coefficients
```

```
coef(fit)

# In-sample predicted classes and probabilities
head(fitted(fit))
head(fitted(fit, type = "response"))

# Fit with a custom label order: labels earlier in the chain are used as
# features for later labels, outputs remain in the original column order
fit_lo <- ccn(x = data$X, y = data$Y, q = 1, lambda = 0.01,
              label_order = c(4, 1, 3, 2))
fit_lo
```

---

ccn_cv *Cross-validation for a Classifier Chain Network*

Description

Performs k-fold cross-validation over a grid of hyperparameters, returning the best model refitted on the full data.

Usage

```
ccn_cv(
  x,
  y,
  q,
  lambda,
  folds = 5L,
  fold_assignments = NULL,
  scoring = ccn_hamming_accuracy,
  seed = NULL,
  label_order = NULL,
  verbose = 0,
  ...
)
```

Arguments

x	A numeric matrix of predictors (n x m).
y	A numeric matrix of binary responses (n x L).
q	Numeric vector; candidate values for the Lq norm exponent. All combinations with lambda are evaluated.
lambda	Numeric vector; candidate regularization penalties. All combinations with q are evaluated.
folds	Positive integer; number of cross-validation folds. Ignored if fold_assignments is provided. Default 5L.

`fold_assignments`
: Optional integer vector of length n assigning each observation to a fold. Allows the same folds to be reused across methods for fair comparison. If `NULL` (default), folds are generated internally by `multilabel_stratified_folds` using `seed`, so that the proportion of each label is kept roughly constant across folds.

`scoring`
: A function returning a single numeric value, called with the named arguments `y_true` (the observed labels of the held out fold), `y_class` (the predicted labels) and `y_prob` (the predicted probabilities). A scoring function that does not need all three should declare only the ones it uses and absorb the rest in `...`, as the built-in ones do. Higher values are assumed to be better. Defaults to `ccn_hamming_accuracy`.

`seed`
: Integer; random seed for fold generation. Ignored if `fold_assignments` is provided. Default `NULL`.

`label_order`
: Candidate chain orders to search over, alongside `q` and `lambda`. A list whose elements are each `NULL` (the column order of `y`), a permutation of `1:L`, or a function `f(x, y)` returning such a permutation. A function is evaluated inside every fit, so each fold derives its order from its own training observations, which a precomputed permutation cannot do. Element names are used to label the results. A single candidate may be passed directly instead of wrapped in a list. If `NULL` (default), no orders are searched and the column order of `y` is used. See `entropy_label_order` for available ordering heuristics.

`verbose`
: The verbosity of the output. The options are: `0` (default) silent and `1` progress bar.

`...`
: Additional arguments passed to `ccn`.

Value

An object of class `"ccn_cv"` containing:

best_model A `"ccn"` object fitted on the full data with the best hyperparameters. When the winning candidate is a function, it is applied to the full data for this refit.

best_params A one-row data frame with the best `q` and `lambda`, and the name of the best `label_order` when more than one candidate order was searched.

scores A data frame with one row per hyperparameter combination and columns for `q`, `lambda`, mean score, and sd of score across folds, plus a `label_order` column when more than one candidate order was searched.

label_orders The list of candidate orders that was searched, named as in the `label_order` column of `scores`.

fold_assignments Integer vector mapping each observation to a fold, for reproducibility and comparison with other methods.

folds The number of folds used.

See Also

`ccn_hamming_accuracy`, `ccn_exact_match`, `ccn_mean_auc`, `ccn_log_likelihood`, `ccn_micro_f1`, `ccn_macro_f1` for built-in scoring functions; `multilabel_stratified_folds` for the fold generator; `entropy_label_order` for candidate chain orders; `predict.ccn_cv`, `coef.ccn_cv`, `fitted.ccn_cv` for working with the best model.

Examples

```
# Generate sample data
set.seed(2)
data <- generate_dataset(n = 200)

# Cross-validate over a small hyperparameter grid
cv <- ccn_cv(x = data$X, y = data$Y, q = c(1, 2), lambda = c(0.01, 0.1),
             folds = 3, seed = 1)
cv

# Best hyperparameters and the model refitted on the full data
cv$best_params
cv$best_model

# Predict directly from the cross-validation object (uses the best model)
head(predict(cv, newdata = data$X))
head(predict(cv, newdata = data$X, type = "response"))

# Coefficients and fitted values of the best model
coef(cv)
head(fitted(cv))

# Use a different built-in scoring function for hyperparameter selection
cv_f1 <- ccn_cv(x = data$X, y = data$Y, q = c(1, 2), lambda = c(0.01, 0.1),
                folds = 3, seed = 1, scoring = ccn_macro_f1)
cv_f1$best_params

# Use a custom scoring function: it must accept the arguments y_true,
# y_class, and y_prob and return a single number, where higher is better.
# This one rewards predicted probabilities that are close to the true labels
soft_accuracy <- function(y_true, y_class, y_prob) {
    mean(1 - abs(y_true - y_prob))
}
cv_custom <- ccn_cv(x = data$X, y = data$Y, q = c(1, 2), lambda = c(0.01, 0.1),
                    folds = 3, seed = 1, scoring = soft_accuracy)
cv_custom$best_params

# Reuse fold assignments for a fair comparison between settings
cv_reuse <- ccn_cv(x = data$X, y = data$Y, q = 1, lambda = c(0.01, 0.1),
                   fold_assignments = cv$fold_assignments)
cv_reuse$best_params
```

`ccn_exact_match` *Exact match ratio scoring function*

Description

Computes the proportion of observations where all labels are predicted correctly, for use as a `scoring` function in `ccn_cv`. This scoring function is closely related to the zero-one loss, which measures the proportion of observations where at least one label is predicted incorrectly.

Usage

```
ccn_exact_match(y_true, y_class, ...)
```

Arguments

y_true	A binary matrix of true labels (n x L).
y_class	A binary matrix of predicted labels (n x L).
...	Ignored, so that this function can also be called with the predicted probabilities supplied by ccn_cv.

Value

A single numeric value between 0 and 1.

See Also

ccn_cv where this function is intended to be used.

Examples

