Assignment #8

Lindsay Brock, Gustavo Esparza, and Brian Schetzsle

October 24, 2019

ISLR 10.3 - Clustering Methods

This section will focus on *Clustering Methods*- a set of techniques utilized for finding subgroups (known as clusters) within a data set. The primary goal for Clustering is to divide observations in a data set into distinct groups that have similar qualities within the groups and have differing qualities outside of the groups. In discussing *similarity* and *difference*, we base these qualities on the data observations themselves (these qualities will be explored within this section). As we are attempting to find structure within the data provided, Clustering is an *unsupervised* method. The previously explored unsupervised method of Principal Component Analysis sought out a low-dimensional representation of the original data, whereas we are now focusing on dividing all of our data into homogeneous subgroups.

10.3.1: K-Means Clustering

There are two main techniques for Clustering: **K-Means Clustering** and **Hierarchical Clustering**; we will begin by exploring the former. K-Means Clustering seeks to group our data into a predetermined (K) amount of clusters. In order to establish this clustering method in clear mathematical language, we must meet the following properties:

1.) The union of our K clusters must represent the entire data set.
2.) The intersection of any two clusters must represent the empty set, indicating there is no overlap between any two clusters.

Now, we will discuss how each of our K clusters are developed. A key element in this process is to minimize the *within-cluster variation*, denoted as $W(C_k)$. This is simply the amount of variation found within a given cluster, and given this objective we are able to observe that the optimal k clusters for a given set of data must achieve the following constraint:

$$\underset{C_1,\dots,C_k}{\text{minimize}}\left\{\sum_{k=1}^{K} W(C_k)\right\}.$$

That is, the total within-cluster variance summed over all k clusters is minimized. This goal is intuitive, but we need a method of measuring within-cluster variance in order to fully visualize this objective.

A useful and common choice for measuring this variance is using *Euclidean Distance* (squared distance in p-dimensional space) defined by:

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

This equation is essentially summing the squared distance between between all i observations in the k^{th} cluster over the entire p-dimensional feature space. Once this sum is obtained, we find semi-average variance by dividing by the total amount of observations in the cluster: $|C_k|$.

Provided our measure of variance, we can now introduce an algorithm for constructing the optimal K clusters. Before jumping into our algorithm, we must briefly introduce the concept of **cluster centroids**. Thus, consider the following equality:

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

This identity uses the dividing cardinality ($|C_k|$) to now represent the feature means for a given k cluster, $\bar{x}_{kj}$. The cluster centroid is essentially a vector that contains all p feature means for the observations in the k^{th} cluster. We can also observe the multiplicative factor of 2 given by double counting the summation over all i observations in our cluster C_k. Using this final equality, here is the algorithm for k-means clustering:

```
K-Means Clustering Algorithm:

1.) Randomly assign an initial cluster group (from 1 to K) to each observation.

2.) Iterate the following process until all observations are no longer assigned to a new cluster:
      (a) For each of the K clusters, compute the respective cluster centroid.
      (b) Assign each observation to the cluster whose centroid is closest (in terms of euclidean distance).
```

As this process is repeated, we find our observations being assigned to clusters that minimize this distance (variance) until the cluster reassignment does not change. This optimum assignment results in the K-Means Clustering that minimizes overall within-cluster variance. It is important to note that the algorithm begins by randomly assigning observations to a cluster, thus differing K-means clusters will be produced as the initial assignment is changed (IE the algorithm finds a local optimal value rather than a global value). In order to ensure that truly optimal clusters are defined, it is recommended to run this algorithm under various initial cluster assignments. The figure below provides the results for six different K-Means Algorithm initial configurations:

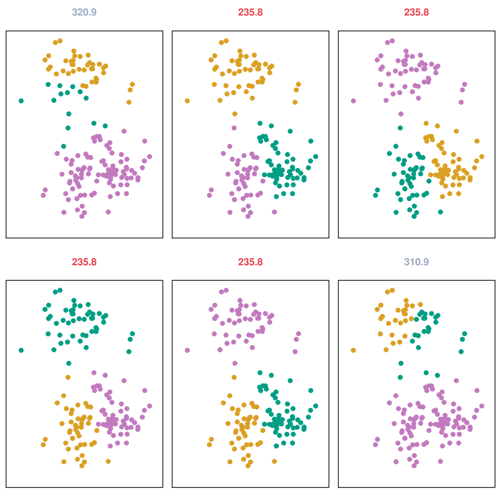


Figure 1: K-Means Algorithm Results

We can see that the six configurations provided different clusters of size K = 3, but four of these different clusters provided the same optimal value of 235.8. Thus, any of these four options are suitable as an optimal clustering.

We have spent some time developing a method for defining our clusters in a variance reducing manner, but there remains the question of selecting the size of K. This is an important factor to consider, and will be discussed later in this section.

10.3.2: Hierarchical Clustering

Hierarchical Clustering is an alternative method for clustering where the amount of total clusters does not need to be specified. This method of clustering provides a more visual perspective that allows for a great deal of flexibility in constructing desired clusters.

The primary tool for Hierarchical Clustering is the **dendrogram**, a graphical upside-down tree where each leaf represent an observation from the data set and branches are then constructed by grouping observations that are similar to one another.

This section will first discuss how to interpret a dendrogram, then we will focus on how to actually construct one. It is important to note that branches developed at the bottom of our tree have observations (leaves) that are more in common with each other than the branches developed at the top of our tree. More specifically, the height at which the branch was formed (known as the point of fusion for the leaves) indicates how different the observations are from each other.

Interpretation of dendrograms can often be initially confusing. Therefore, we will illustrate an example of a dendrogram before continuing with our discussion:

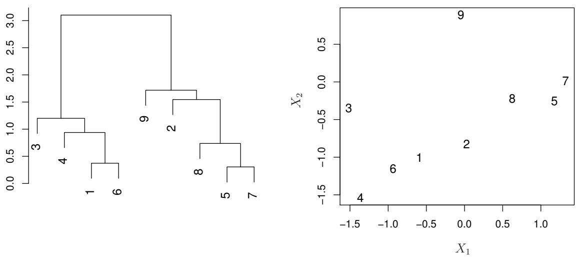


Figure 2: dendrogram Interpretation

On the right, we can observe nine observations from a two-dimensional space. Once again considering Euclidean distance as a measure of dissimilarity, we can see that observation **9** is quite different from all other observations. In addition, the pairs **(7,5)** and **(1,6)** are quite close to each other in the 2-dimensional space, and are therefore very similar. Taking these observations and constructing their respective dendrogram provides the tree on the left. As expected, the leaves that were the first to fuse into branches are the similar pairs **(7,5)** and **(1,6)**. It is interesting to note that observations **9** and **2** were quite distant on the original plot, but now appear quite close on the provided dendrogram. Although they are indeed close in proximity, it is critical to make note of the fact that the branch consisting only of observation **9** is shown as being fused with a chain of previous branches consisting of observations **2,8,5,** and **7**. Furthermore, this fusion occurs near the top of our dendrogram - implying that the two branches are quite dissimilar from one another. It is tempting to inspect the dendrogram and conclude that leaves/branches that are located in close proximity are similar, but we must always consider the vertical height at which the proximity occurs.

Now that the benefits of observing similarities and differences between observations on a dendrogram have been discussed, we can dive into a discussion regarding how to construct clusters from a given dendrogram. Much of the previous discussion regarded the vertical height of our dendrogram, and cluster construction follows in that manner. Specifically, we must decide a height to make a horizontal cutoff line that will decide how the clusters are defined. In effect, the distinct sets of data found *below* the cutoff represent the clusters themselves.

