9. Clustering

Clustering is *unsupervised* learning because we either do not have labels or choose to ignore them for the purpose of learning more about the data. The goal of clustering is to find groups within the data that share common features. A common application is market segmentation – looking for groups of people who are likely to buy a product. Figure 9.1 shows the iris data plotted using Petal.Length and Petal.Width. The goal in clustering is to group them into homogenous clusters.

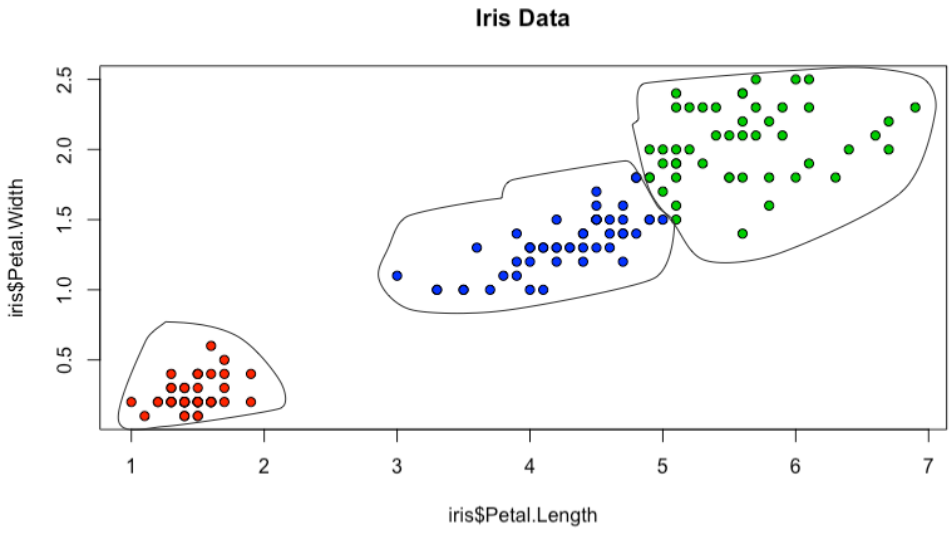

Figure 9.1: Clustering the Iris Data

9.1 Overview

There are numerous clustering algorithms, but in this chapter we focus on two of the most popular:

- k-means clustering - identifies k centers and groups other observations based on nearness to the closest centroid
- hierarchical clustering - uses a distance measure to combine observations into clusters that are organized in a hierarchy

The notion of distance is central to clustering. What do we mean by distance? We can think of this as Euclidean distance but it may be optimized to be some type of variance measure. How are the clustering algorithms discussed in this chapter different from kNN? The kNN algorithm does not cluster all the observations into clusters, it simply looks for nearby observations given a test observation. The kNN algorithm is also different in that it is a supervised algorithm whereas the clustering algorithms discussed in this chapter are unsupervised.

Clustering problems have a typical workflow:

1. preprocess data
2. determine a similarity measure
3. cluster the observations
4. analyze the results
5. repeat from 2 until the clusters are meaningful

Scaling is very important in clustering because we are measuring distance (or variance). Imagine that we are clustering people by their physical characteristics, if they are 2 pounds different in weight we would consider them similar but not if they were 2 feet different in height. The number 2 has no meaning in isolation and will make sense for distance calculations only if scaled within its column. We will use R's scale() function to put all our columns in similar ranges.

9.2 Clustering in R with k-Means

The k-means algorithm is an iterative approach that starts with a random assignment. The algorithm is as follows:

1. random assignment
2. assign each observation to its closest centroid
3. recalculate the centroid
4. repeat from 2 until convergence

There are many variations on k-means. The most important variation is how step 1 is conducted. One approach is to randomly choose k observations to be the means. Another approach is to randomly assign each observation to one of k groups.

The kmeans() function is in R, we don't need to load a package. The first argument specifies the data you want to cluster. Here we are only considering Petal.Length and Petal.Width. The second argument specifies that we want 3 clusters. In the case of the iris data we know there are 3 classes so this is an unusual advantage here. Normally determining the optimal number of clusters is done experimentally. The third argument is the number of starts. Because the algorithm starts with a random assignment, the algorithm is usually run several times with different random assignments. The kmeans() algorithm will select the best one of the 20 starts we requested. We set a seed for reproducible results but the algorithm will have 20 random starts.

