Multivariate analysis in R

Stefan Evert

16 Sep 2018

Contents

1 Preparation

First, load the required support packages:

```
library(corpora)
library(wordspace)
```

as well as further optional packages if you want to follow all steps in this tutorial:

```
library(cluster)
library(MASS)
library(ggplot2)
library(reshape2)
library(Hotelling)
library(ellipse)
library(e1071)
library(Rtsne)
```

If you have RGL installed, you can also display 3D graphics in an interactive session. We use the option `eval=FALSE` on the code chunk below (and all other code chunks with 3D visualizations) so that it isn't executed when rendering the full document offline.

```
library(rgl)
```

We also need to load the additional support functions and data sets. Both files should be in the RStudio project directory.

```
source("multivar_utils.R")
load("unit7_data.rda", verbose=TRUE)
```

```
## Loading objects:
```

```
##   BrownBiber_Matrix
##   BrownBiber_Meta
##   CroCo_Matrix
##   CroCo_Meta
##   CroCo_orig2trans
##   Delta
##   DeltaComplexity
##   DeltaLemma
##   MultiVar_Matrix
##   SyntacticComplexity_Matrix
##   SyntacticComplexity_Meta
```

1.1 A small example matrix

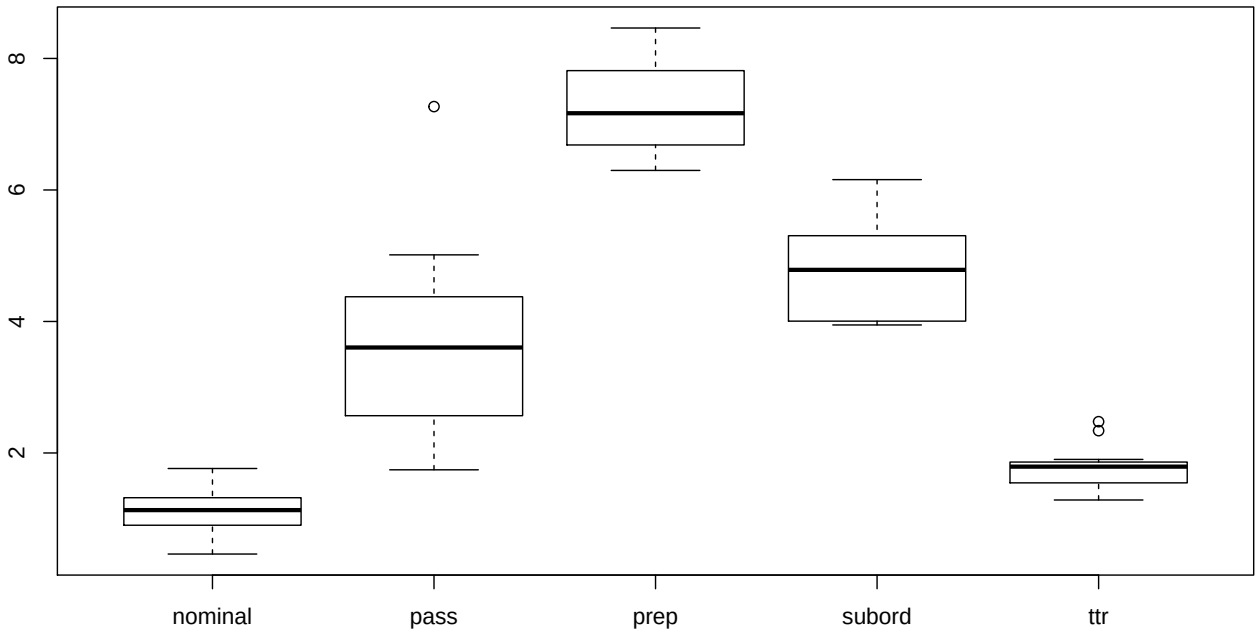
The data file includes a small example feature matrix which we can use to understand various analysis techniques.

```
knitr::kable(MultiVar_Matrix)
```

	nominal	pass	prep	subord	ttr
orig-1	1.205	5.013	6.883	4.483	1.285
orig-2	0.738	2.537	6.486	6.157	1.714
orig-3	1.252	4.462	8.463	4.785	2.476
orig-4	1.105	2.899	8.119	3.966	1.519
orig-5	1.764	4.268	7.167	3.947	1.792
orig-8	1.545	7.268	7.461	5.455	1.572
trans-1	0.463	2.208	6.297	6.089	2.339
trans-2	1.131	2.597	6.307	4.844	1.810
trans-4	0.935	1.744	7.098	4.012	1.403
trans-5	0.867	3.604	7.511	5.154	1.902
trans-7	1.387	4.290	8.211	3.998	1.822

We rename this matrix M for convenience and use a boxplot to assess the ranges and distributions of the individual features.

```
M <- MultiVar_Matrix
boxplot(M)
```



Since the ranges are considerably different, we compute standardized z-scores (Z) ensuring equal contribution of all features to the Euclidean distances.

```
Z <- scale(M)
```

1.2 The CroCo data set: replicating Evert & Neumann (2017)

Again, we assign the data matrix and metdata to shorter names.

```
MC <- CroCo_Matrix
MetaC <- CroCo_Meta
```

Get an overview of the available metadata:

```
head(MetaC)
```

```
##                     id register language status tokens chunks sentences
## 416   gtrans-share-012    share       DE  trans   2529    516        95
## 425  gtrans-speech-008   speech       DE  trans   2760    593       101
## 130   etrans-essay-021    essay       EN  trans   2136    689        95
## 375 gtrans-fiction-002  fiction       DE  trans   3536   1120        89
## 289       go-share-005    share       DE   orig   4157   1078       216
## 234       go-essay-007    essay       DE   orig   2244    720       141
```

There are 8 registers, but Evert & Neumann excluded *instruction*, *tourism* and *fiction*.

```
table(MetaC$register)
```

```
##
##       essay     fiction instruction      popsci       share      speech     tourism
##         102          40          48          42          48          64          66
##         web
##          42
```

1.3 Authorship attribution with Delta measures

We will use unnormalized word forms from the English texts as an example here. Feel free to work with the other languages or lemmatized data (hint: German gives the best authorship attribution results :–).

```
head(Delta$EN, 7)
```

```
## 7 x 7 sparse Matrix of class "dgCMatrix"
##                   ,   the    .   and   to   of    a
## delta_en_001 11733  8251 6010  3232 3418 3590 1941
## delta_en_002  3318  2116 2253  1019 1146  879  739
## delta_en_003  8533  4679 5566  3477 2847 2594 1940
## delta_en_004 14576  6417 6677  5523 5159 4895 3422
## delta_en_005 25912 12026 7836 10949 7971 8144 5315
## delta_en_006 17270  9961 7784  6530 5796 6527 4563
## delta_en_007 11109  7551 6062  4557 3975 4081 3201
```

After selecting the number of most frequent words (*mfw*) as features dimensions, we convert the sparse term-document matrix into a regular R matrix and compute relative word frequencies. Here, we will work with $n = 2000$ mfw, which should give fairly good authorship identification.

```
MA <- as.matrix(Delta$EN$M[, 1:2000])
MetaA <- Delta$EN$rows
text.sizes <- MetaA$f # number of tokens in each text
MA <- scaleMargins(MA, rows = 1 / text.sizes) # relative frequency
```

If you have a very recent version of the **wordspace** package (0.2-2 or newer), you can carry out all these steps and even compute z-scores with the `dsm.score` function:

```
ZA <- dsm.score(subset(Delta$EN, select=(1:2000)),
                score = function(f, f1, ...) 1000 * f / f1,
                scale="standardize", negative.ok=TRUE, matrix.only=TRUE)
```

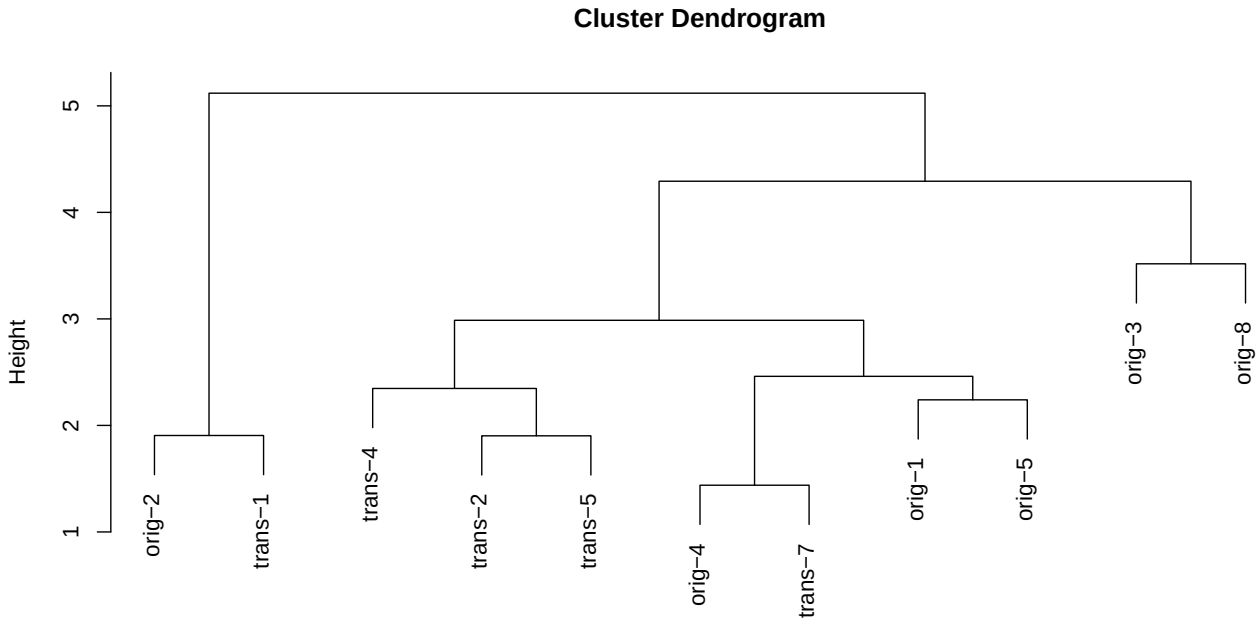
2 Distances and visualization

```
DM <- dist(Z)
round(DM, 2)
```

```
##         orig-1 orig-2 orig-3 orig-4 orig-5 orig-8 trans-1 trans-2 trans-4 trans-5
## orig-2    3.13
## orig-3    3.92   4.16
## orig-4    2.30   3.60   3.05
## orig-5    2.24   4.11   3.07   2.46
## orig-8    2.34   4.03   3.52   3.61   2.79
## trans-1   4.47   1.91   4.16   4.54   4.94   5.12
## trans-2   2.29   1.94   3.59   2.73   2.55   3.64    2.78
## trans-4   2.31   2.94   4.06   1.63   2.96   4.29    3.97    1.98
## trans-5   2.42   2.03   2.38   2.10   2.91   3.11    2.69    1.90    2.35
## trans-7   2.45   4.04   2.10   1.44   1.71   2.88    4.77    2.99    2.75    2.24
```

Hierarchical clustering incrementally groups together the two most similar data points or sub-clusters. It can be visualized in the form of a dendrogram where the height of each subtree corresponds to the "size" of the corresponding cluster.

