


News

The Newsletter of the R Project

Volume 6/3, August 2006

Editorial

by Ron Wehrens and Paul Murrell

Welcome to the third issue of R News for 2006, our second special issue of the year, this time with a focus on uses of R in Chemistry. Thanks go to our guest editor, Ron Wehrens, for doing all of the heavy lifting to get this issue together. Ron describes below the delights that await in the following pages. Happy reading!

Paul Murrell

The University of Auckland, New Zealand

paul.murrell@R-project.org

R has become the standard for statistical analysis in biology and bioinformatics, but it is also gaining popularity in other fields of natural sciences. This special issue of R News focuses on the use of R in chemistry. Although a large number of the Bioconductor packages can be said to relate to chemical sub-disciplines such as biochemistry or analytical chemistry, we have deliberately focused on those applications with a less obvious bioinformatics component.

Rather than providing a comprehensive overview, the issue gives a flavour of the diversity of applications. The first two papers focus on fitting equations that are derived from chemical knowledge, in this case with nonlinear regression. Peter Watkins and Bill Venables show an example from chromatography, where the retention behaviour of carboxylic acids is modelled. Their paper features

some nice examples of how to initialize the optimization. In the next paper, Johannes Ranke describes the **drfit** package for fitting dose-response curves.

The following three papers consider applications that are more analytical in nature, and feature spectral data of various kinds. First, Bjørn-Helge Mevik discusses the **pls** package, which implements PCR and several variants of PLS. He illustrates the package with an example using near-infrared data, which is appropriate, since this form of spectroscopy would not be used today but for the existence of multi-variate calibration techniques. Then, Chris Fraley and Adrian Raftery describe several chemical applications of the **mclust** package for model-based clustering. The forms of spectroscopy here yield images rather than spectra; the examples focus on segmenting microarray images and dynamic magnetic resonance images. Ron Wehrens and Egon Willighagen continue with a paper describing self-organising maps for large databases of crystal structures, as implemented in the package **wccsom**. To compare the spectral-like descriptors of crystal packing, a specially devised similarity measure has to be used.

The issue concludes with a contribution by Rarajshi Guha on the connections between R and the Chemistry Development Kit (CDK), another open-source project that is rapidly gaining widespread popularity. With CDK, it is easy to generate descriptors of molecular structure, which can then be used in R for modelling and predicting properties. The paper includes a description of the **rcdk** package, where

Contents of this issue:

Editorial	1
Non-linear regression for optimising the separation of carboxylic acids	2
Fitting dose-response curves from bioassays and toxicity testing	7

The pls package	12
Some Applications of Model-Based Clustering in Chemistry	17
Mapping databases of X-ray powder patterns	24
Generating, Using and Visualizing Molecular Information in R	28

the key point is the connection between R and Java (which underlies CDK).

Ron Wehrens

*Institute for Molecules and Materials
Analytical Chemistry
The Netherlands
R.Wehrens@science.ru.nl*

Non-linear regression for optimising the separation of carboxylic acids

by Peter Watkins and Bill Venables

In analytical chemistry, models are developed to describe a relationship between a response and one or more stimulus variables. The most frequently used model is the linear one where the relationship is linear in the parameters that are to be estimated. This is generally applied to instrumental analysis where the instrument response, as part of a calibration process, is related to a series of solutions of known concentration. Estimation of the linear parameters is relatively simple and is routinely applied in instrumental analysis. Not all relationships though are linear with respect to the parameters. One example of this is the Arrhenius equation which relates the effect of temperature on reaction rates:

$$k = A \exp(-E_a/RT) \times \exp(\epsilon) \quad (1)$$

where k is the rate coefficient, A is a constant, E_a is the activation energy, R is the universal gas constant, and T is the temperature in degrees Kelvin. As k is the measured response at temperature T , A and E_a are the parameters to be estimated. The last factor indicates that there is an error term, which we assume is multiplicative on the response. In this article we will assume that the error term is normal and homoscedastic, that is, $\epsilon \sim N(0, \sigma^2)$

One way to find estimates for A and E_a is to transform the Arrhenius equation by taking logarithms of both sides. This converts the relationship from a multiplicative one to a linear one with homogeneous, additive errors. In this form linear regression may be used to estimate the coefficients in the usual way.

Note that if the original error structure is not multiplicative, however, and the appropriate model is, for example, as in the equation

$$k = A \exp(-E_a/RT) + \epsilon \quad (2)$$

then taking logarithms of both sides does not lead to a linear relationship. While it may be useful to ignore this as a first step, the optimum estimates can only be obtained using non-linear regression techniques, that is by least squares on the original scale and not in the logarithmic scale. Starting from initial values for the unknown parameters, the estimates are iteratively refined until, it is hoped, the process converges to the maximum likelihood estimates.

This article is intended to show some of the powerful general facilities available in R for non-linear regression, illustrating the ideas with simple, yet important non-linear models typical of those in use in Chemometrics. The particular example on which we focus is one for the response behaviour of a carboxylic acid using reverse-phase high performance liquid chromatography and we use it to optimise the separation of a mixture of acids.

Non-linear regression in general is a very unstructured class of problems as the response function of the regression may literally be any function at all of the parameters and the stimulus variables. In specific applications, however, certain classes of non-linear regression models are typically of frequent occurrence. The general facilities in R allow the user to build up a knowledge base in the software itself that allows the fitting algorithm to find estimates for initial values and to find derivatives of the response function with respect to the unknown parameters automatically. This greatly simplifies the model fitting process for such classes of models and usually makes the process much more stable and reliable.

The working example

Aromatic carboxylic acids are an important class of compounds since many are pharmacologically and biologically significant. Thus it is useful to be able to separate, characterise and quantify these types of compounds. One way to do this is with chromatography, more specifically, reverse phase high performance liquid chromatography (RP-HPLC). Due to the ionic nature of these compounds, analysis by HPLC can be complicated as the hydrogen ion concentration is an important factor for separation of these compounds. [Waksmundzka-Hajnos \(1998\)](#) reported a widely used equation that models the separation of monoprotic carboxylic acids (i.e. containing a single hydrogen ion) depending on the hydrogen ion concentration, $[H^+]$. This is given by

$$k = \frac{(k_{-1} + k_0([H^+]/K_a))}{(1 + [H^+]/K_a)} + \epsilon \quad (3)$$

where k is the capacity factor, k_0 and k_{-1} are the k values for the non-ionised and ionised forms of the

carboxylic acid, and K_a is the acidity constant. In this form, non-linear regression can be applied to Equation 3 to estimate the parameters, k_0 , k_{-1} and K_a . Deming and Turoff (1978) reported HPLC measurements for four carboxylic acids containing a single hydrogen ion; benzoic acid (BA), *o*-aminobenzoic acid (OABA), *p*-aminobenzoic acid (PABA) and *p*-hydroxybenzoic acid (HOBA).

The R data set we use here has a stimulus variable named pH and four responses named BA, OABA, PABA and HOBA, the measured capacity factors for the acids at different pH values. The data set is given as an appendix to this article.

Initial values

While non-linear regression is appropriate for the final estimates, we can pay less attention to the error structure when trying to find initial values. In fact we usually disregard it and simply manipulate the equation to a form where we can find initial values by simple approximation methods. One way to do this for this example, suggested to us by Petra Kuhnert, is to multiply the equation by the denominator and re-arrange to give

$$k[H^+] \approx k_{-1}K_a + kK_a + [H^+]k_0 = \theta + kK_a + [H^+]k_0 \quad (4)$$

where the three unknowns on the right hand side are $\theta = k_{-1}K_a$, K_a and k_0 . The initial values can be found by ignoring the error structure and simply regressing the artificial response, $k[H^+]$ on the notional 'stimulus' variables k and $[H^+]$, with an intercept term. The initial values for k_{-1} , k_0 and K_a then follow simply.

We can in fact do slightly better than this by noting that if the value for the parameter K_a were known, the original equation is then linear in k_{-1} and k_0 . In principle, then, we could use the approximate idea to obtain an initial value for K_a only and then use the entire data set to find initial values for the other two parameters by linear regression with the original equation. Intuitively we might expect this to yield slightly better initial estimates as while the value for K_a is still somewhat doubtful, at least those for k_{-1} and k_0 do in fact respect the appropriate error structure, but sadly there is no guarantee that this will be the case.

Partially linear models

The non-linear fitting function supplied with R is `nls`. Normally the user has to supply initial values for all unknown parameters. However if the linear regression has a form like the one above where, if some of the parameters are known, the regression becomes linear in the others, a special algorithm allows us to capitalise on this fact. The benefit is two-fold: firstly, we need only supply initial values for some of the parameters and secondly, the algorithm is typically more stable than it would otherwise be.

Parameters in the first set, (for our example, just K_a) are called the "non-linear parameters" while the second set are the "linear parameters" (though a better term might be "conditionally linear parameters"). The so-called "partially linear" fitting algorithm requires only that initial values be supplied for the non-linear parameters. To illustrate the entire fitting process, we note that when BA is the response, a reasonable initial value is $K_a = 0.0001$. Using this we can fit the regression as follows:

```
> tmp <- transform(ba, H = 10^(-pH))
> ba.nls <- nls(BA ~ cbind(1, H/Ka)/(1 + H/Ka),
  data = tmp, algorithm = "plinear",
  start = c(Ka = 0.0001), trace = T)
13.25806 : 0.000100 3.531484 54.819291
7.180198 : 4.293692e-05 5.890920e-01 4.197178e+01
1.441950 : 5.600297e-05 1.635335e+00 4.525067e+01
1.276808 : 5.906698e-05 1.831871e+00 4.597552e+01
1.276494 : 5.920993e-05 1.840683e+00 4.600901e+01
1.276494 : 5.921154e-05 1.840783e+00 4.600939e+01
> coef(ba.nls)
      Ka      .lin1      .lin2
5.921154e-05 1.840783e+00 4.600939e+01
```

The first step is for convenience only. Notice that the regression function is specified as a matrix whose columns are, in general, functions of the non-linear parameters. The coefficients for this non-linear "model matrix" are then the estimates of the linear parameters.

The trace output shown above shows at each step the current residual sum of squares and the parameter estimates, beginning with the non-linear parameters. As the model specification makes no explicit reference to the linear parameters, the estimates are named using a simple naming convention, as shown in the names of the final coefficient vector.

The trace output shows that the initial value is not particularly good and the iterative process is struggling, somewhat.

Self-starting non-linear regressions

The idea behind a self-starting non-linear regression model is that we supply to the fitting algorithm enough information for the process to generate a starting value from the data itself. This involves two steps. Firstly we need to write an *initial value routine* which calculates the initial values from the data set. Secondly we need to generate the self-starting object itself, which will use this initial value routine.

We begin with the initial value routine. As this will be called by the fitting algorithm and not directly by the user, it has to use a somewhat obscure convention for both the parameter list it uses and the way it supplies its values. This convention initially appears to be obscure, but it becomes less so with time. Such an initial value routine can be defined for this kind of regression model as follows:

```

SSba.init <- function(mCall, data, LHS) {
#
# k ~ (k_1 + k_0*H/Ka)/(1 + H/Ka); H = 10^(-pH)
#
  H <- 10^(-eval(mCall[["pH"]], data))
  k <- eval(LHS, data)

  Ka <- as.vector(coef(lsfitt(cbind(H, -k),
    H * k, int = TRUE))[3])
  b <- coef(nls(k ~ cbind(1, H/Ka)
    /(1 + H/Ka),
    data = data.frame(k = k, H = H),
    algorithm = "plinear",
    start = c(Ka = Ka)))
  names(b) <- mCall[c("Ka", "k_1", "k_0")]
  b
}

```

The initial comment is a reminder of how the model will be specified in stage two of the process, and is optional. The way the parameters are accessed is conventional.

This initial value function actually goes a step further and fits the non-linear regression itself using the plinear algorithm, so the initial values should be very good indeed! This is the simplest way to capitalise both on the self-starting model and using the plinear model along the way. The model fitting process at the second stage should then converge in one iteration, which can act as a check on the process.

Note that the value supplied by the function is a vector with names. These names are not necessarily the same as in the defining formula, so the final step of giving names to the value object must again do so in a conventional way, which will ensure that actual names used by the user of the self-starting regression will be correctly specified. In other words, the choice of names for the parameters remains open to the user and is not fixed in advance.

To specify a self-starting model, we now proceed as follows:

```

SSba <- selfStart( ~ (k_1 + k_0*10^(-pH)/Ka)
  /(1 + 10^(-pH)/Ka),
  initial = SSba.init,
  parameters = c("Ka", "k_1", "k_0"),
  template = function(pH, k_1, k_0, Ka) {})

```

The first argument is a one-sided formula giving the response function. This now has to respect the name choices used in the initial value routine, of course. The remaining arguments are the initial value routine itself, the parameters as a character vector and a "function template", that is, a dummy function that specifies, in effect, just the argument list. The self-start function itself supplies the innards of this function, so at the end of the call, SSba is a function with this argument list that may be used to specify the right hand side of a non-linear regression.

Fitting the regressions is now a simple task. We fit all four as follows:

```
ba.SS <- nls(BA ~ SSba(pH, k_1, k_0, Ka),
```

```

  data = ba, trace = T)
oaba.SS <- update(ba.SS, OABA ~ .)
paba.SS <- update(ba.SS, PABA ~ .)
hoba.SS <- update(ba.SS, HOBA ~ .)

```

We have suppressed the trace output, but in fact all four (appear to) converge in a single step. Of course the hard work is done inside the initial value function.

Comparison with published values

We can compare the estimates we get from this rather limited data set with published values, at least for the acidity constant, K_a . These values can be found in [Morrison and Boyd \(1993\)](#).

```

> Kas <- sapply(list(ba = ba.SS, oaba = oaba.SS,
  paba = paba.SS, hoba = hoba.SS), coef)[1,]
> signif(Kas, 4)
      ba      oaba      paba      hoba
5.921e-05 6.449e-06 9.631e-06 2.397e-05
> pKas <- scan(quiet = TRUE)
1: 5.8e-05 1.6e-05 1.4e-05 2.6e-05
5:
> KAvalues <- cbind(Calculated = Kas,
  Published = pKas)
> signif(KAvalues, 2)
      Calculated Published
ba      5.9e-05   5.8e-05
oaba    6.4e-06   1.6e-05
paba    9.6e-06   1.4e-05
hoba    2.4e-05   2.6e-05

```

In most cases the agreement is quite good.

Thus, the agreement between K_a values suggests that the calculated parameter estimates can be used for modelling the HPLC retention behaviour of the carboxylic acids. To verify this, the predicted values can be compared with the experimental data. Figure 1 shows the predicted and measured data as a function of $[H^+]$ or $pH = -\log_{10}[H^+]$.

To display the fitted self-starting functions we evaluate them on a finer scale of pH values than we have in the observations themselves. This can be done using the predict function in R, but we find it useful here to evaluate the predictions directly using eval, as we needed to use in the initial value routine.

```

form <- Quote((k_1 + k_0*10^(-pH)/Ka)/
  (1 + 10^(-pH)/Ka))
## evaluate on a fine grid of pH values

at <- function(x, m) c(x, as.list(coef(m)))
pHData <- list(pH = seq(3.5, 6, len = 250))
pHData <- transform(pHData,
  ba = eval(form, at(pHData, ba.SS)),
  oaba = eval(form, at(pHData, oaba.SS)),
  paba = eval(form, at(pHData, paba.SS)),
  hoba = eval(form, at(pHData, hoba.SS)))

## first plot the fitted lines
with(pHData,

```

```

matplot(pH, cbind(ba, oaba, paba, hoba),
        col = 16:19, type = "l", lty = 1,
        ylab = "Capacity factor",
        main = "Retention behaviour")

## then add the observed points
with(ba,
     matpoints(pH, cbind(BA, OABA, PABA, HOBA),
              col = 16:19, pch = 16:19, cex = 0.8))

legend(5.5, 35, c("BA", "OABA", "PABA", "HOBA"),
      text.col = 16:19, col = 16:19, lty = 1,
      pch = 16:19, bty = "n")

```

Notice how `eval` may be used to evaluate a quoted expression, using a list with names as the second argument to provide values for the variables and parameters.

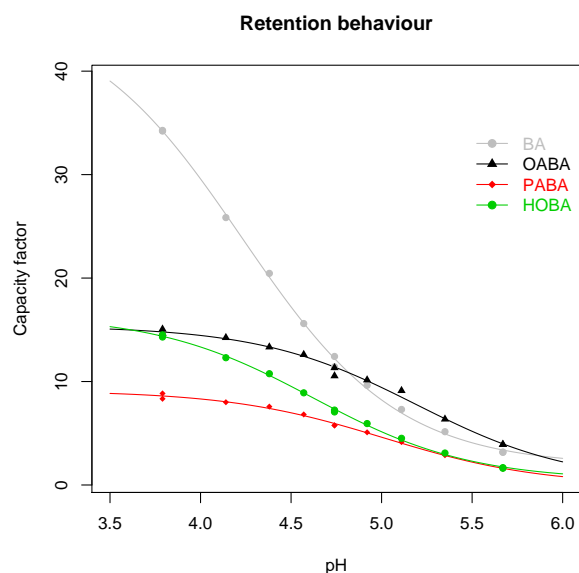


Figure 1: Plot of capacity factor for selected aromatic carboxylic acids against pH

Figure 1 shows that the fit of the predicted values from the self-starting models appears to be quite good with the experimental data.

