04 - Clustering

SYS 4582/6018 | Spring 2019

04-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

```r
library(mixtools)      # install.packages("mixtools")
library(mclust)        # install.packages("mclust")
library(MASS)          # install.packages("MASS")
library(tidyverse)     # dplyr::select() conflict with MASS::select()
```

> 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

We will cover the three most popular methods (in my opinion) which are also the building blocks of the more complex and targeted methods.

1.3 *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.

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

sp	sex	index	FL	RW	CL	CW	BD
O	F	4	12.6	11.5	25.0	28.1	11.5
O	M	42	20.6	14.4	42.8	46.5	19.6
B	F	11	11.0	9.8	22.5	25.7	8.2
O	M	22	15.4	11.1	30.2	33.6	13.5
B	M	10	11.8	10.5	25.2	29.3	10.3
B	M	9	11.8	9.6	24.2	27.8	9.7

- 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

Your Turn #1 : Crab Clustering

What criteria can be used to cluster the crabs?

1.3.1 Process Data

We will construct clustering based on the five morphological features and use the color and sex as the group labels.

```
crabsX = dplyr::select(crabs, FL, RW, CL, CW, BD)
knitr::kable(head(crabsX))
```

FL	RW	CL	CW	BD
8.1	6.7	16.1	19.0	7.0
8.8	7.7	18.1	20.8	7.4
9.2	7.8	19.0	22.4	7.7
9.6	7.9	20.1	23.1	8.2
9.8	8.0	20.3	23.0	8.2

FL	RW	CL	CW	BD
10.8	9.0	23.0	26.5	9.8

```
crabsY = paste(crabs$sp, crabs$sex, sep=":")
head(crabsY)
#> [1] "B:M" "B:M" "B:M" "B:M" "B:M" "B:M"
```

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 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).

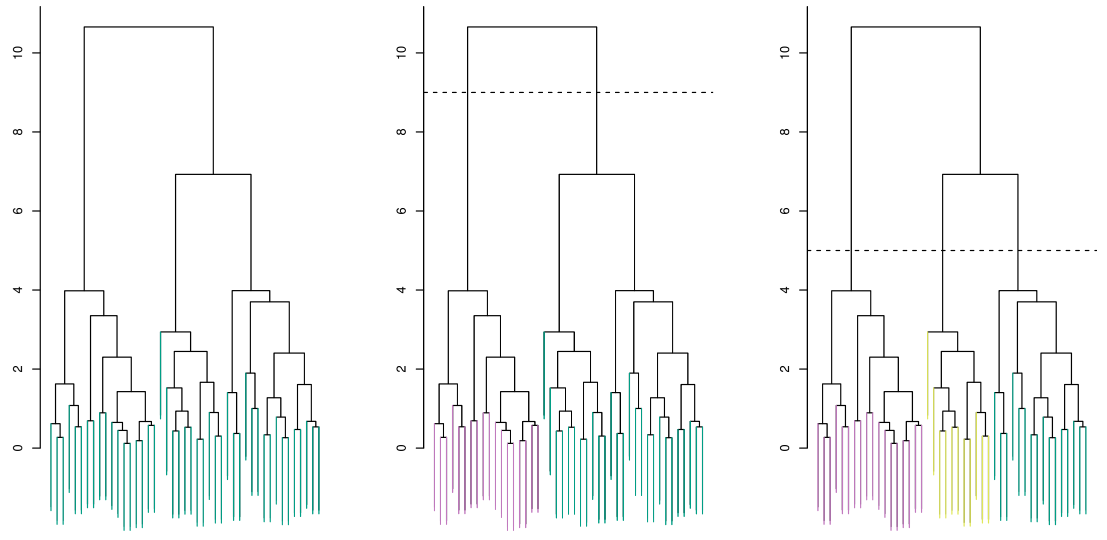
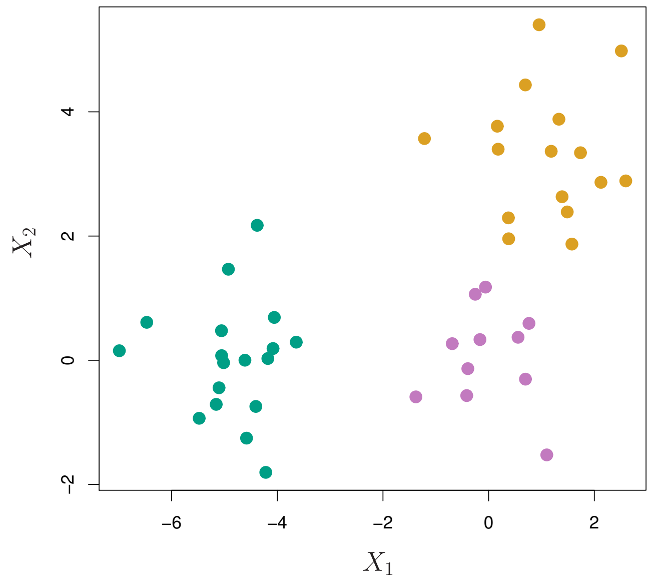
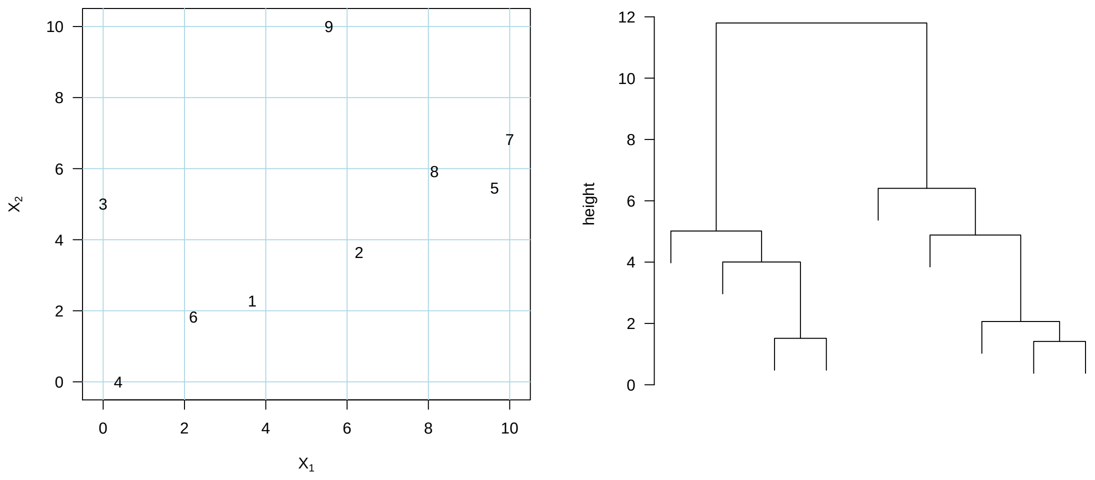


