15 Unsupervised learning

In **supervised learning**, the data always have two parts, X and Y, and we focus on learning how to infer the relationship of X with Y. The outcome Y drives (or "supervises") the learning process. There are multiple inferential goals in surpervised learning. In machine learning, the inference has been mostly about *prediction*.

In **unsupervised learning**, there is no Y, or in other words, no distinction between X and Y. There are multiple goals in unsupervised learing tasks: (1) clustering, (2) density estimation, (3) dimensionality reduction, (4) encoding, (5) association rule learning, etc. Since there is no designated outcome to "supervise" the process, one could view all variables as X, or all variables as the outcome Y (with a constant X across all observations). Many methods for unsupervised learning rely on measures of dissimilarity or similarity, often in a pairwise fashion. Note that kernel SVMs rely on a pairwise similarity measure on X.

15.1 Clustering

Clustering is to divide data into non-overlapping subsets (i.e., a partition) of relatively homogeneous observations. Such a subset is called a **cluster**.

15.1.1 K-means clustering (ISLR 10.3.1)

For the K-means method, we need to have

(1) a measure of **dissimilarity**, $d(x_i, x_{i'})$, between any two points, x_i and $x_{i'}$;
(2) a definition of **centroid** for any set of data points;
(3) a number K, the target number of subsets.

Given a cluster, the within-cluster heterogeneity may be defined as

$$W(C) = \frac{1}{|C|} \sum_{i,i' \in C} d(x_i, x_{i'}), \tag{10.10}$$

where C is the set of indices for the cluster.

A popular choice for $d()$ is the **squared Euclidean distance** $d(x_i, x_{i'}) = \|x_i - x_{i'}\|^2 = \sum_j (x_{ij} - x_{i'j})^2$. This requires **all the features be standardized** unless you have a reason not to. The **centroid** of a cluster is often defined as the average $\frac{1}{|C|} \sum_{i \in C} x_i$. A centroid defined this way is the point x_0 that minimizes $\frac{1}{|C|} \sum_{i \in C} \|x_i - x_0\|^2$, its average squared Euclidean distance from all the observations in the cluster.

Goal: Given K, find a partition of data so that the total within-cluster heterogeneity defined in (10.10) is minimized. Given a partition of data, let $\{C_1, \ldots, C_K\}$ be the sets of indices for the partition. Then the goal is

$$\text{minimize}_{\{C_1,\ldots,C_K\}} \sum_{k=1}^{K} W(C_k). \tag{10.9}$$

K-means algorithm: A generic version of the algorithm is below (shown in ISLR Figure 10.6):

1. Randomly assign the observations to K classes (every class needs to have at least one observation).
2. Iterate the following two steps:
   - Update centroids: Calculate the centroids for the K classes.
   - Update membership: Re-assign every observation to the class represented by the centroid "closest" to it (i.e., that centroid that has the smallest d from the observation).
3. Once converged (i.e., there is no change in membership or centroids), record the partition and its $\sum_k W(C_k)$.
4. Repeat 1–3 multiple times. Select the partition with the smallest $\sum_k W(C_k)$.

Notes:

- This is a greedy algorithm. Steps 1–3 do not guarantee to reach the overall mimimum. Step 4 increases the chance of reaching it.

- The algorithm can also be started with K centroids.
- Starting with K random centroids may lead to less than K clusters if one of the centroids is far away from all observations. Starting with a random partition with K non-empty subsets may also lead to less than K clusters. In `kmeans()`, K distinct observations are chosen as the starting centroids; this guarantees every cluster has at least one observation assigned to it.
- Adding features may bring more information (if they are informative features) or dilute information (if they are noise features). Adding noise features leads to an effect similar to the 'curse of dimensionality'.
- K is a hyperparameter. (I am not aware of any method to help select the K in K-means.)
- When $d()$ is the squared Euclidean distance, the iteration guarantees to reduce $\sum_k W(C_k)$ at every step. This is because for any cluster of m vectors $t_1, \ldots, t_m$, the within-cluster heterogeneity is

