Scientific Computing for Biologists
Hands-On Exercises, Lecture 10

Paul M. Magwene

08 November 2011

K-means Clustering in R

The `kmeans()` function calcualtes standard k-means clusters in R. The input is a data matrix (perhaps transformed before hand) and k, the number of clusters. Alternatively you can specify starting cluster centers. You can also run the algorithm multiple times with different random starting positions by using the `nstart` argument.

```
# generate data set w/two groups (one of size 50, the other of size 75)
# note the different means and std dev between the two groups
> test.data <- rbind(matrix(rnorm(100, mean=0, sd=0.2),ncol=2),
                     matrix(rnorm(150,mean=1,sd=0.5),ncol=2))
> colnames(test.data) <- c("x", "y")
> plot(test.data)
> cl <- kmeans(test.data, 2)
> names(cl)
[1] "cluster"  "centers"  "withinss" "size"
> cl$cluster #  which cluster each object is assigned to
... output deleted ...
> plot(test.data, col = cl$cluster)
> cl$centers  # compare to the "true" means for the groups
           x         y
1 0.009479636 0.1182016
2 1.109641398 1.0427396
> points(cl$centers, col = 1:2, pch = 8, cex=2)

> # what if we pick the wrong number of clusters?
> cl <- kmeans(test.data, 5)
> plot(test.data, col = cl$cluster)
> points(cl$centers, col = 1:5, pch = 8, cex=2)

> # as above but using nstart argument
> cl <- kmeans(test.data, 5, nstart=25)
> plot(test.data, col = cl$cluster)
> points(cl$centers, col = 1:5, pch = 8, cex=2)
```

Applying K-means to the iris data set

Now that we've seen how to apply k-mean clustering to a synthetic data set, let's go ahead and apply it to our old friend the iris data set. Note that this is a four dimensional data set so we'll need to pick a projection in which to depict the cluster. The space of the first two principal components is a natural choice (but note that the fact that we're using the PCA space doesn't impact the k-means clustering in this context).

```
# drop the fifth column (species names)
# we'll assume we know how many groups there are
> k.iris <- kmeans(as.matrix(iris[,-5]), 3)
> iris.pca <- prcomp(iris[,-5])
# the following plot colors the specimens by the group
# they were assigned to by the k-means clustering
> plot(iris.pca$x,col=k.iris$cluster)

# this plot colors speciemsn by k-means grouping
# and chooses plot symbol by real species grouping.
# This can help us quickly pick out the misclasssified
# specimens
> plot(iris.pca$x, col=k.iris$cluster, pch=c(1,2,16)[iris[,5]])
```

Gaussian Mixture Models in R

There are multiple packages for fitting mixture models in R. We'll look at two – `mixtools` and `MCLUST`.

Installing mixtools

The package `mixtools` can be installed via the GUI or the `install.packages` command. A mixtools vignette can be downloaded from the CRAN website.

Using mixtools

We'll look at how to use mixtools using a data set on eruption times for the Old Faithful geyser in Yellowstone National Park (`?faithful` for details). We'll fit a univariate Gaussian mixture model to the time between eruptions data (`faithful$waiting`).

```
# allows us to refer to the variables within waiting time
# without using the standard list "$" syntax
> attach(faithful)

# create a nice histogram
> hist(waiting, main = "Time between Old Faithful eruptions",
xlab = "Minutes", ylab = "", cex.main = 1.5, cex.lab = 1.5, cex.axis = 1.4)

> library(mixtools)
> ?normalmixEM  # read the docs!
> wait.mix <- normalmixEM(waiting)

> names(wait.mix)
[1] "x"          "lambda"     "mu"         "sigma"      "loglik"     "posterior"
[7] "all.loglik" "restarts"   "ft"

# lambda is what we called "pi" in the lecture notes
> wait.mix[c("lambda","mu","sigma")]

> class(wait.mix)
[1] "mixEM"
> ?plot.mixEM  # read about the plotting options for the mixEM object
> plot(wait.mix, density=TRUE)
> plot(wait.mix, loglik=FALSE, density = TRUE, cex.axis = 1.4, cex.lab = 1.4,
 cex.main = 1.8, main2 = "Time between Old Faithful eruptions", xlab2 = "Minutes")
```

Installing MCLUST

The package `MCLUST` is one of another package that provides maximum likelihood based estimation of mixture models. You will need to install the package (and it's dependencies) from the R GUI or using the `install.packages` command.