Figure 1: Left: Cut at height of 11 ($K = 1$ cluster); Middle: Cut at height of 9 ($K = 2$ clusters); Right: Cut at height of 5 ($K = 3$ clusters).



2.1.1 Dendrogram Interpretation

```
#> Error in select(., node, height = V3): unused arguments (node, height = V3)
#> Error in text.default(x = 1:9, y = ht - 1.5, labels = (1:9)[hc$order]): object '
```



Distance Matrix

```
#>        1      2      3      4      5      6      7      8
#> 2  2.962
#> 3  4.570  6.442
#> 4  4.004  6.953  5.014
#> 5  6.759  3.797  9.640 10.746
#> 6  1.514  4.461  3.881  2.595  8.252
#> 7  7.796  4.883 10.164 11.799  1.413  9.246
#> 8  5.771  2.932  8.199  9.768  1.550  7.201  2.063
#> 9  7.955  6.407  7.474 11.264  6.104  8.835  5.466  4.843
```

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 $\binom{n}{2}$ observations.

Iterate: For $k = n-1, n-2, \ldots, 2$:

3. Merge the most *similar* clusters.
4. Update the pairwise dissimilarity between the new cluster and all existing clusters.
5. Set $k = k - 1$

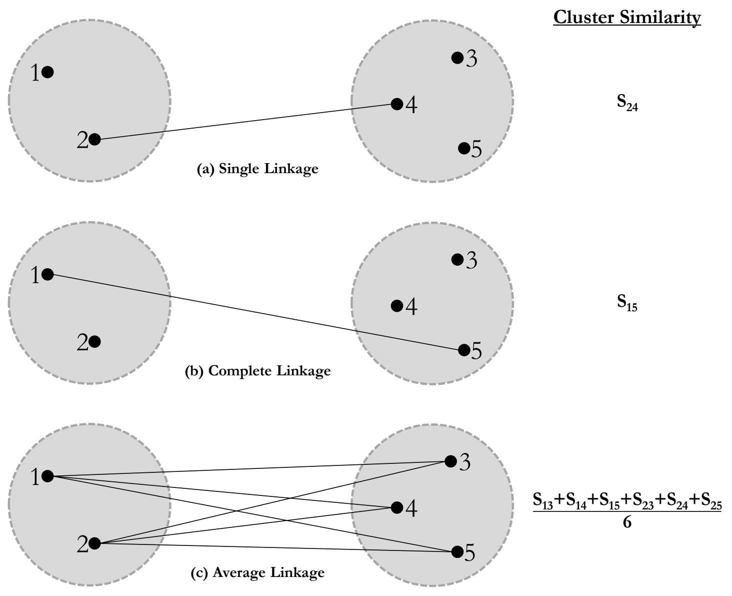
2.2.1 Cluster Dissimilarity

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.

- **Single Linkage/Nearest Neighbor**: The cluster dissimilarity is the *smallest* dissimilarity between pairs in the two sets
- **Copmlete Linkage/Farthest Neighbor**: The cluster dissimilarity is the *largest* dissimilarity between pairs in the two sets
- **Average Linkage**: The cluster dissimilarity is the *average* dissimilarity between pairs in the two sets

```
knitr::include_graphics("figs/clustEX")
```



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

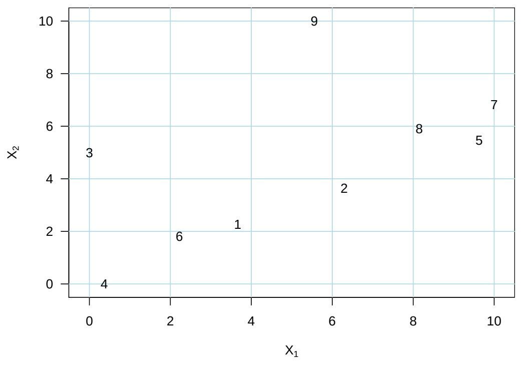
- **Centroid Linkage**: The cluster dissimilarity is the *distance between the cluster centroids*.
- **Ward's Linkage**: The cluster dissimilarity is the *increase in sum of squares* if the clusters were merged.

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

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

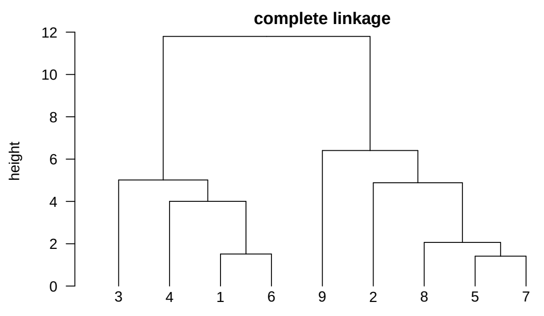
- $x = (x_1, \ldots, x_p)$
- m_A is the centroid of set A
- L_2 norm: $\|x\| = \left(\sum_j x_j^2\right)^{1/2}$
- See (Murtagh and Legendre 2011) for more details and R implementation (`ward.D` and `ward.D2`)

