Mixture Modeling: Mixture of Regressions

Shane Mueller

03/23/2015

# Mixture Modeling

A mixture model is a probabilistic model for representing the presence of sub-populations within an overall population, without requiring that an observed data-set should identify the sub-population to which an individual observation belongs. We will begin by examining mixtures of regression models, which sometimes enable you to identify sets of distinct relationships amongst sub-groups, hidden within a larger population. The goal of mixture modeling is to model your data as a mixture of processes or populations that have distinct patterns of data.

## Example 1:

Suppose we have two processes generating observations: one with a positive relationship between x and y, and a second with a negative. But we don't know which observations came from which group.

1. Random 1000 variables for x and y

set.seed(100)
 x1 <- rnorm(1000)\*50
 x2 <- rnorm(1000)\*50
 y1 <- x1\*3 + 10 +rnorm(1000)\*30
 y2 <- -3\*x2 + 25 + rnorm(1000)\*50
 x <- c(x1,x2)
 y <- c(y1,y2)
par(mfrow=c(1,3))
 plot(x1,y1)
plot(x2,y2)
plot(x,y)



lm(y1~x1)

##
## Call:
## lm(formula = y1 ~ x1)
##
## Coefficients:
## (Intercept) x1
## 9.573 2.976

lm(y2~x2)

##
## Call:
## lm(formula = y2 ~ x2)
##
## Coefficients:
## (Intercept) x2
## 25.196 -3.097

If we try to fit data by using linear regression, we fail:

 plot(x,y)
 lm1 <- lm(y~x)
 summary(lm1)

##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
## Min 1Q Median 3Q Max
## -537.79 -102.55 1.96 102.96 513.75
##
## Coefficients:
## Estimate Std. Error t value Pr(>|t|)
## (Intercept) 18.26748 3.53271 5.171 2.56e-07 \*\*\*
## x 0.08846 0.07026 1.259 0.208
## ---
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 158 on 1998 degrees of freedom
## Multiple R-squared: 0.0007929, Adjusted R-squared: 0.0002928
## F-statistic: 1.586 on 1 and 1998 DF, p-value: 0.2081

 abline(lm1$coefficients,col="green",lwd=3.0)



And from summary(lm1), R-squared = 0.0008135

summary(lm1)

##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
## Min 1Q Median 3Q Max
## -537.79 -102.55 1.96 102.96 513.75
##
## Coefficients:
## Estimate Std. Error t value Pr(>|t|)
## (Intercept) 18.26748 3.53271 5.171 2.56e-07 \*\*\*
## x 0.08846 0.07026 1.259 0.208
## ---
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 158 on 1998 degrees of freedom
## Multiple R-squared: 0.0007929, Adjusted R-squared: 0.0002928
## F-statistic: 1.586 on 1 and 1998 DF, p-value: 0.2081

## Fitting mixtures of regressions using E-M and flexmix

The situation here is nearly the same as for model-based clustering. But instead of having a lot of DVs that we are fitting with a mean and sd for each observation, we have a relationship between IVs and a DV. But in this case, as long as we can produce a ML estimate for a model, we can use the E-M algorithm to identify and estimate the groups.

Reveiw: E-M stands for Expectation-Maximization. It is an iterative process whereby you apply two complementary processes. Suppose you assume there are two sub-groups. First, randomly assign members to either of the sub-groups. Next, you compute a maximum likelihood (ML) estimate for each sub-group. Now, because of random variation, there are likely to be members of one group that are better described by the other group (in terms of likelihood). You then apply the 'maximization' step to re-sort data into the group that better describes them. You repeat this process, re-estimating your models and re-sorting members until nobody is better described by a model that they are not in. Original research on the E-M algorithm proved that it would converge to a local maximum; you might not get the global maximum however, so you typically repeat the process many times and pick the best outcome you find.

This is possible to do by hand, but the 'flexmix' package allows us to do this very easily.

