

# Model-Based Clustering and mclust

Shane T. Mueller [shanem@mtu.edu](mailto:shanem@mtu.edu)

2019-04-01

## Model-Based Clustering

The previous clustering methods used distance as a means of assessing similarity and forming clusters. This makes intuitive sense, but it would be nice to take a more statistical approach. To do so, we would like to consider a model that might have generated the data. We might assume that the data arose from some distribution (e.g., one or more normal distributions), and try to model the data based on this model. This, in general, is called *Model-based clustering*. If we make assumptions about the model that generated the data, (and our assumptions happen to be correct), we can perhaps get better classification and clustering.

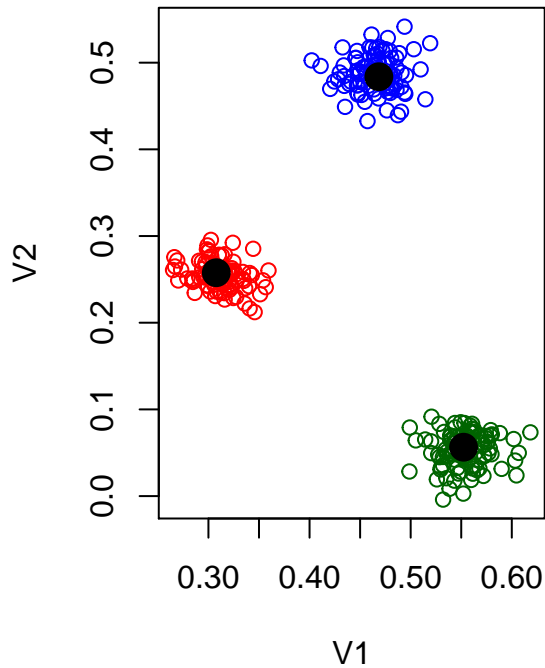
To do this, we will no longer use distance measures, but work with a set of dimensional or feature data. Let's start with a 2D example. We will assume that data are generated from a two-dimensional gaussian model. Here is an example where, within those two dimensions, our data were generated from three groups with different means, but the same standard deviation:

```
set.seed(100)

centers <- matrix(runif(6),ncol=3)
par(mfrow=c(1,2))

data1 <- matrix(c(rnorm(200,mean=centers[,1],sd=.02),
                 rnorm(200,mean=centers[,2],sd=.02),
                 rnorm(200,mean=centers[,3],sd=.02)),
               ncol=2,byrow=T)
colnames(data1) <- c("V1","V2")

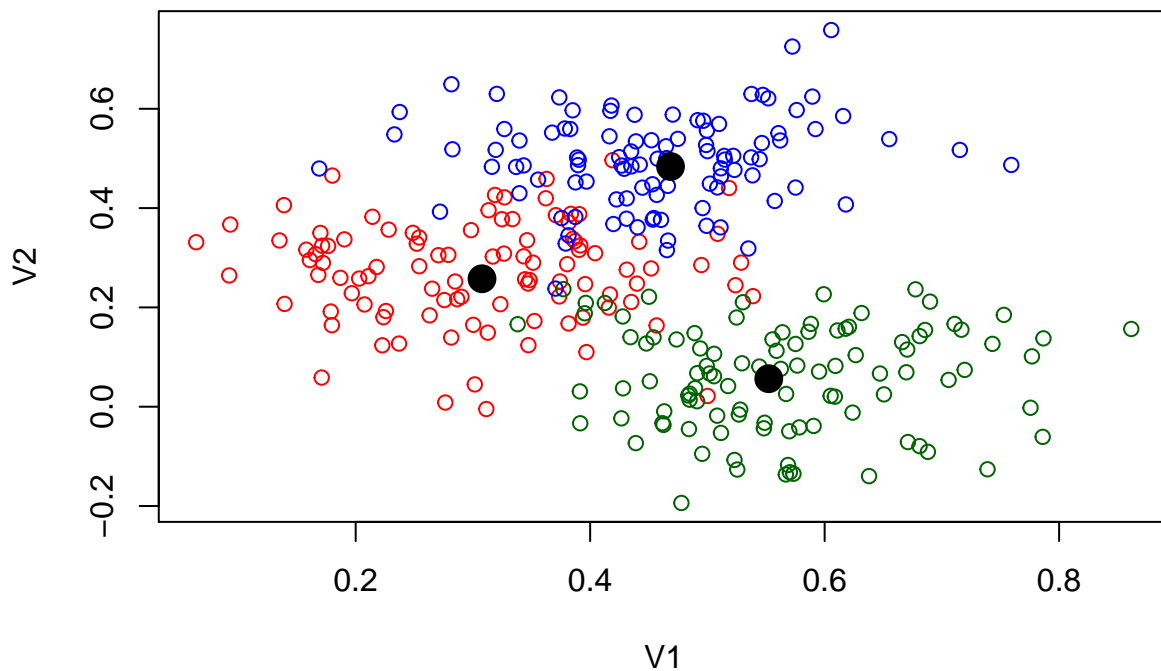
plot(data1,col=rep(c("red","darkgreen","blue"),each=100))
points(t(centers),pch=16,cex=2,col="black")
```



Here, the groups do not overlap at all, but we might imagine a setup where they do:

```
data2 <- matrix(c(rnorm(200, mean = centers[, 1], sd = 0.1), rnorm(200, mean = centers[,
  2], sd = 0.1), rnorm(200, mean = centers[, 3], sd = 0.1)), ncol = 2, byrow = T)

colnames(data2) <- c("V1", "V2")
plot(data2, col = rep(c("red", "darkgreen", "blue"), each = 100))
points(t(centers), pch = 16, cex = 2, col = "black")
```

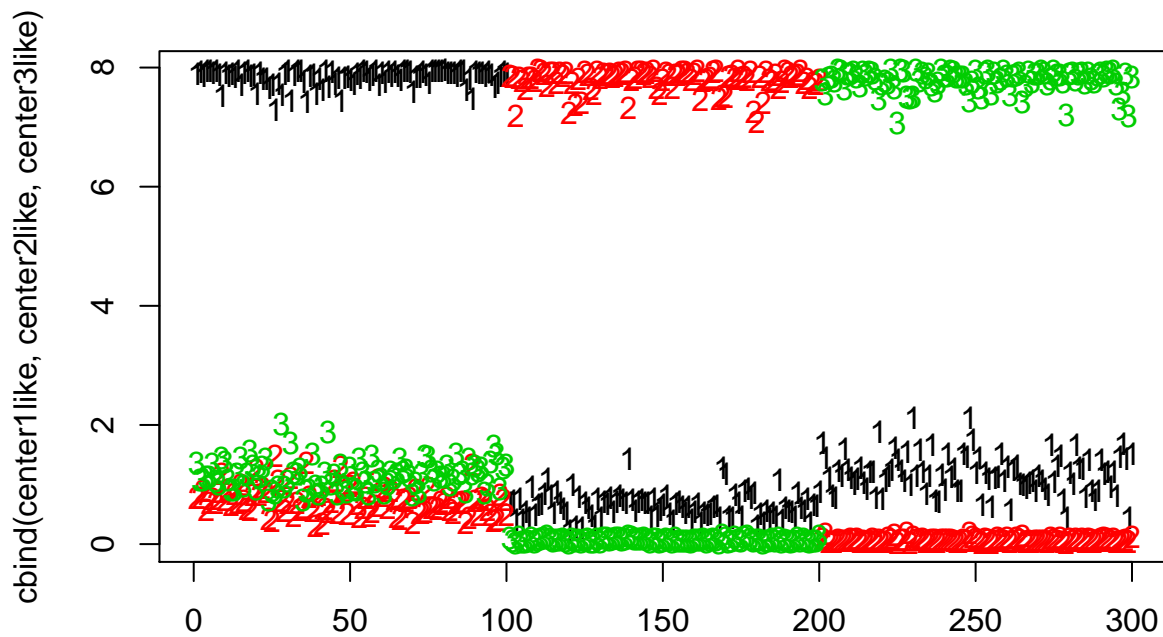


Suppose we assumed the data were from three gaussian distributions, and happened to know the centers of these three distributions, but we don't know which point came from which distribution. We could estimate the likelihood of each point having come from each of the three groups, and use this to help us determine the

group it should have belonged to. To do this, we will just use the multivariate density function from the `mvtnorm` package. We will compute the likelihood of each point for all three models—first for the low-noise data. Here we will do it for `data1`, which is the most easily discriminable:

```
library(mvtnorm)
truth <- rep(1:3, each = 100)
center1like <- dmvtnorm(data1, mean = centers[, 1], sigma = diag(rep(0.02, 2)))
center2like <- dmvtnorm(data1, mean = centers[, 2], sigma = diag(rep(0.02, 2)))
center3like <- dmvtnorm(data1, mean = centers[, 3], sigma = diag(rep(0.02, 2)))

likes <- cbind(center1like, center2like, center3like)
group <- apply(likes, 1, which.max)
matplot(cbind(center1like, center2like, center3like))
```



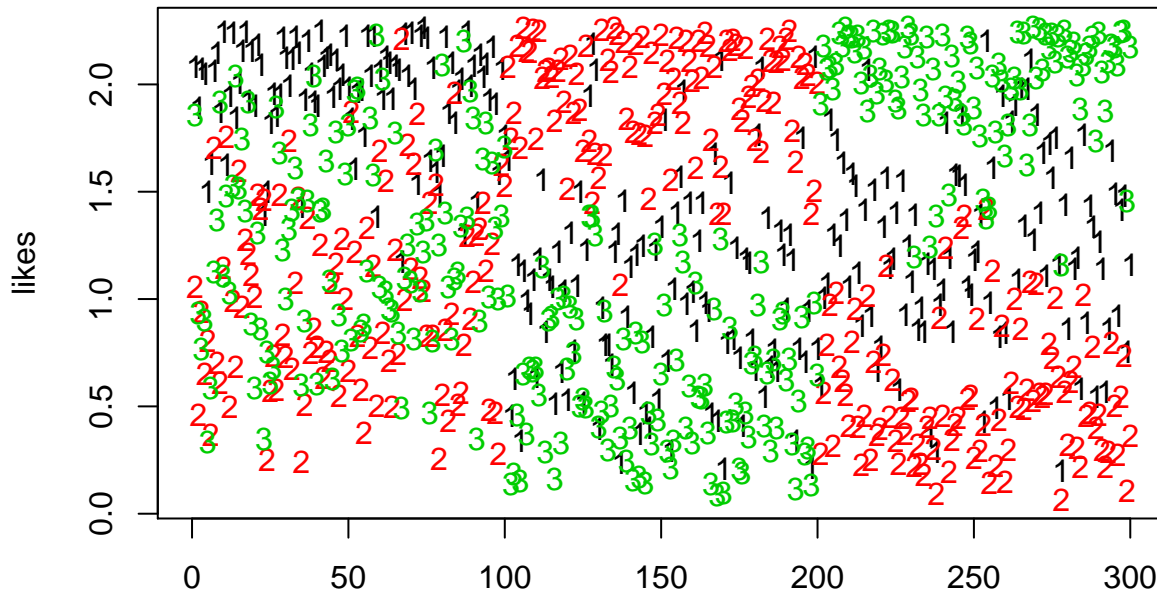
```
table(truth, group)
```

	group		
truth	1	2	3
1	100	0	0
2	0	100	0
3	0	0	100

Notice that the points were sampled in sets of 100, and the likelihood model correctly predicts every point if we categorize by the maximum likelihood group. We can look at the high-noise group `data2`, and it still looks promising, but there might be some errors.

```
center1like <- dmvtnorm(data2, mean = centers[, 1], sigma = diag(rep(0.07, 2)))
center2like <- dmvtnorm(data2, mean = centers[, 2], sigma = diag(rep(0.07, 2)))
center3like <- dmvtnorm(data2, mean = centers[, 3], sigma = diag(rep(0.07, 2)))
likes <- cbind(center1like, center2like, center3like)

matplot(likes)
```



```
group <- apply(likes, 1, which.max)
table(truth, group)
```

	group		
truth	1	2	3
1	83	6	11
2	7	93	0
3	6	0	94

Looking at the confusion table, this is not perfect, but still very good—around 90% accurate, even though we didn't know the group membership to start with. So, if we know that the data are generated from a normal distribution, and if we assume the variance is the same in both dimensions, and if we know the means of the groups, and if we are right about there being three groups, in many conditions we can do very well at recovering the group structure. The general problem is referred to as model-based clustering. Because we treat the data as being generated by a mixture of unknown sources, this is also called 'mixture modeling'.