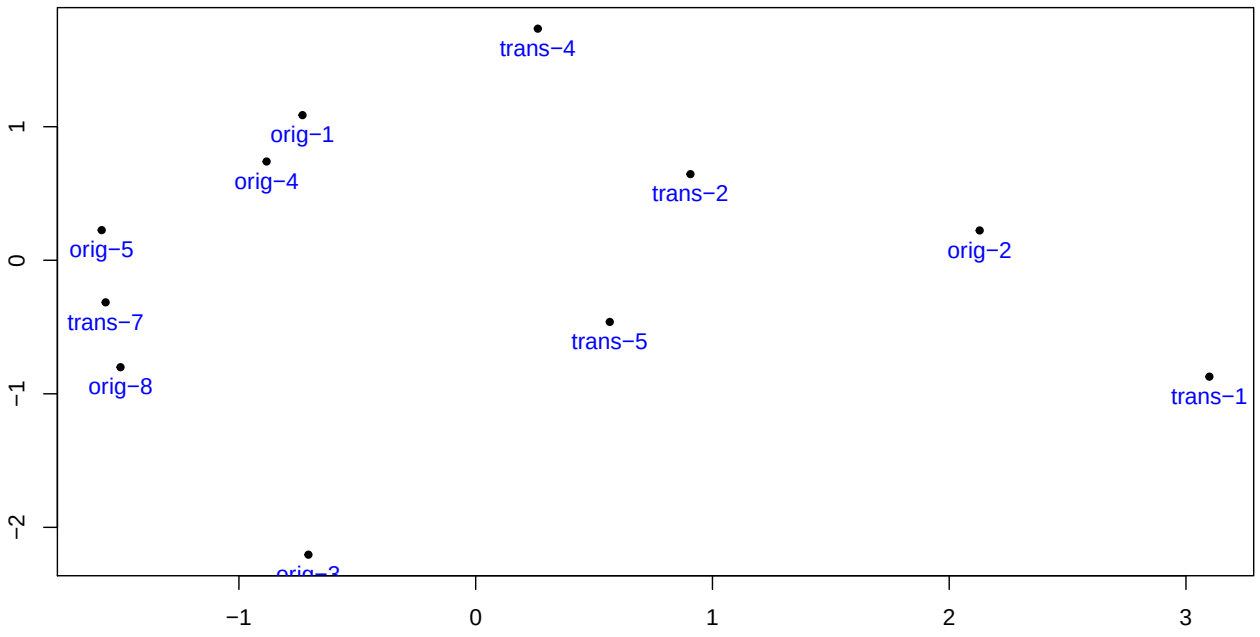
```
clusters <- hclust(DM, method = "complete")
plot(clusters, xlab="", sub="")
```



Q: Try different clustering algorithms (`method` argument) and different distance metrics for `DM`. Do you get substantially different groupings of the texts?

Another visualization approach maps the data points into a two-dimensional (or sometimes three-dimensional) space, attempting to preserve distances as faithfully as possible. A classical technique is multidimensional scaling (MDS). We will return to this issue when we have more interesting data at hand.

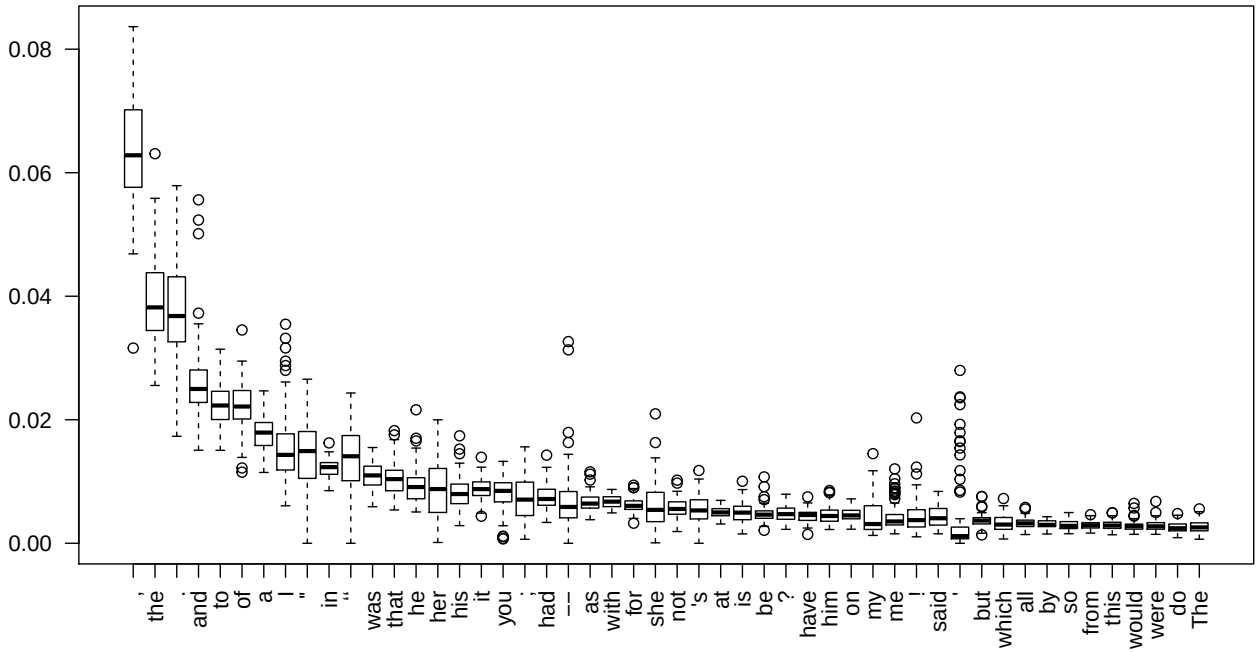
```
coord <- cmdscale(DM)
plot(coord, pch=20, xlab="", ylab="")
text(coord, labels=rownames(M), pos=1, col="blue")
```



3 Authorship attribution

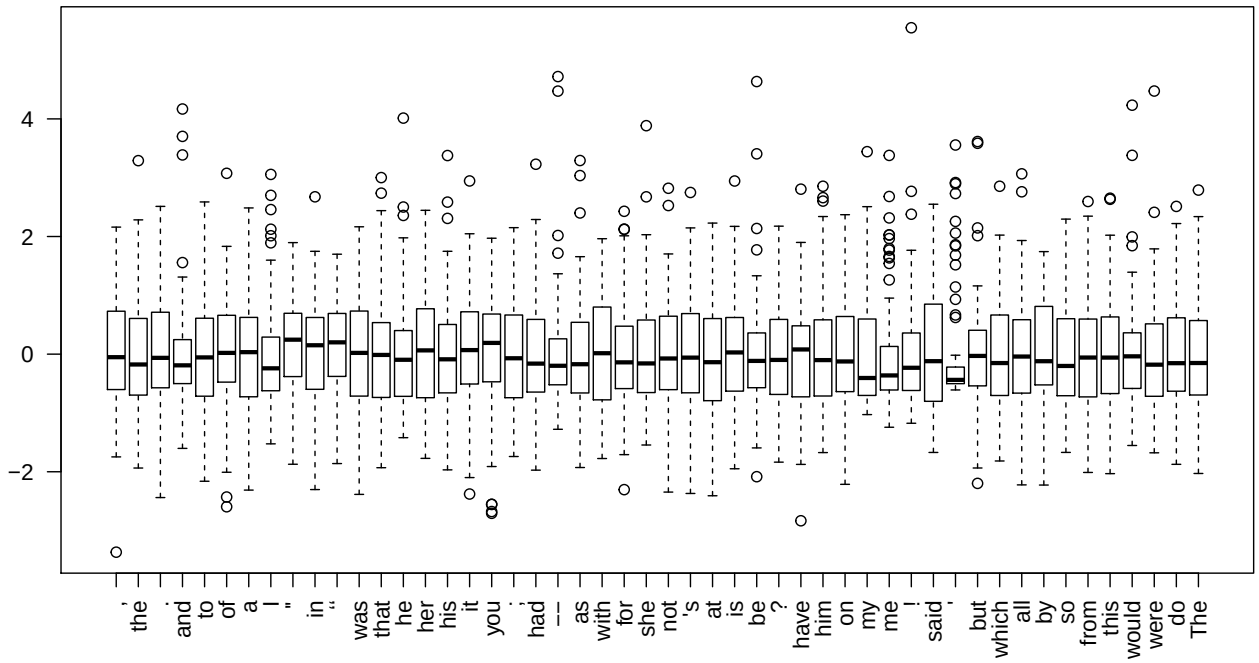
Clustering and low-dimensional visualization of distances should work well for authorship attribution tasks because we expect texts from the same author to group together. Remember that we need to standardize the columns of the feature matrix because otherwise the distances would be determined by a small number of most frequent words.

```
boxplot(MA[, 1:50], las=2)
```



After standardization, each feature makes the same contribution to the squared Euclidean distances.

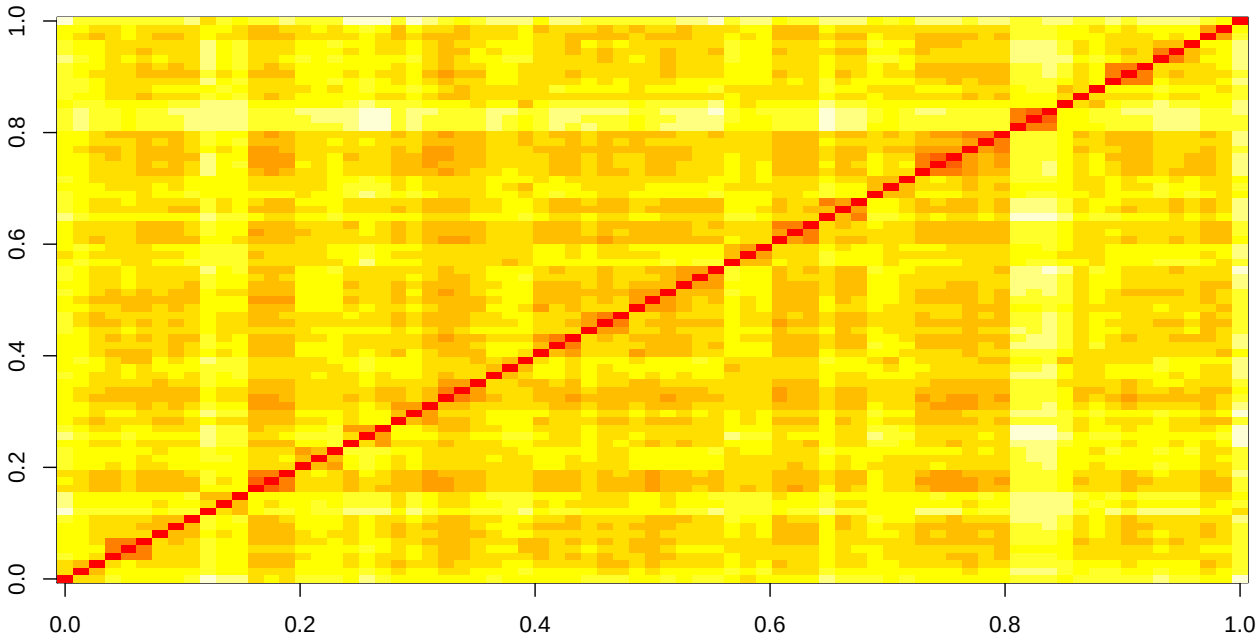
```
ZA <- scale(MA)
boxplot(ZA[, 1:50], las=2)
```



3.1 Clustering

Following Burrows's original algorithm, we use the Manhattan metric rather than Euclidean distances, even though the standardized features will no longer have the same weight. You will be asked later on to try several other distance metrics. The distance table is too large to print, but can be visualized as a coloured map. Note that the texts are sorted by author – can you see corresponding groups in the plot?

```
DMA <- dist(ZA, method="manhattan") # Burrows Delta
image(as.matrix(DMA))
```