### Flexmix modeling:

FlexMix implements a general framework for fitting discrete mixtures of regression models in the R statistical computing environment

We can use flexmix to fit the data as amixture of two processes:

split <- rep(1:2,1000)
plot(x,y)
abline(lm(y[split==1]~x[split==1])$coef)
abline(lm(y[split==2]~x[split==2])$coef)

 library(flexmix)

## Loading required package: lattice



 model <- flexmix(y~x,k=2)
 plot(x,y,col=clusters(model))
 abline(parameters(model)[1:2,1],col="blue",lwd=3)
 abline(parameters(model)[1:2,2],col="green",lwd=3)



plot(model)



summary(model)

##
## Call:
## flexmix(formula = y ~ x, k = 2)
##
## prior size post>0 ratio
## Comp.1 0.494 933 1501 0.622
## Comp.2 0.506 1067 1375 0.776
##
## 'log Lik.' -11251.23 (df=7)
## AIC: 22516.45 BIC: 22555.66

The summary() method gives the estimated prior probabilities, the number of observations assigned to the corresponding clusters, the number of observations where posterior probability > δ for each component (with a default of δ = 10−4 ). This indicates the proportion and number of observations that fit are fit even a little bit by the model, in comparison to those fit best by the model. In the above example, one component has a ratio of around .78--there were 1375 points that had non-zero likelihood of being in that group, and of those 78% were best fit by that group. The distribution of those likelihoods is shown as a rootogram (with the count scale is transformed). Histograms or rootograms of the posterior class probabilities can be used to visually assess the cluster structure. Rootograms are very similar to histograms, the only difference is that the height of the bars correspond to square roots of counts rather than the counts themselves, so that low counts are more visible and peaks less so.

When interpreting rootograms, keep in mind that there is no 'ground truth'. A peak near probability 1 indicates that many of the points are overwhelmingly well-represented by that cluster. A peak near 0 would indicate that many points clearly don't fit the category. Points in the middle indicate a lack of separation--there are points that are only moderately well-described by that group, which should be considered with respect to the number of groups you think exist.

## Example: A mixture of variances

Mixture modeling can be used to identify outliers or select elements that are caused by the same process but have mutch larger variance. Sometimes, skewed distributions with long tails are modeled as mixtures of two gaussians--one to capture the fairly symmetrical controid of the distribution, and a second to capture the long tail. At the end of the day, you may treat the tail as a nuisance and use the centroid model as the one you care about.

1. Random 1000 variables for x and y

 x1 <- rep(1:10,50)
 x2 <- rep(1:10,each=5)
 y1 <- 3\*x1 + 10 +rnorm(500)\*3
 y2 <- 3\*x2 + 10 + rnorm(50)\*50
 x <- c(x1,x2)
 y <- c(y1,y2)
 plot(x,y)



Notice that we have a clear linear relationship, along with a handful of outliers. For a regular maxmimum-likelihood model, these outliers present a problem because they are either very unlikely (punishing the model), or the model's variance must get larger to make them reasonable, making all the other data less likely.

library(flexmix)
model2a <- flexmix(y~x,k=1)
summary(model2a)

##
## Call:
## flexmix(formula = y ~ x, k = 1)
##
## prior size post>0 ratio
## Comp.1 1 550 550 1
##
## 'log Lik.' -2235.205 (df=3)
## AIC: 4476.41 BIC: 4489.34

model2b <- flexmix(y~x,k=2)
summary(model2b)

##
## Call:
## flexmix(formula = y ~ x, k = 2)
##
## prior size post>0 ratio
## Comp.1 0.9012 505 508 0.9941
## Comp.2 0.0988 45 550 0.0818
##
## 'log Lik.' -1649.15 (df=7)
## AIC: 3312.3 BIC: 3342.47

parameters(model2b)

## Comp.1 Comp.2
## coef.(Intercept) 10.178077 1.264176
## coef.x 2.984318 5.765556
## sigma 2.907706 42.810485

clusters(model2b)

## [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 1 1 1 1 1 1 1 1 1 1 1
## [36] 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 1 1 1 1 1 1 1 1 1 1
## [71] 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 1 1 1 1 1 1 1 1 1 1
## [106] 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 1 1 1 1 1 1 1 1 1 1
## [141] 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 1 1 1 1 1 1 1 1 1 1
## [176] 1 1 1 1 1 2 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 1 1 1 1
## [211] 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 1 1 1 1 1 1 1 1 1 1
## [246] 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 1 1 1 1 1 1 1 1 1 1
## [281] 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 1 1 1 1 1 1 1 1 1 1
## [316] 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 1 1 1 1 1 1 1 1 1 1
## [351] 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 1 1 1 1 1 1 1 1 1 1
## [386] 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 1 1 1 1 1 1 1 1 1 1
## [421] 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 1 1 1 1 1 1 1 1 1 1
## [456] 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 1 1 1 1 1 1 1 1 1 1
## [491] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 1 2 2 2 1 2 2 2 2 2 1 2 2 2 1 2 2 2
## [526] 2 2 1 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

In this case, the 2-group model will often select 40/50 of the outliers to be in the secondary group. We'd like to know whether the 2-group solution is any better? We can use a criterion such as BIC to do this.

