gllvmprime: fit a GLLVM model with mixed responses

Guillaume Blanc

2023-01-30

This document showcases how to fit a GLLVM model to a toy dataset using the package `gllvmprime`. This dataset has a matrix of responses `Y` with $n = 1000$ observations on $p = 20$ responses, the first 10 of which are (conditional) Gaussian and the last 10 are (conditional) Bernoulli. What's more, it has some information on one covariate in the matrix `X`, which includes an intercept as a column of 1.

Fitting the dataset consists in calling the `gllvmprime` function. The necessary parameters to give are:

- `Y`: a matrix of responses,
- `q`: the number of latent variables, i.e. the dimensions of the latent space (defaults to 2),
- `family`: the type of responses given by the names of the (conditional) distribution of the responses.

The `family` argument admits either a string if the responses are all of the same type, or, more generally, a p-dimensional vector of strings. The types of conditional distribution currently implemented are `poisson`, `bernoulli`, and `gaussian`.

Other noteworthy arguments of the `gllvmprime` function are:

- `X`, a matrix of covariate,
- `intercept`: a boolean variable which, if set to `intercept=TRUE`, will concatenate a column of ones to the matrix of covariates `X` (and create it if it does not exist), except if `X` already contains an intercept column, in which case the `intercept` argument is ignored,
- `maxit`: the maximum number of iterations,
- `alpha`: the learning rate multiplier, which defaults to `alpha=1` , and which controls the variance of the iterations.

Fit a GLLVM model with mixed responses

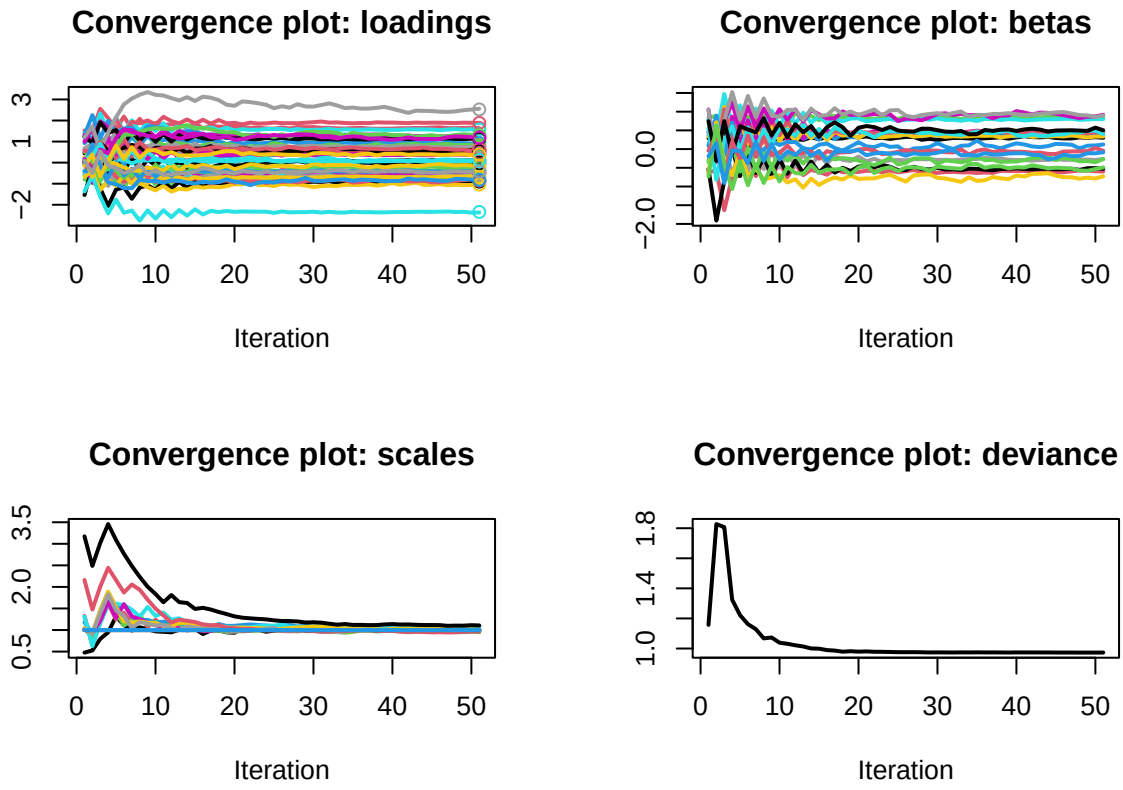
Our toy dataset is first loaded as a list whose elements are the covariates `X` (which includes an intercept) and the responses `Y`. To specify the conditional distribution of the responses, we create a character vector of size p consisting of the names of the distribution. We then run the algorithm once, with the defaults arguments `alpha=1` and `maxit=50`.

```
library(gllvmprime)

# The data is loaded
data <- readRDS("../data/example_1.rds")

# The families are specified.
family <- c(rep("gaussian", 10), rep("binomial", 10))

set.seed(213)
fit <- gllvmprime(data$Y, q=2, X=data$X, family=family)
```