$$W = \frac{1}{m}\sum_{i,j}\|t_i - t_j\|^2 = 2\sum_i \|t_i - \bar{t}\|^2,$$

  where $\bar{t} = \frac{1}{n}\sum_i t_i$. After the m-th round of update, let $c_m : \{1, \ldots, n\} \to \{C_1, \ldots, C_K\}$ be the partitioning function and $\mu_{k,m}$ be the center of the k-th cluster. Then the total within-cluster heterogeneity for this partition is $T_m = 2\sum_{i=1}^n \|x_i - \mu_{c_m(x_i),m}\|^2$. Changing memberships leads to $2\sum_{i=1}^n \|x_i - \mu_{c_{m+1}(x_i),m}\|^2 < T_m$, and changing centroids leads to $T_{m+1} < 2\sum_{i=1}^n \|x_i - \mu_{c_{m+1}(x_i),m}\|^2$. [Proof of the equation above for 1-dimension: On the one hand, $\sum_{i,j}(t_i - t_j)^2 = 2\sum_{i<j}(t_i - t_j)^2 = 2[(m-1)\sum_i t_i^2 - 2\sum_{i<j} t_i t_j]$. On the other hand, $\sum_i (t_i - \bar{t})^2 = \frac{1}{m^2}[m(m-1)\sum_i t_i^2 - 2m\sum_{i<j} t_i t_j] = \frac{1}{m}[(m-1)\sum_i t_i^2 - 2\sum_{i<j} t_i t_j]$.]

The base R has a function `kmeans()` for performing the K-means clustering using squared Euclidean distance. Unfortunately it does not even have the option for scaling the features! So, we need to scale the features before calling `kmeans()`.

```
library(ISLR)
Hitters1 = Hitters[,1:7]
names(Hitters1); dim(Hitters1)

aa = kmeans(Hitters1, 3, nstart=10)   ## K=3; run the algorithm 10 times
aa
str(aa)

all.equal(aa$tot.withinss, sum((Hitters1 - aa$centers[aa$cluster,])^2))  ## True
all.equal(aa$totss, sum((t(Hitters1) - apply(Hitters1,2,mean))^2))       ## True
```

Below we show why the algorithm should be run multiple times (specified with `nstart=`). We print out `tot.withinss` after every run of the algorithm. Note that every run has a different initialization. Partitions with different `tot.withinss` must be different partitions.

```
for(ii in 1:20) {
  aa = kmeans(Hitters1, 3, nstart=1)
  print(aa$tot.withinss)
}
```

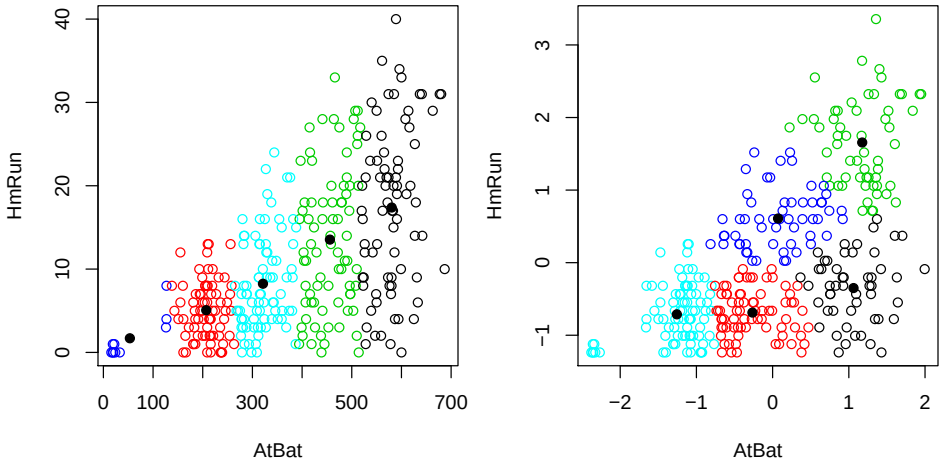
Now we standardize the features. The results are quite different between not scaling and scaling!

```
Hitters2 = scale(Hitters1)
aa = kmeans(Hitters1, 3, nstart=10)
bb = kmeans(Hitters2, 3, nstart=10)
table(aa$cluster, bb$cluster)  ## quite different results!
```