Here is another example to illustrate how clusters are formed from a dendrogram:

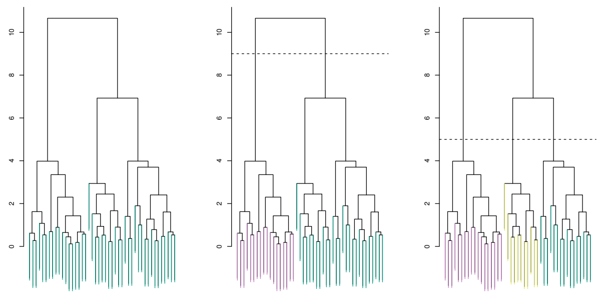


Figure 3: Constructing Clusters From a dendrogram

On the far-most left, we can observe a dendrogram constructed from 45 observations. By a simple inspection, there are several branches constructed at many different heights. The center plot once again displays our dendrogram, but we have now defined a horizontal cut off at a height of about 9. This cutoff crosses two distinct branch lines that then provide two distinct clusters consisting of all observations found beneath each branch line. It is apparent that the clusters are not equal in size, but do in fact construct clusters that are mutually exclusive and collectively represent the entire data set. Now focusing on the rightmost plot, we declare a new cutoff at a height of 5. Now, we have crossed three separate branch lines that produce three separate clusters that still maintain the general properties of valid clusters.

This illustration has been beneficial in displaying how one dendrogram can result in various cluster constructions of many sizes. In practical instances dendrograms are usually used to create clusters by approximating a cutoff height where a suitable amount of clusters are found to be appropriate for the given data. Therefore, the context of the data can be almost as significant to the amount of clusters as the dendrogram itself.

Thus far, examples of hierarchical clusters have been displayed in terms of general dendrogram structure and specific construction via cutoffs. Now, we will examine a useful algorithm that produces the required dendrogram structure. The algorithm is quite simple, and is defined as follows:

Hierarchical Clustering Algorithm

1.) Begin with all n observations and a measure of the total n(n-1)/2 pairwise dissimilarities, usually represented by the euclidean distance between observations. Each observation will act as their own initial cluster.

2.) For i = n, n-1, . . . , 2:

(a) Examine all pairwise inner-cluster dissimilarities among the i clusters and identify the most similar cluster pair. Proceed to fuse this pair and define the fusion height of the dendrogram as the found measure of dissimilarity.

(b) Repeat step (a) for the remaining $i-1$ clusters.

This algorithm is quite straight forward, but remains a bit ambiguous in terms of how to proceed computing dissimilarity once clusters consisting of multiple observations are formed after the initial fusion step. This is where the concept of **linkage** comes into effect. Linkage simply defines the dissimilarity between two groups of observations, and comes in a few variations which are displayed in the table below:

Common Linkages:

Complete:
Compute all pairwise dissimilarities between observations in Cluster A and Cluster B. The **largest** pairwise dissimilarity will define the general dissimilarity between the two clusters.

Single:
Compute all pairwise dissimilarities between observations in Cluster A and Cluster B. The **smallest** pairwise dissimilarity will define the general dissimilarity between the two clusters.

Average:
Compute all pairwise dissimilarities between observations in Cluster A and Cluster B. The **average** pairwise dissimilarity will define the general dissimilarity between the two clusters.

Centroid:
Compute dissimilarity between the centroids (vector of length p representing the feature dissimilarities) for cluster A and cluster B.

Each of these linkage types have their benefits and drawbacks, and we will illustrate the resulting dendrograms for the first three (most common) linkage types in the figure below:

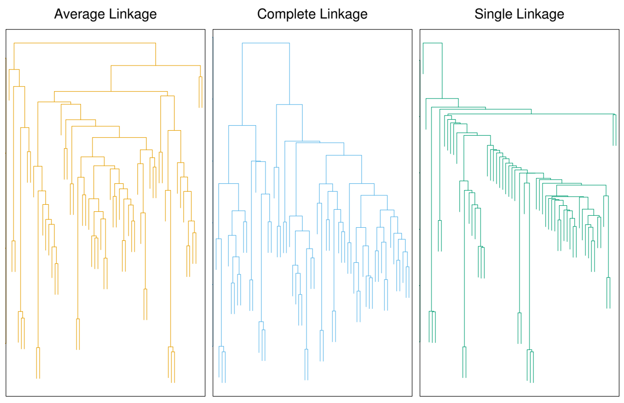


Figure 4: dendrograms By Linkage Type

We can see that Complete Linkage provides the most evenly spread out dendrogram with branches consisting of multiple observations, while Average Linkage is a bit more sparse in terms of branches. Single linkage creates many branches with a single observation, which may not be ideal when constructing clusters to group our data into distinct groups.

Dissimilarity Measures

So far in this section, we have relied on Euclidean distance as our measure of similarity between observations. However, we may wish to choose an alternative measure when the context of our data possess unique qualities. For example, **correlation-based distance** considers two observations to be similar if their features are highly correlated, but not necessarily close in terms of euclidean distance. A useful example for this instance of dissimilarity measure is the tracking of online shopping habits. When attempting to group subjects by their shopping habits, we would not necessarily want to split them apart by how often they shopped (Euclidean Distance) but rather by the similarity of items purchased (Correlation-based).

Beyond considering which measure of dissimilarity to use, we should also consider whether each variable for the observations are scaled prior to performing Hierarchical Clustering. Returning to the example of Online shopping habits, we can consider two separate variables - purchases of socks and purchases of computers. It is obvious that the price of these two items will be widely different, and we also tend to purchase socks much more often than computers. Thus, we may want to consider scaling each variable in the feature space in order to have equal importance for each of these variables when clustering our observations. This example helps establish how important the context of our data is when choosing how to measure dissimilarity.

10.3.3: Practical Issues in Clustering

As a closing point for this discussion of Clustering methods, we will briefly mention some issues that should be considered before effectively performing a clustering process. We have attempted to provide a holistic set of choices for clustering that ensure that many different observational scenarios are able to be grouped, but we have yet to determine if our clustering is actually grouping subsets of our data or just grouping random *noise* found within the data. This will require some form of a *validation* process that can determine statistical significance via a p-value. This text implies that a cluster significance test has no universally accepted methodology, but is currently being established in the Statistics community.

One final concern regarding Clustering is the idea of **outliers** being present within a data set. By definition of clustering, each observation will be forced into a cluster. In addition, this outlier will then effect the general properties (variance) of the associated cluster. This is undesirable since we strive to minimize the within cluster variance, but there exist **mixture models** that take into account the general presence of outliers, which will be discussed in a following section regarding variants of K-Means Clustering.

ESL 14.3 - Cluster Analysis

The goal of cluster analysis is to find natural groupings of observations such that members within a group are more closely related to each other than with members of other groups. Cluster analysis can also be used for hypothesis testing, to test whether a data set breaks into a certain number of clusters. How we determine the similarity and difference between points is central to the task of forming clusters. These concerns have to be informed by the subject matter surrounding the data; there are no general rules applicable to all data sets and determining dissimilarity is highly subjective.

14.3.1: Proximity Matrices

Proximity matrices catalog how similar or dissimilar each data point is from every other data point. If you have n observations then you will need a matrix of size $n \times n$ for all pairs of data. Most clustering algorithms assume a *dissimilarities matrix* D that catalogs how dissimilar or far apart each pair of data are. D is usually assumed to be symmetric, so that the distance between x_{ij} and $x_{i'j}$ is the same as the distance between $x_{i'j}$ and x_{ij}, and the diagonal entries are 0 because there is no distance between a point and itself. D can easily be made to be symmetric by replacing it with $(D + D^T)/2$.

14.3.2: Dissimilarities Based on Attributes

D is constructed by determining how different each data point is from every other data point; because each data point is a collection of p attributes, we have to find a way to aggregate the differences between each of the attributes of any pair of data. We construct a function $d_j(x_{ij}, x_{i'j})$ that will return the difference between point x_i and $x_{i'}$ for the j^{th} attribute and then

$$D(x_i, x_{i'}) = \sum d_j(x_{ij}, x_{i'j})$$

The most common dissimilarity function is squared distance, $d_j(x_{ij}, x_{i'j}) = (x_{ij} - xi'j)^2$. Note that distance isn't well-defined if the j^{th} attribute is categorical (ordinal or nominal). It's also sometimes desirable to give some attributes greater importance in determining the total difference between points by weighting the sum.

