

Tutorial 2

PC and PLS Regression using the pls package

PCR and PLSR can be carried out easily using the package **pls**. Install and active this package. The use is quite straightforward. The optimal number of components is arrived at using leave-one-out cross validation.

Exercise 1: Polyethylene Terephthalate Data

Consider the Polyethylene Terephthalate data found in the data set **yarn**, included in the package **pls**. The data were obtained from a calibration study of polytheylene terephthalate yarns, which are used for textile manufacture and other industrial purposes. Raman near-infrared (NIR) spectroscopy has recently become an important tool in the pharmaceutical and semiconductor industries for investigating structural information on polymers.

The Y variable is the density (measured in kg/m^3) of the yarn. There are 268 explanatory variables, selected from the NIR spectrum of the yarn.

There are 28 observations, so that the X matrix is 28×268 (if centred variables are considered so that the intercept is 0).

Remember to centre (and standardise) the data where appropriate.

1. Perform a principal component analysis on the X variables. How many eigenvalues greater than 1 are there?

You can do a PCA on the X variables (which are the NIR variables) by:

```
> pcxyarn = prcomp(yarn$NIR,center=TRUE,scale=TRUE)
```

Note that the eigenvalues are the *variances* of the principal components and hence the *squares* of the standard deviations. To get the eigenvalues, try:

```
> ev = pcxyarn$sdev^2
> ev
[1] 1.369756e+02 8.278146e+01 4.414949e+01 2.646324e+00 6.006888e-01
3.409703e-01
[7] 2.317032e-01 7.035091e-02 6.121514e-02 4.929247e-02 2.760041e-02
1.866255e-02
[13] 1.048080e-02 8.622384e-03 6.670331e-03 4.678203e-03 4.179166e-03
3.241186e-03
[19] 2.195114e-03 1.725305e-03 1.456335e-03 1.240840e-03 7.721071e-04
6.085844e-04
[25] 3.988832e-04 2.611062e-04 1.453094e-04 2.499623e-28
```

so we see that even though there are 28 variables, there are only 4 eigenvalues greater than 1.

2. Perform a PCR (principal component regression). Try it with 1,3,4,20 components. Is PCR useful here? If so, how many components would you recommend.

From the **pls** package:

```
> pcreg = pcr(density~NIR,ncomp=6,data=yarn)
> summary(pcreg)
Data: X dimension: 28 268
Y dimension: 28 1
Fit method: svdpc
Number of components considered: 6
TRAINING: % variance explained
```

	1 comps	2 comps	3 comps	4 comps	5 comps	6 comps
X	52.17	98.60	99.47	99.70	99.88	99.97
density	5.50	98.15	99.40	99.58	99.95	99.99

uses 6 components. As we can see, already after the second component, 98% of the total sum of squares is explained, but the leading principal component doesn't explain very much of the **density** sum of squares. Even though the first principal component of NIR has a much larger eigenvalue than the others, it does not help much for prediction of **density**; the second principal component is much more important.

3. Perform a PLSR (partial least squares regression). How many PLSR components are needed? (Answer: the final shape of the coefficient estimates can already be discerned by 3 components - that is, the coefficient for the significant components do not change substantially as further components are added. A useful representation is given by 4 components. Is it useful to add further components?)

```
> pcreg = plsreg(density~NIR,ncomp=6,data=yarn)
> summary(pcreg)
Data: X dimension: 28 268
Y dimension: 28 1
Fit method: kernelpls
Number of components considered: 6
TRAINING: % variance explained
```

	1 comps	2 comps	3 comps	4 comps	5 comps	6 comps
X	46.83	98.38	99.46	99.67	99.85	99.97
density	98.12	98.25	99.64	99.97	99.99	99.99

We can see that this is much better than PCR; even after the first component we have explained 98% of the total sum of squares by the model. At the first pass, the PLSR algorithm selects the linear component of the X variables most strongly correlated with the response Y . In this problem, this leads to a pronounced improvement.

4. Try PCR, PLSR again. Centre the variables and compare what happens if you standardise the X variables and if you do not standardise the X variables.
5. How are the results affected if you neither standardise nor centre the variables?

