Running the perf function

Kim-Anh Le Cao

01 September 2014

The function **valid** has been superseded by the **perf** function to avoid some selection bias in the sparse functions. This has been fixed.

Load the latest version of the package 5.0-3

```
# locally:
library(mixOmics, lib='../../mixomics/MyR/')

#library(mixOmics)
```

```
sessionInfo()
```

```
## R version 3.1.0 (2014-04-10)
## Platform: x86_64-apple-darwin10.8.0 (64-bit)
##
## locale:
## [1] en_AU.UTF-8/en_AU.UTF-8/en_AU.UTF-8/C/en_AU.UTF-8/en_AU.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] mixOmics_5.0-3  lattice_0.20-29 MASS_7.3-33
##
## loaded via a namespace (and not attached):
##  [1] digest_0.6.4       evaluate_0.5.5     formatR_0.10
##  [4] grid_3.1.0         htmltools_0.2.4    igraph_0.7.1
##  [7] knitr_1.6          pheatmap_0.7.7     RColorBrewer_1.0-5
## [10] RGCCA_2.0          rgl_0.93.1098      rmarkdown_0.2.46
## [13] stringr_0.6.2      tools_3.1.0        yaml_2.1.13
```

This is an example to announce the transition to the perf function:

```
data(liver.toxicity)
X <- liver.toxicity$gene
Y <- liver.toxicity$clinic

res.pls = pls(X, Y, ncomp = 3, mode = "regression")
valid.pls = valid(res.pls, validation = "Mfold", folds = 5, criterion = "all",
    progressBar = FALSE)
```

```
## The valid function has been supersided by the perf function to evaluate the performance of the PLS, s
##      See ?perf for more details
```

Perf is an S3 method that applies on an object inherited from PLS, sPLS, PLS-DA or sPLS-DA. It supersedes the function 'valid', which we have removed from the package.

Perf retrieves the parameters input from the model run beforehand and assesses the performance of the model.

PLS methods

We will use the liver toxicity data for the PLS and sPLS methods where the Y response is a matrix composed of 10 clinical variables:

```
data(liver.toxicity)
X <- liver.toxicity$gene
Y <- liver.toxicity$clinic
```

Tuning the number of components

The Q2.total can be used to tune the number of components. The rule of thumbs is that a PLS component should be included in the model if its value is greater than or equal to 0.0975 (Tenenhaus, 1998, rule applied in the SIMCA software). After few important components, the Q2.total values should all be less than 0.0975. We have often experiences Q2.total values that do not decrease, which might be due to the large number of variables in the X data set.

Here is an example with PLS, where **ncomp = 3** should be enough for the model:

```
res.pls1 = pls(X, Y, ncomp = 10, mode = "regression")
tune.pls = perf(res.pls1, validation = "Mfold", folds = 5, criterion = "all",
    progressBar = FALSE)
tune.pls$Q2.total
```

```
##    comp 1    comp 2    comp 3    comp 4    comp 5    comp 6    comp 7
##  0.214133  0.040277  0.107942 -0.003466  0.071277 -0.015536 -0.023893
##    comp 8    comp 9   comp 10
## -0.006803  0.015436 -0.022081
```

In order to save some computational time (and output space) we can now rerun the perf function with ncomp = 3 components and look at the other criteria Q2, R2 and MSEP which are calculated for each response variable Y. According to the tuning step from above, we should only consider the columns ncomp = 3, although these more 'local' criteria also give a better idea of the fit for each response variable.

```
res.pls2 = pls(X, Y, ncomp = 3, mode = "regression")
perf.pls = perf(res.pls2, validation = "Mfold", folds = 5, criterion = "all",
    progressBar = FALSE)
perf.pls$Q2
```

```
##                      ncomp 1  ncomp 2   ncomp 3
## BUN.mg.dL.          0.514049  0.06188  0.025144
## Creat.mg.dL.        0.051391 -0.06608  0.009398
## TP.g.dL.           -0.043313  0.01171  0.192171
## ALB.g.dL.          -0.038366  0.12124  0.280171
## ALT.IU.L.           0.589725  0.31626  0.062465
## SDH.IU.L.          -0.001181 -0.11592  0.111836
## AST.IU.L.           0.566242  0.31044  0.095482
## ALP.IU.L.           0.144389 -0.06196 -0.002227
## TBA.umol.L.         0.572407 -0.07397  0.326516
## Cholesterol.mg.dL.  0.221113  0.18298 -0.071998
```

```
perf.pls$R2
```

```
##                       ncomp 1  ncomp 2 ncomp 3
## BUN.mg.dL.          0.5148571 0.544541 0.55580
## Creat.mg.dL.        0.0520316 0.017363 0.03185
## TP.g.dL.            0.0394756 0.016117 0.18939
## ALB.g.dL.           0.0009524 0.106859 0.35000
## ALT.IU.L.           0.5900411 0.723297 0.73915
## SDH.IU.L.           0.0155130 0.004692 0.03441
## AST.IU.L.           0.5665986 0.704510 0.73183
## ALP.IU.L.           0.1490419 0.109437 0.10997
## TBA.umol.L.         0.5725677 0.541088 0.69432
## Cholesterol.mg.dL. 0.2220677 0.363660 0.32065
```

