Exercises 6

Giovanni Zurlo

21/11/2021

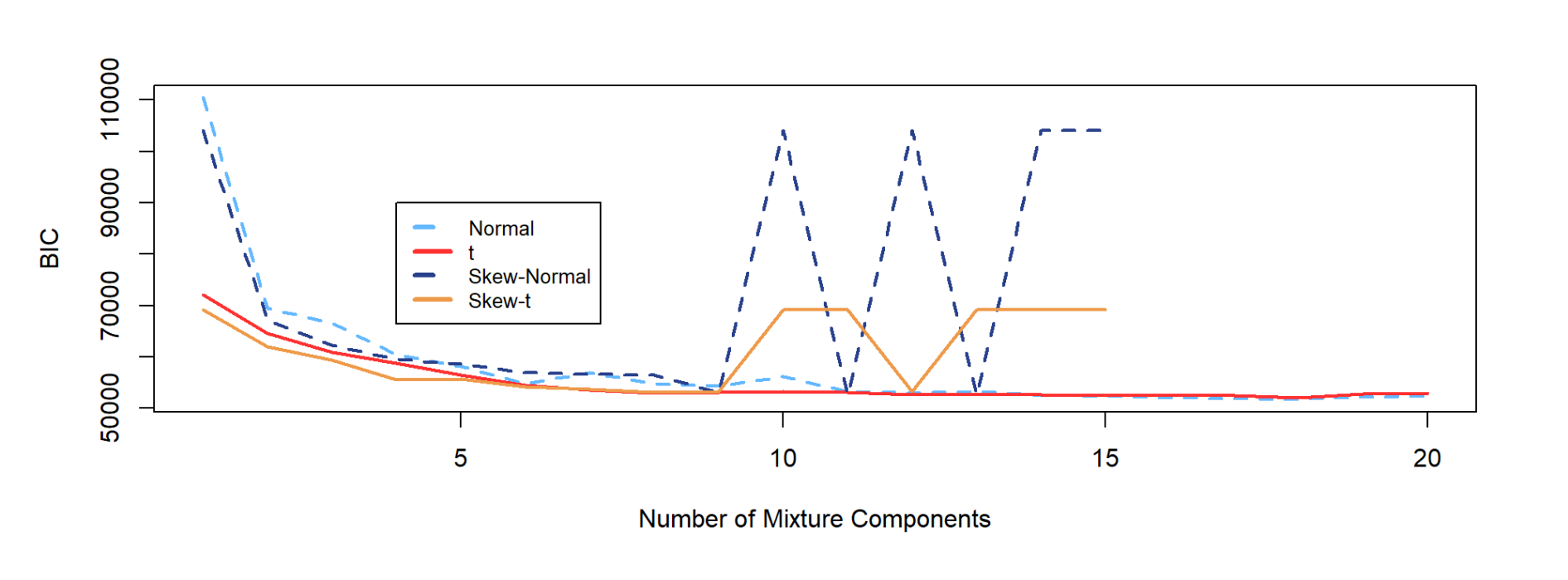
Exercise 1

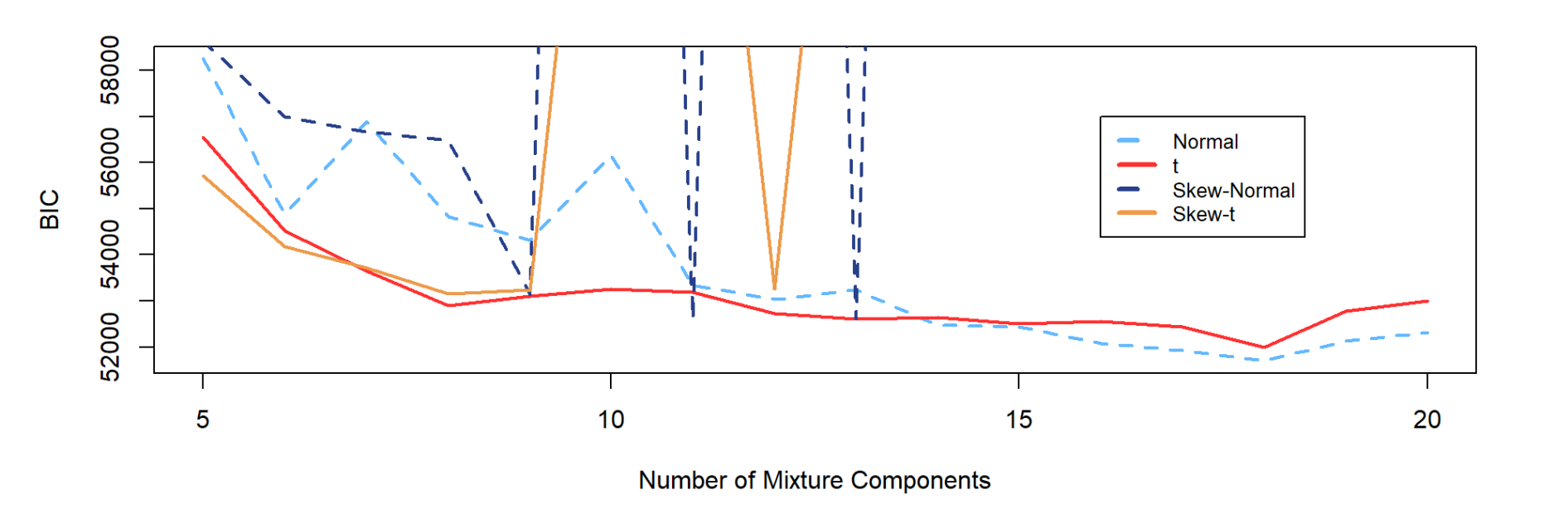
> *Cluster the dataset stars5000.dat using Gaussian mixtures, t-mixtures, skew-normal mixtures, and skew-t mixtures, and decide which clustering you find most convincing, with reasons. Although methods with flexible covariance/shape matrices can in principle handle variables with very different variances, value ranges here are vastly different, and standardisation may help, maybe in a robust manner.*

```
stars<- read.table("stars5000.dat", header=TRUE)
load(file = "Environment3.RData")
str(stars)
```

```
## 'data.frame':    5000 obs. of  6 variables:
##  $ casn  : num  5.85 6.36 7.26 3.41 9.13 ...
##  $ cacont: num  0.399 0.226 0.261 0.301 0.253 ...
##  $ kl1   : num  -2.94 -1.77 -2.69 -3.1 -2.31 ...
##  $ kl2   : num  -0.20434 -0.38198 0.03356 -0.00541 -0.06825 ...
##  $ xh1   : num  1640 1920 1693 1571 1754 ...
##  $ xh2   : num  827 950 920 893 855 ...
```

```
# Robust Positional Standardization ((x-median)/mad)
sstars=clusterSim::data.Normalization(stars,type="n2",normalization="column")
```