Code 9.2.1 — k-Means Clustering. Iris Data.

```
set.seed(1234)
irisCluster <- kmeans(iris[, 3:4], 3, nstart=20)
irisCluster
```

The results are stored in an object named irisCluster. By typing its name at the console you get the following information.

```
K-means clustering with 3 clusters of sizes 50, 48, 52

Cluster means:
  Petal.Length Petal.Width
1     1.462000    0.246000
2     5.595833    2.037500
3     4.269231    1.342308

Clustering vector:
  [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 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
 [51] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 3 3
      3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
[101] 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 3 2 2 2
      2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2

Within cluster sum of squares by cluster:
[1]  2.02200 16.29167 13.05769
 (between_SS / total_SS =  94.3 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"
     "tot.withinss" "betweenss"
[7] "size"         "iter"         "ifault"
```

Notice the 3 clusters it found were 50, 48, 52 so we know that at least 2 observations ended up clustered with a different species than they truly are. That is a moot point because clustering does not have the same goal as classifying. The output also shows the cluster means. We did not scale the iris data because Petal.Length and Petal.Width, our two features, are in the same units. The within-cluster sum of squares measures the variance of the observations within each cluster. A smaller sum of squares indicates a more compact cluster. Notice that the first cluster has a very small value of 2.022. This is the compact red cluster visualized in Figure 9.1. The other clusters are more spread out as indicated by their higher sum of squares numbers of 16 and 13.

Finally, the output shows the components you can look at within the irisCluster object. For example to see the size, type `irisCluster$size` at the console. Most of this information was already supplied in the output.

9.3 Metrics

In most clustering problems we are clustering without knowing what the "true" clustering should be, or if there even is a "true" clustering. How will you know if you have a good clustering or not? Let's first look more precisely at what a cluster is. A cluster is a set of observations that satisfies the following two properties:

- Each of the n observations belongs to one of the clusters: $C_1 \cup C_2 \cup ... \cup C_k = 1, 2, ...n$
- No observation belongs in more than one cluster: $C_i \cap C_k = \emptyset$ for all i, j

One indication that we have a reasonably good clustering is if items within a cluster are homogeneous. One way to measure this is by nearness using a distance metric like Euclidean distance. That is we average over all k clusters, the distance between every pair of points in the cluster for all features p. We want this average distance to be minimal.

$$\sum_k \frac{1}{|C_k|} \sum_{i,j \in C_K} \sum_p (x_{ip} - x_{jp})^2 \tag{9.1}$$

The term *distance* that is commonly used in connection with k-means is intuitive because we can easily visualize it from our everyday experiences. Perhaps a more mathematically convenient term would be *variance*. We want to know the variance of a point from its assigned centroid. For point x_i, we want to quantify its variance from the mean of each centroid. Each centroid has its own mean, so the squared variance from a centroid is:

$$(x_i - \bar{\mu})^2 \tag{9.2}$$

The quantity we seek to minimize is called the **within sum of squares**. This quantity can be output from the results by typing `results$withinss` at the console. This will output the within-ss for all the clusters so if you want an overall withinss, place sum() around the expression. The within-ss indicates how compact clusters are while the between-ss indicates how well separated the clusters are.

Check Your Understanding 9.1 — k-means. Using the Wine Data.

Try the following:

- Load the wine_all.csv file from the github.
- Make sure type and other columns are numeric.
- Normalize the data.
- Perform k-means clustering with k=2.
- What is the correlation between the clusters and the wine type?
- Plot the data with the following code. The graph would be crowded if we used all the data so first we randomly sample 250 observations. We use pH and alcohol for the x, y axis to spread the data out. We use a golden color for the white wine and a reddish color for the red wine. The clusters are identified by triangles and circles.
- Keep in mind we are only plotting on two dimensions in the graph. The data is of what dimension? What does the clustering tell you about white and red wines in n-dimensional space?

```
j <- sample(1:nrow(wine), 250, replace=FALSE)
plot(pH[j], alcohol[j], pch=wine_clust$cluster[j],
     col=c("brown3","goldenrod1")[wine$type[j]+1],
     xlab="alcohol", ylab="pH")
```

9.4 Algorithmic Foundations of k-means

Since there would be k^n ways to divide n observations into k clusters, a direct mathematical approach would be np-complete. It could not be solved in polynomial time. Further, a loss function for k-means woud be non-convex. For this reason we use an approximation algorithm. The k-means heuristic approach will give a good result but it is not guaranteed to be a global optimum. This is

why we use several starts of the algorithm. By finding many local optima we hope to get close to a global optimum.

