Class Discovery with OOMPA

Kevin R. Coombes

May 27, 2008

Contents

1 Introduction

OOMPA is a suite of object-oriented tools for processing and analyzing large biological data sets, such as those arising from mRNA expression microarrays or mass spectrometry proteomics. The *ClassDiscovery* package in OOMPA provides tools to work on the "class discovery" problem. Class discovery is one of the three primary types of applictions of microarrays described by Richard Simon and colleagues. These are unsupervised methods that are intended to uncover the underlying structure in a data set.

2 Getting Started

No one will be surprised to learn that we start by loading the package into the current R session:

```
> library(ClassDiscovery)
```

The main functions and classes in the ClassDiscovery package work either with data matrices or with `ExpressionSet` objects from the BioConductor *Biobase* package. For the first set of examples in this vignette, we will use simulated data that represents three different groups of samples:

```
> d1 <- matrix(rnorm(100 * 10, rnorm(100, 0.5)), nrow = 100, ncol = 10,
+     byrow = FALSE)
> d2 <- matrix(rnorm(100 * 10, rnorm(100, 0.5)), nrow = 100, ncol = 10,
+     byrow = FALSE)
> d3 <- matrix(rnorm(100 * 10, rnorm(100, 0.5)), nrow = 100, ncol = 10,
+     byrow = FALSE)
> dd <- cbind(d1, d2, d3)
> rm(d1, d2, d3)
```

Because the "raw data" is small by microarray standards, we can use the image command to take a look at it. The *ClassDiscovery* package includes several color maps that are more common in the microarray literature than the color maps that ship with R. These include a green-black-red colormap (obtained via the `redgreen` function), a blue-gray-yellow colormap (from `blueyellow`), shades of red, green, or blue (from `redscale`, `greenscale`, or `bluescale`, respectively) and a "jet" color map (from `jetColors`) that recapitulates the standard MATLAB color map. Two examples are shown in Figure 1.

3 Distances and Clustering

The `dist` function includes a variety of distance metrics commonly used by statisticians. However, it does **not** include the most commonly used metric in the microarray literature, which is based on the Pearson correlation coefficient. In addition, `dist` wants to compute distances between rows, not columns. In most microarray applications, the convention is to store the samples as columns and the genes as rows, and we are typically more interested in clustering the samples. So, we have written a function called `distanceMatrix` that solves both these problems. All the existing distance metrics in `dist` are available through `distanceMatrix`, but some new ones are added. The first example clusters the samples using Pearson correlation (Figure 2). As you can see, the imposed three-group structuire is visible in the dendrogram. Similar results are obtained using Spearman rank correlation instead of the Pearson correlation (Figure 3).

3.1 Colored Clusters

Sometimes we want the known group structure to stand out clearly in a dendrogram, indicated perhaps by color. In our example, we have ten samples from each of three groups, in order. Using `plotColoredClusters`, we can plot the dendrogram with each group indicated using a different color (Figure 4).

```
> par(mfrow = c(2, 1))
> image(1:nrow(dd), 1:ncol(dd), dd, xlab = "genes", ylab = "samples",
+     col = redgreen(64))
> image(1:nrow(dd), 1:ncol(dd), dd, xlab = "genes", ylab = "samples",
+     col = jetColors(64))
> par(mfrow = c(1, 1))
```