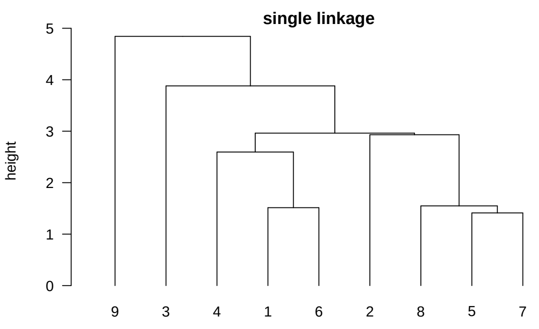
2.2.2 Example

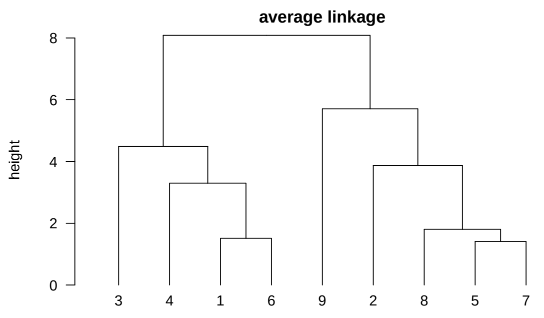


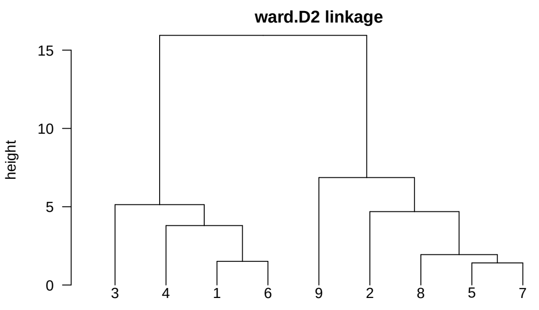
Distance Matrix

```
#>      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
```







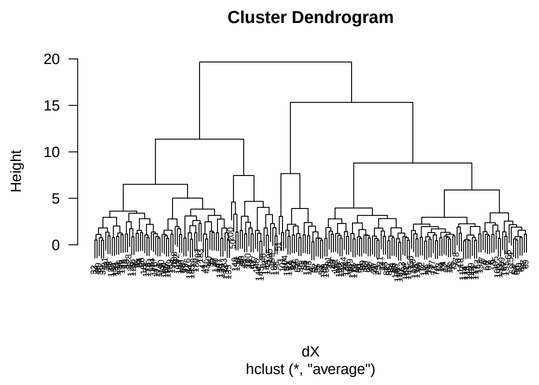


2.2.3 R Implementation

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

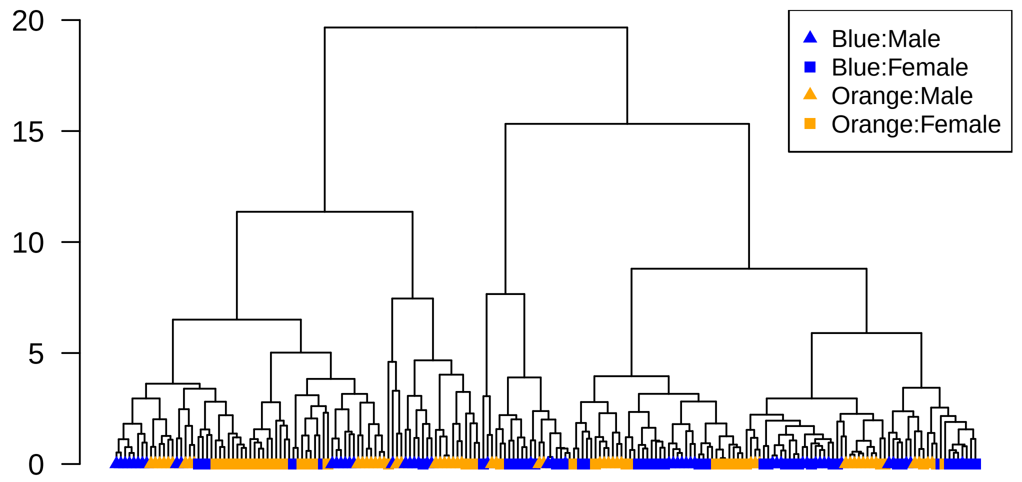
- It takes a *distance* object. This is obtained by running `dist()`.

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



- Some additional visualization is available if `hc` is converted to a `dendrogram` object

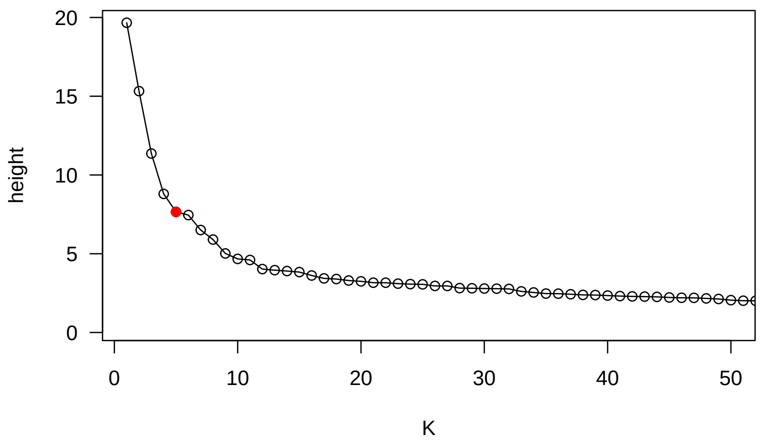
```
plot(as.dendrogram(hc), las=1, leaflab="none")
ord = hc$order
labels = crabsY[ord]
colors = ifelse(str_detect(labels, "B"), "blue", "orange")
shapes = ifelse(str_detect(labels, "M"), 17, 15)
n = nrow(crabsX)
points(1:n, rep(0, n), col=colors, pch=shapes, cex=.8)
legend("topright", c("Blue:Male", "Blue:Female", "Orange:Male", "Orange:Female"),
       pch=c(17, 15, 17, 15), col=c("blue", "blue", "orange", "orange"), cex=.8)
```