The two iterative steps of the k-means algorithm can be viewed in a more general sense as an example of the Expectation-Maximization algorithm which estimates parameters from data. The Expectation step computes the probability of the data given the parameters. The Maximization step then computes a better estimate of the parameters. The EM steps are repeated until convergence.

The k-means algorithm as viewed from the EM perspective:

1. E step: each observation is assigned to the closest centroid, the most likely custer
2. M step: the centroids are recomputed

The EM algorithm is widely used in machine learning in cases where parameters cannot be directly calculated.

9.5 Finding k

In unsupervised clustering, we do not know beforehand what the optimal value of k will be. We have to find it experimentally. The notebook `9-3-kmean-k` in github provides an example of finding k using synthetic data. The rnorm() function was used to create 60 points, in 3 distributions with centers (10, 3), (27, 2) and (41, 5). These are the "true" clusters but the regions overlap a little. We plot the unclustered data with different shapes for each distribution. Then we try clustering with different values of k. Figure 9.2 shows the results of kmeans() with k=3. The shapes represent the "true" distributions. We see one purple square between the orange and purple clusters and 3 green circles between the purple and green. The bottom graph shows the results when k=4. The overlap between the purple and green in the top figure has become its own cluster.

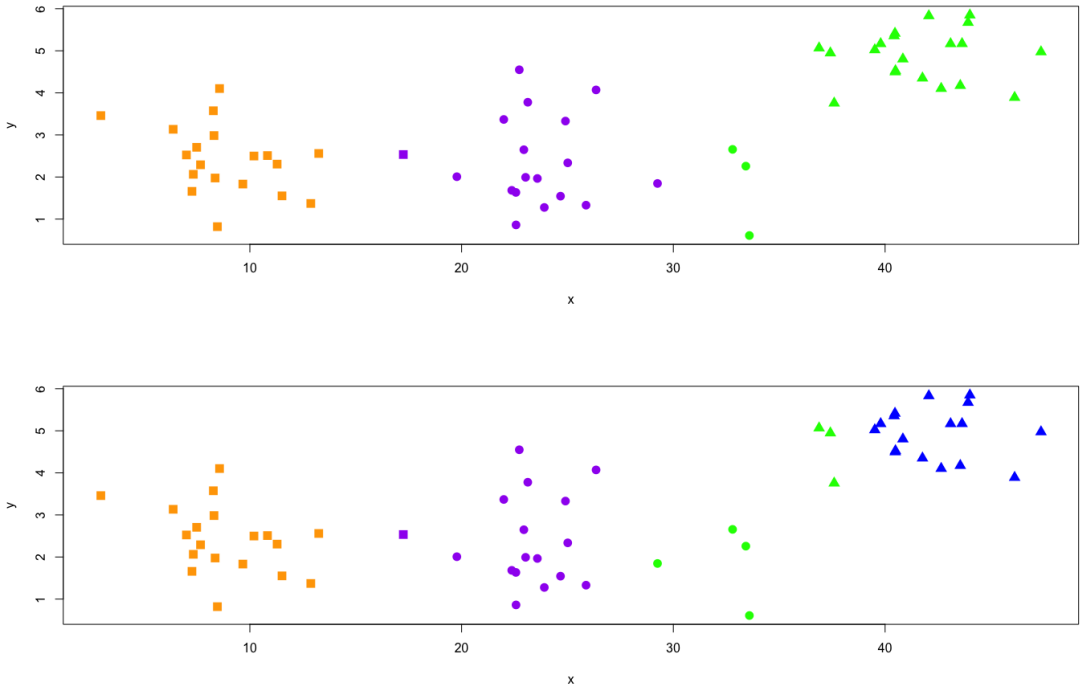

Figure 9.2: Clustering with k-means with k=3 (top) and k=4 (bottom)

We know we created the data from 3 distributions but they are not perfectly separated clusters.

In the online notebook we tried k=2, 3, 4, and 5, and printed the withinss:

```
[1] "k=2:  2530.74905628694"
[1] "k=3:  611.695448018061"
[1] "k=4:  373.0421592289"
[1] "k=5:  284.426492959912"
```

It seems there is a dramatic drop from k=2 to k=3 then it gradually decreases. It makes sense that the larger the number of clusters, the smaller the within-ss. After all, if k=n (each observation in its own cluster) then withn-ss would be 0.

