CSI 2300: Lecture 23 – Clustering and Multidimensional Scaling

Outline

In this lecture, we'll look at *clustering* and *multidimensional scaling*.

- Introduction
- K-means clustering
- Hierarchical clustering
- Multidimensional scaling

What is Clustering? What is Multidimensional Scaling?

Today we look at some related, but very different topics: clustering and multidimensional scaling. In brief:

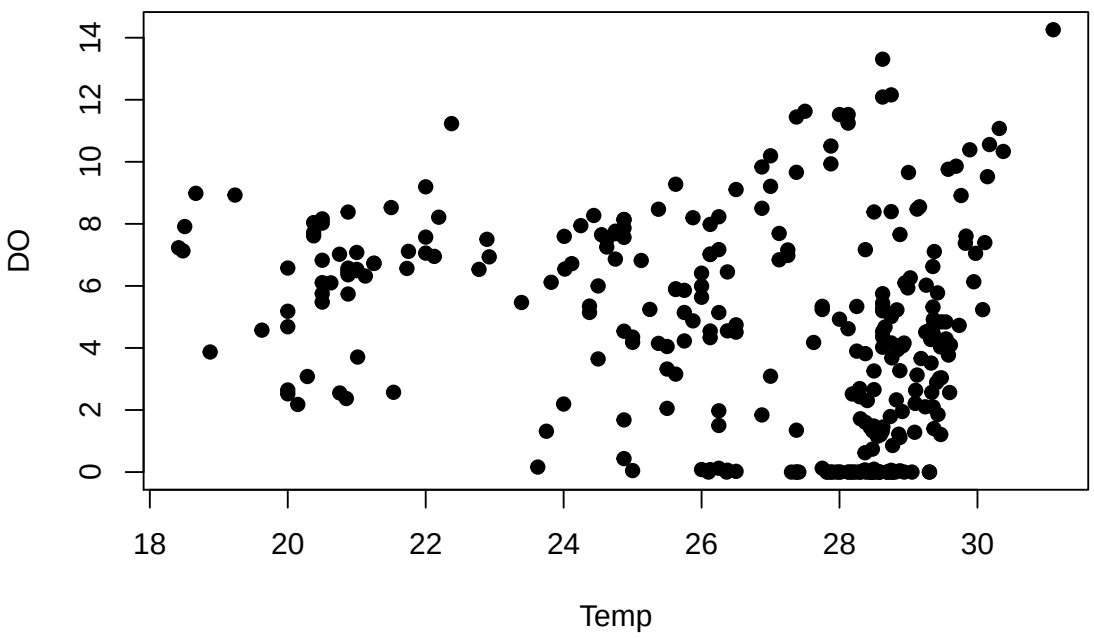
- **Clustering** is forming groups from observations (or from pairwise distances of observations). There are many clustering algorithms; we'll look at two: k-means, and hierarchical clustering.
- **Multidimensional scaling** is constructing a vector representation of data which is useful for viewing its structure, from pairwise distances.

One common theme here is the lack of a *dependent variable*. In regression we were trying to predict something; with clustering and multidimensional scaling we are not. We're simply trying to figure out underlying patterns in the data in some way. This helps us:

- Summarize the data – by grouping similar observations
- Compress the data – we can replace similar observations with their group ID
- Visualize the data – we can see different relationships in the data

Data Representation – Original Space vs Pairwise Distances

In this lecture, we're going to think about two kinds of data representations. The first, which we'll call "original space", is one we are used to: observations that are numeric, such as physical measurements (temperature readings, water flow rates, ammonia levels) or locations (GPS latitude, longitude) – anything that can be quantified. We have seen this in regression problems. We think of having n observations, with d variables for each observation. One of the things we can do with original space data is view it in a scatterplot (at least, we can view 2 or 3 variables at a time).