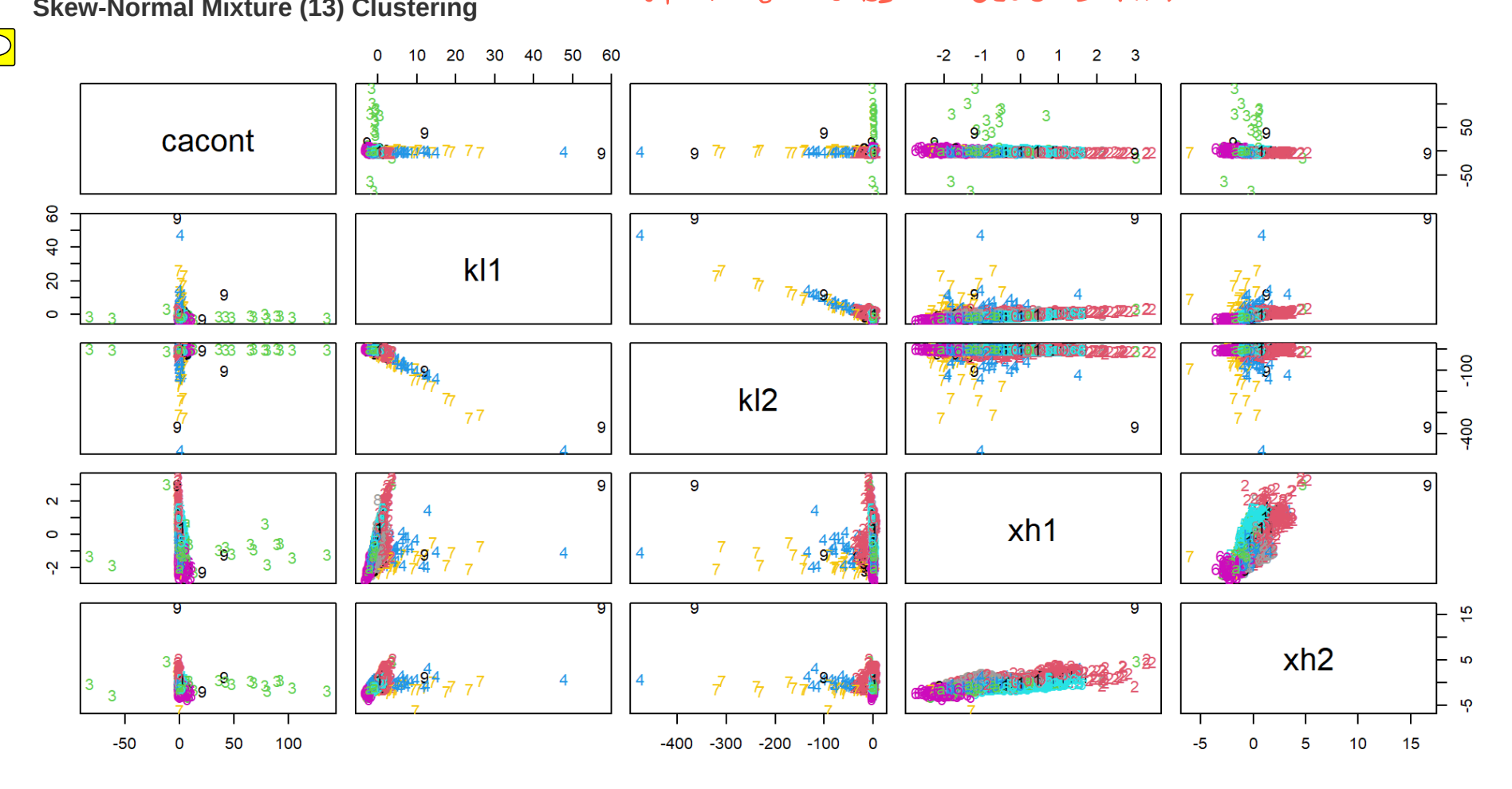
Suggested Optimal Mixtures

	Gaussian	t	Skew-Normal	Skew-t
# of Components	18.00	18.0	13.00	8.00
BIC	51700.16	51993.3	52600.61	53156.49

A robust normalization based on the median and MAD was applied to the data. For each type of density, I fitted mixtures with up to 15 or 20 components; the best ones are reported in the table above. First, we can observe that mixtures of skewed distribution are able to fit data better with a lower number of components, but they're also more difficult to estimate: "spikes" in the BIC series indicate that constraints on the covariance matrices were introduced in order to obtain the convergence of the E-M Algorithm. The maximum number of allowed iterations was set to 5000 (5x the default value) since otherwise most skew-t and skew-normal models flagged a convergence error (*"1"*) in the output list.
It is important to recall that BIC is somehow unreliable when choosing between mixtures of different shapes, and it is biased towards a higher number of components due to overclustering of data. I then followed a visual validation.