### Bayesian Information Criterion (BIC)

The BIC is a statistic that is based on both the likelihood of the data AND a punishing factor related to the number of parameters used. If you have three parameters for a default model (intercept, slope, and variance), you would have six for a model with two clusters, and nine for one with three. To use the BIC statistic, you can just compare across models and determine which is the smallest. This model provides the best trade-off between accuracy and complexity. However, sometimes the differences are small and you still may prefer a more complex model for other reasons.

In the above example, the BIC is given as part of the output of each model, or we can get it directly:

BIC(model2a)

## [1] 4489.34

BIC(model2b)

## [1] 3342.47

Here, model2b has a much smaller BIC, so we would prefer this one--the two-group model. We'd like to be able to automate this, looking across a number of options; we'd also like to run this many times (and with more than just 2 groups) and see if there are any better soluetions. Here is a way to do both. In the stepFlexmix function, it allows you to specify the numbers of clusters to test, and the number of repetitions at each configuration. We can then select the one with the smallest BIC using the getModel() function

models <- stepFlexmix(y~x,k=1:5,nrep=10)

## 1 : \* \* \* \* \* \* \* \* \* \*
## 2 : \* \* \* \* \* \* \* \* \* \*
## 3 : \* \* \* \* \* \* \* \* \* \*
## 4 : \* \* \* \* \* \* \* \* \* \*
## 5 : \* \* \* \* \* \* \* \* \* \*

models

##
## Call:
## stepFlexmix(y ~ x, k = 1:5, nrep = 10)
##
## iter converged k k0 logLik AIC BIC ICL
## 1 2 TRUE 1 1 -2235.205 4476.410 4489.340 4489.340
## 2 9 TRUE 2 2 -1649.150 3312.300 3342.470 3364.701
## 3 33 TRUE 3 3 -1648.722 3319.444 3366.853 3902.890
## 4 200 FALSE 4 4 -1645.662 3321.324 3385.973 3957.889
## 5 200 FALSE 5 5 -1645.332 3328.663 3410.552 4188.121

summary(models)

## Length Class Mode
## 1 stepFlexmix S4

plot(models)



model <- getModel(models,"BIC")
model

##
## Call:
## stepFlexmix(y ~ x, k = 2, nrep = 10)
##
## Cluster sizes:
## 1 2
## 45 505
##
## convergence after 9 iterations

plot(model)



summary(model)

##
## Call:
## stepFlexmix(y ~ x, k = 2, nrep = 10)
##
## prior size post>0 ratio
## Comp.1 0.0988 45 550 0.0818
## Comp.2 0.9012 505 508 0.9941
##
## 'log Lik.' -1649.15 (df=7)
## AIC: 3312.3 BIC: 3342.47

parameters(model)

## Comp.1 Comp.2
## coef.(Intercept) 1.265448 10.178074
## coef.x 5.765109 2.984321
## sigma 42.807090 2.907605

clusters(model)

## [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [36] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [71] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [106] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [141] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [176] 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 2 2 2 2 2 2 2 2 2
## [211] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [246] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [281] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [316] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [351] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [386] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [421] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [456] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [491] 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 2 1 1 1 2 1 1 1 1 1 2 1 1 1 2 1 1 1
## [526] 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

This does pretty well. There are a few things to examine. First, we can look at the post>0 ratio. This sets a very small criterion and asks how many of the data points have likelihood greater than that small level (post>0). It then asks how many are placed in the category based on maximum likelihood. For the larger component, we have a high proportion--almost all of the points that are described by the larger component at all are placed in that category. The second component is less good--almost all of the points could be described by that component, but are just better descibed by the other component. This makes sense in our case, because we have two overlapping distributions. Other times, we might want to have mutually-exclusive clusters.