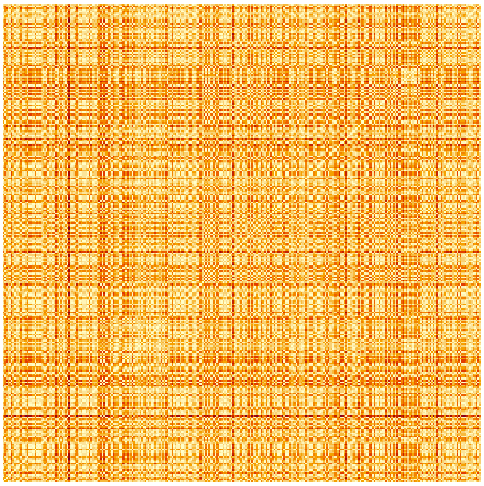


Some algorithms (e.g. k-means) operate on this original space data. However, for other algorithms – hierarchical clustering, multidimensional scaling – we'll use **pairwise distances**.

A pairwise distance matrix is just what it sounds like: instead of n observations with d variables, the representation is an $n \times n$ matrix – where entry (i, j) represents the distance between observation i and observation j. We no longer have the original d variables per observation (or we are ignoring them).

Sometimes pairwise distance matrices arise naturally, such as when asking consumers to tell which of pairs of products they think are similar / dissimilar. Other times we can construct them directly from the original n observations with the `dist()` function in R.

```
eml_dist_matrix <- dist(eml_small)
image(as.matrix(eml_dist_matrix), axes=F, xlab="", ylab="", asp=1,
      main="Pairwise distances between EML observations")
```



With those representations in mind, let's get going.

Partitional Clustering with K-means

The k-means algorithm is a classical algorithm which:

- finds k clusters in the data, each of which is represented by a *cluster center* (sometimes called "centroid")
- *partitions* the data into k clusters – each observation is assigned to exactly one of the k clusters – the one whose center is nearest
- operates on *original space data*
- takes as input the *observations* and the *number of clusters* (k)

It is a "hill climbing" algorithm which starts with some initial set of cluster center positions, and then repeatedly does the following:

- For each observation, assign it to its nearest cluster center.
- Move each cluster center to the average of the observations that were just assigned to it.

This is best illustrated through a video (this one from bitLectures): https://www.youtube.com/watch?v=5I3Ei69I40s

What is k-means doing?

Recall that linear regression was finding a line that minimizes the sum of squared residuals – (observation − prediction)^2 – between the observed y values and the line (or plane).

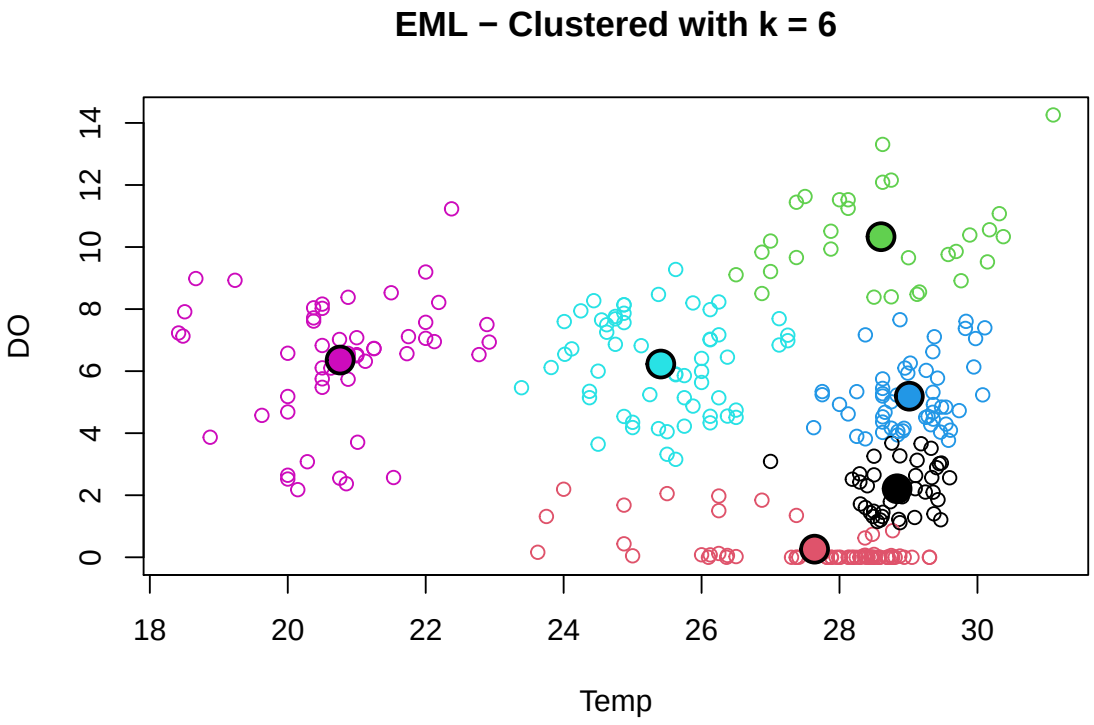
k-means is doing something similar: it is trying to minimize the sum of squared distances between each observation and its nearest cluster center. That is, it is trying to move the centers c to minimize this quantity, known as the **distortion**:

