Online appendix for the paper: “Bayesian Paired-Comparison with the bpcs package”

David Issa Mattos and Érika Martins Silva Ramos

2021-01-20

Contents

Chapter 1

Foreword

This is the online appendix for the paper *"Bayesian Paired-Comparison with the bpcs package"*. It contains a commented and reproducible code for all the analysis, tables and plots presented in the paper.

In the beginning of each study, we show a few lines of the original datasets. These datasets are either available through a link in the paper or through asking the authors directly. Therefore we do not provide or distribute the data in this appendix (or in the code repository).

1.1 Installation of the bpcs package

This appendix was compiled with the version 1.1.0 of the bpcs package.

The development and latest version of the bpcs package can be installed directly from Github with:

```
remotes::install_github('davidissamattos/bpcs')
```

The latest CRAN release can be installed with:

```
install.packages('bpcs')
```

We emphasize here that the CRAN version might some versions behind the latest stable release in Github.

1.2 Session info

This appendix is compiled automatically and the following session information was used to generate this appendix:

```r
sessionInfo()
```

```
## R version 4.0.3 (2020-10-10)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Big Sur 10.16
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] knitr_1.30      forcats_0.5.0   stringr_1.4.0   dplyr_1.0.2
##  [5] purrr_0.3.4     readr_1.3.1     tidyr_1.1.2     tibble_3.0.4
##  [9] ggplot2_3.3.3   tidyverse_1.3.0 bpcs_1.1.0
##
## loaded via a namespace (and not attached):
##  [1] httr_1.4.2            jsonlite_1.7.2        modelr_0.1.8
##  [4] RcppParallel_5.0.2    StanHeaders_2.21.0-7  assertthat_0.2.1
##  [7] BiocManager_1.30.10   rvcheck_0.1.8         stats4_4.0.3
## [10] blob_1.2.1            cellranger_1.1.0      yaml_2.2.1
## [13] pillar_1.4.7          backports_1.2.1       glue_1.4.2
## [16] digest_0.6.27         RColorBrewer_1.1-2    rvest_0.3.6
## [19] colorspace_2.0-0      htmltools_0.5.1       pkgconfig_2.0.3
## [22] rstan_2.21.2          broom_0.7.0           haven_2.3.1
## [25] bookdown_0.21         scales_1.1.1          processx_3.4.5
## [28] generics_0.1.0        ellipsis_0.3.1        withr_2.3.0
## [31] cli_2.2.0             magrittr_2.0.1        crayon_1.3.4
## [34] readxl_1.3.1          evaluate_0.14         ps_1.5.0
## [37] badger_0.0.9          fs_1.5.0              fansi_0.4.1
## [40] xml2_1.3.2            pkgbuild_1.2.0        tools_4.0.3
## [43] loo_2.4.1             prettyunits_1.1.1     hms_0.5.3
## [46] lifecycle_0.2.0       matrixStats_0.57.0    V8_3.4.0
## [49] munsell_0.5.0         reprex_0.3.0          callr_3.5.1
## [52] compiler_4.0.3        rlang_0.4.10          grid_4.0.3
## [55] rstudioapi_0.13       rmarkdown_2.6         gtable_0.3.0
## [58] codetools_0.2-16      inline_0.3.17         DBI_1.1.0
## [61] curl_4.3              R6_2.5.0              gridExtra_2.3
## [64] rstantools_2.1.1.9000 lubridate_1.7.9       dlstats_0.1.3
## [67] stringi_1.5.3         parallel_4.0.3        Rcpp_1.0.5
## [70] vctrs_0.3.6          dbplyr_1.4.4         tidyselect_1.1.0
## [73] xfun_0.20
```

Chapter 2

Basic functionality

Although a large overview of the functionality of the `bpcs` is available in the official documentation (https://davidissamattos.github.io/bpcs) we provide here a short example based on the paper:

> Luckett, Curtis R., Sara L. Burns, and Lindsay Jenkinson. "Estimates of relative acceptability from paired preference tests." Journal of Sensory Studies 35.5 (2020): e12593.

```
library(bpcs)
library(tidyverse)
library(knitr)
options(mc.cores = parallel::detectCores())
rstan::rstan_options(auto_write = TRUE)
set.seed(99)
```

2.1 Reading and preparing the data

This paper analyzes food preferences using paired comparisons (and compare different methods). The original data was made available in the paper and it can be found in the link:

```
d <- readxl::read_xlsx(path='data/PREF_DATA.xlsx', sheet = 'Pizza')
```

Below we see a fragment of how the dataset looks like:

```
sample_n(d, size=5) %>%
  kable(caption="Example of the pizza data frame") %>%
  kableExtra::kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"))
```

This data is in a aggregated format. That is each row contains more than one observation. For example, the first row shows that Tombstone was voted 16 times

Table 2.1: Example of the pizza data frame

Prod1	Prod2	Win1	Win2
Tombstone	Red Barron	16	21
Freschetta	DiGiorno	20	18
aKroger	Tombstone	17	23
Freschetta	Tombstone	22	16
DiGiorno	aKroger	26	13

against Red Barron and Red Barron was voted 21 times against Tombstone

To use with the bpcs package, we need to expand it to a single contest per row, in a long format. We can use the function `expand_aggregated_data` of the bpcs package exactly for this purpose.

This leads to a data frame with 380 rows (same number of wins for 1 and 2). This function adds an id column to the data, so each row is uniquely represented (important if you will do some transformations later). Below we examplify how the expanded data looks like

```
dpizza <- expand_aggregated_data(d = d, player0 = 'Prod1', player1 = 'Prod2', wins0 =
# renaming the columns
colnames(dpizza) <- c('Prod0','Prod1', 'y', 'contest_id')
# creating a short table to exemplify
sample_n(dpizza, size = 10) %>%
  kable(caption = 'Sample of the expanded pizza data set') %>%
  kableExtra::kable_styling(bootstrap_options = c("striped", "hover", "condensed", "res
  kableExtra::scroll_box(width = "100%")
```

\begin{table}

\caption{(#tab:expand_data2)Sample of the expanded pizza data set}

Prod0	Prod1	y	contest_id
aKroger	Red Barron	1	362
Freschetta	DiGiorno	1	226
DiGiorno	Red Barron	0	128
aKroger	Red Barron	1	358
Red Barron	Freschetta	0	269
Tombstone	Red Barron	1	20
DiGiorno	Tombstone	1	68
Freschetta	aKroger	1	265
Freschetta	DiGiorno	0	210
DiGiorno	aKroger	0	88

\end{table}

2.2 The Bayesian Bradley-Terry analysis

Now that we have the data in the correct format we can use bpcs package to model the Bayesian Bradley-Terry Model. It is a good practice to save the result fitted model in a file right after sampling. Some models might take several minutes to fit and you probably don't want to keep re-fitting the model always. After saving you can just read the model and continue your analysis instead of re-fitting. The `save_bpc_model` is a wrapper function around the `saveRDS` function with a few smaller checks. Few free to use any. To read you can use the `load_bpc_model` or the `readRDS` functions.

Let's run the simplest Bayesian Bradley-Terry model:

```r
m <- bpc(data = dpizza,
         player0 = 'Prod0',
         player1 = 'Prod1',
         result_column = 'y',
         solve_ties = 'none',
         model_type = 'bt',
         iter=3000)
save_bpc_model(m, 'pizza','fittedmodels')
```

To load:

```r
m <- load_bpc_model('fittedmodels/pizza.RDS')
```

2.2.1 Diagnostics

After sampling, we can investigate the convergence of the model and the predictive posterior with shinystan. Convergence checks are already available in the fitted model, but for the posterior checks we need to first calculate the posterior predictive values with the `posterior_predictive` function. This function returns a list with two values, the `y` (original fitted values) and the `y_pred` (posterior predictve). Both are in the correct format to use with shinystan.

We save them to the global environment and then we load it in shinystan directly (through the GUI).

```r
#posterior predictive check
y_pp <- posterior_predictive(m)
y <- y_pp$y
y_pred <- y_pp$y_pred
```

```r
launch_shinystan(m)
```

Since everything looks fine we can proceed with the analysis.

2.2.2 Summary information

The `summary` function in the command line provides some tables that help understand the model, such as the summary of the parameters, the probability of winning, and the ranking table.

```
summary(m)
```

```
## Estimated baseline parameters with HPD intervals:
##
##
## Table: \label{tab:unnamed-chunk-11}Parameters estimates
##
## Parameter               Mean   HPD_lower   HPD_higher
## -------------------  ------  ----------  -----------
## lambda[Tombstone]     -0.20       -3.02         2.37
## lambda[DiGiorno]       0.26       -2.39         2.99
## lambda[Freschetta]     0.16       -2.54         2.82
## lambda[Red Barron]     0.18       -2.50         2.84
## lambda[aKroger]       -0.41       -3.11         2.24
## NOTES:
## * A higher lambda indicates a higher team ability
##
## Posterior probabilities:
## These probabilities are calculated from the predictive posterior distribution
## for all player combinations
##
##
## Table: \label{tab:unnamed-chunk-11}Estimated posterior probabilites
##
## i            j             i_beats_j   j_beats_i
## -----------  -----------  ----------  ----------
## aKroger      DiGiorno           0.62        0.38
## aKroger      Freschetta         0.70        0.30
## aKroger      Red Barron         0.69        0.31
## aKroger      Tombstone          0.62        0.38
## DiGiorno     Freschetta         0.48        0.52
## DiGiorno     Red Barron         0.57        0.43
## DiGiorno     Tombstone          0.44        0.56
## Freschetta   Red Barron         0.53        0.47
## Freschetta   Tombstone          0.44        0.56
## Red Barron   Tombstone          0.36        0.64
##
## Rank of the players' abilities:
## The rank is based on the posterior rank distribution of the lambda parameter
##
##
```

```
## Table: \label{tab:unnamed-chunk-11}Estimated posterior ranks
##
## Parameter             MedianRank   MeanRank   StdRank
## --------------------  -----------  ---------  --------
## lambda[DiGiorno]                1       1.63      0.80
## lambda[Freschetta]              2       2.28      0.84
## lambda[Red Barron]              2       2.19      0.83
## lambda[Tombstone]               4       4.05      0.52
## lambda[aKroger]                 5       4.84      0.37
```

The presented tables can also be obtained individually as data frame. Additionally, for all of them it is possible to obtain the posterior distribution.

The package has some publication-ready functions to create plots and tables. The format option can set it to latex, html, pandoc etc.