Note that the model estimates completely different parameters for each group; we'd probably want to just estimate a different variance parameter, and to do that we'd need either to do this by hand or use another flexmix model (possibly a custom model)--it might not even be possible.

## Example: learners and non-learners

In this example, suppose that you have tested a group of people repeatedly. Maybe, some people get fatigued in this situation, and so their data get worse over time; another group learn and so their data get better over time. On average, the performance may be flat, but this could hide two separate groups. But these groups might not be labeled, and they may not be easily detectable.

learners <- t(matrix(5+ .5\*(1:10) + 1.5\*rnorm(100),10))
fatigues <- t(matrix(10- .5 \*(1:10) + 1.5\*rnorm(100),10))
data <- rbind(learners,fatigues)
par(mfrow=c(1,3))
matplot(t(learners),pch=16,col="black",main="Learners")
matplot(t(fatigues),pch=16,col="black",main="Fatigued")
matplot(t(data),pch=16,col="black")

 Although we have two different groups, without knowing about their identity, we might just look at the data and declare that they are all the same. Of course, if they were all the same, we would see some that have a positive slope, and some that have a negative slope. Seeing some with positive and some with negative slope doesn't necessarily tell us we have two groups though. For example, here is a a data set created from a single group

data2 <- matrix(5 + rnorm(200)\*2,20)

pars1 <- matrix(0,ncol=2,nrow=20)
pars2 <- matrix(0,ncol=2,nrow=20)
pred <- 1:10
for(i in 1:20)
 {
 pars1[i,] <- lm(data[i,]~pred)$coef
 pars2[i,] <- lm(data2[i,]~pred)$coef
 }

par(mfrow=c(1,2))
plot(pars1,xlab="intercepts",ylab="slopes",main="Two groups")
plot(pars2,xlab="intercepts",ylab="slopes",main="One group")

 By comparing the slope and intercept of each person, we don't see a lot of difference. But notice that when we have two groups, that if we fit a regression to each person, the resulting parameters appear to cluster into two groups, with a gap in the middle. When we have one group, they tend to vary more smoothly. A mixture model should be able to take advantage of this.

Howeever, to do so we need to reshape the data

library(reshape2)
data <- as.data.frame(data)
data$sub <- 1:20
long <- melt(data,id.vars = c("sub"))
long$time <- rep(1:10,each=20)

long[1:10,]

## sub variable value time
## 1 1 V1 7.063576 1
## 2 2 V1 2.823134 1
## 3 3 V1 7.248888 1
## 4 4 V1 4.946792 1
## 5 5 V1 2.429878 1
## 6 6 V1 3.299892 1
## 7 7 V1 8.329819 1
## 8 8 V1 6.503791 1
## 9 9 V1 6.385533 1
## 10 10 V1 2.337430 1

Here, each predictor variable and subject are fitted separately. Now, we can make a repeated-measures regression with the following formula, which will aggregate across subject--exactly what we want. The points of each subject are classified all-or-none into one or the other cluster.

f <- flexmix(value~time|as.factor(sub), data = long, k = 2,
 control = list(iter.max=10))
parameters(f)

## Comp.1 Comp.2
## coef.(Intercept) 4.9070797 10.3819897
## coef.time 0.5356787 -0.5669244
## sigma 1.5453145 1.5904027

clusters(f)

## [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2
## [36] 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
## [71] 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1
## [106] 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
## [141] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2
## [176] 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2

table(clusters(f),long$sub)

##
## 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
## 1 10 10 10 10 10 10 10 10 10 10 0 0 0 0 0 0 0 0 0 0
## 2 0 0 0 0 0 0 0 0 0 0 10 10 10 10 10 10 10 10 10 10

f1 <- flexmix(value~time|as.factor(sub), data = long, k = 1,
 control = list(iter.max=10))

## Example: Curved versus straight models

Mixtures don't always come from the same process or type of model. Here is a one in which one component is linear, and a second is curved.

 x1 <- rnorm(1000)\*5
x2 <- rnorm(1000)\*5
 y1 <- 3\*x1 + 10 +rnorm(1000)\*5
 y2 <- 10+ 5\*x2 + .3\*x2^2 + rnorm(1000)\*5
 x <- c(x1,x2)
 y <- c(y1,y2)
 plot(x,y)