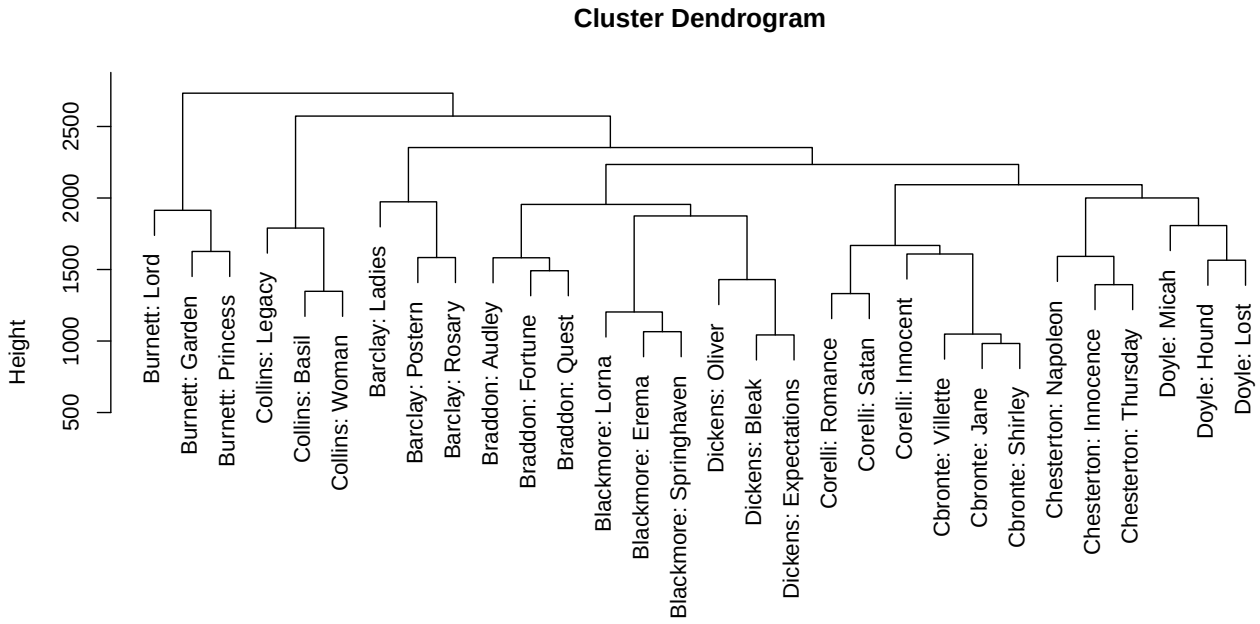
For now, we will only look at the first 10 authors so the clustering dendrogram is readable. Note how we use `droplevels()` to tell R to forget about the authors no longer included in the data set.

```
ZA10 <- ZA[1:30, ]
DMA10 <- dist(ZA10, method="manhattan")
MetaA10 <- droplevels(MetaA[1:30, ])
```

Q: Can you figure out how to take a *random* sample of 10 authors?

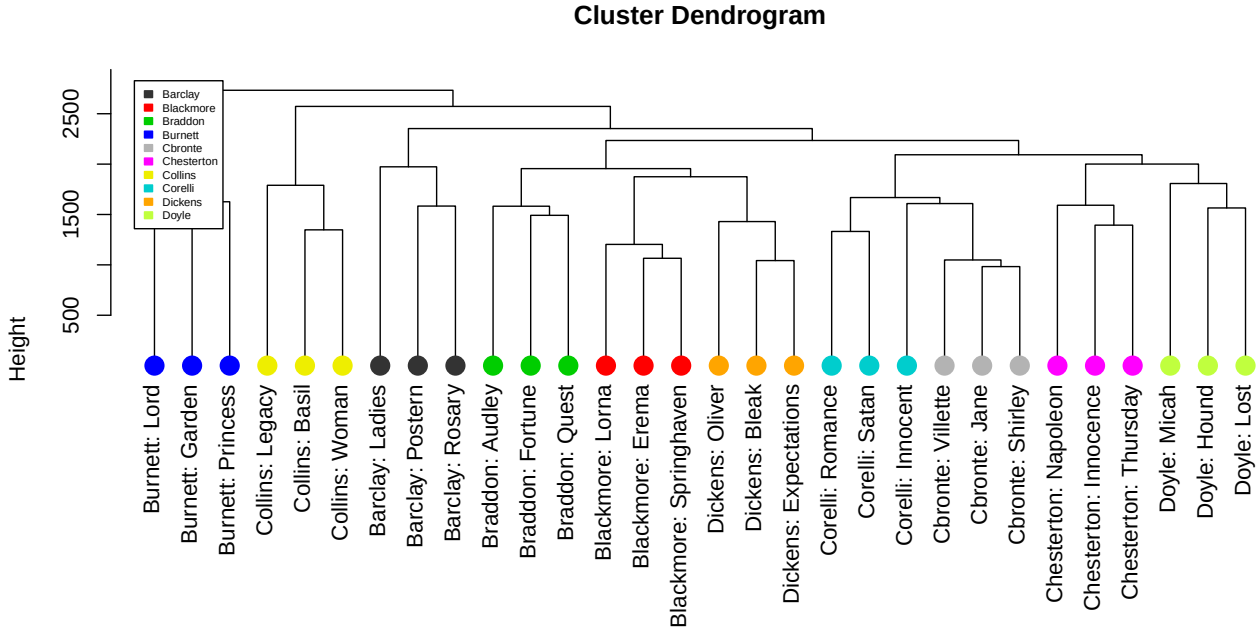
Compute a hierarchical clustering of the texts and plot the dendrogram:

```
clusters <- hclust(DMA10) # see ?hclust for the default method
plot(clusters, xlab="", sub="", labels=MetaA10$label)
```



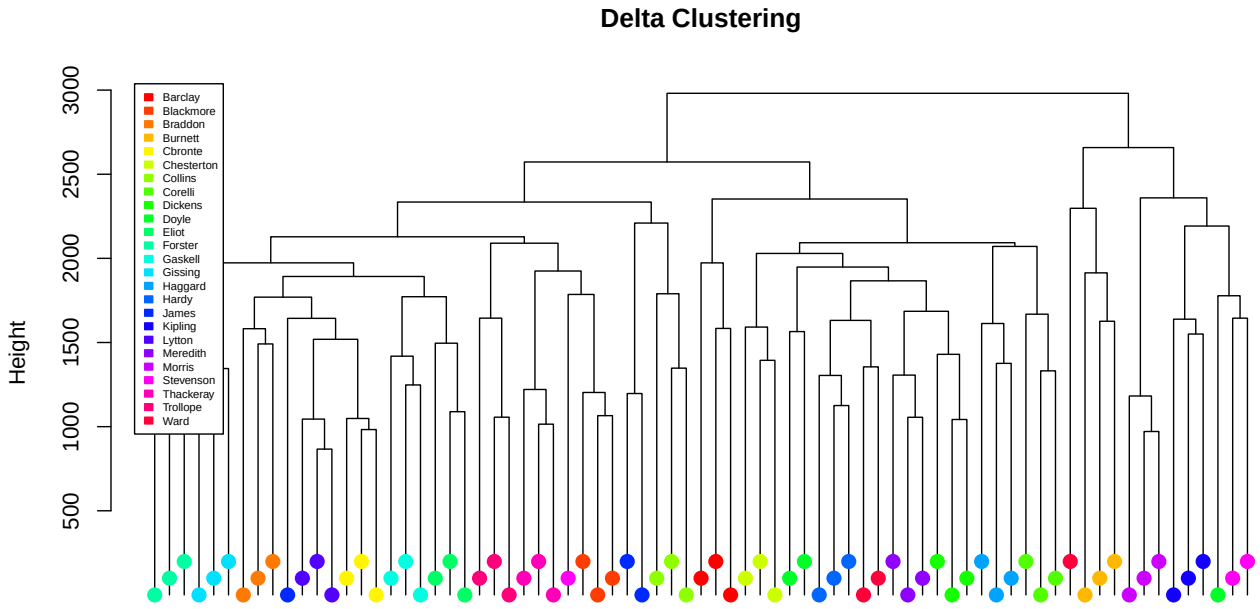
This looks promising, but it's difficult to see whether novels from the same author always cluster together. The `mvar.clust()` function provides a colour-coded visualization. It takes the standardized feature matrix as input and carries out the clustering itself, controlled by options `method`, `metric` and `p`.

```
mvar.clust(clusters,
           labels=MetaA10$label, col=MetaA10$author,
           cex=2, legend.cex=.5)
```



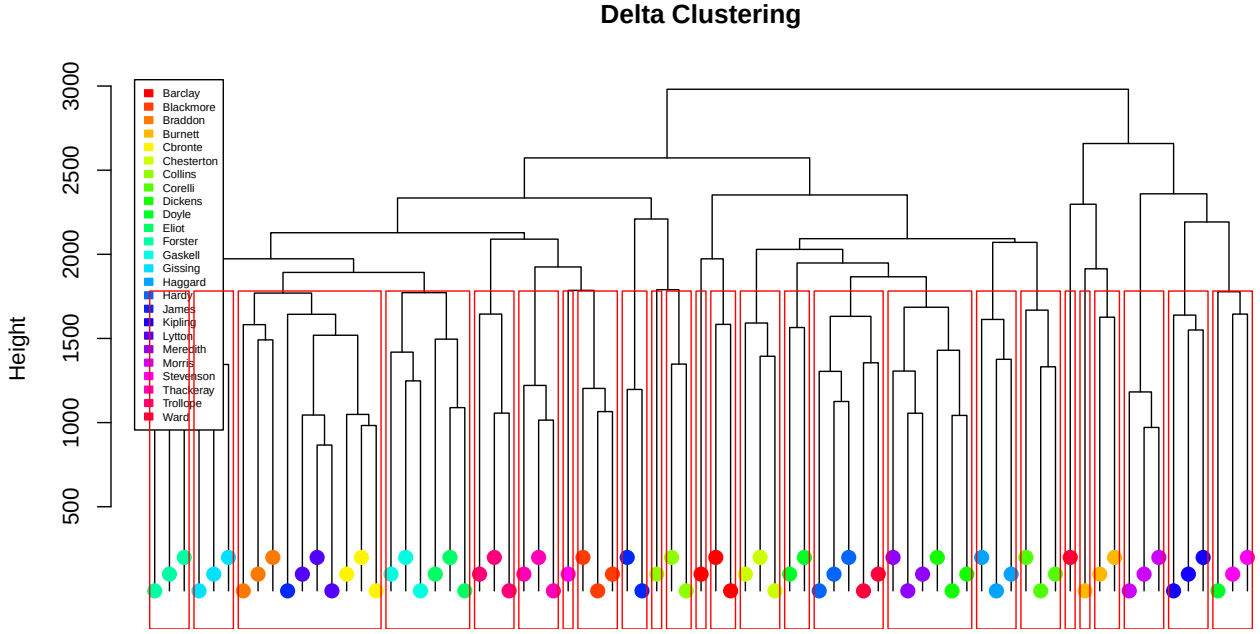
In order to colour-code all 25 authors, we need a suitable colour palette). If the dendrogram becomes too crowded, we can omit the labels and stagger the indicator dots using the `period` and `spread` options

```
clusters <- hclust(DMA)
mvar.clust(clusters, labels=NULL,
           col=MetaA$author, col.vals=rainbow(25),
           period=3, cex=1.5, legend.cex=.5, main="Delta Clustering")
```



Since we know that there are exactly 25 authors in our data set, it makes sense to split the hierarchical clustering into 25 flat clusters by cutting the dendrogram at a suitable height. We need to re-run `mvar.clust()` because each code chunk is evaluated on its own and cannot add to a previous plot.

```
mvar.clust(clusters, labels=NULL,
           col=MetaA$author, col.vals=rainbow(25),
           period=3, cex=1.5, legend.cex=.5, main="Delta Clustering")
rect.hclust(clusters, k=25, border="red") # visualization of the cut
```




```
cluster.id <- cutree(clusters, k=25)
```

