University College Dublin

Research Project

Developing an R Shiny Application for Model-Based Clustering and Classification

Authors:
Alan Jeffares
Aralt O' Callaghan

Student Numbers:
14329336
14507617

May 25, 2018

Abstract

This report details the development of a web application for model-based clustering and classification. The application is developed using the shiny package in the R environment. Clustering and classification in the application is identical to that implemented in the mclust package taking a parametric, model-based approach to the task. Background to both concepts is discussed as well as an introduction to the shiny package and a brief overview of the mclust package.

Simplicity and efficiency were the two main goals throughout development with the final application intended to be user-friendly and intuitive. The final application is summarised in detail with an example use case given in a step by step format.

Contents

1 Introduction

One of the drawbacks of developing a package in any computer languages is that users without literacy in that particular language are unable to utilize the functionality of the package being developed. The shiny package (Chang et al. [4]) offers a solution to this issue in the R framework by allowing developers to build interactive web applications that require no code knowledge from users, in principle. This report discusses the development of one such application for mclust (Fraley et al. [10], Fraley and Raftery [8]), a contributed R package for model-based clustering and classification.

Clustering considers the problem of determining the intrinsic structure of data when no information other than the observed values is available. Model-based clustering (Fraley and Raftery [7]) is a parametric approach to clustering where it is assumed that the data are generated by a mixture of underlying probability distributions in which each component represents a different cluster.

Classification differs from clustering in that the true class labels of the data are known and are used to learn the underlying structure of the data. Model-based classification [8] involves relaxing assumptions from mixture discriminant analysis to essentially perform model-based clustering on each class.

The application that was developed aims to include the key functionality of the the mclust package while presenting it in an intuitive, user-friendly and concise format. The development process involved adapting the mclust package to fit into the shiny framework while taking careful steps to ensure that the efficiency and functionality was maintained as much as possible.

This report is organised as follows. In Section 2, a background is given to all relevant theory and software including discussions on model-based clustering, model-based classification, R shiny and the mclust package. An overview of the application that was eventually developed is included in Section 3, including an overview of the applications interface, some further detail on a number of the key features and an insight into the architecture behind the application. A step by step sample use case of clustering with the application is included in the appendix. A final section summarises and indicates extensions to the application.

2 Background

2.1 Model-Based Clustering

Model-Based Clustering considers the problem of determining the structure of unlabelled data without prior information about the number or composition of clusters. The assumption is made that clusters or components are generated from Gaussian distributions with varying geometric properties with partitions determined by the EM (expectation-maximisation) algorithm. Different parameterisations and cross-cluster constraints may be considered and they are compared based on the Bayesian Information Criterion (BIC). A brief overview of this process is given in this section based on the approach taken by Fraley and Raftery [7].

2.1.1 Probability Models for Cluster Analysis

The primary assumption in model based clustering is that the data is generated by a mixture of underlying probability distributions in which each component represents a different group or cluster. Given observations $\mathbf{X} = (\mathbf{x}_1, ..., \mathbf{x}_n)$, let $f_k(\mathbf{x}_i|\theta_k)$ be the density of an observation $\mathbf{x}_i$ from the kth component. The *mixture likelihood* approach maximises

$$L_M(\theta_1, ..., \theta_G; \tau_1, ..., \tau_G|\mathbf{x}) = \prod_{i=1}^{n} \sum_{k=1}^{G} \tau_k f_k(\mathbf{x}_i|\theta_k)$$

where τ_k is the prior probability that an observation belongs to the kth component. We consider the case where the probability density function is multivariate normal which reduces to the case where clusters are ellipsoidal, with means μ_k and covariance Σ_k determining their geometric characteristics. In this case $\theta_k = (\mu_k, \Sigma_k)$. This density has the form

$$f_k(\mathbf{x}_i|\mu_k, \Sigma_k) = \frac{\exp\{-\frac{1}{2}(\mathbf{x}_i - \mu_k)^T \Sigma_k^{-1}(\mathbf{x}_i - \mu_k)\}}{(2\pi)^{\frac{p}{2}}|\Sigma_k|^{\frac{1}{2}}} \tag{1}$$

Banfield and Raftery [2] proposed a reparameterization of the covariance matrix Σ_k which allows us to specify that some, but not all, features be the same for all clusters. This eigenvalue decomposition takes the form

$$\Sigma_k = \lambda_k D_k A_k D_k^T$$

D_k is an orthogonal matrix of eigenvectors that determines the orientation of the corresponding ellipsoid. A_k is a diagonal matrix with elements proportional to the

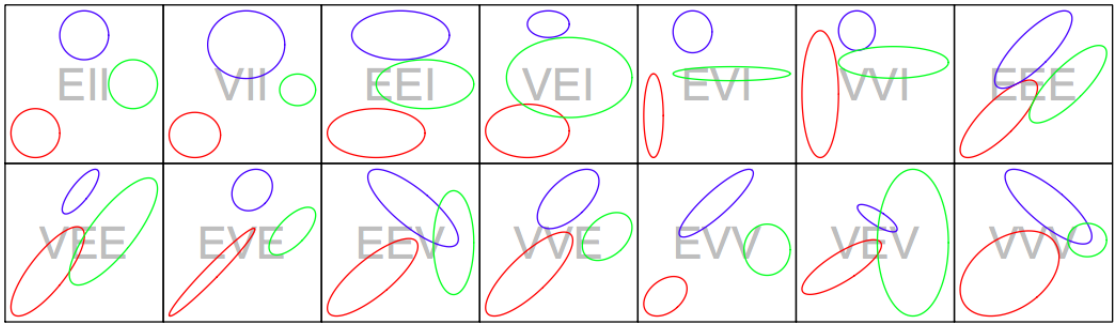


Figure 1: Plot of different parameterizations of the decomposed covariance matrix. Letters correspond to volume, shape and orientation respectively. E implies equal across all components, V implies variable across components while I implies the identity matrix.

eigenvalues which determines the shape of the ellipsoid. λ_k is a scalar that specifies the volume of the ellipsoid. A visualisation of the 14 possible combinations of constraints on the covariance matrix is given in Figure 1.