sPLS

With sPLS, we first specify the number of variables to select on each component. Note that for ncomp = 1, the Q2.total should be equal to 'NA'.

```
res.spls = spls(X, Y, ncomp = 3, keepX = c(10, 5, 10), keepY = c(5, 8, 4), mode = "regression")
perf.spls = perf(res.spls, validation = "Mfold", folds = 5, criterion = "all",
    progressBar = FALSE)
perf.spls$Q2.total
```

```
##   comp 1   comp 2   comp 3
##  0.28701  0.14132 -0.07031
```

```
perf.spls$MSEP
```

```
##                     ncomp 1 ncomp 2 ncomp 3
## BUN.mg.dL.           0.5143  0.5081  0.5967
## Creat.mg.dL.         1.0166  1.0819  1.1927
## TP.g.dL.             1.1322  0.7739  0.7844
## ALB.g.dL.            1.1302  0.6046  0.6234
## ALT.IU.L.            0.1217  0.1266  0.1410
## SDH.IU.L.            1.0743  1.0527  1.1541
## AST.IU.L.            0.1385  0.1489  0.1554
## ALP.IU.L.            0.9334  0.9404  1.0110
## TBA.umol.L.          0.3491  0.2260  0.2248
## Cholesterol.mg.dL.   0.6082  0.5635  0.5668
```

```
perf.spls$R2
```

```
##                       ncomp 1  ncomp 2  ncomp 3
## BUN.mg.dL.          0.4775343 0.484310 0.420737
## Creat.mg.dL.        0.0002572 0.001439 0.006895
## TP.g.dL.            0.1975223 0.226394 0.223684
## ALB.g.dL.           0.0997524 0.386800 0.370379
## ALT.IU.L.           0.8763600 0.872267 0.858173
## SDH.IU.L.           0.0064601 0.024065 0.011427
```

```
## AST.IU.L.          0.8593764 0.850492 0.843669
## ALP.IU.L.          0.0858100 0.081470 0.069903
## TBA.umol.L.        0.6523868 0.772846 0.773367
## Cholesterol.mg.dL. 0.3825104 0.430367 0.427878
```

