K-Means Clustering

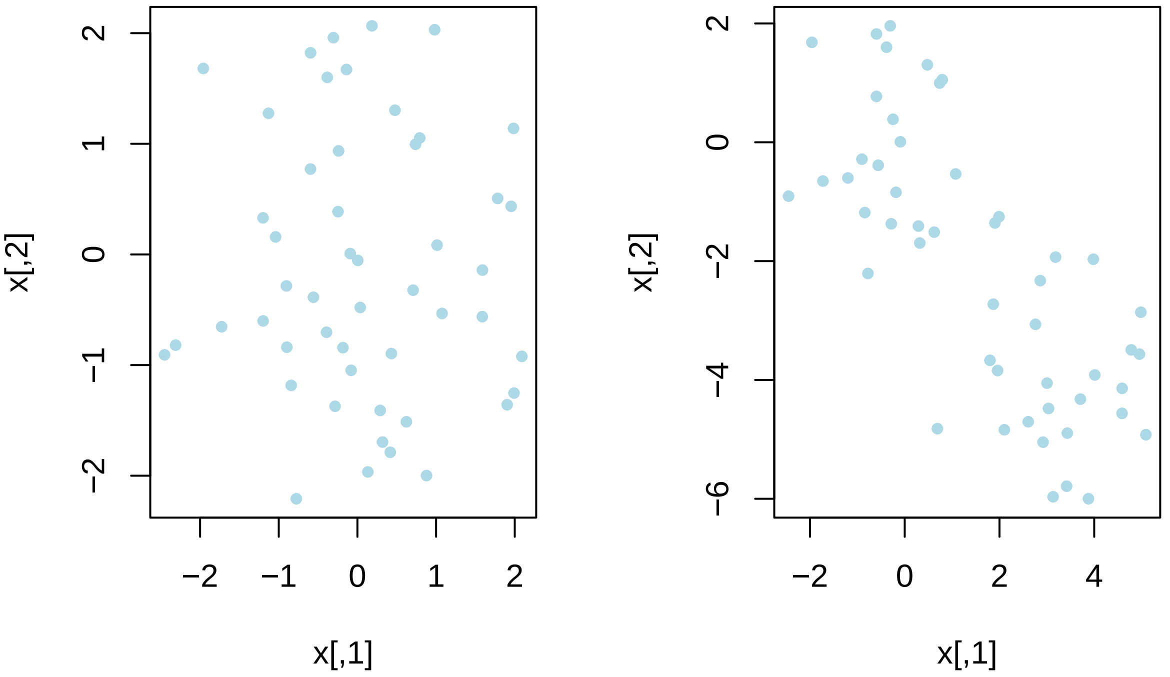
Trevor Hastie and Robert Tibshirani

Here, I am adapting part of the lab associated with Chapter 12 of the textbook.

K-Means Clustering

The function `kmeans()` performs K-means clustering in `R`. We begin with a simple simulated example in which there truly are two clusters in the data: the first 25 observations have a mean shift relative to the next 25 observations.

```r
set.seed(2)
x <- matrix(rnorm(50 * 2), ncol = 2)
par(mfrow = c(1, 2))
plot(x, col="lightblue", pch=20)
x[1:25, 1] <- x[1:25, 1] + 3
x[1:25, 2] <- x[1:25, 2] - 4
plot(x, col="lightblue", pch=20)
```



We now perform K-means clustering with $K = 2$.

```r
km.out <- kmeans(x, 2, nstart = 20)
```

The options above are the following:

- 2 stands for `centers` which as an integer is interpreted as the number of clusters
- `nstart=20` means that `R` is instructed to look at 20 different initial configurations and then picks out the one with the best "objective"

