A tutorial for Discriminant Analysis of Principal Components (DAPC) using *adegenet* 2.0.0

Thibaut Jombart, Caitlin Collins *

Imperial College London

MRC Centre for Outbreak Analysis and Modelling

June 23, 2015

Abstract

This vignette provides a tutorial for applying the Discriminant Analysis of Principal Components (DAPC [1]) using the *adegenet* package [2] for the R software [3]. This methods aims to identify and describe genetic clusters, although it can in fact be applied to any quantitative data. We illustrate how to use `find.clusters` to identify clusters, and `dapc` to describe the relationships between these clusters. More advanced topics are then introduced, such as advanced graphics, assessing the stability of DAPC results and using supplementary individuals.

*tjombart@imperial.ac.uk, caitlin.collins12@imperial.ac.uk

Contents

1 Introduction

Investigating genetic diversity using multivariate approaches relies on finding synthetic variables built as linear combinations of alleles (i.e. new-variable = $a_1$allele$_1$ + $a_2$allele$_2$ + ... where a_1, a_2 etc. are real coefficients) and which reflect as well as possible the genetic variation amongst the studied individuals. However, most of the time we are not only interested in the diversity amongst individuals, but also and possibly more so in the diversity between groups of individuals. Typically, one will be analysing individual data to identify populations, or more largely genetic clusters, and then describe these clusters.

A problem occuring in traditional methods is they usually focus on the entire genetic variation. Genetic variability can be decomposed using a standard multivariate ANOVA model as:

$$\text{total variance} = (\text{variance between groups}) + (\text{variance within groups})$$

or more simply, denoting $\mathbf{X}$ the data matrix:

$$VAR(\mathbf{X}) = B(\mathbf{X}) + W(\mathbf{X})$$

Usual approaches such as Principal Component Analysis (PCA) or Principal Coordinates Analysis (PCoA / MDS) focus on $VAR(\mathbf{X})$. That is, they only describe the global diversity, possibly overlooking differences between groups. On the contrary, DAPC optimizes $B(\mathbf{X})$ while minimizing $W(\mathbf{X})$: it seeks synthetic variables, the *discriminant functions*, which show differences between groups as best as possible while minimizing variation within clusters.

2 Identifying clusters using `find.clusters`

2.1 Rationale

DAPC in itself requires prior groups to be defined. However, groups are often unknown or uncertain, and there is a need for identifying genetic clusters before describing them. This can be achieved using k-means, a clustering algorithm which finds a given number (say, k) of groups maximizing the variation between groups, $B(\mathbf{X})$. To identify the optimal number of clusters, k-means is run sequentially with increasing values of k, and different clustering solutions are compared using Bayesian Information Criterion (BIC). Ideally, the optimal clustering solution should correspond to the lowest BIC. In practice, the 'best' BIC is often indicated by an elbow in the curve of BIC values as a function of k.

While k-means could be performed on the raw data, we prefer running the algorithm after transforming the data using PCA. This transformation has the major advantage of reducing the number of variables so as to speed up the clustering algorithm. Note that this does not imply a necessary loss of information since all the principal components (PCs) can be retained, and therefore all the variation in the original data. In practice however, a reduced number of PCs is often sufficient to identify the existing clusters, while making the analysis essentially instantaneous.

2.2 In practice

Identification of the clusters is achieved by `find.clusters`. This function first transforms the data using PCA, asking the user to specify the number of retained PCs interactively unless the argument `n.pca` is provided. Then, it runs k-means algorithm (function `kmeans` from the *stats* package) with increasing values of k, unless the argument `n.clust` is provided, and computes associated summary statistics (by default, BIC). See `?find.clusters` for other arguments.

`find.clusters` is a generic function with methods for `data.frame`, objects with the class `genind` (usual genetic markers) and `genlight` (genome-wide SNP data). Here, we illustrate its use using a toy dataset simulated in [1], `dapcIllus`:

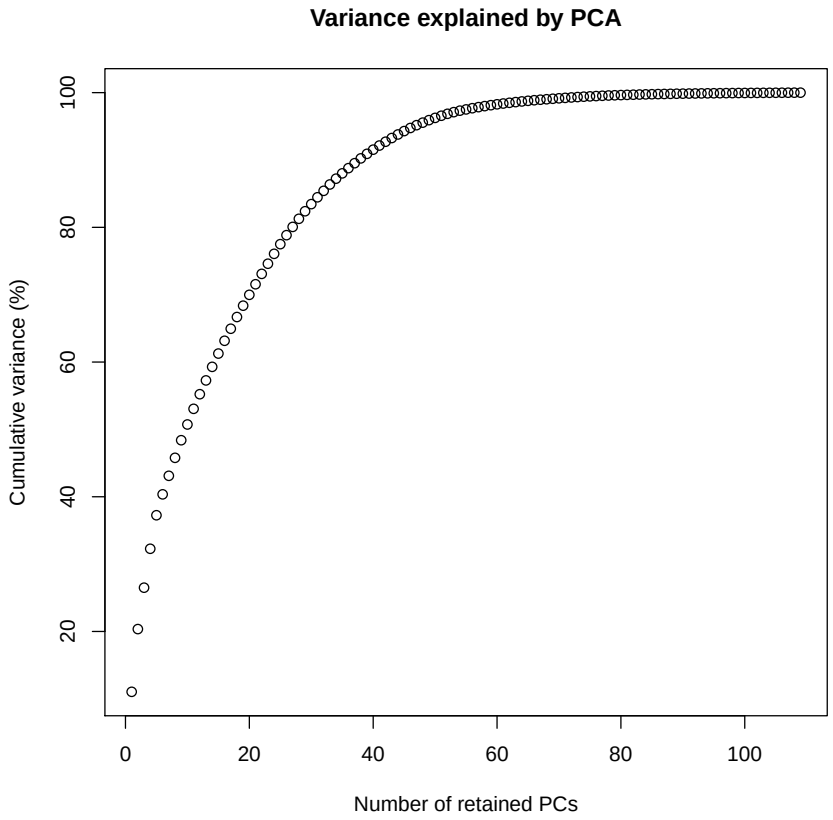
```
library(adegenet)
data(dapcIllus)
class(dapcIllus)

## [1] "list"

names(dapcIllus)

## [1] "a" "b" "c" "d"
```

`dapcIllus` is a list containing four datasets; we shall only use the first one:

```
x <- dapcIllus$a
x

## /// GENIND OBJECT /////////
##
##  // 600 individuals; 30 loci; 140 alleles; size: 405.7 Kb
##
##  // Basic content
##    @tab:  600 x 140 matrix of allele counts
##    @loc.n.all: number of alleles per locus (range: 2-8)
##    @loc.fac: locus factor for the 140 columns of @tab
##    @all.names: list of allele names for each locus
##    @ploidy: ploidy of each individual  (range: 2-2)
##    @type:  codom
##    @call: read.fstat(file = file, missing = missing, quiet = quiet)
##
##  // Optional content
##    @pop: population of each individual (group size range: 100-100)
```

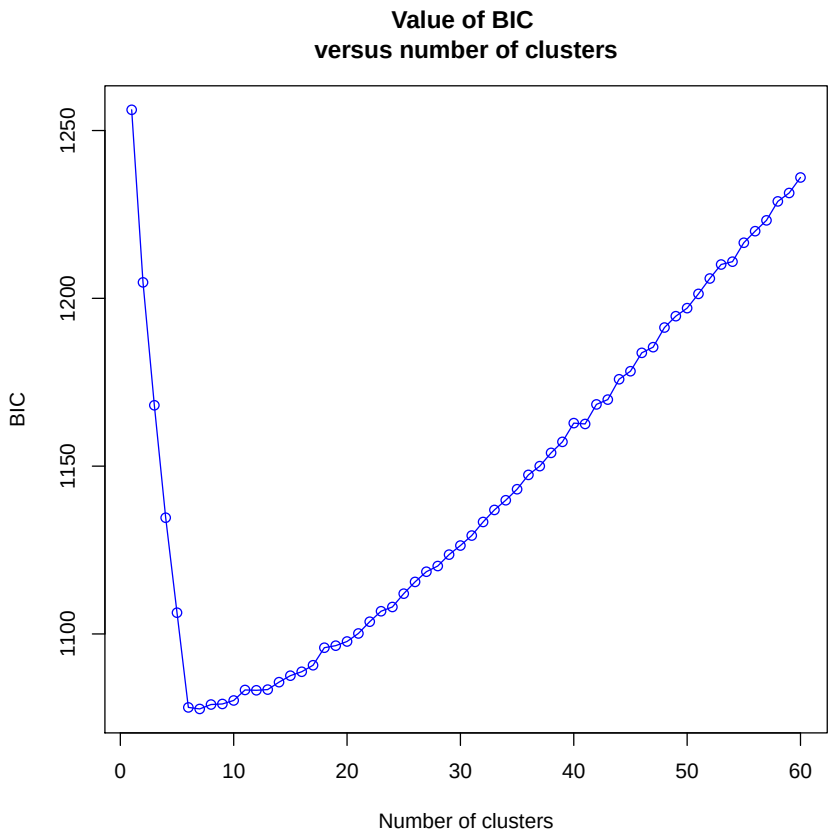
`x` is a dataset of 600 individuals simulated under an island model (6 islands) for 30 microsatellite markers. We use `find.clusters` to identify clusters, although true clusters are, in this case, known (and accessible using `pop(x)`). We specify that we want to evaluate up to $k = 40$ groups (`max.n.clust=40`):

```
grp <- find.clusters(x, max.n.clust=40)
```



The function displays a graph of cumulated variance explained by the eigenvalues of the PCA. Apart from computational time, there is no reason for keeping a small number of components; here, we keep all the information, specifying to retain 200 PCs (there are actually less PCs —around 110—, so all of them are kept).

Then, the function displays a graph of BIC values for increasing values of k:



This graph shows a clear decrease of BIC until $k = 6$ clusters, after which BIC increases. In this case, the elbow in the curve also matches the smallest BIC, and clearly indicates 6 clusters should be retained. In practice, the choice is often trickier to make for empirical dataset.

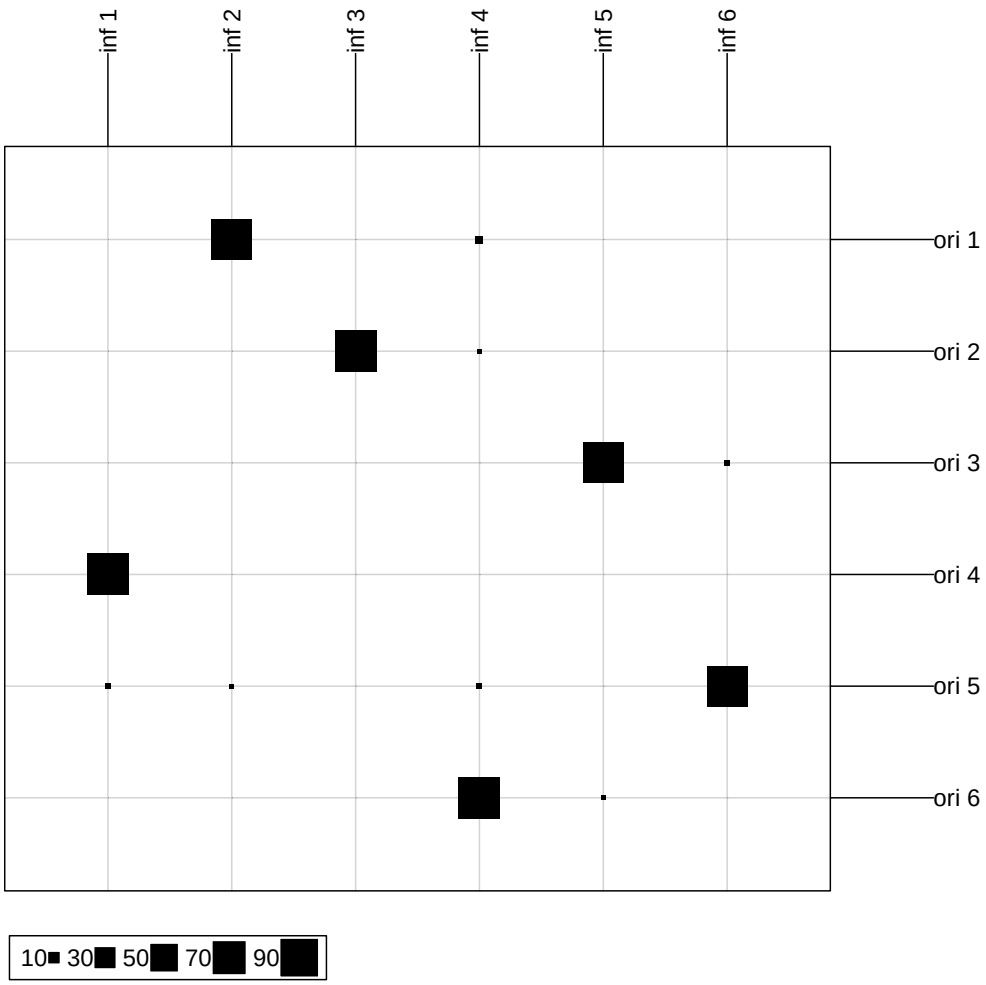
The output of `find.clusters` is a list:

```
names(grp)

## [1] "Kstat" "stat"  "grp"   "size"

head(grp$Kstat, 8)

## NULL

grp$stat

## NULL

head(grp$grp, 10)

##  1  2  3  4  5  6  7  8  9 10
##  2  2  2  4  2  2  2  2  2  2
## Levels: 1 2 3 4 5 6
```

```
grp$size

## [1] 102  98  99 105  99  97
```

The components are respectively the chosen summary statistics (here, BIC) for different values of k (slot `Kstat`), the selected number of clusters and the associated BIC (slot `stat`), the group memberships (slot `grp`) and the group sizes (slot `size`). Here, since we know the actual groups, we can check how well they have been retrieved by the procedure. Actual groups are accessed using `pop`:

```
table(pop(x), grp$grp)

##
##        1   2   3   4   5   6
##   P1   0  97   0   3   0   0
##   P2   0   0  99   1   0   0
##   P3   0   0   0   0  98   2
##   P4 100   0   0   0   0   0
##   P5   2   1   0   2   0  95
##   P6   0   0   0  99   1   0

table.value(table(pop(x), grp$grp), col.lab=paste("inf", 1:6),
            row.lab=paste("ori", 1:6))
```

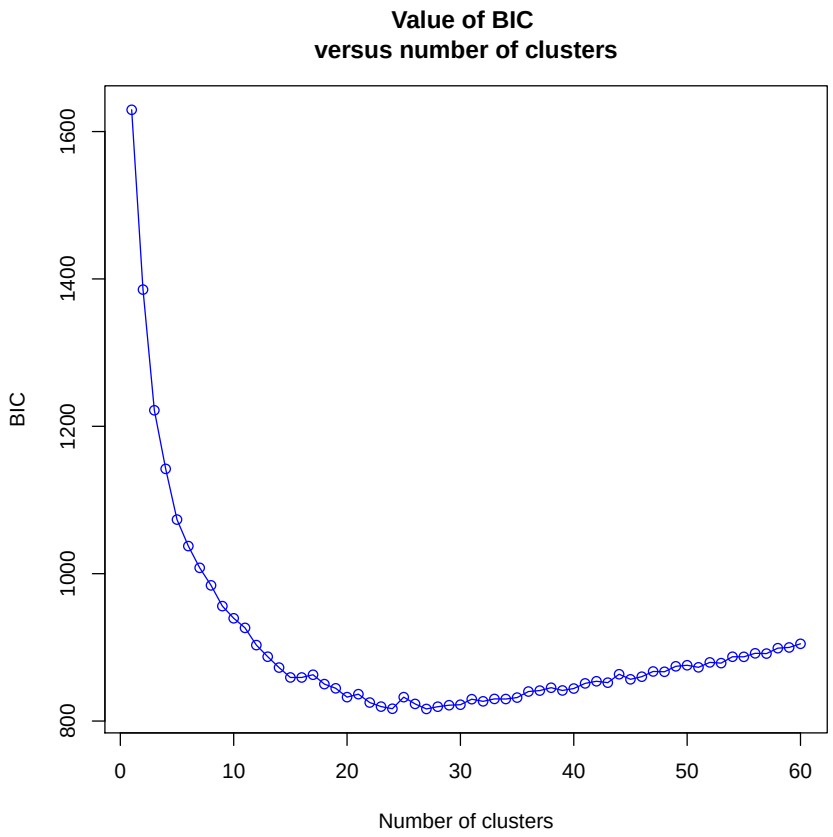


Rows correspond to actual groups ("ori"), while columns correspond to inferred groups ("inf"). Here, we can see that original groups have nearly been perfectly identified by the method.

