R codes (with simulated data) for the Bayesian sequential updating and area calculation between posterior probability sequences

Patricia Gilholm

This document outlines the R code used for the Bayesian sequential updating and area between posterior probability sequences as described in the paper "Identifying latent subgroups of children with developmental delay using Bayesian sequential updating and Dirichlet process mixture modelling". The data used in this document is a simulated data set for demonstration, and therefore the results in this document will not be the same as that of the paper. Please refer to the paper for instructions on how to access the data used in the paper. This walkthrough also demonstrates how to post process the MCMC chains to obtain the optimal clusters. The code to run the DPMM is not presented here, however a link to access the code is provided.

```
#libraries
library(tidyverse)
library(logitnorm)
library(RcmdrPlugin.KMggplot2)
library(coda)
library(mcclust)
library(alluvial)
library(cluster)
library(reshape2)
```

The simulated data is located in a csv file labelled simulated_data.csv. There are 4 columns in the data:

- participant_id: 100 participants labelled 1 through 100, and a reference participant labelled "ref".
- dimension: 2 dimensions labelled A and B.
- timepoint: 20 timepoints labelled 1 through 20.
- status: A binary variable indicating success = 1 or failure = 0 for the observation at each timepoint. The probably of success, θ, for each participant and each dimension were randomly drawn from a uniform distribution, $\theta \sim U[0,1]$.

```
data<-read_csv("simulated_data.csv", col_types =list(col_factor(), col_factor(),
                                                     col_double(), col_double()))
data$participant_id<-as.factor(data$participant_id)
data$dimension<-as.factor(data$dimension)
head(data, n=10)
```

```
## # A tibble: 10 x 4
##    participant_id dimension timepoint status
##    <fct>          <fct>         <dbl>  <dbl>
##  1 1              A                 1      0
##  2 1              A                 2      0
##  3 1              A                 3      0
```

```
##  4 1              A                 4      0
##  5 1              A                 5      1
##  6 1              A                 6      0
##  7 1              A                 7      0
##  8 1              A                 8      0
##  9 1              A                 9      0
## 10 1              A                10      0
```

Bayesian sequential updating

The first step of the Bayesian sequential updating is to get the columns z, N, a, b set up for the calculation of the posterior mean.

```
#get z
data_z<- data %>%
         group_by(participant_id, dimension)%>%
         mutate(z = cumsum(status))

#get N
data_N_step_1<- data_z %>%
                ungroup()%>%
                mutate(N = rep(1, length.out = nrow(data_z)))

data_N<- data_N_step_1 %>%
         group_by(participant_id, dimension)%>%
         mutate(N=cumsum(N))
#get a
data_a_step_1 <- data_N %>%
          group_by(participant_id, dimension)%>%
          mutate(a = ifelse(status == 1, 1, 0))
#make sure first a for each functional domain is 1
data_a_step_2 <- data_a_step_1 %>%
  group_by(participant_id, dimension)%>%
  mutate(a = ifelse(timepoint == min(timepoint), 1, a))

data_a<- data_a_step_2 %>%
         group_by(participant_id, dimension)%>%
         mutate(a = cumsum(a))

#get b
data_b_step_1 <- data_a %>%
  group_by(participant_id, dimension)%>%
  mutate(b = ifelse(status == 0, 1, 0))
#make sure first a for each functional domain is 1
data_b_step_2 <- data_b_step_1 %>%
  group_by(participant_id, dimension)%>%
  mutate(b = ifelse(timepoint == min(timepoint), 1, b))

data_b<- data_b_step_2 %>%
  group_by(participant_id, dimension)%>%
  mutate(b = cumsum(b))
```

The next step is to calculate the posterior mean, $\frac{z+a}{N+a+b}$, for each row.

```
data_with_postmean<- data_b %>%
                     mutate(posterior_mean = (z+a)/(N+a+b))
head(data_with_postmean[,c(4:9)], n=10)
```

```
## # A tibble: 10 x 6
##    status     z     N     a     b posterior_mean
##     <dbl> <dbl> <dbl> <dbl> <dbl>          <dbl>
##  1      0     0     1     1     1          0.333
##  2      0     0     2     1     2          0.2
##  3      0     0     3     1     3          0.143
##  4      0     0     4     1     4          0.111
##  5      1     1     5     2     4          0.273
##  6      0     1     6     2     5          0.231
##  7      0     1     7     2     6          0.2
##  8      0     1     8     2     7          0.176
##  9      0     1     9     2     8          0.158
## 10      0     1    10     2     9          0.143
```

