06 - Clustering

Hierarchical, K-Means, Gaussian Mixture Models

SYS 6018 | Fall 2020

06-clustering.pdf

Contents

1 Clustering Intro

1.1 Required R Packages

We will be using the R packages of:

- `tidyverse` for data manipulation and visualization
- `mixtools` for mixture modeling
- `mclust` for model-based clustering
- `MASS` for the crabs data
- `broom` for tidying model output

  Some of the figures in this presentation are taken from "An Introduction to Statistical Learning, with applications in R" (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.

1.2 Clustering

Cluster analysis divides data into groups (clusters) that are meaningful, useful, or both. [ITDM 7]

- If meaningful, then clusters should capture the natural structure of the underlying data generating process.
    - biological taxonomy
- Alternatively, clustering can be useful for summarizing or reducing the size/complexity of the data.
    - market segmentation for targeted marketing

1.2.1 Clustering: Two Views

1. Find homogeneous subgroups
    - Partition data into groups such that objects in the same group are *similar* to each other, while objects in different groups are *dissimilar*
2. Statistical Clustering
    - Partition data into groups such that objects in the same groups are from the same *distribution*

1.2.2 The Field of clustering

There are probably more approaches/algorithms for clustering than any other area of data mining.

- Different criteria on which to base cluster analysis

    - E.g., Cluster plants on color, size, shape, geography

- Different ways to evaluate a clustering solution

- Different goals of clustering: understanding, data reduction, etc.

- The large variety of data types which can be clustered

    - rectangular, network, time series, functional, etc

The three most popular methods (in my opinion) which are also the building blocks of the more complex and targeted methods are:

1. Hierarchical Clustering

2. K-means

3. Model Based Clustering

1.3   Example: *Leptograpsus variegatus*

The *Leptograpsus variegatus*, or purple rock crab, can take on a blue or orange coloring in some parts of Australia.

Campbell, N.A. and Mahon, R.J. (1974) collected 5 morphological measurements on 200 crabs from the species *Leptograpsus variegatus*. These are recorded in the `crabs` dataset in the `MASS` R package.

```
1  crabs = MASS::crabs
```

sp	sex	index	FL	RW	CL	CW	BD
B	M	25	15.0	11.9	32.5	37.2	13.6
O	M	25	16.3	11.6	31.6	34.2	14.5
O	F	20	17.1	14.5	33.1	37.2	14.6
B	F	19	12.0	11.1	25.4	29.2	11.0
B	M	17	13.1	10.6	28.2	32.3	11.0
O	M	19	14.7	11.1	29.0	32.1	13.1

- columns
    - `sp` B=blue, O=orange
    - `sex` M=male, F=female
    - `FL`, `RW`, `CL`, `CW`, `BD` morphological measurements
    - `index` not important for us
    - see `?MASS::crabs` for details

We will construct clustering based on the five morphological features and use the color and sex as the group labels (which we will pretend are unknown).

1.4   Example: Old Faithful

The old faithful geyser in Yellowstone National Park is one of the most regular geysers in the park. The waiting time between eruptions is between 35 and 120 mins.

[Live Streaming Webcam with eruption predictions](#)

Because the nearby Yellowstone Lodge is nice and warm in the winter, and serves good ice cream in the summer, you may be distracted from stepping outside to watch the eruption. Let's see if we can determine

the best time to leave the cozy lodge and go outside to watch the next eruption.

Your Turn #1 : Old Faithful

The `eruptions` column is the *duration of the previous eruption* and the `waiting` column is the *time between eruptions*.

```
1  #-- Load the Old Faithful data
2  X = datasets::faithful
3
4  #-- Scatterplot
5  ggplot(X) + geom_point(aes(eruptions, waiting)) + theme_light()
```
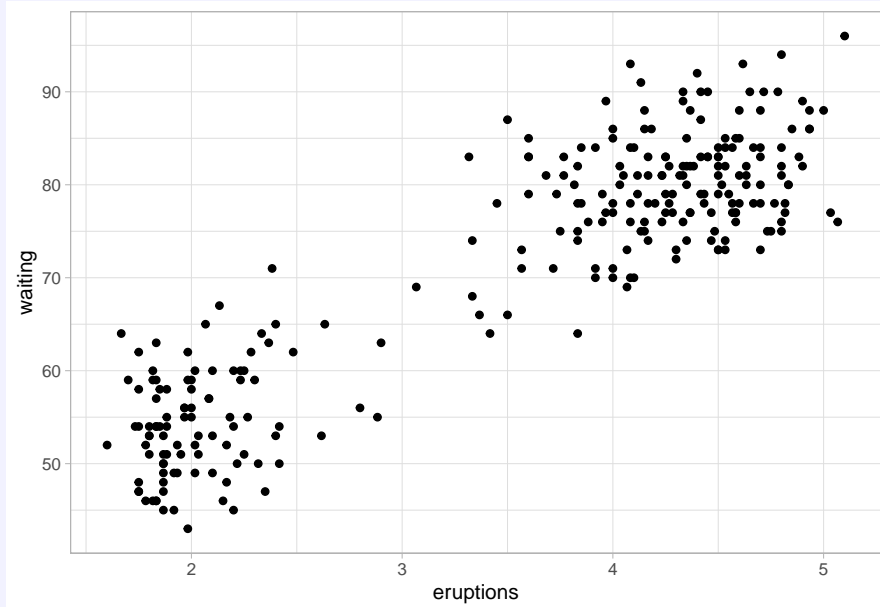

1. What patterns do you see?
2. How many clusters are there? Interpret each cluster.
3. Think about how to estimate this *bivariate* density.

2 Hierarchical Clustering

Hierarchical clustering creates a set of *hierarchical* (or nested) clusters based on a *dissimilarity/distance* metric.

The resulting structure is represented by a dendrogram (*tree-drawing*), which resembles a (usually upside-down) tree.
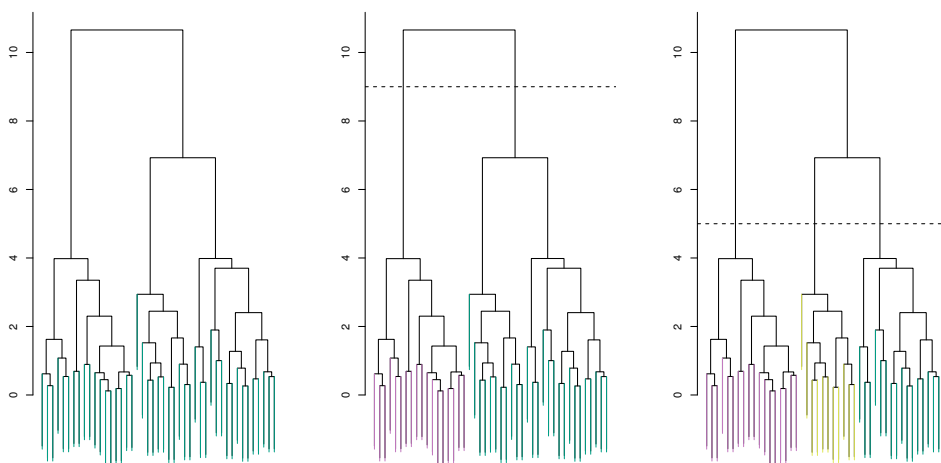
2.1 Dendrogram

Below are 45 observations, in 2D space, from 3 different classes (shown by color).
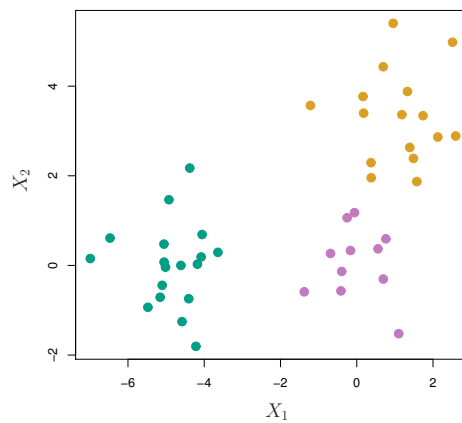
The following dendrogram is constructed from *agglomerative hierarchical clustering*:

- The *pairwise dissimilarity* is defined as the Euclidean distance between points.
- The *cluster dissimilarity* is defined as the largest dissimilarity between two clusters (complete linkage).

Left: Cut at height of 11 ($K = 1$ cluster); Middle: Cut at height of 9 ($K = 2$ clusters); Right: Cut at height of 5 ($K = 3$ clusters).

2.1.1   Dendrogram Interpretation
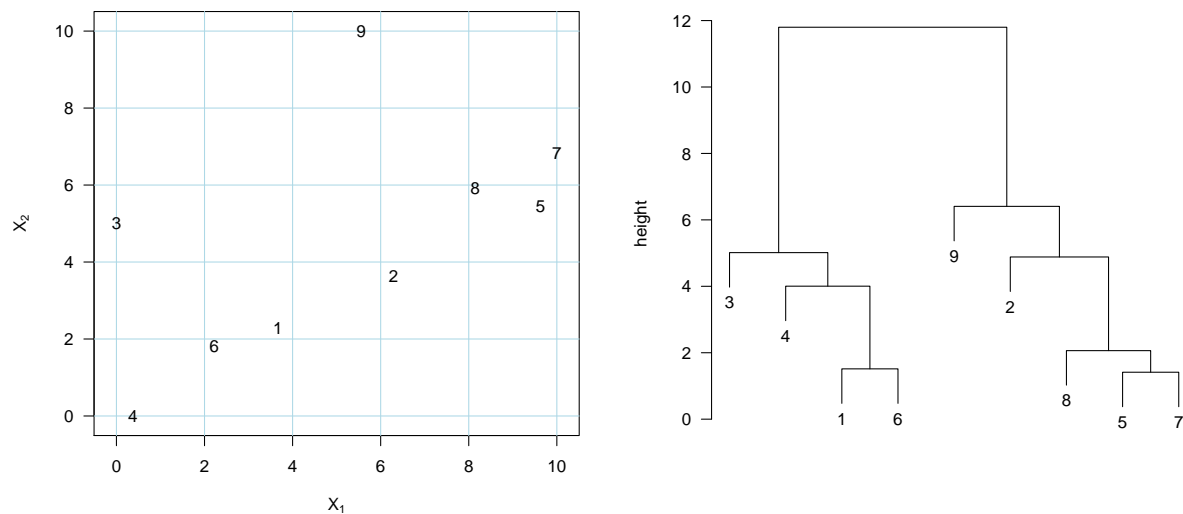
Distance Matrix

```
       1      2      3      4      5      6      7      8
2   2.96
3   4.57   6.44
4   4.00   6.95   5.01
5   6.76   3.80   9.64  10.75
6   1.51   4.46   3.88   2.60   8.25
7   7.80   4.88  10.16  11.80   1.41   9.25
8   5.77   2.93   8.20   9.77   1.55   7.20   2.06
9   7.95   6.41   7.47  11.26   6.10   8.83   5.47   4.84
```

2.2   Agglomerative (Greedy) Hierarchical Clustering Algorithm

The basic hierarchical clustering algorithm takes a *greedy*, *sequential* approach to forming the nested groups.

Algorithm: Agglomerative Hierarchical Clustering

Initialize
1. Begin with n observations and treat each observation as a unique cluster. Set the number of clusters $k = n$.
2. Calculate the *dissimilarity* between all $\binom{n}{2}$ clusters.

Iterate: For $k = n - 1, n - 2, \ldots, 2$:
3. Merge the most *similar* (or least *dissimilar*) clusters.
4. Update the pairwise dissimilarity between the new cluster and all existing clusters.
5. Set $k = k - 1$

2.2.1   Cluster Dissimilarity

There are a few common approaches to calculate the dissimilarity between *clusters* (or sets).

The first three are based on the pairwise dissimilarity/similarity between observations.

- Let A and B be two clusters (collection of points) and d_{ij} the dissimilarity/distance between observations i and j.

Single Linkage/Nearest Neighbor

The cluster dissimilarity is the *smallest* dissimilarity between pairs in the two sets

$$D_{\text{single}}(A, B) = \min\{d_{ij} : i \in A, j \in B\}$$
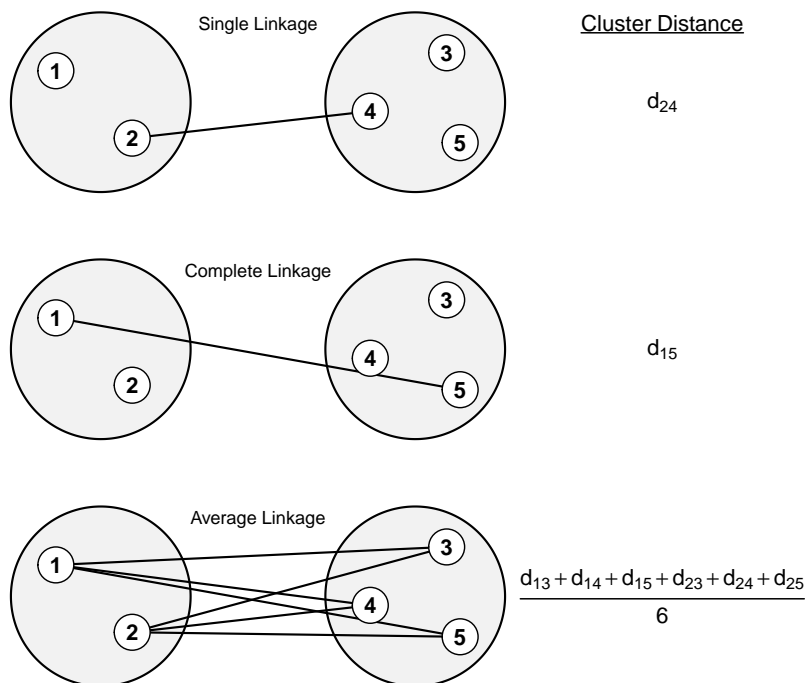
Complete Linkage/Farthest Neighbor

The cluster dissimilarity is the *largest* dissimilarity between pairs in the two sets

$$D_{\text{complete}}(A, B) = \max\{d_{ij} : i \in A, j \in B\}$$

Average Linkage

The cluster dissimilarity is the *average* dissimilarity between pairs in the two sets

$$D_{\text{avg}}(A, B) = \text{avg}\{d_{ij} : i \in A, j \in B\}$$

Another common approach is to base the cluster dissimilarity score on the *centroid* and *compactness* of the observations in the cluster.