2.1.2 The EM Algorithm

The EM algorithm [5] is a general approach to maximum likelihood in the presence of incomplete data. The "complete" data are considered to be $y_i = (\mathbf{x}_i, \mathbf{z}_i)$, where z_i with

$$z_{ik} = \begin{cases} 1 & \text{if } \mathbf{x}_i \text{ belongs to group } k \\ 0 & \text{otherwise} \end{cases}$$

constitutes the "missing" data. It is clear that the resulting *complete-data loglikelihood* is

$$l(\theta_k, \tau_k, z_{ik}|\mathbf{x}) = \sum_{i=1}^{n} \sum_{k=1}^{G} z_{ik}[\log \tau_k f_k(\mathbf{x}_i|\theta_k)] \tag{2}$$

This approach is sufficient for applying the EM algorithm in which the expectation step calculates $\hat{z}_{ik} = E[z_{ik}|\mathbf{x}_i, \theta_1, ..., \theta_G]$ and the maximisation step seeks to maximise the terms $\hat{\tau}_k$, $\hat{\mu}_k$, and $\hat{\Sigma}_k$ given the current estimate $\hat{z}_{ik}$ and Equation (2). Each iteration of the EM algorithm gives new estimates of τ_k, μ_k and Σ_k with higher log-likelihood than the previous ones. This iteration between E-step and M-step is repeated until convergence is obtained.

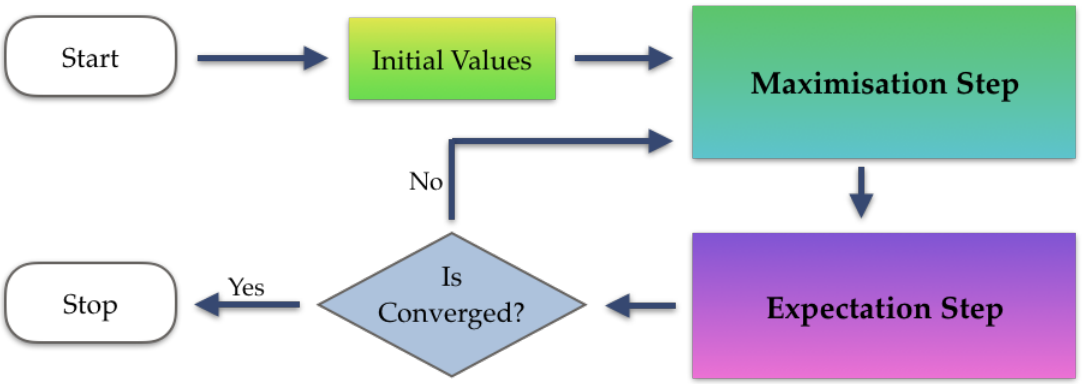


Figure 2: The EM algorithm

In practise the EM algorithm is used with good results regardless of initialisation, however performance can be improved by a better choice of the initial group membership z_{ik}. Agglomerative Hierarchical Clustering aims to give a better than random choice of initial group memberships for each number of possible groups. This process works by beginning with each observation in a cluster by itself and at each stage merging the two nearest clusters using a maximum-likelihood criterion as a distance metric. The resulting tree may be cut at any choice of number of groups to obtain initial group memberships. The entire EM process is summarised in Figure 2.

2.1.3 Model-Based Strategy for Clustering

The approach to the general model-based strategy discussed in this section can be summarised in 5 steps [7].

- Determine the maximum number of clusters to consider and a set of candidate parameterisations (see Figure 1) of the Gaussian model to consider.
- Do agglomerative hierarchical clustering for the unconstrained Gaussian model and obtain the initialisation of classifications for each number of clusters.
- Do EM algorithm for each parameterisation with each number of clusters.
- Compute the BIC score for each combination of number of clusters and parameterisations with optimal parameters from previous step. BIC is defined as $BIC = 2l_m(x, \hat{\theta}) - m_m \log(n)$ where m_m is the number of independent parameters to be estimated in the model. This gives a matrix of BIC values corresponding to each possible combination of parameterisation and number of clusters.

- Plot the BIC values for each model. A decisive first local maximum indicates strong evidence for a model.

2.2 Model-Based Classification

Discriminant analysis, is a supervised statistical technique where the class labels of observations are used to learn the structure of the data. The G number of groups or classes is known, as well as the group membership of each observation. Knowing the structure of the data then allows for the classification of new observations. This section describes how linear discriminant analysis (LDA) can be extended to quadratic discriminant analysis (QDA) to eigenvalue decomposition discriminant analysis (EDDA) and then to mixture discriminant analysis (MDA), as discussed in Fraley and Raferty 2002 [8].

2.2.1 Discriminant Analysis Background

Linear, quadratic and mixture discriminant analysis all have a statistical modelling basis - they are parametric methods. They are probabilistic due to the assumption that the observations in the kth class are generated by a probability distribution specific to that class $f_k(.)$. In order to classify a new observation into one of the G known classes, the posterior probability of belonging to each group given the data must be known. A new observation will be classified as belonging to the class for which it has the largest posterior probability, hence minimising the misclassification rate. If π_k is the proportion of observations in class k, then using Bayes theorem the posterior probability can be re-expressed as

$$\mathbb{P}(k|\mathbf{x}_i) = \frac{\pi_k f_k(\mathbf{x}_i|\theta)}{\sum_{k\prime=1}^{G} \pi_{g\prime} f_{k\prime}(\mathbf{x}_i|\theta)}$$

Linear discriminant analysis begins with two main assumptions: observations in the kth class are generated by a multivariate normal probability distribution specific to that class; and also covariance matrices are equal across classes. Taking the log of the posterior probability and using the assumption of equal covariance matrices leads to cancellations allowing the posterior probability to be reformulated in terms of a discriminant function

$$\log(\pi_k f_k(\mathbf{x}_i|\mu_k, \Sigma_k)) = \log(\pi_k) + \mathbf{x}_i^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k = \delta_k(\mathbf{x}_i)$$

Log is a monotonic, increasing function, hence a new observation will be classified as belonging to the class for which it has the largest discriminant function. The

Table 1: Covariance matrix constraints across the three different discriminant analysis methods where the notation is the same as that used in Figure 1

$\Sigma_i = \lambda_k D_k A_K D_K^T$			
	$\boldsymbol{\lambda}$	**A**	**D**
LDA	E	E	E
QDA	V	V	V
EDDA	E/V	I/E/V	I/E/V

boundary lines between classes (where the probability of belonging to one class is equal to the probability of belonging to another class) are therefore linear. Quadratic discriminant analysis is an extension of LDA in which the assumption of equal covariance matrices no longer applies (Σ_k is no longer constant) The convenient cancellations therefore no longer hold leading to the following discriminant function

$$\log(\pi_k f_k(\mathbf{x}_i|\mu_k, \Sigma_k)) = \log(\pi_k) - \frac{1}{2} log|\Sigma_k| - \frac{1}{2}(\mathbf{x}_i - \mu_k)^T \Sigma_k^{-1}(\mathbf{x}_i - \mu_k) = \delta_k(\mathbf{x}_i)$$

where the boundary lines are no longer linear due to non-constant covariance matrices.

