Assignment 3 Group 14

September 2017, Multivariate Statistics

R. van Eeuwijk [454161] R. Hammink [435001] C. van Veen [432890] M. Koops [432409] P. de Wet [427863]

Methodology

We apply multidimensional scaling (MDS) on consumption data of 34 individuals for 8 brands of soft drinks. MDS can be used as a means of visualization in which the distances between the different observations, or in this case the difference between all individuals, are preserved as well as possible in p dimensions. If $p = 2$ or $p = 3$ those variables can then be plotted in a 2D or 3D plot. Individuals who are plotted close together are regarded as having similar preferences over soft drinks.

The first step is to compute a $n \times n$ dissimilarity matrix $\mathbf{\Lambda}$ of the original data, of which the entries δ_{ij} denote the Euclidean distance between row i and j (i.e. individual i and j) where it holds that $\delta_{ij} = 0$ for $i = j$, $\delta_{ij} = \delta_{ji}$, $\delta_{ij} \geq 0$ and $\delta_{ij} \leq \delta_{ik} + \delta jk$. Let $\mathbf{X}$ be a $n \times p$ matrix, with p preferably equal to 2 or 3 (for visualization). Define Kruskal's raw Stress function as

$$\sigma_r(\mathbf{X}) = \sum_{i<j}(\delta_{ij} - d_{ij}(\mathbf{X})) = \eta_\delta^2 + \eta^2(\mathbf{X}) - 2\rho(\mathbf{X}) \tag{1}$$

Here $d_{ij}(\mathbf{X})$ is the distance between row i and j of matrix $\mathbf{X}$. (1) can be normalized by dividing by $\sum_{i<j}\delta_{ij}^2$ such that it takes values between 0 and 1 and eases interpretation (denoted by $\sigma_n(\mathbf{X})$). Now the point is to choose a configuartion of $\mathbf{X}$ such that $\sigma_n(\mathbf{X})$ is minimized. This cannot be done analytically, so the majorization algorithm SMACOF may be used. It needs $\mathbf{\Lambda}$, an initial $\mathbf{X}_0$ and ϵ (typically very small, e.g. 10^{-6}) as input, and computes the initial Stress value. Then $\mathbf{X}_0$ is improved iteratively: at the k-th iteration $\mathbf{X}_k = n^{-1}\mathbf{B}_{\mathbf{X}_{k-1}}\mathbf{X}_{k-1}$ and the algorithm stops when $\sigma_n(\mathbf{X}_k) - \sigma_n(\mathbf{X}_{k-1}) < \epsilon$. The matrix $\mathbf{B}$ corresponding to a particular $\mathbf{X}_k$ has elements equal to $-\frac{\delta_{ij}}{d_{ij}(\mathbf{X}_k)}$ except when $d_{ij}(\mathbf{X}_k) = 0$ for $i \neq j$, and for the diagonal it has elements $b_{ii} = -\sum_{j\neq i}^{n}\frac{\delta_{ij}}{d_{ij}(\mathbf{X}_k)}$

The decision on dimensionality can be made on the basis of Stress value after convergence. Kruskal's rule-of-thumb stipulates that a Stress value of 0.1 is fair, 0.05 is good, etc, so a p-dimensional solution with Stress 0.05 should be preferred over a $(p-1)$-dimensional solution with Stress 0.1. A plot of Stress and the number of dimensions can also be used, and one should include p dimensions if an 'elbow' occurs (i.e. suddenly a less decreasing slope) at dimension p. Of course common sense can also be used, if it becomes difficult to interpret an additional dimension, then it does not really mean anything.

After an $\mathbf{X}$ is chosen, a Shepard plot can show how well the resulting transformation fits the original data. The dissimilarities (horizontal axis) are plotted with e.g. a solid point against the distances (vertical axis) with e.g. an open point. The difference represents the error, so the closer the points are together the more accurate (i.e. the lower the stress).