Rerunning the algorithm with various values of k is tedious, let's do that in a function to try k = 2 to 9. The function in the code below reruns kmeans() with each value of k in 2:9, keeping track of the sum of the within-ss from the results. The plot is shown in Figure 9.3. Notice that within-ss is decreasing but that there is an "elbow" at k=3. The value of k at the elbow is chosen for k because k values higher than that are just reflecting the observation that larger numbers of clusters result in smaller clusters and consequently lower within-ss.

Code 9.5.1 — Finding k. Using a Function.

```
plot_withinss <- function(df, max_clusters){
  withinss <- rep(0, max_clusters-1)
  for (i in 2:max_clusters){
    set.seed(1234)
    withinss[i] <- sum(kmeans(df, i)$withinss)
  }
  plot(2:max_clusters, withinss[2:max_clusters], type="o",
       xlab="K", ylab="Within Sum Squares")
}
plot_withinss(df, 9)
```

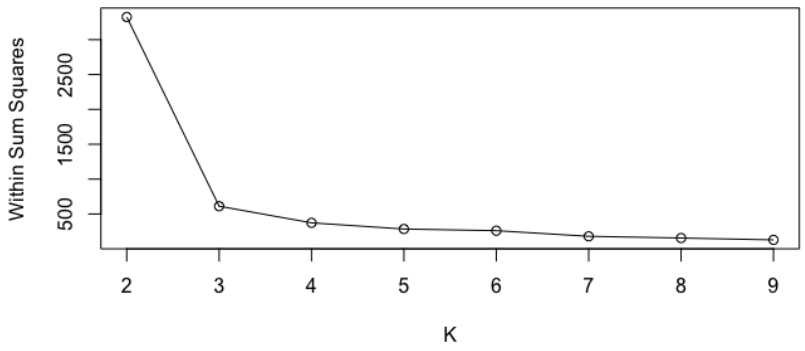

Figure 9.3: Finding K

9.5.1 NbClust

We wrote a simple function above to help us determine the best value for k. There is a very sophisticated function in package `NbClust` that provides information on over 30 indices. The online notebook uses NbClust to confirm that the best k is 3. Figure 9.4 shows that k=3 was selected by the majority of the criteria.

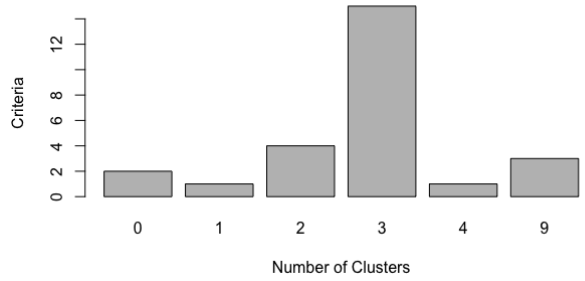

Figure 9.4: Finding K with 30 Indices in NbClust

9.6 Hierarchical Clustering

As we have seen, one of the difficulties of using k-means is that we have to specify k beforehand and it takes quite a bit of exploration to find the optimal k. Hierarchical clustering is a very different type of clustering. We do not have to specify how many clusters we have before running it. Another unique thing about hierarchical clustering is that it creates a dendogram of the clustering, which you can see in Figure 9.5.

A notebook in the github provides an example of hierarchical clustering with the nutrient data set in package `flexclust`. This data set lists the energy, protein, fat, calcium and iron for 27 prepackaged foods. Since the 5 variables are in different units, they are scaled first. R has a built-in dist() function that calculates Euclidean distance by default for a matrix. Once we have the distance, we input this into the hclust() function, built into R. Then the dendogram is plotted.

