10 Unsupervised Learning

Andrew Liang

11/25/2020

Notes

Challenges of Unsupervised Learning

- tends to be much more subjective
- often perform as part of *exploratory data analysis*
- no way to check our work since we don't know the true answer

Principal Components Analysis

- when faced with large set of correlated variables, PCA allows us to summarize using smaller number of variables that expalains the most variability
- directions in feature space along which original data *varies the most*
  - also defines lines and subspaces that are *as close as possible* to data cloud
- serves as a tool for data viz

Given n observations with p features $X_1, \ldots, X_p$, we can do some EDA with scatterplots, but with large p, then the number of plots becomes very big

We want to find low-dimensional representation of the data that captures as much of the info as possible

First principal component of a set of features $X_1, \ldots, X_p$ is the normalized linear combination:

$$Z_1 = \phi_{11} X_1 + \ldots + \phi_{p1} X_p$$

where $\sum_{j=1}^{p} \phi_{j1}^2 = 1$

- $\phi_1 = (\phi_{11} \phi_{21} \ldots \phi_{p1})^T$ is the principal component loadings vector for the first component
  - defines direction in feature space along data which vary the most
- important to scale the variables first, since we're using the variance in the data
  - only scale if variables are measured in different units
  - typically scale each variable to have standard deviation one before performing PCA

Uniqueness of Principal Components

- each PC loading vector is unique, up to a sign flip
  - meaning two different software packages will yield the same PC loadings, though the signs may differ
  - flipping sign *does not* effect the PC loading as the PC loading is a line that extends in either direction

Proportion of Variance Expalained

- how much info is lost by projecting observations onto first few PC?
  - how much *variance* is not contained in the first few PCs?
- an positive amount of proportion of variance explained (PVE) is in each loadings, while the cumulative PVE can just be summed up in all the loadings
- Ultimate goal is to use as few principal components to get a good unerstanding of the data
  - can use a *scree* plot to visualize this
  - no well-accepted objective way to decide how many PCs are enough
    - main reason why PCA is generally used as an EDA approach

Clustering Methods

- broad set of techniques to finding *subgroups* (clusters) in a dataset
- must define what it means for two or more observations to be *similar* of *different*
- instead of looking at variability such as in PCA, clustering aims to find homogenous subgroups among the observations

K-Means Clustering

- must first specify desired number of clusters K, then the algorithm will assign each observation to exactly one of K clusters
- must satisfy two properties:
  - each observation must belong to at least one of the K clusters
  - clusters are non-overlapping (no observation belongs to more than one cluster)
- a *good* clustering is one for which the *within-cluster variation* $W(C_k)$ is as small pas possible
  - we try to solve:

$$\min_{C_1,\dots,C_K} \sum_{j=1}^{K} W(C_k)$$

hence the goal is to partition observations into K clusters such that the total within-cluster variation (summed over all K clusters) is minimized

- to define the within-cluster variation, the common choice is to use *squared Euclidean distance*:

$$W(C_k) = \frac{1}{|C_k|} = \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2$$

where $|C_k|$ is the number of obs in the kth cluster. Thus, the within-cluster variation is just a sum of all pairwise squared Euclidean distances between observations in the kth cluster, divided by the total number of observations in the kth cluster. And so K-Means tries to do the optimization problem:

$$\min_{C_1,\dots,C_K} \left\{ \sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2 \right\}$$

Now we try to find an algorithm to solve the above, which is the K-means clustering:

1. Randomly assign a number from 1 to K, to each of the observations. These serve as initial cluster assignments for observations
2. Iterate until cluster assignments stop changing:
    - for each K clusters, computer cluster *centroid*. The kth cluster centroid is the vector of the p feature means for observations in the kth cluster
    - assign each observation to the cluster whose centroid is closest (using Euclidean distance)

The challenge with K-Means is actually choosing K, the number of prespecified clusters

Hierarchical Clustering

- does not require to choose K
- creates an attractive tree-based representation of observations, called a *dendrogram*

