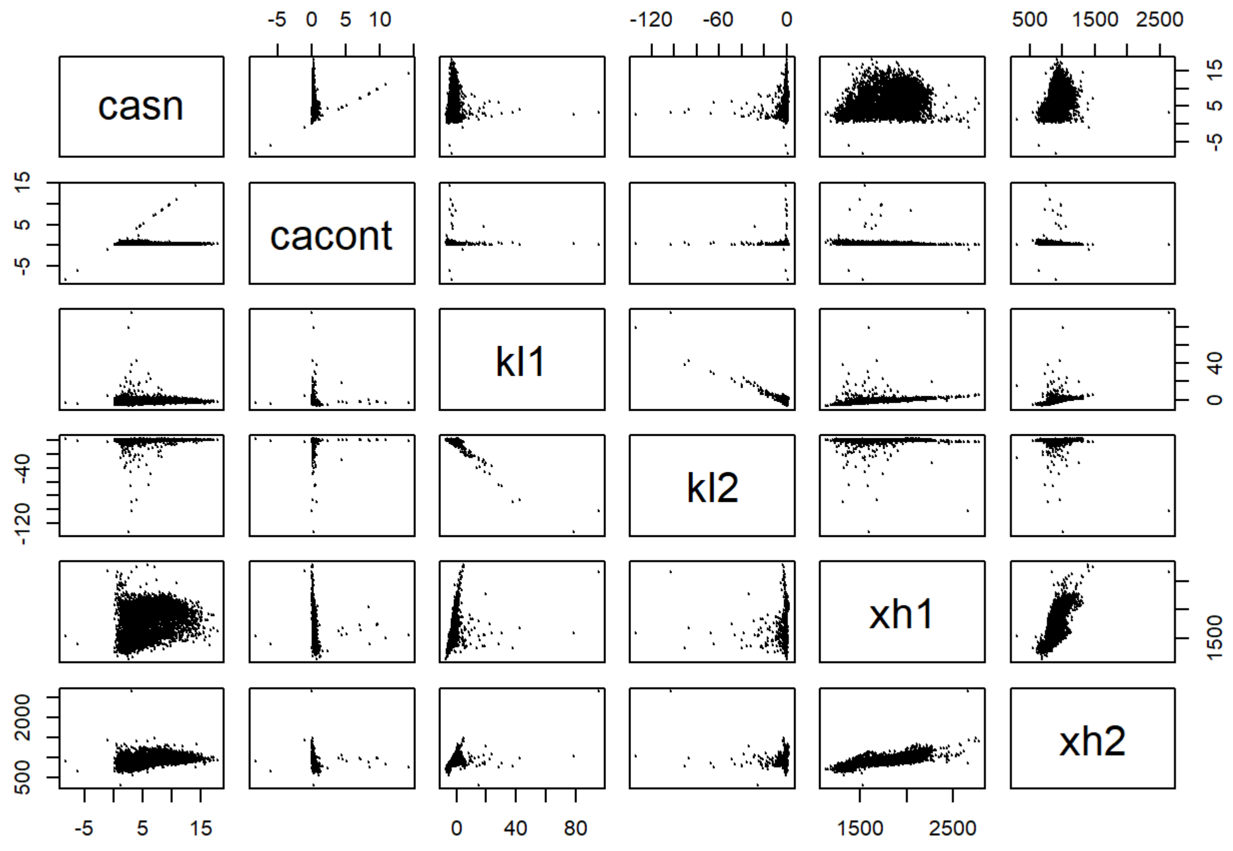
Assignment 6

Sebastian Veuskens

Exercise 1

```
library(mixsmsn)

stars5000 <- read.table("data/stars5000.dat", header=TRUE)

pairs(stars5000, pch=20, cex=0.1)
```



```
# For subsampling with bug
stars1000_ind <- sample(1:5000, 1000)
stars1000 <- stars5000[stars1000_ind,]
```

Standardization

```
data <- stars1000

data_st <- scale(data) # standard scaling

medians <- apply(data, 2, median)
mads <- apply(data, 2, mad)
data_centered <- sweep(data, 2, medians)
data_md <- sweep(data, 2, mads, FUN = "/") # median scaling

dist_st <- dist(data_st)
dist_md <- dist(data_md)
```

Determine DF

```
# In this case, we use 3 degrees of freedom to have a heavy-tailed distribution that still ha
s a variance
df = 3
```

Clustering

```
library(mixsmsn)

cluster_t <- smsn.search(data_st, df, criteria = "bic", family = "t")
cluster_ts <- smsn.search(data_st, df, criteria = "bic", family = "Skew.t")
cluster_n <- smsn.search(data_st, 10, criteria = "bic", family = "Normal")
cluster_ns <- smsn.search(data_st, 1, criteria = "bic", family = "Skew.normal")
```