The next step is to calculate the upper and lower 95% higest density limits. I used the function in Kruschke's source code from the book "Doing Bayesian Data Analysis". Visit https://sites.google.com/site/doingbayesiandataanalysis/software-installation to obtain the code.

```
source("BernBeta.R")
source("DBDA2E-utilities.R")
```

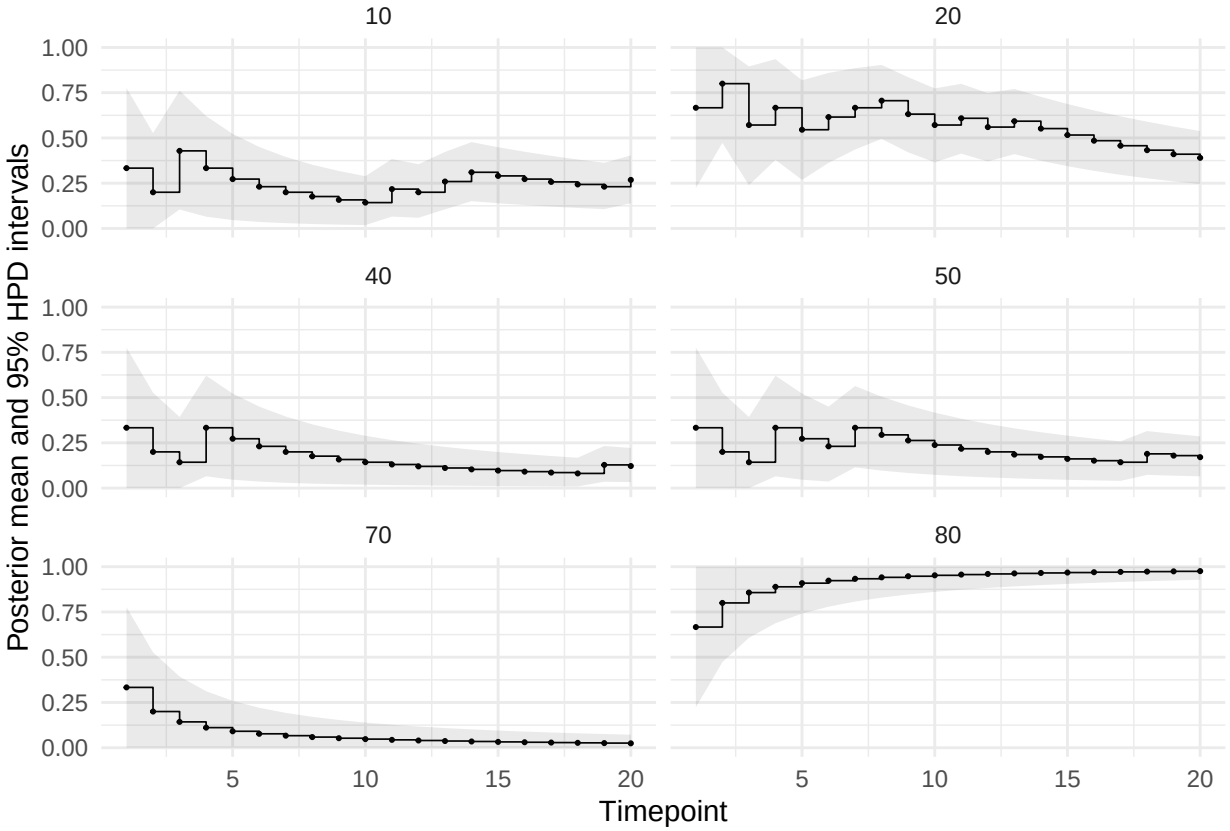
```
##
## ********************************************************************
## Kruschke, J. K. (2015). Doing Bayesian Data Analysis, Second Edition:
## A Tutorial with R, JAGS, and Stan. Academic Press / Elsevier.
## ********************************************************************
```

```
HDI_output<- apply(data_with_postmean[,c("a", "b", "N", "z")], 1,
                   function(x) HDIofICDF(ICDFname=qbeta,shape1=x[1]+x[4],
                                         shape2 = x[2]+x[3]-x[4],
                                         credMass = 0.95, tol=1e-8))

HDI_output<- as.data.frame(HDI_output)

HDI_output<-t(HDI_output)
colnames(HDI_output)<- c("low_HDI", "High_HDI")

HDI_output<-as.data.frame(HDI_output)

data_with_postmean$low_HDI<- HDI_output$low_HDI
data_with_postmean$high_HDI<- HDI_output$High_HDI
```

