Genome-Wide Association Studies

Caitlin Collins[*], Thibaut Jombart

Imperial College London

MRC Centre for Outbreak Analysis and Modelling

August 6, 2015

Abstract

This practical provides an introduction to Genome-Wide Association Studies (GWAS) in R . First, we will examine population structures within the data. Second, we will test for associations between a genome-wide SNP panel and our phenotypic trait of interest: antibiotic resistance. We will carry out the test of association and perform feature selection with three separate methods: (i) The *univariate* **Fisher's exact test**, (ii) the *multivariate* penalized regression technique **LASSO**, and (iii) with an extension of the *multivariate* factorial method **DAPC**. Following these initial tests of association, we will correct for population stratification via two methods (PCA and DAPC). We will re-run GWAS, applying all three methods of association testing and feature selection to the "corrected" dataset, and compare the results.

[*] `caitlin.collins12@imperial.ac.uk`

Contents

1 The data

Before we begin, we will need to (install and) load the required packages.

```
library(devtools)
install_github("thibautjombart/adegenet")
library(adegenet)

install.packages("glmnet", dep=TRUE)
library(glmnet)
```

```
## Loading required package:  ade4
##
##    /// adegenet 2.0.1 is loaded ////////////
##
##    > overview:  '?adegenet'
##    > tutorials/doc/questions:  'adegenetWeb()'
##    > bug reports/feature resquests:  adegenetIssues()
##
##
## Loading required package:  Matrix
##
## Attaching package:  'Matrix'
##
## The following objects are masked from 'package:base':
##
##    crossprod, tcrossprod
##
## Loading required package:  foreach
## Loaded glmnet 2.0-2
```

The simulated data used in this practical is saved under the name "simGWAS.Rdata". The dataset is in `R` 's binary format (extension `RData`), which uses compression to store data efficiently.

Please set your working directory to the location of the "simGWAS" file on your computer (replace `‘‘PATH/TO/WORKING_DIRECTORY’’` in the code below with the correct path for your computer). Then, using `get(load())`, read the data into `R` . A new object, `simGWAS`, should appear in your `R` environment:

```
setwd("PATH/TO/WORKING_DIRECTORY/")
simGWAS <- get(load("./simGWAS.Rdata"))
```

Let's take a quick look at the format and dimensions of of `simGWAS`.

```
class(simGWAS)

## [1] "list"

names(simGWAS)

## [1] "snps" "phen"
```

The object **simGWAS** is a list containing the two components required for any GWAS: (i) genetic variables stored in **$snps**, a matrix of Single Nucleotide Polymorphism (SNPs) data, and (ii) the phenotype of the different sampled isolates, stored in **$phen**.

The SNPs data has a modest size by GWAS standards:

```
print(object.size(simGWAS$snps), unit="Mb")

## 7.8 Mb
```

Note: If we were dealing with a larger SNP dataset, we might want to work with our data in the **genlight** format, which can reduce RAM requirements by over 50 times! (See **adegenetTutorial("genomics")** for details.)

Let's take a look at the object contained in **$snps**...

```
class(simGWAS$snps)

## [1] "matrix"

dim(simGWAS$snps)

## [1]    95 10050

simGWAS$snps[1:10,1:8]

##            L00001 L00002 L00003 L00004 L00005 L00006 L00007 L00008
## isolate-1       0      1      1      1      0      1      0      1
## isolate-2       1      1      1      1      0      1      0      0
## isolate-3       1      1      1      1      0      1      0      0
## isolate-4       1      1      0      1      0      0      1      0
## isolate-5       1      1      1      1      0      1      0      0
## isolate-6       0      1      1      1      0      1      0      0
## isolate-7       0      1      1      1      0      1      1      0
## isolate-8       1      1      0      1      0      1      0      0
## isolate-9       1      0      0      1      0      1      1      0
## isolate-10      0      1      1      1      1      1      1      1
```

And we see that it contains a matrix with 95 individuals (haploid bacterial isolates) in the rows, and 10,050 binary SNPs in the columns (alleles coded as 0/1).

For each of the 95 individuals, a phenotype is stored in the factor `$phen`:

```
class(simGWAS$phen)

## [1] "factor"

length(simGWAS$phen)

## [1] 95

simGWAS$phen

##  [1] S S S S S R R S S S S R S S S S S R S R S S S R S S S R R S R R R R R
## [36] S S R R S S S R R R S S R R R R R R R S S S R R R R R S S S R R S R
## [71] R R R S S R S R S S S S S S S S S S S S S S S R S
## Levels: R S

table(simGWAS$phen)

##
##  R  S
## 40 55
```

In this analysis, the phenotype of interest is antibiotic resistance. The “R” and “S” in the above table stand for the two levels of this phenotype: “Resistant” and “Susceptible”.

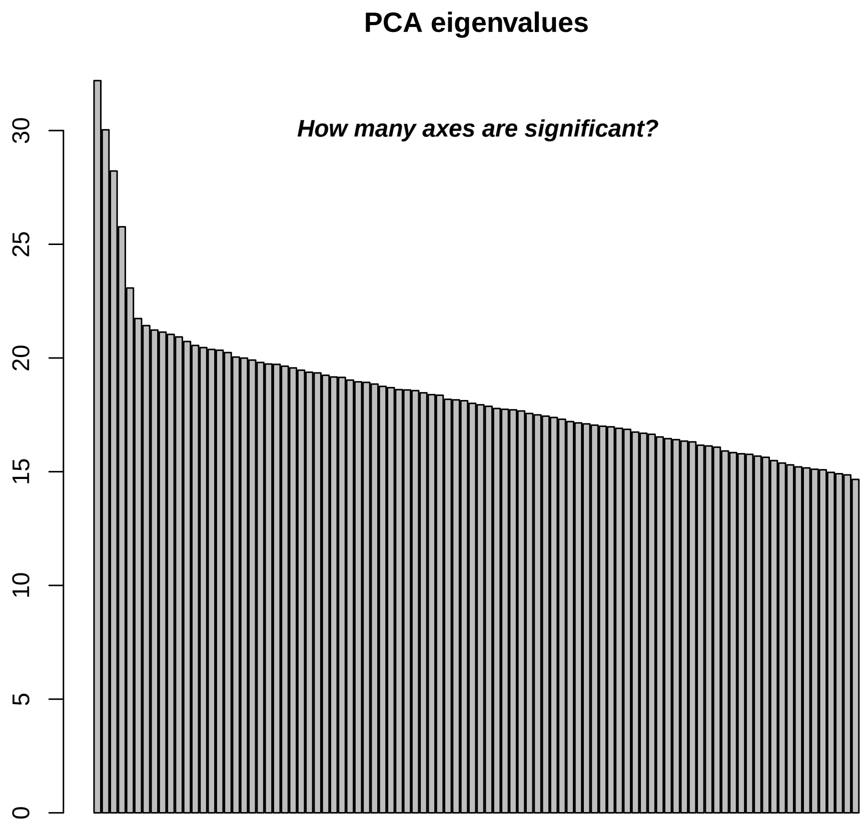
To simplify further commands, we create the new objects `snps` and `phen` from `simGWAS`:

```
snps <- simGWAS$snps
phen <- factor(simGWAS$phen)
```

2 First assessment of the genetic diversity

Principal Component Analysis (PCA) is a very powerful tool for reducing the diversity contained in massively multivariate data into a few synthetic variables (the principal components — PCs). We will run PCA with the `dudi.pca` function from the *ade4* package (a dependency of *adegenet*), specifying that variables should not be scaled (`scale=FALSE`) to unit variances (this is only useful when variables have inherently different scales of variation, which is not the case here):

```
pca1 <- dudi.pca(snps, scale=FALSE)
```



The method displays a screeplot (barplot of eigenvalues) to help the user decide how many PCs should be retained. The general rule is to retain only the largest eigenvalues, after which non-structured variation results in smoothly decreasing eigenvalues. How many PCs would you retain here?

The object `pca1` contains various information.

```
pca1

## Duality diagramm
## class: pca dudi
## $call: dudi.pca(df = snps, scale = FALSE, scannf = FALSE, nf = 5)
##
## $nf: 5 axis-components saved
## $rank: 94
## eigen values: 32.2 30.03 28.22 25.77 23.08 ...
##   vector length mode    content
## 1 $cw    10050  numeric column weights
## 2 $lw    95     numeric row weights
## 3 $eig   94     numeric eigen values
##
##   data.frame nrow  ncol  content
## 1 $tab       95    10050 modified array
## 2 $li        95    5     row coordinates
## 3 $l1        95    5     row normed scores
## 4 $co        10050 5     column coordinates
## 5 $c1        10050 5     column normed scores
## other elements: cent norm
```

Most importantly:

- `pca1$eig` contains the eigenvalues of the analysis, representing the amount of information contained in each PC.
- `pca1$li` contains the principal components.
- `pca1$c1` contains the principal axes (loadings of the variables).

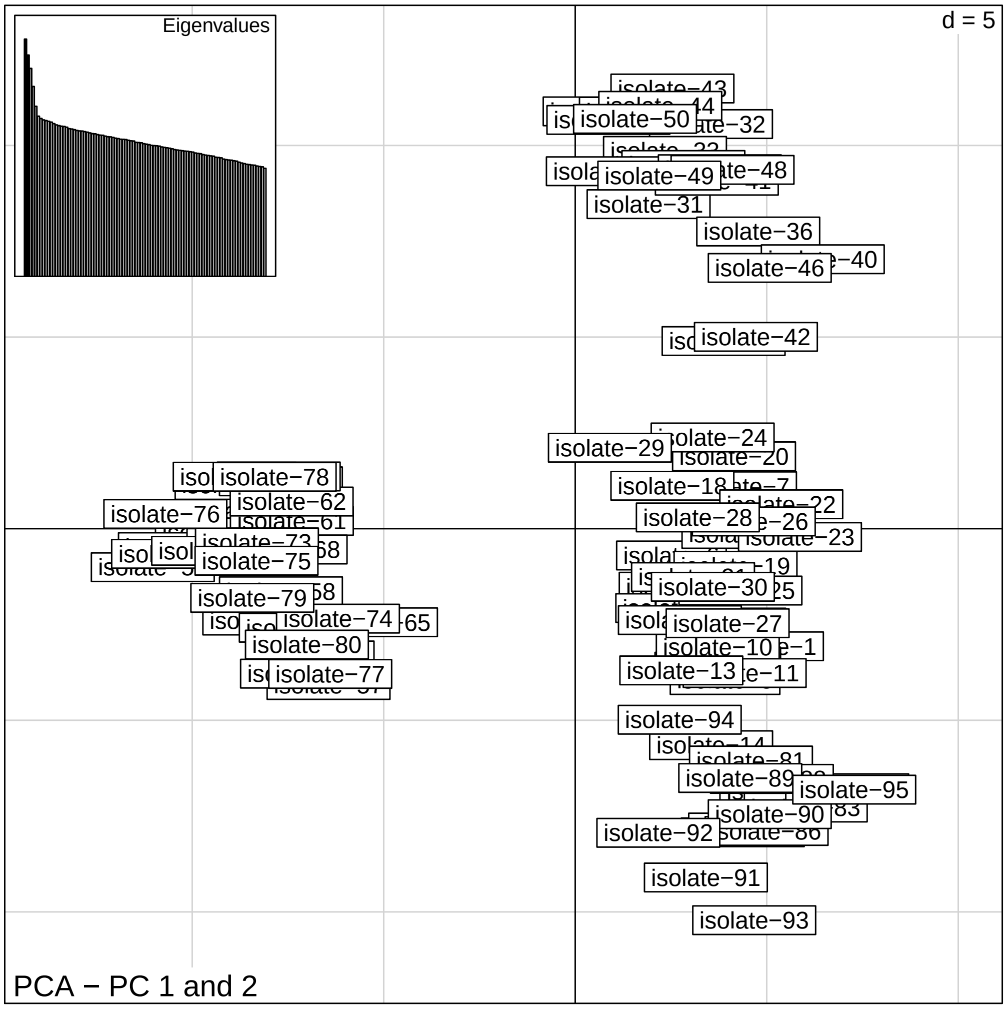
Use `head()` to examine the first few elements of each of these:

```
head(pca1$eig)

## [1] 32.19587 30.03250 28.22215 25.76779 23.07734 21.73465

head(pca1$li)

##                Axis1     Axis2      Axis3      Axis4      Axis5
## isolate-1 5.048201 -3.068272  -9.205946  -6.499608  0.7487427
## isolate-2 3.513306 -3.601273  -8.578199 -10.092913 -3.6415792
## isolate-3 3.911874 -3.951249  -8.893619  -7.501878 -2.4698956
## isolate-4 3.930861 -3.465554  -8.462628  -5.833693 -0.0363412
## isolate-5 3.755635 -2.966284  -8.378713  -8.771672 -1.7296293
## isolate-6 2.511967 -0.699552 -10.329327  -2.952949  9.8714639
```

```
head(pca1$c1)

##                  CS1           CS2           CS3           CS4
## L00001 -0.004642219  1.077777e-02 -0.0013576456 -0.0110171225
## L00002  0.000926134 -1.036157e-02 -0.0026777380 -0.0089670704
## L00003 -0.005606760 -3.342560e-03  0.0023200728 -0.0011928040
## L00004  0.002716849  1.250353e-04 -0.0005926661 -0.0001794188
## L00005 -0.003184920 -4.223935e-04  0.0049759458 -0.0062691897
## L00006 -0.001926306  5.648339e-05  0.0039054812 -0.0133130370
##                  CS5
## L00001 -0.0048392443
## L00002  0.0154976521
## L00003  0.0004408119
## L00004 -0.0021188560
## L00005 -0.0005538742
## L00006 -0.0068934484
```

Because of the large number of variables, the usual biplot (function `scatter`) is useless to visualize the results (try `scatter(pca1)` if unsure). Instead, we can represent PCs using `s.label`:

```
s.label(pca1$li, sub="PCA - PC 1 and 2")
add.scatter.eig(pca1$eig,5,1,2, ratio=.26, posi="topleft")
```



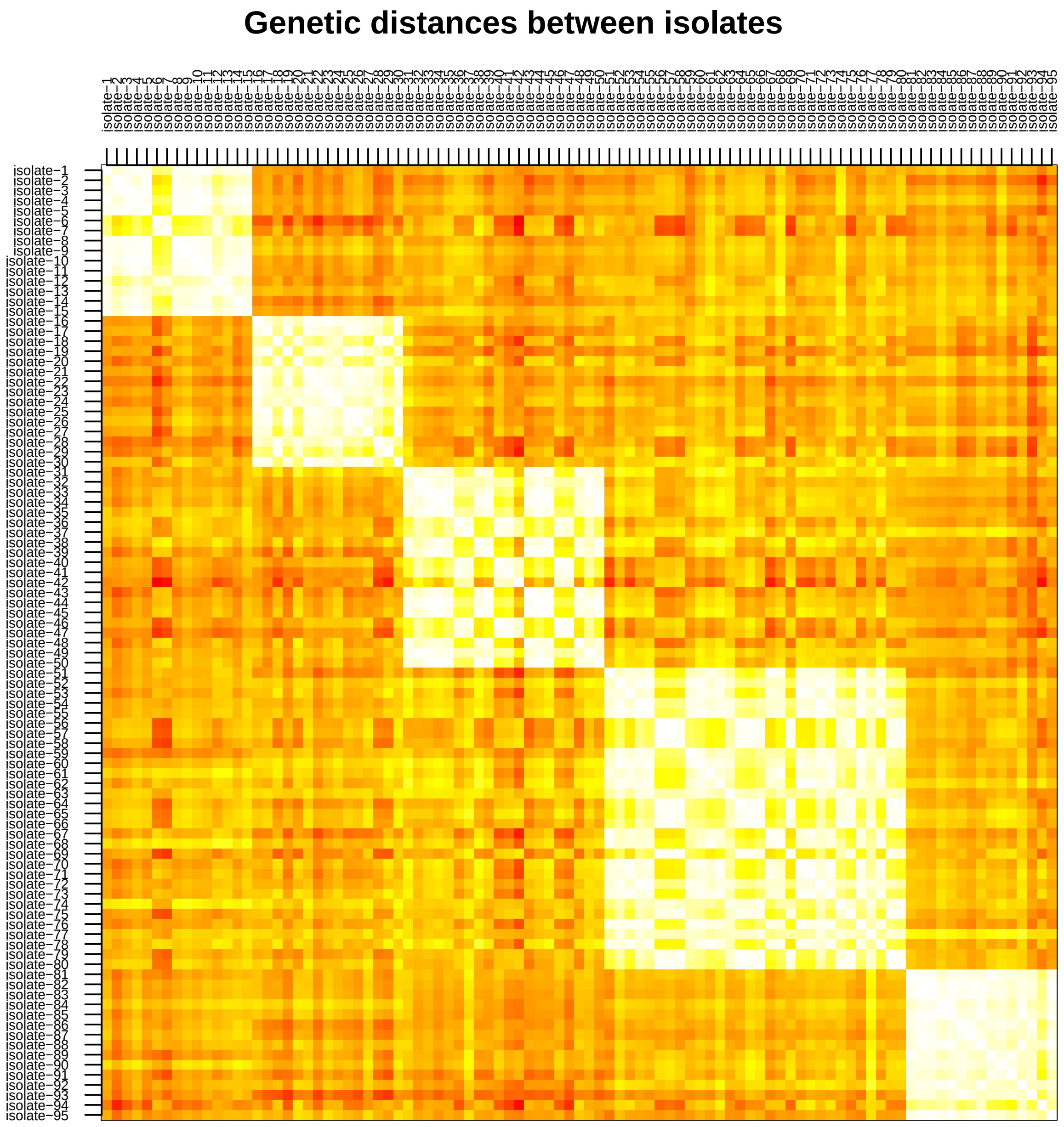
What can you say about the genetic relationships between the isolates? Are there indications of distinct lineages of bacteria? If so, how many lineages would you count?

For a more quantitative assessment of this clustering, we derive squared Euclidean distances between isolates (function `dist`) and use hierarchical clustering with complete linkage (`hclust`) to define tight clusters:

```
D <- dist(pca1$li[,1:5])^2
clust <- hclust(D, method="complete")
```

We can plot the distances stored in the `dist` object, `D`, in a heatmap with the following commands (which should look familiar from the intro-phylo practical).

