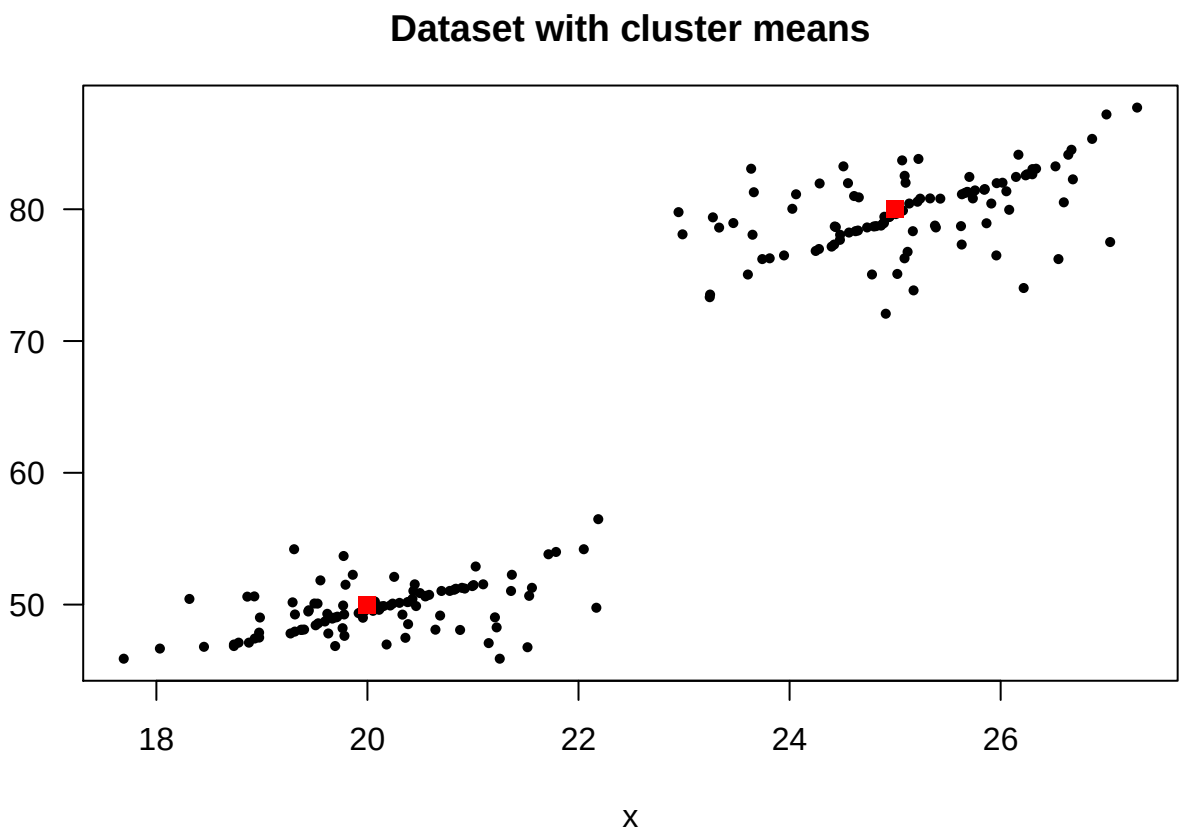
Overfitting Analysis Mclust

Renato Frey - renato.frey@unibas.ch

12 June, 2019

Step 1: Generate 2d-dataset with two obvious clusters.

```
set.seed(123)

m_x1 <- 20
m_x2 <- 25
m_y1 <- 50
m_y2 <- 80
means <- data.frame(x=c(m_x1, m_x2), y=c(m_y1, m_y2))

x1 <- sort(rnorm(100, mean=m_x1, sd=1))
y1 <- sort(rnorm(100, mean=m_y1, sd=2))
ind1 <- sample(1:100, 50)
ind2 <- sample(1:100, 50)
y1[ind1] <- y1[ind2]

x2 <- sort(rnorm(100, mean=m_x2, sd=1))
y2 <- sort(rnorm(100, mean=m_y2, sd=3))
ind3 <- sample(1:100, 50)
ind4 <- sample(1:100, 50)
y2[ind3] <- y2[ind4]

x <- c(x1, x2)
y <- c(y1, y2)
d <- cbind(x,y)
write.csv(d, file="simdat.csv")

par(mar=c(4,3,3,1))
plot(d, las=1, pch=16, cex=.7, main="Dataset with cluster means")
points(means, pch=15, cex=1.2, col="red")
```