2.3 Choosing the number of clusters, K

- There are several approaches; see ITDM Chapter 7.5
- One approach is to look for regions in the dendrogram where gaps/changes appear in the height of merges.
  - For Ward's method this corresponds to the change is SSE, which is somewhat reasonable

```
n=length(hc$height)      # get number of merges
plot(n:1, hc$height, type='o', xlab="K", ylab="height", las=1,
     xlim=c(1, 50))
points(5, hc$height[n-4], col="red", pch=19) # K=5
```



Your Turn #2

1. Where do you have to cut the dendrogram to get $K = 5$ clusters?
2. Since we have labels, 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.,

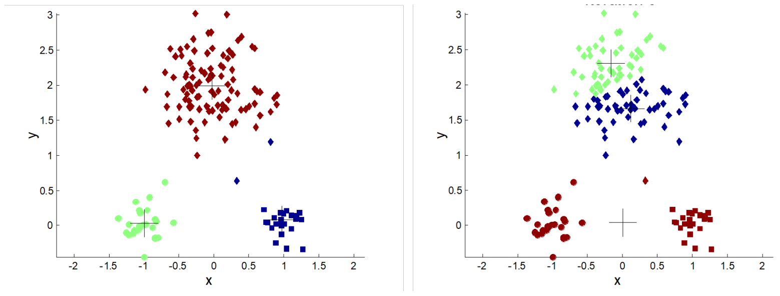
  - Scale so all features have mean of zero and standard deviation of one. In R, this is done with `scale()` function.
  - Scale to be between $[0, 1]$
  - Scale by quantile.

- Should all the features/variables be used in to calculate the distance?
- Other transformations
  - PCA (Principal component analysis). See ISL chapter 10.1.
- 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 hierarchcical clustering structure, *prototype methods* seek a set of K points $(m_1, \dots, m_K)$ 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 tradeoff 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.

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

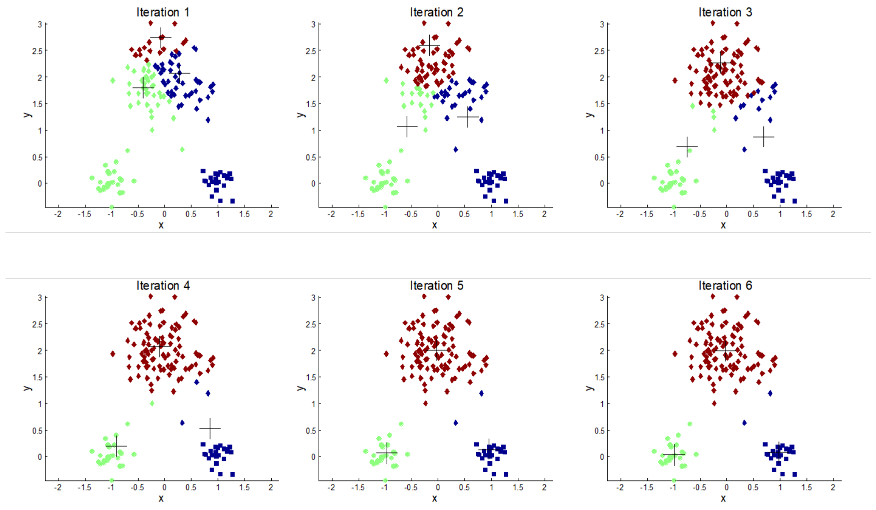
3. Update the centroids.

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

- Notation:

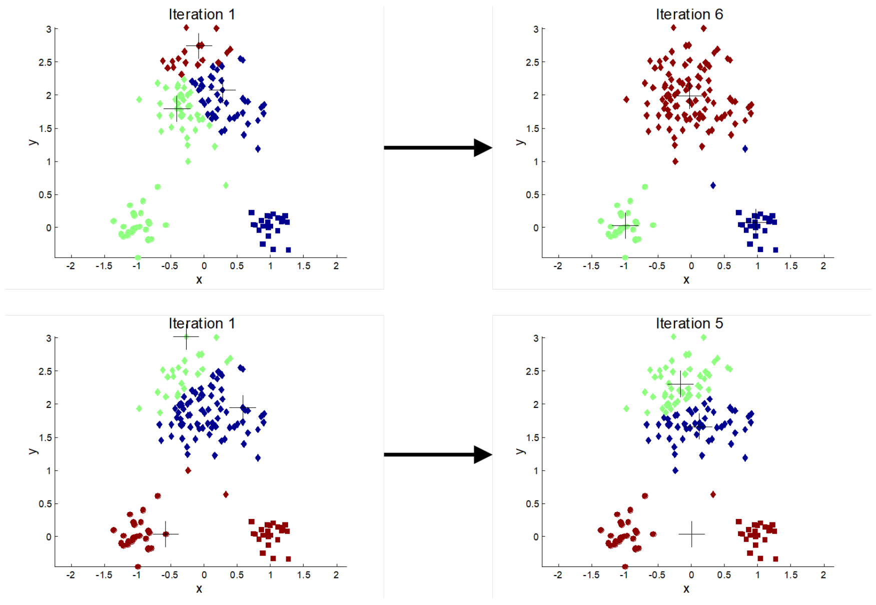
- $x_i \in \mathbf{R}^p$ is the ith observation (point in p dimensional Euclidean space)
- $m_k \in \mathbf{R}^p$ is the k^{th} prototype
- The L_2 norm: $\|x - y\| = \left(\sum_{j=1}^{p}(x_j - y_j)^2\right)^{1/2}$