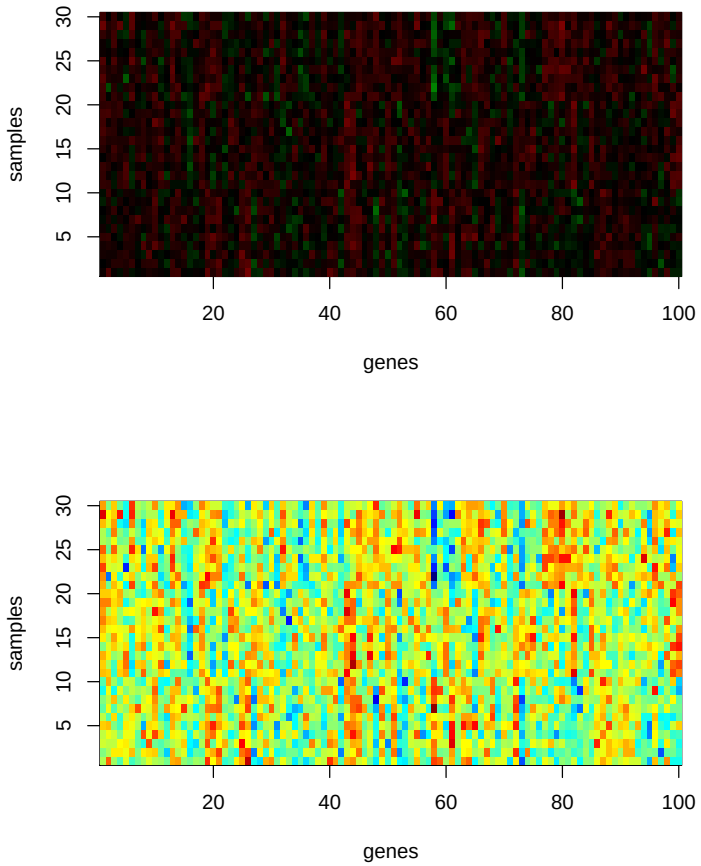

Figure 1: Images of the simulated data using two different color maps.

```
> pearson <- hclust(distanceMatrix(dd, "pearson"), "average")
> plclust(pearson)
```

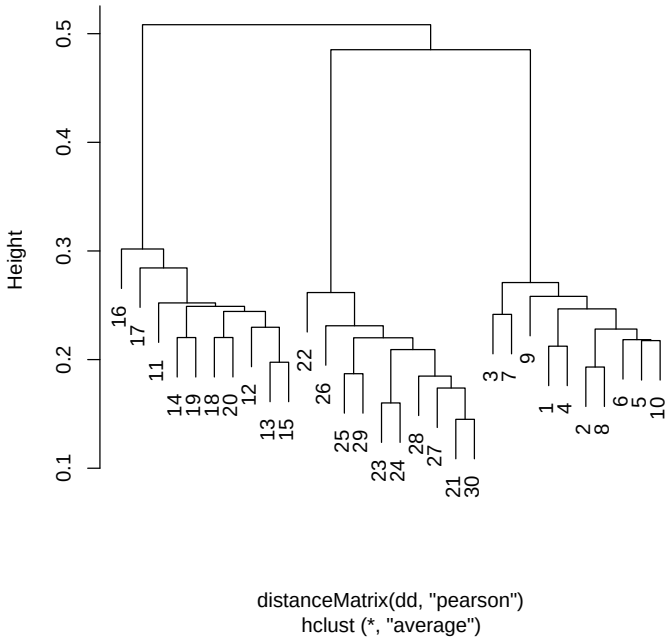

Figure 2: Hierarchical clustering using Pearson correlation to define distances.

```
> spear <- hclust(distanceMatrix(dd, "spearman"), "average")
> plclust(spear)
```

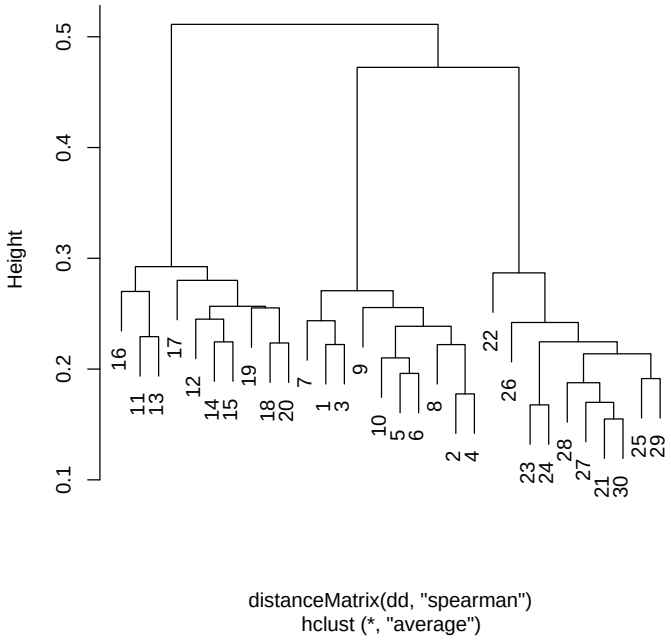

Figure 3: Hierarchical clustering using Spearman rank correlation to define distances.

```
> myColors <- rep(c("red", "blue", "purple"), each = 10)
> myLabels <- paste("Sample", 1:30)
> plotColoredClusters(pearson, myLabels, myColors)
```