2.3 How many clusters are there really in the data?

Although the most frequently asked when trying to find clusters in genetic data, this question is equally often meaningless. Clustering algorithms help making a caricature of a complex reality, which is most of the time far from following known population genetics models. Therefore, we are rarely looking for actual panmictic populations from which the individuals have been drawn. Genetic clusters can be biologically meaningful structures and reflect interesting biological processes, but they are still models.

A slightly different but probably more meaningful question would be: "How many clusters are useful to describe the data?". A fundamental point in this question is that clusters are merely tools used to summarise and understand the data. There is no longer a "true k", but some values of k are better, more efficient summaries of the data than others. For instance, in the following case:



, the concept of "true k" is fairly hypothetical. This does not mean that clutering algorithms should necessarily be discarded, but surely the reality is more complex than a few clear-cut, isolated populations. What the BIC decrease says is that 10-20 clusters would provide useful summaries of the data. The actual number retained is merely a question of personnal taste.

3 Describing clusters using `dapc`

3.1 Rationale

DAPC aims to provide an efficient description of genetic clusters using a few synthetic variables. These are constructed as linear combinations of the original variables (alleles) which have the largest between-group variance and the smallest within-group variance. Coefficients of the alleles used in the linear combination are called *loadings*, while the synthetic variables are themselves referred to as *discriminant functions*.

Moreover, being based on the Discriminant Analysis, DAPC also provides membership probabilities of each individual for the different groups based on the retained discriminant functions. While these are different from the admixture coefficients of software like STRUCTURE, they can still be interpreted as proximities of individuals to the different clusters. Membership probabilities also provide indications of how clear-cut genetic clusters are. Loose clusters will result in fairly flat distributions of membership probabilities of

individuals across clusters, pointing to possible admixture.

Lastly, using the allele loadings, it is possible to represent new individuals (which have not participated to the analysis) onto the factorial planes, and derive membership probabilities as welll. Such individuals are referred to as *supplementary individuals*.

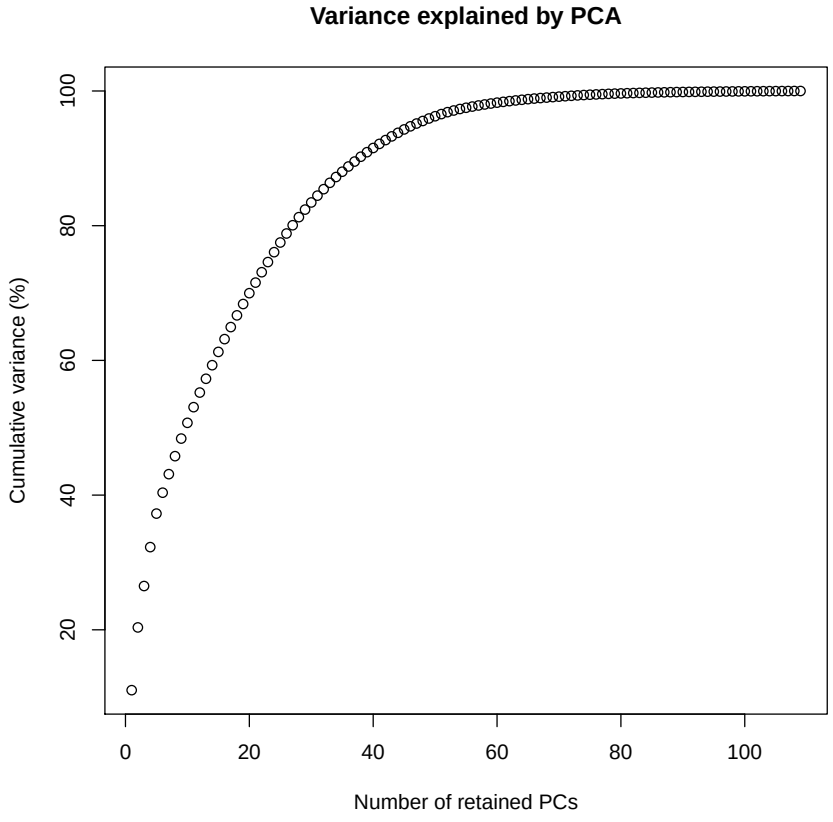
3.2 In practice

DAPC is implemented by the function `dapc`, which first transforms the data using PCA, and then performs a Discriminant Analysis on the retained principal components. Like `find.clusters`, `dapc` is a generic function with methods for `data.frame`, and objects with the class `genind` (usual genetic markers) and `genlight` (genome wide SNP data).

We run the analysis on the previous toy dataset, using the inferred groups stored in `grp$grp`:

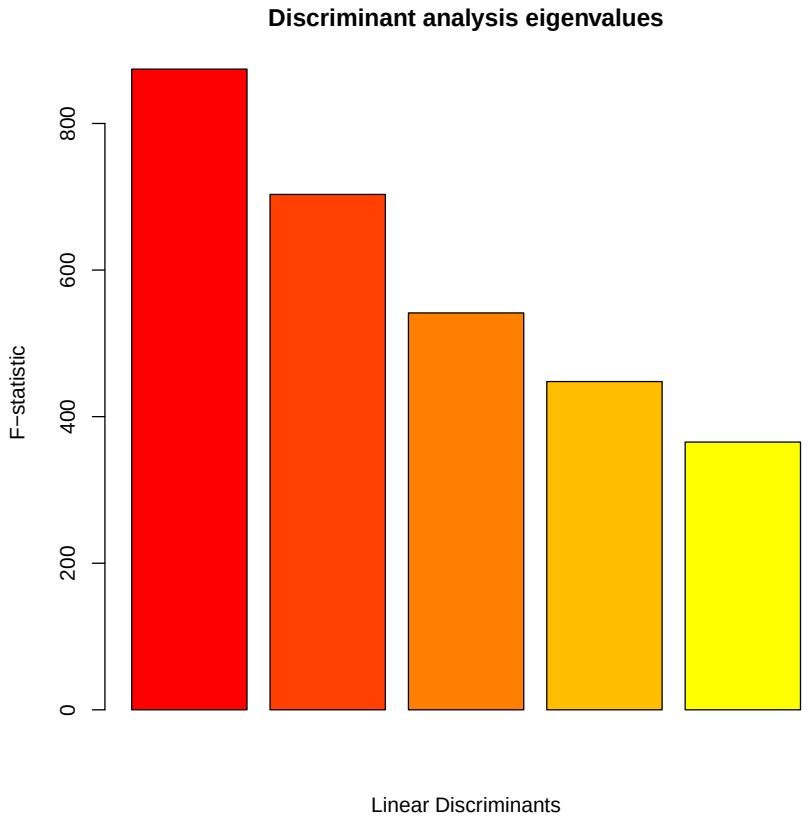
```
dapc1 <- dapc(x, grp$grp)
```

The method displays the same graph of cumulated variance as in `find.cluster`. However, unlike k-means, DAPC can benefit from not using too many PCs. Indeed, retaining too many components with respect to the number of individuals can lead to over-fitting and unstability in the membership probabilities returned by the method (see section below about the stability of membership probabilities).



The bottomline is therefore retaining a few PCs without sacrificing too much information. Here, we can see that little information is gained by adding PCs after the first 40. We therefore retain 40 PCs.

Then, the method displays a barplot of eigenvalues for the discriminant analysis, asking for a number of discriminant functions to retain (unless argument `n.da` is provided).



For small number of clusters, all eigenvalues can be retained since all discriminant functions can be examined without difficulty. Whenever more (say, tens of) clusters are analysed, it is likely that the first few dimensions will carry more information than the others, and only those can then be retained and interpreted.

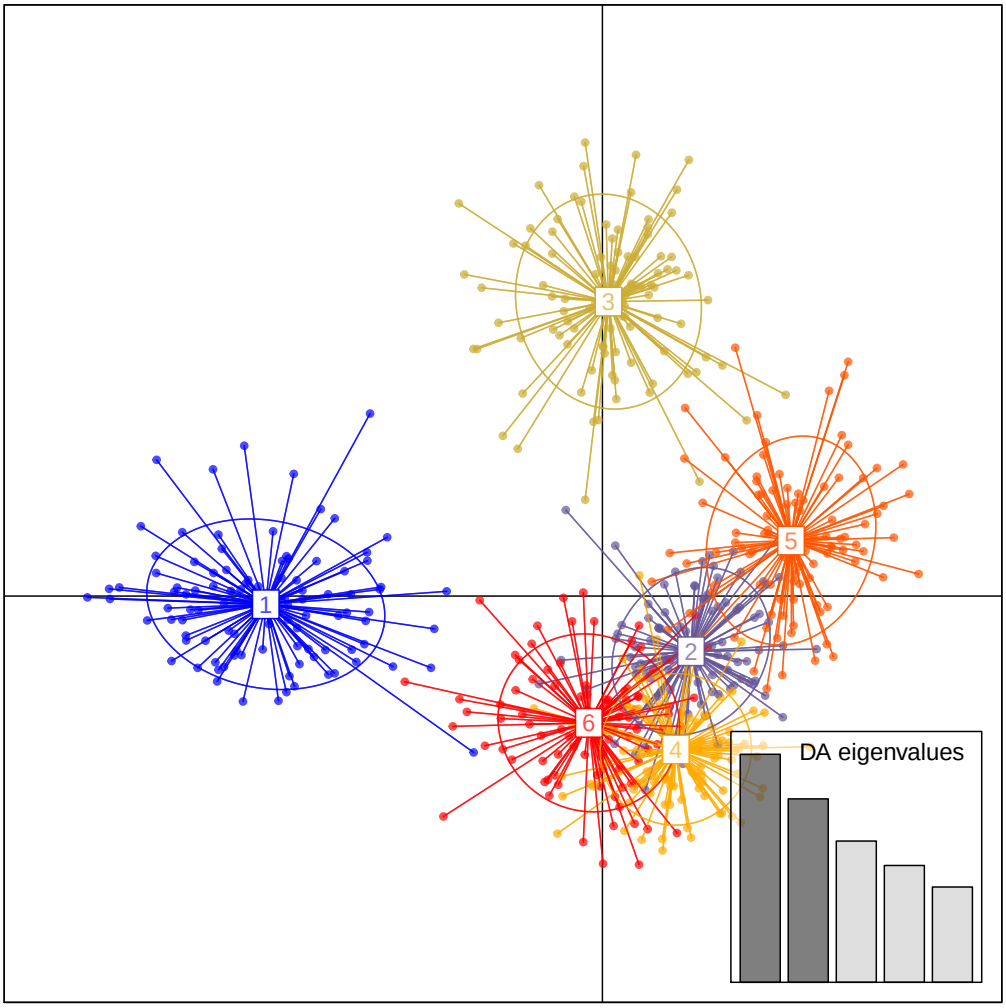
The object `dapc1` contains a lot of information:

```
dapc1

##  ################################################
##  # Discriminant Analysis of Principal Components #
##  ################################################
## class: dapc
## $call: dapc.genind(x = x, pop = grp$grp, n.pca = 40, n.da = 100)
##
## $n.pca: 40 first PCs of PCA used
```

```
## $n.da: 5 discriminant functions saved
## $var (proportion of conserved variance): 0.915
##
## $eig (eigenvalues): 874.1 703.2 541.5 447.9 365.3  vector    length content
## 1 $eig      5      eigenvalues
## 2 $grp      600    prior group assignment
## 3 $prior    6      prior group probabilities
## 4 $assign   600    posterior group assignment
## 5 $pca.cent 140    centring vector of PCA
## 6 $pca.norm 140    scaling vector of PCA
## 7 $pca.eig  109    eigenvalues of PCA
##
##   data.frame    nrow ncol
## 1 $tab          600  40
## 2 $means        6    40
## 3 $loadings     40   5
## 4 $ind.coord    600  5
## 5 $grp.coord    6    5
## 6 $posterior    600  6
## 7 $pca.loadings 140  40
## 8 $var.contr    140  5
##   content
## 1 retained PCs of PCA
## 2 group means
## 3 loadings of variables
## 4 coordinates of individuals (principal components)
## 5 coordinates of groups
## 6 posterior membership probabilities
## 7 PCA loadings of original variables
## 8 contribution of original variables
```

For details about this content, please read the documentation (`?dapc`). Essentially, the slots `ind.coord` and `grp.coord` contain the coordinates of the individuals and of the groups used in scatterplots. Contributions of the alleles to each discriminant function are stored in the slot `var.contr`. Eigenvalues, corresponding to the ratio of the variance between groups over the variance within groups for each discriminant function, are stored in `eig`. Basic scatterplots can be obtained using the function `scatterplot`:

```
scatter(dapc1)
```