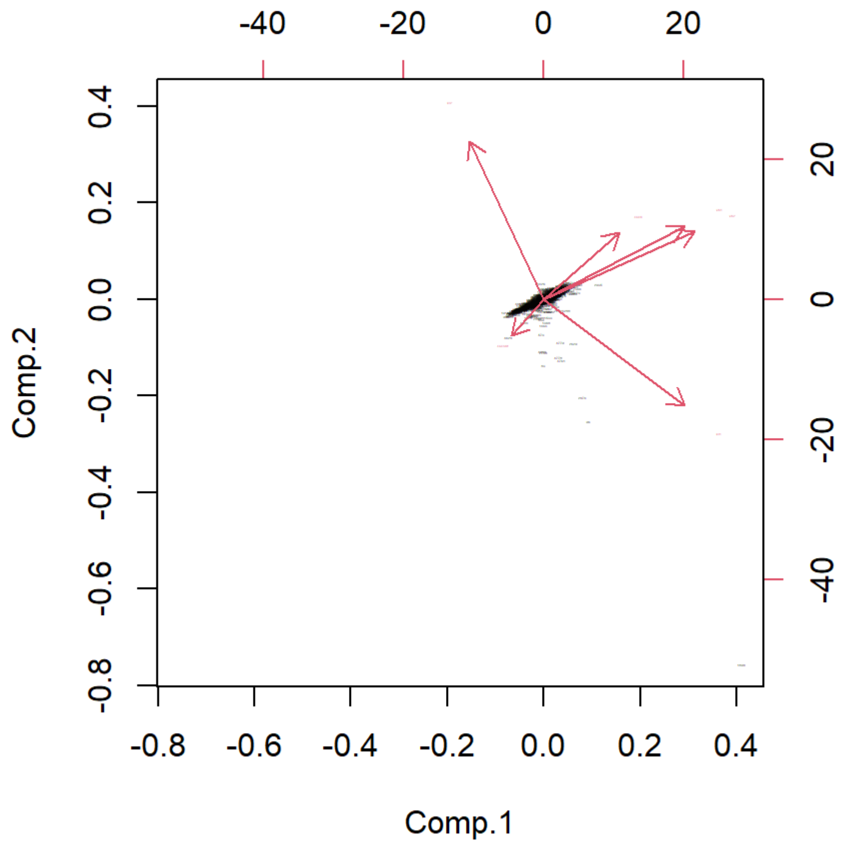
Comparison

Visual separation

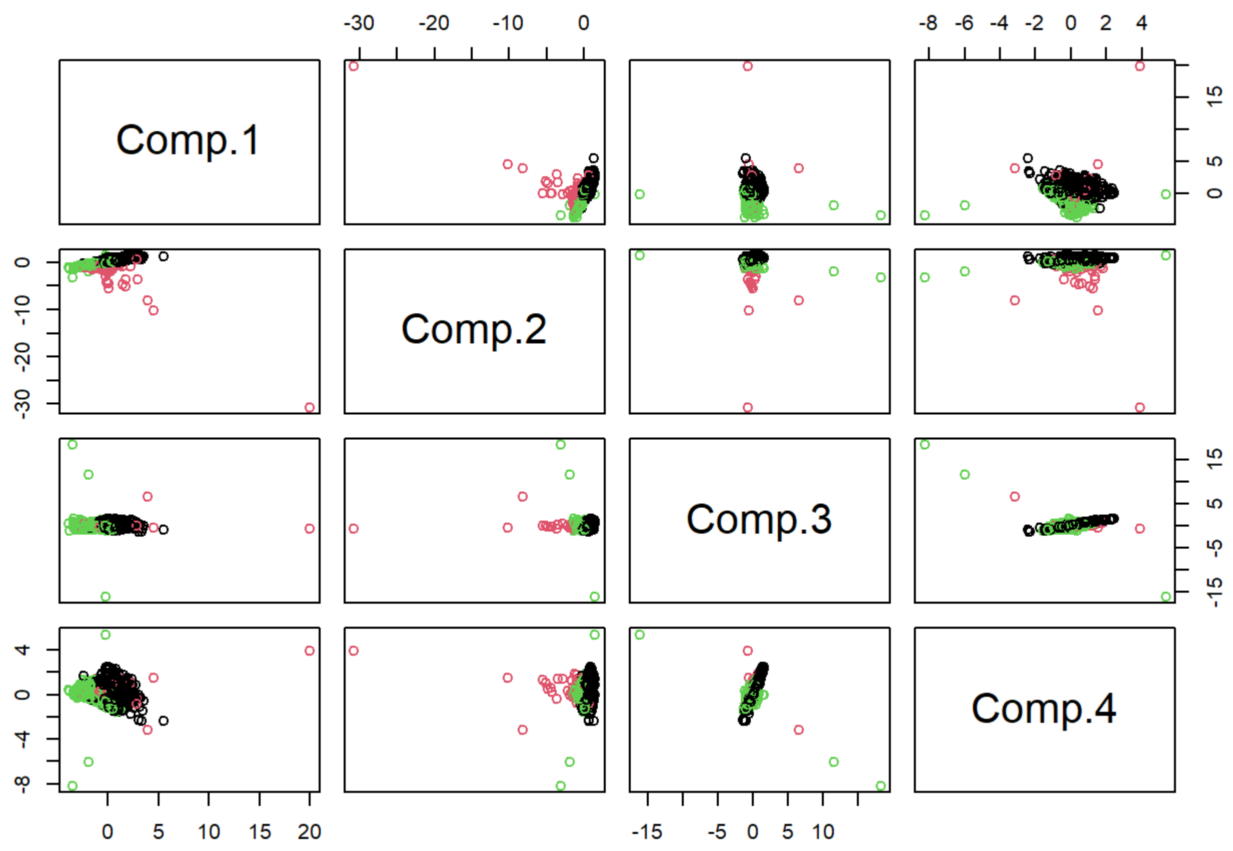
```
# Principal Components
pr_data <- princomp(data_st)
summary(pr_data)
```

```
## Importance of components:
##                           Comp.1    Comp.2    Comp.3    Comp.4     Comp.5
## Standard deviation     1.5418338 1.2860554 1.0226103 0.8444369 0.43329115
## Proportion of Variance 0.3966052 0.2759323 0.1744631 0.1189646 0.03132153
## Cumulative Proportion  0.3966052 0.6725375 0.8470006 0.9659652 0.99728675
##                            Comp.6
## Standard deviation     0.12752733
## Proportion of Variance 0.00271325
## Cumulative Proportion  1.00000000
```