Diagnostics

The residual plot (predicted - measured *vs* predicted) (Figure 2) suggests that there may be some outliers. This is particularly notable for BA and OABA.

Further presentation of results

A 'window diagram' is a graphical method for showing chromatographic data (Cooper and Hurtubise, 1985), and is suitable for visualizing the degree of chromatographic separation. The degree of separation can be optimised by maximising the selectivity, α , of the most difficult to resolve peak pairs as a function

of pH. The selectivity factor, α_{ij} , is defined by:

$$\alpha_{ij} = k_i/k_j \quad (5)$$

where k_i and k_j are the respective capacity factors. For their data, Deming and Turoff (1978) ensured that α was greater than or equal to 1. In the cases where α was found to be less than 1 then the reciprocal was taken and used for α . This amounts to defining α_{ij} as

$$\alpha_{ij} = \max(k_i, k_j) / \min(k_i, k_j) \quad (6)$$

```

alpha <- function(ki, kj) pmax(ki,kj)/pmin(ki,kj)
pHData <- transform(pHData,
  aB0 = alpha( ba, oaba),
  aBP = alpha( ba, paba),
  aBH = alpha( ba, hoba),
  aOP = alpha(oaba, paba),
  aOH = alpha(oaba, hoba),
  aPH = alpha(paba, hoba))

```

The window diagram (Figure 3) can be produced by plotting the selectivity factor, α_{ij} , as a function of pH.

```

with(pHData,
     matplot(pH, cbind(aB0, aBP, aBH, aOP, aOH, aPH),
            type = "l", col = 16:21, lty = 1:6,
            main = "Window diagram",
            ylab = "Selectivity factor"))
abline(v = 4.5, lty = 4)

leg.txt <-
  c("aB0", "aBP", "aBH", "aOP", "aOH", "aPH")
legend(5.5, 4.5, leg.txt, col = 16:21,
      text.col = 16:21, lty = 1:6, bty = "n")

```

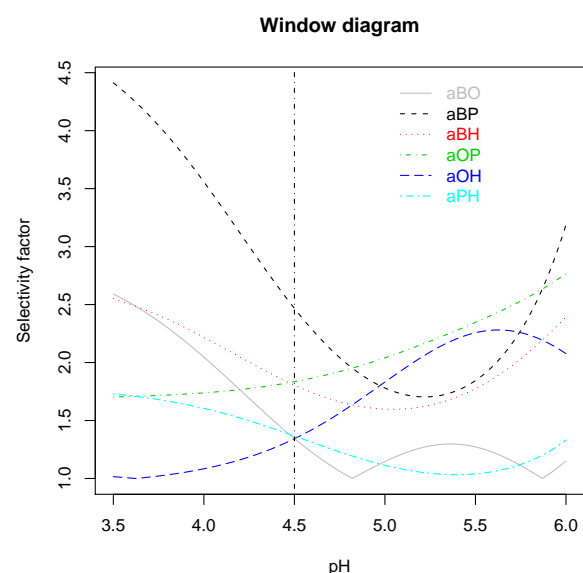


Figure 3: Window diagram for all six combinations of carboxylic acids

The conditions that give α as unity represent a minimum performance for the chromatographic separation while separations with $\alpha > 1$ represent better separation for that pair of compounds. Thus, we

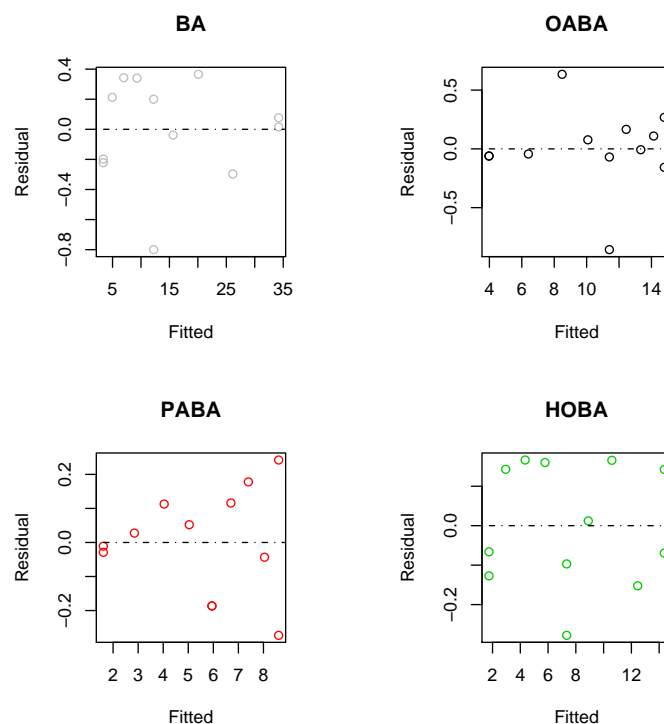


Figure 2: Residual plots for non-linear regression for BA, OABA, PABA and HOBA

seek the point where the best separation occurs for the worst separated compounds. Inspection of Figure 3 shows that this occurs at pH 4.5.

Final remarks

In conclusion, non-linear regression has been used to model the retention behaviour of four related carboxylic acids using Equation 1. Using the predicted values for each acid, the optimum point for separation of the four compounds was identified and used to find the optimal pH.

The example is simple, but sufficient to show the power of non-linear regression in Chemometric applications, and, more importantly, the value of investing some time and effort to produce self-starting model functions for specific classes of regressions. The example also shows some of the power and flexibility of R itself to investigate the data, to fit the models, to check the diagnostics, to predict from the estimates and to present the results.

Bibliography

H. A. Cooper and R. J. Hurtubise. Solubility parameter and window diagram optimization methods for the reversed-phase chromatographic optimization of complex hydroxyl aromatic mixtures. *J. Chrom. A*, 328, 81–91, 1985. 5

S. N. Deming and M. L. H. Turoff. Optimisation of reverse-phase liquid chromatographic separation of weak organic acids. *Anal. Chem.*, 58, 546–548, 1978. 3, 5, 7

R. T. Morrison and R. N. Boyd. *Organic Chemistry*. Allyn and Bacon, Newton, Mass, 5th edition, 1993. 4

M. Waksmundzka-Hajnos. Chromatographic separations of aromatic carboxylic acids. *J. Chrom. B*, 717, 93–118, 1998. 2

Peter Watkins
CSIRO Food Science Australia
peter.watkins@csiro.au

Bill Venables
CSIRO Mathematical and Information Sciences
bill.venables@csiro.au

Appendix

The data set used for this article is shown in the table. It is taken from [Deming and Turoff \(1978\)](#) where it was published in retention times. For this article, the data has been converted to their corresponding capacity factors.

pH	BA	OABA	PABA	HOBA
3.79	34.21	15.06	8.85	14.30
3.79	34.27	14.64	8.33	14.52
4.14	25.85	14.24	8.00	12.30
4.38	20.46	13.33	7.58	10.76
4.57	15.61	12.61	6.82	8.91
4.74	12.42	11.33	5.76	7.24
4.74	11.42	10.55	5.76	7.06
4.92	9.64	10.15	5.09	5.94
5.11	7.30	9.12	4.15	4.52
5.35	5.15	6.36	2.88	3.09
5.67	3.18	3.92	1.60	1.68
5.67	3.18	3.92	1.58	1.62

Fitting dose-response curves from bioassays and toxicity testing

by Johannes Ranke

Introduction

During the development of new chemicals, but also in risk assessment of existing chemicals, they have to be characterized concerning their potential to harm biological organisms. Characterizing chemicals according to this potential has many facets and requires various types of experiments. One of the most important types is the dose-response experiment.

In such experiments, the responses of biological organisms to different doses¹ are observed in a quantitative way. Examples of the observed variables (endpoints of toxicity) are the length of wheat seedlings after being exposed to different concentrations of the chemical substance for a defined time interval, the activity of luminescent bacteria, the ability of cell cultures to reduce a specific dye, the growth rate according to number of individuals or biomass, the number of viable offspring and many others.

These observed variables have in common that a reference magnitude for healthy and viable organisms can be defined (normalised response level $r = 1$), and that the magnitude of the variable (response) is limited by a zero response ($r = 0$) where the maximum of the effect is observed. The **drfit** package covers the case where there is a continuum of possible response values between 0 and 1 (inclusive). Additionally, responses above 1 are frequently observed due to variability or as the result of stimulation by a subtoxic dose, and even responses below 0 may be present, depending on the type of data and the applied preprocessing.

If the responses are binomial, such as life and death for a number of individuals, it is advisable

to choose the readily available glm fitting procedures (generalized linear models), where the probit and logit links are already built-in (e.g. Chapter 7.2 in [Venables and Ripley \(2002\)](#)) or to look into the **drc** package.

Dose-response relationships for continuous response tests can generally be expressed as

$$r = f(d, \vec{p}) + \epsilon \quad (1)$$

where r is the normalised response at dose d , $f(d, \vec{p})$ is the model function with parameter vector \vec{p} , and ϵ is the error variable describing the variability in the observations not explainable by the model function $f(d, \vec{p})$.

This article shows how different model functions $f(d, \vec{p})$ can be conveniently fitted to such dose-response data using the R package **drfit**, yielding the vector of parameters \vec{p} that gives the description of the data with the least residual error. The fitting can be carried out for many substances with a single call to the main function `drfit`.

The results that the user will probably be most interested in are the doses at which a response of 50 % relative to healthy control organisms is to be expected (termed ED_{50}), as this is a very robust parameter describing the toxicity of the substance toward the organism investigated.

The **drfit** package internally uses the R function `nls` for nonlinear regression analysis as detailed by [Bates and Watts \(1988\)](#). Confidence intervals for the model parameters are calculated by the `confint.nls` function from the **MASS** package as described in [Venables and Ripley \(2002\)](#).

drfit defines a dose-response data representation as a special case of an R dataframe, facilitates fitting standard dose-response models (probit, logit,

¹ The term dose is used here in a generalised way, referring to doses in the strict sense like mg oral intake per kg body weight as well as to measured concentrations in aquatic toxicity tests or nominal concentrations in cell culture assays.

weibull and linlogit at the time of this writing), and a function to produce different types of plots of the data as well as the fitted curves.

Optionally, the raw data can be kept in an external database connected by **RODBC**. This has proven to be useful if the data of a large number of dose-response experiments have to be evaluated, as for example in bioassays based on microtiter plates.

Recently, the R package **drc** containing similar functionalities to **drfit** has been uploaded to CRAN. Unfortunately, I have noticed the existence of this package only during the preparation of this article, after having maintained **drfit** on CRAN for almost one and a half years. Maybe in the future it will be possible to join forces.

In this introductory article, it is explained how the input data must be formatted, how dose-response curves are fitted to the data using the **drfit** function and in what ways the data and the models can be plotted by the **drplot** function. Since the package was actively developed during the preparation of this article, the reader is advised to upgrade to the latest **drfit** version available. Note that $R \geq 2.1.0$ is needed for recent **drfit** versions.

Collecting dose-response data

The **drfit** function expects the dose-response data as a data frame containing at least a factor called 'substance', a vector called 'unit' containing the unit used for the dose, a column 'response' with the response values of the test system normalized using the "natural" zero response as 0, and the response of the control organisms as a "natural" 1. Therefore, values outside this interval, and especially values above 1 may occur in the normalized data. An example of such data can be easily obtained from the built-in dataset **XY**.

```
> library(drfit)
> data(XY)
> print(XY,digits=2)
  nr.  substance dose unit fronds response
1   1   Control  0 mg/L  174   1.050
2   2   Control  0 mg/L  143   0.973
3   3   Control  0 mg/L  143   0.973
4   4 Substance X  10 mg/L  147   0.983
5   5 Substance X  10 mg/L  148   0.986
6   6 Substance X  10 mg/L  148   0.986
7   7 Substance X  100 mg/L   63   0.651
8   8 Substance X  100 mg/L   65   0.663
9   9 Substance X  100 mg/L   55   0.598
10  10 Substance X  300 mg/L   20   0.201
11  11 Substance X  300 mg/L   22   0.238
12  12 Substance X  300 mg/L   25   0.288
13  13 Substance X 1000 mg/L   13   0.031
14  14 Substance X 1000 mg/L   16   0.113
15  15 Substance X 1000 mg/L   16   0.113
16  16   Control  0 mg/L  153   0.999
```

```
17 17   Control  0 mg/L  144   0.975
18 18   Control  0 mg/L  163   1.024
19 19 Substance Y  10 mg/L   20   0.201
20 20 Substance Y  10 mg/L   19   0.180
21 21 Substance Y  10 mg/L   21   0.220
22 22 Substance Y 100 mg/L   13   0.031
23 23 Substance Y 100 mg/L   12   0.000
24 24 Substance Y 100 mg/L   13   0.031
25 25 Substance Y 300 mg/L   12   0.000
26 26 Substance Y 300 mg/L   12   0.000
27 27 Substance Y 300 mg/L   14   0.061
28 28 Substance Y 1000 mg/L   12   0.000
29 29 Substance Y 1000 mg/L   12   0.000
30 30 Substance Y 1000 mg/L   12   0.000
```

Normalisation of the response data is not done within the **drfit** package. It can either be carried out with a typical spreadsheet file, with some extra lines of R code, or by an external procedure, while or before the data is read into a database.

If the data is collected and normalised using MS Excel, it can be easily transferred to R by saving it in CSV format, and reading it in using the R function `read.csv2` or alternatively by the `read.xls` function from the **gdata** package. If OpenOffice.org Calc is being used, and the default values are used for exporting the data in CSV format, the function `read.csv` is very helpful.

Figure 1 shows a possible spreadsheet layout for capturing dose-response data including both the observed endpoint (number of fronds in this case) and the normalized response values.

Total growth inhibition is in this case the natural lower limit of the response and the response will therefore be zero if the number of duckweed (*Lemna minor*) fronds stays at the initial level n_0 during the observation time. The natural reference for the healthy organisms (response=1) is in this case given by the growth rate of the controls μ_c , calculated by

$$\mu_c = \frac{\ln(\bar{n}_c) - \ln(n_0)}{t - t_0} \quad (2)$$

where \bar{n}_c is the mean number of fronds in the control experiments after the observation time. The growth rates μ_i are calculated in the same way, and the normalized responses are then easily obtained by

$$r_i = \frac{\mu_i}{\mu_c} \quad (3)$$

If the spreadsheet from Figure 1 (which can be found at <http://www.uft.uni-bremen.de/chemie/ranke/data/drfit/>) were exported by writing a CSV file, this file could be processed by something like

```
> d <- read.csv('sampledata.csv',skip=2,dec=',')
```


depending on the path to the CSV file, the number of lines before the column headings and the decimal separator used.

	A	B	C	D	E	F
1	Concentration-response data for the Lemna growth tes					
2						
3	nr.	substance	dose	unit	fronds	response
4	1	Control	0	mg/L	174	1,0496
5	2	Control	0	mg/L	143	0,9726
6	3	Control	0	mg/L	143	0,9726
7	4	Substance X	10	mg/L	147	0,9834
8	5	Substance X	10	mg/L	148	0,9861
9	6	Substance X	10	mg/L	148	0,9861
10	7	Substance X	100	mg/L	63	0,6509
11	8	Substance X	100	mg/L	65	0,6631
12	9	Substance X	100	mg/L	55	0,5976
13	10	Substance X	300	mg/L	20	0,2005
14	11	Substance X	300	mg/L	22	0,2379
15	12	Substance X	300	mg/L	25	0,2881
16	13	Substance X	1000	mg/L	13	0,0314
17	14	Substance X	1000	mg/L	16	0,1129
18	15	Substance X	1000	mg/L	16	0,1129
19	16	Control	0	mg/L	153	0,9991
20	17	Control	0	mg/L	144	0,9754
21	18	Control	0	mg/L	163	1,0240
22	19	Substance Y	10	mg/L	20	0,2005
23	20	Substance Y	10	mg/L	19	0,1804
24	21	Substance Y	10	mg/L	21	0,2197
25	22	Substance Y	100	mg/L	13	0,0314
26	23	Substance Y	100	mg/L	12	0,0000
27	24	Substance Y	100	mg/L	13	0,0314
28	25	Substance Y	300	mg/L	12	0,0000

Figure 1: Data structure for a typical toxicity test in OpenOffice Calc. Note that the response column is calculated (see text).

Fitting and plotting

A quick result for a compatible dataframe can usually be obtained by a simple call to `drfit`

```
> rXY <- drfit(XY)
```

The contents of the dataframe `rXY` containing the results of the fitting procedure are shown in Figure 2. Each fitted dose-response model (usually only one per substance) produces one line. The number of dose levels `nd1` is reported, the total number of data points used for the model fitting `n`, the decadic logarithms of the lowest dose `l1d` and the highest dose `h1d` tested.

The next column contains the type of the dose-response model fitted (probit, logit, weibull or linlogit) or, if not applicable, a classification of the substance data as “active” (if the response at the lowest dose is < 0.5), “inactive” (if the response at the highest dose is > 0.5) or “no fit”.

The log ED_{50} is given with its confidence interval as calculated by the `confint.nls` function from the **MASS** package. This only works if the log ED_{50} is one of the model parameters. Therefore, in the case of the weibull model, no confidence interval is given.