We can plot some of the sequences to see how the posterior mean gets updated over time. This next section displays the Bayesian sequental updating for participants 10, 20, 40, 50, 70 and 80 in dimension A.

```
#first filter some examples
sub_six<- filter(data_with_postmean, participant_id %in% c(10, 20, 40, 50, 70, 80))
```

```
sub_six_A<- filter(sub_six, dimension=="A")
sub_six_A$participant_id<- as.factor(sub_six_A$participant_id)

#plot six kids
ggplot(data=sub_six_A, aes(x = timepoint, y=posterior_mean, group=participant_id))+
  geom_point(size=0.4)+
  geom_step(size=0.3)+
  geom_ribbon(aes(ymin=low_HDI, ymax= high_HDI), linetype=2, alpha=0.1)+
  facet_wrap(~participant_id, nrow=3, ncol=2)+
  theme_minimal()+
  theme(legend.position="none")+
  labs(x ="Timepoint", y = "Posterior mean and 95% HPD intervals")
```



You can see for some participants the probability of success increases over time, and for others the probability decreases over time. For some participants there is more variability in the responses, which results in wider HDI's. You can also see the 95% highest density interval narrows over time as there is more certainty for the observations over time.

Next we extract the data from the reference participant and join the posterior mean as its own column. The reference participant has status = 1 for every observation.

```
ref_participant<- filter(data_with_postmean, participant_id == "ref")
ref_participant_select_columns<- ref_participant %>%
                            ungroup() %>%
                            dplyr::select(dimension, N, ref_post = posterior_mean)
```

```
data_with_ref<- left_join(data_with_postmean, ref_participant_select_columns)
```