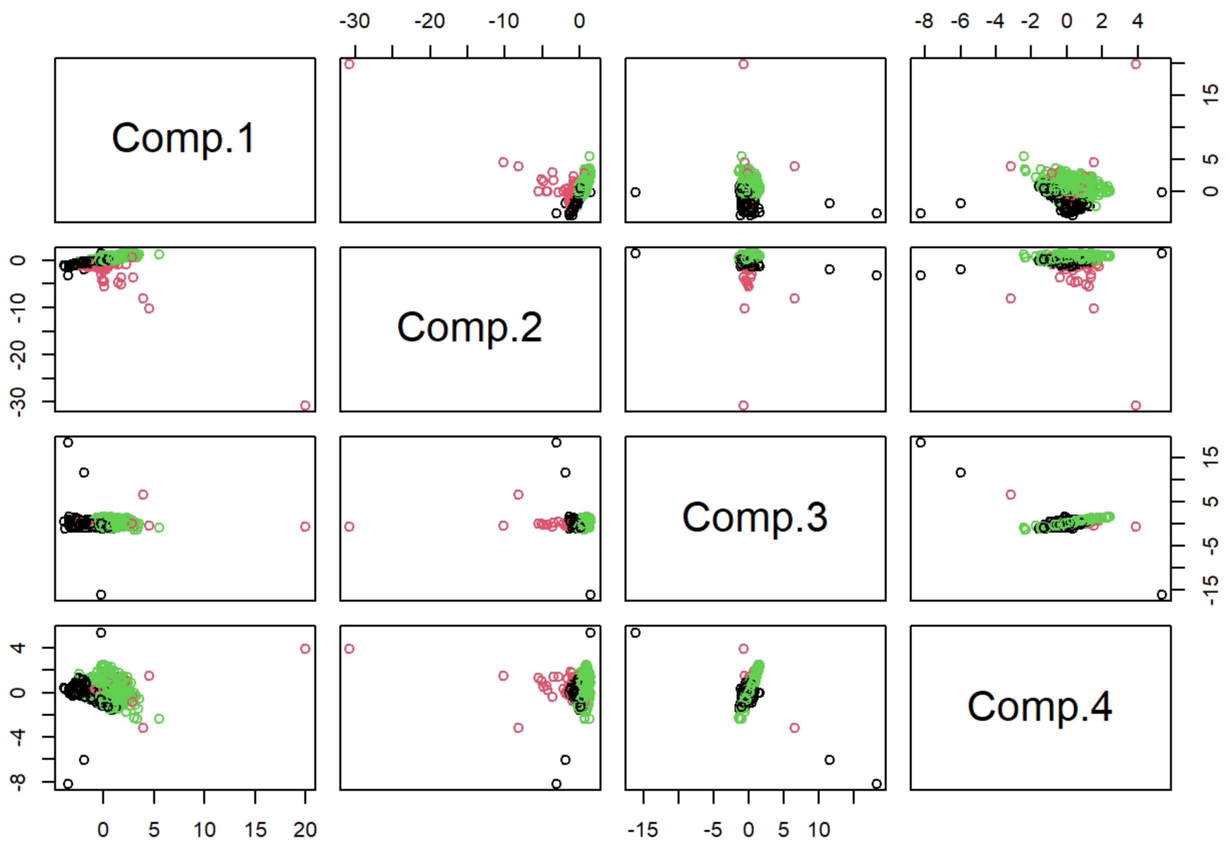
```
biplot(pr_data, cex=0.1)
```



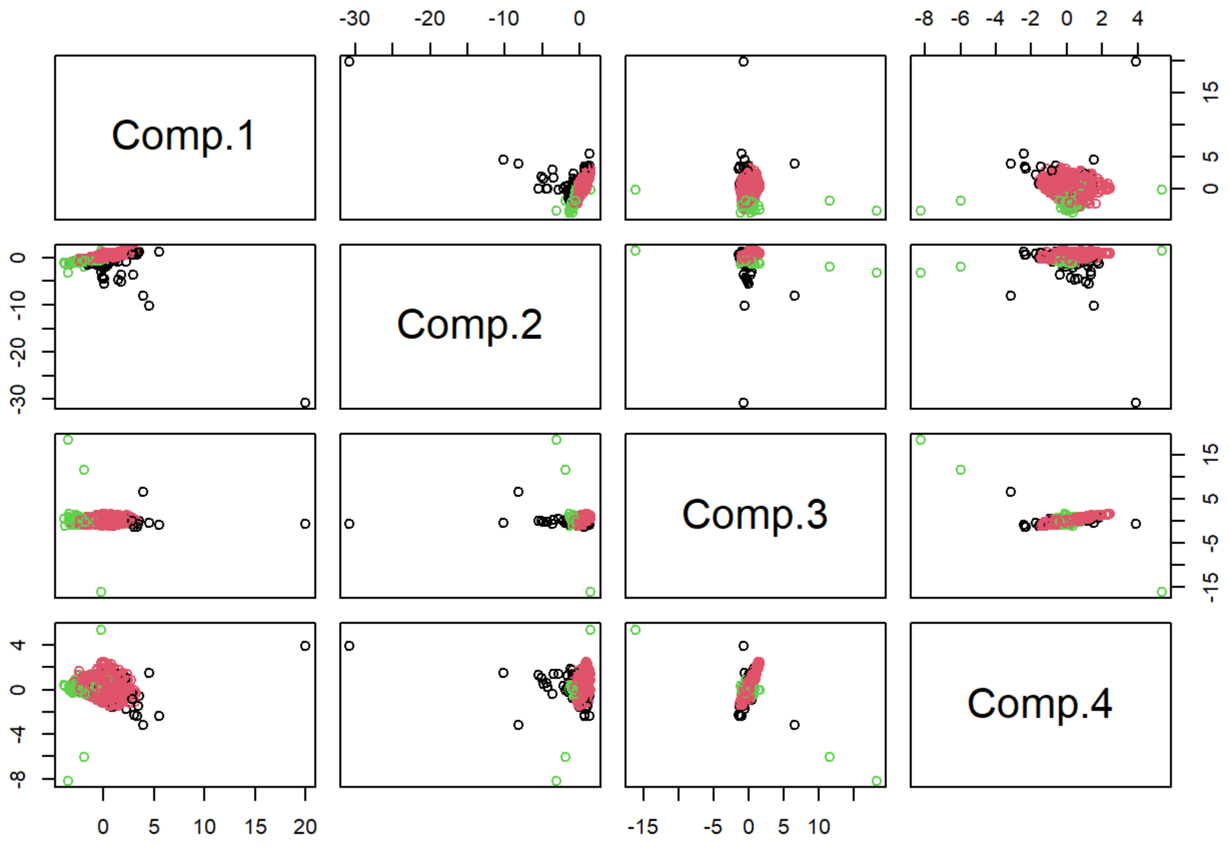
```
pairs(pr_data$scores[,1:4], col=cluster_t$best.model$group)
```



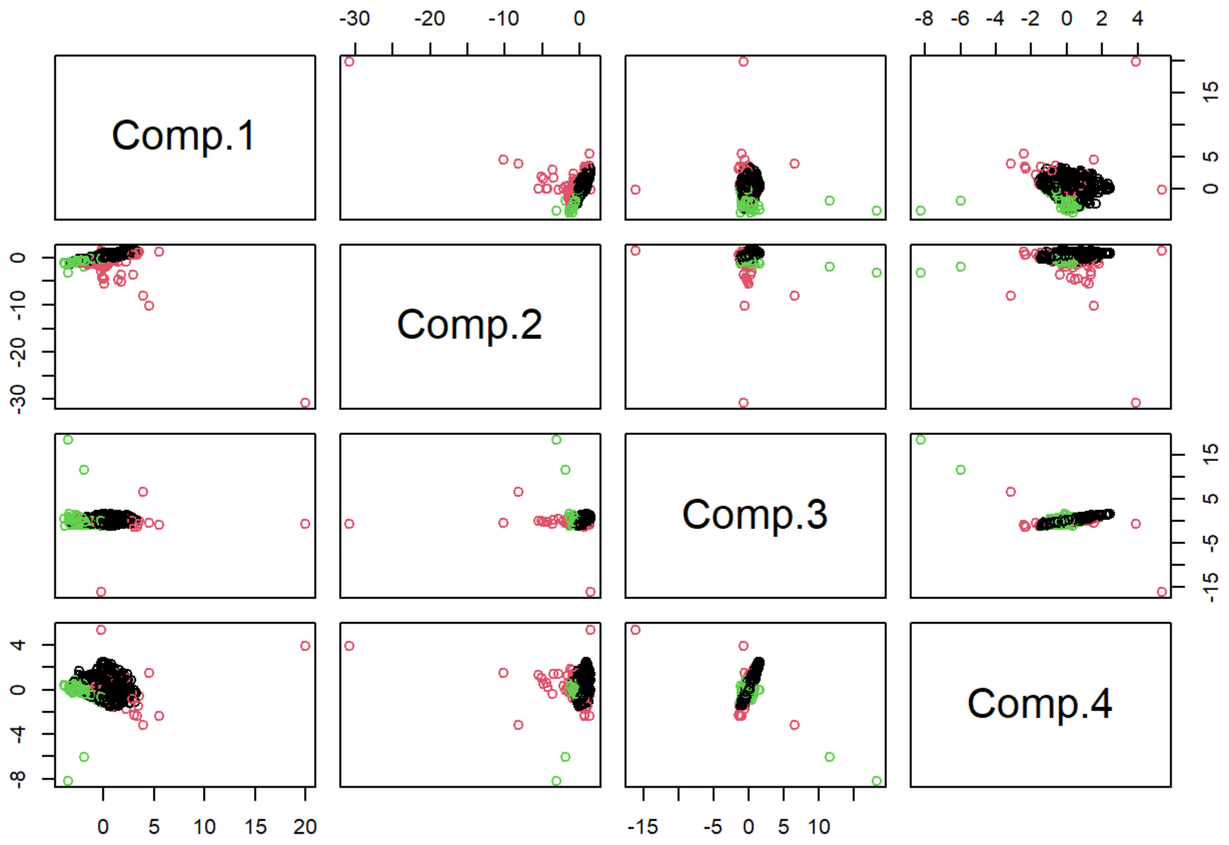
```
pairs(pr_data$scores[,1:4], col=cluster_ts$best.model$group)
```