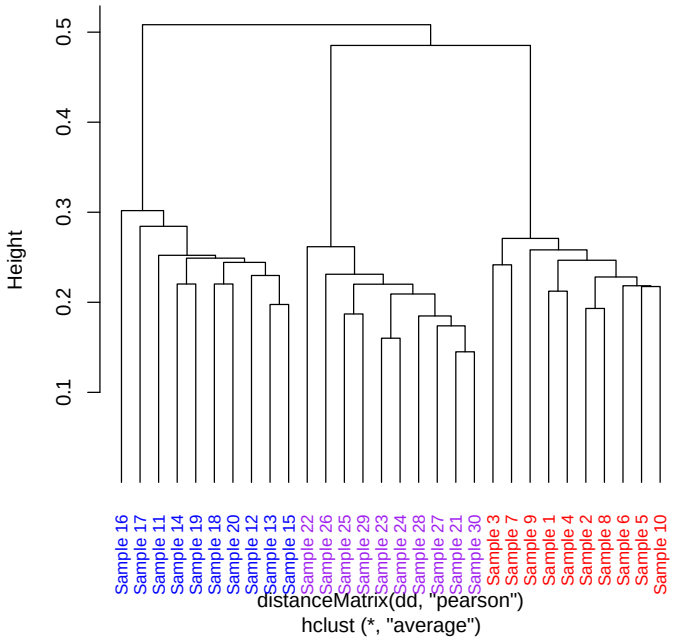

Figure 4: Clustering by Pearson correlation, with the true group structure coded by color.

4 Checking the Robustness of Clusters

One important factor about clustering routines is that they always produce clusters, whether or not clusters are truly present in the data. Thus, it is important to have some tools available to try to determine if the clusters are believable. One way to approach this problem, as decribed by Kerr and Churchill, is to repeat the clustering using a bootstrap. If we are trying to cluster samples, for example, we can use a bootstrap to repeatedly resample the genes used for the clustering, and count how many times each pair of samples ends up in the same branch of the dendrogram. Here is an example, using hierarchical clustering with Pearson correlation and average linkage. Figure 5 displays the results, using a color map that ranges from pure blue (the samples are in the same branch 0% of the time) to pure yellow (the samples are in the same branch 100% of the time).

```
> boot <- BootstrapClusterTest(dd, cutHclust, nTimes = 200, k = 4,
+     metric = "pearson", method = "average", verbose = FALSE)
> summary(boot)

Number of bootstrap samples: 200.
Number of rows sampled: 100.
A BootstrapClusterTest object.

Call:
         BootstrapClusterTest(data = dd, FUN = cutHclust, nTimes = 200, verbose = FALSE, k =

Agreement levels:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.0000  0.0000  0.2859  0.8975  1.0000
```

The default `image` display calls the `heatmap` function, which reclusters the samples based on the bootstrap distance results. You can override this by supplying a starting dendrogram, as in Figure 6.

Because the cluster test requires you to cut the dendrogram at a prespecified level to produce k clusters, the results may be different for different values of k. They can also be different if you change the metric or linkage method. You can also use other clustering methods; the functions `cutPam`, `cutKmeans`, and `cutRepeatedKmeans` can be used instead of `cutHclust`. If you want to write your own version of these functions, they should take an argument `data` to specify the data matrix and an argument `k` to specify the number of desired clusters, and should return a numeric vector containing the class labels (in the range 1 to k) for the samples. Additional optional arguments to control the clustering algorithm can be added as desired, as long as they are given sensible default values.