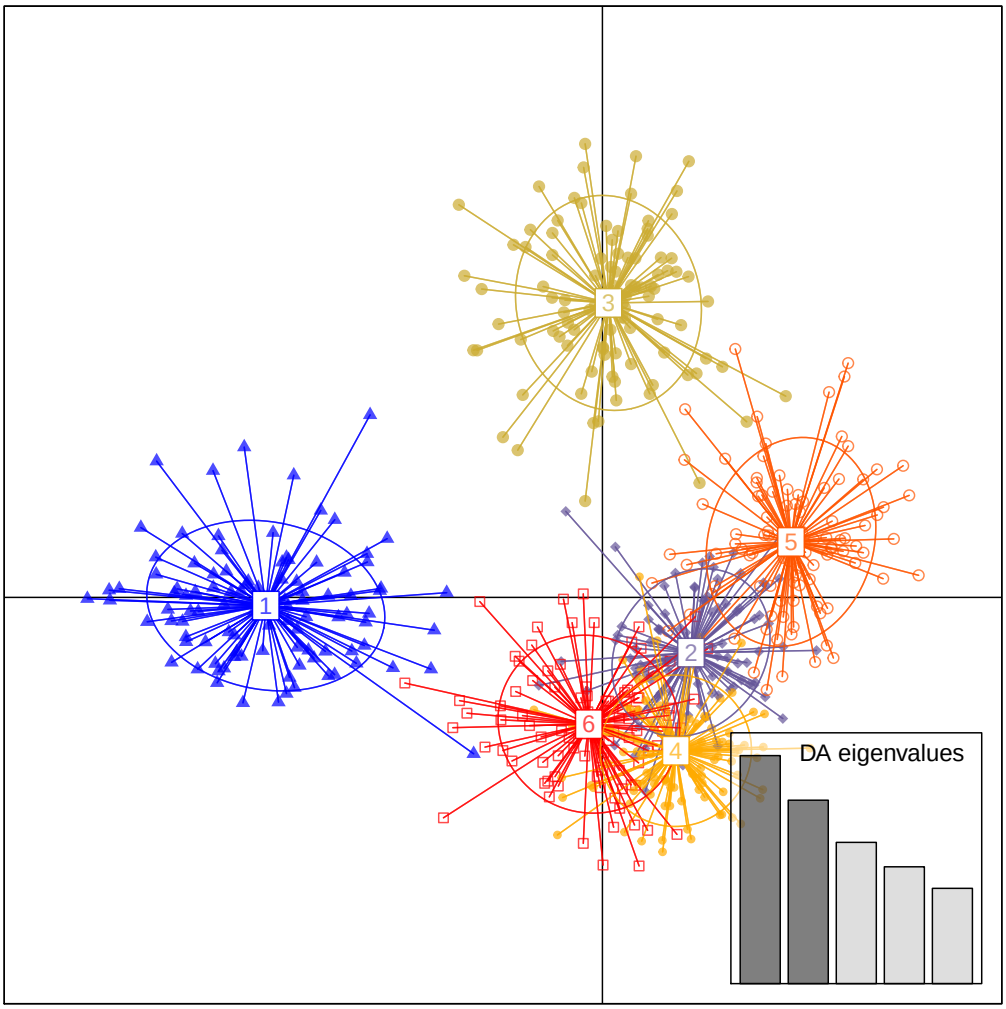
The obtained graph represents the individuals as dots and the groups as inertia ellipses. Eigenvalues of the analysis are displayed in inset. These graphs are fairly easy to customize, as shown below.

3.3 Customizing DAPC scatterplots

DAPC scatterplots are the main result of DAPC. It is therefore essential to ensure that information is displayed efficiently, and if possible to produce pretty figures. Possibility are almost unlimited, and here we just illustrate a few possibilities offered by `scatter`. Note that `scatter` is a generic function, with a dedicated method for objects produced by `dapc`. Documentation of this function can be accessed by typing `?scatter.dapc`.

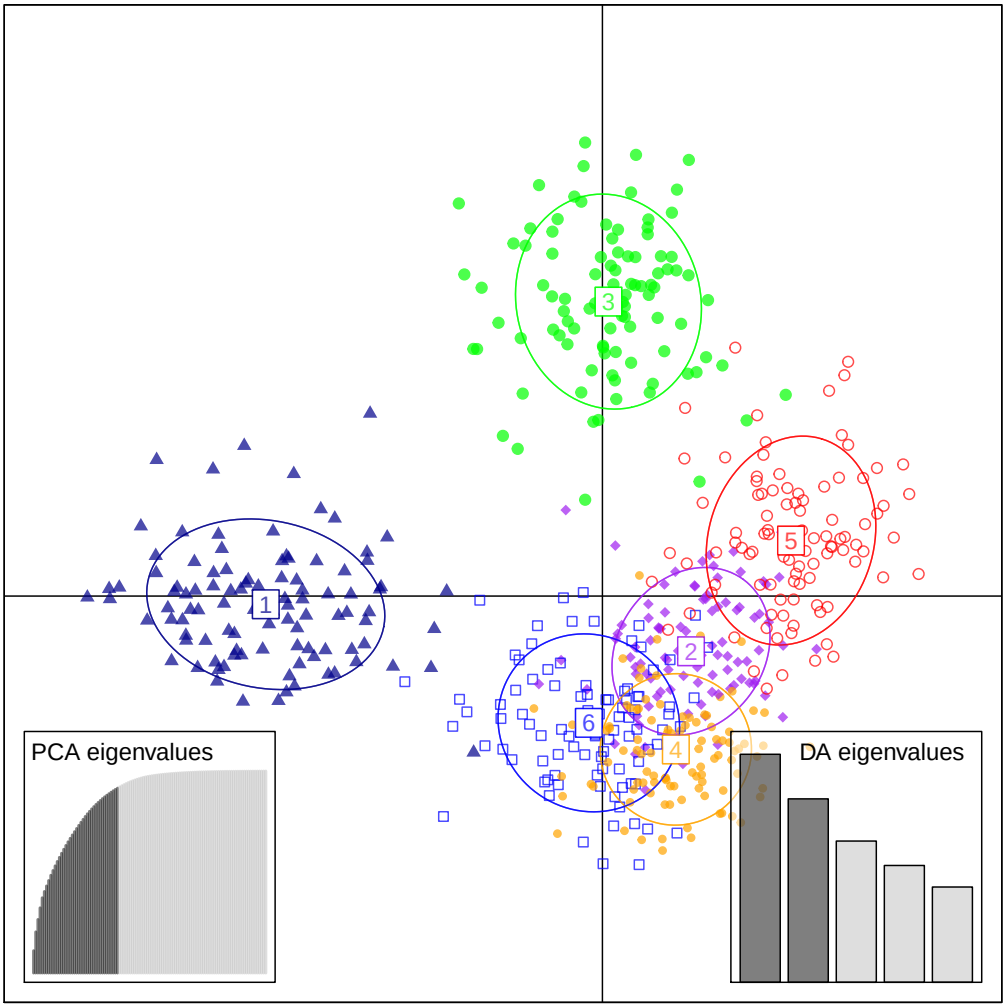
We illustrate some graphical possibilities trying to improve the display of the analysis presented in the previous section. While the default background (grey) allows to visualize rainbow colors (the default palette for the groups) more easily, it is not so pretty and is probably better removed for publication purpose. We also move the inset to a more appropriate place where it does not cover individuals, and use different symbols for the groups.

```
scatter(dapc1, posi.da="bottomright", bg="white", pch=17:22)
```



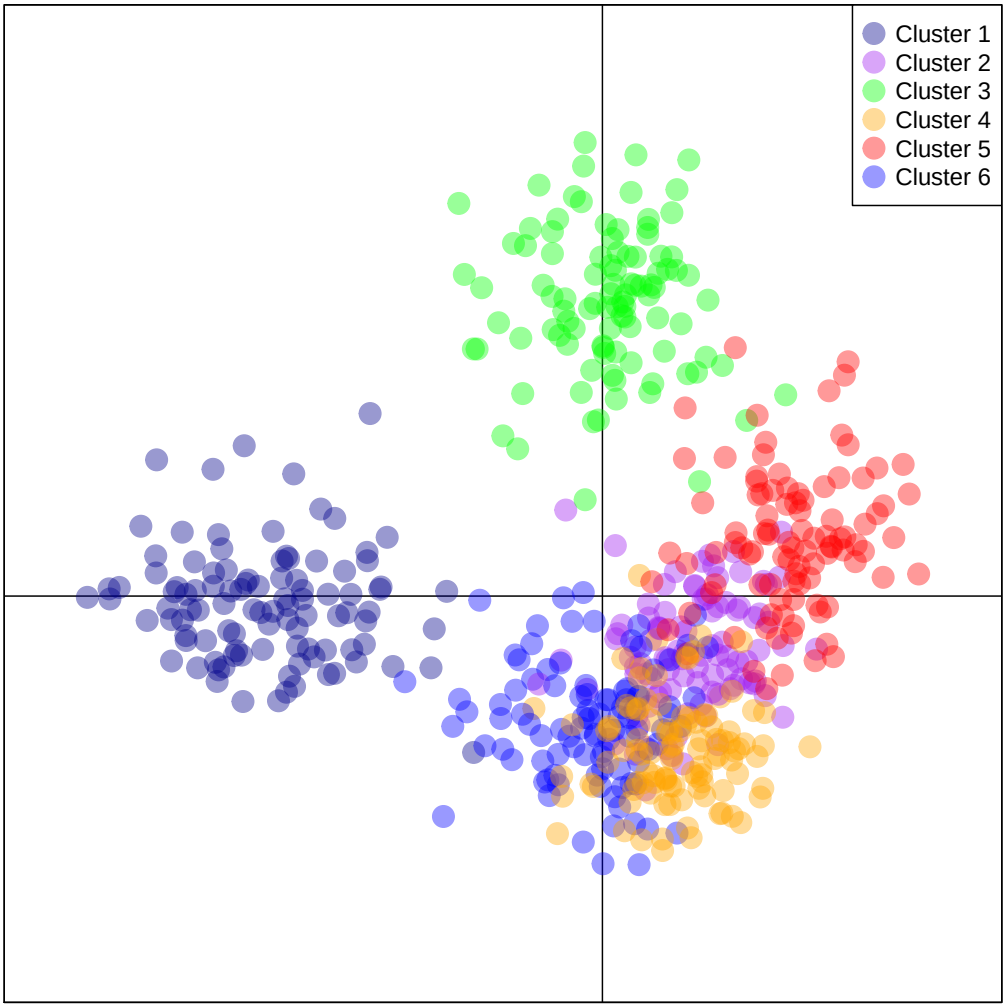
This is still not entirely satisfying: we need to define other colors more visible over a white background, and we can remove the segments linking the points to their ellipses:

```
myCol <- c("darkblue","purple","green","orange","red","blue")
scatter(dapc1, posi.da="bottomright",  bg="white",
        pch=17:22, cstar=0, col=myCol, scree.pca=TRUE,
        posi.pca="bottomleft")
```



Another possibility is remove the labels within the ellipses and add a legend to the plot. We also use the same symbol for all individuals, but use bigger dots and transparent colours to have a better feel for the density of individuals on the factorial plane.

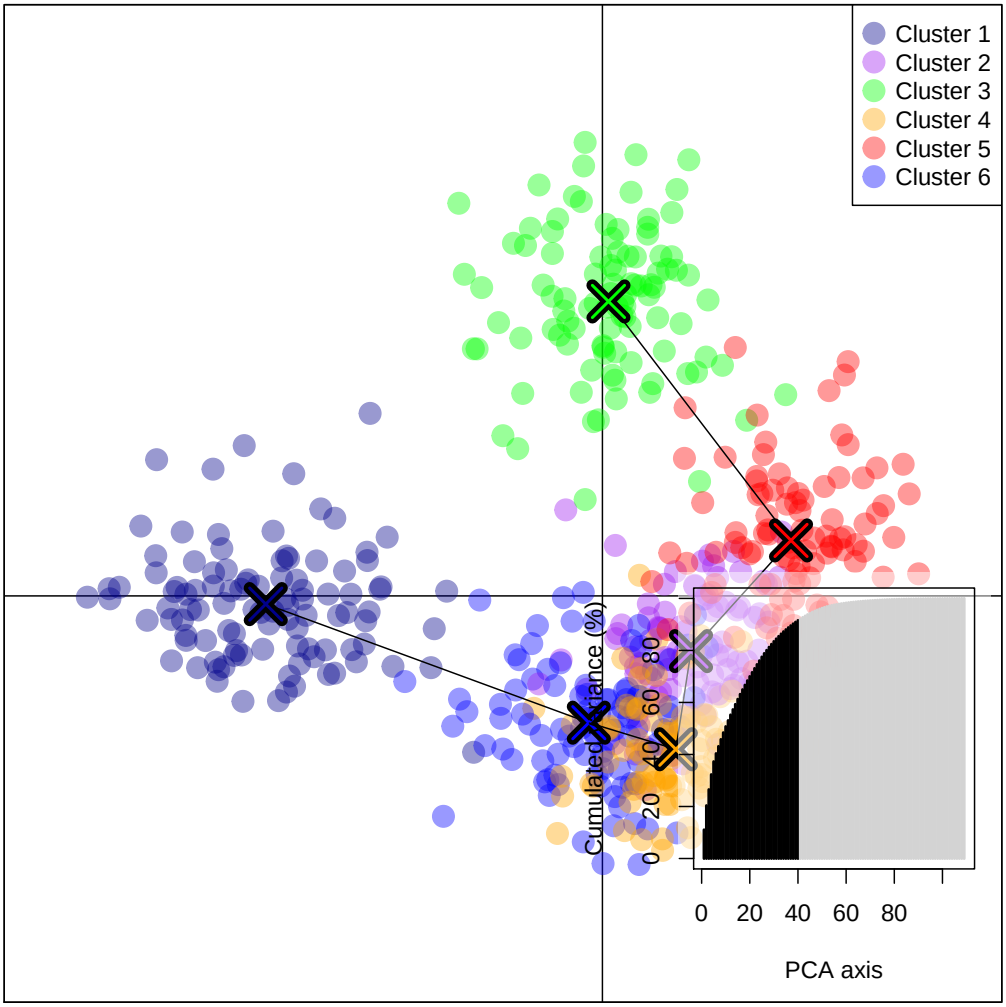
```
scatter(dapc1, scree.da=FALSE, bg="white", pch=20,  cell=0, cstar=0, col=myCol, solid=.4
        cex=3,clab=0, leg=TRUE, txt.leg=paste("Cluster",1:6))
```



We can also add a minimum spanning tree based on the (squared) distances between populations within the entire space. This allows one to bear in mind the actual proximities between populations inside the entire space, which are not always well represented in susbsets of discriminant functions of lesser rank. We also indicate the centre of each group with crosses. Lastly, we remove the DAPC eigenvalues, not very useful in this case, and replace them manually by a graph of PCA eigenvalues retained in dimension-reduction step (retained eigenvalues in black, similar to using `scree.pca=TRUE`).