Test with leave one out CV for a subset of individuals. For ncomp = 1, the Q2. total is equal to NA.

```
X.sub <- liver.toxicity$gene[liver.toxicity$treatment[, "Dose.Group"] == 1500,
    ]
Y.sub <- liver.toxicity$clinic[liver.toxicity$treatment[, "Dose.Group"] == 1500,
    ]
dim(X.sub)
```

```
## [1]   16 3116
```

```
dim(Y.sub)
```

```
## [1] 16 10
```

```
# with pls
res.pls2 = pls(X.sub, Y.sub, mode = "regression")
perf.pls2 = perf(res.pls2, validation = "loo", criterion = "all", progressBar = FALSE)
perf.pls2$Q2.total
```

```
##  comp 1  comp 2
## 0.08672 0.05018
```

```
perf.pls2$MSEP
```

```
##                    ncomp 1 ncomp 2
## BUN.mg.dL.          0.5806  0.5656
## Creat.mg.dL.        1.0968  1.2913
## TP.g.dL.            1.1665  0.8159
## ALB.g.dL.           1.1538  0.7976
## ALT.IU.L.           0.6321  0.6669
## SDH.IU.L.           1.0525  1.1465
## AST.IU.L.           0.6684  0.7136
## ALP.IU.L.           1.0757  1.1521
## TBA.umol.L.         0.5894  0.4602
## Cholesterol.mg.dL.  0.5461  0.5227
```

```
perf.pls2$R2
```

```
##                    ncomp 1 ncomp 2
## BUN.mg.dL.         0.41319 0.42102
## Creat.mg.dL.       0.16008 0.36672
## TP.g.dL.           0.70332 0.14845
## ALB.g.dL.          0.43995 0.15915
## ALT.IU.L.          0.33238 0.29172
## SDH.IU.L.          0.06180 0.12067
```

```
## AST.IU.L.           0.29280 0.24470
## ALP.IU.L.           0.02385 0.07594
## TBA.umol.L.         0.39480 0.52568
## Cholesterol.mg.dL. 0.43170 0.44769
```

```
# with spls
res.spls2 = spls(X.sub, Y.sub, ncomp = 2, keepX = c(50, 50), keepY = c(5, 8),
    mode = "regression")
perf.spls2 = perf(res.spls2, validation = "loo", criterion = "all", progressBar = FALSE)
perf.spls2$Q2.total
```

```
## comp 1 comp 2
## 0.1403 0.1236
```

```
perf.spls2$MSEP
```

```
##                    ncomp 1 ncomp 2
## BUN.mg.dL.          0.5396  0.5450
## Creat.mg.dL.        1.1296  1.3419
## TP.g.dL.            1.1188  0.5598
## ALB.g.dL.           1.1150  0.4879
## ALT.IU.L.           0.4928  0.5078
## SDH.IU.L.           1.0929  1.1476
## AST.IU.L.           0.5220  0.5438
## ALP.IU.L.           1.2804  1.4075
## TBA.umol.L.         0.4020  0.2280
## Cholesterol.mg.dL.  0.3662  0.2938
```

```
perf.spls2$R2
```

```
##                    ncomp 1 ncomp 2
## BUN.mg.dL.         0.42552 0.41907
## Creat.mg.dL.       0.20886 0.36197
## TP.g.dL.           0.35350 0.40327
## ALB.g.dL.          0.41361 0.48406
## ALT.IU.L.          0.47936 0.46583
## SDH.IU.L.          0.03338 0.05607
## AST.IU.L.          0.44906 0.43054
## ALP.IU.L.          0.25008 0.30300
## TBA.umol.L.        0.57418 0.77024
## Cholesterol.mg.dL. 0.60966 0.68855
```

sPLS and stability of selection

An interesting output that we have added with the function perf is a 'stability frequency'. For each training step (cross validation run) the specified number of variables are selected (**keepX** and **keepY**) on the training data set. Since each training set is different, a different list of selected features is obtained. We can therefore calculate the frequency of selection of of the selected variables.

```
perf.spls$features$stable.X
```

```
## $`comp 1`
## A_42_P620915 A_42_P705413  A_43_P10606  A_43_P14131  A_43_P22616
##          1.0          1.0          1.0          1.0          1.0
##  A_43_P23376 A_42_P675890 A_42_P758454 A_42_P840776  A_43_P17415
##          1.0          0.6          0.6          0.6          0.6
## A_42_P802628 A_42_P578246 A_42_P809565 A_42_P825290  A_43_P10003
##          0.4          0.2          0.2          0.2          0.2
##  A_43_P11570  A_43_P11724
##          0.2          0.2
##
## $`comp 2`
## A_42_P591665 A_42_P505480 A_42_P576823  A_43_P11285 A_42_P678904
##          1.0          0.8          0.8          0.8          0.6
## A_42_P843603 A_42_P477643 A_42_P794613  A_43_P12842
##          0.4          0.2          0.2          0.2
##
## $`comp 3`
## A_42_P622652  A_43_P17889 A_42_P504507 A_42_P606133 A_42_P656513
##          0.6          0.6          0.4          0.4          0.4
## A_42_P466316 A_42_P473074 A_42_P491505 A_42_P563285 A_42_P658758
##          0.2          0.2          0.2          0.2          0.2
## A_42_P668600 A_42_P693789 A_42_P720241 A_42_P757032 A_42_P757296
##          0.2          0.2          0.2          0.2          0.2
## A_42_P760952 A_42_P790250 A_42_P802481  A_43_P10703  A_43_P11409
##          0.2          0.2          0.2          0.2          0.2
##  A_43_P11417  A_43_P11999  A_43_P12859  A_43_P13050  A_43_P13368
##          0.2          0.2          0.2          0.2          0.2
##  A_43_P13563  A_43_P14234  A_43_P15239  A_43_P15503  A_43_P15653
##          0.2          0.2          0.2          0.2          0.2
##  A_43_P16009  A_43_P16147  A_43_P16246  A_43_P16271  A_43_P16637
##          0.2          0.2          0.2          0.2          0.2
##  A_43_P16946  A_43_P17412  A_43_P20298  A_43_P20814  A_43_P21283
##          0.2          0.2          0.2          0.2          0.2
##  A_43_P21538  A_43_P21963  A_43_P22375
##          0.2          0.2          0.2
```

```
# perf.spls$features$stable.Y
```