In some cases, there are not enough rows for a bootstrap resample to adequately reflect the distribution. To deal with this case, we can perform a similar reclustering process, where we add Gaussian white noise to the data matrix

```
> image(boot, col = blueyellow(64))
```

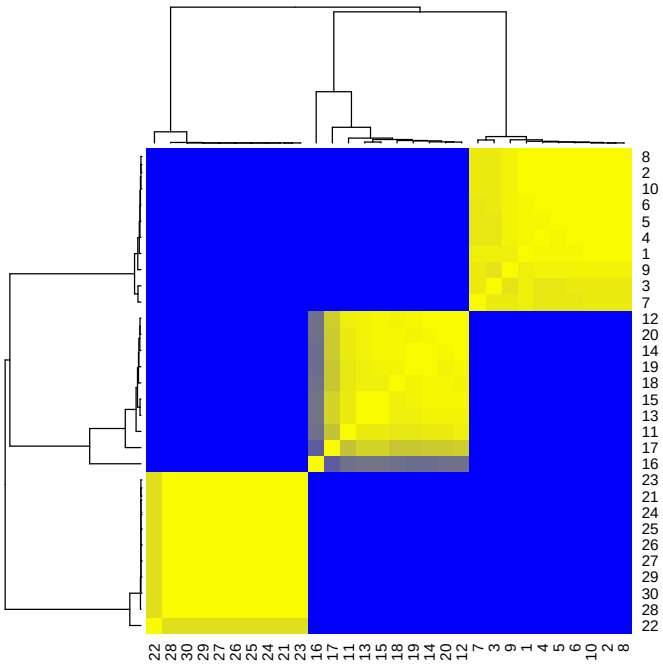

Figure 5: Heatmap of the results of a bootstrap cluster test.

```
> image(boot, dendrogram = pearson, col = blueyellow(64))
```

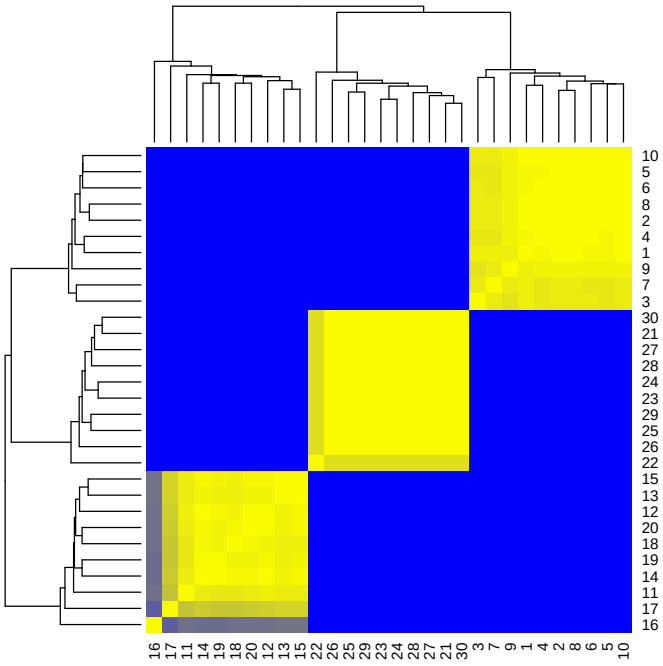

Figure 6: Heatmap of the results of a bootstrap cluster test of the significance of hierarchical clustering using Pearson correlation and average linkage.

instead of using a bootstrap. Here is an example, using k-means to do the clustering (Figure 7).

```
> kper <- PerturbationClusterTest(dd, cutKmeans, k = 4, nTimes = 100,
+     noise = 1, verbose = FALSE)
> summary(kper)

Number of perturbation samples: 100.
Noise level: 1.
A PerturbationClusterTest object.

Call:
         PerturbationClusterTest(data = dd, FUN = cutKmeans, nTimes = 100, noise = 1, verbos

Agreement levels:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.000   0.000   0.010   0.262   0.745   0.970
```

5 Principal Components Analysis

Principal components analysis (PCA) provides an alternative way to see which samples are close to one another. One has to be careful when performing PCA on large data sets, since the default behavior of the `princomp` function is to start by constructing a possibly gigantic covariance matrix. We have implemented the algorithm using a singular value decomposition (SVD) on the original data matrix, which is computationally much more efficient. When there are known classes (as in our example) we can easily display them in color (Figure 8).