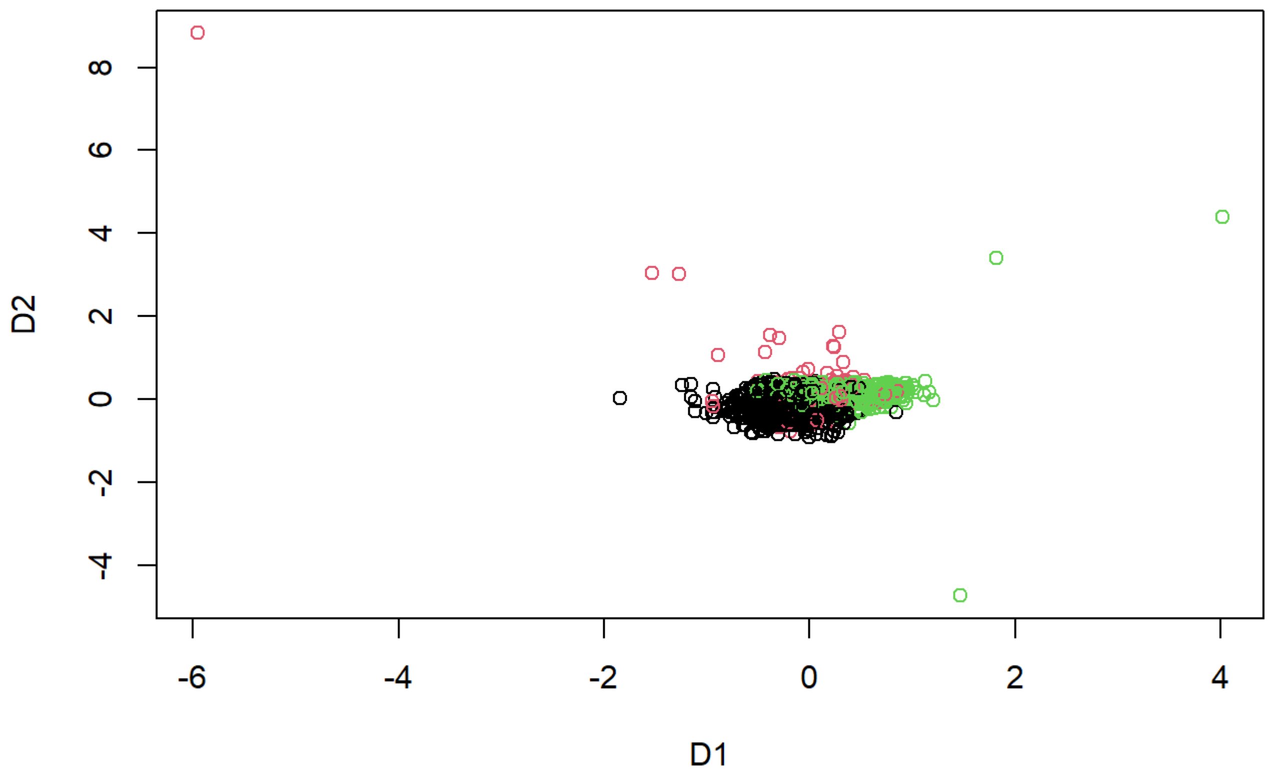
```
pairs(pr_data$scores[,1:4], col=cluster_n$best.model$group)
```



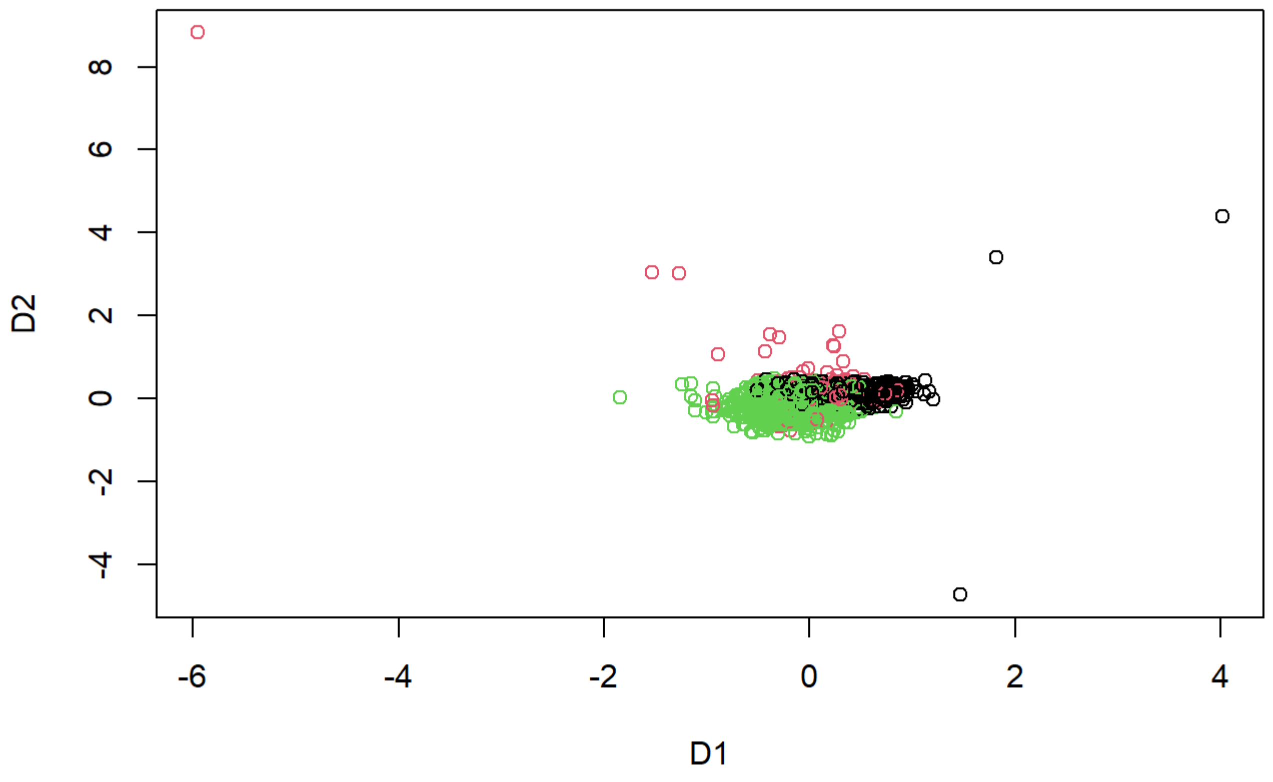
```
pairs(pr_data$scores[,1:4], col=cluster_ns$best.model$group)

# MDS
library(smacof)
```



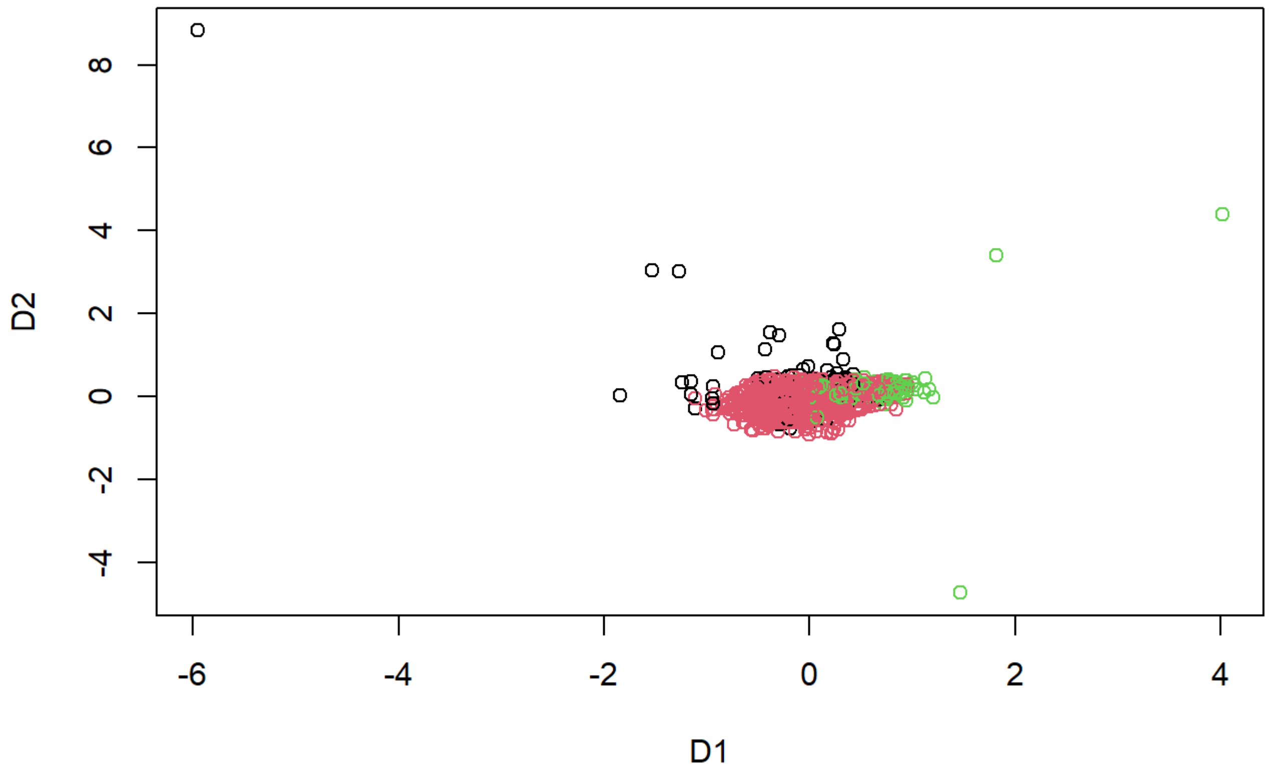
```
data_mds <- mds(dist(data_st), ndim=2)

plot(data_mds$conf, col=cluster_t$best.model$group)
```