$$\sum_{i=1}^{n}(x_i - c_{(i)})^2$$

where x_i is a vector representing the ith observation, and $c_{(i)}$ is a vector representing the cluster center that is closest to x_i. (If the term "vector" does not feel familiar, think "point in space".)

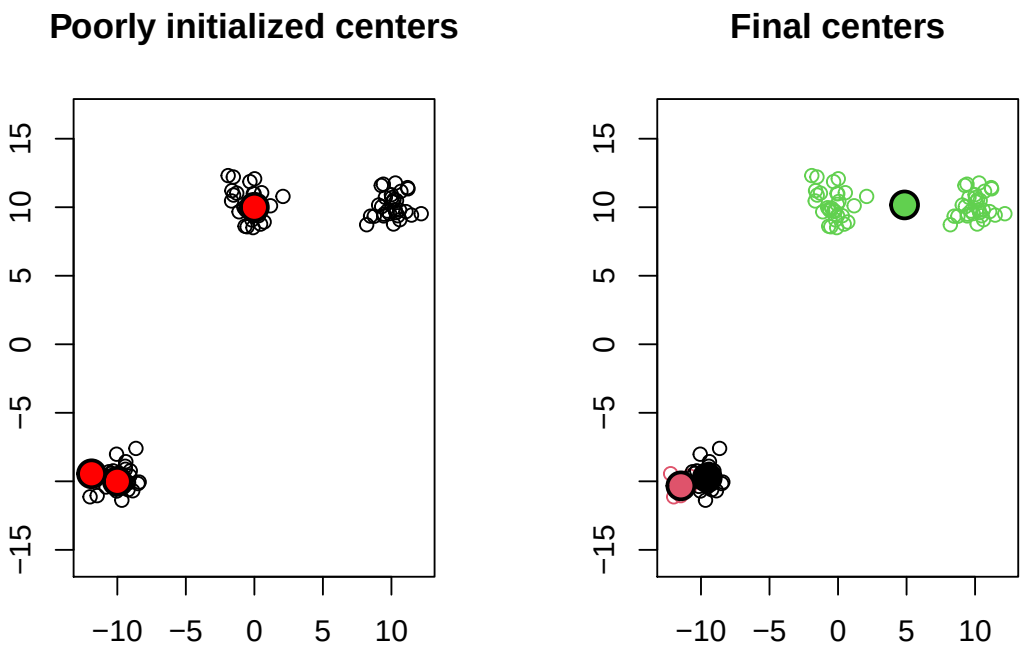
Example with EML Data

```
set.seed(1)
k <- 6
eml_fit <- kmeans(eml_small, k)
plot(DO ~ Temp, data=eml_small, col=eml_fit$cluster, pch=21,
     main=paste("EML - Clustered with k =", k))
points(eml_fit$centers[,"Temp"], eml_fit$centers[,"DO"],
       pch=21, bg=1:k, cex=2, lwd=2)
```