Interpreting a Dendrogram

- each *leaf* of a dendrogram represents one of the observations
- as you move up to the branches, leaves begin to *fuse* with each other, meaning that the observations are similar to one another
- observations that fuse later (near the stump of the tree) can be quite different
- we draw conclustions about the similarity of two observations based on the location on the *vertical axis* NOT the *horizontal axis*
- we make *cuts* horizontally on the tree and that gives us the number of clusters shown below the cut
    - we can make between 1 and up to n different clusters this way
    - choice of where to cut is not clear
    - because of the difficulty of where to cut, hierarchical clustering may yield worse results than K-means

Hierarchical Clustering Algorithm

1. Begin with n observations and a measure (such as Euclidean distance) of all the $\binom{n}{2} = n(n-1)/2$ pairwise dissimilarities. Treat *each* observation as its own cluster
2. For $i = n, n-1, \ldots, 2$:
    - Examine all pairwise inter-cluster dissimilarities among the i clusters and identify the pair of clusters that are least dissimilar (that is, most similar). Fuse these two clusters. Dissimilarity between these two clusters represnt the height in the dendrogram at which fusion should be placed
    - Computer the new pairwise inter-cluster dissimilarities among the remaining $i-1$ clusters

- One important aspect is defining the dissimilarity between groups, which is called the *linkage*
    - four common types of linkages - *complete*, *average*, *single*, and *centroid*
        * average and complete are generally preferred

Choice of Dissimilarity Measure

- may sometimes use *correlation-based distances* for dissimilarity measures
    - meaning we're much more interested in the similarities of features between each observations, rather than its pure observed value

Validating the Clusters

- we want to truly know if clustering finds true subgroups or as a result of *clustering the noise*

Other Considerations

- clustering methods generally not very robust to noise in data

Applied

PCA

We perform PCA on the US Arrests data, scaling the data:

Let's first see the variable names and their means:

```
apply(USArrests, 2, mean)
```

```
##   Murder  Assault UrbanPop     Rape
##    7.788  170.760   65.540   21.232
```

```
pr.out <- prcomp(USArrests, scale = T)
names(pr.out)
```

```
## [1] "sdev"     "rotation" "center"   "scale"    "x"
```

center and scale correspond to means and standard deviations of variables that were used for scaling preior to PCA

```
pr.out$center
```

```
##   Murder  Assault UrbanPop     Rape
##    7.788  170.760   65.540   21.232
```

```
pr.out$scale
```

```
##    Murder   Assault  UrbanPop      Rape
##  4.355510 83.337661 14.474763  9.366385
```

Rotation provides the principal loadings:

```
pr.out$rotation
```

```
##                 PC1        PC2        PC3         PC4
## Murder   -0.5358995  0.4181809 -0.3412327  0.64922780
## Assault  -0.5831836  0.1879856 -0.2681484 -0.74340748
## UrbanPop -0.2781909 -0.8728062 -0.3780158  0.13387773
## Rape     -0.5434321 -0.1673186  0.8177779  0.08902432
```