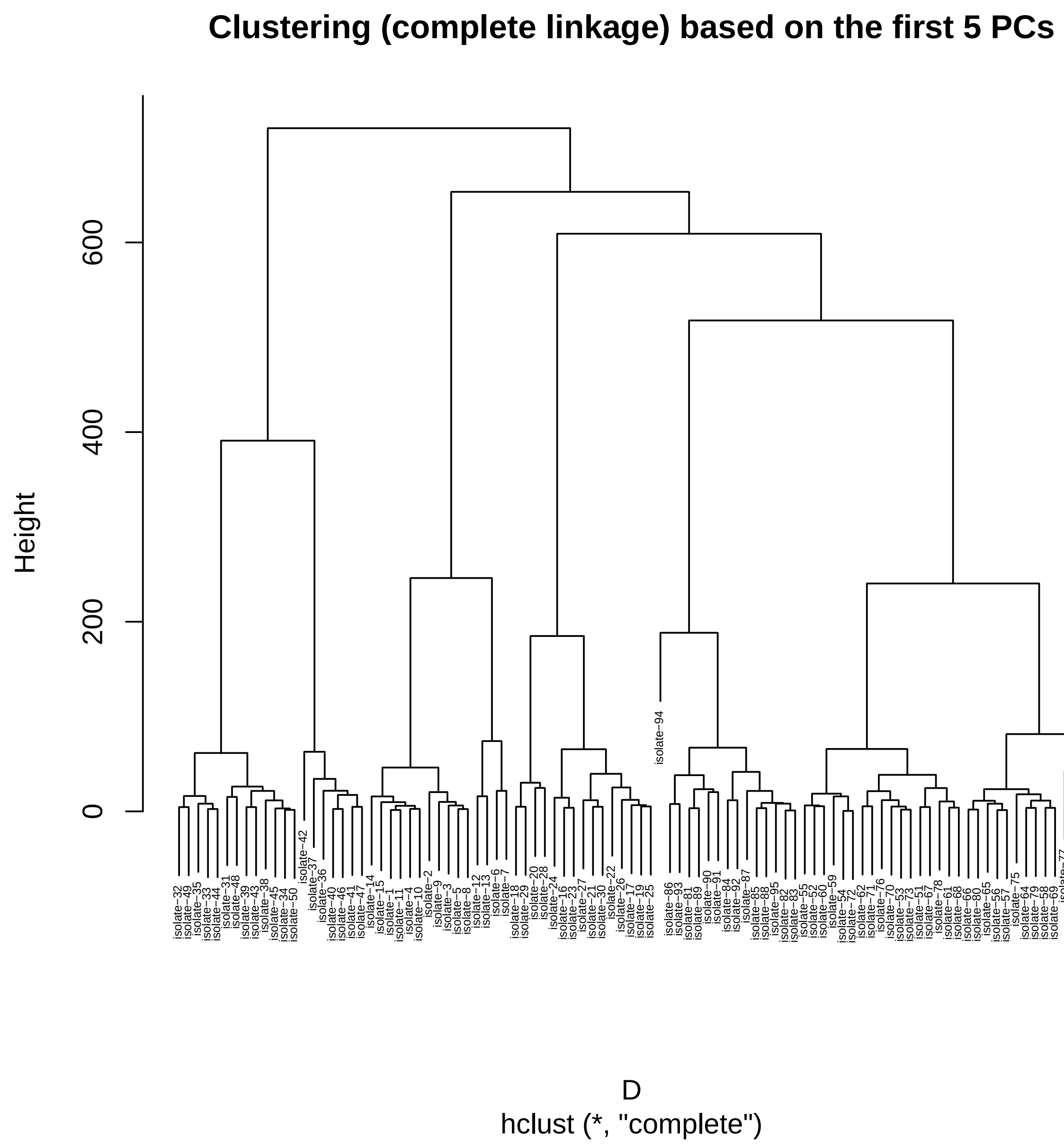
```
temp <- as.data.frame(as.matrix(D))
temp <- t(as.matrix(D))
temp <- temp[,ncol(temp):1]

par(mar=c(1,5,5,1))
image(x=1:95, y=1:95, temp, col=rev(heat.colors(nlevels(as.factor(D)))),
      xaxt="n", yaxt="n",
      xlab="",ylab="")
axis(side=2, at=1:95, lab=rev(rownames(snps)), las=2, cex.axis=.46)
axis(side=3, at=1:95, lab=rownames(snps), las=2, cex.axis=.46)

title("Genetic distances between isolates", outer=TRUE, line=-1)
```



```
## Reset those pesky margins!
par(mar=c(5.1,4.1,4.1,2.1))
```

Based on this distance matrix, what do you predict the topology of a complete-linkage hierarchical clustering tree will look like?

```r
plot(clust,
     main="Clustering (complete linkage) based on the first 5 PCs", cex=.4)
```



How many major clusters are there in the data?

Use `cutree` to define clusters based on the dendrogram:

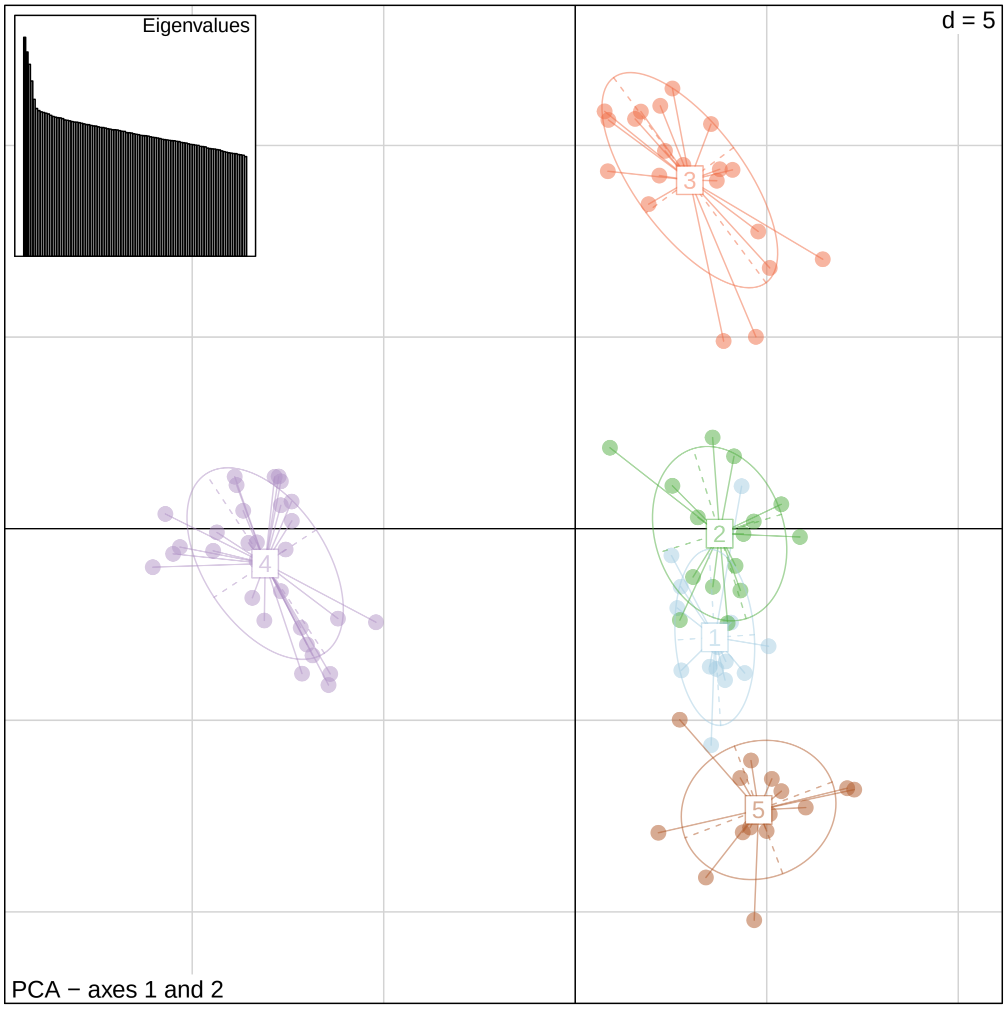
```r
pop <- factor(cutree(clust, k=5))
```

Taking a look at the object `pop`, we see that it is a factor containing 5 levels. Using `table`, we can see how many individuals belong to each cluster.

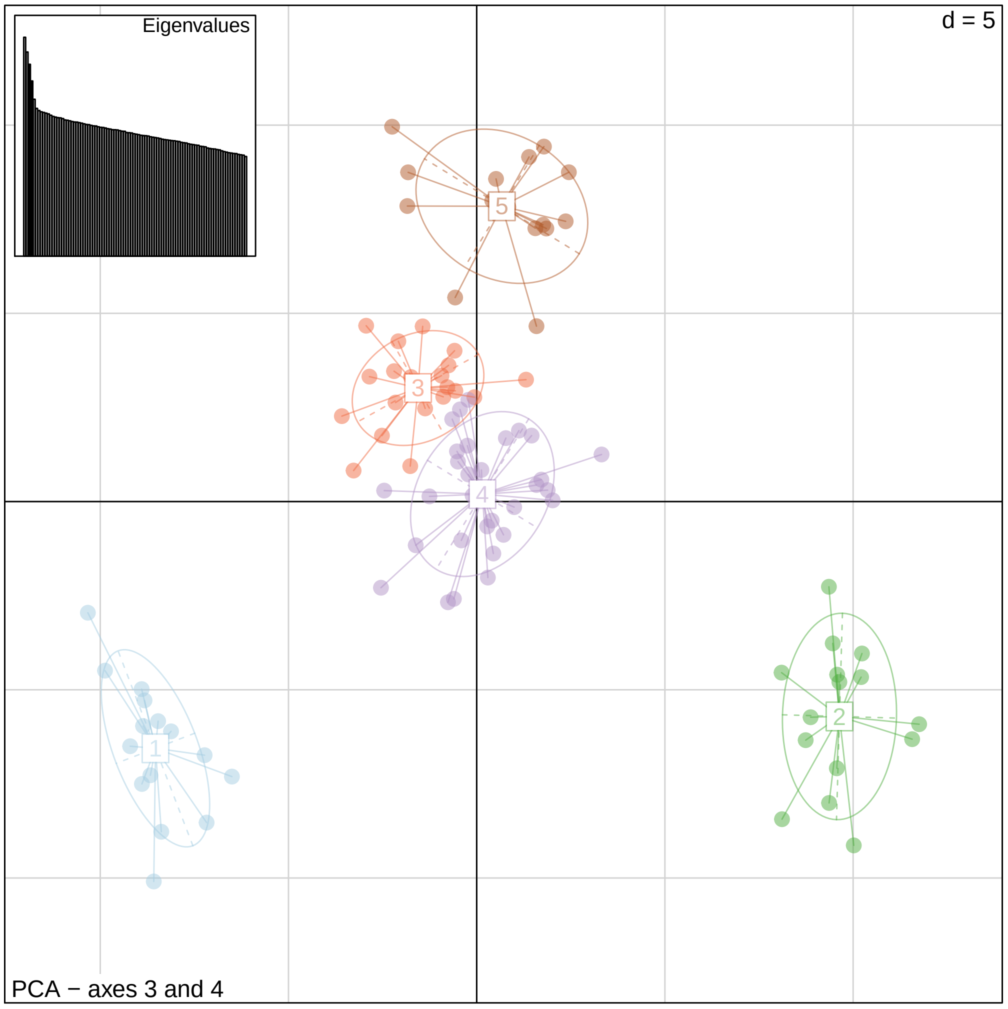
```
head(pop,20)

##  isolate-1  isolate-2  isolate-3  isolate-4  isolate-5  isolate-6
##          1          1          1          1          1          1
##  isolate-7  isolate-8  isolate-9 isolate-10 isolate-11 isolate-12
##          1          1          1          1          1          1
## isolate-13 isolate-14 isolate-15 isolate-16 isolate-17 isolate-18
##          1          1          1          2          2          2
## isolate-19 isolate-20
##          2          2
## Levels: 1 2 3 4 5

table(pop)

## pop
##  1  2  3  4  5
## 15 15 20 30 15
```

Having defined the subpopulation cluster to which each individual isolate belongs, we can now use `s.class` to represent these groups on top of the PCs with different colours and ellipses:

```
s.class(pca1$li, fac=pop, col=transp(funky(5)), cpoint=2,
        sub="PCA - axes 1 and 2")

add.scatter.eig(pca1$eig,5,1,2, ratio=.24, posi="topleft")
```

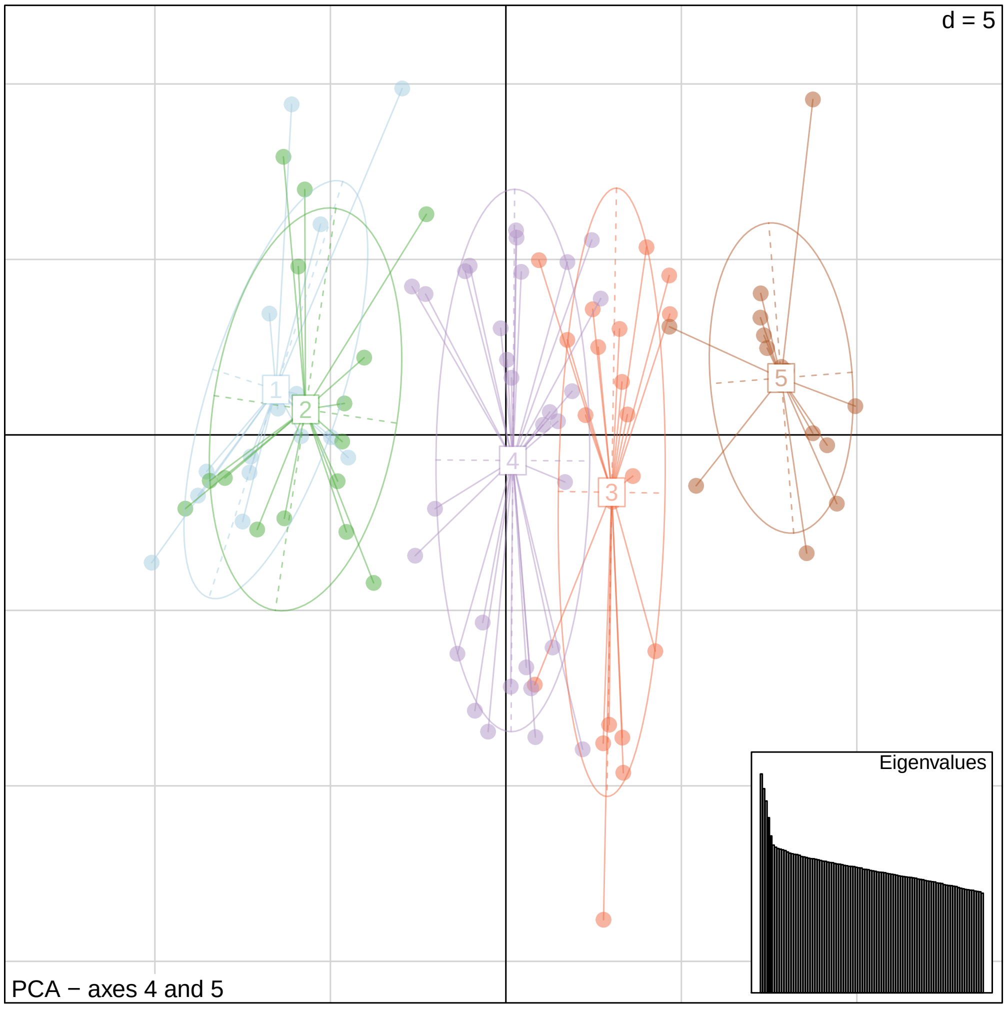


Do the same for PCs 3 and 4. (If you are unsure about how to change which axes are displayed, use `?s.class` to get a list of the arguments available for this function.)



Looking at the above two plots, what can you say about the contribution of each of the first four retained PCs to our assessment of the population structure?

Let's also take a look at PC 5:



What sort of variation does the 5th PCA axis represent? Does it contribute meaningfully to the separation of our 5 main subpopulation clusters?