Area calculation

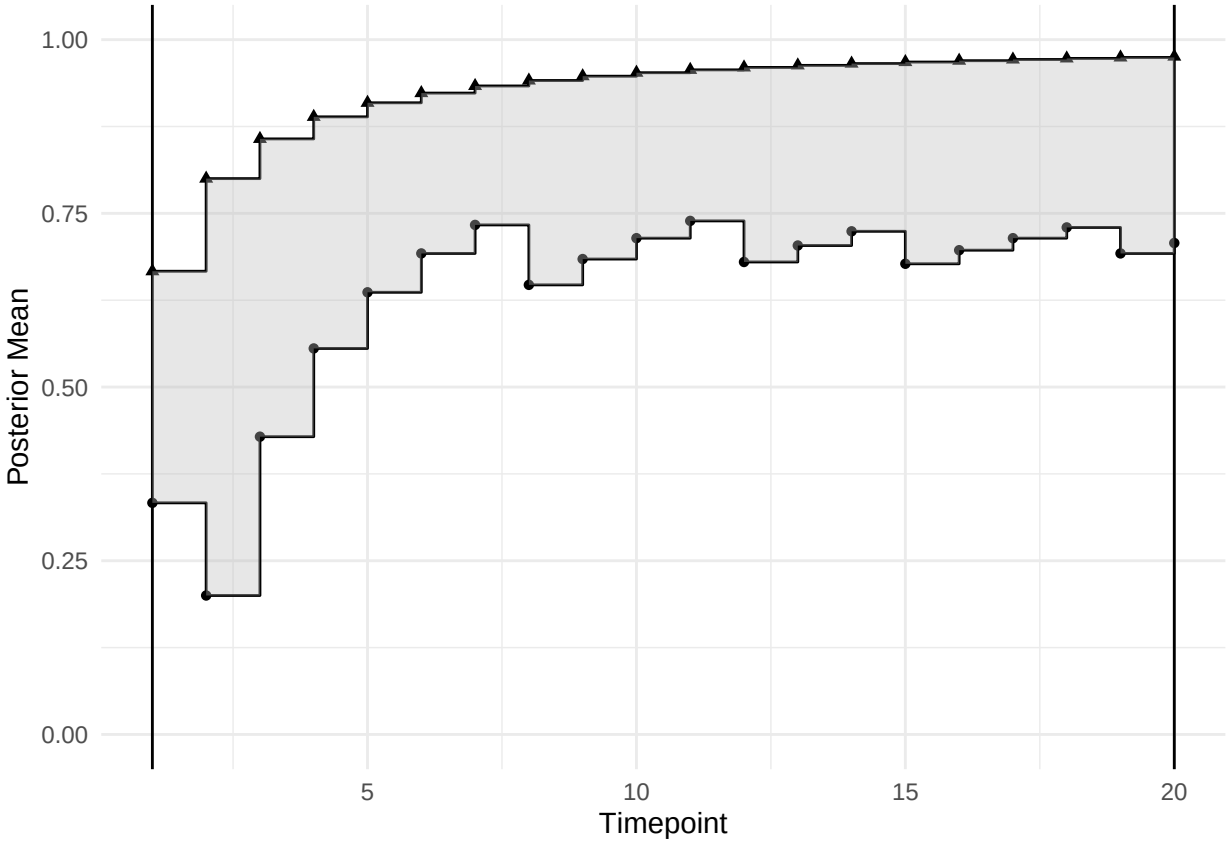
Finally, the last step is to calculate the areas and rescale them. In the simulated data all of the participants have 20 timepoints for every dimension. In the paper this was not the case, so rescaling ensured we could compare sequences of different lengths.

```
#function for areas
area<- function(x, curve_one, curve_two){
  a<-c()
  for(i in 1:length(x)){

    a[i]<-(curve_two[i]- curve_one[i])*(x[i+1] - x[i])
    result<-sum(a, na.rm=T)
  }
  result
}

#calculate absolute areas
data_with_area<-data_with_ref %>%
  group_by(participant_id, dimension)%>%
  mutate(area_curves = area(N, posterior_mean, ref_post))


#rescale areas
rescaled_area<-  data_with_area %>%
  group_by(participant_id, dimension) %>%
  mutate(n_time = n_distinct(timepoint))%>%
  mutate(rescaled_area = area_curves/n_time)
```

Now we have calculated the areas we can plot an example of what we just calculated.

```
example_area<- filter(data_with_postmean, participant_id %in% c("60", "ref"))

example_dim_A<- filter(example_area, dimension == "A")

example_dim_A$participant_id<- as.factor(example_dim_A$participant_id)



example_dim_A<- example_dim_A%>%
                dplyr::group_by(timepoint)%>%
                mutate(ymin = min(posterior_mean),
                       ymax = max(posterior_mean))

#plot area between curves


ggplot(data = example_dim_A, aes(x = timepoint, y = posterior_mean,
                                 group = participant_id))+
```

```
  geom_point(aes(shape= participant_id))+
  geom_step()+
  theme_minimal()+
  geom_vline(xintercept = c(1, 20))+
  RcmdrPlugin.KMggplot2::geom_stepribbon(aes(ymin = ymin, ymax = ymax),
                                         fill="grey", alpha = 0.2)+
  xlim(1,20)+
  ylim(0.0,1.0)+
  labs(x = "Timepoint", y = "Posterior Mean")+
  theme(legend.position="none")
```



Here we have calculated the absolute area between participant 60 (bottom line) and the reference curve (top line). The shaded region is the area that is calculated and this is then rescaled by the number of observations.