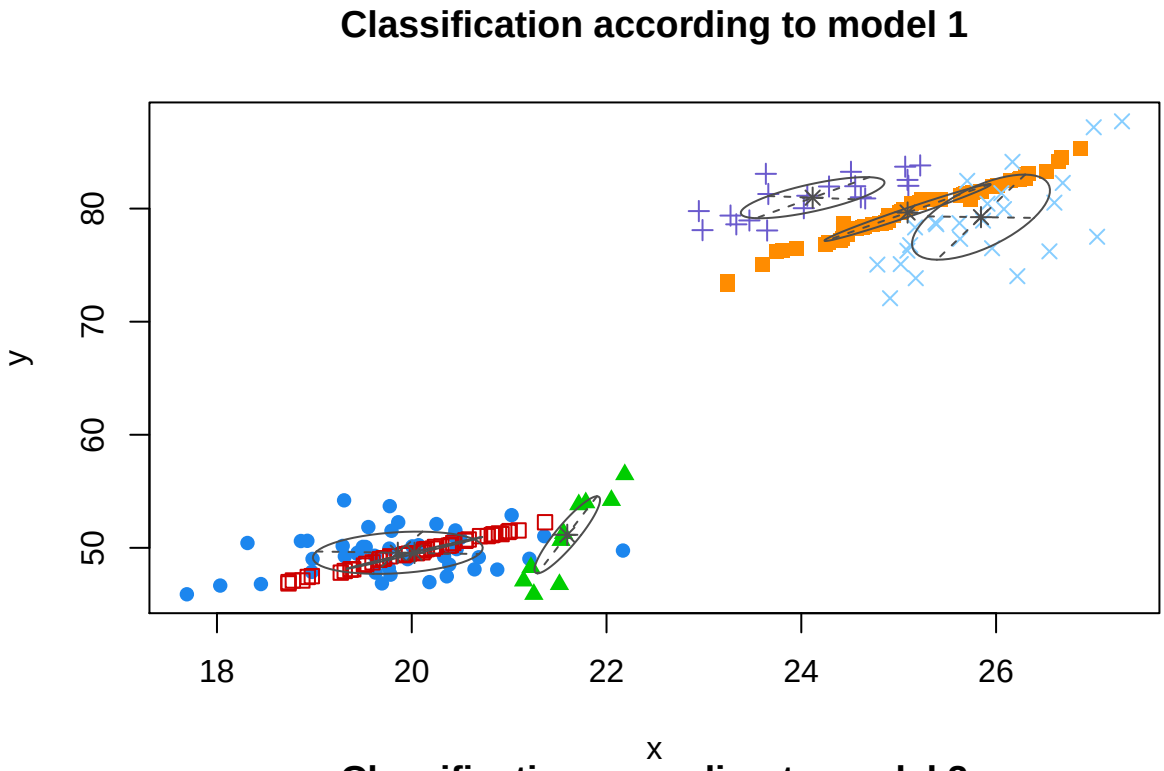


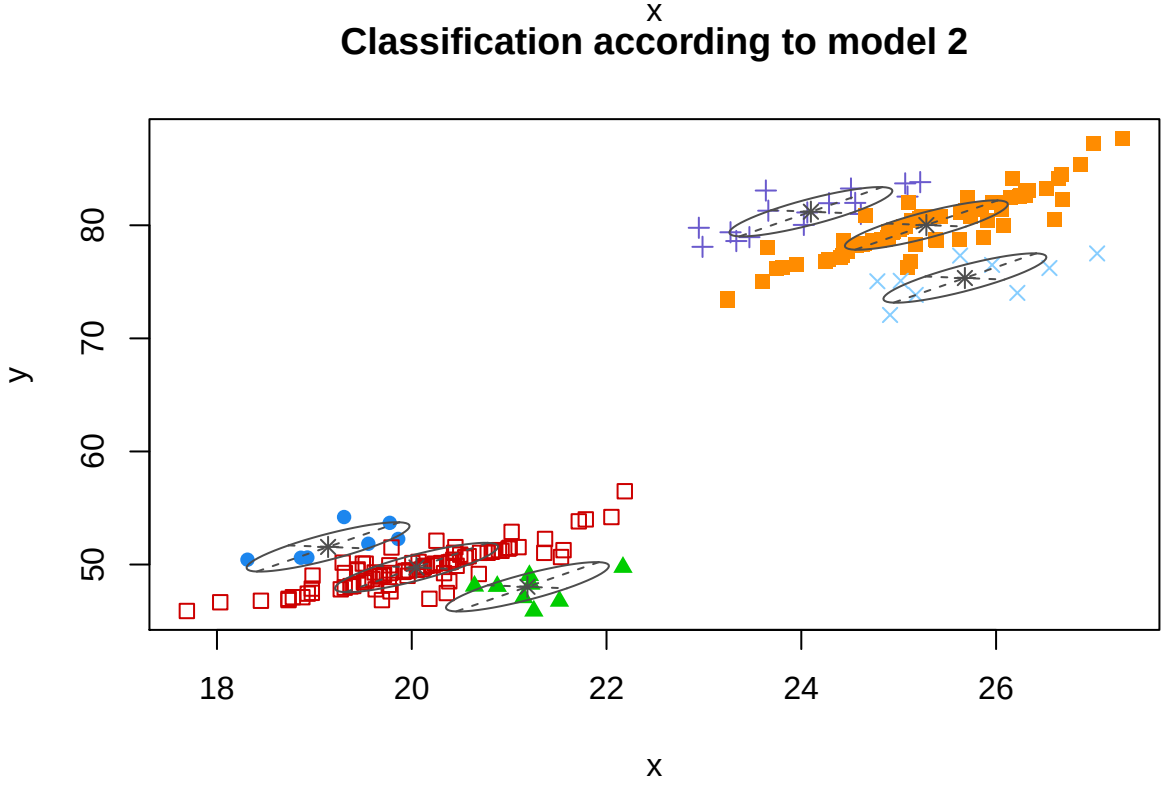
Step 2: Use Mclust to fit three different GMMs.

