Package: clustord (via r-universe)

November 7, 2026

Type Package

Title Cluster Ordinal Data via Proportional Odds or Ordered Stereotype

Version 2.0.1

Date 2026-03-03

Maintainer Louise McMillan <louise.mcmillan@vuw.ac.nz>

Description Biclustering, row clustering and column clustering using the proportional odds model (POM), ordered stereotype model (OSM) or binary model for ordinal categorical data. Fernández, D., Arnold, R., Pledger, S., Liu, I., & Costilla, R. (2019) <doi:10.1007/s11634-018-0324-3>.

License GPL-3

URL https://vuw-clustering.github.io/clustord/

Depends R (>= 3.5.0), stats, utils

Imports Rcpp (>= 1.0.1), MASS, nnet, flexclust, methods

LinkingTo Rcpp, RcppArmadillo

Suggests knitr, formatR, rmarkdown, testthat (>= 2.1.0), multgee, parallel

Encoding UTF-8

RoxygenNote 7.3.2

VignetteBuilder knitr, formatR

Repository https://vuw-clustering.r-universe.dev

Date/Publication 2026-03-03 07:11:58 UTC

RemoteUrl https://github.com/vuw-clustering/clustord

RemoteRef HEAD

RemoteSha d6343dafe812c6df6acdde1e9f19aa5200b2e402

Contents

calc_cluster_comparisons 2
calc_SE_rowcluster 3
clustord 4
mat_to_df 20
osm 21
reorder.clustord 24
rerun 26

Index **29**

`calc_cluster_comparisons` *Calculate comparison measures between two sets of clustering results*

Description

Given two sets of posterior probabilities of membership for clusters, calculate three measures to compare the clustering memberships.

Usage

```
calc_cluster_comparisons(ppr1, ppr2)
```

Arguments

`ppr1`	Posterior probabilities of cluster membership, named `row_cluster_probs` or `column_cluster_probs` in the output of `clustord`. If you have performed bi-clustering, then `ppr1` should be the clustering results for just one of the dimensions i.e. just the row clustering results, or just the column clustering results. The rows of `ppr1` give the entries that have been clustered, and each column corresponds to one cluster.
`ppr2`	Posterior probabilities of cluster membership from a different clustering run, which will be compared to `ppr1`.

Details

The three measures are the Adjusted Rand Index (ARI), the Normalised Variation of Information (NVI) and the Normalised Information Distance (NID).

The three measures are documented in

Value

A list with components:

`ARI`: Adjusted Rand Index.

`NVI`: Normalised Variation of Information.

`NID`: Normalised Information Distance.

References

Fernández, D., & Pledger, S. (2016). Categorising count data into ordinal responses with application to ecological communities. Journal of agricultural, biological, and environmental statistics (JABES), 21(2), 348–362.

`calc_SE_rowcluster` *Calculate standard errors of clustering parameters.*

Description

Calculate SE of parameters fitted using `clustord`.

Usage

```
calc_SE_rowcluster(long_df, clust_out, control_optim = default_control_optim())

calc_SE_bicluster(long_df, clust_out, control_optim = default_control_optim())
```

Arguments

`long_df`	The data frame, in long format, as passed to `clustord`.
`clust_out`	A `clustord` object.
`control_optim`	control list for the `optim` call within the M step of the EM algorithm. See the control list Details in the `optim` manual for more info.

Details

Use `calc_SE_rowcluster` to calculate SE for row clustering and column clustering, or `calc_SE_bicluster` to calculate SE for biclustering.

Calculates SE by running `optimHess` (see `optim`) on the incomplete-data log-likelihood to find the hessian at the fitted parameter values from `clustord`. Then the square roots of the diagonal elements of the negative inverse of the hessian are the standard errors of the parameters i.e. `SE <- sqrt(diag(solve(-optim_hess))`.

Note that SE values are **only** calculated for the independent parameters. For example, if the constraint on the row clustering parameters is set to constraint_sum_zero = TRUE, where the last row clustering parameter is the negative sum of the other parameters, SE values will only be calculated for the first RG-1 parameters, the independent ones. This applies similarly to individual column effect coefficients, etc.

The function requires an input which is the output of `clustord`, which includes the component `out_parvec`, the final vector of independent parameter values from the EM algorithm, which will correspond to a subset of the parameter values in `out_parlist`.

Value

The standard errors corresponding to the elements of `clust_out$out_parvec`.

Functions

- `calc_SE_rowcluster()`: SE for rowclustering
- `calc_SE_bicluster()`: SE for biclustering

`clustord` *Likelihood-based clustering using Ordered Stereotype Models (OSM), Proportional Odds Models (POM) or Binary Models*

Description

Likelihood-based clustering with parameters fitted using the EM algorithm. You can perform clustering on rows or columns of a data matrix, or biclustering on both rows and columns simultaneously. You can include any number of covariates for rows and covariates for columns. Ordinal models used in the package are Ordered Stereotype Model (OSM), Proportional Odds Model (POM) and a dedicated Binary Model for binary data.

Usage

```
clustord(
  formula,
  model,
  RG = NULL,
  CG = NULL,
  long_df,
  init_parvec = NULL,
  init_pi = NULL,
  init_kappa = NULL,
  control_EM = default_control_EM(),
  optim_method = "L-BFGS-B",
  control_optim = default_control_optim(),
  constraint_sum_zero = TRUE,
  start_from_simple_model = TRUE,
  parallel_starts = FALSE,
  nstarts = 5,
  verbose = FALSE
)
```

Arguments

`formula`	model formula (see 'Details').
`model`	`"OSM"` for Ordered Stereotype Model or `"POM"` for Proportional Odds Model or `"Binary"` for binary data model.
`RG`	number of row clustering groups.
`CG`	number of column clustering groups.

`long_df` data frame with at least three columns, `Y` and `ROW` and `COL`. Each row in the data frame corresponds to a single cell in the original data matrix; the response value in that cell is given by `Y`, and the row and column indices of that cell in the matrix are given by `ROW` and `COL`. Use `mat_to_df` to create this data frame from your data matrix of responses. `mat_to_df` also allows you to supply data frames of row or column covariates which will be incorporated into `long_df`.

`init_parvec` (default `NULL`) vector of starting parameter values for the model.

Note: if the user enters an initial vector of parameter values, it is **strongly recommended** that the user also check the values of `init_parlist` in the output object, to **make sure that the constructed parameters are as expected**.

If `NULL`, starting parameter values will be generated automatically.

See 'Details' for definitions of the parameters used for different models.

`init_pi` (default `NULL`) starting parameter values for the proportions of observations in the different row clusters.

If `NULL`, starting values will be generated automatically.

User-specified values of `init_pi` must be of length `(RG-1)` because the final value will be automatically calculated so that the values of `pi` sum to 1.

`init_kappa` (default `NULL`) starting parameter values for the proportions of observations in the different column clusters.

If `NULL`, starting values will be generated automatically.

User-specified values of `init_kappa` must be of length `(CG-1)` because the final value will be automatically calculated so that the values of `kappa` sum to 1.

`control_EM` (default = `list(maxiter=50, EM_likelihood_tol=1e-4, EM_params_tol=1e-2, params_stopping=TRUE, maxiter_start=10, keep_all_params=FALSE, epsilon=1e-6)`) list of parameters controlling the EM algorithm.

`maxiter` controls how many EM iterations of the main EM algorithm are used to fit the chosen submodel.

`EM_likelihood_tol` is the tolerance for the stopping criterion for the **log-likelihood** in the EM algorithm. The criterion is the absolute change in the **incomplete** log-likelihood since the previous iteration, scaled by the size of the dataset `n*p`, where `n` is the number of rows in the data matrix and `p` is the number of columns in the data matrix. The scaling is applied because the incomplete log-likelihood is predominantly affected by the dataset size.