Next, we reorganise the area data so that it is in the correct format for the DPMM analysis.

```
#spread into a data frame of just areas
data_with_area_wide<- dplyr::select(rescaled_area, participant_id, dimension,
                                    rescaled_area)%>%
                      unique()%>%
                      spread(dimension, rescaled_area, fill=NA)

#remove child 1111
data_with_area_wide<- filter(data_with_area_wide, participant_id != "ref")
#reorganise levels of participant_id
data_with_area_wide$participant_id<-factor(data_with_area_wide$participant_id,
```

```
                                          levels = c(1:100))
data_with_area_wide<-arrange(data_with_area_wide, participant_id)
```

Because we will transform the data using a logit-normal transformation, we first need to change any zeros to a small number - I have changed them to the smallest non-zero area for each variable.

```
#replace 0's with the smallest areas
sort(data_with_area_wide$B)
data_with_area_wide$A[data_with_area_wide$A == 0] <-0.008123948
data_with_area_wide$B[data_with_area_wide$B == 0] <- 0.02553204
#transform data
logit_data<- data_with_area_wide %>%
  mutate(logit_A = logit(A),
         logit_B = logit(B))
```

DPMM

The next step of the analysis is to perform the Dirichlet process mixture modelling. To do this I have used code publicly available on Nicole White's Github repository. Please visit https://github.com/nicolemwhite/spike_sorting_DPM to access the R code for this step.

Post processing clusters

I have fit the DPMM to the logit_data file using the code at the link above. I have saved the MCMC draws in the RDS file "MCMC_traces_simdata.rds". I only ran 1 chain for 10,000 iterations and used the following priors: $\mu = \bar{y}$, $N_0 = 0.1$, $c_0 = 3$, $C_0 = \Sigma_y$ and $\alpha \sim Gamma(1, 1)$.

The following section outlines how to post process the MCMC draws to find the optimal cluster composition.The RDS file contains the MCMC draws for z (the cluster assignment label for each observation at each draw), K (the number of clusters at each draw) and α (the discrimination parameter for each draw).

```
output<-readRDS("MCMC_traces_simdata.rds")

#extract K
output_k<-output$K
output_k<-as.mcmc(output_k)

#extract alpha
output_alpha<-output$alpha
output_alpha<-as.mcmc(output_alpha)

#extract z
output_z<-output$z
output_z<-as.mcmc(output_z)
```

To assess the convergence of the chains for K and alpha these functions in the coda package are useful which prints the traceplot, effective sample size and the autocorrelation for the chain. If you have run more than 1 chain (recommended) you can also calculate the Gelman-rubin statistic using the function gelman.diag.

```
traceplot(output_k)
effectiveSize(output_k)
autocorr(output_k)

traceplot(output_alpha)
effectiveSize(output_alpha)
autocorr(output_alpha)

combined_chains<- mcmc.list(chain1_k, chain2_k, chain3_k)
traceplot(combined_chains)
gelman.diag(combined_chains)
gelman.plot(combined_chains)
```

Once the chains have converged, the next step is to find the optimal clusters. First, we need to calculate the posterior similarity matrix. This matrix is a $n \times n$ matrix where each element is the proportion over all draws that two observations were placed in the same cluster. We then calculate the dissimilarity matrix from the similarity matrix.