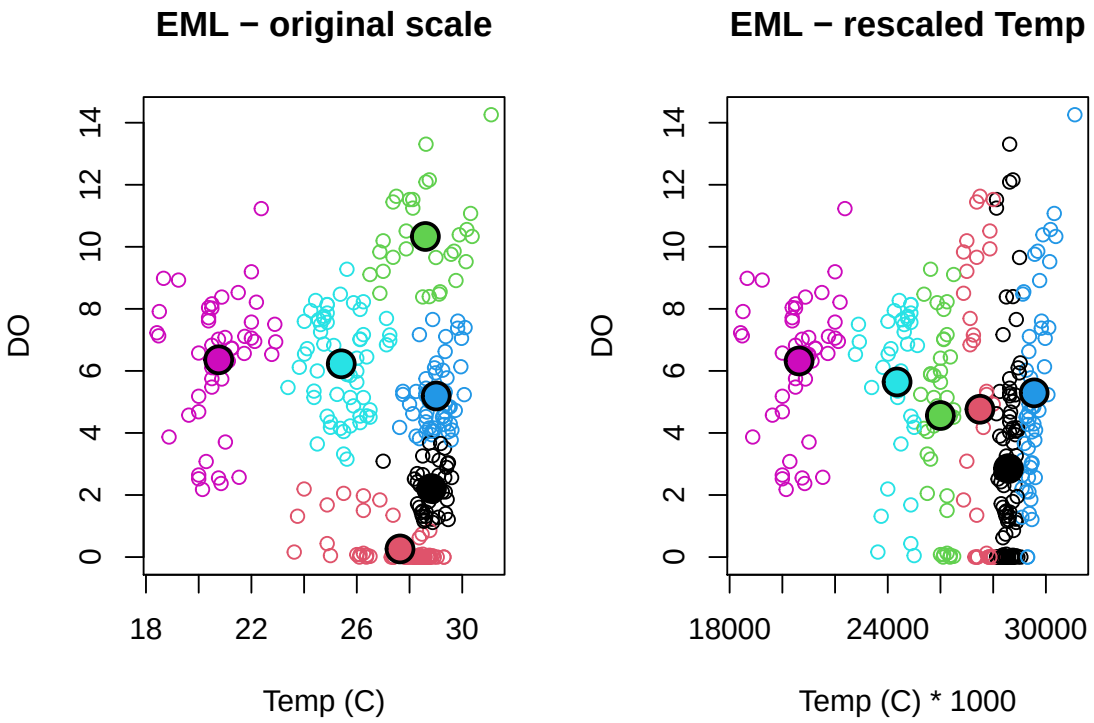


Comments

- Getting stuck: k-means is a hill-climbing algorithm which requires an initial set of center positions and then iteratively improves. It is **not** guaranteed to find the best solution (the center locations that provide the lowest sum of squares). It can get "stuck" in poor solutions. Here's an example:



- **Random restarts**: because k-means can get stuck, it is common to try different "random restarts", using different initial positions, and then take the best one. In R, the number of restarts is specified with the `nstart` parameter.
  - (If you really want to geek out on this stuff, the best way to initialize the centers – which R does not use by default – is the so-called "k-means++" algorithm https://en.wikipedia.org/wiki/K-means%2B%2B.)
- **Convergence**: k-means is guaranteed to converge, but can be stopped early. The default for R is to run 10 iterations; to increase that you can specify `iter.max`.
- **Distances and scale**: using different variable scales will lead to different distances, and therefore different clusterings.



- **What is K?** – One of the most common questions asked is "how many clusters should I use"? The answer is not always straightforward. Some suggestions:
  - Use domain knowledge to answer the question "what is a natural number of clusters for this problem?"
  - What if you use $k = n$ (i.e. every observation gets a cluster)?
  - There are many methods that try different values of k, and then pick one based on the curve of the distortion.
  - See my [Dr. Hamerly] dissertation.

Hierarchical Agglomerative Clustering with hclust

Hierarchical agglomerative clustering is a totally different method for clustering. It takes as input a distance matrix, and *not* a number of clusters. It works as follows:

- Initially every observation is in its own cluster.
- While more than one cluster exists:
  - Join together the two most similar clusters

Varying the definition of "most similar" changes the algorithm. If we have two clusters A and B, then the distance between A and B is:

- **single-link**: the distance between their **closest pair of observations**: $d(A, B) = \min_{a \in A, b \in B} d(a, b)$
- **average-link**: the **average distance between each pair of observations**: $d(A, B) = \text{average}(d(a, b))$ for $a \in A$ and $b \in B$.
- **complete-link**: the distance between their **furthest pair of observations**: $d(A, B) = \max_{a \in A, b \in B} d(a, b)$.