2.2.2 Eigenvalue Decomposition Discriminant Analysis

Eigenvalue decomposition discriminant analysis (Bensmail and Celeux 1996 [3]) is very similar to the previous two methods except there is more flexibility than LDA while at the same time allowing more structure than the unconstrained covariance matrix underlying QDA. Using the eigenvalue decomposition

$$\Sigma_k = \lambda_k D_k A_k D_k^T$$

constraints can be imposed on the covariance matrices for each class. There are therefore 14 different models whereby the volume, shape and orientation may be constrained as in model-based clustering (as discussed in 2.1.1). This is summarised in Table 1.

2.2.3 Mixture Discriminant Analysis

Hastie and Tibrishani 1996 [11] suggested an alternative model-based approach to discriminant analysis whereby the density for each class itself is allowed to be a

mixture of multivariate normal distributions

$$f_j(\mathbf{x}|\theta_k) = \sum_{k=1}^{G_j} \pi_{jk} f_k(\mathbf{x}|\mu_{jk}, \Sigma_{jk})$$

while assuming all component covariance matrices are the same ($\Sigma_{jk} = \Sigma$) and the number of mixture components for each class is known in advance.

Relaxing the assumption of known number of mixture components as well as the assumption of equal covariance matrices by allowing them to vary both within and between classes, perhaps with cross-component constraints, would result in the data determining which parameterization of the covariance matrix and which number of mixture components is optimal for each class. This generalisation of MDA is called MclustDA and combines [3] and [11] into essentially performing model-based clustering on each class.

2.3 R Shiny

Shiny [4] is an open source R (`https://www.r-project.org`) package that may be used to build interactive web applications (apps) directly from R. Unlike most other application development software, Shiny requires no knowledge in Java, Javascript, PHP, CSS or HTML. Shiny only requires familiarity with the R programming language.

2.3.1 Structure of a Shiny App

A Shiny application's primary use case is to make a block of R code accessible without knowledge of R and interactive to predefined changes in code. An intuitive (although unlikely to be efficient) approach to shiny is to consider the case when a fully functional R script is available and is required to be moulded into a shiny web application. There are two primary components to this (and every) application:

- The Server - This contains an adapted version of the original R script and may be thought of as the internal mechanics behind the application.
- The User Interface (UI) - This is where the front-end is defined e.g. how the user interacts with the application and the general aesthetics.

In order for the UI to interact with the server, certain parts of the server such as variables or plots must become *reactive*. For example a reactive variable, `x`, may

be required to take values from 1-10 based on the users input via an interactive slider. In this case `x` is a reactive variable and must be treated as such in the server. There are numerous reactive inputs and outputs that may be defined in the UI section. Inputs include checkboxes, sliders and file uploaders with different inputs being appropriate for different purposes. Outputs are generally similar to the usual R outputs such as tables, plots and images.

In order to maximise the speed efficiency in a Shiny application it is important to only use the minimal amount of computation required for any given interaction from the user. A change in a single input generally shouldn't require running the entire server function from start to finish. A toy example is the case when a shiny application is developed with two inputs: training data and test data. If the applications purpose was to fit a model to the training data and then use that model to make predictions on the test data, *efficient coding* will run the code in such a way as to minimise computation. A interaction in the training data input will require most of the server to be run, however an interaction in the test data input should not require the model to be fit again. In other words, the model should be connected with both the training data input and the test data input but should only be reactive to the training data input.

2.3.2 Basic Example

In this section a very simple Shiny app is detailed in order to demonstrate the theory described in 2.3.1. The app is designed to plot a histogram of a user specified number of standard normally distributed observations. In other words there is one user input (number of observations) and a single output (a histogram). The back-end is included in Figure 3(a) where there are two main commands

- `ui <- fluidpage()` - A function with arguments corresponding to the user interface.
- `server <- function(input, output){}` - A function containing the server section of the application.

A slider was selected as the method for allowing a user to input the number of observations. The command `sliderInput()` was used to achieve this with the argument `inputId = "num"` enabling the server to access this variable as `input$num`. Similarly, the UI contains the function `to` plot a histogram in the output. Now the server function can simply use the usual R commands for plotting a histogram wrapped in the command `renderPlot({})` in order to define the plot as being

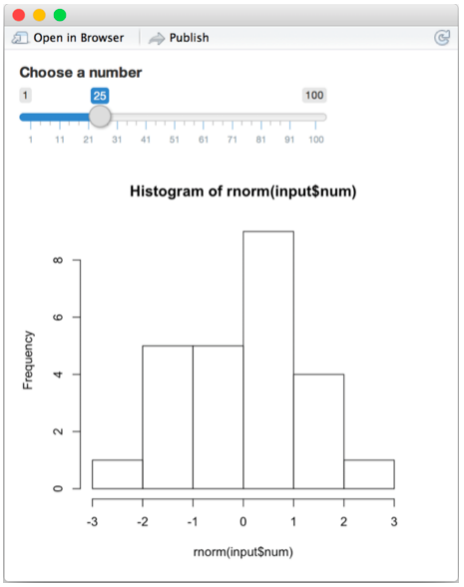
```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}

shinyApp(ui = ui, server = server)
```

(a) Input code



(b) The application

Figure 3: A very simple Shiny application that plots a histogram of a user specified number of standard normally distributed observations

reactive to (or *listening for*) changes in the `input$num` variable. The resulting shiny application is included in Figure 3(b).

2.4 Mclust

Mclust (Fraley et al. [10], [8]) is a contributed R package for model-based clustering, classification, and density estimation based on finite normal mixture modelling. The package contains functionality for implementing model-based clustering (as described in Section 2.1) as well as model-based classification (as described in Section 2.2). There is additional functionality for displaying and visualising the models along with clustering and classification results. The Mclust package is compatible with Shiny (as described in Section 2.3) and, although source code needed to be adjusted to fit in with the Shiny framework, the application described in Section 3 is entirely based on the functionality of the Mclust package. The package may be accessed via `library(mclust)`. The model-based clustering functionality may be accessed with the command `Mclust()` while the model-based classification may be accessed with the command `MclustDA()`.

Figure 4: The clustering demo section of the shiny application

3 Software Demonstration

3.1 Application Overview

What follows is a summary of the Shiny Mclust Application that was developed for this project. This section contains a brief overview of the applications features while Section 3.2 looks into some of the key components of the application in more detail with Section 3.3 examining how the application was designed to maximise efficiency. The application is hosted on the open source shiny server and may be accessed at `https://mclust.shinyapps.io/apphere`. The application is roughly split into two sections, clustering and classification.