Parameters with HPDI

```r
get_parameters_table(m, format='html', caption = 'Parameter estimates with HPDI', digits = 3)
```

Parameter estimates with HPDI

Parameter

Mean

HPD_lower

HPD_higher

lambda[Tombstone]

-0.199

-3.023

2.373

lambda[DiGiorno]

0.264

-2.388

2.994

lambda[Freschetta]

0.158

-2.541

2.821

lambda[Red Barron]

0.175

-2.502

2.836

lambda[aKroger]

-0.414

-3.111

2.241

Probabilities

```
get_probabilities_table(m, format='html', caption='Probability of selectiig a brand of
```

Probability of selectiig a brand of pizza over the other

i

j

i__beats__j

j__beats__i

aKroger

DiGiorno

0.64

0.36

aKroger

Freschetta

0.58

0.42

aKroger

Red Barron

0.63

0.37

aKroger

Tombstone

0.50

0.50

DiGiorno

Freschetta

0.43

0.57

DiGiorno

Red Barron

0.43

0.57

DiGiorno

Tombstone

0.37

0.63

Freschetta

Red Barron

0.46

0.54

Freschetta

Tombstone

0.34

0.66

Red Barron

Tombstone

0.31

0.69

Rank

```
get_rank_of_players_table(m, caption='Rank of Pizza', format='html')
```

Rank of Pizza

Parameter

MedianRank

MeanRank

StdRank

lambda[DiGiorno]

1

1.630

0.776

lambda[Freschetta]

2

2.260

0.839

lambda[Red Barron]

2

2.209

0.847

lambda[Tombstone]

4

4.062

0.524

lambda[aKroger]

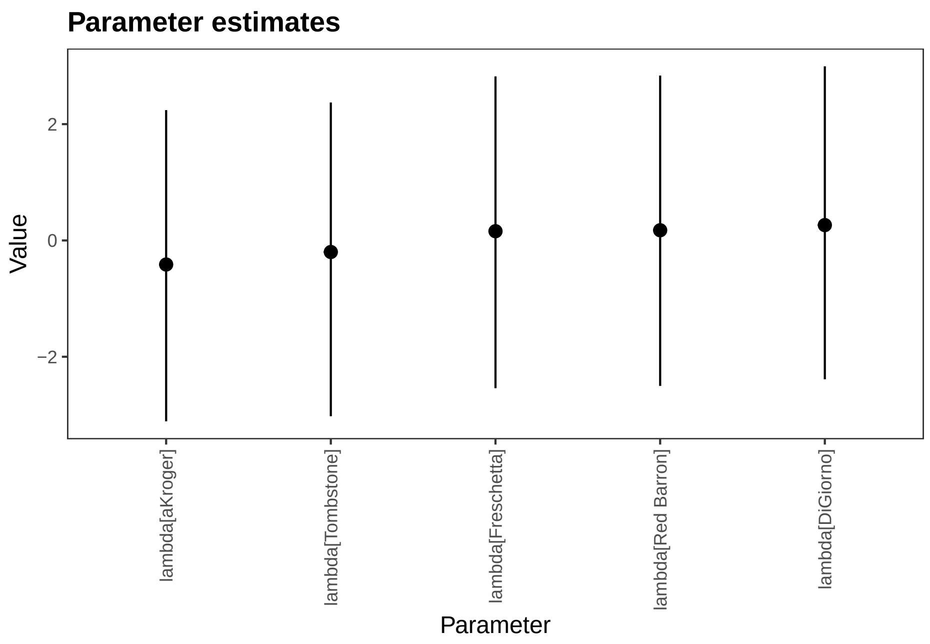
5

4.839

0.381

Plot

```r
plot(m, rotate_x_labels=T)
```



We can also obtain information criteria to compare different models that fit the same data.

Below we show how to obtain the WAIC

```r
get_waic(m)
```

```
##
## Computed from 8000 by 380 log-likelihood matrix
##
##           Estimate  SE
## elpd_waic   -259.8 3.9
## p_waic         4.0 0.1
## waic         519.6 7.8
```

This chapter show some basic functionality of the package. The next chapters will show how to apply these functions to solve more interesting and complex problems. We also recommend checking the package documentation at: https://davidissamattos.github.io/bpcs/index.html

Chapter 3

Re-analysis 1

This reanalysis is based on the paper:

Iwasa, Kazunori, et al. "Visual perception of moisture is a pathogen detection mechanism of the behavioral immune system." Frontiers in Psychology 11 (2020): 170.

The data of this paper can be obtained from the repository: https://osf.io/5quj9/

```
library(bpcs)
library(tidyverse)
options(mc.cores = parallel::detectCores())
rstan::rstan_options(auto_write = TRUE)
set.seed(99)
```

3.1 Importing the data

First let's read the data:

```
d <- readxl::read_xlsx('data/moisture.xlsx', sheet = 'Exp02_BTmodel')
```

Below we show a sample of how the original data looks like:

```
sample_n(d, size = 5) %>%
  kable(caption = 'Sample of the original data') %>%
  kableExtra::scroll_box(width = "100%")
```

The data is in the aggregated format. So let's expand it to the long format

```
d_moisture <- expand_aggregated_data(d, player0 = 'player1', player1='player2', wins0 = 'win1', w
```

Now the data looks like this:

Table 3.1: Sample of the original data

player1	player2	win1	win2
image7	image6	131	109
image5	image6	18	222
image7	image2	226	14
image4	image1	239	1
image5	image8	20	220

```r
sample_n(d_moisture, size = 20) %>%
  kable(caption = 'The data in the long format') %>%
  kableExtra::scroll_box(width = "100%")
```

3.2 Analysis with the Bradley-Terry model and the order effect model

Although this is multiple judgment case, the dataset in the aggregated format does not provide information of how each individual voted, therefore we cannot compensate this effect. Therefore, we will create an analysis with only the Bradley-Terry model and the Bradley-Terry model with order effect

First, let's sample the Bradley-Terry model

```r
m_moisture <-
  bpc(
    d_moisture,
    player0 = 'player0',
    player1 = 'player1',
    result_column = 'y',
    model_type = 'bt',
    iter = 3000
  )
save_bpc_model(m_moisture,'m_moisture','fittedmodels')
```

Low let's sample the model with order effect.

Although the authors said that the order of the images was inverted to compensate order effect, we can still estimate if there is an order effect or not in the choice.

But first we need to create a column indicating if there was order effect for that case or not. In this problem, we just indicate with a column of ones that all instances could have an order effect. Not that the package marks the order effect relative to player1. So if the values should be interpret as such.

Table 3.2: The data in the long format

player0	player1	y	rowid
image2	image7	1	2922
image5	image2	0	7102
image7	image6	1	11490
image2	image8	1	3200
image1	image3	1	358
image8	image4	1	12704
image6	image3	0	8973
image8	image3	0	12308
image6	image4	0	9188
image5	image2	0	7071
image5	image1	0	6724
image4	image7	1	6409
image8	image1	0	11986
image4	image1	0	5278
image3	image7	1	4694
image4	image5	1	5972
image1	image8	1	1533
image4	image8	1	6560
image1	image3	1	398
image3	image5	1	4228

```r
d_moisture <- d_moisture %>%
  dplyr::mutate(z1 = 1)
```

```r
m_moisture_order <-
  bpc(
    d_moisture,
    player0 = 'player0',
    player1 = 'player1',
    result_column = 'y',
    z_player1 = 'z1',
    model_type = 'bt-ordereffect',
    iter = 3000
  )
save_bpc_model(m_moisture_order,'m_moisture_order','fittedmodels')
```

3.3 Diagnostics

Checking convergence of the models

```r
launch_shinystan(m_moisture)
```

```r
launch_shinystan(m_moisture_order)
```

3.3.1 Tables and plots

First, lets get a table for the parameters (to export it to Latex just utilize the format option)

```r
get_parameters_table(m_moisture, format = 'html', caption = 'Parameters estimates for
```

Parameters estimates for the simple Bradley-Terry model

Parameter	Mean	HPD_lower	HPD_higher
lambda[image1]	-4.461	-6.507	-2.307
lambda[image2]	-2.350		

-4.459

-0.257

lambda[image3]

-0.038

-2.138

2.061

lambda[image4]

0.153

-1.933

2.273

lambda[image5]

0.313

-1.805

2.377

lambda[image6]

1.855

-0.243

3.974

lambda[image7]

2.032

-0.069

4.128

lambda[image8]

3.044

0.849

5.060

```r
get_parameters_table(
  m_moisture_order,
  params = c('lambda', 'gm'),
  format = 'html',
  caption = 'Parameters estimates for the Bradley-Terry model with order effect'
)
```

Parameters estimates for the Bradley-Terry model with order effect

Parameter	Mean	HPD_lower	HPD_higher
lambda[image1]	-4.524	-6.526	-2.428
lambda[image2]	-2.410	-4.338	-0.281
lambda[image3]	-0.097	-2.029	2.020
lambda[image4]	0.095	-1.834	2.226
lambda[image5]	0.254	-1.679	2.377
lambda[image6]	1.799	-0.107	3.955
lambda[image7]	1.974		

0.017

4.083

lambda[image8]

2.988

1.026

5.100

gm

0.001

-0.054

0.057

We can see from the table of the order effect that the gamma parameter is very close to zero, indicating that there is no order effect

Now lets compute the posterior ranks of the images based on the first BT model

```
r <- get_rank_of_players_df(m_moisture)
```

Generating a table with the images

```
r %>%
  mutate(Image = c("data/moisture/Stimuli/image08.jpg",
                   "data/moisture/Stimuli/image07.jpg",
                   "data/moisture/Stimuli/image06.jpg",
                   "data/moisture/Stimuli/image05.jpg",
                   "data/moisture/Stimuli/image04.jpg",
                   "data/moisture/Stimuli/image03.jpg",
                   "data/moisture/Stimuli/image02.jpg",
                   "data/moisture/Stimuli/image01.jpg") %>% pander::pandoc.image.return()) %>%
  knitr::kable(caption = "Rank of the images based on moisture content", format='html', booktabs=
```

Rank of the images based on moisture content

Parameter

MedianRank

MeanRank

StdRank

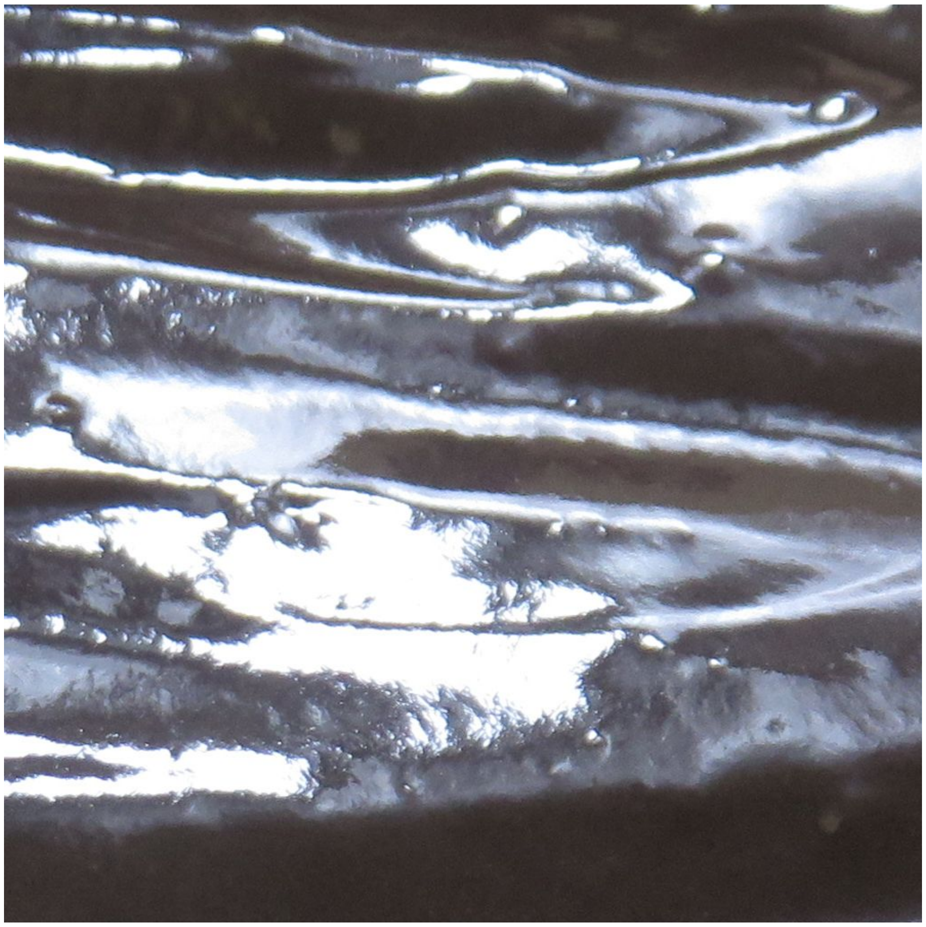
Image

lambda[image8]