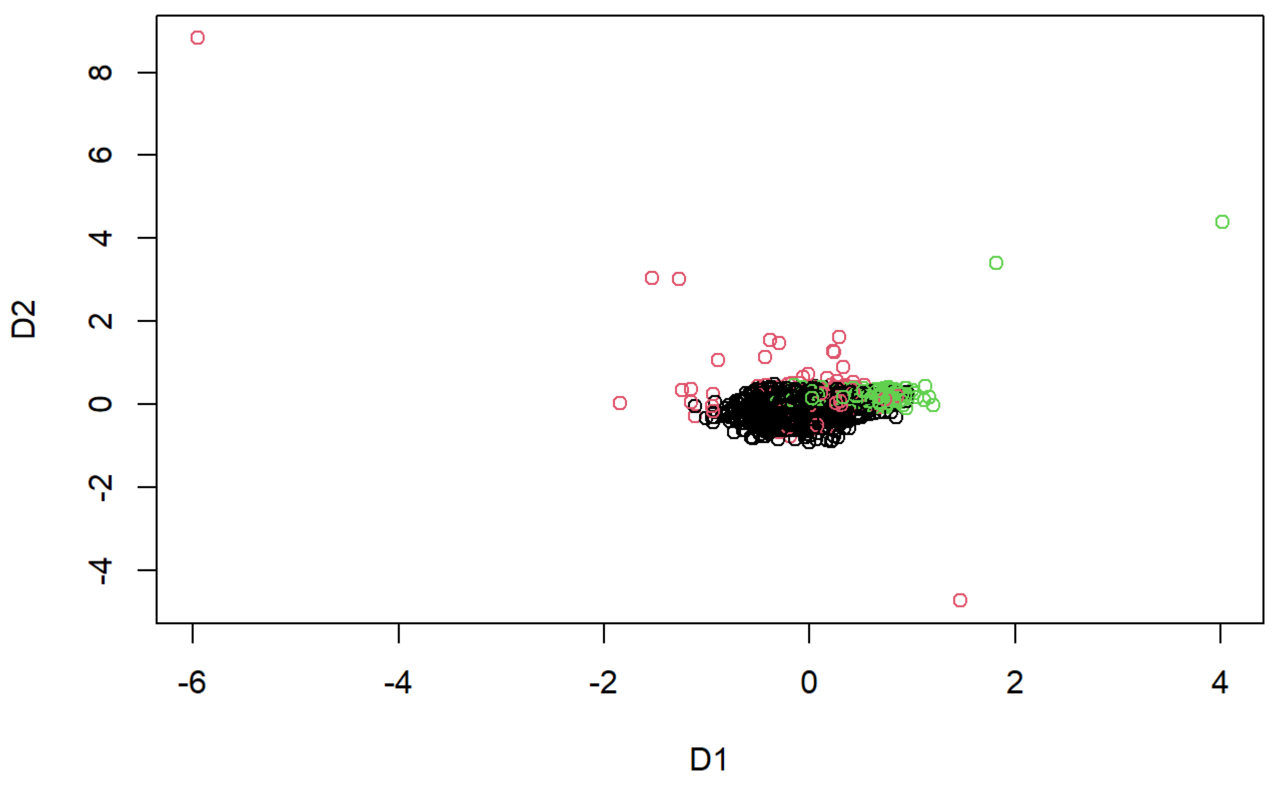
```
plot(data_mds$conf, col=cluster_ts$best.model$group)
```



```
plot(data_mds$conf, col=cluster_n$best.model$group)
```



```
plot(data_mds$conf, col=cluster_ns$best.model$group)
```



From the visual inspection, all clusters look fairly similar. In each clustering, a “noise” cluster is present, while the other two clusters show pretty much the same data.

Goodness of fit measures

```
library(cluster)

summary(silhouette(cluster_t$best.model$group, dist=dist_st))$avg.width
```

```
## [1] 0.2454327
```

```
summary(silhouette(cluster_ts$best.model$group, dist=dist_st))$avg.width
```

```
## [1] 0.227955
```

```
summary(silhouette(cluster_n$best.model$group, dist=dist_st))$avg.width
```

```
## [1] 0.2079847
```

```
summary(silhouette(cluster_ns$best.model$group, dist=dist_st))$avg.width
```

```
## [1] 0.2116848
```

According to the ASW, the multivariate t-distribution is the best clustering. Since the visual inspection did not suggest a certain clustering, the decision is completely based on the ASW and the multivariate t-distribution is chosen as the final clustering.

It is important to mention, that especially in the Gaussian mixture models, the ASW might lead to misleading results, since clusters do not necessarily have to have a low within-cluster distance.

Exercise 2

Clustering

```
library(mclust)

bd_method <- function(data, sample_size = 1000, G=1:20) {
    sample_ind <- sample(1:5000, sample_size)
    sample_data <- data[sample_ind,]
    m_clus_sample <- Mclust(sample_data, G=G)
    pred <- predict.Mclust(m_clus_sample, data[-sample_ind,])
    pred_class <- numeric(5000)
    pred_class[sample_ind] <- m_clus_sample$classification
    pred_class[-sample_ind] <- pred$classification
    result <- list()
    result$m_clus <- m_clus_sample
    result$classification <- pred_class
    result$data <- data
    result$sample_data <- sample_data
    return(result)
}

time_bd <- system.time(m_clus_1000 <- bd_method(stars5000))
time_mc <- system.time(m_clus_5000 <- Mclust(stars5000, G=1:20))
```

Comparison

```
# Time
time_bd
```

```
##    user  system elapsed
##   27.47    0.03   31.05
```

```
time_mc
```

```
##    user  system elapsed
##  140.74    0.31  152.05
```