3.1.1 Clustering

The clustering section of the shiny application includes the main features of the mclust (Section 2.4) implementation of model-based clustering (Section 2.1). The key features of the clustering demo section of the application are included in this section with numbers corresponding to those in Figure 4.

1. The Navigation Bar - Allows users to easily navigate between clustering, classification and about sections of the application with embedded menu's for choosing between a demo and a user uploaded dataset.
2. Dataset Input - Dropdown menu allowing users to choose between four built in datasets for analysis in the demo section.
3. Variable Selection - Dynamic dropdown menu allowing users to select variables to be plotted on either axis dependant on dataset. Discussed in more detail in Section 3.2.1.
4. Number of Clusters - Allows users to choose between minimum and maximum number of clusters to be considered using a slider with two handles.
5. Clustering Plot Type - Radio buttons allowing user to select which plot is to be displayed from the clustering performed on the dataset. Discussed in more detail in Section 3.2.2.
6. Output Tabs - Tabsets allowing user to select either graphic outputs or summary details of the clustering performed on the dataset.

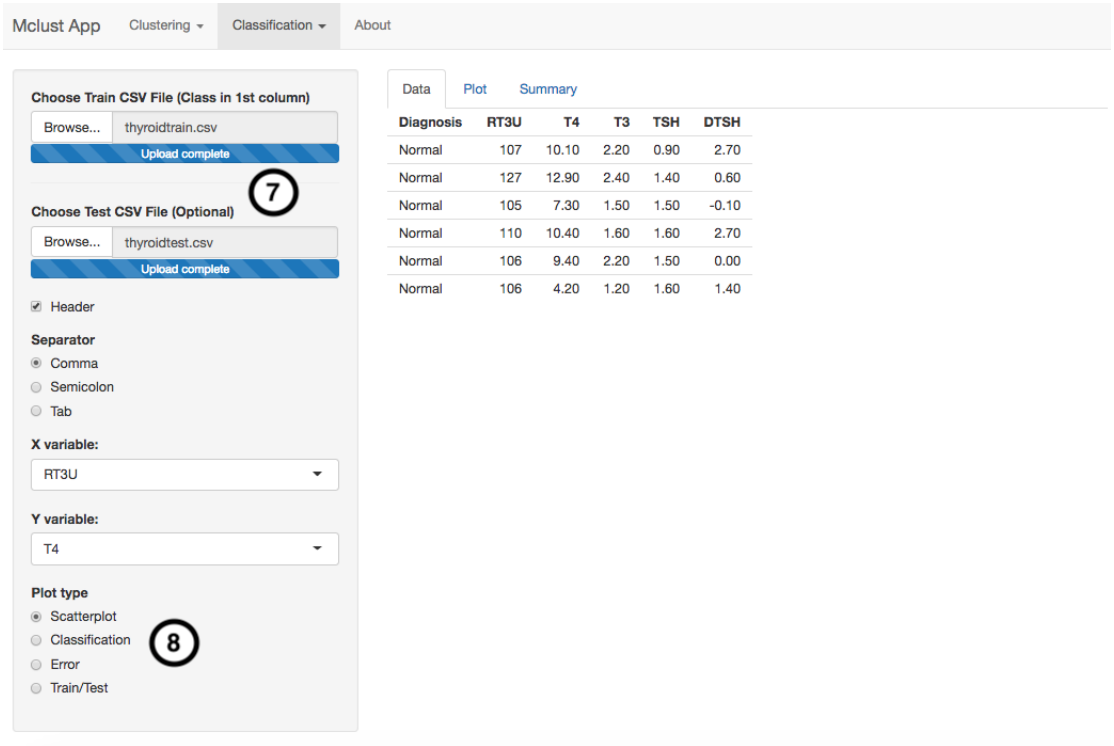


Figure 5: The classification upload section of the shiny application

3.1.2 Classification

The classification section of the shiny application includes the main features of the mclust (Section 2.4) implementation of model-based classification (Section 2.2). This section describes two new features in addition to those included in clustering (excluding *"4. Number of Clusters"* as the number of classes is known in supervised learning). The key features of the classification upload section of the application are included in this section with numbers corresponding to those in Figure 5.

7. Dataset Upload - A collection of settings available in clustering and classification that allows users to upload local datasets. Discussed in more detail in Section 3.2.3.
8. Classification Plot Type - Radio buttons allowing user to select which plot is to be displayed from the classification performed on the dataset. Discussed in more detail in Section 3.2.2.

3.2 Feature Details

3.2.1 Dynamic User Interface

Dynamic user interfaces require that, in addition to the server function being reactive to changes in the input, other inputs in the UI are also reactive to changes in the input. In the shiny mclust application demo section it was required that the variable selection input was reactive to the dataset input. The dropdown menus were required to display the variables relevant to the dataset that was currently selected. The `renderUI()` function is used in the server section of the application in order to allow relevant inputs to be defined in the server rather than the UI thus allowing them to be dynamically reactive to other inputs.

3.2.2 Plot Type

Plots of the clustering and classification results are some of the most important outputs of the application. The clustering section outputs four plots as described by Fraley and Raftery [9]

- Classification - Plot of the final clusterings of the data.
- BIC - Plot of BIC values used for choosing the number of clusters/ parameterisation combination, with number of clusters on the x-axis, BIC scores on y-axis and parameterisations differentiated by colour and symbol.

- Uncertainty - Plot of classification uncertainty with bigger observations indicating a higher uncertainty.
- Density - Contour plot of estimated density.

The classification section also outputs four different plots

- Scatterplot - Plot of training data with points marked based on the known classification. Ellipses corresponding to covariances of mixture components are also drawn.
- Classification - Plot of data with points marked based on the predicted classification.
- Error - Plot of data with misclassified points marked as black filled shapes where the shape corresponds to the class the observation was incorrectly classified as.
- Train/Test - Plot of training and test data with points marked according to whether they belong to the training set or the test set. Only available in the upload section.

The application produces all plots in two dimensions based on the variables selection input. This functionality required adapting the mclust source code as the package only plots matrices of plots for most of the desired plot types. This adds considerable complexity to the server as it requires taking the output from the mclust model and applying the custom two dimensional plot functions to this output.