The cluster IDs can then be compared with the "gold standard" authors. A simple approach is to label each cluster with the most frequent author.

```
gold <- as.character(MetaA$author) # gold standard authors as strings
cluster.auth <- majorityLabels(cluster.id, gold) # authors assigned by majority labelling
```

```
rbind(cluster.auth, gold)
```

```
##              1         2         2         3           3           3           4
## cluster.auth "Barclay" "Barclay" "Barclay" "Blackmore" "Blackmore" "Blackmore" "Braddon"
## gold         "Barclay" "Barclay" "Barclay" "Blackmore" "Blackmore" "Blackmore" "Braddon"
##              4         4         5         6         5         4         4         4
## cluster.auth "Braddon" "Braddon" "Burnett" "Burnett" "Burnett" "Braddon" "Braddon" "Braddon"
## gold         "Braddon" "Braddon" "Burnett" "Burnett" "Burnett" "Cbronte" "Cbronte" "Cbronte"
##              7           7           7           8         9         8         10
## cluster.auth "Chesterton" "Chesterton" "Chesterton" "Collins" "Collins" "Collins" "Corelli"
## gold         "Chesterton" "Chesterton" "Chesterton" "Collins" "Collins" "Collins" "Corelli"
##              10        10        11        11        11        12      12      13
## cluster.auth "Corelli" "Corelli" "Dickens" "Dickens" "Dickens" "Doyle" "Doyle" "Stevenson"
## gold         "Corelli" "Corelli" "Dickens" "Dickens" "Dickens" "Doyle" "Doyle" "Doyle"
##              14      14      14      15        15        15        14        14
## cluster.auth "Eliot" "Eliot" "Eliot" "Forster" "Forster" "Forster" "Eliot"   "Eliot"
## gold         "Eliot" "Eliot" "Eliot" "Forster" "Forster" "Forster" "Gaskell" "Gaskell"
##              14        16        16        16        17        17        17        18
## cluster.auth "Eliot"   "Gissing" "Gissing" "Gissing" "Haggard" "Haggard" "Haggard" "Hardy"
## gold         "Gaskell" "Gissing" "Gissing" "Gissing" "Haggard" "Haggard" "Haggard" "Hardy"
##              18      18      19      4         19      20        20        20        4
## cluster.auth "Hardy" "Hardy" "James" "Braddon" "James" "Kipling" "Kipling" "Kipling" "Braddon"
## gold         "Hardy" "Hardy" "James" "James"   "James" "Kipling" "Kipling" "Kipling" "Lytton"
##              4         4         11         11         11         21       21       21
## cluster.auth "Braddon" "Braddon" "Dickens"  "Dickens"  "Dickens"  "Morris" "Morris" "Morris"
## gold         "Lytton"  "Lytton"  "Meredith" "Meredith" "Meredith" "Morris" "Morris" "Morris"
##              13          22          13          23          23          23
## cluster.auth "Stevenson" "Stevenson" "Stevenson" "Thackeray" "Thackeray" "Thackeray"
## gold         "Stevenson" "Stevenson" "Stevenson" "Thackeray" "Thackeray" "Thackeray"
##              24         24         24         18      18      25
## cluster.auth "Trollope" "Trollope" "Trollope" "Hardy" "Hardy" "Ward"
## gold         "Trollope" "Trollope" "Trollope" "Ward"  "Ward"  "Ward"
```

An intuitive quantitative measure of the clustering quality is *purity*, i.e. the proportion of novels assigned to the correct author by the majority labels.

```
n.correct <- sum(cluster.auth == gold)
round(100 * n.correct / length(gold), 2) # in percent
```

```
## [1] 78.67
```

> **Q:** Can you explain why clustering purity is "optimistic", i.e. it will report a higher quality than is actually achieved?

A better quantitative measure is the adjusted Rand index (ARI), used e.g. by Evert *et al.* (2017) for the evaluation of authorship attribution tasks.

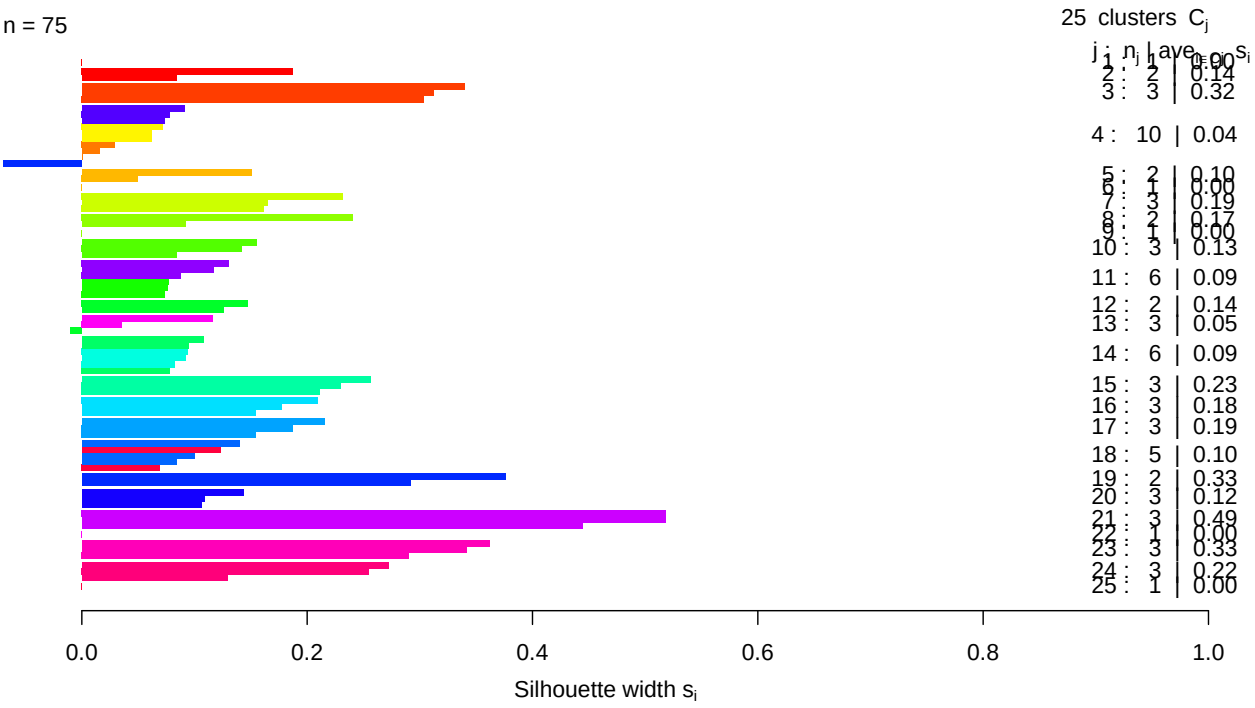
```
adjustedRandIndex(cluster.id, MetaA$author)
```

```
## [1] 0.6028249
```

The **cluster** package can compute silhouette widths as an indicator of clustering quality. Colouring the bars by author is a bit tricky, though. Note that bars for points with a silhouette width of 0 (e.g. for all single-point clusters) are not visible.

```
sil <- silhouette(cluster.id, DMA)
rownames(sil) <- MetaA$author
sil <- sortSilhouette(sil)
```

```
col.map <- rainbow(25)
names(col.map) <- levels(MetaA$author)
plot(sil, col=col.map[rownames(sil)])
```



> **Q:** If a certain number of clusters are desired, it is often better to compute such a "flat" clustering directly. A robust algorithm is PAM (*partitioning around medoids*) implemented in the `pam()` function from **cluster**. Can you work out how to obtain cluster IDs and examine the clustering quality?
>
> **Q**: The **cluster** package also offers algorithms `agnes()` and `diana()` for hierarchical clustering. Can you make them work with the procedure above? (Hint: you will need to call the function `as.hclust()` at some point.)

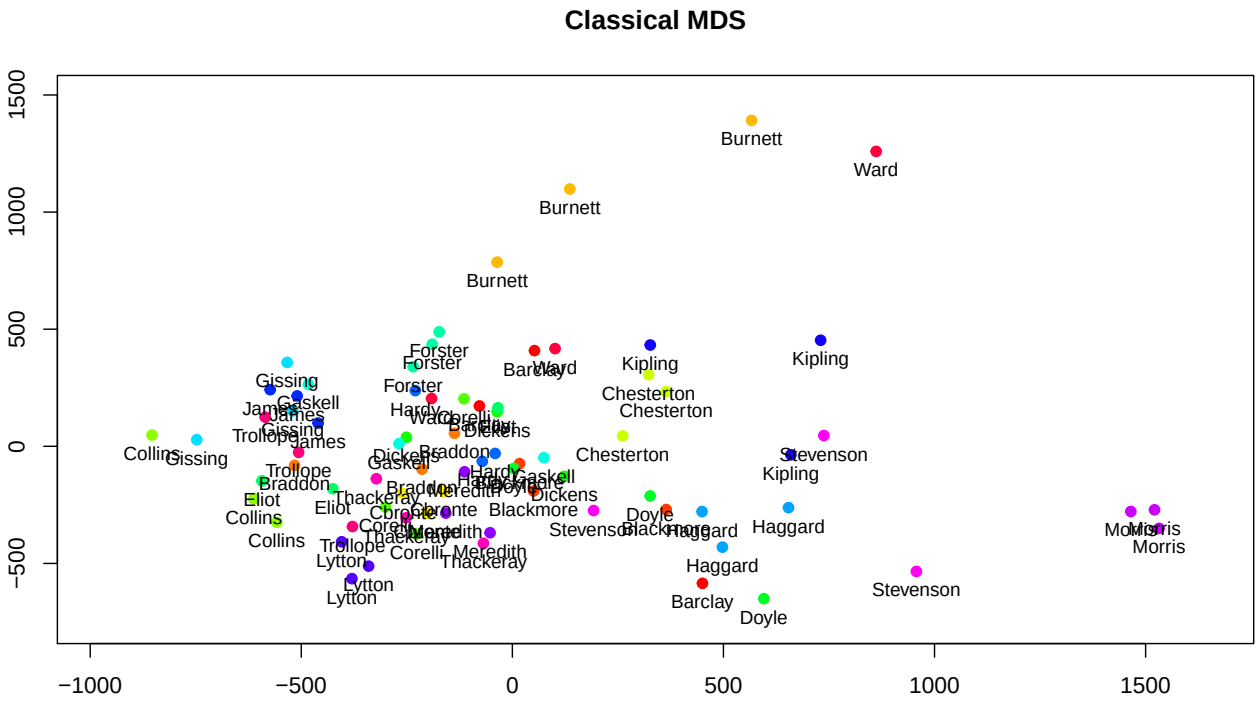
3.2 Topological maps

Multidimensional scaling (MDS) visualizes high-dimensional data in the form of a topological map, which attempts to display data points near each other that are close in the original space. Since we will want to re-do this plot with different mapping algorithms (and perhaps other parameter settings), it's time to define an ad-hoc function – note that most parameters are hard-coded to the data we're currently interested in, which makes the function much easier to write and use.

```
delta.map <- function (xy, main="") {
  xr <- expand.range(xy[, 1], by=.05) # add 5% margin for labels
  yr <- expand.range(xy[, 2], by=.05)
  col.vec <- rainbow(25)[MetaA$author]
  plot(xy, pch=20, cex=1.5, col=col.vec, main=main,
       xlab="", ylab="", xlim=xr, ylim=yr)
  text(coord, labels=MetaA$author, pos=1, cex=.8)
}
```