```
# Generate sample data
set.seed(2)
data <- generate_dataset(n = 200)

# Fit a classifier chain network and obtain predictions
fit <- ccn(x = data$X, y = data$Y, q = 1, lambda = 0.01)
y_class <- predict(fit, newdata = data$X, type = "class")
y_prob <- predict(fit, newdata = data$X, type = "response")

# Scoring functions that compare predicted labels to observed labels
ccn_hamming_accuracy(data$Y, y_class)
ccn_exact_match(data$Y, y_class)
ccn_micro_f1(data$Y, y_class)
ccn_macro_f1(data$Y, y_class)

# Scoring functions that work on the predicted probabilities
ccn_log_likelihood(data$Y, y_prob)
ccn_mean_auc(data$Y, y_prob)

# Use a scoring function to select hyperparameters in cross-validation
cv <- ccn_cv(x = data$X, y = data$Y, q = 1, lambda = c(0.01, 0.1),
             folds = 3, seed = 1, scoring = ccn_mean_auc)
cv$best_params
```

`ccn_hamming_accuracy` *Hamming accuracy scoring function*

Description

Computes the proportion of correctly predicted individual labels, for use as a `scoring` function in `ccn_cv`.

Usage

```
ccn_hamming_accuracy(y_true, y_class, ...)
```

Arguments

`y_true`	A binary matrix of true labels (n x L).
`y_class`	A binary matrix of predicted labels (n x L).
`...`	Ignored, so that this function can also be called with the predicted probabilities supplied by `ccn_cv`.

Value

A single numeric value between 0 and 1.

See Also

`ccn_cv` where this function is intended to be used.

Examples

```
# Generate sample data
set.seed(2)
data <- generate_dataset(n = 200)

# Fit a classifier chain network and obtain predictions
fit <- ccn(x = data$X, y = data$Y, q = 1, lambda = 0.01)
y_class <- predict(fit, newdata = data$X, type = "class")
y_prob <- predict(fit, newdata = data$X, type = "response")

# Scoring functions that compare predicted labels to observed labels
ccn_hamming_accuracy(data$Y, y_class)
ccn_exact_match(data$Y, y_class)
ccn_micro_f1(data$Y, y_class)
ccn_macro_f1(data$Y, y_class)

# Scoring functions that work on the predicted probabilities
ccn_log_likelihood(data$Y, y_prob)
ccn_mean_auc(data$Y, y_prob)

# Use a scoring function to select hyperparameters in cross-validation
```

```
cv <- ccn_cv(x = data$X, y = data$Y, q = 1, lambda = c(0.01, 0.1),
             folds = 3, seed = 1, scoring = ccn_mean_auc)
cv$best_params
```

`ccn_log_likelihood` *Mean log-likelihood scoring function*

Description

Computes the mean per-label log-likelihood under the predicted probabilities, for use as a `scoring` function in `ccn_cv`. Averaging over label-observation pairs rather than over observations puts the score on the same scale as `ccn_hamming_accuracy` and keeps it comparable across data sets with a different number of labels.

Usage

```
ccn_log_likelihood(y_true, y_prob, ...)
```

Arguments

`y_true`	A binary matrix of true labels (n x L).
`y_prob`	A numeric matrix of predicted probabilities (n x L).
`...`	Ignored, so that this function can also be called with the predicted labels supplied by `ccn_cv`.

Details

Probabilities are clipped away from 0 and 1 before taking the log, so that a confidently wrong prediction contributes a large but finite penalty instead of `-Inf`.

Value

A single numeric value; less negative values are better.

See Also

`ccn_cv` where this function is intended to be used.

Examples

```
# Generate sample data
set.seed(2)
data <- generate_dataset(n = 200)

# Fit a classifier chain network and obtain predictions
fit <- ccn(x = data$X, y = data$Y, q = 1, lambda = 0.01)
y_class <- predict(fit, newdata = data$X, type = "class")
y_prob <- predict(fit, newdata = data$X, type = "response")
```

```
# Scoring functions that compare predicted labels to observed labels
ccn_hamming_accuracy(data$Y, y_class)
ccn_exact_match(data$Y, y_class)
ccn_micro_f1(data$Y, y_class)
ccn_macro_f1(data$Y, y_class)

# Scoring functions that work on the predicted probabilities
ccn_log_likelihood(data$Y, y_prob)
ccn_mean_auc(data$Y, y_prob)

# Use a scoring function to select hyperparameters in cross-validation
cv <- ccn_cv(x = data$X, y = data$Y, q = 1, lambda = c(0.01, 0.1),
             folds = 3, seed = 1, scoring = ccn_mean_auc)
cv$best_params
```

`ccn_macro_f1` *Macro-F1 scoring function*

Description

Computes the macro-averaged F1 score, where F1 is computed per label and then averaged. This weights each label equally regardless of class imbalance.

Usage

```
ccn_macro_f1(y_true, y_class, ...)
```

Arguments

`y_true`	A binary matrix of true labels (n x L).
`y_class`	A binary matrix of predicted labels (n x L).
`...`	Ignored, so that this function can also be called with the predicted probabilities supplied by `ccn_cv`.

Value

A single numeric value between 0 and 1.

See Also

`ccn_cv` where this function is intended to be used.

Examples

```
# Generate sample data
set.seed(2)
data <- generate_dataset(n = 200)

# Fit a classifier chain network and obtain predictions
fit <- ccn(x = data$X, y = data$Y, q = 1, lambda = 0.01)
y_class <- predict(fit, newdata = data$X, type = "class")
y_prob <- predict(fit, newdata = data$X, type = "response")

# Scoring functions that compare predicted labels to observed labels
ccn_hamming_accuracy(data$Y, y_class)
ccn_exact_match(data$Y, y_class)
ccn_micro_f1(data$Y, y_class)
ccn_macro_f1(data$Y, y_class)

# Scoring functions that work on the predicted probabilities
ccn_log_likelihood(data$Y, y_prob)
ccn_mean_auc(data$Y, y_prob)