Finally, the residual sum of squares `sigma` is listed and the fitted parameters `a` and `b`, or, in the case of the three parameter model `linlogit`, the parameters `a`, `b` and `c` are listed.

Once the `drfit` function has been successfully called and the result assigned a name (`rXY` in this case), dose-response plots for the fitted data can easily be created using the `drplot` function. The following example produces a single graph (`overlay=TRUE`) with the fitted dose-response curves and raw data (`dtype="raw"`) for all substances and fitted models in dataframes `XY` and `rXY` using color (`bw=FALSE`). Additionally, the scatter of the responses in control experiments can be displayed, by setting the argument `ctype` to “std” or “conf”: as shown in Figure 3.

```
> drplot(rXY,XY,overlay=TRUE,bw=FALSE,
  ylim=c("auto",1.3),dtype="raw", ctype="conf")
```

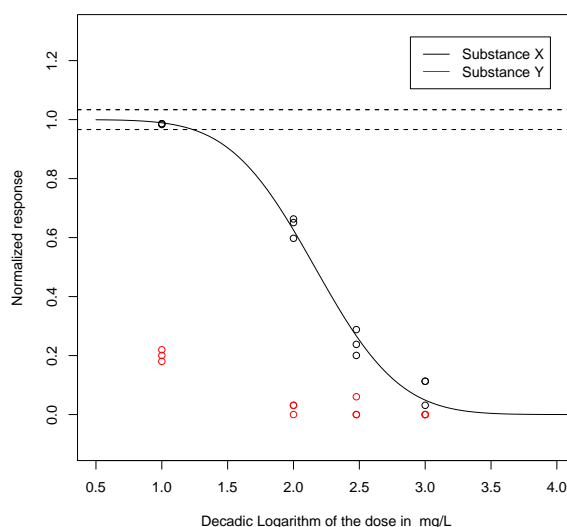


Figure 3: Output of the `drplot` function for the sample data `XY` from the package.

If the user prefers to view the raw data with error bars, the argument `dtype` can be set to “std” for showing standard deviations (default) or “conf” for showing confidence intervals.

In the following, the analysis of a somewhat more complicated, but also more interesting example is illustrated, which has been published by [Ranke et al. \(2004\)](#) before the basic `drfit` code was packaged.

First, dose-response curves with the default settings of `drfit` are generated as shown in Figure 4.

```
> data(IM1xIPC81)
> dIM <- IM1xIPC81
> rIM <- drfit(dIM)
> drplot(rIM,dIM,overlay=TRUE,bw=FALSE)
```

```
> print(rXY,digits=2)
  Substance nd1  n  lld  lhd  mtype logED50 2.5% 97.5% unit sigma  a  b
1   Control   1  6 -Inf -Inf inactive    NA    NA    NA mg/L   NA  NA  NA
2 Substance X   4 12   1   3  probit   2.2  2.1  2.2 mg/L 0.041 2.2 0.51
3 Substance Y   4 12   1   3  active    NA    NA    NA mg/L   NA  NA  NA
```

Figure 2: Contents of the dataframe containing the results from the fitting procedure for example data from the package (see text for explanations).

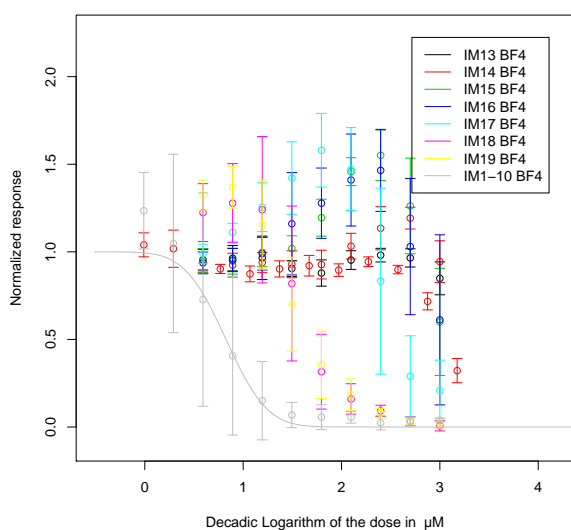


Figure 4: Dose-response plot showing the toxicities in a homologous series of compounds and the fitted probit model for IM1-10 BF4.

The graph shows that only one dose-response curve is fitted with the built-in default arguments of the `drfit` function and that the legend is interfering with the data. It is obvious that for almost all substances in this data, response values > 1 are caused in a certain dose range, a phenomenon which is called hormesis. In order to properly model such data, the so-called linear-logistic dose-response model has been introduced by [Brain and Cousens \(1989\)](#). The `drfit` package makes use of it in the parameterization suggested by [van Ewijk and Hoekstra \(1993\)](#), which allows for a convenient calculation of confidence intervals of the ED_{50} .

To include the linear-logistic model (`linlogit` in `drfit` terminology) in the fitting procedure and list the results including confidence intervals for a confidence level of 90 % two-sided, one simply calls

```
> rIM2 <- drfit(dIM,linlogit=TRUE,level=0.9,
  chooseone=FALSE)
```

First, the `linlogit` argument causes the `linlogit` model to be additionally tried. Then, the argument `chooseone=FALSE` leads to reporting one line

for each fitted model. If the argument `chooseone` is set to `TRUE` (default), only the first convergent dose-response model (probit and `linlogit` in this case) from the somewhat arbitrary sequence `linlogit > probit > logit > weibull` is reported.

The dataframe with the results shown in [Figure 5](#) accordingly lists all instances of fitted models, and gives confidence intervals for the log ED_{50} values.

Then, a customized plot can be generated:

```
> drplot(rIM2,dIM,overlay=TRUE,bw=FALSE,
  xlim=c("auto",5))
```

The `xlim` argument to `drplot` fixes the interference between legend and data. Furthermore, the plot produced in the above example shown in [Figure 6](#) shows two fitted dose-response curves for the substance IM1-10 BF4 (grey lines), one for the probit and one for the `linlogit` model.

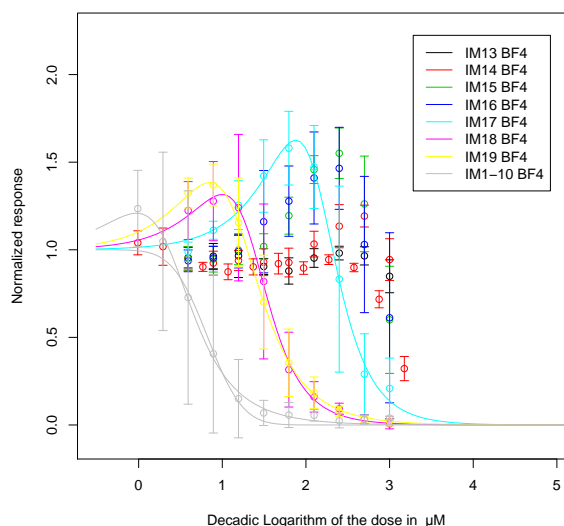


Figure 6: Dose-response plot showing the dose-response curves for a homologous series of compounds and all fitted `linlogit` and probit models.

External databases

Certain screening bioassays can be carried out with relatively low investments of time and money, so

```
> print(rIM2,digits=2)
  Substance ndl  n   lld lhd  mtype logED50  5% 95%  unit sigma  a  b  c
1  IM13 BF4   9  81  0.592 3.0 inactive    NA  NA  NA microM  NA  NA  NA  NA
2  IM14 BF4  20 216 -0.010 3.2  no fit    NA  NA  NA microM  NA  NA  NA  NA
3  IM15 BF4   9 135  0.592 3.0 inactive    NA  NA  NA microM  NA  NA  NA  NA
4  IM16 BF4   9 108  0.592 3.0 inactive    NA  NA  NA microM  NA  NA  NA  NA
5  IM17 BF4   9  81  0.592 3.0 linlogit  2.58 2.52 2.65 microM  0.24 2.58 2.30 0.015
6  IM18 BF4   9 135  0.592 3.0 linlogit  1.68 1.63 1.73 microM  0.23 1.68 2.24 0.057
7  IM19 BF4   9  81  0.592 3.0 linlogit  1.65 1.61 1.69 microM  0.15 1.65 1.98 0.110
8 IM1-10 BF4  11 162 -0.010 3.0 linlogit  0.77 0.70 0.84 microM  0.30 0.77 1.94 0.458
9 IM1-10 BF4  11 162 -0.010 3.0  probit   0.83 0.75 0.90 microM  0.31 0.83 0.33  NA
```

Figure 5: Contents of the dataframe containing the results from the fitting procedure for the chain length data IM1xIPC81 from the package (see text for explanations).

large volumes of dose-response data can build up (high-throughput screening/high-content screening). The `drfit` package makes it possible to retrieve data stored in databases accessible by ODBC using the `RODBC` package internally. Since `RODBC` works on Windows, Mac OS X and Unix platforms, the code is platform- and database independent to a high degree.

For storing cytotoxicity data in a MySQL database, the following minimum database definition is advisable:

```
CREATE TABLE `cytotox` (
  `pk` int(11) unsigned NOT NULL auto_increment,
  `plate` int(11) NOT NULL default '0',
  `experimentator` varchar(40) NOT NULL
    default '',
  `substance` varchar(100) NOT NULL default '',
  `celltype` varchar(20) NOT NULL default '',
  `conc` float NOT NULL default '0',
  `unit` set('unit1','...') default 'unit1',
  `viability` float NOT NULL default '0',
  `performed` date NOT NULL
    default '0000-00-00',
  `ok` set('not ok','ok','?','no fit')
    default '?',
  PRIMARY KEY (`pk`),
)
```

The `pk` and the `performed` data field are not interpreted by the package, databases with any other columns missing might work but have not been tested.

The column called `viability` contains the normalised response that has been calculated at the time of the data import into the database. Of course, the Data Source has to be configured to be a valid and accessible ODBC DSN on the platform used, e.g. by installing and configuring `unixodbc` and `myodbc` under Linux or `MyODBC` under Windows. This also involves setting up the MySQL server to listen to network connections, if it is not located on the local computer, and adequate MySQL user privileges.

With such a setup, the `drdata` function from the package can be used to conveniently retrieve data

from the database and evaluate it with the `drfit` and `drplot` functions:

```
> s <- c("Sea-Nine", "TBT", "ZnPT2")
> d <- drdata(s, experimentator = "fstock",
  whereClause="performed < 2006-04-04")
> r <- drfit(d, linlogit=TRUE)
> drplot(r, d, dtype="none",
  bw=FALSE, overlay=TRUE)
```

The `whereClause` argument to the `drdata` function allows for flexible selection of data to be used for the analysis, e.g. by using comparison operators on columns containing dates as illustrated in the above example.

Additionally, the use of the argument `dtype="none"` to the `drplot` function is shown, which leads to the display of the fitted models only, without the data, as shown in Figure 7.

In the UFT Center of Environmental Research and Technology, we use the `drfit` package for regular batch-processing of all our dose-response data from several bioassays for a substance library of more than 200 compounds. The results are in turn written to a database, and the `drplot` function is used to create updated dose-response plots every time the raw data has been augmented. The whole process of fitting all data and producing the plots takes less about 1 minute on an 1600 MHz AMD Sempron PC for the cytotoxicity data for 227 substances, provided that the new data has been checked by the `checkplate` and `checksubstance` functions, which allow for an easy validation of experimental dose-response data generated by plate-reader bioassays stored in a `drfit` conformant MySQL database.

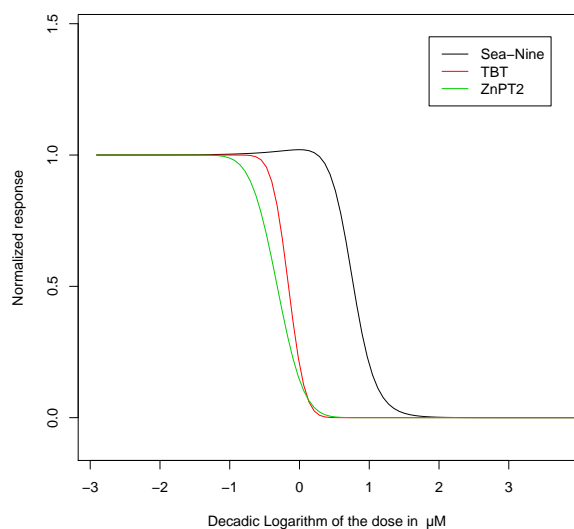


Figure 7: Dose-response plot showing the fitted dose-response curves for three antifouling biocides in the cytotoxicity assay fitted with the linlogit model.

The whole system provides the basis for analysis of the toxicity data, e.g. by (Quantitative) Structure-Activity Relationships (SAR/QSAR), which may provide a deeper chemical understanding of the interaction of the chemicals with biological organisms.

The pls package

by Bjørn-Helge Mevik

Introduction

The `pls` package implements *Partial Least Squares Regression* (PLSR) and *Principal Component Regression* (PCR). It is written by Ron Wehrens and Bjørn-Helge Mevik.

PCR is probably well-known to most statisticians. It consists of a linear regression of one or more responses Y onto a number of principal component scores from a predictor matrix X (Næs and Martens, 1988).

PLSR is also a linear regression onto a number of components from X , but whereas principal component analysis maximizes the variance of the scores, PLS maximizes the covariance between the scores and the response. The idea is that this should give components that are more relevant for the response. Typically, PLSR achieves the same (or smaller) pre-

Bibliography

- D. M. Bates and D. G. Watts. *Nonlinear Regression Analysis and its Applications*. Wiley Series in Probability and Mathematical Statistics. Wiley, New York, 1988. 7
- P. Brain and R. Cousens. An equation to describe dose responses where there is stimulation of growth at low doses. *Weed Research*, 29:93–96, 1989. 10
- J. Ranke, K. Mölter, F. Stock, U. Bottin-Weber, J. Poczobutt, J. Hoffmann, B. Ondruschka, J. Filser, and B. Jastorff. Biological effects of imidazolium ionic liquids with varying chain lengths in acute *Vibrio fischeri* and WST-1 cell viability assays. *Ecotoxicology and Environmental Safety*, 28(3):396–404, 2004. 9
- P. H. van Ewijk and J. A. Hoekstra. Calculation of the EC50 and its confidence interval when subtoxic stimulus is present. *Ecotoxicology and Environmental Safety*, 25:25–32, 1993. 10
- W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Statistics and Computing. Springer, New York, 2002. 7

Johannes Ranke

Department of Bioorganic Chemistry

UFT Center for Environmental Research and Technology

University of Bremen jranke@uni-bremen.de

diction error as PCR, with fewer components. A good introduction to PLSR and PCR can be found in Martens and Næs (1989). A review of PLSR is given in Wold et al. (2001) (in fact, all of that issue of Chemolab is dedicated to PLSR). Frank and Friedman (1993) provides a more technical treatment, from a statistical viewpoint.

PLSR and PCR are commonly used in situations where there are collinearities or near-collinearities in X , for instance when there are more variables than observations. This is a very common situation in fields like chemometrics, where various types of spectroscopic data are often used to predict other measurements.

There are other regression methods that can be applied to such data, for instance ridge regression. Studies have indicated that in terms of prediction error, ridge regression can perform slightly better than PLSR. However, one of the major advantages of PLSR and PCR is interpretation. In addition to a prediction equation, one gets score and loading vectors

for each component. These can be plotted and interpreted by the field expert. There is a strong focus on graphical assessment and interpretation in fields like chemometrics.

Typical usage

To illustrate the usage of the package, we will use a data set published in Kalivas (1997). It consists of octane number and *near infrared* (NIR) spectra of 60 gasoline samples. Each NIR spectrum consists of 401 diffuse reflectance measurements from 900 to 1700 nm. The spectra are shown in Figure 1.

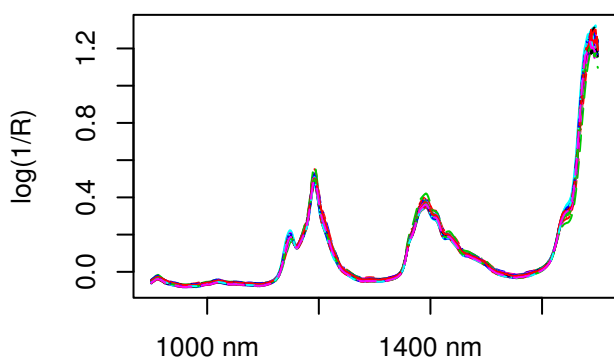


Figure 1: NIR spectra

The user interface is inspired by `lm`, and has a formula interface and methods for `plot`, `predict`, `fitted`, etc. The main modelling functions are `pls` and `pcr`, which fit PLSR and PCR models, respectively. They are both simple wrappers for `mvr`, and return a fitted model as an object of class "mvr". The object usually contains fit results for models with 0, 1, ..., `ncomp` components, where `ncomp` is specified in the call. Because the score vectors are mean centered and orthogonal, one can inspect submodels by selecting subsets of the components, without having to refit the model. Many extraction and plot functions support this through the argument `comps` (or, for a few functions, `ncomp`).

It is customary to use *cross-validation* (CV) (Stone, 1974) or test set validation to determine how many components to use in the regression. The modelling functions can perform cross-validation, and the results are stored in the model object.