We can also select the pairs of principal components (PCs) that we want to display (Figure 9), although the default display of the first two is the one you usually want. In order to figure out how many PCs are important, we can use a "screeplot" (Figure 10). In our example, the first two components appear to carry almost all of the information in the data set.

PCA is fundamentally a geometric procedure based on linear algebra, since it works by choosing a convenient set of directions to serve as axes in a high dimensional space. In some applications, the PCs are used as features (sometimes called "metagenes") to build a classification model. In order to apply these kinds of classifiers to new data sets, you have to be able to project new samples into the same principal component space. To illustrate how this works, we simulate some new data that does not really belong to any of the three classes, and we use the `predict` method to project it into the principal component space. We can then plot the resultsd of the PCA and overlay the projected points (Figure 11).

```
> newdata <- matrix(rnorm(10 * 100), ncol = 10)
> projected <- predict(spca, newdata)
```

```
> image(kper, col = redgreen(128))
```

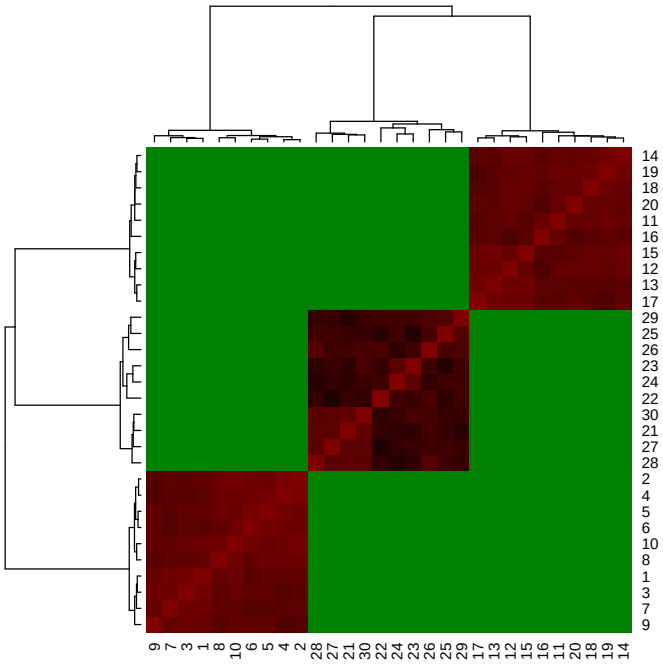

Figure 7: Heatmap of the reproducibility of clustering using k-means under repeated perturbations of the data.

```
> trueClasses <- factor(rep(c("A", "B", "C"), each = 10))
> spca <- SamplePCA(dd, trueClasses)
> plot(spca, col = c("red", "blue", "purple"))
```

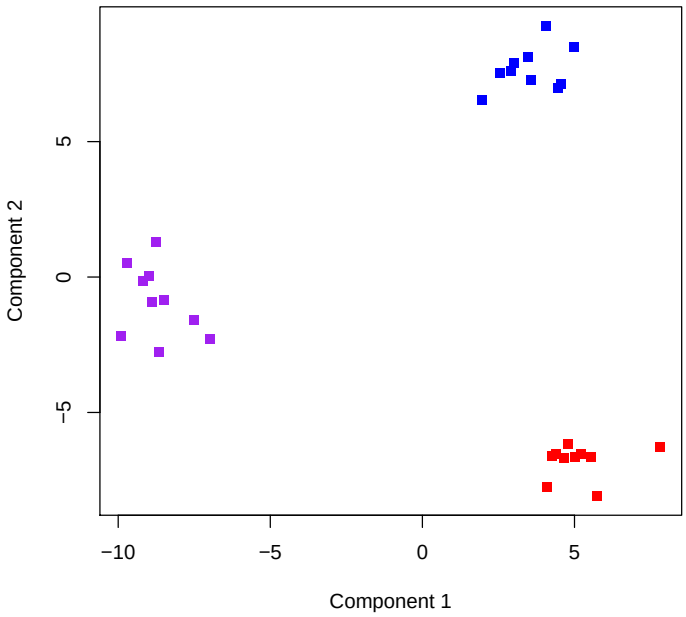

Figure 8: PCA plot of the first two principal components.

```
> plot(spca, col = c("red", "blue", "purple"), which = c(1, 3))
```