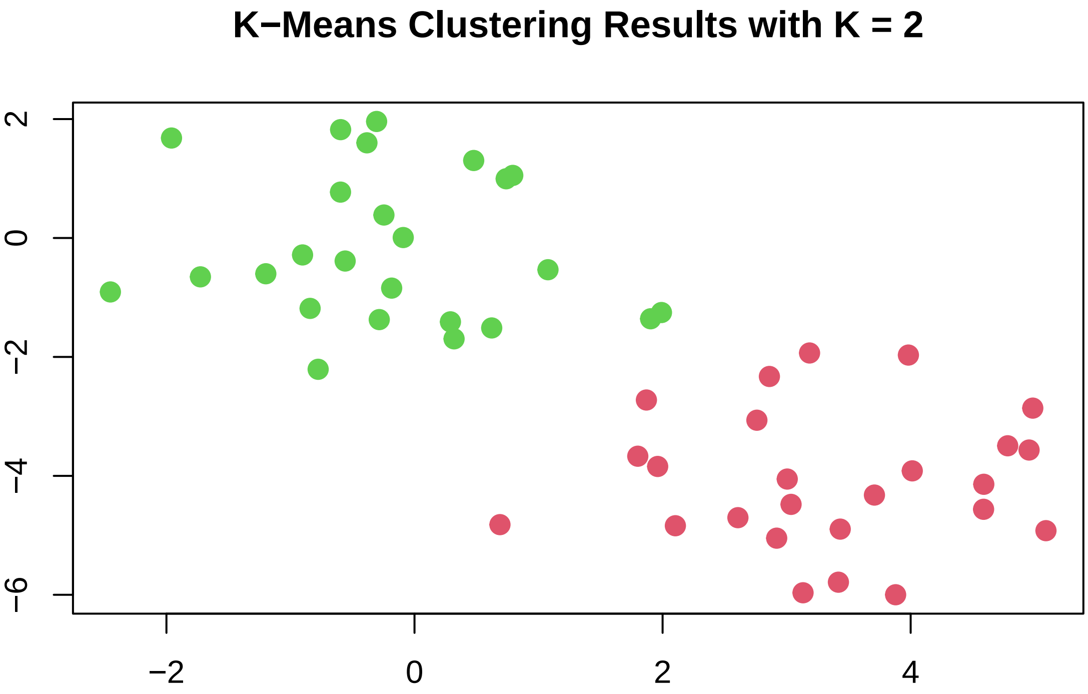
The cluster assignments of the 50 observations are contained in `km.out$cluster`.

```
km.out$cluster
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2
## [39] 2 2 2 2 2 2 2 2 2 2 2 2
```

The K-means clustering perfectly separated the observations into two clusters even though we did not supply any group information to `kmeans()`. We can plot the data, with each observation colored according to its cluster assignment.

```
#par(mfrow = c(1, 2))
plot(x, col = (km.out$cluster + 1),
    main = "K-Means Clustering Results with K = 2",
    xlab = "", ylab = "", pch = 20, cex = 2)
```



Here the observations can be easily plotted because they are two-dimensional. If there were more than two variables then we could instead perform PCA and plot the first two principal components score vectors.