You can also use absolute difference as the measure of dissimilarity between points for quantitative attributes, so $d_j(x_{ij}, x_{i'j}) = |x_{ij} - xi'j|$ or even some monotonic function of this absolute difference. You can also use the correlation between data points across all their quantitative attributes, but it can be shown that this is equivalent to using a function of the squared difference.

For ordinal data that has M categories in some order, what is typically done is replacing the M original values with $\frac{i-\frac{1}{2}}{M}, i = 1, ..., M$ and then treating that value as a quantitative variable. This new attribute takes on values in $(0, 1)$ and has the limitation that it treats the difference between consecutive levels as fixed.

For categorical variables, the difference between categories must be defined explicitly. This is typically done with an $M \times M$ matrix (if the j^{th} attribute is categorical with M distinct values). The diagonal elements are the difference between two objects in the same category, so should be 0, and the off-diagonal elements are the difference between categories; and easy and common choice is 1 but you can emphasize difference between some categories with a larger number. Like, the difference between dog and coyote could be 1 but the difference between dog and salmon could be 20.

14.3.3: Object Dissimilarity

Once we are able to measure the difference between points for all the p attributes, we can combine them with a weighted sum to get the elements of D.

$$D(x_i, x_{i'}) = \sum w_j * d_j(x_{ij}, x_{i'j})$$

By setting $w_j = \frac{1}{\bar{d}_j}$, where $\bar{d}_j$ is the average difference between all points for the j^{th} attribute, we can make every attribute equal in importance to the overall dissimilarity between points. This is not always desirable, for instance when an attribute makes clustering of observations easy. It's better to give higher weights to attributes that are more relevant to the resulting clusters.

In real-life scenarios there is often missing values for some attributes. You can either drop those pairs of observations if an attribute is missing or fill it in with the mean or median from the data. Another way that the book doesn't touch on but which we learned about it computational stats is using the EM algorithm to iteratively fill in missing data. Categorical attributes can simply use "missing" as a new category.

14.3.4: Clustering Algorithms

The book introduces three type of algorithms used to define clusters in data. *Combinatorial algorithms* seek clusters directly from the data without making any assumptions on the distribution of the data. *Mixture modeling* assumes the data are iid samples from some population that is modeled as a mixture of several sub-populations. The components of these sub-populations are then found using maximum likelihood or corresponding Bayesian methods. *Mode seekers* take a non-parametric approach and seek out the most likely modes for clusters and then define the boundaries of each cluster as the region of the parameter space closest to each cluster's mode.

14.3.5: Combinatorial Algorithms

Combinatorial algorithms try to find a mapping, or *encoder* $k = C(i)$, that assigns each data point x_i to a cluster k. The number of clusters is determined before the algorithm starts. The algorithm uses the dissimilarity matrix D and minimizes some "loss" function, a measure of how well the mapping assigned each point to a cluster. One common way of finding the best mapping is by minimizing the "scatter" of data within each cluster. This within cluster scatter is defined as

$$W(C) = \frac{1}{2} \sum_{k=1}^{K} \sum_{C(i)=k} \sum_{C(i')=k} d(x_i, x_{i'})$$

In words, that is the sum of all distances of pairs of points in each cluster k, summed over all clusters. The $\frac{1}{2}$ is because each pair is double-counted. Minimizing the scatter within clusters is the same as maximizing the scatter between clusters, because the total distance of all data points is constant. To go through all possible combinations of mappings is only feasible for small data sets. Algorithms have been developed to find a good mapping without iterating through all possibilities; these start at some initial mapping and make small changes that improve the loss function until no further improvements can be made.

14.3.6: K-means

One popular method for finding the mapping that minimizes within-cluster scatter is the K-means algorithm. In order to use it, the predictors must all be quantitative and the squared euclidean distance is used for the dissimilarity function $d_j(x_{ij}, x_{i'j})$. Thus the algorithm seeks to minimize

$$W(C) = \frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\sum_{C(i')=k}||x_i - x_i'||^2$$

$$= \sum_{k=1}^{K} N_k \sum_{C(i)=k}||x_i - \bar{x}_i||^2$$

which is the total distance of each point from its cluster's mean.

The K-means algorithm is quite simple; it starts with some mapping C and calculates the mean for each cluster. It then reassigns each point to the cluster with the closest mean, then recalculates the means until the reassignments don't change. This method can get stuck in sub-optimal cluster assignments so the book suggests two possible improvements: After the algorithm has converged, check to make sure that the within-group scatter isn't improved by switching any single data point to another group; also, run the algorithm many times with different initial mappings C and then choose from among all the final mappings that one with smallest within-group scatter.

14.3.7: Gaussian Mixtures as Soft K-means Clustering

An application of the EM algorithm is to find the probability that a data point was generated from a particular Gaussian component in a mixture of several components. In the E-step, the probability of belonging to each component is calculated based on the current means of those components. In the M-step, the means of the components are recomputed based on the probabilities of that data points belonging to each component. The number of components and the covariance matrix of those components is determined beforehand; if we say there are K mixture components and the covariance matrix is $\sigma\boldsymbol{I}$ then we will get probabilistic cluster assignments using EM as opposed to deterministic cluster assignments with K-means. As $\sigma \to 0$, EM becomes equivalent to K-means.

14.3.8: Example - Human Tumor Microarray Data

The book concludes the topic of K-means clustering with an example of applying this method to human tumor microarray data. The data consists of 64 samples each with 6830 gene expressions measurements (so n=64 and p=6830). Each sample has a label for the type of cancer the patient it was drawn from had. The goal is to try and cluster the 64 samples based on similarity in gene expressions and see if the labels fall into the same clusters *post hoc*. We would hope to find some similarities in gene expressions for cancers of the same type.

The number of clusters K must be determined before-hand, so the book iterates through $K = 1, ..., 10$ and compares the total within-cluster scatter of the final clustering assignment for each K. There was no K that stood out, so the book just looked at $K = 3$ and found that many of the cancers did organize themselves into one of the three clusters. Some problems with K-means clustering in this application that the book highlights are that there is no explicit relationship between elements in the same cluster and the cluster assignments can change arbitrarily as K changes; two items in the same cluster when $K = 4$ will not necessarily be in the same cluster when $K = 3$.

14.3.9: Vector Quantization

The K-means clustering algorithm is also useful in image and signal compression, specifically *vector quantization* (VQ). When storing an image on a computer, it can be stored with varying degrees of clarity depending on how much storage space you would like to use (more pixels with higher quality requiring more storage which can be compressed down in size but will effect the quality of the picture). Each pixel is a number corresponding to the color it should represent in the picture. VQ in this situation decomposes the original high quality 1024x1024 pixel image into 512x512 small blocks of 2x2 pixels. Since each block of four holds four numbers, it is a vector in $\mathbb{R}^4$ and this is where the K-means clustering algorithm is run. As the value of K (number of clusters) decreases, so does the clarity of the picture. Each of the 512x512 *points* (blocks/clusters) is approximated by its closest *codeword*, or cluster centroid, which are collected into a *codebook*. This method is appropriately named the *encoding* step.

To create the compressed image, the codebook of Kx4 real numbers must be created and each 2x2 block must be given the codebook entry that approximates it. This step of approximating the original image and creating a compressed version using the centroids is called the *decoding* step.

Now that we know how it works, it is clear that VQ ends up being an good method of approximation to compress photographs because in higher quality photos, groups of pixels end up looking the same. Since they are close if not exactly the same color, we can create one block to represent a group with multiple pointers to that block, thus cutting down the required storage space for a picture. The method described here is called a *lossy* compression. As it is technically *distorting* (degrading or reducing) the original image, it would be important to measure this distortion which can be done using the mean squared error and can be visualized using a rate/distortion curve. This is not the only method of VQ currently used today and it may be of use to research the other generalizations that have been developed, like the tree-structured VQ.