One can use the argument `verbose = TRUE` to display convergence information (in terms of deviance) as the algorithm proceeds. Since every additional iteration may potentially be useful to reduce noise, the algorithm will not stop early if some convergence criterion is reached; instead, it will run the number of iterations

specified by the `maxit`argument, here set to the default value of `100`. We now manually check the convergence of the algorithm:

```
plot(fit)
```



The top-left plot represents the values of the loadings for all the 50 iterations, the top-right plot represents those of the fixed-effect coefficients, the bottom-left plot those of the scale parameters, and the bottom-right plot represents the evolution of the deviance, i.e., the fit of our model. While our estimator does not aim at minimizing the deviance, we have found after significant testing that it provides a very good indicator of the convergence of the algorithm: the deviance plot of a well fitted model would not show any obvious trend and be relatively stable. From the above graph, it seems that the iterates have stabilized around their solution.

For best results, fitting a model using `gllvmprime` should be iterative: after the first fit, the user should use the visual diagnostic tools to assess the convergence, and, if necessary, update the fit using the `update`. This process is iterated until the user is satisfied with the fit. On the one hand, this procedure requires some work on the user's side; on the other, it provides the reassurance that the algorithm has properly converged.

To try and improve on the result, let us use the `update` function. This function admits the same arguments as `gllvmprime`, and will use as default the values of the arguments right before the end of the last call of either `gllvmprime` or `update`. That is, it will start the fitting process where it was just left off in the previous call.

This gives the opportunity to change the `alpha` argument to either a lower value, in order to reduce the noise, or to a larger one to make faster progress. If left to its default value, `update` will continue the fit using a starting value of `alpha` given by the last value of alpha that was used, which is available in `fit$controls$alpha`. That is, the following two lines return the same value:

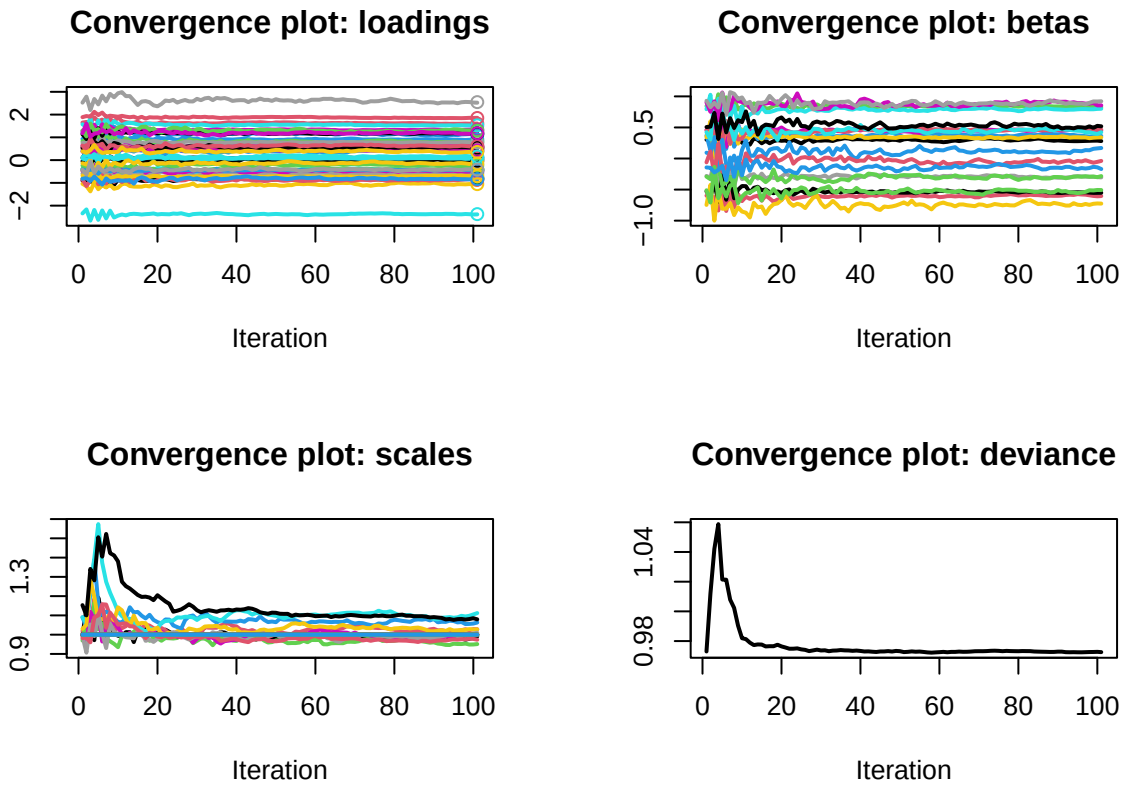
```
fit_1 <- update(fit)
fit_2 <- update(fit, alpha=fit$controls$alpha)
```

