FIT2086 Studio 10
Clustering and Mixture Modelling

Daniel F. Schmidt

October 1, 2018

Contents

1 Introduction

Studio 10 gives us a brief introduction to clustering in R using the `mclust` package.

During your Studio session, your demonstrator will go through the answers with you, both on the board and on the projector as appropriate. Any questions you do not complete during the session should be completed out of class before the next Studio. Complete solutions will be released on the Friday after your Studio.

2 Mixture Modelling with Gaussian Mixtures

To get started, begin by installing the `mclust` package in R. This will give us access to some Gaussian mixture modelling software for univariate and multivariate Guassian distributions, and brings with it several useful datasets for playing around with. Install the package and then load it.

1. The first dataset we will look at is a diabetes data set containing three measurements made on 145 non-obese adult patients classified into three groups. To load this

   ```
   data(diabetes)
   ```

   Have a look at the dataset.

2. Now let us split the dataset into the classes and the features. First, let us look at the (marginal) distribution of the classes

   ```
   class = diabetes$class
   table(class)
   ```

   In this dataset, we know the different groups that the people belong to. However, in general clustering applications we do not know classes and want to use the features to try and discover if there are categories within the data ("intrinsic classification"). Let us extract only the features using

   ```
   X = diabetes[,-1]
   ```

   We can use the software to produce a nice visualisation of the features in a series of scatterplots. To do this, use the command

   ```
   clp <- clPairs(X, class, lower.panel = NULL)
   clPairsLegend(0.1, 0.3, class = clp$class, col = clp$col, pch = clp$pch)
   ```

   This produces a scatter-plot for all variables against each other, with the points colour coded by the diabetes category to which they belong.

3. Now, let us fit a Gaussian mixture model to the features and see how many classes it discovers. Initially, we will restrict each class to be modelled using a different univariate normal distribution for each column of the features. This is equivalent to using a multivariate normal distribution with a diagonal covariance matrix. To do this, we run

   ```
   mod1 = Mclust(X, modelNames = "VVI")
   ```

   The option `"VVI"` tells the clustering software to allow each class in the mixture model to have varying volume (each class can have a different covariance matrix), varying shape (each univariate normal distribution for the features in each class can have a different variance) and be diagonal.

The software tries a number of classes and chooses to use the one with the smallest Bayesian information criterion (BIC) score. Recall that the BIC is a penalized likelihood method which adds a penalty proportional to the number of parameters to the negative log-likelihood to form a goodness-of-fit score. Note that in this package, the authors use the log-likelihood instead of the negative log-likelihood, so they subtract the penalty rather than adding it; we are therefore looking for large values of BIC in this case, rather than small ones (confusing ...).

To examine the model that was fitted, we can use

```
summary(mod1, parameters = TRUE)
```

Scrolling through the output, we can see that the BIC score has suggested four classes as optimal. You can see the mixing probabilities (weights for each of the classes/proportions of individuals in the data allocated to each class), the means for each of the features in each of the classes, and the covariance matrices for each of the four classes. Note that because of the `I` in the `"VVI"` specification, the covariance matrices are all diagonal (zero covariance between features). To see how our data is distributed between the classes we can use

```
plot(mod1, what = "classification")
```

This shows the way in which the datapoints are placed into the four classes that have been discovered. We can see how this lines up with the diabetes categories, we can use

```
table(class, mod1$classification)
```

We see that there appears to be little overlap between the diabetes categories within the four classes, but that two classes are required to approximate the `Overt` category. This is suggestive that our diagonal covariance structure is possible too strict.

4. We know that there are actually three diabetes categories in this data, though our mixture model discovered four. There can be two reasons for this discrepancy: (i) the three categories, which were very likely determined by humans using reasoning and observation, are actually not sufficient to describe the data, or (ii) the restriction to diagonal covariance matrices means our mixture model classes are incapable of compactly describing the data. We can actually ask the `mclust` package to try the entire range of different covariance matrix structures. To see the different models we can use

```
help(mclustModelNames)
```

To ask `mclust` to try all possible combinations of covariance matrices, use

```
BIC <- mclustBIC(X)
BIC
```

We can see this displays the BIC scores for models with between 1 and 9 components/classes, for the multiple different covariance matrix structures. If we use

```
summary(BIC)
```

we can see the top few models (number of classes, covariance structure) in terms of the BIC score. We can see that the `VVI` model we specified above is not in the top three. What are the top three model structures?

5. We can also plot this information

   ```
   plot(BIC)
   ```

   From this we can see that the very restrictive models (for example, `EEI`, which is spherical covariance matrix, equal volume for each class, i.e., equivalent to k-means) do very poorly in this data, as they have very poor BIC scores. In contrast, the most flexible covariance model `VVV`, which has arbitrary covariance structure, achieves the best BIC score when there are three classes.

6. We can ask `mclust` to use this information to fit a good mixture model to our data. Do this using

   ```
   mod1 = Mclust(X, x = BIC)
   summary(mod1, parameters = TRUE)
   ```

   Using this output, answer the following questions:

   (a) What proportion of people are in class 1?

   (b) What are the average glucose levels in the three classes?

7. Let us now see how the discovered classes match up against the known diabetes categories. First, we can tabulate the classes the individuals have been assigned to against their diabetes category:

   ```
   table(class, mod1$classification)
   ```

   We see that the three classes discovered by the mixture modelling software closely correspond to the three different diabetes categories. This is quite impressive, given that the software did not have access to this information, and instead discovered it indirectly by clustering. Of course, in many situations in real practice such clean correspondence does not usually occur, but this demonstrates what is possible. We can visualise the clustering using

   ```
   plot(mod1, what = "classification")
   ```

   Which of the three three diabetes categories seems easiest to discover as a seperate class?

3 k-means Clustering

In this question you are asked to program a simple implementation of the k-means clustering algorithm.

1. Write an R function that takes a dataframe, and a number of clusters to fit, and tries to find the best centroids for these clusters using the k-means algorithm. The code provided in `my.k.means.R` will give you a starting point. You can use the `sample()` function to generate random integers. Your function should return:

   - The centroids of the k clusters
   - The number of the cluster to which each datapoint is assigned
   - The sum of the distances from each point to its closest cluster centroid

2. Try running your k-means algorithm on the data provided in the file `kmeans.test.csv` with $k = 3$. After it has run, scatterplot the points colour coded by the cluster they belong to. You can do this using

   ```
   plot(X, col=Cluster)
   ```

   Run it several times. How often does it get stuck in a poor local minima?