1

1.000

0.0000000

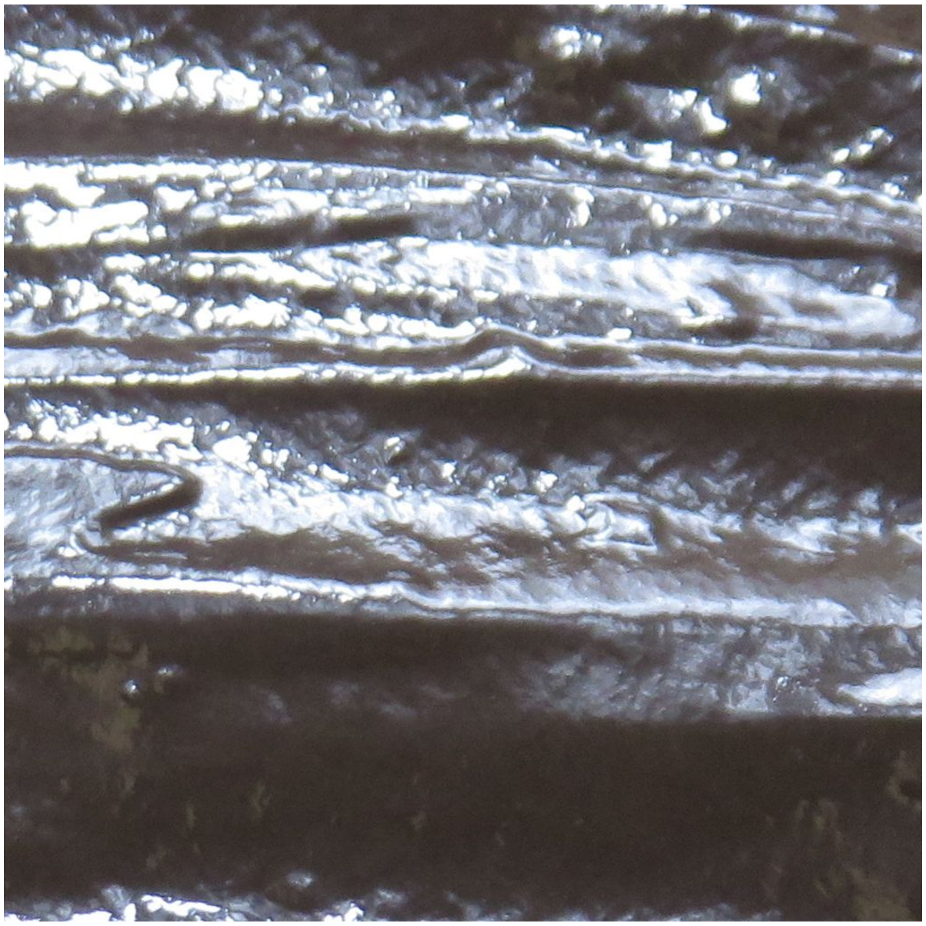


lambda[image7]

2

2.003

0.0547174

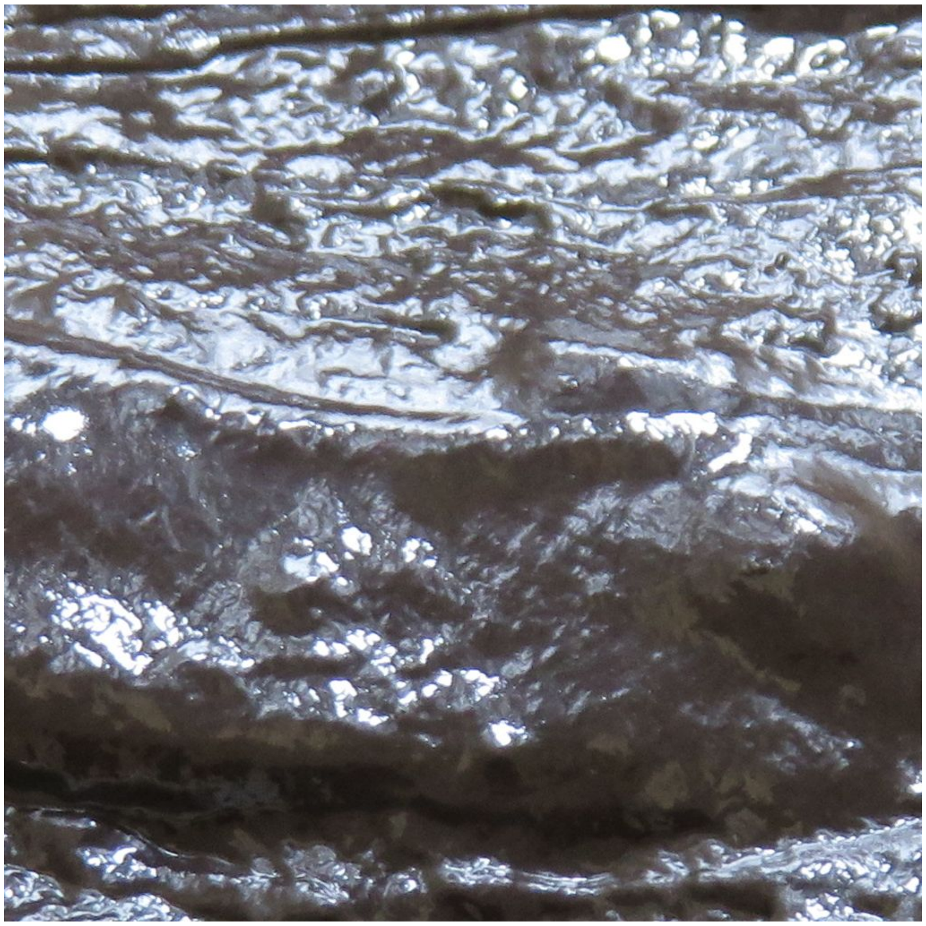


lambda[image6]

3

2.997

0.0547174

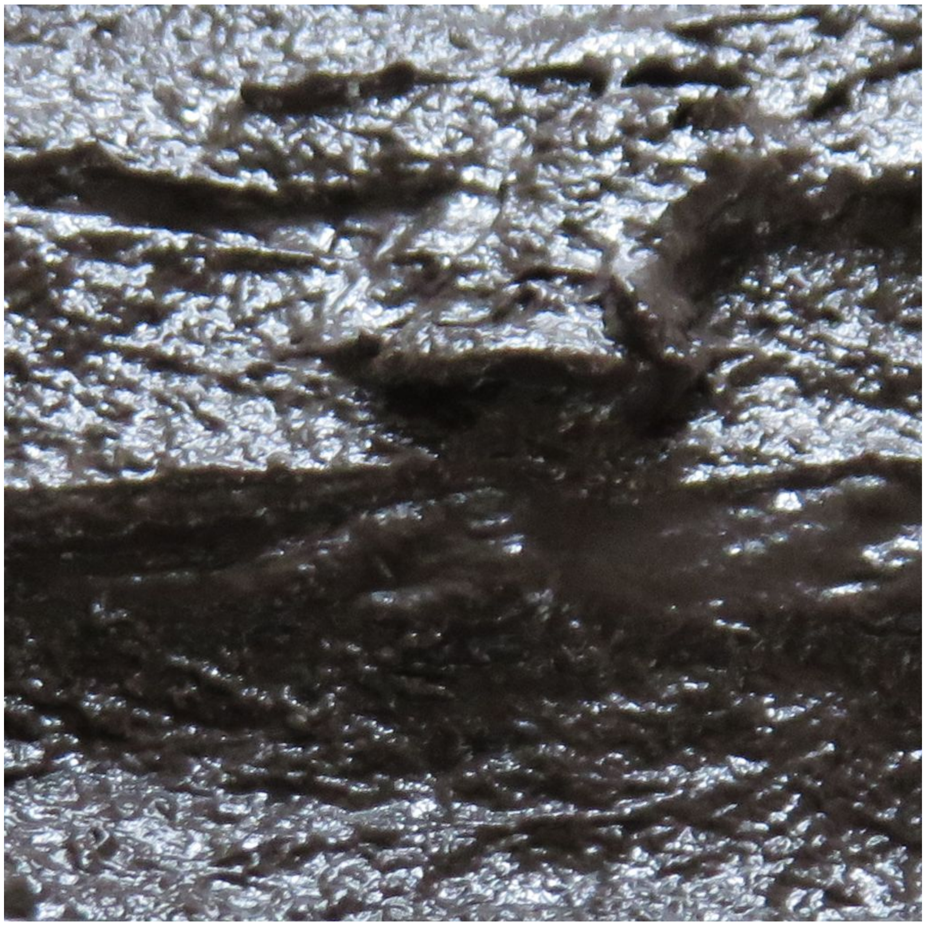


lambda[image5]

4

4.008

0.0891288

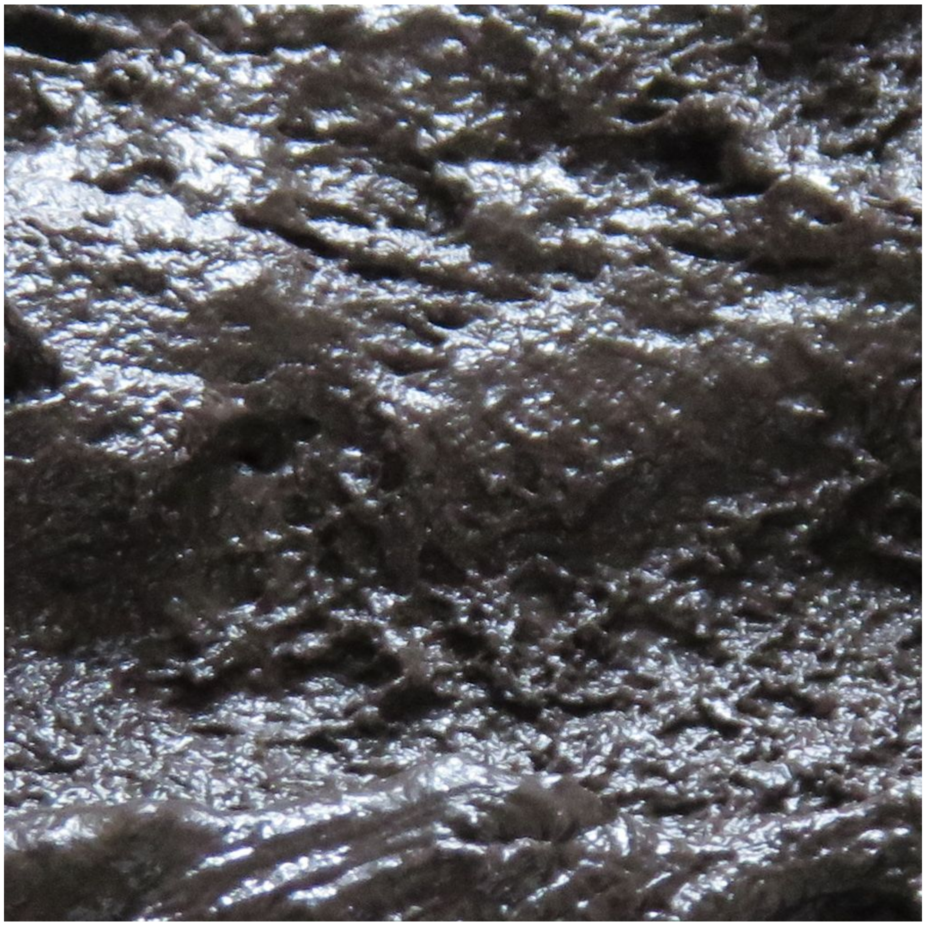


lambda[image4]