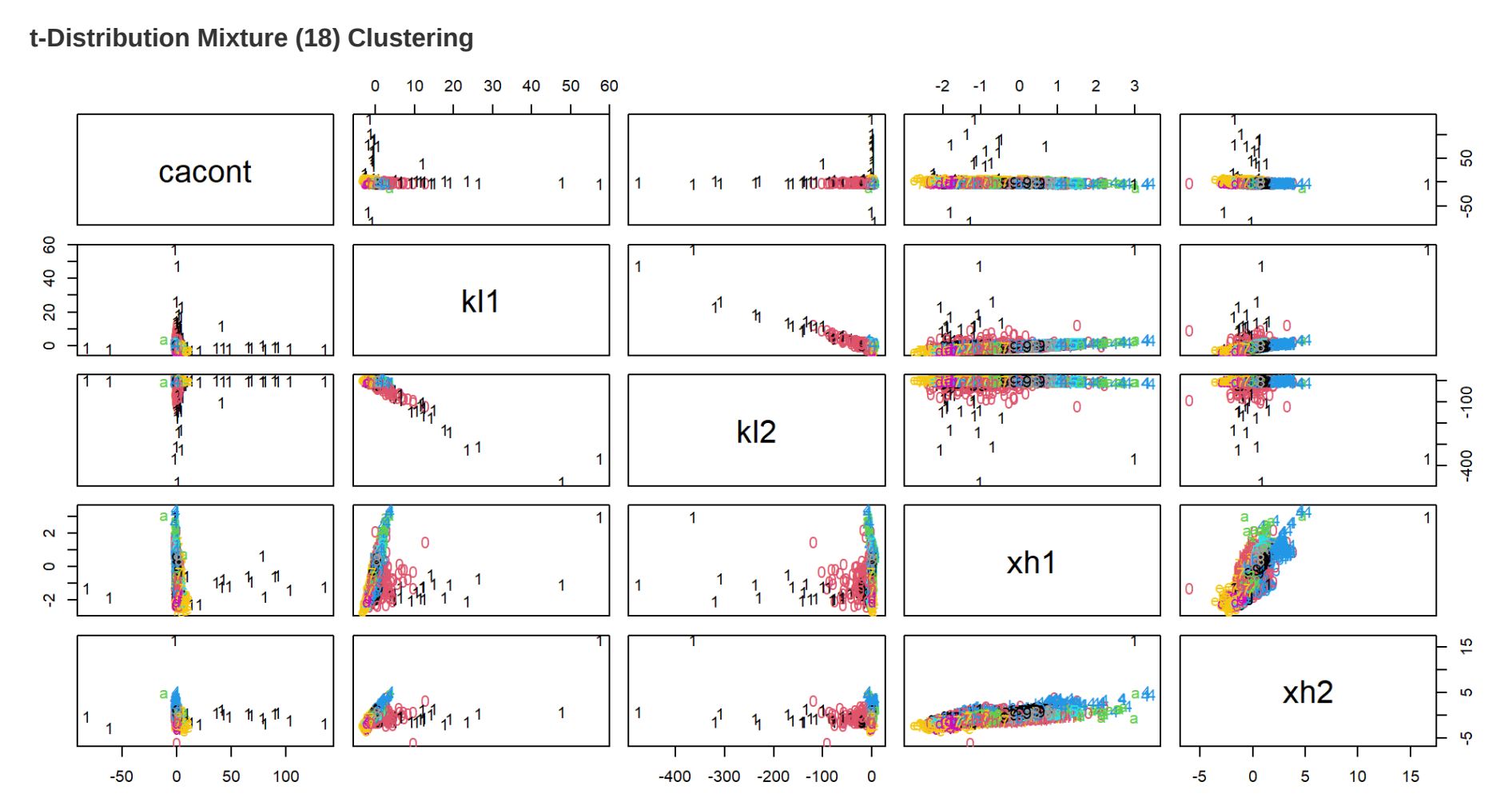
Skew-Normal Mixture (13) Clustering

UP TO 8 COLORS → CLUCOLS(...)