Functions

Euclidean Distance Function

```
euclid <- function(x1,x2)
{
  sum_sq=0
```

```
  p=length(x1)
  for (i in 1:p)
  {
    sum_sq=sum_sq+(x1[i]-x2[i])^2
  }
  return(sqrt(sum_sq))
}

euclidDistance <- function(X)
{
  n=nrow(X)
  D<-matrix(0,n,n)
  for (i in 1:n)
  {
    for (j in 1:n)
    {
      D[i,j]=euclid(X[i,],X[j,])
      D[j,i]=D[i,j]
    }
  }
  return(D)
}
```

SMACOF Function

```
smacof_fun <- function(D, X_0, eps=10^-6)
{
  n=nrow(X_0)
  p=ncol(X_0)
  k=1
  Z_k<-X_0 # set Z_k equal to initial random Z
  normstress=(0.5*sum(D^2)+0.5*sum(euclidDistance(Z_k)^2)-sum(D*euclidDistance(Z_k))) /
    (0.5*sum(D^2)) # scalar 0.5 is for i<j in summation
  print(normstress)
  repeat
  {
    normstress_lag=normstress # store current normstress to k-1 normstress
    k=k+1
    F<-D/euclidDistance(Z_k)
    F[is.nan(F)]=0 # set those elements that have NaN due to division by 0 equal to 0
    B_k<-diag(rowSums(F))-F
    X_k<-(1/n)*B_k%*%Z_k #compute improved X_k
    Z_k<-X_k
    normstress=(0.5*sum(D^2)+0.5*sum(euclidDistance(Z_k)^2)-sum(D*euclidDistance(Z_k))) /
      (0.5*sum(D^2))
    print(normstress)
    stress <<- normstress
    if (normstress_lag-normstress<=eps) # stop when improvement is smaller than eps
    {
      break
    }
  }
  return(Z_k) # returns final configuration
```

```
}
```

MDS Results

Euclidean Distance

The following code checks if the results of our own dissimilarity function equals the output of the standard `dist` function in R.

```
dissim_matrix = euclidDistance(consumption) # create dissimilarity matrix
# Compare results of own function with standard distance function
r_dist = as.matrix(dist(consumption, method="euclidean"))
all(dissim_matrix == r_dist, TRUE) # check if each element is equal
```

```
## [1] TRUE
```

MDS

Next, we compare results of our own function with the standard output of the `smacof` package in R.

Stress values:

```
cat(paste("Own Stress-1:", formatC(stress, digits=4)))
```

```
## Own Stress-1: 0.04362
```

```
cat(paste("Package Stress-1:", formatC(smacof_package$stress, digits=4)))
```

```
## Package Stress-1: 0.1976
```

The obtained stress values are quite different.

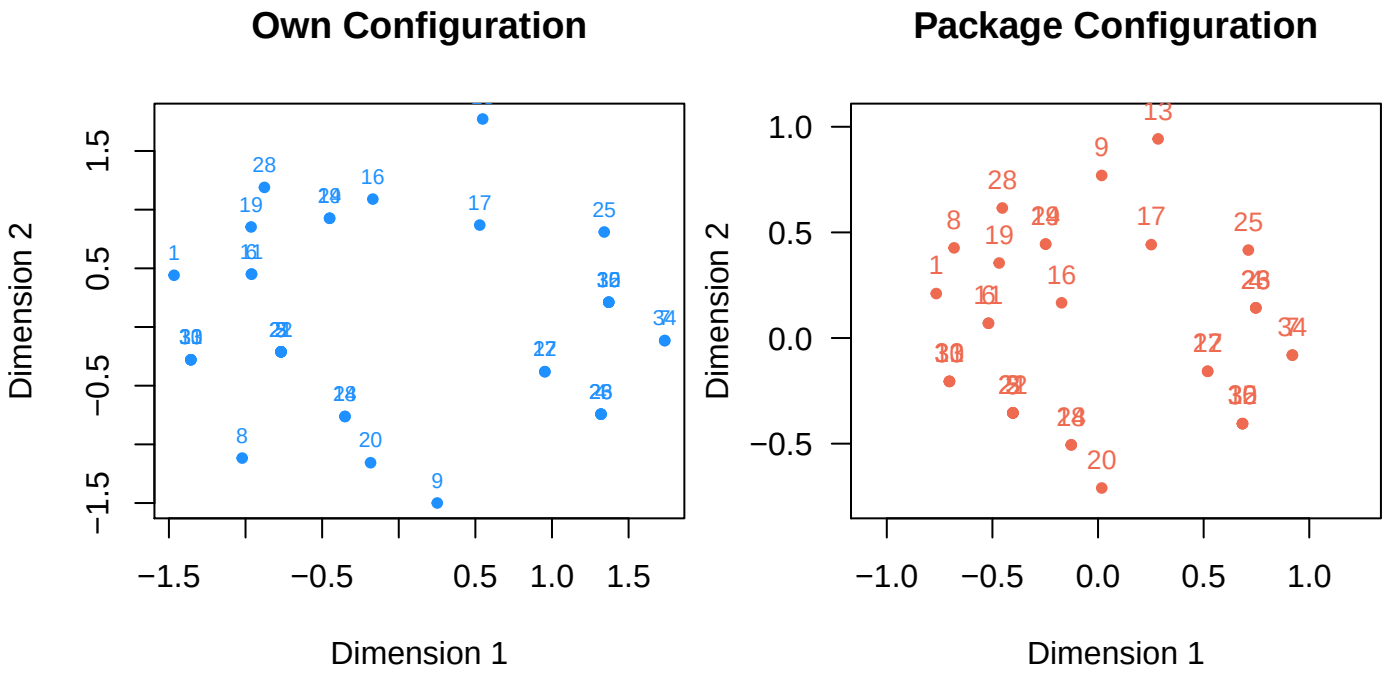
The following table reports the results of our own function and the standard function. Again we find results are not comparable.

```
comparison = cbind(smacof_own, smacof_package$conf)
colnames(comparison) = c("Own D1", "Own D2", "Package D1", "Package D2")
kable(comparison)
```