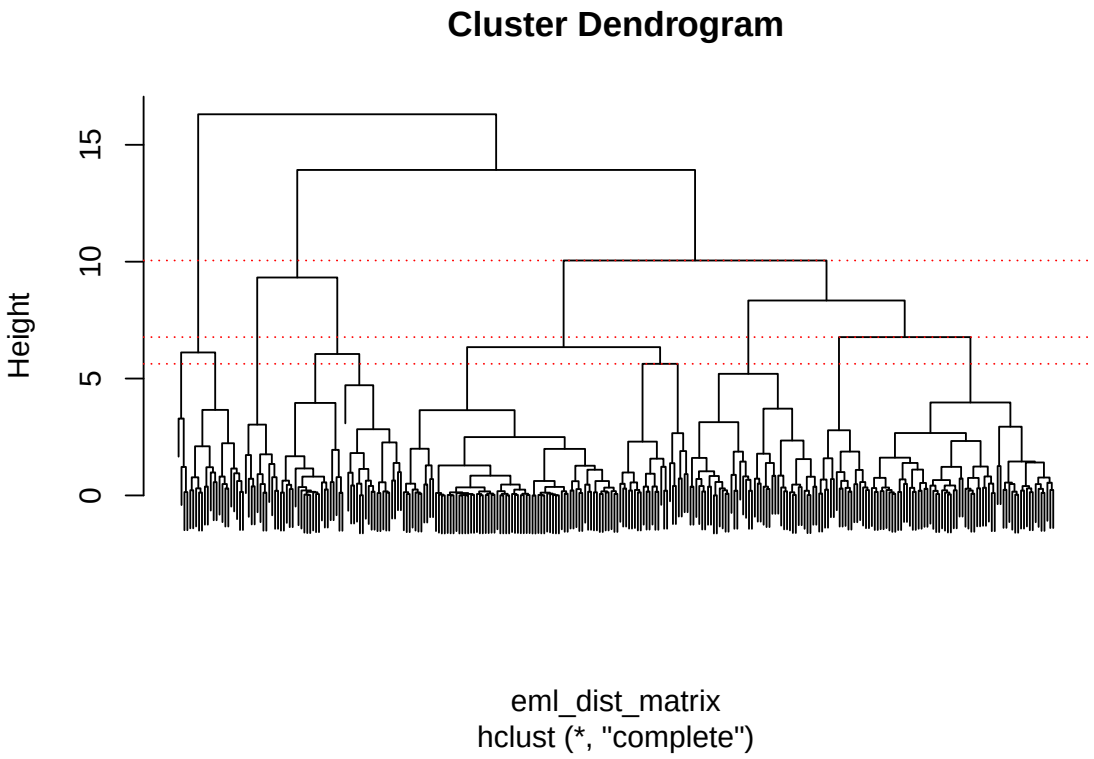
The default for R is *complete-link*.