### ##Finite mixture modeling

One common variety of this solution works similarly to k-means clustering, and is referred to as finite mixture modeling. Finite refers to assumptions about how many groups we have, and mixture modeling refers to the mixture of distributions assumptions. Unlike k-means (and more like fanny), finite mixture modeling will permit each item to have a likelihood of being a member of each group.

Inference for FMM—finding where the groups are—can happen in a number of ways. The simplest method uses a technique called Expectation-Maximization, but Bayesian approaches are also popular. Aside from the method for identifying clusters, the Bayesian approach will typically produce a distribution of solutions, whereas E-M will produce a single maximum-likelihood solution. In addition, many Bayesian approaches have explored infinite mixture models—permitting the model to have as many groups as you need, and letting the Bayesian inference determine the likely number of groups.

E-M works very similarly to the iterative process of k-means. You begin with a random assignment of items to groups. You then perform Expectation step—estimating group parameters via maximum likelihood and likelihoods for the points given the parameters. Then you perform maximization—sorting each group into the cluster that maximized likelihood. You repeat this until the process converges, and statisticians have proved that it converges under many flexible conditions.

Let's try this by hand, with data2. Do it multiple times and see the different ways it turns out.

```

group <- sample(1:3, 300, replace = T)

## Repeat the following until it converges:
par(mfrow = c(1, 2))
for (i in 1:6) {
  ## recompute centers and spread based on these groupings (maximum-likelihood
  ## estimation)
  groupmeans <- aggregate(data2, list(group), mean)[, -1]
  groupspd <- aggregate(data2, list(group), sd)

  plot(data2, col = group, main = "Current configuration")
  points(groupmeans, pch = 16, cex = 2, col = 1:3)

  ## estimate likelihood for each grouping:
  center1like <- dmvnorm(data2, mean = unlist(groupmeans[1, ]), sigma = diag(groupspd[1,
    2:3]))
  center2like <- dmvnorm(data2, mean = unlist(groupmeans[2, ]), sigma = diag(groupspd[2,
    2:3]))
  center3like <- dmvnorm(data2, mean = unlist(groupmeans[3, ]), sigma = diag(groupspd[3,
    2:3]))

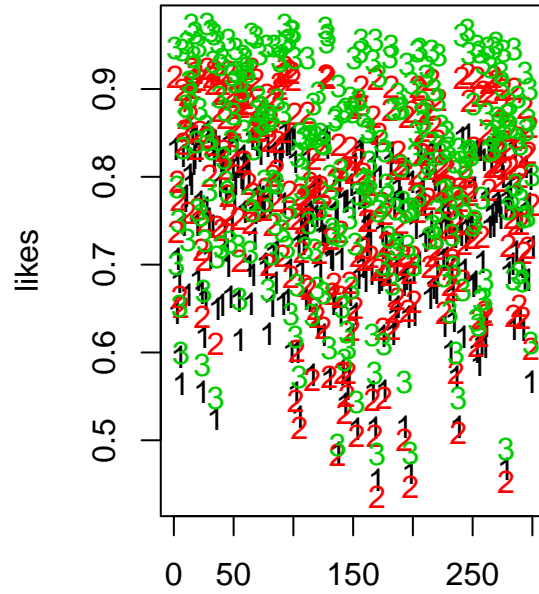
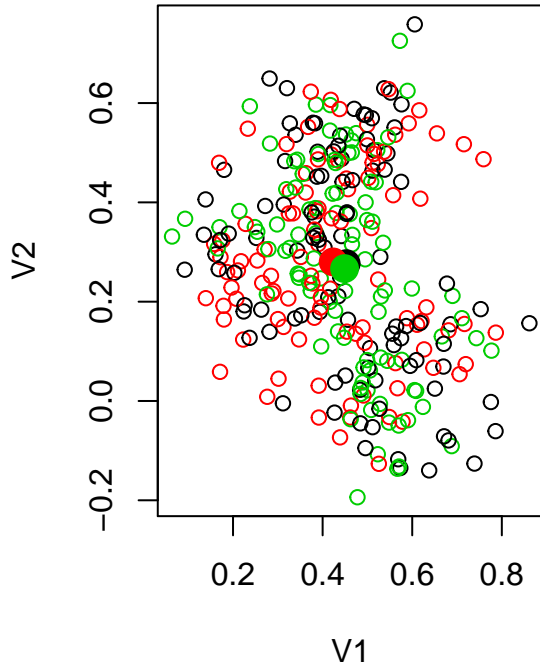
  # center2like <-
  # dmvnorm(data2, mean=groupmeans[2,], sigma=diag(groupspd[2,2:3])) center3like
  # <- dmvnorm(data2, mean=groupmeans[3,], sigma=diag(groupspd[3,2:3]))
  likes <- cbind(center1like, center2like, center3like)

  matplot(likes)
  ## redo group assignment
  group <- apply(likes, 1, which.max)

  print(groupmeans)
  print(groupspd)
  print(table(truth, group))
}

```

## Current configuration

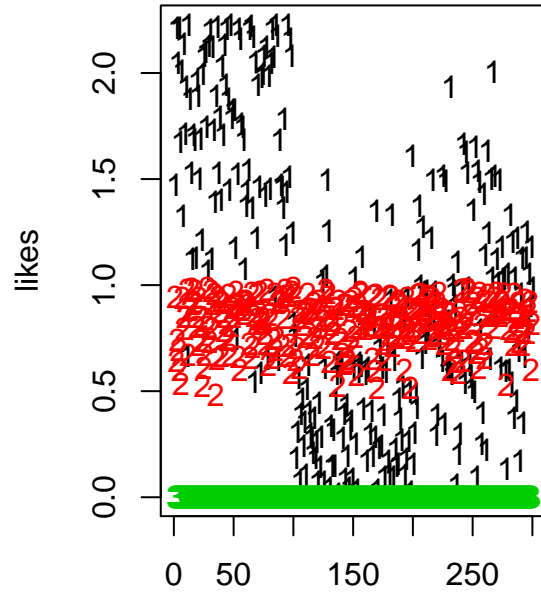
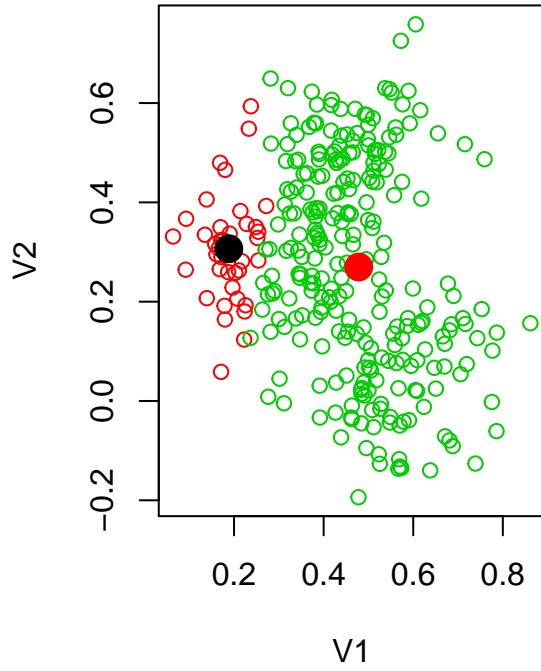


```

      V1      V2
1 0.4526602 0.2765555
2 0.4235476 0.2800561
3 0.4489721 0.2665703
  Group.1      V1      V2
1      1 0.1583307 0.2206970
2      2 0.1572064 0.1879290
3      3 0.1311509 0.2035941
  group
truth  2  3
1     34 66
2      0 100
3      4  96

```

## Current configuration

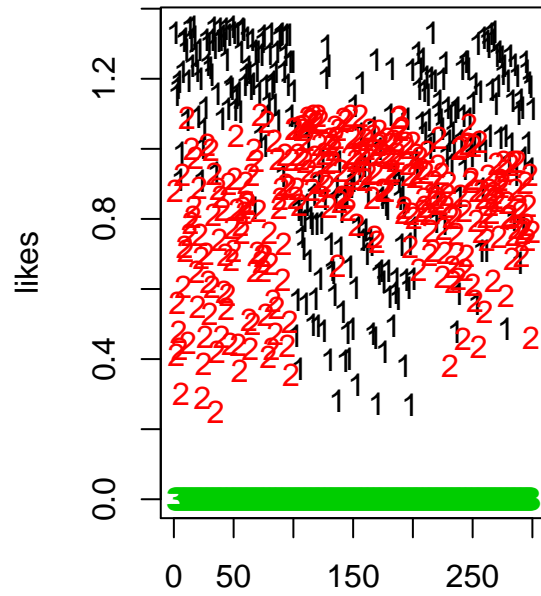
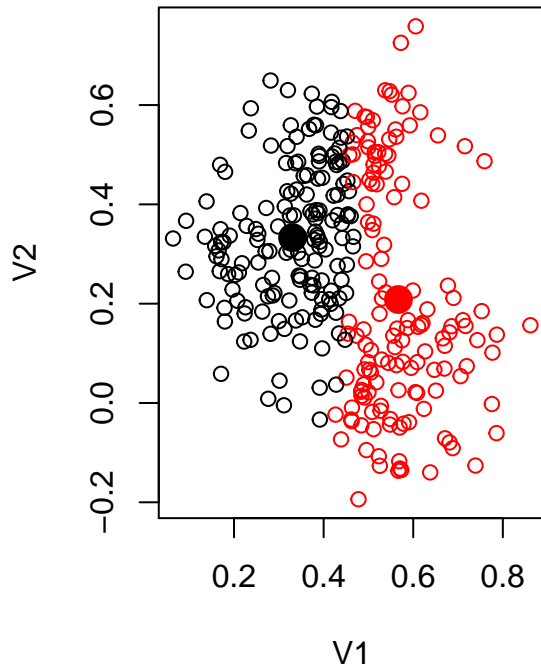


```

      V1      V2
1 0.1885021 0.3065139
2 0.4784820 0.2698116
  Group.1      V1      V2
1      2 0.04707345 0.1076460
2      3 0.12105606 0.2141792
  group
truth 1 2
      1 92 8
      2 12 88
      3 55 45

```

## Current configuration



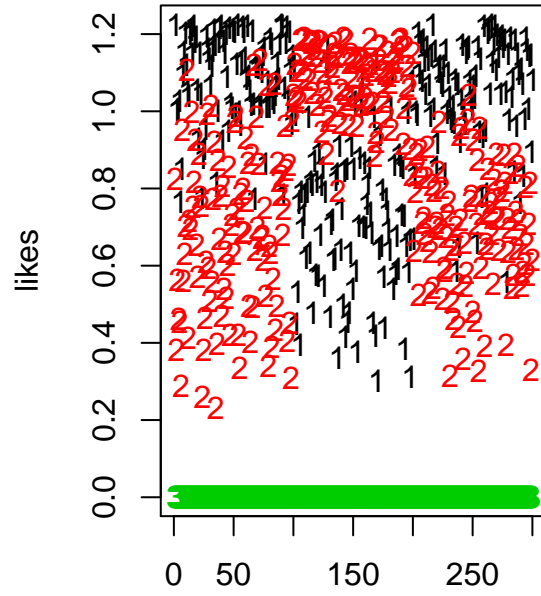
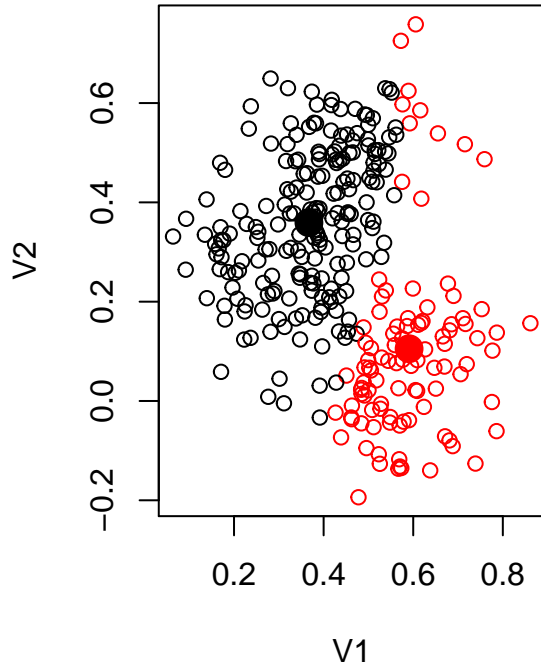
```

      V1      V2
1 0.3307062 0.3330692
2 0.5669723 0.2083699
  Group.1      V1      V2
1      1 0.09607889 0.1443800
2      2 0.08731583 0.2389049
  group
truth 1 2
      1 97 3
      2 14 86
      3 89 11