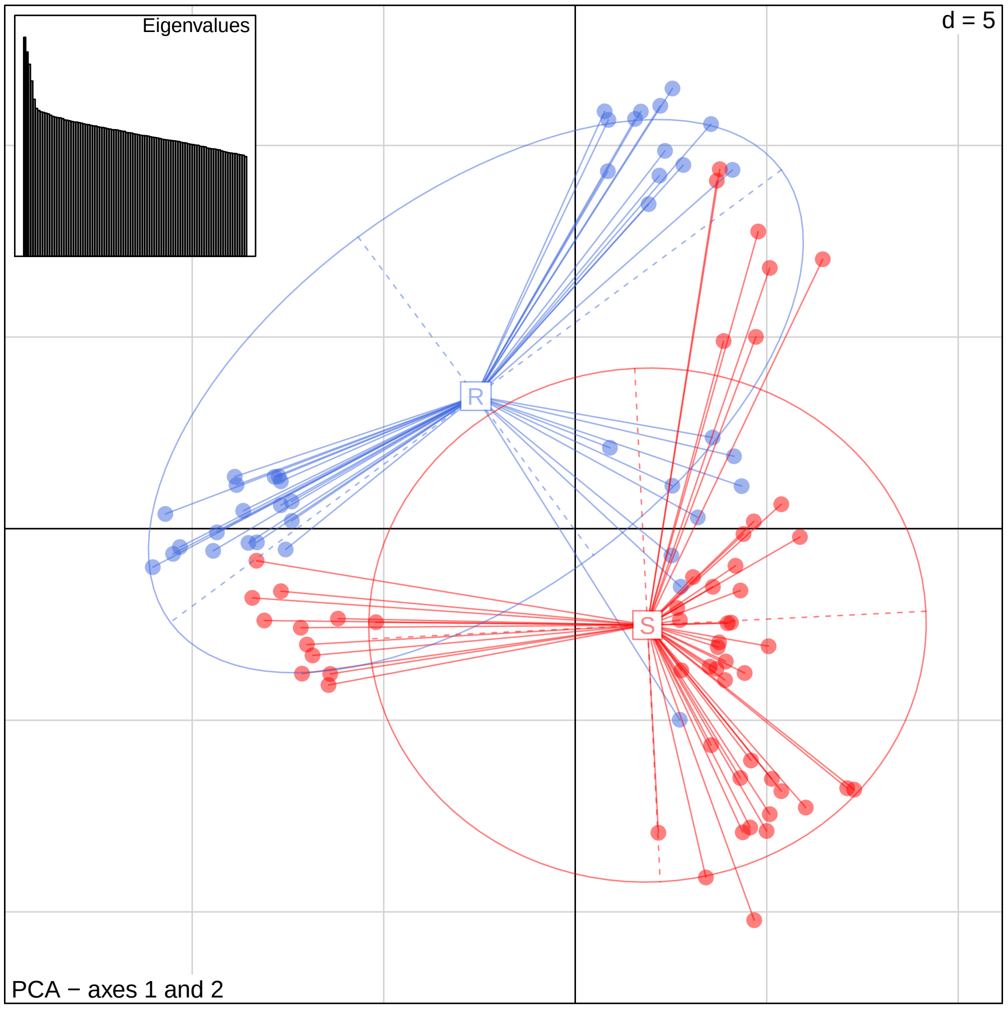
3 Assessing the extent of population stratification

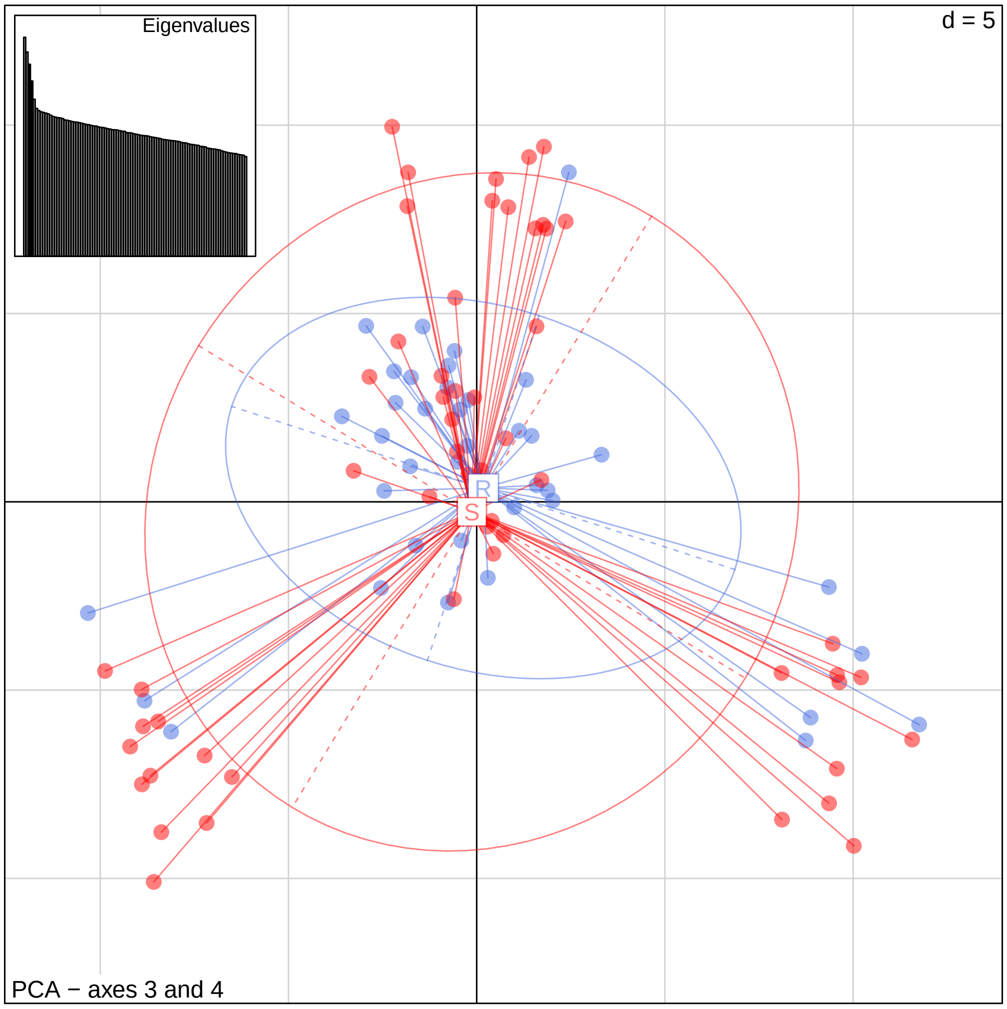
The data contained in `phen` indicate whether isolates are susceptible or resistant to a given antibiotic (`S`/`R`):

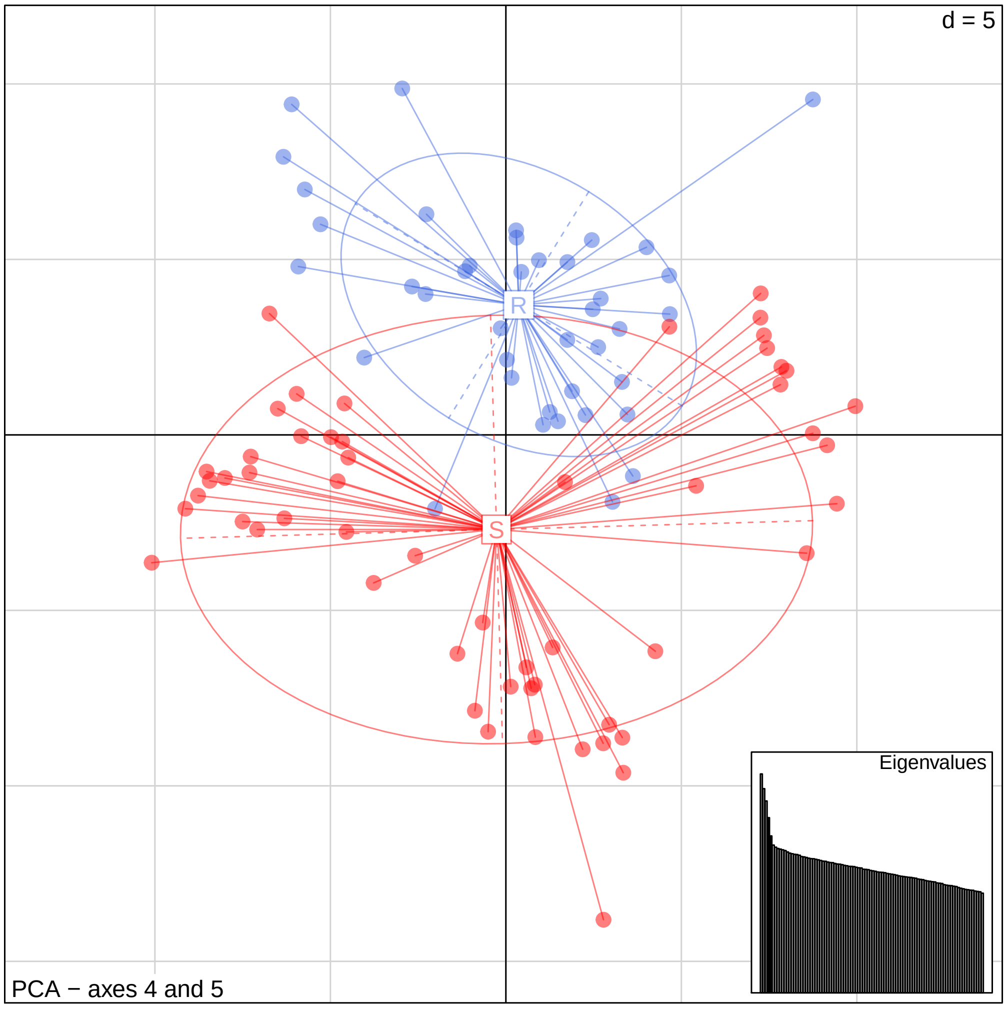
```
head(phen,10)

##  [1] S S S S S R R S S S
## Levels: R S
```

Above, we used `s.class` to visualise the main subpopulation clusters along the significant axes of PCA. Now let's use it to visualise our two *phenotypic* clusters on each of the first 5 PCs. Start by plotting the phenotypic clusters along the first two PCA axes with the following commands, and then do the same for axes 3 and 4, then 4 and 5.

```
## PCA axes 1 and 2 ##
s.class(pca1$li, fac=phen, col=transp(c("royalblue","red")), cpoint=2,
        sub="PCA - axes 1 and 2")

add.scatter.eig(pca1$eig,5,1,2, ratio=.24, posi="topleft")
```


Eigenvalues
d = 5
R
S
PCA − axes 1 and 2


Eigenvalues
d = 5
R
S
PCA − axes 3 and 4



Looking at these plots, would you say that antibiotic resistance is correlated to the main genetic features of these isolates?

Take a look at the following cross-tabulation of phenotypic groups and subpopulation clusters:

```
table(phen, pop)

##     pop
## phen  1  2  3  4  5
##    R  3  5 13 18  1
##    S 12 10  7 12 14
```

What do you think? Confirm or reject your hypothesis by performing a standard Chi-square test to check if there is an association between genetic clusters and resistance:

```
chisq.test(table(phen, pop), simulate=TRUE)

##
```

```
##  Pearson's Chi-squared test with simulated p-value (based on 2000
##  replicates)
##
## data:  table(phen, pop)
## X-squared = 19.4498, df = NA, p-value = 0.0004998
```

So what do you conclude?

Do you think we should correct for population stratification in our GWAS analysis? The most common method of correcting for population stratification in GWAS is to control for variation along the significant axes of PCA. If we adopt this approach, what complications might we run into? (Hint: Look at the two `s.class` plots showing PCA axes 4 and 5.)

4 GWAS

Before we think about correcting for population stratification, it is always a good idea to do a naive test of association (ie. without any correction for population stratification).

We will test for association with three different methods that were introduced during the lecture:

1. The *univariate* **Fisher's exact test**.
2. The *multivariate* **LASSO** method.
3. The *multivariate* **DAPC-based** approach to feature selection.

4.1 Univariate method

The first method we will use to test for association in this dataset is the most standard approach used in GWAS: the univariate **Fisher's exact test**.

We can run this test with a simple one-line command:

```
pval <- apply(snps, 2, function(e)
              fisher.test(table(factor(e, levels=c(0,1)), phen))$p.value)
```

Please take a moment to make sure you understand what each part of this function is doing. *What is meant by the number 2? What is the "e" doing?*

Let's take a look at the `pval` object we just created. What is the smallest p-value found by the Fisher's exact test? And how many p-values achieve "significance" at alpha=0.05?

```
min(pval)
# 1.002828e-27

length(which(pval < 0.05))
# 460
```

Are there really 460 SNPs that are significantly associated with antibiotic resistance in this dataset? What have we forgotten to do?

Recall that because the Fisher's exact test is a *univariate* approach, we are carrying out one Fisher test for every SNP in our dataset (that's 10,050 tests!). This means that we also need to correct for multiple testing. We will do this with two different methods and compare the results.

4.1.1 Bonferroni correction

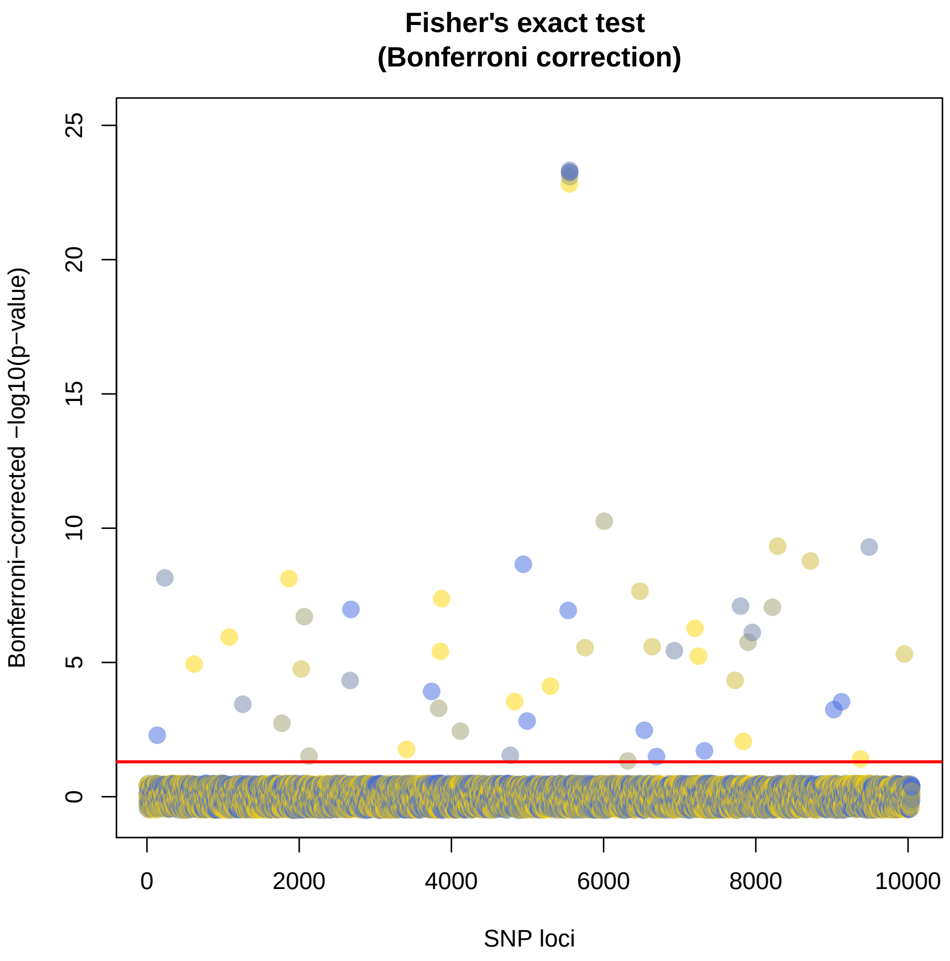
We will first try using the Bonferroni method to correct for multiple testing.

To do this, we can either divide our "significance threshold" p-value (ie. 0.05) by the number of tests (ie. the number of SNP loci), or we can scale up our p-values with the function `p.adjust`:

```
pval.corrected.bonf <- p.adjust(pval, method="bonferroni")
```

To visualise the results from this test, we can use the following commands to generate a Manhattan Plot (the plot most commonly used to represent the results of GWAS analyses).

```
log.pval <- -log10(pval.corrected.bonf)
set.seed(1)
log.pval <- jitter(log.pval, amount=0.5)

plot(log.pval,
     col = transp(azur(5)),
     pch = 19,
     cex = 1.5,
     ylim=c(-0.5, 25),
     main="Fisher's exact test \n(Bonferroni correction)",
     xlab="SNP loci", ylab="Bonferroni-corrected -log10(p-value)")

thresh <- -log10(0.05)
abline(h=thresh, col = "red", lwd=2)
```



Each point in the plot represents a p-value. Note that the colours of the points are not meaningful: this is just a way of distinguishing neighbouring points from one another.

How many SNPs have we identified as significant by the Fisher's exact test with Bonferroni correction?

```
res <- which(pval.corrected.bonf < 0.05)

length(res)
# 53
```

Which loci have been selected?

```
res

## L00135 L00234 L00621 L01081 L01259 L01773 L01866 L02027 L02068 L02128
##    135    234    621   1081   1259   1773   1866   2027   2068   2128
## L02669 L02680 L03411 L03740 L03833 L03856 L03871 L04118 L04774 L04831
##   2669   2680   3411   3740   3833   3856   3871   4118   4774   4831
## L04945 L04995 L05301 L05535 L05551 L05552 L05553 L05554 L05555 L05757
##   4945   4995   5301   5535   5551   5552   5553   5554   5555   5757
## L06008 L06318 L06477 L06535 L06637 L06695 L06929 L07201 L07246 L07325
##   6008   6318   6477   6535   6637   6695   6929   7201   7246   7325
## L07727 L07799 L07836 L07898 L07954 L08218 L08287 L08717 L09025 L09125
##   7727   7799   7836   7898   7954   8218   8287   8717   9025   9125
## L09376 L09489 L09952
##   9376   9489   9952
```

Because the loci in this dataset are named simply `‘‘L0001’’` to `‘‘L10050’’`, we may not want or need to keep the loci names when storing our result. It will be a little easier to examine the list of results if we just store the *indices* of the SNPs selected and set the names of the selected loci to `NULL`.

```
names(res) <- NULL
```

Before moving on to the next test, create a list in which to store the results from the various association tests we will perform as we continue through the practical. Store the result from this analysis in that list and give it an appropriate name so that later in the practical we can compare it the the results from other association tests.

```
sigSNPs <- list()
sigSNPs[[1]] <- list(res)
names(sigSNPs)[[1]] <- "univariate"
names(sigSNPs$univariate)[[1]] <- "bonferroni"
```