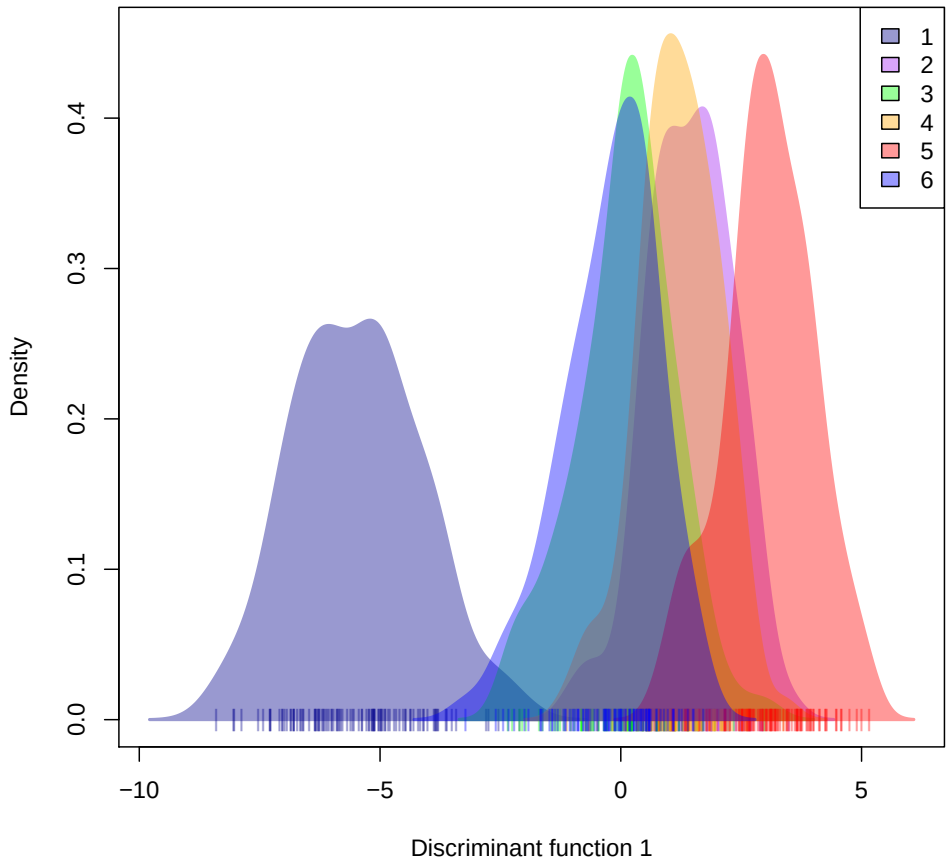
```
scatter(dapc1, ratio.pca=0.3, bg="white", pch=20,  cell=0,
        cstar=0, col=myCol, solid=.4, cex=3, clab=0,
        mstree=TRUE, scree.da=FALSE, posi.pca="bottomright",
        leg=TRUE, txt.leg=paste("Cluster",1:6))

par(xpd=TRUE)
points(dapc1$grp.coord[,1], dapc1$grp.coord[,2], pch=4,
       cex=3, lwd=8, col="black")
points(dapc1$grp.coord[,1], dapc1$grp.coord[,2], pch=4,
       cex=3, lwd=2, col=myCol)
```

```
myInset <- function(){
    temp <- dapc1$pca.eig
    temp <- 100* cumsum(temp)/sum(temp)
    plot(temp, col=rep(c("black","lightgrey"),
                c(dapc1$n.pca,1000)), ylim=c(0,100),
         xlab="PCA axis", ylab="Cumulated variance (%)",
         cex=1, pch=20, type="h", lwd=2)
}

add.scatter(myInset(), posi="bottomright",
            inset=c(-0.03,-0.01), ratio=.28,
            bg=transp("white"))
```



Lastly, note that `scatter` can also represent a single discriminant function, which is especially useful when only one of these has been retained (e.g. in the case $k = 2$). This is achieved by plotting the densities of individuals on a given discriminant function with different colors for different groups:

```
scatter(dapc1,1,1, col=myCol, bg="white",
        scree.da=FALSE, legend=TRUE, solid=.4)
```



3.4 Interpreting variable contributions

In DAPC, the variable actually analyzed are principal components of a PCA. Loadings of these variables are generally uninformative, since PCs themselves do not all have straightforward interpretations. However, we can also compute contributions of the alleles, which can turn out to be very informative. In general, there are many alleles and their contribution is best plotted for a single discriminant function at a time.

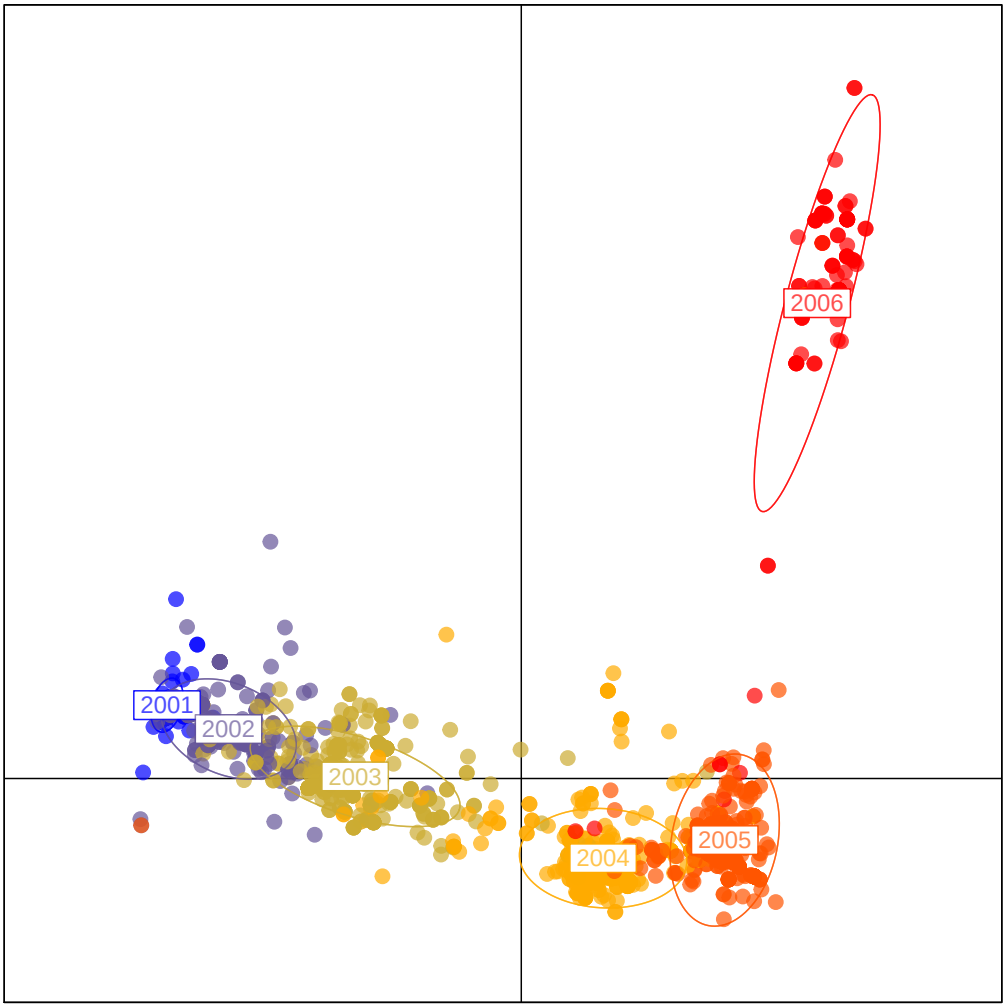
Variable contributions are stored in the `var.contr` slot of a `dapc` object. They can be plotted using `loadingplot`. We illustrate this using the seasonal influenza dataset `H3N2`, which contains 1903 isolates genotyped for 125 SNPs located in the hemagglutinin segment (see `?H3N2`):

```
data(H3N2)
H3N2
```

```
## /// GENIND OBJECT /////////
##
##  // 1,903 individuals; 125 loci; 334 alleles; size: 3.4 Mb
##
##  // Basic content
##    @tab:  1903 x 334 matrix of allele counts
##    @loc.n.all: number of alleles per locus (range: 2-4)
##    @loc.fac: locus factor for the 334 columns of @tab
##    @all.names: list of allele names for each locus
##    @ploidy: ploidy of each individual  (range: 1-1)
##    @type:  codom
##    @call: .local(x = x, i = i, j = j, drop = drop)
##
##  // Optional content
##    @other: a list containing: x  xy  epid

pop(H3N2) <- H3N2$other$epid
dapc.flu <- dapc(H3N2, n.pca=30,n.da=10)
```

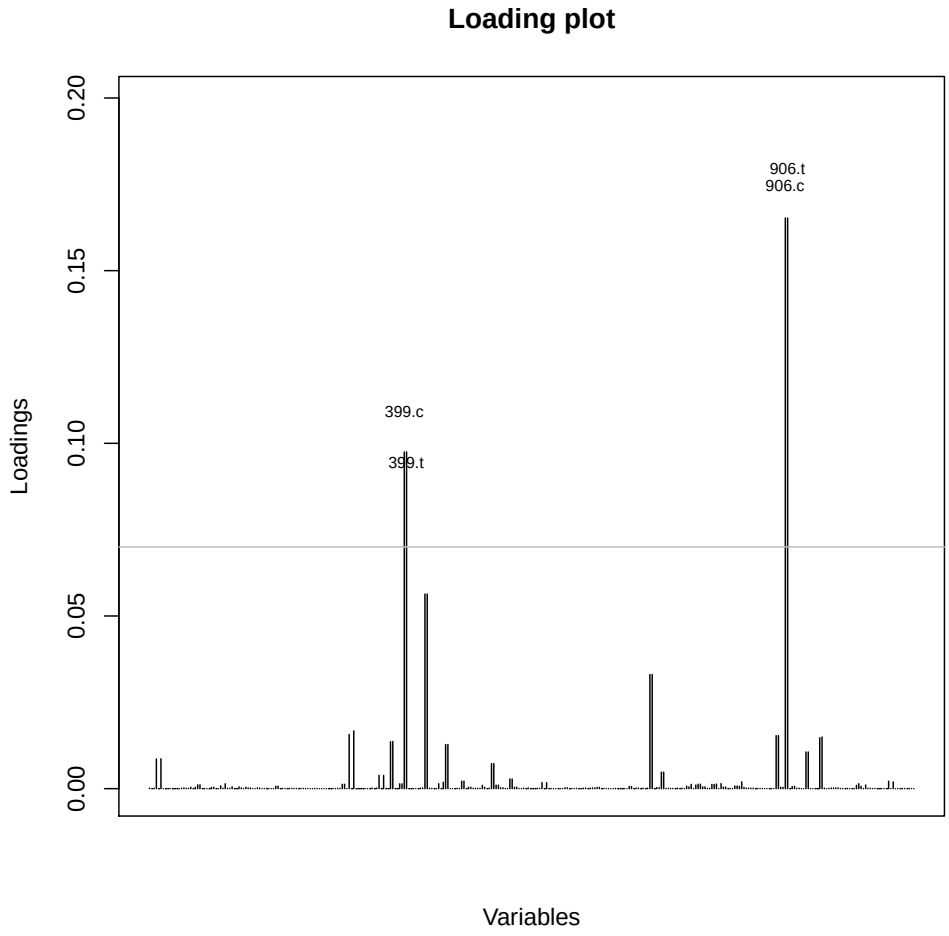
The first discriminant function shows the temporal evolution of the influenza virus, while the second one shows the originality of 2006 strains.

```
myPal <- colorRampPalette(c("blue","gold","red"))
scatter(dapc.flu, col=transp(myPal(6)), scree.da=FALSE,
        cell=1.5, cex=2, bg="white",cstar=0)
```



We can assess which alleles most highlight the originality of 2006 using `loadingplot`:

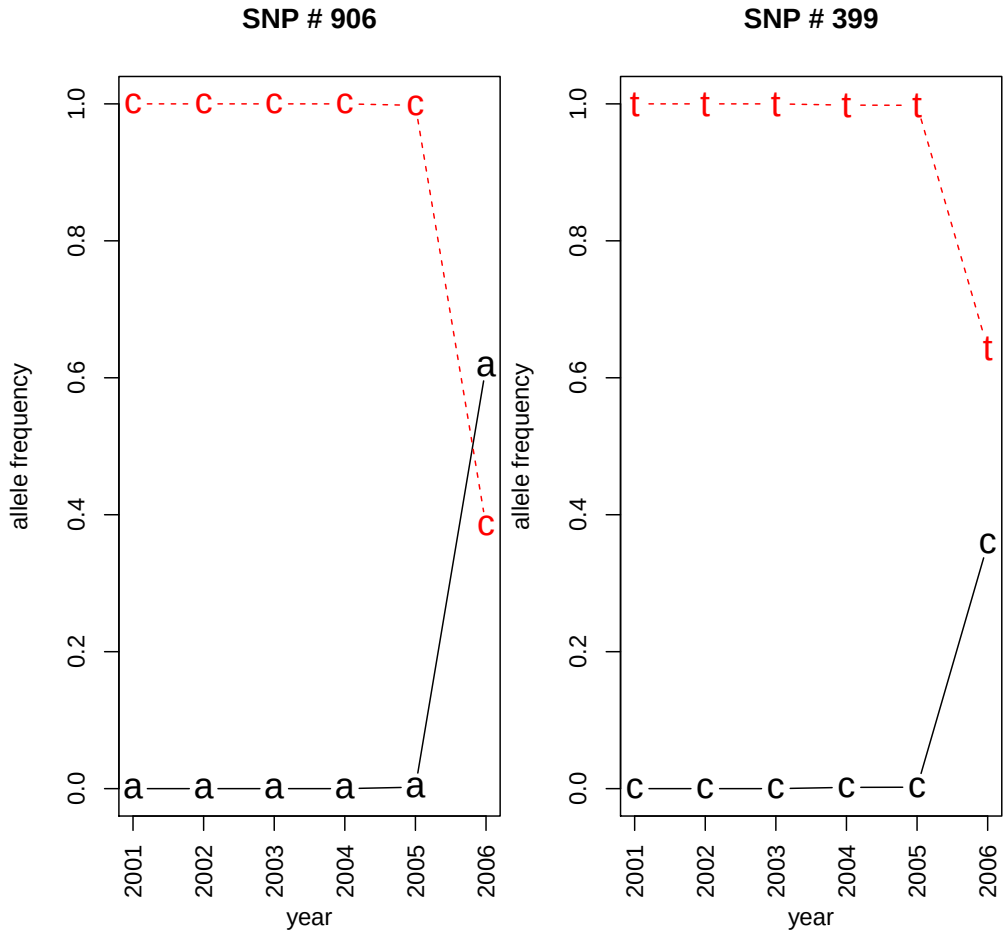
```
set.seed(4)
contrib <- loadingplot(dapc.flu$var.contr, axis=2,
                       thres=.07, lab.jitter=1)
```



`temp` is a list invisibly returned by `loadingplot` which contains the most contributing alleles (i.e., contributions above a given threshold – argument `threshold`). In this case, SNPs `906` and `399` reflect most the temporal evolution of the virus. We can look into their allele frequencies over 2002-2006:

```
freq399 <- tab(genind2genpop(H3N2[loc=c("399")]),freq=TRUE)

##
##  Converting data from a genind to a genpop object...
##
## ...done.

freq906 <- tab(genind2genpop(H3N2[loc=c("906")]),freq=TRUE)

##
##  Converting data from a genind to a genpop object...
##
## ...done.
```

```
par(mfrow=c(1,2), mar=c(5.1,4.1,4.1,.1),las=3)
matplot(freq906, pch=c("a","c"), type="b",
        xlab="year",ylab="allele frequency", xaxt="n",
        cex=1.5, main="SNP # 906")
axis(side=1, at=1:6, lab=2001:2006)
matplot(freq399, pch=c("c","t"), type="b", xlab="year",
        ylab="allele frequency", xaxt="n", cex=1.5,
        main="SNP # 399")
axis(side=1, at=1:6, lab=2001:2006)
```



In both cases, a new allele appeared in 2005 at a very low frequency, and reached high or even dominant frequencies a year later. Irrespective of the mecanism underlying these changes (drift or selection), this illustrates that in seasonal influenza, specific nucleotides can undergo drastic changes within only a couple of years.