```



## Current configuration

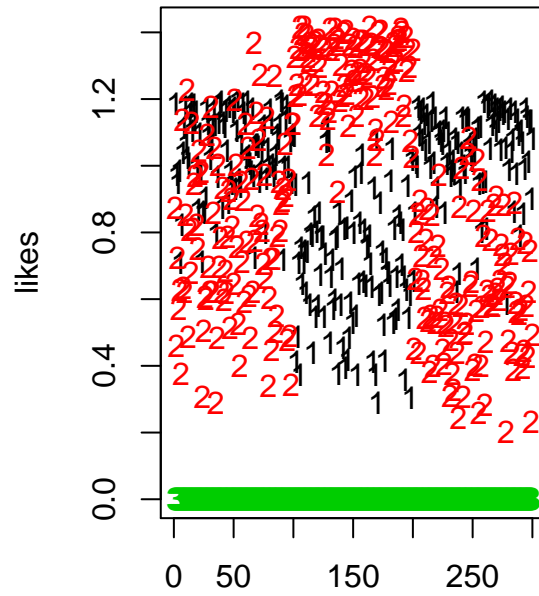
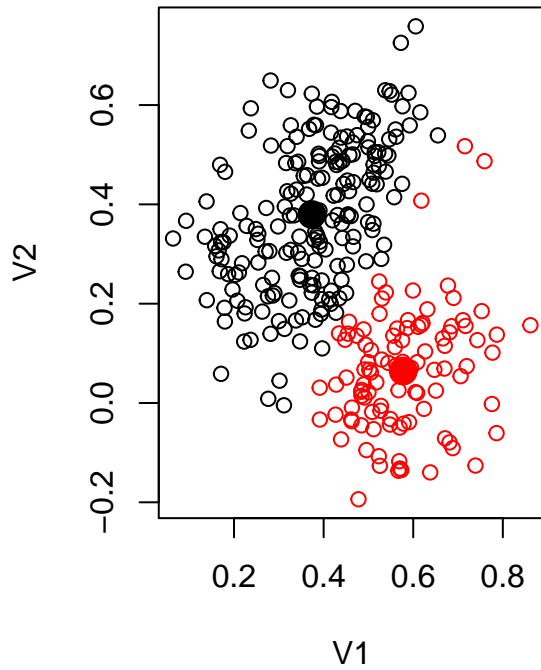


```

      V1      V2
1 0.3672293 0.3590982
2 0.5907951 0.1051853
  Group.1      V1      V2
1      1 0.11276098 0.1492019
2      2 0.09184077 0.1937960
  group
truth 1 2
      1 96 4
      2 7 93
      3 97 3

```

## Current configuration

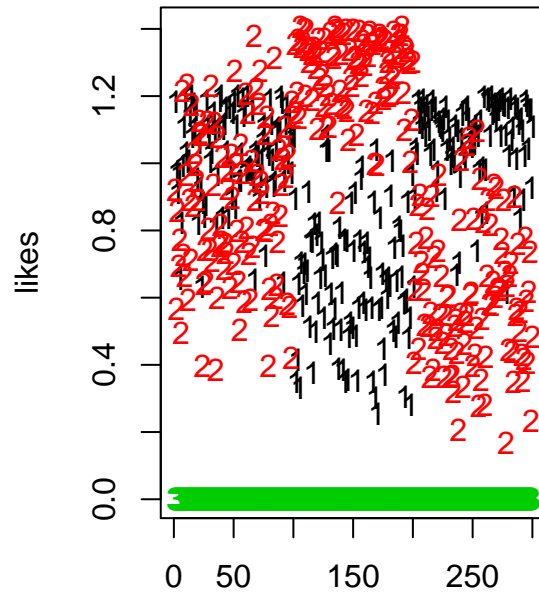
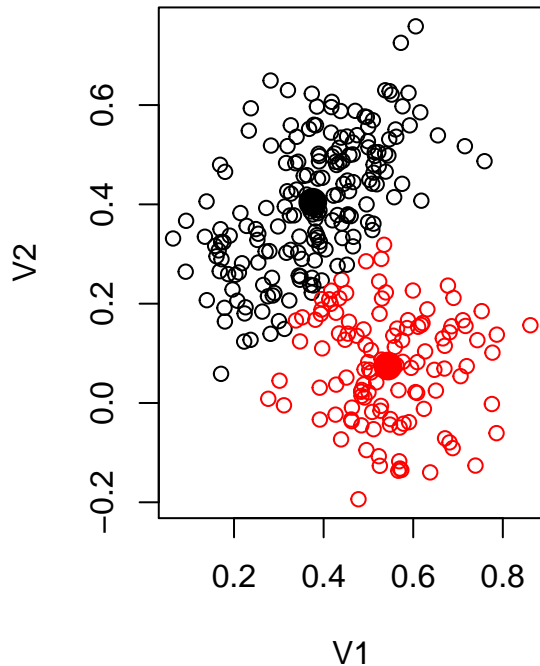


```

      V1      V2
1 0.3737522 0.37954943
2 0.5777493 0.06428281
  Group.1      V1      V2
1      1 0.1209175 0.1468932
2      2 0.1011815 0.1241568
  group
truth 1 2
      1 82 18
      2 1 99
      3 99 1

```

## Current configuration



```

      V1      V2
1 0.3757964 0.40420512
2 0.5434782 0.07434605
  Group.1      V1      V2
1      1 0.1313784 0.1334730
2      2 0.1154892 0.1106536
  group
truth 1 2
1 72 28
2 0 100
3 98 2

```

Notice by looking at the table, that the name of the groups might not be right, but it usually does a good job of categorizing by the fourth round. Sometimes, a group disappears—eaten by another group. We can look at the parameters to identify each model. Note that although the data were created with independence and equal variance, the model estimates the variance of each dimension separately.

The groupsd values should all be around 0.07, which is what generated them. Notice also that the centers are estimated very well.

groupsd

```

  Group.1      V1      V2
1      1 0.1313784 0.1334730
2      2 0.1154892 0.1106536

```

groupmeans

```

      V1      V2
1 0.3757964 0.40420512
2 0.5434782 0.07434605

```

```
centers
```

```
      [,1]      [,2]      [,3]
[1,] 0.3077661 0.55232243 0.4685493
[2,] 0.2576725 0.05638315 0.4837707
```

We have assumed a normal distribution, which is spherical, and correctly re-estimated the parameters of that distribution and a hidden piece of information—the group membership. This is how E-M is sometimes used to estimate hidden, latent, or missing information; here the missing information is the group membership variable.

We should also recognize that we have made several assumptions here that might not be true, and perhaps not made others that could have been true. For example:

- Should all groups have the same variability?
- Should a group have the same variability in each direction?
- Should there be covariance, to permit angled ellipses?

Also, we haven't yet solved the problem of figuring out how many groups there should be. Because we are fitting a likelihood model, likelihood metrics can be used. It is common to use BIC, which will balance model complexity and goodness of fit, to determine how many groups to use.

Finally, if you the solution to the model is going to be very dependent on starting conditions. There is only one 'best' solution, but we may not find it unless we run the model a few hundred times from different configurations.

For these reasons, it is usually better to use a built-in library that handles all these issues. the `mclust` and the `flexmix` libraries are both very good. We will focus on `mclust`, which is faster and offers a lot of built-in models. The `flexmix` library is more flexible, but sometimes requires you to build a model driver by hand. `mclust` focuses primarily on gaussian mixture modeling, and is very fast.

```
library(mclust)
```

```
model1 <- Mclust(data2)
summary(model1)
```

```
-----
Gaussian finite mixture model fitted by EM algorithm
-----
```

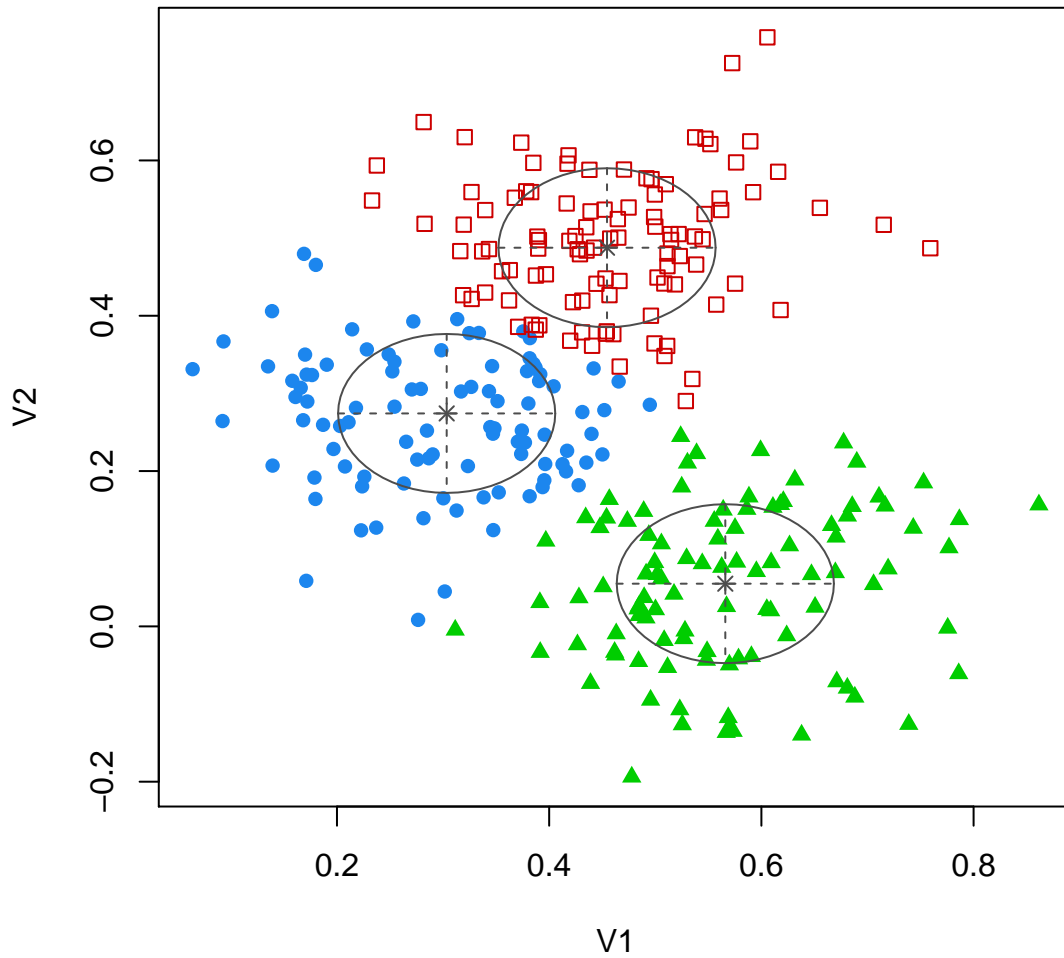
```
Mclust EII (spherical, equal volume) model with 3 components:
```

```
log-likelihood  n df      BIC      ICL
      260.7488 300  9 470.1636 399.5207
```

```
Clustering table:
```