5

4.993

0.0946571

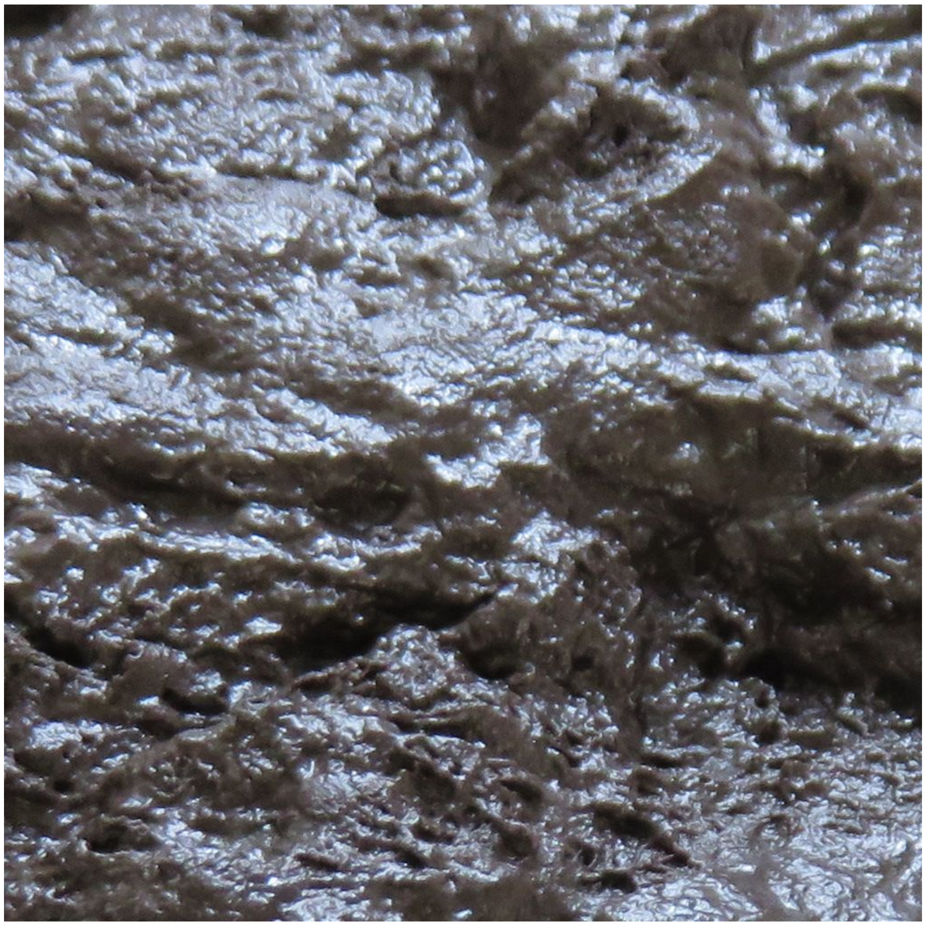


lambda[image3]

6

5.999

0.0316228

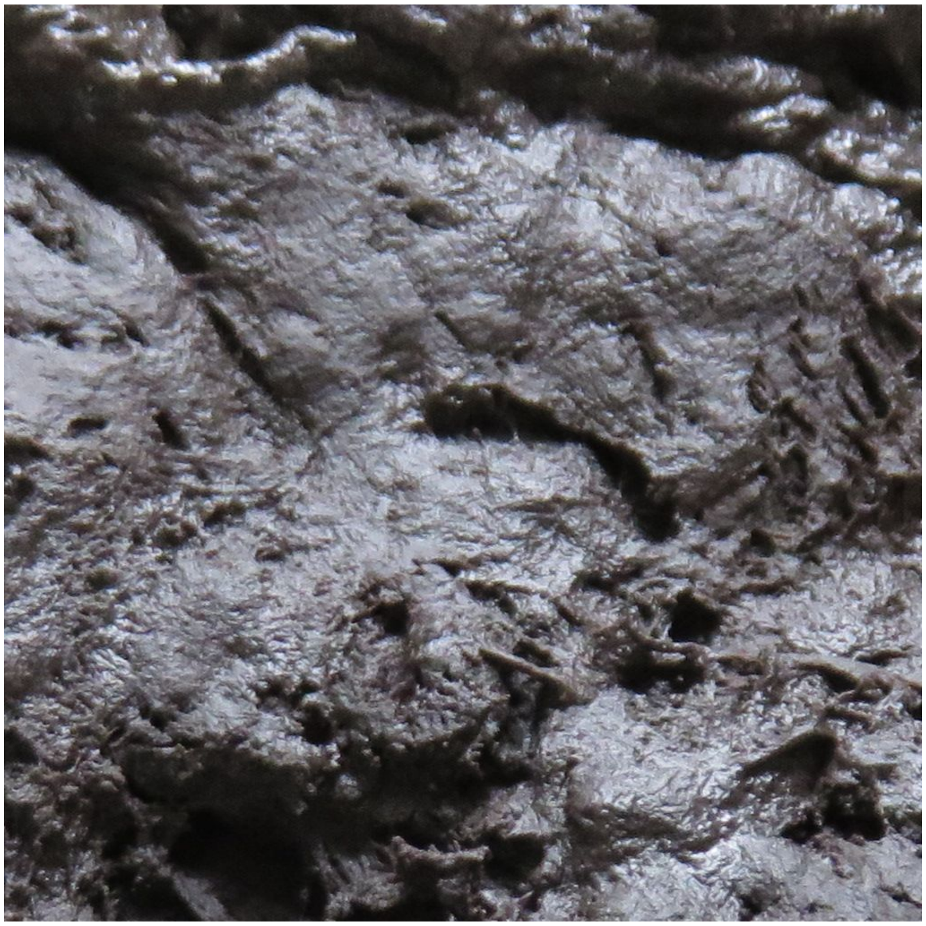


lambda[image2]

7

7.000

0.0000000



lambda[image1]

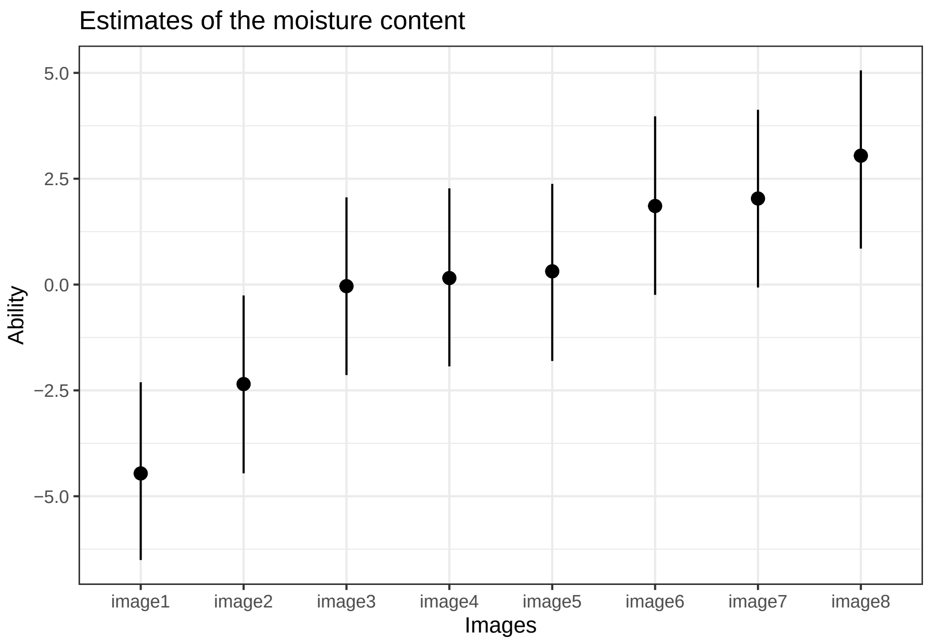
8

8.000

0.0000000

Now lets get a caterpillar style plot

```r
get_parameters_plot(
  m_moisture,
  HPDI = T,
  title = 'Estimates of the moisture content',
  xaxis = 'Images',
  yaxis = 'Ability',
  rotate_x_labels = F,
  APA = F
) + scale_x_discrete(
  labels = c(
    "image1",
    "image2",
    "image3",
    "image4",
    "image5",
    "image6",
    "image7",
    "image8"
  )
) + theme_bw()
```



3.3.2 WAIC

Calculating the WAIC of both models

```
get_waic(m_moisture)
```

```
##
## Computed from 8000 by 13440 log-likelihood matrix
##
##           Estimate    SE
## elpd_waic  -4066.1  78.8
## p_waic         6.9   0.2
## waic        8132.2 157.6
```

```
get_waic(m_moisture_order)
```

```
##
## Computed from 8000 by 13440 log-likelihood matrix
##
##           Estimate    SE
## elpd_waic  -4067.1  78.9
## p_waic         7.8   0.2
## waic        8134.2 157.7
```

We can see that the WAIC of the models are quite similar and that the model without the order effect has a slightly smaller WAIC and less parameters. Therefore we will select it.

Chapter 4

Re-analysis 2

This reanalysis is based on the paper:

> Huskisson, S.M., Jacobson, S.L., Egelkamp, C.L. et al. Using a Touchscreen Paradigm to Evaluate Food Preferences and Response to Novel Photographic Stimuli of Food in Three Primate Species (Gorilla gorilla gorilla, Pan troglodytes, and Macaca fuscata). Int J Primatol 41, 5–23 (2020). https://doi.org/10.1007/s10764-020-00131-0

```r
library(bpcs)
library(tidyverse)
library(knitr)
options(mc.cores = parallel::detectCores())
rstan::rstan_options(auto_write = TRUE)
set.seed(99)
```

4.1 Importing the data

The data from this paper was made available upon request and below we exemplify a few rows of how the original dataset looks like

```r
d <- readr::read_csv('data/touchscreen.csv')
```

Previewing how the data looks like

```r
dplyr::sample_n(d, size = 10) %>%
  knitr::kable(caption='Sample of how the dataset looks like') %>%
  kableExtra::kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"))
  kableExtra::scroll_box(width = "100%")
```

Now we need to modify a bit the data frame to create a column with the results as 0 and 1.

Table 4.1: Sample of how the dataset looks like

date	species_type	SubjectCode	Sex	Trial	image1	image2	image_chosen	conc
10/12/17	Gorilla	Gorilla1	Male	2	Ap	Cu	Ap	ApC
8/31/17	Gorilla	Gorilla3	Male	16	Ca	To	To	CaT
8/16/17	Gorilla	Gorilla5	Female	16	Cu	Gr	Cu	CuG
11/13/17	Chimpanzee	Chimpanzee3	Female	14	Ap	Ca	Ca	ApC
4/11/17	Gorilla	Gorilla3	Male	15	Ca	Cu	Ca	CaC
1/22/18	Macaque	Macaque7	Female	1	Ca	Pe	Pe	CaP
1/5/17	Gorilla	Gorilla4	Male	28	Ca	Tu	Tu	CaT
3/3/17	Chimpanzee	Chimpanzee1	Female	19	Cu	Gr	Gr	CuG
12/4/17	Chimpanzee	Chimpanzee1	Female	27	Ap	Tu	Tu	ApT
5/1/17	Gorilla	Gorilla6	Male	14	Ca	Tu	Ca	CaT

Creating a numerical result vector with 0 for image1 and 1 for image2

```r
d <- d %>%
  dplyr::mutate(y =ifelse(image_chosen==image1, 0, 1))
```

Adding names to the abbreviations

```r
#image1
d$image1 <- dplyr::recode(d$image1, "Ca" = 'Carrot')
d$image1 <- dplyr::recode(d$image1, "Cu" = 'Cucumber')
d$image1 <- dplyr::recode(d$image1, "Tu" = 'Turnip')
d$image1 <- dplyr::recode(d$image1, "Gr" = 'Grape')
d$image1 <- dplyr::recode(d$image1, "To" = 'Tomato')
d$image1 <- dplyr::recode(d$image1, "Ap" = 'Apple')
d$image1 <- dplyr::recode(d$image1, "Jp" = 'Jungle Pellet')
d$image1 <- dplyr::recode(d$image1, "Ce" = 'Celery')
d$image1 <- dplyr::recode(d$image1, "Gb" = 'Green Beans')
d$image1 <- dplyr::recode(d$image1, "Oa" = 'Oats')
d$image1 <- dplyr::recode(d$image1, "Pe" = 'Peanuts')

#image2
d$image2 <- dplyr::recode(d$image2, "Ca" = 'Carrot')
d$image2 <- dplyr::recode(d$image2, "Cu" = 'Cucumber')
d$image2 <- dplyr::recode(d$image2, "Tu" = 'Turnip')
d$image2 <- dplyr::recode(d$image2, "Gr" = 'Grape')
d$image2 <- dplyr::recode(d$image2, "To" = 'Tomato')
d$image2 <- dplyr::recode(d$image2, "Ap" = 'Apple')
d$image2 <- dplyr::recode(d$image2, "Jp" = 'Jungle Pellet')
d$image2 <- dplyr::recode(d$image2, "Ce" = 'Celery')
d$image2 <- dplyr::recode(d$image2, "Gb" = 'Green Beans')
d$image2 <- dplyr::recode(d$image2, "Oa" = 'Oats')
```