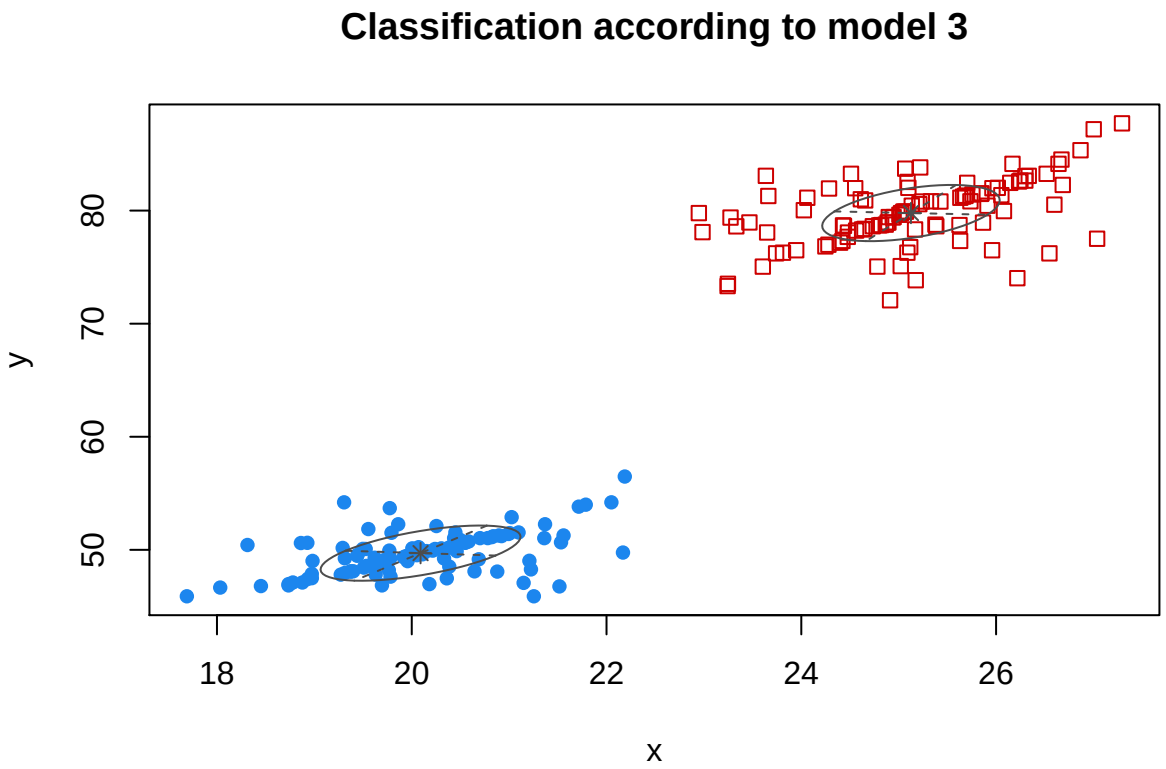
```
library(mclust)
```

```
## Package 'mclust' version 5.4.3
## Type 'citation("mclust")' for citing this R package in publications.
```

```
m1 <- Mclust(d, G=1:10)
m2 <- Mclust(d, G=1:10, modelNames=c("EEE"))
m3 <- Mclust(d, G=1:10, modelNames=c("EEV"))

# save the cluster means
cmeans_m1 <- t(summary(m1)$mean)
cmeans_m2 <- t(summary(m2)$mean)
cmeans_m3 <- t(summary(m3)$mean)

for (i in 1:3) {
  plot(get(paste("m", i, sep="")), what="classification")
  title(paste("Classification according to model", i))
}
```