Code 9.6.1 — Hierarchical Clustering. Nutrient Data.

```
d <- dist(nutrient.scaled)
fit.average <- hclust(d, method="average")
plot(fit.average, hang=-1, cex=.8,
     main="Hierarchical Clustering")
```

9.6.1 The Algorithm

There are many variations of hierarchical clustering, here is the "bottom-up" version:

1. Place each observation in its own cluster
2. Calculate the distance between each cluster and every other cluster
3. Combine the two closest clusters
4. Repeat steps 2 and 3 until all clusters merge into one cluster

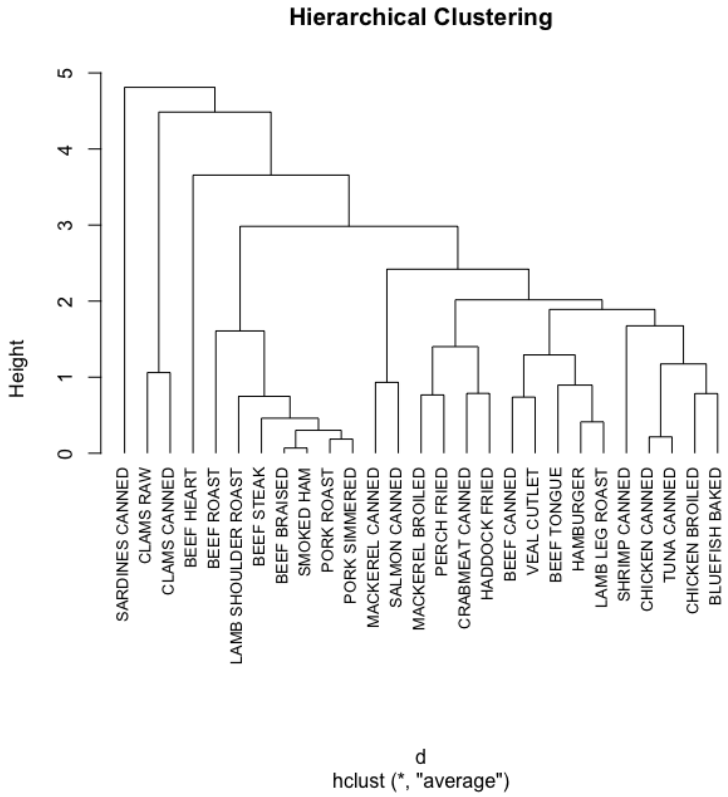

Figure 9.5: Hierarchical Clustering on Nutrient Data

How do you measure distance between clusters? There are 3 different types of measurements, called linkage, that we can use in hierarchical clustering:

1. single linkage: the shortest distance between any points in the clusters; tends to create elongated clusters
2. complete linkage: the longest distance between any points in the clusters; more compact but sensitive to outliers
3. average linkage: the average distance between points in the clusters

9.6.2 Cutting the Dendogram

Figure 9.6 shows the dendogram with 3 possible cuts in colored lines. The blue line would result in two clusters, while the red and green lines would result in 3 and 5 cluters, respectively. The built-in cutree() function can be used to cut a tree created by hclust(). The red line cut in Figure 9.6 could be created by command cutree(fit.average, 3), where fit.average is the hierarchical clustering and 3 is the number of clusters to retrieve. The cutree() function returns a vector of indices of the cluster to which each observation belongs.

The online notebook for hierarchical clustering shows how to loop through possible cuts and evaluate the clusters. After visually inspecting the clusters to see if they make sense for the task at hand, you might consider printing tables of the clusters as shown in the online notebook. Another option is to use some kind of clustering metric. There are dozens of clustering metrics and that topic is beyond the scope of this book. We will briefly discuss one clustering metric that is available via the NbClust package. The Rand index is a measure of cluster purity that ranges from -1 to +1. Rand Index close to 0 means that the clustering was random.