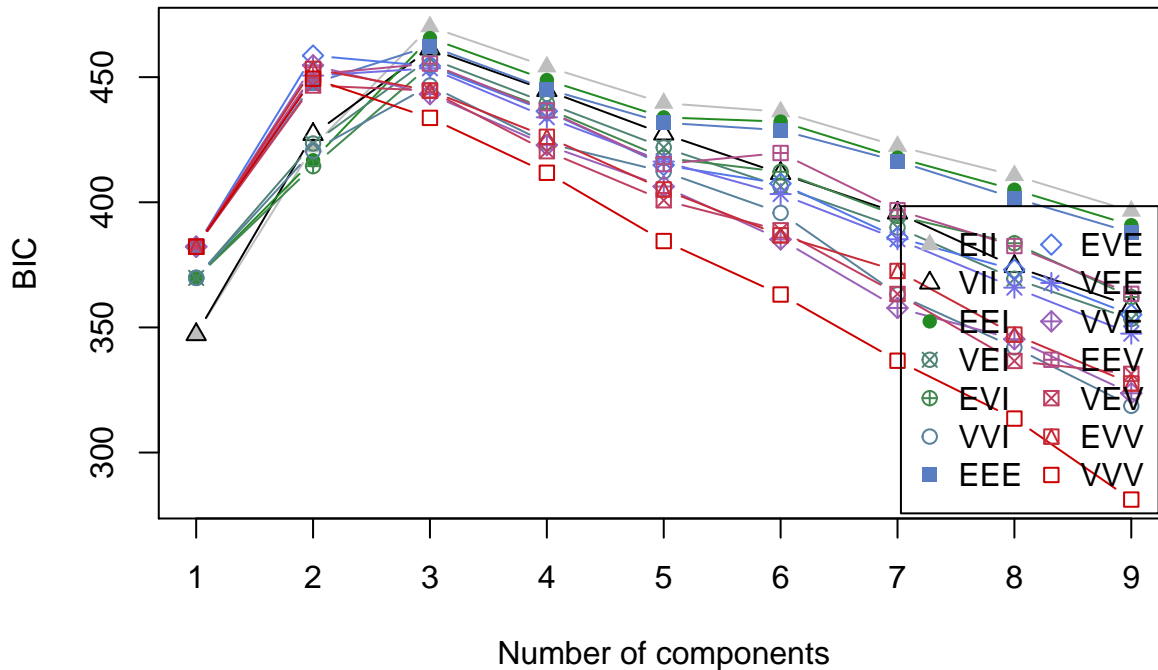
```
  1  2  3
97 104 99
```

```
plot(model1, what = c("classification"))
```



Without asking, this identified that we had three components, showed how many were in each group, and produced likelihood and BIC scores. It also states it used model 'EII'. Unpacking this, EII refers to the type of gaussian variance model. There are many different options. The model actually fit all the different variance options, and compared them via a BIC score to choose the best:

```
plot(model1, what = "BIC")
```



We can see about 14 different models here. There are also some that don't apply to multivariate mixture models. From the help file for `mclustModelNames`:

The following models are available in package `mclust`:

```

univariate mixture
"E" = equal variance (one-dimensional)
"V" = variable variance (one-dimensional)
multivariate mixture
"EII" = spherical, equal volume
"VII" = spherical, unequal volume
"EEI" = diagonal, equal volume and shape
"VEI" = diagonal, varying volume, equal shape
"EVI" = diagonal, equal volume, varying shape
"VVI" = diagonal, varying volume and shape
"EEE" = ellipsoidal, equal volume, shape, and orientation
"EVE" = ellipsoidal, equal volume and orientation (*)
"VEE" = ellipsoidal, equal shape and orientation (*)
"VVE" = ellipsoidal, equal orientation (*)
"EEV" = ellipsoidal, equal volume and equal shape
"VEV" = ellipsoidal, equal shape
"EVV" = ellipsoidal, equal volume (*)
"VVV" = ellipsoidal, varying volume, shape, and orientation
single component
"X" = univariate normal
"XII" = spherical multivariate normal
"XXI" = diagonal multivariate normal
"XXX" = ellipsoidal multivariate normal

```

In general, the lower the model, the more complex it is, and so the more parameters needed and the greater improvement needed to justify using it. The model picked EII—spherical equal volume. Spherical indicates the variance is equal in all directions, and equal volume indicates that all variances are assumed to be equal. In contrast to spherical, diagonal would permit each dimension to have a different variance, and ellipsoidal will permit covariance terms to orient the ellipse in different directions. On the other hand, we can constrain

aspects of shape and volume (size) across groups, forcing them to be the same or allowing them to differ.

We can see in the parameters how the constraints work out. The variance-sigma values are all identical. They could differ, or we could permit covariance.

```
model1$parameters
```

```
$pro
```

```
[1] 0.3284201 0.3408899 0.3306900
```

```
$mean
```

```
      [,1]      [,2]      [,3]  
V1 0.3033886 0.4545137 0.56600799  
V2 0.2742091 0.4876100 0.05498639
```

```
$variance
```

```
$variance$modelName
```

```
[1] "EII"
```

```
$variance$d
```

```
[1] 2
```

```
$variance$G
```

```
[1] 3
```

```
$variance$sigma
```

```
, , 1
```

```
      V1      V2  
V1 0.01047352 0.00000000  
V2 0.00000000 0.01047352
```

```
, , 2
```

```
      V1      V2  
V1 0.01047352 0.00000000  
V2 0.00000000 0.01047352
```

```
, , 3
```

```
      V1      V2  
V1 0.01047352 0.00000000  
V2 0.00000000 0.01047352
```

```
$variance$Sigma
```

```
      V1      V2  
V1 0.01047352 0.00000000  
V2 0.00000000 0.01047352
```

```
$variance$sigma2
```

```
[1] 0.01047352
```

```
$variance$scale
```

```
[1] 0.01047352
```

We can force use of another model—for example one with covariance.

```
model2 <- Mclust(data2, modelNames = "VVI")
summary(model2)
```

```
-----
Gaussian finite mixture model fitted by EM algorithm
-----
```

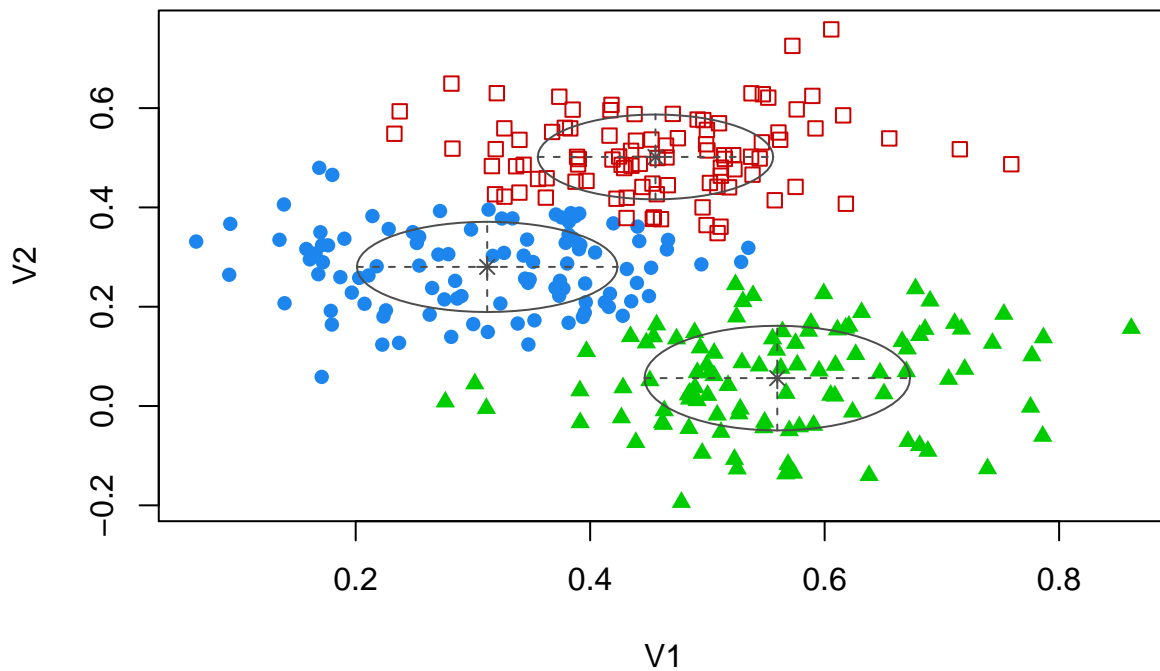
Mclust VVI (diagonal, varying volume and shape) model with 3 components:

```
log-likelihood  n df      BIC      ICL
      263.2538 300 14 446.6547 383.0861
```

Clustering table:

```
  1  2  3
104 95 101
```

```
plot(model2, what = c("classification"))
```



```
model2$parameters
```

```
$pro
[1] 0.3429819 0.3174936 0.3395244
```

```
$mean
      [,1]      [,2]      [,3]
V1 0.3121680 0.4557030 0.55960752
V2 0.2801813 0.5016499 0.05623381
```

```
$variance
$variance$modelName
[1] "VVI"
```



```
$variance$d
[1] 2
```

```
$variance$G
[1] 3
```

```
$variance$sigma
, , 1
```

```
      V1      V2
V1 0.01234934 0.00000000
V2 0.00000000 0.008257076
```

```
, , 2
```

```
      V1      V2
V1 0.01007904 0.00000000
V2 0.00000000 0.007299181
```

```
, , 3
```

```
      V1      V2
V1 0.0128181 0.00000000
V2 0.0000000 0.01110417
```

```
$variance$sigma2
[1] 0.010097991 0.008577223 0.011930396
```

```
$variance$scale
[1] 0.010097991 0.008577223 0.011930396
```

```
$variance$shape
      1      2      3
[1,] 1.2229501 1.1750939 1.0744067
[2,] 0.8176949 0.8509958 0.9307462
```

It did not make much of a difference permitting these to differ. They did change slightly, but values are all very similar across dimensions and groups. We can get the classification response out of the model directly.

```
table(truth, model1$classification)
```

```
truth  1  2  3
      1 83 11  6
      2  7  0 93
      3  7 93  0
```

```
table(truth, model2$classification)
```

```
truth  1  2  3
      1 85  7  8
      2  7  0 93
      3 12 88  0
```

## Exercise:

- Build mclust models from the letter-feature data.
- Build mclust models from the personality data.

For each model, select the type of model you feel is most reasonable, and identify how many groups of people/sounds there are. Then, look at the average sound/person from each category and see if there is something that the category captures.

```
# Letter-Feature data:
ratings <- read.table("keren.txt")
```

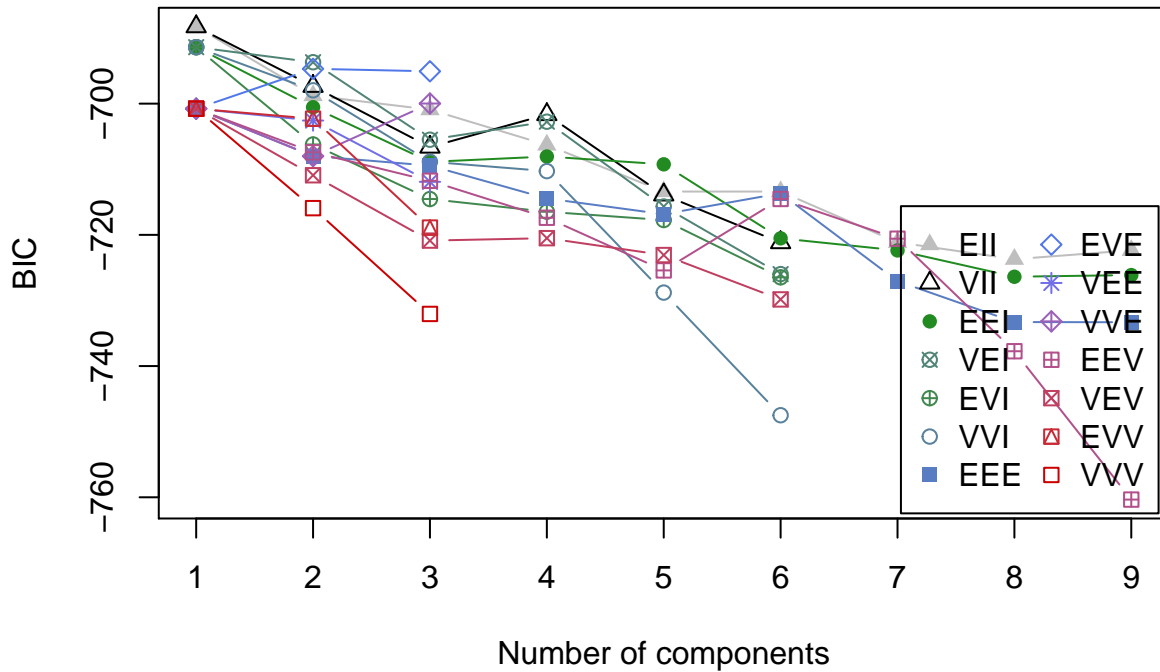
## Using mixture modeling with similarity/Distance data