Let us continue with the final value of `alpha` of the previous fit, which is the default behavior of `update`. Finally, let us increase the number of iterations to 100 using `maxit=100` to obtain our final estimate.

```
fit <- update(fit, maxit=100, alpha=fit$controls$alpha)
```

We now again plot the new fit:

```
plot(fit)
```



The convergence is good: there is no obvious trends in the convergence plots and the deviance plot seems to have stabilized. The scale parameters still showcase a bit of variance, but this does not affect the loadings and fixed coefficients much, if at all. Updating the fit once more using `fit <- update(fit, alpha=fit$controls$alpha/2)` will stabilize the estimate of the scale parameters. Note that the final value of the parameter reported by the algorithm is computed as the average of the last 20 iteration values, which further reduces the noise in the final estimate.

Displaying the results

The values of the estimated parameter are available in `fit$parameters`:

- the estimate of the loading matrix is found in `fit$parameters$A`
- the estimate of the fixed effect coefficient matrix (if any) is found in `fit$parameters$B`
- the estimate of the loading matrix is found in `fit$parameters$A`
- the estimate of the scale vector is found in `fit$parameters$phi`

What's more, the estimated scores (the imputing values) are found in `fit$Z`, and the estimated means (the fitted values) in `fit$mean`. The parameters can be be visualized by printing the `fit` object. The `print` method accepts an extra parameter, `n`, which takes `n=10` as default and which limits the printing of the

parameter values to the first `n` lines. Since we have 20 responses, we set this value to `n=20` in order to see all parameters.

```
print(fit, n=20)
#> Fitted GLLVM model:
#> --------------------
#> Loadings:
#> # A tibble: 20 x 1
#>      A[,1]    [,2]
#>      <dbl>   <dbl>
#>  1  0.0939  0.0687
#>  2  1.61    1.33
#>  3 -0.783  -0.700
#>  4  0.901   0.776
#>  5  1.53    1.17
#>  6  0.626   0.534
#>  7 -0.448  -0.403
#>  8 -0.571  -0.439
#>  9 -0.824  -2.37
#> 10  1.85    0.425
#> 11  0.800  -0.148
#> 12  1.12    2.55
#> 13 -0.410   0.558
#> 14 -0.481   0.615
#> 15 -1.06    1.30
#> 16  0.519  -0.831
#> 17 -0.0236  0.160
#> 18 -0.864   1.18
#> 19 -0.226   0.360
#> 20  0.398  -0.418
#>
#> Fixed coefficients:
#> # A tibble: 20 x 1
#>      B[,1]   [,2]
#>      <dbl>  <dbl>
#>  1  0.289  -0.536
#>  2 -0.594  -0.609
#>  3  0.836  -0.395
#>  4  0.372   0.103
#>  5  0.793   0.775
#>  6  0.457   0.662
#>  7  0.340   0.518
#>  8 -0.297  -0.720
#>  9 -0.546  -0.127
#> 10  0.464  -0.454
#> 11 -0.298   0.712
#> 12  0.127  -0.198
#> 13  0.440   0.537
#> 14  0.889  -0.501
#> 15 -0.733   0.902
#> 16  0.898  -0.486
#> 17  0.513  -0.342
#> 18 -0.0492  0.626
#> 19 -0.527   0.343
```

```
#> 20 -0.148  -0.782
#>
#> Scale parameters:
#>  [1] 0.9872025 1.0244272 0.9560316 1.0655146 1.0976791 0.9948955 1.0270613
#>  [8] 0.9950984 1.0861449 0.9730701 1.0000000 1.0000000 1.0000000 1.0000000
#> [15] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
#>
#> Numerical controls:
#>  Number of iterations:   maxit = 100
#>  Batch size:         batch_size = 1000
#>  End value for alpha:    alpha = 0.1915646
```