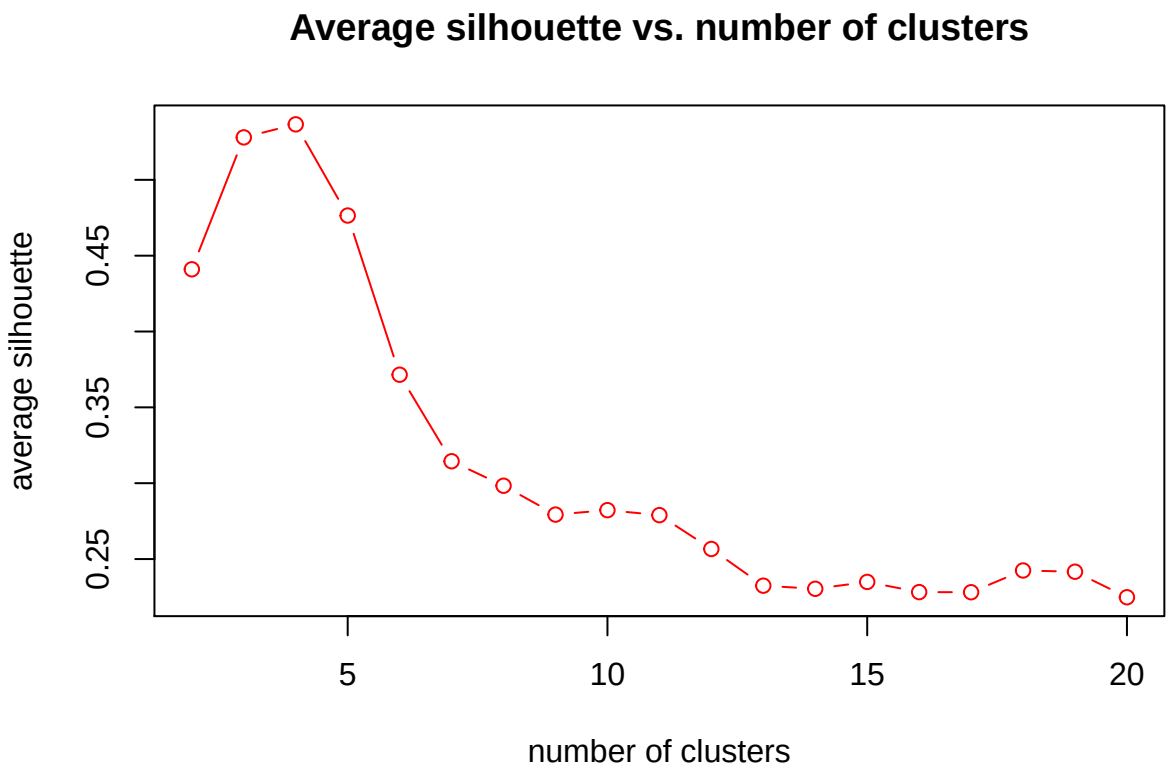
```
#calculate posterior similarity matrix
sim_mat<-comp.psm(t(output_z))
dissim<- 1 - sim_mat
dissim<-as.dist(dissim)
```

If you have used more than 1 chain you can calculate the posterior similarity matrix for each chain seperately and then calculate the average over all the posterior similarity matrices using the following code.

```
#combine PSM  for each chain in a list
list_PSM<- list(sim_mat1, sim_mat2, sim_mat3)

#calculate average for each element to join the chains
mean_PSM<- apply(simplify2array(list_PSM), 1:2, mean)
```

Once we have the dissimilarity matrix we can use this as input for the PAM (Partitioning around medoids) clustering. We choose the best number of clusters by calculating the average silhouette width for 2-20 clusters and selecting the maximum.

```
#get silhouette widths for range of clsuter sizes using PAM
av.sil.Width <- function(k) {
  pr.pam <- pam(dissim, k = k)
  return(pr.pam$silinfo$avg.width)
}
k <- 2:20
avg.width <- apply(as.matrix(k), 1, FUN = av.sil.Width)
plot(k, avg.width, xlab = "number of clusters", ylab = "average silhouette",
     type = "b", col = "red", main = "Average silhouette vs. number of clusters")
```