Because mixture modeling works within a dimensional or feature space, the most popular versions will not work if you have a distance measure (like we used for hierarchical clustering). Without developing new algorithms, one way you can still apply tool like Mclust is to use MDS to infer a dimensional space, and then apply finite mixture modeling to that. The choice of  $k$  might matter in these cases, but you are not restricted to choosing a 2-dimensional solution. Here is an example using the letter distance data:

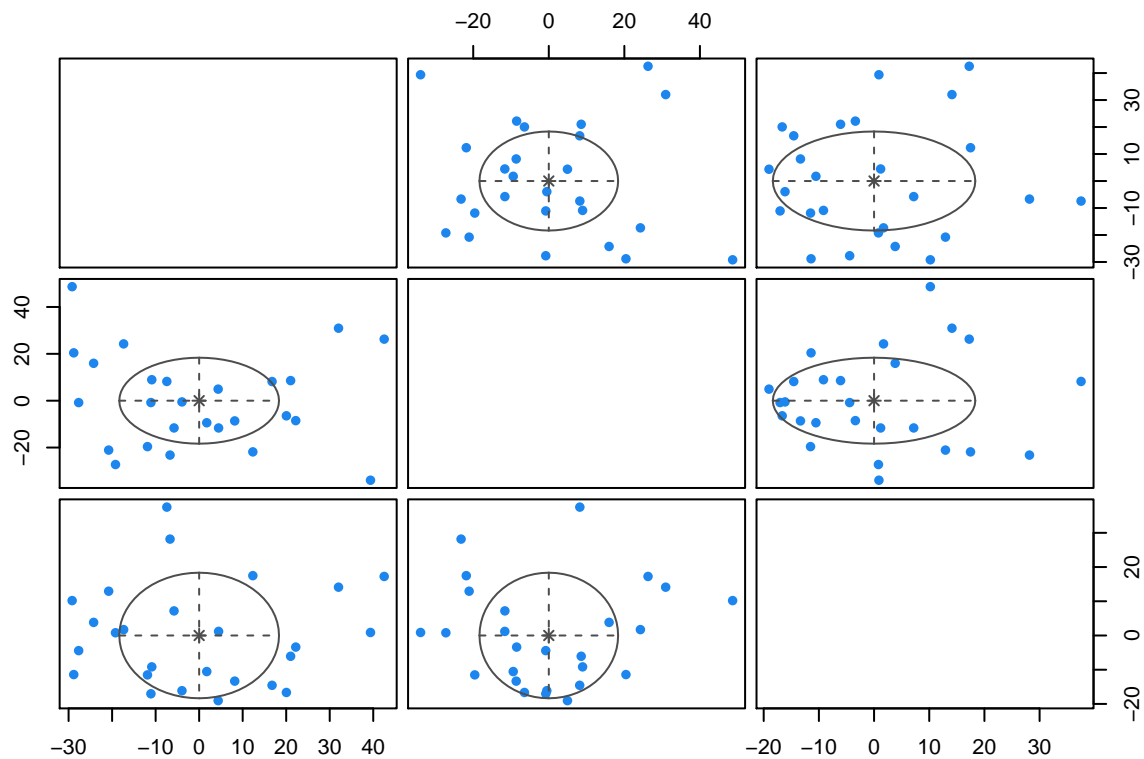
```
library(MASS)
d_dist <- read.csv("plaindist.csv")
d_dist <- d_dist[, -1]
colnames(d_dist) <- LETTERS
rownames(d_dist) <- LETTERS
x <- isoMDS(as.dist(d_dist), k = 3)
```

```
initial value 21.183195
iter 5 value 14.763980
iter 10 value 13.824625
iter 10 value 13.812779
iter 10 value 13.807954
final value 13.807954
converged
```

```
l <- Mclust(x$points)
plot(l, what = "BIC")
```



```
plot(1, what = "classification")
```

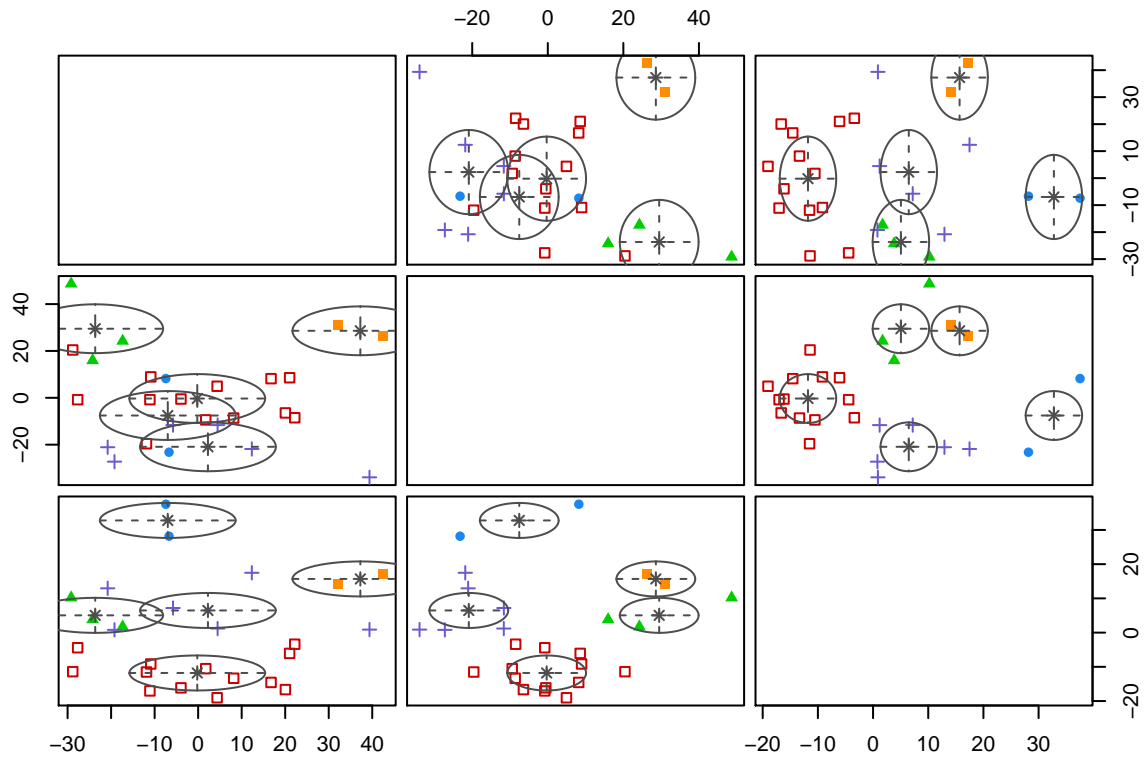


Here, the one-group solution was always the best. Notice that because we used 3 dimensions, when we plot the mclust solution, it shows us cluster breakdown for each pair.

Although the algorithm is very similar conceptually to k-means clustering, our solution prefers a single solution, because the BIC criterion is very conservative. Part of this is that the BIC incorporates number of observations into its measure. In this case though, we've already averaged across our observations, and there may have been dozens of participants and hundreds of trials. Here, BIC is probably not really appropriate, because it takes  $N$  as the number of entities we are clustering. We can always force Mclust to use a specified

number of cluster if we think that is better than choosing the best via BIC.

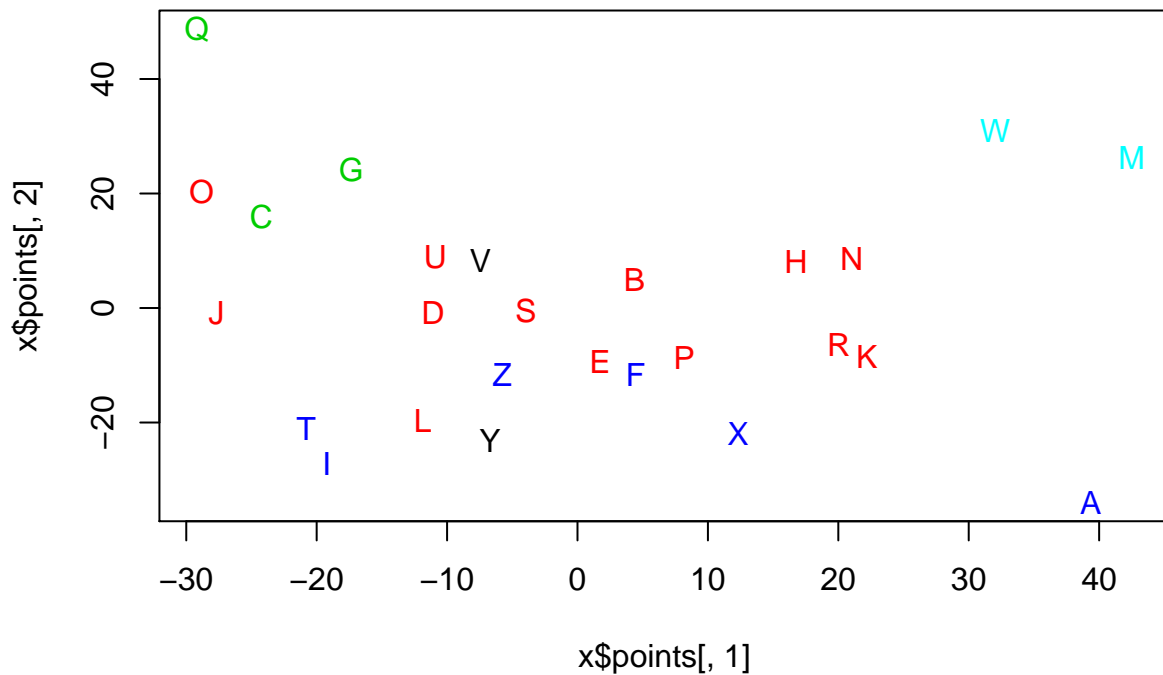
```
15 <- Mclust(x$points, G = 5)
plot(15, what = "classification")
```



Notice

that plotting this will not show the actual letters, but we can improvise:

```
plot(x$points[, 1], x$points[, 2], type = "n")
text(x$points[, 1], x$points[, 2], LETTERS, col = 15$classification)
```



## Exercise: Mixture model with personality data

1. Use mclust to find clusters of personalities.
2. Use mclust to find clusters of personality questions.

```
data <- read.csv("bigfive.csv")

dat.vals <- data[, -1] ##remove subject code
qtype <- c("E", "A", "C", "N", "O", "E", "A", "C", "N", "O", "E", "A", "C",
          "N", "O", "E", "A", "C", "N", "O", "E", "A", "C", "N", "O", "E", "A", "C",
          "N", "O", "E", "A", "C", "N", "O", "E", "A", "C", "N", "O", "O", "A", "C",
          "O")
valence <- c(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, 1, 1, 1, -1,
            1, -1, 1)
add <- c(6, 0, 0)[valence + 2]

tmp <- dat.vals
reversed <- t(t(tmp) * valence + add)
## reverse code questions:
bytype <- reversed[, order(qtype)]
key <- sort(qtype)
colnames(bytype) <- paste(key, 1:44, sep = "")

## we may need to impute missing dat. If so, a value of 3 is a quick
## reasonable guess:

bytype[is.na(bytype)] <- 3
```

## Hierarchical Model-based Clustering with mclust

mclust also provides a hierarchical model-based agglomerative clustering. This will still use a limited number of models to characterize a group, but attempt to identify a hierarchical structure. Like normal mclust, hc have varieties hcE, hcV, hcEII, hcVII, hcEEE, hcVVV, that can be specified by modelName.

```
hh <- hc(data2, modelName = "EII")
hclass(hh)[1:10, 1:10]

      2 3 4 5 6 7 8 9 10 11
[1,] 1 1 1 1 1 1 1 1 1 1
[2,] 1 1 2 2 2 2 2 2 2 2
[3,] 1 1 2 2 2 2 3 3 3 3
[4,] 1 1 2 2 2 2 2 2 2 2
[5,] 1 1 2 2 2 2 3 3 3 3
[6,] 1 1 2 2 2 2 2 2 2 2
[7,] 1 1 1 1 3 3 4 4 4 4
[ reached getOption("max.print") -- omitted 3 rows ]
```