# Use a scoring function to select hyperparameters in cross-validation
cv <- ccn_cv(x = data$X, y = data$Y, q = 1, lambda = c(0.01, 0.1),
             folds = 3, seed = 1, scoring = ccn_mean_auc)
cv$best_params
```

`ccn_mean_auc` *Mean AUC scoring function*

Description

Computes the mean per-label AUC, for use as a `scoring` function in `ccn_cv`.

Usage

```
ccn_mean_auc(y_true, y_prob, ...)
```

Arguments

`y_true`	A binary matrix of true labels (n x L).
`y_prob`	A numeric matrix of predicted probabilities (n x L).
`...`	Ignored, so that this function can also be called with the predicted labels supplied by `ccn_cv`.

Value

A single numeric value between 0 and 1.

See Also

ccn_cv where this function is intended to be used.

Examples

```
# Generate sample data
set.seed(2)
data <- generate_dataset(n = 200)

# Fit a classifier chain network and obtain predictions
fit <- ccn(x = data$X, y = data$Y, q = 1, lambda = 0.01)
y_class <- predict(fit, newdata = data$X, type = "class")
y_prob <- predict(fit, newdata = data$X, type = "response")

# Scoring functions that compare predicted labels to observed labels
ccn_hamming_accuracy(data$Y, y_class)
ccn_exact_match(data$Y, y_class)
ccn_micro_f1(data$Y, y_class)
ccn_macro_f1(data$Y, y_class)

# Scoring functions that work on the predicted probabilities
ccn_log_likelihood(data$Y, y_prob)
ccn_mean_auc(data$Y, y_prob)

# Use a scoring function to select hyperparameters in cross-validation
cv <- ccn_cv(x = data$X, y = data$Y, q = 1, lambda = c(0.01, 0.1),
             folds = 3, seed = 1, scoring = ccn_mean_auc)
cv$best_params
```

---

ccn_micro_f1 *Micro-F1 scoring function*

Description

Computes the micro-averaged F1 score, where the confusion matrix is accumulated across all labels before computing F1. This weights each label-observation pair equally, so labels with more positive instances contribute more to the score.

Usage

```
ccn_micro_f1(y_true, y_class, ...)
```

Arguments

y_true	A binary matrix of true labels (n x L).
y_class	A binary matrix of predicted labels (n x L).
...	Ignored, so that this function can also be called with the predicted probabilities supplied by ccn_cv.

Value

A single numeric value between 0 and 1.

See Also

ccn_cv where this function is intended to be used.

Examples

```
# Generate sample data
set.seed(2)
data <- generate_dataset(n = 200)

# Fit a classifier chain network and obtain predictions
fit <- ccn(x = data$X, y = data$Y, q = 1, lambda = 0.01)
y_class <- predict(fit, newdata = data$X, type = "class")
y_prob <- predict(fit, newdata = data$X, type = "response")

# Scoring functions that compare predicted labels to observed labels
ccn_hamming_accuracy(data$Y, y_class)
ccn_exact_match(data$Y, y_class)
ccn_micro_f1(data$Y, y_class)
ccn_macro_f1(data$Y, y_class)

# Scoring functions that work on the predicted probabilities
ccn_log_likelihood(data$Y, y_prob)
ccn_mean_auc(data$Y, y_prob)

# Use a scoring function to select hyperparameters in cross-validation
cv <- ccn_cv(x = data$X, y = data$Y, q = 1, lambda = c(0.01, 0.1),
             folds = 3, seed = 1, scoring = ccn_mean_auc)
cv$best_params
```

---

coef.ccn *Extract coefficients from a Classifier Chain Network*

Description

Unlike predictions and fitted values, which are returned in the original column order of y, the coefficients are in chain order: position l refers to the label that occupies position l in the chain, that is column label_order[l] of y. The names carried by b, the columns of W, and both margins of C identify which label each position belongs to. The rows of W are named after the columns of x.

Usage

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

Arguments

object	A fitted "ccn" object.
...	Further arguments passed to or from other methods.

Value

A list with elements b (intercepts), W (coefficients of the explanatory variables), and C (coefficients of the label predictions).

Examples

```
# Generate sample data
set.seed(2)
data <- generate_dataset(n = 200)

# Fit a classifier chain network
fit <- ccn(x = data$X, y = data$Y, q = 1, lambda = 0.01)

# Print a compact overview and a full summary of the model
print(fit)
summary(fit)

# With many explanatory variables, W dominates the summary and
# can be left out
summary(fit, which = c("b", "C"))

# Coefficients: intercepts (b), coefficients of the explanatory
# variables (W), and coefficients of the label predictions (C)
coef(fit)

# Fitted classes and fitted probabilities
head(fitted(fit))
head(fitted(fit, type = "response"))

# Response residuals: y minus the fitted probabilities
head(residuals(fit))
```

---

coef.ccn_cv *Extract Coefficients from a Cross-validated Classifier Chain Network*

Description

Extracts the coefficients of the best model selected by cross-validation. Equivalent to calling coef.ccn on object$best_model.

Usage

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

Arguments

object	A fitted "ccn_cv" object.
...	Further arguments passed to coef.ccn.

Value

A list with elements b (intercepts), W (coefficients of the explanatory variables), and C (coefficients of the label predictions).

See Also

ccn_cv, coef.ccn.

Examples

```
# Generate sample data
set.seed(2)
data <- generate_dataset(n = 200)

# Cross-validate over a small hyperparameter grid
cv <- ccn_cv(x = data$X, y = data$Y, q = c(1, 2), lambda = c(0.01, 0.1),
             folds = 3, seed = 1)
cv

# Best hyperparameters and the model refitted on the full data
cv$best_params
cv$best_model

# Predict directly from the cross-validation object (uses the best model)
head(predict(cv, newdata = data$X))
head(predict(cv, newdata = data$X, type = "response"))