3.2.1 Example



3.3 Initialization

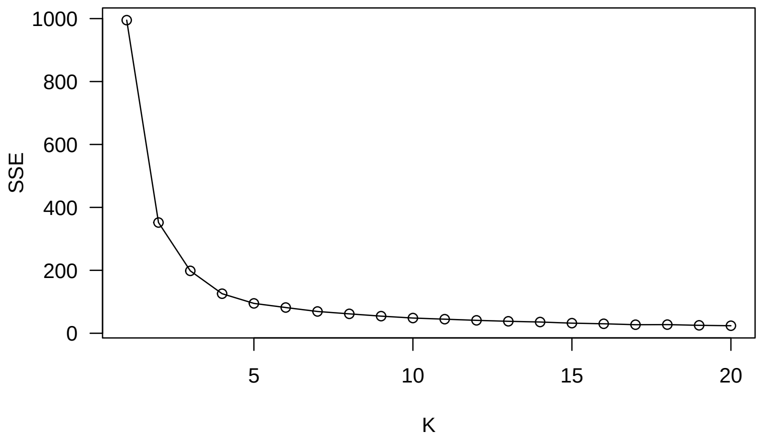
- K-means may only find *local* solutions
- Important to run with several initializations
- There are some strategies to help
  - initialize with hierarchical clustering
  - sequentially choose prototypes that are farthest away form existing centroids



3.4 Choosing K

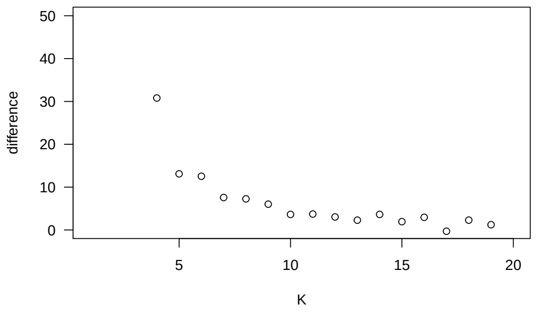
- Run K-means for several values of K and examine the SSE (sum of squared error); also known as *within cluster scatter*.

```
X = scale(crabsX)     # scale crabs data

#-- 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")
```

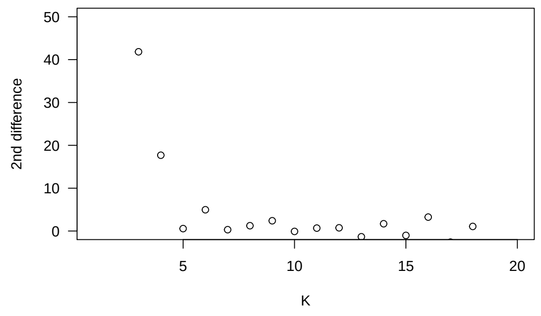


- Same warnings as with hierarchical clustering
- Look for *elbow* in plot of SSE vs. K

```
#-- Plot 1-step differences
dif1 = SSE - lead(SSE)   # SSE(K) - SSE(K-1)
plot(dif1, ylab="difference", xlab="K", las=1,
     ylim=c(0, 50))
```



```
#-- Plot linear (2nd) differences
dif2 = SSE - 2*lead(SSE) + lead(SSE, 2)   # SSE(K) -2SSE(K-1) +SSE(K)
plot(dif2, ylab="2nd difference", xlab="K", las=1,
     ylim=c(0, 50))
```

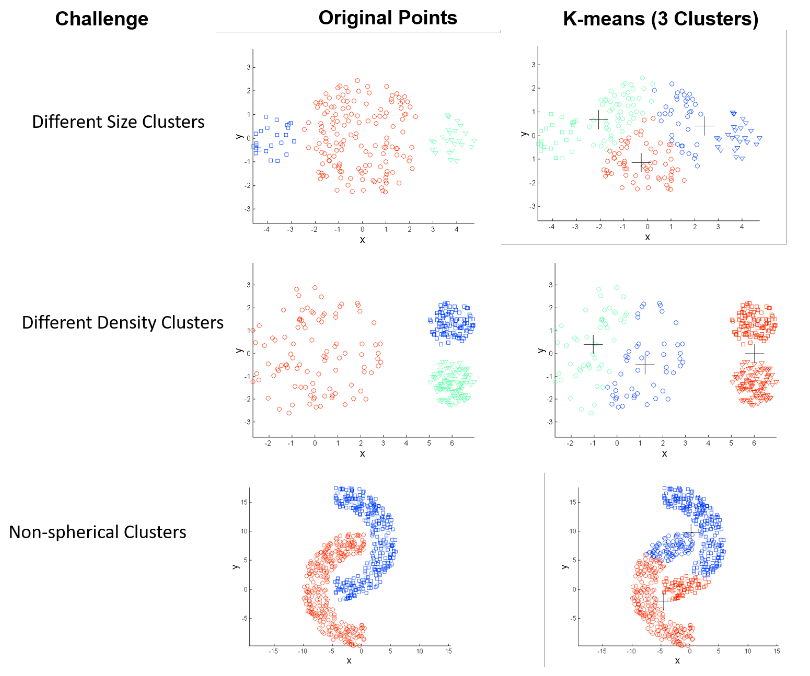


```
#-- Evaluate to truth (both indicate K=5 isn't bad)
km = kmeans(X, centers=5, nstart=25)  # choose K=6
table(true=crabsY, est=km$cluster)
#>       est
#> true    1  2  3  4  5
#>   B:F 15 12  1 16  6
#>   B:M 10  7  5 13 15
#>   O:F 13  2 15  5 15
#>   O:M  7  4  9 15 15
```

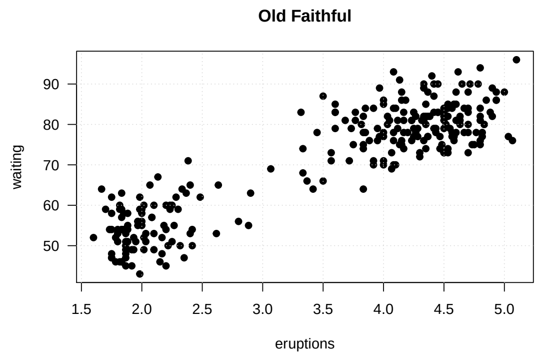
- Or use one of many statistical type tests (e.g. [Hamerly & Elkan, 2003])

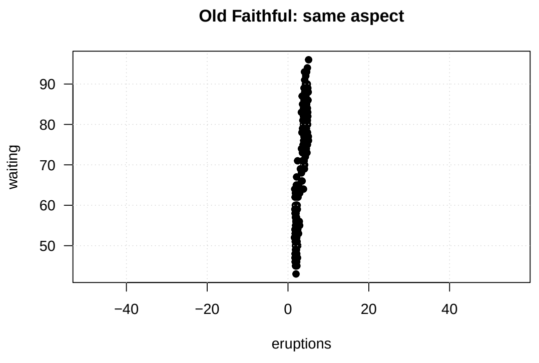
3.5 K means finds balanced, spherical clusters

- 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.