Centroid Linkage

The cluster dissimilarity is the *distance between the cluster centroids*.

$$D_{\text{centroid}}(A, B) = \|m_A - m_B\|$$

- m_A is the centroid of set A
- L_2 norm: $\|x\| = \left(\sum_j x_j^2\right)^{1/2}$ (Euclidean distance)

Ward's Linkage

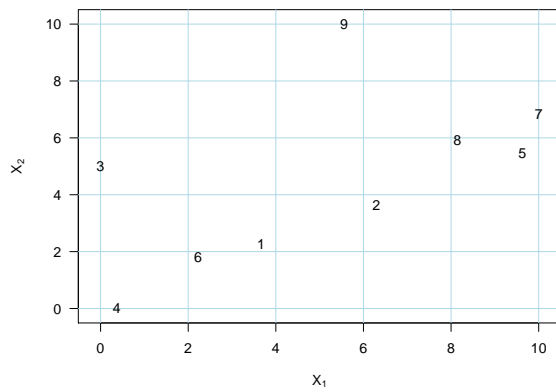The cluster dissimilarity is the *increase in sum of squares* if the clusters were merged.

Let A and B be two clusters. Ward's linkage uses the dissimilarity score

$$
\begin{aligned}
D_{\text{ward}}(A, B) &= \sum_{i \in A \cup B} \|x_i - m_{A \cup B}\|^2 - \sum_{i \in A} \|x_i - m_A\|^2 - \sum_{i \in B} \|x_i - m_B\|^2 \\
&= \frac{n_A n_B}{n_A + n_B} \|m_A - m_B\|^2
\end{aligned}
$$

- $x = (x_1, \ldots, x_p)$
- m_A is the centroid of set A
- L_2 norm: $\|x\| = \left(\sum_j x_j^2\right)^{1/2}$
  - $\|x\|^2$ is the *squared* Euclidean distance
- See (Murtagh and Legendre 2011) for more details and R implementation (`ward.D` and `ward.D2`)
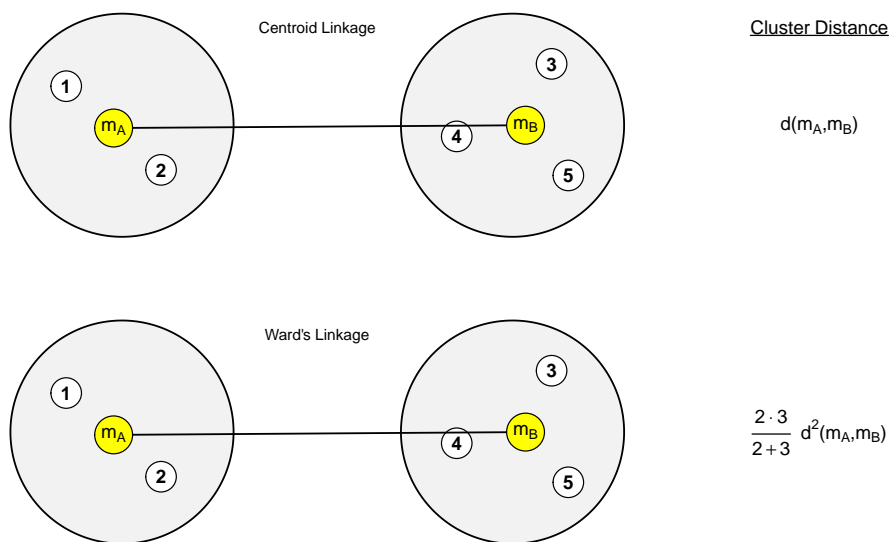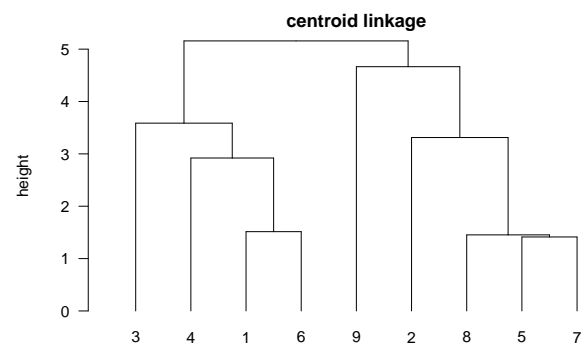
2.2.2   Example

Distance Matrix

	1	2	3	4	5	6	7	8
2	2.96							
3	4.57	6.44						
4	4.00	6.95	5.01					
5	6.76	3.80	9.64	10.75				
6	1.51	4.46	3.88	2.60	8.25			
7	7.80	4.88	10.16	11.80	1.41	9.25		
8	5.77	2.93	8.20	9.77	1.55	7.20	2.06	
9	7.95	6.41	7.47	11.26	6.10	8.83	5.47	4.84

ward.D2 linkage
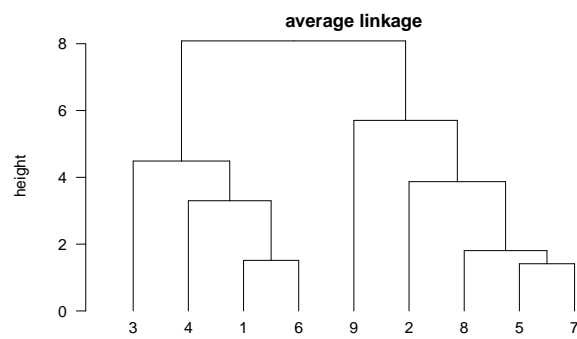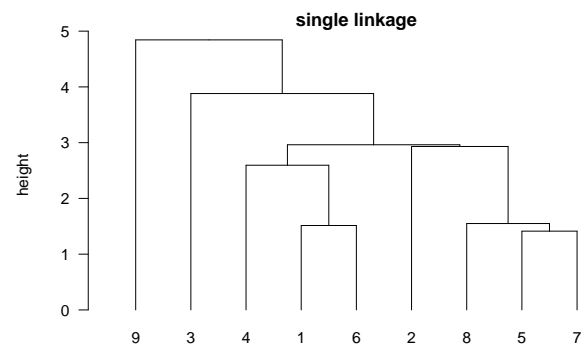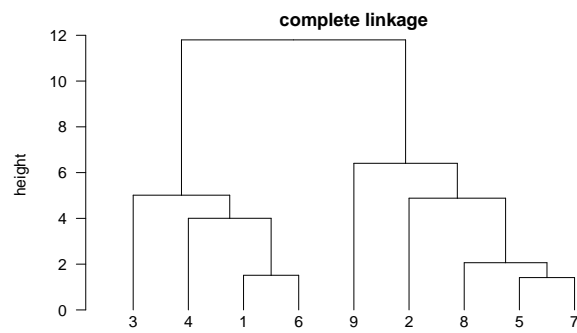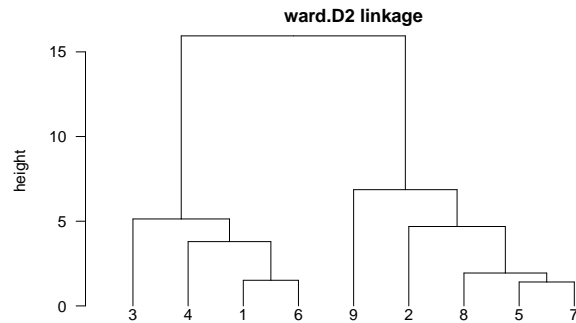
2.2.3   R Implementation

```
1  crabs = MASS::crabs                                    # get crabs data
2  crabsX = dplyr::select(crabs, FL, RW, CL, CW, BD)  # extract features
3  crabsY = paste(crabs$sp, crabs$sex, sep=":")        # get true labels
```

- The R function `hclust()` will run basic Hierarchical Clustering.

- It takes a *distance* object. A distance object is generated by calling `dist()` on a matrix or data frame.

 - This contains all pairwise distances between the rows of your data.
  - Data be numeric data and have no missing values.
  - Several different types of distances are available (e.g., Euclidean, manhattan). Whenever you see distances being used think about scaling!

```
1  dX = dist(crabsX, method="euclidean")   # calculate distance
2  hc = hclust(dX, method="average")        # average linkage
3  plot(hc, las=1, cex=.6)
```
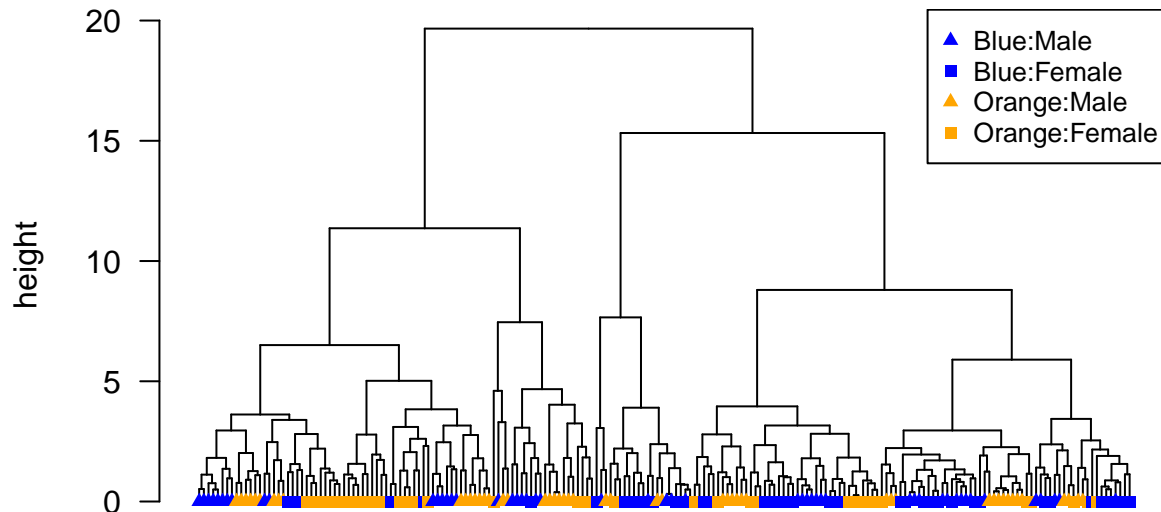
Cluster Dendrogram

dX
hclust (*, "average")
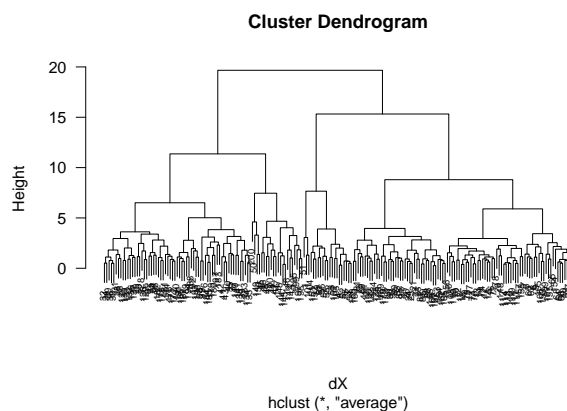
- Some additional visualization is available if `hc` is converted to a `dendrogram` object

```
1  plot(as.dendrogram(hc), las=1, leaflab="none", ylab="height")
2  ord = hc$order
3  labels = crabsY[ord]
4  colors = ifelse(str_detect(labels, "B"), "blue", "orange")
5  shapes = ifelse(str_detect(labels, "M"), 17, 15)
6  n = nrow(crabsX)
```

```
7   points(1:n, rep(0, n), col=colors, pch=shapes, cex=.8)
8   legend("topright", c("Blue:Male", "Blue:Female", "Orange:Male", "Orange:Female"),
9           pch=c(17, 15, 17, 15), col=c("blue", "blue", "orange", "orange"), cex=.8)
```


2.3   Choosing the number of clusters, K

- There are **many** approaches; see ITDM Chapter 7.5 for some approaches.

- One approach is to look for regions in the dendrogram where gaps/changes appear in the height of merges.