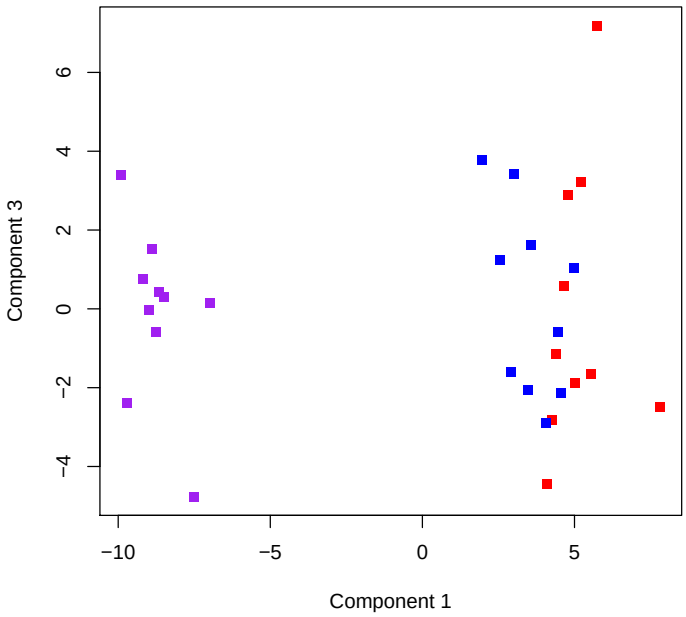

Figure 9: PCA plot of the first and third principal components.

```
> screeplot(spca)
```

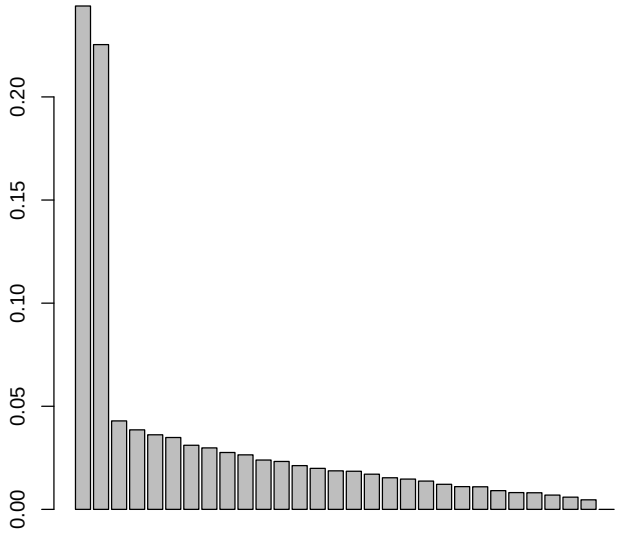

Figure 10: A "screeplot" of the PCA, which shows the percentage of variance explained by each component.

```
> plot(spca, col = c("red", "blue", "purple"))
> points(projected[, 1], projected[, 2], pch = 16)
```

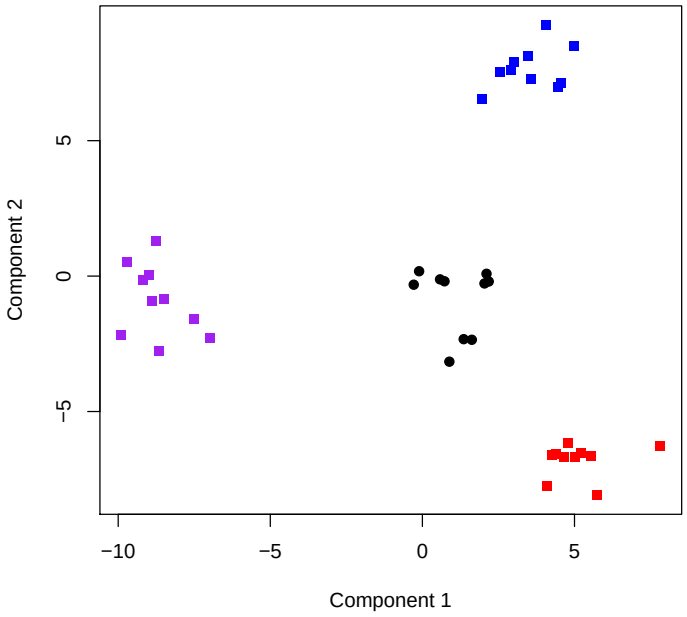

Figure 11: PCA plot, along with projections of a new data set.