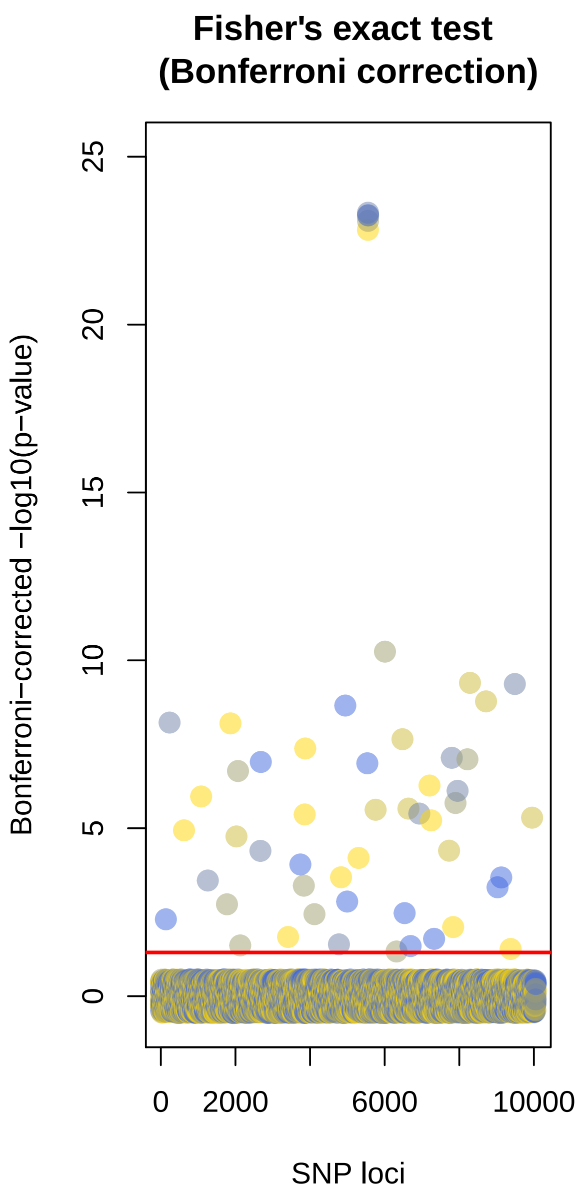
4.1.2 FDR correction

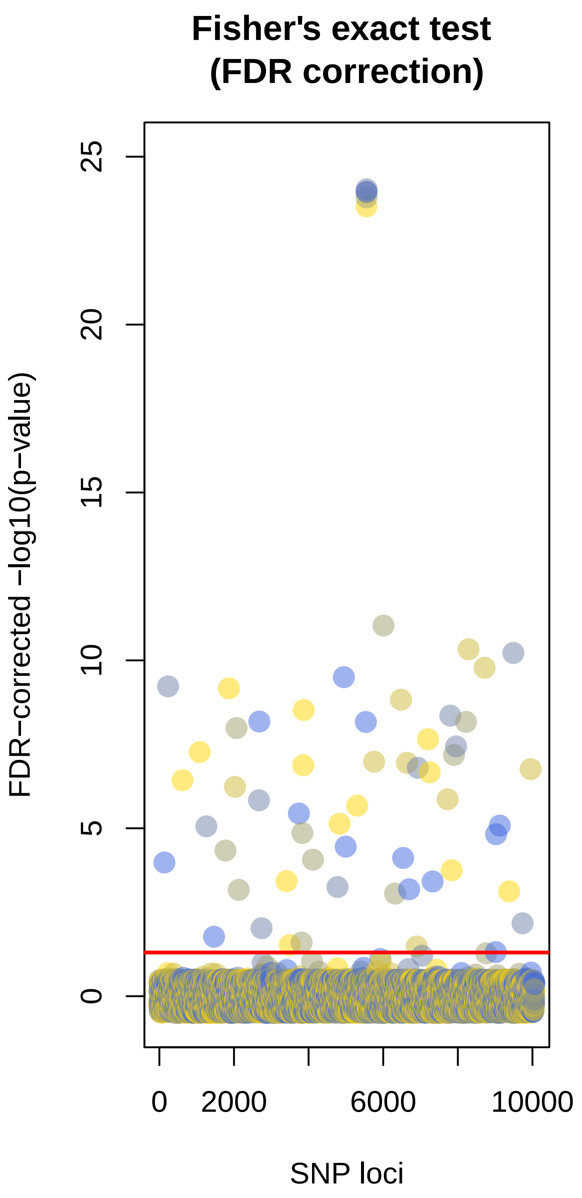
Let’s try to make a prediction before we get the results obtained by the False Discovery Rate correction for multiple testing: Do you think that we will identify more or fewer SNPs as significant associations with the FDR correction, as compared with the number we just found with the Bonferroni correction?

The False Discovery Rate correction for multiple testing is less straightforward than the Bonferroni correction, but it can just as easily be carried out with the function `p.adjust`:

```
pval.corrected.fdr <- p.adjust(pval, method="fdr")
```

Now, working with `pval.corrected.fdr` instead of `pval.corrected.bonf`, make another Manhattan plot to visualise the results from the Fisher’s exact test with FDR correction. For ease of comparison, you may want to use the command `par(mfrow=c(1,2))` to allow for the generation of side-by-side Manhattan plots (type `?par` for more info).





What can you say about the two methods of correcting for multiple testing by comparing these two plots?

How many associated SNPs do we find by using the FDR method of correcting for multiple testing?
Which loci are these?

```
res <- which(pval.corrected.fdr < 0.05)
names(res) <- NULL

length(res)
# 59
```

```
res

##  [1]  135  234  621 1081 1259 1465 1773 1866 2027 2068 2128 2669 2680 2739
## [15] 3411 3496 3740 3813 3833 3856 3871 4118 4774 4831 4945 4995 5301 5535
## [29] 5551 5552 5553 5554 5555 5757 6008 6318 6477 6535 6637 6695 6897 6929
## [43] 7201 7246 7325 7727 7799 7836 7898 7954 8218 8287 8717 9025 9125 9376
## [57] 9489 9734 9952
```

Remember to store your result in `sigSNPs`...

```
sigSNPs$univariate[[2]] <- res
names(sigSNPs$univariate)[[2]] <- "fdr"
```

4.2 Multivariate methods

4.2.1 LASSO

To test for association with the LASSO penalized regression method, we use the function `cv.glmnet` from package `glmnet`.

```
set.seed(1)
LASSO <- cv.glmnet(snps, phen,
                   family="binomial",
                   lambda.min.ratio=0.01, alpha=1)
```

(What does the "cv" in the `cv.glmnet` function stand for? And why have we set `lambda.min.ratio` to 0.01?)

The LASSO method generates coefficients for each variable, though the majority of these will have been shrunk to zero in the penalization step. We extract these coefficients from the model generated by `cv.glmnet` with the function `coef`.

```
beta <- as.vector(t(coef(LASSO, s="lambda.min")))
```

We retrieve the results of the implicit feature selection performed by LASSO by identifying the variables that have non-zero coefficients.

```
res <- which(beta[-1] !=0)

coefs.LASSO <- beta[-1][res]
names(coefs.LASSO) <- colnames(snps)[res]
```

Taking a look at the SNPs selected, we see...

```
length(res)
# 4

res
# 5551 5552 5553 5555
```

Quite the disagreement with the results from the univariate approach.

A standard visualisation tool for the LASSO method is a plot of the fraction of deviance explained versus the values of the coefficients retained in the LASSO regression model.

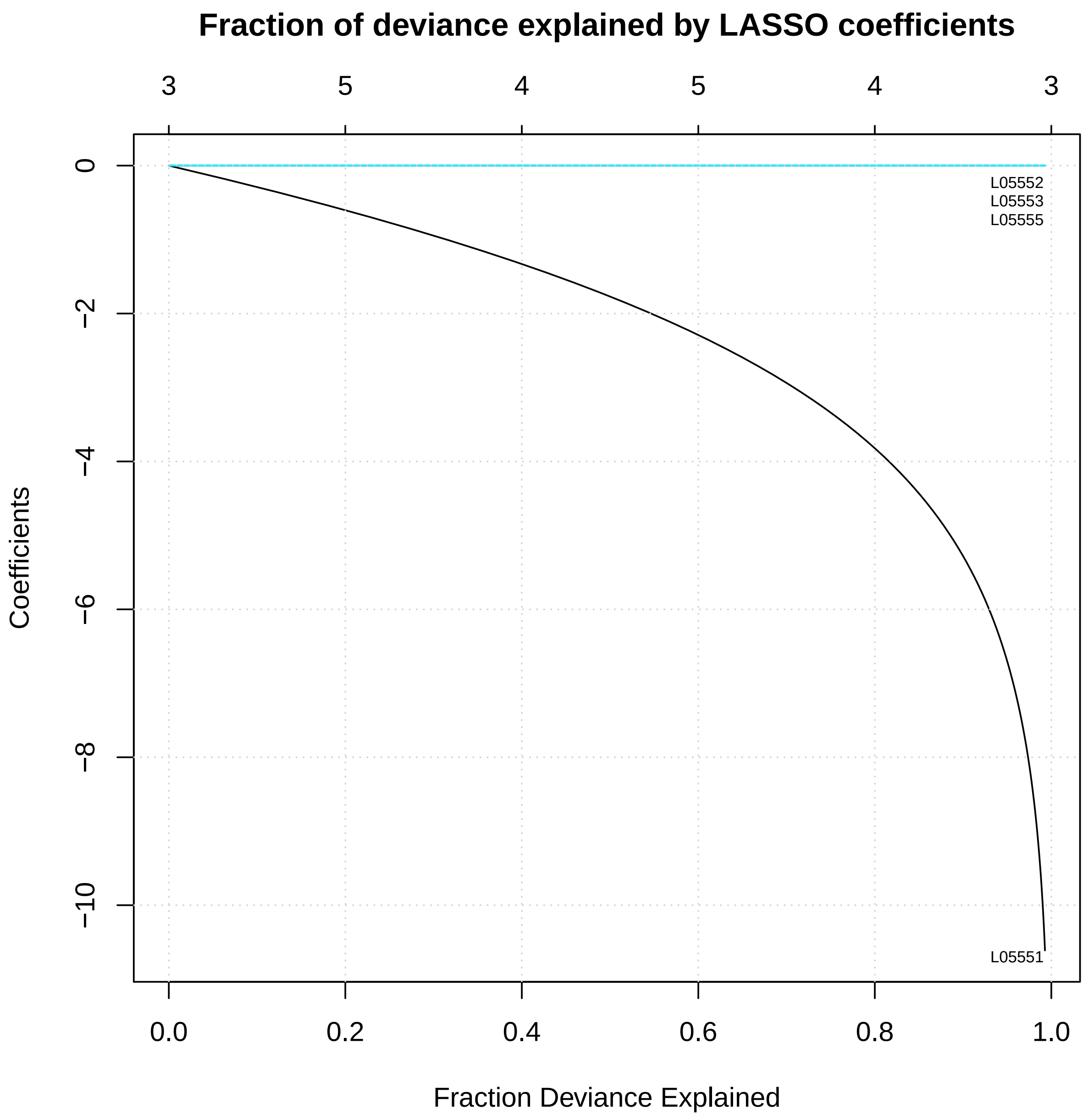
First, get the fitted `glmnet` object from within the results of the LASSO cross-validation:

```r
fit <- LASSO$glmnet.fit
```

Then, plot the deviance explained by the coefficients retained. (Note: So we don't waste too much time fidgeting around with this ugly plot, I've included the standard plotting command, as well as some modified code below this that will allow us to see the labels more clearly for this example.)

```r
## standard code:
plot(fit, xvar = "dev", label = TRUE)
```

```r
## improved code for this example:
y.pos <- coefs.LASSO-c(0.2, 0.25, 0.5, 0.75)

plot(fit, xvar = "dev", label = FALSE)
text(x=1.01, y=y.pos,
     labels=names(coefs.LASSO), col="black", pos=2, cex=0.6)
grid()
title("Fraction of deviance explained by LASSO coefficients", line=3)
```



Still not a very pretty plot, but it does tell us something...

Let's take a closer look at the coefficients assigned to each of the SNPs selected by LASSO.

```
coefs.LASSO
#  5551          5552          5553          5555
# -1.051829e+01 -9.203177e-15 -3.122506e-14 -3.911350e-14
```

Note that (the absolute value of) the coefficient for the first SNP is substantially larger than the coefficients for the other 4 SNPs selected by LASSO, which are very near zero. What is your interpretation of these coefficients? What do they tell you about the relative importance of each of the SNPs selected?

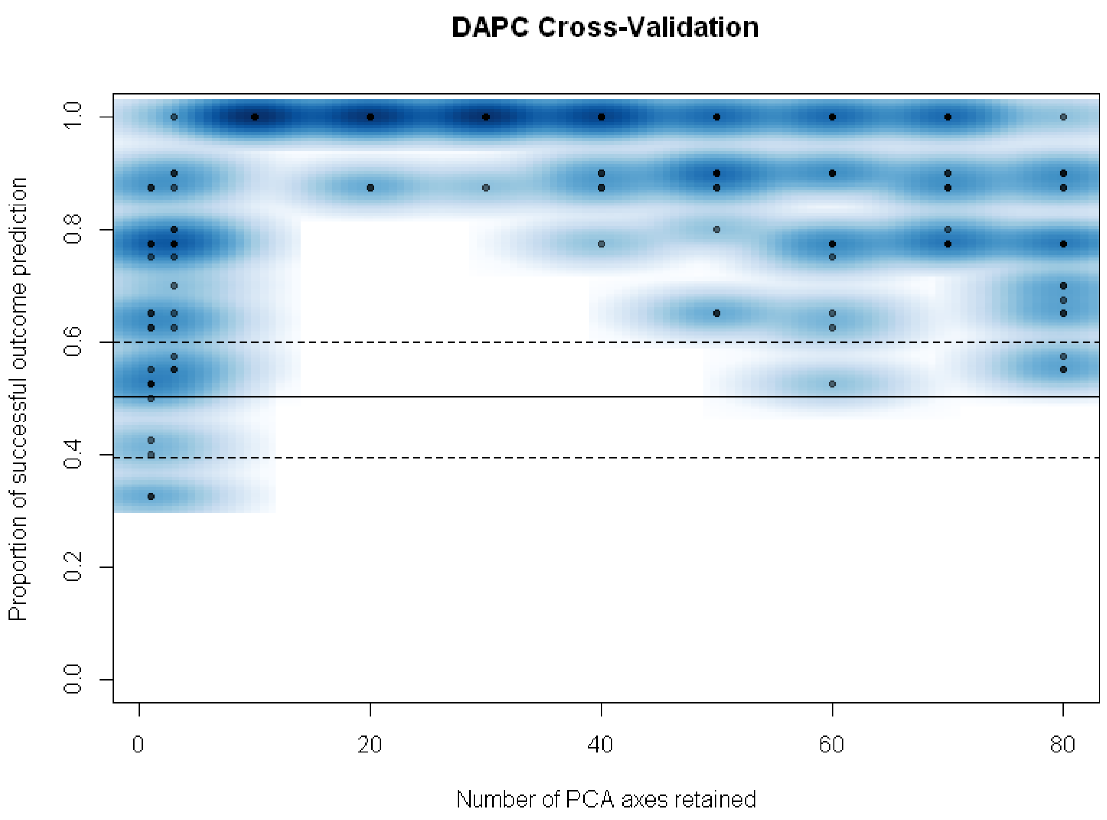
Before we move on, remember to store these results in our `sigSNPs` list:

```
sigSNPs[[2]] <- list(res)
names(sigSNPs)[[2]] <- "multivariate"
names(sigSNPs$multivariate)[[1]] <- "lasso"
```

4.2.2 DAPC-based feature selection

We begin the DAPC approach to feature selection by running cross-validation to help us select the number of PCs of PCA to retain that will maximize our ability to discriminate between our two phenotypic groups.

```
set.seed(1)
xval1 <- xvalDapc(snps, phen,
                  n.pca=c(c(1,3), seq(10, 80, 10)),
                  n.rep=20) # may take a moment...
```



Based on the plot generated by xvalDapc, do you trust that cross-validation has been successful in selecting a model that is useful in assigning individuals to the correct phenotypic group?

Let's take a look at the object `xval1` containing the results of cross-validation:

```
xval1[2:6]
```

```
## $`Median and Confidence Interval for Random Chance`
##      2.5%       50%     97.5%
## 0.3954545 0.5034091 0.6011080
##
## $`Mean Successful Assignment by Number of PCs of PCA`
##       1       3      10      20      30      40      50      60      70
## 0.60000 0.75500 1.00000 0.98750 0.99375 0.96625 0.90750 0.86250 0.89375
##      80
## 0.75750
##
## $`Number of PCs Achieving Highest Mean Success`
## [1] "10"
##
## $`Root Mean Squared Error by Number of PCs of PCA`
##          1          3         10         20         30         40
## 0.43178409 0.27122408 0.00000000 0.03952847 0.02795085 0.07137051
##         50         60         70         80
## 0.13896942 0.19921722 0.14328730 0.27214426
##
## $`Number of PCs Achieving Lowest MSE`
## [1] "10"
```

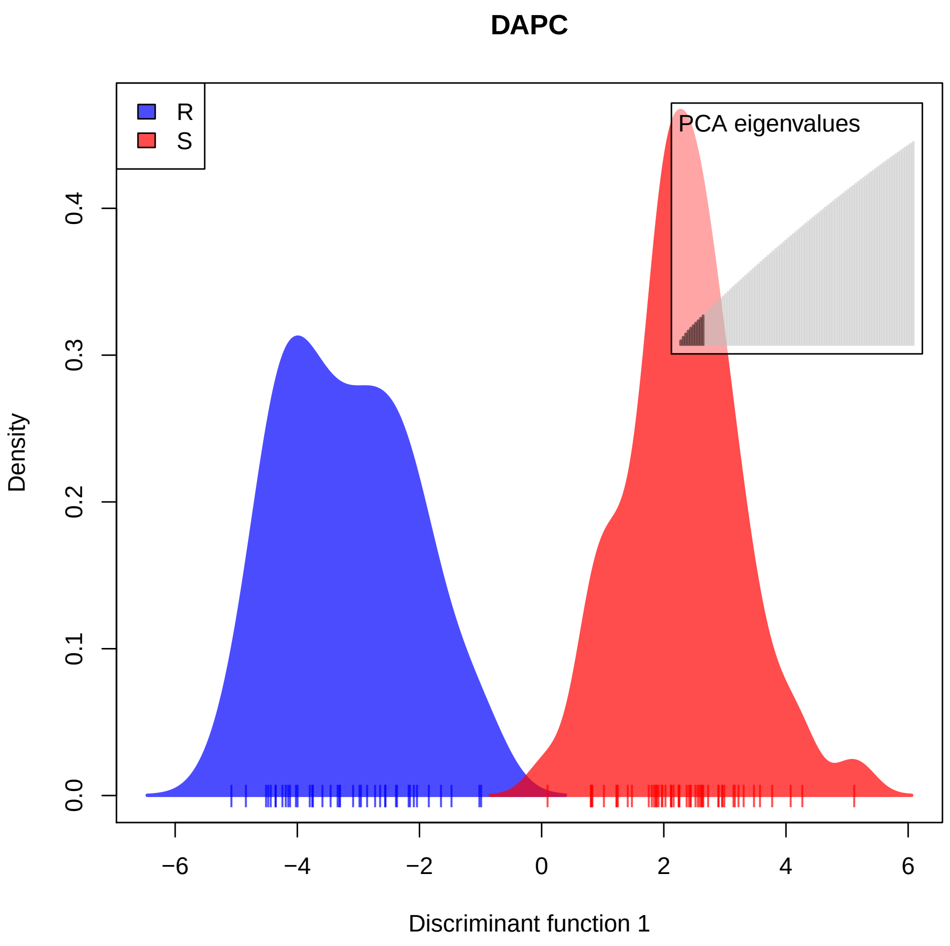
How many PCs of PCA should you retain in your DAPC? For this analysis (and for all Case-Control GWAS analyses), how many DA axes do you need?