# Coefficients and fitted values of the best model
coef(cv)
head(fitted(cv))

# Use a different built-in scoring function for hyperparameter selection
cv_f1 <- ccn_cv(x = data$X, y = data$Y, q = c(1, 2), lambda = c(0.01, 0.1),
                folds = 3, seed = 1, scoring = ccn_macro_f1)
cv_f1$best_params

# Use a custom scoring function: it must accept the arguments y_true,
# y_class, and y_prob and return a single number, where higher is better.
# This one rewards predicted probabilities that are close to the true labels
soft_accuracy <- function(y_true, y_class, y_prob) {
    mean(1 - abs(y_true - y_prob))
}
cv_custom <- ccn_cv(x = data$X, y = data$Y, q = c(1, 2), lambda = c(0.01, 0.1),
                    folds = 3, seed = 1, scoring = soft_accuracy)
cv_custom$best_params
```

```
# Reuse fold assignments for a fair comparison between settings
cv_reuse <- ccn_cv(x = data$X, y = data$Y, q = 1, lambda = c(0.01, 0.1),
                   fold_assignments = cv$fold_assignments)
cv_reuse$best_params
```

`conditional_entropy_matrix` *Pairwise Conditional Entropies of the Labels*

Description

Computes the conditional entropy of every label given every other label, estimated by plugging the joint frequency table of the two labels into the definition. Entry `[i, j]` of the result is $H(y_j \mid y_i)$, the uncertainty left in label `j` once label `i` is known, in bits. The diagonal is zero.

Usage

```
conditional_entropy_matrix(y)
```

Arguments

`y`	A numeric matrix of binary responses (n x L), values in `{0, 1}`.

Details

Row sums and column sums answer different questions. The row sum $\sum_j H(y_j \mid y_i)$ is small when label `i` explains the other labels well, while the column sum $\sum_i H(y_j \mid y_i)$ is small when the other labels explain label `j` well. `entropy_label_order` builds chain orders from these sums.

A label that is constant contributes zero entropy, and conditioning on it leaves the entropy of the other label unchanged.

Value

An L x L numeric matrix of conditional entropies in bits, with `dimnames` taken from the column names of `y`, or `L1` up to `LL` when `y` carries none.

References

Jun, X., Lu, Y., Lei, Z. and Guolun, D. (2019). Conditional entropy based classifier chains for multi-label classification. *Neurocomputing*, 335, 185-194.

See Also

`entropy_label_order`, which turns this matrix into a chain order for the `label_order` argument of `ccn`.

Examples

```
# Generate sample data
set.seed(2)
data <- generate_dataset(n = 200)

# Pairwise conditional entropies of the labels, in bits: entry [i, j] is
# H(y_j | y_i), the uncertainty left in label j once label i is known. Row sums
# say how much a label explains the others, column sums how much the others
# explain it, and the ordering rules below aggregate one or the other.
H <- conditional_entropy_matrix(data$Y)
round(H, 3)
round(rowSums(H), 3)
round(colSums(H), 3)

# Each rule turns that matrix into a chain order. Different rules often agree
# on a given dataset, and they need not agree on a subsample of it.
for (method in c("cebcc1", "cebcc2", "cebcc3", "cebcc4", "ebcc")) {
    cat(method, ":", entropy_label_order(data$Y, method), "\n")
}

# Fit with an order computed up front
order_1 <- entropy_label_order(data$Y, method = "cebcc1")
fit <- ccn(x = data$X, y = data$Y, q = 1, lambda = 0.01,
           label_order = order_1)
fit$label_order

# Or hand ccn the rule itself. label_order also accepts a function f(x, y)
# returning a permutation, evaluated at the start of every fit, so the order
# always comes from the observations that fit actually sees.
cebcc1_order <- function(x, y) entropy_label_order(y, method = "cebcc1")

fit_rule <- ccn(x = data$X, y = data$Y, q = 1, lambda = 0.01,
                label_order = cebcc1_order)
fit_rule$label_order

# On a subsample the same rule can pick a different order, which is exactly
# what makes it worth deferring the choice to fit time
sub <- ccn(x = data$X[1:60, ], y = data$Y[1:60, ], q = 1, lambda = 0.01,
           label_order = cebcc1_order)
sub$label_order

# Cross-validate the rules against each other and against the column order of
# y (the NULL candidate). Passing rules rather than permutations keeps the
# comparison honest: a permutation computed on all of y has already seen the
# validation folds, which flatters it. Any function works here, so a rule of
# your own is just another candidate.
cebcc3_order  <- function(x, y) entropy_label_order(y, method = "cebcc3")
reverse_order <- function(x, y) rev(seq_len(ncol(y)))

cv <- ccn_cv(x = data$X, y = data$Y, q = c(1, 2), lambda = 0.01, folds = 3,
             seed = 1,
             label_order = list(default  = NULL,
```

```
                                cebcc1   = cebcc1_order,
                                cebcc3   = cebcc3_order,
                                reversed = reverse_order))
cv$scores
cv$best_params

# The winning candidate is refitted on the full data, so a rule is applied
# again there rather than reusing a fold's order
cv$best_model$label_order
```

`entropy_label_order` *Chain Order Derived from the Entropy of the Labels*

Description

Determines the order in which labels are chained from the labels alone, following Jun et al. (2019). All rules place labels that are more certain earlier in the chain, so that error propagates from well-determined labels toward poorly determined ones rather than the other way around. The result is a permutation suitable for the `label_order` argument of `ccn`.

Usage