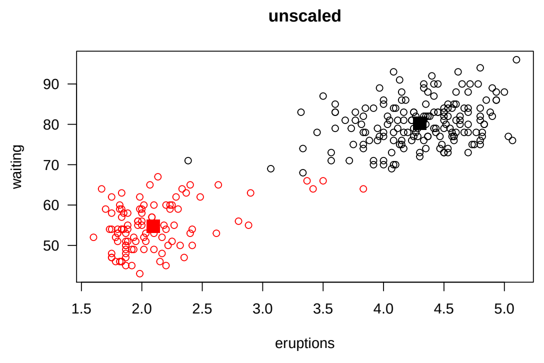


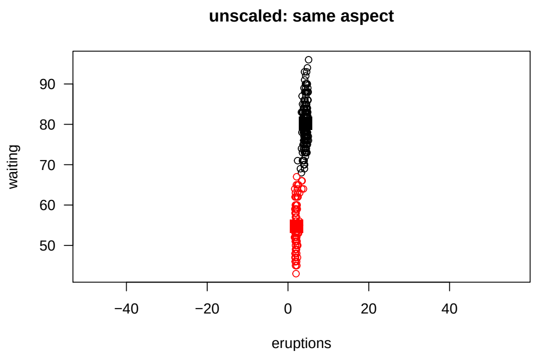
3.5.1 Old Faithful Example



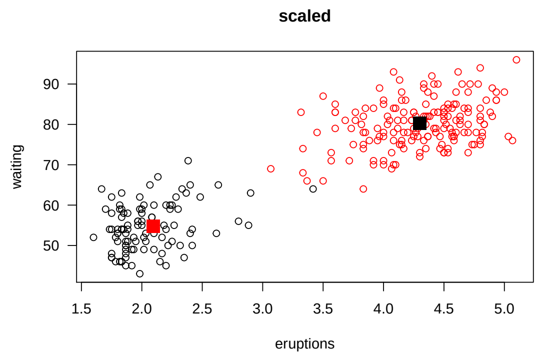


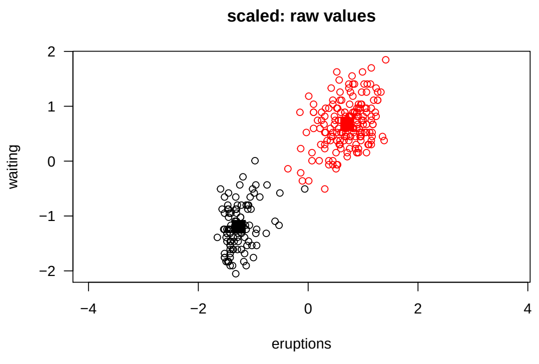
Unscaled Solution





Scaled Solution





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.

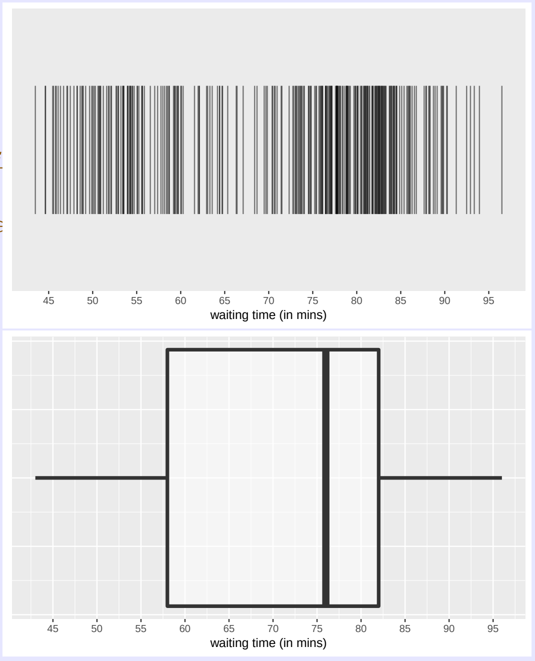
Live Streaming Webcam with eruption predictions

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 go out and watch.