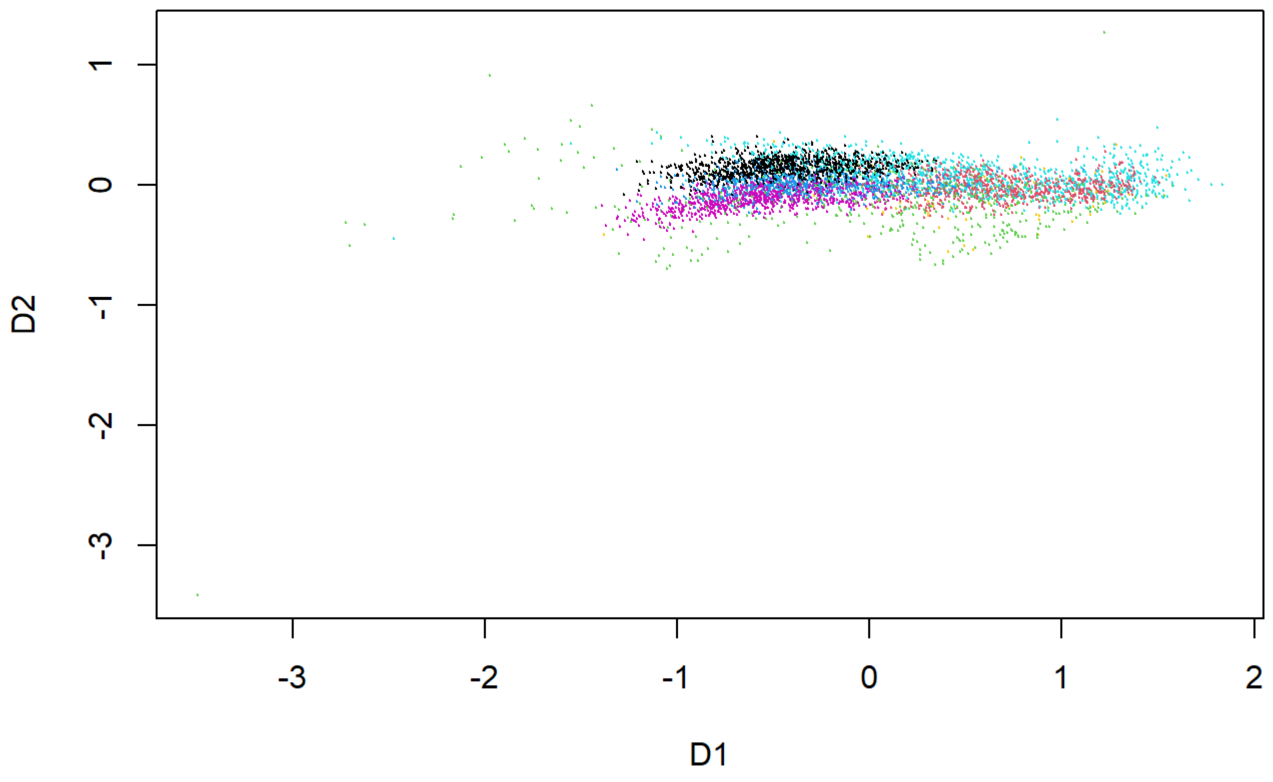
```
time_bd[3] * 5 / time_mc[3]
```

```
##  elapsed
## 1.021046
```

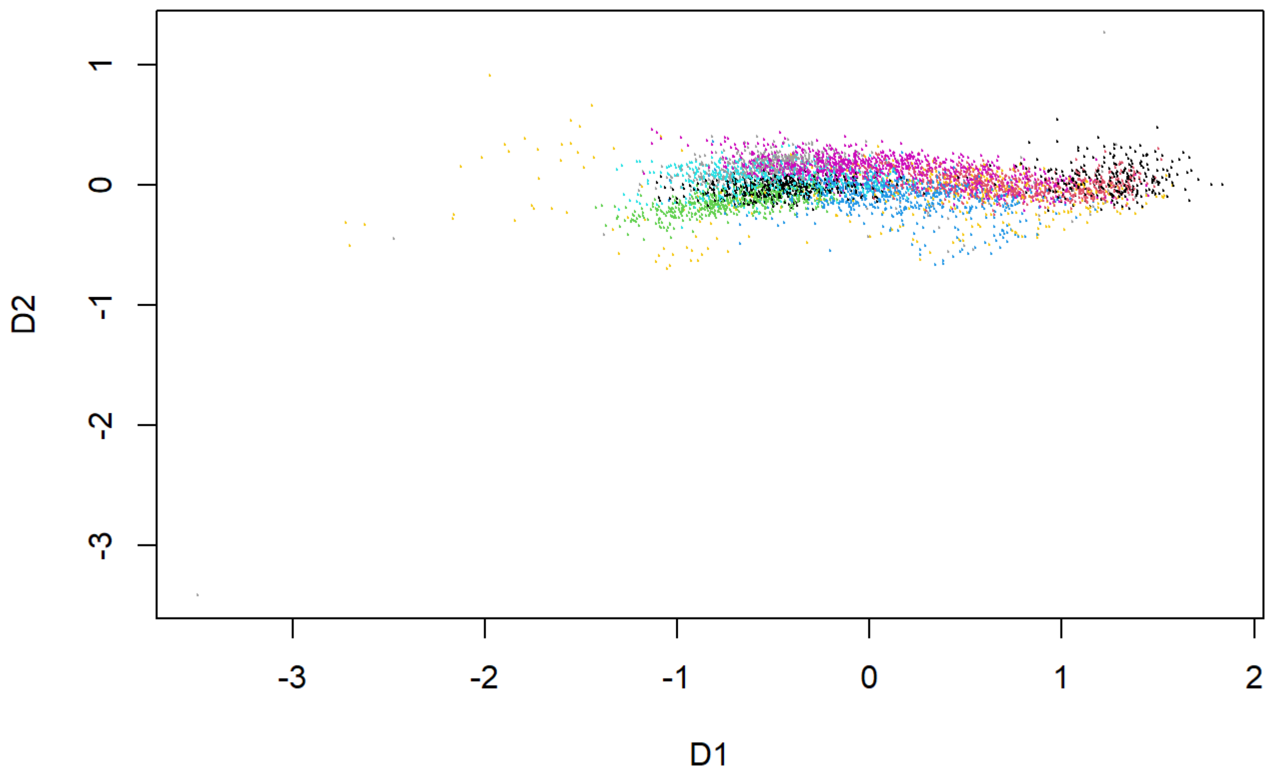
```
# Adjusted Rand Index
ari_clusters <- adjustedRandIndex(m_clus_1000$classification, m_clus_5000$classification)
ari_clusters
```

```
## [1] 0.3906156
```

```
# Visual Inspectation
library(smacof)
mds_stars5000 <- mds(dist(stars5000), ndim=2)
plot(mds_stars5000$conf, col=m_clus_1000$classification, cex=0.01)
```



```
plot(mds_stars5000$conf, col=m_clus_5000$classification, cex=0.01)
```



The time for Mclust on the sampled data is much faster than on the original data. However, overall (elapsed) runtime seems to increase almost linearly with the number of observations, since in this case the following equation holds **runtime(5000_obs) ~= runtime(1000_obs) * 5**. Thus, the decrease of the number of observations is directly proportional to the number of observations here.

The ARI is comparably small, meaning the resulting clusters are fairly different. In this case, we tried to achieve a clustering on only a sample of the data that is as similar to the clustering on the whole data as possible. According to the ARI, the clusterings differ significantly and thus indicate a “worse” performance of the clustering on the sampled data. Here, an assumption is that more data leads to a better clustering, and thus we can consider the clustering on the whole dataset as the “true” clustering that we want to achieve.

