Image analysis in R

Tommaso Leonardi, tl344@ebi.ac.uk

November 27, 2013

For this tutorial we are going to need some external libraries, which can be installed and loaded with:

```
install.packages("tiff")
install.packages("grid")
install.packages("ggplot2")
source("http://bioconductor.org/biocLite.R")
biocLite("EBImage")
library("EBImage")
library("tiff")
library("grid")
library("ggplot2")
```

We can now load the image that we want to analyse:

```
tif_files <- as.Image(readTIFF("astrocytes.tiff"))
colorMode(tif_files) <- "color"
nuc <- t(tif_files[, , 1])
cel <- t(tif_files[, , 3])
img <- rgbImage(green = cel, blue = nuc)
grid.raster(img)
```

The nuc object now contains the channel with the DAPI staining, while the cel object contains the channel with the Vimentin staining. The last two commands combine the two channels in a single RGB image and display it (Figure 1). From this image, we want to determine:

1. The number of cells
2. The average signal intensity per cell
3. The distribution of the areas of the cells

1 Segmenting nuclei

The first step is to identify the nuclei based on the DAPI staining (Figure 2). The simplest approach is to just apply a threshold on the image:

```
nmask <- nuc > 0.3
```

Which would generate a binary image as in Figure 3. However, the EBImage package provides the thresh function, which is slightly more elaborate (type '?thresh' for more info) and gives better results.

```
nmask <- thresh(nuc, w = 35, h = 35, offset = 0.15)
grid.raster(nmask)
```