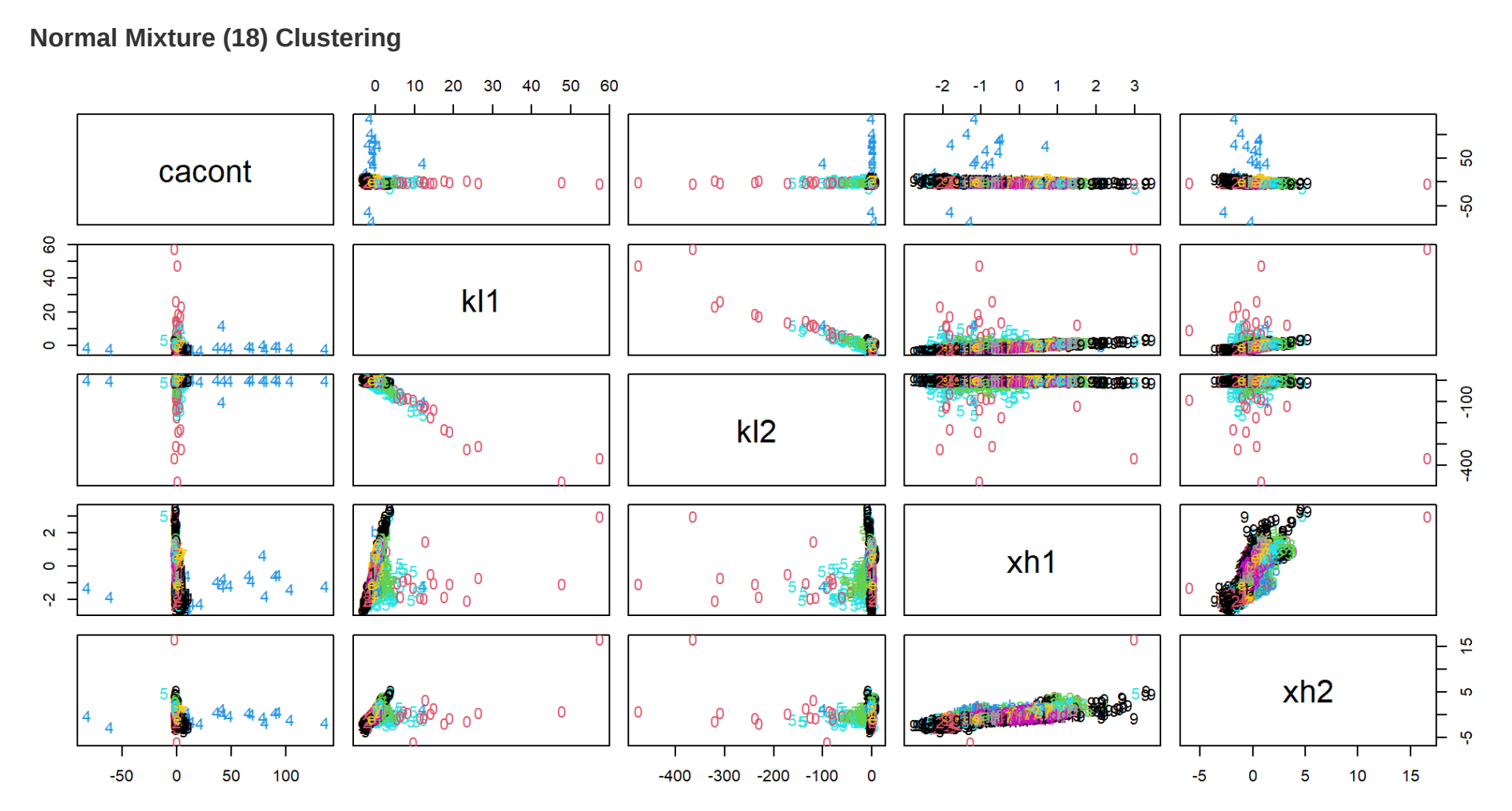
	1	2	3	4	5	6	7	8	9	10	11	12	13
Frequency	629	214	17	26	624	346	35	367	9	779	534	535	885

t-Distribution Mixture (18) Clustering



	1	2	3	4	5	7	8	9	10	11	12	13	14	15	16	17	18
Frequency	39	352	361	267	198	438	460	748	127	140	256	173	221	208	583	43	386

Normal Mixture (18) Clustering



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Frequency	637	234	106	20	48	382	433	306	60	18	104	127	421	638	248	450	280	488

Skewed distributions mixtures proved to be much more flexible than their symmetric counterparts. As you can see from the first pairs plot (*casn* variable was excluded), extreme values fall in four different small clusters (**3-4-7-9**). On the other hand, with a 18 t-distributions mixture model, these are all categorized in a "noise cluster" (**1**) while overclustering occurs in the core group of points. I would avoid such clustering since it seems unable to differentiate among outliers in different dimensions.
The 18 Normals Mixture at least categorizes them in two clusters (**4** and **10**) and leads to the lowest observed BIC. I would personally prefer the Skew-Normal or a Skew-t Mixture Model since they seem more suited to this kind of data, despite higher BIC values and some fitting issues (to be solved maybe with alternative initialization techniques or more specific parameters constraints).