14.3.10: K-medoids

There are a few restrictions required for the K-means algorithm such as squaring the Euclidean distance (which gives the most influence to the largest distances in turn giving outliers excessive influence) and not allowing qualitative variables which can be removed at the expense of computation. This algorithm can be altered to use subjective dissimilarities $D(x_i, x_{i'})$ by replacing the step that minimizes the squared Euclidean with one that optimizes it with respect to $\{m_1, \ldots, m_K\}$. The first step finds the observation in a cluster assignment C that minimizes the total distance to the other observations of that cluster,

$$i_k^* =^{argmin}_{\{i:C(i)=k\}} \sum_{C(i')=k} D(x_i, x_{i'}),$$

and sets this observation as the estimate of the cluster center m_k, for $k = 1, \ldots, K$. The i in the equation above represents the index of that observation. The second step assigns the original observations to one of the current closest cluster centers (the estimates $\{m_1, \ldots, m_K\}$) to minimize the total error, $C(i) =^{argmin}_{1\leq k\leq K} D(x_i, m_k)$. Finally, this is done repeatedly until cluster centers no longer change. This algorithm can be used for both attribute and proximity matrix data.

14.3.11: Practical Issues

The discussion of both K methods above brings us to the important topic of selecting the number of clusters and the initial starting value for both algorithms. Picking starting values can be done many ways such as simply selecting random values or, with a more calculated approach, using forward stepwise assignment. Selecting the number of clusters depends entirely on the situation as there is usually a predetermined number of clusters, or areas we wish to group, when the data is presented. The book gives an excellent example of sales people splitting their customer database. With K many sales people, they may need to split up the data into K groups where the customers in each group are as similar as possible. This can be extended to the use of descriptive statistical analysis where we may wish to see if the data falls into naturally forming groups of similarities. It may not be as clear as the sales example, but the number of clusters would need to be estimated based on what we know of the data. This can be done by looking at the cluster dissimilarity W_K corresponding to a selected number of clusters. For each choice of K, a W is calculated and they are plotted against one another. Looking for the "kink" (sharp decrease) in the graph and its corresponding K value allows us to determine the number of clusters. With some research there are various other methods that can be used, such as the *Gap statistic*.

14.3.12: Hierarchical Clustering

Another clustering method that does not require us to choose a specific number of clusters or appropriate starting values is hierarchical clustering, which alternately calls for user specified dissimilarity measures between disjoint groups of observations. These measures are based on pairwise differences between the observations in two of the groups. This method receives its name because it produces hierarchical levels where each of the lower level groups are merged in the next level up until they become one group of all observations at the top of the hierarchy. The lowest hierarchy level contains a single observation per group and because of this, will have $N-1$ levels. This method includes two types based on the direction of the hierarchy, *glomerative* which merges clusters from the bottom to top and *divisive* which divides clusters from top to bottom. Agglomeration methods start at the bottom and merge a single pair of clusters per level (selecting the pair of clusters with the smallest intergroup dissimilarity) until it reaches a single group at the top. Divisive methods work in the opposite direction and split a single cluster at each level, selecting the cluster that will yield the largest between-group dissimilarity.

Since each level of the hierarchy produces a different number of disjoint clusters, this is where user selection comes in to play as they must decide on the appropriate level that represents a natural grouping of the observations. To aid in this selection the previously mentioned Gap statistic can also be used in this scenario.

These two methods can be visualized using a rooted binary tree where the root node represents the entire data set, the following *parent* nodes are the different levels and groups (each node split into *daughter nodes*), and the terminal nodes at the very bottom are all of the individual observations (*singleton clusters*). The heights of the nodes are proportional to the intergroup dissimilarity of the two daughter nodes, getting smaller and smaller all the way down to the terminal nodes. Most often, this dissimilarity is *monotone*, or uniform, at each level and it increases up the node levels. The hierarchical tree is called a *dendrogram* as seen in the figure below and it is a major reason for the popularity of this clustering method as it an excellent tool for interpretation of the data itself.

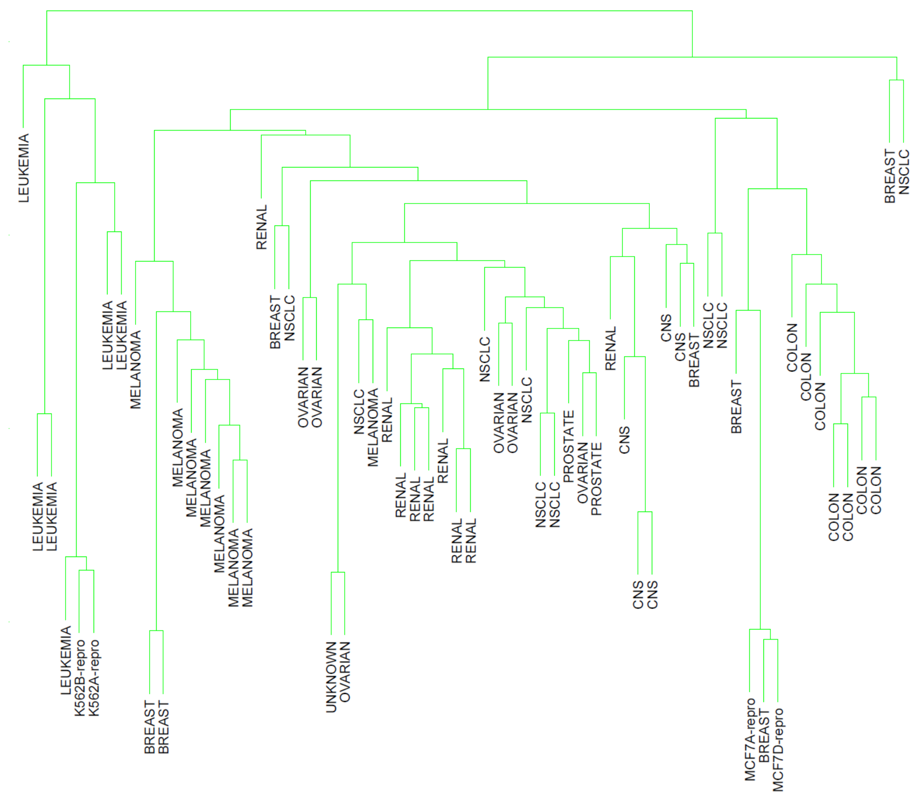


FIGURE 14.12. *Dendrogram from agglomerative hierarchical clustering with average linkage to the human tumor microarray data.*

It is also useful to cut the dendrogram horizontally at certain heights. This is the same as stopping the hierarchical clustering algorithm at a certain iteration based on user defined intergroup dissimilarity criteria. This provides interesting insight into the data as it partitions it into specific disjoint clusters. Groups that are merged higher in the tree (say above the cut off point) are candidates for natural clusters. This may also generate a clustering hierarchy (not to be confused with hierarchical clustering) which is clusters nested within clusters.

Although it is an extremely useful depiction of the hierarchical methods, using it to interpret the data should be done with caution. Different hierarchical methods will produce different trees just as small changes in the data can, as well. There are also situations where a hierarchical structure is produced when there is in fact no natural grouping of the data to begin with. How well the hierarchy represents the data can be assessed by the *cophenetic correlation coefficient*, which is the correlation between the $N(N-1)/2$ pairwise observation dissimilarities ($d_{ii'}$) from the data and the computed *cophenetic* dissimilarities ($C_{ii'}$) from the dendrogram. The cophenetic dissimilarity of the two observations (i, i') is the intergroup dissimilarity where they are first joined together to form a cluster. If that correlation is not high, or in other words the cophenetic dissimilarities do not match the pairwise observation dissimilarities, it does not prove the tree to be a good representation of that data and should not be used to directly describe it. Instead these trees should be used to represent the clustering structure determined by the respective algorithm used *on* the data.