6 Mosaics: red-green heatmaps

<Sarcasm> When Mike Eisen and colleagues invented clustering, they also introduced the now-ubiquitous red-green heatmaps beloved by color-blind researchers around the world. As a result, everyone working with microarrays has to be able to produce the same kinds of pictures in order to be taken seriously. </Sarcasm>

Our answer to this challenge is the `Mosaic` class. We can use our simulated data as an example, following the usual R convention in which we first construct an object and then print it or display it (Figure 12). The summary tells us which distance metrics and linkage methods were used to contruct the object. The plot command then gives a simple interface to get the desired figure.

```
> mose <- Mosaic(dd, sampleMetric = "spearman", geneMetric = "pearson")
> summary(mose)

My mosaic, an object of the Mosaic class.

Call:
        Mosaic(data = dd, sampleMetric = "spearman", geneMetric = "pearson")

Sample dendrogram constructed with "average" linkage and "spearman" distance metric.

Gene dendrogram constructed with "average" linkage and "pearson" distance metric.
```

Implementation Note: The hardest part of this whole thing was being able to control the aspect ratio of the heatmap. This feature is not part of the `heatmap` function in the *stats* package. Our solution was to modify the code from that function to produce a new function that we call `aspectHeatmap`. This modification added even more parameters to a function that was already almost unusable in the hands of novice R users. (This claim is based on three years experience teaching a course on the analysis of microarray data to a mixture of statisticians and biologists.) The *Mosaic* class hides most of this complexity; the only things you really neeed to know about are the `hExp` and `wExp` parameters that act as expansion factors for the height and width of the figure, respectively.

7 Class discovery with ExpressionSets

As we mentioned earlier, the main functions in the *ClassDiscovery* work with ExpressionSets as well as with plain old data matrices. For example, we can load a sample data set from the *Biobase* package.

```
> library(Biobase)
> data(sample.ExpressionSet)
> s <- sample.ExpressionSet
> s
```

```
> plot(mose, hExp = 3, col = redgreen(64))
```

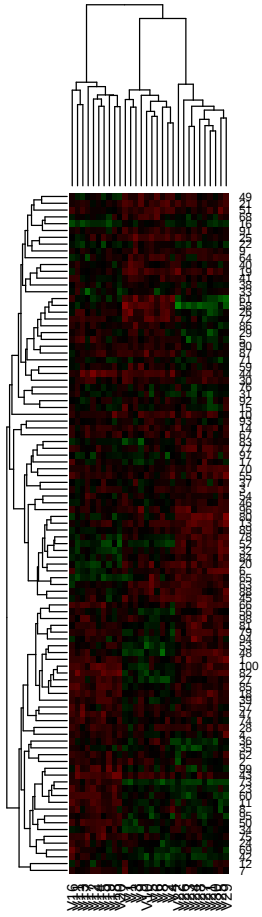

Figure 12: Red-green heatmap based on two-way clustering of the data.

```
> plclust(hclust(distanceMatrix(s, "pearson"), "average"))
```