We also output the final variable selection given the input arguments **keepX** and **keepY** on the full data set, along with the weight values from the loading vector.

```
perf.spls$features$final.X
```

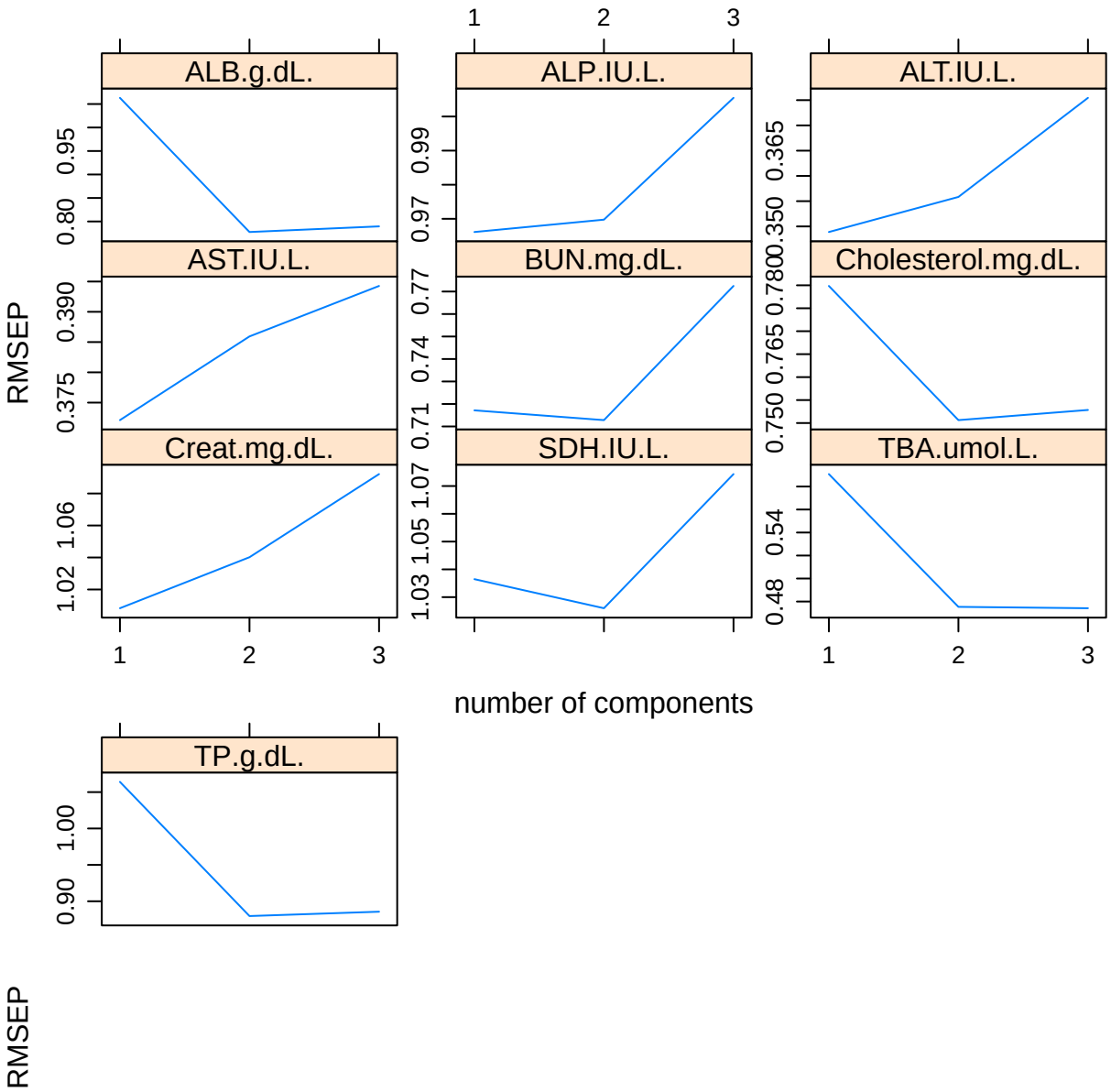
```
## $`comp 1`
##              value.var.X
## A_42_P620915     0.54789
## A_43_P14131      0.53920
## A_43_P23376      0.37728
## A_43_P22616      0.32534
## A_43_P10606      0.27336
```

```
## A_43_P17415      0.23237
## A_42_P705413     0.13863
## A_42_P802628     0.08216
## A_42_P840776     0.06827
## A_42_P675890     0.03932
##
## $`comp 2`
##              value.var.X
## A_42_P591665     -0.7181
## A_42_P576823     -0.5238
## A_43_P11285      -0.3390
## A_42_P678904     -0.2911
## A_42_P505480     -0.1011
##
## $`comp 3`
##              value.var.X
## A_42_P622652     0.78117
## A_42_P656513     0.33316
## A_42_P504507     0.31844
## A_43_P17889      0.24404
## A_42_P467372     0.21493
## A_43_P22375      0.17676
## A_43_P13368      0.13226
## A_42_P757296     0.09641
## A_42_P606133     0.08656
## A_43_P10703      0.07814
```

```
# perf.spls$features$final.Y
```