3.2.3 Dataset Upload

The application has the functionality for a user to upload their own dataset in CSV format, for both clustering and classification. There are two features to allow flexibility in how a dataset is uploaded: a reactive checkbox input notifying whether or not the first row contains column names (header) and a reactive radio buttons option determining how the CSV file is separated (comma, semi-colon or tab). A tabset entitled 'Data' displays the head of the data, allowing the user to ensure the data has been read in appropriately. Within the clustering section of the application it is possible to upload a dataset with the class labels as the first column of the dataset. In this case a confusion matrix will be printed in the summary section. Both testing and training datasets can be uploaded as part of the classification upload section, whereby class labels must be in the first column. Uploading a testing dataset is optional.

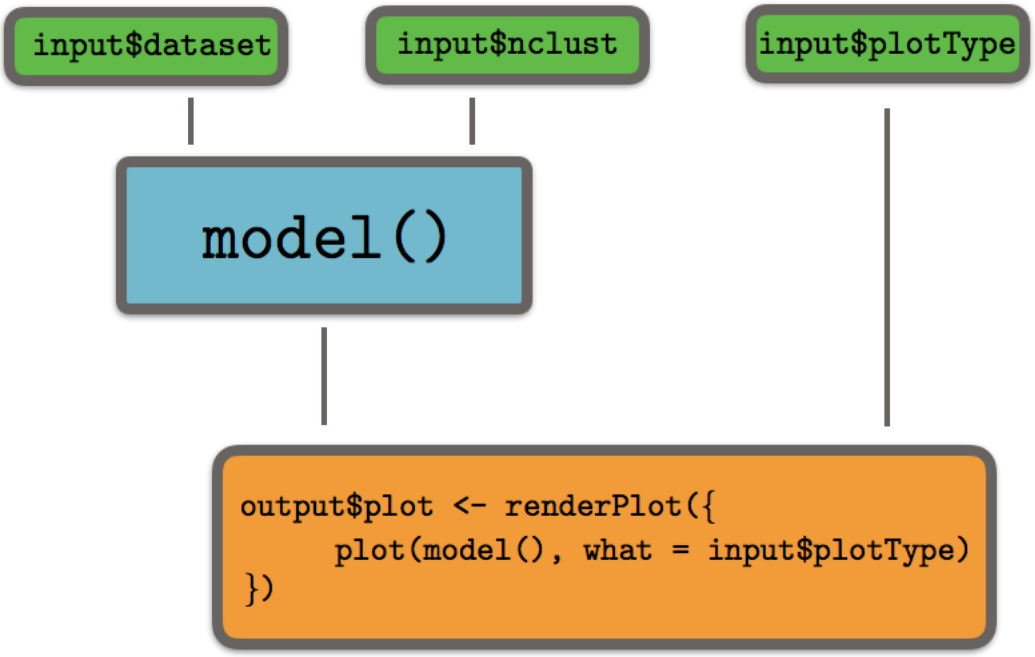


Figure 6: A simplified version of the architecture behind the shiny mclust application.

3.3 Application Architecture

As discussed in Section 2.3.1, a challenge in designing any shiny application is managing the reactivity within the application in such a way as to maximise the *efficiency*. In other words, any change in input should run the minimal amount of computation in order to perform it's task. This approach of minimising computation is taken throughout the shiny mclust application most notably in fitting the model as detailed in this section.

Figure 6 details a simplified version of a subsection of the applications architecture for clustering. In this subsection there are three of the input nodes included as described in 3.1.1: dataset input, number of clusters input and plot type input. The `model()` processing step is where the mclust package is used to apply model based clustering to the data. This is the most computationally heavy part of the process. The bottom processing step represents a reactive function that plots the results from the model. Connections between processing steps are denoted by lines. The reactive plot function leads to different amounts of computation depending on which input node updates and may be split into two distinct cases.

The first case occurs when either the dataset or the number of clusters input is updated. Before any change in the inputs, all nodes and processing steps may be considered up to date or *valid*. As one of the inputs is adjusted that node and each subsequent processing step downstream are notified that they are out of date or *invalid*. No code has been run at this stage but the entire network has been updated on its validity. The next step is for the code to run and this operates from the bottom of the network. As `output$plot` is invalid, it runs requiring both `model()` and the plot type input. The plot type input is still valid and thus returns the same value from its cached memory while `model()` is invalid and must run again using the current values from its input nodes. Clearly in this case there is no way of avoiding computing both processing steps.

The second case occurs when the plot type input is updated. In this case the only downstream processing step to be notified of its invalidity is the `output$plot`. The `output$plot` is now required to run (just as in the first case) and looks upstream for the values it requires. This time `model()` is valid and returns its cached value without running while the plot type input returns its new value.

This simplified version of the application demonstrates that subtle differences in network architecture can lead to substantial differences in performance. The first case demonstrates the situation where there is no option but to re run the model due to changes in the model essential inputs. The second case demonstrates the situation where the change in input only requires different output from *the same* model. Ensuring that connections are sparse is not initially intuitive as it is not generally required when writing typical R code and it is not difficult to inadvertently connect nodes unnecessarily (in this example an unintentional connection might occur between the input plot and the `model()`). Conscious effort was made throughout the application to ensure maximum efficiency in the network by ensuring minimal connections throughout.

4 Discussion

This report has described how multivariate normal mixture models can be used in both clustering and classification methodologies. In the case of clustering, model-based agglomerative hierarchical clustering is used to initialise the EM algorithm for various parameterizations and number of clusters, with Bayesian Information Criterion used to determine the best combination. In the case of classification, the assumptions in Hastie and Tibrishani [11], of known number of mixture components and of equal covariance matrices are relaxed, to essentially perform model-based clustering on each class.

An introduction to R shiny and mclust, detailing the main features of both, is given to precede a detailed overview of how the two have been combined to produce the mclust shiny application. A software demonstration takes the reader through both sections of the application, clustering and classification, explaining some of the key features, with the architecture of the application explained in detail.

Careful attention was made to balance simplicity with functionality while attempting to exploit the main components of mclust. Features of mclust such as the ability to choose model names in clustering and the option of Eigenvalue Decomposition Discriminant Analysis (EDDA) in classification, were not deemed necessary with the potential users of the application's background kept in mind, although these could be easily implemented in a future version.

Care was made to ensure the speed of the application was consistent with the speed of mclust in the R framework. A large dataset of 7195 observations and 22 features, relating to anuran species recognition through their calls (Dheeru et al. [6]), was obtained with the intent of pushing the capabilities of the application. Model-based clustering was performed on this dataset in both the R framework and through the application where no discrepancy in time was observed.

Limitations of the application include the constrained summary output, mirroring a difficulty with mclust itself. A possible extension of the application would be to determine a user-friendly way of displaying the extensive output from mclust such as the mixing proportions, matrix of means and covariance matrices.

References

[1] E. Anderson. The species problem in iris. *Annals of the Missouri Botanical Garden*, 23(3):457–509, 1936.