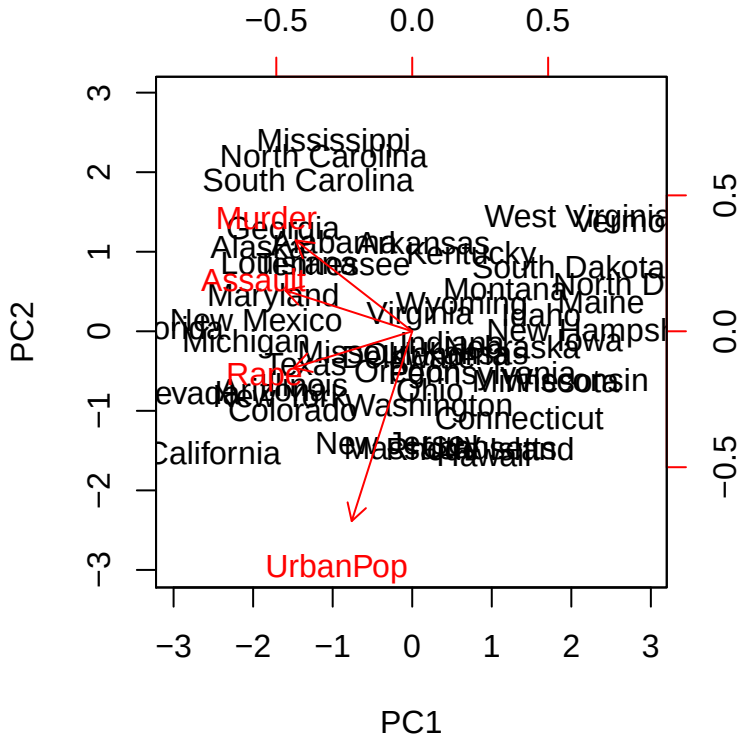
The matrix "x" has the principal component score vectors

```
head(pr.out$x)
```

```
##                  PC1        PC2         PC3          PC4
## Alabama    -0.9756604  1.1220012 -0.43980366  0.154696581
## Alaska     -1.9305379  1.0624269  2.01950027 -0.434175454
## Arizona    -1.7454429 -0.7384595  0.05423025 -0.826264240
## Arkansas    0.1399989  1.1085423  0.11342217 -0.180973554
## California -2.4986128 -1.5274267  0.59254100 -0.338559240
## Colorado   -1.4993407 -0.9776297  1.08400162  0.001450164
```

Let's plot the first two principal components:

```
biplot(pr.out, scale = 0)
```



We can see the standard deviation of each pcincipal component as follows:

```
pr.out$sdev
```

```
## [1] 1.5748783 0.9948694 0.5971291 0.4164494
```

Variance of each principal component can be computed:

```
pr.var <- pr.out$sdev^2
pr.var
```

```
## [1] 2.4802416 0.9897652 0.3565632 0.1734301
```

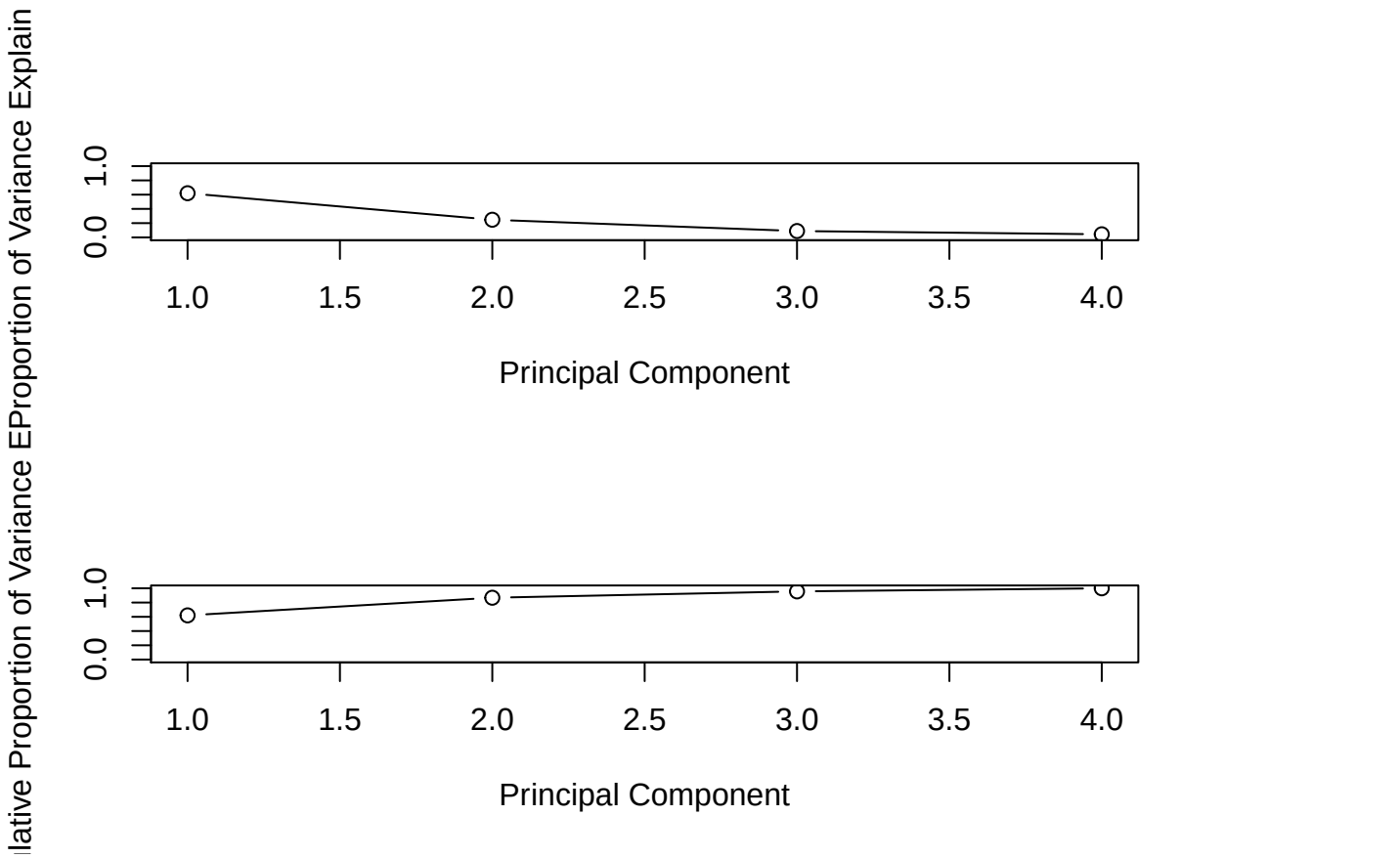
Then the proportion of variance explained by each principal component can be computed:

```
pve <- pr.var/sum(pr.var)
pve
```

```
## [1] 0.62006039 0.24744129 0.08914080 0.04335752
```