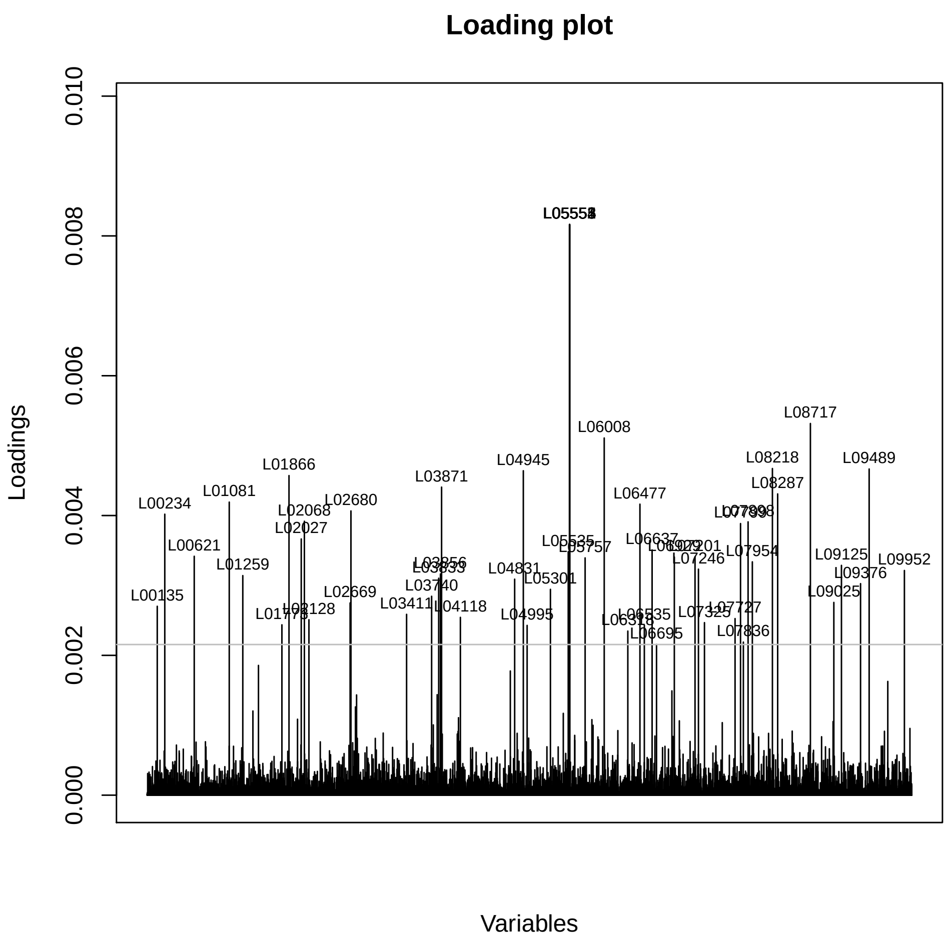
The last element of the output of `xvalDapc` is a `dapc` object generated with the optimal number of PCs, as indicated by RMSE. Store this in an object called `dapc1`.

```
dapc1 <- xval1[[7]]
```

We can now use the function `snpzip` to perform feature selection and visualise our results. (NOTE: Due to recent updates in the `hclust` package, the name being used to indicate Ward's minimum variance method has been changed. `R` will complain about this, but it is safe to ignore these warnings.)

```
result <- snpzip(snps, dapc1,
                method="ward", xval.plot = FALSE,
                plot = TRUE, loading.plot = TRUE)

## The "ward" method has been renamed to "ward.D"; note new "ward.D2"
```


DAPC
R
S
PCA eigenvalues
Density
0.0
0.1
0.2
0.3
0.4
−6
−4
−2
0
2
4
6
Discriminant function 1



```r
par(ask=FALSE) # allow plots to display without hitting ENTER
```

Looking at the plot entitled “DAPC”, would you say that the DAPC approach has been successful in distinguishing our two phenotypic groups along the principal axis?

We can check this explicitly by looking at the `summary` of `dapc1`:

```r
summary(dapc1)

## $n.dim
## [1] 1
##
## $n.pop
## [1] 2
##
## $assign.prop
## [1] 1
##
```

```
## $assign.per.pop
## R S
## 1 1
##
## $prior.grp.size
##
##  R  S
## 40 55
##
## $post.grp.size
##
##  R  S
## 40 55
```

Thinking back to the `s.class` plot we made earlier to visualise the phenotypic groups in PCA space, would you say that DAPC has been more or less effective in separating these two phenotypic groups than PCA (along any axis or axes)? Why?

Let's take a look at the output of `snpzip`:

```
result

## $`Number of PCs of PCA retained`
## [1] 10
##
## $FS
## $FS$`Number of selected vs. unselected alleles`
## [1]   52 9998
##
## $FS$`List of selected alleles`
##  [1]  135  234  621 1081 1259 1773 1866 2027 2068 2128 2669 2680 3411 3740
## [15] 3833 3856 3871 4118 4831 4945 4995 5301 5535 5551 5552 5553 5554 5555
## [29] 5757 6008 6318 6477 6535 6637 6695 6929 7201 7246 7325 7727 7799 7836
## [43] 7898 7954 8218 8287 8717 9025 9125 9376 9489 9952
##
## $FS$`Names of selected alleles`
##  [1] "L00135" "L00234" "L00621" "L01081" "L01259" "L01773" "L01866"
##  [8] "L02027" "L02068" "L02128" "L02669" "L02680" "L03411" "L03740"
## [15] "L03833" "L03856" "L03871" "L04118" "L04831" "L04945" "L04995"
## [22] "L05301" "L05535" "L05551" "L05552" "L05553" "L05554" "L05555"
## [29] "L05757" "L06008" "L06318" "L06477" "L06535" "L06637" "L06695"
## [36] "L06929" "L07201" "L07246" "L07325" "L07727" "L07799" "L07836"
## [43] "L07898" "L07954" "L08218" "L08287" "L08717" "L09025" "L09125"
## [50] "L09376" "L09489" "L09952"
##
```

```
## $FS$`Contributions of selected alleles to discriminant axis`
##      L00135      L00234      L00621      L01081      L01259      L01773
## 0.002702390 0.004019747 0.003417851 0.004192544 0.003141309 0.002436959
##      L01866      L02027      L02068      L02128      L02669      L02680
## 0.004571981 0.003665454 0.003918528 0.002509789 0.002749745 0.004066029
##      L03411      L03740      L03833      L03856      L03871      L04118
## 0.002587192 0.002845342 0.003103492 0.003164939 0.004406861 0.002543660
##      L04831      L04945      L04995      L05301      L05535      L05551
## 0.003089871 0.004641114 0.002428758 0.002944800 0.003482731 0.008163206
##      L05552      L05553      L05554      L05555      L05757      L06008
## 0.008163206 0.008163206 0.008163206 0.008163206 0.003394389 0.005109407
##      L06318      L06477      L06535      L06637      L06695      L06929
## 0.002347548 0.004162878 0.002432528 0.003507859 0.002155828 0.003406501
##      L07201      L07246      L07325      L07727      L07799      L07836
## 0.003411794 0.003233062 0.002468220 0.002526155 0.003885590 0.002189902
##      L07898      L07954      L08218      L08287      L08717      L09025
## 0.003910026 0.003337914 0.004672914 0.004309669 0.005317029 0.002757332
##      L09125      L09376      L09489      L09952
## 0.003287357 0.003026235 0.004666147 0.003214336
```

With a result much more similar to that given by the Fisher's exact test than to the result given by LASSO, we have selected 52 SNPs as significant associations with DAPC-based feature selection using Ward's minimum variance clustering method.

Isolate the element of `result` containing the selected alleles and store it in our `sigSNPs` list:

```r
res <- result$FS[[2]]

sigSNPs$multivariate[[2]] <- res
names(sigSNPs$multivariate)[[2]] <- "dapc"
```

5 Correcting for population stratification with PCA

The objective in correcting for population stratification is to remove the *between*-group variation separating individuals in the study sample into subpopulation clusters.

Recall that earlier we identified 5 major clusters:

```
table(pop)

## pop
##  1  2  3  4  5
## 15 15 20 30 15
```

PCA is the most common method of correcting for population stratification in GWAS.

As we have already run a PCA to visualise the population structure in our first assessment of genetic diversity, we have already generated the object `pca1` that we will use to correct our SNPs matrix for population stratification. We do this by regressing along the significant axes of PCA. In our case, this means we will regress along the first 5 PCs of PCA.

```
snps.corrected <- apply(snps, 2, function(e)
  residuals(lm(e~pca1$li[,1]+pca1$li[,2]+pca1$li[,3]+
                 pca1$li[,4]+pca1$li[,5]))) # may take a minute
```

Let's inspect our corrected SNPs matrix:

```
dim(snps.corrected)
# 95 10050
range(snps.corrected)
# -1.137694  1.279894
```

```
snps.corrected[1:10,1:5]

##                L00001       L00002     L00003       L00004      L00005
## isolate-1  -0.6239781  0.079521595  0.2102712 -0.007840968 -0.05342640
## isolate-2   0.3146597  0.112919596  0.1960766 -0.013179369 -0.08662247
## isolate-3   0.3540695  0.113155207  0.2004474 -0.011457889 -0.06703873
## isolate-4   0.3796632  0.096568617 -0.7979056 -0.005859113 -0.05711164
## isolate-5   0.3330204  0.102025735  0.1998281 -0.009510705 -0.07723301
## isolate-6  -0.5795854 -0.006171606  0.2067841  0.018053703 -0.02762617
## isolate-7  -0.6244806 -0.004455539  0.2144205  0.012633779 -0.05202696
## isolate-8   0.3477955  0.089695614 -0.8056878 -0.007025073 -0.08183554
## isolate-9   0.3507693 -0.884919196 -0.7995953 -0.007655277 -0.07580913
## isolate-10 -0.6442575  0.096202834  0.2011892 -0.006948981  0.93407128
```

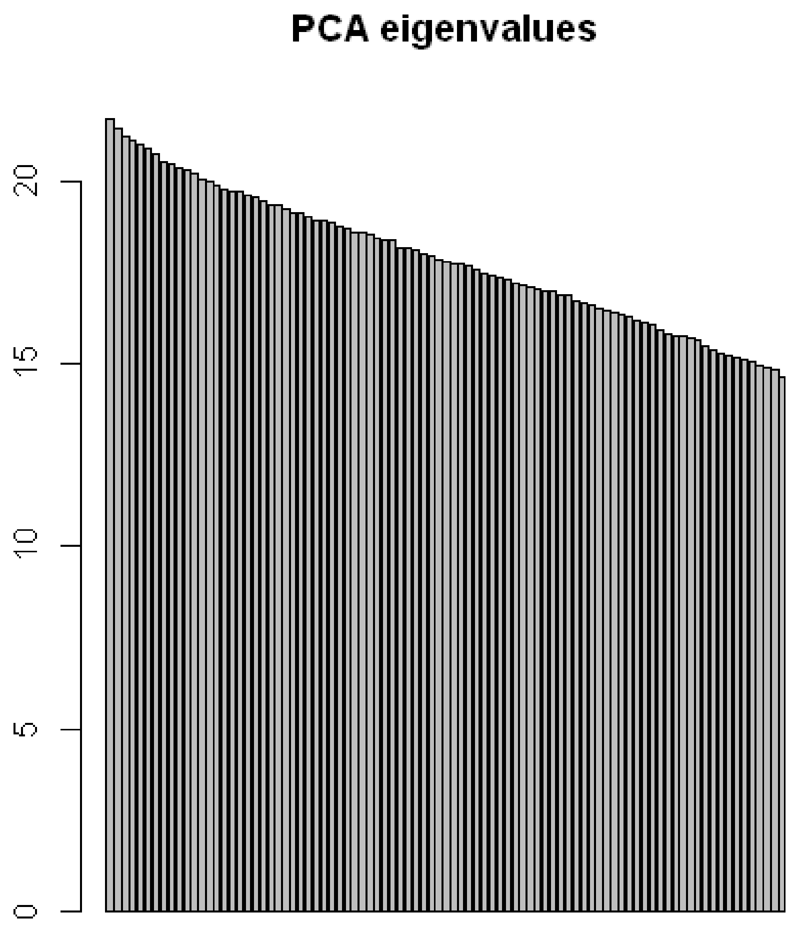
What kind of variable does our corrected SNPs matrix contain?

To visually assess whether our correction for population stratification has been successful, we can run a second PCA analysis, this time with the *corrected* SNPs matrix:

```
pca2 <- dudi.pca(snps.corrected, scale=FALSE, scannf=FALSE, nf=5)
```

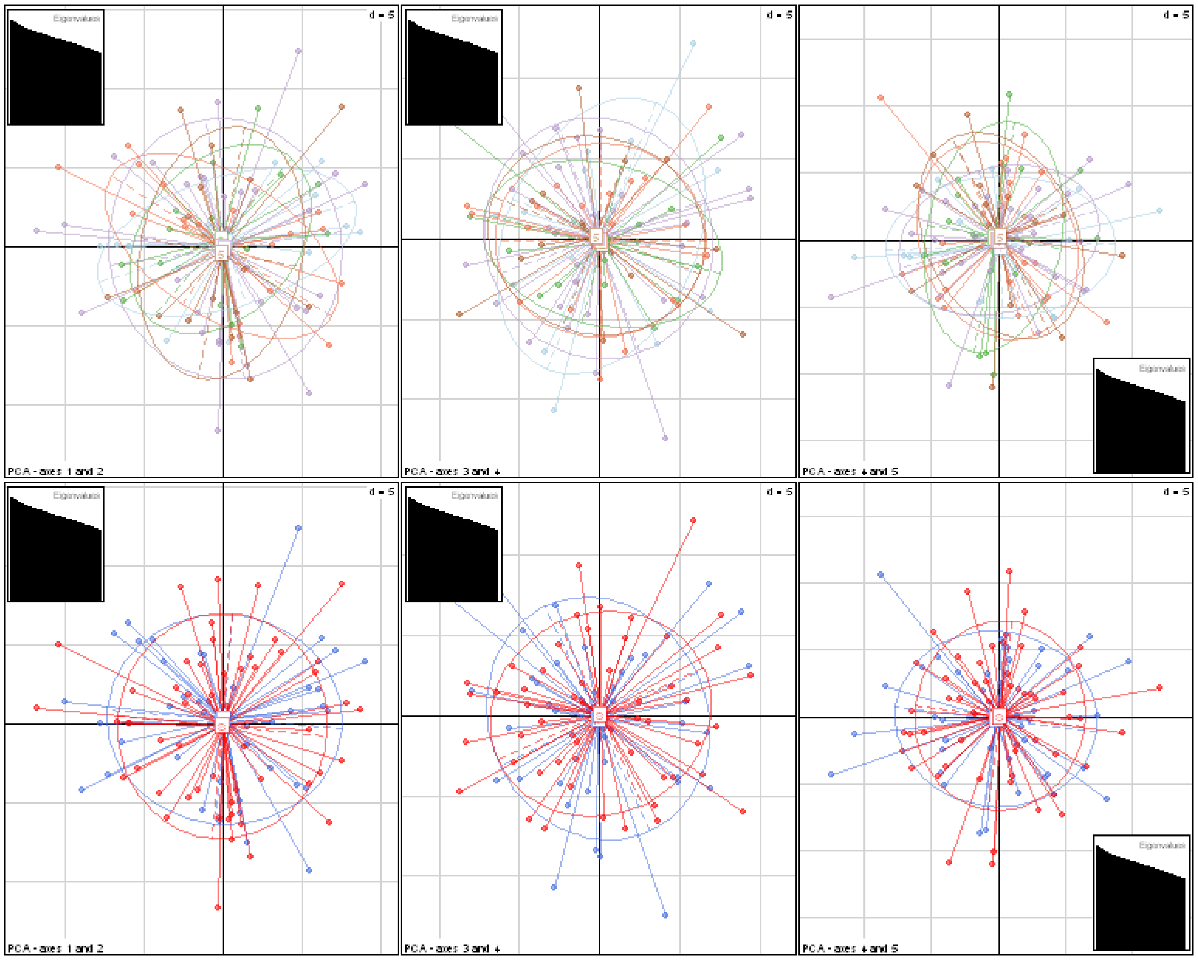
First, we can take a look at the eigenvalues for `pca2`.

```
barplot(pca2$eig, main="PCA eigenvalues")
```



What do you notice about these eigenvalues? What can you infer from this?

Using `s.class` and `par(mfrow)`, let's examine the structure among our subpopulation clusters and phenotypic groups along the first 5 axes of PCA:



Have we removed the variation separating our subpopulation clusters? In other words, do you think we have been successful in correcting for population stratification?

What has happened to the variation separating our phenotypic groups? What impact do you predict this will have on the results of GWAS analyses run after correcting for population stratification via PCA (in this example)?

Let's go and see for ourselves...

6 GWAS after correcting with PCA

6.1 Univariate method

While both multivariate methods for association testing and feature selection will be directly repeatable in application to our newly corrected SNPs matrix, our univariate approach is no longer valid!

Why do you think the Fisher's exact test is no longer appropriate here?

Instead of Fisher's exact test we will use an alternative univariate approach that consists of two stages. First, we generate a simple linear model between each column of our corrected SNPs matrix and our phenotypic trait. Second, we run an analysis of variance (ANOVA) on each model generated, specifying a Chi-squared test of association. From this,we can retrieve a p-value for the significance of association between each corrected SNP and the resistance phenotype.

```r
pval2 <- numeric(0)
for(i in 1:ncol(snps.corrected)){
  foo <- suppressWarnings(glm(phen ~ snps.corrected[,i], family="binomial"))
  ANOVA <- anova(foo, test="Chisq")
  pval2[i] <- ANOVA$"Pr(>Chi)"[2]
} # end for loop
```

Take a look at the smallest p-value, and determine the number of p-values that are significant at alpha=0.05.

```r
min(pval2, na.rm=TRUE)

## [1] 0.0005274102

# 0.0005274102

length(which(pval2 < 0.05))

## [1] 5

# 5
```

Oh dear. We are barely achieving significance at any locus and we have not yet corrected for multiple testing!
We will now correct for multiple testing as we did in our initial univariate analysis, using the Bonferroni correction and False Discovery Rate. We can use the same code to do this as before we corrected for population stratification.