Classification according to model 1
y
50
60
70
80
18
20
22
24
26
x

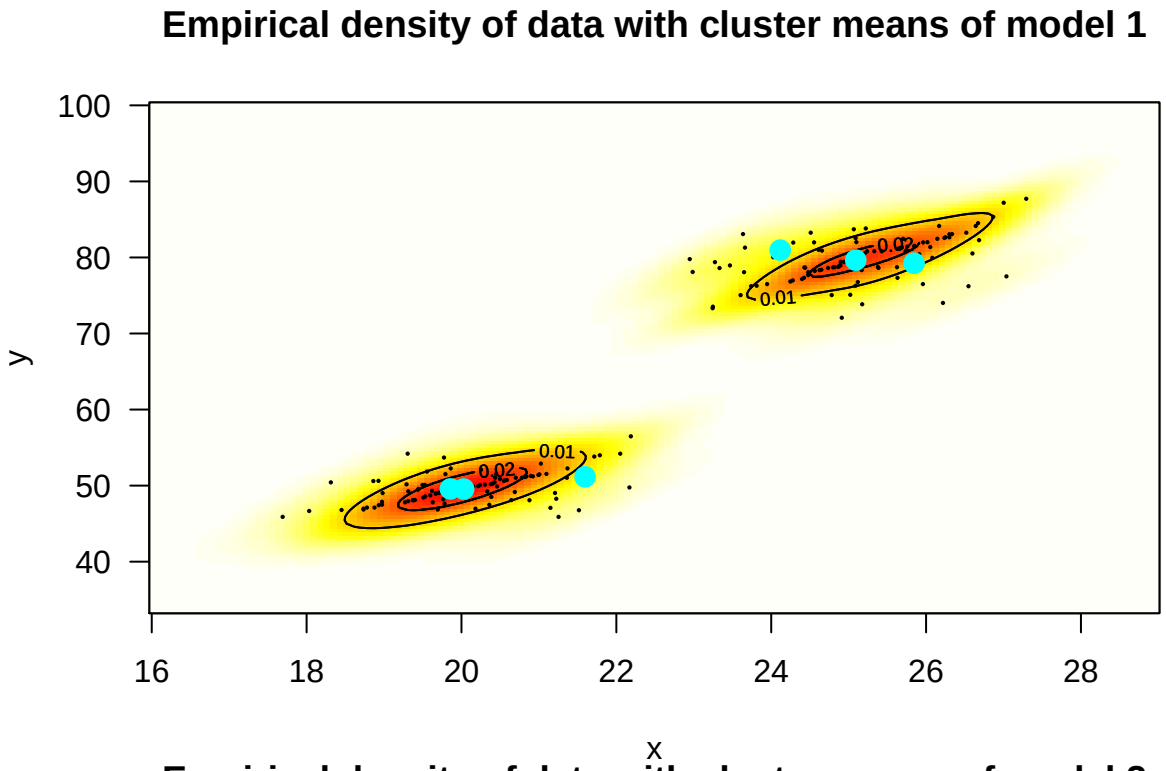

Classification according to model 2
y
50
60
70
80
18
20
22
24
26
x

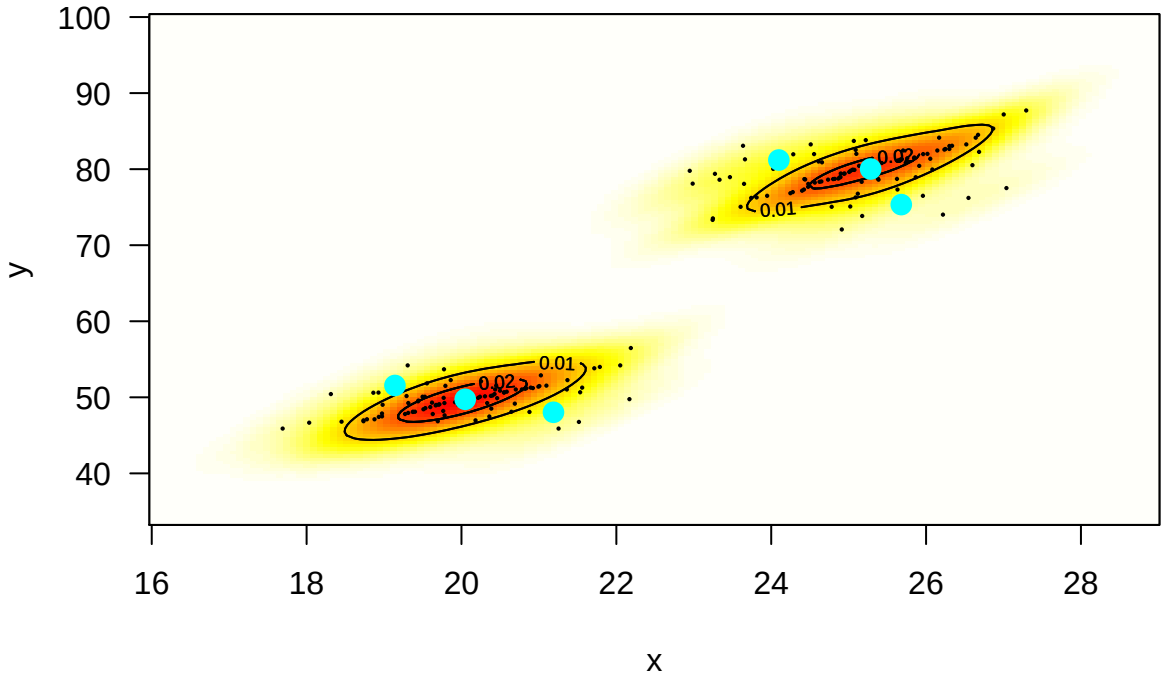


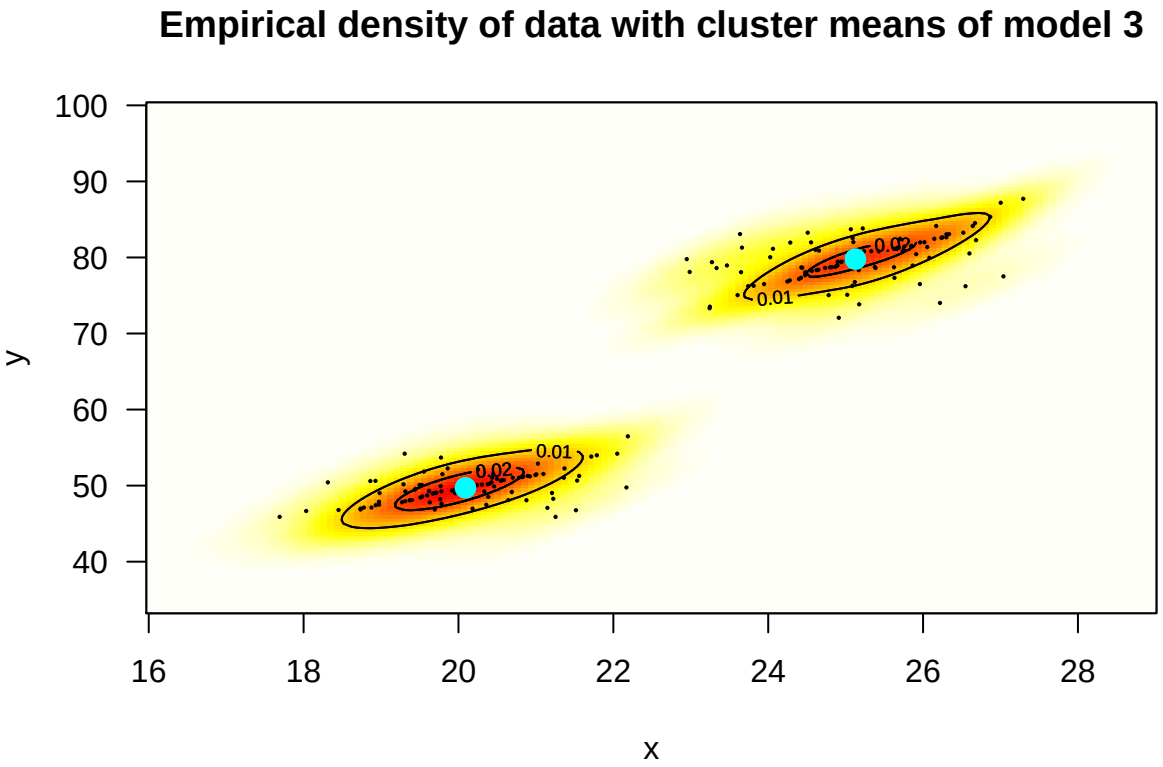
For models 1 and 2, the plots above show that too many clusters are estimated, with several cluster means being very close to each other (small euclidean distance between cluster means).

Step 3: Use a Gaussian kernel to estimate the empirical density in the dataset.

```
library(ks)
dens <- kde(d, binned=F)

# save the points where the density was evaluated (will be needed below)
evalp <- expand.grid(dens$eval.points[[1]], dens$eval.points[[2]])
colnames(evalp) <- c("x", "y")

# predict the density at the cluster means
dens_cmeans_m1 <- predict(dens, x=cmeans_m1)
dens_cmeans_m2 <- predict(dens, x=cmeans_m2)
dens_cmeans_m3 <- predict(dens, x=cmeans_m3)

# plot the empirical density and add the cluster means
for (i in 1:3) {
  plot(dens, display="image", las=1,
       main=paste("Empirical density of data with cluster means of model", i))
  points(d, pch=16, cex=0.3, col="black")
  myconts <- round(seq(0.0001, max(dens$estimate), length.out=8), 2)
  plot(dens, col="black", add=T, abs.cont=myconts)
  points(get(paste("cmeans_m", i, sep="")), col="cyan", pch=16, cex=1.5)
}
```


Empirical density of data with cluster means of model 1
100
90
80
70
60
50
40
y
16
18
20
22
24
26
28
x
0.01
0.02
0.01
0.02


Empirical density of data with cluster means of model 2
100