```
entropy_label_order(
  y,
  method = c("cebcc1", "cebcc2", "cebcc3", "cebcc4", "ebcc")
)
```

Arguments

`y`	A numeric matrix of binary responses (n x L), values in `{0, 1}`.
`method`	Character; the ordering rule, one of `"cebcc1"` (default), `"cebcc2"`, `"cebcc3"`, `"cebcc4"`, or `"ebcc"`. See details.

Details

Writing `H` for the matrix returned by `conditional_entropy_matrix`, the methods are:

`"cebcc1"` The label with the smallest row sum of `H` is placed at the end of the chain. The best performing rule in Jun et al. (2019), hence the default.

`"cebcc2"` The label with the largest row sum is placed at the beginning of the chain.

`"cebcc3"` The label with the smallest column sum is placed at the beginning of the chain.

`"cebcc4"` The label with the largest column sum is placed at the end of the chain.

`"ebcc"` Labels are sorted by increasing marginal entropy $H(y_l)$, ignoring the dependence between labels.

The four conditional entropy rules apply their criterion greedily: after placing a label, its row and column are dropped and the sums are recomputed over the labels not yet placed. The rules therefore differ even where they agree on the criterion. Ties are resolved toward the lowest remaining label index, so the result can depend on the column order of `y`.

Deriving an order from the same labels that are later used for evaluation leaks information. Passing a function to the `label_order` argument of `ccn` avoids that, because every fit then derives its order from the observations it is given; see `ccn_cv` for cross-validating several rules against each other.

Value

An integer vector of length L containing a permutation of `1:L`, in chain order.

References

Jun, X., Lu, Y., Lei, Z. and Guolun, D. (2019). Conditional entropy based classifier chains for multi-label classification. *Neurocomputing*, 335, 185-194.

See Also

`conditional_entropy_matrix` for the underlying quantities, `ccn` for the `label_order` argument, and `ccn_cv` for selecting among several orders.

Examples

```
# Generate sample data
set.seed(2)
data <- generate_dataset(n = 200)

# Pairwise conditional entropies of the labels, in bits: entry [i, j] is
# H(y_j | y_i), the uncertainty left in label j once label i is known. Row sums
# say how much a label explains the others, column sums how much the others
# explain it, and the ordering rules below aggregate one or the other.
H <- conditional_entropy_matrix(data$Y)
round(H, 3)
round(rowSums(H), 3)
round(colSums(H), 3)

# Each rule turns that matrix into a chain order. Different rules often agree
# on a given dataset, and they need not agree on a subsample of it.
for (method in c("cebcc1", "cebcc2", "cebcc3", "cebcc4", "ebcc")) {
    cat(method, ":", entropy_label_order(data$Y, method), "\n")
}

# Fit with an order computed up front
order_1 <- entropy_label_order(data$Y, method = "cebcc1")
fit <- ccn(x = data$X, y = data$Y, q = 1, lambda = 0.01,
           label_order = order_1)
fit$label_order

# Or hand ccn the rule itself. label_order also accepts a function f(x, y)
# returning a permutation, evaluated at the start of every fit, so the order
```

```
# always comes from the observations that fit actually sees.
cebcc1_order <- function(x, y) entropy_label_order(y, method = "cebcc1")

fit_rule <- ccn(x = data$X, y = data$Y, q = 1, lambda = 0.01,
                label_order = cebcc1_order)
fit_rule$label_order

# On a subsample the same rule can pick a different order, which is exactly
# what makes it worth deferring the choice to fit time
sub <- ccn(x = data$X[1:60, ], y = data$Y[1:60, ], q = 1, lambda = 0.01,
           label_order = cebcc1_order)
sub$label_order

# Cross-validate the rules against each other and against the column order of
# y (the NULL candidate). Passing rules rather than permutations keeps the
# comparison honest: a permutation computed on all of y has already seen the
# validation folds, which flatters it. Any function works here, so a rule of
# your own is just another candidate.
cebcc3_order  <- function(x, y) entropy_label_order(y, method = "cebcc3")
reverse_order <- function(x, y) rev(seq_len(ncol(y)))

cv <- ccn_cv(x = data$X, y = data$Y, q = c(1, 2), lambda = 0.01, folds = 3,
             seed = 1,
             label_order = list(default  = NULL,
                                cebcc1   = cebcc1_order,
                                cebcc3   = cebcc3_order,
                                reversed = reverse_order))
cv$scores
cv$best_params

# The winning candidate is refitted on the full data, so a rule is applied
# again there rather than reusing a fold's order
cv$best_model$label_order
```

`fitted.ccn` *Extract fitted values from a Classifier Chain Network*

Description

Extract fitted values from a Classifier Chain Network

Usage

```
## S3 method for class 'ccn'
fitted(object, type = c("class", "response"), ...)
```

Arguments

`object`	A fitted "ccn" object.

type	Character; "class" (default) returns binary predictions, "response" returns probabilities.
...	Further arguments passed to or from other methods.

Value

An n x L matrix of fitted values in the original column order of y.

Examples

```
# Generate sample data
set.seed(2)
data <- generate_dataset(n = 200)

# Fit a classifier chain network
fit <- ccn(x = data$X, y = data$Y, q = 1, lambda = 0.01)

# Print a compact overview and a full summary of the model
print(fit)
summary(fit)

# With many explanatory variables, W dominates the summary and
# can be left out
summary(fit, which = c("b", "C"))

# Coefficients: intercepts (b), coefficients of the explanatory
# variables (W), and coefficients of the label predictions (C)
coef(fit)

# Fitted classes and fitted probabilities
head(fitted(fit))
head(fitted(fit, type = "response"))

# Response residuals: y minus the fitted probabilities
head(residuals(fit))
```

---

fitted.ccn_cv *Extract Fitted Values from a Cross-validated Classifier Chain Network*

Description

Extracts the fitted values of the best model selected by cross-validation, which is refitted on the full data. Equivalent to calling fitted.ccn on object$best_model.

Usage

```
## S3 method for class 'ccn_cv'
fitted(object, type = c("class", "response"), ...)
```

Arguments

object	A fitted "ccn_cv" object.
type	Character; "class" (default) returns binary predictions, "response" returns probabilities.
...	Further arguments passed to fitted.ccn.

Value

An n x L matrix of fitted values in the original column order of y.

See Also

ccn_cv, fitted.ccn.

Examples

```
# Generate sample data
set.seed(2)
data <- generate_dataset(n = 200)

# Cross-validate over a small hyperparameter grid
cv <- ccn_cv(x = data$X, y = data$Y, q = c(1, 2), lambda = c(0.01, 0.1),
             folds = 3, seed = 1)
cv

