mixomics

Barry Digby

29/11/2018

To do list

Column1/Row1 of counts(dds). This contains an unannotated gene '.' It is possible that this column collected all counts for genes that could not be assigned to correct gene_name (t_data.ctab files contains a gene_name field. In some rows, the gene name is '.'). I have decided to remove this column from analysis. It contained millions of counts, investigate further..

```
## Loading required package: S4Vectors

## Loading required package: stats4

## Loading required package: BiocGenerics

## Loading required package: parallel

##
## Attaching package: 'BiocGenerics'

## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB

## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs

## The following objects are masked from 'package:base':
##
##     anyDuplicated, append, as.data.frame, basename, cbind,
##     colMeans, colnames, colSums, dirname, do.call, duplicated,
##     eval, evalq, Filter, Find, get, grep, grepl, intersect,
##     is.unsorted, lapply, lengths, Map, mapply, match, mget, order,
##     paste, pmax, pmax.int, pmin, pmin.int, Position, rank, rbind,
##     Reduce, rowMeans, rownames, rowSums, sapply, setdiff, sort,
##     table, tapply, union, unique, unsplit, which, which.max,
##     which.min

##
## Attaching package: 'S4Vectors'

## The following object is masked from 'package:base':
##
##     expand.grid

## Loading required package: IRanges

## Loading required package: GenomicRanges

## Loading required package: GenomeInfoDb

## Loading required package: SummarizedExperiment
## Loading required package: Biobase

## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
##     'browseVignettes()'. To cite Bioconductor, see
##     'citation("Biobase")', and for packages 'citation("pkgname")'.

## Loading required package: DelayedArray

## Loading required package: matrixStats

##
## Attaching package: 'matrixStats'

## The following objects are masked from 'package:Biobase':
##
##     anyMissing, rowMedians

## Loading required package: BiocParallel

##
## Attaching package: 'DelayedArray'

## The following objects are masked from 'package:matrixStats':
##
##     colMaxs, colMins, colRanges, rowMaxs, rowMins, rowRanges

## The following objects are masked from 'package:base':
##
##     aperm, apply

## Loading required package: MASS

## Loading required package: lattice

## Loading required package: ggplot2

##
## Loaded mixOmics 6.6.0
##
## Thank you for using mixOmics! Learn how to apply our methods with our tutorials on www.mixOmics.org,
## Questions: email us at mixomics[at]math.univ-toulouse.fr
## Bugs, Issues? https://github.com/mixOmicsTeam/mixOmics/issues
## Cite us:  citation('mixOmics')
```

Read in Data

Set working Dir, use **tximport** to import the t_data.ctab files into R.
Construct experiment design using DESeq, run DESeq. Using the DESeq function is useful here, as under the hood it carrries out normalization (divide the counts by the size factors); a required preprocessing step before using mixOmics.

In this step we will also carry out minimal prefiltering. The mixOmics manual states for prefiltering: “for RNA-seq data, by removing consistently low counts”.

```r
directory <- "/Users/barrydigby/Desktop/read_tables/"

data <- c("/Users/barrydigby/Desktop/read_tables/H84S2Ctrl_S7/t_data.ctab",
          "/Users/barrydigby/Desktop/read_tables/H84WTCtrl_S1/t_data.ctab",
          "/Users/barrydigby/Desktop/read_tables/H85S2Ctrl_S8/t_data.ctab",
          "/Users/barrydigby/Desktop/read_tables/H85WTCtrl_S2/t_data.ctab",
          "/Users/barrydigby/Desktop/read_tables/H86S2Ctrl_S9/t_data.ctab",
          "/Users/barrydigby/Desktop/read_tables/H86WTCtrl_S3/t_data.ctab")

tmp <- read.table(data[1], header = TRUE)
names(data) <- c("S2Ctrl_S7", "WTCtrl_S1", "S2Ctrl_S8", "WTCtrl_S2", "S2Ctrl_S9", "WTCtrl_S3")
tx2gene <- tmp[, c("t_name", "gene_name")]
txi <- tximport(data, type = "stringtie", tx2gene = tx2gene)
```

```
## reading in files with read_tsv

## 1 2 3 4 5 6
## summarizing abundance
## summarizing counts
## summarizing length
```

```
sampleNames <- c("S2Ctrl_S7", "WTCtrl_S1", "S2Ctrl_S8", "WTCtrl_S2", "S2Ctrl_S9", "WTCtrl_S3")
sampleGroup <- c("S2", "WT", "S2", "WT", "S2", "WT")
sampleTable <- data.frame(sampleName = sampleNames, type = sampleGroup)

rownames(sampleTable) <- colnames(txi$counts)
ddsTxi <- DESeqDataSetFromTximport(txi, sampleTable, design = ~ type)
```

```
## using counts and average transcript lengths from tximport
```

```
keep <- rowSums(counts(ddsTxi)) >= 30 # 6 samples, min 5 reads per column.
dds1 <- ddsTxi[keep,]
dds1$type <- relevel(dds1$type, ref = "WT")
dds <- DESeq(dds1)
```

```
## estimating size factors
## using 'avgTxLength' from assays(dds), correcting for library size
## estimating dispersions
## gene-wise dispersion estimates
## mean-dispersion relationship
## final dispersion estimates
## fitting model and testing
```

Pre processing the count table

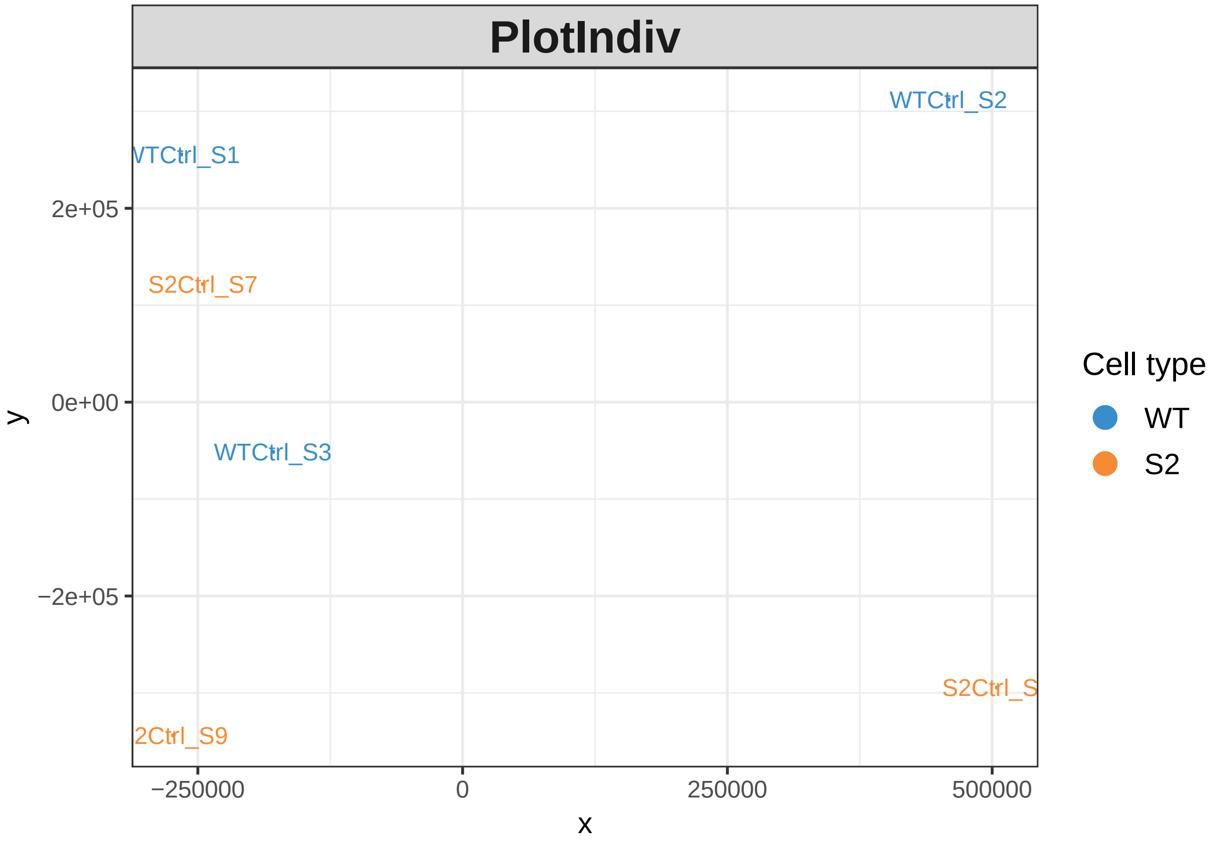
MixOmics manual: Our methods use matrix decomposition techniques. Therefore, the numeric data matrix or data frames have n observations or samples in rows and p predictors or variables (e.g. genes, proteins, OTUs) in columns.
To satisfy this requirement, simply transpose the count matrix. Set normalized=TRUE and replaced=FALSE. This will omit the counts marked as outliers.

```
tcounts <- t(counts(dds, normalized=TRUE, replaced=FALSE))

tcounts <- tcounts[,-1]
```