Exercise 4

Generate dataset

```
prodcat <- c("PR","MB","AB","N") # product categories
eventcat <- c("S","M","C","P","N") # event categories
prodp <- eventp <- list()
# Category probabilities within the two mixture components (clusters)
prodp[[1]] <- c(0.4,0.4,0.1,0.1)
prodp[[2]] <- c(0.2,0.1,0.4,0.3)
eventp[[1]] <- c(0.5,0.2,0.1,0.1,0.1)
eventp[[2]] <- c(0.1,0.1,0.1,0.3,0.4)
# The first 400 observations from component 1, then 600 from component 2:
n1 <- 400
n2 <- 600
n <- n1+n2
# Initialisation (provide a data frame of the correct type and size
# without already having the data):
consumers <- data.frame(prod=factor(c(prodcat,rep(NA,n-4)),levels=prodcat),
event=factor(c(eventcat,rep(NA,n-5)),levels=eventcat))
# Generation of the data; you may want to set a seed here.
consumers$prod[1:n1] <- sample(prodcat,n1,replace=TRUE,prob=prodp[[1]])
consumers$event[1:n1] <- sample(eventcat,n1,replace=TRUE,prob=eventp[[1]])
consumers$prod[(n1+1):n] <- sample(prodcat,n2,replace=TRUE,prob=prodp[[2]])
consumers$event[(n1+1):n] <- sample(eventcat,n2,replace=TRUE,prob=eventp[[2]])
# You can run table(consumers) or str(consumers) to see what you got.
table(consumers)
```

```
##     event
## prod  S  M  C  P  N
##   PR 98 35 27 61 65
##   MB 95 33 23 26 35
##   AB 46 30 23 73 94
##   N  35 29 31 61 80
```

```
str(consumers)
```

```
## 'data.frame':    1000 obs. of  2 variables:
##  $ prod : Factor w/ 4 levels "PR","MB","AB",..: 2 2 1 2 1 2 2 1 1 2 ...
##  $ event: Factor w/ 5 levels "S","M","C","P",..: 5 4 1 1 1 1 1 2 1 1 ...
```

```
true_labels <- numeric(1000)
true_labels[1:n1] <- 1
true_labels[(n1+1):n] <- 2
```

Cluster dataset

```
library(fpc)
set.seed(12345) # For flexmixdruns call

flex_consumers <- flexmixedruns(consumers, continuous = 0, discrete=2, n.cluster=1:5)
```

```
## k=  1  new best fit found in run  1
## k=  1  BIC=  5889.174
## k=  2  new best fit found in run  1
## k=  2  new best fit found in run  2
## Nonoptimal or repeated fit found in run  3
## Nonoptimal or repeated fit found in run  4
## Nonoptimal or repeated fit found in run  5
## Nonoptimal or repeated fit found in run  6
## Nonoptimal or repeated fit found in run  7
## Nonoptimal or repeated fit found in run  8
## k=  2  new best fit found in run  9
## Nonoptimal or repeated fit found in run  10
## Nonoptimal or repeated fit found in run  11
## Nonoptimal or repeated fit found in run  12
## Nonoptimal or repeated fit found in run  13
## Nonoptimal or repeated fit found in run  14
## Nonoptimal or repeated fit found in run  15
## Nonoptimal or repeated fit found in run  16
## Nonoptimal or repeated fit found in run  17
## Nonoptimal or repeated fit found in run  18
## Nonoptimal or repeated fit found in run  19
## Nonoptimal or repeated fit found in run  20
## k=  2  BIC=  5854.98
## k=  3  new best fit found in run  1
## k=  3  new best fit found in run  2
## Nonoptimal or repeated fit found in run  3
## k=  3  new best fit found in run  4
## Nonoptimal or repeated fit found in run  5
## Nonoptimal or repeated fit found in run  6
## k=  3  new best fit found in run  7
## Nonoptimal or repeated fit found in run  8
## Nonoptimal or repeated fit found in run  9
## Nonoptimal or repeated fit found in run  10
## Nonoptimal or repeated fit found in run  11
## Nonoptimal or repeated fit found in run  12
## Nonoptimal or repeated fit found in run  13
## Nonoptimal or repeated fit found in run  14
## Nonoptimal or repeated fit found in run  15
## Nonoptimal or repeated fit found in run  16
## Nonoptimal or repeated fit found in run  17
## Nonoptimal or repeated fit found in run  18
## Nonoptimal or repeated fit found in run  19
## Nonoptimal or repeated fit found in run  20
## k=  3  BIC=  5906.18
## k=  4  new best fit found in run  1
## k=  4  new best fit found in run  2
## Nonoptimal or repeated fit found in run  3
## Nonoptimal or repeated fit found in run  4
## Nonoptimal or repeated fit found in run  5
## Nonoptimal or repeated fit found in run  6
## Nonoptimal or repeated fit found in run  7
## Nonoptimal or repeated fit found in run  8
## Nonoptimal or repeated fit found in run  9
## Nonoptimal or repeated fit found in run  10
## Nonoptimal or repeated fit found in run  11
## Nonoptimal or repeated fit found in run  12
## k=  4  new best fit found in run  13
## Nonoptimal or repeated fit found in run  14
## Nonoptimal or repeated fit found in run  15
## Nonoptimal or repeated fit found in run  16
## Nonoptimal or repeated fit found in run  17
## Nonoptimal or repeated fit found in run  18
## Nonoptimal or repeated fit found in run  19
## Nonoptimal or repeated fit found in run  20
## k=  4  BIC=  5960.633
## k=  5  new best fit found in run  1
## k=  5  new best fit found in run  2
## Nonoptimal or repeated fit found in run  3
## k=  5  new best fit found in run  4
## Nonoptimal or repeated fit found in run  5
## Nonoptimal or repeated fit found in run  6
## k=  5  new best fit found in run  7
## Nonoptimal or repeated fit found in run  8
## Nonoptimal or repeated fit found in run  9
## Nonoptimal or repeated fit found in run  10
## k=  5  new best fit found in run  11
## Nonoptimal or repeated fit found in run  12
## Nonoptimal or repeated fit found in run  13
## Nonoptimal or repeated fit found in run  14
## k=  5  new best fit found in run  15
## Nonoptimal or repeated fit found in run  16
## Nonoptimal or repeated fit found in run  17
## Nonoptimal or repeated fit found in run  18
## Nonoptimal or repeated fit found in run  19
## Nonoptimal or repeated fit found in run  20
## k=  5  BIC=  6015.89
```

```
num_clusters <- which.min(flex_consumers$bicvals) # Minimum BIC is best here

flex_cluster <- flex_consumers$flexout[[num_clusters]]@cluster
flex_cluster
```

```
##    [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 2 1 1 1 1 1
##   [38] 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 2
##   [75] 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1
##  [112] 2 2 1 2 2 2 1 1 2 1 1 1 1 1 1 1 1 1 2 2 1 2 1 1 1 1 1 1 1 2 1 1 1 1 2 1 1
##  [149] 1 1 1 1 2 2 1 1 1 2 2 1 1 1 1 2 2 1 1 1 1 1 2 1 1 1 1 1 1 2 2 1 1 1 1 1 2
##  [186] 1 1 1 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [223] 1 1 1 1 2 1 1 1 1 1 2 1 1 1 2 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [260] 1 1 1 1 1 1 1 2 1 2 1 1 2 1 1 1 1 2 1 2 2 1 1 1 2 1 2 1 2 1 1 1 1 2 1 1 1
##  [297] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2 1 1 2 1 1 1 2
##  [334] 2 2 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1 1 1 2 1
##  [371] 1 1 1 2 2 1 1 2 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 2 1 1 1 1 1 1 2 2 2 2 1 2 2
##  [408] 2 2 1 2 2 1 2 2 2 2 2 2 2 2 2 1 2 2 2 1 1 2 2 2 2 2 2 2 2 1 2 2 2 2 2 1 2
##  [445] 1 2 2 2 2 2 2 1 2 2 2 2 2 2 2 1 2 2 2 1 2 2 1 2 2 1 2 1 2 2 2 1 2 1 2 2 2
##  [482] 1 2 2 1 2 1 2 2 2 2 2 2 1 2 2 2 2 2 2 2 1 2 2 2 2 1 2 2 2 2 2 2 1 1 1 2 2
##  [519] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 1 2 2 2 1 2 2 2 2 2 2 2
##  [556] 1 2 2 2 2 2 2 2 2 1 1 1 1 2 2 2 1 2 2 2 1 2 2 2 1 2 1 1 1 1 2 2 2 1 2 2 2
##  [593] 2 2 1 1 1 2 2 2 2 2 2 2 1 2 2 2 2 2 1 2 2 2 2 2 2 2 1 2 2 2 2 1 2 2 1 2 2
##  [630] 1 2 2 2 2 1 2 2 2 2 2 1 2 2 2 1 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 1
##  [667] 1 2 2 1 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 1 2 2 2 2 2 1 2 1 1 2 1 1 2 1 2 2
##  [704] 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 1 2 1 2 2 1 2 1 2 2 2 2 2 2 2 2 2 1 2 2 2
##  [741] 2 2 1 2 2 1 2 1 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 1 2 2 1 2 2 2 2 2 2 2 1 2 2
##  [778] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [815] 2 2 1 2 1 2 2 2 2 1 1 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 1 2 1 1 2 2 2 2 2
##  [852] 1 2 1 2 2 1 2 2 2 1 2 2 2 1 1 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 1 1 1 2 1 2
##  [889] 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 1 2 2 1 2 2 2 2 2 2 2 2 2 1 2 2 2 2 1 2 1
##  [926] 2 2 2 2 2 2 2 2 1 2 2 1 1 2 1 2 2 2 1 1 1 2 2 1 2 2 2 2 1 2 2 2 1 1 2 1 2
##  [963] 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 1 1 2 2 2 2 2 2 2 1 2 2 1 2 1 2 2 2 1 2 2
## [1000] 2
```

```
adjustedRandIndex(flex_cluster, true_labels)
```

```
## [1] 0.3664086
```

```
flex_components <- flex_consumers$flexout[[num_clusters]]@components

pred_com1_par1 <- flex_components[[1]][[1]]@parameters$pp[[1]]
pred_com1_par2 <- flex_components[[1]][[1]]@parameters$pp[[2]]
pred_com2_par1 <- flex_components[[2]][[1]]@parameters$pp[[1]]
pred_com2_par2 <- flex_components[[2]][[1]]@parameters$pp[[2]]

# Predicted probabilities
pred_com1_par1
```

```
## [1] 0.1275716 0.4022984 0.1006938 0.3694362
```

```
pred_com1_par2
```

```
## [1] 0.1004453 0.1521760 0.1268606 0.1090291 0.5114890
```

```
pred_com2_par1
```

```
## [1] 0.38526677 0.04804316 0.35257678 0.21411329
```

```
pred_com2_par2
```

```
## [1] 0.10706265 0.10530893 0.40077198 0.31747160 0.06938484
```

```
# True probabilities
prodp[[1]]
```

```
## [1] 0.4 0.4 0.1 0.1
```

```
eventp[[1]]
```

```
## [1] 0.5 0.2 0.1 0.1 0.1
```

```
prodp[[2]]
```

```
## [1] 0.2 0.1 0.4 0.3
```

```
eventp[[2]]
```

```
## [1] 0.1 0.1 0.1 0.3 0.4
```

```
# Compare probabilities
sum(abs(pred_com1_par1 - prodp[[1]]))
```

```
## [1] 0.5448568
```

```
sum(abs(pred_com2_par1 - prodp[[1]]))
```

```
## [1] 0.7333801
```

```
sum(abs(pred_com1_par1 - prodp[[2]]))
```

```
## [1] 0.7434692
```

```
sum(abs(pred_com2_par1 - prodp[[2]]))
```

```
## [1] 0.3705335
```

```
sum(abs(pred_com1_par2 - eventp[[1]]))
```

```
## [1] 0.8947574
```

```
sum(abs(pred_com2_par2 - eventp[[1]]))
```

```
## [1] 1.036487
```

```
sum(abs(pred_com1_par2 - eventp[[2]]))
```

```
## [1] 0.3819418
```

```
sum(abs(pred_com2_par2 - eventp[[2]]))
```

```
## [1] 0.6612303
```