We can use two features to demonstrate the difference between scaling and not scaling features. With no scaling, `AtBat` has a wider range (16–687) and thus a much higher impact than `HmRun` (range 0–40). With no scaling, we effectively assume every increment in `AtBat` is equivalent to every increment in `HmRun`.

```
Hitters3 = Hitters1[,c(1,3)]
Hitters4 = scale(Hitters3)
aa = kmeans(Hitters3, 5, nstart=10)
bb = kmeans(Hitters4, 5, nstart=10)
```

```
table(aa$cluster, bb$cluster)

plot(Hitters3, col=aa$cluster); points(aa$centers, pch=19)
plot(Hitters4, col=bb$cluster); points(bb$centers, pch=19)
```



The **implementation** of the K-means algorithm in software often is not the algorithm above, but an optimized version that is faster and achieves exactly the same results as the one above. Optimizing the K-means algorithm is a classical topic in computer science.

A **clarification on various concepts**: It is helpful to distinguish these related concepts: (1) a goal, (2) a method, (3) an implementation algorithm of the method. For example,

(1) Our goal is to identify the partition that minimizes (10.9);
(2) The K-means algorithm is actually a method to achieve or approximate our goal;
(3) The steps as described in the method are an algorithm; this algorithm is often relatively easy to understand. The method in (2) may be achieved with different implemention algorithms. Some of them are more efficient (with respect to speed or memory usage) but may not be as easy to understand.

Sometimes a method is called an algorithm (as in here), which can cause confusions.

Similarly, for statistical models, it is helpful to distinguish these related concepts: (1) a model, (2) a model fitting criterion, (3) an algorithm to fit the model. For example,

(1) We wish to fit a linear model $y = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p + \epsilon$;
(2) We use the lasso criterion with $\lambda = 10$ (alternative criteria: least squares, maximum likelihood);
(3) We use gradient descent to fit the model (alternative algorithms: direct formula, Newton–Raphson).

The Keras syntax makes some of these distinctions, with its `layer` step mostly overlapping with (1) and (2), and its `compile` and `fit` steps overlapping with (3).

15.1.2 Hierarchical clustering (ISLR 10.3.2)

For the hierarchical clustering method, we need to have

(1) a measure of **dissimilarity**, $d(C_i, C_j)$, between any two non-overlapping *subsets*, C_i and C_j;
(2) either of the following:
- a **threshold** for dissimilarity (a height in dendrogram) to determine if two subsets belong to the same cluster;
- a number K, the target number of subsets.

Hierarchical clustering algorithm:

1. Start from n singletons (a singleton is a subset containing a single observation).
2. Repeat the following until a single set is reached:

- Among all the current subsets, merge the two subsets that have the smallest $d(C_i, C_j)$ (Because the smaller $d(C_i, C_j)$ the more similarity between C_i and C_j.)
- Record that $d(C_i, C_j)$.

3. Use a threshold value for $d(C_i, C_j)$ to cut the tree to obtain branches as clusters (as shown in ISLR Figure 10.9).

Notes:

- Hierarchical clustering is a *bottom-up* approach. In contrast, trees are built *top-down*.
- The results can be drawn as a **dendrogram**, with height $d(C_i, C_j)$ for the fuse point between C_i and C_j.
- The threshold value is a hyperparameter.
- ISLR Figures 10.10 and 10.11 illustrate this algorithm.

Definition of $d(C_i, C_j)$: The measure $d(C_i, C_j)$ may not be easily defined directly on all subsets. It is often easier to define a measure of dissimilarity between two points, $d(x_i, x_j)$. Then $d(C_i, C_j)$ can be defined through $d(x_i, x_j)$, in one of the following ways (called **linkage**):

- $d(C_1, C_2) = \max_{i \in C_1, j \in C_2} d(x_i, x_j)$ (**complete** linkage)
- $d(C_1, C_2) = \min_{i \in C_1, j \in C_2} d(x_i, x_j)$ (**single** linkage)
- $d(C_1, C_2) = \text{average}_{i \in C_1, j \in C_2} d(x_i, x_j)$ (**average** linkage)
- $d(C_1, C_2) = d(\text{centroid}_1, \text{centroid}_2)$ (**centroid** linkage)