Unsupervised Analysis

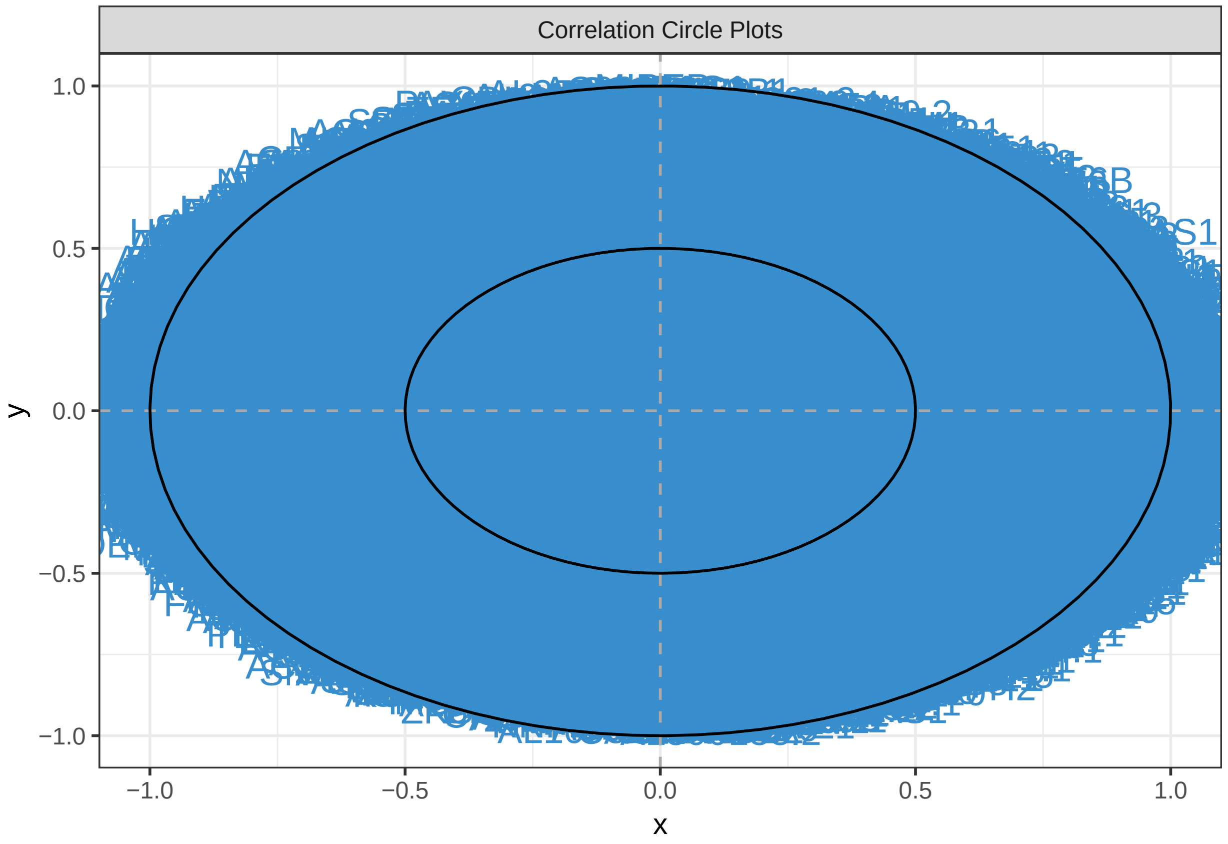
Before starting supervised analysis, it is advisable to check the data using unsupervised techniques such as PCA:

```r
X <- tcounts

ctab.pca <- pca(X)  # 1 Run the method
plotIndiv(ctab.pca, group = dds$type, legend = TRUE, legend.title = "Cell type")
```



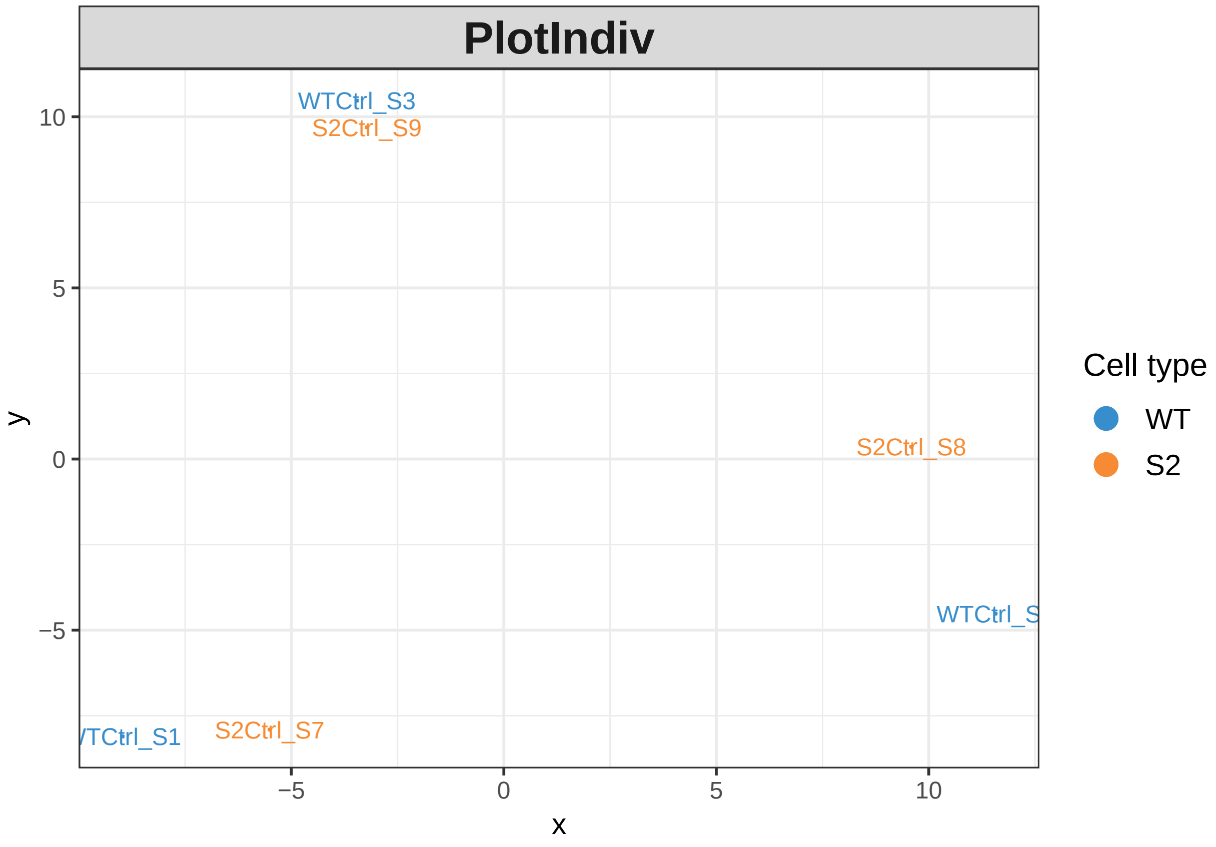
Next we will plot the variables driving variation in the samples:
This will result in a messy correlation plot. We will have to set an arbitrary threshold to select for the top number of variables.

```r
plotVar(ctab.pca)
```