  - The height corresponds to the dissimilarity between the merged clusters, so a large jump in height corresponds to a high dissimilarity between the solutions for k and $k + 1$ clusters
  - For Ward's method this corresponds to the change in the Sum of Squared Errors (SSE) for a merge
  - For Complete Linkage, the height corresponds to the dissimilarity between the most dissimilar points in the two clusters

```
1   tibble(height = hc$height, K = row_number(-height)) %>%
2     ggplot(aes(K, height)) +
3     geom_line() +
4     geom_point(aes(color = ifelse(K == 5, "red", "black"))) +
5     scale_color_identity() +
6     coord_cartesian(xlim=c(1, 50))
```
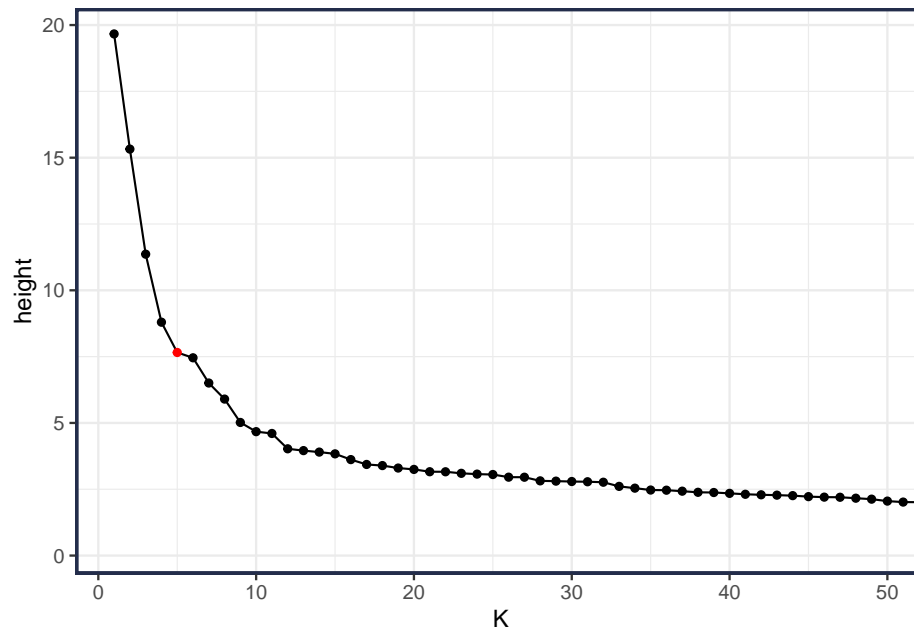
> ### Your Turn #2
>
> 1. Where do you have to cut the dendrogram to get $K = 5$ clusters?
> 2. Since we have labels (crab color/sex), how could we evaluate how well the clustering performed?

- The function `cutree()` will extract the membership vector for a given k clusters or h height.

```
yhat = cutree(hc, k=5)
head(yhat)
1 2 3 4 5 6
1 1 1 1 1 2
```

- Confusion Matrix

```
table(est=yhat, true=crabsY)
     true
est  B:F  B:M  O:F  O:M
  1   10    5    2    4
  2   18   15    6   15
  3   14    7   13    6
  4    7   16   24   16
  5    1    7    5    9
```

2.4 Details and Considerations

- Choice of dissimilarity/distance can be crucial

- Should the variables/features be standardized? E.g.,

  - Scale so all features have mean of zero and standard deviation of one. In R, this is done with `scale()` function.
  - Scale to be between $[0, 1]$
  - Scale by quantile/rank

- Should all the features/variables be used in to calculate the distance? Or more generally, should the features/variables be weighted?

    – See ESL 14.3.3 for an interesting discussion on feature weights

- Other transformations

    – PCA (Principal component analysis). See ISL chapter 10.1.
    – Autoencoders (based on ANNs)

- What type of linkage should be used?

- What value of K to use?

    – There are several, more quantitative, methods to select K. They all make strong assumptions.
    – See Gap Statistic paper for an approach that compares each solution to what is obtained in a *null* model (of no clustering)
    – Could consider a resampling (e.g., bootstrap or cross-validation) approach

- **These can have a substantial impact on your resulting analysis**

- View clustering results with skepticism

    – Results may not have much stability. Vary the data just a little and can get a very different solution
    – Most clustering results that you will see are based on trying many different K, distance/dissimilarity, and linkage methods, to get a pleasing solution
    – this hunting (p-hacking) does not usually lead to repeatable patterns in the data

3   K-means clustering

3.1   Prototype Methods

- Instead of seeking a hierarchical clustering structure, *prototype methods* seek a set of K points $(m_1, \ldots, m_K)$ that best represent the n data points.
    - The prototypes don't have to be existing observations
    - The K prototypes can be thought of as representing K *clusters*
    - Prototype methods can be used for data reduction (see ESL 14.3.9)
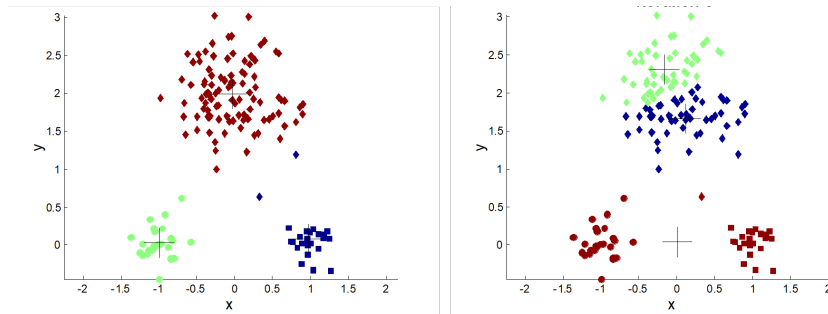
- The prototypes ($\{m_k\}$) should be determined by optimization. The prototypes should *best represent* the data.

- *Best* is, of course, determined by the application

    - For data compression, the choice of K and $\{m_k\}$ is based on the trade-off between fidelity and storage size.
    - For clustering, the choice of prototypes can give us different insights into the data structure

- The following concepts emerge:

    1. Each point should be represented by the prototype that *best represents* it.
    2. The prototypes should be determined so that they *best represent* the points assigned to it.

3.2   K-means

- K-means formalizes these concepts into an algorithm

- In K means, the K cluster *centroids* are the prototypes

    1. The kth centroid represents the n_k observations assigned to that cluster
    2. A point is assigned to the centroid that is *nearest*

Algorithm: K means

Initialize
   1. Choose K initial centroids
      - $\{m_k\}_{k=1}^{K}$

Repeat until convergence:
   2. Assign the observations to the *nearest* centroid (using squared Euclidean distance).

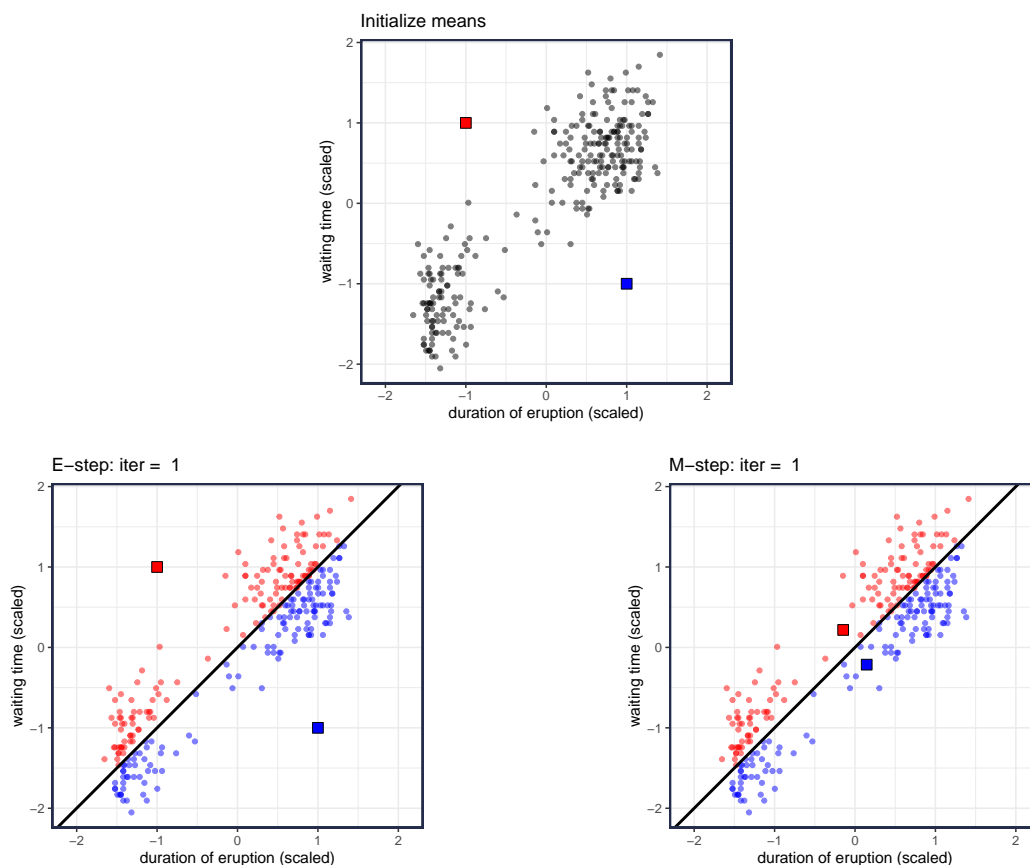$$g_i = \arg\min_{1 \leq k \leq K} \|x_i - m_k\|^2$$

   3. Update the centroids according to the points assigned to them.

$$m_k = \arg\min_{m} \sum_{i: g_i = k} \|x_i - m\|^2$$

- Notation:
  - $x_i \in \mathbf{R}^p$ is the ith observation (point in p dimensional Euclidean space)
  - $m_k \in \mathbf{R}^p$ is the kth prototype
  - $g_i \in 1, 2, \ldots, K$ is the cluster label for observation i
  - The L_2 norm: $\|x - y\| = \left(\sum_{j=1}^{p} (x_j - y_j)^2 \right)^{1/2}$

3.2.1   Example: Old Faithful
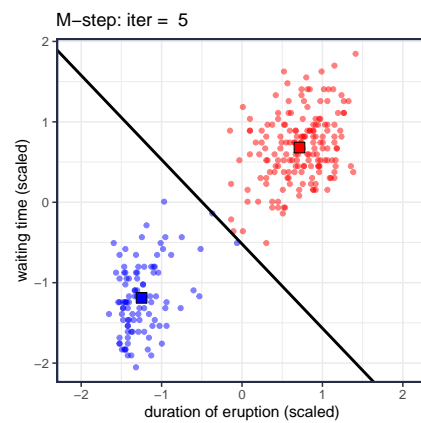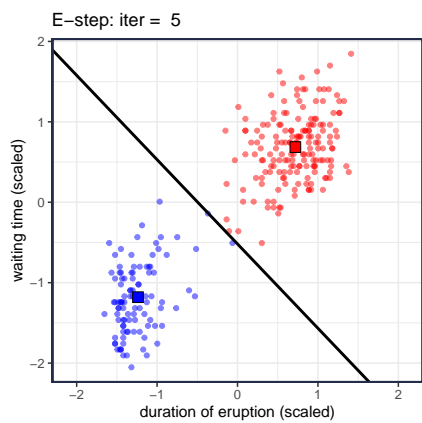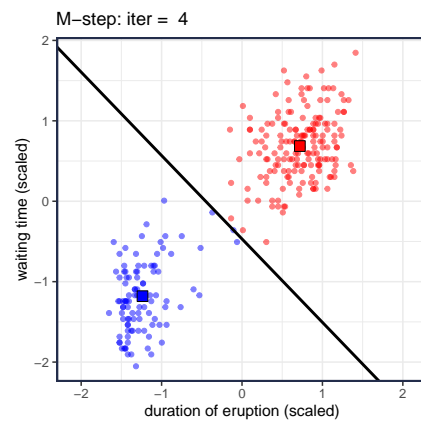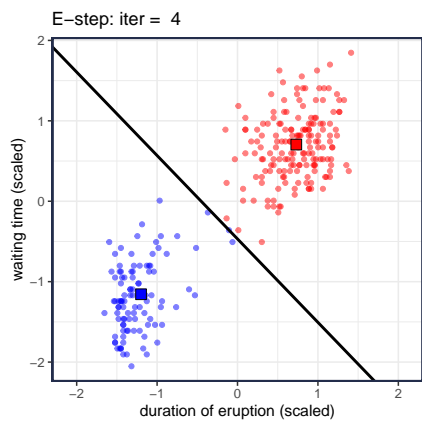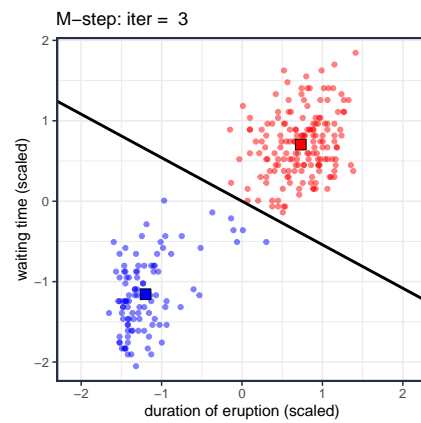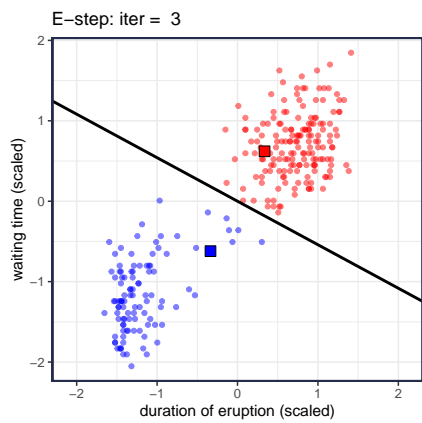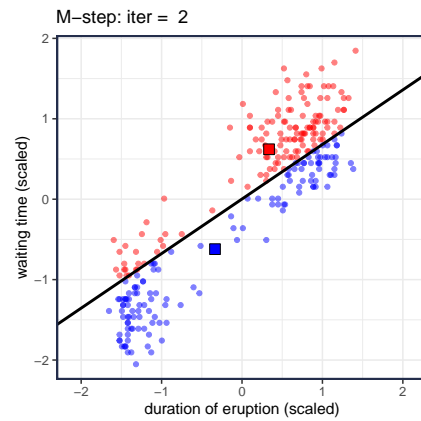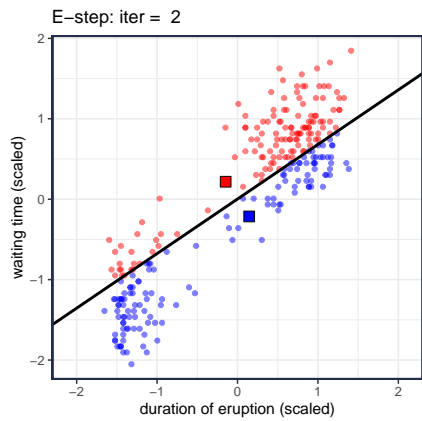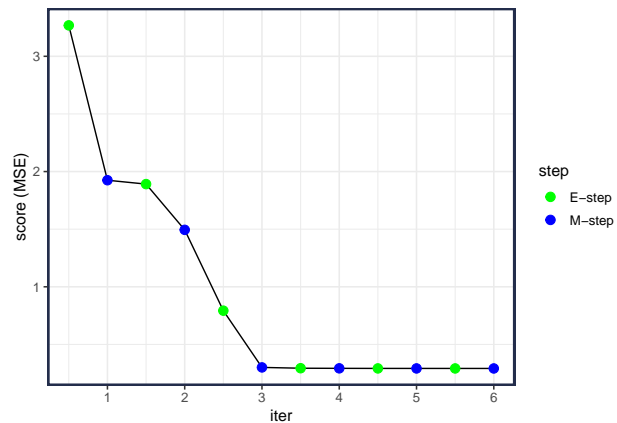
3.3   Initialization