3.5 Interpreting group memberships

Besides scatterplots of discriminant functions, group memberships of DAPC can be exploited. Note that caution should be taken when interpreting group memberships of a DAPC based

on too many PCs, as there are risks of overfitting the discriminant functions (see section below). But despite this possible bias, group memberships can be used as indicators of how clear-cut genetic clusters are. Note that this is most useful for groups defined by an external criteria, i.e. defined biologically, as opposed to identified by k-means. It is less useful for groups identified using `find.clusters`, since we expect k-means to provide optimal groups for DAPC, and therefore both classifications to be mostly consistent.

Membership probabilities are based on the retained discriminant functions. They are stored in `dapc` objects in the slot `posterior`:

```
class(dapc1$posterior)

## [1] "matrix"

dim(dapc1$posterior)

## [1] 600   6

round(head(dapc1$posterior),3)

##   1     2 3     4 5 6
## 1 0 1.000 0 0.000 0 0
## 2 0 1.000 0 0.000 0 0
## 3 0 1.000 0 0.000 0 0
## 4 0 0.016 0 0.984 0 0
## 5 0 1.000 0 0.000 0 0
## 6 0 1.000 0 0.000 0 0
```

Each row corresponds to an individual, each column to a group. This information can be summarized using `summary` on the `dapc` object:

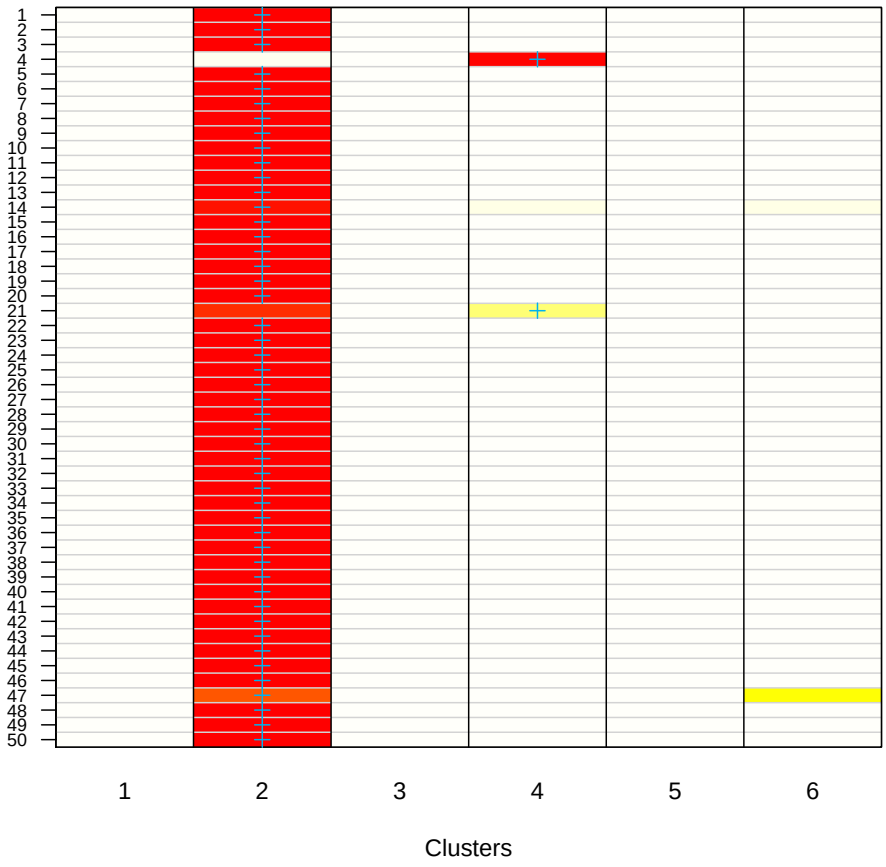
```
summary(dapc1)

## $n.dim
## [1] 5
##
## $n.pop
## [1] 6
##
## $assign.prop
## [1] 0.9966667
##
## $assign.per.pop
##         1         2         3         4         5         6
## 0.9901961 1.0000000 1.0000000 0.9904762 1.0000000 1.0000000
##
```

```
## $prior.grp.size
##
##   1   2   3   4   5   6
## 102  98  99 105  99  97
##
## $post.grp.size
##
##   1   2   3   4   5   6
## 101  99  99 105  99  97
```

The slot `assign.per.pop` indicates the proportions of successful reassignment (based on the discriminant functions) of individuals to their original clusters. Large values indicate clear-cut clusters, while low values suggest admixed groups.

This information can also be visualized using `assignplot` (see `?assignplot` for display options); here, we choose to represent only the first 50 individuals to make the figure readable:

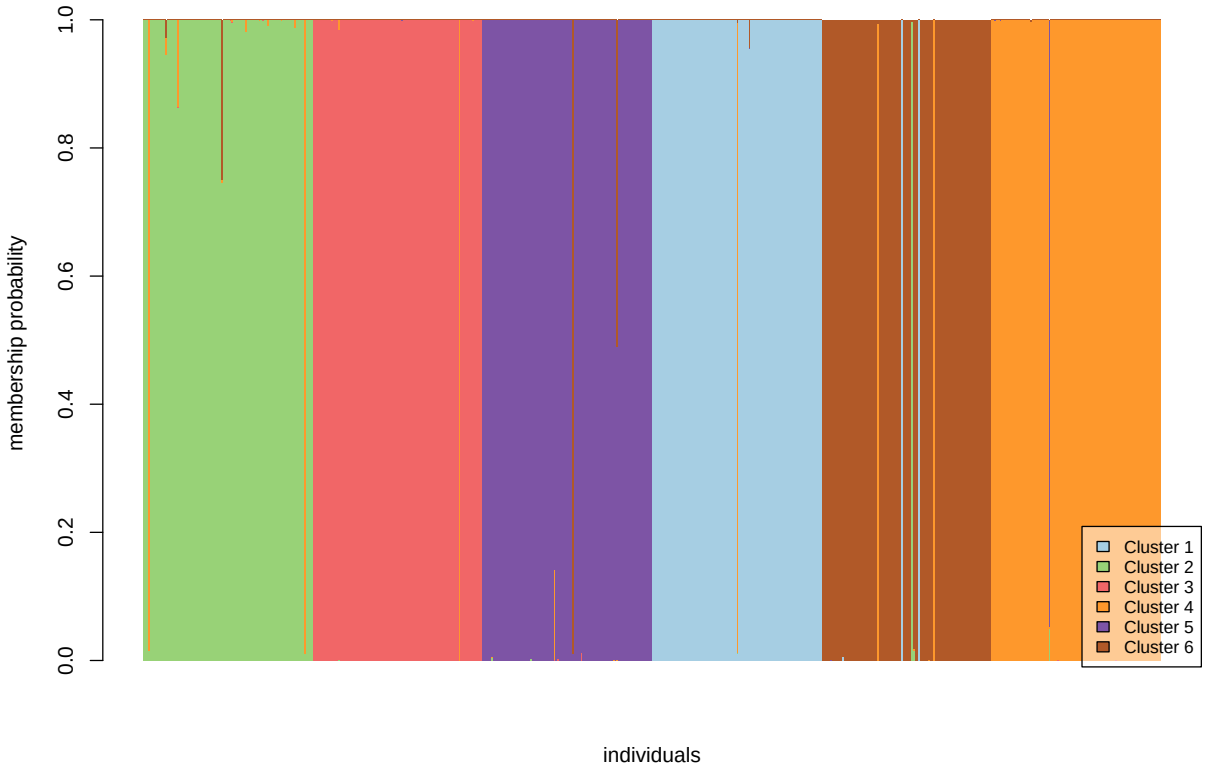
```
assignplot(dapc1, subset=1:50)
```



This figure is the simple graphical translation of the `posterior` table above. Heat colors represent membership probabilities (red=1, white=0); blue crosses represent the prior cluster provided to DAPC. Here in most individuals, DAPC classification is consistent with the original clusters (blue crosses are on red rectangles), except for one discrepancy in individual 21, classified in group 1 while DAPC would assign it to group 3. Such figure is particularly useful when prior biological groups are used, as one may infer admixed or misclassified individuals.

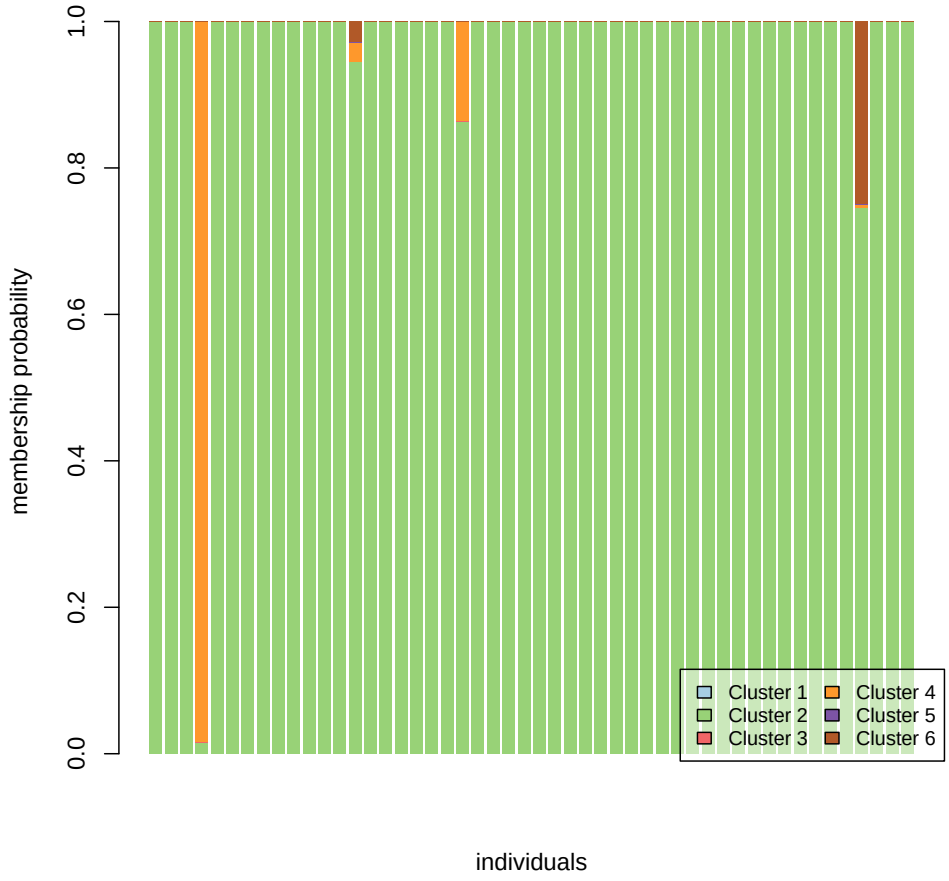
Note that this information can also be plotted in a STRUCTURE-like (!) way using `compoplot` (see `?compoplot` to customize the plot). We can plot information of all individuals to have a global picture of the clusters composition.

```
compoplot(dapc1, posi="bottomright",
          txt.leg=paste("Cluster", 1:6), lab="",
          ncol=1, xlab="individuals", col=funky(6))
```



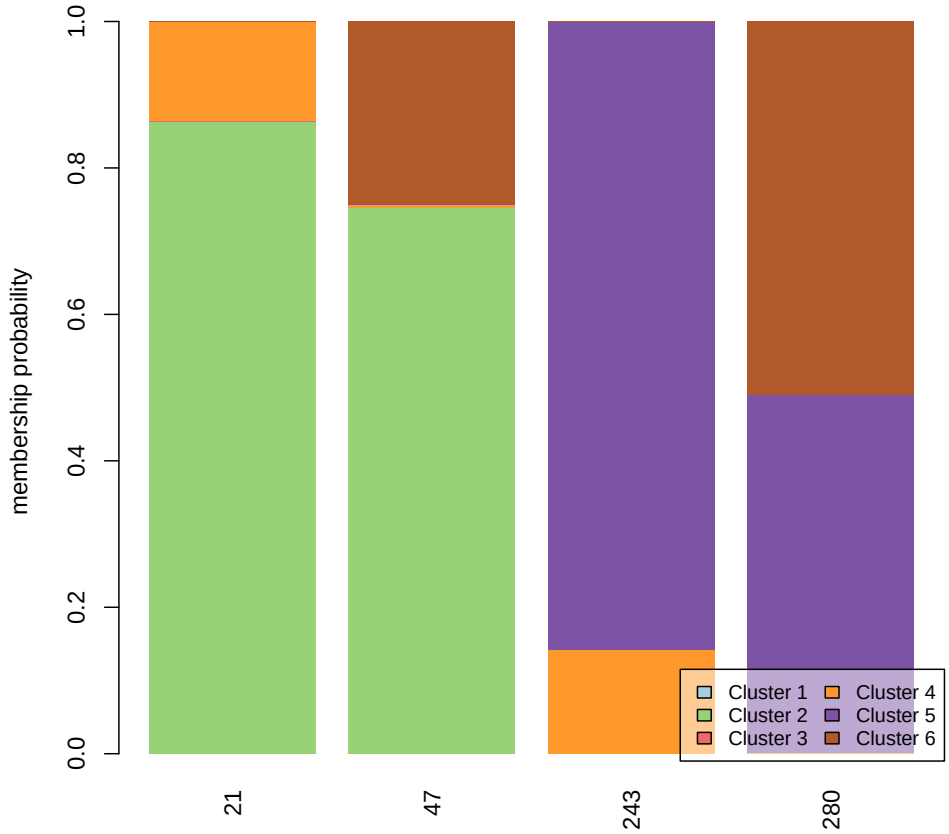
We can also have a closer look at a subset of individuals; for instance, for the first 50 individuals:

```
compoplot(dapc1, subset=1:50, posi="bottomright",
          txt.leg=paste("Cluster", 1:6), lab="",
          ncol=2, xlab="individuals", col=funky(6))
```



Obviously, we can use the power of R to lead our investigation further. For instance, which are the most 'admixed' individuals? Let us consider as admixed individuals having no more than 90% of probability of membership in a single cluster:

```
temp <- which(apply(dapc1$posterior,1, function(e) all(e<0.9)))
temp

##  21  47 243 280
##  21  47 243 280

compoplot(dapc1, subset=temp, posi="bottomright",
          txt.leg=paste("Cluster", 1:6),
          ncol=2, col=funky(6))
```