Table 4.2: Sample of how the gorilla dataset looks like

date	species_type	SubjectCode	Sex	Trial	image1	image2	image_chosen	concat_test
3/3/17	Gorilla	Gorilla2	Male	31	Cucumber	Tomato	To	CuTo
11/8/17	Gorilla	Gorilla6	Male	6	Tomato	Turnip	To	ToTu
3/20/17	Gorilla	Gorilla2	Male	6	Grape	Tomato	To	GrTo
3/8/17	Gorilla	Gorilla6	Male	20	Carrot	Grape	Ca	CaGr
8/23/17	Gorilla	Gorilla5	Female	4	Cucumber	Grape	Cu	CuGr
5/25/17	Gorilla	Gorilla4	Male	19	Carrot	Cucumber	Ca	CaCu
8/3/17	Gorilla	Gorilla6	Male	14	Carrot	Tomato	To	CaTo
8/23/17	Gorilla	Gorilla5	Female	24	Cucumber	Grape	Gr	CuGr
12/7/16	Gorilla	Gorilla2	Male	28	Cucumber	Turnip	Cu	CuTu
5/25/17	Gorilla	Gorilla1	Male	4	Carrot	Cucumber	Ca	CaCu

```
d$image2 <- dplyr::recode(d$image2, "Pe" = 'Peanuts')
```

Separating the data into three datasets. One for each species.

```
macaque <- d %>%
  dplyr::filter(species_type=='Macaque')

chip <- d %>%
  dplyr::filter(species_type=='Chimpanzee')

gor <- d %>%
  dplyr::filter(species_type=='Gorilla')
```

Below we show a few lines of each dataset:

```
dplyr::sample_n(gor, size = 10) %>%
  knitr::kable(caption='Sample of how the gorilla dataset looks like') %>%
  kableExtra::kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"))
  kableExtra::scroll_box(width = "100%")
```

```
dplyr::sample_n(chip, size = 10) %>%
  knitr::kable(caption='Sample of how the chimpanzees dataset looks like') %>%
  kableExtra::kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"))
  kableExtra::scroll_box(width = "100%")
```

```
dplyr::sample_n(macaque, size = 10) %>%
  knitr::kable(caption='Sample of how the macaques dataset looks like') %>%
  kableExtra::kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"))
  kableExtra::scroll_box(width = "100%")
```

Table 4.3: Sample of how the chimpanzees dataset looks like

date	species_type	SubjectCode	Sex	Trial	image1	image2	image_chosen	c
3/1/18	Chimpanzee	Chimpanzee4	Male	3	Apple	Carrot	Ap	[illegible]
12/7/17	Chimpanzee	Chimpanzee1	Female	18	Carrot	Tomato	To	C
4/26/17	Chimpanzee	Chimpanzee3	Female	7	Cucumber	Turnip	Cu	C
11/1/17	Chimpanzee	Chimpanzee1	Female	19	Apple	Grape	Gr	[illegible]
7/26/17	Chimpanzee	Chimpanzee3	Female	20	Cucumber	Grape	Cu	C
10/26/17	Chimpanzee	Chimpanzee1	Female	7	Apple	Grape	Gr	[illegible]
9/18/17	Chimpanzee	Chimpanzee4	Male	27	Carrot	Grape	Gr	C
12/8/16	Chimpanzee	Chimpanzee1	Female	12	Cucumber	Turnip	Cu	C
12/14/17	Chimpanzee	Chimpanzee3	Female	14	Tomato	Turnip	To	T
1/12/17	Chimpanzee	Chimpanzee4	Male	21	Cucumber	Turnip	Tu	C

Table 4.4: Sample of how the macaques dataset looks like

date	species_type	SubjectCode	Sex	Trial	image1	image2	in
10/6/17	Macaque	Macaque6	Female	10	Jungle Pellet	Celery	Jp
09-06-2017@11-43	Macaque	Macaque1	Male	12	Carrot	Celery	C
04-30-2018@11-54	Macaque	Macaque1	Male	8	Peanuts	Oats	P
8/7/17	Macaque	Macaque3	Female	1	Peanuts	Celery	C
05-04-2018@11-27	Macaque	Macaque1	Male	1	Celery	Green Beans	G
1/16/18	Macaque	Macaque7	Female	14	Carrot	Peanuts	C
04-02-2018@11-27	Macaque	Macaque1	Male	28	Carrot	Green Beans	C
9/7/17	Macaque	Macaque6	Female	2	Carrot	Celery	C
8/14/17	Macaque	Macaque2	Female	14	Carrot	Peanuts	P
5/24/18	Macaque	Macaque4	Female	17	Jungle Pellet	Oats	Jp

4.2 Simple Bradley-Terry model

Now that the data is ready let's fit three simple Bayesian Bradley-Terry models

```
m1_macaque <-
  bpc(
    macaque,
    player0 = 'image1',
    player1 = 'image2',
    result_column = 'y',
    model_type = 'bt',
    priors = list(prior_lambda_std = 1.0),
    iter = 3000
  )
save_bpc_model(m1_macaque, 'm1_macaque', 'fittedmodels')

m1_chip <-
  bpc(
    chip,
    player0 = 'image1',
    player1 = 'image2',
    result_column = 'y',
    model_type = 'bt',
    priors = list(prior_lambda_std = 1.0),
    iter = 3000
  )
save_bpc_model(m1_chip, 'm1_chip', 'fittedmodels')

m1_gor <-
  bpc(
    gor,
    player0 = 'image1',
    player1 = 'image2',
    result_column = 'y',
    model_type = 'bt',
    priors = list(prior_lambda_std = 1.0),
    iter = 3000
  )
save_bpc_model(m1_gor, 'm1_gor', 'fittedmodels')
```

4.2.1 Assessing the fitness of the model

Here we are illustrating how to conduct the diagnostic analysis for one model only. Since the shinystan app does not appear in the compiled appendix we are just representing it here once for the Chimpanzees model. Note that it is still possible to use the bayesplot package to generate static figures if needed.

```
#First we get the posterior predictive in the environment
pp_m1_chip <- posterior_predictive(m1_chip, n = 100)
y_pp_m1_chip <- pp_m1_chip_gor$y
ypred_pp_m1_chip <- pp_m1_chip$y_pred
```

Then we launch shinystan to assess convergence and validity of the model

```
launch_shinystan(m1_chip)
```

4.2.2 Getting the WAIC

Before we start getting the parameters tables and etc let's get the WAIC so we can compare with the next model (with random effects)

```
get_waic(m1_macaque)
```

```
##
## Computed from 8000 by 8400 log-likelihood matrix
##
##           Estimate    SE
## elpd_waic  -3550.8  56.7
## p_waic         5.0   0.1
## waic        7101.6 113.4
```

```
get_waic(m1_chip)
```

```
##
## Computed from 8000 by 5400 log-likelihood matrix
##
##           Estimate   SE
## elpd_waic  -3599.7 17.0
## p_waic         5.1  0.0
## waic        7199.5 33.9
```

```
get_waic(m1_gor)
```

```
##
## Computed from 8000 by 8100 log-likelihood matrix
##
##           Estimate   SE
## elpd_waic  -4883.7 36.0
## p_waic         5.0  0.1
## waic        9767.4 71.9
```

4.3 Bradley-Terry model with random effects for individuals

Let's add the cluster SubjectCode as a random effects in our model

```
m2_macaque <-
  bpc(
    macaque,
    player0 = 'image1',
    player1 = 'image2',
    result_column = 'y',
    model_type = 'bt-U',
    cluster = c('SubjectCode'),
    priors = list(
      prior_lambda_std = 1.0,
      prior_U1_std = 1.0
    ),
    iter = 3000
  )
save_bpc_model(m2_macaque, 'm2_macaque', 'fittedmodels')

m2_chip <-
  bpc(
    chip,
    player0 = 'image1',
    player1 = 'image2',
    cluster = c('SubjectCode'),
    result_column = 'y',
    model_type = 'bt-U',
    priors = list(
      prior_lambda_std = 1.0,
      prior_U1_std = 1.0
    ),
    iter = 3000
  )
save_bpc_model(m2_chip, 'm2_chip', 'fittedmodels')

m2_gor <-
  bpc(
    gor,
    player0 = 'image1',
    player1 = 'image2',
    cluster = c('SubjectCode'),
    result_column = 'y',
    model_type = 'bt-U',
    priors = list(
      prior_lambda_std = 1.0,
      prior_U1_std = 1.0
    ),
    iter = 3000
  )
save_bpc_model(m2_gor, 'm2_gor', 'fittedmodels')
```

Of course we should run diagnostic analysis on the models. For the gorillas

```
launch_shinystan(m2_gor)
```

4.3.1 Getting the WAIC

Before we start plotting tables let's get the WAIC for each random effects model and compare with the first models without random effects

```
get_waic(m2_macaque)
```

```
##
## Computed from 8000 by 8400 log-likelihood matrix
##
##           Estimate    SE
## elpd_waic  -3356.6  57.1
## p_waic        33.0   0.8
## waic        6713.2 114.2
```

```
get_waic(m2_chip)
```

```
##
## Computed from 8000 by 5400 log-likelihood matrix
##
##           Estimate   SE
## elpd_waic  -3359.9 24.1
## p_waic        19.6  0.3
## waic        6719.7 48.3
```

```
get_waic(m2_gor)
```

```
##
## Computed from 8000 by 8100 log-likelihood matrix
##
##           Estimate   SE
## elpd_waic  -4396.1 40.1
## p_waic        29.2  0.4
## waic        8792.2 80.2
```

Below I just copied the result of the WAIC into a data frame to create the tables. Of course this process could be automated.

Table 4.5: Comparison of the WAIC of the Bradley-Terry model and the Bradley-Terry model with random effects on the subjects for each specie

	WAIC	
Specie	Bradley-Terry	Bradley-Terry with random effects
Macaques	7101.6	6713.2
Chimpanzees	7199.5	6719.7
Gorillas	9767.4	8792.2

```
waic_table <-
  data.frame(
    Species = c('Macaques', 'Chimpanzees', 'Gorillas'),
    BT = c(7101.6, 7199.5, 9767.4),
    BTU = c(6713.2, 6719.7, 8792.2)
  )
```

The kableExtra package provides some nice tools to create tables from R directly to Latex

```
kable(
  waic_table,  booktabs=T,
  caption = 'Comparison of the WAIC of the Bradley-Terry model and the Bradley-Terry model with r
  col.names = c('Specie', 'Bradley-Terry', 'Bradley-Terry with random effects')
) %>%
  kableExtra::add_header_above(c(" " = 1, "WAIC" = 2)) %>%
  kableExtra::kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"))
  kableExtra::scroll_box(width = "100%")
```

We can see that the random effects model perform much better than the simple BT model by having a much lower WAIC. Therefore from now we will use only the random effects model to generate our tables and plots

4.3.2 Parameter tables and plots

Now let's create some plots and tables to analyze and compare the models

4.3.2.1 Parameters table

Creating a nice table of the parameters.