- K-means may only find *local* solutions

- Important to run with several initializations

    - Use the initialization that gives the best performance

- There are some strategies to help

    - initialize with hierarchical clustering
    - sequentially choose prototypes that are farthest away from existing centroids
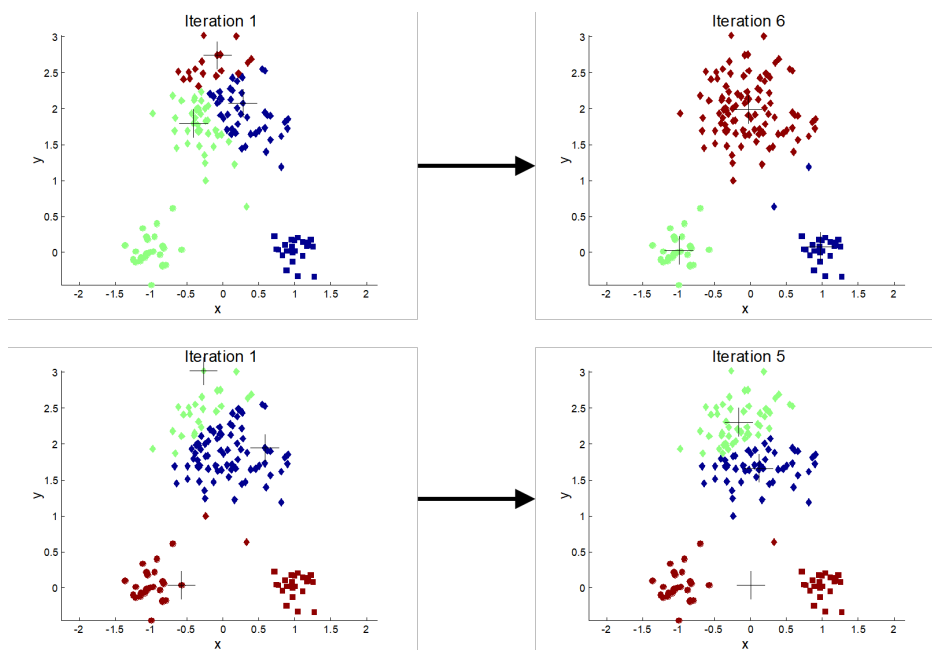
3.4   Choosing K

- Run K-means for several values of K and examine the SSE (sum of squared error); also known as *within cluster scatter* or *within group sum of squares*.
    - Or mean squared error (MSE = SSE/n)

$$\text{SSE} = \sum_{k=1}^{K} \sum_{i:g_i=k} \|x_i - m_k\|^2$$

$$= \sum_{k=1}^{K} \sum_{i=1}^{n} \mathbb{1}(g_i = k) \|x_i - m_k\|^2$$

$$= \sum_{k=1}^{K} W(k)$$

- $W(k)$ is the SSE for cluster k
  - Notice that each step, both E and M, will reduce (technically, will not increase) the SSE

```r
X = scale(crabsX)      # scale crabs data (mean=0, sd=1)

#-- Run kmeans for multiple K
Kmax = 20                                    # maximum K
SSE = numeric(Kmax)                          # initiate SSE vector
for(k in 1:Kmax){
  km = kmeans(X, centers=k, nstart=25)       # use 25 starts
  SSE[k] = km$tot.withinss                   # get SSE
}


#-- Plot results
plot(1:Kmax, SSE, type='o', las=1, xlab="K")
title("K-means for crabs data")
```

K−means for crabs data
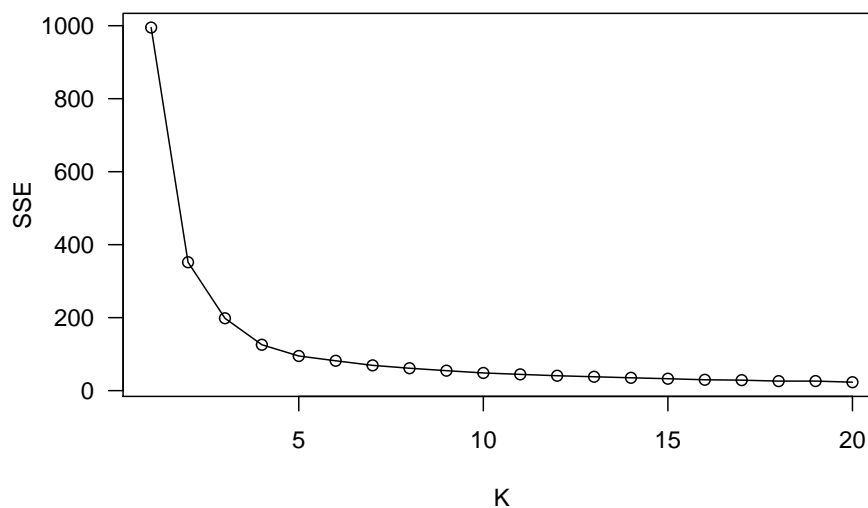
- Same warnings as with hierarchical clustering

Methods for choosing K

1. Look for *elbow* in plot of SSE vs. K

2. Use one of many statistical type tests (e.g. [Hamerly & Elkan, 2003])

3. Instead of K means, use the more flexible Gaussian Mixture Models!

   • Then use AIC/BIC and other statistical measures to select K

3.4.1   Evaluating K-means using known labels

```
#-- Evaluate the classification ability of K-means
km = kmeans(X, centers=5, nstart=25)   # use K=5
table(true=crabsY, est=km$cluster)
      est
true    1   2   3   4   5
  B:F   6  16  12  15   1
  B:M  15  13   7  10   5
  O:F  15   5   2  13  15
  O:M  15  15   4   7   9
```

3.5   K means finds balanced, spherical clusters

   • K means tends to find *balanced* clusters
      – clusters are around the same size (number of observations)
      – there is no mechanism to permit small/large clusters; everything is based on SSE
   • K means tends to find *spherical* clusters
      – because Euclidean distance is used, clusters will be more spherical; larger K is needed to fit
      – **Importance of scaling to treat each variable equivalently** using Euclidean distance
   • Important to scale appropriately. E.g., standardize all columns to have equal variance.
      – In R, the function scale(X) will transform all columns to have mean 0 and variance of 1.
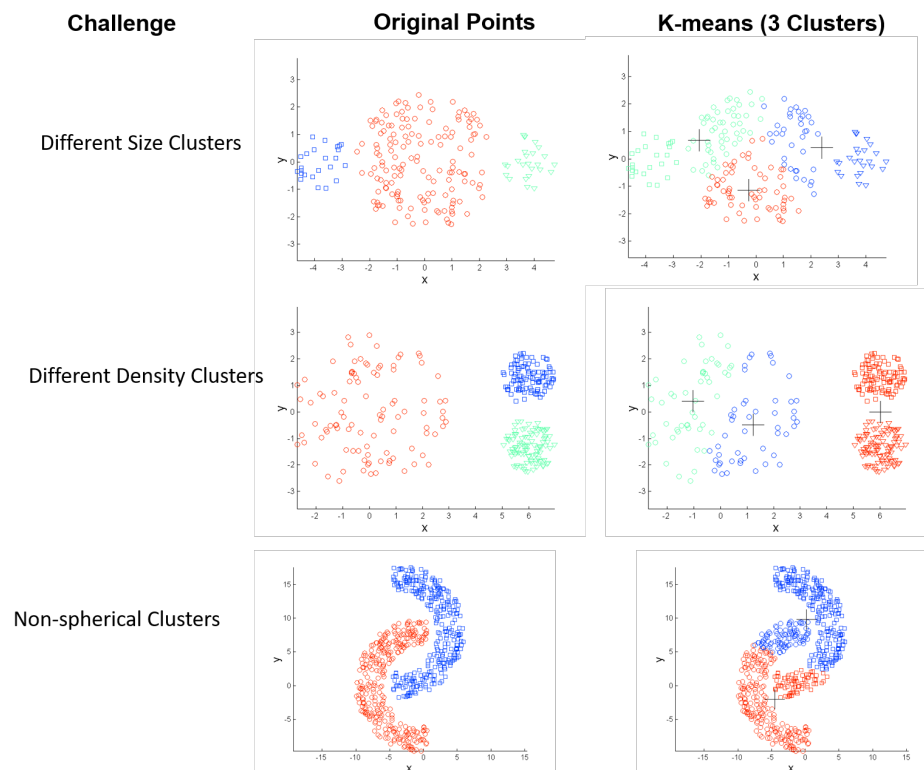
3.5.1 Old Faithful Example
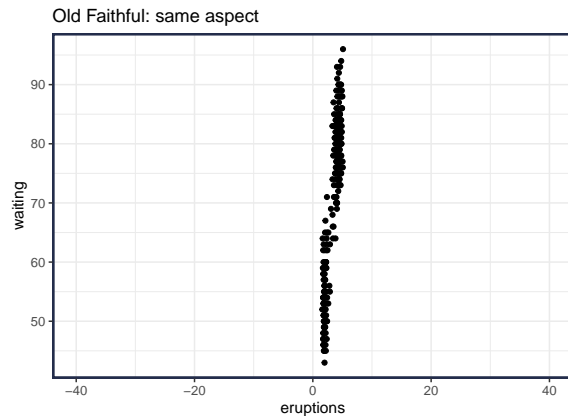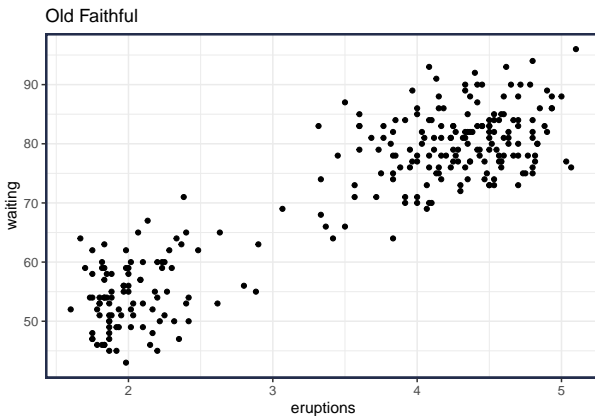
Unscaled Solution
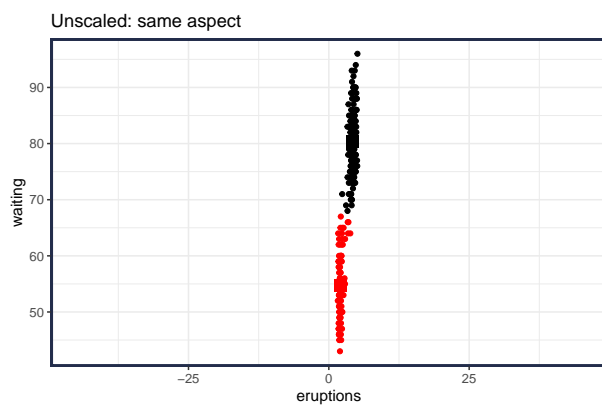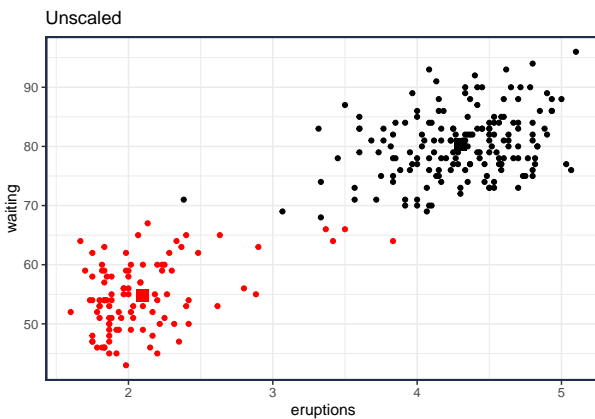
Scaled Solution (mean=0, sd=1)
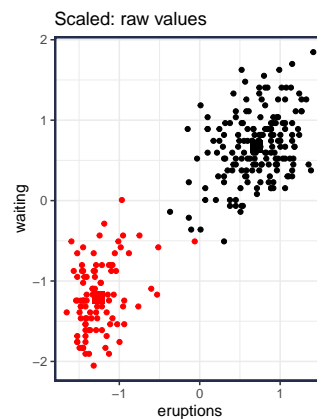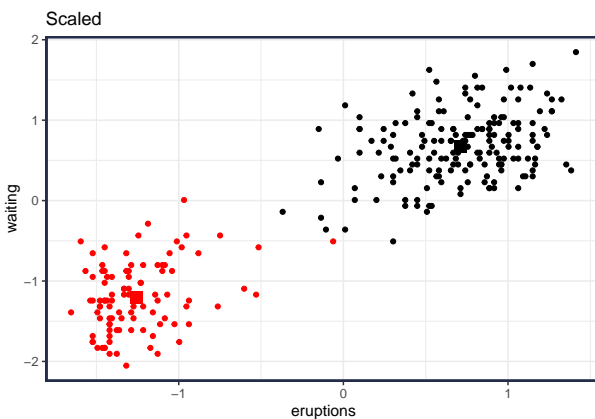
- Notice when we **do not** scale the data, the waiting time dominates the distance calculation
  - Thus, the *eruption* variable has practically no impact on the resulting clustering (i.e., clustering is only based on *waiting time*)
- By scaling the data (so each feature has the same variance), the resulting clustering can better adapt to the natural structure