A plot function is proposed to output the different criteria per response variable.

```
plot(perf.spls, criterion = c("RMSEP"), type = "l")
```



PLS-DA methods

PLS-DA perform a classification task (as opposed to a regression task with PLS). Here the Y response is a factor indicating the class of each individual. Here we take the dose group as a Y factor.

```
data(liver.toxicity)
X <- liver.toxicity$gene
Y <- as.factor(liver.toxicity$treatment[, "Dose.Group"])  # take a factor here
summary(Y)
```

```
##   50  150 1500 2000
##   16   16   16   16
```

Estimating the error rate

In addition to the number of components, the user needs to specify the prediction method to be used with PLS-DA (see ?predict.plsda and ?perf). The criterion that is computed is the classification error rate averaged across all cross-validated runs. Note if M fold cross validation is performed, this result should be averaged across several repetitions (in the example below we only ran it once).

```
res.plsda = plsda(X, Y, ncomp = 10)
tune.plsda = perf(res.plsda, method.predict = "all", validation = "Mfold", folds = 5,
    progressBar = FALSE)

# ! note that the function perf should be rerun several times and the error
# rates obtained below should then be averaged
tune.plsda$error.rate
```

```
##          max.dist centroids.dist mahalanobis.dist
## ncomp 1    0.7038         0.4372           0.4372
## ncomp 2    0.6103         0.4538           0.3923
## ncomp 3    0.4346         0.2795           0.2949
## ncomp 4    0.3269         0.2654           0.1859
## ncomp 5    0.2321         0.2808           0.1705
## ncomp 6    0.2167         0.2487           0.2026
## ncomp 7    0.2795         0.2641           0.2795
## ncomp 8    0.2500         0.2641           0.2949
## ncomp 9    0.2013         0.2795           0.2641
## ncomp 10   0.2013         0.2474           0.2013
```

Example with leave-one-out cross-validation

```
res.plsda2 = plsda(X, Y, ncomp = 10)
tune.plsda2 = perf(res.plsda2, method.predict = "all", validation = "loo", progressBar = FALSE)

# ! note that the function perf should be rerun several times and the error
# rates obtained below should then be averaged
tune.plsda2$error.rate
```

```
##          max.dist centroids.dist mahalanobis.dist
## ncomp 1    0.8281         0.4375           0.4375
## ncomp 2    0.4062         0.4062           0.3750
## ncomp 3    0.2656         0.2969           0.1875
## ncomp 4    0.3438         0.3125           0.2500
## ncomp 5    0.2812         0.3438           0.2031
## ncomp 6    0.2344         0.3125           0.1562
```

```
## ncomp 7    0.2344          0.2656          0.2188
## ncomp 8    0.2188          0.2500          0.2344
## ncomp 9    0.1875          0.2500          0.2812
## ncomp 10   0.1875          0.2656          0.2344
```

The classification error rate might reach a minimum when a sufficient number of components is included in the model. This output is especially useful when using the sparse PLS-DA, for example if we want to assess the performance of sPLS-DA with a specified number of variables to select (**keepX**).