# Best hyperparameters and the model refitted on the full data
cv$best_params
cv$best_model

# Predict directly from the cross-validation object (uses the best model)
head(predict(cv, newdata = data$X))
head(predict(cv, newdata = data$X, type = "response"))

# Coefficients and fitted values of the best model
coef(cv)
head(fitted(cv))

# Use a different built-in scoring function for hyperparameter selection
cv_f1 <- ccn_cv(x = data$X, y = data$Y, q = c(1, 2), lambda = c(0.01, 0.1),
                folds = 3, seed = 1, scoring = ccn_macro_f1)
cv_f1$best_params

# Use a custom scoring function: it must accept the arguments y_true,
# y_class, and y_prob and return a single number, where higher is better.
# This one rewards predicted probabilities that are close to the true labels
soft_accuracy <- function(y_true, y_class, y_prob) {
    mean(1 - abs(y_true - y_prob))
}
cv_custom <- ccn_cv(x = data$X, y = data$Y, q = c(1, 2), lambda = c(0.01, 0.1),
                    folds = 3, seed = 1, scoring = soft_accuracy)
cv_custom$best_params
```

```
# Reuse fold assignments for a fair comparison between settings
cv_reuse <- ccn_cv(x = data$X, y = data$Y, q = 1, lambda = c(0.01, 0.1),
                   fold_assignments = cv$fold_assignments)
cv_reuse$best_params
```

`generate_dataset` *Generate a synthetic multilabel classification dataset*

Description

Generates correlated explanatory variables and multilabel binary outcomes with label interdependence. The dataset always has $m = 3$ explanatory variables and $L = 4$ binary labels.

Usage

```
generate_dataset(n)
```

Arguments

`n`	Integer. Number of observations.

Details

The explanatory variables are sampled from a multivariate normal distribution with compound-symmetry covariance (variance 2, covariance 0.4). Each label is generated using a logistic model with random coefficients drawn from $U(-2, 2)$ and optional dependence on all previous labels via coefficients drawn from $U(-4, 4)$.

Value

A list with:

X Numeric matrix of explanatory variables ($n \times 3$).

Y Binary integer matrix of multilabel outcomes ($n \times 4$).

Examples

```
# Generate sample data
set.seed(123)
data <- generate_dataset(n = 100)

# Explanatory variables (n x 3) and binary labels (n x 4)
dim(data$X)
dim(data$Y)

head(data$X)
head(data$Y)
```

`multilabel_stratified_folds` *Stratified fold assignments for multi-label data*

Description

Splits observations into `folds` groups by iterative stratification, keeping the proportion of positive examples of every label as close as possible to its proportion in the full data. This produces folds in which each label typically has both classes present, which is desirable for cross-validation of multi-label models. The result plugs directly into the `fold_assignments` argument of `ccn_cv`.

Usage

```
multilabel_stratified_folds(y, folds = 5L, seed = NULL)
```

Arguments

`y`	A numeric matrix of binary responses (n x L), values in `{0, 1}`.
`folds`	Positive integer; number of folds. Default `5L`.
`seed`	Integer or `NULL`; random seed controlling the visiting order and the tie-breaking. `NULL` (default) uses the current random number stream.

Details

The algorithm is that of Sechidis, Tsoumakas and Vlahavas (2011), "On the Stratification of Multi-Label Data". Rows are placed one label at a time, rarest label first, each into the fold that most needs an example of that label. Every tie is broken at random, as in the original paper, so repeated calls give different (equally stratified) assignments. `seed = NULL` draws from the current random number stream (set `set.seed` beforehand for reproducibility); a `seed` gives a reproducible assignment.

Value

An integer vector of length n assigning each observation to a fold in `1:folds`.

References

Sechidis, K., Tsoumakas, G. and Vlahavas, I. (2011). On the Stratification of Multi-Label Data. In *Machine Learning and Knowledge Discovery in Databases* (ECML PKDD), pp. 145-158.

See Also

`ccn_cv`, which uses this to generate folds when no `fold_assignments` are supplied.

predict.ccn *Predict from a Classifier Chain Network*

Description

Predict from a Classifier Chain Network

Usage

```
## S3 method for class 'ccn'
predict(
  object,
  newdata = NULL,
  type = c("class", "response"),
  threshold = 0.5,
  ...
)
```

Arguments

object	A fitted "ccn" object.
newdata	Numeric matrix of predictors (n x m). If NULL, returns the fitted values from training.
type	Character; the type of prediction. "class" (default) returns a binary 0/1 matrix. "response" returns predicted probabilities.
threshold	Numeric; decision threshold for converting probabilities to class predictions. Either a scalar applied to all labels or a vector of length L for per-label thresholds. Default 0.5. Ignored when type = "response".
...	Further arguments passed to or from other methods.

Value

An n x L matrix of predictions, in the original column order of y used during training.

See Also

ccn for fitting a model.

Examples

```
# Generate sample data and split into training and test sets
set.seed(2)
data <- generate_dataset(n = 250)
train <- 1:200

# Fit a classifier chain network on the training data
fit <- ccn(x = data$X[train, ], y = data$Y[train, ], q = 1, lambda = 0.01)
```

```
# Predicted classes and probabilities for the test data
x_test <- data$X[-train, ]
head(predict(fit, newdata = x_test))
head(predict(fit, newdata = x_test, type = "response"))

# Without newdata, predictions for the training data are returned
head(predict(fit))

# Custom decision threshold, either a scalar for all labels or one
# threshold per label
head(predict(fit, newdata = x_test, threshold = 0.3))
head(predict(fit, newdata = x_test, threshold = c(0.3, 0.5, 0.5, 0.7)))
```

`predict.ccn_cv` *Predict from a Cross-validated Classifier Chain Network*

Description

Computes predictions from the best model selected by cross-validation. Equivalent to calling `predict.ccn` on `object$best_model`.

Usage