> • Gaussian Mixture Models provide a natural way to handle input features that have different scales
>   – I.e., **use mixture models instead of k-means**

3.6   R Code for K-means

The R function `kmeans()` implements the K means algorithm.

- Inputs (primary):
    - `x`: numeric matrix or data frame. No missing values allowed. You probably want `x` to be scaled.
    - `centers`: Two options: (1) the number of clusters or (2) a set of initial centers. Default uses (1) and selects random rows of `x` to initiate.
    - `nstart`: (only used if `centers` is a number) the number of times to re-run the algorithm with different initialization. Will return the run with the best performance.
- Outputs (primary):
    - `cluster`: integer representing the cluster label
    - `centers`: the cluster centers
    - `size`: number of points in each cluster
    - performance metrics: `withinss`, `tot.wighinss`, `betweenss`, `totss`

```r
crabsX %>%                    # data
  scale() %>%                 # scale the data
  kmeans(centers = 4, nstart = 100) # run k-means with 4 centers; 100 different initializations
#> K-means clustering with 4 clusters of sizes 41, 71, 28, 60
#>
#> Cluster means:
#>         FL       RW       CL       CW       BD
#> 1   1.3767   1.2778   1.3597   1.3388   1.3855
#> 2   0.3329   0.3512   0.3502   0.3637   0.3143
#> 3  -1.5533  -1.5791  -1.6046  -1.6143  -1.5700
#> 4  -0.6098  -0.5519  -0.5947  -0.5919  -0.5861
#>
#> Clustering vector:
#>    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17   18   19   20
#>    3    3    3    3    3    3    3    3    3    4    4    4    4    4    4    4    4    4    4    4
#>   21   22   23   24   25   26   27   28   29   30   31   32   33   34   35   36   37   38   39   40
#>    4    4    4    4    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
#>   41   42   43   44   45   46   47   48   49   50   51   52   53   54   55   56   57   58   59   60
#>    2    2    2    1    1    1    1    1    1    1    3    3    3    3    3    3    3    3    3    3
#>   61   62   63   64   65   66   67   68   69   70   71   72   73   74   75   76   77   78   79   80
#>    3    3    4    4    4    4    4    4    4    4    4    4    4    4    4    4    4    4    4    4
#>   81   82   83   84   85   86   87   88   89   90   91   92   93   94   95   96   97   98   99  100
#>    4    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    1    1    1
#> 101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120
#>    3    3    3    3    3    4    4    4    4    4    4    4    4    4    4    4    4    4    4    4
#> 121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140
#>    4    4    2    2    2    2    2    2    2    2    2    2    2    2    2    2    1    1    1
#> 141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160
#>    1    1    1    1    1    1    1    1    1    1    3    3    4    4    4    4    4    4    4    4
#> 161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180
#>    4    2    2    2    2    2    2    2    2    2    2    2    2    2    2    1    2    2    2    2
#> 181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200
#>    2    2    2    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
#>
#> Within cluster sum of squares by cluster:
```

```
36   #> [1] 38.09 43.49 16.55 27.58
37   #>  (between_SS / total_SS =  87.4 %)
38   #>
39   #> Available components:
40   #>
41   #> [1] "cluster"     "centers"      "totss"        "withinss"     "tot.withinss"
42   #> [6] "betweenss"    "size"         "iter"         "ifault"
```

The `broom` package provides some nice functions to quickly get relevant info. See the tidy kmeans tutorial for details.

```
1   library(broom) # for tidy(), augment(), glance()
```

```
1   #-- Scale data and implement kmeans
2   scaled_data = scale(crabsX)   # scale data
3   fit = kmeans(scaled_data, centers = 4, nstart = 100)   # run kmeans
```

```
1   #-- augment() adds the cluster labels to data
2   augment(fit, scaled_data)
3   #> # A tibble: 200 x 6
4   #>      FL     RW     CL     CW     BD .cluster
5   #>   <dbl> <dbl> <dbl> <dbl> <dbl> <fct>
6   #> 1 -2.14 -2.35 -2.25 -2.21 -2.05 1
7   #> 2 -1.94 -1.96 -1.97 -1.98 -1.94 1
8   #> 3 -1.83 -1.92 -1.84 -1.78 -1.85 1
9   #> 4 -1.71 -1.88 -1.69 -1.69 -1.70 1
10  #> 5 -1.65 -1.84 -1.66 -1.70 -1.70 1
11  #> 6 -1.37 -1.45 -1.28 -1.26 -1.24 1
12  #> # ... with 194 more rows
```

```
1   #-- tidy() summarizes the info about each cluster
2   tidy(fit)
3   #> # A tibble: 4 x 8
4   #>       FL     RW     CL     CW     BD  size withinss cluster
5   #>    <dbl>  <dbl>  <dbl>  <dbl>  <dbl> <int>    <dbl> <fct>
6   #> 1 -1.55  -1.58  -1.60  -1.61  -1.57    28     16.6 1
7   #> 2  1.38   1.28   1.36   1.34   1.39    41     38.1 2
8   #> 3 -0.610 -0.552 -0.595 -0.592 -0.586   60     27.6 3
9   #> 4  0.333  0.351  0.350  0.364  0.314   71     43.5 4
```

```
1   #-- glance() summarizes the info about the entire model
2   glance(fit)
3   #> # A tibble: 1 x 4
4   #>   totss tot.withinss betweenss  iter
5   #>   <dbl>        <dbl>     <dbl> <int>
6   #> 1   995         126.      869.     2
```

4 Mixture Models

4.1 Example: Old Faithful

The old faithful geyser in Yellowstone National Park is one of the most regular geysers in the park. The waiting time between eruptions is between 35 and 120 mins.

- We used kernel density estimation (KDE) to estimate the density of waiting times in a previous section.
  - See the Shiny App

Density Histograms and Kernel Density Estimation
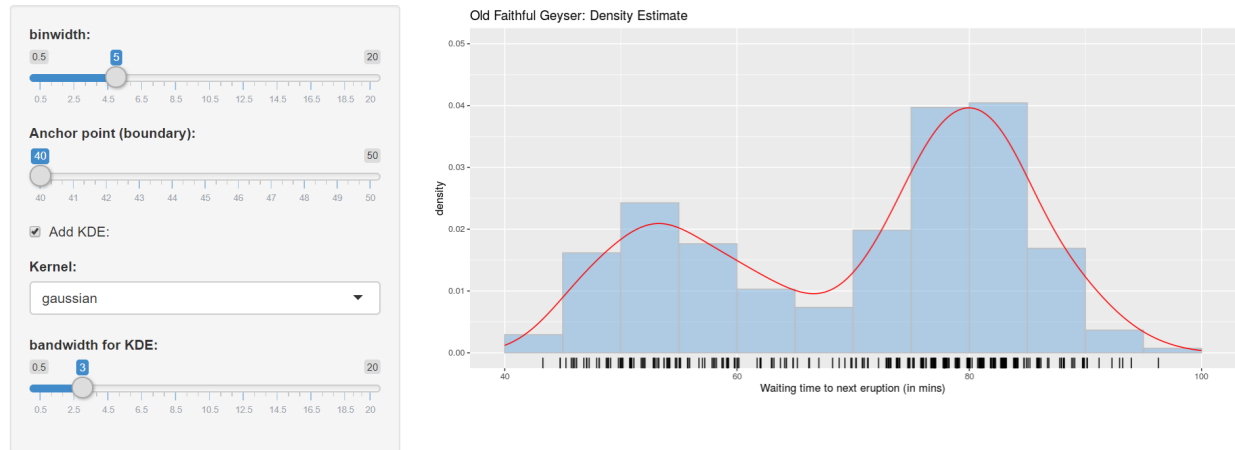
- We used KDE because we couldn't think of any parametric distribution that could capture the bi-modal structure of the waiting time distribution.

- While KDE can be a great approach, another idea is to use a mixture of parametric distributions.

  - These can also capture complex densities (like multiple modes), but also provide a nice intuitive interpretation (i.e., data comes from several sub-populations) which form the basis for clustering

4.2 Finite Mixture Models

Mixture models combine several parametric models to produce a more complex, yet easy to interpret distribution.

- natural representation when data come from K different clusters/groups

$$f(x) = \sum_{k=1}^{K} \pi_k \, f_k(x)$$
$$= \pi_1 \, f_1(x) + \pi_2 \, f_2(x) + \ldots + \pi_K \, f_K(x)$$

- $0 \le \pi_k \le 1$, $\sum_{k=1}^{K} \pi_k = 1$
- $f_k(x)$ is a parametric pdf/pmf

- Think of π_k as the probability that an observation comes from group k
- Think of $f_k(x)$ as the density of group k

> • Usually written $f_k(x) = f(x; \theta_k)$
>   - i.e., all component densities are of the same family, but have different parameter values

4.3 Gaussian Mixture Model (GMM): Univariate

Consider a two-component ($K = 2$) mixture of univariate Gaussian distributions (GMM):

$$f(x; \theta) = \pi f_1(x; \theta_1) + (1 - \pi) f_2(x; \theta_2)$$
$$= \pi \mathcal{N}(x; \mu_1, \sigma_1) + (1 - \pi) \mathcal{N}(x; \mu_2, \sigma_2)$$

- $\mathcal{N}(x; \mu, \sigma) = \frac{1}{\sigma \sqrt{2\pi}} e^{-(x-\mu)^2 / 2\sigma^2}$ is the pdf for a univariate Gaussian distribution
- $0 \le \pi \le 1$
- $\theta = (\pi, \mu_1, \sigma_1, \mu_2, \sigma_2)$
  - There are 5 parameters to estimate

4.3.1 Example: Old Faithful

Your Turn #3

Manually select parameters for the Gaussian Mixture Model. Use the Shiny app to help you decide.