We can see that the first two components make up about 86% of all the variance. We can plot this PVE by each component, as well as the cumulative PVE:

```
par(mfrow = c(2,1))
plot(pve, xlab = "Principal Component", ylab = "Proportion of Variance Explained", ylim = c(0,1), type =
plot(cumsum(pve), xlab = "Principal Component", ylab = "Cumulative Proportion of Variance Expalined", yl
```



K Means Clustering

The function kmeans() performs K-means clustering. We first simulate an example where there are two true clusters in a data:

```r
set.seed(1)
x <- matrix(rnorm(50*2), ncol=2)
x[1:25,1] <- x[1:25,1] + 3
x[1:25,2] <- x[1:25,2] - 4
```

Now we perform K-means clustering with $K = 2$:

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

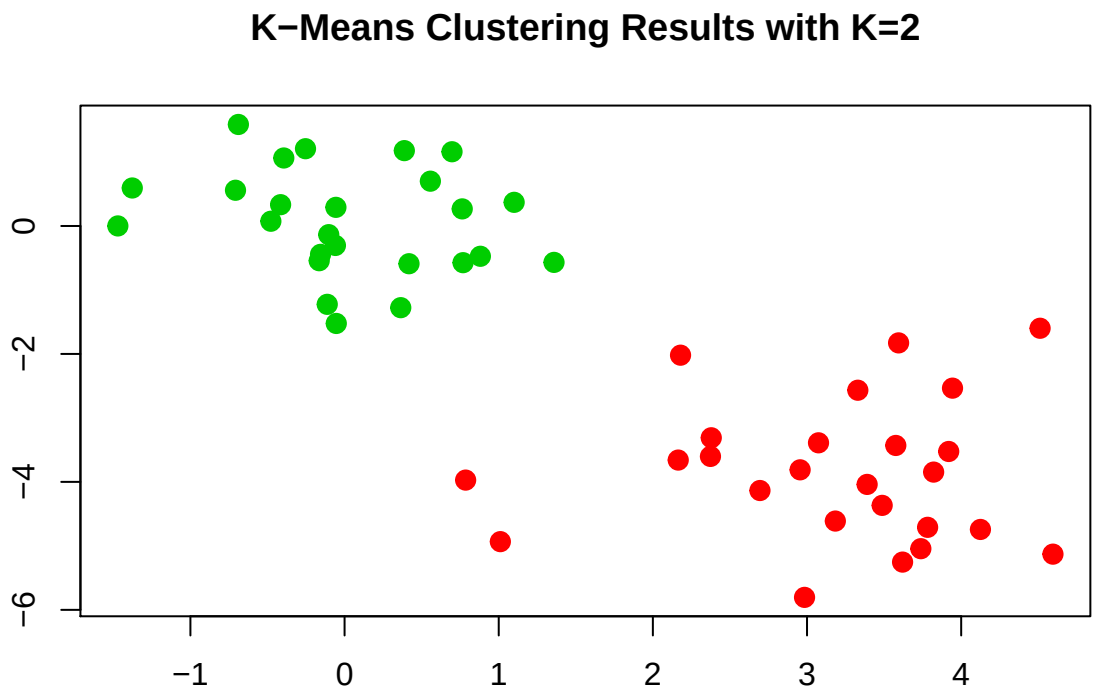
We can see the assignments of the cluster here:

```r
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 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
```