We will illustrate the use of `pls` by doing a PLSR analysis of the gasoline data. A typical way of calling `pls` is:

```
> gas1 <- pls(oct ~ NIR, ncomp = 10,
+ data = gasoline, validation = "LOO")
```

This fits a model with 10 components, and includes *leave-one-out* (LOO) cross-validated predictions (Lachenbruch and Mickey, 1968). One can extract different elements of the fitted model with functions like `coef`, `scores`, `loadings`, etc.

The print method for "mvr" objects shows the type of model and fit algorithm, while the `summary` method gives more details about the fit and validation results (if any):

```
> summary(gas1)
```

```
Data:      X dimension: 60 401
          Y dimension: 60 1
Fit method: kernelppls
Number of components considered: 10
```

VALIDATION: RMSEP

```
Cross-validated using 60 leave-one-out segs.
      (Intercept)  1 comps  2 comps
CV      1.543      1.328   0.3813
adjCV   1.543      1.328   0.3793
      3 comps  4 comps  5 comps  6 comps
CV      0.2579   0.2412   0.2412   0.2294
adjCV   0.2577   0.2410   0.2405   0.2288
      7 comps  8 comps  9 comps 10 comps
CV      0.2191   0.2280   0.2422   0.2441
adjCV   0.2183   0.2273   0.2411   0.2433
```

TRAINING: % variance explained

	1 comps	2 comps	3 comps	4 comps
X	70.97	78.56	86.15	95.4
oct	31.90	94.66	97.71	98.0
	5 comps	6 comps	7 comps	8 comps
X	96.12	96.97	97.32	98.1
oct	98.68	98.93	99.06	99.1
	9 comps	10 comps		
X	98.32	98.71		
oct	99.20	99.24		

The validation results here are *Root Mean Squared Error of Prediction* (RMSEP). There are two cross-validation estimates: 'CV' is the ordinary CV estimate, and 'adjCV' is a bias-corrected CV estimate (Mevik and Cederkvist, 2004) (For a LOO CV, there is virtually no difference).

As an alternative, one can plot the RMSEPs in order to decide how many components to use. The function `RMSEP` returns an object of class "mvrVal", with its own print and plot methods. `RMSEP` can also calculate a test set estimate, if a test set is provided with the `newdata` argument. (If one prefers MSE, one can use the function `MSEP`.)

```
> plot(RMSEP(gas1), legendpos = "topright")
```

This plots the estimated RMSEPs as functions of the number of components (Figure 2). The `legendpos` argument adds a legend at the indicated position. Three components seem to be enough. This gives an RMSEP of 0.258. As mentioned in the introduction, the main practical difference between PCR and PLSR is that PCR often needs more components than PLSR to achieve the same prediction error. On this data set, PCR would need four components to achieve the same RMSEP.

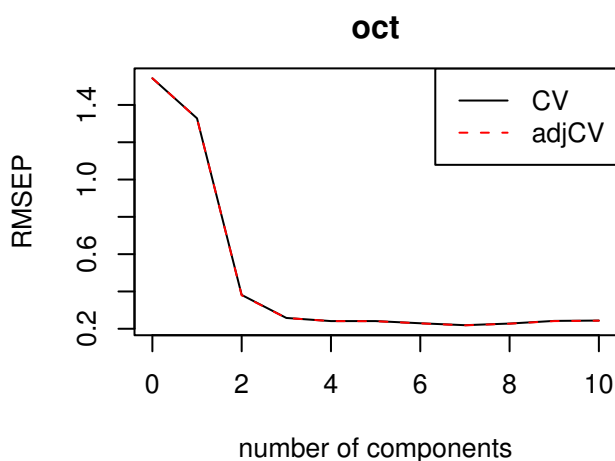


Figure 2: Cross-validated RMSEP curves

Once the number of components has been chosen, one can inspect different aspects of the fit by plotting predictions, scores, loadings, etc. The default plot is a prediction plot:

```
> plot(gas1, ncomp = 3, asp = 1, line = TRUE)
```

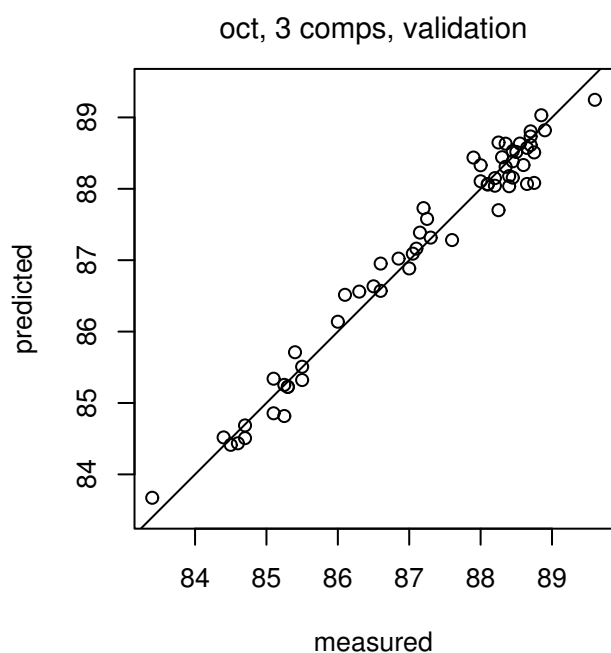


Figure 3: Cross-validated predictions

This shows the cross-validated predictions with three components versus measured values (Figure 3). We have chosen an aspect ratio of 1, and to draw a target line. One can plot fitted values instead of cross-validated predictions, by using the argument `which = "train"`.

Other plots can be selected with the argument `plottype`:

```
> plot(gas1, plottype = "scores",
```

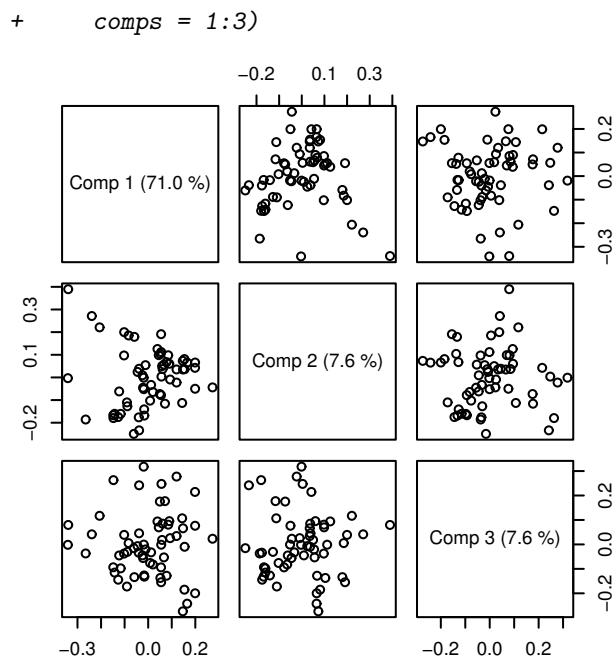


Figure 4: Score plot

This gives a pairwise plot of the score values (Figure 4). Score plots are much used to look for patterns, groups or outliers in the data. (For instance, plotting the two first components for a model built on the NIR dataset included in `pls`, clearly indicates the experimental design of the data.) The numbers in parentheses after the component labels are the relative amount of X-variance explained by each component. The explained variances can be extracted explicitly with `explvar(gas1)`.

Another common plot is the loading plot (Figure 5), which is much used for component interpretation, by looking for known spectral peaks or profiles:

```
> plot(gas1, "loadings", comps = 1:3,
+      legendpos = "topleft")
> abline(h = 0)
```

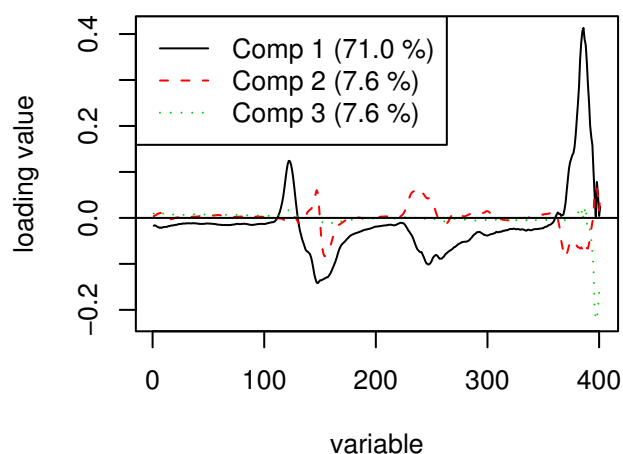


Figure 5: Loading plot

The package also implements biplots and correlation loadings plots. All the plots can also be accessed directly with functions `predplot`, `scoreplot`, etc.

Multi-response models

Multi-response models are handled automatically by the `fit`, `extraction` and `plot` functions in **pls**. The package contains a small sensory data set with 5 quality parameters `Quality` and 6 sensory attributes `Panel` for 16 olive oil samples. To see whether the sensory attributes can be explained by the quality measurements, one can do a regression of `Panel` on `Quality`:

```
> data(sensory)
> olive1 <- plsr(Panel ~ Quality,
+   data = sensory, val = "LOO")
```

The `plot` method for "mvrVal" objects gives one validation curve for each response (Figure 6):

```
> plot(RMSEP(olive1), nCols = 2)
```

The `nCols = 2` argument tells the underlying `plot.mvrVal` function to use two columns instead of three for the panels. The `syrup` attribute is best explained by a single component, but overall, two components seem optimal. The reduction in RMSEP is quite moderate for all responses, which is not unusual for sensory data. This can also be seen in a prediction plot (Figure 7):

```
> plot(olive1, ncomp = 2, asp = 1,
+   line = TRUE, nCols = 2)
```

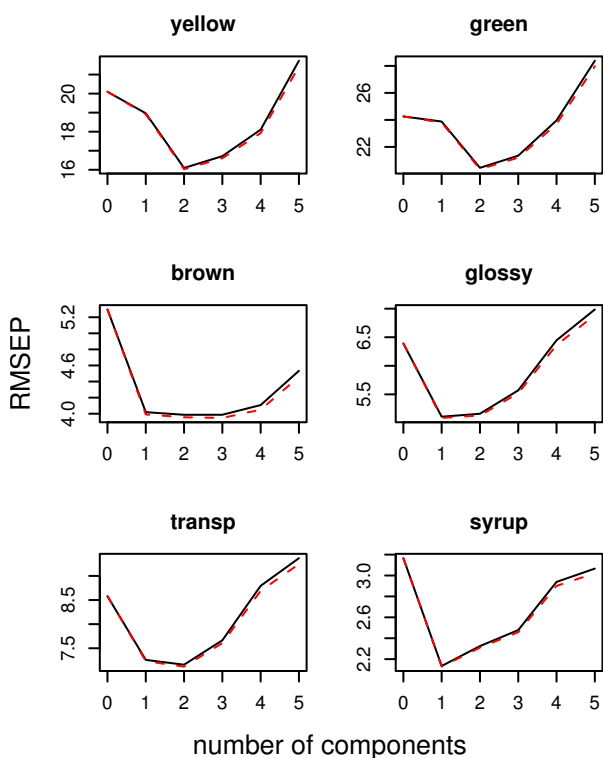


Figure 6: Cross-validated RMSEP curves

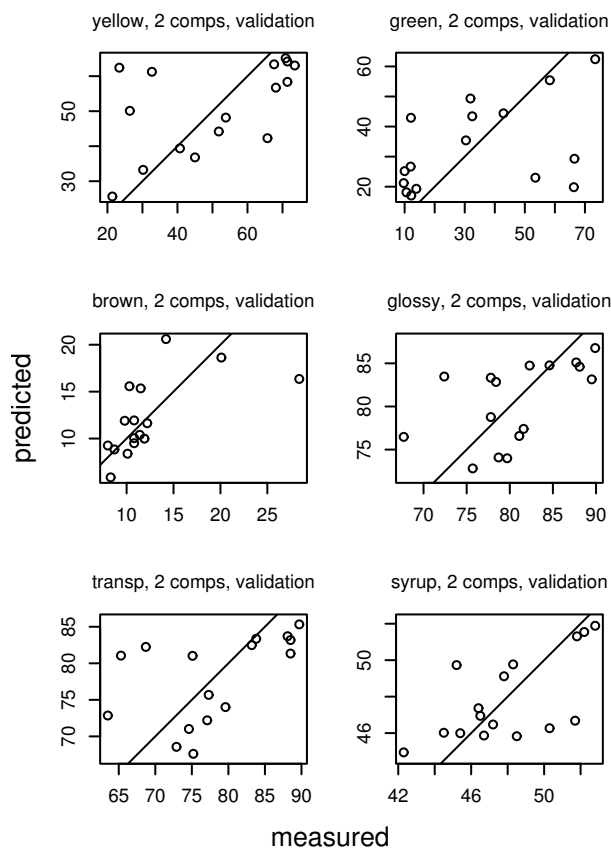


Figure 7: Cross-validated predictions

A correlation loadings plot can tell us which predictors are adequately described by the model, and which predictors correspond to the different components:

```
> corrplot(olive1, comps = 1:2,
+   labels = "names")
```

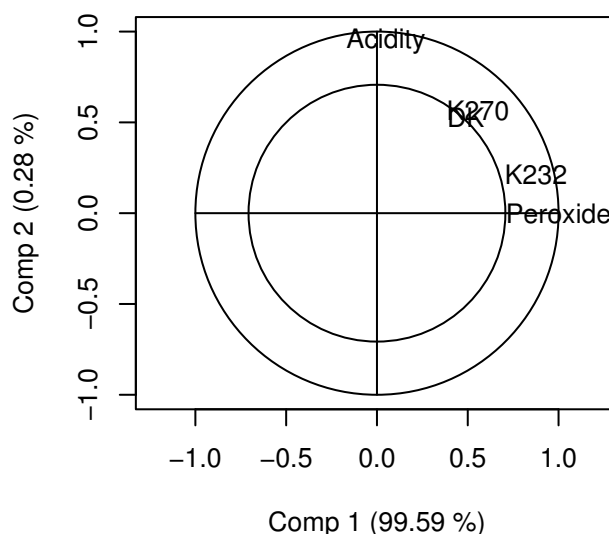


Figure 8: Correlation loadings plot

Figure 8 indicates that all predictors are quite well described, and that Peroxide and K232 are most correlated with component 1, while Acidity corresponds to the second component. The `labels = "names"` argument labels the points with the variable names. They can also be labelled with numbers. The default is the standard plot symbol. The percentages give the amount of X-variance explained by each component. We see that the first component explains almost all variance of X. However, the RMSEPs in Figure 6 show that two components are needed to explain all responses adequately.

Flexible cross-validation

The cross-validation in `pls` can handle quite complex situations, with respect to both the segments and the treatment of the data.

Often, the observations in a data set are grouped in some way or another, for instance when there are replicates. In such cases, each cross-validation segment should consist of entire groups. As an example, assume that the gasoline data actually consists of three replicate measurements performed on 20 samples, and that the replicates are stored in consecutive rows in the data table. This can be handled with

```
> gas2 <- update(gas1, validation = "CV",
+   segment.type = "consecutive",
+   length.seg = 3)
```

which gives 20 consecutive segments of size three. One can also choose interleaved or randomly selected (the default) segments. If one specifies `length.seg`, the number of segments are calculated, and *vice versa* if `segments` is specified. Care is taken so that the segment sizes differ at most by 1.

For full flexibility, e.g. for unequal segment sizes, the segments can be specified explicitly, as a list of index vectors. See `?mvrCv` for details.

In spectroscopic applications, the spectra often have to be preprocessed in order to eliminate artefacts such as light scattering effects. This is often done by taking the first or second derivative of the spectra, or using *Multiplicative Scatter Correction* (MSC). Currently, `pls` has a function `msc` which implements MSC. It can be used directly in model formulas: `gas2 <- pls(oct ~ msc(NIR), ...)`. This is implemented such that the same preprocessing is applied to new data in `predict(gas2, newdata = new.data)`.

Formulas like this pose a problem for the built-in cross-validation: For efficiency reasons, the formula is evaluated only once in `mvr`, on the complete data set, even when the built-in CV is used. However, the scatter correction ought to be re-calculated for each CV segment, because it depends on the whole data set. This can be done by using the more general (but slower) function `crossval` for cross-validation.

It takes an "mvr" object and returns a cross-validated object:

```
> gas2 <- pls(oct ~ msc(NIR), ncomp = 6,
+   data = gasoline)
> gas2 <- crossval(gas2, length.seg = 1)
> RMSEP(gas2)
```

	(Intercept)	1 comps	2 comps	
CV	1.543	1.296	0.3975	
adjCV	1.543	1.296	0.3971	
	3 comps	4 comps	5 comps	6 comps
CV	0.2516	0.2400	0.2212	0.2266
adjCV	0.2514	0.2389	0.2208	0.2263

As can be seen, there seems to be little effect of the MSC on these data. In fact, Figure 1 indicates that there is little scatter in the spectra.

Internals and Extensions

There are quite a few algorithms for calculating PLS scores and loadings. Most of them are equivalent for single-response models, and give more or less the same results for multi-response models. The `pls` package currently implements three of the more well-known algorithms: kernel PLS, SimPLS and the orthogonal scores algorithm.

Under the hood, `mvr` first handles the formula and data, and then sends the data to an underlying fit function, as determined by the method argument.