[2] J. D. Banfield and A. E. Raftery. Model-based gaussian and non-gaussian clustering. *Biometrics*, pages 803–821, 1993.

[3] H. Bensmail and G. Celeux. Regularized gaussian discriminant analysis through eigenvalue decomposition. *Journal of the American statistical Association*, 91(436):1743–1748, 1996.

[4] W. Chang, J. Cheng, J. Allaire, Y. Xie, and J. McPherson. *shiny: Web Application Framework for R*, 2017. R package version 1.0.5.

[5] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

[6] D. Dheeru and E. Karra Taniskidou. UCI machine learning repository, 2017.

[7] C. Fraley and A. E. Raftery. How many clusters? which clustering method? answers via model-based cluster analysis. *The computer journal*, 41(8):578–588, 1998.

[8] C. Fraley and A. E. Raftery. Model-based clustering, discriminant analysis, and density estimation. *Journal of the American statistical Association*, 97(458):611–631, 2002.

[9] C. Fraley and A. E. Raftery. Mclust version 3: an r package for normal mixture modeling and model-based clustering. Technical report, WASHINGTON UNIV SEATTLE DEPT OF STATISTICS, 2006.

[10] C. Fraley, A. E. Raftery, T. B. Murphy, and L. Scrucca. mclust version 4 for r: Normal mixture modeling for model-based clustering, classification, and density estimation. 2012. *University of Washington: Seattle*.

[11] T. Hastie and R. Tibshirani. Discriminant analysis by gaussian mixtures. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 155–176, 1996.

Appendix

Clustering Example with Shiny Mclust Application

This section details a step by step example of the model-based approach to clustering the iris dataset (Anderson [1]) within the shiny mclust application which involves 150 observations in which measurements are taken in centimeters of the variables: sepal length and width and petal length and width for 50 flowers from each of 3 species of iris. Figure 7 depicts how the application opens on a user's interface, to mclust performing model-based clustering on the preloaded iris dataset, in which the classification of sepal length versus sepal width is displayed.

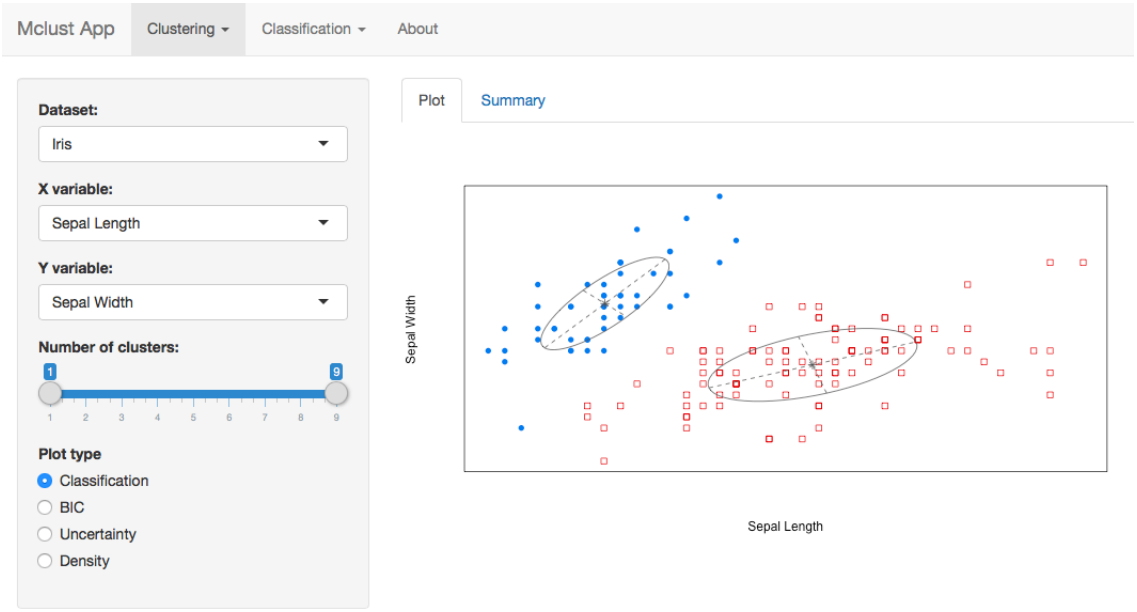


Figure 7: The opening page of the shiny mclust application.

Changing the x-variable from sepal length to petal length, in Figure 8, will update the plot, so rather than re-running the model, the same output will be reused to update the output.

In Figure 9, the minimum and maximum number of clusters is changed - by means of the input slider - to 3 and 7 respectively. As the model is reactive to changes in the number of clusters, the server re-runs the model with the new parameters and the new output is displayed, showing a 3 cluster solution.

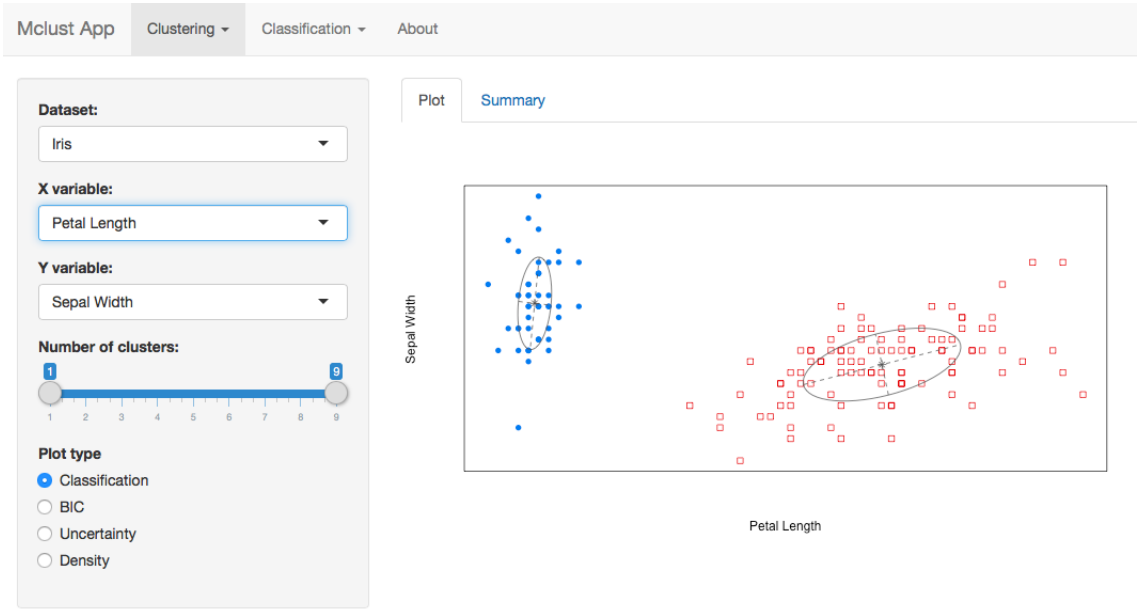


Figure 8: Changing the x-variable from Sepal Length to Petal Length.