`EM_params_tol` is the tolerance for the stopping criterion for the **parameters** in the EM algorithm. This is a tolerance for the **sum** of the scaled parameter changes from the last iteration, i.e. the tolerance is not for any individual parameter but for the sum of changes in all the parameters. Thus the default tolerance is 1e-2. The individual parameter criteria are the absolute differences between the exponentiated absolute parameter value at the current timestep and the exponentiated absolute parameter value at the previous timestep, as a proportion of the exponentiated absolute parameter value at the current timestep. The exponentiation is to rescale parameter values that are close to zero.

If, for example, there are around 5 independent parameter values, then at the point of convergence using default tolerances for the log-likelihood and the parameters, each parameter will have a scaled absolute change since the previous

iteration of about 1e-4; if there are 20 or 30 independent parameters, then each will have a scaled aboslute change of about 1e-6.

`params_stopping`: if `FALSE`, indicates that the EM algorithm should only check convergence based on the change in incomplete-data log-likelihood, relative to the current difference between the complete-data and incomplete-data log-likelihoods, i.e. `abs(delta_lli)/abs(llc[iter] - lli[iter])`; if `TRUE`, indicates that as well as checking the likelihood criterion, the EM algorithm should also check whether the relative change in the exponentials of the absolute values of the current parameters is below the tolerance `EMstoppingpar`, to see whether the parameters and the likelihood have both converged.

`maxiter_start` controls how many EM iterations are used when fitting the simpler submodels to get starting values for fitting models with interaction.

`keep_all_params`: if `TRUE`, keep a record of parameter values (including pi_r and kappa_c) for every EM iteration.

`rerun_estep_before_lli`: if `TRUE`, and only when using biclustering, rerun the E-step before calculating the incomplete-data log-likelihood. The EM algorithm runs the E-step to estimate the cluster membership probabilities and then runs the M-step to estimate the parameters by maximising the complete-data log-likelihood (LLC), and then recalculates the estimated incomplete-data log-likelihood (LLI) to check for convergence. The biclustering LLI approximation uses the cluster membership probabilities so will be slightly more accurate if these cluster membership probabilities are recalculated using the very latest parameter estimates. So the `TRUE` setting for this control recalculates the cluster memberships before calculating the LLI approx.

`use_latest_lli`: Kept for backwards compatibility with original version of clustord algorithm. Original version of the algorithm had this set to `FALSE`, which keeps the best LLI from any previous iteration rather than the latest LLI. The default, `TRUE`, instead keeps the latest LLI even if there was a better LLI in a previous iteration. This is appropriat: In row clustering the exact LLI is used and the EM algorithm creators proved the LLI should always not decrease with each iteration. In biclustering the approximation to the LLI may be very inaccurate when the algorithm is a long way from convergence, so even when the value of LLI appears to be very high in early iterations this may not be accurate, so it is better to take the latest value.

For `columnclustering`, the parameters saved from each iteration will NOT be converted to column clustering format, and will be in the row clustering format, so `alpha` in `EMstatus$params_every_iteration` will correspond to beta_c and `pi` will correspond to kappa.

`epsilon`: default 1e-6, small value used to adjust values of pi, kappa and theta that are too close to zero so that taking logs of them does not create infinite values.

`optim_method` (default "L-BFGS-B") method to use in optim within the M step of the EM algorithm. Must be one of "L-BFGS-B", "BFGS", "CG" or "Nelder-Mead" (i.e. not the SANN method).

`control_optim` control list for the `optim` call within the M step of the EM algorithm. See the control list Details in the `optim` manual for more info. Please note that although `optim`, by default, uses `pgtol=0` and `factr=1e7` in the L-BFGS-B method,

clustord, by default, alters these to pgtol=1e-4 and factr=1e11, but you can use this control_optim argument in clustord to revert them to the defaults if you want. The reason for the change is that the chosen values in clustord reduce the tolerance on the log-likelihood function optimization in order to speed up the algorithm, and because the log-likelihood is on the scale of 1e4 for <100 rows in the data matrix and 1e6 for 5000 rows in the data matrix, tolerance at the default optim scale is not as important as the choice of model type and structure or the number of starting points. If one model is better than another, it will probably have a likelihood that is better by about the size of the data matrix, which is far larger than the tolerance in the optimization. If one starting point is better than another, it will probably have a likelihood that is better on about 1/10th or 1/100th the size of the data matrix, which is still far larger than the tolerance in the optimization. If you need accurate parameter estimates, firstly make sure to try more starting points, then perform model selection, and then finally rerun the chosen model with finer tolerance, e.g. the optim defaults, pgtol=0 and factr=1e7.

constraint_sum_zero

(default TRUE) if TRUE, use constraints that cluster effects sum to zero; if FALSE, use constraints that the cluster effect for the first cluster will be 0. Both versions have the same constraints for joint row-column cluster effects: these effects are described by a matrix of parameters gamma_rc, indexed by row cluster and column cluster indices, and the constraints are that the final column of gamma_rc is equal to the negative sum of the other columns (so gamma columns sum to zero) and first row of gamma_rc is equal to the negative sum of the other rows (so gamma rows sum to zero).

start_from_simple_model

(default TRUE) if TRUE, fit a simpler clustering model first and use that to provide starting values for all parameters for the model with interactions; if FALSE, use the more basic models to provide starting values only for init_pi and init_kappa. If the full model has interaction terms, then simpler models are ones without the interactions. If the model has individual row/column effects alongside the clusters, then simpler models are ones without the individual row/column effects. If the full model has covariates, then simpler models are ones without the covariates (to get starting values for the cluster parameters), and ones with the covariates but no clustering (to get starting values for the covariates).

parallel_starts

(default FALSE) if TRUE, by generating multiple random starts, those random starts will be parallelised over as many cores as are available. For example, on a personal computer this will be one fewer than the number of cores in the machine, to make sure one is left for system tasks external to R.

nstarts (default 5) number of random starts to generate, if generating random starting points for the EM algorithm.

verbose (default FALSE) changes how much is reported to the console during the algorithm's progress. If TRUE, reports the incomplete-data log-likelihood at every EM algorithm iteration and the trace information from nnet::multinom() during the process of fitting initial values for the mu parameters. If FALSE, the incomplete log-likelihood is only reported every 10 iterations of the EM algorithm and the initial fitting reporting is suppressed. Regardless of the verbosity

setting the algorithm reports each of the random starts and whenever it finds a better log-likelihood than all previous starts, and it also reports when it is fitting simpler models to find starting values for the parameters vs fitting the final, more complex model. If wanting the detailed output from `optim()`, use `control_optim=list(trace=X)`, where X is 1 to 6, with 6 being the highest level of verbosity for the L-BFGS-B algorithm.

Details

You can select your own input parameters, or starting values will be generated by running kmeans or by fitting simpler models and feeding the outputs into the final model as starting values.

The starting point for clustering is a data matrix of response values that are binary or categorical. You may also have a data frame of covariates that are linked to the rows of the data matrix, and may also have a data frame of covariates that are linked to the columns of the data matrix.

For example, if clustering data from fishing trawls, where the rows are trawl events and columns are species caught, then you could also supply a gear covariate linked to the rows, representing gear used on each trawl event, and could additionally supply species covariates linked to the columns, representing auxiliary information about each species. There is no requirement to provide any covariates, and you can provide only row covariates, or only column covariates.

Before running `clustord`, you need to run `mat_to_df` to convert the data matrix into a long form data frame. The data frame needs to have at least three columns, `Y` and `ROW` and `COL`. Each row in the data frame corresponds to a single cell in the original data matrix; the response value in that cell is given by `Y`, and the row and column indices of that cell in the matrix are given by `ROW` and `COL`.

`mat_to_df` also allows you to supply data frames of row or column covariates which will be incorporated into `long_df`.

Then, to run the `clustord` function, you need to enter your chosen formula and model, and the number of clusters you want to fit. The formula model_structure is akin to that for `glm`, but with a few restrictions. You can include any number of covariates in the same way as for a multiple regression model, though unlike for `glm`, you can include both row and column covariates.

Note that, unlike `glm`, you should not specify a `family` argument; the `model` argument is used instead.

`formula` argument details

In the following description of different models, the Binary model is used for simplicity when giving the mathematical descriptions of the models, but you can use any of the following models with the Ordered Stereotype or Proportional Odds Models as well.