Common choices for the definition of dissimilarity $d(x_i, x_j)$ are:

(1) Euclidean distance (often requiring *feature standardization* to make sense);
(2) One minus correlation coefficient (e.g., Pearson or Spearman correlation). The correlation is between two observations across all features, and so *feature standardization* is often necessary for the results to make sense.
(3) One minus Jaccard index, if the data are binary indicating presence or absence of features. The **Jaccard index** between two observations is (# features present in both) / (# features present in either). In R package `vegan`, `vegdist()` can calculate this and other dissimilarity indices.

Notes:

- Examples in ISLR Figure 10.12; also see below.
- Single linkage often has the signature pattern of adding one observation at a time.
- Different measures of dissimilarity can reflect quite differently on whether two observations are "similar" or not. ISLR Figure 10.13 shows an example for the definitions (1) and (2) above.

The base R has `hclust()` for hierarchical clustering, which by default uses complete linkage. It takes a `dist` object, which can be generated by calling `dist()` or `as.dist()`. We can use `cutree()` to cut a `hclust` tree into clusters. The function `dist()` by default computes the Euclidean distance between every pair of observations.

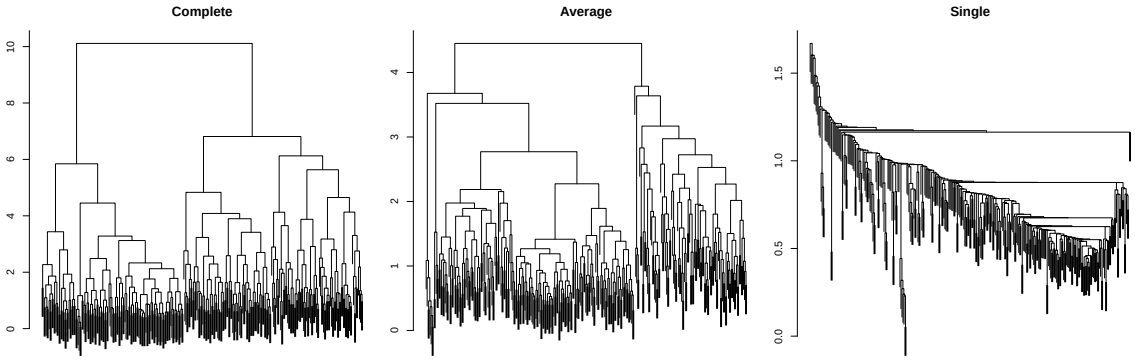
```
library(ISLR)
Hitters1 = Hitters[,1:7]
Hitters2 = scale(Hitters1)  ## scaling the features
dist1 = dist(Hitters2)      ## pairwise distance
dim(Hitters2)               ## 322 x 7
str(dist1)                  ## 51681 = 322 * 321 / 2
sqrt(sum((Hitters2[1,]-Hitters2[2,])^2))  ## dist between 1st and 2nd observations
```

One can obtain a partition by specifying either the height in a dendrogram or the target number of subsets.

```
cluster1 = cutree(hclust(dist1), h=5.7)   ## height 5.7
cluster2 = cutree(hclust(dist1), k=5)     ## K=5
table(cluster1); table(cluster2)
table(cluster1, cluster2)  ## same clustering
```