Exercise 2

> *In a situation with 10 variables and 4 mixture components, what is the number of free parameters for …*

a *"VVV"* Gaussian MM assuming fully flexible covariance matrices

```
p = 10
K = 4

(K-1) + K*(p + p*(p+1)/2)
```

```
## [1] 263
```

a *"VII"* Gaussian MM assuming spherical covariance matrices

```
# Only one free parameter in each spherical covariance matrix
(K-1) + K*(p + 1)
```

```
## [1] 47
```

an *"EEE"* Gaussian MM with flexible covariance matrix assumed equal

```
# Only one free flexible covariance matrix common to each mixture component
(K-1) + K*p + p*(p+1)/2
```

```
## [1] 98
```

a Fully Flexible Skew-Normal Mixture

```
# We only add the number of delta skewness parameters
(K-1) + K*(p + p*(p+1)/2) + K*p
```

```
## [1] 303
```

a Fully Flexible Mixture of Multivariate *t* Distributions

```
# We have 4 degrees of freedom parameters
(K-1) + K*(p + p*(p+1)/2) + K
```

```
## [1] 267
```

a Fully Flexible Mixture of Skewed *t* Distributions

```
# We add 4 10-dimensional skewness parameters vectors
(K-1) + K*(p + p*(p+1)/2) + K*p + K
```

```
## [1] 307
```

a Mixture of Skew-*t* distributions with equal skewness parameters, degrees of freedom and Σ-matrices

```
# Several constraints allow us to reduce the number of free parameters by 2/3
((K-1) + K*p + p*(p+1)/2 + p + 1)
```

```
## [1] 109
```

Exercise 3

> Consider the following density of a mixture of two one-dimensional uniform distributions:
>
> $$f_\eta(x) = \pi u_{[a_1,b_1]}(x) + (1-\pi)u_{[a_2,b_2]}(x)$$
>
> where
>
> $$\eta = (\pi, a_1, a_2, b_1, b_2), 0 < \pi < 1, a_1 < b_1, a_2 < b_2$$
>
> Prove that the parameters of this model are not identifiable by proposing parameter vectors η1 and η2 such that:
>
> $$f_{\eta 1}(x) = f_{\eta 2}(x)$$

Given two parameter vectors:

$$\eta_1 = (\pi_1 = \frac{1}{3}, \frac{1}{4}, \frac{1}{2}, \frac{1}{2}, 1) and \eta_2 = (\pi_2 = \frac{2}{3}, \frac{1}{4}, \frac{3}{4}, \frac{3}{4}, 1)$$

It can be proven that, for all x:

$$f_{\eta 1}(x) = f_{\eta 2(x)}$$

$$\frac{1}{3} \cdot u_{[0.25,0.5]}(x) + (1 - \frac{1}{3}) \cdot u_{[0.5,1]}(x) = \frac{2}{3} \cdot u_{[0.25,0.75]}(x) + (1 - \frac{2}{3}) \cdot u_{[0.75,1]}(x)$$

$$\frac{4}{3} \cdot 1(x \in [\frac{1}{4}, \frac{1}{2}]) + \frac{4}{3} \cdot 1(x \in [\frac{1}{2}, 1]) = \frac{4}{3} \cdot 1(x \in [\frac{1}{4}, \frac{3}{4}] + \frac{4}{3} \cdot 1(x \in [\frac{3}{4}, 1]))$$

Exercise 4

> *Implement the big data method explained above, and apply it to the stars5000 data from Exercise 1. Use ns = 1000, take the time for this method's execution. Also take the time for running Mclust on those data. Compare the running times and the results of the big data method and of standard Mclust (it would be a good result for the big data method if results are very similar, but the run time is much faster).*