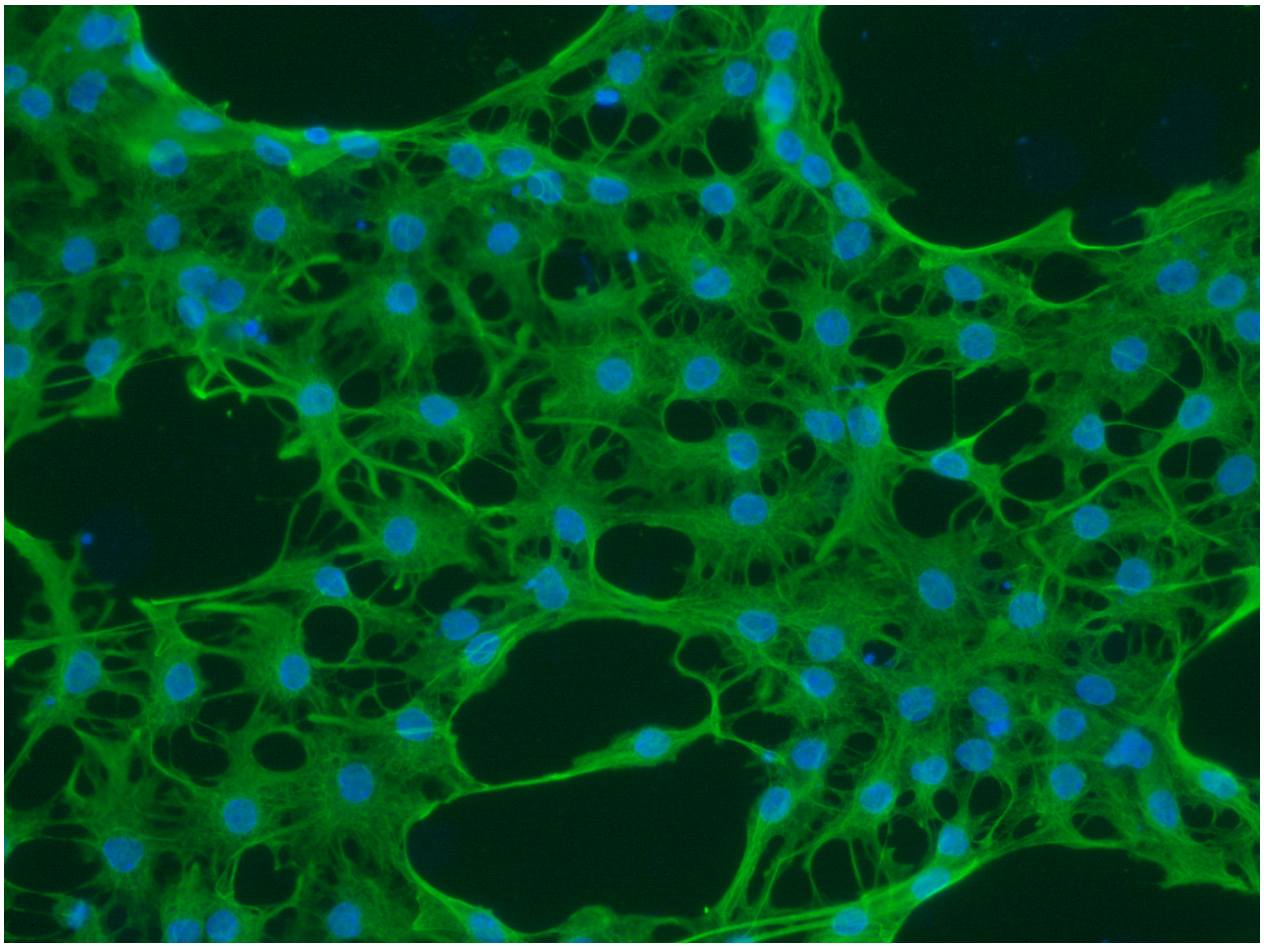

Figure 1: The starting image that we want to analyse

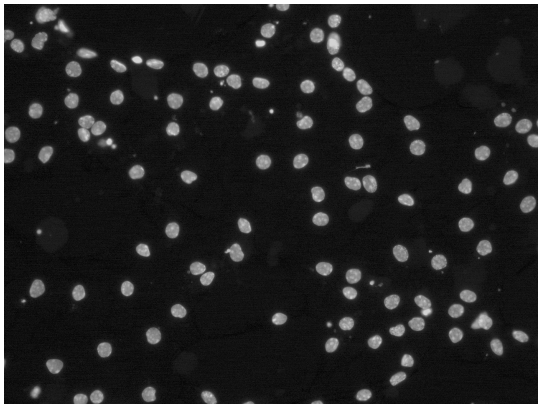

Figure 2: Object nuc (i.e., DAPI channel)

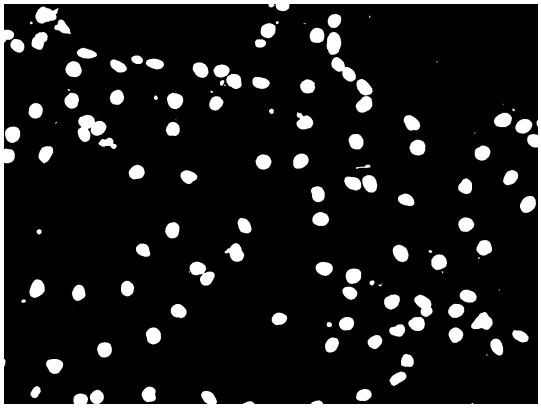

Figure 3: Simple thresholding

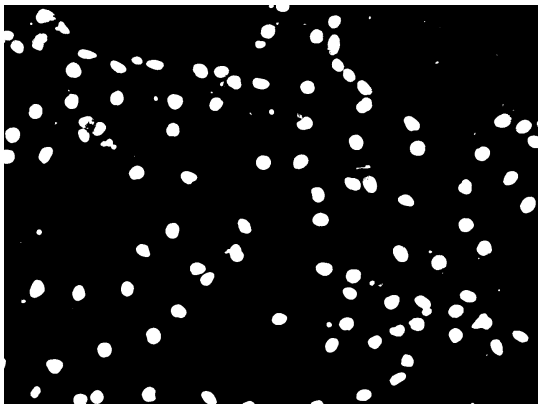

Figure 4: Nuclei thresholded using the thresh() function in EBImage

The thresh function compares the image (nuc) to its filtered (smoothed) version. The w and h parameters in thresh() define the width and the height of the filtering window, while offset specifies the required difference in intensity between the original and filtered versions. The result is displayed in Figure 4.

```
nmask <- opening(nmask, makeBrush(15, shape = "disc"))
nmask <- fillHull(nmask)
nmask <- bwlabel(nmask)
grid.raster(nmask)
```

The opening() function performs an erosion followed by a dilation[1] with a filter of size 15. As you can see in Figure 5, the effect if that we get rid of several small objects that are clearly noise. The last two lines perform to simple actions: fillHull fills the holes inside the segmented nuclei (i.e. sets to 1 all the 0s that are surrounded by 1s), while bwlabel assigns a label to each nucleus.