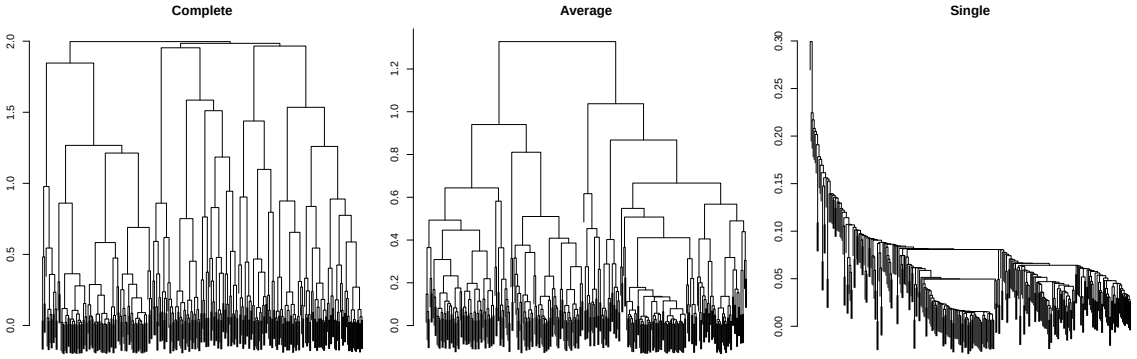
The dendrograms are:

```
plot(hclust(dist1, method='complete'), labels=F, xlab='', main="Complete")
plot(hclust(dist1, method='average'), labels=F, xlab='', main="Average")
plot(hclust(dist1, method='single'), labels=F, xlab='', main="Single")
```



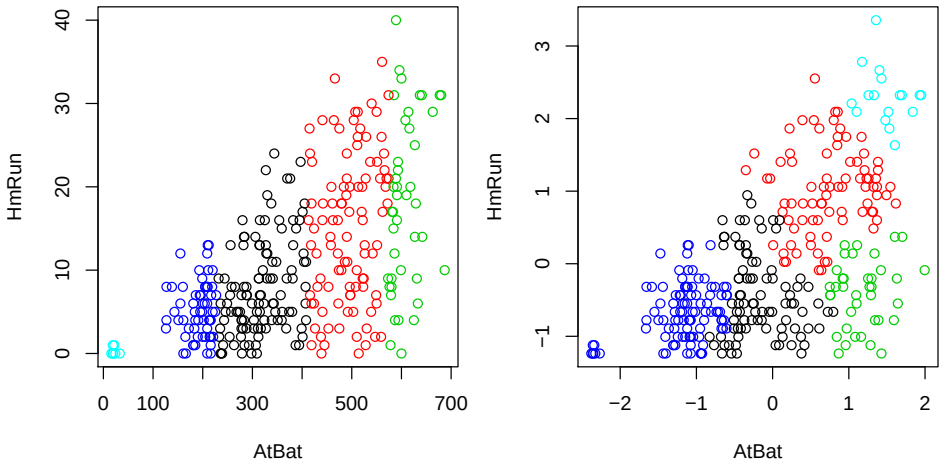
Now we define the dissimilarity as 1 - Pearson correlation, which has a range $[0, 2]$.

```
dist2 = as.dist(1 - cor(t(Hitters2)))

plot(hclust(dist2, method='complete'), labels=F, xlab='', main="Complete")
plot(hclust(dist2, method='average'), labels=F, xlab='', main="Average")
plot(hclust(dist2, method='single'), labels=F, xlab='', main="Single")
```