Form our experience, the prediction distance 'max.dist' seems the outperform the other distances in most cases. This is the distance that we choose in this example:

```
res.splsda = splsda(X, Y, ncomp = 3, keepX = c(50, 40, 20))
perf.splsda = perf(res.splsda, method.predict = "max.dist", validation = "Mfold",
    folds = 5, progressBar = FALSE)

# ! note that the function perf should be rerun several times and the error
# rates obtained below should then be averaged
perf.splsda$error.rate
```

```
##         max.dist
## ncomp 1   0.5641
## ncomp 2   0.2487
## ncomp 3   0.1859
```

Example with leave-one-out cross-validation

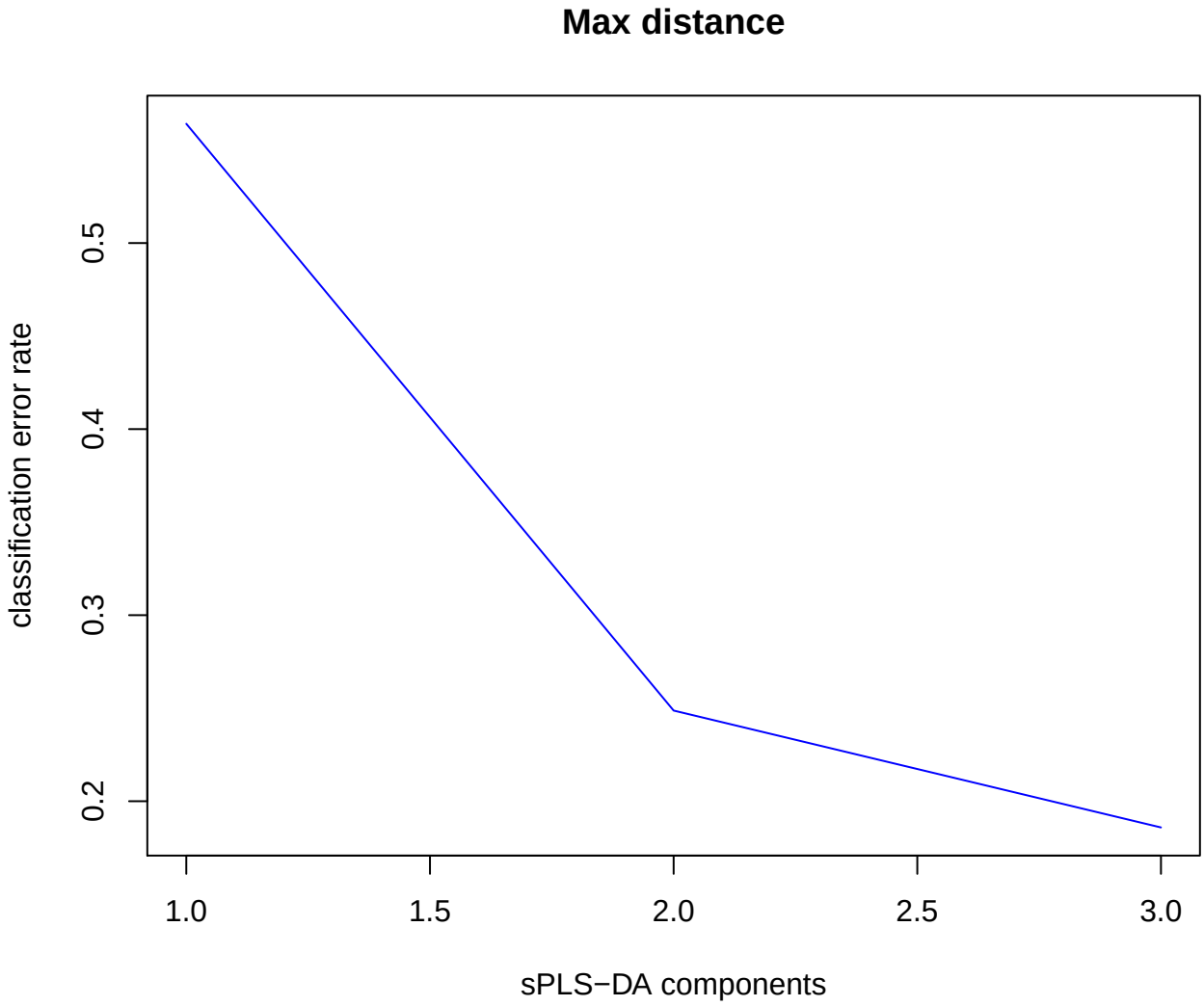
```
res.splsda2 = splsda(X, Y, ncomp = 3, keepX = c(50, 40, 20))
perf.splsda2 = perf(res.splsda2, method.predict = "max.dist", validation = "loo",
    progressBar = FALSE)

# ! note that the function perf should be rerun several times and the error
# rates obtained below should then be averaged
perf.splsda2$error.rate
```

```
##         max.dist
## ncomp 1   0.7500
## ncomp 2   0.1094
## ncomp 3   0.1719
```