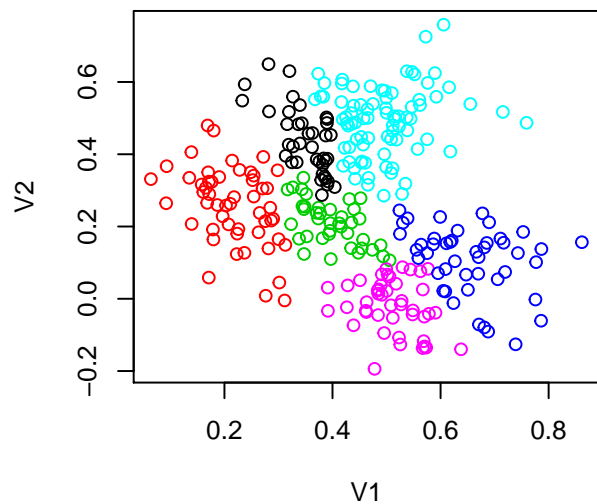
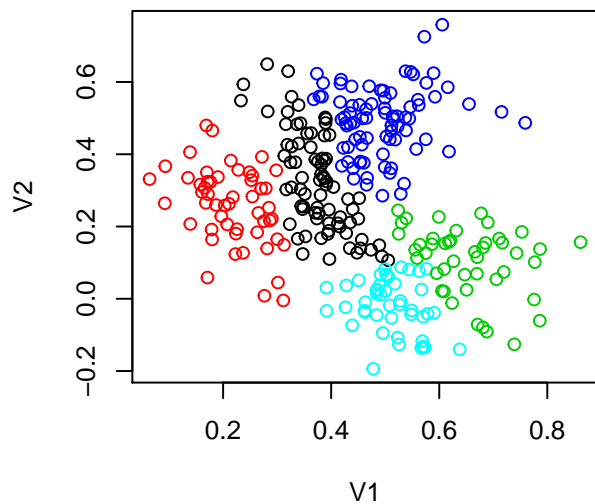
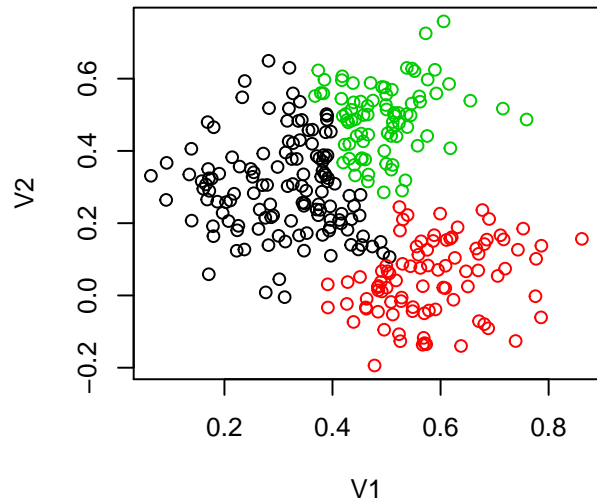
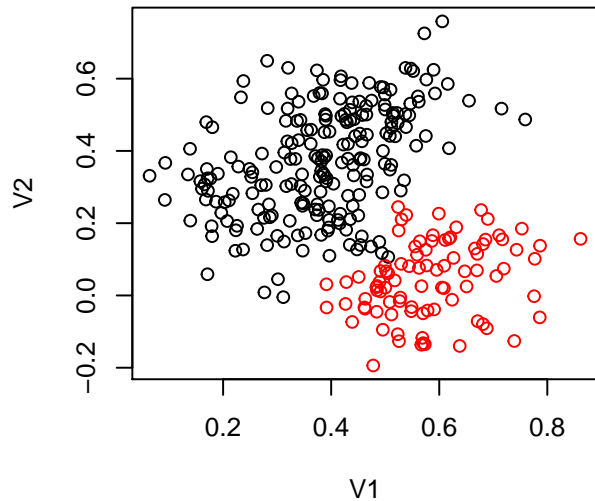
This will have N x N-1 matrix, where each column specifies a different level of hierarchical cluster. We can plot each layer consecutively, so you can see how each cluster gets divided.

```
hc <- hclass(hh)
par(mfrow = c(2, 2))
```

```

plot(data2, col = hc[, 1])
plot(data2, col = hc[, 2])
plot(data2, col = hc[, 4])
plot(data2, col = hc[, 5])

```



Notice how the clusters are always divided into two clusters. This provides an interesting hybrid approach, and also a means for determining how deeply to cluster—using the BIC criterion

```
mclustBIC(hh)
```

Bayesian Information Criterion (BIC):

	EII	VII	EVI	VEI	EVI	VVI
1	-6220.742	-6220.742	-5298.825	-5298.825	-5298.825	-5298.825
2	NA	NA	NA	NA	NA	NA

Top 3 models based on the BIC criterion:

	EVI,1	EVI,1	VEI,1
	-5298.825	-5298.825	-5298.825

## mclustBIC(hc)

Bayesian Information Criterion (BIC):

	EII	VII	E EI	VEI	EVI	VVI	EEE
1	-959320.7	-959320.7	-881575.4	-881575.4	-881575.4	-881575.4	-470545.3
2	-861126.2	-861502.0	-796147.6	-791355.9	-795965.0	-791238.7	NA
3	-811380.8	-808621.2	-753977.7	-752387.8	-749557.1	-746602.3	NA
4	-801432.5	-782705.7	-744076.8	-734538.7	NA	NA	NA
5	-780928.0	-766767.1	-729084.9	-730959.8	NA	NA	NA
	EVE	VEE	VVE	EEV	VEV	EVV	VVV
1	-470545.3	-470545.3	-470545.3	-470545.3	-470545.3	-470545.3	-470545.3
2	NA	NA	NA	NA	NA	NA	NA
3	NA	NA	NA	NA	NA	NA	NA
4	NA	NA	NA	NA	NA	NA	NA
5	NA	NA	NA	NA	NA	NA	NA

[ reachedgetOption("max.print") -- omitted 4 rows ]

Top 3 models based on the BIC criterion:

EEE,1	EEV,1	EVE,1
-470545.3	-470545.3	-470545.3

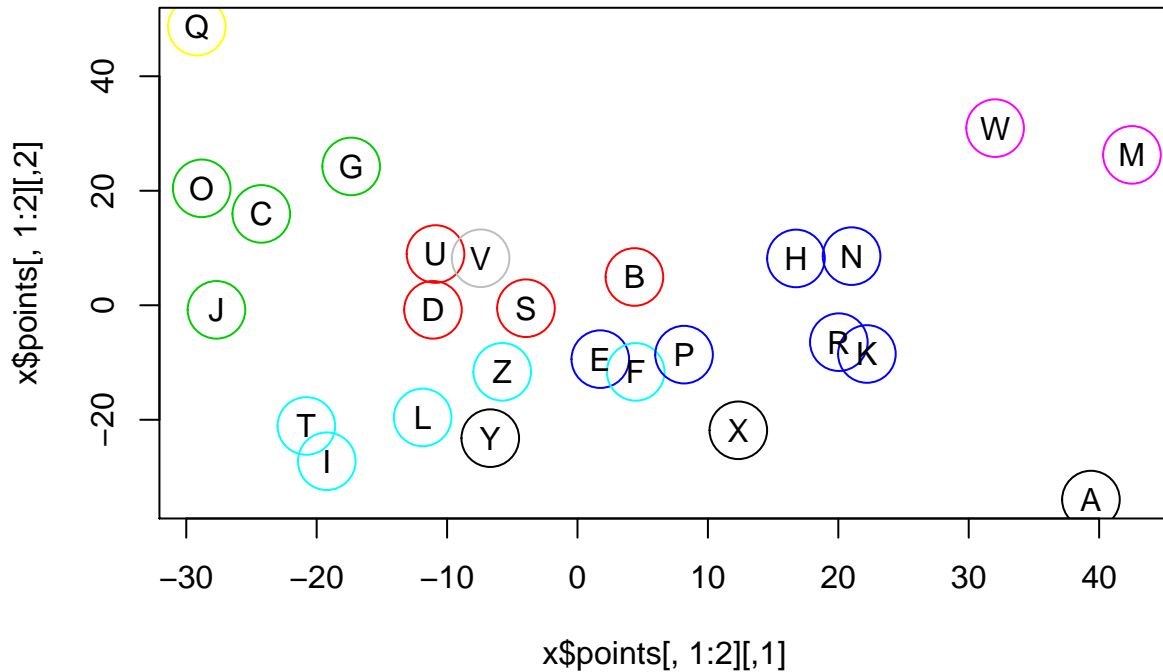
It is not clear what the first `mclustBIC(hh)` does, but the second one examines BIC values across many different values. For the complex models with elaborate covariance, we get a single group—more than that and we probably run out of parameters. This looks to be the best model overall, but it sort of forces us to use a meaning of cluster we might not be comfortable with. The simple models like EII appear to continue to improve as we add groups.

## Hierarchical clustering of letter similarity

Although `mclust` seems to fail for the letters (because of the BIC conservatism), maybe a hierarchical approach would be better. We will start with our 8-D mds solution, and plot by the first two dimensions.

```
hh <- hc(x$points, modelName = "EII")
classes <- hclass(hh)

plot(x$points[, 1:2], cex = 4, col = classes[, 8])
text(x$points[, 1], x$points[, 2], LETTERS)
```



This

seems reasonable, and shows the advantage of the conditional splitting in a hierarchical approach.

## Model-based clustering for non-gaussian distributions: The flexmix library

Mixtures of gaussians are very widely used, but we sometimes need more flexibility. Just as with regression, sometimes we don't want to use the gaussian error distribution. A good example is for binary data, although likert-scales apply as well. Or consider data where you might have a rank-order constraint. Or if you were to model a pattern of shots on a target or a dartboard, you might model a mixture of a gaussians (for the ability of a person) and a low-density uniform (for mistakes that were haphazard), so that you can remove and account for any outliers while still capturing the precision of the person.

The flexmix library provides a set of tools for building much more general mixture models. There are many built-in options, but you can also create 'drivers' for custom models and mixtures of models.

A reasonable model for binary behavior such as voting is a binomial model. Let's take another look at the Michigan Senate voting data, that we have previously looked at. Can we detect party affiliation by modeling the data as a mixture of binomials?

We could model each person with a single binomial coefficient— each person has a tendency to vote for or against a set of issues, modeled as a probability. But for this case, let's assume that a "voting bloc" has a tendency to vote similarly, and people belong to a voting block. Then, we can use mixture modeling to identify which voting block they belong to, and what the voting block looks like.

To use flexmix, we need to specify the model we want, which is the FLXMCbinary. There are dozens of models, including gaussian, and others that we will examine in the next unit.

```
library(flexmix)

data.raw <- read.csv("misenate.csv")
key <- read.csv("misenate-key.csv", header = F)

keep <- !is.na(rowMeans(data.raw[, -(1:2)]))
```



```

data <- data.raw[keep == TRUE, ]
party <- data$Party
votes <- as.matrix(data[, -(1:2)])

## model with a single group is easy;
model1 <- flexmix(votes ~ 1, k = 1, model = FLXMCmbinary())
model2 <- flexmix(votes ~ 1, k = 2, model = FLXMCmbinary())

table(party, clusters(model2))

```

```

party 1 2
  D 44 0
  R  1 58

```

```

model4 <- flexmix(votes ~ 1, k = 4, model = FLXMCmbinary())
table(party, clusters(model4))

```

```

party 1 2 3 4
  D  0 27 17 0
  R 15  0  1 43

```

```

parameters(model4)

```

	Comp.1	Comp.2	Comp.3	Comp.4
center.Q1	5.348405e-01	9.313701e-01	0.77183095	6.741074e-01
center.Q2	0.000000e+00	5.905867e-01	0.66859034	0.000000e+00
center.Q3	1.992939e-01	7.399120e-01	0.72369105	1.163977e-01
center.Q4	6.807134e-02	1.000000e+00	0.61644679	8.844886e-01
center.Q5	7.357308e-01	3.735129e-02	0.32880745	2.775276e-01
center.Q6	5.105258e-40	3.293337e-01	0.33919558	6.987509e-02
center.Q7	1.327471e-01	1.000000e+00	1.00000000	4.658339e-02
center.Q8	8.517025e-112	8.521891e-01	0.66871324	4.658339e-02
center.Q9	4.024658e-01	9.626335e-01	0.89038814	9.534770e-01
center.Q10	6.637355e-02	8.520439e-01	0.66892604	2.329170e-02
center.Q11	0.000000e+00	1.000000e+00	1.00000000	0.000000e+00
center.Q12	0.000000e+00	1.494085e-01	0.16456531	0.000000e+00
center.Q13	6.728783e-02	3.155603e-02	0.50168151	2.297086e-02
center.Q14	1.331203e-01	1.000000e+00	0.89041337	7.917867e-01
center.Q15	6.637355e-02	1.000000e+00	1.00000000	0.000000e+00
center.Q16	1.327471e-01	1.000000e+00	1.00000000	0.000000e+00
center.Q17	6.669385e-02	1.000000e+00	1.00000000	4.647099e-02
center.Q18	0.000000e+00	1.000000e+00	0.78082674	2.329170e-02