```
## S3 method for class 'ccn_cv'
predict(
  object,
  newdata = NULL,
  type = c("class", "response"),
  threshold = 0.5,
  ...
)
```

Arguments

`object`	A fitted `"ccn_cv"` object.
`newdata`	Numeric matrix of predictors (n x m). If `NULL`, returns the fitted values from training.
`type`	Character; the type of prediction. `"class"` (default) returns a binary 0/1 matrix. `"response"` returns predicted probabilities.
`threshold`	Numeric; decision threshold for converting probabilities to class predictions. Either a scalar applied to all labels or a vector of length L for per-label thresholds. Default `0.5`. Ignored when `type = "response"`.
`...`	Further arguments passed to `predict.ccn`.

Value

An n x L matrix of predictions, in the original column order of y used during training.

See Also

ccn_cv, predict.ccn.

Examples

```
# Generate sample data
set.seed(2)
data <- generate_dataset(n = 200)

# Cross-validate over a small hyperparameter grid
cv <- ccn_cv(x = data$X, y = data$Y, q = c(1, 2), lambda = c(0.01, 0.1),
             folds = 3, seed = 1)
cv

# Best hyperparameters and the model refitted on the full data
cv$best_params
cv$best_model

# Predict directly from the cross-validation object (uses the best model)
head(predict(cv, newdata = data$X))
head(predict(cv, newdata = data$X, type = "response"))

# Coefficients and fitted values of the best model
coef(cv)
head(fitted(cv))

# Use a different built-in scoring function for hyperparameter selection
cv_f1 <- ccn_cv(x = data$X, y = data$Y, q = c(1, 2), lambda = c(0.01, 0.1),
                folds = 3, seed = 1, scoring = ccn_macro_f1)
cv_f1$best_params

# Use a custom scoring function: it must accept the arguments y_true,
# y_class, and y_prob and return a single number, where higher is better.
# This one rewards predicted probabilities that are close to the true labels
soft_accuracy <- function(y_true, y_class, y_prob) {
    mean(1 - abs(y_true - y_prob))
}
cv_custom <- ccn_cv(x = data$X, y = data$Y, q = c(1, 2), lambda = c(0.01, 0.1),
                    folds = 3, seed = 1, scoring = soft_accuracy)
cv_custom$best_params

# Reuse fold assignments for a fair comparison between settings
cv_reuse <- ccn_cv(x = data$X, y = data$Y, q = 1, lambda = c(0.01, 0.1),
                   fold_assignments = cv$fold_assignments)
cv_reuse$best_params
```

---

print.ccn *Print a Classifier Chain Network*

Description

Print a Classifier Chain Network

Usage

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

Arguments

x	A fitted "ccn" object.
...	Further arguments passed to or from other methods.

Value

Invisibly returns x, the fitted "ccn" object. Called for the side effect of printing a compact overview of the model: its dimensions, hyperparameters, training loss, label names, and chain order.

Examples

```
# Generate sample data
set.seed(2)
data <- generate_dataset(n = 200)

# Fit a classifier chain network
fit <- ccn(x = data$X, y = data$Y, q = 1, lambda = 0.01)

# Print a compact overview and a full summary of the model
print(fit)
summary(fit)

# With many explanatory variables, W dominates the summary and
# can be left out
summary(fit, which = c("b", "C"))

# Coefficients: intercepts (b), coefficients of the explanatory
# variables (W), and coefficients of the label predictions (C)
coef(fit)

# Fitted classes and fitted probabilities
head(fitted(fit))
head(fitted(fit, type = "response"))

# Response residuals: y minus the fitted probabilities
head(residuals(fit))
```

print.ccn_cv *Print a Classifier Chain Network Cross-validation*

Description

Print a Classifier Chain Network Cross-validation

Usage

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

Arguments

x	A fitted "ccn_cv" object.
...	Further arguments passed to or from other methods.

Value

Invisibly returns x, the "ccn_cv" object. Called for the side effect of printing the number of folds, the size of the hyperparameter grid, the selected hyperparameters, and the score of every grid point.

Examples

```
# Generate sample data
set.seed(2)
data <- generate_dataset(n = 200)

# Cross-validate over a small hyperparameter grid
cv <- ccn_cv(x = data$X, y = data$Y, q = c(1, 2), lambda = c(0.01, 0.1),
             folds = 3, seed = 1)
cv

# Best hyperparameters and the model refitted on the full data
cv$best_params
cv$best_model

# Predict directly from the cross-validation object (uses the best model)
head(predict(cv, newdata = data$X))
head(predict(cv, newdata = data$X, type = "response"))

# Coefficients and fitted values of the best model
coef(cv)
head(fitted(cv))

# Use a different built-in scoring function for hyperparameter selection
cv_f1 <- ccn_cv(x = data$X, y = data$Y, q = c(1, 2), lambda = c(0.01, 0.1),
                folds = 3, seed = 1, scoring = ccn_macro_f1)
cv_f1$best_params
```

```
# Use a custom scoring function: it must accept the arguments y_true,
# y_class, and y_prob and return a single number, where higher is better.
# This one rewards predicted probabilities that are close to the true labels
soft_accuracy <- function(y_true, y_class, y_prob) {
    mean(1 - abs(y_true - y_prob))
}
cv_custom <- ccn_cv(x = data$X, y = data$Y, q = c(1, 2), lambda = c(0.01, 0.1),
                    folds = 3, seed = 1, scoring = soft_accuracy)
cv_custom$best_params

# Reuse fold assignments for a fair comparison between settings
cv_reuse <- ccn_cv(x = data$X, y = data$Y, q = 1, lambda = c(0.01, 0.1),
                   fold_assignments = cv$fold_assignments)
cv_reuse$best_params
```

`residuals.ccn` *Extract residuals from a Classifier Chain Network*

Description

Returns the difference between the binary response matrix and the fitted probabilities.

Usage

```
## S3 method for class 'ccn'
residuals(object, ...)
```

Arguments

`object`	A fitted `"ccn"` object.
`...`	Further arguments passed to or from other methods.

Value

An n x L numeric matrix of response residuals (`y - fitted probabilities`) in the original column order of `y`.

Examples