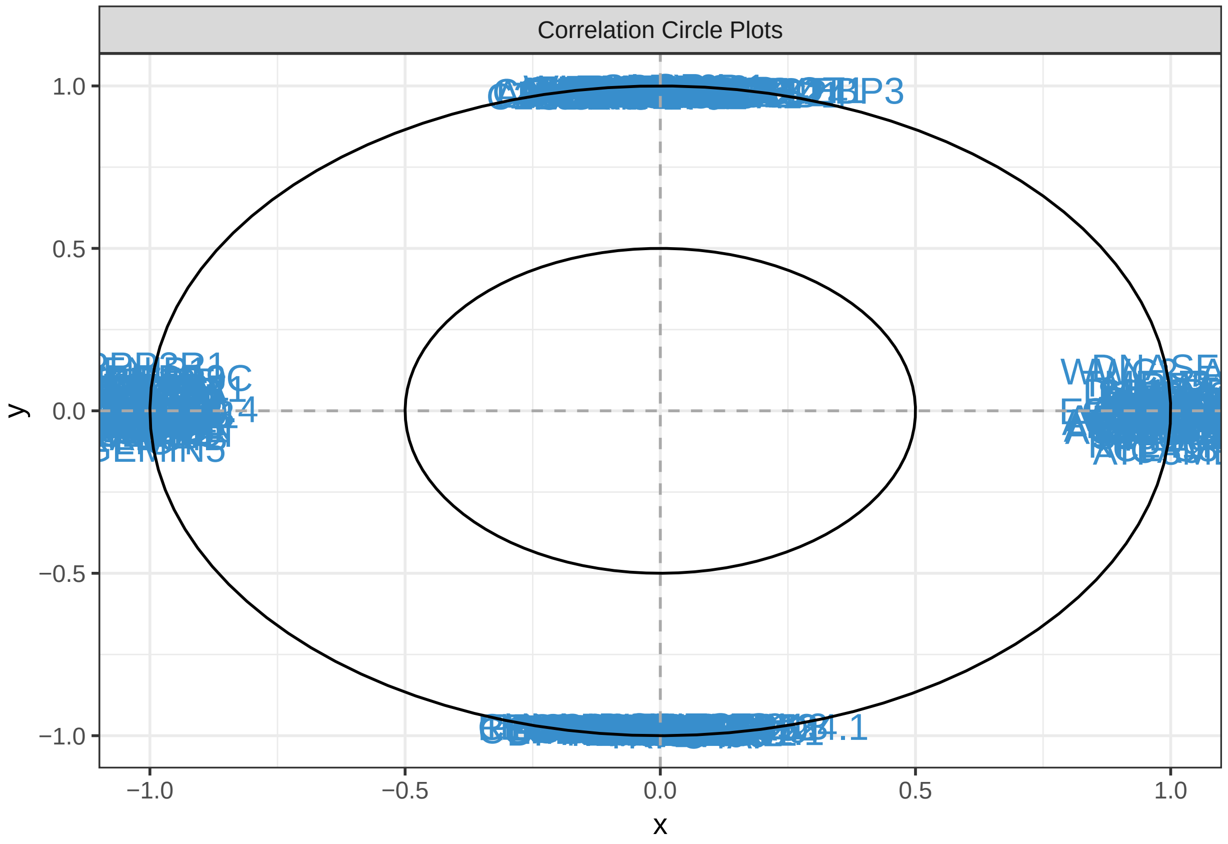


Lets look at the top 100 variables:

```
top100.spca <- spca(X, keepX=c(100,100))

plotIndiv(top100.spca, group = dds$type, legend = TRUE, legend.title = "Cell type")
```



```
plotVar(top100.spca)
```



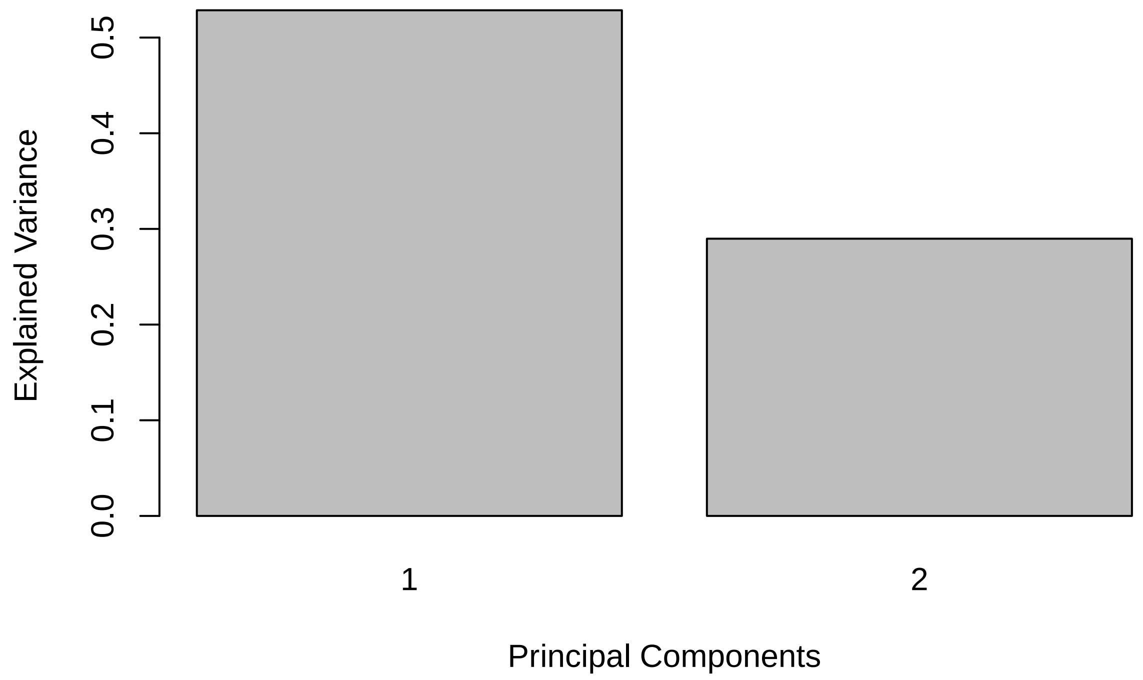
Extract the top 100 variables:

```
top100.comp1 <- selectVar(top100.spca, comp = 1)$value

top100.comp2 <- selectVar(top100.spca, comp=2)$value

# output to file for comaprison with supervised learning output downstream:

write.csv(top100.comp1, "/Users/barrydigby/Desktop/sPLSDA/top100_vars_comp1_UNSUP.csv")
write.csv(top100.comp2, "/Users/barrydigby/Desktop/sPLSDA/top100_vars_comp2_UNSUP.csv")
```

Variance explained and choice of number of components:

In the previous step we only selected comp=1 and comp=2. This is because the optimum number of components chosen by the model is ncomp=2. This means that the first two components explain most of the variance. This result is not surprising given there are only 2 cell types, WT and S2+.

```
plot(ctab.pca)
```