In the `formula` argument, the response must be exactly `Y`. You cannot use any functions of `Y` as the response, nor can you include `Y` in any terms on the right hand side of the formula. `Y` is the name in `clustord` of the response values in the original data matrix.

The `formula` argument has 4 special variables: `ROWCLUST`, `COLCLUST`, `ROW` and `COL`. There are some restrictions on how these can be used in the formula, as they are not covariates, but instead act as indicators of the clustering model_structure you want to use.

All other variables in the formula will be any covariates that you want to include in the model, and these are unrestricted, and can be used in the same way as in `glm`.

`ROWCLUST` and `COLCLUST` are used to indicate what row clustering model structure you want, and what column clustering model structure you want, respectively. The inclusion of `ROWCLUST` as a

single term indicates that you want to include a row clustering effect in the model. In the simplest row clustering model, for Binary data with **row clustering** effects only, the basic function call would be

`clustord(Y ~ ROWCLUST, model="Binary", long_df=long_df)`

and the model fitted would have the form:

Logit(P(Y = 1)) = mu + rowc_coef_r

where mu is the intercept term, and rowc_coef_r is the row cluster effect that will be applied to every row from the original data matrix that is a member of row cluster r. The inclusion of `ROWCLUST` corresponds to the inclusion of rowc_coef_r.

Note that we are not using notation involving greek letters, because (a) we ran out of letters for all the different types of parameters in the model and (b) with this many parameters, it would be difficult to remember which ones are which.

Similarly to row clustering, the formula `Y ~ COLCLUST` would perform **column clustering**, with model Logit(P(Y = 1)) = mu + colc_coef_c, where colc_coef_c is the column cluster effect that will be applied to every column from the original data matrix that is a member of column cluster c.

Including both `ROWCLUST` and `COLCLUST` in the same formula indicates that you want to perform biclustering, i.e. you want to cluster the rows and the columns of the original data matrix simultaneously. If included without interaction, then the terms just correspond to including rowc_coef_r and colc_coef_c in the model:

The formula

`Y ~ ROWCLUST + COLCLUST`

is the simplest possible **biclustering** model,

Logit(P(Y = 1)) = mu + rowc_coef_r + colc_coef_c

If you want to include interaction between the rows and columns, i.e. you want to perform block biclustering where each block corresponds to a row cluster r and a column cluster c, then that model has a matrix of parameters indexed by r and c.

`clustord(Y ~ ROWCLUST*COLCLUST, model="Binary", ...)` has the model

Logit(P(Y = 1)) = mu + rowc_colc_coef_rc

This model can instead be called using the equivalent formula

`Y ~ ROWCLUST + COLCLUST + ROWCLUST:COLCLUST`.

You can instead use the formula `Y ~ ROWCLUST:COLCLUST`. Mathematically, this is equivalent to the previous two. In regression, the models would not be equivalent but in clustering, they are equivalent, and have the same number of independent parameters overall. If you include the main effects, then that reduces the number of independent parameters in the interaction term compared to if you just use the interaction term (see below section about `init_parvec`).

You cannot include just one of the main effects alongside the interaction term, i.e. you cannot use `Y ~ ROWCLUST + ROWCLUST:COLCLUST` or `Y ~ COLCLUST + ROWCLUST:COLCLUST`. This is for simplicity in the code, and to avoid confusion when interpreting the results.

However, `clustord` allows a lot more flexibility than this. The variables `ROW` and `COL` are used to indicate that you want to also include **individual row or column effects**, respectively.

For example, if you are clustering binary data that indicates the presence/ absence of different species (columns) at different trawl events (rows), and you know that one particular species is

incredibly common, then you can include column effects in the model, which will allow for the possibility that two columns may correspond to species with different probabilities of appearing in the trawl.

You can add individual column effects along with row clustering, or you can add individual row effects along with column clustering. The formula for row clustering with individual column effects (without interaction) is

`Y ~ ROWCLUST + COL`

which corresponds to Binary model

Logit(P(Y = 1)) = mu + rowc_coef_r + col_coef_j

So if two cells from the data matrix are in the same row cluster, but in different columns, they will not have the same probability of Y = 1.

You can also add interaction between the individual row/column effects and the clustering effects.

If you still want to be able to see the row cluster and column effects separately, then you use `Y ~ ROWCLUST*COL` or `Y ~ ROWCLUST + COL + ROWCLUST:COL` (these are both the same), which have model

Logit(P(Y = 1)) = mu + rowc_coef_r + col_coef_j + rowc_col_coef_rj

As before, rowc_coef_r and col_coef_j are the row cluster effects and individual column effects, and rowc_col_coef_rj are the interaction terms.

Alternatively, you can use the mathematically-equivalent formula

`Y ~ ROWCLUST:COL` which has model

Logit(P(Y = 1)) = mu + rowc_col_coef_rj

where the row cluster effects and individual column effects are absorbed into the matrix rowc_col_coef_rj. These models are the same mathematically, the only differences between them are in how they are constrained (see below in the section about the `init_parvec` argument) and how they should be interpreted.

Note that if you were using covariates, then it would not be equivalent to leave out the main effects and just use the interaction terms, but the clustering models don't work quite the same as regression models with covariates.

Equivalently, if you want to cluster the columns, you can include individual row effects alongside the column clusters, i.e.

`Y ~ COLCLUST + ROW` or `Y ~ COLCLUST + ROW + COLCLUST:ROW`,

depending on whether you want the interaction terms or not.

You are **not** able to include individual row effects with row clusters, or include individual column effects with column clusters, because there is not enough information in ordinal or binary data to fit these models. As a consequence, you cannot include individual row or column effects if you are doing biclustering, e.g.

`Y ~ ROWCLUST + COLCLUST + ROW` or `Y ~ ROwCLUST + COLCLUST + COL`

are not permitted.

From version 1 of the package, you can now also include **covariates** alongside the clustering patterns. The basic way to do this is include them as additions to the clustering model_structure. For example, including one row covariate `xr` to a row clustering model would have the formula

`Y ~ ROWCLUST + xr`

with Binary model Logit(P(Y = 1)) = mu + rowc_coef_r + row_coef_1*xr_i

where row_coef_1 is the coefficient of xr_i, just as in a typical regression model.

Additional row covariates can also be included, and you can include interactions between them, and functions of them, as in regression models, e.g.

`Y ~ ROWCLUST + xr1*log(xr2)`

which would have the Binary model

Logit(P(Y = 1)) = mu + rowc_coef_r + row_coef1*xr1_i + row_coef2*log(xr2_i) + row_coef3*xr1_i*log(xr2_i)

If instead you want to add column covariates to the model, they work in the same way after they've been added to the `long_df` data frame using `mat_to_df`, but they are indexed by j instead of i. Simplest model, with one single column covariate xc, would have formula

`Y ~ ROWCLUST + xc`

with Binary model Logit(P(Y = 1)) = mu + rowc_coef_r + col_coef1*xc_j

You can use any functions of or interactions between column covariates, just as with row covariates. You can similarly add row or column covariates to column clustering or biclustering models.

You can include **interactions between covariates** and `ROWCLUST` or `COLCLUST` in the formula. But these are **not quite** the same as interactions between covariates. The formula

`Y ~ ROWCLUST*xr`

where `xr` is some row covariate, corresponds to the Binary model

Logit(P(Y = 1)) = mu + rowc_coef_r + cov_coef*xr_i + rowc_row_coef_r1*xr_i

What this means is that there is a term in the linear predictor that involves the row covariate xr (which has the index i because it is a row covariate), and each cluster (indexed by r) has a different coefficient for that covariate (as distinct from the non-interaction covariate models above, which have the same coefficients for the covariates regardless of which cluster the row is in).

This is different from interaction terms involving only covariates, where two or more covariates appear multiplied together in the model and then have a shared coefficient term attached to them. In a clustering/covariate interaction, the row or column clustering pattern controls the coefficients rather than adding a different type of covariate.