Concentrating on agglomerative clustering where there is one less cluster on the way to the top, a measure of dissimilarity must be determined between two clusters. If G and H represent each cluster, the dissimilarity $d(G, H)$ between them is calculated using the set of pairwise dissimilarities $d_{ii'}$ where i is G and i' is H. One form of agglomerative clustering called *single linkage* (aka *nearest-neighbor*) says the intergroup dissimilarity measure between G and H is actually that of the most similar pair of the groups, rather the closest.

$$d_{SL}(G, H) = ^{min}_{i \in G, i' \in H}\, d_{ii'}$$

Another form is called the *complete linkage* (*furthest-neighbor*) and says just the opposite, that the intergroup dissimilarity measure is that of the least similar pair, or the furthest.

$$d_{CL}(G, H) =_{i \in G, i' \in H}^{max} d_{ii'}$$

Group average clustering lands right in the middle, using the average dissimilarity of the two groups, where N_G and N_H in the equation below are the number of observations in each group.

$$d_{GA}(G, H) = \frac{1}{N_G N_H} \sum_{i \in G} \sum_{i' \in H} d_{ii'}$$

The plots below represent the effect each of these measures of dissimilarity have on the shape of the trees. The three methods will end up producing the same results if the data dissimilarities naturally have an apparent and clear clustering tendency, with groups that are extremely similar are well separated.

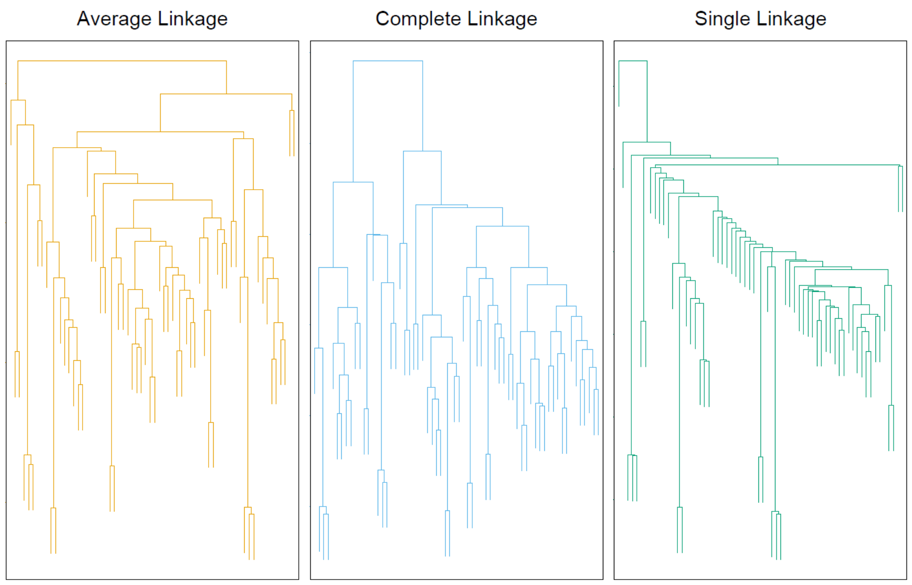


FIGURE 14.13. *Dendrograms from agglomerative hierarchical clustering of human tumor microarray data.*

A drawback of single linkage is the *chaining* phenomenon it produces, as seen in the dendrogram on the far right. As its only requirement to consider two clusters close together is that a single dissimilarity of a pair of observations be small, it is predisposed to combine observations that may be quite largely dissimilar by linking a series of closer observations together just like a chain. This gives the cluster a large *diameter* (largest dissimilarity among members of a cluster) and can result in a violation of the compactness (observation similarity) property of each cluster. On the other hand, complete linkage can produce extremely compact clusters as its only restriction is that two clusters are considered close if the dissimilarity of the farthest pair is small. This gives its clusters extremely small diameters and violates the closeness property of clusters, where observations in one cluster are closer to those in another cluster than they are to some members of their own cluster. Most of the time, group average clustering settles as a good compromise to both methods by creating *relatively* compact clusters that are *relatively* far apart.

When a study requires a relatively small number of clusters to be evaluated, divisive clustering obviously has the advantage over agglomerative methods as it would be easier to split data only a few times versus merging the data multiple times. It is used by iteratively splitting the data using any one of the combinatorial methods, i.e. K-means or K-moids with K=2. A downside of this method is that each step is dependent on the starting configuration given by the previous split. It may also generate a splitting sequence that is not monotonistic which is needed for dendrograms. To avoid these issues a specific algorithm was suggested by Mac- naughton Smith et al. in 1965. It says to begin by placing all observations in one cluster (call it G) and selecting the one observation that has the largest average dissimilarity from all the other observations. This one alone creates a second cluster called H. In the next iteration, the observation in G whose average distance from those in H, minus the average distance from the remaining observations in G is the largest, is then moved to H. This loop continues until the difference becomes negative which signals the loop to stop. The result is the second level of the hierarchy which is a split of the original data into two groups, G and H. This can be done to create even more levels by performing this algorithmic split on one of the clusters in the previous level. To choose which cluster to split, a candidate could be the one with the largest diameter (suggested by Kaufman and Rousseeuw in 1990) or the one with the largest average dissimilarity ($\bar{d}_G = \frac{1}{N_G^2} \sum_{i \in G} \sum_{i' \in G} d_{ii'}$). In these cases, either all clusters become singletons at the lowest level or all members of each cluster have a dissimilarity value of zero to determine the final level of the hierarchy.

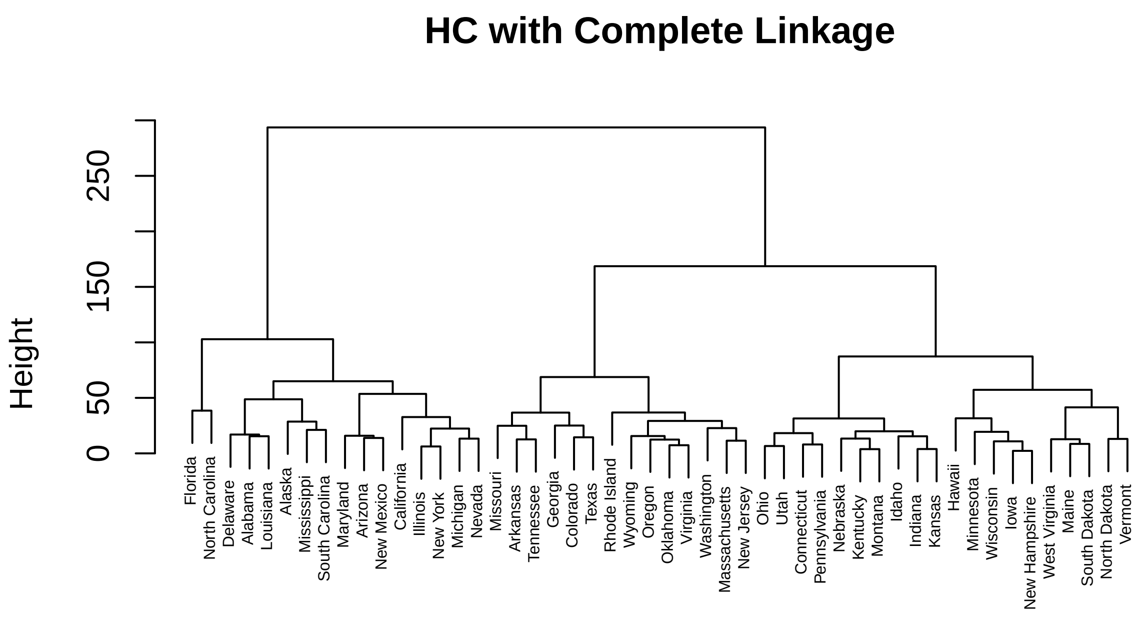
ISLR Problems

Consider the *USArrests* data. We will now perform hierarchical clustering on the states.