90
80
70
60
50
40
y
16
18
20
22
24
26
28
x
0.01
0.02
0.01
0.02



Step 4: Generate a null-model to estimate the "saturation" of the identified clusters: specifically, shuffle the dataset 100 times (by column, that is, keeping the marginal distributions of each variable constant, but removing any clusters).

```
null_m1 <- NULL
null_m2 <- NULL
null_m3 <- NULL
null_model_z <- NULL
for (i in 1:100) {

  # shuffle dataset
  d_shf <- apply(d, 2, sample)

  # estimate density
  dens_shf <- kde(d_shf, binned=F)

  # get (and store) the density at the cluster centers
  shf_cmeans_m1 <- predict(dens_shf, x=cmeans_m1)
  null_m1 <- rbind(null_m1, shf_cmeans_m1)

  shf_cmeans_m2 <- predict(dens_shf, x=cmeans_m2)
  null_m2 <- rbind(null_m2, shf_cmeans_m2)

  shf_cmeans_m3 <- predict(dens_shf, x=cmeans_m3)
  null_m3 <- rbind(null_m3, shf_cmeans_m3)

  # get (and store) the density at each actual datapoint
  shf_z <- predict(dens_shf, x=evalp)
  null_model_z <- rbind(null_model_z, shf_z)
}
```

Step 5: Determine a "non-parametric" p-value for each cluster, that is, the proportion of simulation runs in which the density at the cluster centers were LARGER (under the null-model) than the density of the clusters centers in the actual dataset

```
pvals_m1 <- rowMeans(apply(null_m1, 1, function(x) {x > dens_cmeans_m1}))
pvals_m2 <- rowMeans(apply(null_m2, 1, function(x) {x > dens_cmeans_m2}))
pvals_m3 <- rowMeans(apply(null_m3, 1, function(x) {x > dens_cmeans_m3}))
print(pvals_m1)
```

```
## [1] 0.00 0.00 0.00 0.07 0.00 0.00
```