These fit functions can be called directly, which can be useful when speed is important, or when using PLSR or PCR as building blocks in other algorithms. As a simple example, a "quick 'n dirty" cross-validation can be made like this:

```
> X <- gasoline$NIR
> y <- gasoline$oct
> n <- nrow(X)
> A <- 10
> cvpreds <- matrix(nrow = n, ncol = A)
> for (i in 1:n) {
+   fit <- kernelpls.fit(X[-i,], y[-i],
+     ncomp = A, stripped = TRUE)
+   for (j in 1:A)
+     cvpreds[i,j] <- (X[i,] -
+       fit$Xmeans) %*%
+       fit$coefficients[,j]
+   cvpreds[i,] <- cvpreds[i,] + fit$Ymeans
+ }
> sqrt(colMeans((y - cvpreds)^2))

[1] 1.3281674 0.3813088 0.2578943
[4] 0.2411522 0.2411555 0.2294477
[7] 0.2191377 0.2279735 0.2421662
[10] 0.2440551
```

This is of course identical to the LOO CV results in `summary(gas1)`.

Another example can be found in the package **Ispls**, which is available on CRAN. **Ispls** uses ordinary least squares regression and PLSR to fit a response to a sequence of matrices (Jørgensen et al., 2005).

Bibliography

- I. E. Frank and J. H. Friedman. A statistical view of some chemometrics regression tools. *Technometrics*, 35(2):109–148, 1993. 12
- K. Jørgensen, B.-H. Mevik, and T. Næs. Combining designed experiments with several blocks of spectroscopic data. Submitted, 2005. 17
- J. H. Kalivas. Two data sets of near infrared spectra. *Chemometrics and Intelligent Laboratory Systems*, 37: 255–259, 1997. 13
- P. A. Lachenbruch and M. R. Mickey. Estimation of error rates in discriminant analysis. *Technometrics*, 10(1):1–11, 1968. 13
- H. Martens and T. Næs. *Multivariate Calibration*. Wiley, Chichester, 1989. 12
- B.-H. Mevik and H. R. Cederkvist. Mean squared error of prediction (MSEP) estimates for principal component regression (PCR) and partial least squares regression (PLSR). *Journal of Chemometrics*, 18(9):422–429, 2004. 13
- T. Næs and H. Martens. Principal component regression in NIR analysis: Viewpoints, background details and selection of components. *Journal of Chemometrics*, 2:155–167, 1988. 12
- M. Stone. Cross-validatory choice and assesment of statistical predictions. *Journal of the Royal Statistical Society, Series B—Methodological*, 36:111–147, 1974. 13
- S. Wold, M. Sjöström, and L. Eriksson. PLS-regression: a basic tool of chemometrics. *Chemometrics and Intelligent Laboratory Systems*, 58(2):109–130, 2001. 12

Bjørn-Helge Mevik
IKBM, Norwegian University of Life Sciences,
Ås, Norway.
bjorn-helge.mevik@umb.no

Some Applications of Model-Based Clustering in Chemistry

by Chris Fraley and Adrian E. Raftery

Interest in clustering has experienced a recent surge due to the emergence of new areas of application. Prominent among these is the analysis of images resulting from new technologies involving chemical processes, such as microarray or proteomics data, and contrast-enhanced medical imaging. Clustering is applied to the image data to produce segmentations that are appropriately interpretable. Other applications include minefield detection (Dasgupta and Raftery 1998; Stanford and Raftery 2000), finding flaws in textiles (Campbell et al. 1997; 1999), grouping coexpressed genes (Yeung et al. 2001), *in vivo* MRI of patients with brain tumors (Wehrens et al. 2002), and statistical process control (Thissen et al. 2005).

The use of clustering methods based on probability models rather than heuristic procedures is becoming increasingly common due to recent advances in methods and software for model-based clustering, and the fact that the results are more easily interpretable. Finite mixture models (McLachlan and Peel, 2000), in which each component probability corresponds to a cluster, provide a principled statistical approach to clustering. Models that differ in the

number of components and/or component distributions can be compared using statistical criteria. The clustering process estimates a model for the data that allows for overlapping clusters, as well as a probabilistic clustering that quantifies the uncertainty of observations belonging to components of the mixture.

The R package **mclust** (Fraley and Raftery 1999, 2003) implements clustering based on normal mixture models. The main clustering functionality is provided by the function `EMclust`, together with its associated `summary` and `plot` methods. Users can specify various parameterizations of the variance or covariance of the normal mixture model, including spherical and diagonal models in the multivariate case, along with the desired numbers of mixture components to consider. The mixture parameters are estimated via the EM algorithm (Dempster et al. 1977; McLachlan and Krishnan 1997), initialized by model-based hierarchical clustering (Banfield and Raftery 1993; Fraley 1998). The best model is selected according to the Bayesian Information Criterion or BIC (Schwarz 1978), a criterion that adds a penalty to the loglikelihood that increases with the number of parameters in the model.

In this article, we discuss an application of model-

based clustering to diabetes diagnosis from glucose and insulin levels in blood plasma. We also discuss two applications in image segmentation. In the first, model-based clustering is used to give an initial segmentation of microarray images for signal extraction. In the second, model-based clustering is used to segment a dynamic breast MR image to reveal possible tumors.

Model-based Clustering

In model-based clustering, the data x are viewed as coming from a mixture density $f(x) = \sum_{k=1}^G \tau_k f_k(x)$, where f_k is the probability density function of the observations in group k , and τ_k is the probability that an observation comes from the k th mixture component ($0 < \tau_k < 1$ for all $k = 1, \dots, G$ and $\sum_k \tau_k = 1$).

Each component is usually modeled by the normal or Gaussian distribution. In the univariate case, component distributions are characterized by the mean μ_k and the variance σ_k^2 , and have the probability density function

$$\phi(x_i; \mu_k, \sigma_k^2) = \frac{1}{\sqrt{2\pi\sigma_k^2}} \exp\left\{-\frac{(x_i - \mu_k)^2}{2\sigma_k^2}\right\}. \quad (1)$$

In the multivariate case, component distributions are characterized by the mean μ_k and the covariance matrix Σ_k , and have the probability density function

$$\phi(x_i; \mu_k, \Sigma_k) = \frac{\exp\{-\frac{1}{2}(x_i - \mu_k)^T \Sigma_k^{-1} (x_i - \mu_k)\}}{\sqrt{\det(2\pi\Sigma_k)}}. \quad (2)$$

The likelihood for data consisting of n observations assuming a Gaussian mixture model with G multivariate mixture components is

$$\prod_{i=1}^n \sum_{k=1}^G \tau_k \phi(x_i; \mu_k, \Sigma_k). \quad (3)$$

For reviews of model-based clustering, see McLachlan and Peel (2000) and Fraley and Raftery (2002).

For a fixed number of components G , the model parameters τ_k , μ_k , and Σ_k can be estimated using the EM algorithm initialized by hierarchical model-based clustering (Dasgupta and Raftery 1998; Fraley and Raftery 1998). Data generated by mixtures of multivariate normal densities are characterized by groups or clusters centered at the means μ_k , with increased density for points nearer the mean. The corresponding surfaces of constant density are ellipsoidal.

Geometric features (shape, volume, orientation) of the clusters are determined by the covariances Σ_k , which may also be parametrized to impose cross-cluster constraints. There are a number of possible parameterizations of Σ_k , many of which are implemented in **mclust**. Common instances include $\Sigma_k =$

λI , where all clusters are spherical and of the same size; $\Sigma_k = \Sigma$ constant across clusters, where all clusters have the same geometry but need not be spherical; and unrestricted Σ_k , where each cluster may have a different geometry.

Banfield and Raftery (1993) proposed a general framework for geometric cross-cluster constraints in multivariate normal mixtures by parametrizing covariance matrices through eigenvalue decomposition in the following form:

$$\Sigma_k = \lambda_k D_k A_k D_k^T, \quad (4)$$

where D_k is the orthogonal matrix of eigenvectors, A_k is a diagonal matrix whose elements are proportional to the eigenvalues, and λ_k is an associated constant of proportionality. Their idea was to treat λ_k , A_k and D_k as independent sets of parameters, and either constrain them to be the same for each cluster or allow them to vary among clusters. When parameters are fixed, clusters will share certain geometric properties: D_k governs the orientation of the k th component of the mixture, A_k its shape, and λ_k its volume, which is proportional to $\lambda_k^d \det(A_k)$. The model options available in **mclust** are summarized in Table 1.

A 'best' model for the data can be estimated by fitting models with differing parameterizations and/or numbers of clusters to the data by maximum likelihood, and then applying a statistical criterion for model selection. The Bayesian Information Criterion or BIC (Schwarz 1978) is the model selection criterion provided in the **mclust** software; the 'best' model is taken to be the one with the highest BIC value.

Example 1: Diabetes Diagnosis from Glucose and Insulin Levels

We first illustrate the use of **mclust** on the diabetes dataset (Reaven and Miller 1979) giving three measurements for each of 145 subjects:

glucose	-	plasma glucose response to oral glucose
insulin	-	plasma insulin response to oral glucose
sspg	-	steady-state plasma glucose (measures insulin resistance)

This dataset is included in the **mclust** package. The subjects were clinically diagnosed into three groups: normal, chemically diabetic, and overtly diabetic. The diagnosis is given in the first column of the diabetes dataset, which is excluded from the cluster analysis.

The following code computes the BIC curves using the function `EMclust` and then plots them (see Figure 1, upper left):

```
> data(diabetes)
> diBIC <- EMclust(diabetes[, -1])
> plot(diBIC)
```

Table 1: Parameterizations of the multivariate Gaussian mixture model available in **mclust**. In the column labeled '# covariance parameters', d denotes the dimension of the data, and G denotes the number of mixture components. The total number of parameters for each model can be obtained by adding Gd parameters for the means and $G - 1$ parameters for the mixing proportions.

identifier	Model	# covariance parameters	Distribution	Volume	Shape	Orientation
EII	λI	1	Spherical	=	=	NA
VII	$\lambda_k I$	G	Spherical	=	=	NA
EEI	λA	d	Diagonal	=	=	axes
VEI	$\lambda_k A$	$G + (d-1)$	Diagonal	=	=	axes
EVI	λA_k	$1 + G(d-1)$	Diagonal	=	=	axes
VVI	$\lambda_k A_k$	Gd	Diagonal	=	=	axes
EEE	λDAD^T	$d(d+1)/2$	Ellipsoidal	=	=	=
EEV	$\lambda D_k A D_k^T$	$1 + (d-1) + G[d(d-1)/2]$	Ellipsoidal	=	=	=
VEV	$\lambda_k D_k A D_k^T$	$G + (d-1) + G[d(d-1)/2]$	Ellipsoidal	=	=	=
VVV	$\lambda_k D_k A_k D_k^T$	$G[d(d+1)/2]$	Ellipsoidal	=	=	=

EII VII EEI VEI EVI VVI EEE EEV VEV VVV
 "A" "B" "C" "D" "E" "F" "G" "H" "I" "J"

The model parameters can then be extracted via the `summary` function, and results can be plotted using the function `coordProj` as follows:

```
> diS <- summary(diBIC,diabetes[,-1])
> coordProj(diabetes[,-1], dims=c(2,3),
            mu=diS$mu, sigma=diS$sigma,
            type="classification",
            classification=diS$classification)
```

The summary object `diS` contains the parameters and classification for the best (highest BIC) model. The function `coordProj` can be used to plot the data and **mclust** classification, marking the means and drawing ellipses (with axes) corresponding to the variance for each group (see Figure 1, lower left).

For this data, model-based clustering chooses a model with three components, each having a different covariance. Moreover, the corresponding three-group classification matches the three clinically diagnosed groups with 88% accuracy.

The uncertainty of a classification can also be assessed in model-based clustering. The function `uncerPlot` can be used to display the uncertainty of misclassified objects when there is a known classification for comparison. More generally, the function `coordProj` can be used to display the relative uncertainty of a classification:

```
> uncerPlot(diS$z,diabetes[,1])
> coordProj(diabetes[,-1], dims=c(2,3),
            mu=diS$mu, sigma=diS$sigma,
            type="uncertainty",
            uncertainty=diS$uncertainty)
```

The resulting plots are shown in Figure 1, upper right and lower right. In this case, the misclassified data points tend to be among the most uncertain.

Example 2: Microarray Image Segmentation

Microarray technology is now a widely-used tool in a number of large-scale assays. While many array platforms exist, a common method for making DNA arrays consists of printing the single-stranded DNA representing the genes on a solid substrate using a robotic spotting device. In the two-color array, the cDNA extracted from the experimental and control samples are first labelled using the Cy3 (green) and Cy5 (red) fluorescent dyes. Then they are mixed and hybridized with the arrayed DNA spots. After hybridization, the arrays are scanned at the corresponding wavelengths separately to obtain the images corresponding to the two channels. The fluorescence measurements are used to determine the relative abundance of the mRNA or DNA in the samples.

The quantification of the amount of fluorescence from the hybridized sample can be affected by a variety of defects that occur during both the manufacturing and processing of the arrays, such as perturbations of spot positions, irregular spot shapes, holes in spots, unequal distribution of DNA probe within spots, variable background, and artifacts such as dust and precipitates. Ideally these events should be automatically recognized in the image analysis, and the estimated intensities adjusted to take account of them.

Li et al. (2005) proposed a robust model-based method for processing microarray images so as to estimate foreground and background intensities. It starts with an automatic gridding algorithm that uses a sliding window to find the peaks and valleys. Then model-based clustering is applied to the (univariate) sum of the intensities of the two channels measuring the red and green signals to provide an initial segmentation. Based on known information about the data, it is assumed there can be no more than three groups in the model (background, fore-

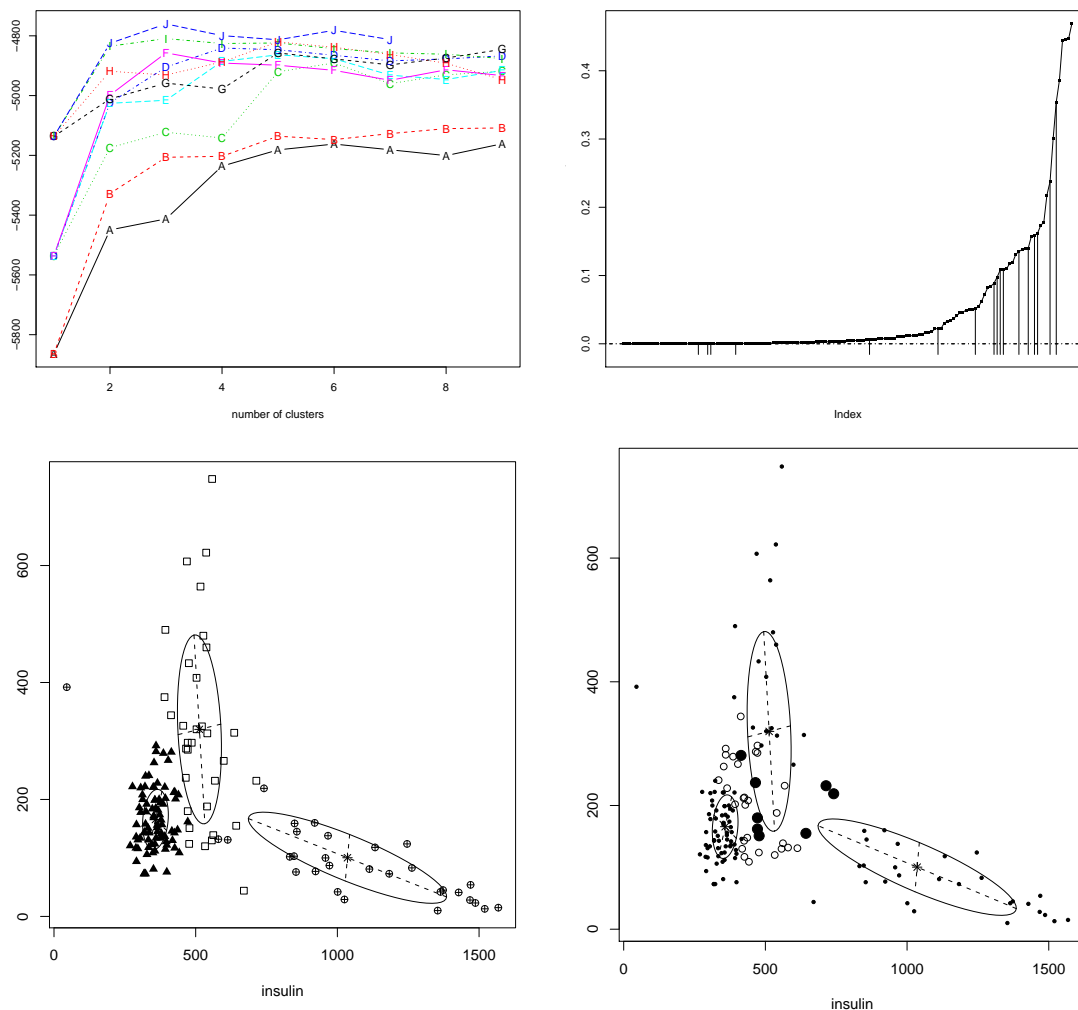


Figure 1: Upper left: BIC computed by EMclust for the 10 available model parameterizations and up to 9 clusters for the diabetes dataset. Different letters encode different model parameterizations, as output from the plot method. The 'best' model is taken to be the one with the highest BIC among the fitted models. Lower left: A projection of the diabetes data, with different symbols indicating the classification corresponding to the best model as computed by EMclust. The component means are marked and ellipses with axes are drawn corresponding to their covariances. In this case there are three components, each with a different covariance. Upper right: Uncertainty of the classification of each observation in the best model. Observations are ordered by increasing uncertainty along the horizontal axis. Vertical lines indicate misclassified observations, which in this case tend to be among the most uncertain. Lower right: A projection of the diabetes data showing classification uncertainty. Larger symbols indicate the more uncertain observations.

ground, uncertain). If there is more than one group, connected components below a certain threshold in size are removed (designated as unknown) from the brightest group as a precaution against artifacts. The procedure is depicted in Figure 2.