Here, we fit two polynomial models, this might work, because a linear model is just a subset of a polynomial model

model3 <- flexmix(y~poly(x,2),k=2)
summary(model3)

##
## Call:
## flexmix(formula = y ~ poly(x, 2), k = 2)
##
## prior size post>0 ratio
## Comp.1 0.488 829 1865 0.445
## Comp.2 0.512 1171 1860 0.630
##
## 'log Lik.' -6466.389 (df=9)
## AIC: 12950.78 BIC: 13001.19

 plot(x,y,col=c("red","green")[clusters(model3)])

ord <- order(x)
 points(x[ord],predict(model3)[[1]][ord],type="l",col="black",lwd=6)
 points(x[ord],predict(model3)[[1]][ord],type="l",col="red",lwd=3)
 points(x[ord],predict(model3)[[2]][ord],type="l",col="black",lwd=6)
 points(x[ord],predict(model3)[[2]][ord],type="l",col="green",lwd=3)

 We can look at basic statistical tests with the following trick:

model3 <- flexmix(y~poly(x,2),k=2)
rm1 <- refit(model3)
summary(rm1)

## $Comp.1
## Estimate Std. Error z value Pr(>|z|)
## (Intercept) 10.38961 0.27447 37.8528 <2e-16 \*\*\*
## poly(x, 2)1 671.99344 9.90587 67.8379 <2e-16 \*\*\*
## poly(x, 2)2 10.55195 8.51599 1.2391 0.2153
## ---
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1
##
## $Comp.2
## Estimate Std. Error z value Pr(>|z|)
## (Intercept) 18.21285 0.25799 70.594 < 2.2e-16 \*\*\*
## poly(x, 2)1 1151.43156 8.74820 131.619 < 2.2e-16 \*\*\*
## poly(x, 2)2 479.74923 7.91929 60.580 < 2.2e-16 \*\*\*
## ---
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

You can see that for one model, it is quadratic, but for the other the quadratic term is not significant.

## Control parameters

If you want to set a maximum number of iterations = 10 in model3,

>model31 <- flexmix(y~poly(x,2),k=2,control = list(iter.max=10))

Another control parameter is minprior. it is the minimum prior probability that components are enforced to have. Components falling below this threshold (the current default is 0.05) are removed during EM iteration to avoid numerical instabilities for components containing only a few observations. Using a minimum prior of 0 disables component removal. Here is a regression model with two latent classes. We can compare it to the actual class at the end:

data("NPreg")
 m1 <- flexmix(yp ~ x + I(x^2), data = NPreg, k = 4, control = list(minprior = 0.2))
 summary(m1)

##
## Call:
## flexmix(formula = yp ~ x + I(x^2), data = NPreg, k = 4, control = list(minprior = 0.2))
##
## prior size post>0 ratio
## Comp.1 0.521 122 180 0.678
## Comp.2 0.479 78 200 0.390
##
## 'log Lik.' -439.1564 (df=9)
## AIC: 896.3128 BIC: 925.9976

plot(m1)



clusters(m1)

## [1] 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 2 1 1 1 2 2 2 1 2 1 1 1 1 1 2 1 2 2 2 1
## [36] 1 1 2 2 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 2 1 1 1 2 1 2 1 1 1 1 1 1
## [71] 1 1 2 1 1 1 2 1 1 2 1 1 1 2 1 1 1 1 2 1 2 1 1 1 1 1 1 1 1 1 2 1 2 1 2
## [106] 2 2 2 2 2 2 1 1 1 1 2 1 1 1 2 1 2 1 2 1 1 1 2 1 2 2 2 1 1 1 2 2 2 1 2
## [141] 2 2 2 1 1 1 1 1 2 1 2 2 1 1 2 1 1 2 2 2 2 1 1 1 2 2 2 1 2 2 2 2 1 2 2
## [176] 1 2 2 1 1 1 1 2 2 1 2 1 2 2 2 1 2 1 2 1 1 1 2 1 2

table(NPreg$class,clusters(m1))

##
## 1 2
## 1 75 25
## 2 47 53

## References

Leisch, F. (2004). Flexmix: A general framework for finite mixture models and latent class regression in R. Journal of Statistical Software, 1-18.