Density Histograms and Kernel Density Estimation
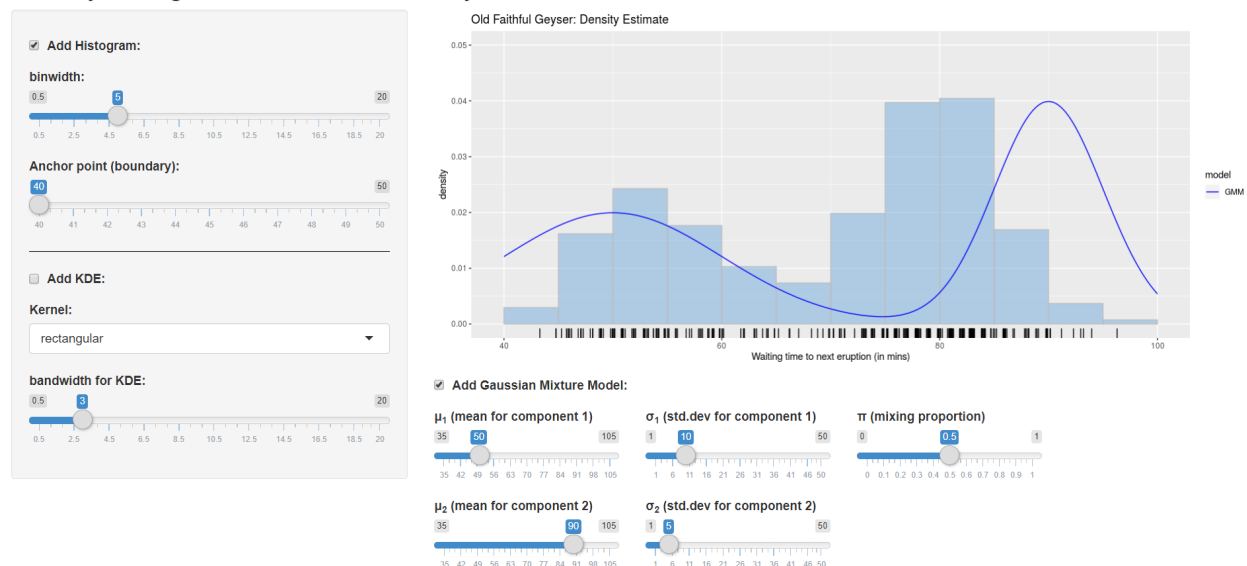
4.4 Simulation (Generative Model)

- Data can be simulated from mixture models in a straightforward manner

> ### Algorithm: Simulate Data from Gaussian Mixture Model (GMM)
>
> To simulate n observations from a K component Gaussian mixture model (with known parameters):
>
> $$f(x;\theta) = \sum_{k=1}^{K} \pi_k f_k(x;\theta_k)$$
> $$= \sum_{k=1}^{K} \pi_k f_k(x;\theta_k) \mathcal{N}(x;\mu_k,\sigma_k)$$
>
> For i in $1, 2, \ldots, n$
> 1. Draw a label from a categorical distribution:
>
> $$g_i \sim Cat(\pi_1, \ldots, \pi_K)$$
>
> 2. Draw the value from a Gaussian, conditional on the group label/parameters
>
> $$x_i \mid g_i{=}k \sim N(\mu_k, \sigma_k)$$

To implement this more efficiently in R, I would do the following:

1. Draw all n labels from a multinomial distribution:

$$g \sim MN(n, \pi_1, \ldots, \pi_K)$$

2. For k in $1, \ldots, K$
    - Draw n_k observations from $N(\mu_k, \sigma_k)$
    - where $n_k = \sum_{i=1}^{n} \mathbb{1}(g_i = k)$

  - See the companion Clustering R code for an example.

> ### Your Turn #4
>
> 1. What is the expected value of n_k?
> 2. What is the expected value of X (i.e., the overall mean)?

4.5   EM Algorithm

The details are found in the assigned readings ESL 8.5.1 and Gaussian Mixture Models: 11.1-11.3, so we will review only the basics.

Notation:

  - Let $g_i \in \{1, 2, \ldots, K\}$ be the (unknown) group/component identifier.

    – π_k is prior probability that any observation is from component k

  - The observed data is $D = \{X_1, X_2, \ldots, X_n\}$

  - The parameters $\theta = (\pi_1, \ldots, \pi_K, \theta_1, \ldots, \theta_K)$

    – note: because we have the constraint that $\sum_k \pi_k = 1$, we only need to estimate $K - 1$ of the π's

- The **responsibility** r_{ik} is the posterior probability that event i came from component k:

$$r_{ik} = \Pr(g_i = k | D, \theta)$$
$$= \frac{P(D|g_i = k, \theta_k)\pi_k}{\sum_{j=1}^{K} P(D|g_i = j, \theta_j)\pi_j}$$

- The responsibilities are weights: $\sum_{k=1}^{K} r_{ik} = 1 \ \forall i$, which represent the probability that events come from the components, conditional on the parameters θ.

- *EM* stands for *Expectation-Maximization*
    - *E-step*: calculate the responsibilities
    - *M-step*: estimate parameters using new responsibilities as weights
    - Iterate until convergence

Algorithm: EM Algorithm

Initialize
  1. Set θ to something reasonable.

Repeat until convergence:
  2. E-step: update r_{ik}, using θ, for $i = 1, 2, \ldots, n$ and $k = 1, \ldots, K$.
  3. M-step: update θ using r_{ik} (maximizing the *expected* log-likelihood)

For Gaussian components:

- Estimate like usual (MLE), except with *weighted* observations:

$$\hat{n}_k = \sum_{i=1}^{n} r_{ik}$$
$$\hat{\pi}_k = \hat{n}_k/n$$
$$\hat{\mu}_k = \frac{1}{\hat{n}_k} \sum_{i=1}^{n} r_{ik} x_i$$
$$\hat{\sigma}_k^2 = \frac{1}{\hat{n}_k} \sum_{i=1}^{n} r_{ik}(x_i - \hat{\mu}_k)^2$$

- The above is specific to univariate Gaussian mixture models.

- The assigned reading gives the mathematical details about what exactly EM is doing (e.g., what the *expected likelihood* is and why it is *maximized*)

4.5.1   Example: Old Faithful
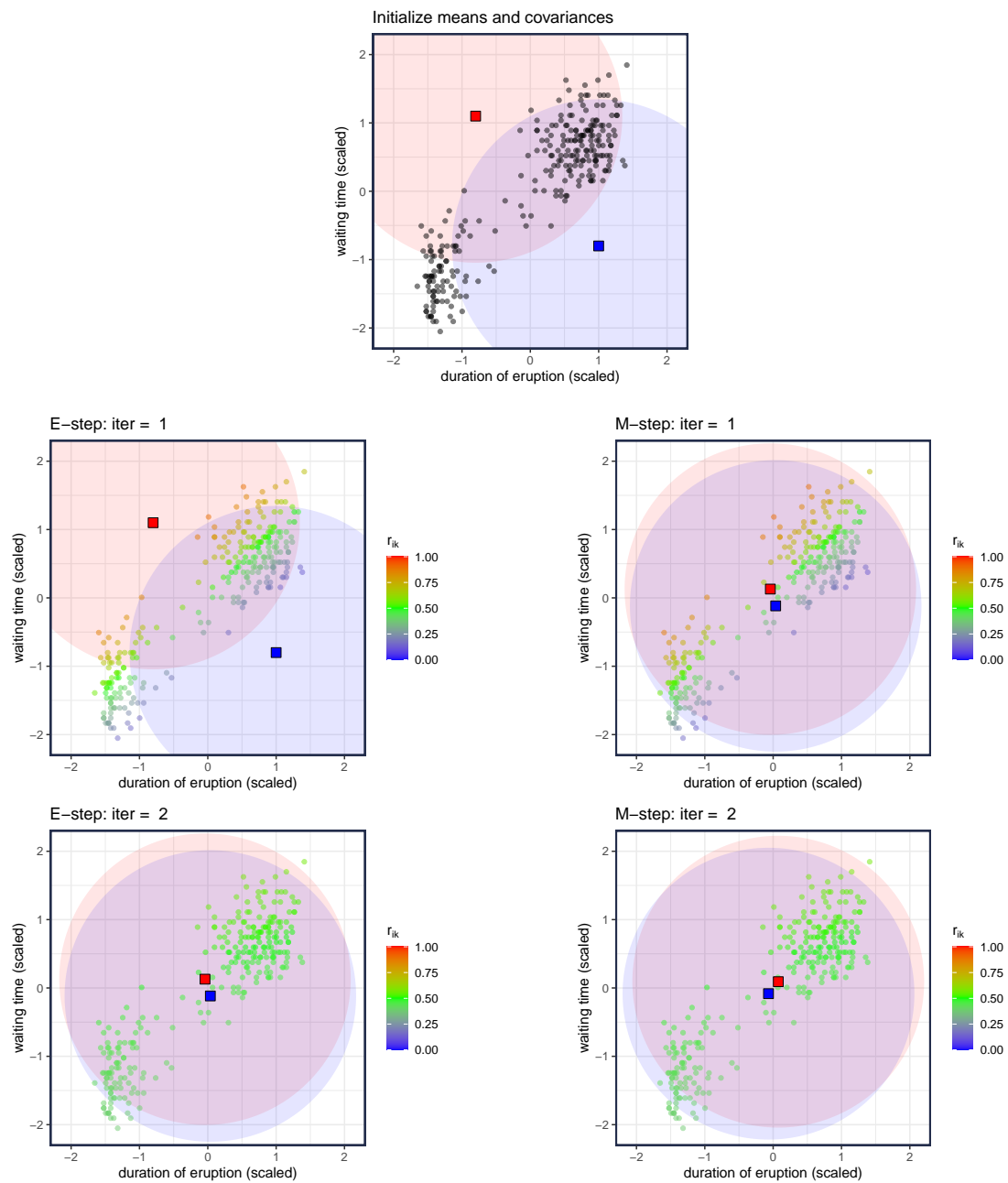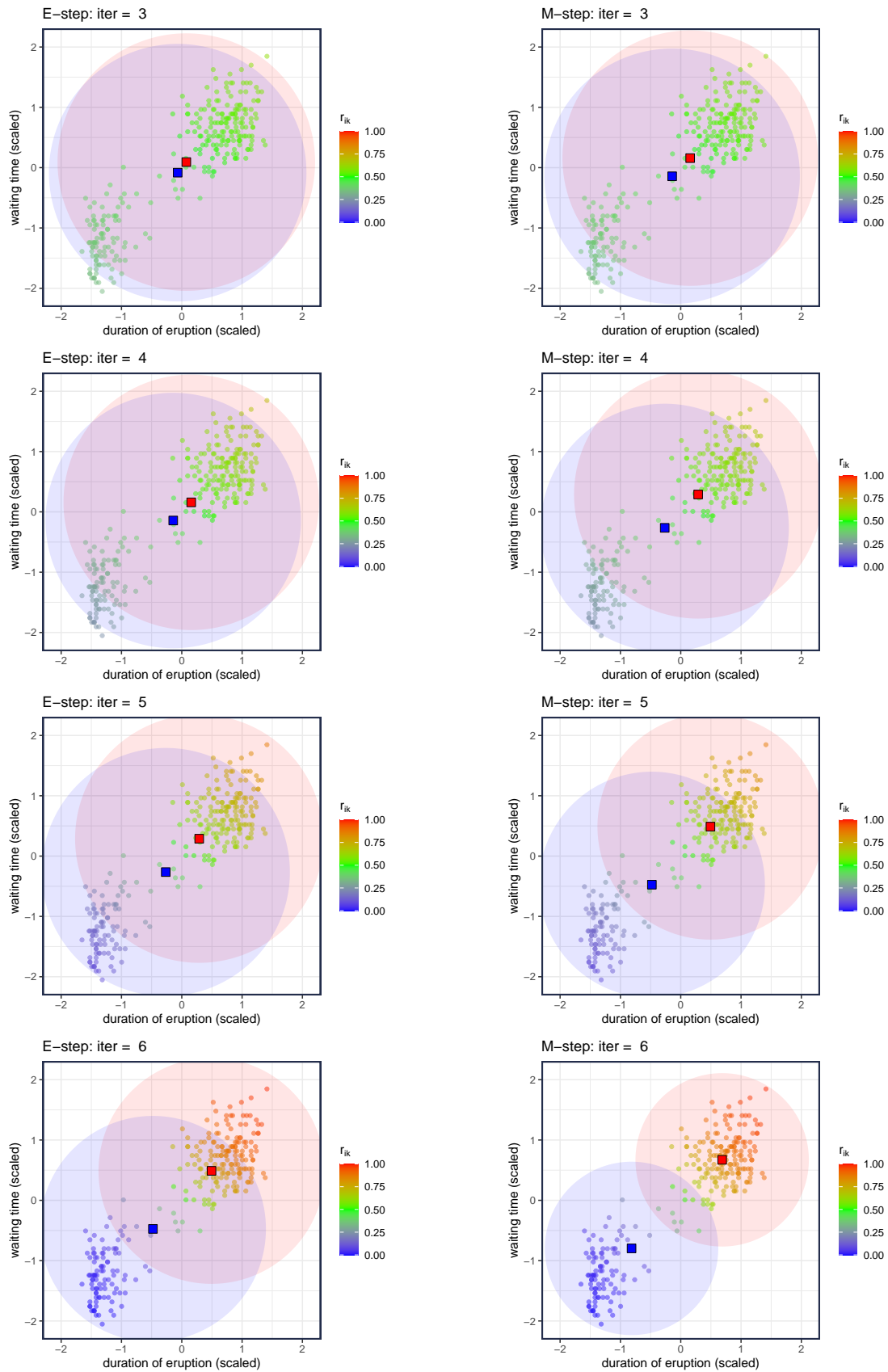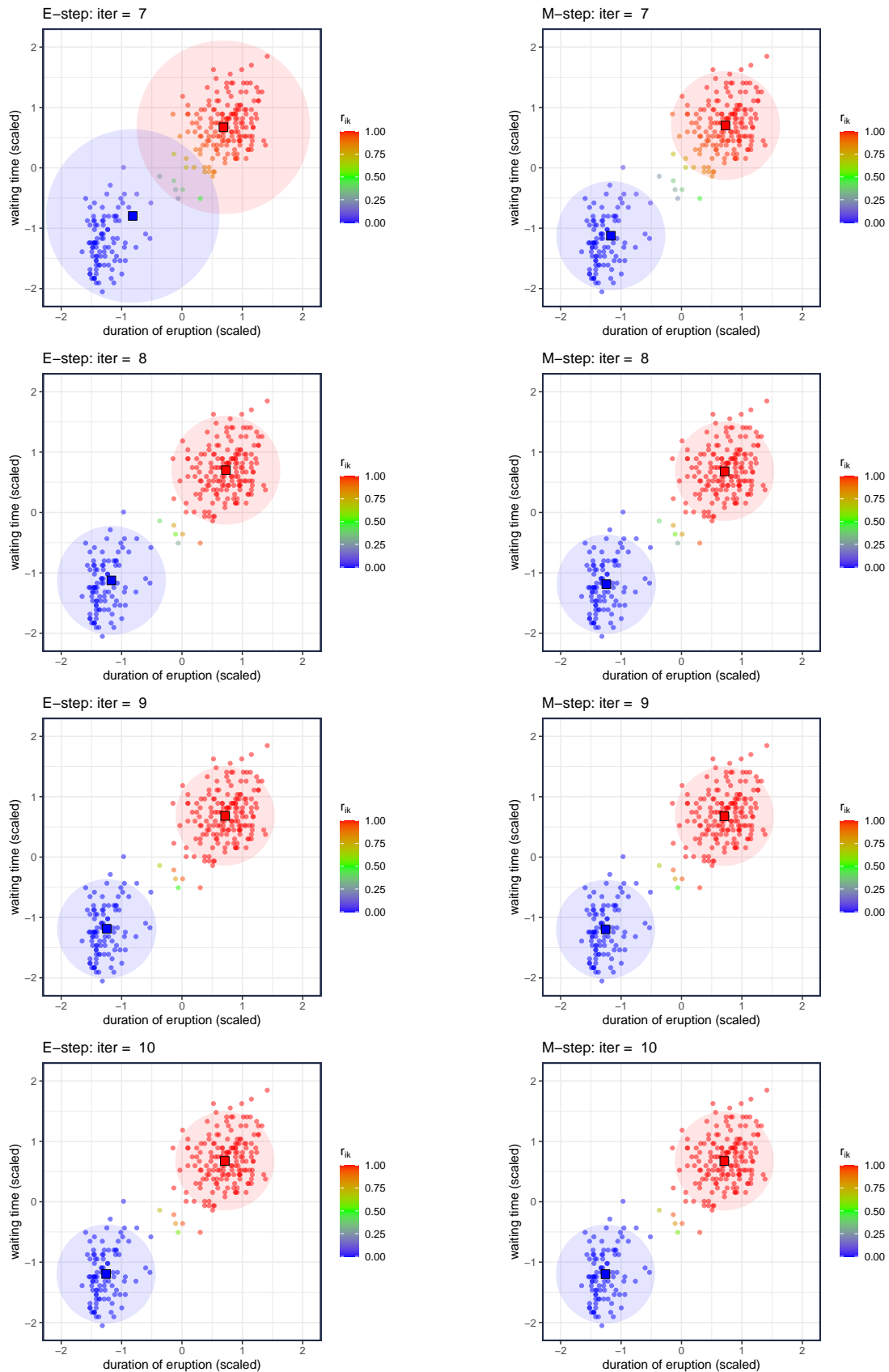
- This example uses a common (spherical, volume) variance-covariance matrix for all components.
  - This is a strong restriction, but makes GMM very close to k-means
  - Allowing more flexibility in the variance-covariance matrix will give more complex models.