An implementation is available in Bioconductor (see <http://www.bioconductor.org>). The package is called **spotSegmentation**, and consists of two basic functions:

spotgrid: determines spot locations in blocks within microarray slides

spotseg: determines foreground and background signals within individual spots

1. Automatic gridding.
2. Model-based clustering for ≤ 3 groups.
3. Threshold connected components.
4. Foreground / background determination:
 - If there is more than one group, the foreground is taken to be the group of highest mean intensity and the background the group of lowest mean intensity.
 - If there is only one group, it is assumed that no foreground signal is detected.

Figure 2: Basic Procedure for Model-based Segmentation of Microarray Blocks.

These functions will be illustrated on the `spotSegTest` dataset supplied with the package, which consists of a portion of the first block from the first microarray slide image from van't Wout et al. (2003). This data set is a data frame, with two columns, one from each of the two channels of absorption intensities. The `spotSegTest` dataset can be obtained via the `data` command once the **spotSegmentation** package is installed.

```
> data(spotSegTest)
```

Because the data are encoded for compact storage, they need to be transformed as follows in order to extract the intensities:

```
> dataTrans <- function(x)
  (256*256-1-x)^2*4.71542407E-05
> chan1 <- matrix(dataTrans(spotSegTest[,1]),
  144, 199)
> chan2 <- matrix(dataTrans(spotSegTest[,2]),
  144, 199)
```

Note that this transformation is specific to this data; in general stored image data must be converted as needed to image intensities. The function `spotgrid` can be used to divide the microarray image block into a grid separating the individual spots.

```
> Grid <- spotgrid( chan1, chan2, rows = 4,
  cols = 6, show = FALSE)
> Grid
$rowcut
[1] 17 50 77 104 139

$colcut
[1] 12 41 66 94 123 151 183
```

Here we have used the knowledge that there are 4 rows and 6 columns in this subset of spots from the microarray image. The `show` option allows display of the gridded image.

The individual spots can now be segmented using the function `spotseg`, which does model-based clustering for up to 3 groups via **mclust** followed by a connected component analysis. The following segments all spots in the block:

```
Seg <- spotseg( chan1, chan2,
  Grid$rowcut, Grid$colcut)
plot(Seg)
```

The corresponding plot is shown in Figure 3.

Example 3: Dynamic MRI Segmentation

Dynamic contrast-enhanced magnetic resonance imaging (MRI) is emerging as a powerful tool for the diagnosis of breast abnormalities (e.g. Hylton 2005). Because of the high reactivity of breast carcinomas

after gadolinium injection, this technology has the potential to allow differentiation between malignant and benign tissues. Its unique ability to provide morphological and functional information can be used to assist in the differential diagnosis of lesions that other methods find questionable. It is currently used as a complementary diagnostic modality in breast imaging. However, data acquisition, postprocessing, image analysis and interpretation of dynamic breast MRI are still active areas of research. Forbes et al. (2006) developed a region of interest (ROI) selection method that combines model-based clustering with Bayesian morphology (Forbes and Raftery 1999), to produce a classification of the data for potential use in diagnosis.

Each dynamic MR image consists of 25 sequential images recording signal intensity after gadolinium injection. Instead of working directly with the image data, they are summarized in terms of five derived variables considered to be of significance in cancer diagnosis:

- **Time to peak:** the time at which the signal peaks.
- **Difference at peak:** absolute increase of intensity between the beginning of the signal and the time at which the signal peaks.
- **Enhancement slope:** in units of intensity/time.
- **Maximum step:** maximum change between two adjacent dynamic samples.
- **Washout slope:** in units of intensity/time.

Model-based classification for up to four groups is then applied to this data to segment the image. The choice of four groups is based on knowledge about the data: the main distinguishable components in breast tissue are blood vessels, air, fat, and possibly lesions or tumors. Figure 4 gives an example of model-based clustering applied to multivariate data derived as described above from dynamic contrast-enhanced breast MRI. Further steps using Bayesian morphology may then be applied to smooth the resulting image.

Summary

The contributed R package **mclust** implements parameter estimation for normal mixture models with and without constraints, with higher-level functions for model-based clustering and discriminant analysis. It includes functions for displaying the fitted models and clustered data.

The Bioconductor package **spotSegmentation** uses **mclust** to determine foreground and background of spots in microarray images.

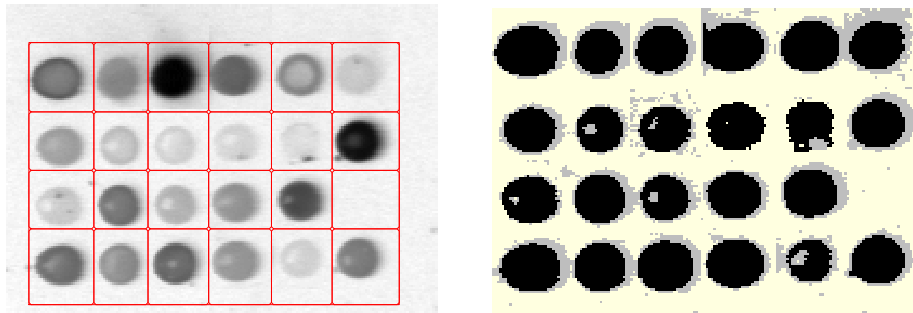


Figure 3: Left: The sum of channel signals from a portion of a microarray block containing HIV data, with the grid produced by `spotgrid` superimposed. Right: the corresponding segmented spots produced by `spotseg`, based on the grid produced by `spotgrid`. The color scheme is as follows: *black* denotes the spots, *yellow* denotes background, *gray* denotes pixels of uncertain classification.



Figure 4: Reference image (left) and four-class `mclust` classification (right). The tumor area is shown in red, with the colors assigned automatically according to the size of the mean difference at peak for pixels within each cluster.

Model-based clustering can be used successfully in a variety of technologies involving chemical processes, including image segmentation for cDNA microarrays and dynamic contrast-enhanced MR.

Bibliography

- J. D. Banfield and A. E. Raftery. Model-based Gaussian and non-Gaussian clustering. *Biometrics*, 49: 803–821, 1993.
- J. G. Campbell, C. Fraley, F. Murtagh, and A. E. Raftery. Linear flaw detection in woven textiles using model-based clustering. *Pattern Recognition Letters* 18, 1539–1548, 1997.
- J. G. Campbell, C. Fraley, D. Stanford, F. Murtagh, and A. E. Raftery. Model-based methods for real-time textile fault detection. *International Journal of Imaging Systems and Technology* 10, 339–346, 1999.
- A. Dasgupta and A. E. Raftery. Detecting features in spatial point processes with clutter via model-based clustering. *Journal of the American Statistical Association*, 93:294–302, 1998.
- A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood for incomplete data via the EM algorithm (with discussion). *Journal of the Royal Statistical Society, Series B*, 39:1–38, 1977.
- F. Forbes, N. Peyrard, C. Fraley, D. Georgian-Smith, D. Goldhaber, and A. Raftery. Model-based region-of-interest selection in dynamic breast MRI. *Journal of Computer Assisted Tomography*, 30:576–687, 2006.
- F. Forbes and A. E. Raftery. Bayesian morphology: Fast unsupervised Bayesian image analysis. *Journal of the American Statistical Association*, 94:555–568, 1999.
- C. Fraley. Algorithms for model-based Gaussian hierarchical clustering. *SIAM Journal on Scientific Computing*, 20:270–281, 1998.
- C. Fraley and A. E. Raftery. How many clusters? Which clustering method? - Answers via model-based cluster analysis. *The Computer Journal*, 41: 578–588, 1998.
- C. Fraley and A. E. Raftery. MCLUST: Software for model-based cluster analysis. *Journal of Classification*, 16:297–306, 1999.
- C. Fraley and A. E. Raftery. Model-based clustering, discriminant analysis and density estimation. *Journal of the American Statistical Association*, 97:611–631, 2002.
- C. Fraley and A. E. Raftery. Enhanced software for model-based clustering, density estimation, and discriminant analysis: MCLUST. *Journal of Classification*, 20:263–286, 2003.
- N. Hylton. Magnetic resonance imaging of the breast: Opportunities to improve breast cancer management. *Journal of Clinical Oncology*, 23(8): 1678–1684, March 10 2005.
- Q. Li, C. Fraley, R. E. Bumgarner, K. Y. Yeung, and A. E. Raftery. Donuts, scratches, and blanks: Robust model-based segmentation of microarray images. *Bioinformatics*, 21:2875–2882, 2005.
- G. J. McLachlan and T. Krishnan. *The EM Algorithm and Extensions*. Wiley, 1997.
- G. J. McLachlan and D. Peel. *Finite Mixture Models*. Wiley, 2000.
- G. M. Reaven and R. G. Miller. An attempt to define the nature of chemical diabetes using a multi-dimensional analysis. *Diabetologia*, 16:17–24, 1979.
- G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6:461–464, 1978.
- D. Stanford and A. E. Raftery. Principal curve clustering with noise. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22, 601–609, 2000.
- U. Thissen, H. Swierenga, A. P. de Weijer, R. Wehrens, W. J. Melssen and L. M. .C. Buydens, Multivariate statistical process control using mixture modeling. *Journal of Chemometrics*, 19:23–31, 2005.
- R. Wehrens and A. W. Simonetti and L. M. .C. Buydens, Mixture modeling of medical magnetic resonance data. *Journal of Chemometrics*, 16:274–282, 2002.
- A. B. van't Wout, G. K. Lehrman, S. A. Mikeeva, G. C. O'Keefe, M. G. Katze, R. E. Bumgarner, G. K. Geiss, and J. I. Mullins. Cellular gene expression upon human immunodeficiency type 1 infection of CD4(+)-T-cell lines. *Journal of Virology*, 77(2):1392–1402, January 2003.
- K. Y. Yeung, C. Fraley, A. Murua, A. E. Raftery, and W. L. Ruzzo. Model-based clustering and data transformation for gene expression data. *Bioinformatics* 17, 977–987, 2001.

Chris Fraley
Department of Statistics
University of Washington
fraley@stat.washington.edu

Adrian E. Raftery
Department of Statistics
University of Washington
raftery@stat.washington.edu

Mapping databases of X-ray powder patterns

by Ron Wehrens and Egon Willighagen

With the advent of high-throughput analysis methods, the number of large databases containing information on chemical compounds has increased rapidly. They are being used for many different purposes. In the pharmaceutical industry, for example, databases containing hundreds of thousands of drug-like compounds are being used to help identify promising new drug candidates. Obviously, the need for efficient data mining tools is increasing as well. It is especially difficult to assess how the separate objects in the database relate. Visualisation of the contents is of prime importance but not trivial; R offers a rich environment for this kind of exploratory analysis.

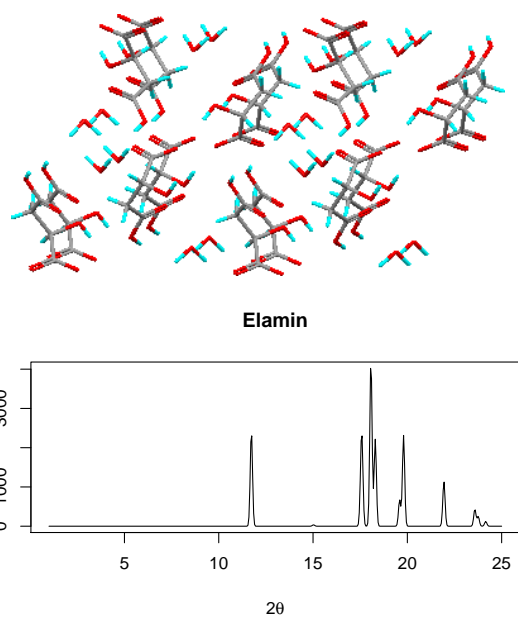


Figure 1: Crystal structure of ELAMIN (top) and associated powder pattern (bottom).

We show an example (Wehrens et al., 2005) of mapping part of the Cambridge Structural Database (CSD) (Allen, 2002) to a Kohonen Self-Organising Map (Kohonen, 2001). This database currently contains nearly 400,000 crystal structures. To assess the similarity of the crystal packing of these structures, many options are open. One can, e.g., compare the unit cell parameters (i.e. lengths of, and angles between, the three main axes that define the smallest replicated unit – the unit cell). Although this is easy and quick, there are several disadvantages to this approach. Apart from the fact that there is no unique definition of the unit cell, this representation does not

capture information about the positions of the atoms within the cell. Therefore, we use X-ray powder diffractograms, which contain all packing information. In Figure 1, the crystal structure of compound ELAMIN is shown, together with the powder pattern. The shape of the unit cell determines peak positions, and the electron density within the unit cell determines peak heights.

Crucial is the similarity measure used to compare powder patterns: it must above all capture similarities in peak position, where it should be noted that the number of peaks in patterns of different compounds may be quite different. This is achieved by a measure called the weighted cross-correlation (WCC) (de Gelder et al., 2001). It takes one parameter, the triangle width, the maximal shift that will still give a positive contribution to the similarity when comparing peaks.

A self-organising map consists of a regular grid of units, where each unit has a “codebook vector”, initially a random selection of patterns from the data set. During the training of the self-organising map, patterns are presented in random order. Each time, the unit that is most similar to the presented pattern is determined. The codebook vectors of this unit and of all other units within a certain neighbourhood are updated to become more similar to the presented pattern. Both the neighbourhood and the learning rate are decreased during training. At the end of the training, only the winning unit is updated; this, in fact, is a k-means clustering. The codebook vectors then can be interpreted as cluster centers. New objects can easily be projected in the map by comparing their patterns to all codebook vectors. Such a two-dimensional map is ideally suited for visual inspection and may show relations that otherwise may have gone unnoticed.

All this is implemented in package `wccsom`, available from CRAN. It is based on the SOM function by B.D. Ripley in package `class`, but has a number of extra features, most notably plotting functions, and the use of the WCC similarity function. The data used in this paper are proprietary (the CSD is a commercial database) but the package contains two smaller sets of powder patterns, provided by René de Gelder (Crystallography Department, Radboud University Nijmegen).

Example

For illustration purposes, we use a small dataset of 205 simulated powder patterns. Each pattern consists of 481 variables (2θ values). The set is con-

structured by searching the CSD for the compounds that are very similar to a set of twelve seed compounds; each group of compounds is considered to be a separate class. Powder patterns for three of the twelve classes are shown in Figure 2. ELAMIN is the seed compound of class 12. Obviously, not all classes are equally well defined: the spread in class three is much larger than in the other two.

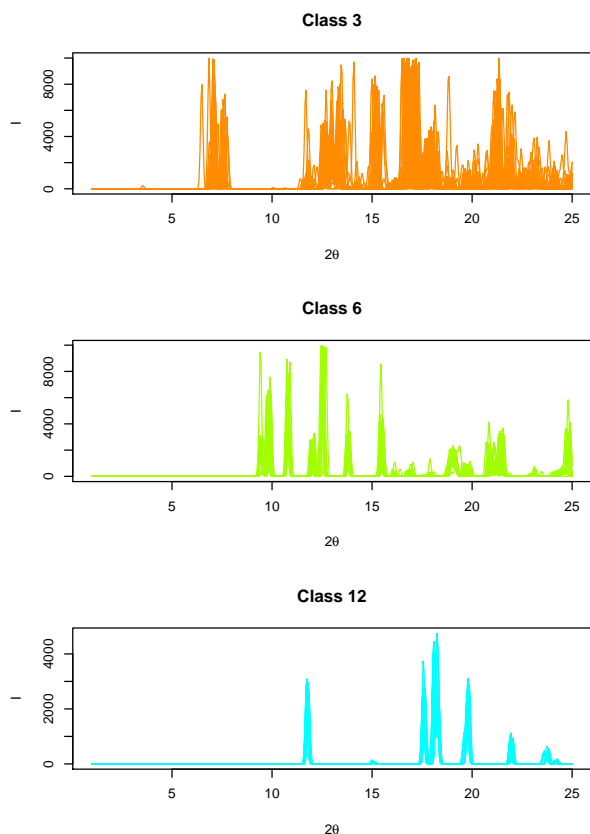


Figure 2: Powder patterns of classes 3, 6, and 12 (top to bottom).

The patterns are stored in a 205 x 481 matrix `testX`. They are mapped to a six-by-six Kohonen map by the following piece of code:

```
> library(wccsom)
> set.seed(7)
> somnet <- WCCSOM(testX,
+ somgrid(6, 6, "hexagonal"),
+ trwidth=20)
```

The triangle width for the WCC similarity function in this case is twenty points, corresponding to one degree 2θ . The convergence of the network can be assessed after training by plotting the mean change in similarity of the mapped object to the winning codebook vectors for each epoch (i.e. a complete presentation of the training set).

```
> plot(somnet, type="changes")
```

This is shown in Figure 3. In this case, the changes at the end of the training are very

small; note that this is also the result of the decrease in learning parameter α . The training is followed by a k-means clustering, which essentially performs a fine-tuning of the network.