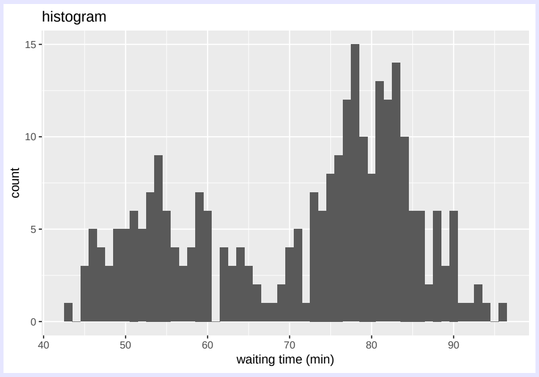
Your Turn #3 : Old Faithful

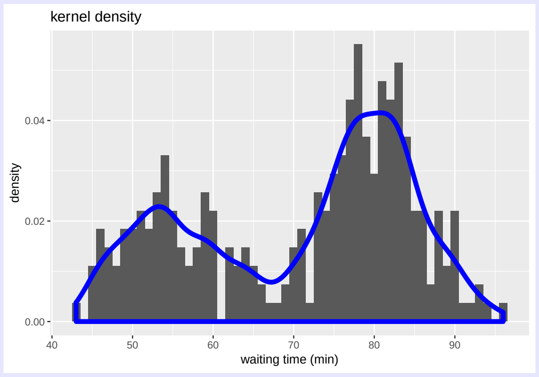
The data, summary statistics, and plots below represent a sample of *waiting times*, the time (in min) between Old Faithful eruptions.

```
#-- Load the Old Faithful data
wait = datasets::faithful$waiting

#-- Calculate summary stats
length(wait)                # sample si
#> [1] 272
summary(wait)               # six numbe
#>    Min. 1st Qu.  Median    Mean
#>    43.0    58.0    76.0    70.9
mean(wait)                  # mean
#> [1] 70.9
sd(wait)
#> [1] 13.59
median(wait)
#> [1] 76
quantile(wait, probs=c(.25,.50,.75))
#> 25% 50% 75%
#>  58  76  82
```



```
#-- Put data into a data.frame/tibble for use with ggplot
wait.df = tibble(wait)

#-- Make a ggplot object
pp = ggplot(wait.df, aes(x=wait)) + xlab("waiting time (min)")

#-- Histogram
pp + geom_histogram(binwidth = 1) + ggtitle("histogram")

#-- overlay kernel density plot
pp + geom_histogram(binwidth = 1, aes(y=stat(density))) +   # *density* histogram
  geom_density(bw=2, size=2, color="blue") + ggtitle("kernel density")
```





1. What can you say about the shape of the distribution?
2. Would a Gaussian (i.e., Normal) Distribution be a good choice for modeling the distribution of these data?
3. What would you recommend?

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 different clusters/groups

$$f(x) = \sum_{k=1}^{K} \pi_k \, f_k(x)$$

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

4.3 Univariate Gaussian Mixture Model

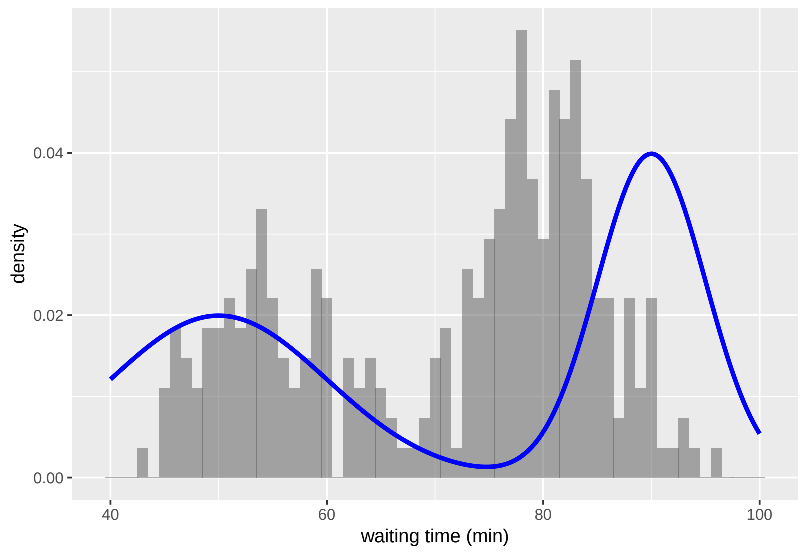
Consider a two-component ($K = 2$) mixture of Gaussian distributions:

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

- $0 \leq \pi \leq 1$
- $\theta = (\pi, \mu_1, \sigma_1, \mu_2, \sigma_2)$

4.3.1 Example: Old Faithful

```r
#-- Function to calculate Gaussian mixture pdf
dnmix <- function(theta1, theta2, w=.5, x.seq=seq(-4, 4, length=100)){
  f1 = dnorm(x.seq, mean=theta1[1], sd=theta1[2])
  f2 = dnorm(x.seq, mean=theta2[1], sd=theta2[2])
  fmix = f1*w + f2*(1-w)
  return(fmix)
}

#-- Set parameters
theta1 = c(mu=50, sigma=10)        # parameters for component 1
theta2 = c(mu=90, sigma=5)         # parameters for component 2
w = .5                             # mixture weight

#-- Make data for plotting
x.seq = seq(40, 100, length=200)
f = dnmix(theta1, theta2, w, x.seq)
data.mix = tibble(x.seq, f)

#-- Make plot
pp + geom_histogram(binwidth = 1, aes(y=stat(density)), alpha=.5) +
  geom_line(data=data.mix, aes(x=x.seq, y=f), color="blue", size=1.25)
```



Your Turn #4

What parameters do you suggest? Modify the code above to help you decide.

4.4 EM Algorithm

The details are found in the assigned reading Gaussian Mixture Models: 11.1-11.3, so we will just cover the basics.

Notation:

- Let $g_i \in \{1, 2, \dots, K\}$ be the (unknown) group/component identifier.
  - π_k is prior probability that any observation is from component k
- The data $D = \{X_1, X_2, \dots, X_n\}$
- The **responsibilities** are the posterior probability that event i came from component k:

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

- 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

EM Algorithm

Initialize

1. Set θ to something reasonable.

Repeat until convergence:

2. E-step: update r_{ik}, using θ, for $i = 1, 2, \dots, n$ and $k = 1, \dots, K$.
3. M-step: update θ using r_{ik}

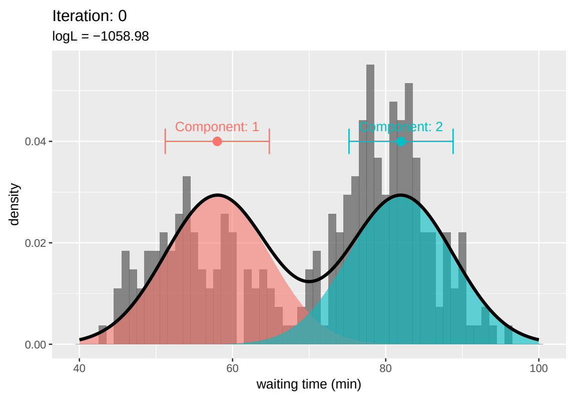
For Gaussian components:

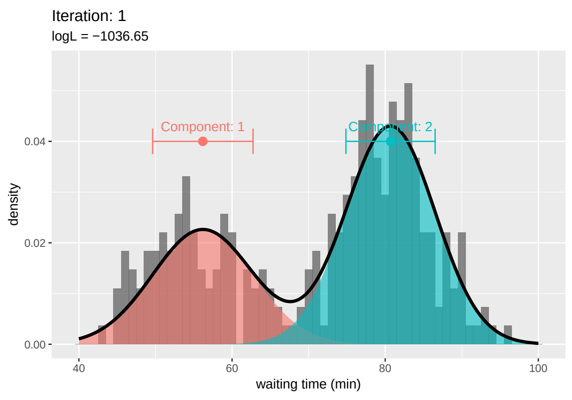
- Estimate like usual, except with *weighted* observations:

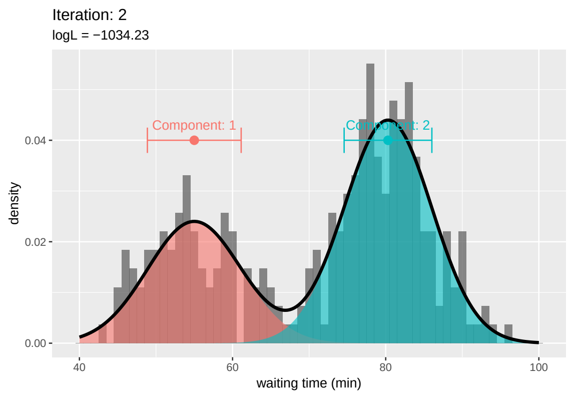
$$\begin{aligned} n_k &= \sum_{i=1}^{n} r_{ik} \\ \pi_k &= n_k / n \\ \mu_k &= \frac{1}{n_k} \sum_{i=1}^{n} r_{ik} x_i \\ \sigma_k &= \frac{1}{n_k} \sum_{i=1}^{n} r_{ik} (x_i - \mu_k)^2 \end{aligned}$$

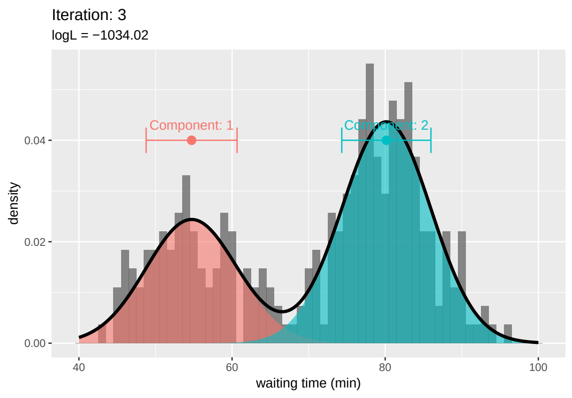
4.4.1 R package mixtools

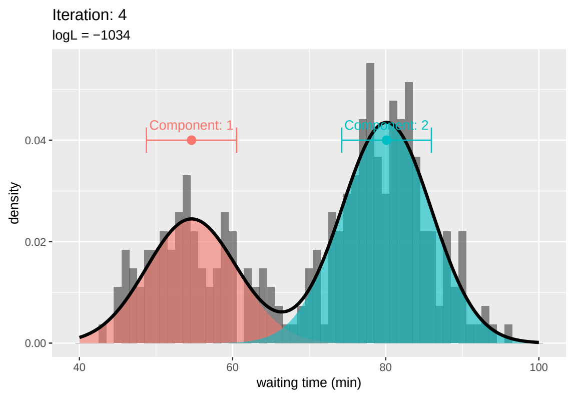
```
library(mixtools)
gauss_mix = normalmixEM(wait, k=2)   # 2 component gaussian mixture
#> number of iterations= 30

(w = gauss_mix$lambda)        # prior probabilities (pi)
#> [1] 0.3609 0.6391
(mu = gauss_mix$mu)           # component means
#> [1] 54.61 80.09
(sigma = gauss_mix$sigma)     # component standard deviations
#> [1] 5.871 5.868
r = gauss_mix$posterior       # responsibiliites matrix
```

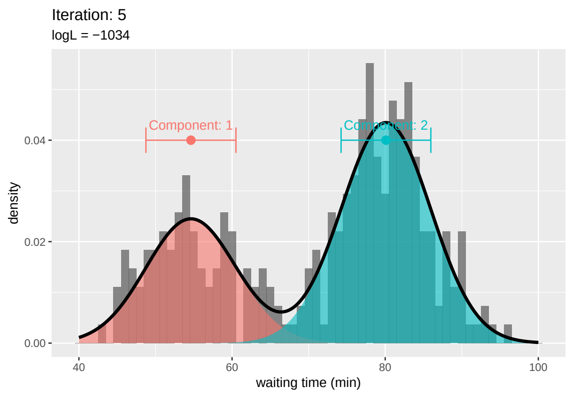

Iteration: 0
logL = −1058.98
Component: 1
Component: 2
density
0.00
0.02
0.04
40
60
80
100
waiting time (min)

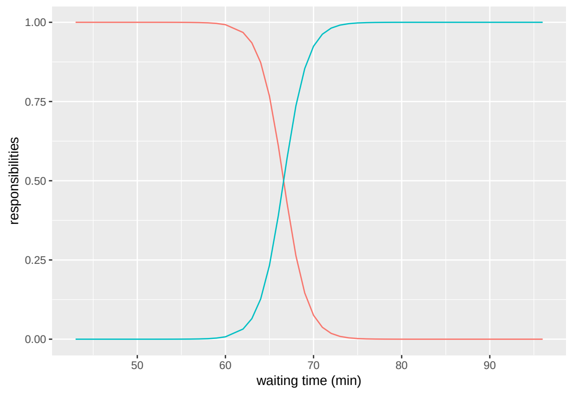

Iteration: 1
logL = −1036.65
Component: 1
Component: 2
density
waiting time (min)


Iteration: 2
logL = −1034.23
Component: 1
Component: 2
density
waiting time (min)


Iteration: 3
logL = −1034.02
Component: 1
Component: 2
density
waiting time (min)


Iteration: 4
logL = −1034
Component: 1
Component: 2
density
waiting time (min)


Iteration: 5
logL = −1034
Component: 1
Component: 2
density
waiting time (min)


responsibilities
0.00
0.25
0.50
0.75
1.00
50
60
70
80
90
waiting time (min)