Own D1	Own D2	Package D1	Package D2
-1.4663363	0.4409579	-0.7657509	0.2105149
-0.7684427	-0.2113898	-0.4030474	-0.3544824
-0.7684427	-0.2113898	-0.4030474	-0.3544824
1.3199038	-0.7427340	0.7468103	0.1427764
-0.7684427	-0.2113898	-0.4030474	-0.3544824
-0.9607224	0.4507365	-0.5186115	0.0699132
1.7358739	-0.1156938	0.9200999	-0.0809280
-1.0217191	-1.1169682	-0.6815143	0.4267332
0.2513965	-1.4995806	0.0173577	0.7696792
-1.3566886	-0.2790760	-0.7034959	-0.2052928
-0.9607224	0.4507365	-0.5186115	0.0699132
0.9535228	-0.3806917	0.5190619	-0.1575052
0.5477002	1.7727091	0.2843211	0.9426043
-0.4511579	0.9265952	-0.2479663	0.4447524
1.3706166	0.2112566	0.6838560	-0.4053922
-0.1690867	1.0900862	-0.1724805	0.1667020
0.5285695	0.8688509	0.2519806	0.4421090
-0.3509772	-0.7623926	-0.1268236	-0.5058112
-0.9636023	0.8526097	-0.4680323	0.3550566
-0.1838210	-1.1564957	0.0176562	-0.7099116
-0.7684427	-0.2113898	-0.4030474	-0.3544824
-0.7684427	-0.2113898	-0.4030474	-0.3544824
1.3199038	-0.7427340	0.7468103	0.1427764
-0.3509772	-0.7623926	-0.1268236	-0.5058112
1.3410809	0.8095614	0.7116704	0.4162949
1.3199038	-0.7427340	0.7468103	0.1427764
0.9535228	-0.3806917	0.5190619	-0.1575052
-0.8765421	1.1897715	-0.4530025	0.6155125
-0.4511579	0.9265952	-0.2479663	0.4447524
1.3706166	0.2112566	0.6838560	-0.4053922
-1.3566886	-0.2790760	-0.7034959	-0.2052928
1.3706166	0.2112566	0.6838560	-0.4053922
-1.3566886	-0.2790760	-0.7034959	-0.2052928
1.7358739	-0.1156938	0.9200999	-0.0809280

The following figures visualize the outputs. Our own configurion plot is on the left, the standard package output is on the right.

```
plot(smacof_own, col="dodgerblue", main="Own Configuration", xlab="Dimension 1",
     ylab="Dimension 2", pch=20)
text(smacof_own[,1], smacof_own[,2], labels = row.names(consumption), cex=0.7, col="dodgerblue",
     adj=c(0.5, -1))
plot(smacof_package, plot.type = "confplot", col="coral2", label.conf = list(label = TRUE, col = "coral
     las = "1", main="Package Configuration")
```



As we have already seen in the table, results of the two approaches are not comparable. We are unsure why our own function produces different results. Most likely there is a minor error in our function resulting in wrong values. Unfortunately, we were unable to find an error in our smacof function.