We can see that a total of 2 clusters are assigned even though we didn't specify any group info to kmeans(). We can plot this:

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



If there are more than two variables we can perform PCA and plot the first two principal component score vectors. Let's try $K = 3$ on the same example:

```r
set.seed(1)
km.out <- kmeans(x,3,nstart=20)
km.out
```

```
## K-means clustering with 3 clusters of sizes 5, 25, 20
##
## Cluster means:
##         [,1]        [,2]
## 1 3.51171162 -2.10935842
## 2 0.03223135  0.06924384
## 3 3.08290361 -4.26589906
##
## Clustering vector:
##  [1] 3 3 3 3 1 1 3 3 3 3 1 3 3 3 3 3 3 1 3 1 3 3 3 3 3 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
##
## Within cluster sum of squares by cluster:
## [1]  3.740301 28.534174 27.901401
##  (between_SS / total_SS =  84.7 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

nstart argument specifies the times the algorithm randomly assigns each point to a cluster. Let's compare nstart=1 and nstart = 20:

```r
set.seed(1)

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

```
## [1] 60.56297
```

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

```
## [1] 60.17588
```

The values denote the within-cluster sums of squares, and as we can see, there isn't a big difference between the two, but it's always recommeneded to use a large value of nstart (20 or 50)

Hierarchical Clustering

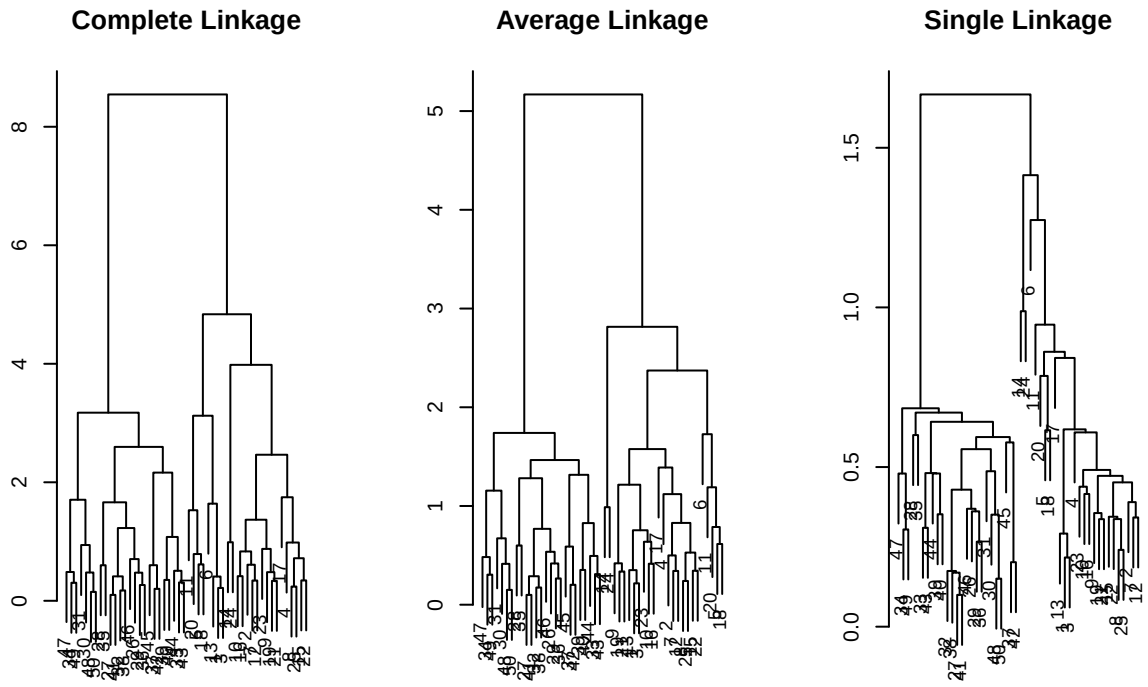
We start off by using various linkagees:

```r
hc.complete <- hclust(dist(x), method = "complete")
hc.average <-  hclust(dist(x), method = "average")
hc.single <-  hclust(dist(x), method = "single")
```