First let's put all species in the same data frame

```
df1 <- get_parameters(m2_macaque, n_eff = F)
df2 <- get_parameters(m2_chip, n_eff = F)
df3 <- get_parameters(m2_gor, n_eff = F)
```

```
df1 <- df1 %>% dplyr::mutate(Species='Macaque')
df2 <- df2 %>% dplyr::mutate(Species='Chimpanzees')
df3 <- df3 %>% dplyr::mutate(Species='Gorilla')

#appending the dataframes
df <- rbind(df1,df2,df3)

#Removing the individual random effects parameters otherwise the table will be much bi
df <- df %>%
  filter(!startsWith(Parameter,'U1['))

#Re-coding the parameters (removing the lambda) so it reads nicer in the table.
df$Parameter <- dplyr::recode(df$Parameter, 'lambda[Carrot]'='Carrot')
df$Parameter <- dplyr::recode(df$Parameter, 'lambda[Grape]'='Grape')
df$Parameter <- dplyr::recode(df$Parameter, 'lambda[Turnip]'='Turnip')
df$Parameter <- dplyr::recode(df$Parameter, 'lambda[Cucumber]'='Cucumber')
df$Parameter <- dplyr::recode(df$Parameter, 'lambda[Tomato]'='Tomato')
df$Parameter <- dplyr::recode(df$Parameter, 'lambda[Apple]'='Apple')
df$Parameter <- dplyr::recode(df$Parameter, 'lambda[Jungle Pellet]'='Jungle Pellet')
df$Parameter <- dplyr::recode(df$Parameter, 'lambda[Green Beans]'='Green Beans')
df$Parameter <- dplyr::recode(df$Parameter, 'lambda[Celery]'='Celery')
df$Parameter <- dplyr::recode(df$Parameter, 'lambda[Oats]'='Oats')
df$Parameter <- dplyr::recode(df$Parameter, 'lambda[Peanuts]'='Peanuts')
```

Now let's create the table by removing the species column and adding some row Headers for the species

```
(df %>% select(-Species) %>%
kable( caption = 'Parameters of the model with random effects', digits = 2, col.names =
kableExtra::kable_styling(bootstrap_options = c("striped", "hover", "condensed", "respo
  kableExtra::pack_rows("Macaque", 1, 6) %>%
  kableExtra::pack_rows("Chimpanzees", 7, 13) %>%
  kableExtra::pack_rows("Gorilla", 14, 20) %>%
  kableExtra::scroll_box(width = "100%")
)
```

4.3.2.2 Rank table

Rank of the food preferences

```
rank1 <- get_rank_of_players_df(m2_macaque)
rank2 <- get_rank_of_players_df(m2_chip)
rank3 <- get_rank_of_players_df(m2_gor)

#appending the dataframes
rank_all <- rbind(rank1,rank2,rank3)
```

Table 4.6: Parameters of the model with random effects

Parameter	Mean	HPD lower	HPD upper
Macaque			
Carrot	0.12	-0.81	1.02
Celery	-2.28	-3.16	-1.34
Jungle Pellet	1.24	0.35	2.14
Oats	-0.90	-1.85	-0.02
Green Beans	-0.17	-1.04	0.77
U1_std	0.57	0.42	0.75
Chimpanzees			
Apple	0.02	-0.97	1.06
Tomato	0.32	-0.71	1.30
Carrot	-0.22	-1.26	0.76
Grape	0.60	-0.39	1.59
Cucumber	-0.34	-1.33	0.63
Turnip	-0.44	-1.44	0.56
U1_std	0.72	0.48	1.00
Gorilla			
Apple	0.03	-0.98	0.97
Carrot	-0.11	-1.12	0.83
Grape	0.86	-0.11	1.87
Tomato	0.85	-0.10	1.84
Cucumber	-0.71	-1.66	0.33
Turnip	-0.93	-1.86	0.08
U1_std	0.79	0.57	1.02

```
#Re-coding the parameters (removing the lambda) so it reads nicer in the table.
rank_all$Parameter <- dplyr::recode(rank_all$Parameter, 'lambda[Carrot]'='Carrot')
rank_all$Parameter <- dplyr::recode(rank_all$Parameter, 'lambda[Grape]'='Grape')
rank_all$Parameter <- dplyr::recode(rank_all$Parameter, 'lambda[Turnip]'='Turnip')
rank_all$Parameter <- dplyr::recode(rank_all$Parameter, 'lambda[Cucumber]'='Cucumber')
rank_all$Parameter <- dplyr::recode(rank_all$Parameter, 'lambda[Tomato]'='Tomato')
rank_all$Parameter <- dplyr::recode(rank_all$Parameter, 'lambda[Apple]'='Apple')
rank_all$Parameter <- dplyr::recode(rank_all$Parameter, 'lambda[Jungle Pellet]'='Jungl
rank_all$Parameter <- dplyr::recode(rank_all$Parameter, 'lambda[Green Beans]'='Green B
rank_all$Parameter <- dplyr::recode(rank_all$Parameter, 'lambda[Celery]'='Celery')
rank_all$Parameter <- dplyr::recode(rank_all$Parameter, 'lambda[Oats]'='Oats')
rank_all$Parameter <- dplyr::recode(rank_all$Parameter, 'lambda[Peanuts]'='Peanuts')
```

Now let's create the rank table

```
(rank_all %>%
kable( caption = 'Ranking of the food preferences per specie with model with random ef
kableExtra::kable_styling(bootstrap_options = c("striped", "hover", "condensed", "resp
  kableExtra::pack_rows("Macaque", 1, 6) %>%
  kableExtra::pack_rows("Chimpanzees", 7, 12) %>%
  kableExtra::pack_rows("Gorilla", 13, 18) %>%
  kableExtra::scroll_box(width = "100%")
)
```

4.3.2.3 Plot

Now let's use ggplot to create a plot comparing both types of model. The simple BT and the BT with Random effects

First we need to prepare the data frames for plotting. For the simple BT model (called old)

```
df1_old <- get_parameters(m1_macaque, n_eff = F)
df2_old <- get_parameters(m1_chip, n_eff = F)
df3_old <- get_parameters(m1_gor, n_eff = F)

df1_old <- df1_old %>% dplyr::mutate(Species='Macaque', Model='Simple')
df2_old <- df2_old %>% dplyr::mutate(Species='Chimpanzees', Model='Simple')
df3_old <- df3_old %>% dplyr::mutate(Species='Gorilla', Model='Simple')

#appending the dataframes
df_old <- rbind(df1_old,df2_old,df3_old)

#Removing the individual random effects parameters
df_old <- df_old %>%
  filter(!startsWith(Parameter,'U'))
```

Table 4.7: Ranking of the food preferences per specie with model with random effects

Food	Median Rank	Mean Rank	Std. Rank
Macaque			
Peanuts	1	1.01	0.10
Jungle Pellet	2	1.99	0.11
Carrot	3	3.16	0.37
Green Beans	4	3.85	0.38
Oats	5	4.99	0.10
Celery	6	6.00	0.00
Chimpanzees			
Grape	1	1.53	0.85
Tomato	2	2.27	1.12
Apple	3	3.26	1.25
Carrot	4	4.24	1.28
Cucumber	5	4.70	1.18
Turnip	5	5.00	1.11
Gorilla			
Grape	2	1.55	0.60
Tomato	2	1.57	0.61
Apple	3	3.38	0.74
Carrot	4	3.72	0.72
Cucumber	5	5.12	0.70
Turnip	6	5.66	0.56

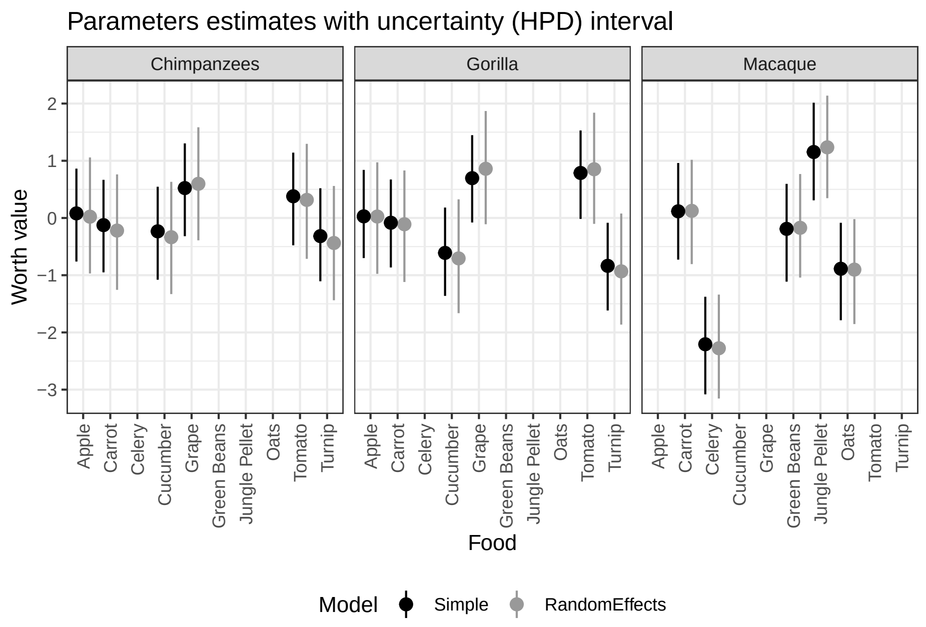
```
#Re-coding some parameters so it reads nicer in the figure
df_old$Parameter <- dplyr::recode(df_old$Parameter, 'lambda[Carrot]'='Carrot')
df_old$Parameter <- dplyr::recode(df_old$Parameter, 'lambda[Grape]'='Grape')
df_old$Parameter <- dplyr::recode(df_old$Parameter, 'lambda[Turnip]'='Turnip')
df_old$Parameter <- dplyr::recode(df_old$Parameter, 'lambda[Cucumber]'='Cucumber')
df_old$Parameter <- dplyr::recode(df_old$Parameter, 'lambda[Tomato]'='Tomato')
df_old$Parameter <- dplyr::recode(df_old$Parameter, 'lambda[Apple]'='Apple')
df_old$Parameter <- dplyr::recode(df_old$Parameter, 'lambda[Jungle Pellet]'='Jungle Pel
df_old$Parameter <- dplyr::recode(df_old$Parameter, 'lambda[Green Beans]'='Green Beans
df_old$Parameter <- dplyr::recode(df_old$Parameter, 'lambda[Celery]'='Celery')
df_old$Parameter <- dplyr::recode(df_old$Parameter, 'lambda[Oats]'='Oats')
df_old$Parameter <- dplyr::recode(df_old$Parameter, 'lambda[Peanuts]'='Peanuts')
```

For the BT with random effects

```
df1 <- get_parameters(m2_macaque, n_eff = F)
df2 <- get_parameters(m2_chip, n_eff = F)
df3 <- get_parameters(m2_gor, n_eff = F)

df1 <- df1 %>% dplyr::mutate(Species='Macaque', Model='RandomEffects')
df2 <- df2 %>% dplyr::mutate(Species='Chimpanzees', Model='RandomEffects')
df3 <- df3 %>% dplyr::mutate(Species='Gorilla', Model='RandomEffects')

#appending the dataframes
df <- rbind(df1,df2,df3)

#Removing the individual random effects parameters
df <- df %>%
  filter(!startsWith(Parameter,'U'))

#Re-coding some parameters so it reads nicer in the figure
df$Parameter <- dplyr::recode(df$Parameter, 'lambda[Carrot]'='Carrot')
df$Parameter <- dplyr::recode(df$Parameter, 'lambda[Grape]'='Grape')
df$Parameter <- dplyr::recode(df$Parameter, 'lambda[Turnip]'='Turnip')
df$Parameter <- dplyr::recode(df$Parameter, 'lambda[Cucumber]'='Cucumber')
df$Parameter <- dplyr::recode(df$Parameter, 'lambda[Tomato]'='Tomato')
df$Parameter <- dplyr::recode(df$Parameter, 'lambda[Apple]'='Apple')
df$Parameter <- dplyr::recode(df$Parameter, 'lambda[Jungle Pellet]'='Jungle Pellet')
df$Parameter <- dplyr::recode(df$Parameter, 'lambda[Green Beans]'='Green Beans')
df$Parameter <- dplyr::recode(df$Parameter, 'lambda[Celery]'='Celery')
df$Parameter <- dplyr::recode(df$Parameter, 'lambda[Oats]'='Oats')
df$Parameter <- dplyr::recode(df$Parameter, 'lambda[Peanuts]'='Peanuts')
```