Finally, we can annotate the original image with the position of the segmented nuclei (Figure 6):

```
colorMode(nmask) <- "Grayscale"
nucgray <- channel(nuc, "rgb")
nucgray <- paintObjects(nmask, nucgray, col = "#ff00ff")
grid.raster(nucgray)
```

2 Segmenting the cell bodies

We first make a mask based on the intensity of the Vimentin channell:

```
ctmask <- opening(cel > 0.3, makeBrush(5, shape = "disc"))
```

And then using the propagate() function we determine the object boundaries in the cel image based previously identified seeds (i.e. the segmented nuclei). The result is displayed in Figure 7.

[1] In very simple terms, erode() removes a bit from the border of every object, while dilate adds a bit to the border of every object. The trick here is that for small objects, the erosion completely removes them.

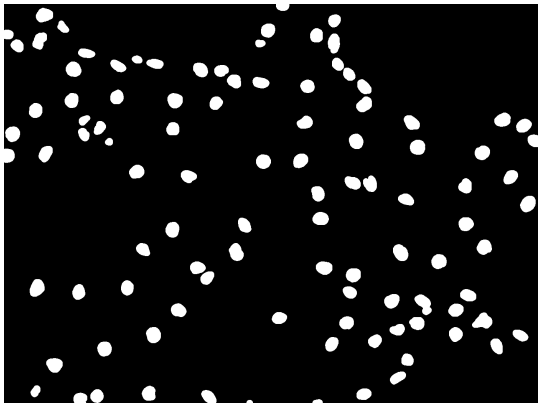

Figure 5: Segmented nuclei after opening() and fillHull()

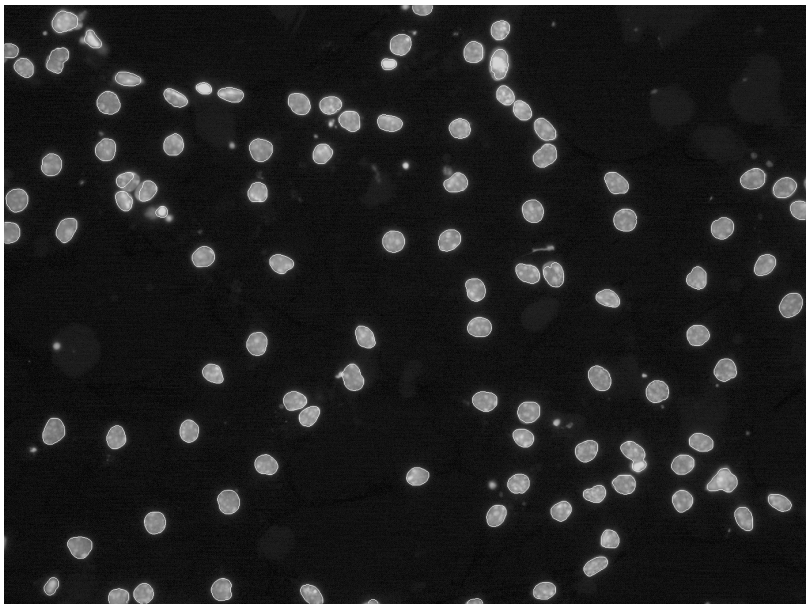

Figure 6: Final image showing segmented nuclei

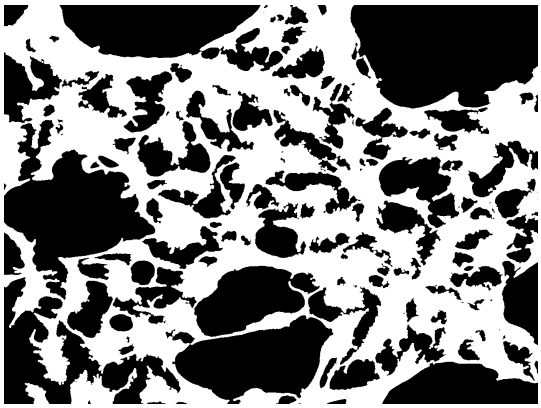

Figure 7: Segmented cell bodies

```
cmask <- propagate(cel, seeds = nmask, mask = ctmask, lambda = 0.1)
cmask <- fillHull(cmask)
grid.raster(cmask)
```