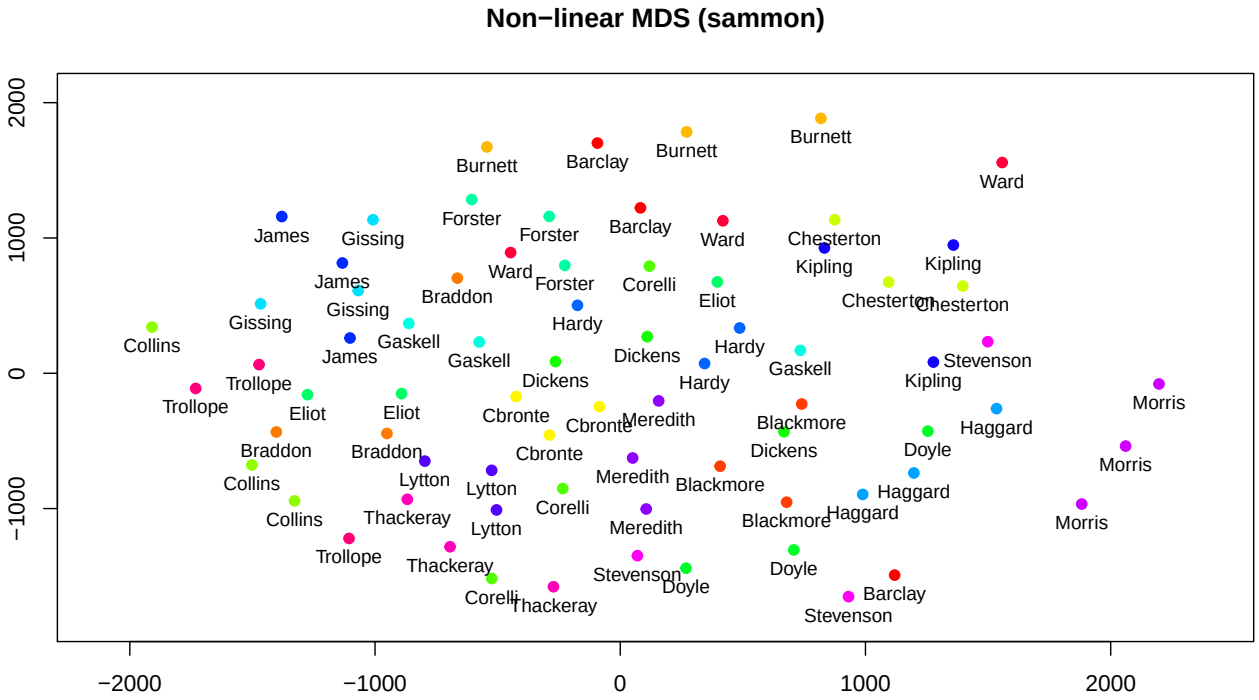
The classical MDS algorithm uses a linear mapping that cannot represent data sets with complex high-dimensional structure.

```
coord <- cmdscale(DMA)
delta.map(coord, main="Classical MDS")
```



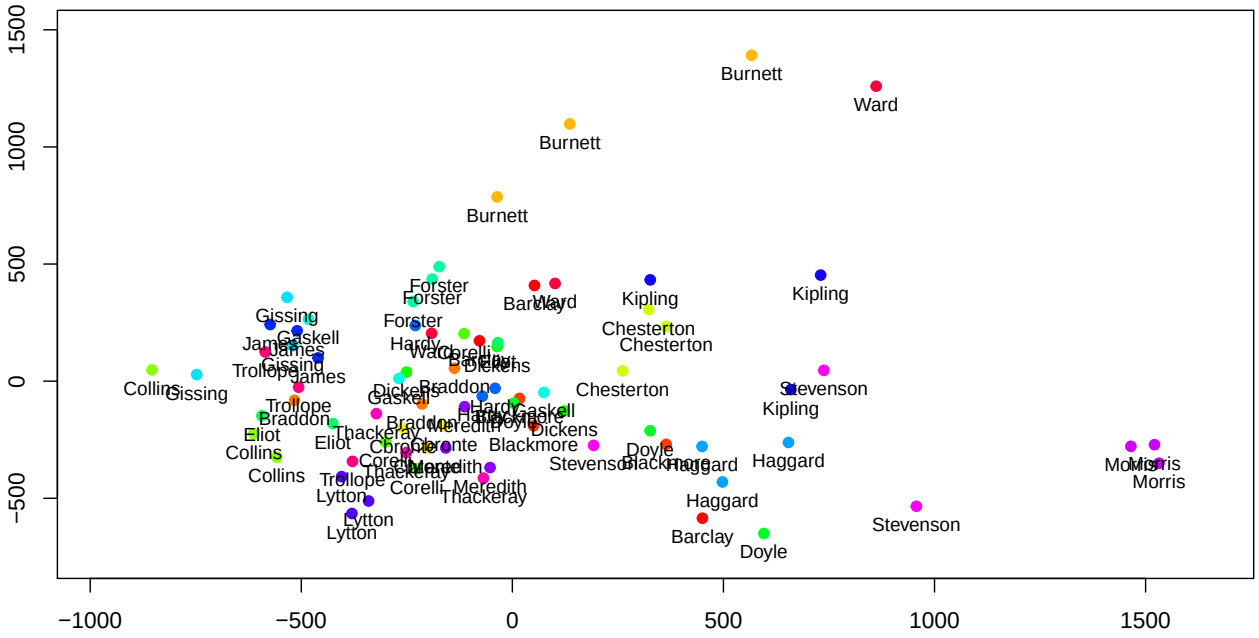
The **MASS** package offers two versions of non-linear MDS. The Sammon algorithm attempts to preserve distances and thus tends to spread out points evenly if all pairwise distances are fairly large. It fails to show interesting topological structure in such cases.

```
coord <- sammon(DMA, trace=FALSE)$points
delta.map(coord, main="Non-linear MDS (sammon)")
```



The other algorithm (attributed to Kruskal) allows for more structure in the map (by penalizing larger distances less severely). However, it can be fairly instable and sometimes fails to improve over classical MDS (used as its initial configuration).

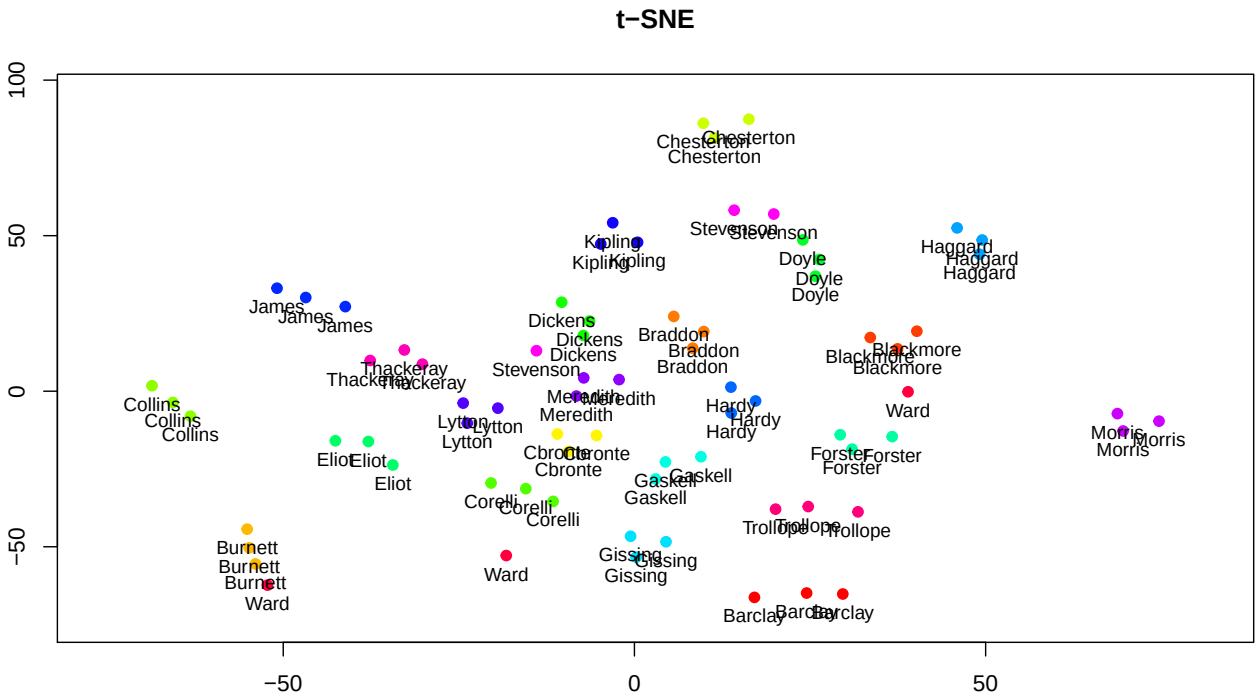
```
coord <- isoMDS(DMA, trace=FALSE)$points
delta.map(coord, main="Non-linear MDS (isoMDS)")
```



A more sophisticated approach attempts to produce a topological mapping that preserves neighbourhood structure but allows for a substantial distortion of the larger geometry. The state-of-the-art algorithm t-SNE (*t-distributed stochastic neighbour embedding*) is currently very popular. We use an implementation in the R

package **Rtsne**. For smaller data sets, the `perplexity` parameter needs to be reduced from its default value of 30. One big disadvantage is that the stochastic algorithm can produce entirely different visualizations depending on the random seed (try setting it to `1`).

```
set.seed(1984)
coord <- Rtsne(DMA, perplexity=10)$Y
delta.map(coord, main="t-SNE")
```



3.3 Exercise

> Explore other distance metrics, clustering methods and visualization parameters. Researchers have found that Delta performs especially well with angular distance (also known as *cosine similarity*). You can computer angular distance with the `dist.matrix()` function from **wordspace** (it is the default setting if no `method` argument is specified), but remember to set `as.dist=TRUE` if you want to use the resulting distance matrix with `hclust()`.

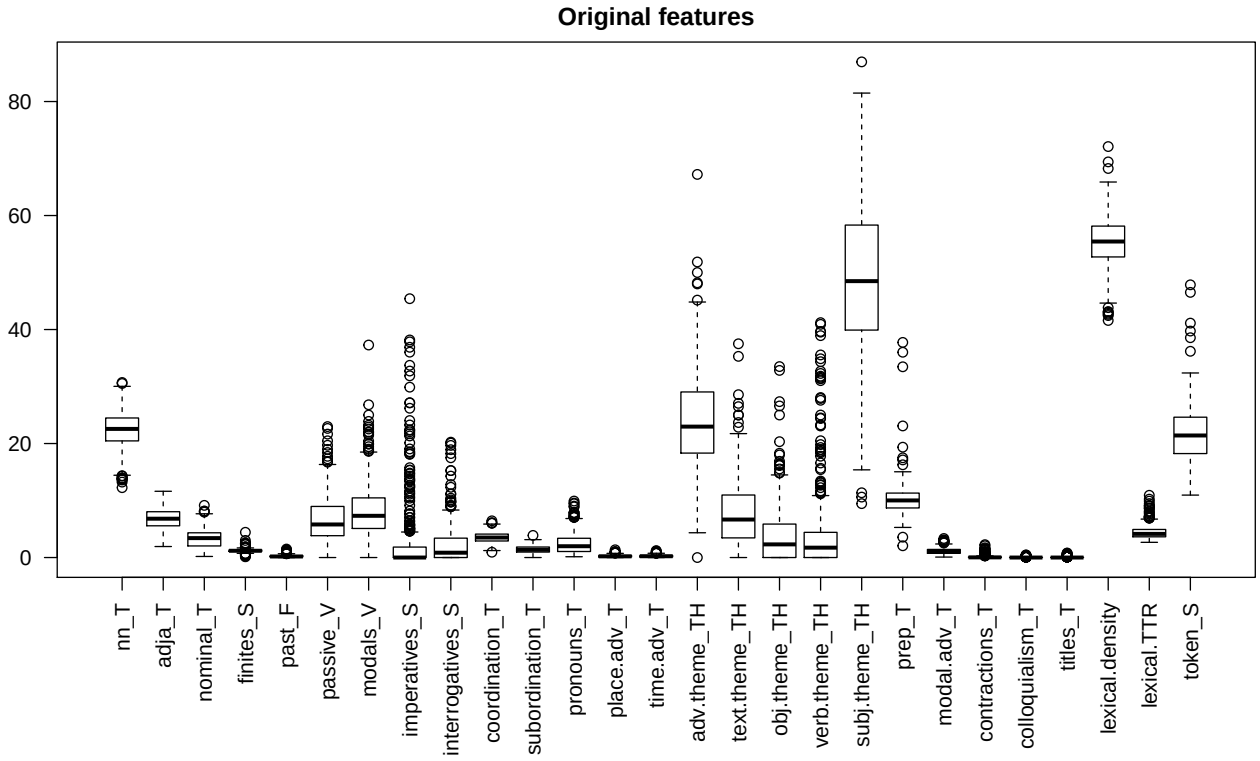
As a starting point, take a look at the parameter settings explored by Evert *et al.* (2017). They found that clustering quality often depends on the number n_w of most frequent words included in the vector representation. Can you work out how to adjust n_w?