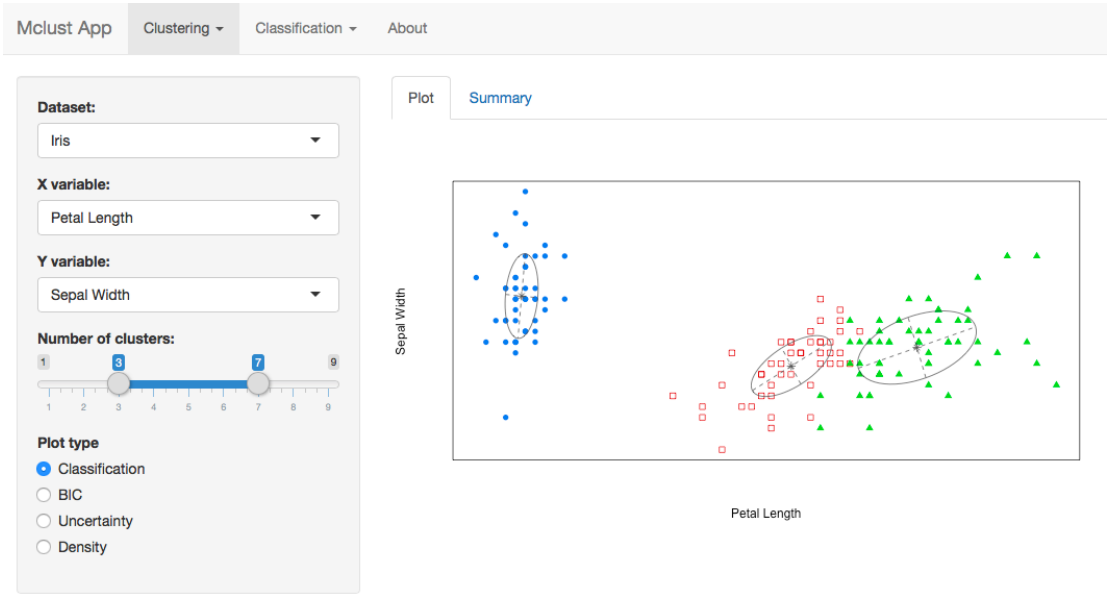


Figure 9: Adjusting the number of clusters to consider and examining the resulting classification plot.

Changing the radio button option of plot type from classification to BIC displays a plot of the BIC scores for each number of clusters between 3 and 7 on the x-axis and each parameterisation differentiated by colour and symbol, on the y-axis. A VEV model with 3 clusters is chosen, due to having the highest BIC score as seen in Figure 10.

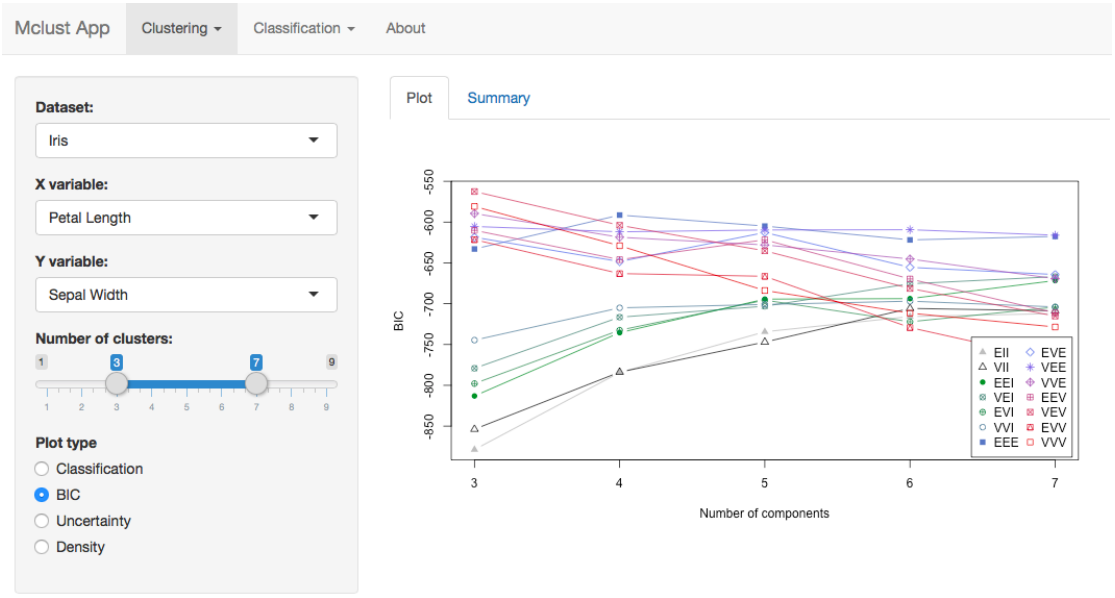


Figure 10: Examining the output plots - BIC.

Figure 11 displays a change in plot type from BIC to uncertainty. Due to the soft-clustering nature of model-based clustering, a point will be assigned to its respective cluster with a certain probability. The lower the probability of a point belonging in the cluster it has been assigned to, the larger the point being displayed will be.

The final plot type output displays the density estimation for each cluster, as demonstrated in Figure 12. Each cluster is generated from an unobservable multivariate normal probability density function with parameter estimates specific to that cluster. These estimated densities are displayed using a contour plot superimposed on the classification plot.