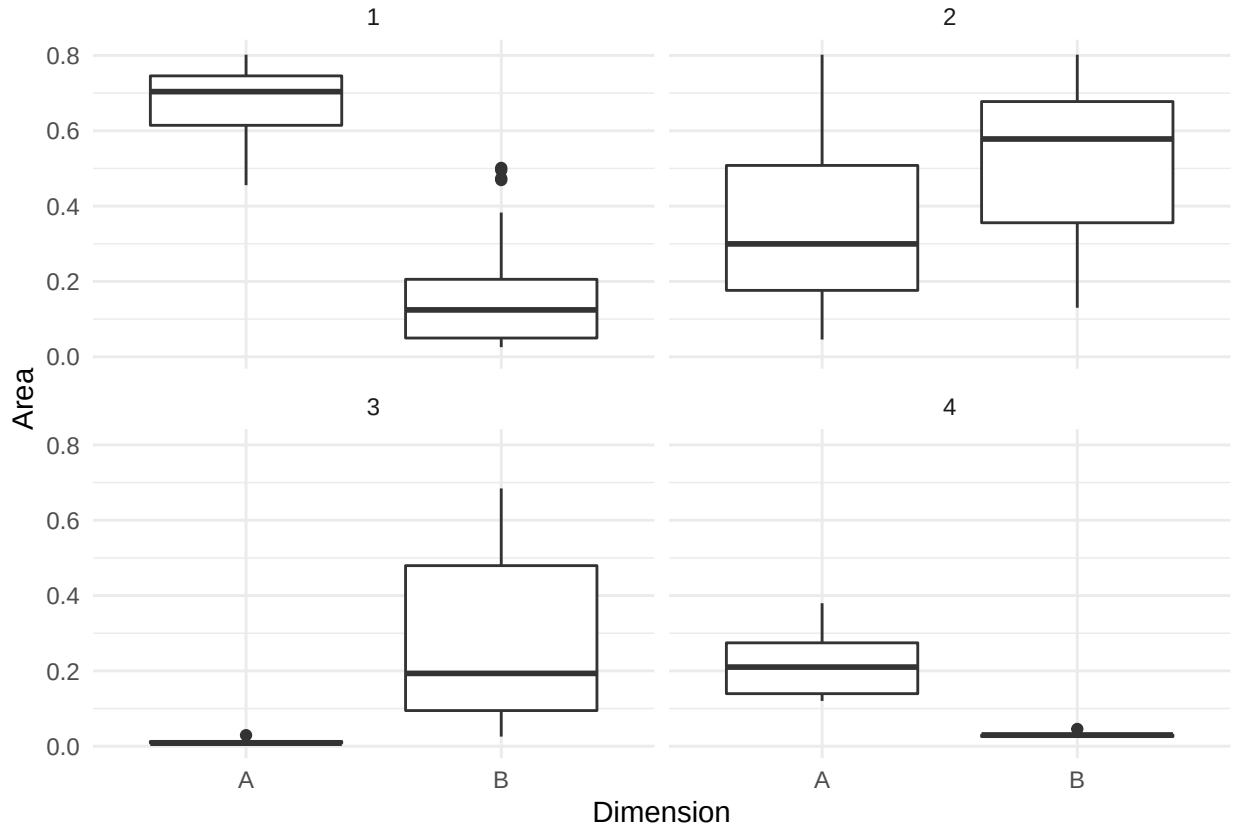
Here the maximum number of clusters is 4 clusters. We run the PAM clustering specifying 4 clusters, extract the cluster labels and plot the profiles for each cluster.

```
#use PAM method to method to find optimal clusterings
#get PAM clusters
pam_mean<-pam(dissim, k=4)
cl_pam_mean<- pam_mean$clustering
#number of participants in each cluster
summary(factor(cl_pam_mean))
```

```
##  1  2  3  4
## 31 54 10  5
```

```
#combine the cluster labels with the logit_data file

logit_data<-as.matrix(logit_data)

data_with_clusters<-cbind(logit_data, cl_pam_mean)

data_with_clusters<- as.data.frame(data_with_clusters)
#only select the columns with original areas for display

data_with_clusters<-dplyr::select(data_with_clusters, c(1,2,3,6))

#restructure the data frame to long format
```

```r
data_long <- melt(data_with_clusters, id.vars=c("participant_id", "cl_pam_mean"))
data_long$variable <- as.factor(data_long$variable)
data_long$cl_pam_mean<- as.factor(data_long$cl_pam_mean)
data_long$value<-as.numeric(data_long$value)
data_long$participant_id<-as.factor(data_long$participant_id)



ggplot(data=data_long, aes(x = variable, y = value, group = variable))+
  geom_boxplot()+
  facet_wrap(~cl_pam_mean)+
  theme_minimal()+
  labs(x = "Dimension", y = "Area")
```



The four clusters that were identified from the DPMM are shown in the boxplot above. Cluster one contains 31 participants and is characterised by high areas for dimension A and low areas for dimension B. The second cluster contains 54 participants with larger areas for dimension B than dimenson A. Cluster three has 10 participants and is characterised by an area of zero for dimension A (meaning that they have the same probability sequence as the reference observation) and more variability in their area for dimension B. Finally, cluster four contains 5 participants who have an area of zero for dimension B. These results demonstrate the DPMM's tendency to reveal smaller clusters that don't exaclty fit the profile for the larger clusters.