- Evert, Stefan; Proisl, Thomas; Jannidis, Fotis; Reger, Isabella; Pielström, Steffen; Schöch, Christof; Vitt, Thorsten (2017). Understanding and explaining Delta measures for authorship attribution. *Digital Scholarship in the Humanities*, **22**(suppl_2), ii4–ii16. https://doi.org/10.1093/llc/fqx023

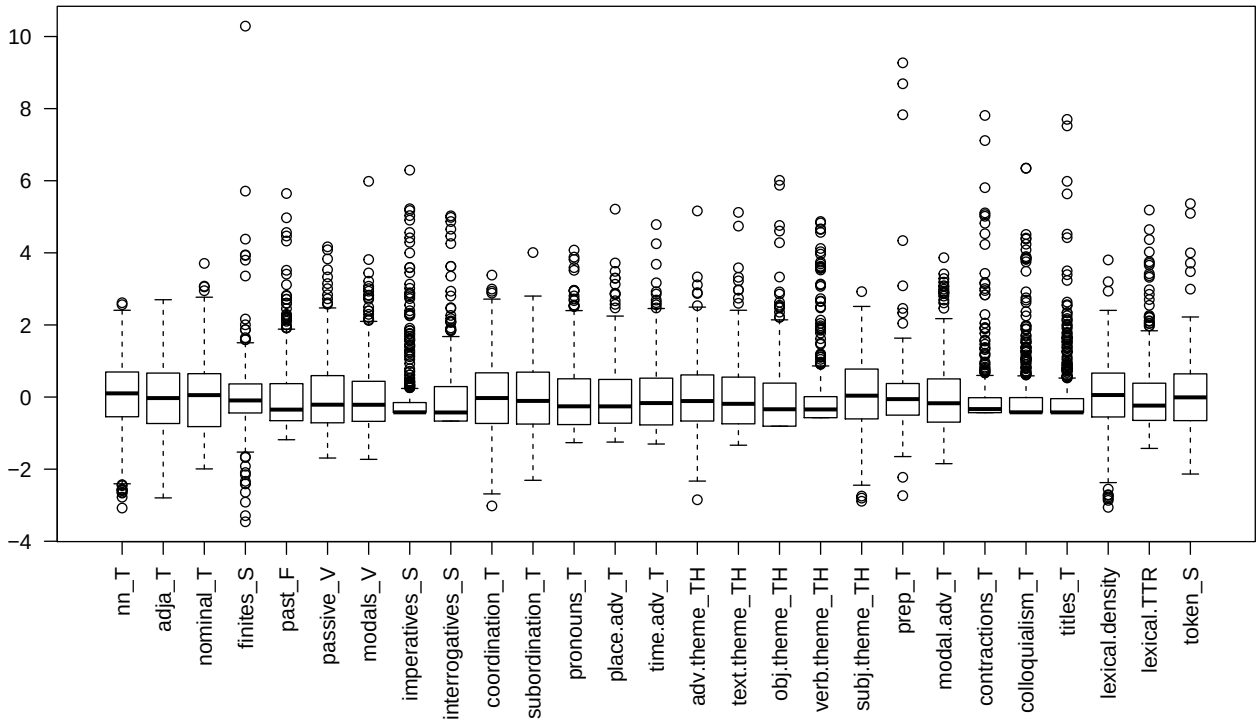
4 CroCo: registers and translation effects

Even though the grammatical features of the CroCo data set do not follow a Zipfian distribution like word frequencies, standardization is essential because different features cover entirely different frequency ranges.

```
par(mar=c(7, 2, 2, 1)) # make room for labels
boxplot(MC, las=2, main="Original features")
```



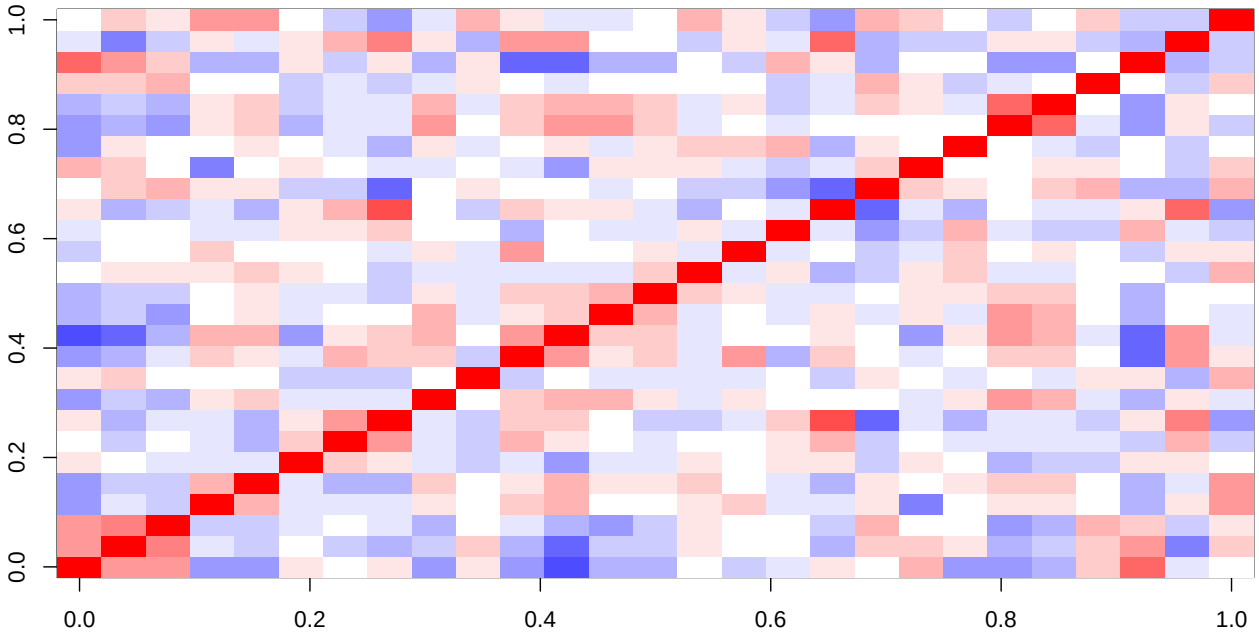

```
ZC <- scale(MC)
par(mar=c(7, 2, 2, 1))
boxplot(ZC, las=2, main="Standardized features")
```



Another preliminary step is to check for collinearities and other overly strong correlations between the features. If there are such strong correlations or conspicuous large blocks in the plot, some features may need to be

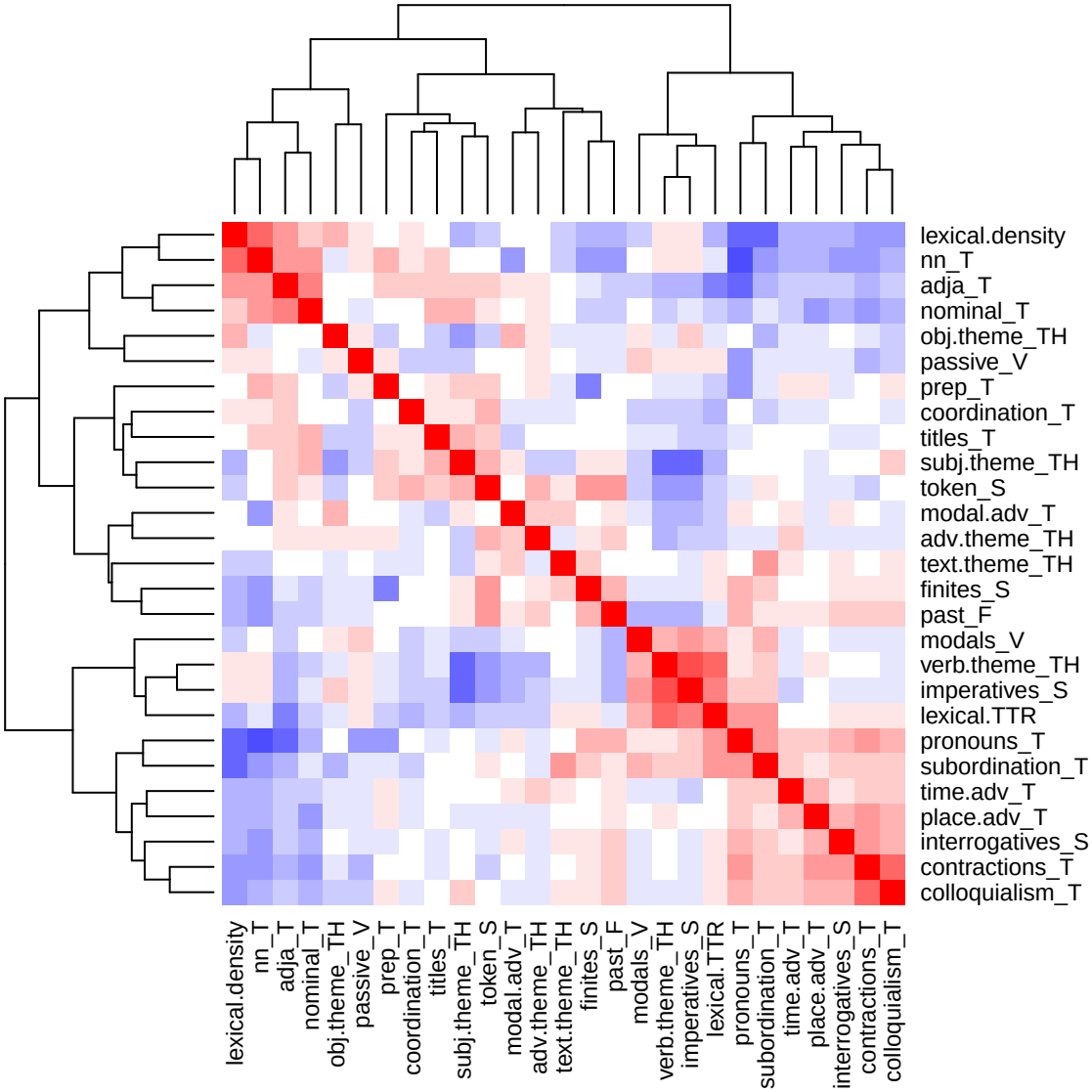
excluded or a different unit of measurement may be required. For interpretability, we need to specify the range $[-1, 1]$ of possible correlation scores and choose a suitable colour scale.

```
cor.colours <- c(
  hsv(h=2/3, v=1, s=(10:1)/10), # blue = negative correlation
  rgb(1,1,1),                   # white = no correlation
  hsv(h=0, v=1, s=(1:10/10))    # red = positive correlation
)
image(cor(ZC), zlim=c(-1, 1), col=cor.colours)
```



A "heatmap" plot automatically groups correlated variables using a clustering algorithm, so the visualization is much easier to interpret.

```
heatmap(cor(ZC), symm=TRUE, zlim=c(-1,1), col=cor.colours, margins=c(7,7))
```



Q: Try to visualize the CroCo data set using MDS or t-SNE, indicating registers, languages or translation status with different colours or shapes. Can you pick out any meaningful patterns? (Hint: the `mvar.pairs()` function introduced below displays metadata categories in a similar way as `mvar.clust()`.)