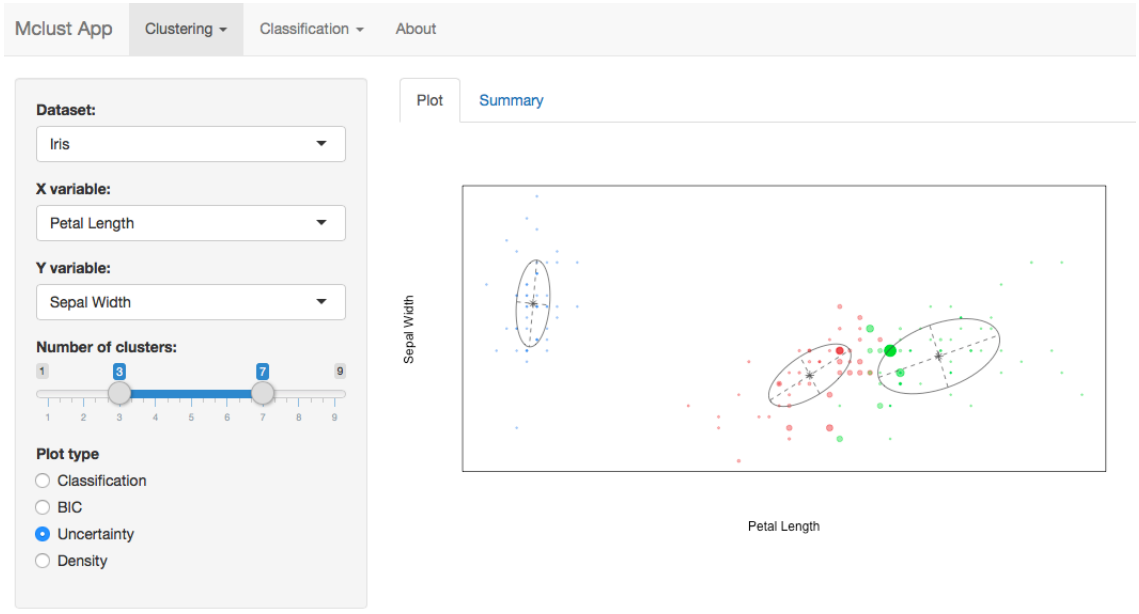


Figure 11: Examining the output plots - Uncertainty.

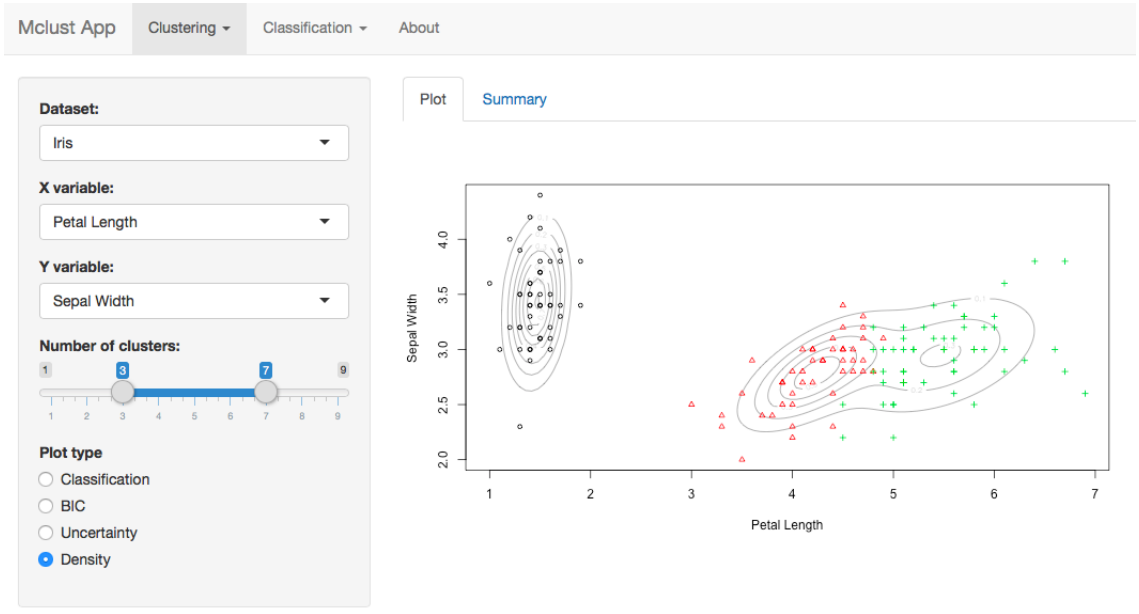


Figure 12: Examining the output plots - Density.

The final part of the output returned by mclust is the summary features. As demonstrated in Figure 13, the summary tab displays how many clusters have been chosen based on the user-defined number of possible clusters to consider (3-7), the type of parameterisation (VEV corresponding to: variable volume, equal shape and variable orientation), the BIC score (-562.5522) corresponding to the chosen parameterisation and number of clusters, and finally the clustering table (50-45-55).

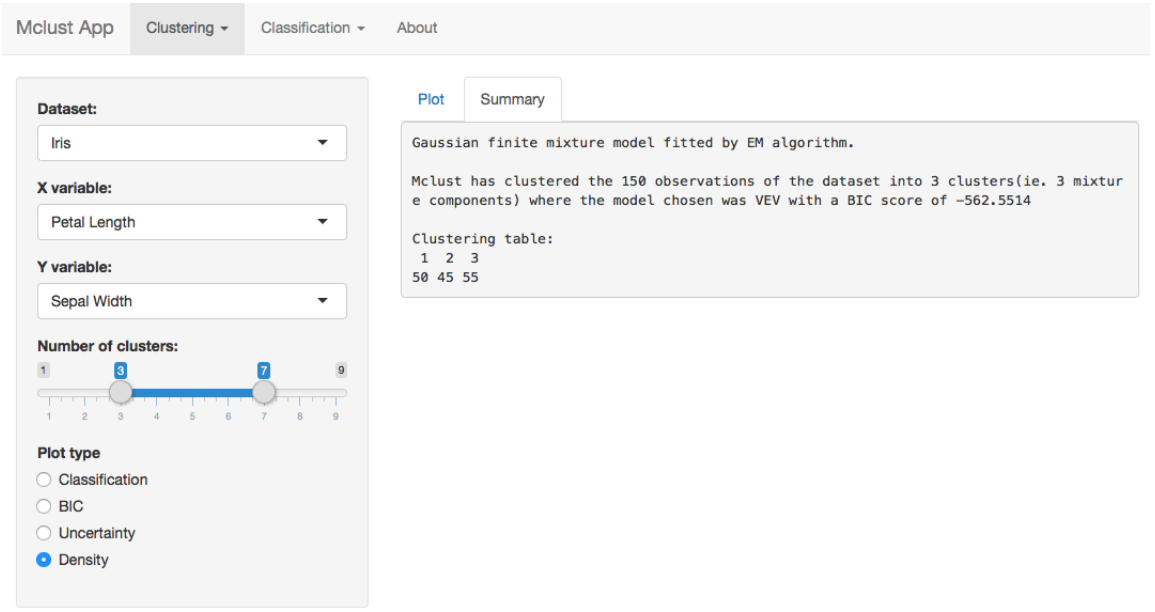


Figure 13: Viewing the text summary output of the selected clustering.