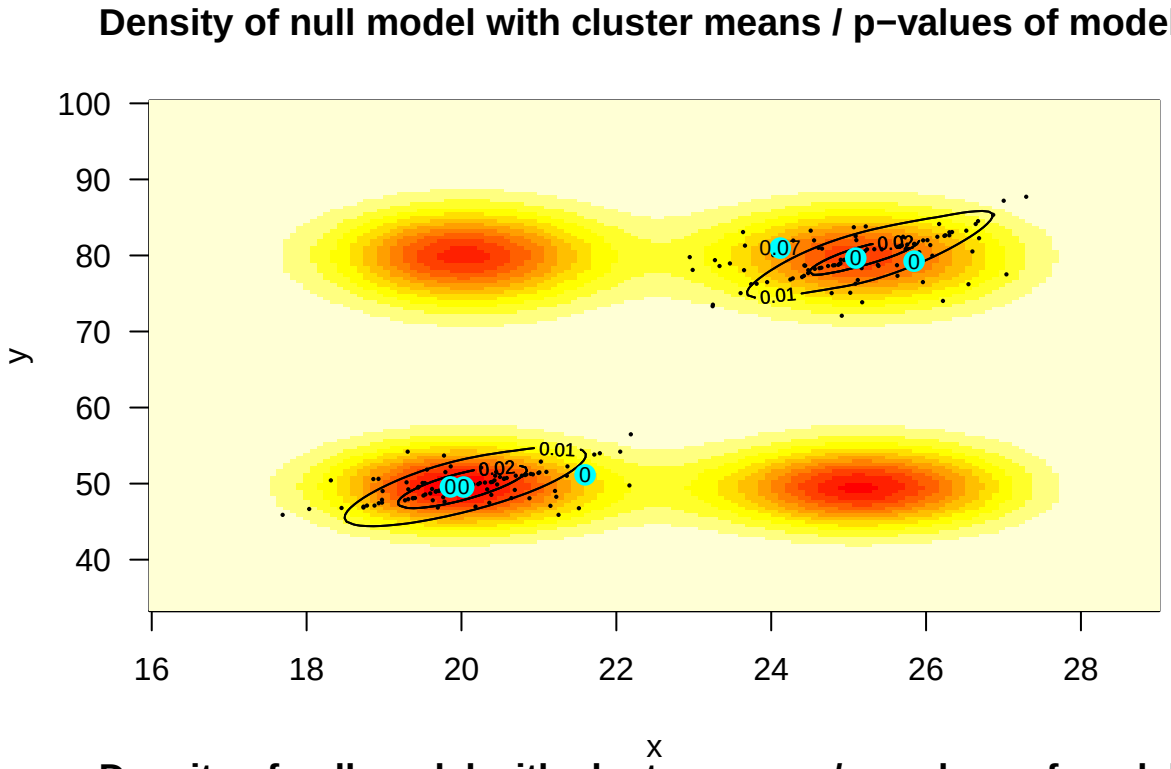
```
print(pvals_m2)
```

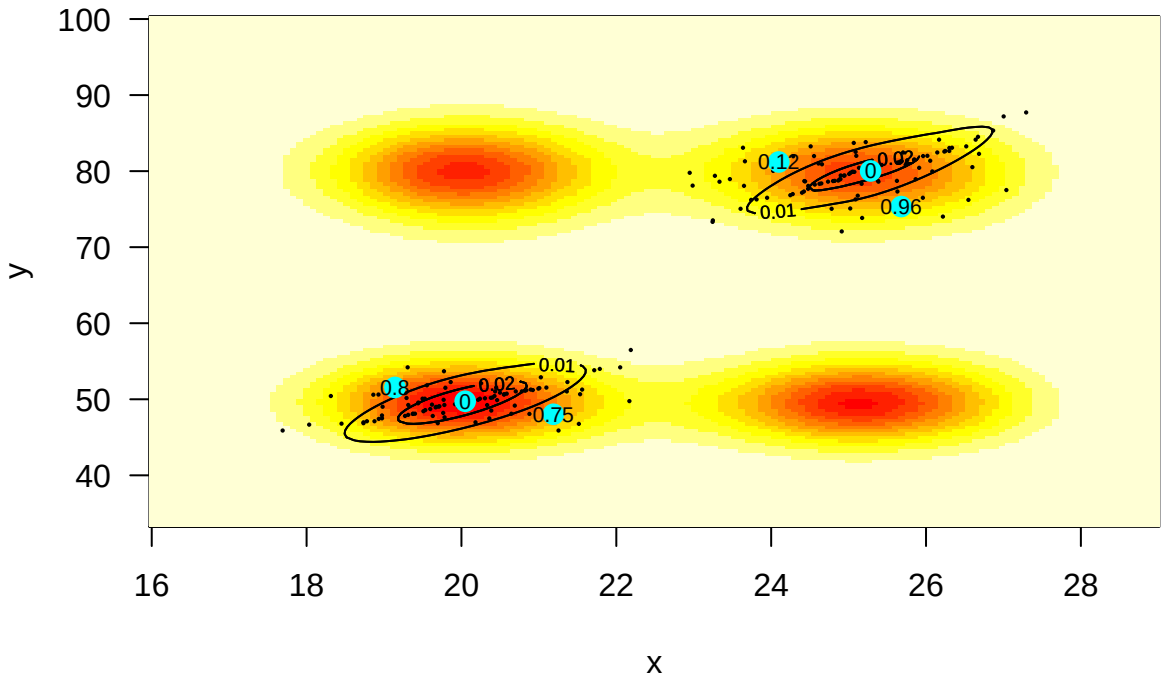
```
## [1] 0.80 0.00 0.75 0.12 0.00 0.96
```