4.1 Projection with PCA

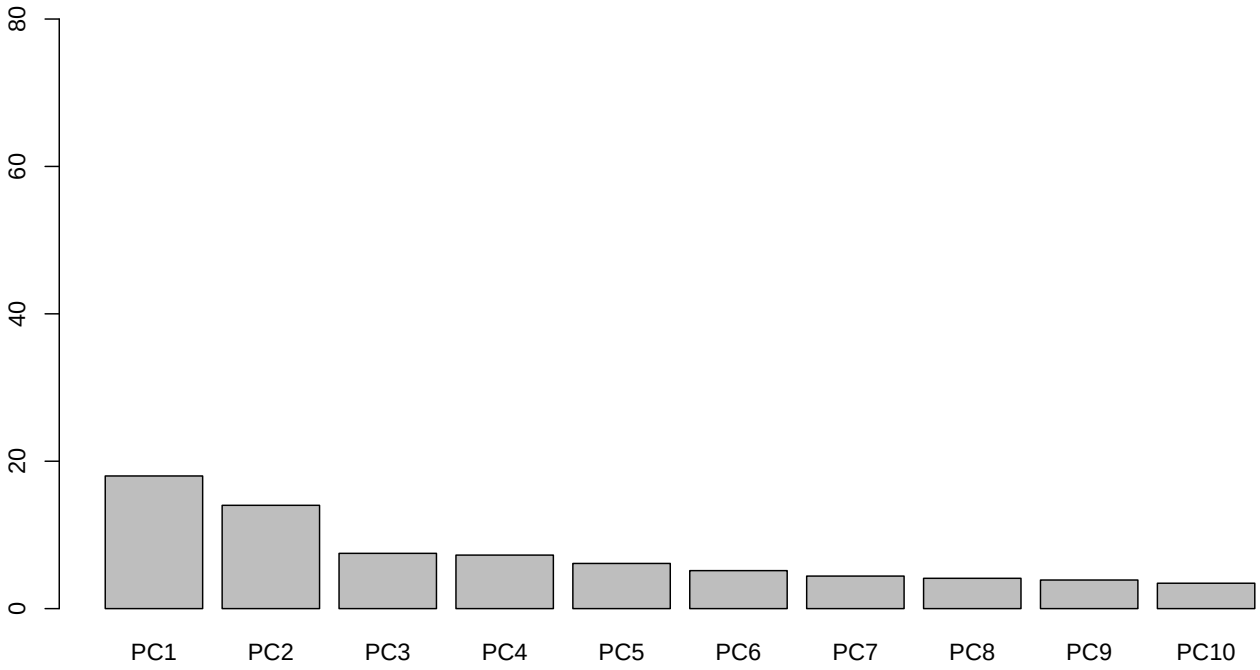
We understand orthogonal projection as a lower-dimensional **perspective** on the geometric configuration of data points in high-dimensional space. This only makes sense in combination with the Euclidean metric, because the orthogonal projection decomposes squared Euclidean distance. A group of support functions (such as `mvar.space()` below) helps you to work with projections without having to carry out all the low-level matrix operations.

The method of **principal component analysis** (PCA) finds a perspective that preserves as much of the distance information as possible. It is not necessary to choose the number of dimensions in advance: PCA returns a sequence of orthogonal basis vectors that represent increasingly larger optimal subspaces. This operation is performed automatically when creating a new `mvar.space` object.

```
PCA <- mvar.space(ZC)
```

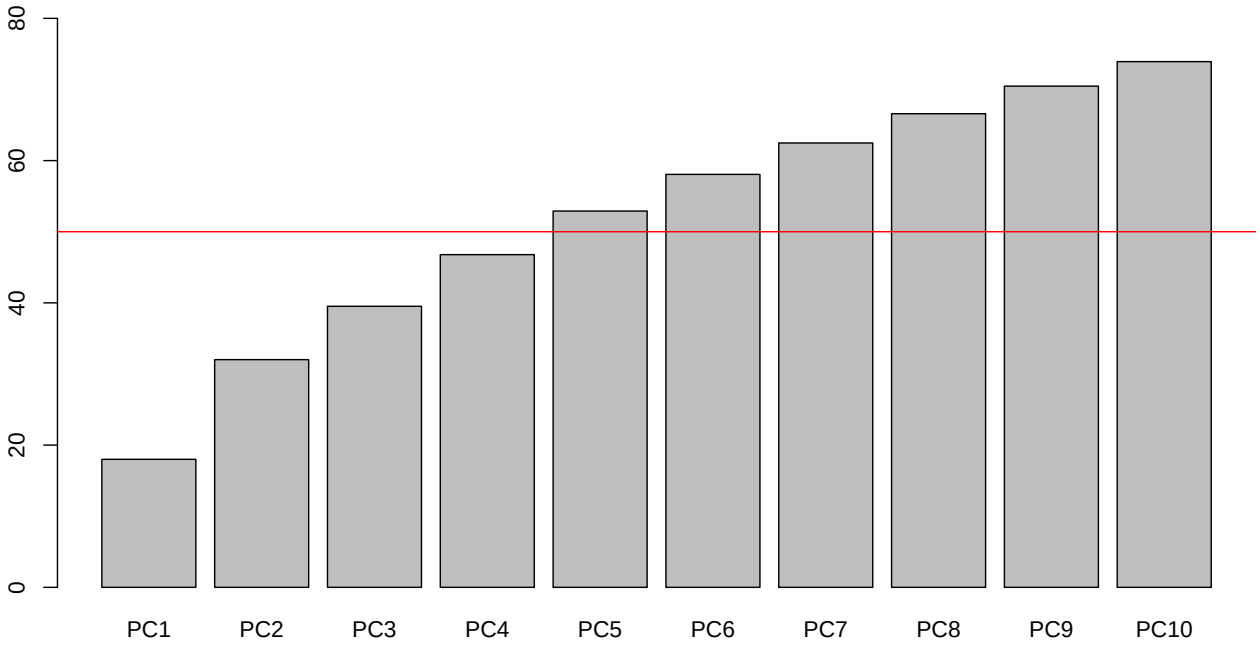
The proportion of variance (R^2) for each basis dimension shows how much of the distance information it captures. Let us look at the first 10 latent dimensions.

```
r2 <- mvar.R2(PCA, dim=1:10)
barplot(r2, ylim=c(0, 80))
```



If we project into the first three dimensions, the total variance captured by the projection is the sum of the first three R^2 values. The barplot below shows that the first 5 dimensions already contain more than half of the (squared Euclidean) distance information.

```
barplot(cumsum(r2), ylim=c(0, 80))
abline(h=50, col="red")
```



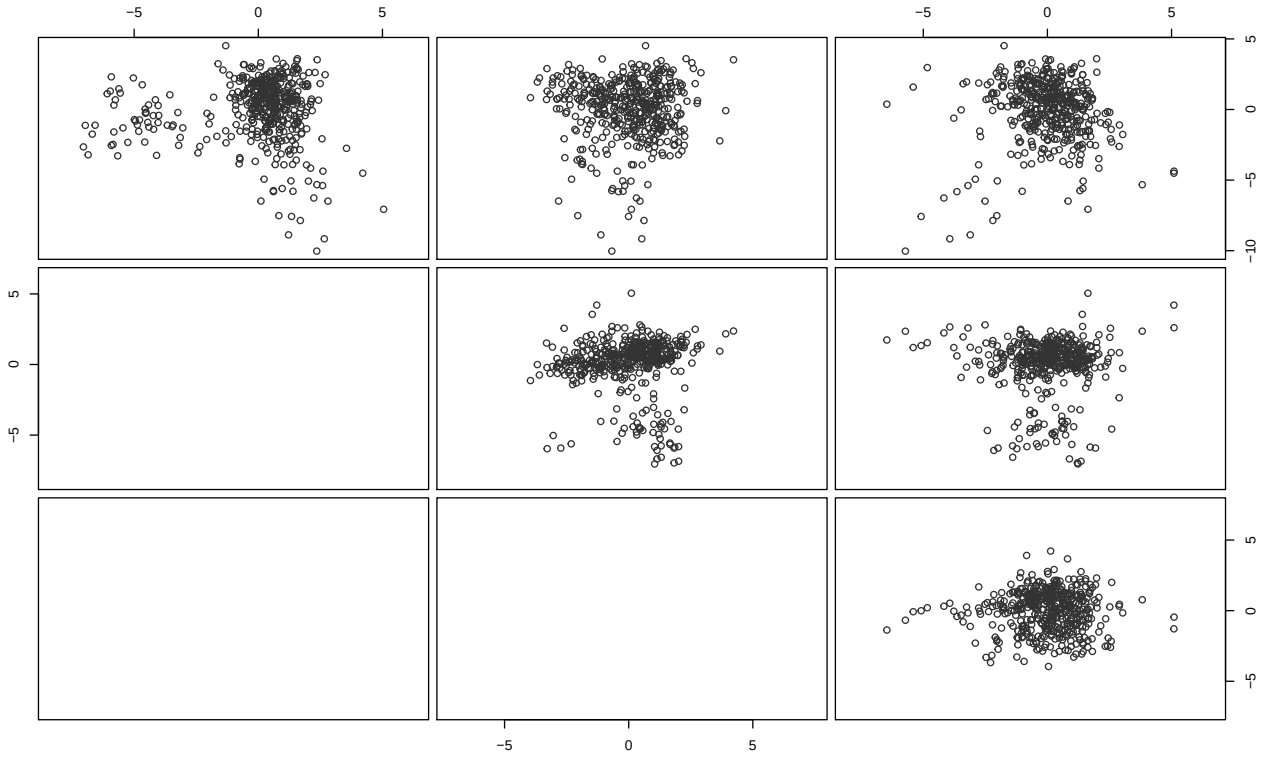
An `mvar.space` object represents a decomposition of the high-dimensional feature space into a **target subspace** and an orthogonal **complement space**, whose basis vectors are determined by the PCA algorithm. In this case, we did not specify a target space, so we obtain a PCA for the complement space.

```
PCA
```

```
## mvar.space object representing projection of 452 x 27 data matrix into 0-dimensional subspace
```

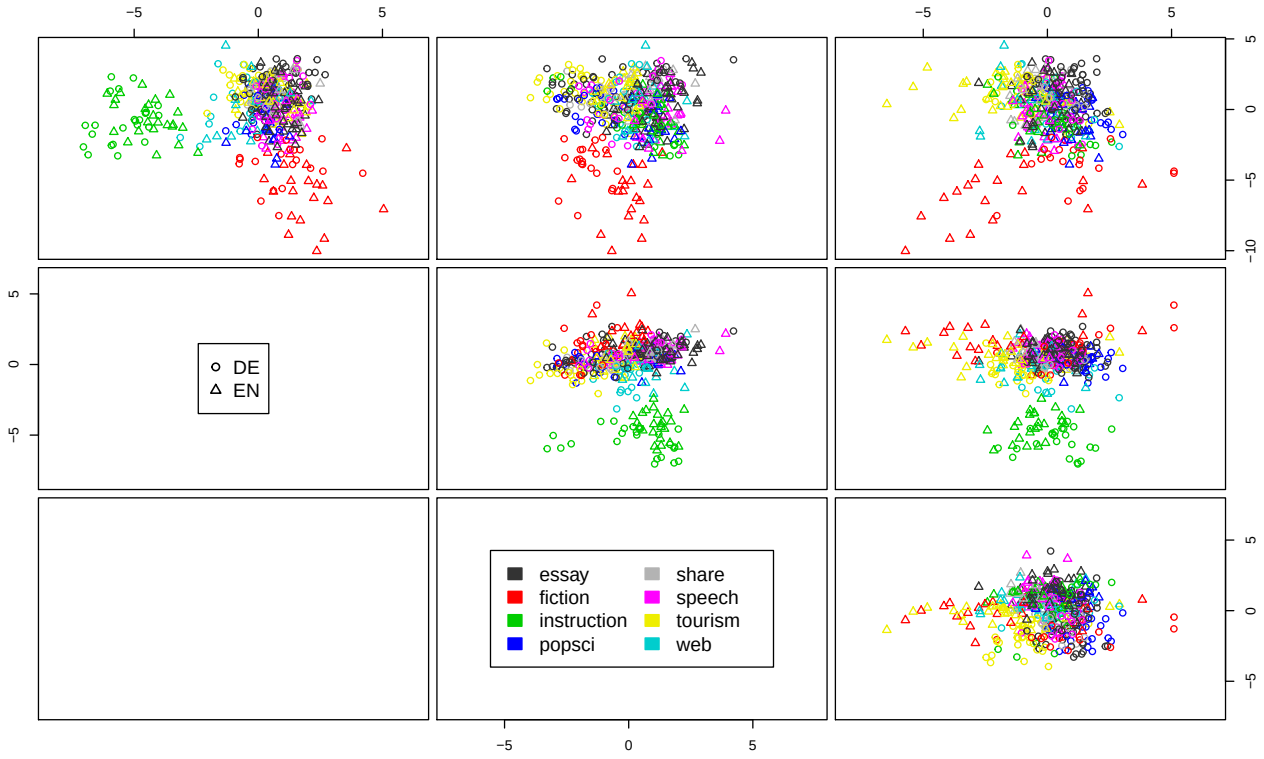
Let us look at the projection into the first four PCA dimensions. Specify `space="both"` to obtain dimensions of the complement space in addition to the (here non-existent) target space. A scatterplot matrix for these dimensions can be drawn with `mvar.pairs()`, which should always be used with the options shown below.

```
Proj4 <- mvar.projection(PCA, space="both", dim=1:4)
mvar.pairs(Proj4, compact=TRUE, iso=TRUE)
```