I wrote a function for fitting mixture models to large datasets:

```
MclustBD <-function(x, ns = 2000, seed = 1234, showProgress = F, Summary = T,...){
  # The ... arguments are passed on to the Mclust function
  # Setting a seed is required for reproducibility
  # Some default values are specified for seed and subset size
  require(mclust)
  start_time <- Sys.time()
  set.seed(seed)
  # Draw a random subset of ns observations
  index <- sample(1:nrow(x), size=ns)
  rsubs <- x[index,]
  extsubs <- x[-index,]
  # Fitting Mclust to the subset
  fit=Mclust(rsubs,verbose = showProgress,...)
  # Extending the fitted model to all observations
  pred <- predict.Mclust(fit,extsubs)
  end_time <- Sys.time()
  # Printing Running Time & Summary
  if(showProgress == T){
  print(paste("Running Time: ", round(end_time - start_time,3)),quote = F)}
  if(Summary == T){print(summary(fit))}

  # Returning a list with comprehensive results
  out <- list()
  out$classification <- integer(nrow(x))
  out$classification[index] <- fit$classification
  out$classification[-index] <- pred$classification
  out$probability <- matrix(nrow = nrow(x),ncol = fit$G)
  out$probability[index,] <- fit$z
  out$probability[-index,] <- pred$z
  out$fit <- fit
    return(out)}
```

Mclust on a random subset (*n = 1000*)

```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust VVV (ellipsoidal, varying volume, shape, and orientation) model with 7
## components:
##
##  log-likelihood    n  df      BIC       ICL
##       -5155.595 1000 195 -11658.2 -11813.38
##
## Clustering table:
##   1   2   3   4   5   6   7
##  99 116 187 179 168 233  18
```

```
## [1] Overall Predictions (Out-of-Sample = 4000):
```

```
##
##          1    2    3    4    5    6    7
##   [1,]  520  593  944  822  883 1165   73
```

Mclust on the entire dataset (*n = 5000*)

```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust VVV (ellipsoidal, varying volume, shape, and orientation) model with 15
## components:
##
##  log-likelihood    n  df       BIC       ICL
##       -24202.19 5000 419 -51973.09 -54039.68
##
## Clustering table:
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
## 249 477 259 488 109 457 400 328 288 358 434 342 405 369  37
```

```
# Computing ARI to compare the obtained clusterings
adjustedRandIndex(subs.fit$classification,full$classification)
```

```
## [1] 0.4018085
```

Optimal Gaussian Mixture - Subset vs. Full

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	0	54	0	83	2	0	0	0	0	2	1	378	0	0
2	118	0	203	0	0	249	1	0	16	0	0	0	4	2	0
3	0	15	1	1	0	180	337	0	53	2	347	0	5	3	0
4	0	419	0	0	0	0	37	0	0	356	0	0	8	2	0
5	131	43	1	0	0	25	25	111	187	0	0	0	0	360	0
6	0	0	0	487	0	0	0	217	32	0	85	341	1	2	0
7	0	0	0	0	26	1	0	0	0	0	0	0	9	0	37

Results are quite different… Standard Mclust function selected a *'VVV'* mixture model with *K = 15* while the big data method missed some of them, identifying 7 components and returning much worse results. Things don't change significantly if we force both function to fit a specific covariance model. Let's now compare the computational effort (as measured by the elapsed time) when fitting a 15-components *'VVV'* model:

```
# I integrated a running time measure in the function above
subs.fit15 <- MclustBD(sstars, G=15, modelNames="VVV", ns=1000, showProgress = T, Summary = F)
```

```
## [1] Running Time:  1.111
```

```
# Running time as difference between system time before & after the execution
start_time <- Sys.time()
full15 <- Mclust(sstars, G=15, modelNames="VVV", verbose = F)
end_time <- Sys.time()
print(paste("Running Time: ", round(end_time - start_time,3)), quote=F)
```

```
## [1] Running Time:  7.159
```

We can see how remarkable is the time saving, already with small samples. Let's now repeat the comparison by doubling the size of the random sample *(ns = 2000)*:

```
# I integrated a running time measure in the function above
subs.fit15 <- MclustBD(sstars, G=15, modelNames="VVV", ns=2000, showProgress=T, Summary=F)
```

```
## [1] Running Time:  5.089
```

```
# Running time as difference between system time before & after the execution
start_time <- Sys.time()
full15 <- Mclust(sstars, G=15, modelNames="VVV", verbose = F)
end_time <- Sys.time()
print(paste("Running Time: ", round(end_time - start_time,3)), quote=F)
```

```
## [1] Running Time:  6.579
```

```
# Computing ARI to compare the obtained clusterings
adjustedRandIndex(subs.fit$classification,full$classification)
```

```
## [1] 0.4018085
```

Using ns = 2000 does not win that much time while still leading to pretty different, and potentially worse, clustering results.