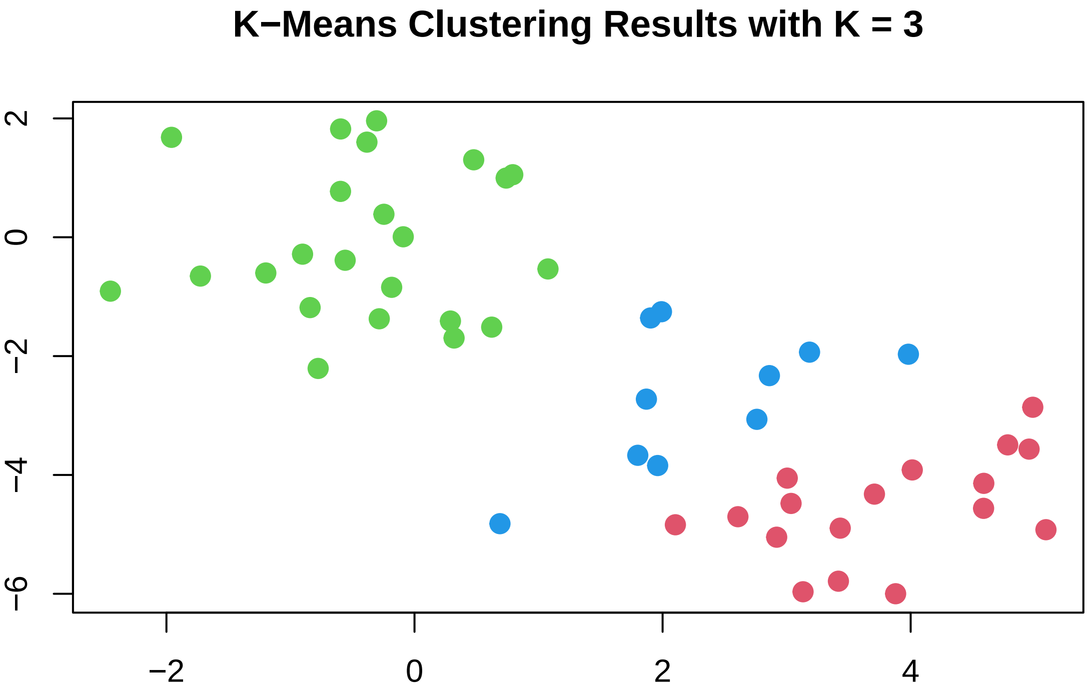
In this example, we knew that there really were two clusters because we generated the data. However, for real data, in general we do not know the true number of clusters. We could instead have performed K-means clustering on this example with $K = 3$.

```
set.seed(4)
km.out <- kmeans(x, 3, nstart = 20)
km.out
```

```
## K-means clustering with 3 clusters of sizes 17, 23, 10
##
## Cluster means:
##         [,1]        [,2]
## 1  3.7789567 -4.56200798
## 2 -0.3820397 -0.08740753
## 3  2.3001545 -2.69622023
##
## Clustering vector:
```

```
##  [1] 1 3 1 3 1 1 1 3 1 3 1 3 1 3 1 3 1 1 1 1 1 3 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2
## [39] 2 2 2 2 2 3 2 3 2 2 2 2
##
## Within cluster sum of squares by cluster:
## [1] 25.74089 52.67700 19.56137
##  (between_SS / total_SS =  79.3 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

```
plot(x, col = (km.out$cluster + 1),
    main = "K-Means Clustering Results with K = 3",
    xlab = "", ylab = "", pch = 20, cex = 2)
```



When $K = 3$, K-means clustering splits up the two clusters.

To run the `kmeans()` function in `R` with multiple initial cluster assignments, we use the `nstart` argument. If a value of `nstart` greater than one is used, then K-means clustering will be performed using multiple random assignments in Step~1 of Algorithm 12.2, and the `kmeans()` function will report only the best results. Here we compare using `nstart = 1` to `nstart = 20`.

```
set.seed(4)
km.out <- kmeans(x, 3, nstart = 1)
km.out$tot.withinss
```

```
## [1] 104.3319
```

```
km.out <- kmeans(x, 3, nstart = 20)
km.out$tot.withinss
```

```
## [1] 97.97927
```

Note that `km.out$tot.withinss` is the total within-cluster sum of squares, which we seek to minimize by performing K-means clustering (Equation 12.17). The individual within-cluster sum-of-squares are contained in the vector `km.out$withinss`.

We *strongly* recommend always running K-means clustering with a large value of `nstart`, such as 20 or 50, since otherwise an undesirable local optimum may be obtained.

When performing K-means clustering, in addition to using multiple initial cluster assignments, it is also important to set a random seed using the `set.seed()` function. This way, the initial cluster assignments in Step~1 can be replicated, and the K-means output will be fully reproducible.