We can now plot the dendrograms:

```
par(mfrow = c(1,3))

plot(hc.complete, main = "Complete Linkage", xlab="", ylab= "", sub= "", cex = .9)
plot(hc.average, main = "Average Linkage", xlab="", ylab= "", sub= "", cex = .9)
plot(hc.single, main = "Single Linkage", xlab="", ylab= "", sub= "", cex = .9)
```



To determine the cluster labels for each observation associated with the given cut of dendrogram, we use the cutree() function:

```
cutree(hc.complete, 2)
```

```
##  [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
```

```
cutree(hc.average, 2)
```

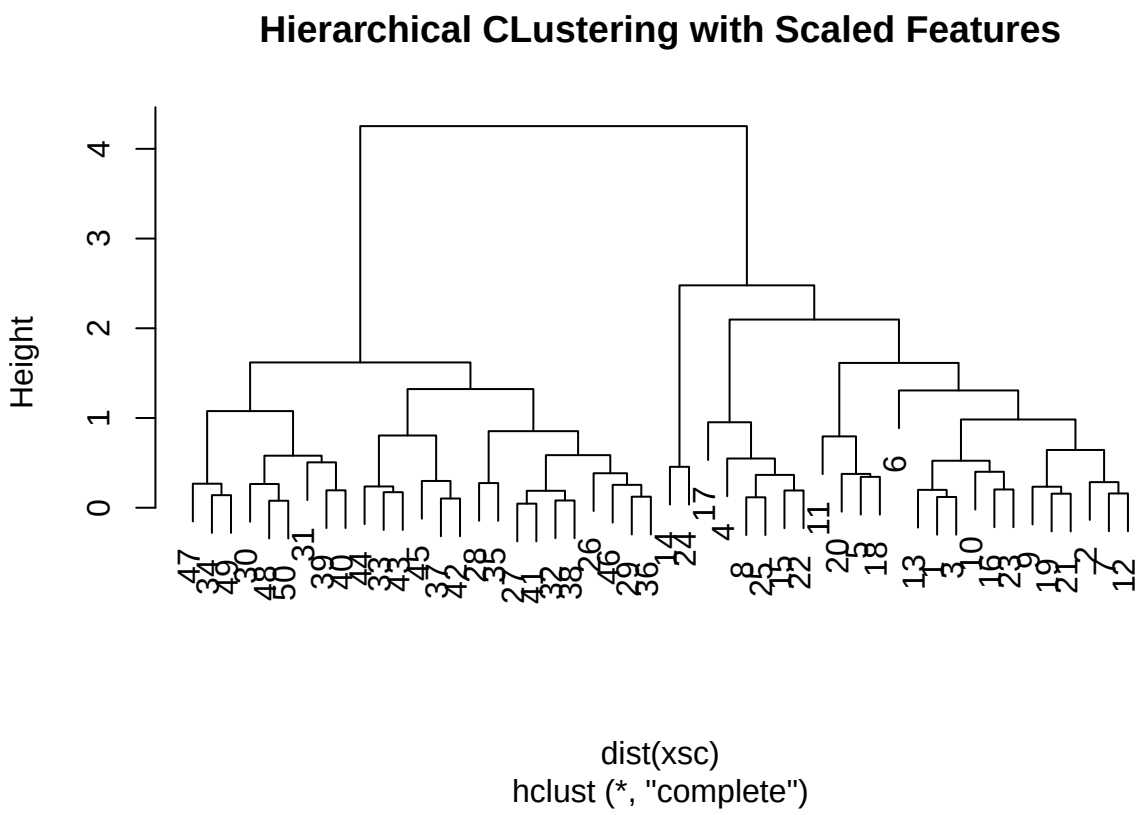
```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2
## [39] 2 2 2 2 2 2 2 2 2 2 2 2
```

```
cutree(hc.single, 2)
```

```
##  [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
```