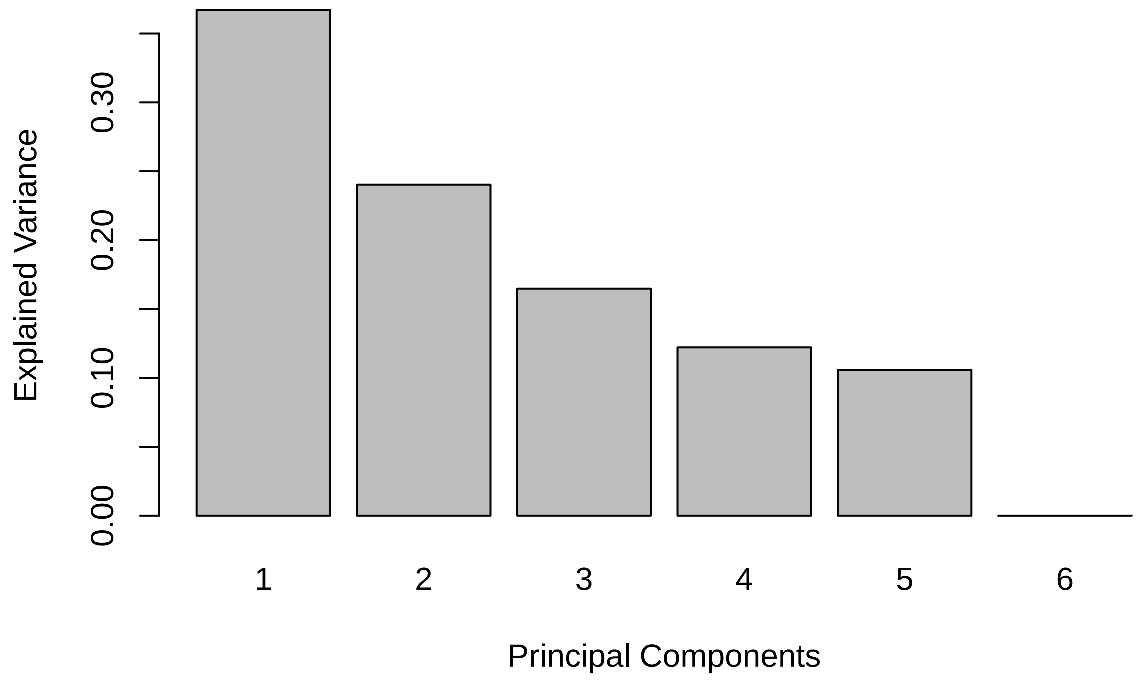


Supervised analysis

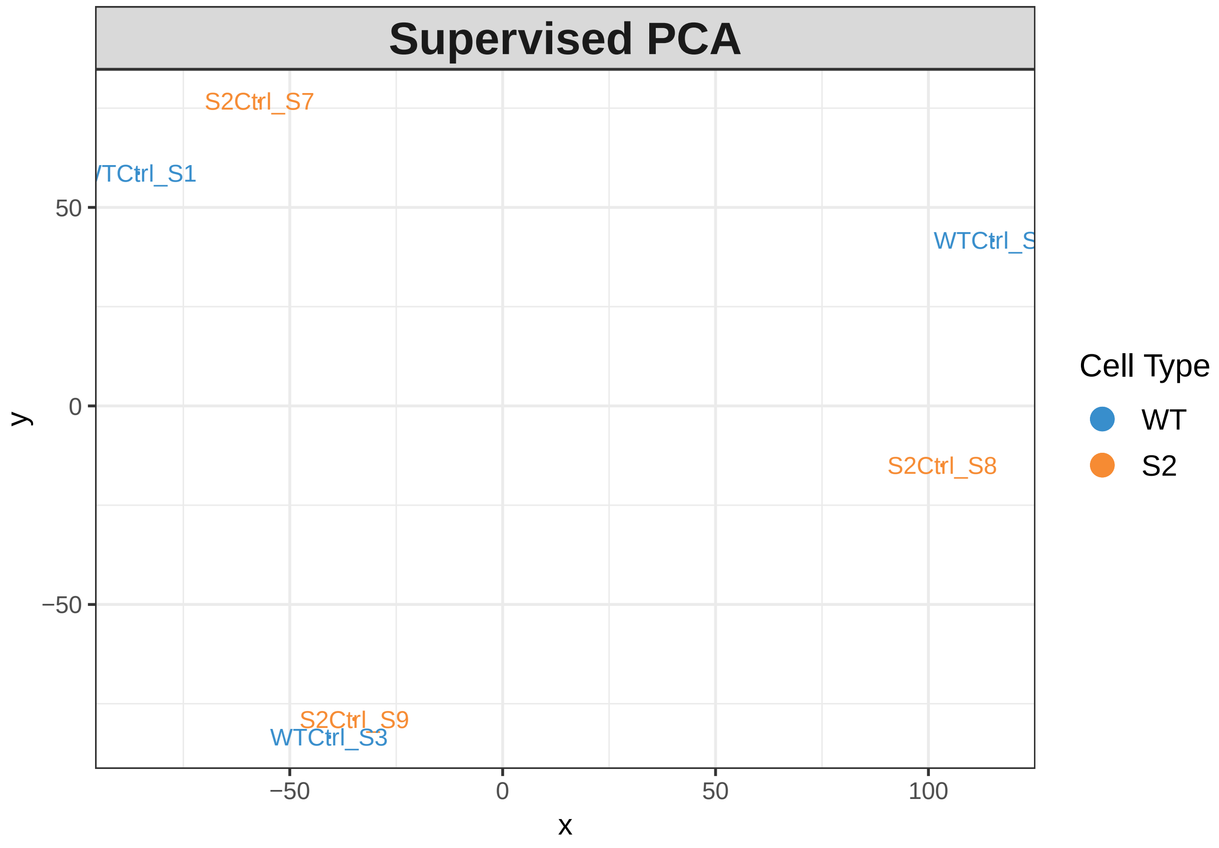
PCA

This time we can set the number of components, just to test if any others could acccount for variation in the data:
In the plot we can see that ncomp could be set to 5.

```
ctab.pca.sup = pca(X, ncomp = 6, center = TRUE , scale = TRUE)
# center = shift variables to be zero centered
plot(ctab.pca.sup)
```



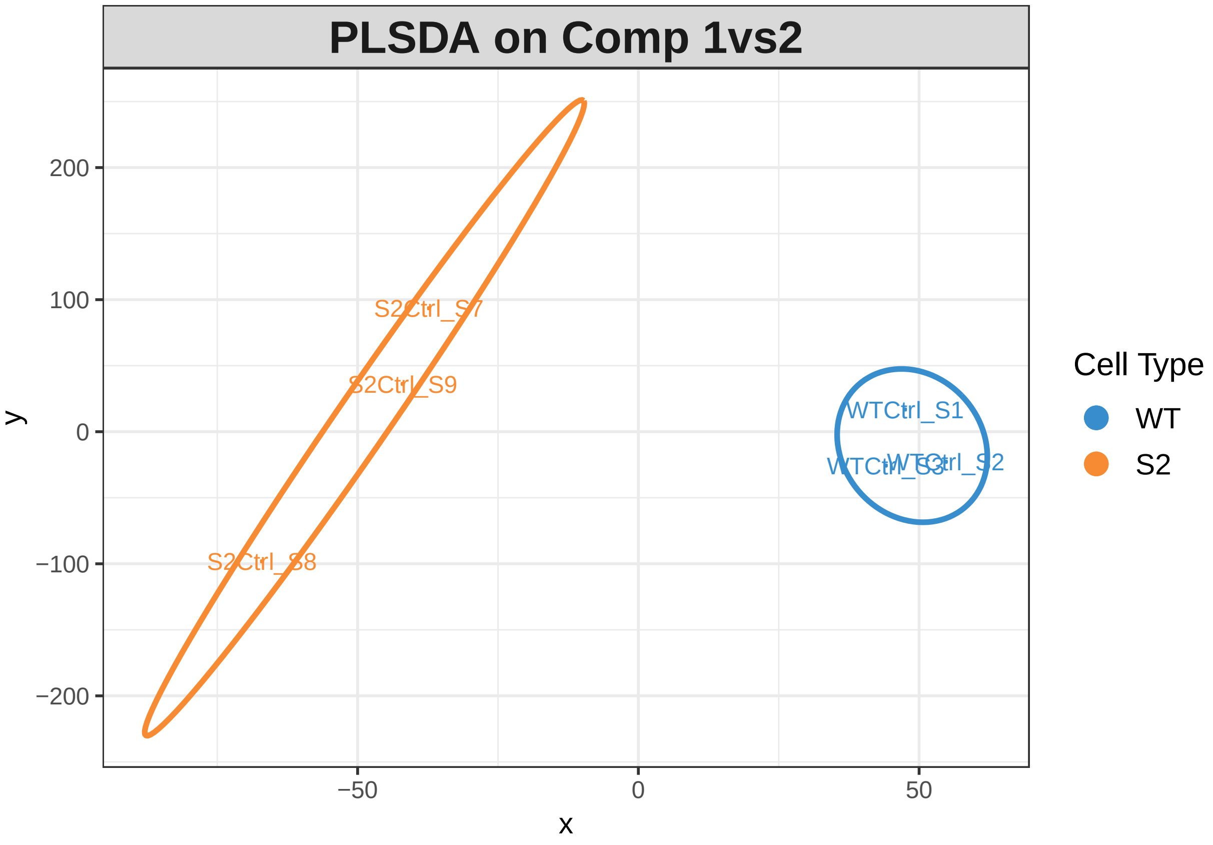
```r
plotIndiv(ctab.pca.sup, group = dds$type, ind.names = TRUE,
          legend = TRUE, legend.title = "Cell Type", title = 'Supervised PCA')
```