4.5.2 Issues in Fitting Mixture Models

- Like in K-means, the EM algorithm may converge to a *local* instead of global solution
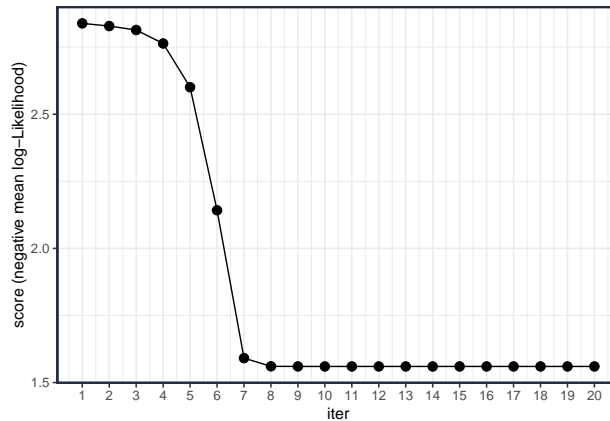 - Run several times with different initializations for θ
- Degenerate distributions
  - It's possible, especially in higher dimensions, to get component variances close to zero (e.g., when very few observations assigned to a cluster)
  - Just ignore these solutions, and run again with different initializations
  - Most software will do this automatically
- Add constraints (e.g., on the variance matrix) will help limit the complexity of the fitted models and can reduce the chances of finding degenerate component distributions

4.6 Multivariate Gaussian Mixture Models (GMM)

A multivariate normal random variable $(\mathbf{X} = X_1, \ldots, X_p)$ has the joint pdf:

$$f(\mathbf{x}; \mu, \Sigma) = \frac{1}{(2\pi)^{p/2}|\Sigma|^{-1/2}} \exp\left[-\frac{1}{2}(\mathbf{x} - \mu)^\mathsf{T} \Sigma^{-1}(\mathbf{x} - \mu)\right]$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} \qquad \mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_p \end{bmatrix}$$

$$
\begin{aligned}
\Sigma &= \begin{bmatrix}
Var(X_1) & Cov(X_1, X_2) & \dots & Cov(X_1, X_p) \\
Cov(X_2, X_1) & Var(X_2) & \dots & Cov(X_2, X_p) \\
\vdots & \vdots & \ddots & \vdots \\
Cov(X_p, X_1) & Cov(X_p, X_2) & \dots & Var(X_p)
\end{bmatrix} \\
&= \begin{bmatrix}
\sigma_1^2 & \sigma_{12} & \dots & \sigma_{1p} \\
\sigma_{21} & \sigma_2^2 & \dots & \sigma_{2p} \\
\vdots & \vdots & \ddots & \vdots \\
\sigma_{p1} & \sigma_{p2} & \dots & \sigma_p^2
\end{bmatrix}
\end{aligned}
$$

- $|\Sigma|$ is the *determinant* of the variance-covariance matrix Σ
    - $\det(\Sigma) = \prod_{j=1}^{p} \lambda_i$ where λ are eigenvalues of Σ
- Σ controls the orientation, shape, and volume of the density contours
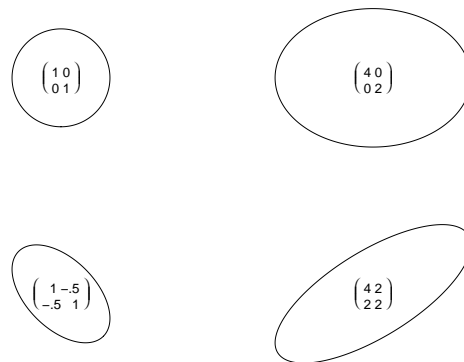
4.6.1 Example: Old Faithful

The R package `mixtools` provides a function `mvnormalmixEM()` to estimate parameters in a Gaussian mixture model using EM.

```r
library(mixtools)

#-- Use old faithful data
X = datasets::faithful

#-- Fit K=2 component mixture model
GMM = mvnormalmixEM(X, k=2)   # use mvnormalmixEM()
#> number of iterations= 18
```

```r
(w = GMM$lambda)                # estimated weights
#> [1] 0.3559 0.6441
```

```r
(mu = GMM$mu)                   # estimate means
#> [[1]]
#> [1]  2.036 54.479
#>
#> [[2]]
#> [1]  4.29 79.97
```

```r
(Sigma = GMM$sigma)             # estimated covariance matrix
#> [[1]]
```
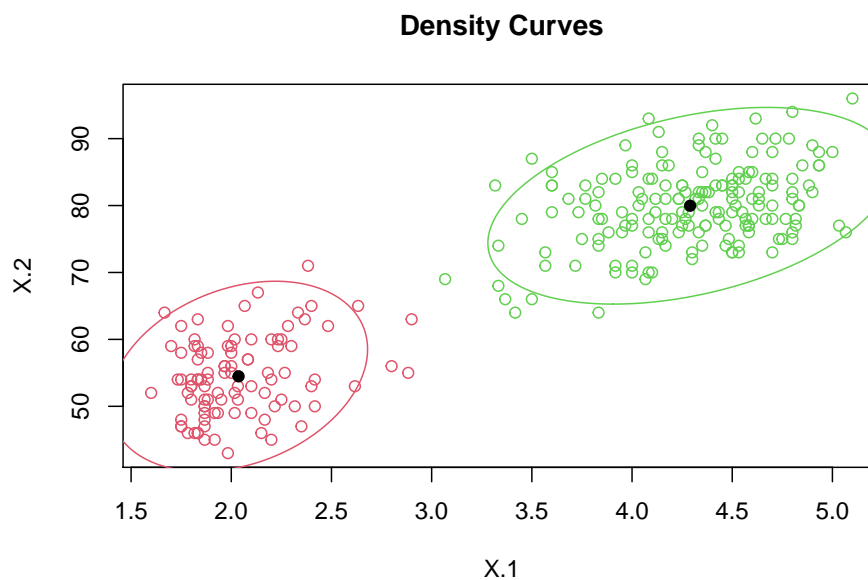
```
#>          [,1]    [,2]
#> [1,] 0.06917  0.4352
#> [2,] 0.43517 33.6973
#>
#> [[2]]
#>          [,1]    [,2]
#> [1,] 0.1700  0.9406
#> [2,] 0.9406 36.0462
```

```
sapply(Sigma, det)              # determinant
#> [1] 2.141 5.242
```

```
#-- Plot 95% contours
plot(GMM, whichplots = 2, alpha=.05) # 95% is 1-alpha
```

Density Curves

4.7   Mixture Models vs. LDA/QDA

Recall in the discussion of Linear/Quadratic Discriminant Analysis (LDA/QDA) each class was modeled by a Gaussian distribution. In LDA every class shared the same covariance matrix while in QDA each class had its own covariance matrix.

- Because the class labels were available, we could directly estimate the parameters.
- LDA is a simpler model (less parameters to estimate) that results in a linear boundary.
- QDA is more complex (higher variance) and thus more likely to over-fit
- We used hold-out/CV to find which model predicts best

For Gaussian mixture models (and model-based clustering) we aren't able to observe the labels, but choose to use the same type of Gaussian distributions to model the data.

- We can also restrict all components to share the same covariance matrix (simple model) or let the covariance matrices be unconstrained (more complex)
- The same bias/variance trade-off exists in this unsupervised setting

5 Model-Based Clustering

- Model-based clustering takes the *statistical view* of clustering:
    - observations are in a group if they are from the same distribution
- Model based clustering can be thought of as an extension of K-means that relaxes the restrictions
    - Can fit data with non-spherical clusters, different sizes, different variances (densities)
    - While Gaussian components are most common, any other types of distributions are possible

---

Algorithm: Model Based Clustering

For k in $1, 2, \ldots, K$:
    a. Fit a mixture model with k components
    b. Record the log-likelihood of the best solution
    c. Choose the k and model parameters that optimize some criterion (e.g., BIC)
Using the best model $(\hat{K}, \hat{\theta})$, the *responsibilities*, $r_{ik}|\hat{\theta}$ allow hard or soft classification of each observation into a cluster.
- E.g., hard classification
$$\hat{g}_i = \arg\max_{1 \leq k \leq K} r_{ik}$$
- E.g., soft classification
$$\Pr(g_i = k|\hat{\theta}) = r_{ik}(\hat{\theta}_k)$$

5.1 Choosing K in Model-Based Clustering

- Back to the old problem of choosing the number of components K.
    - We have a few more options in this statistical setting

1. One approach is the *elbow* method with a plot of log-likelihood and K.
    - the log-likelihood will always increase as K increases
2. Use the *Bayesian Information Criterion (BIC)*.
    - m is the *number of parameters estimated by the model*
    - $\hat{L}$ is the maximum likelihood (i.e., the likelihood using the optimal parameter values)
    - Choose the model with the *lowest* BIC

$$BIC(m) = -2\ln\hat{L} + m\ln(n)$$

- Note: In multivariate settings, m can vary depending on the model constraints (e.g., $\Sigma = \sigma I_p$)

- This penalizes the *complexity* of the model, where complexity is defined by the number of estimated parameters

    - Add constraints in the model will reduce the number of estimated parameters (and hence reduce the model complexity)

3. Use resampling (e.g., cross-validation, bootstrap) along with any loss function

5.2 Example with `mclust`

- The R package `mclust` provides many functions for running Gaussian model based clustering.