4 On the stability of group membership probabilities

4.1 When and why group memberships can be unreliable

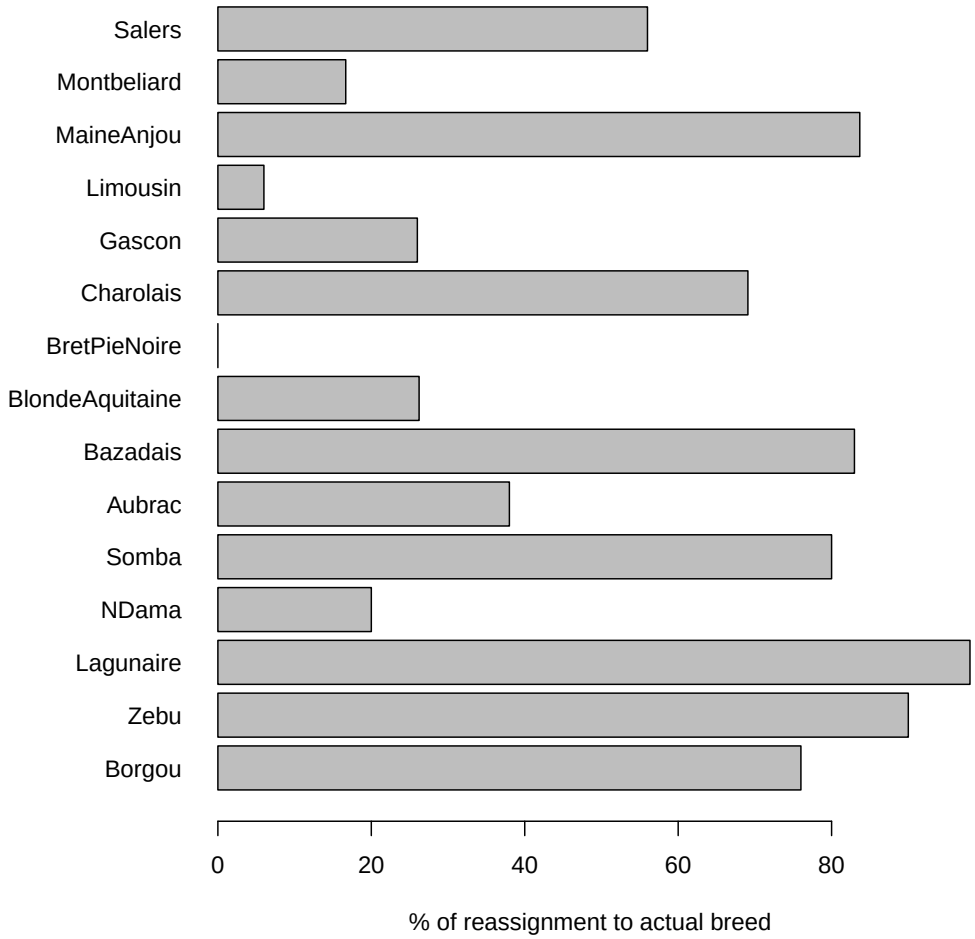
In DAPC, discriminant functions are linear combinations of variables (principal components of PCA) which optimize the separation of individuals into pre-defined groups. Based on the retained discriminant functions, it is possible to derive group membership probabilities, which can be interpreted in order to assess how clear-cut or admixed the clusters are. In attempting to summarise high-dimensional data in a small number of meaningful discriminant functions, DAPC must manage a trade-off. If too *few* PCs (with respect to the number of individuals) are retained, useful information will be excluded from the analysis, and the resultant model will not be informative enough to accurately discriminate between groups. By contrast, if too *many* PCs are retained, this will have a destabilising effect on the coefficients extimated, leading to problems of overfit. In such cases, the model is able to describe all of the data in such detail that it becomes flexible enough to discriminate almost perfectly between any possible clusters. As a result, membership probabilities can become drastically inflated for the best-fitting cluster, resulting in

apparent perfect discrimination. At the same time, however, the excessively complex model loses its ability to generalise to new or unseen data, as reflected in a loss of predictive capacity.

This point can be illustrated using the `microbov` dataset (704 cattles of 15 breeds typed for 30 microsatellite markers). We first examine the % of successful reassignment (i.e., quality of discrimination) for different numbers of retained PCs. First, retaining only 3 PCs during the dimension-reduction step, and all discriminant functions:

```
data(microbov)
microbov

## /// GENIND OBJECT /////////
##
##  // 704 individuals; 30 loci; 373 alleles; size: 1.2 Mb
##
##  // Basic content
##    @tab:  704 x 373 matrix of allele counts
##    @loc.n.all: number of alleles per locus (range: 5-22)
##    @loc.fac: locus factor for the 373 columns of @tab
##    @all.names: list of allele names for each locus
##    @ploidy: ploidy of each individual  (range: 2-2)
##    @type:  codom
##    @call: genind(tab = truenames(microbov)$tab, pop = truenames(microbov)$pop)
##
##  // Optional content
##    @pop: population of each individual (group size range: 30-61)
##    @other: a list containing: coun  breed  spe

temp <- summary(dapc(microbov, n.da=100, n.pca=3))$assign.per.pop*100
```

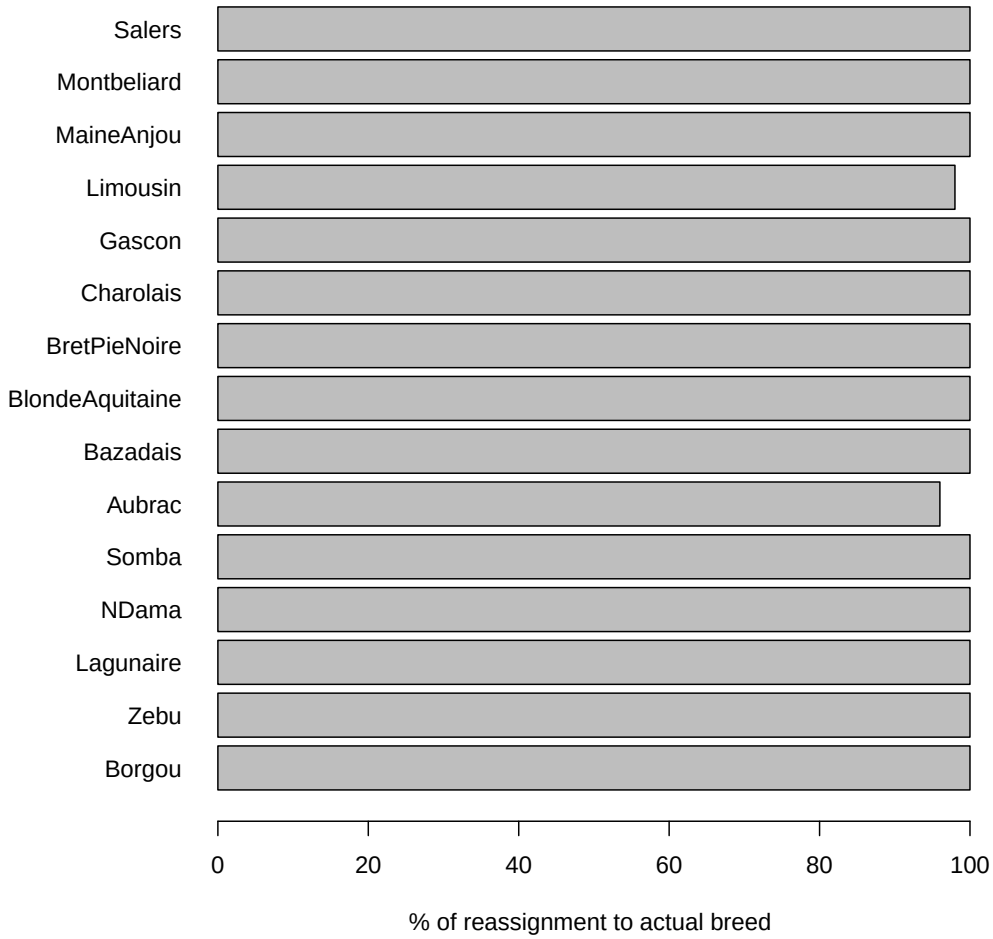
```
par(mar=c(4.5,7.5,1,1))
barplot(temp, xlab="% of reassignment to actual breed",
        horiz=TRUE, las=1)
```



We can see that some breeds are well discriminated (e.g. Zebu, Lagunaire, *geq* 90%) while others are entirely overlooked by the analysis (e.g. Bretone Pie Noire, Limousin, *leq* 10%). This is because too much genetic information is lost when retaining only 3 PCs. We repeat the analysis, this time keeping 300 PCs:

```
temp <- summary(dapc(microbov, n.da=100, n.pca=300))$assign.per.pop*100
```

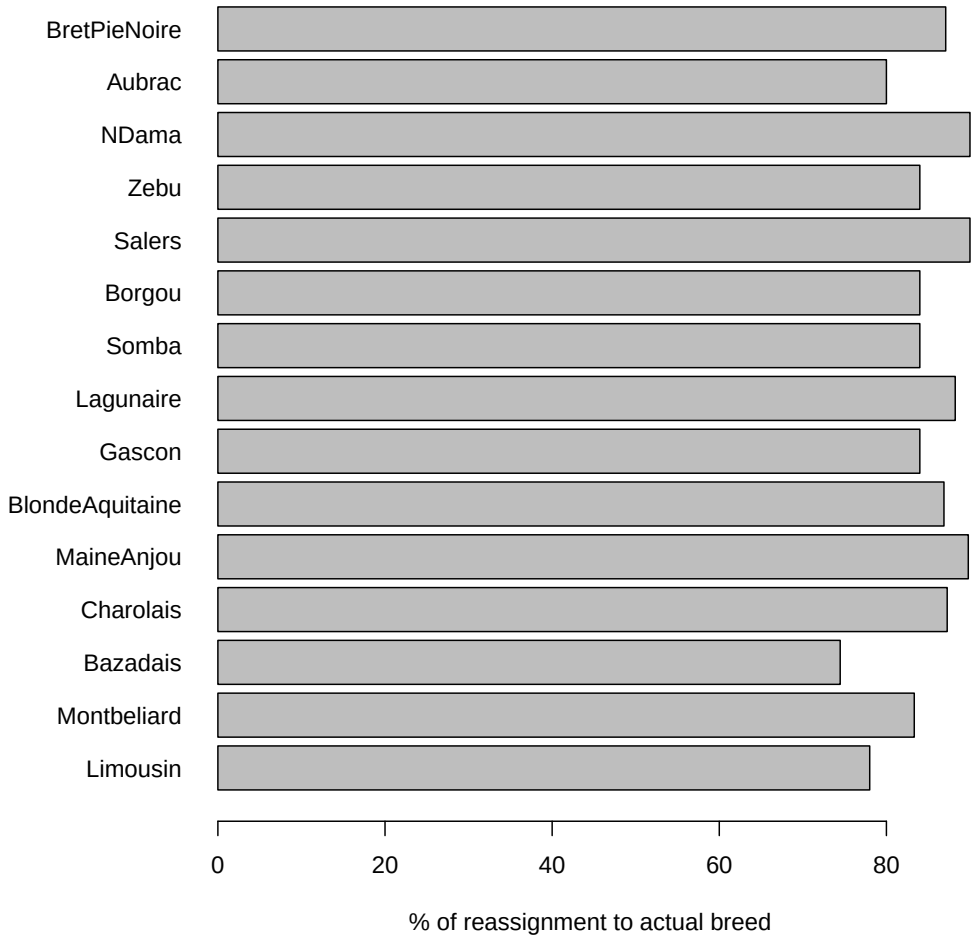
```
par(mar=c(4.5,7.5,1,1))
barplot(temp, xlab="% of reassignment to actual breed", horiz=TRUE, las=1)
```



We now obtain almost 100% discrimination for all groups. Is this result satisfying? Actually not. The number of PCs retained is so large that the resultant discriminant functions could in fact model any structure, and virtually any set of clusters would be well discriminated. This can be illustrated by running the analysis using randomised groups:

```
x <- microbov
pop(x) <- sample(pop(x))
temp <- summary(dapc(x, n.da=100, n.pca=300))$assign.per.pop*100
```

```
par(mar=c(4.5,7.5,1,1))
barplot(temp, xlab="% of reassignment to actual breed", horiz=TRUE, las=1)
```



Groups have been randomised, and yet we find that we still get very good discrimination. This clearly illustrates the trade-off we have described: DAPC requires enough PCs to secure a space with sufficient power of discrimination but must also avoid retaining too many dimensions that lead to over-fitting.

4.2 Using the *a*-score

The trade-off between power of discrimination and over-fitting can be measured by the *a*-score, which is simply the difference between the proportion of successful reassignment of the analysis (observed discrimination) and values obtained using random groups (random discrimination). It can be seen as the proportion of successful reassignment corrected for the number of retained PCs. It is implemented by `a.score`, which relies on repeating the DAPC analysis using randomized groups, and computing *a*-scores for each group, as well as the average *a*-score:

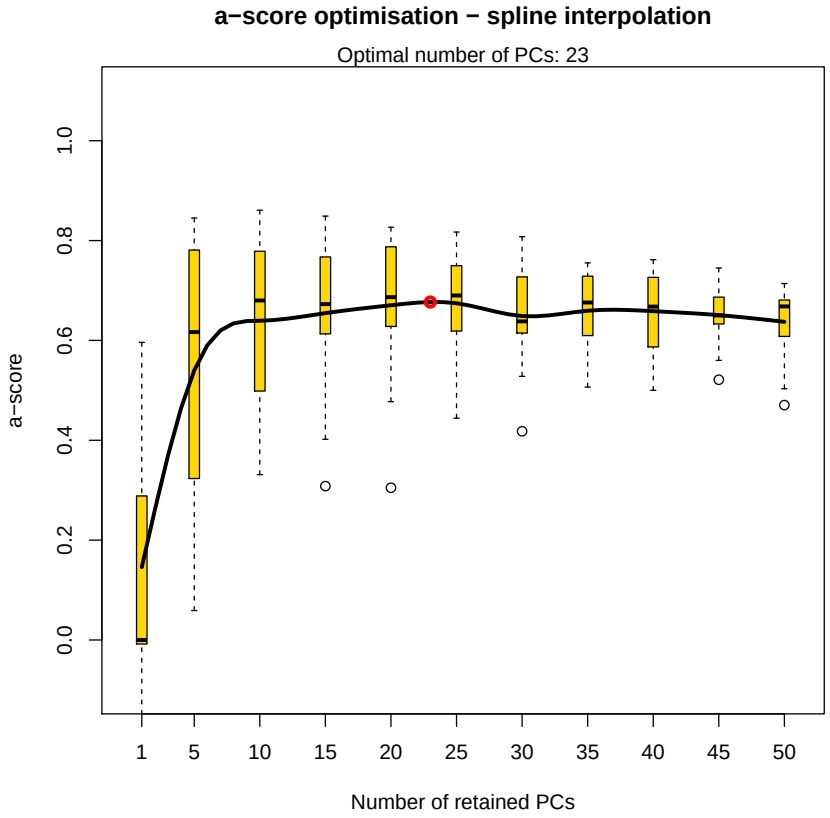
```
dapc2 <- dapc(microbov, n.da=100, n.pca=10)
temp <- a.score(dapc2)
names(temp)
```

```
## [1] "tab"       "pop.score" "mean"

temp$tab[1:5,1:5]

##       Borgou Zebu Lagunaire     NDama Somba
## sim.1   0.74 0.84 0.8235294 0.5666667  0.58
## sim.2   0.76 0.72 0.8627451 0.5333333  0.78
## sim.3   0.60 0.78 0.8627451 0.5000000  0.70
## sim.4   0.64 0.74 0.8627451 0.5333333  0.70
## sim.5   0.72 0.76 0.9607843 0.5000000  0.80

temp$pop.score

##          Borgou            Zebu       Lagunaire           NDama
##       0.6380000       0.7420000       0.8509804       0.5166667
##           Somba          Aubrac        Bazadais BlondeAquitaine
##       0.7180000       0.4940000       0.8382979       0.3016393
##    BretPieNoire       Charolais          Gascon        Limousin
##       0.4741935       0.5563636       0.6640000       0.4280000
##      MaineAnjou     Montbeliard          Salers
##       0.8653061       0.6733333       0.7720000

temp$mean

## [1] 0.6355187
```