6.1.1 Bonferroni correction

```r
pval.corrected.bonf <- p.adjust(pval2, method="bonferroni")
res <- which(pval.corrected.bonf < 0.05)
```

As you may have predicted, there are no SNPs above the threshold of significance:

```r
length(res)

## [1] 0
```

Nevertheless, we will make a new list in which to store the results of our association tests performed after correcting for population stratification with PCA...

```r
sigSNPs.PCA <- list()
```

And store our results with the same organisation as before:

```r
sigSNPs.PCA[[1]] <- list(res)
names(sigSNPs.PCA)[[1]] <- "univariate"
names(sigSNPs.PCA$univariate)[[1]] <- "bonferroni"
```

6.1.2 FDR correction

We do the same for the FDR correction.

```r
pval.corrected.fdr <- p.adjust(pval2, method="fdr")
res <- which(pval.corrected.fdr < 0.05)
```

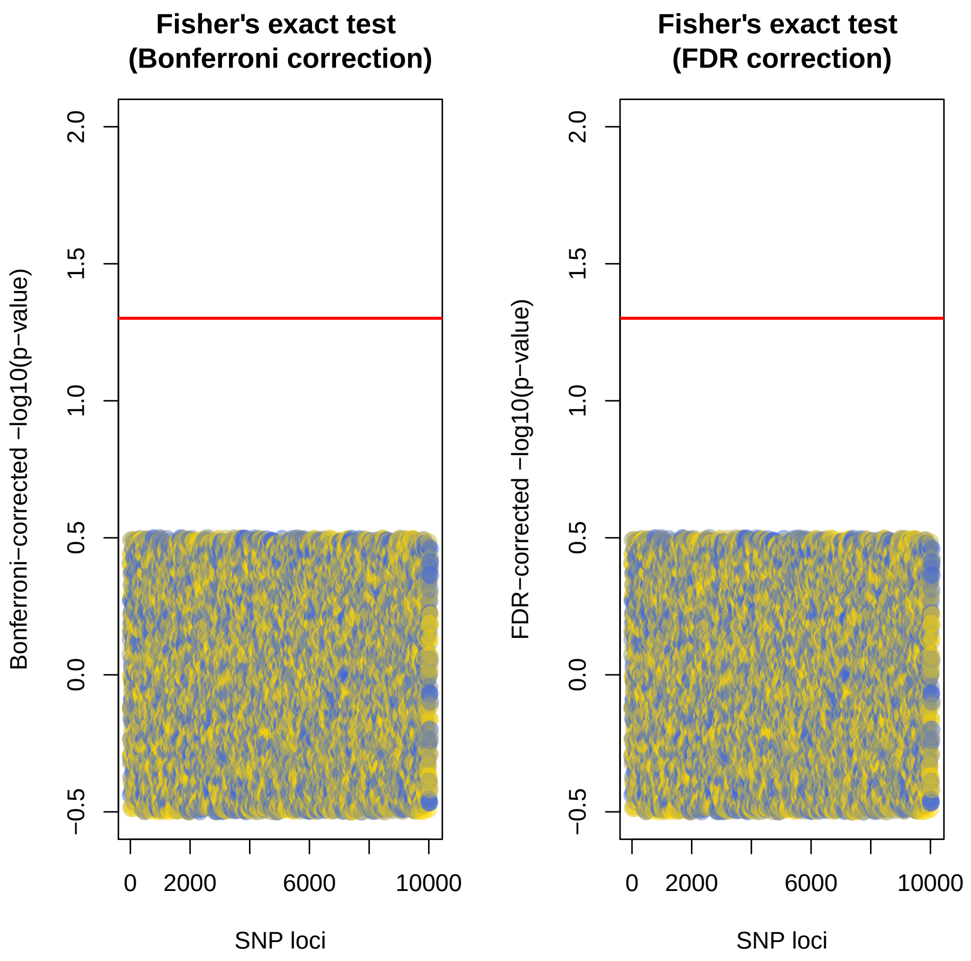
Even this less conservative approach does not have the power to detect any associations in our dataset following the correction for population stratification via regression along the significant axes of PCA.

```r
length(res)

## [1] 0
```

Again, while we have found no significant SNPs, we still store the result in `sigSNPs.PCA`.

```r
sigSNPs.PCA$univariate[[2]] <- res
names(sigSNPs.PCA$univariate)[[2]] <- "fdr"
```

Despite our lack of findings with either method of correcting for multiple testing, we can still compare the the Bonferroni- and FDR-corrected p-values with side-by-side Manhattan plots:



Looking at these plots, it is clear that, regardless of the method of correcting for multiple testing, we had no chance of finding any significant associations with the univariate approach.

6.2 Multivariate methods

6.2.1 LASSO

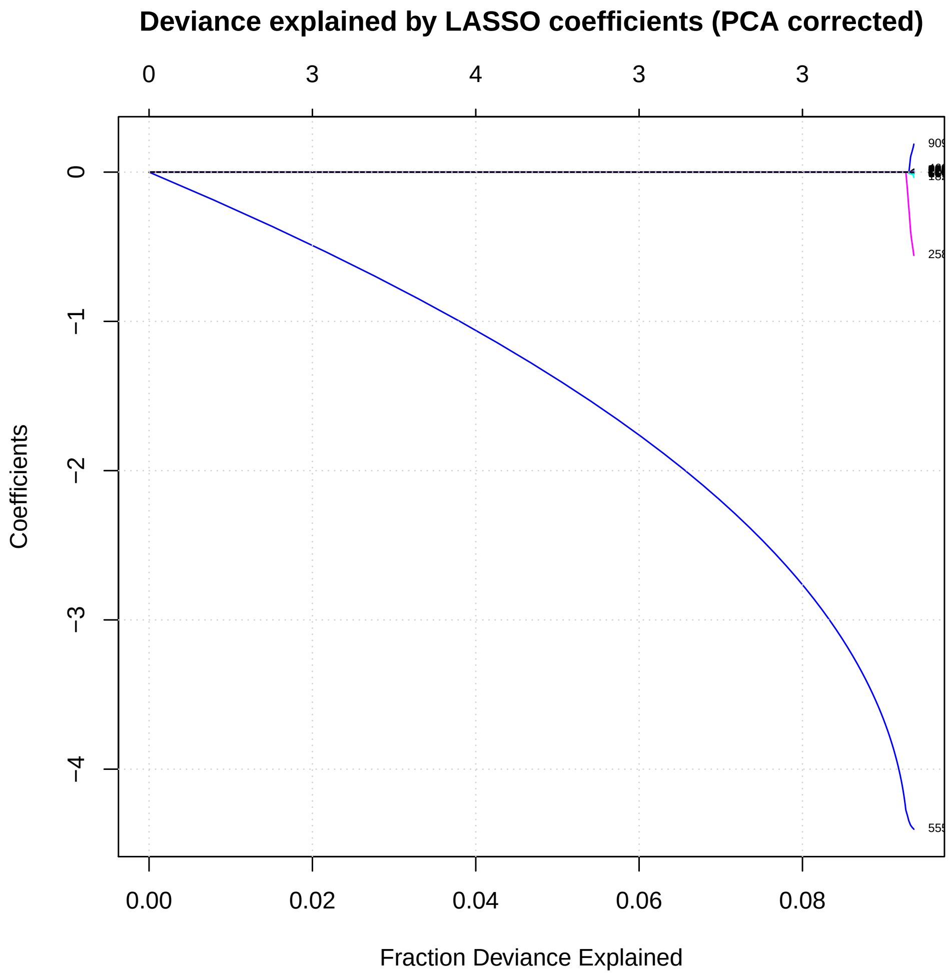
The LASSO method can be performed using the exact same code as before (replacing `snps` with `snps.corrected`).

```
fit <- LASSO$glmnet.fit
```

Then, plot the deviance explained by the shrinking coefficients.

```
## standard code:
plot(fit, xvar = "dev", label = TRUE)
```

```r
grid()
title("Deviance explained by LASSO coefficients (PCA corrected)", line=3)
```



So, does LASSO do any better than the Fisher's exact test?

```r
res <- which(beta[-1] !=0)
```

```r
length(res)
# 0
```

Sadly no. While we can see in the plot that SNP 5551 (which was selected as significant by LASSO before we corrected for population stratification) has the largest non-zero coefficient (absolute value), LASSO is unable to build a model that explains a sufficient fraction of the deviance in the phenotype to deem any of its coefficients "significant".

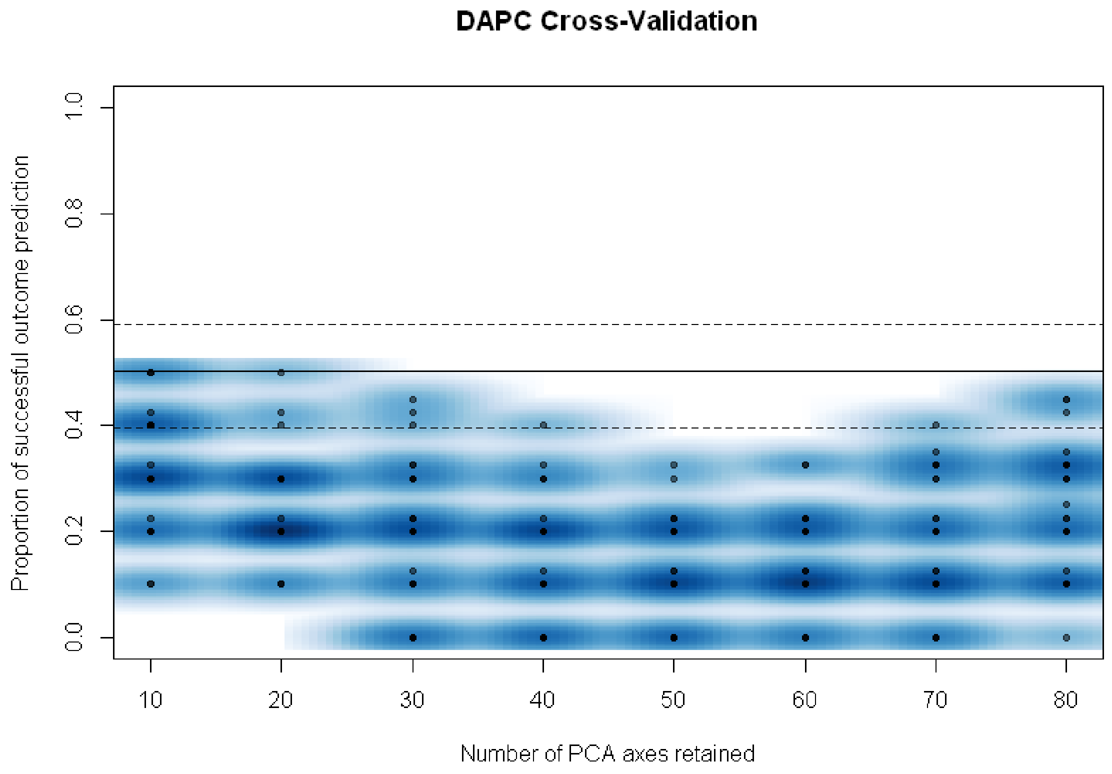
```r
sigSNPs.PCA[[2]] <- list(res)
names(sigSNPs.PCA)[[2]] <- "multivariate"
names(sigSNPs.PCA$multivariate) <- "lasso"
```

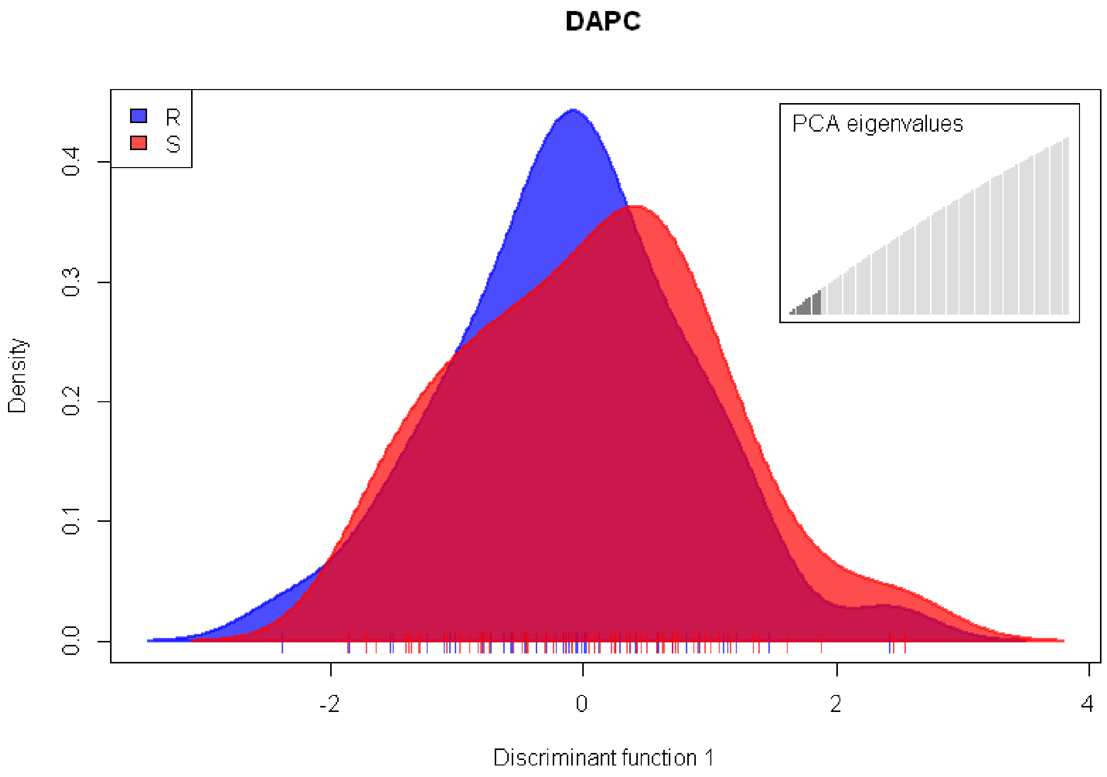
6.2.2 DAPC-based feature selection

Now we try with the remaining method: the DAPC-based feature selection approach.

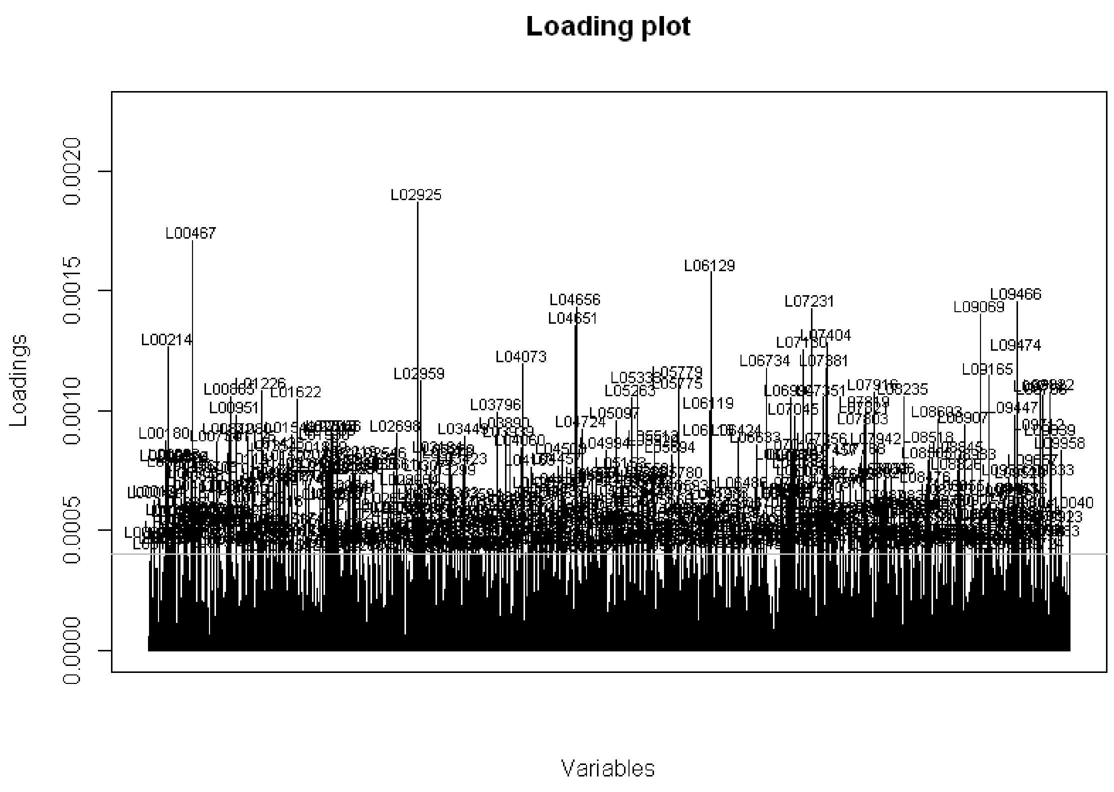
Previously, we ran cross-validation and then set the first two arguments of the function `snpzip` to be `snps` and `dapc1`. Instead of providing `snpzip` with a SNPs matrix and a `dapc` object, we could have simply input a SNPs matrix and a group factor (in the case of GWAS, this would be *phen*). If `snpzip` sees that its second argument is a group factor rather than a `dapc` object, it will automatically perform cross-validation with `xvalDapc`.

```r
result <- snpzip(snps.corrected, phen,
                 xval.plot=TRUE, plot=TRUE, loading.plot=TRUE,
                 method="ward")
```





DAPC
R
S
PCA eigenvalues
Density
0.0
0.1
0.2
0.3
0.4
-2
0
2
4
Discriminant function 1



Notice that while we get *no* separation between groups, and our cross-validation consistently fails to accurately predict the group membership of individuals, `snpzip` will *still* find some SNPs (in this case, *loads* of SNPs!), because there will always be a set of more-significantly-contributing variables to the discriminant axis.