Now we can merge them in a single data frame for ggplot

```
#appending the dataframes
out <- rbind(df, df_old)
#To order in ggplot we need to specify the order in the levels. We want to place the first model
out$Model<-factor(out$Model, levels=c('Simple','RandomEffects'))

# Defining a black-gray palette:
cbp1 <- c("#000000", "#999999")

#Using the pointrange function to define the HPD intervals
ggplot(out, aes(x=Parameter))+
  geom_pointrange(aes(
        ymin = HPD_lower,
        ymax = HPD_higher,
        y = Mean,
        group=Model,
        color=Model),
           position=position_dodge(width=1))+ #separating the two models (so they are not plotted
  labs(title = 'Parameters estimates with uncertainty (HPD) interval',
       y = 'Worth value',
        x = 'Food')+
  facet_grid(~Species) + #Dividing the plot into three by species
  theme_bw()+ # A black and white theme
  # scale_x_discrete(guide = guide_axis(n.dodge = 2)) +
  theme(legend.position="bottom")+
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))+#small adjustments to the
  scale_colour_manual(values = cbp1) #applying the color palette
```



4.3.2.4 Probability of selecting a novel stimuli

We will create a the table of the predictions of selecting a novel stimuli (compared to the trained ones). This is replication of table II of the original paper (of course the results are not the same since we are using different models and estimation values)

For that, we first create a data frame of the new predictions for each species. Since our model uses random effects and we would need to specify each random effect to make the predictions we will do something slightly different. We will consider that the random effects will be zero, that is, which is equivalent to the average value of the random effects (remember that it has a mean of zero). One way to achieve this is by using the obtained coefficients in a submodel. This can be done in the bpcs package by using the model_type option.

We ask for the data frame instead of the table because we will assemble the table manually.

```
# Create a data frame with all the pairs of food that we want to calculate
pairs_gor <-
  data.frame(
    image1 = c(
      'Apple',
      'Apple',
      'Apple',
      'Apple',
      'Tomato',
      'Tomato',
      'Tomato',
      'Tomato'
    ),
    image2 = c(
      'Cucumber',
      'Grape',
      'Turnip',
      'Carrot',
      'Cucumber',
      'Grape',
      'Turnip',
      'Carrot'
    )
  )

pairs_chip <-
  data.frame(
    image1 = c(
      'Apple',
      'Apple',
      'Apple',
      'Apple',
      'Tomato',
      'Tomato',
      'Tomato',
      'Tomato'
    ),
    image2 = c(
      'Cucumber',
      'Grape',
      'Turnip',
      'Carrot',
      'Cucumber',
      'Grape',
      'Turnip',
      'Carrot'
    )
  )


pairs_macaque <-
  data.frame(
    image1 = c(
      'Oats',
      'Oats',
      'Oats',
      'Oats',
      'Green Beans',
      'Green Beans',
      'Green Beans',
      'Green Beans'
    ),
    image2 = c(
      'Celery',
      'Jungle Pellet',
      'Peanuts',
      'Carrot',
      'Celery',
      'Jungle Pellet',
      'Peanuts',
      'Carrot'
    )
  )


prob_chip <-
  get_probabilities_df(
    m2_chip,
    newdata = pairs_chip,
    model_type = 'bt') #here we are assuming zero for the random effects

prob_macaque <-
  get_probabilities_df(
    m2_macaque,
    newdata = pairs_macaque,
    model_type = 'bt')

prob_gor <-
  get_probabilities_df(
    m2_gor,
    newdata = pairs_gor,
    model_type = 'bt')

#merging in a single df
prob_table <- rbind(prob_gor, prob_chip, prob_macaque)

#since this is  a time consuming table to make let's also save it
saveRDS(prob_table, 'prob_table.RDS')
```

Now we can create the table

```
prob_table <- prob_table %>%
  mutate(Probability = i_beats_j,
         OddsRatio = Probability / (1 - Probability))

prob_table %>%
  dplyr::select(i, j, Probability, OddsRatio) %>%
  kable(
    caption = 'Posterior probabilities of the novel stimuli i being selected over the trained sti
    booktabs = T,
    digits = 2,
    col.names = c('Item i', 'Item j', 'Probability', 'Odds Ratio')
  ) %>%
  kableExtra::kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"))
  kableExtra::pack_rows("Gorilla", 1, 8) %>%
  kableExtra::pack_rows("Chimpanzee", 9, 16) %>%
  kableExtra::pack_rows("Macaque", 17, 24) %>%
  kableExtra::scroll_box(width = "100%")
```

Table 4.8: Posterior probabilities of the novel stimuli i being selected over the trained stimuli j

Item i	Item j	Probability	Odds Ratio
Gorilla			
Apple	Cucumber	0.78	3.55
Apple	Grape	0.35	0.54
Apple	Turnip	0.75	3.00
Apple	Carrot	0.57	1.33
Tomato	Cucumber	0.80	4.00
Tomato	Grape	0.48	0.92
Tomato	Turnip	0.79	3.76
Tomato	Carrot	0.69	2.23
Chimpanzee			
Apple	Cucumber	0.64	1.78
Apple	Grape	0.42	0.72
Apple	Turnip	0.55	1.22
Apple	Carrot	0.51	1.04
Tomato	Cucumber	0.54	1.17
Tomato	Grape	0.37	0.59
Tomato	Turnip	0.69	2.23
Tomato	Carrot	0.64	1.78
Macaque			
Oats	Celery	0.76	3.17
Oats	Jungle Pellet	0.11	0.12
Oats	Peanuts	0.06	0.06
Oats	Carrot	0.23	0.30
Green Beans	Celery	0.88	7.33
Green Beans	Jungle Pellet	0.26	0.35
Green Beans	Peanuts	0.10	0.11
Green Beans	Carrot	0.45	0.82

Chapter 5

Re-analysis 3

This re-analysis is from the study

> Marton, Giulia, et al. "Patients' health locus of control and preferences about the role that they want to play in the medical decision-making process." Psychology, Health & Medicine (2020): 1-7

```
library(bpcs)
library(tidyverse)
library(knitr)
library(loo)
options(mc.cores = parallel::detectCores())
rstan::rstan_options(auto_write = TRUE)
set.seed(99)
```

5.1 Importing the data

The data from this paper was made available upon request and below we exemplify a few rows of how the original dataset looks like. Let's starting importing the data.

```
d<-read.table("data/MHLC.txt", sep="\t", header=T)
d<-as.data.frame(d)
sample_n(d, size=5) %>%
  kable(caption = 'Sample of rows from the original data') %>%
  kableExtra::kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"))
  kableExtra::scroll_box(width = "100%")
```

As we can see, the data is in a wide format. Before we pivot it to longer let's add a column that indicates the subject ID.

Table 5.1: Sample of rows from the original data

AB	AC	BC	AD	BD	CD	AE	BE	CE	DE	GENDER	MHLC_INTERNAL	MHI
1	1	1	1	1	1	1	1	1	1	2	16	
1	1	1	1	1	0	0	0	0	0	2	23	
1	1	1	0	0	0	0	0	0	0	1	19	
1	1	1	1	1	0	1	0	0	0	2	19	
1	1	1	1	0	0	0	0	0	0	1	14	

```
d <- d %>% mutate(SubjectID = row_number())
```

Now let's pivot it to the longer format

```
cols_to_pivot<-colnames(d)[1:10]
d_longer<-tidyr::pivot_longer(d, cols=all_of(cols_to_pivot), names_to='comparison', val
```

Now let's divide the comparison into two vectors (choice0 and choice1). So it fits the bpcs format

```
comp_cols <- str_split_fixed(d_longer$comparison, "", 2)
d_longer$choice0 <- comp_cols[,1]
d_longer$choice1 <- comp_cols[,2]
```

The data frame now looks like this:

```
dplyr::sample_n(d_longer, size=10) %>%
  kable() %>%
  kableExtra::kable_styling(bootstrap_options = c("striped", "hover", "condensed", "res
  kableExtra::scroll_box(width = "100%")
```

GENDER	MHLC_INTERNAL	MHLC_CHANCE	MHLC_DOCTORS	MHLC_OTHER_PE
1	22	6	15	
2	20	17	8	
2	30	30	15	
2	23	8	14	
2	23	18	13	
1	20	17	14	
2	21	23	13	
2	20	23	8	
2	19	19	17	
2	33	31	8	

Now that we have setup the data frame correctly for the bpcs package, we can use it to model the problem.

Let's just rename a few values so it is easier to understand.

```
#choice0
d_longer$choice0 <- recode(d_longer$choice0, 'A'='Active')
d_longer$choice0 <- recode(d_longer$choice0, 'B'='Active-Collaborative')
d_longer$choice0 <- recode(d_longer$choice0, 'C'='Collaborative')
d_longer$choice0 <- recode(d_longer$choice0, 'D'='Passive-Collaborative')
d_longer$choice0 <- recode(d_longer$choice0, 'E'='Passive')
#choice1
d_longer$choice1 <- recode(d_longer$choice1, 'A'='Active')
d_longer$choice1 <- recode(d_longer$choice1, 'B'='Active-Collaborative')
d_longer$choice1 <- recode(d_longer$choice1, 'C'='Collaborative')
d_longer$choice1 <- recode(d_longer$choice1, 'D'='Passive-Collaborative')
d_longer$choice1 <- recode(d_longer$choice1, 'E'='Passive')
```

Our final step in the preparation of the dataset is to standardize the values of the MHLOC scales. The values provided in the dataset correspond to the sum of the values of the scale and ranges from 3-18 or from 6-36. Therefore we will scale them accordingly for more stable inference

```
d_longer <- d_longer %>%
  mutate(Internal = scale(MHLC_INTERNAL),
         Chance = scale(MHLC_CHANCE),
         Doctors = scale(MHLC_DOCTORS),
         OtherPeople = scale(MHLC_OTHER_PEOPLE))
```

The final modified dataset looks like:

```
sample_n(d_longer, size=10) %>%
  kable() %>%
  kableExtra::kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"))
  kableExtra::scroll_box(width = "100%")
```

GENDER	MHLC_INTERNAL	MHLC_CHANCE	MHLC_DOCTORS	MHLC_OTHER_PEOPLE	Ag
2	18	11	9	7	5
1	18	31	15	8	4
2	15	17	15	9	2
1	18	31	15	8	4
2	23	18	13	11	5
1	21	13	11	7	5
2	25	14	14	10	6
2	30	14	10	9	6
2	15	15	12	8	3
1	15	14	9	6	N

Now we can proceed with the analysis.

5.2 Simple Bradley-Terry models

Let's start with a simple BT without considering any subject predictors. Just to evaluate the average probability of people being Active, Active-Collaborative, Collaborative, Passive-Collaborative or Passive.

```r
m1 <-
  bpc(
    d_longer,
    player0 = 'choice0',
    player1 = 'choice1',
    result_column = 'y',
    model_type = 'bt',
    priors = list(prior_lambda_std = 1.0),
    iter = 3000
  )
save_bpc_model(m1, 'm_hloc', 'fittedmodels')
```

Let's investigate model convergence with shinystan. Everything seems fine.

```r
launch_shinystan(m1)
```

Now let's get the waic

```r
m1_waic <- get_waic(m1)
m1_waic
```

```
##
## Computed from 8000 by 1530 log-likelihood matrix
##
##           Estimate   SE
## elpd_waic   -711.2 21.9
## p_waic         3.9  0.2
## waic        1422.4 43.9
```

5.3 Subject predictors model

We have different HLC that can be used as predictors for the response. Let's create a single model with all the predictors.

```r
m2 <-
  bpc(
    d_longer,
    player0 = 'choice0',
    player1 = 'choice1',
    result_column = 'y',
    subject_predictors = c('Internal', 'Chance', 'Doctors', 'OtherPeople'),
    model_type = 'bt-subjectpredictors',
    priors = list(prior_lambda_std = 1.0,
                  prior_S_std = 1.0),
    iter = 3000
  )
save_bpc_model(m2, 'm_mhloc_subjectpred', 'fittedmodels')
```

Diagnostics

```
launch_shinystan(m2)
```

```
m2_waic <- get_waic(m2)
```

5.4 Subject predictors and random effects

Now let's create a third model that also compensate for multiple judgment with a random effects variable.