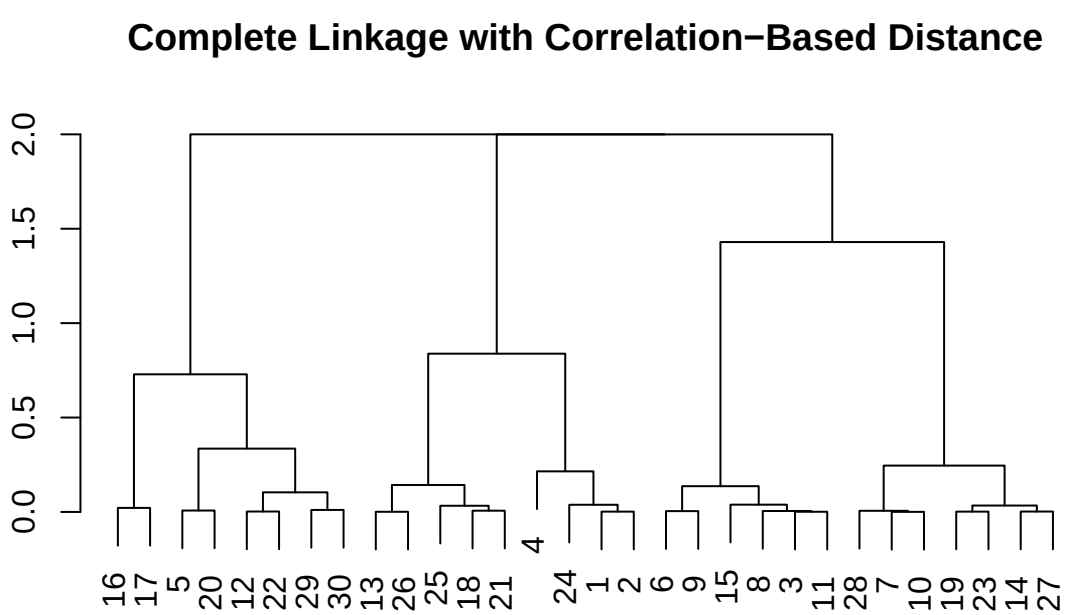
To scale the variables before performing hierarchical clustering, we use scale() function:

```
xsc <- scale(x)
plot(hclust(dist(xsc), method = "complete"), main = "Hierarchical CLustering with Scaled Features")
```



Correlation-based distance can be computed using as.dist(). This only makes sense for data with at least 3 features since absolute correlation between any two observation with measurements on two features is always 1. We cluster a three-dimensional dataset:

```
x <- matrix(rnorm(30*3), ncol = 3)
dd <- as.dist(1-cor(t(x)))
plot(hclust(dd, method = "complete"), main = "Complete Linkage with Correlation-Based Distance", xlab="
```


Complete Linkage with Correlation−Based Distance
0.0
0.5
1.0
1.5
2.0
16
17
5
20
12
22
29
30
13
26
25
18
21
4
24
1
2
6
9
15
8
3
11
28
7
10
19
23
14
27
hclust (*, "complete")