In fact, in this case, because there is not an obvious set of SNPs whose loadings stand out strongly from the rest, the "ward" clustering method selects an extremely large number of SNPs.

```
res <- result$FS[[2]]
```

```
length(res)
# 519
```

```
sigSNPs.PCA[[2]][[2]] <- res
names(sigSNPs.PCA$multivariate)[[2]] <- "dapc"
```

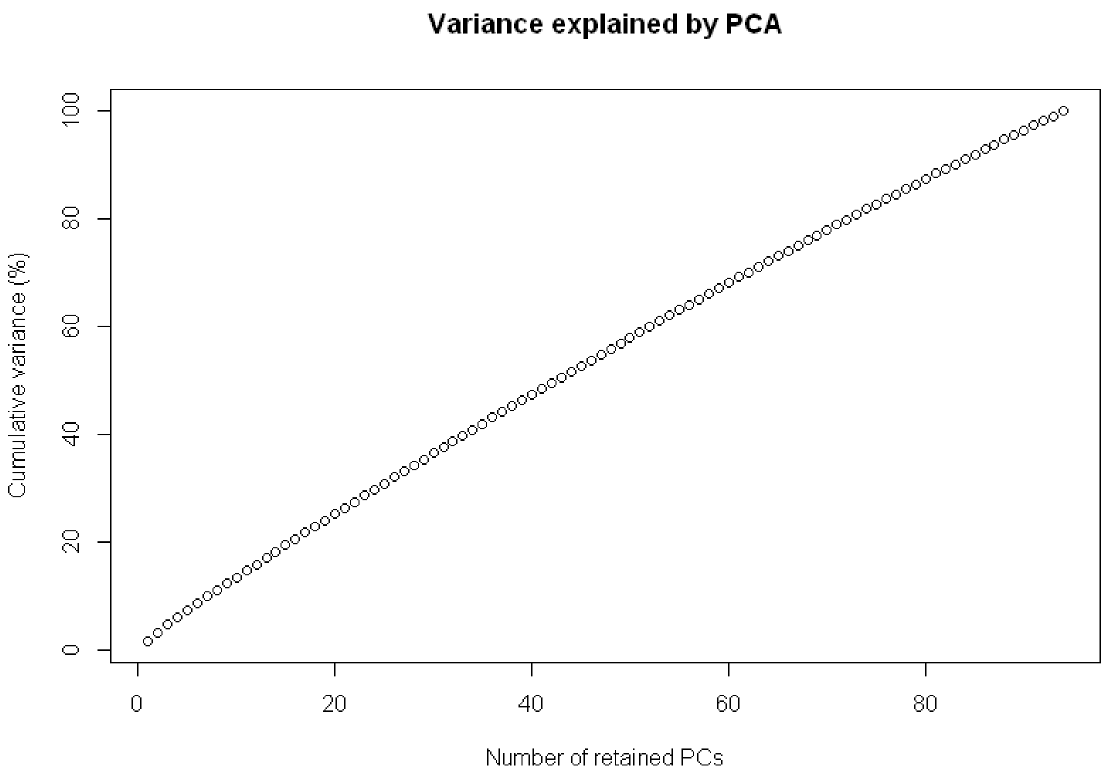
Given our abysmal lack of success with GWAS after correcting for population stratification using the significant axes of PCA, we can try again with a different approach.

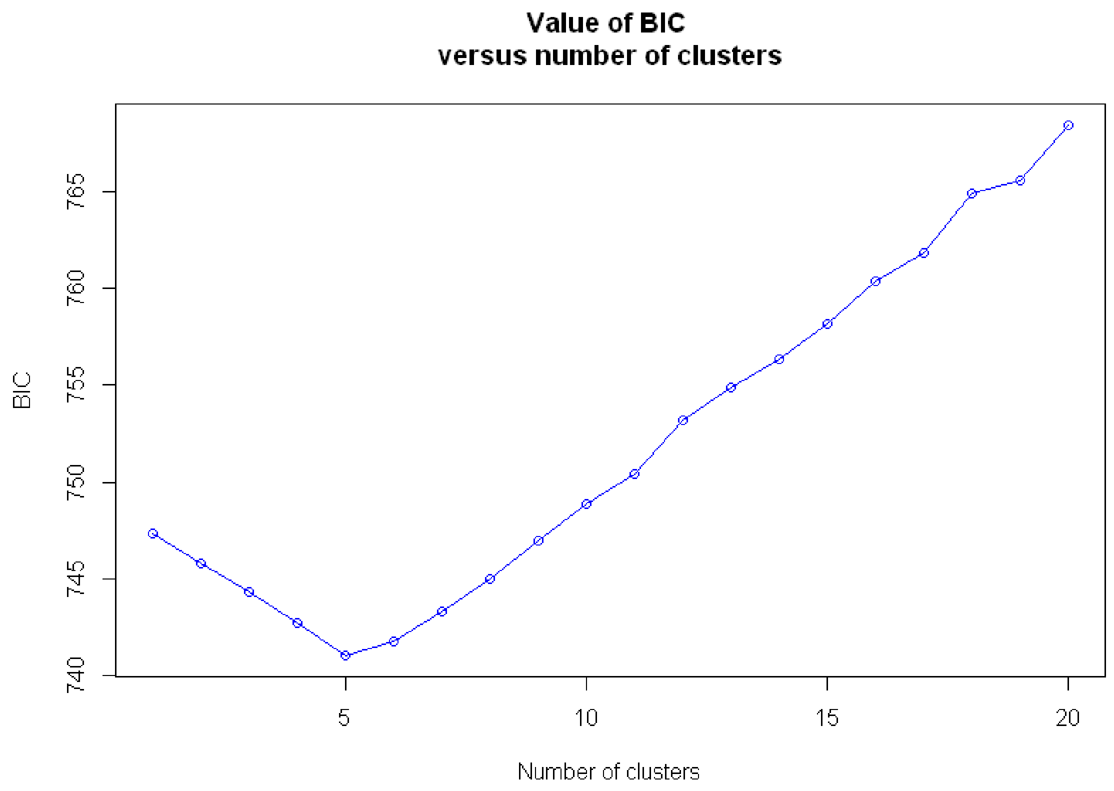
7 Correcting for population stratification with DAPC

Using DAPC to correct for population stratification in GWAS, while less common in the GWAS literature, is strongly supported by the following theoretical justification: Because, in correcting for population stratification, we are trying to remove the population structure that is separating the individuals in our study sample into subpopulation groups, it makes sense to use a method that focuses only on the *between*-group structure.

While we have already identified that there are 5 major subpopulation clusters represented in our dataset (visible in both PCA space and in our complete-linkage hierarchical clustering phylogenetic tree), we can also check with `find.clusters` to see if 5 is an appropriate value of "k" for this dataset.
Note: Because the data we are working with in this practical is a simulated dataset, variance accumulates nearly linearly with increasing numbers of PCs. This means that there is no obvious number of PCs at which a plateau in the cumulative variance is reached. When asked below to choose the number of PCs to retain, 20 will be enough to see an elbow in the BIC plot to follow.

```
grp <- find.clusters(snps, max.n.clust=20)
## Choose the number of PCs to retain:
## 20

## Choose the number of clusters:
## 5
```


Variance explained by PCA
Cumulative variance (%)
Number of retained PCs
0
20
40
60
80
100



Once again, we have support for 5 clusters.

We can use `table.value` to check that the suggested 5 groups correspond to the pops we originally identified with `cutree`.
Looks good!

Using our `pop` clusters as the group factor in DAPC, we can generate a new DAPC object after performing cross-validation to optimise the discrimination between these subpopulations:

```
xval.pop <- xvalDapc(snps, pop)
```

```
xval.pop[2:6]

## $`Median and Confidence Interval for Random Chance`
##      2.5%       50%     97.5%
## 0.1215833 0.2000000 0.2817500
##
## $`Mean Successful Assignment by Number of PCs of PCA`
##        10        20        30        40        50        60        70
## 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 0.9755556
##        80
## 0.7244444
##
## $`Number of PCs Achieving Highest Mean Success`
## [1] "10"
##
## $`Root Mean Squared Error by Number of PCs of PCA`
##         10          20          30          40          50          60
## 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
##         70          80
## 0.06554614 0.34221501
##
## $`Number of PCs Achieving Lowest MSE`
## [1] "10"

dapc.pop <- xval.pop[[7]]
```

As we did when correcting with PCA, we regress along the axes of DAPC. When correcting with the DAPC approach, we do not need to determine how many axes are "significant": we will always correct with (k - 1) axes.

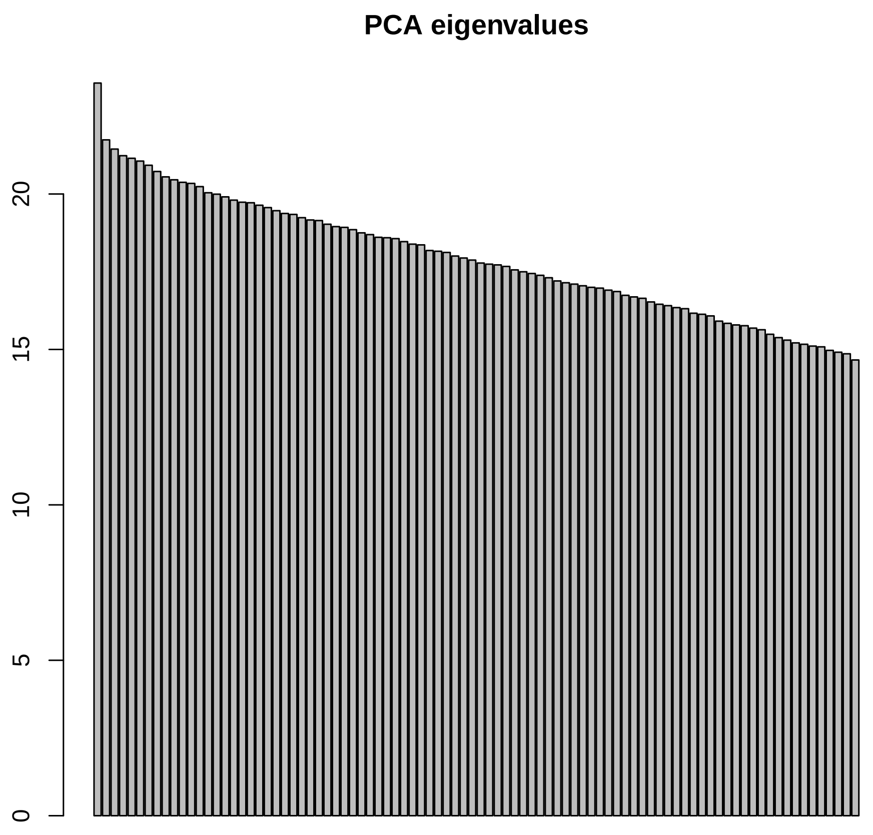
```
snps.corrected <- apply(snps, 2, function(e)
  residuals(lm(e~dapc.pop$ind.coord[,1]+
                 dapc.pop$ind.coord[,2]+
                 dapc.pop$ind.coord[,3]+
                 dapc.pop$ind.coord[,4])))
```

As we did in the previous section, we use PCA as a visual diagnostic tool to check whether our correction for population stratification has been successful.

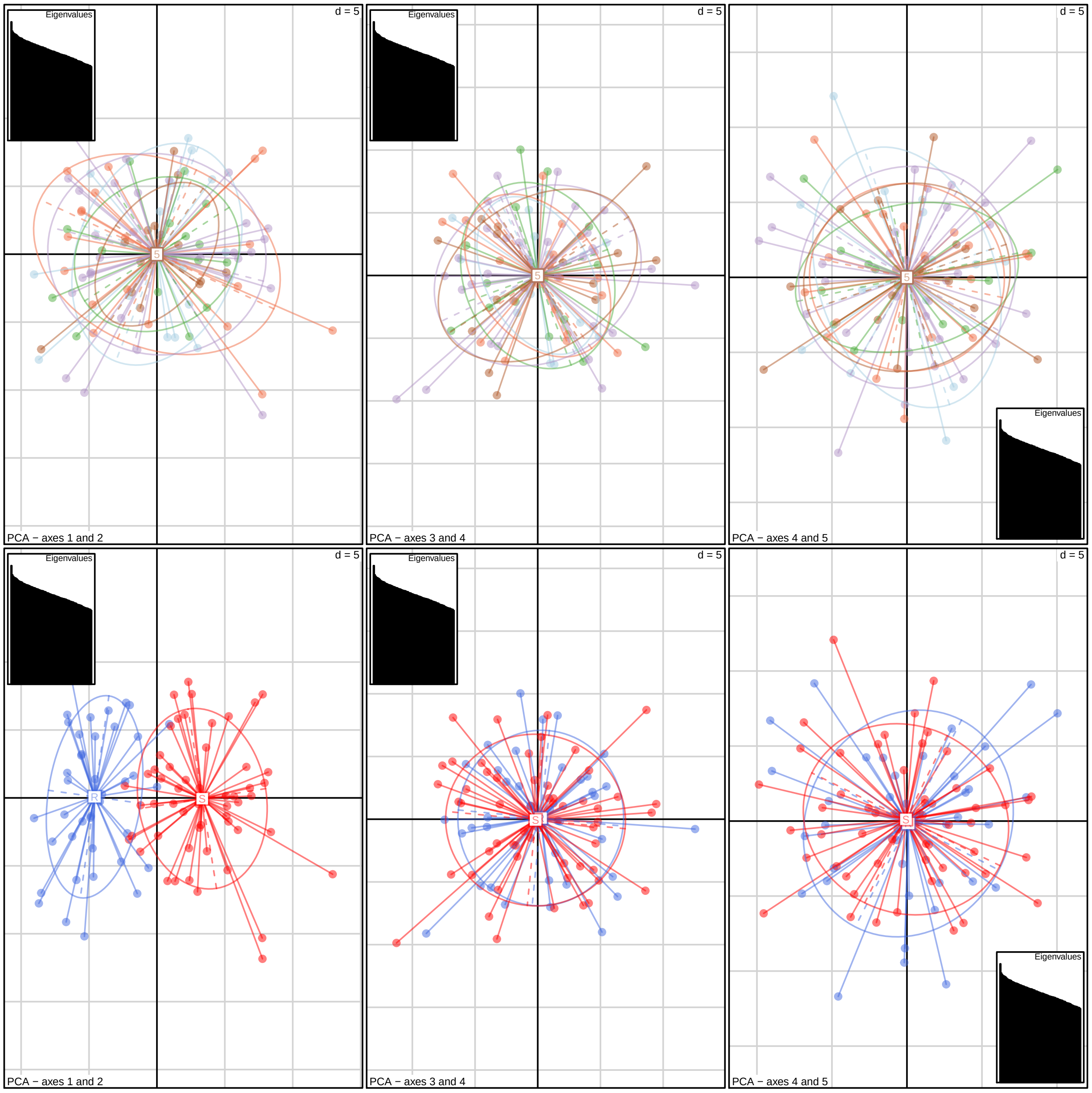
```
pca2 <- dudi.pca(snps.corrected, scale=FALSE)
```

Looking at the eigenvalues for `pca2`, what do you notice?

```
barplot(pca2$eig, main="PCA eigenvalues")
```



As you did before, use `s.class` to examine the population and phenotypic structure present along the first 5 axes of PCA.



8 GWAS after correcting with DAPC

Repeat the code above to get the results of GWAS with each method, and store these in another list, `sigSNPs.DAPC`.

8.1 Univariate method

```
pval2 <- numeric(0)
for(i in 1:ncol(snps.corrected)){
  foo <- suppressWarnings(glm(phen ~ snps.corrected[,i], family="binomial"))
  ANOVA <- anova(foo, test="Chisq")
  pval2[i] <- ANOVA$"Pr(>Chi)"[2]
} # end for loop
```

Take a look at the smallest p-value, and determine the number of p-values that are significant at alpha=0.05.

```
min(pval2, na.rm=TRUE)

## [1] 5.773208e-30

# 5.773208e-30

length(which(pval2 < 0.05))

## [1] 405

# 405
```

Excellent, we have nearly retained all of the power from our initial GWAS!
Now correct for multiple testing using the Bonferroni correction and False Discovery Rate. (We can use the same code to do this as before.)