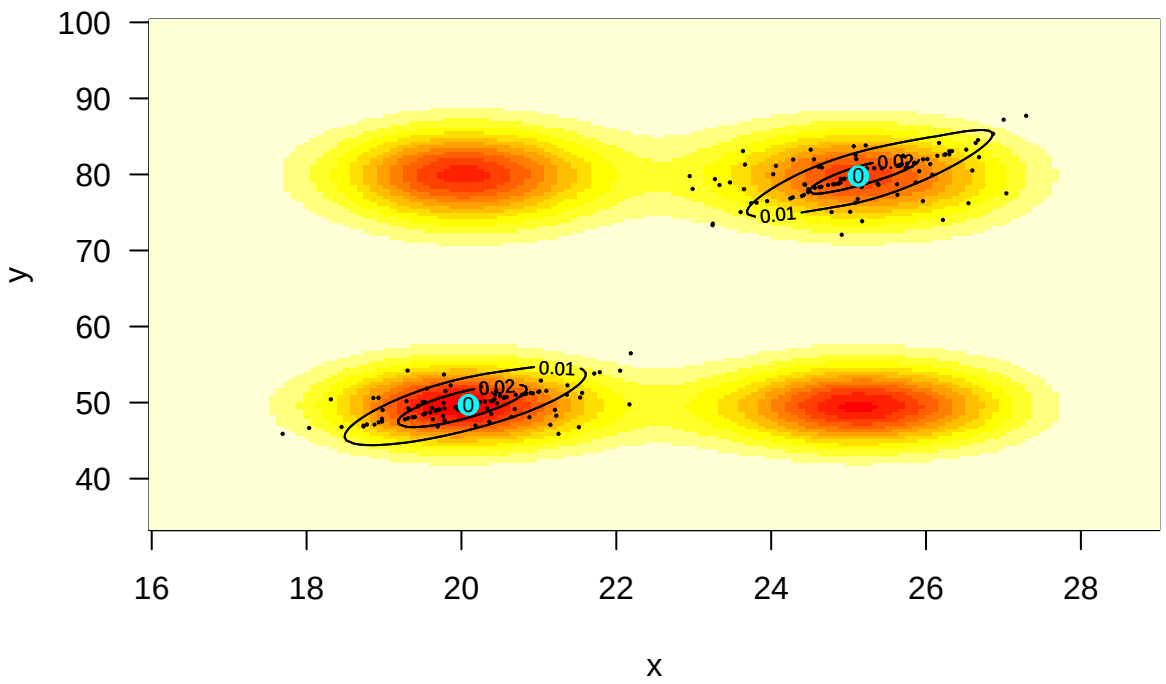
```
print(pvals_m3)
```

```
## [1] 0 0
```

```
# get the average density across all the shuffled datasets (i.e., the null-model)
tmp <- colMeans(null_model_z)
xs <- unique(evalp$x)
ys <- unique(evalp$y)
out <- as.data.frame(matrix(NA, nrow=length(xs), ncol=length(ys)))
rownames(out) <- xs
colnames(out) <- ys
for (k in 1:length(tmp)) {
  out[as.character(evalp$x[k]), as.character(evalp$y[k])] <- tmp[k]
}

# plot the average density add p-values to each of the cluster means
for (i in 1:3) {
  image(x=xs, y=ys, z=1-as.matrix(out), las=1, xlab="x", ylab="y",
        main=paste("Density of null model with cluster means / p-values of model", i))
  points(d, pch=16, cex=0.3, col="black")
  plot(dens, col="black", add=T, abs.cont=myconts)
  points(get(paste("cmeans_m", i, sep="")), col="cyan", pch=16, cex=1.5)
  text(round(get(paste("pvals_m", i, sep="")), 2),
       x=get(paste("cmeans_m", i, sep=""))[,1],
       y=get(paste("cmeans_m", i, sep=""))[,2], cex=.7)
}
```


Density of null model with cluster means / p−values of model 1
100
90
80
70
60
50
40
y
16
18
20
22
24
26
28
x
0.07
0
0.02
0
0.01
0.01
0.02
0
0
0


Density of null model with cluster means / p−values of model 2
100
90
80
70
60
50
40
y
16
18
20
22
24
26
28
x
0.12
0
0.02
0.01
0.96
0.01
0.8
0.02
0
0.75



Interim conclusions:

For model 1 this saturation analysis does not recognize that there exist several spurious clusters, because most of them happen to have means right at the peaks of the empirical density.

For model 2, this analysis works because some of the spurious clusters have means slightly outside the peaks of the empirical density, that is, where the density under the null-model peaks "higher" sufficiently often.

For model 3, the two cluster means are at the peaks of the empirical density, thus correctly resulting in very small p-values.