This model adds 153*4 variables, so sampling will take longer and be a bit more complex.

```
m3 <-
  bpc(
    d_longer,
    player0 = 'choice0',
    player1 = 'choice1',
    result_column = 'y',
    subject_predictors = c('Internal', 'Chance', 'Doctors', 'OtherPeople'),
    cluster = c('SubjectID'),
    model_type = 'bt-subjectpredictors-U',
    priors = list(prior_lambda_std = 1.0,
                  prior_S_std = 1.0,
                  prior_U1_std = 1.0),
    iter = 3000
  )
save_bpc_model(m3, 'm_mhloc_subjectpred_U', 'fittedmodels')
```

Diagnostics

```
launch_shinystan(m3)
```

```
m3_waic <- get_waic(m3)
```

5.5 Comparing the WAIC

Let's compare the WAIC of the three models

```
loo::loo_compare(m1_waic,m2_waic, m3_waic)
```

```
##        elpd_diff se_diff
## model3    0.0       0.0
## model2 -226.7      14.8
## model1 -248.8      16.1
```

5.6 Plots and tables

Now that we see that all models have proper convergence and that the model m3 has a better fit we will generate some plots and tables to help understand the problem

```
lambda <- get_parameters(m3, params = 'lambda', n_eff = F)
Spar <- get_parameters(m3, params = 'S', n_eff = F)
U_std <- get_parameters(m3, params = 'U1_std', n_eff = F)
```

Let's create a custom table based on these parameters. But first we will rename the parameters a bit so the table reads a bit better. We are here removing the S and the lambda from the parameter

```
lambda$Parameter <-  stringr::str_sub(lambda$Parameter,start = 8, end=-2)
Spar$Parameter <- stringr::str_sub(Spar$Parameter,start = 3, end=-2)
```

5.6.1 Table lambda and U

Now we can create the table

```
#merge the datasets
df_table <- rbind(lambda, U_std)
#creating the table
df_table %>%
  kable(caption = 'Lambda parameters of the model and the random effects standard devi
        digits = 2,
        col.names = c('Parameter', 'Mean','HPD lower', 'HPD lower')) %>%
  kableExtra::kable_styling(bootstrap_options = c("striped", "hover", "condensed", "res
  kableExtra::scroll_box(width = "100%")
```

5.6.2 Subject predictors table

```
S <- Spar %>% tidyr::separate(Parameter,c('Role','MHLOC'), sep=",")
S$MHLOC <- recode(S$MHLOC, 'OtherPeople'= 'Other people')
```

```
S %>%
  dplyr::arrange(Role) %>%
  select(-Role) %>%
```

Table 5.2: Lambda parameters of the model and the random effects standard deviation

Parameter	Mean	HPD lower	HPD lower
Active	-2.11	-3.04	-1.08
Active-Collaborative	1.40	0.44	2.38
Collaborative	3.21	2.08	4.21
Passive-Collaborative	0.85	-0.07	1.84
Passive	-3.32	-4.43	-2.34
U1_std	2.32	1.89	2.78

```
kable(format='html',
      caption = 'Subject predictors parameters by role',
      digits = 2,
      col.names = c('Parameter', 'Mean','HPD lower', 'HPD lower'),
      booktabs=T) %>%
kableExtra::kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"))
kableExtra::pack_rows("Active", 1, 4) %>%
kableExtra::pack_rows("Active-Collaborative", 5, 8) %>%
kableExtra::pack_rows("Collaborative", 9, 12) %>%
kableExtra::pack_rows("Passive-Collaborative", 13, 16) %>%
kableExtra::pack_rows("Passive", 17, 20) %>%
kableExtra::scroll_box(width = "100%")
```

Subject predictors parameters by role

Parameter

Mean

HPD lower

HPD lower

Active

Internal

-0.09

-1.11

0.85

Chance

-0.11

-1.14

0.86

Doctors

0.17

-1.50

0.50

Other people

0.57

-1.19

0.80

Active-Collaborative

Internal

-0.07

-0.97

0.98

Chance

-0.48

-1.10

0.86

Doctors

-0.74

-1.44

0.48

Other people

0.10

-1.30

0.71

Collaborative

Internal

-0.51

-1.03

0.93

Chance

-0.31

-0.90

1.09

Doctors

0.03

-0.78

1.19

Other people

0.14

-1.76

0.33

Passive-Collaborative

Internal

-0.01

-0.88

1.14

Chance

0.11

-0.84

1.15

Doctors

0.27

-0.51

1.48

Other people

0.69

-0.35

1.72

Passive

Internal

-0.18

-0.97

0.97

Chance

-0.05

-1.07

0.88

Doctors

-0.06

-0.72

1.23

Other people

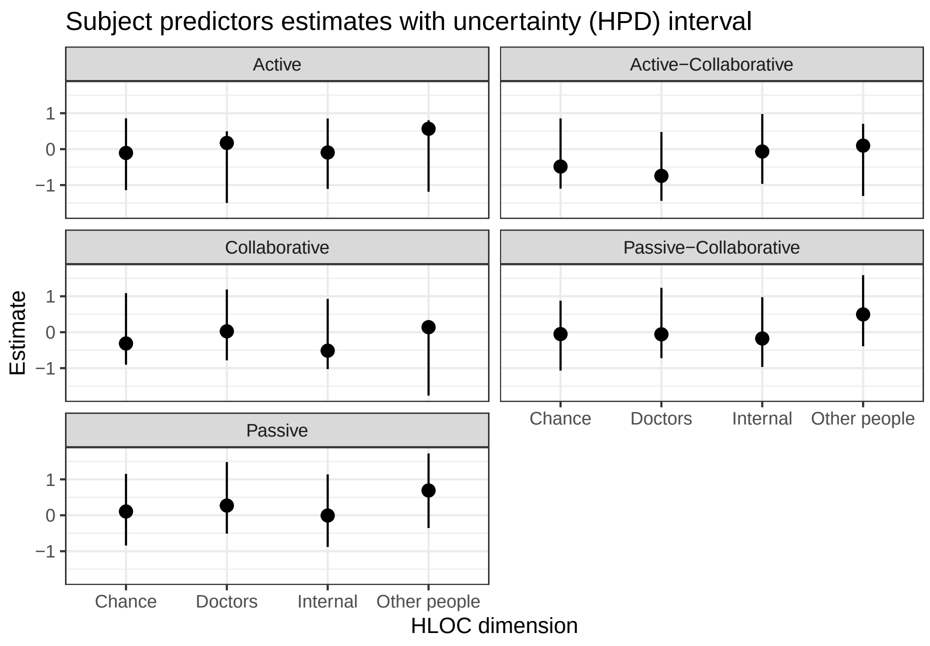
0.49

-0.39

1.59

5.6.3 Plot

```
S$Role <- factor(S$Role, levels = c('Active', 'Active-Collaborative', 'Collaborative',
ggplot(S, aes(x=MHLOC))+
  geom_pointrange(aes(
        ymin = HPD_lower,
        ymax = HPD_higher,
        y = Mean,
        group=MHLOC))+
  facet_wrap(~Role, nrow = 3) + #Dividing the plot into three by species
  labs(title = 'Subject predictors estimates with uncertainty (HPD) interval',
       y = 'Estimate',
        x = 'HLOC dimension')+
  theme_bw()+ # A black and white theme
  # scale_x_discrete(guide = guide_axis(n.dodge = 2)) +
  theme(legend.position="bottom") #small adjustments to the theme
```



5.6.4 Probability tables

First let's create a data frame with the cases we want to investigate. We will utilize the model_type option in the probabilities table so we can average out the values of the random effects in the subjects. In this table, we will only investigate a few of the conditions, but of course it is possible to do a much more expansive analysis

Let's create a new data frame with the conditions we want to investigate. Mainly we are just investigating: * Between the choice of Active and Passive how does going from -2 to 2 standard deviations over the mean influences the probability in the Internal and the Doctors * Between the choice of Active-Collaborative and Collaborative how does going from -2 to 2 standard deviations over the mean influences the probability in the Internal and the Doctors * Between the choice of Collaborative and Passive-Collaborative how does going from -2 to 2 standard deviations over the mean influences the probability in the Internal and the Doctors

```
newdata <- tibble::tribble(~choice0, ~choice1, ~Internal, ~Chance, ~Doctors, ~OtherPeople,
                           "Active", "Passive", 0, 0, 0, 0,
                           "Active", "Passive", -2, 0, 0, 0,
                           "Active", "Passive", 2, 0, 0, 0,
                           "Active", "Passive", 0, 0, -2, 0,
                           "Active", "Passive", 0, 0, 2, 0,
                           "Active-Collaborative", "Collaborative", 0, 0, 0, 0,
                           "Active-Collaborative", "Collaborative", -2, 0, 0, 0,
                           "Active-Collaborative", "Collaborative", 2, 0, 0, 0,
                           "Active-Collaborative", "Collaborative", 0, 0, -2, 0,
                           "Active-Collaborative", "Collaborative", 0, 0, 2, 0,
                           "Collaborative", "Passive-Collaborative", 0, 0, 0, 0,
                           "Collaborative", "Passive-Collaborative", 0, -2, 0, 0,
                           "Collaborative", "Passive-Collaborative", 0, 2, 0, 0,
                           "Collaborative", "Passive-Collaborative", 0, 0, 0, -2,
                           "Collaborative", "Passive-Collaborative", 0, 0, 0, 2
                           )

#Now we can calculate the probabilities
prob_hloc <-
  get_probabilities_df(
    m3,
    newdata = newdata,
    model_type = 'bt-subjectpredictors') #here we are assuming zero for the random eff
```

```
prob_hloc %>%
  select(-j_beats_i) %>%
  rename(Probability=i_beats_j) %>%
  kable(caption = "Probabilities of selecting role i instead of j based on changes of t
      digits = 2,
      booktabs = T) %>%
  kableExtra::kable_styling(bootstrap_options = c("striped", "hover", "condensed", "res
  kableExtra::add_header_above(c("Roles" = 2, "HLOC dimensions" = 4, " "=1)) %>%
  kableExtra::scroll_box(width = "100%")
```

Table 5.3: Probabilities of selecting role i instead of j based on changes of the values of the HLOC dimensions

Roles		HLOC dimensions				
i	j	Internal	Chance	Doctors	OtherPeople	Probability
Active	Passive	0	0	0	0	0.85
Active	Passive	-2	0	0	0	0.80
Active	Passive	2	0	0	0	0.71
Active	Passive	0	0	-2	0	0.96
Active	Passive	0	0	2	0	0.31
Active-Collaborative	Collaborative	0	0	0	0	0.14
Active-Collaborative	Collaborative	-2	0	0	0	0.21
Active-Collaborative	Collaborative	2	0	0	0	0.25
Active-Collaborative	Collaborative	0	0	-2	0	0.39
Active-Collaborative	Collaborative	0	0	2	0	0.10
Collaborative	Passive-Collaborative	0	0	0	0	0.86
Collaborative	Passive-Collaborative	0	-2	0	0	0.84
Collaborative	Passive-Collaborative	0	2	0	0	0.94
Collaborative	Passive-Collaborative	0	0	0	-2	0.99
Collaborative	Passive-Collaborative	0	0	0	2	0.43