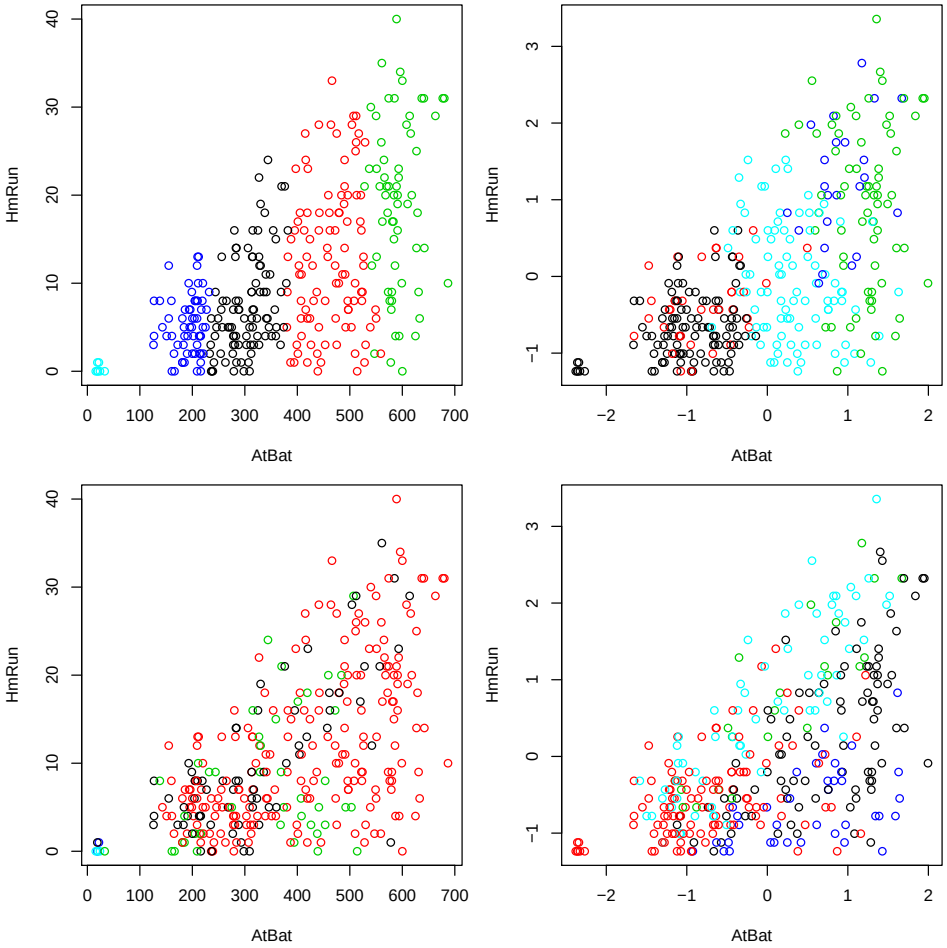
We now use two features to demonstrate the difference between scaling and not scaling features.

```
Hitters3 = Hitters1[,c(1,3)]
Hitters4 = scale(Hitters3)
aa = cutree(hclust(dist(Hitters3)), k=5)
bb = cutree(hclust(dist(Hitters4)), k=5)
table(aa, bb)
plot(Hitters3, col=aa); plot(Hitters4, col=bb)
```



Finally we use all 7 features and compare the results between Euclidean distance and 1-cor and between non-scaling and scaling. The results are quite different. We can plot the results with 2 features as the backdrop.

```
aa = cutree(hclust(dist(Hitters1)), 5)                ## no scaling, Euclidean
bb = cutree(hclust(dist(Hitters2)), 5)                ## scaled, Euclidean
cc = cutree(hclust(as.dist(1-cor(t(Hitters1)))), 5)   ## no scaling, 1-cor
dd = cutree(hclust(as.dist(1-cor(t(Hitters2)))), 5)   ## scaled, 1-cor
table(aa, cc); table(bb, dd)

plot(Hitters3, col=aa); plot(Hitters4, col=bb)
plot(Hitters3, col=cc); plot(Hitters4, col=dd)
```



15.1.3 Clustering using a mixture model

For the mixture model approach, we need to have

(1) a family of distributions (with parameters θ) for the clusters;
(2) a number K, the target of subsets.

In a **mixture model**, we assume the data are from a mixture of K distributions with unknown parameters θ_k and unknown mixture probabilities π_k:

$$h(y|\phi) = \sum_{k=1}^{K} \pi_k f(y|\theta_k) \quad \text{subject to} \quad \pi_k \geq 0, \quad \sum_{k=1}^{K} \pi_k = 1,$$

where $\phi = (\pi_1, \ldots, \pi_K, \theta_1, \ldots, \theta_K)$. Once we have estimated all the parameters, we can calculate the posterior

probability for each observation to belong to a class k,

$$\hat{p}_k = P(k|y, \hat{\phi}) = \frac{\hat{\pi}_k f(y|\hat{\theta}_k)}{\sum_k \hat{\pi}_k f(y|\hat{\theta}_k)},$$

and assign the observation to the class with the largest posterior probability.