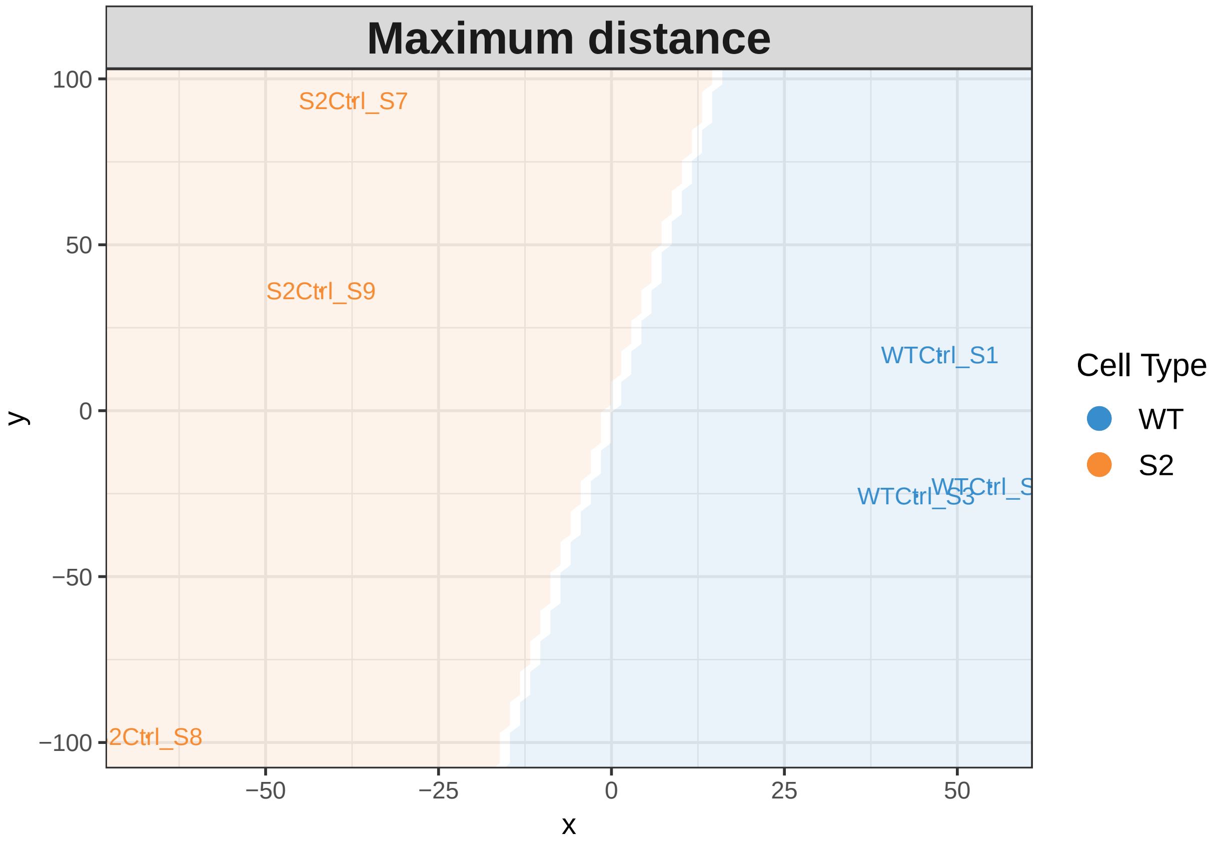
PLS-DA analysis

The PLS-DA model is fitted with 5 components, and will be tested to evaluate the number of components to select for the final model.
First, we can display PCA across component 1 & 2. The PLSDA method shows clear seperation between the two classes of cell type.

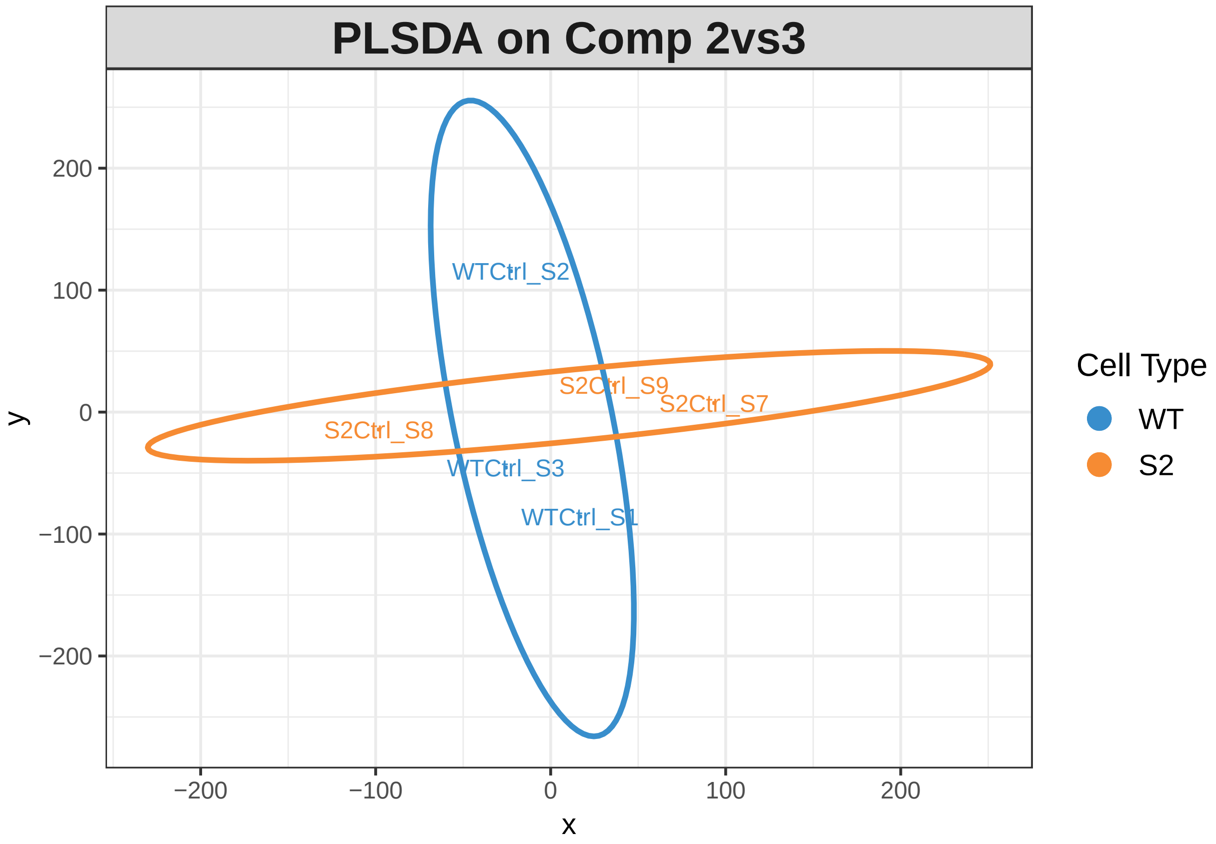
```
Y <- as.factor(dds$type)

ctab.plsda <- plsda(X, Y, ncomp = 5)

plotIndiv(ctab.plsda , comp = 1:2,
          group = Y, ind.names = TRUE,
          ellipse = TRUE, legend = TRUE,
          legend.title= "Cell Type", title = 'PLSDA on Comp 1vs2')
```



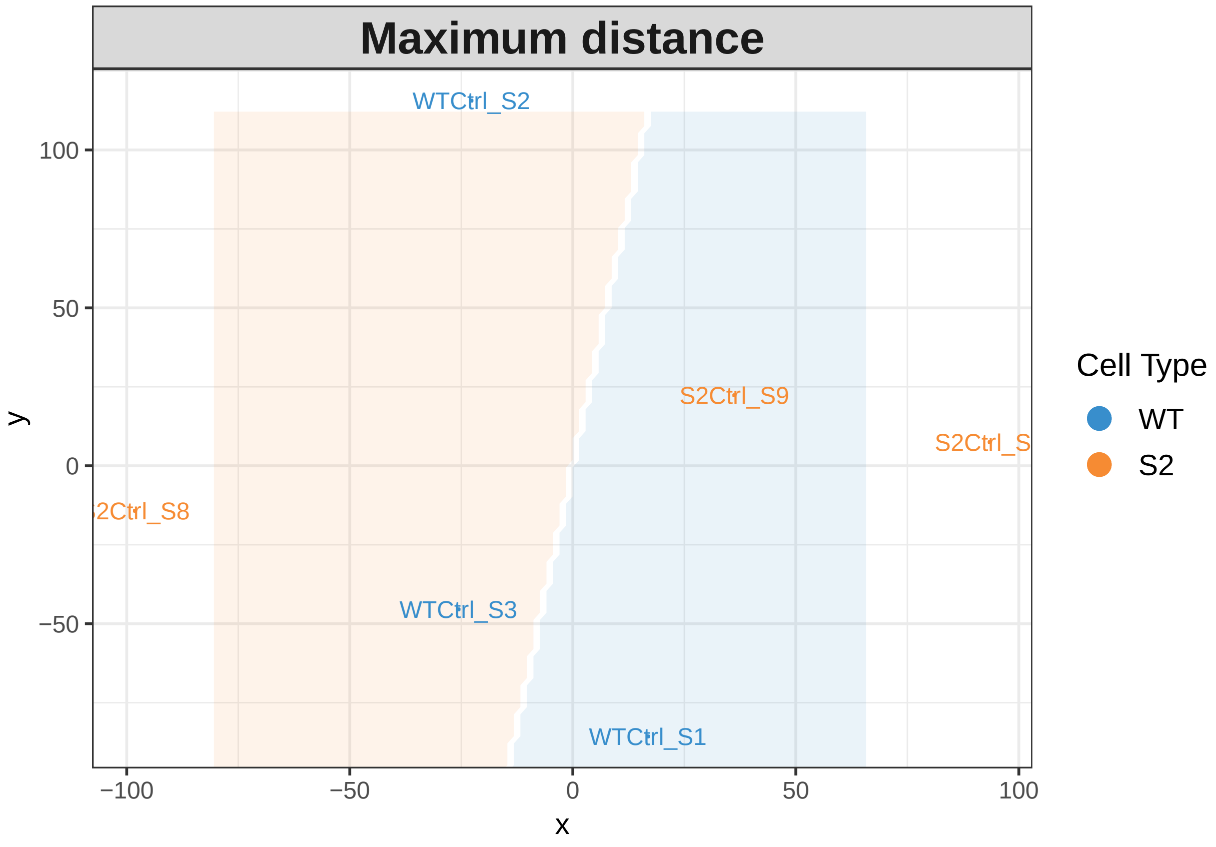
```
background = background.predict(ctab.plsda, comp.predicted=2, dist = "max.dist")

plotIndiv(ctab.plsda, comp = 1:2,
          group = Y, ind.names = TRUE, title = "Maximum distance",
          legend = TRUE, legend.title = "Cell Type",  background = background)
```



```
plotIndiv(ctab.plsda , comp = 2:3,
          group = Y, ind.names = TRUE,
          ellipse = TRUE, legend = TRUE,
          legend.title= "Cell Type", title = 'PLSDA on Comp 2vs3')
```