Finally, we can combine our segmented nuclei and cell bodies in a single image (Figure 8):

```
colorMode(cmask) <- "Grayscale"
res <- paintObjects(cmask, img, col = "#FF0000")
res <- paintObjects(nmask, res, col = "#ffff00")
grid.raster(res)
```

3 Computing features

Now that we know the boundaries of each cell we can easily calculate features such as cell size, mean signal intensity, etc... This is easily done using the computeFeatures() function in EBImage. This function comes in three flavours: one to calculate intensities, one to calculate shape and one to calculate moment (i.e. position related things) Let's compute the features and combine the interesting ones in a single dataframe:

```
basic <- computeFeatures.basic(cmask, cel)
shape <- computeFeatures.shape(cmask)
moment <- computeFeatures.moment(cmask)
all_features <- as.data.frame(cbind(basic, shape, moment))
all_features <- all_features[, c(1, 9, 16, 17)]
head(all_features)

##   b.mean s.area m.majoraxis m.eccentricity
## 1 0.4048   2318       91.04         0.9089
## 2 0.4325   5364      107.81         0.8000
## 3 0.4563   5784      114.34         0.7483
## 4 0.4896   3196       87.79         0.7983
## 5 0.4185  10045      241.35         0.9090
## 6 0.3972   7731      154.50         0.8181
```