Note that the pure cluster effect rowc_coef_r is also included in the model automatically, in the same way that a regression formula `Y ~ x1*x2` would include the individual x1 and x2 effects as well as the interaction between x1 and x2.

The coefficients for row clusters interacting with row coefficients are named `row_cluster_row_coef` in the output of `clustord` because you can also have coefficients for interactions between row clustering and column covariates, or column clustering and row covariates, or column clustering and column covariates. Row clustering interacting with column covariates would look something like

`Y ~ ROWCLUST*xc`

with Binary model Logit(P(Y = 1)) = mu + rowc_coef_r + rowc_col_coef_r1*xc_j

The other combinations of clustering and covariates work similarly. `rowc_col_coef_rl` and the other similar coefficients have two indices. Their first index is the index of the cluster, and their second index is the index of the covariate among the list of covariates interacting with that direction of clustering. So if there are two row covariates `xr1` and `xr2` interacting with three row clusters, that gives you 6 coefficients:

rowc_col_coef_11, rowc_col_coef_12, rowc_col_coef_21,

rowc_col_coef_22, rowc_col_coef_31, rowc_col_coef_32.

and you can also have a three-way interaction between row cluster and those two covariates, which would add the coefficients rowc_col_coef_r3 for the xr1:xr2 term.

You can instead add covariates that interact with column clusters, which will have parameters colc_row_coef_cm, where m here indexes just the covariates interacting with column cluster.

If you have covariates interacting with row clusters and other covariates interacting with column clusters, then you will have parameters rowc_cov_coef_rl **and** colc_cov_coef_cm.

An example of this is the model

Y ~ ROWCLUST + xr1 + ROWCLUST:xr1 + xc1 + COLCLUST + COLCLUST:log(xc1)

This has main effects for row clusters and column clusters, i.e. ROWCLUST and COLCLUST. It also has two covariate terms not interacting with clusters, xr1 and xc1. It also has 1 covariate term interacting with row clusters, xr1, with coefficients rowc_cov_coef_r1, and 1 covariate term interacting with column clusters, log(xc1), with coefficients colc_cov_coef_c1.

Restrictions on formula

The primary restriction on the formula argument is that that you **cannot** use functions of ROW, COL, ROWCLUST or COLCLUST, such as log(ROW) or COLCLUST^2. That is because they are not covariates, and cannot be manipulated like that; instead, they are indicators for particular elements of the clustering model_structure.

If performing biclustering, i.e. if ROWCLUST and COLCLUST are both in the model, and you want to include the interaction between them, then you can use the interaction between them on its own, or you can include both main effects, but you are not allowed to use just one main effect alongside the interaction. That is, you can use

Y ~ ROWCLUST + COLCLUST + ROWCLUST:COLCLUST or Y ~ ROWCLUST*COLCLUST,

or you can use Y ~ ROWCLUST:COLCLUST, and these two types of biclustering model will have different parameter constraints (see below under init_parvec details), but you **cannot** use

Y ~ ROWCLUST + ROWCLUST:COLCLUST or Y ~ COLCLUST + ROWCLUST:COLCLUST

As stated above, you also cannot include individual row effects alongside row clustering, and you cannot use individual column effects alongside column clustering, i.e. if ROWCLUST is in the formula, then ROW **cannnot** be in the formula, and if COLCLUST is in the formula then COL **cannot** be in the formula.

If you are including COL with ROWCLUST, then you can include the interaction between them but that is the **only** permitted interaction term that involves COL, and similarly the interaction between ROW and COLCLUST is the **only** permitted interaction term that involves ROW. But you can include those interactions in the form

Y ~ ROWCLUST + COL + ROWCLUST:COL or as Y ~ ROWCLUST*COL, or as Y ~ ROWCLUST:COL.

These are the only permitted uses of the COL term, and there are equivalent constraints on the inclusion of ROW.

As stated above, you can include interactions between ROWCLUST or COLCLUST and covariates, but you **cannot** include three-way interactions between ROWCLUST, COLCLUST and one or more covariates are **not permitted** in clustord, mostly because of the prohibitive number of parameter values that would need to be fitted, and the difficulty of interpreting such a model. That is, you cannot use

formulae such as `Y ~ ROWCLUST*COLCLUST*xr`, which would have Binary model Logit(P(Y = 1)) = mu + bi_cluster_row_coef_rc1*xr_i.

`model` **argument details**

The three models available in `clustord` are the Binary model, which is a Bernoulli model equivalent to the binary model in the package `clustglm`, the Proportional Odds Model (POM) and the Ordered Stereotype Model (OSM).

Many Binary model examples have been given above, which have the general form

logit(P(Y = 1)) = mu + «linear terms»

where the linear terms can include row or column clustering effects, individual row or column effects, and row or column covariates, with or without interactions with row or column clustering.

The Proportional Odds Model and the Ordered Stereotype Model have the same model_structure for the linear terms, but the overall model equation is different.

The Proportional Odds Model (`model = "POM"`) has the form

logit(P(Y <= k)) = log(P(Y <= k)/P(Y > k)) = mu_k - «linear terms»

So the simplest POM for row clustering would be

logit(P(Y <= k)) = mu_k - rowc_coef_r

and the model including individual column effects and no interactions would be

logit(P(Y <= k)) = mu_k - rowc_coef_r - col_coef_j

Note that the linear-term coefficients have negative signs for the Proportional Odds Models. This is so that as the row cluster index increases, or as the column index increases, Y is more likely to fall at higher values (see Ch4 of Agresti, 2010).

The Ordered Stereotype model (`model = "OSM"`) has the form

log(P(Y = k)/P(Y = 1)) = mu_k + phi_k(«linear terms»)

So the simplest OSM for row clustering would be

log(P(Y = k)/P(Y = 1)) = mu_k + phi_k*rowc_coef_r

and the model including individual column effects and no interactions would be

log(P(Y = k)/P(Y = 1)) = mu_k + phi_k(rowc_coef_r + col_coef_j)

Note that the OSM is **not** a cumulative logit model, unlike the POM. The model describes the log of the kth level relative to the first level, which is the baseline category, but the patterns for k = 2 may be different than the patterns for k = 3. They are linked, because the linear terms will be the same, but they may not have the same shape. In this sense, the OSM is more flexible/less restrictive than the POM.

See Anderson (1984) for the original definition of the ordered stereotype model, and see Fernández et al. (2016) for the application to clustering.

The phi_k parameters may be treated as "score" parameters. After fitting the OSM, the fitted phi_k values can give some indication of what the true separation is between the categories. Even if the default labelling of the categories is from 1 to n, that doesn't mean that the categories are actually equally spaced in reality. But the fitted phi_k values from the OSM can be treated as data-derived numerical labels for the categories. Moreover, if two categories have very similar fitted phi_k values, e.g. if phi_2 = 0.11 and phi_3 = 0.13, that suggests that there is not enough information in the data to distinguish between categories 2 and 3, and so you might as well merge them into a single category to simplify the model-fitting process and the interpretation of the results.

`init_parvec` **argument details**

Initvect is the vector of starting values for the parameters, made up of sections for each different type of parameter in the model. Note that the length of each section of init_parvec is the number of **independent** parameter values, not the overall number of parameter values of that type.

If you want to supply a vector of starting values for the EM algorithm, you need to be careful how many values you supply, and the order in which you include them in `init_parvec`, and you should **CHECK** the output list of parameters (which is the full set of parameter values, including dependent ones, broken up into each type of parameter) to check that your init_parvec model_structure is correct for the formula you have specified.

For example, the number of `mu` values will always be 1 fewer than the number of categories in the data, and the remaining value of mu is dependent on those q-1 values. In the OSM for data with 3 categories, the first value of mu for category 1 will be 0, and then the other 2 values of mu for categories 2 and 3 will be the independent values of mu. For the POM for data with 5 categories, the first 4 values of mu will be the independent values and then the last value of mu is infinity, because the probability of Y being in category 5 is defined as 1 minus the sum of the probabilities for the other 4 levels.

`q` is the number of levels in the values of y, `n` is the number of rows in the original data matrix, and `p` is the number of columns in the original data matrix.