8.1.1 Bonferroni correction

```
pval.corrected.bonf <- p.adjust(pval2, method="bonferroni")
res <- which(pval.corrected.bonf < 0.05)
```

We now have 43 SNPs above the significance threshold!

```
length(res)
# 43
```

We'll make a new list for this round of association tests:

```
sigSNPs.DAPC <- list()
```

And store our results with the same organisation as before:

```
sigSNPs.DAPC[[1]] <- list(res)
names(sigSNPs.DAPC)[[1]] <- "univariate"
names(sigSNPs.DAPC$univariate)[[1]] <- "bonferroni"
```

8.1.2 FDR correction

We do the same for the FDR correction.

```r
pval.corrected.fdr <- p.adjust(pval2, method="fdr")
res <- which(pval.corrected.fdr < 0.05)
```

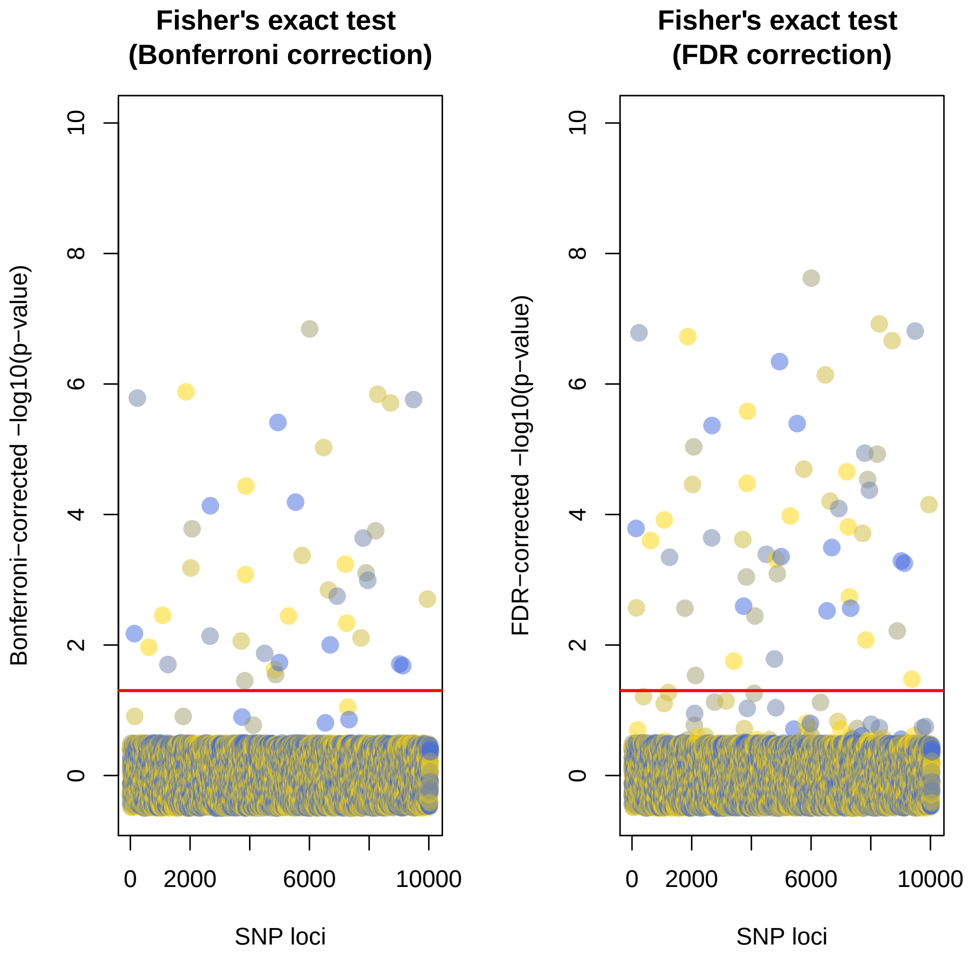
Naturally, we have found more significant SNPs with the FDR correction.

```r
length(res)
# 59
```

We store the result in `sigSNPs.DAPC`.

```r
sigSNPs.DAPC$univariate[[2]] <- res
names(sigSNPs.DAPC$univariate)[[2]] <- "fdr"
```

And we compare the the Bonferroni- and FDR-corrected p-values with side-by-side Manhattan plots once again:



8.2 Multivariate methods

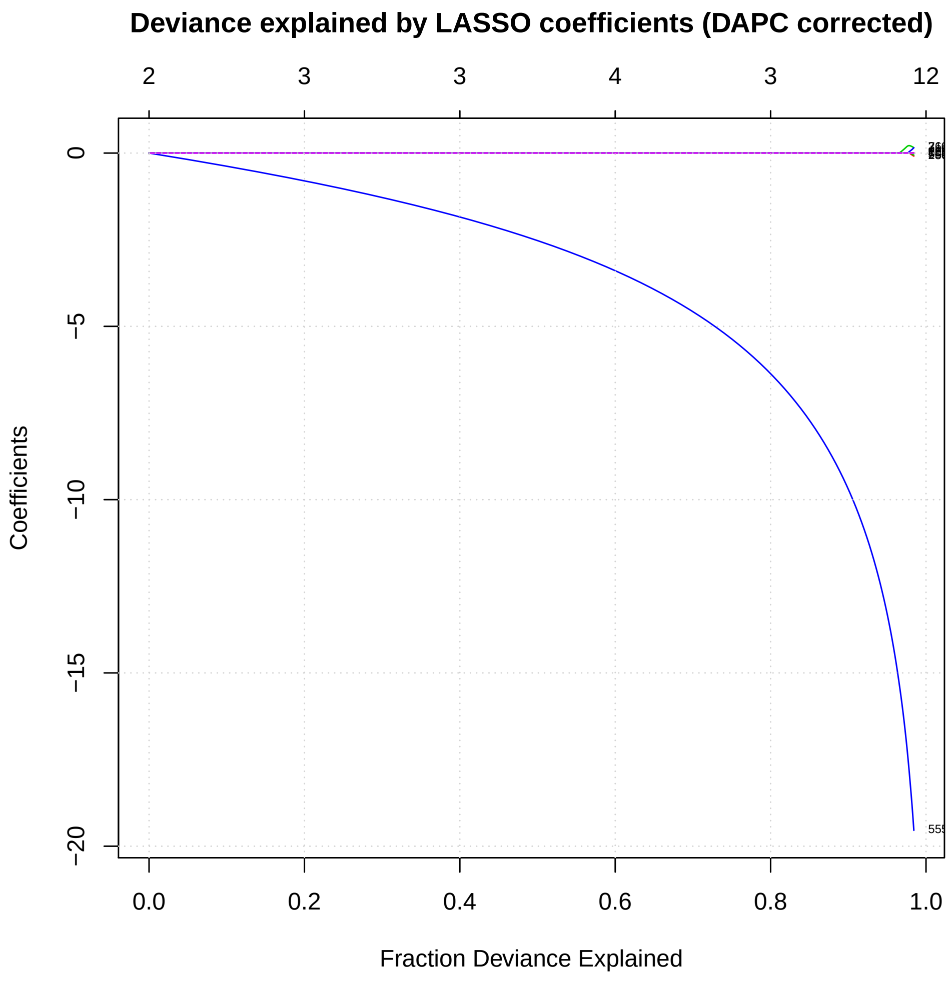
8.2.1 LASSO

The LASSO method can be performed using the exact same code as before (replacing `snps` with `snps.corrected`).
We pull out the fit from our `glmnet` object.

```
fit <- LASSO$glmnet.fit
```

And plot the deviance explained by the shrinking coefficients.

```
## standard code:
plot(fit, xvar = "dev", label = TRUE)
grid()
title("Deviance explained by LASSO coefficients (DAPC corrected)", line=3)
```



Good! This time we have reached the necessary predictive power to expect some non-zero coefficients.
So, how many did LASSO find this time?

```
res <- which(beta[-1] !=0)
```

```
length(res)
# 5
```

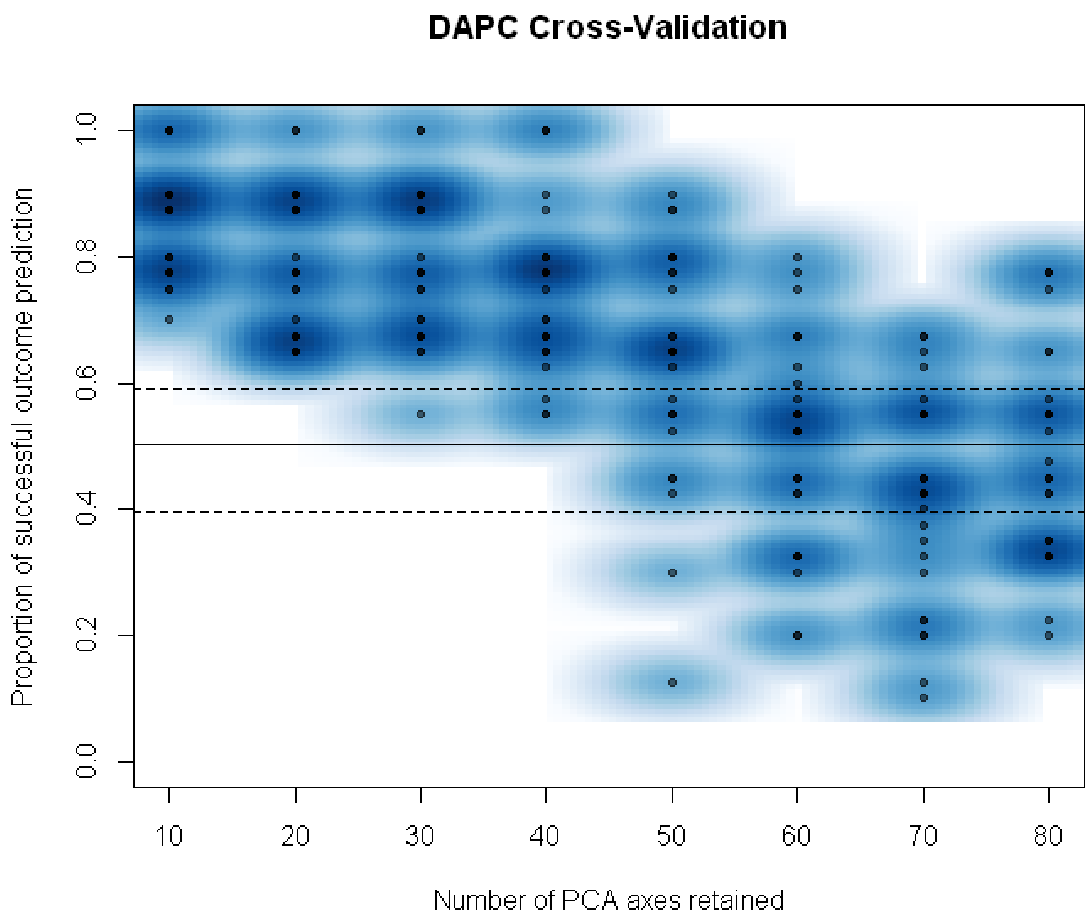
Aha! This time we find one more SNP than we did before correcting for populaiton stratification.

```
sigSNPs.DAPC[[2]] <- list(res)
names(sigSNPs.DAPC)[[2]] <- "multivariate"
names(sigSNPs.DAPC$multivariate) <- "lasso"
```

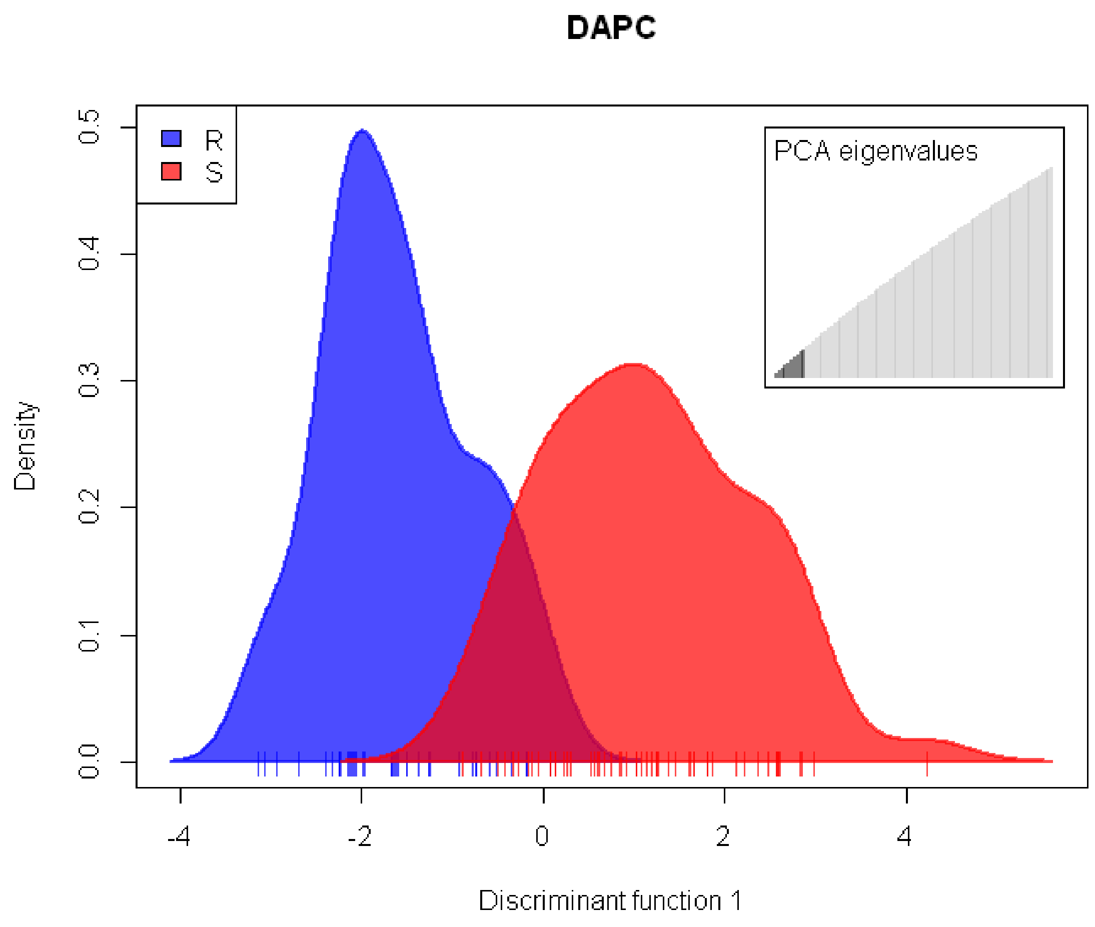
8.2.2 DAPC-based feature selection

We can again run the DAPC-based feature selection approach with `snpzip`.

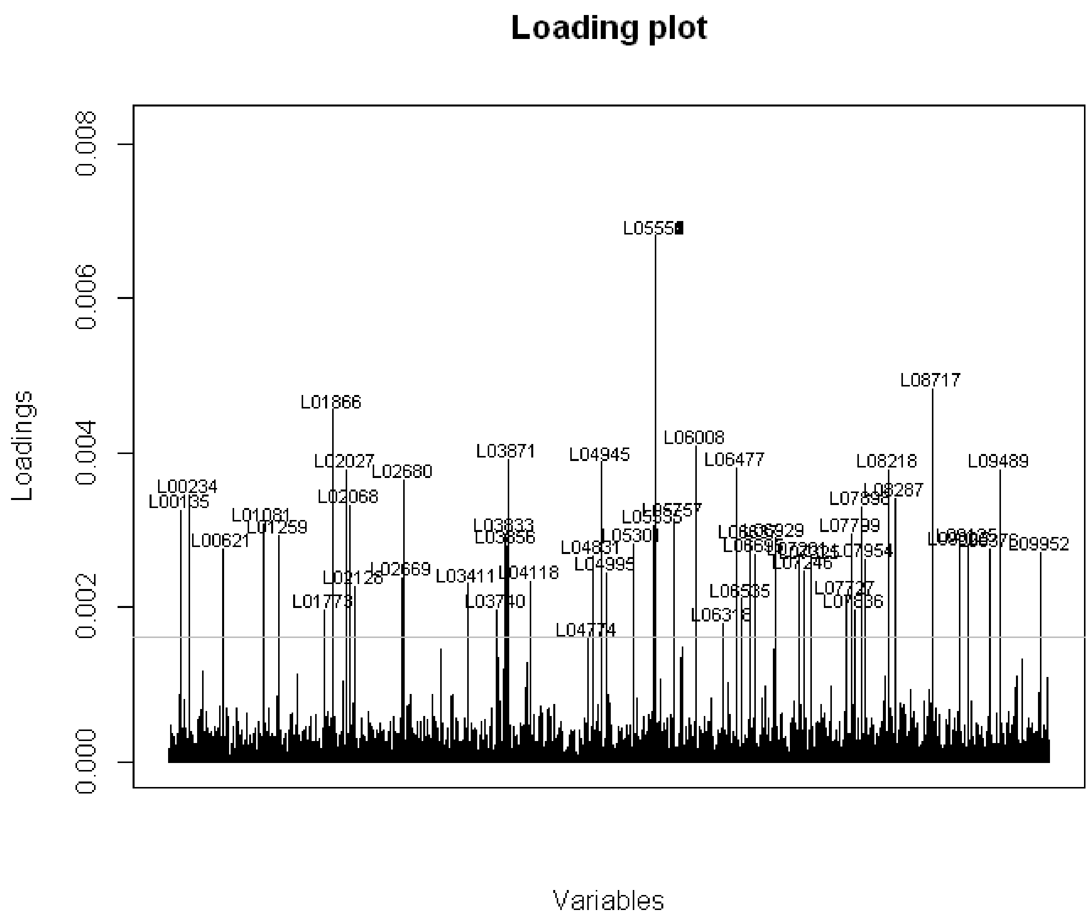
```
result <- snpzip(snps.corrected, phen,
                 xval.plot=TRUE, plot=TRUE, loading.plot=TRUE,
                 method="ward")

par(ask=FALSE)
```



Excellent. We are able to identify a number of PCs that enables us to predict the group membership of individuals with a reasonable degree of accuracy.



We also see a reasonbly good separation of the two phenotypic groups along the discriminant axis, though there is still some overlap in the middle.



And our loadingplot seems much more reasonable this time around. Once again, we pull out the SNPs selected from the output of `snpzip`.

```
res <- result$FS[[2]]
```

And we see that we have selected 53 significant SNPs.

```
length(res)
# 53
```

Finally, we store these in our list `sigSNPs.DAPC`

```
sigSNPs.DAPC[[2]][[2]] <- res
names(sigSNPs.DAPC$multivariate)[[2]] <- "dapc"
```

8.3 Answers!

We are just about finished here!
In a real GWAS study, in fact, you *would* be finished at this point. And, if the results you had found with different methods did not agree, then you would be left scratching your head (or, if you were feeling productive, heading for the lab). In the real world, beyond making informed inferences based on your set(s) of results, the only way to know whether the candidate SNPs you identified are *truly* associated with the phenotype in question is to confirm the association with traditional "bottom-up" laboratory experiments.
Lucky for you, this is not "real life", *per se*. Instead, I, the God of All Simulations for this practical, happen to have the answers.
To get the numbers of the columns in the SNP matrix which contain the truly associated SNPs, all you need to do is the following:

```
set.seed(1) # important!
snps.assoc <- sample(c(1:10050), 50)
snps.assoc <- sort(c(c(snps.assoc), c(5551:5555)))
snps.assoc

##  [1]  135  234  621 1081 1259 1773 1866 2027 2068 2128 2669 2680 3411 3740
## [15] 3813 3833 3856 3871 4118 4774 4831 4945 4995 5301 5535 5551 5552 5553
## [29] 5554 5555 5757 6008 6318 6477 6535 6637 6695 6897 6929 7201 7246 7325
## [43] 7727 7799 7836 7898 7954 8218 8287 8717 9025 9125 9376 9489 9952
```

Et voila! There were, after all, 55 truly associated SNPs to be found.
Armed with this information, you may now make a more informed comparison of the performance of the GWAS methods we tried out. Sensitivity and specificity are two of the most commonly used metrics of evaluation for classification tests.
To calculate sensitivity and specificity for each classification test, you will need to measure the following four quantities for the test in question:

- **TP**: The number of True Positives (ie. n.times you called a SNP "significant" when it truly was).
- **TN**: The number of True Negatives (ie. n.times you *ruled out* an insignificant SNP when you truly should have).
- **FP**: The number of False Positives (ie. n.times you called a SNP "significant" when it, in fact, was not).
- **FN**: The number of False Negatives (ie. n.times you ruled out a SNP and called it "insignificant" when it was, in fact, significant).

The following formulas can then be used to compute the performance metrics:

```r
sensitivity <- (TP / (TP + FN))

specificity <- (TN / (TN + FP))
```

Now compare! What do you think makes the different trade-offs between sensitivity and specificity advantageous or disadvantageous? Are there any circumstances in which these trade-offs might cause you to favour certain methods over others?