We next estimate a variational Bayesian gaussian mixture model, as obtained from using the Python-package scikit-learn (the following code illustrates the key steps in Python):

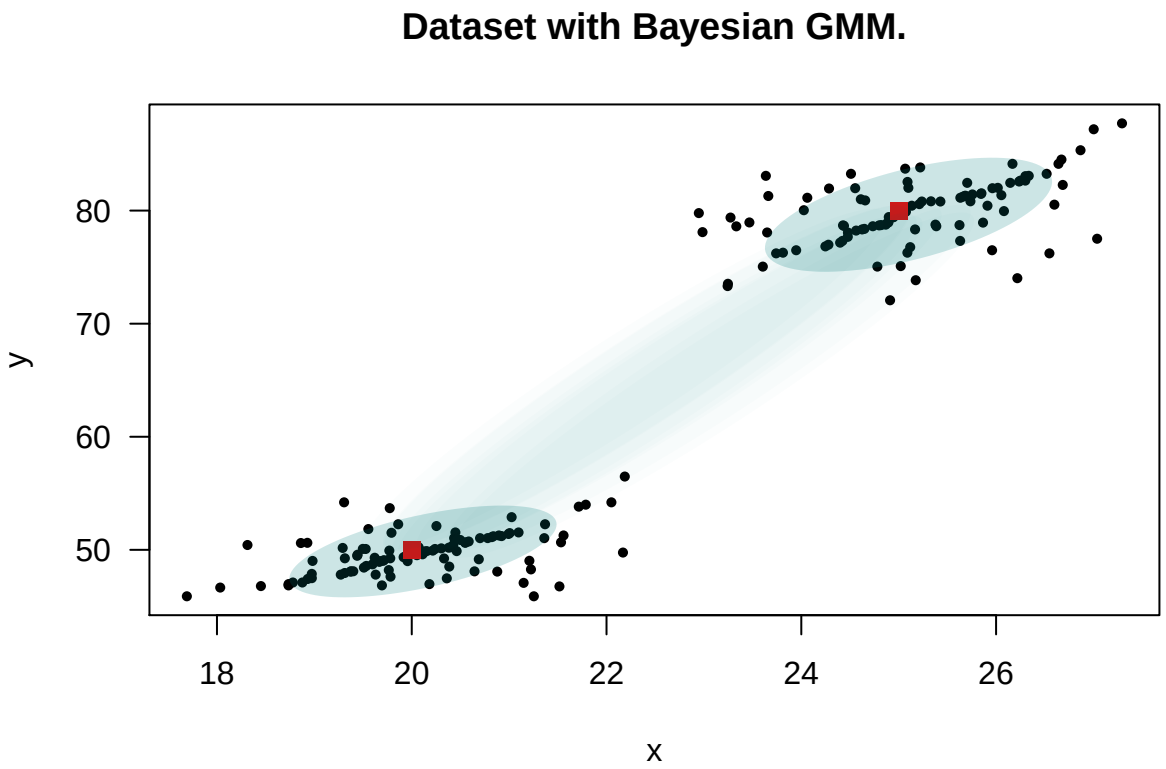
```
  import sklearn.mixture as sklm
  gmm = sklm.BayesianGaussianMixture(n_components = 10,
                                     n_init = 5,
                                     covariance_type = 'full')
  c_means = gmm.means_
  c_cov = gmm.covariances_
  c_weights = gmm.weights_
  likelihood = gmm.lower_bound_
```

The following code reads in and visualizes the identified clusters:

```
plot(d, las=1, pch=16, cex=.7, main="Dataset with Bayesian GMM.")
points(means, pch=15, cex=1.2, col="red")

# read values from BayesianGaussianMixture (scikit-learn)
means <- read.csv("../python/gmm/out/means.csv", header=F)
weights <- unlist(read.csv("../python/gmm/out/weights.csv", header=F))
print(round(weights, 4))
```

```
##    V11    V12    V13    V14    V15    V16    V17    V18    V19   V110
## 0.5022 0.4973 0.0004 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
```

```r
# add constant to visualize all clusters (even the ones we really small weights)
weights <- weights + .04
weights <- weights / max(weights) * .2

if (T) {
  means[3:10,1] <- means[3:10,1] + runif(8, min=-.5, max=.5)
  means[3:10,2] <- means[3:10,2] + runif(8, min=-.5, max=.5)
}

for (i in 1:nrow(means)) {
  covars <- read.csv(paste("../python/gmm/out/covars_c", i-1, ".csv", sep=""),
                     header=F)

  m <- as.numeric(means[i,])
  c <- matrix(rev(unlist(covars)), nrow=nrow(covars))

  e <- eigen(c)
  eig_vecs <- e$vectors
  unit_eig_vec <- eig_vecs[,2] #/ norm(eig_vecs)
  angle <- atan2(unit_eig_vec[2], unit_eig_vec[1])
  angle <- (180 * angle / pi) * - 1
  eig_vals <-  sqrt(2) * sqrt(e$values)


  library(plotrix)
  draw.ellipse(x=m[1], y=m[2], a=eig_vals[1], b=eig_vals[2], angle=angle,
               deg=T, lwd=0.05, col=rgb(0,.5,.5, alpha = weights[i]), border=0)
}
```



In conclusion, even though the variational Bayesian implementation did fit up to 10 clusters (with covariances fully varying) the two correct clusters were recovered with each cluster being assigned a weight close to .5 (and all remaining eight spurious clusters were assigned weights virtually equal to 0).