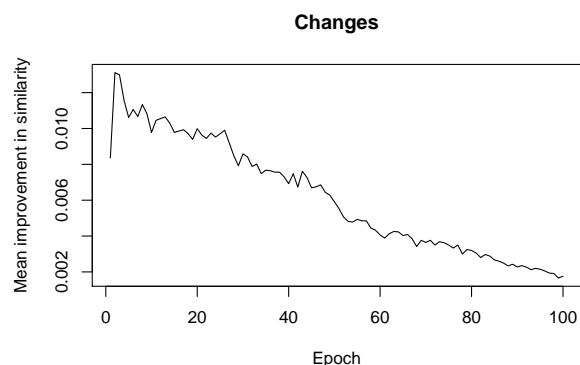


Figure 3: Convergence of network training: the plot shows the increase in similarity between the winning unit codebook vectors and the mapped objects during training.

Several other plots are available:

```
> mycols <- rainbow(12, start=0, end=.7)
> par(mfrow=c(2,2))
> plot(somnet, type="counts", main="Counts")
> plot(somnet, type="quality",
+ main="Quality")
> plot(somnet, type="mapping",
+ main="Mapping",
+ labels=classes, col=mycols[classes])
> plot(somnet, type="codes", main="Codes")
```

This leads to the plots in Figure 4. The figure in the top left shows how many objects are mapped to each unit: gray indicates an empty unit. In the figure labelled "Quality", the mean similarity of objects to the corresponding codebook vector is indicated in colour; the blue lines give an indication of the standard deviations. A line pointing downwards indicates the minimal variation in this set; a line pointing upward the maximal. Units containing only one object are usually very similar to that object; no variation is indicated in such a case. Which type of object is mapped to what unit is shown in the figure at the bottom left: `classes` is just a class vector of length 205. With the exception of class three, which is structurally the most diverse class, all objects of one class are mapped to one unit or one group of adjacent units. Finally, the unit codes are shown in the bottom right plot. As one would expect, they look very much like powder patterns.

Since the classes are chosen to be well separated, this is not a difficult example. However, using Euclidean distances or correlations as (dis)similarity measures does not lead to a good mapping: the twelve classes are mixed up completely.

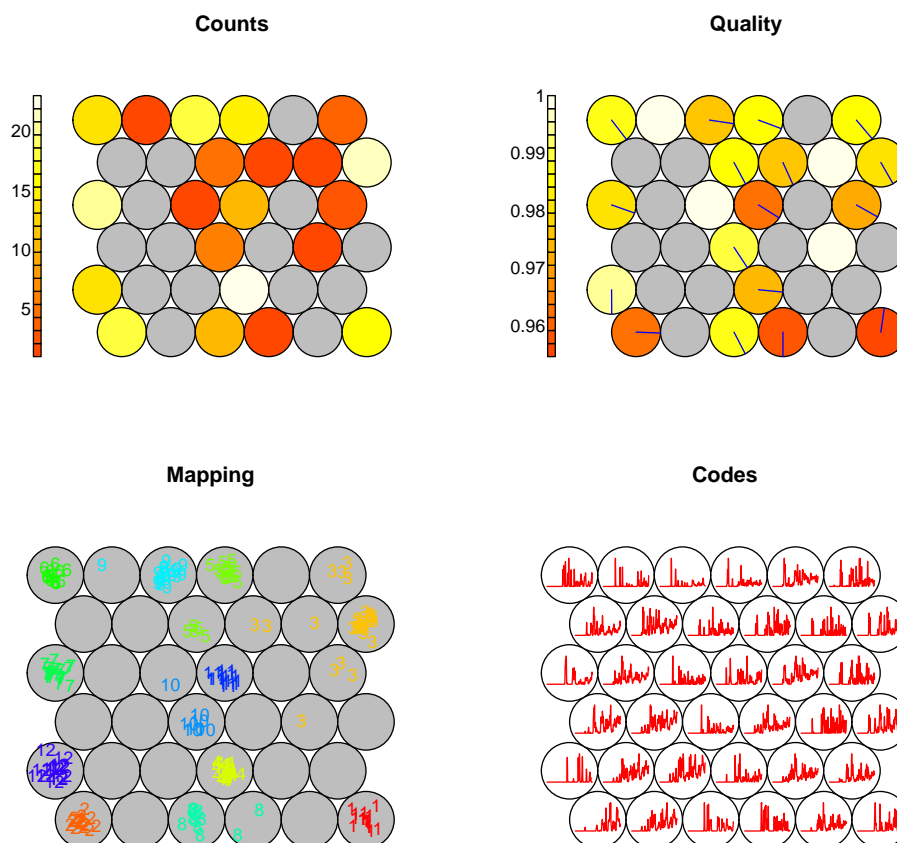


Figure 4: Several plotting types available for WCCSOM objects.

A more interesting case is presented by mapping all 2303 steroid-like structures present in the CSD (November 2004 release) to a 16x16 hexagonal grid. Apart from the plots shown earlier, the summary function can be used to get more information from the mapping. In particular, it is possible to get information on what objects are mapped to what unit, and how smooth the map is, i.e. how different the codebook vectors of neighbouring units are. Moreover, it is possible to summarize other information that may be present, such as reduced cell parameters. The seed structure for class 3 in the previous example, ECARAB, is also a steroid. It is mapped to unit 241 in the steroid map. We can check what other steroids are mapped there, and can compare their reduced cell parameters.

```
> set.seed(1)
> steroid.net <- WCCSOM(steroidX,
+   gr = somgrid(16, 16, "hexagonal"),
+   trwidth=20)
> options(digits=2)
> summary(steroid.net, type="unit",
+   nr = 241,
+   properties = steroid.props[,1:6])
Agreement with codebook vector of
25 objects mapped to cell 241:
      wccs   a   b   c alpha beta gamma
```

bifqai01	0.98	8.2	14	14	114	90	90
ecarab	0.99	8.1	14	14	114	90	90
eripuu	0.97	8.3	14	14	114	90	90
eripuu01	0.98	8.3	14	14	114	90	90
...							
zuzdon	0.99	8.1	14	14	114	90	90
zuzdut	0.97	8.1	14	14	116	90	90
zuzfab	0.95	8.1	14	14	116	90	90

All compounds mapped to this unit have very similar reduced cell parameters and very high similarities to the codebook vector. Class 5 of the test data set, which is also a steroid class but with a different symmetry (axis lengths of 7.2, 13.6, and 25.6 Å, respectively, and all angles α , β and γ equal to 90 degrees), is mapped in a totally different area of the map.

Although the compounds from the small test dataset are not all steroids, they still can be mapped to the steroid map:

```
> classif <- wccassign(steroid.net, testX)
> par(mfrow=c(2,1))
> plot(steroid.net, "mapping",
+   classif=classif,
+   labels=classes, col=mycols[classes])
> plot(steroid.net, "quality",
+   classif=classif)
```

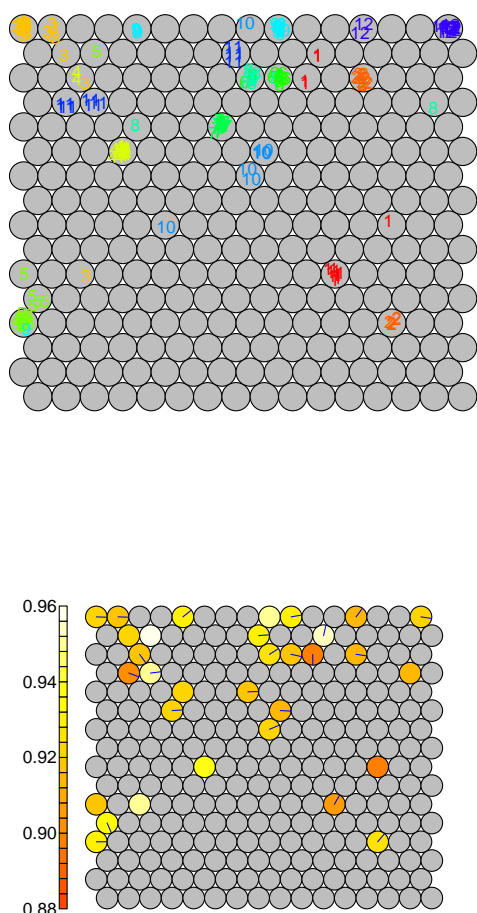


Figure 5: Mapping the test data set to the steroid map. Bottom figure: quality plot.

This leads to the plot in Figure 5. There is considerably more spread, mainly because of the much larger variability in powder patterns in the steroid data set. Note that the mapping quality is also lower.

Upscaling

Training the steroid data set takes roughly 45 minutes on a 2.0 GHz AMD Opteron processor. Obviously, training a network for all structures in the CSD presents problems regarding memory as well as calculation time. However, one of the advantages of self-organising maps is that one can train in steps: first use a subset of the complete database for initial training, and then gradually add new patterns until all have been presented to the net several times. An added advantage is that one can easily update the map when new releases of the database become available. Care must be taken, however, not to destroy other information in the network.

Still, it is a challenging task, and several tricks

should be applied to increase the speed of training. One obvious improvement is to use a growing network instead of a decreasing neighbourhood; another is to do the bulk of the training with fewer variables. For this, functions `expand.som` and `bucket/debucket` are available in package `wccsom`. Note that the triangle width for the WCC function should be adapted accordingly. In the following example, we first decrease the resolution of the powder patterns by a factor of five, and train a small four-by-four network. This network is expanded twice so that a similar-sized network as in the above example is obtained.

```
> steroid.Xsm <- bucket(steroidX, 5)
> somnet <- WCCSOM(steroid.Xsm,
+   gr=somgrid(4, 4, "hexagonal"),
+   rlen=20, radius=4, trwidth=4)
> for (i in 1:2) {
+   map.exp <- expand.som(somnet)
+   somnet <- WCCSOM(X, gr=map.exp$grid,
+     rlen=20, radius=4,
+     nhbrdist = map.exp$nhbrdist,
+     init = t(map.exp$codes))
+ }
```

Finally, the original resolution is restored (function `debucket` uses linear interpolation), and a few training epochs are used to fine-tune the codebook vectors. The results are comparable to the original map.

```
> sternet.codes <- debucket(t(somnet$codes),
+   ncol(steroidX))
> somnet.final <- WCCSOM(steroidX,
+   gr = somnet$grid, rlen=20,
+   radius=4, nhbrdist = map.exp$nhbrdist,
+   init = sternet.codes)
> options(digits = 2)
> ecarabindex <-
+   which(dimnames(steroidX)[[1]] ==
+     "ecarab")
> summary(somnet.final, type="object",
+   nr = ecarabindex,
+   properties = steroid.props[,1:6])
```

Agreement of object ecarab with
23 other objects mapped to cell 6:

	wccs	a	b	c	alpha	beta	gamma
bifqai01	0.98	8.2	14	14	114	90	90
eripuq	0.96	8.3	14	14	114	90	90
eripuq01	0.97	8.3	14	14	114	90	90
...							
zuzdon	0.99	8.2	14	14	114	90	90
zuzdut	0.97	8.1	14	14	116	90	90
zuzfab	0.95	8.1	14	14	116	90	90

Again, all compounds mapped to this unit share the same reduced cell parameters. Indeed, the set of compounds mapped to this unit is almost identical to the one found earlier. By this procedure, training time could be reduced by fifty percent. For larger data sets and, especially, larger nets, speed improvements are even bigger.

One further potential improvement that we have investigated but so far have been less successful with, is the use of stick patterns, rather than semi-continuous patterns consisting of equidistant points. Not only will this use less memory (on average 40 – 100 peaks are present in one pattern), but the calculation of the similarity function is faster, too. Moreover, it allows for easy selection of the most prominent features only, again decreasing computational requirements. However, the crucial updating step during training so far has proved difficult to define.

Applications

The trained map may be used to visualize the contents of the database in a variety of ways. For individual compounds, one can show areas that contain similar crystal packings. Pair-wise comparisons need not to be performed for the whole database, which is computationally very expensive: one can concentrate on the codebook vectors of the map. All compounds that map to similar units are candidates for further investigation.

A particular advantage of using semi-continuous powder patterns is that experimental patterns can directly be mapped. No peak picking is required. One can even assign chemical meaning to some characteristics of the unit vectors. If many peaks are present in the low 2θ ranges, it means that the cell volume is quite large. For other forms of spectroscopy, some peaks may even be interpreted directly. All this may aid in the elucidation of structure parameters of unknown compounds.

One can also use the map as a means of stratified sampling: a small set of compounds (e.g. one for each unit) can be selected that covers the complete chemical space. In polymorph prediction, this feature can be used to reduce the number of structures before (expensive) energy minimisation.

As a final example, trained maps can be used for database quality control. Although many steps in the sequence from measuring data to storing the results in a database can be automated and in fact are, there is still more than enough opportunity for error. Compounds with either very unrealistic powder patterns

or properties very dissimilar to compounds mapped to the same units are candidates for further investigation.

In conclusion, the examples in this paper show how the versatility of the R environment can contribute to a better understanding of the connections in large databases of molecules. The necessary speed is obtained by using compiled code; the trained maps are stored as R objects which can easily be interrogated, even by inexperienced R users, and on which new objects can easily be mapped. One could even envisage a web-based application similar to examples mentioned in (Kohonen, 2001).

Acknowledgements

René de Gelder is acknowledged for his crystallographic input; Willem Melssen for providing expertise on self-organising maps.

Bibliography

- F. H. Allen. The Cambridge Structural Database: a quarter of a million crystal structures and rising. *Acta Crystallogr.*, B58:380–388, 2002. 24
- R. de Gelder, R. Wehrens, and J. A. Hageman. A generalized expression for the similarity spectra: application to powder diffraction pattern classification. *J. Comput. Chem.*, 22(3):273–289, 2001. 24
- T. Kohonen. *Self-Organizing Maps*. Number 30 in Springer Series in Information Sciences. Springer, Berlin, 3 edition, 2001. 24, 28
- R. Wehrens, W.J. Melssen, L. Buydens, and R. de Gelder. Representing structural databases in a self-organizing map. *Acta Cryst.*, B61:548–557, 2005. 24

*Institute for Molecules and Materials
Analytical Chemistry
The Netherlands*

R.Wehe@science.ru.nl

Generating, Using and Visualizing Molecular Information in R

by Rajarshi Guha

Introduction

R, as a statistical computing environment, has a wide variety of functionality that makes it suitable for

modeling purposes in a number of fields. In the case of cheminformatics we are often presented with a large amount of information, from which chemical meaning and insight must be extracted, using computational tools. In many cases, problems in chem-

informatics are essentially data mining problems in which the data is chemical information. This information can be experimentally (such as assay data) or computationally generated. In the latter case, one requires access to a toolkit or library that is able to generate chemical information such as fingerprints, similarity values or molecular descriptors. A number of such toolkits are available such as JOELib, OEChem and the CDK (Steinbeck et al., 2006, 2003). In most situations one must write a program using the toolkit to generate chemical information which can then be imported into an R session and subsequently analyzed.

In terms of a consistent workflow, it would be useful to be able to access cheminformatics functionality from within the R environment itself. One example of such an application would be the development of a virtual screening pipeline which involves accessing and manipulating structures, evaluation of molecule descriptors and fingerprints and then the development of predictive models. In the interests of code reuse, we would prefer not to implement cheminformatics functionality within an R package but instead reuse established toolkits. The rest of this article will describe how the CDK project can be integrated into the R environment to provide cheminformatics functionality resulting in a seamless workflow for chemical data mining.

It should be noted that some commercial products do use R as a backend. However the combination of R and the CDK is attractive since both are Open-Source products and given that one is a full fledged programming language and the other is an extensive programming library, their combination provides the user with a large amount of flexibility in terms of data access, generation and modeling.

Requirements

As noted, we focus on the integration of the CDK project with R. The CDK project is a Java framework for cheminformatics development. Consequently, to access the classes and methods of the CDK libraries from R, we require an R-Java bridge. This requirement is satisfied by using the SJava package (Temple-Lang, 2005). The examples in this article have been tested with SJava 0.68 and R 2.1.0 running on Fedora Core 3 and 4. In addition, a recent build of the CDK project is required and either all the individual jar files or else the single comprehensive jar file must be placed in the user's CLASSPATH variable. Finally in order to visualize molecular structures a recent build of the Jmol (Howard, 2005) jar file as well as a copy of the JChemPaint (Krause et al., 2000) jar file should also be placed in the users CLASSPATH.

If one looks at the documentation for the CDK project it is clear that it provides a very wide variety of cheminformatics functionality. Performing

certain tasks using the CDK can be quite involved (such as loading arbitrary file formats) whereas other tasks can consist of a single call to a static method (such as getting a fingerprint). Though SJava provides the means to access all the functionality of the CDK, many tasks can become tedious to perform in the R environment. Furthermore, the CDK methods will generally return non-primitive Java objects which do not have a corresponding R type. Thus the user must keep track of R and Java objects. To alleviate these problems we have provided a Java library and associated R wrapper functions, known as the `rcdk` library available in the form of an R package from <http://cheminfo.informatics.indiana.edu/~rguha/code/R/index.html#rcdk>. It provides functions to perform a number of common tasks without having to directly access the CDK API via SJava. The aim of the package is to remove some of the tedious conversions between Java types and R types as well as simplify a number of tasks which would otherwise require the user to be more than casually familiar with the CDK API. Table 1 summarizes the R functions currently available in the `rcdk` package.