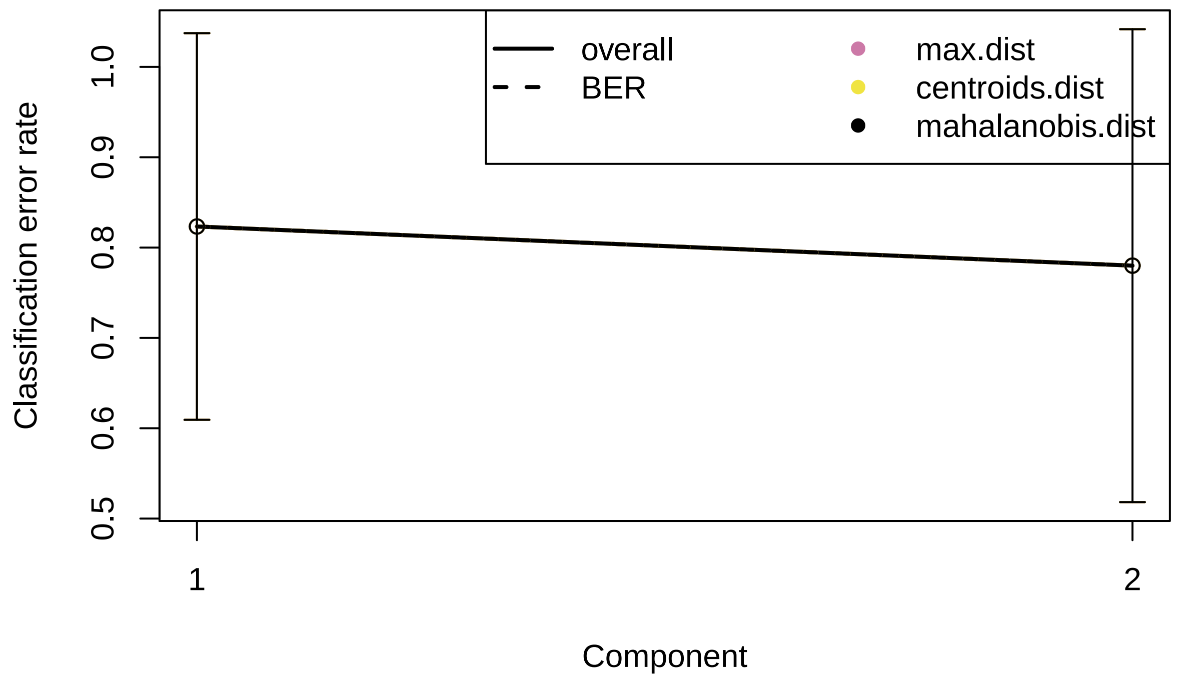


```
background = background.predict(ctab.plsda, comp.predicted=2, dist = "max.dist")

plotIndiv(ctab.plsda, comp = 2:3,
          group = Y, ind.names = TRUE, title = "Maximum distance",
          legend = TRUE, legend.title = "Cell Type",  background = background)
```



Testing components 2 vs 3 above, we can see that the distinctions are not obvious. I am expecting the tuning process to chose 2 components maximum.
To test the performance of the PLSDA model, we use the **perf** function. The perf function uses (3) fold cross validation to estimate the classification error rate. This is used to assess the optimal number of components to use in or final model.

```
#perf.plsda.ctab <- perf(ctab.plsda, validation = "Mfold", folds = 3,
  #                 progressBar = TRUE, auc = TRUE, nrepeat = 50)

#folds dropped from 5 to 3 to avoid error messages.

# system is computationally singular: reciprocal condition number = 4.52019e-19
# Failed on component 3. Need to reset ncomp=2 for model to run.

ctab.plsda <- plsda(X, Y, ncomp = 2)

perf.plsda.ctab <- perf(ctab.plsda, validation = "Mfold", folds = 3,
                  progressBar = FALSE, auc = TRUE, nrepeat = 50)

plot(perf.plsda.ctab, col = color.mixo(5:7), sd = TRUE, legend.position = "horizontal")
```

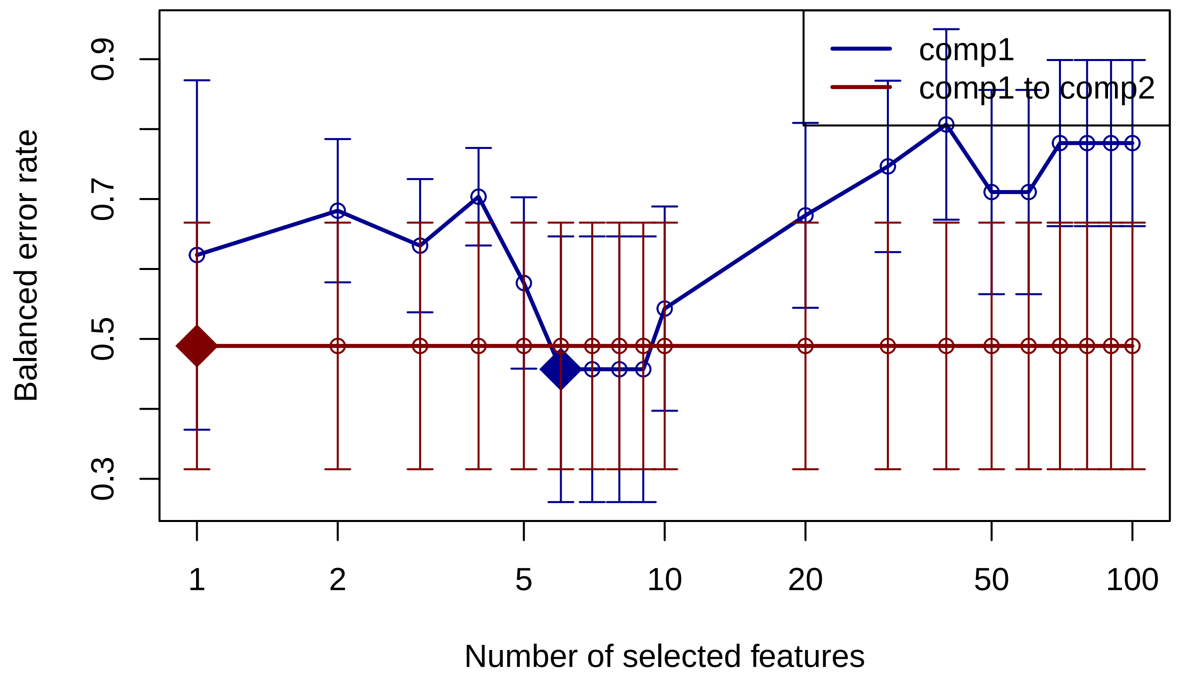


We can now use the *tune* function to assess the optimal number of variables to select on each component.

```
list.keepX <- c(1:10,  seq(20, 100, 10))

tune.splsda.ctab <- tune.splsda(X, Y, ncomp = 2,
                                 validation = 'Mfold',
                                 folds = 3, dist = 'max.dist', progressBar = FALSE,
                                 measure = "BER", test.keepX = list.keepX,
                                 nrepeat = 50)
```