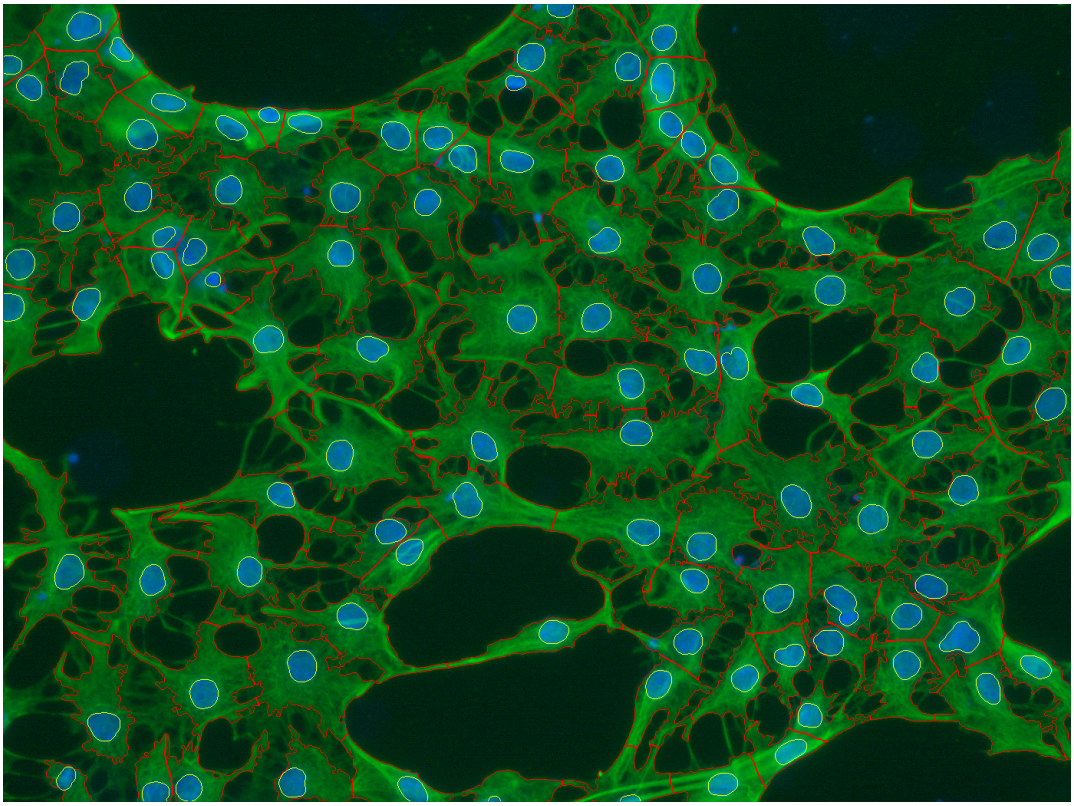

Figure 8: Segmented cell bodies

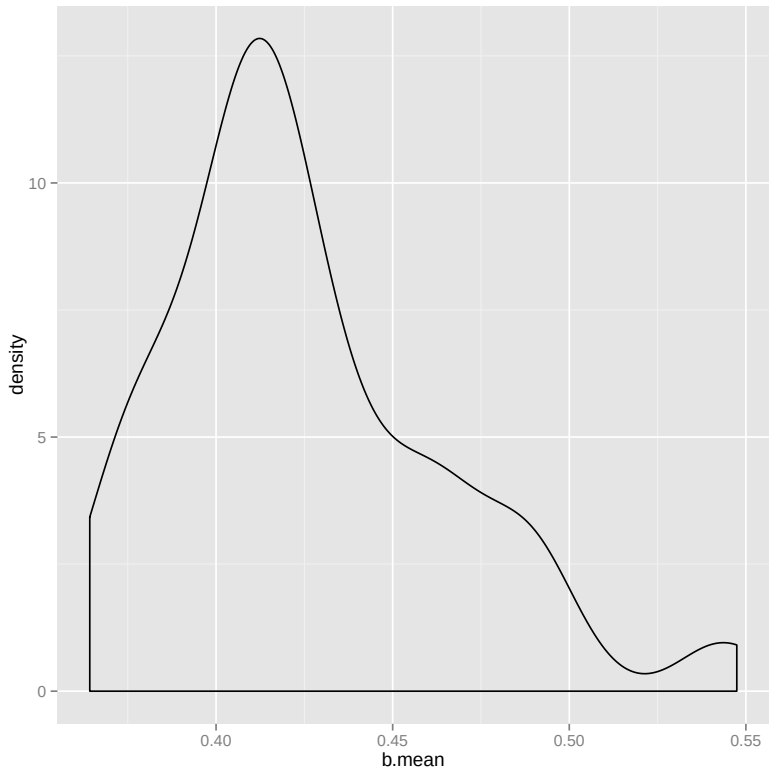

Figure 9: Distribution of Vimentin intensity

The first column is the cell number, followed by the mean Vimentin intensity, the area of the cell, the length of the major axis (in pixels) and the eccentricity. It's now easy to play around with the data; Figure 9, 10 and 11 show the distribution of Vimentin intensity, the distribution of the areas of the cells and a scatterplot with the area plotted against the intensity.

```
ggplot(all_features, aes(b.mean)) + geom_density()
```

```
ggplot(all_features, aes(s.area)) + geom_density()
```

```
ggplot(all_features, aes(x = s.area, y = b.mean)) + geom_point()
```

If we had two different populations of cells in the image, we might also be able to separate them using PCA as in Figure 12.

```
fit <- princomp(all_features, cor = TRUE)
biplot(fit)
```