Generating & using molecular information

As described above, R is well suited to manipulating and analyzing data. The first step, however, is to obtain the data. For cheminformatics problems an important source of data are chemical structures, from which we can generate fingerprints, molecular descriptors and so on. Easy access to the CDK API using the SJava package provides the user with a streamlined workflow, whereby statistical and chemical information can be generated and manipulated within a single environment. In this section we present two examples of how we can access the CDK classes to obtain information regarding chemical structure and then use the data for modeling purposes.

A common task in a cheminformatics setting is clustering. When clustering molecules, we usually consider a variety of structural features of the molecules in question. One approach is to calculate a set of molecular descriptors. Another common approach is to evaluate binary fingerprints. Thus for example, we may have a collection of structure files. To perform a fingerprint based clustering, we would load the files into the R environment and then call the fingerprint method present in the CDK API. The return value of this method is a `java.util.BitSet` object which must be converted to a form usable by R. This is easily performed by parsing the String representation of the `BitSet` object. With the help of the

Function	Description
<code>edit.molecule</code>	Brings up the JChemPaint 2D structure editor and returns the final structure(s) as a CDK Molecule object
<code>get.desc.values</code>	Extracts the numerical values from a CDK DescriptorValue object returned by a descriptors calculate method
<code>get.fingerprint</code>	Returns a list of numeric objects or a single numeric object containing the positions of the bits that are set to 1 for the specified molecules fingerprint. Uses the default settings for the CDK Fingerprinter class
<code>load.molecules</code>	Loads one or more molecular structure files from disk and returns a list of CDK Molecule objects
<code>view.molecule</code>	Brings up a Jmol window displaying the molecule contained in the specified file. A Jmol script string can also be supplied
<code>view.molecule.table</code>	Displays multiple molecular structure files and associated numerical data in tabular format

Table 1: A summary of the functions available in the **rdck** package.

rdck package functions this can be easily achieved in a few lines of code:

```
> fnames <- list.files(pattern='*.sdf')
> molecules <- load.molecules(fnames)
> fprinter <- JNew('Fingerprinter')
> fplist <- lapply(molecules,
+ function(mol,fpr) {
+   .Java(fpr, 'getFingerprint', mol)
+ }, fpr=fprinter)
>
```

To convert the fingerprint returned by the CDK to a form usable by R we can use the following code:

```
> s <- gsub('{ }', '',
+         fplist[[1]]$toString())
> s <- strsplit(s, split=',')[[1]]
> s <- as.numeric(s)
```

The `load.molecule` function encapsulates a number of calls to the CDK API via SJava to load a molecular structure file. Since the CDK API provide a single static method to obtain fingerprints, the call using SJava is simple enough that a wrapper function is not required. The **rdck** package provides a convenience function, `get.fingerprint`, that encapsulates the above call to the CDK library and parsing of the return value to a numeric.

The code snippet above converts the fingerprint to a numeric object and multiple fingerprints can be aggregated into a list object and then manipulated using the binary fingerprint package (Guha, 2005a). This package allows one to obtain a distance matrix for the set of fingerprints using the Tanimoto metric which can then be used as input to the numerous clustering routines available in R. For a more detailed discussion of this application the reader is referred to Guha (2005c).

Another important task in the field of cheminformatics is QSAR modeling. In statistical terms this is simply the development of predictive models using feature vectors obtained from molecular struc-

ture information. Clearly, the R environment is well suited for the development of QSAR models. To build QSAR models one needs a set of feature vectors characterizing various aspects of molecular structure. These features are generally termed molecular descriptors and one can find references to a huge variety of descriptors in the cheminformatics literature (Todeschini and Consonni, 2002) and a number of packages are available to perform descriptor calculation (Jurs et al., 1979; Chemical Computing Group Inc., 2004; Todeschini et al., 2005). The CDK contains a number of classes implementing a variety of descriptor routines, including constitutional (atom and bond counts), topological (Zagreb index, χ indices), geometrical (moments of inertia, gravitational index) and whole molecule (BCUT) descriptors. The design of the CDK descriptor package allows the user to automatically evaluate all the available descriptors, though this is still a work in progress. Let us consider an example in which a specific descriptor is to be evaluated, such as the BCUT (Pearlman and Smith, 1999) descriptor. By default the descriptor routine will return the highest and lowest eigenvalues of the property weighted Burden matrices. Thus the simplest way to use this descriptor is:

```
> desc <- .JavaConstructor('BCUTDescriptor')
> dval <- desc$calculate(molecule)
```

Here molecule is an object of class Molecule that has been previously obtained from a disk file or from the structure editor (see below). The return value of the `calculate()` method is an object of class DescriptorValue which contains both the numerical values of the descriptor as well as extra information regarding implementation and literature references. Though it is simple enough to extract the numerical values from this object, the design of the CDK descriptor package can make such extraction a tedious process in the R environment. The **rdck** package provides a convenience function to extract the descriptor value from the object returned by `calculate()`

as follows:

```
> dnum <- get.desc.values(dval)
```

The return value of this function is a numeric object which can contain one or more elements depending on how many values were calculated by the descriptor routine. A more detailed description of this application, including parameter specification, can be found in [Guha \(2005c\)](#). Once a set of descriptors have been calculated, they can be converted to a `data.frame` and used as input to feature selection and model development functionality provided by R.

Molecular editing

In many cheminformatics problems we start out with a set of molecular structures. In general we use a structure editor to make modifications to the structures. Models based on structural information must now be updated. Since, in general, structure editors are external programs it would be useful to be able to access a structure editor from within the R environment. Another situation is when one would like to predict a property of a new molecule using a QSAR model built in R. In both cases, being able to access a structure editor from within R leads to a more consistent and efficient workflow. This can be achieved by using the JChemPaint ([Krause et al., 2000](#)) module of the CDK project. JChemPaint can be used as a standalone 2D structure diagram editor. However, being part of the CDK project, it can also be embedded in other programs and can be queried to obtain the structures being drawn. These features allow it to be used from within R. The `rcdk` package provides a convenience function to bring up the structure editor and return the structure that was drawn. Its usage is simply

```
> molecule <- edit.molecule()
```

In addition, it is also possible to supply a structure to the editor and modify it. The return value is a Java object of class `Molecule`. This is not meant to be used directly in R but can be passed to other methods in the CDK libraries. It is important to note that the structure returned will only have 2D coordinates. For many cases, in which only connectivity information is required, such as QSAR models built from topological descriptors, this is sufficient. However this approach will not be useful for cases where 3D geometries are required. The CDK has recently been enhanced with the development of a 3D coordinate generation package. We are currently extending the `rcdk` package to make use of this functionality and thus allow the user to generate reasonable 3D structures from within the R environment.

Molecular visualization

The previous section has discussed how one can draw and edit molecular structures using the JChemPaint application from within the R environment. However another important task in a cheminformatics workflow is the visualization of 3D molecular structures. This can be achieved by using Jmol which is an open source program for visualizing a wide variety of molecular structures. This project also utilizes a number of CDK classes. As a result it can read a wide variety of molecular structure file formats as well as handle data structures returned by CDK classes.

As in the case of structure editing, the `rcdk` project provides Java classes that handle the details of instantiating a Jmol window and loading structure files. The associated R source file provides R functions that wrap these Java classes.

Currently the `rcdk` package provides two methods to view 3D structures. The first method takes in a single character variable containing the path to the file to view. It can also optionally take a command string which is passed onto Jmol which evaluates it as a script. Thus, to view a structure one could simply write

```
> view.molecule('data/dan001.xyz')
```

A screenshot of the resultant viewer is shown in [Fig. 7](#).

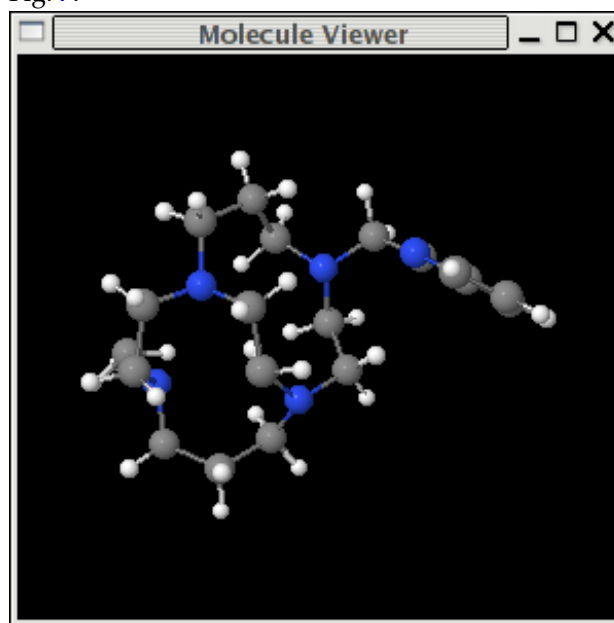


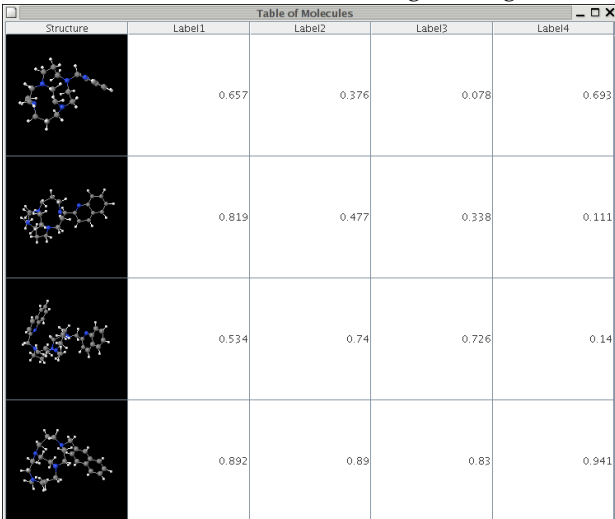
Figure 1: A screenshot of the single molecule viewer.

For tasks such as QSAR modeling one is usually working with a collection of molecules. In this case it would be useful to be able to view molecular structures and associated information (such as molecular descriptor values) in a table. This can be easily achieved by creating a table containing Jmol instances. The `rcdk` package provides a Java class

and associated R function wrapper that performs this task. Thus to view the structures and associated information for a set of molecules one could write:

```
> fnames <- c('data/dan001.xyz',
+ 'data/dan002.xyz', 'data/dan003.xyz',
+ 'data/dan004.xyz')
>
> cnames <- c('Structure', 'Label1',
+ 'Label2', 'Label3', 'Label4')
>
> moldata <- data.frame(matrix(
+ runif(4*4), nrow=4))
>
> view.molecule.table(fnames, cnames, moldata)
```

The resultant table is shown in Fig. 7. However, since each Jmol instance is a full fledged molecular viewer this can be a strain on resources. A future extension to the `rcdk` package will allow the use of the 2D structure diagram generator class present in the CDK, which would result in a much more light weight table.



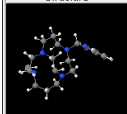
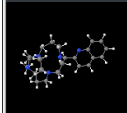


Structure	Label1	Label2	Label3	Label4
	0.657	0.376	0.078	0.693
	0.819	0.477	0.338	0.111
	0.534	0.74	0.726	0.14
	0.892	0.89	0.83	0.941

Figure 2: A screenshot of the table view.

Accessing R from the CDK

This article has focused on accessing the cheminformatics functionality of the CDK from within the R environment. In many cases it is advantageous to be able to access the statistical functionality of R from within a CDK based program. To facilitate this, the CDK contains a set of modeling classes which allow the user to access a number of R routines. The classes are based on SJava and a detailed description of the design of these classes can be found in Guha (2005b). An example of an application using these classes can be found in Guha and Jurs (2005). Currently, the CDK modeling package provides access to linear regression (`lm`), neural networks (`mnet`) and PLS (`p1s.pcr`). Future work on these classes will include other modeling routines (`randomForest`, `lda`,

etc.). A downside of the use of the SJava package is that the modeling functionality is effectively restricted to the Linux platform. Though SJava can be used on Windows, installation can be problematic. In addition, the SJava package is not multi-threaded and as a result all instances of the CDK modeling classes share the same R session. Currently, the design of the classes takes this into account, but this adds an extra layer of complexity. Finally, extending the CDK modeling classes to include other R routines is quite tedious since it essentially involves the design of Java classes which are one-to-one mappings of R objects. One approach that is being considered is the use of the Rserve package (Urbanek, 2005). This package would allow for a much simpler design of the CDK modeling classes and avoids a number of problems associated with the SJava package. The downside is that it requires that the user manually run the Rserve daemon, either remotely or locally. As a result, unless the Rserve daemon is running, the CDK modeling classes will not be of any use.

Conclusions

This article has attempted to highlight the use of the CDK within the R environment for the purposes of cheminformatics tasks. Using the SJava package, the user has access to the wide array of cheminformatics functionality present in the CDK as well as allied projects such as JChemPaint and Jmol. The integration of the CDK project with R results in a very useful workflow for chemical data mining problems. As shown above, one can calculate fingerprints or molecular descriptors and then perform a variety of modeling tasks using the statistical functionality of R. Coupled with Jmol and JChemPaint, one can visualize both statistical results as well as molecular structures. Furthermore, the use of the SJava package allows the CDK project to utilize the statistical functionality of R from within CDK based programs.

Though the user can directly access CDK classes and methods, this can become cumbersome. As a result the `rcdk` package was designed to alleviate the tedium of some common tasks such as loading molecular structure files and visualizing 3D structures. The package consists of a set of Java classes in the form of a jar file and R wrapper functions. Future work involves the addition of a number of helper functions for common cheminformatics tasks as well making the various functions more robust in terms of error handling as well as what objects can be accepted.

Bibliography

Chemical Computing Group Inc. Molecular Operating Environment (MOE 2004.03), 2004. 30

- R. Guha. <http://cheminfo.informatics.indiana.edu/~rguha/code/R/index.html#bfp>, September 2005a. 30
- R. Guha. Using R to Provide Statistical Functionality for QSAR Modeling in CDK. *CDK News*, 2:7–13, 2005b. 32
- R. Guha. Using the CDK as a Backend to R. *CDK News*, 2:2–6, 2005c. 30, 31
- R. Guha and P. Jurs. Integrating R with the CDK for QSAR modeling. In *230th American Chemical Society Meeting & Conference*, Washington D.C., 2005. 32
- M. Howard. <http://www.jmol.org>, September 2005. 29
- P. Jurs, J. Chou, and M. Yuan. *Computer Assisted Drug Design*, chapter Studies of Chemical Structure Biological Activity Relations Using Pattern Recognition. American Chemical Society, Washington D.C., 1979. 30
- S. Krause, E. Willighagen, and C. Steinbeck. JChemPaint - Using the Collaborative Forces of the Internet to Develop a Free Editor for 2D Chemical Structures. *Molecules*, 5:93–98, 2000. 29, 31
- R. Pearlman and K. Smith. Metric Validation and the Receptor-Relevant Subspace Concept. *J. Chem. Inf. Comput. Sci.*, 39:28–35, 1999. 30
- C. Steinbeck, Y. Han, S. Kuhn, O. Horlacher, E. Luttmann, , and E. Willighagen. The Chemistry Development Kit (CDK): An Open-Source Java Library for Chemo- and Bioinformatics. *J. Chem. Inf. Comput. Sci.*, 43:493–500, 2003. 29
- C. Steinbeck, C. Hoppe, S. Kuhn, M. Floris, R. Guha, and E. Willighagen. Recent Developments of the Chemistry Development Kit (CDK) - An Open-Source Java Library for Chemo- and Bioinformatics. *Curr. Pharm. Des.*, in press, 2006. 29
- D. Temple-Lang. <http://www.omegahat.org/RSJava>, September 2005. 29
- R. Todeschini and V. Consonni. *Handbook of Molecular Descriptors*. Wiley-VCH, Berlin, 2002. 30
- R. Todeschini, V. Consonni, and M. Pavan. *Dragon*, 2005. 30
- S. Urbanek. <http://stats.math.uni-augsburg.de/Rserve>, September 2005. 32

Rajarshi Guha
Pennsylvania State University
rxg218@psu.edu

Editor-in-Chief:

Paul Murrell
Department of Statistics
The University of Auckland
Private Bag 92019
Auckland

Editorial Board:

Torsten Hothorn and John Fox.

Editor Programmer's Niche:

Bill Venables

Editor Help Desk:

Uwe Ligges

Email of editors and editorial board:

firstname.lastname@R-project.org

R News is a publication of the R Foundation for Statistical Computing. Communications regarding this publication should be addressed to the editors. All articles are copyrighted by the respective authors. Please send submissions to regular columns to the respective column editor and all other submissions to the editor-in-chief or another member of the editorial board. More detailed submission instructions can be found on the R homepage.

R Project Homepage:

<http://www.R-project.org/>

This newsletter is available online at

<http://CRAN.R-project.org/doc/Rnews/>