For Binary,

There is one independent value for `mu`, i.e. `q = 2`.

Ignore `phi`, which is not used in the Binary model.

For OSM,

The starting values for `mu_k` are length `q-1`, and the model has `mu_1 = 0` always, so the init_parvec values for `mu` will become `mu_2`, `mu_3`, etc. up to `mu_q`.

The starting values for `phi_k` are length `q-2`.

Note that the starting values for `phi` do not correspond directly to `phi`, because `phi` is restricted to being increasing and between 0 and 1, so instead the starting values are treated as elements `u[2:q-1]` of a vector `u` which can be between `-Inf` and `+Inf`, and then

`phi[2] <- expit(u[2])` and

`phi[k] <- expit(u[2] + sum(exp(u[3:k])))` for k between 3 and q-1

`(phi[1] = 0 and phi[q] = 1)`.

For POM,

The starting values for `mu_k` are length `q-1`, but the starting values do not correspond directly to `mu_k`, because `mu_k` is restricted to being increasing, i.e. the model has to have `mu_1 <= mu_2 <= ... mu_q = +Inf`

So instead of using the init_parvec values directly for `mu_k`, the 2nd to (q-1)th elements of init_parvec are used to construct `mu_k` as follows:

```
mu_1 <- init_parvec[1]
```

```
mu_2 <- init_parvec[1] + exp(init_parvec[2])
```

```
mu_3 <- init_parvec[1] + exp(init_parvec[2]) + exp(init_parvec[3])
```

... and so on up to `mu_{k-1}`, and `mu_k` is infinity, because it is not used directly to construct the probability of Y = q.

Thus the values that are used to construct mu_k can be unconstrained, which makes it easier to specify init_parvec and easier to optimize the parameter values.

Ignore phi, which is not used in POM.

For **all three models**,

The starting values for rowc_coef_r are length RG-1, where RG is the number of row clusters. The final row cluster parameter is dependent on the others (see the input parameter info for constraint_sum_zero), whereas if it were independent it would be colinear with the mu_k parameters and thus not identifiable.

Similarly the starting values for colc_coef_c are length CG-1, where CG is the number of column clusters, to avoid problems of colinearity and nonidentifiability.

If you have biclustering with an interaction term between row clusters and column clusters, then the number of independent values in the matrix of interaction terms depends on whether you include the main effects of row and column clusters separately. That is, if you use the biclustering model

Y ~ ROWCLUST + COLCLUST + ROWCLUST:COLCLUST, or equivalently

Y ~ ROWCLUST*COLCLUST,

then the main effect term ROWCLUST has RG-1 independent parameters in init_parvec, and COLCLUST has CG-1 independent parameters in init_parvec, and ROWCLUST:COLCLUST will have (RG - 1)*(CG - 1) independent parameter values. The final matrix of interaction terms will be constrained to have its last row equal to the negative sum of the other rows, and the last column equal to the negative sum of the other columns.

On the other hand, if you want to use only the interaction term and not the main effects (which for the clustering model is mathematically equivalent), i.e.

Y ~ ROWCLUST:COLCLUST,

then that matrix of interaction terms will have RG*CG - 1 independent parameters, i.e. more independent parameters than if you included the main effects.

If you have column effects alongside row clusters (they are not permitted alongside column clusters), without interactions, i.e. the formula Y ~ ROWCLUST + COL with Binary model Logit(P(Y = 1)) = mu + rowc_coef_r + col_coef_j then the row cluster coefficients have RG - 1 independent parameters, and the column effect coefficients have p - 1 independent parameters, where p is the number of columns in the original data matrix, i.e. the maximum value of long_df$COL.

If you include the interaction term, then the number of independent parameters again depends on whether you just use the interaction term, or include the main effects.

In the formula Y ~ ROWCLUST + COL + ROWCLUST:COL or its equivalent with "*", the interaction term will have (RG - 1)*(p-1) independent parameters.

If you instead use the formula Y ~ ROWCLUST:COL, then the interaction term will have RG*p - 1 independent parameters. Either way, the total number of independent parameters in the model will be RG*p.

Similarly, if you have row effects alongside column clusters, without interactions, i.e. the formula

Y ~ COLCLUST + ROW,

with Binary model Logit(P(Y = 1)) = mu + colc_coef_c + row_coef_i

then the column cluster coefficients will have CG - 1 independent parameters, and the row coefficients will have n-1 independent parameters, where n is the number of rows in the original data matrix, i.e. the maximum value of long_df$ROW.

If you include the interaction term alongside the main effects, i.e.

`Y ~ COLCLUST + ROW + COLCLUST:ROW`, or its equivalent with "*", the interaction term will have `(CG - 1)*(n-1)` independent parameters.

If you instead use the formula `Y ~ COLCLUST:ROW`, that interaction coefficient matrix will have `CG*n - 1` independent parameters.

Any covariate terms included in the formula will be split up by `clustord` into the covariates that interact with row clusters, the covariates that interact with column clusters, and the covariates that do not interact with row or column clusters.

The number of independent parameters for row-cluster-interacting covariates will be `RG*L`, where `L` is the number of terms involving row clusters and covariates after any "*" terms have been expanded.

So in this formula, for example,

`Y ~ ROWCLUST*xr1 + xr2 + ROWCLUST:log(xc1)`

where xr1 and xr2 are row covariates, and xc1 is a column covariate, the fully expanded formula would be

`Y ~ ROWCLUST + xr1 + xr2 + ROWCLUST:xr1 + ROWCLUST:log(xc1)`

and the terms interacting with `ROWCLUST` would be `ROWCLUST:xr1` and `ROWCLUST:log(xc1)`, so there would be `RG*2` independent coefficients for those covariates.

The number of independent parameters for column-cluster-interacting covariates will be `CG*M`, where `M` is the number of terms involving column clusters and covariates after any "*" terms have been expanded.

So this formula, for example,

`Y ~ I(xr1^2) + COLCLUST*xc1 + COLCLUST:xc2:xc3 + COLCLUST*xr1`

would be expanded as

`Y ~ COLCLUST + xr1 + I(xr1^2) + xc1 + COLCLUST:xc1 + COLCLUST:xc2:xc3 + COLCLUST:xr1`

and the terms interacting with `COLCLUST` would be `COLCLUST:xc1`, `COLCLUST:xc2:xc3` and `COLCLUST:xr1`, so there would be `CG*3` independent coefficients for those covariates.

The number of independent parameters for covariates that do not interact with row or column clusters will be the same as the number of those covariate terms, after any "*" terms have been expanded.

So this formula, for example,

`Y ~ ROWCLUST*xr1 + xr2 + ROWCLUST:log(xc1) + COLCLUST*xc1`

would be expanded as

`Y ~ ROWCLUST +COLCLUST + xr1 + xr2 + xc1 + ROWCLUST:xr1 + ROWCLUST:log(xc1) + COLCLUST:xc1`,

so there would be 3 independent coefficients for the terms `xr1, xr2, xc1`.

Note that there are **no intercept** terms for the coefficients, because those are incorporated into the parameters `mu_k`.

The **order of the** `init_parvec` **entries** is as follows, and any entries that are not included in the formula will be ignored and not included in `init_parvec`. That is, you should NOT provide values in `init_parvec` for components that are not included in the formula.