Exploring pls further

We explore the **pls** package further using three data sets, the **yarn** data set described above, the *oliveoil* data set, which contains 5 quality measurements (**chemical**), 6 sensory variables (**sensory**) made on 16 olive oil samples. We concentrate mostly on the **gasoline** data set which contains the octane number **octane** and the NIR spectra (**NIR**) of 60 gasoline samples.

```
data("yarn")
data("oliveoil")
data("gasoline")
```

We use the first 50 gasoline observations for training and the remaining 10 to test the predictor.

```
gasTrain <- gasoline[1:50, ]
gasTest <- gasoline[51:60, ]
gas1 <- plsr(octane ~ NIR, ncomp = 10, data = gasTrain, validation = "LOO")
summary(gas1)
```

The LOO validation is leave-one-out; a predictor computed using 49 of the 50; the mean squared error determined by the prediction of the remaining. CV denotes the ordinary CV estimate, adjCV the bias corrected estimate; these should be approximately the same for LOO. RMSEP stands for root mean squared error of the predictor; let us see how it depends on the number of components.

```
plot(RMSEP(gas1), legendpos = "topright")
plot(gas1, ncomp = 2, asp = 1, line = TRUE)
plot(gas1, plottype = "scores", comps = 1:3)
```

The explained variables can be extracted explicitly with **explvar**:

```
explvar(gas1)
```

The loading plot is extremely useful:

```
plot(gas1, "loadings", comps = 1:2, legendpos = "topleft",
     labels = "numbers", xlab = "nm")
abline(h = 0)
```

The aim of the fitted model is to predict future behaviour:

```
predict(gas1, ncomp = 2, newdata = gasTest)
```

and since we actually have the values we can calculate the RMSEP for the test set.

```
RMSEP(gas1, newdata = gasTest)
```

the `plsr` function returns a model of class `mvr`:

```
dens1 <- plsr(density ~ NIR, ncomp = 5, data = yarn)
```

If the response term of the model is a matrix, a multi-response model is fitted:

```
dim(oliveoil$sensory)
plsr(sensory ~ chemical, data = oliveoil)
```

The `update` function is useful for adding explanatory variables:

```
trainind <- which(yarn$train == TRUE)
dens2 <- update(dens1, subset = trainind)
dens3 <- update(dens1, ncomp = 10)
```

If `scale` is `TRUE`, then each variable is standardised. If it is a numeric vector, each variable is divided by the corresponding number.

```
olive1 <- plsr(sensory ~ chemical, scale = TRUE, data = oliveoil)
```

The function `msc` produces a *multiplicative scatter corrected* matrix, which is sometimes more useful. Does it give a better predictor?

```
gas2 <- plsr(octane ~ msc(NIR), ncomp = 10, data = gasTrain)
predict(gas2, ncomp = 3, newdata = gasTest)
```

We can do cross validation with ‘leave 10 out’.

```
gas2.cv <- crossval(gas2, segments = 10)
plot(MSEP(gas2.cv), legendpos = "topright")
summary(gas2.cv, what = "validation")
```

The *plotype* command enables us to plot several regressions simultaneously, each with a different number of explanatory variables.

```
plot(gas1, plotype = "coef", ncomp = 1:3, legendpos = "bottomleft",  
     labels = "numbers", xlab = "nm")
```

Prediction is straightforward using the `predict` command.

```
predict(gas1, ncomp = 2:3, newdata = gasTest[1:5, ])
```

```
predict(gas1, comps = 2, newdata = gasTest[1:5, ])
```

and `drop` is useful for comparing a smaller model with a larger model.

```
drop(predict(gas1, ncomp = 2:3, newdata = gasTest[1:5, ]))
```

Predictions can be plotted using the `predplot` command:

```
predplot(gas1, ncomp = 2, newdata = gasTest, asp = 1, line = TRUE)
```

The fit functions can be called directly if, for example, one does not want the overhead of formula and data handling repeating fits.

```
X <- gasTrain$NIR  
Y <- gasTrain$octane  
ncomp <- 5  
cvPreds <- matrix(nrow = nrow(X), ncol = ncomp)  
for (i in 1:nrow(X)) {  
  fit <- simpls.fit(X[-i, ], Y[-i], ncomp = ncomp, stripped = TRUE)  
  cvPreds[i, ] <- (X[i, ] - fit$Xmeans) %*% drop(fit$coefficients) +  
    fit$Ymeans  
}  
sqrt(colMeans((cvPreds - Y)^2))
```