We can now extract the optimum number of components, stored in **ncomp**.
The optimum number of variabels to select is denoted as a diamond on the plot:

```
ncomp <- tune.splsda.ctab$choice.ncomp$ncomp
select.keepX <- tune.splsda.ctab$choice.keepX[1:ncomp]


plot(tune.splsda.ctab, col = color.jet(2))
```



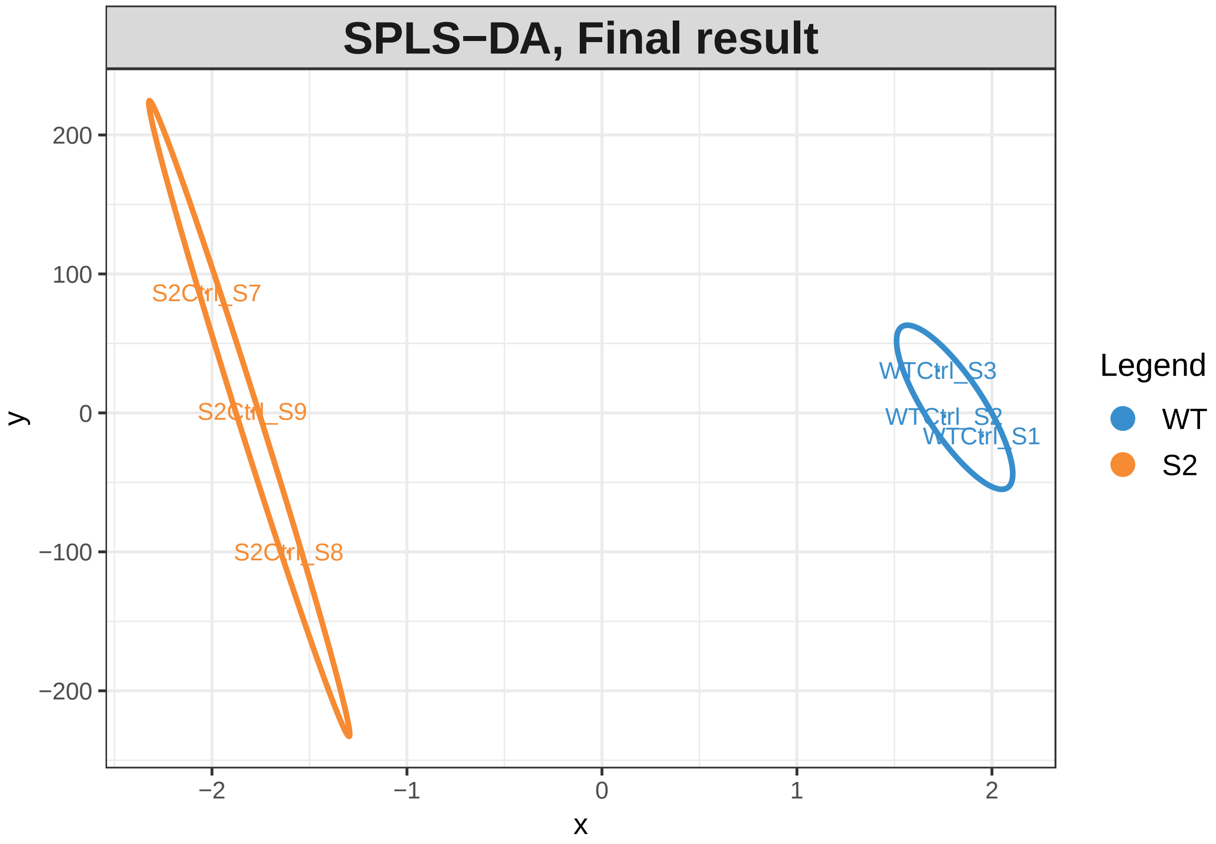
```
ncomp
```

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

sPLSDA

After tuning our paramters, we can now run the final model:
note: the model cannot run on ncomp=1. It was increased to 2..

```
ctab.splsda.final <- splsda(X, Y, ncomp = 2, keepX = select.keepX)


plotIndiv(ctab.splsda.final, ind.names = TRUE, legend=TRUE,
          ellipse = TRUE, title="SPLS-DA, Final result")
```

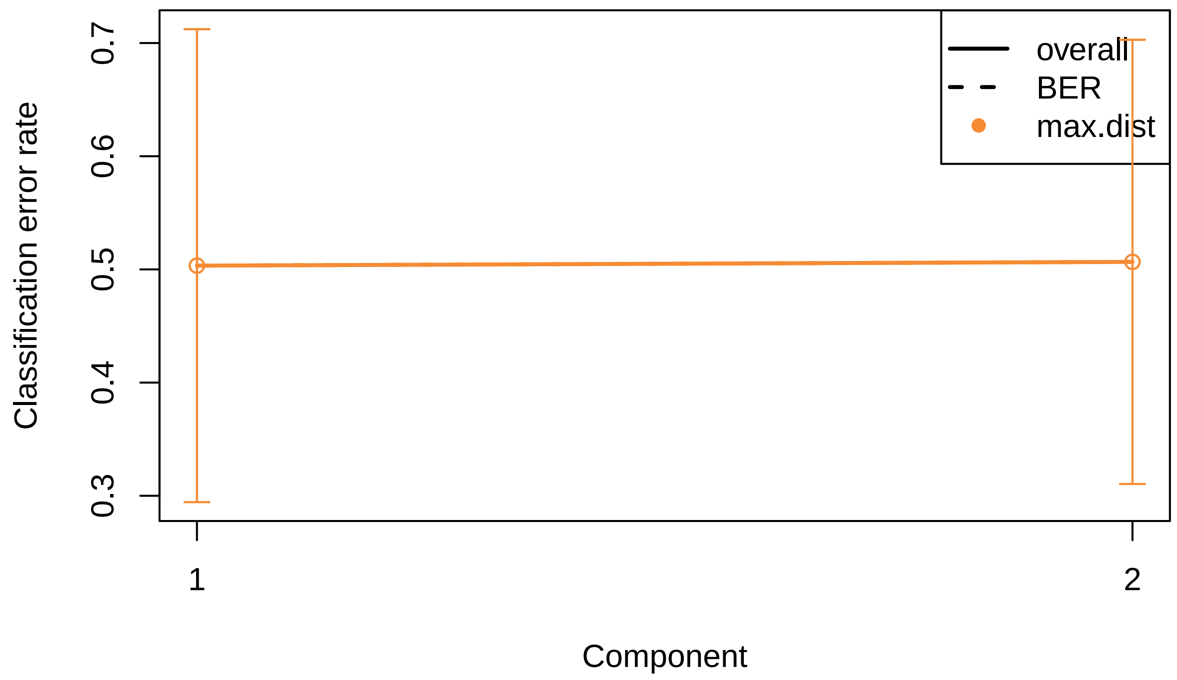


Assess the performance of the final model:

```
perf.ctab.final <- perf(ctab.splsda.final, validation = "Mfold", folds = 3,
                        dist = 'max.dist', nrepeat = 50,
                        progressBar = FALSE)

perf.ctab.final$error.rate
```