As with the sPLS we can also output the final variable selection (on the full data set) as well as the variable selected repeatedly across the cross-validation runs:

```
# perf.splsda$features$stable perf.splsda$features$final
```

```
matplot(perf.splsda$error.rate, type = 'l', col = 'blue', xlab = 'sPLS-DA components',
        ylab = 'classification error rate', main = 'Max distance')
```



Tuning the number of variables to select with sPLS-DA

Below is an example of code to run to tune the number of variables to select with sPLS-DA. There are few things to take into account during the tuning step: - vary the number of variables to select - this code below only assesses the model for one cross-validation procedure. **It should be averaged across several runs** - the tuning should be performed iteratively, 1 component at a time.

Note: these computations may take a long time to run!

For component 1:

```
n <- nrow(X)
M <- 5
list.keepX <- c(seq(10, 200, 10))
error1 <- vector(length = length(list.keepX))
names(error1) <- list.keepX

ncomp = 1
for (i in 1:length(list.keepX)) {
    model <- splsda(X, Y, ncomp = ncomp, keepX = c(list.keepX[i]))
    error1[i] <- perf(model, method.predict = "max.dist", validation = "Mfold",
```

```
        folds = M, progressBar = FALSE)$error.rate
}

error1
```

```
##     10     20     30     40     50     60     70     80     90    100
## 0.5923 0.5615 0.5936 0.6423 0.6103 0.5436 0.6577 0.5744 0.5154 0.5782
##    110    120    130    140    150    160    170    180    190    200
## 0.6231 0.5936 0.5756 0.6731 0.5936 0.5321 0.6103 0.5487 0.7179 0.6551
```