The number of retained PCs can be chosen so as to optimize the a-score; this is achived by `optim.a.score`:

```
dapc2 <- dapc(microbov, n.da=100, n.pca=50)
```

```
temp <- optim.a.score(dapc2)
```

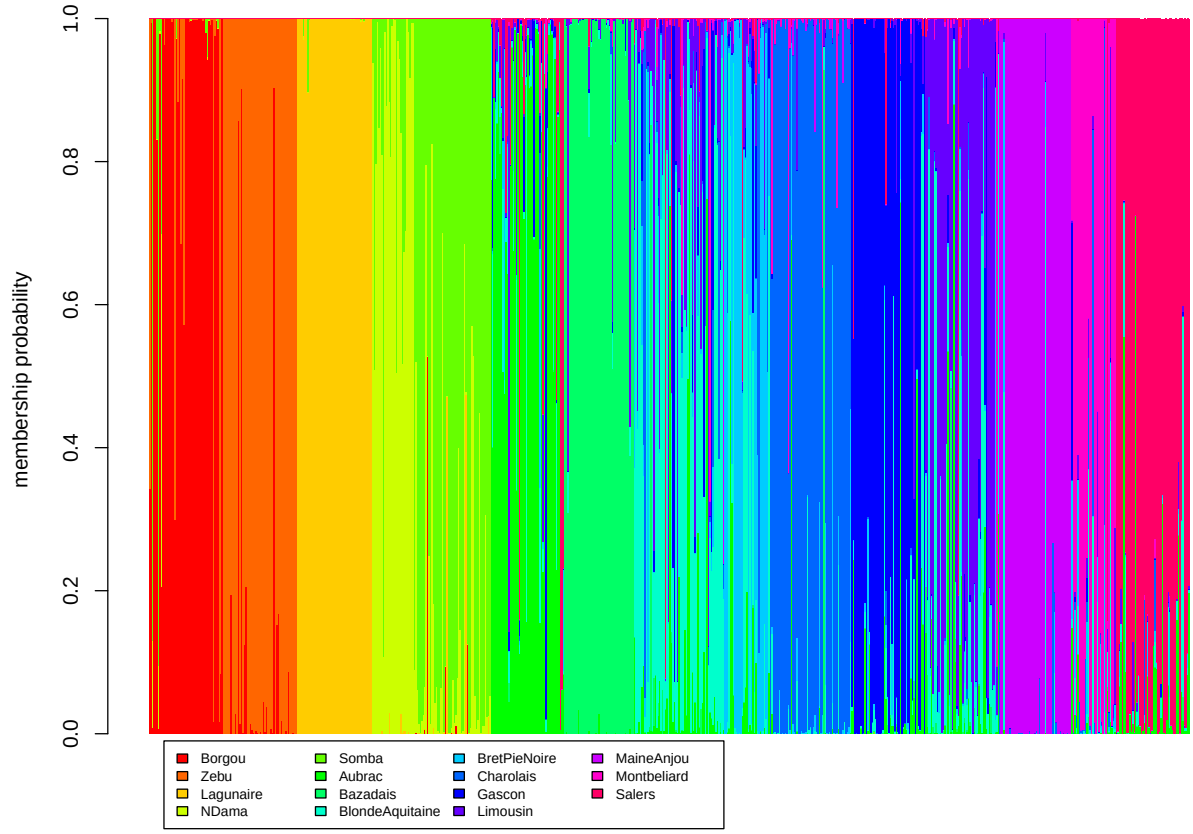


Since evaluating solutions for 1, 2, ... 100 retained PCs is unusefully computer-intensive, as a first approximation the method evaluates a few numbers of retained PCs in this range, and uses spline interpolation to approximate the optimal number of PCs to retain. Then, one can evaluate all solutions within a restrained range using the argument `n.pca`. For the `microbov` dataset, we should probably retain between 10 and 30 PCs during the dimension-reduction step.

We perform the analysis with 20 PCs retained, and then map the membership probabilities as before:

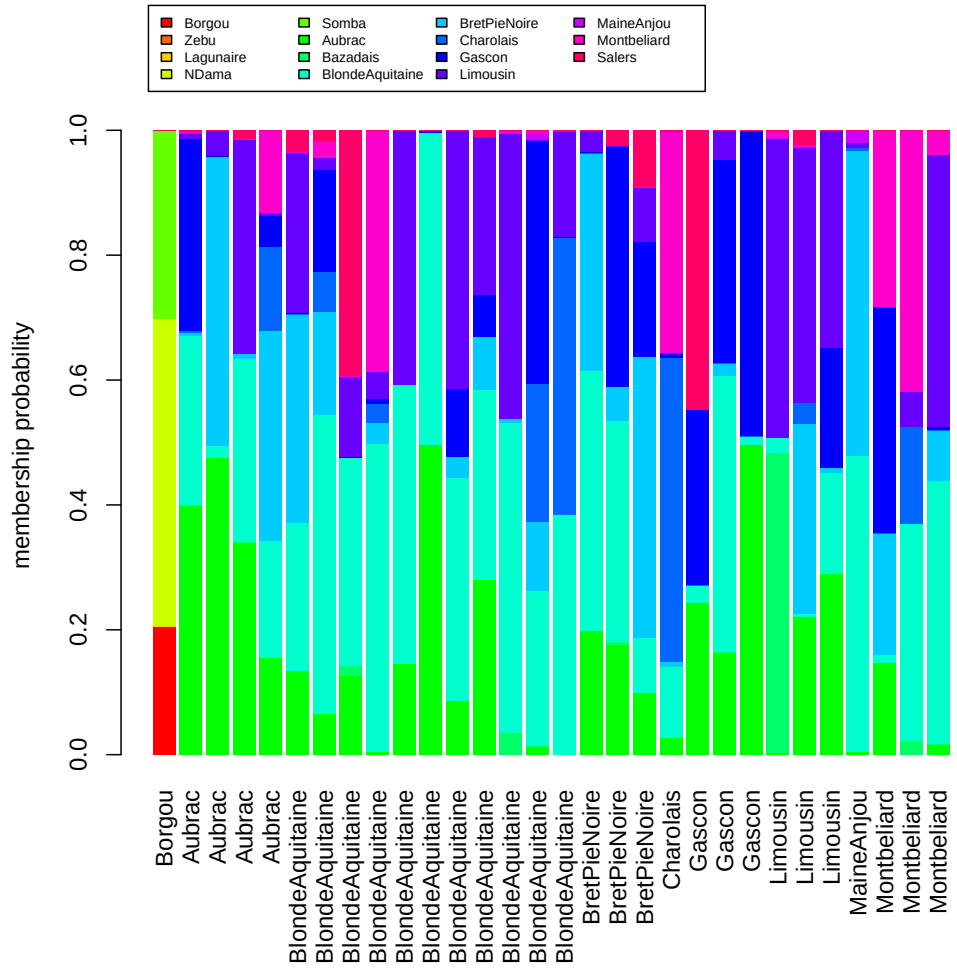
```
dapc3 <- dapc(microbov, n.da=100, n.pca=20)
myCol <- rainbow(15)
```

```
par(mar=c(5.1,4.1,1.1,1.1), xpd=TRUE)
compoplot(dapc3, lab="", posi=list(x=12,y=-.01), cleg=.7)
```



And as before, we can further investigate admixed individuals, which we arbitrarily define as those having no more than 0.5 probability of membership to any group:

```
temp <- which(apply(dapc3$posterior,1, function(e) all(e<0.5)))
temp

##  AFBIBOR9511  FRBTAUB9062  FRBTAUB9070  FRBTAUB9078  FRBTAUB9225
##            9          233          241          249          265
## FRBTBDA29851 FRBTBDA29856 FRBTBDA29879 FRBTBDA35248 FRBTBDA35256
##          329          334          354          361          363
## FRBTBDA35259 FRBTBDA35267 FRBTBDA35278 FRBTBDA35281 FRBTBDA35877
##          365          368          372          374          382
## FRBTBDA35941  FRBTBPN1906  FRBTBPN1913  FRBTBPN1915 FRBTCHA15957
##          386          405          409          411          422
## FRBTGAS14183  FRBTGAS9173  FRBTGAS9200 FRBTLIM30832 FRBTLIM30839
##          477          498          520          543          550
## FRBTLIM30855  FRBTMA25298  FRBTMBE1496  FRBTMBE1514  FRBTMBE1544
##          566          579          625          636          651

lab <- pop(microbov)
par(mar=c(8,4,5,1), xpd=TRUE)
compoplot(dapc3, subset=temp, cleg=.6, posi=list(x=0,y=1.2), lab=lab)
```



Admixture appears to be the strongest between a few breeds (Blonde d'Aquitaine, Bretonne Pie-Noire, Limousine and Gascone). Some features are fairly surprising; for instance, the last individual is fairly distant from its cluster, but has almost 50% chances of being assigned to two other breeds.

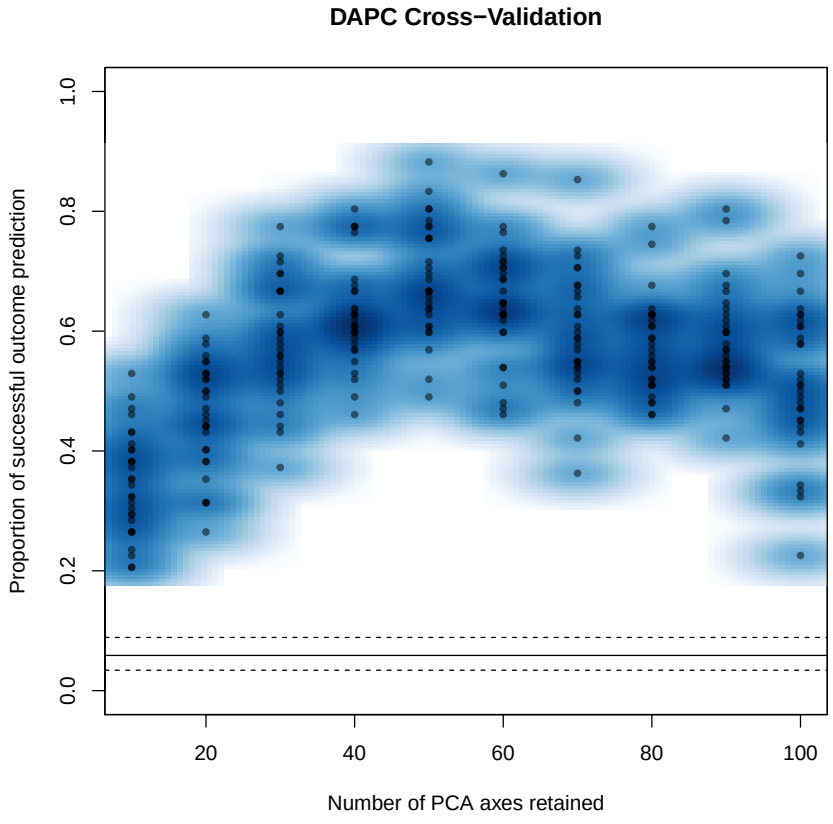
4.3 Using cross-validation

Carrying out a DAPC requires the user to define the number of PCs retained in the analysis. As discussed above, this is not a trivial decision, as the number of PCs can have a substantial impact on the results of the analysis. Cross-validation (carried out with the function `xvalDapc`) provides an objective optimisation procedure for identifying the 'golidlocks point' in the trade-off between retaining too few and too many PCs in the model. In cross-validation, the data is divided into two sets: a training set (typically comprising 90% of the data) and a validation set (which contains the remainder (by default, 10%) of the data). With `xvalDapc`, the validation set is selected by stratified random sampling: this ensures that at least one member of each group or population in the original data is represented in both training and validation sets.

DAPC is carried out on the training set with variable numbers of PCs retained, and the degree to which the analysis is able to accurately predict the group membership of excluded

individuals (those in the validation set) is used to identify the optimal number of PCs to retain. At each level of PC retention, the sampling and DAPC procedures are repeated `n.rep` times. (By default, we perform 30 replicates, though it should be noted that for large datasets, performing large numbers of replicates may be computationally intensive). Here is an example using the `nancycats` dataset:

```
data(nancycats)
x <- nancycats
mat <- as.matrix(na.replace(x, method="mean"))
grp <- pop(x)

xval <- xvalDapc(mat, grp, n.pca.max = 300, training.set = 0.9,
                 result = "groupMean", center = TRUE, scale = FALSE,
                 n.pca = NULL, n.rep = 30, xval.plot = TRUE)
```




```
xval[2:6]

## $`Median and Confidence Interval for Random Chance`
##       2.5%        50%      97.5%
```

```
## 0.03300235 0.05885587 0.09214083
##
## $`Mean Successful Assignment by Number of PCs of PCA`
##        10        20        30        40        50        60        70
## 0.3313725 0.4699346 0.6016340 0.6418301 0.6669935 0.6633987 0.6147059
##        80        90       100
## 0.5843137 0.5366013 0.5460784
##
## $`Number of PCs Achieving Highest Mean Success`
## [1] "50"
##
## $`Root Mean Squared Error by Number of PCs of PCA`
##        10        20        30        40        50        60        70
## 0.6744832 0.5389899 0.4083935 0.3694574 0.3374078 0.3483820 0.3990449
##        80        90       100
## 0.4244845 0.4678296 0.4615173
##
## $`Number of PCs Achieving Lowest MSE`
## [1] "50"
```

When `xval.plot` is `TRUE`, a scatterplot of the DAPC cross-validation is generated. The number of PCs retained in each DAPC varies along the x-axis, and the proportion of successful outcome prediction varies along the y-axis. Individual replicates appear as points, and the density of those points in different regions of the plot is displayed in blue.
As one might expect (or hope) for an optimisation procedure, the results of cross-validation here take on an arc-like shape. Predictive success is sub-optimal with both too few and too many retained PCA axes. At the apex of this arc, we that we are able to achieve 60% - 70% predictive success and an associated root mean squared error (RMSE) of 30% - 40%. While in this example, the number of PCs associated with the highest mean success is also associated with the lowest MSE, this is not always the case. Based on the model validation literature, we recommend using the number of PCs associated with the lowest RMSE as the 'optimum' `n.pca` in the DAPC analysis. Hence, we return this `dapc` object as the seventh component of the output of `xvalDapc`.