Example

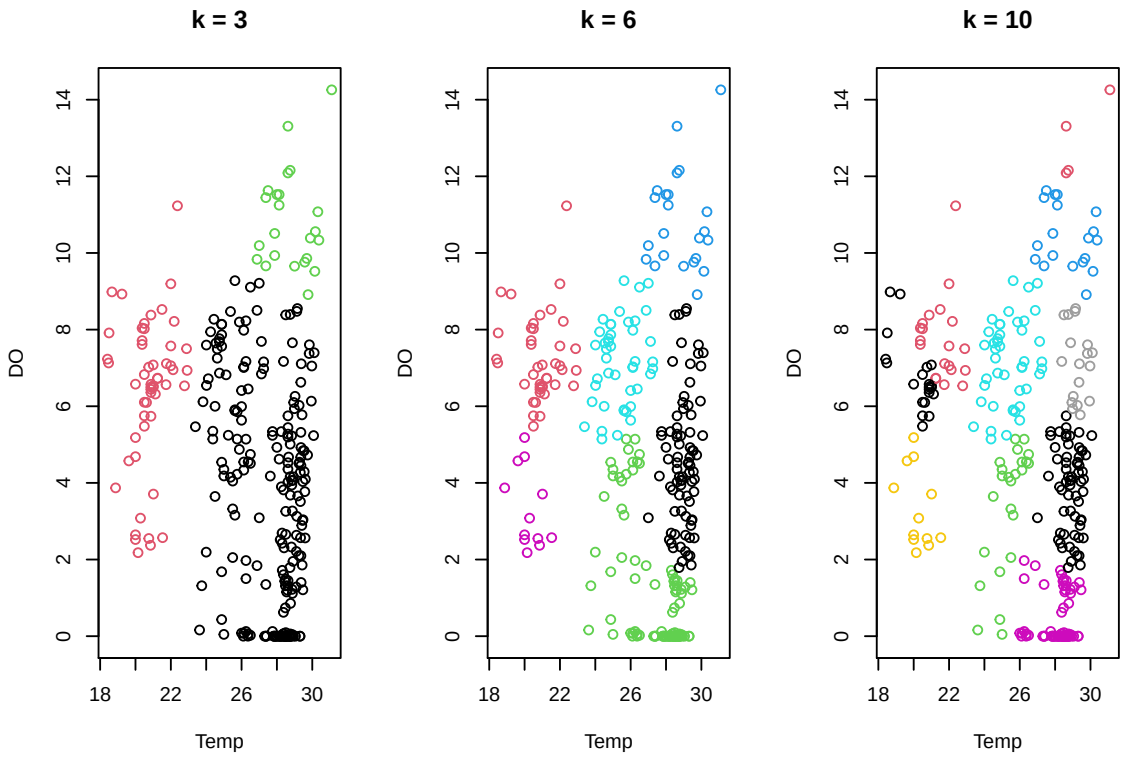
```
# point out what the data looks like clustered into different k values
k_vals <- c(3, 6, 10)

# cluster the data, plot the dendrogram, and then plot the break points for
# various k values
eml_complete <- hclust(eml_dist_matrix)
plot(eml_complete, labels=F)

n <- length(eml_complete$height)
for (k in k_vals) { abline(h=eml_complete$height[n - k + 1], col="red", lty=3) }
```




```
# show the clusterings of those different k values on the original data
par(mfrow=c(1, length(k_vals)))
for (k in k_vals) {
    members <- cutree(eml_complete, k=k)
    plot(eml_small, col=members, main=paste("k =", k))
}
```



Comments

Hierarchical clustering:

- Good for pairwise distance data.
- Good if you don't want to specify k in advance. You can investigate the dendrogram for "natural breaks".
- Does not provide guarantees about the quality of its solutions (like k-means), and is in fact not optimizing a global clustering criterion.
- Provides the same answer for the same input.