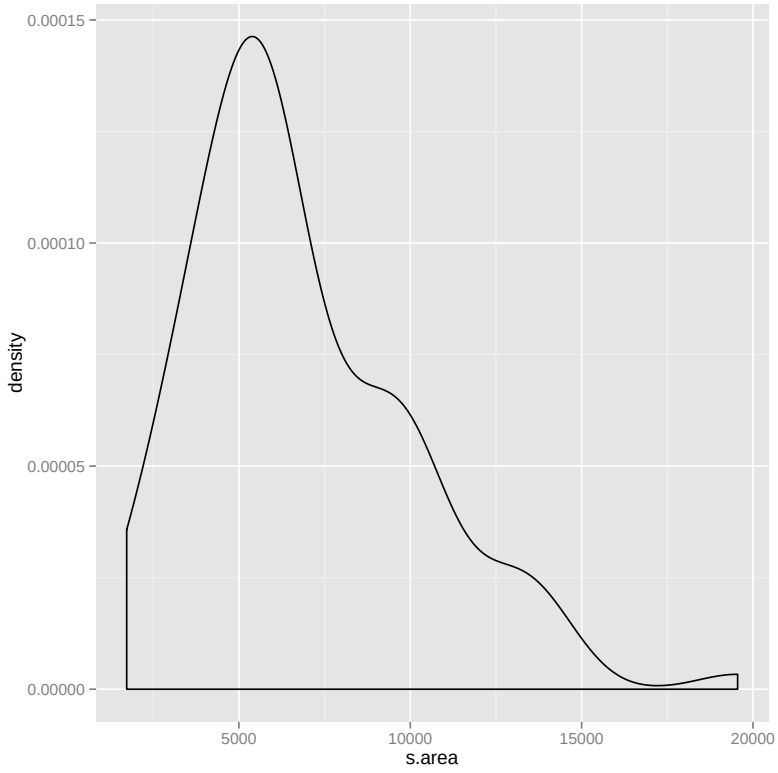

Figure 10: Distribution of areas

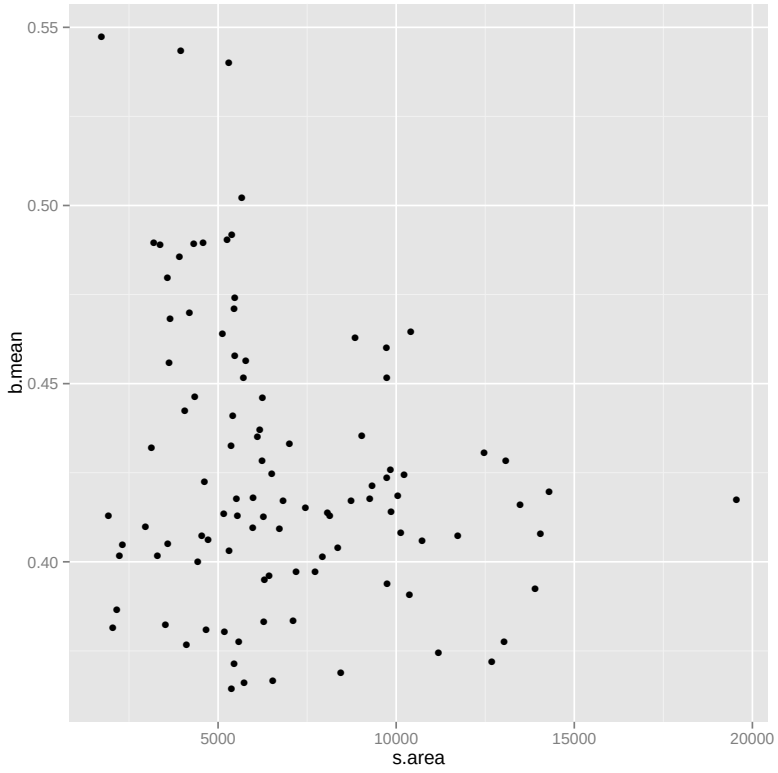

Figure 11: The area of each cell is plotted against its mean intensity

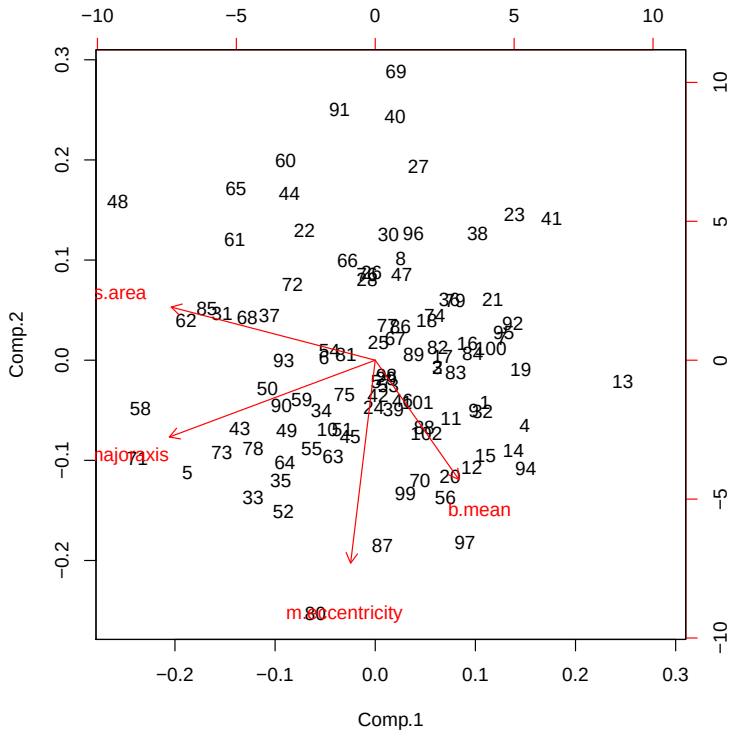

Figure 12: Biplot after Principal Component Analysis