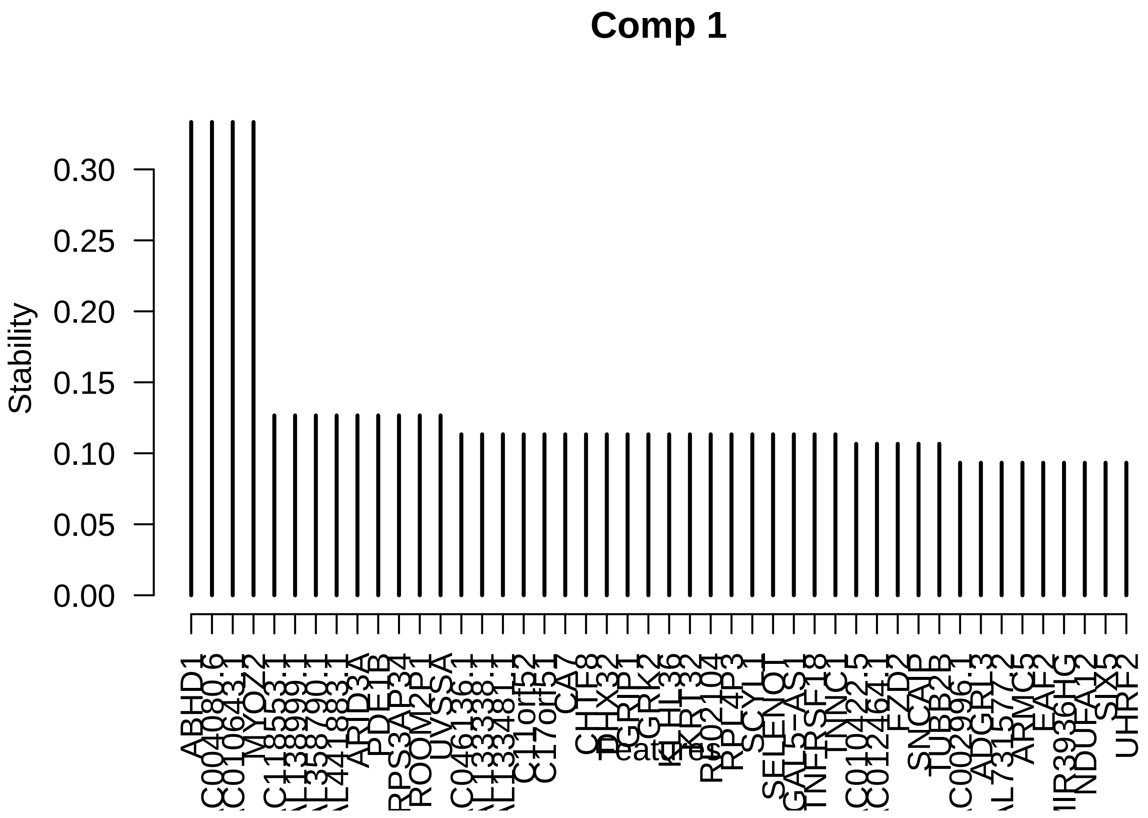
```
## $overall
##         max.dist
## comp 1 0.5033333
## comp 2 0.5066667
##
## $BER
##         max.dist
## comp 1 0.5033333
## comp 2 0.5066667
```

```
plot(perf.ctab.final, col = color.mixo(2))
```



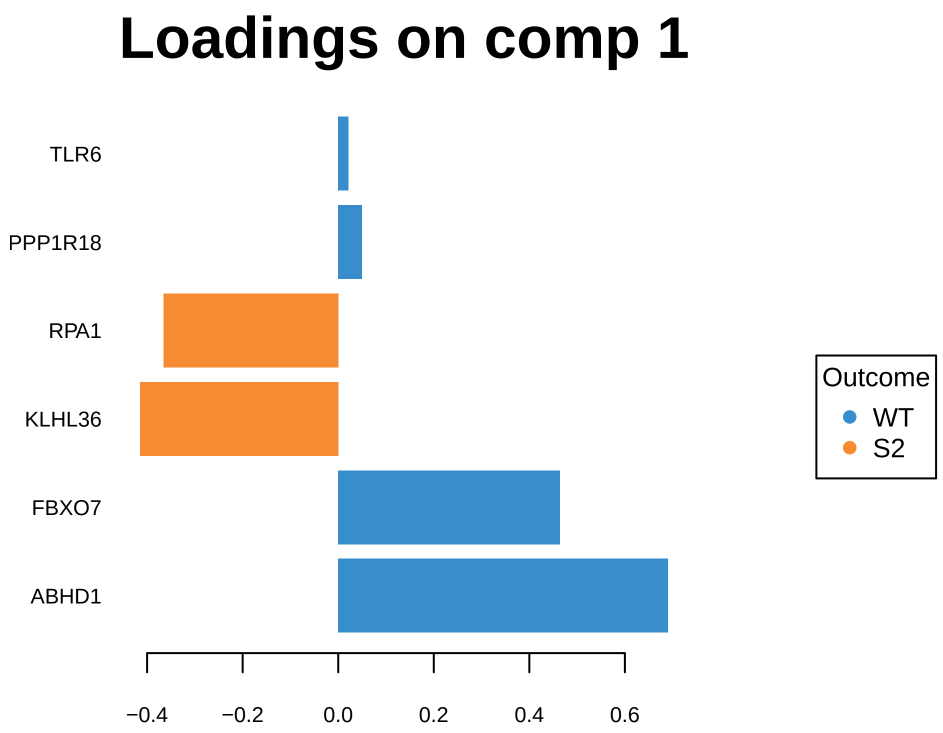
Assess the stability of the run:

```r
par(mfrow=c(1,1))
plot(perf.ctab.final$features$stable[[1]], type = 'h', ylab = 'Stability',
     xlab = 'Features', main = 'Comp 1', las =2)
```

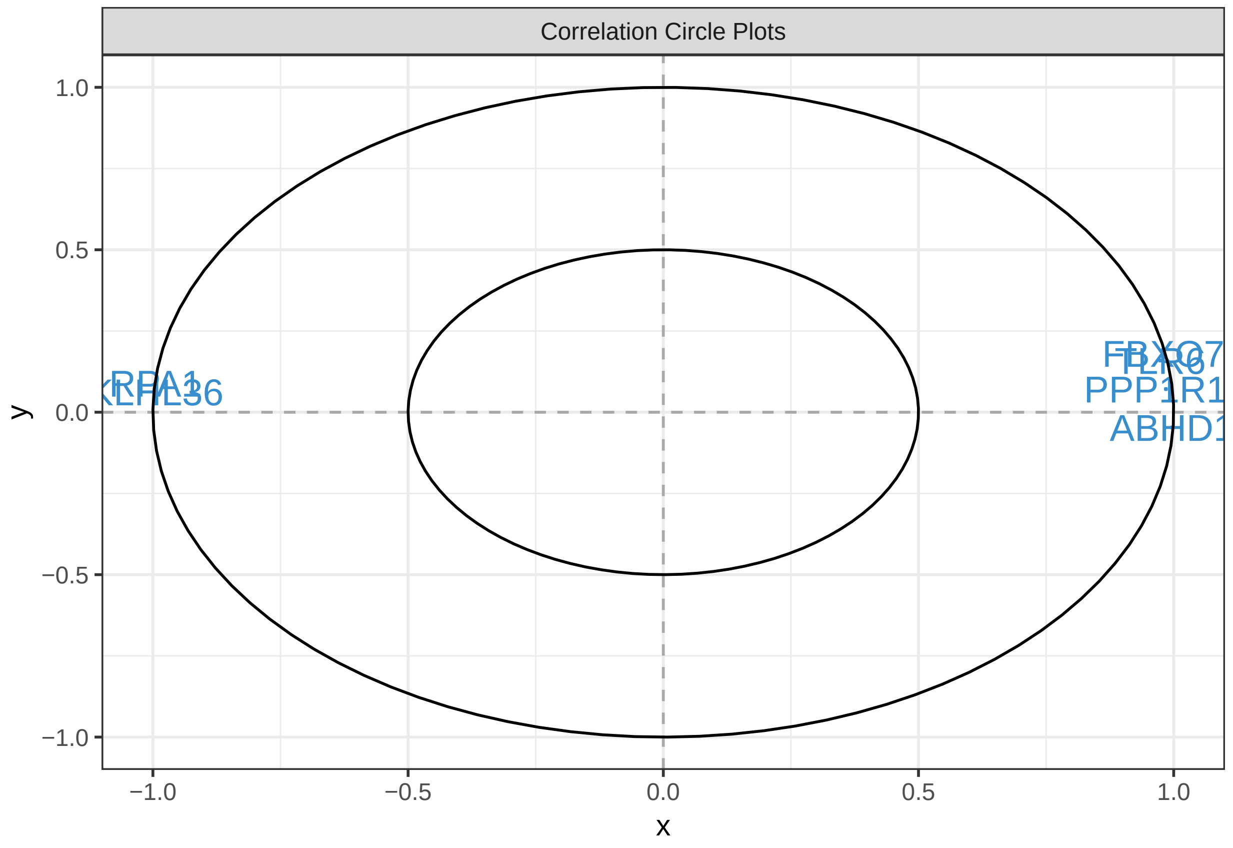


Assess the 6 loadings on comp 1:

```r
plotLoadings(ctab.splsda.final, comp = 1, title = 'Loadings on comp 1',
             contrib = 'max', method = 'mean')
```



```r
plotVar(ctab.splsda.final, comp.select = 1)
```



```r
top6.final <- selectVar(ctab.splsda.final, comp = 1)$value
top6.final
```

```
##            value.var
## ABHD1    0.69056375
## FBXO7    0.46361416
## KLHL36  -0.41457985
## RPA1    -0.36525043
## PPP1R18  0.04923599
## TLR6     0.02179706
```

```r
write.csv(top6.final, "/Users/barrydigby/Desktop/sPLSDA/top_genes_sPLSDA.csv")
```