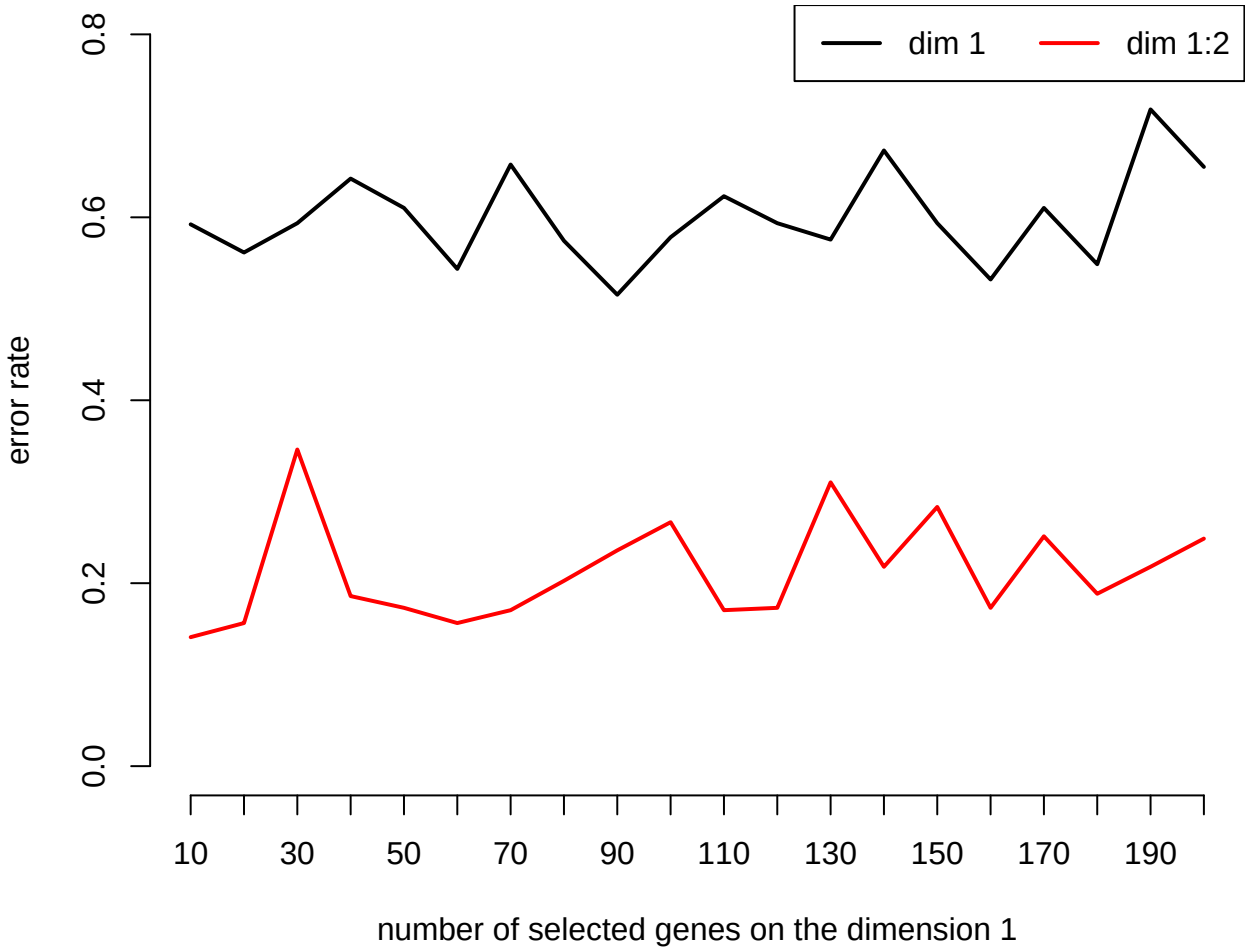
Given the previous results, we can decide to set the first keepX value to, say, 20 and continue to tune the next component:

```
ncomp = 2
error2 <- vector(length = length(list.keepX))
names(error2) <- list.keepX

# tuned value for the first component
keepX1 = 20


for (i in 1:length(list.keepX)) {
    model <- splsda(X, Y, ncomp = ncomp, keepX = c(keepX1, list.keepX[i]))
    error2[i] <- t(perf(model, method.predict = "max.dist", validation = "Mfold",
        folds = M, progressBar = FALSE)$error.rate)[ncomp]
}
error2
```

```
##     10     20     30     40     50     60     70     80     90    100
## 0.1410 0.1564 0.3462 0.1859 0.1731 0.1564 0.1705 0.2026 0.2359 0.2667
##    110    120    130    140    150    160    170    180    190    200
## 0.1705 0.1731 0.3103 0.2179 0.2833 0.1731 0.2513 0.1885 0.2179 0.2487
```

Plot the error rate for either component 1, or the combination of component 1 and 2:

```
matplot(cbind(error1, error2), type = "l", axes = FALSE, xlab = "number of selected genes on the dimens
    ylab = "error rate", lwd = 2, lty = 1, col = 1:2, ylim = c(0, 0.8))
axis(1, c(1:length(list.keepX)), labels = list.keepX)
axis(2)
legend("topright", lty = 1, lwd = 2, legend = c("dim 1", "dim 1:2"), horiz = TRUE,
    col = 1:2)
```


dim 1
dim 1:2
error rate
0.0
0.2
0.4
0.6
0.8
10
30
50
70
90
110
130
150
170
190
number of selected genes on the dimension 1