```
# Generate sample data
set.seed(2)
data <- generate_dataset(n = 200)

# Fit a classifier chain network
fit <- ccn(x = data$X, y = data$Y, q = 1, lambda = 0.01)

# Print a compact overview and a full summary of the model
print(fit)
```

```
summary(fit)

# With many explanatory variables, W dominates the summary and
# can be left out
summary(fit, which = c("b", "C"))

# Coefficients: intercepts (b), coefficients of the explanatory
# variables (W), and coefficients of the label predictions (C)
coef(fit)

# Fitted classes and fitted probabilities
head(fitted(fit))
head(fitted(fit, type = "response"))

# Response residuals: y minus the fitted probabilities
head(residuals(fit))
```

`summary.ccn` *Summarize a Classifier Chain Network*

Description

Prints an overview of the fitted model followed by the estimated coefficients. Each set of coefficients carries the names of the labels and of the explanatory variables. For a model with many explanatory variables `W` dominates the output, in which case `which` can be used to leave it out.

Usage

```
## S3 method for class 'ccn'
summary(object, which = c("b", "W", "C"), digits = 4, ...)
```

Arguments

`object`	A fitted `"ccn"` object.
`which`	Character vector selecting which sets of coefficients are printed: any subset of `"b"` (intercepts), `"W"` (coefficients of the explanatory variables), and `"C"` (coefficients of the label predictions). All three are printed by default.
`digits`	Number of decimal places used for the coefficients. Default `4`.
`...`	Further arguments passed to or from other methods.

Details

Of `C` only the rows and columns that hold coefficients are printed: the first row and the last column are omitted, since the first label in the chain has no predecessors and the last precedes none. Entries on and above the diagonal are printed blank. `coef` returns the full `L` by `L` matrix, with `NA` in those positions.

Value

Invisibly returns `object`, the fitted `"ccn"` object. Called for the side effect of printing the overview and the selected sets of coefficients.

Examples

```
# Generate sample data
set.seed(2)
data <- generate_dataset(n = 200)

# Fit a classifier chain network
fit <- ccn(x = data$X, y = data$Y, q = 1, lambda = 0.01)

# Print a compact overview and a full summary of the model
print(fit)
summary(fit)

# With many explanatory variables, W dominates the summary and
# can be left out
summary(fit, which = c("b", "C"))

# Coefficients: intercepts (b), coefficients of the explanatory
# variables (W), and coefficients of the label predictions (C)
coef(fit)

# Fitted classes and fitted probabilities
head(fitted(fit))
head(fitted(fit, type = "response"))

# Response residuals: y minus the fitted probabilities
head(residuals(fit))
```

---

summary.ccn_cv *Summarize Classifier Chain Network Cross-Validation Results*

Description

Summarize Classifier Chain Network Cross-Validation Results

Usage

```
## S3 method for class 'ccn_cv'
summary(object, ...)
```

Arguments

`object`	A fitted `"ccn_cv"` object.
`...`	Further arguments passed to or from other methods.

Value

Invisibly returns `object`, the `"ccn_cv"` object. Called for the side effect of printing the cross-validation results, which are identical to the output of `print`.

Examples

```
# Generate sample data
set.seed(2)
data <- generate_dataset(n = 200)

# Cross-validate over a small hyperparameter grid
cv <- ccn_cv(x = data$X, y = data$Y, q = c(1, 2), lambda = c(0.01, 0.1),
             folds = 3, seed = 1)
cv

# Best hyperparameters and the model refitted on the full data
cv$best_params
cv$best_model

# Predict directly from the cross-validation object (uses the best model)
head(predict(cv, newdata = data$X))
head(predict(cv, newdata = data$X, type = "response"))

# Coefficients and fitted values of the best model
coef(cv)
head(fitted(cv))

# Use a different built-in scoring function for hyperparameter selection
cv_f1 <- ccn_cv(x = data$X, y = data$Y, q = c(1, 2), lambda = c(0.01, 0.1),
                folds = 3, seed = 1, scoring = ccn_macro_f1)
cv_f1$best_params

# Use a custom scoring function: it must accept the arguments y_true,
# y_class, and y_prob and return a single number, where higher is better.
# This one rewards predicted probabilities that are close to the true labels
soft_accuracy <- function(y_true, y_class, y_prob) {
    mean(1 - abs(y_true - y_prob))
}
cv_custom <- ccn_cv(x = data$X, y = data$Y, q = c(1, 2), lambda = c(0.01, 0.1),
                    folds = 3, seed = 1, scoring = soft_accuracy)
cv_custom$best_params

# Reuse fold assignments for a fair comparison between settings
cv_reuse <- ccn_cv(x = data$X, y = data$Y, q = 1, lambda = c(0.01, 0.1),
                   fold_assignments = cv$fold_assignments)
cv_reuse$best_params
```

Index

∗ **datasets**
amphibians, 3

amphibians, 3

ccn, 4, *7*, *18*, *20, 21*, *27*
ccn_cv, *5*, 6, *8–15*, *17*, *21*, *24*, *26*, *29*
ccn_exact_match, *7*, 8
ccn_hamming_accuracy, *7*, 10, *11*
ccn_log_likelihood, *7*, 11
ccn_macro_f1, *7*, 12
ccn_mean_auc, *7*, 13
ccn_micro_f1, *7*, 14
CCNR *(*CCNR-package*)*, 2
CCNR-package, 2
coef, *33*
coef.ccn, *5*, 15, *16, 17*
coef.ccn_cv, *7*, 16
conditional_entropy_matrix, 18, *20, 21*

entropy_label_order, *5*, *7*, *18*, 20

fitted.ccn, *5*, 22, *23, 24*
fitted.ccn_cv, *7*, 23

generate_dataset, 25

multilabel_stratified_folds, *7*, 26

predict.ccn, *5*, 27, *28, 29*
predict.ccn_cv, *7*, 28
print, *35*
print.ccn, 29
print.ccn_cv, 31

residuals.ccn, *5*, 32

set.seed, *26*
summary.ccn, *5*, 33
summary.ccn_cv, 34