1) mu (or values used to construct mu, POM only)

2) values used to construct phi (OSM only)

3) row cluster coefficients

4) column cluster coefficients

5) [matrix] bicluster coefficients (i.e. interaction between row and column clusters)

6) individual row coefficients

7) individual column coefficients

8) [matrix] interactions between row clusters and individual column coefficients

9) [matrix] interactions between column clusters and individual row coefficients

10) [matrix] row-cluster-specific coefficients for covariates interacting with row clusters

11) [matrix] column-cluster-specific coefficients for covariates interacting with column clusters

12) coefficients for covariates that do not interact with row or column clusters

Any entries marked as [matrix] will be constructed into matrices by filling those matrices row-wise, e.g. if you want starting values 1:6 for a matrix of 2 row clusters and 3 covariates interacting with those row clusters, the matrix of coefficients will become

`1 2 3`

`4 5 6`

For the formula `Y ~ ROWCLUST*COLCLUST`, where the matrix of interactions between row and column clusters has `(RG - 1)*(CG - 1)` independent parameters, the last row and column of the matrix will be the negative sums of the rest, so e.g. if you have 2 row clusters and 3 column clusters, there will only be 2 independent values, so if you provide the starting values -0.5 and 1.2, the final matrix of parameters will be:

`column cluster 1 column cluster 2 column cluster 3`

`row cluster 1 -0.5 1.2 -0.7`

`row cluster 2 0.5 -1.2 0.7`

If the matrix is a matrix relating to row clusters, then the row clusters are in the rows, and if it's a matrix relating to column clusters but not row clusters, then the column clusters are in the rows, i.e. the matrix of coefficients for column clusters interacting with individual row effects will have the rows of the matrix corresponding to the clusters, i.e. the matrix would be indexed colc_row_coef_ci, c being the column cluster index and i being the row index.

Similarly, if the matrix is a matrix relating to column clusters and covariates, then the rows of the matrix will correspond to the column clusters, i.e. the matrix would be indexed colc_cov_coef_cl, c being the column cluster index and l being the covariate index.

If using biclustering with interaction between row and column clusters, then the row clusters will be the rows and the column clusters will be the columns, i.e. the matrix would be indexed rowc_colc_coef_rc, r being the row cluster index and c being the column cluster index.

Value

A `clustord` object, i.e. a list with components:

`info`: Basic info n, p, q, the number of parameters, the number of row clusters and the number of column clusters, where relevant.

`model`: The model used for fitting, "OSM" for Ordered Stereotype Model, "POM" for Proportional Odds Model, or "Binary" for Binary model.

EMstatus: a list containing the latest iteration iter, latest incomplete-data and complete-data log-likelihoods new_lli and new_llc, the best incomplete-data log-likelihood best_lli and the corresponding complete-data log-likelihood, llc_for_best_lli, and the parameters for the best incomplete-data log-likelihood, params_for_best_lli, indicator of whether the algorithm converged converged, and if the user chose to keep all parameter values from every iteration, also params_every_iteration.

Note that for **biclustering**, i.e. when ROWCLUST and COLCLUST are both included in the model, the **incomplete** log-likelihood is calculated using the entropy approximation, and this may be **inaccurate** unless the algorithm has converged or is close to converging. So beware of using the incomplete log-likelihood and the corresponding AIC value **unless the EM algorithm has converged**.

Also note that out_parvec unpacks the dependent elements of matrices of effects, such as the row cluster and column cluster interaction effects rowc_colc and other interaction effects, by row: the first elements of the matrix in out_parvec would be the first *row* of the matrix, etc. By contrast, EMstatus$params_every_iteration simply unlists out_parlist, which happens automatically by column, so the first column of each matrix before the second column, etc. so if it contains columns rowc_colc1, rowc_colc2, rowc_colc3 etc. they are the first *column* of the rowc_colc matrix of parameters. So if using EMstatus$params_every_iteration to check specific parameter values during each iteration of the algorithm, be careful which ones you're looking at.

criteria: the calculated values of AIC, BIC, etc. from the best incomplete-data log-likelihood.

epsilon: the very small value (default 1e-6) used to adjust values of pi and kappa and theta that are too close to zero, so that taking logs of them does not produce infinite values. Use the control_EM argument to adjust epsilon.

constraints_sum_zero: the chosen value of constraints_sum_zero.

param_lengths: vector of total number of parameters/coefficients for each part of the model, labelled with the names of the components. The value is 0 for each component that is not included in the model, e.g. if there are no covariates interacting with row clusters then the rowc_cov_coef value will be 0. If the component is included, then the value given will include any dependent parameter/coefficient values, so if column clusters are included then the colc_coef value will be CG, whereas the number of independent values will be CG - 1.

init_parvec: the initial *vector* of parameter values, either specified by the user or generated automatically. This vector has only the **independent** values of the parameters, not the full set.

out_parvec: the final *vector* of parameter values, containing only the independent parameter values from out_parlist.

init_parlist: the initial list of parameters, constructed from the initial parameter vector init_parvec. Note that if the initial vector has been incorrectly specified, the values of init_parlist may not be as expected, and they should be checked by the user.

out_parlist: fitted values of parameters.

pi, kappa: fitted values of pi and kappa, where relevant.

row_cluster_probs, column_cluster_probs: the posterior probabilities of membership of the row clusters and the column clusters, where relevant.

rowc_mm, colc_mm, cov_mm: the model matrices for, respectively, the covariates interacting with row clusters, the covariates interacting with column clusters, and the covariates not interacting with row or column clusters (i.e. the covariates with constant coefficients). Note that one row of each model matrix corresponds to one row of long_df.

row_clusters, column_clusters: the assigned row and column clusters, where relevant, where each row/column is assigned to a cluster based on maximum posterior probability of cluster membership (row_cluster_probs and column_cluster_probs).

row_cluster_members, column_cluster_members: vectors of assigned members of each row or column cluster, where each row/column is assigned to a cluster based on maximum posterior probability of cluster membership (row_cluster_probs and column_cluster_probs)

reordered: Boolean indicating whether or not the outputs have been reordered in order of row cluster and/or column cluster effects. FALSE for direct outputs of clustord(), will be changed to TRUE after running reorder() on clustord output objects.

References

Fernandez, D., Arnold, R., & Pledger, S. (2016). Mixture-based clustering for the ordered stereotype model. *Computational Statistics & Data Analysis*, 93, 46-75.

Anderson, J. A. (1984). Regression and ordered categorical variables. *Journal of the Royal Statistical Society: Series B (Methodological)*, 46(1), 1-22.

Agresti, A. (2010). *Analysis of ordinal categorical data* (Vol. 656). John Wiley & Sons.

Examples

```
set.seed(1)
long_df <- data.frame(Y=factor(sample(1:3,5*20,replace=TRUE)),
               ROW=factor(rep(1:20,times=5)),COL=rep(1:5,each=20))

# Model Log(P(Y=k)/P(Y=1))=mu_k+phi_k*rowc_coef_r with 3 row clustering groups:
clustord(Y~ROWCLUST,model="OSM",3,long_df=long_df,
             control_EM=list(maxiter=2,maxiter_start=2), nstarts=2)

# Model Log(P(Y=k)/P(Y=1))=mu_k+phi_k*(rowc_coef_r + col_coef_j)
#    with 3 row clustering groups:
clustord(Y~ROWCLUST+COL,model="OSM",3,long_df=long_df,
             control_EM=list(maxiter=2,maxiter_start=2), nstarts=2)

# Model Logit(P(Y <= k))=mu_k-rowc_coef_r-col_coef_j-rowc_col_coef_rj
#    with 2 row clustering groups:
clustord(Y~ROWCLUST*COL,model="POM",RG=2,long_df=long_df,
             control_EM=list(maxiter=2,maxiter_start=2), nstarts=2)

# Model Log(P(Y=k)/P(Y=1))=mu_k+phi_k*(colc_coef_c)
#    with 3 column clustering groups:
clustord(Y~COLCLUST,model="OSM",CG=3,long_df=long_df,
             control_EM=list(maxiter=2,maxiter_start=2), nstarts=2)

# Model Log(P(Y=k)/P(Y=1))=mu_k+phi_k*(colc_coef_c + row_coef_i)
#    with 3 column clustering groups:
clustord(Y~COLCLUST+ROW,model="OSM",CG=3,long_df=long_df,
             control_EM=list(maxiter=2,maxiter_start=2), nstarts=2)

# Model Log(P(Y=k)/P(Y=1))=mu_k+phi_k*(rowc_coef_r + colc_coef_c)
#    with 3 row clustering groups and 2 column clustering groups:
```

```
clustord(Y~ROWCLUST+COLCLUST,model="OSM",RG=3,CG=2,long_df=long_df,
             control_EM=list(maxiter=2), nstarts=1)

# Model Logit(P(Y<=k))=mu_k-rowc_coef_r-colc_coef_c-rowc_colc_coef_rc
#    with 2 row clustering groups and 4 column clustering groups, and
#    interactions between them:
clustord(Y~ROWCLUST*COLCLUST, model="POM", RG=2, CG=4,
             long_df=long_df,control_EM=list(maxiter=2), nstarts=1,
             start_from_simple_model=FALSE)
```

`mat_to_df` *Converting matrix of responses into a long-form data frame and incorporating covariates, if supplied.*

Description

Converting matrix of responses into a long-form data frame and incorporating covariates, if supplied.