Geometric perspectives on large data sets are often difficult to interpret without reference to metadata categories such as genre, language and translation status. The scatterplot function accepts a metadata table (which must correspond to the data points in the feature mtarix, of course) so the information shown by plot symbol and colour can be selected by specifying the corresponding column names of the table.

```
mvar.pairs(Proj4, Meta=MetaC, compact=TRUE, iso=TRUE,
           col=register, pch=language)
```



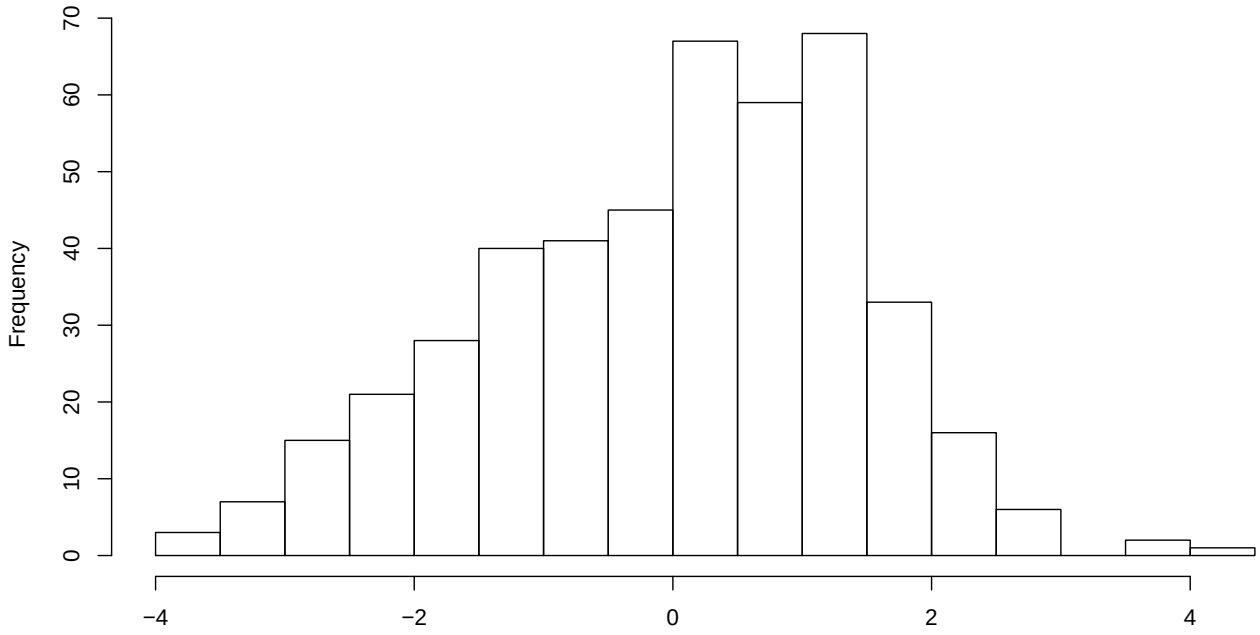
If you have a working installation of the **rgl** package, you can also view an interactive 3D visualization. The `mvar.3d` displays metadata information in the same way as `mvar.pairs`, so it is easy to switch between the scatterplot matrix and the 3D view. By default the first three dimensions are shown and arranged so that the front, left and top view correspond to the top left panels of the scatterplot matrix.

```
mvar.3d(Proj4, Meta=MetaC, iso=TRUE, legend=TRUE,
        col=register, pch=language, size=.1)
view3d(theta=0, phi=0, zoom=.7) # reset to front view
```

4.2 Exploring the PCA dimensions

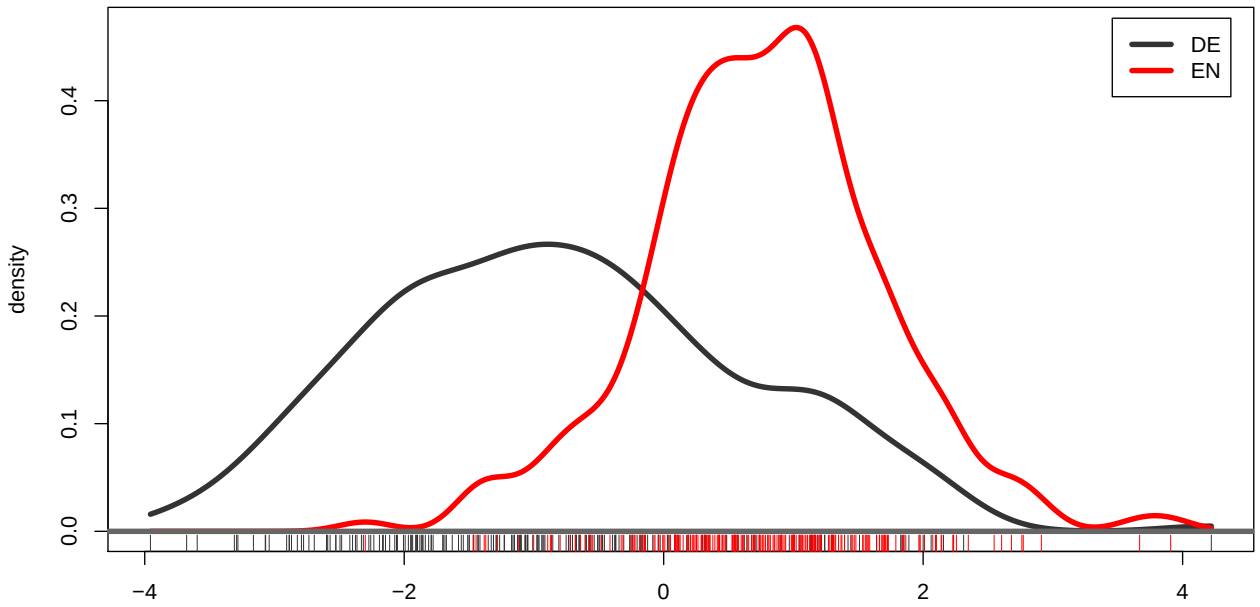
The visualization suggests that PCA dimension 3 separates between English and German texts. In order to take a closer look at the distribution of texts along this axis, we can examine the coordinates along this axis.

```
hist(Proj4[, 3], breaks=20, xlab="", main="")
```



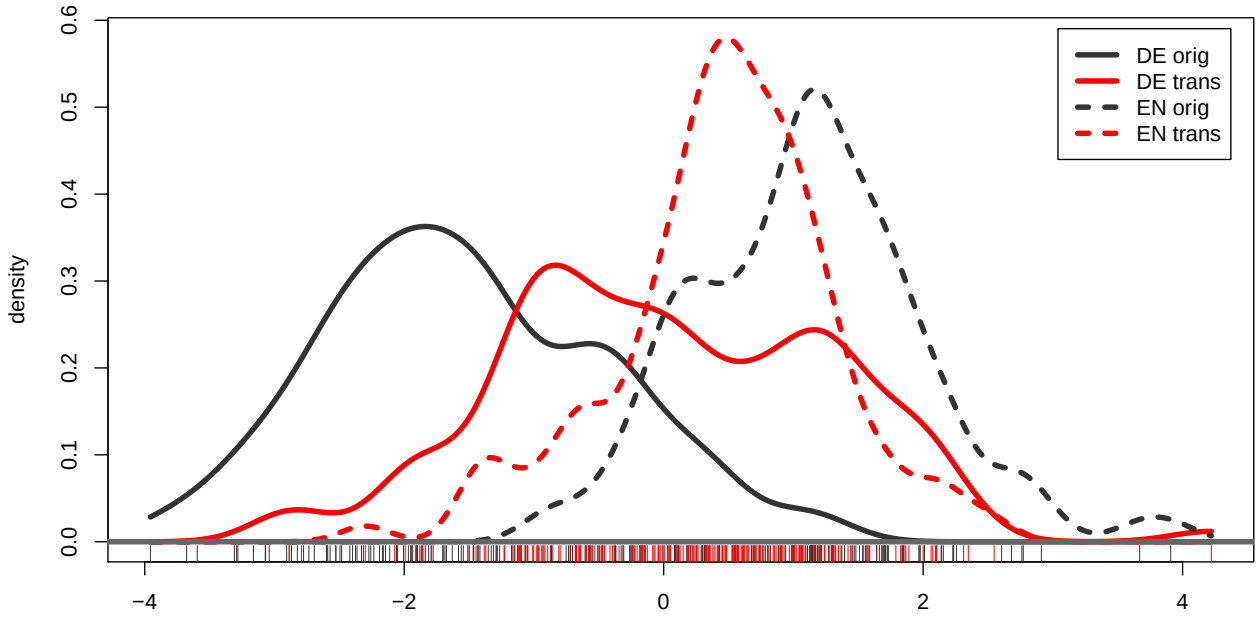
Such a visualization is of little use without separating the German and English texts. The utility function plots separate distributions for the specified categories and allows us to select an arbitrary axis in the space. Here, the axis is simply the third basis vector of the 4-dimensional subspace.

```
discriminant.plot(Proj4, axis.vector(4, 3), MetaC$language,
                  rug=TRUE, xlab="")
```



If we want to separate the distribution by both language and translation status, we need combined categories. Setting line colours and styles appropriately gives the visual impression of two separate categorizations.

```
MetaC <- transform(MetaC, lang_status=paste(language, status))
col.vals <- corpora.palette("simple")[c(1,2,1,2)]
lty.vals <- c("solid", "solid", "32", "32")
discriminant.plot(Proj4, axis.vector(4, 3), MetaC$lang_status,
                  rug=TRUE, xlab="", col.vals=col.vals, lty.vals=lty.vals)
```



The **feature weights** for each PCA dimension are simply the coordinates of the corresponding basis vector. Let us visualize the weights for the first 3 PCA dimensions in a combined barplot. Some fiddling with graphics parameters is necessary to obtain a nicely readable plot. We need to transpose the weights matrix (`t(weights)`) in order to get the desired grouping in the barplot.

```
weights <- mvar.basis(PCA, space="both", dim=1:3)
par(mar=c(8, 4, 0, 0))
barplot(t(weights), beside=TRUE, col=corpora.palette("muted")[2:4],
        las=2, ylim=c(-.5, .5), legend=TRUE, args.legend=list(x="top"))
```