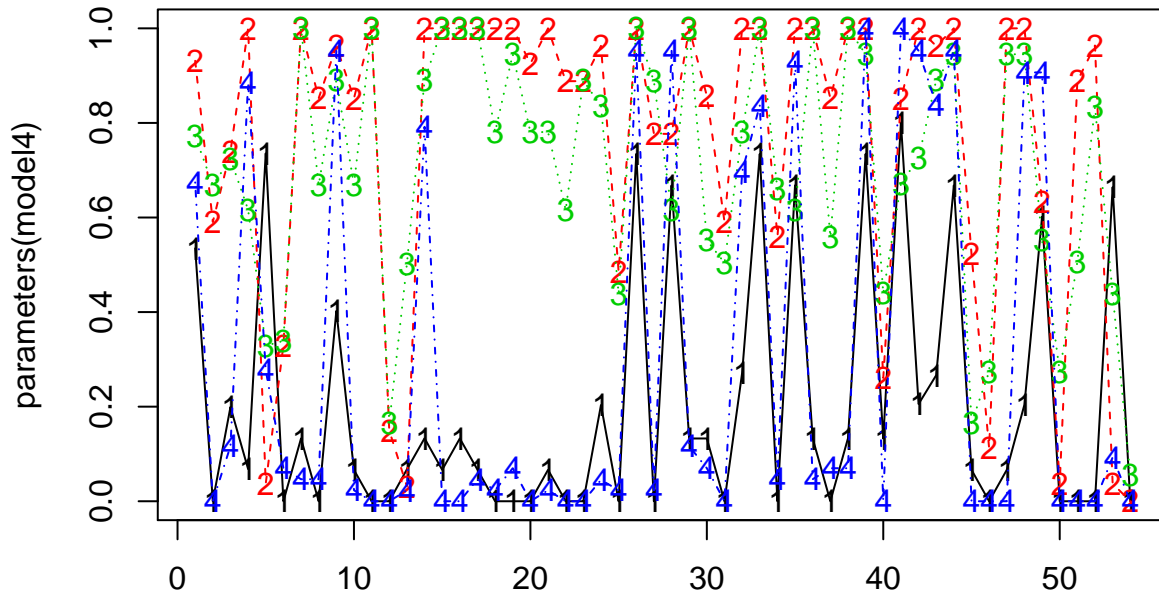
[ reached getOption("max.print") -- omitted 36 rows ]

```

matplot(parameters(model4), type = "o", main = "Voting by block on each issue")

```

## Voting by block on each issue



We find that as the number of groups grows, we get different answers different times, and we start losing groups. Also, fitting each model size separately is a bit tedious. Because we can get a solution each time, we'd like to repeat the model from a bunch of different random starting configurations, and just keep the best model we find.

The `stepflexmix` function takes care of a lot of this for us. We can specify a range of `k`, which tells us the number of groups to explore. We can also select `nrep`, which tells us how many times to fit the model at each level of `k`. 20 is fine for exploring, but you may try `nrep` values as large as 1000 if you want to be confident that are getting a consistently good solution.

```
set.seed(100)

models <- stepFlexmix(votes ~ 1, k = 1:6, model = FLXMCmvbinary(), nrep = 100,
  drop = FALSE)
```

```
1 : * * * * *
2 : * * * * *
3 : * * * * *
4 : * * * * *
5 : * * * * *
6 : * * * * *
```

```
models
```

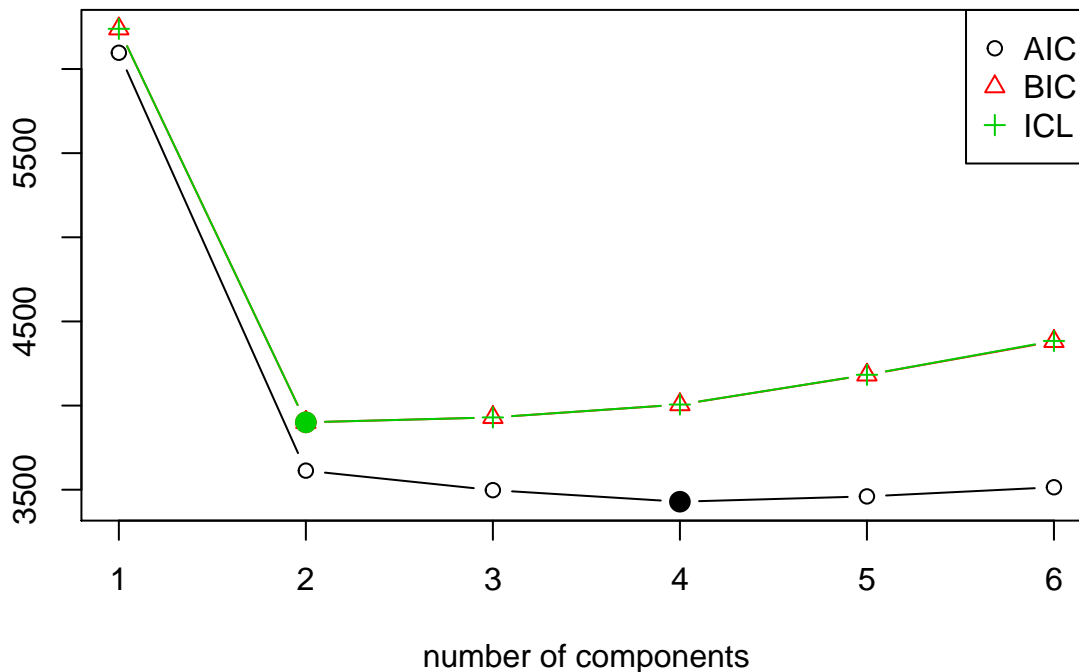
Call:

```
stepFlexmix(votes ~ 1, model = FLXMCmvbinary(), k = 1:6,
  nrep = 100, drop = FALSE)
```

iter	converged	k	k0	logLik	AIC	BIC	ICL
1	TRUE	1	1	-2994.339	6096.677	6238.952	6238.952
2	TRUE	2	2	-1697.609	3613.218	3900.404	3900.404
3	TRUE	3	3	-1584.674	3497.348	3929.443	3929.559
4	TRUE	4	4	-1495.365	3428.730	4005.735	4006.206

```
5 38 TRUE 5 5 -1456.109 3460.218 4182.133 4182.933
6 18 TRUE 6 6 -1428.280 3514.560 4381.385 4384.173
```

```
plot(models)
```



Unlike the mclust library, flexmix plots BIC so that lower is better. Here, the BIC tends to favor two groups, but the AIC favors 5 or more groups. Notice that drop, which defaults to TRUE, can prevent groups from being completely dropped. If we look at the output, we see that even for this case, the ultimate value of  $k$  that was achieved is often not the same as the starting value ( $k_0$ ), and we never ended up with more than 6 groups. Notice also that the best solutions have slightly different BIC values—even within 6-group solutions. This means that we would probably want to increase  $nrep$  to make sure we consistently find the best group.

A number of functions let us extract the model and extract information from them model. We can extract the best model using `getModel`; if we specify the statistic (“BIC” or “AIC”) it will pick out that model; if we specify a number, it will pick out the model with that number of components.

```
bestmodel <- getModel(models, "BIC")
bestmodel
```

```
Call:
stepFlexmix(votes ~ 1, model = FLXMCmvbinary(), k = 2, nrep = 20,
  drop = FALSE)
```

```
Cluster sizes:
 1  2
45 58
```

```
convergence after 5 iterations
```

```
summary(bestmodel)
```

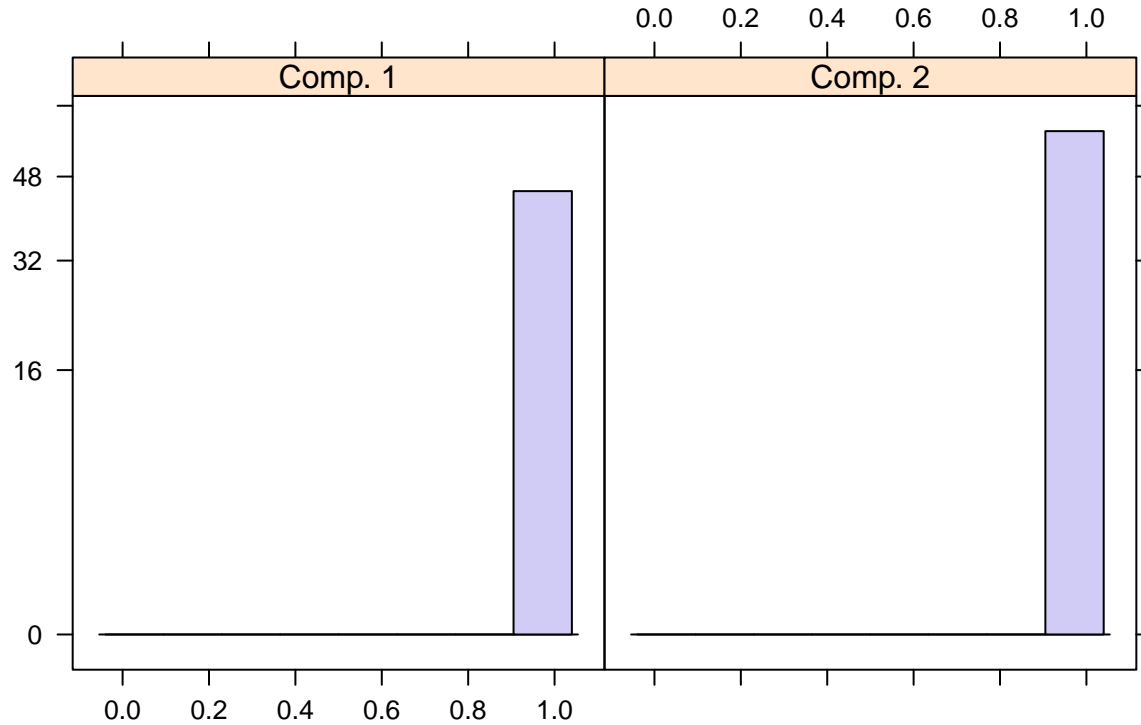
```
Call:
stepFlexmix(votes ~ 1, model = FLXMCmvbinary(), k = 2, nrep = 20,
  drop = FALSE)
```

	prior	size	post>0	ratio
Comp.1	0.437	45	45	1
Comp.2	0.563	58	58	1

'log Lik.' -1697.609 (df=109)  
AIC: 3613.218 BIC: 3900.404

```
plot(bestmodel)
```

### Rootogram of posterior probabilities > 1e-04



```
parameters(bestmodel)
```

	Comp.1	Comp.2
center.Q1	0.86666667	6.379310e-01
center.Q2	0.62222222	2.413488e-88
center.Q3	0.73333333	1.379310e-01
center.Q4	0.84444444	6.724138e-01
center.Q5	0.15555556	3.965517e-01
center.Q6	0.33333333	5.172414e-02
center.Q7	1.00000000	6.896552e-02
center.Q8	0.77777778	3.448276e-02
center.Q9	0.93333333	8.103448e-01
center.Q10	0.77777778	3.448276e-02
center.Q11	1.00000000	2.413488e-88
center.Q12	0.15555556	2.035220e-149
center.Q13	0.22222222	3.448276e-02
center.Q14	0.95555556	6.206897e-01
center.Q15	1.00000000	1.724138e-02
center.Q16	1.00000000	3.448276e-02
center.Q17	1.00000000	5.172414e-02