10.9 - a)

Using hierarchical clustering with complete linkage and Euclidean distance, cluster the states.

```
data = USArrests
attach(data)

#HC using complete linkage and dissimilarity measure of Euclidean dist.
#dist() is used to compute nxn inter-observation Euclidean distance matrix
hc.complete = hclust(dist(data), method = "complete")

#dendrograms
plot(hc.complete, main = "HC with Complete Linkage", xlab = "", sub = "", cex = 0.5)
```



Above is a dendrogram of hierarchical clustering using complete linkage where the root node represents the data as one group of all the states and each node represents a split in the previous group until the terminal nodes at the end are split into individual states.

10.9 - b)

Cut the dendrogram at a height that results in three distinct clusters. Which states belong to which clusters?

```r
cut = cutree(hc.complete, 3)
cluster1 = cut[which(cut == '1')]
cluster2 = cut[which(cut == '2')]
cluster3 = cut[which(cut == '3')]
```

Cutting the dendrogram at a height that produces three clusters assigns each state to the following groups:

Cluster 1

```
 [1] "Alabama"        "Alaska"         "Arizona"        "California"
 [5] "Delaware"       "Florida"        "Illinois"       "Louisiana"
 [9] "Maryland"       "Michigan"       "Mississippi"    "Nevada"
[13] "New Mexico"     "New York"       "North Carolina" "South Carolina"
```

Cluster 2

```
 [1] "Arkansas"      "Colorado"      "Georgia"       "Massachusetts"
 [5] "Missouri"      "New Jersey"    "Oklahoma"      "Oregon"
 [9] "Rhode Island"  "Tennessee"     "Texas"         "Virginia"
[13] "Washington"    "Wyoming"
```

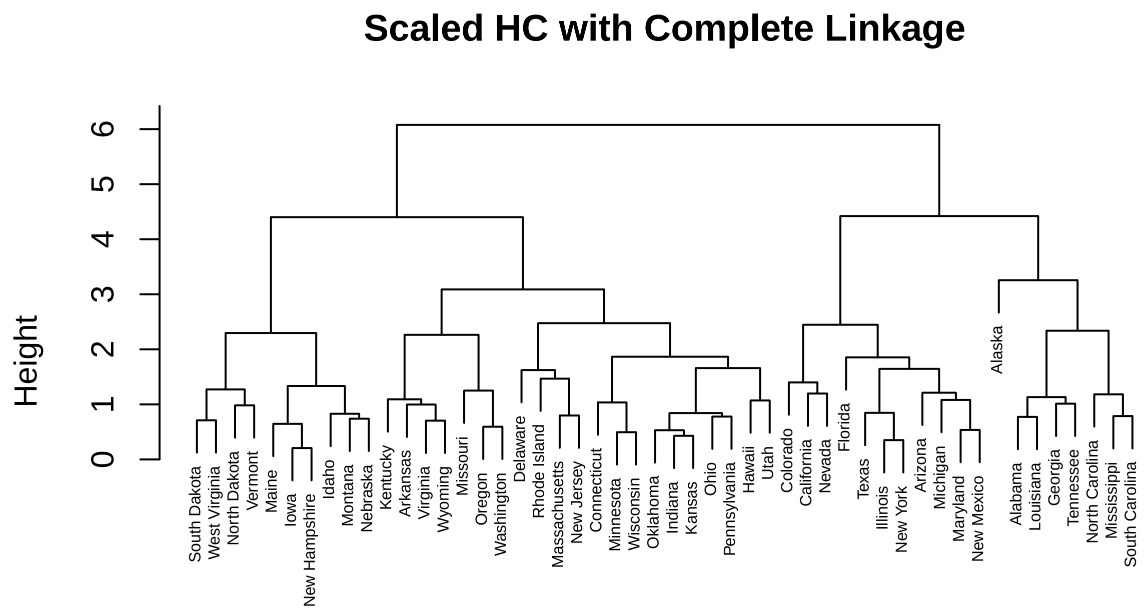
Cluster 3

```
 [1] "Connecticut"   "Hawaii"        "Idaho"         "Indiana"
 [5] "Iowa"          "Kansas"        "Kentucky"      "Maine"
 [9] "Minnesota"     "Montana"       "Nebraska"      "New Hampshire"
[13] "North Dakota"  "Ohio"          "Pennsylvania"  "South Dakota"
[17] "Utah"          "Vermont"       "West Virginia" "Wisconsin"
```

10.9 - c)

Hierarchically cluster the states using complete linkage and Euclidean distance, after scaling the variables to have standard deviation one.

```r
data.scale = scale(data)
hc.complete.scale = hclust(dist(data.scale), method = "complete")
plot(hc.complete.scale, main = "Scaled HC with Complete Linkage",
     xlab = "", sub = "", cex = 0.5)
```



Comparing the dendrogram above with the unscaled version, it seems that the splits are more even and the final terminal nodes have a different order and clustering of states.

10.9 - d)

What effect does scaling the variables have on the hierarchical clustering obtained? In your opinion, should the variables be scaled before the inter-observation dissimilarities are computed? Provide a justification for your answer.

```r
cluster_1 = cutree(hc.complete.scale, k = 3)[cutree(hc.complete.scale, k = 3) == 1]
cluster_2 = cutree(hc.complete.scale, k = 3)[cutree(hc.complete.scale, k = 3) == 2]
cluster_3 = cutree(hc.complete.scale, k = 3)[cutree(hc.complete.scale, k = 3) == 3]
```

Here are three distinct clusters found using standardized observations:

Cluster 1:

```
[1] "Alabama"        "Alaska"         "Georgia"        "Louisiana"
[5] "Mississippi"    "North Carolina" "South Carolina" "Tennessee"
```

Cluster 2:

```
 [1] "Arizona"    "California" "Colorado"   "Florida"    "Illinois"
 [6] "Maryland"   "Michigan"   "Nevada"     "New Mexico" "New York"
[11] "Texas"
```

Cluster 3:

```
 [1] "Arkansas"      "Connecticut"   "Delaware"      "Hawaii"
 [5] "Idaho"         "Indiana"       "Iowa"          "Kansas"
 [9] "Kentucky"      "Maine"         "Massachusetts" "Minnesota"
[13] "Missouri"      "Montana"       "Nebraska"      "New Hampshire"
[17] "New Jersey"    "North Dakota"  "Ohio"          "Oklahoma"
[21] "Oregon"        "Pennsylvania"  "Rhode Island"  "South Dakota"
[25] "Utah"          "Vermont"       "Virginia"      "Washington"
[29] "West Virginia" "Wisconsin"     "Wyoming"
```

Scaling our variables significantly scaled the height at which our clusters are formed, and we can also observe that different clusters are formed when cutting the scaled tree.

Scaling should be performed before dissimilarities are computed because our variables are not on the same scale and would therefore have an uneven influence in the measure of dissimilarity. Here, Urbanpop is given as a percentage while all other are provided in instance per 100,000.

```r
head(USArrests)
```

```
           Murder Assault UrbanPop Rape
Alabama      13.2     236       58 21.2
Alaska       10.0     263       48 44.5
Arizona       8.1     294       80 31.0
Arkansas      8.8     190       50 19.5
California    9.0     276       91 40.6
Colorado      7.9     204       78 38.7
```

In this problem, you will generate simulated data, and then perform PCA and K-means clustering on the data.

10.10 - a)

Generate a simulated data set with 20 observations in each of three classes (i.e. 60 observations total), and 50 variables.

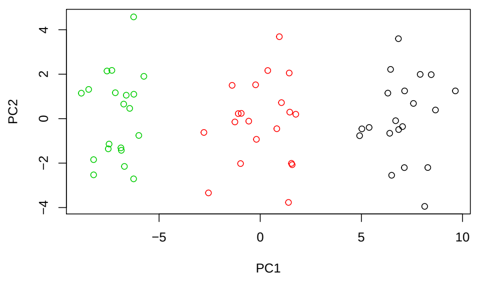
```
set.seed(533)
X = rbind(matrix(rnorm(20*50, mean=1), ncol = 50),
          matrix(rnorm(20*50, mean=2), ncol = 50),
          matrix(rnorm(20*50, mean=3), ncol = 50))

class=c(rep(1,20),rep(2,20),rep(3,20))
```

The data was generated using the rnorm function producing 50 variables (columns) and 60 observations where the first set of 20 rows are generated to be of class 1 with a mean of 1, the second set of 20 rows to be of class 2 with a mean of 2, and the final set of 20 rows to be of class 3 with a mean of 3.