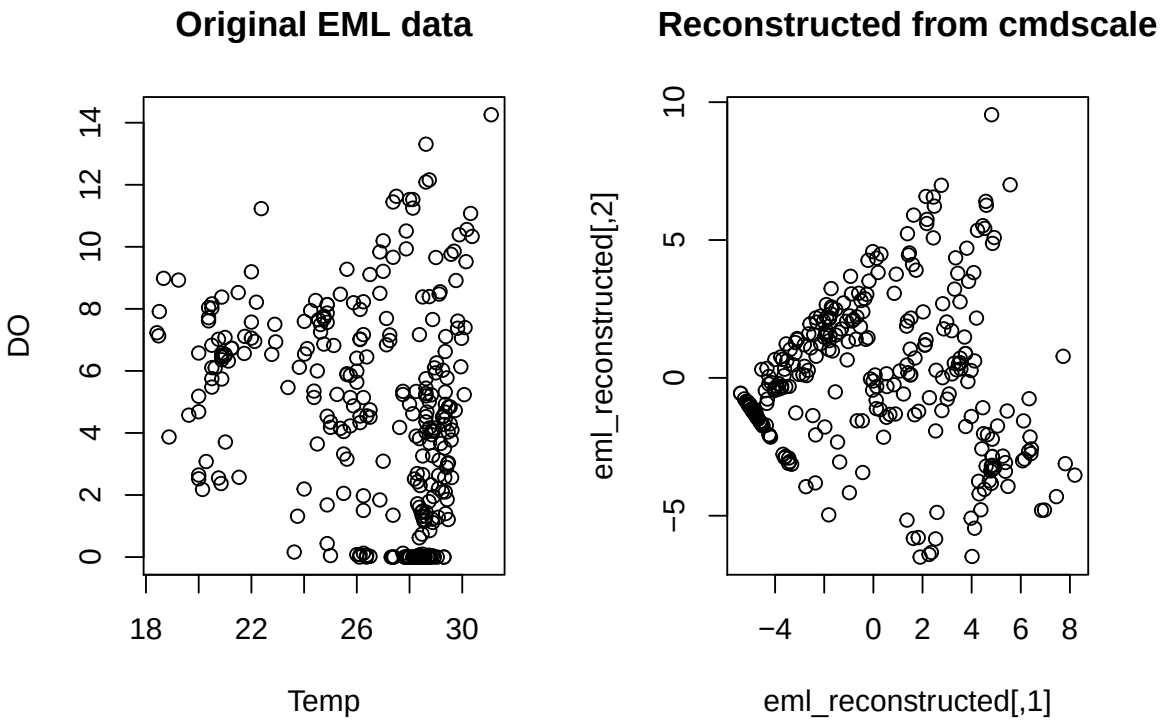
Multidimensional Scaling with cmdscale

This problem is somewhat the opposite of hierarchical clustering. The problem of multidimensional scaling is:

- Input: an $n \times n$ matrix of pairwise distances
- Output: an $n \times d$ representation of the n observations in a d-dimensional vector space.

When you read "dimension" here, just think "variable".

```
eml_reconstructed <- cmdscale(eml_dist_matrix, k=2) # here "k" is dimension
par(mfrow=c(1, 2))
plot(eml_small, main="Original EML data")
plot(eml_reconstructed, main="Reconstructed from cmdscale")
```



Note that the reconstructed data looks different. Why? It only has the pairwise distance matrix. It does **not** have access to:

- the original variables themselves, or even the *number* of original variables;
- the scales of the original variables;
- the orientation or origin of the original observations;
- etc.

Regardless, this is a good approach if we want to take **pairwise data** and turn it into something where we can visualize what's going on, or feed it into a vector-based algorithm (like k-means or regression).

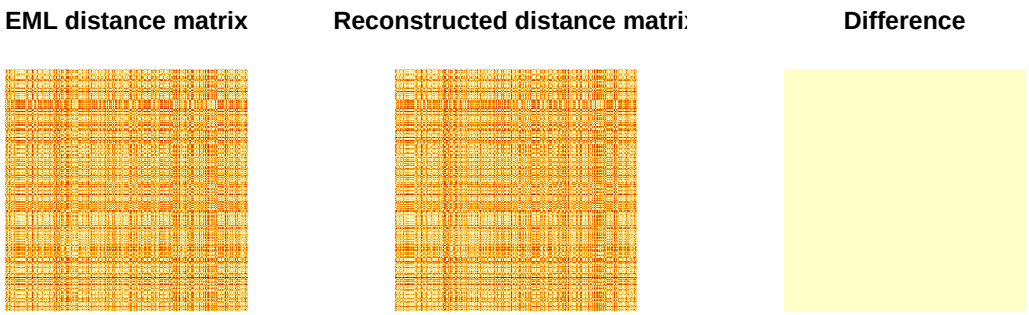
What is Multidimensional Scaling Doing?

It's trying to find n vectors where, when you compute their pairwise distances, looks as similar as possible to the original pairwise distance matrix. How this works is really interesting but beyond the scope of this lecture.

But for now, we can at least compare the results we got for EML:

```
eml_reconstructed_dist_matrix <- dist(eml_reconstructed)

par(mfrow=c(1, 3), pty="s")
# use the same coloring scale for all three plots
zlim <- c(0, max(eml_dist_matrix))
image(as.matrix(eml_dist_matrix), axes=F, xlab="", ylab="", asp=1, zlim=zlim,
      main="EML distance matrix")
image(as.matrix(eml_reconstructed_dist_matrix), axes=F, xlab="", ylab="", asp=1, zlim=zlim,
      main="Reconstructed distance matrix")
image(as.matrix(eml_dist_matrix - eml_reconstructed_dist_matrix), axes=F, xlab="", ylab="", asp=1, zlim=z
      main="Difference")
```



Comments

- If you don't know how many dimensions to select, you should choose carefully based on domain knowledge.
- Two dimensions is good for visualization, but more dimensions may be needed to capture the real structure of the data.
- The constructed variables are not necessarily physically meaningful. If the original data had variables, the constructed variables may be vastly different (rotated, shifted, scaled, combined, etc.).