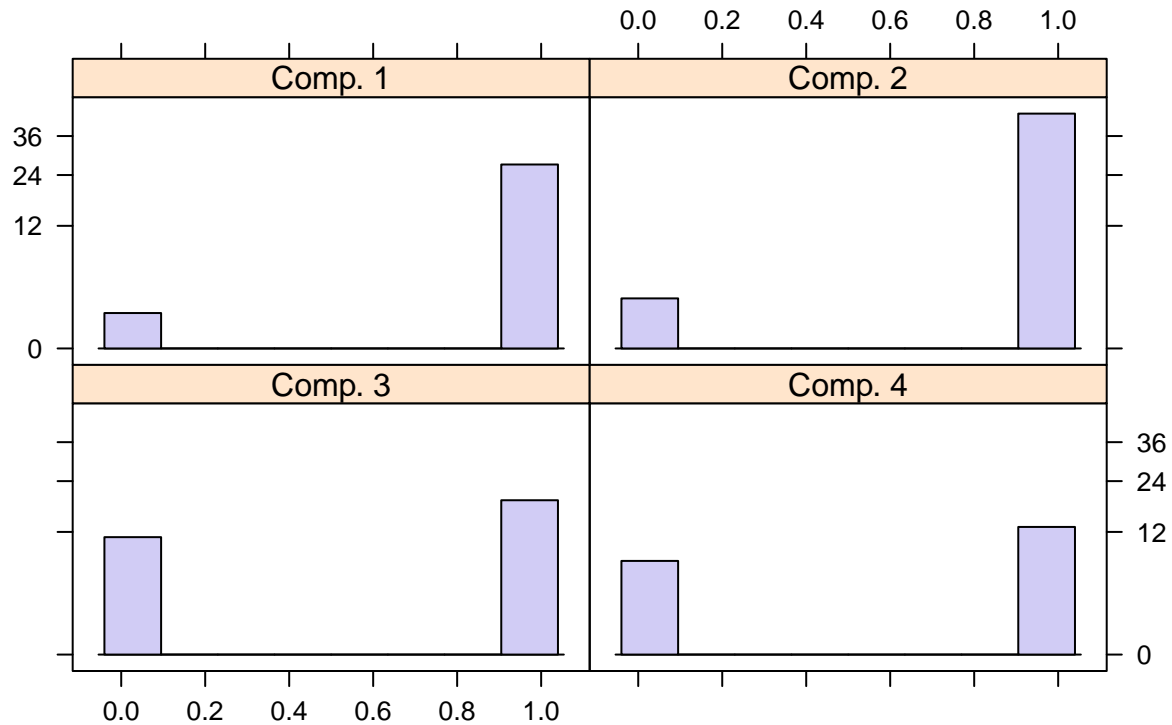
```
Call:
stepFlexmix(votes ~ 1, model = FLXMCmvbinary(), k = 4, nrep = 100,
  drop = FALSE)
```

	prior	size	post>0	ratio
Comp.1	0.260	27	28	0.964
Comp.2	0.427	44	46	0.957
Comp.3	0.186	19	30	0.633
Comp.4	0.126	13	20	0.650

```
'log Lik.' -1495.365 (df=219)
AIC: 3428.73 BIC: 4005.735
```

```
plot(bestmodel)
```

**Rootogram of posterior probabilities > 1e-04**



```
parameters(bestmodel)
```

	Comp.1	Comp.2	Comp.3	Comp.4
center.Q1	0.92543658	6.590821e-01	7.913708e-01	5.385580e-01
center.Q2	0.74512302	0.000000e+00	4.177597e-01	0.000000e+00
center.Q3	0.88836108	1.363851e-01	4.781118e-01	1.537639e-01
center.Q4	0.88818374	8.633822e-01	7.391798e-01	7.821707e-02
center.Q5	0.18636030	2.949596e-01	1.043286e-01	7.706432e-01
center.Q6	0.52172066	6.819295e-02	5.228919e-02	1.476413e-175
center.Q7	1.00000000	6.819295e-02	1.000000e+00	0.000000e+00
center.Q8	1.00000000	4.546196e-02	4.261963e-01	0.000000e+00
center.Q9	0.92545570	9.545387e-01	8.956720e-01	3.849528e-01
center.Q10	0.99991387	4.546196e-02	4.263168e-01	0.000000e+00
center.Q11	1.00000000	0.000000e+00	9.478360e-01	0.000000e+00

```
center.Q12 0.22363065 0.000000e+00 5.216714e-02 0.000000e+00
center.Q13 0.25779840 2.251728e-02 1.608398e-01 7.760340e-02
center.Q14 1.00000000 7.728181e-01 8.956720e-01 7.699984e-02
center.Q15 1.00000000 0.000000e+00 1.000000e+00 0.000000e+00
center.Q16 1.00000000 0.000000e+00 1.000000e+00 7.688061e-02
center.Q17 1.00000000 4.546196e-02 1.000000e+00 1.727036e-103
center.Q18 1.00000000 2.273098e-02 7.391801e-01 0.000000e+00
[ reached getOption("max.print") -- omitted 36 rows ]
```

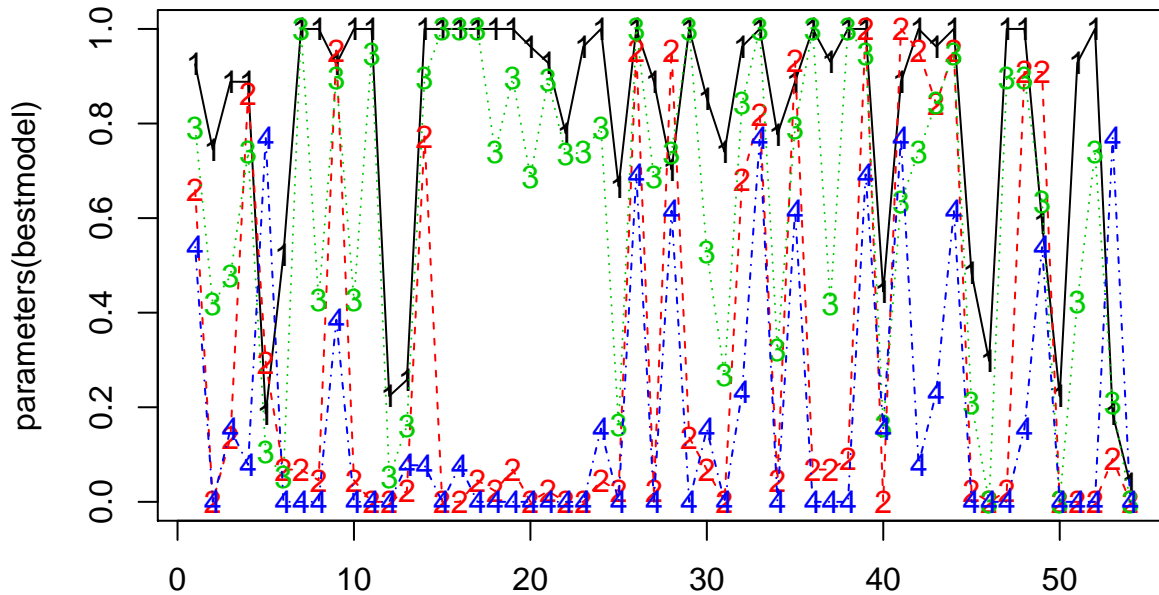
```
clusters(bestmodel)
```

```
[1] 4 4 4 4 4 4 4 2 4 2 4 4 2 2 4 2 2 2 2 4 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 3 3 3 3 3 3 3 3 3 3 3 3
[71] 3 3 3 3 3
[ reached getOption("max.print") -- omitted 28 entries ]
```

```
table(party, clusters(bestmodel))
```

```
party 1 2 3 4
D 27 0 17 0
R 0 44 2 13
```

```
matplot(parameters(bestmodel), type = "o")
```



If forced to classify the politicians into 4 parties, we get one group of democrats, two republicans, and one mixed outlier group. We could maybe sort the issues (since they are arbitrary) to get a better picture. We can make a table with declared party membership to determine who belongs in which group. Then, we can arrange issues using another clustering solution.

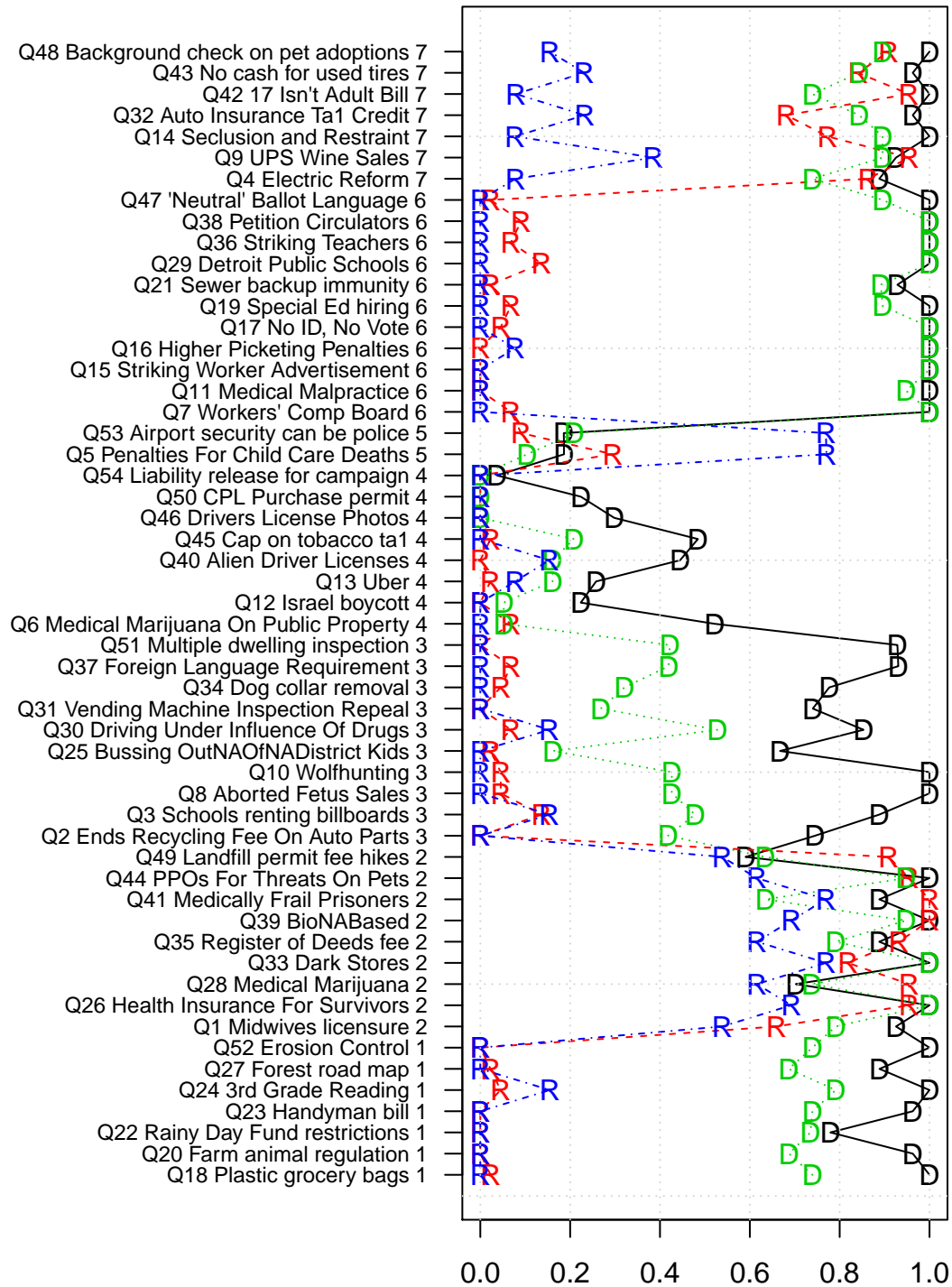
```
set.seed(101)
tab <- table(party, clusters(bestmodel))
gparty <- c("D", "R")[apply(tab, 2, which.max)]
p <- parameters(bestmodel)

grp <- kmeans(p, centers = 7)
ord <- order(grp$cluster)
par(mar = c(3, 25, 2, 1), las = 1)
```

```

matplot(p[ord, ], 1:54, pch = gparty, type = "o", yaxt = "n", ylab = "", xlab = "Agreement with rater on",
grid()
axis(2, 1:54, paste(paste(key$V1, key$V2)[ord], sort(grp$cluster)), cex.axis = 0.75)

```



This clustered questions into 7 groups, then just displayed them in order of the (arbitrary) group number. Looking at the bills, it is difficult to determine why they clustered together, or why the 4-party solution divided parties on specific issues, but someone who follows Michigan politics may learn important things from this solution.