10.10 - b)

Perform PCA on the 60 observations and plot the first two principal component score vectors. Use a different color to indicate the observations in each of the three classes.

```
pca = prcomp(X)
plot(pca$x[,c(1,2)],col=class)
```



Using a different color for each class we can observe the three distinct clusters within our two principal components.

10.10 - c)

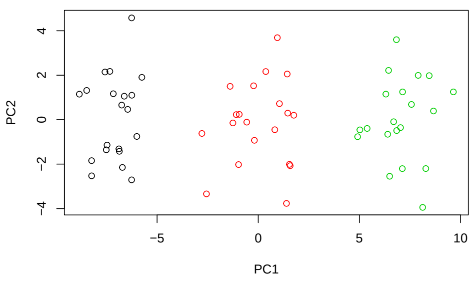
Perform K-means clustering of the observations with K = 3. How well do the clusters that you obtained in K-means clustering compare to the true class labels?

```
true = c(rep(1, 20), rep(2, 20), rep(3, 20))

set.seed(533)
km.out=kmeans(X,3,nstart = 50)
table(true,km.out$cluster)
```

```
true  1  2  3
   1  0  0 20
   2  0 20  0
   3 20  0  0
```

The K means clustering matches perfectly with the true class labels as we see that all cell values are either 20 or zero. We would normally expect to see the matches (values of 20) along the diagonal but K-means arbitrarily numbers the clusters which means the true class label may not match the label given by the clustering function. Below is a plot of the first two PC score vectors and again, using a different color for each class we can observe the three distinct clusters within our two principal components.

```
plot(pca$x[,1:2],col=km.out$cluster)
```



10.10 - d)

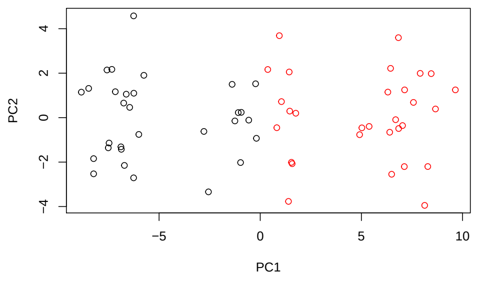
Perform K-means clustering with K = 2. Describe your results.

```
set.seed(533)
km.out = kmeans(X, 2,nstart = 50)
table(true,km.out$cluster)
```

```
true  1  2
   1  0 20
   2 10 10
   3 20  0
```

When using a cluster size of K = 2, the original labels for clusters 2 and 3 are merged into one cluster. With the lack of a third column, one of the true classes is split in half and merged into column 1 and 2. This is also reinforced by the plot of the first two PC score components below, where the observation colors have been modified to represent the classes they have been merged into.

```
plot(pca$x[,c(1,2)],col=km.out$cluster)
```



10.10 - e)

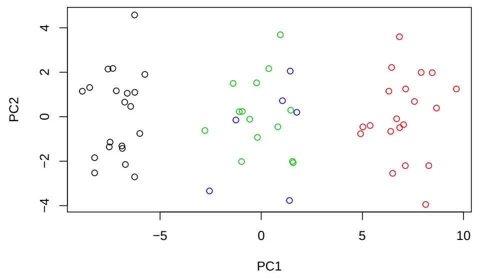
Now perform K-means clustering with K = 4, and describe your results.

```
set.seed(533)
km.out = kmeans(X, 4,nstart = 50)
table(true, km.out$cluster)
```

```
true  1  2  3  4
   1  0 20  0  0
   2  0  0 14  6
   3 20  0  0  0
```

Using a cluster size of K = 4 splits one of our clusters into two distinct clusters, as seen by the addition of the fourth column where two of the columns values always add to 20. Repeating this process shows that the split cluster changes from iteration to iteration indicating the splits produced by K-means are not always identical. Below is the plot of the first two PC score components, where the observation colors have been modified (blue and green) to represent the classes they have been split into.

```r
plot(pca$x[,c(1,2)],col=km.out$cluster)
```



10.10 - f)

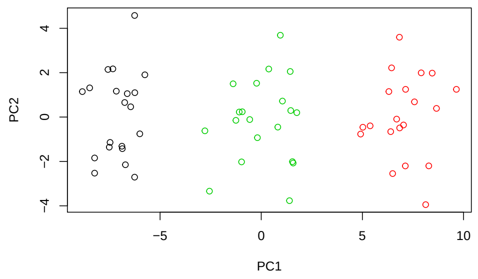
Now perform K-means clustering with K = 3 on the first two principal component score vectors, rather than on the raw data.

```r
set.seed(533)
km.out = kmeans(pca$x[, 1:2], 3,nstart=50)
table(true, km.out$cluster)
```

```
true  1  2  3
   1  0 20  0
   2  0  0 20
   3 20  0  0
```

The table above shows how well K-means clustered the first two PC score vectors compared to their true class. As the table cells are all zeros and 20's, using K-means on the first two principal components still results in a perfect cluster matching.

```r
plot(pca$x[,c(1,2)],col=km.out$cluster)
```



10.10 - g)

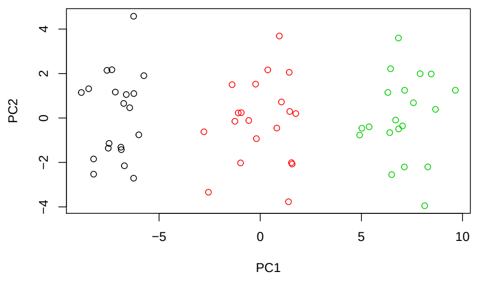
Using the scale() function, perform K-means clustering with K = 3 on the data after scaling each variable to have standard deviation one. How do these results compare to those obtained in (b)? Explain.

```r
set.seed(533)
km.out = kmeans(scale(X), 3, nstart = 50)
table(true, km.out$cluster)
```

```
true  1  2  3
   1  0  0 20
   2  0 20  0
   3 20  0  0
```

After scaling the data and performing K-means, it does not seem to effect the result as we still end up with perfect matches as in Part B (also depicted in the plot below). Intuitively, it seems as though we should end up with slightly different results and even some mismatches but after multiple attempts we end up with the same perfectly matched K-means clustering result. This could be due to the way in which the data was generated, i.e. not random enough, etc.

```r
plot(pca$x[,c(1,2)],col=km.out$cluster)
```



On the book website, www.StatLearning.com, there is a gene expression data set (Ch10Ex11.csv) that consists of 40 tissue samples with measurements on 1,000 genes. The first 20 samples are from healthy patients, while the second 20 are from a diseased group.