NOTE: `mclust` uses an alternative definition of BIC (it uses the *negative* of what is stated above):

$$BIC_{\mathrm{mclust}}(m) = 2\ln \hat{L} - m\ln(n)$$

- So for `mclust` the best model is the one that **maximizes** their BIC

```
# Use the mclust package to help search across all K's
library(mclust)
mix = Mclust(X, verbose=FALSE)    # fit series of models
summary(mix)                       # finds 3 clusters with EEE
#> ----------------------------------------------------
#> Gaussian finite mixture model fitted by EM algorithm
#> ----------------------------------------------------
#>
#> Mclust EEE (ellipsoidal, equal volume, shape and orientation) model with 3
#> components:
#>
#>  log-likelihood   n df   BIC    ICL
#>           -1126 272 11 -2314 -2358
#>
#> Clustering table:
#>    1    2    3
#>   40   97  135
```

```
plot(mix, what="BIC")              # Note: maximize BIC = 2logL - m log n
plot(mix, what="classification")
plot(mix, what="uncertainty")
plot(mix, what="density")
```
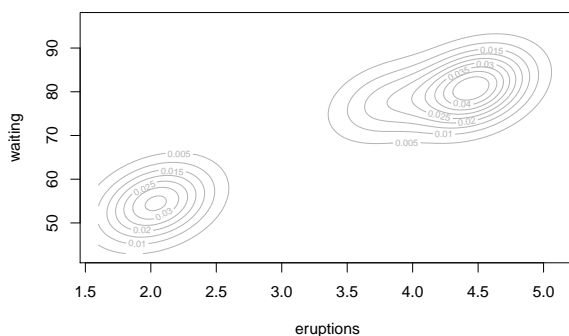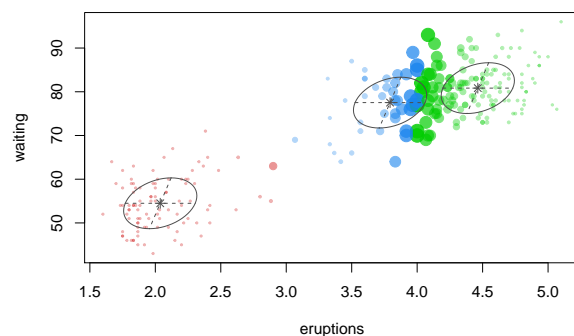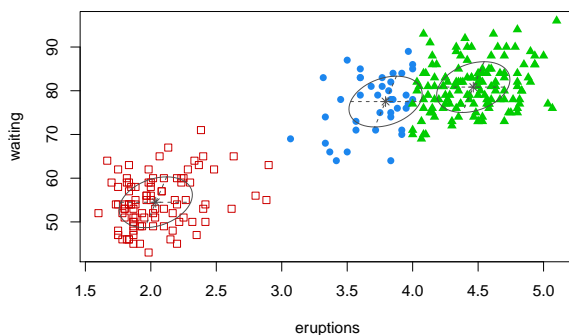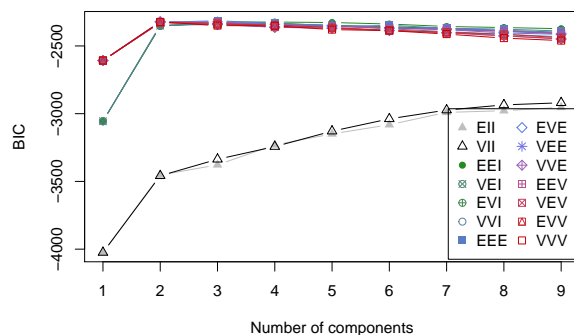
```r
summary(mix, parameters=TRUE)
#> ----------------------------------------------------
#> Gaussian finite mixture model fitted by EM algorithm
#> ----------------------------------------------------
#>
#> Mclust EEE (ellipsoidal, equal volume, shape and orientation) model with 3
#> components:
#>
#>  log-likelihood   n df   BIC    ICL
#>           -1126 272 11 -2314 -2358
#>
#> Clustering table:
#>   1   2   3
#>  40  97 135
#>
#> Mixing probabilities:
#>      1      2      3
#> 0.1657 0.3564 0.4780
#>
#> Means:
#>               [,1]   [,2]   [,3]
#> eruptions   3.793  2.038  4.463
#> waiting    77.521 54.491 80.833
#>
#> Variances:
#> [,,1]
#>           eruptions waiting
#> eruptions   0.07825  0.4802
#> waiting     0.48020 33.7671
#> [,,2]
#>           eruptions waiting
#> eruptions   0.07825  0.4802
#> waiting     0.48020 33.7671
#> [,,3]
#>           eruptions waiting
#> eruptions   0.07825  0.4802
#> waiting     0.48020 33.7671
```

5.2.1 Constraints in `mclust`

The `mclust` package allows many types of constraints (14 to be precise) on the covariance matrices, $\{\Sigma_k\}$.

- This helps find models that have just the right complexity.
- By default, the `Mclust()` function can search over all possible covariance constraints
- Read the mclust 5 paper for more details

Model	Σ_k	Distribution	Volume	Shape	Orientation
EII	λI	Spherical	Equal	Equal	—
VII	$\lambda_k I$	Spherical	Variable	Equal	—
EEI	λA	Diagonal	Equal	Equal	Coordinate axes
VEI	$\lambda_k A$	Diagonal	Variable	Equal	Coordinate axes
EVI	λA_k	Diagonal	Equal	Variable	Coordinate axes
VVI	$\lambda_k A_k$	Diagonal	Variable	Variable	Coordinate axes
EEE	$\lambda D A D^\top$	Ellipsoidal	Equal	Equal	Equal
EVE	$\lambda D A_k D^\top$	Ellipsoidal	Equal	Variable	Equal
VEE	$\lambda_k D A D^\top$	Ellipsoidal	Variable	Equal	Equal
VVE	$\lambda_k D A_k D^\top$	Ellipsoidal	Variable	Variable	Equal
EEV	$\lambda D_k A D_k^\top$	Ellipsoidal	Equal	Equal	Variable
VEV	$\lambda_k D_k A D_k^\top$	Ellipsoidal	Variable	Equal	Variable
EVV	$\lambda D_k A_k D_k^\top$	Ellipsoidal	Equal	Variable	Variable
VVV	$\lambda_k D_k A_k D_k^\top$	Ellipsoidal	Variable	Variable	Variable

Table 3: Parameterisations of the within-group covariance matrix Σ_k for multidimensional data available in the **mclust** package, and the corresponding geometric characteristics.
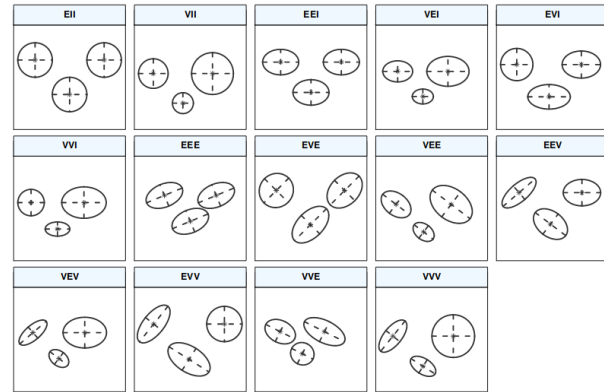
Figure 2: Ellipses of isodensity for each of the 14 Gaussian models obtained by eigen-decomposition in case of three groups in two dimensions.

5.2.2 Tidy Model-based clustering

The `broom` package provides some nice functions to quickly get relevant info. See the tidy Mclust help for details.

```r
library(broom) # for augment(), tidy(), glance()

#-- augment() adds MAP class label and associated probability
augment(mix, X)
#> # A tibble: 272 x 4
#>   eruptions waiting .class .uncertainty
#>       <dbl>   <dbl> <fct>         <dbl>
#> 1       3.6      79 1          2.82e- 2
#> 2       1.8      54 2          8.60e-13
#> 3      3.33      74 1          3.26e- 3
#> 4      2.28      62 2          3.14e- 7
#> 5      4.53      85 3          1.17e- 2
#> 6      2.88      55 2          3.09e- 3
#> # ... with 266 more rows
```

```r
#-- tidy() summarizes each component (but no var-cov matrix yet)
tidy(mix)
#> # A tibble: 3 x 5
#>   component  size proportion mean.eruptions mean.waiting
#>       <int> <int>      <dbl>          <dbl>        <dbl>
#> 1         1    40      0.166           3.79         77.5
#> 2         2    97      0.356           2.04         54.5
#> 3         3   135      0.478           4.46         80.8
```

```r
#-- glance() summarizes the best model
glance(mix)
#> # A tibble: 1 x 7
#>   model     G   BIC logLik    df hypvol  nobs
#>   <chr> <int> <dbl>  <dbl> <dbl>  <dbl> <int>
#> 1 EEE       3 -2314. -1126.    11     NA   272
```

5.3 Gaussian MBC vs. K-means

5.3.1 Univariate Data

Consider first the univariate setting: $x \in \mathbb{R}$

- Recall the updates for K-means
    1. Given the cluster centroids $(\mu_1, \ldots, \mu_K)$, assign cluster label for each event:

$$g_i = \arg\min_{1 \leq k \leq K} \|x - \mu_k\|^2$$

    2. Given the labels, $(g_1, \ldots, g_n)$, update the cluster centroids:

$$\mu_k = \arg\min_m \sum_{i:g_i=k} \|x_i - m\|^2$$

Your Turn #5

1. What are the unknown parameters in K-means?

2. Derive a closed form expression for $\hat{\mu}_k$ (given $\{g_i\}$)

- Consider a GMM (Gaussian Mixture Model) where all components have the same variance.
    1. E-step: update r_{ik}, using θ, for $i = 1, 2, \ldots, n$ and $k = 1, \ldots, K$.
    2. M-step: update θ using r_{ik}

$$n_k = \sum_{i=1}^{n} r_{ik}$$

$$\hat{\pi}_k = n_k/n$$

$$\hat{\mu}_k = \frac{1}{n_k} \sum_{i=1}^{n} r_{ik} x_i$$

$$\hat{\sigma}_k^2 = \frac{1}{n_k} \sum_{i=1}^{n} r_{ik}(x_i - \hat{\mu}_k)^2 \qquad \text{For Multivariate: } \hat{\Sigma}_k = \frac{1}{n_k} \sum_{i=1}^{n} r_{ik}(\mathbf{x}_i - \hat{\mu}_k)(\mathbf{x}_i - \hat{\mu}_k)^\mathsf{T}$$

Your Turn #6

1. What are the unknown parameters in the univariate K component GMM?

2. Estimate the GMM with the constraints:
   a. (Equal size) $\pi_k = 1/K$ for all k
   b. (Equal variance) $\sigma_k = \sigma$ for all k

5.3.2 Multivariate Normal

- K-means is roughly equivalent to the setting:
  - $\pi_k = 1/K$ for all k
  - $\Sigma_k = \sigma I_p$
  - use "hard" assignments

K means

$$\hat{g}_i = \underset{1 \le k \le K}{\arg\min} \; \|X_i - \mu_k\|^2$$

"hard assignment"

$$\hat{r}_{ik} = [0, 0, ..., 1, 0, ..., 0]$$

one-hot encoding

GMM (EM)

$$\hat{r}_{ik} = \frac{\pi_k \, f(x_i | \theta_k)}{\sum_{j=1}^{k} \pi_j \, f(x_i | \theta_j)}$$

"soft assignment"

$$\hat{r}_{ik} = [.01, .04, ..., .12]$$

E-step
given all
model parameters
$\hat{\Theta}$

$$\hat{n}_k = \sum_{i=1}^{n} \mathbb{1}(\hat{g}_i = k)$$

$$\hat{\mu}_k = \frac{1}{\hat{n}_k} \sum_{i=1}^{n} \mathbb{1}(\hat{g}_i = k) X_i$$

$$= \frac{1}{\hat{n}_k} \sum_{i:\hat{g}_i = k} X_i$$

M-step
given assignments
$\{\hat{g}_i\}_{i=1}^{n}$ or $\{\hat{r}_{ik}\}_{i,k}$

$$\hat{n}_k = \sum_{i=1}^{n} \hat{r}_{ik}$$

$$\hat{\pi}_k = \hat{n}_k / n$$

$$\hat{\mu}_k = \frac{1}{\hat{n}_k} \sum_{i=1}^{n} \hat{r}_{ik} X_i$$

$$\hat{\sigma}_k^2 = \frac{1}{\hat{n}_k} \sum_{i=1}^{n} \hat{r}_{ik} (X_i - \mu_k)^2$$

$$\text{MVN}: \hat{\Sigma}_k = \frac{1}{\hat{n}_k} \sum_{i=1}^{n} \hat{r}_{ik} (X_i - \mu_k)(X_i - \mu_k)^T$$

---

constraints: $\pi_k = 1/K \;\; \forall k \;, \; \sigma_k^2 = \sigma^2 \;\; \forall k$

$$\text{M-step} \rightarrow \hat{\sigma}^2 = \sum_{k=1}^{K} \left(\frac{\hat{n}_k}{n}\right) \hat{S}_k$$

$$\hat{S}_k = \frac{1}{\hat{n}_k} \sum_{i=1}^{n} \hat{r}_{ik} (X_i - \hat{\mu}_k)^2$$

$$\text{E-step} \rightarrow \hat{r}_{ik} = \frac{\left(\frac{1}{k}\right) \frac{1}{\hat{\sigma}\sqrt{2\pi}} e^{-\frac{(X_i - \hat{\mu}_k)^2}{2\hat{\sigma}^2}}}{\sum_{j=1}^{k} \left(\frac{1}{k}\right) \frac{1}{\hat{\sigma}\sqrt{2\pi}} e^{-(X_i - \hat{\mu}_j)^2 / 2\hat{\sigma}^2}}$$

$$= \exp\left\{-\frac{1}{2}\left(\frac{X_i - \hat{\mu}_k}{\hat{\sigma}}\right)^2\right\} \Big/ \sum_{j=1}^{k} (\cdot)$$

$$\hat{g}_i = \underset{k}{\arg\min} \, (X_i - \hat{\mu}_k)^2$$

Hard
Assignment

$$\hat{g}_i = \underset{k}{\arg\min} \, \frac{(X_i - \hat{\mu}_k)^2}{\hat{\sigma}^2}$$