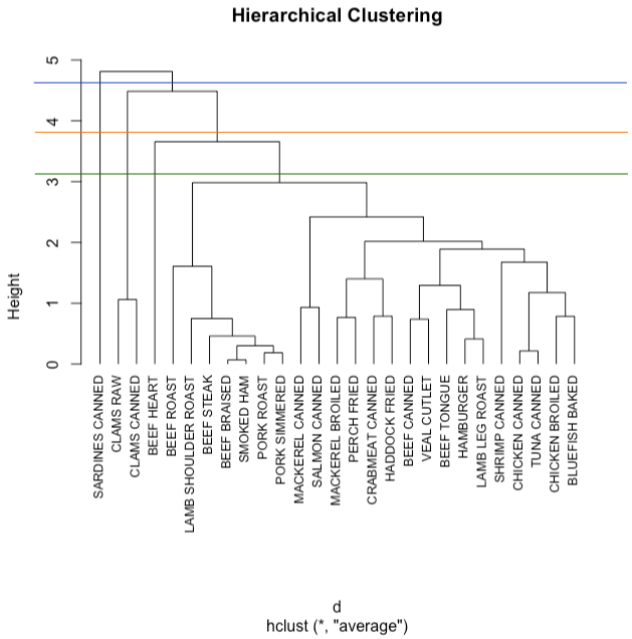

Figure 9.6: Cutting a Dendogram

9.6.3 Advantages and Disadvantages of Hierarchical Clustering

Hierarchical clustering tends to bog down with a lot of data. Another potential disadvantage is that it is a greedy algorithm. Once an observation is in a cluster, it is stuck there. The algorithm does not try several starts as k-means does.

So when should you use hierarchical clustering, also called hierarchical aggolomorative clustering? If you suspect there is some hierarchical structure in data, then this algorithm may find it.

Check Your Understanding 9.2 — Hierarchical Clustering. Using the same normalized wine data as above, try the following:

- Perform hierarchical clustering on the entire wine data. Describe the dendogram.
- Randomly sample 10 observations and perform hierarchical clustering. Compare observations that were clustered together to identify any patterns.

■

9.7 Summary

Although data exploration is an important first step in any machine learning approach, it is vitally important in clustering. You have to get to know your data before you can begin to select features for clustering and in the end to evaluate your results. Otherwise you are just seeing elephants in clouds.

The algorithms explored in this chapter are examples of unsupervised machine learning, specifically clustering algorithms. How are these techniques used? Clustering is helpful when a lot of data is available but it is not labeled. Clustering can find patterns in the data that would be difficult for humans to find. Use case examples include: finding customer clusters for marketing, finding clusters of home types and values for city planning, clustering species in biology, identifying high crime areas in police work, and much more.

9.7.1 Quick Reference

Reference 9.7.1 kmeans

```
# scale df first if necessary
# specify k=3 and  nstart=25
results <- kmeans(df, 3, nstart=25)
# examine results
results
```

Reference 9.7.2 Hierarchical

```
# scale df first if necessary
d <- dist(df)
fit.average <- hclust(d, method="average")
# plot the dendogram
plot(fit.average, hang=-1, cex=.8,
     main="Hierarchical Clustering")
```

9.7.2 Lab

Problem 9.1 — Practice k-means on the Wine Data. Try the following:

1. Use the wine data referenced in this chapter, which is available on the github.
2. We saw in the chapter how the wine type, which is a factor being treated as an integer, pulled the data into two clusters. This is not helpful in learning more about the data. For this exercise, remove the red/white type column.
3. Normalize the data.
4. Create a plot of within-ss to help determine the best k.
5. Perform k-means with this value.

Problem 9.2 — Practice Clustering on the Glass Data. Try the following:

- Load the Glass data set from package mlbench. Research this data set and write a brief description of the columns.
- Run kNN with k=7 on the unscaled data to predict glass type from the other columns.
- Now scale the data and see if your performance improves or not.
- Try different values of k to see if you find higher accuracy.
- Run the k-means algorithm with k=7. Comment on the result of the clustering. Make a table of values like this: `table(df$Type, glass_cluster$cluster)` and comment on any patterns you see.
- Visually check how homogenous the clusters are by printing the clusters and types of glass in each cluster.
- Use the plot_withinss function from this chapter to find the best cluster on the data.
- Use the NbClust() function to find the suggested value of k. Cluster with this value.
- Compare the Rand index values for different clusterings that you have tried.
- Perform hierarchical clustering on the data. Try cuts at 3, 4, 5, and 6, comparing the Rand index at each.
- Comment on which cut you think is best, and why.

9.7.3 Exploring Concepts

Problem 9.3 How is the kNN algorithm different from k-means?

Problem 9.4 Contrast the meaning of "k" in kNN, k-means, and k-fold cross validation.

Problem 9.5 Is the k-means algorithm guaranteed to find the optimal clustering? Why or why not?

Problem 9.6 Earlier it was stated that hierarchical clustering is a greedy algorithm. What does this mean, and why is it a potential disadvantage?

9.7.4 Going Further

The book *Practical Guide to Cluster Analysis in R* by A. Kassambara is partially available here: http://www.sthda.com/

The site also has a page devoted to hierarchical clustering: `http://www.sthda.com/english/wiki/print.php?id=237`