10.11 - a)

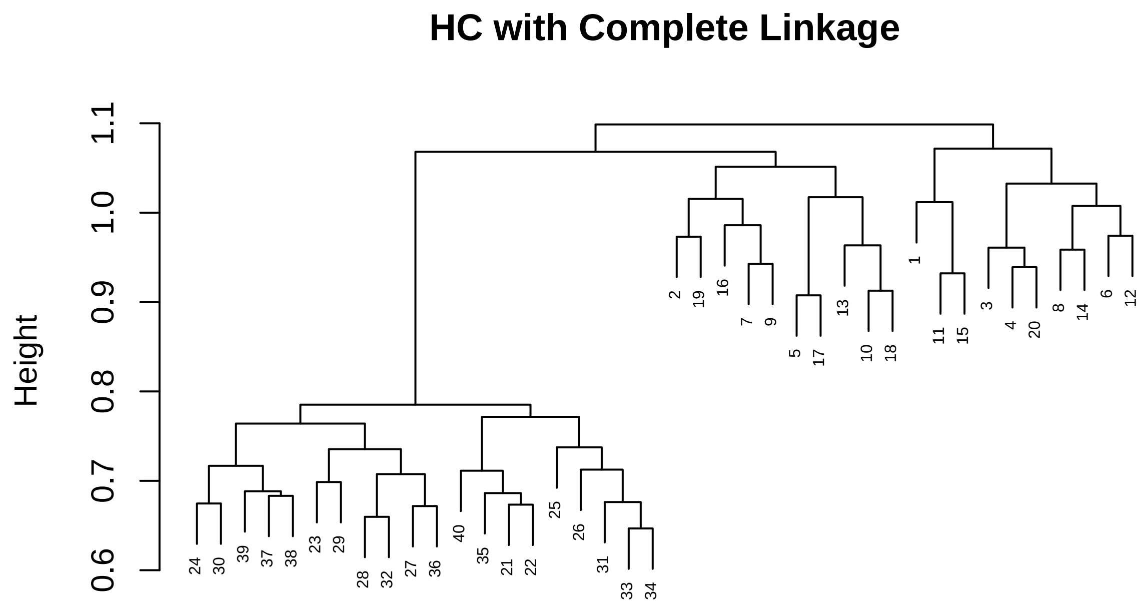
Load in the data using read.csv(). You will need to select header=F.

```
data = read.csv("Ch10Ex11.csv", header = F)
names(data) = c(1:40)
```

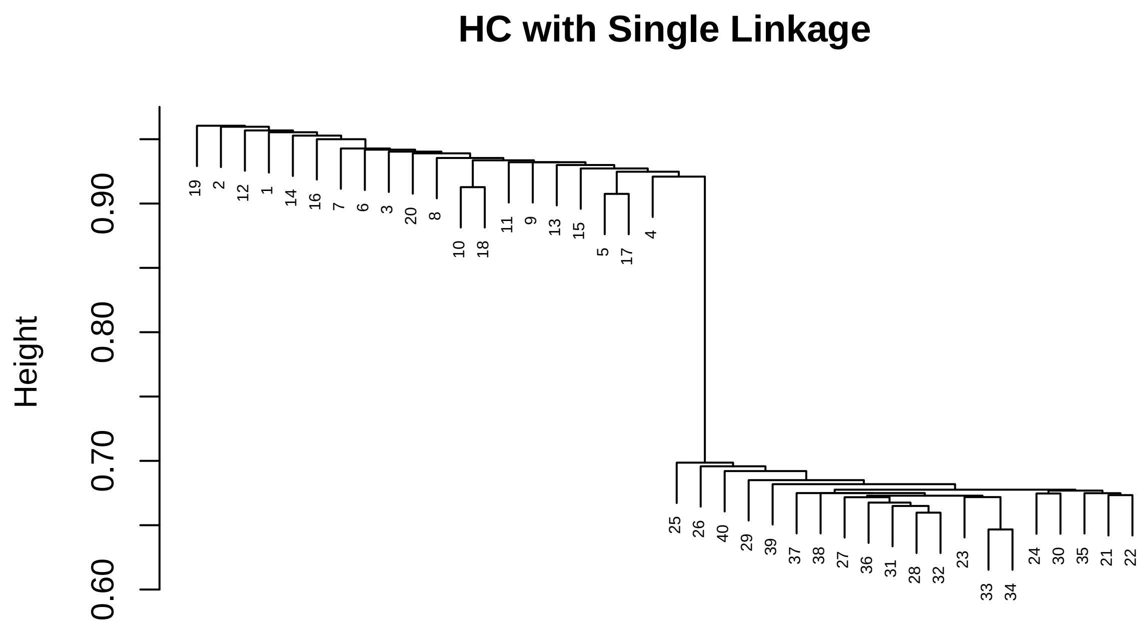
10.11 - b)

Apply hierarchical clustering to the samples using correlation based distance, and plot the dendrogram. Do the genes separate the samples into the two groups? Do your results depend on the type of linkage used?

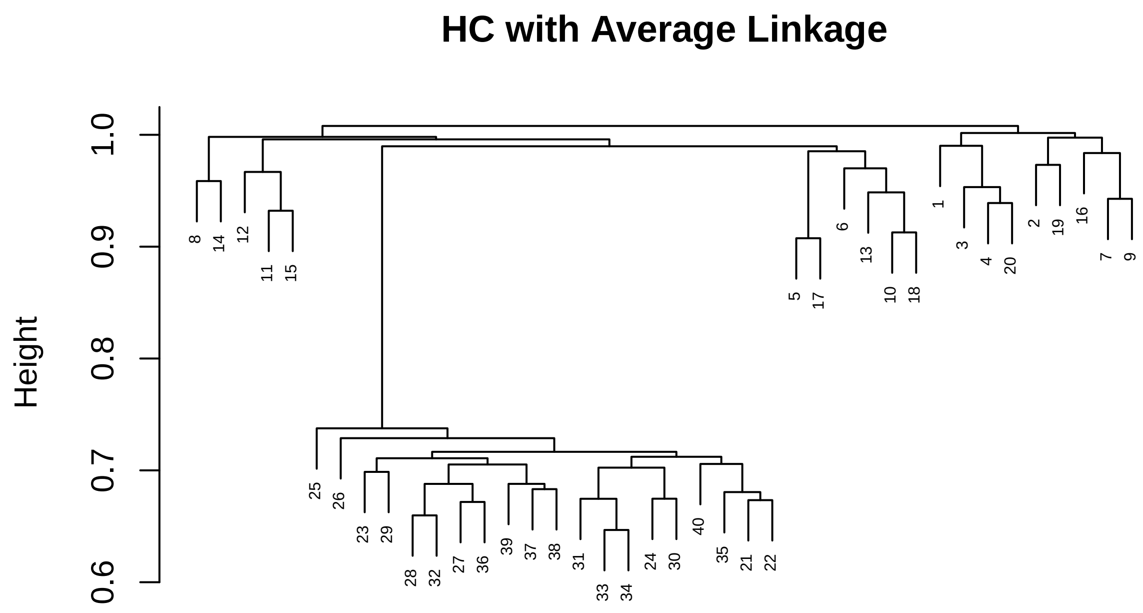
```
cor.dis = as.dist(1-cor(data))

hc.complete = hclust(cor.dis, method = "complete")
plot(hc.complete, main = "HC with Complete Linkage", xlab = "", sub = "", cex = 0.5)
```



```
hc.single = hclust(cor.dis, method = "single")
plot(hc.single, main = "HC with Single Linkage", xlab = "", sub = "", cex = 0.5)
```



```
hc.average = hclust(cor.dis, method = "average")
plot(hc.average, main = "HC with Average Linkage", xlab = "", sub = "", cex = 0.5)
```



Above are the dendrograms using correlation-based distance with single, average and complete linkage, respectively. It is clear that all three methods easily group the diseased samples together; the healthy samples, though, are very dissimilar and do not group easily with each other. This is promising, though, because if we impose 21 cluster and then *post-hoc* reclassify all the samples not in the diseased group as being in the healthy group, then we get the two desired groups. This result does not depend on the linkage used.

10.11 - c)

Your collaborator wants to know which genes differ the most across the two groups. Suggest a way to answer this question, and apply it here.

Principal Component Analysis is useful if we are interested in observing which genes contribute the most variance in the data. We will use each variable's PC rotation (loading) value because the rotation of the variable in that component adjusts for the amount of variance it contributes to the data set. The variables with the highest variance and thus rotation values will differ the most among the two groups.

```r
#the data is transposed since we need the variable names as rows
pr.out=prcomp(t(data))
head(order(abs(rowSums(pr.out$rotation)), decreasing = T))
```

By computing the loadings for each observation and ordering them by magnitude, we are able to find the top 5 genes with the most variance, which are:

```
[1] 865  68 911 428 624  11
```