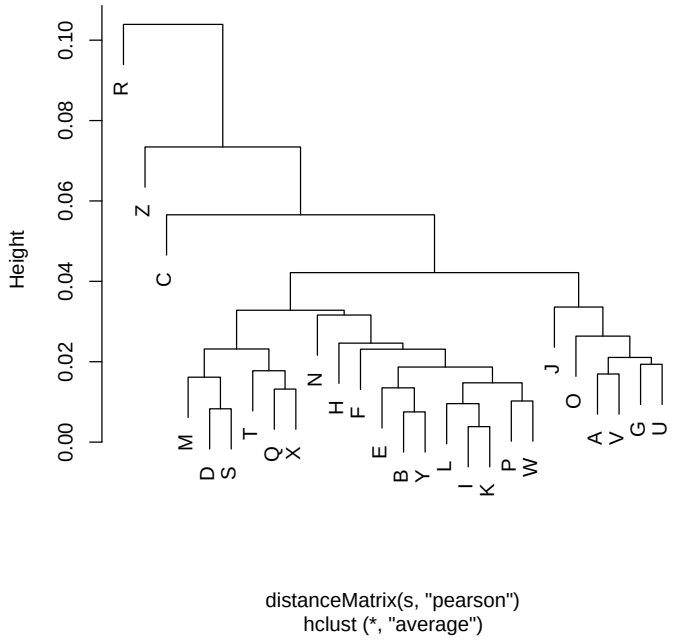

Figure 13: Hierarchical clustering of a sample ExpressionSet.

```
ExpressionSet (storageMode: lockedEnvironment)
assayData: 500 features, 26 samples
  element names: exprs, se.exprs
phenoData
  sampleNames: A, B, ..., Z  (26 total)
  varLabels and varMetadata description:
    sex: Female/Male
    type: Case/Control
    score: Testing Score
featureData
  featureNames: AFFX-MurIL2_at, AFFX-MurIL10_at, ..., 31739_at  (500 total)
  fvarLabels and fvarMetadata description: none
experimentData: use 'experimentData(object)'
Annotation: hgu95av2
```

```
> plot(SamplePCA(s))
```

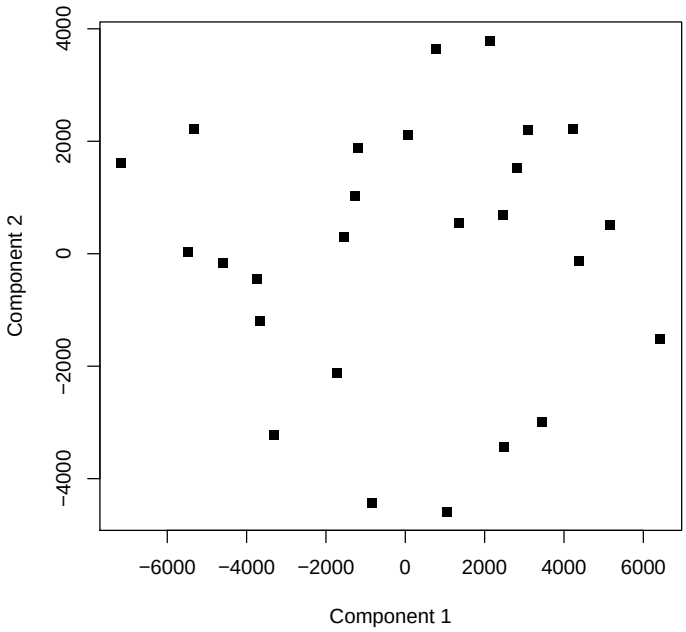

Figure 14: Plot of the first two principal components of a sample ExpressionSet.

```
> plot(Mosaic(s), hExp = 3, col = blueyellow(64))
```

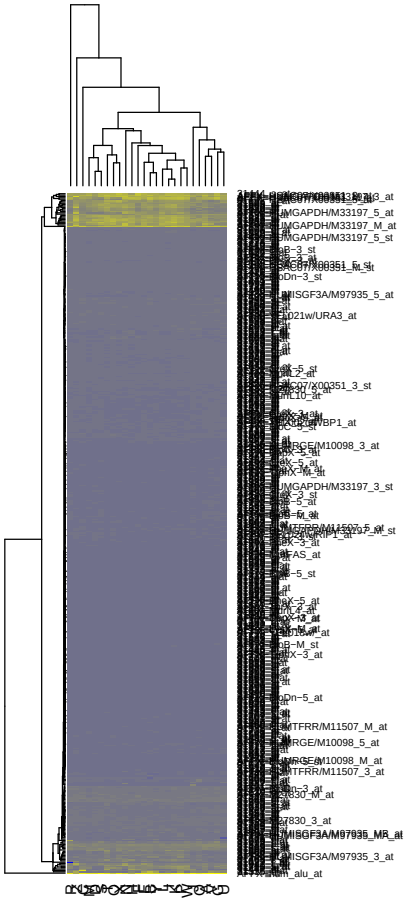

Figure 15: Plot of the first two principal components of a sample ExpressionSet.