Usage

```
mat_to_df(mat, xr_df = NULL, xc_df = NULL)
```

Arguments

`mat`	matrix of responses to be clustered
`xr_df`	optional data frame of covariates corresponding to the rows of `mat`. Each row of `xr_df` corresponds to one row of `mat`, and each column of `xr_df` is a covariate.
`xc_df`	optional data frame of covariates corresponding to the columns of `mat`. Each row of `xc_df` corresponds to one **column** of `mat`, and each column of `xc_df` is a covariate.

Value

A data frame with columns `Y`, `ROW` and `COL`, and additional columns for covariates from `xr_df` and `xc_df`, if included.

The `Y` column of the output contains the entries in `mat`, with one row in the output per one cell in `mat`, and the `ROW` and `COL` entries indicate the row and column of the data matrix that correspond to the given cell. Any cells that were NA are left out of the output data frame.

If `xr_df` is supplied, then there are additional columns in the output corresponding to the columns of `xr_df`, and the values for each covariate are repeated for every entry that was in the corresponding row of the data matrix.

Similarly, if `xc_df` is supplied, there are additional columns in the output corresponding to the columns of `xc_df`, and the values for each covariate are repeated for every entry that was in the corresponding column of the data matrix.

osm *Ordinal data regression using the Ordered Stereotype Model (OSM).*

Description

Fit a regression model to an ordered factor response. The model is NOT a logistic or probit model because the link function is not the logit, but the link function is log-based.

Usage

```
osm(
  formula,
  data,
  weights,
  start,
  ...,
  subset,
  na.action,
  Hess = FALSE,
  model = TRUE
)
```

Arguments

formula	a formula expression as for regression models, of the form response ~ predictors. The response should be a factor (preferably an ordered factor), which will be interpreted as an ordinal response, with levels ordered as in the factor. The model must have an intercept: attempts to remove one will lead to a warning and be ignored. An offset may be used. See the documentation of formula for other details.
data	a data frame, list or environment in which to interpret the variables occurring in formula.
weights	optional case weights in fitting. Default to 1.
start	initial values for the parameters. See the Details section for information about this argument.
...	additional arguments to be passed to optim, most often a control argument.
subset	expression saying which subset of the rows of the data should be used in the fit. All observations are included by default.
na.action	a function to filter missing data.
Hess	logical for whether the Hessian (the observed information matrix) should be returned.
model	logical for whether the model matrix should be returned.

Details

This function should be used in a very similar way to `MASS::polr`, and some of the arguments are the same as `polr`, but the ordinal model used here is less restrictive in its assumptions than the proportional odds model. However, it is still parsimonious i.e. it uses only a small number of additional parameters compared with the proportional odds model.

This model is the *ordered stereotype* model (Anderson 1984, Agresti 2010)

It is more flexible than the proportional odds model but only adds a handful of additional parameters. It is not a cumulative model, being instead defined in terms of the relationships between each of the higher categories and the lowest category that is treated as the reference category.

Each of the higher categories has its own intercept term, mu_k, which is similar to the zeta parameters in `polr`, but in the OSM each higher category also has its own scaling parameter, phi_k, which adjusts the effect of the covariates on the response. This allows the effect of the covariates on the response to be slightly different for each category of the response, thus making the model more flexible than the proportional odds model.

The final set of parameters are coefficients for each of the covariates, and these are equivalent to the coefs in `polr`. Higher or more positive values of the coefficients increases the probability of the response being in the higher categories, and lower or more negative values of the coefficients increase the probability of the response being in the lower categories.

The overall model takes the following form:

log(P(Y = k | X)/P(Y = 1 | X)) = mu_k + phi_k*beta_vec^T x_vec

for k = 2, ..., q, where x_vec is the vector of covariates for the observation Y.

mu_1 is fixed at 0 for identifiability of the model, and the phi_k parameters are constrained to be ordered (giving the model its name) in the following way:

0 = phi_1 <= phi_2 <= ... <= phi_k <= ... <= phi_q = 1.

(The unordered stereotype model restricts phi_1 and phi_q but allows the remaining phi_k to be in any order, and this is suitable for fitting the model for nominal data. However, this package does not provide that option, as it is already available in other packages which can fit the stereotype model.)

After fitting the model, the estimated values of the intermediate phi_k values indicate a suitable numerical spacing of the ordinal response categories that is based on the data. The spacings indicate how much distinct information each of the corresponding levels provide. For example, if you have five response categories and the fitted phi values are `(0, 0.04,0.6, 0.62, 1)` then this indicates that levels 1 and 2 provide very similar information about the effect of the covariates on the response, and levels 3 and 4 provide very similar information as each other. The meaning of this is that you could simplify the response by combining levels 1 and 2 and combining levels 3 and 4 (i.e. reduce the levels to 1, 3 and 5) and you would still be able to estimate the beta coefficients with similar accuracy.

Another use for the phi_k values is that if you want to carry out further analysis of the response, treating it as a numerical variable, then the phi values are a better choice of numerical values for the response categories than the default values 1 to q.

`start` argument values: `start` is a vector of start values for estimating the model parameters.

The first part of the `start` vector is starting values for the coefficients of the covariates, the second part is starting values for the mu values (per-category intercepts), and the third part is starting values for the raw parameters used to construct the phi values.

The length of the vector is [number of covariate terms] + [number of categories in response variable - 1] + [number of categories in response variable - 2]. Every one of the values can take any real value.

The second part is the starting values for the mu_k per-category intercept parameters, and since mu_1 is fixed at 0 for identifiability, the number of non-fixed mu_k parameters is one fewer than the number of categories.

The third part of the starting vector is a re-parametrization used to construct starting values for the estimated phi parameters such that the phi parameters observe the ordering restriction of the ordered stereotype model, but the raw parameters are not restricted which makes it easier to optimise over them. phi_1 is always 0 and phi_q is always 1 (where q is the number of response categories). If the raw parameters are u_2 up to u_(q-1), then phi_2 is constructed as expit(u_2), phi_3 is expit(u_2 + exp(u_3)), phi_4 is expit(exp(u_3) + exp(u_4)) etc. which ensures that the phi_k values are non-decreasing.

This code was adapted from file MASS/R/polr.R copyright (C) 1994-2013 W. N. Venables and B. D. Ripley Use of transformed intercepts contributed by David Firth The osm and osm.fit functions were written by Louise McMillan, 2020.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 or 3 of the License (at your option).

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

A copy of the GNU General Public License is available at http://www.r-project.org/Licenses/

Value

An object of class `"osm"`. This has components

`beta` the coefficients of the covariates, with NO intercept.

`mu` the intercepts for the categories.

`phi` the score parameters for the categories (restricted to be ordered).

`deviance` the residual deviance.

`fitted.values` a matrix of fitted values, with a column for each level of the response.

`lev` the names of the response levels.

`terms` the `terms` structure describing the model.

`df.residual` the number of residual degrees of freedom, calculated using the weights.

`edf` the (effective) number of degrees of freedom used by the model

`n, nobs` the (effective) number of observations, calculated using the weights.

`call` the matched call.

`convergence` the convergence code returned by `optim`.

`niter` the number of function and gradient evaluations used by `optim`.

`eta`

`Hessian` (if `Hess` is true). Note that this is a numerical approximation derived from the optimization proces.

`model` (if `model` is true), the model used in the fitting.