Notes:

- Because this is a likelihood-based method, AIC and BIC can be calculated to help select K.
- The R package `flexmix` has a function `stepFlexmix()` for this. See the next section for an example.
- This method is similar to the mixure model described for LDA/QDA. However, the latter is for supervised learning, with data that has known classes for the observations, and that often has known probabilities π_k.

This method is a special case of **latent class regression**, in which we assume that given x, the outcome y is from a finite mixture distribution with K components,

$$h(y|x, \phi) = \sum_{k=1}^{K} \pi_k f(y|x, \theta_k) \quad \text{subject to} \quad \pi_k \geq 0, \quad \sum_{k=1}^{K} \pi_k = 1.$$

When there are no x (or a constant x across all observations), **this regression problem becomes a clustering problem**! Again, we can calculate the posterior probability for each observation as

$$P(k|x, y, \hat{\phi}) = \frac{\hat{\pi}_k f(y|x, \hat{\theta}_k)}{\sum_k \hat{\pi}_k f(y|x, \hat{\theta}_k)},$$

and assign the observation to the class with the largest posterior probability.

For both methods, an **expectation-maximization** (EM) algorithm can be used to obtain the parameter estimates. EM algorithms are iterative, looping through an E-step and an M-step to update parameter estimates until convergence. Specifically, after initializing the parameters, we iterate between the following two steps:

- E-step: Given current $\hat{\phi}$, calculate $\hat{p}_{ik} = P(k|x_i, y_i, \hat{\phi})$. Then update $\hat{\pi}_k = \frac{1}{n}\sum_{i=1}^{n} \hat{p}_{ik}$ $(k = 1, \ldots, K)$.
- M-step: Maximize $l_k(\theta_k) = \sum_i \hat{p}_{ik} \log f(y_i|x_i, \theta_k)$ to obtain new $\hat{\theta}_k$ $(k = 1, \ldots, K)$.

[Math details: Consider the full data $\{(y_i, x_i, g_i)\}$, where g_i is the unobserved class for observation i. The log-likelihood for the full data is $l = \sum_i \log f(y_i|x_i, \theta_{g_i})$. The E-step is to calculate the expectation $l_E = E_{G|(Y,X)}(l)$ under the current estimate of the conditional distribution of $G|(Y, X)$. Then $l_E = \sum_i \sum_k \hat{p}_{ik} \log f(y_i|x_i, \theta_k) = \sum_k l_k(\theta_k)$. The M-step is to maximize $l_E = \sum_k l_k(\theta_k)$ with respect to all the parameters, which is equivalent to maximizing $l_k(\theta_k)$ for all k.]

15.1.4 Example: Clonal detection with cancer single-cell sequencing

Gawad et al. Dissecting the clonal origins of childhood acute lymphoblastic leukemia by single-cell genomics. PNAS. 2014 111:17947–17952. (paper; code)