5 Using supplementary individuals

5.1 Rationale

Statistically speaking, supplementary individuals are observations which do not participate in constructing a model, but which we would like to predict using a model fitted on other ("training") data. In the context of DAPC, we may know groups for most individuals, but some individuals could be of unknown or uncertain group. In this case, we need to exclude individuals from the analysis, and then project them as supplementary individuals onto the discriminant functions. The only requirement for this operation is that supplementary individuals have been typed for the same loci as the rest of the dataset.

Technically, using supplementary individuals consists in transforming the new data using the centring and scaling of the "training data", and then using the same discriminant coefficients as for the contributing individuals to predict the position of the new individuals onto the discriminant functions.

5.2 In practice

We will illustrate the practice of supplementary individuals using the cattle breeds data previously analyzed (`microbov` dataset). We first split the dataset into two parts: one used for the analysis, and one used as supplementary individuals:

```
data(microbov)
set.seed(2)
kept.id <- unlist(tapply(1:nInd(microbov), pop(microbov),
                         function(e) sample(e, 20,replace=FALSE)))
x <- microbov[kept.id]
x.sup <- microbov[-kept.id]
nInd(x)

## [1] 300

nInd(x.sup)

## [1] 404
```

`x` is a `genind` containing the data to be analyzed; `x.sup` contains the supplementary individuals.

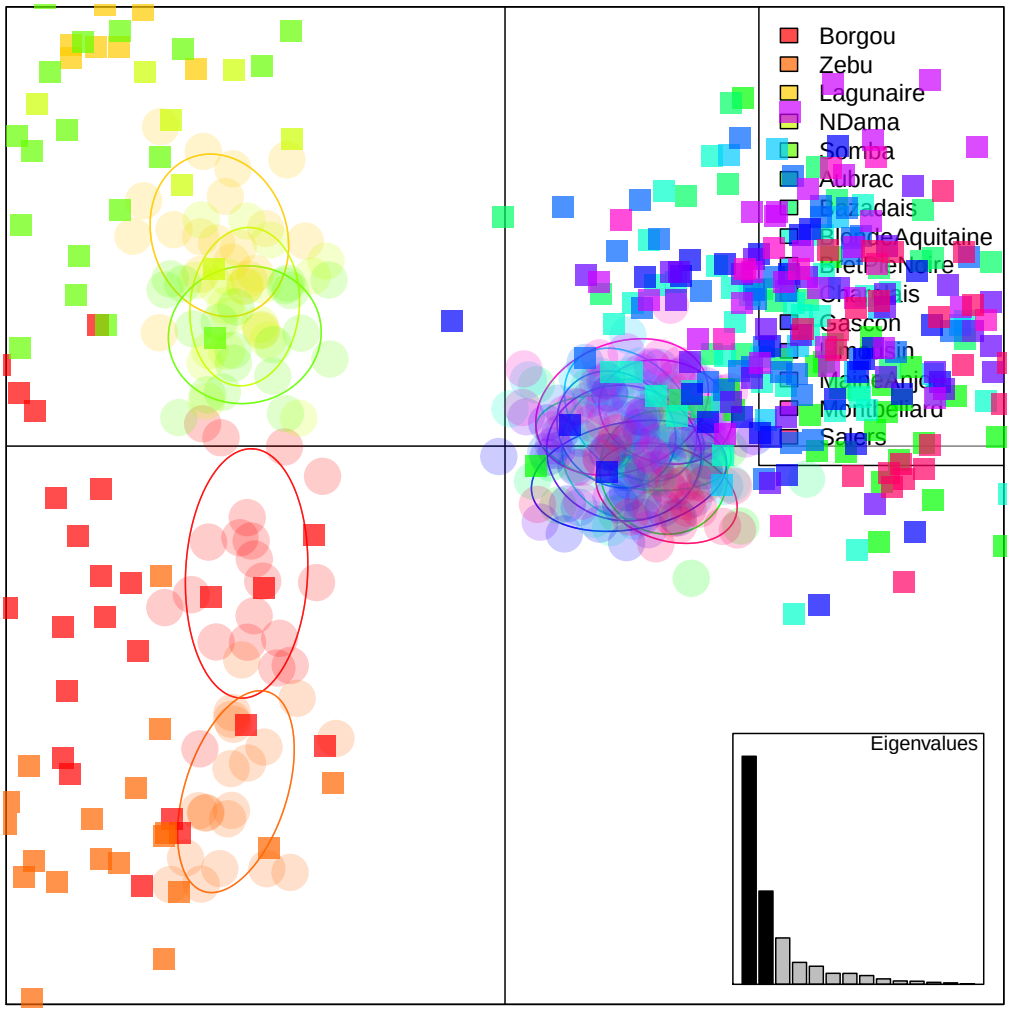
We perform the DAPC of `x`, and use `predict` to predict results for the supplementary individuals:

```
dapc4 <- dapc(x,n.pca=20,n.da=15)
pred.sup <- predict.dapc(dapc4, newdata=x.sup)
names(pred.sup)

## [1] "assign"     "posterior"  "ind.scores"

head(pred.sup$assign)

## [1] Zebu   Zebu   Borgou Borgou NDama  Borgou
## 15 Levels: Borgou Zebu Lagunaire NDama Somba Aubrac ... Salers

pred.sup$ind.scores[1:5,1:3]

##                   LD1       LD2        LD3
## AFBIBOR9503 -6.515239 -7.748033  1.7886050
## AFBIBOR9504 -3.611382 -6.015427  1.3014342
```

```
## AFBIBOR9506 -8.106407 -2.605669  1.8573153
## AFBIBOR9507 -8.560284 -1.805413  0.3964035
## AFBIBOR9511 -8.158395  2.427946 -0.9648335

round(pred.sup$posterior[1:5, 1:5],3)

##             Borgou  Zebu Lagunaire NDama Somba
## AFBIBOR9503  0.000 1.000         0 0.000 0.000
## AFBIBOR9504  0.337 0.663         0 0.000 0.000
## AFBIBOR9506  1.000 0.000         0 0.000 0.000
## AFBIBOR9507  0.969 0.000         0 0.000 0.031
## AFBIBOR9511  0.000 0.000         0 0.967 0.033
```

The list `pred.sup` contains all the predictions about the new data based on the analysis stored in `dapc4`. The slot `assign` contains the assignment of new individuals to groups; `ind.scores` contains the coordinates of the new individuals on the discriminant functions; `posterior` contains the posterior membership probabilities. We can visualize the information by different ways. First, we can represent the new individuals using a scatterplot:

```
col <- rainbow(length(levels(pop(x))))
col.points <- transp(col[as.integer(pop(x))],.2)
scatter(dapc4, col=col, bg="white", scree.da=0, pch="",
        cstar=0, clab=0, xlim=c(-10,10), legend=TRUE)
par(xpd=TRUE)
points(dapc4$ind.coord[,1], dapc4$ind.coord[,2], pch=20,
       col=col.points, cex=5)
col.sup <- col[as.integer(pop(x.sup))]
points(pred.sup$ind.scores[,1], pred.sup$ind.scores[,2], pch=15,
       col=transp(col.sup,.7), cex=2)
add.scatter.eig(dapc4$eig,15,1,2, posi="bottomright", inset=.02)
```

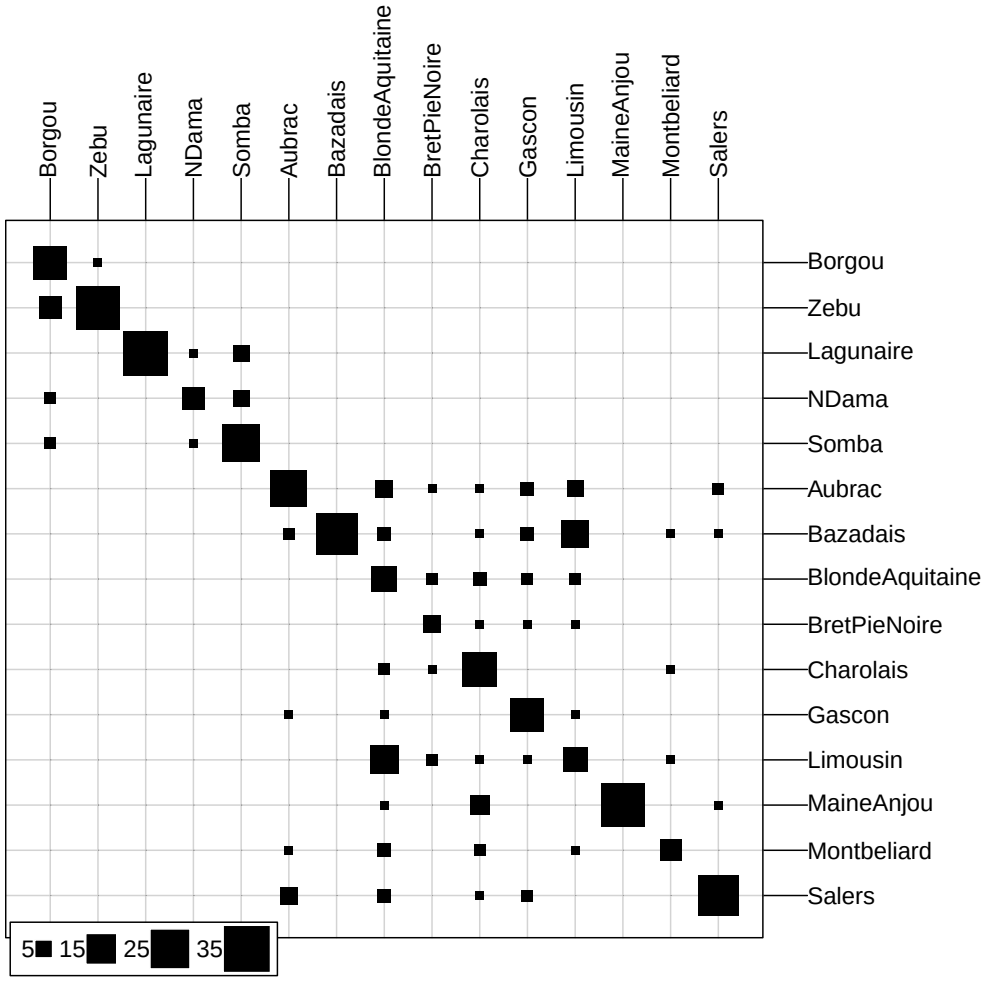


Light dots and ellipses correspond to the original analysis, while more solid squares indicate supplementary individuals. Results are fairly satisfying:

```
mean(as.character(pred.sup$assign)==as.character(pop(x.sup)))

## [1] 0.6905941
```

Around 69% of individuals have been assigned to their actual cluster. For more details about which breed was assigned to which cluster, we can display the contingency table of the actual cluster *vs* the inferred one:

```
table.value(table(pred.sup$assign, pop(x.sup)), col.lab=levels(pop(x.sup)))
```



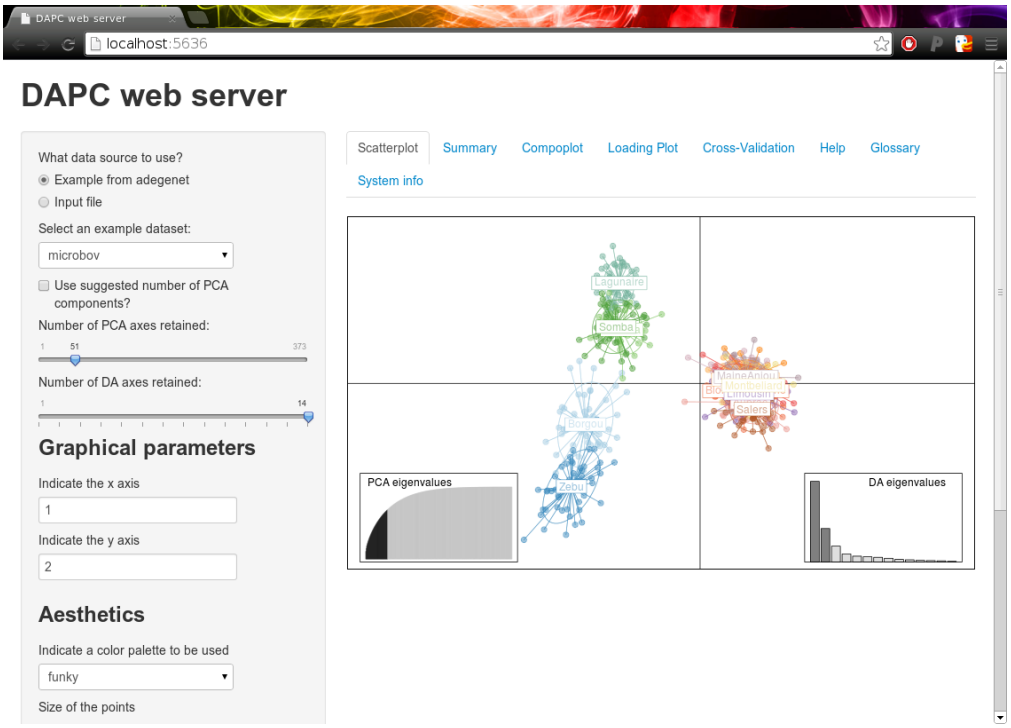
Columns correspond to actual clusters of the supplementary individuals, while rows correspond to inferred clusters. Overall, groups are fairly well retrieved, but we can notice that individuals of Blonde d'Aquitaine breed are poorly identified compared to other breeds.

6 A web interface for DAPC

As of version 1.4-0 of adegenet, an interactive web interface for DAPC is distributed with the package. It is started by typing:

```
adegenetServer(what = "DAPC")
```

This will open up the web browser used by default in R. This application should ressemble:



The corresponding address (e.g. "http://localhost:3076/") can be copied and pasted into a different web browser if needed. This interface is best used with google chrome: `http://www.google.com/chrome/` For further information, look at the documentation within the application.

References

[1] Jombart T, Devillard S and Balloux, F (2010). Discriminant analysis of principal components: a new method for the analysis of genetically structured populations. *BMC Genetics* 11: 94.

[2] Jombart, T. (2008) adegenet: a R package for the multivariate analysis of genetic markers. *Bioinformatics* 24: 1403-1405.

[3] R Development Core Team (2011). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.