Using MCLUST

We're going to use two data set to illustrate some of `MCLUST`'s capabilities – the iris data set we've worked with before, and the bivariate version of the Old Faithful data set. We'll start off with the old faithful data set.

```
> plot(faithful$eruptions, faithful$waiting)
```

From visual inspection of the bivariate scatter plot, it looks like there two clusters. Let's apply the `Mclust` function and see what it suggests:

```
> library(mclust)
> fclust <- Mclust(faithful)
> fclust

 best model: elliposidal, equal variance with 3 components
```

Now that we've running the mixture model, let's look at the results graphically. The following call to `plot` will produce a series of plots.

```
> plot(fclust, data=faithful)
```

The first plot gives is a diagnostic plot that shows the likelihood of the model as a function of the number of groups (see BIC below). In general, when considering many possible models you want to pick the simplest model that has a highest likelihood. The second graphically represents the classification. The third plots highlights those objects for which the cluster assignment is most uncertain. The fourth plot gives a graphical representation of the Gaussian densities.

The `Mclust` function used a likelihood criterion called the "Bayesian Information Criterion" (BIC) to estimate the number of components (clusters) in the mixture model. By this criterion it suggested 3 components. BIC, like other information criteria (the Akaike Information Criterion is another popular one), is designed to help choose among parametric models with different numbers of parameters. It tries to choose the simplest model that provides a good fit to the data.

```
> names(fclust)
 [1] "modelName"      "n"              "d"              "G"              "BIC"
 [6] "bic"            "loglik"         "parameters"     "z"              "classification"
[11] "uncertainty"
> ?mclust  # check out the docs to read about all the returned parameters
```

Of course you don't have to accept the number of clusters that the `Mclust` function estimated. Here's how you'd calculate the mixture model with a user determined number of clusters:

```
> fclust2 <- Mclust(faithful, G=2)
> plot(fclust2, data=faithful)
```

Now let's generate some diagnostic plots for the mixture model:

```
> plot(fclust, data=faithful)
```

If you wanted to generate some of those plots individually you can do the following:

```
# generate a plot showing the classifications predicted by mixture model
> mclust2Dplot(data = faithful, what = "classification", identify = TRUE,
    parameters = fclust$parameters, z = fclust$z)
```

See the docs for the `mclust2Dplot` function for other options.

Mixture Models for the Iris data set

The `MCLUST` package includes the function `clPairs`, a very nice extension of the `pairs` function, for creating scatter plot matrices with group information. The following code illustrates this:

```
> names(iris) # remind yourself of the variables in the iris data set
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"

# 5th variable is the Species classification
> clPairs(data=iris[,-5], classification=iris[,5])
```

Let's see what `Mclust` makes of the iris data set:

```
> iclust <- Mclust(iris[,-5])
> iclust

 best model: ellipsoidal, equal shape with 2 components
> plot(iclust, data=iris[,-5])
```

Blind to the actual group structure the BIC suggests just two components, whereas we know there are three groups (though *I. versicolor* is thought to be an allopolyploid hybrid; see Kim et al. (2007) Ann Bot, 100: 219-224.). Examine the first graph produced by the `plot` call above to see how the 2 and 3 component models compare with respect to the BIC.

Now let's see how the mixture model does when we give it the true number of clusters:

```
> iclust3 <- Mclust(iris[,-5], G=3)
> plot(iclust3,data=iris[,-5])
```

To calculate the classification error rate we can compare the estimated clustering to the "true" (known) classification with the `classError` function (`?classError` for details):

```
> classError(iclust3$classification, iris[,5])
$misclassified
[1] 69 71 73 78 84

$errorRate
[1] 0.03333333
```

The `uncerPlot` command allows us to visualize the uncertainty implied by the mixture model to see how uncertain the model was about the misclassified samples.

```
> uncerPlot(iclust3$z, iris[,5])
```

In the uncertainty plot the vertical lines indicate the misclassified samples. As you can see those tend to be among the observations that the mixture model was most uncertain about with respect to which component they belonged to.

More details on MCLUST

See the MCLUST docs for in depth discussion of the use of `MCLUST`. The examples illustrated above were drawn from this documentation.