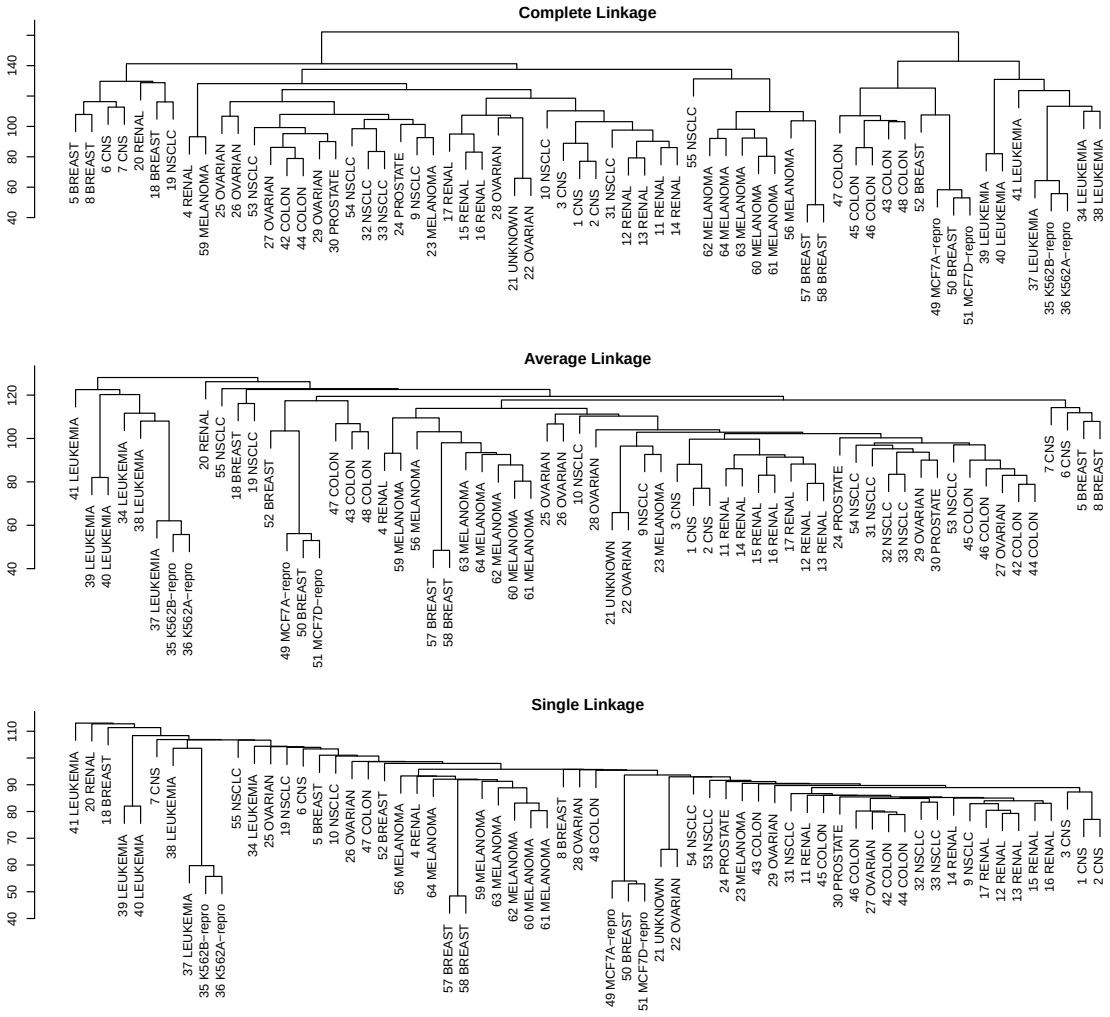
15.1.5 Example: NCI60 data

From Ross et al. Nature Genetics, 2000 24:227–234. See http://genome-www.stanford.edu/nci60/ for more details and some of the clustering plots.

```
library(ISLR)
nci.labs=NCI60$labs
nci.data=NCI60$data
nci.labs = paste(1:64, nci.labs)  ## add indices to labels for the plots
dim(nci.data)  ## 64 x 6830
hist(nci.data)

sd.data = scale(nci.data)  ## standardize the columns (genes)
data.dist = dist(sd.data)  ## default is Euclidean distance
```

We perform hierarchical clustering on the data.

```
hc1 = hclust(data.dist)                    ## complete linkage by default
plot(hc1, labels=nci.labs, main="Complete Linkage", xlab="", ylab="", sub="")
#abline(h=hc1$height[1:3], col='grey')      ## add the first 3 heights

hc2 = hclust(data.dist, method="average")  ## average linkage
plot(hc2, labels=nci.labs, main="Average Linkage", xlab="", ylab="", sub="")

hc3 = hclust(data.dist, method="single")   ## single linkage
plot(hc3, labels=nci.labs, main="Single Linkage", xlab="", ylab="", sub="")
```



The information necessary for the plots are stored in the results.

```
names(hc1)
hc1$order    ## the order of observations to be drawn
hc1$height
dim(hc1$merge); hc1$merge[1:10,]  ## singletons are negative; merged subsets are positive
```

15.1.6 Other clustering methods and R packages

There are several other clustering methods and several R packages for clustering. Here is an incomplete list.

- `flexmix` for admixture modeling.

- `NbClust`
- `GMD`
- Affinity propagation (AP) (Frey and Dueck, Science 2007, 315:972–976)
- X-means: An extension of K-means