`na.action` the NA function used

`xlevels` factor levels from any categorical predictors

References

Agresti, A. (2010). *Analysis of ordinal categorical data* (Vol. 656). John Wiley & Sons.

Anderson, J. A. (1984). Regression and ordered categorical variables. *Journal of the Royal Statistical Society: Series B (Methodological)*, 46(1), 1-22.

See Also

[MASS::polr()]

`reorder.clustord` *Reorder row or column clusters in order of increasing (or decreasing) cluster effects.*

Description

The label-switching problem in model-based clustering is that results with the clusters are in a different order are mathematically equivalent to each other, and the EM algorithm does not distinguish between them. For example, two row clustering results with different starting points on the *same* data may assign all of the observations to the same clusters both times, but the group of observations labelled as cluster 1 in the first result is labelled as cluster 3 in the second result. Similarly, for column clustering a group of variables can be labelled cluster 4 in the first result and cluster 1 in the second result.

Usage

```
## S3 method for class 'clustord'
reorder(x, type, decreasing = FALSE, ...)
```

Arguments

`x`	Object of class `clustord`, the output from a `clustord` run.
`type`	Whether to reorder the row cluster effects (`"row"`), column cluster effects (`"column"`), or both (`"both"`).
`decreasing`	(default FALSE) One or two element vector, indicating which direction to sort in. If one element, then all clusters being reordered will be reordered in the same direction. Default is increasing (i.e. `decreasing = FALSE`, as for the base function `sort()`). If two element vector is used, which is only permissible when `type = "both"`, the first direction will be used for the **row clusters** and the second direction will be used for the **column clusters**.
`...`	optional: extra arguments.

Details

It is often useful to reorder the clusters to show them in order of cluster effect size, because this makes any display of the features of those clusters a bit easier for people to read.

Moreover, if you perform multiple replicate runs of `clustord` with the same settings and want to be able to summarise the results, e.g. by providing the mean estimated parameter values, then you will need to reorder the cluster results so that in all of the replicate runs the first cluster is the one with the most negative cluster effect, etc.

Note that if you order the cluster effects in increasing order, the first one will **not** necessarily be the *smallest*. If using the default constraint that the cluster effects must sum to zero, the first cluster effect in increasing order will be the **most negative** and the last will be the **most positive**.

If you use the argument `constraint_sum_zero = FALSE`, which uses the first-element-is-zero constraint for cluster effects, and you sort the clusters in increasing order (i.e. with default `decreasing = FALSE`, then after reordering the clusters in increasing order the first one will be 0 and the second one will be the smallest non-zero effect. However, if you use the argument `constraint_sum_zero = FALSE` and sort with `decreasing = TRUE`, then the first element **will still be zero** because the model is fitted with that first element always set to zero, so it is special and reordering will not stop it being the first element.

Note that this function CANNOT be used if you have used interaction terms without the main cluster effects e.g. if you included `ROWCLUST:x1` in the formula for clustering but did not include `ROWCLUST` as another term (and similarly for `COLCLUST`).

Value

An object of class `clustord`, with all the relevant elements reordered in order of cluster effects. See clustord for more info about the contents of `clustord` objects. The `clustord` object will gain an extra field, `reordered = TRUE`. Elements of `clustord` object that may be reordered (which ones are reordered depends on whether row clusters are being reordered and whether column clusters are being reordered: - `out_parlist` (the final list of estimated parameter values) - `row_cluster_proportions` and/or `column_cluster_proportions` - `row_cluster_probs` and/or `column_cluster_probs` - `out_parvec` - `row_cluster_members` and `row_clusters` and/or `column_cluster_members` and `column_clusters` - `EMstatus$params_for_best_lli` - `EMstatus$params_every_iteration`, if using option `control_EM$keep_all_params` - `start.par`

.

Examples

```
set.seed(1)
long_df <- data.frame(Y=factor(sample(1:3,5*20,replace=TRUE)),
               ROW=rep(1:10,times=10),COL=rep(1:10,each=10))
results_original <- clustord(Y ~ ROWCLUST + COLCLUST, model="OSM",
                             RG=3, CG=2, long_df=long_df,
                             control_EM=list(maxiter=2))
results_original$out_parlist
# $mu
# mu_1      mu_2      mu_3
# 0.0000000 0.2053150 0.4107883
#
# $phi
```

```
# phi_1     phi_2     phi_3
# 0.0000000 0.6915777 1.0000000
#
# $rowc
# rowc_1      rowc_2      rowc_3
# 0.07756500  0.09247161 -0.17003661
#
# $colc
# colc_1      colc_2
# 0.07130783 -0.07130783

## Run reorder type "row" to reorder based on row cluster effects,
## in increasing order by default
results.reorder <- reorder(results_original, type="row")
results.reorder$out_parlist

## Run reorder type "column" to reorder based on column cluster effects,
## in decreasing order
results.reorder <- reorder(results_original, type="column", decreasing=TRUE)
results.reorder$out_parlist

## Run reorder type "row" to reorder based on row and column cluster effects,
## with row effects in increasing order and column effects in decreasing
## order
results.reorder <- reorder(results_original, type="both", decreasing=c(FALSE,TRUE))
results.reorder$out_parlist
```

`rerun` *Rerun clustord using the results of a previous run as the starting point.*

Description

This function is designed for two purposes. (1) You tried to run clustord and the results did not converge. You can supply this function with the previous results and the previous data object, and it will carry on running clustord from the endpoint of the previous run, which is quicker than starting the run again from scratch with more iterations.

Usage

```
rerun(
  results_original,
  long_df,
  control_EM = NULL,
  verbose = FALSE,
  control_optim = NULL
)
```

Arguments

results_original	The results of the previous run that you want to use as a starting point. The model, number of clusters, and final parameter values will be used, and the cluster controls such as maxiter will be reused unless the user specifies new values. But the row cluster and/or column cluster memberships will NOT be reused, and nor will the dataset, so you can change the dataset slightly and the rest of the details will be applied to this new dataset.
long_df	The dataset to use for this run, which may be slightly different to the original. Please note that the only compatibility check performed is comparing the sizes of the original and new datasets, and it is up to the user to check that the new dataset is sufficiently similar to the old one.
control_EM	Options to use for this run such as maxiter (number of EM iterations). Note that "maxiter_start" will not be relevant as this run will not generate random starts, it will run from the end parameters of the other run. See clustord documentation for more info.
verbose	(default FALSE) changes how much is reported to the console during the algorithm's progress. See clustord documentation for more info.
control_optim	Options to use for this run within optim(), which is used to estimate the parameters during each M-step. See clustord documentation for more info.

Details

(2) The previous result converged, but you have changed the dataset slightly, and want to rerun from the previous endpoint to save time.

Either way, you call the function in the same way, supplying the previous results object and a dataset, and optionally a new number of iterations ('control_EM=list(maxiter=XXX)', where 'XXX' is the new number of iterations.)

The output parameters of the old result will be used as the new initial parameters.

Value

An object of class clustord. See clustord for more info.

Examples

```
set.seed(1)
long_df <- data.frame(Y=factor(sample(1:3,5*20,replace=TRUE)),
               ROW=rep(1:20,times=5),COL=rep(1:5,each=20))
results_original <- clustord(Y ~ ROWCLUST, model="OSM", RG=4,
                             long_df=long_df, control_EM=list(maxiter=2))
results_original$EMstatus$converged
# FALSE

## Since original run did not converge, rerun from that finishing point and
## allow more iterations this time
results_new <- rerun(results_original, long_df, control_EM=list(maxiter=10))
```

```
## Alternatively, if dataset has changed slightly then rerun from the
## previous finishing point to give the new results a helping hand
long_df.new <- long_df[-c(4,25,140),]
results_new <- rerun(results_original, long_df.new)
```

Index

calc_cluster_comparisons, 2
calc_SE_bicluster *(*calc_SE_rowcluster*)*, 3
calc_SE_rowcluster, 3
clustord, *2, 3*, 4, *25*, *27*

mat_to_df, *5*, *8*, *11*, 20

optim, *3*
osm, 21

reorder.clustord, 24
rerun, 26