The number of clusters is predicted correctly, since we know that two different components created the consumer dataset. This is is a quality indicator of the algorithm.

However, the ARI with the true labels is not very strong and suggest, that a lot of samples were classified incorrectly.

From the comparison of the probabilities, the clustering does not reproduce very accuratly the underlying true probabilities. All predicted marginal probabilities differ significantly from the true marginal probabilities for each component.

Apply K-Means clustering algorithm

```
library(cluster)
dist_simple <- daisy(consumers, metric="gower", type=list(factor=c(1,2)))
pam_consumers <- pam(dist_simple, k=2)

adjustedRandIndex(pam_consumers$cluster, true_labels)
```

```
## [1] 0.109603
```

The clustering is not very good. It suggests that the clustering is barely better than random cluster assignment.

Simple matching leads to the following distance matrix:

1. Distance 0 to exact duplicates (same product and event)
2. Distance 0.5 to all observations that have either the same product OR Event
3. Distance 0 to all other observations

The data is made out of 2 clusters with variables with (4, 5) different categories. This means that, in the case of PAM, at least 4 out of 20 different combinations of the variables of the observations have the (maximum) distance 1 to both centroids. The different distance values are very limited (three different possibilities). This makes it hard to find well-separated clusters in general.

Assignment 6 ex 3

24 November 2023 12:38

Meanings of letters

1. Volume
2. Shape
3. Orientation

one volume parameter

diagonal matrix with p values and one constraint $(\det(\cdot) = 1)$

Formulas:

	I	E	V
Volume	0	1	k
Shape	0	$p-1$	$k\cdot(p-1)$
Orientation	0	$\frac{p\cdot(p-1)}{2}$	$k\cdot\frac{p\cdot(p-1)}{2}$
a_k	$p\cdot k$	$p\cdot k$	$p\cdot k$
λ_k	0	p	$p\cdot k$
ν_k	0	1	k
π_k	$k-1$	$k-1$	$k-1$

df of orthonormal matrix

Solutions: $K=4$ $p=10$

a) VVV (GM): No (λ_k, ν_k), Volume $=4$, Shape $=36$, Orientation: 180, $a_k=40$, $\pi_k=3$
$\Rightarrow \boxed{df=263}$

b) VII (GM): No (λ_k, ν_k), Volume $=4$, Shape $=0$, Orientation: 0, $a_k=40$, $\pi_k=3$
$\Rightarrow \boxed{df=47}$

c) Fully flexible skew-normal: No (ν_k), Volume $=4$, Shape $=36$, Orientation: 180, $a_k=40$, $\pi_k=3$, $\lambda_k=40$
$\Rightarrow \boxed{df=303}$

d) Fully flexible t: No (λ_k), Volume $=4$, Shape $=36$, Orientation: 180, $a_k=40$, $\pi_k=3$, $\nu_k=4$
$\Rightarrow \boxed{df=267}$

e) skew-t distributions (all equal): Volume $=1$, Shape $=9$, Orientation: 45, $a_k=40$, $\pi_k=3$, $\lambda_k=10$, $\nu_k=1$
$\Rightarrow \boxed{df=109}$

Summary:

a) $\rightarrow$ 263
b) $\rightarrow$ 47
c) $\rightarrow$ 303

b) $\rightarrow$ 47

c) $\rightarrow$ 303

d) $\rightarrow$ 267

e) $\rightarrow$ 109