Statistische Klassifikationsverfahren
Eine Fallstudie

Dr. Matthias Kohl
Lehrstuhl für Stochastik

29. Mai 2008

Inhaltsverzeichnis

1 Einleitung 2

2 Vorverarbeitung 3

3 Statistische Klassifikation 6

1 Einleitung

Wir wollen am Beispiel eines realen Datensatzes die Vorgehensweise für die Bestimmung eines Klassifikators demonstrieren. Wir wählen hierfür den Datensatz von Kirschner-Schwabe et al. (2006) [16]. Die Genexpressionsdaten von 60 in der Studie eingeschlossenen Patienten finden sich in der Gene Expression Omnibus (GEO) Datenbank [7] unter der Nummer GSE4698. Die Hybridisierungen wurden mittels dem "Affymetrix GeneChip Human Genome U133 Array Set HG-U133A" durchgeführt. Weitere Einzelheiten zu diesem Chip findet man unter http://www.affymetrix.com/support/technical/byproduct.affx?product=hgu133. Wir lesen die Daten aus den sog. CEL-Dateien ein. Jede der 60 Dateien ist ca. 11.5 MByte groß, was insgesamt nahezu 700 MByte an Daten ergibt. Für das Einlesen verwenden wir die Funktion `read.affybatch` aus dem Bioconductor-Paket *affy* [9]. Die Annotation – d.h., die Zuordnung der Genproben zu den Genen – findet sich für diesen Chip im Bioconductor-Paket *hgu133acdf* [10]. Da das Einlesen und die folgenden Berechnungen äußert zeitaufwendig sind, verwenden wir bei rechenintensiven Schritten zusätzlich das Bioconductor-Paket *weaver* [8], mit dem es möglich ist, innerhalb von Sweave [17] Ergebnisse zwischenzuspeichern.

```
R > library(affy)
R > library(hgu133acdf)
R > fn <- list.celfiles(path = "GSE4698_RAW/", full.names=TRUE)

R > rawData <- read.affybatch(filenames = fn)
```

Das Object `rawData` der S4-Klasse *AffyBatch* ist von folgender Dimension und Größe.

```
R > rawData

AffyBatch object
size of arrays=712x712 features (13 kb)
cdf=HG-U133A (22283 affyids)
number of samples=60
number of genes=22283
annotation=hgu133a
notes=

R > dim(exprs(rawData))

[1] 506944     60
```

Die klinischen Daten zu dieser Studie finden sich unter http://clincancerres.aacrjournals.org/cgi/data/12/15/4553/DC1/3. Wir haben diese in eine csv-Datei überführt und werden diese im Folgenden einlesen.

```
R > Patienten <- read.csv("PatientenCharakteristik.csv")
```

Die Daten von solchen Gen-Chips müssen vor einer weitergehenden Analyse einer gewissen Vorverarbeitung unterzogen werden, deren Ziel es ist system-bedingte Fehler zu reduzieren bzw. nach Möglichkeit weitestgehend zu eliminieren. Mehr hierzu findet man zum Beispiel unter http://compdiag.molgen.mpg.de/ngfn/docs/2008/mar/beissbarth_cDNA_QCPP_2008mar.pdf.

2 Vorverarbeitung

Wir verwenden zur Vorverarbeitung die Methode, die auch in der Publikation von Kirschner-Schwabe et al. (2006) [16] verwendet wurde. In einem ersten Schritt verwenden wir das Verfahren von Huber et al. (2002) [14], welches eine Varianzstabilisierung der Daten zum Ziel hat. Anschließend verwenden wir die Methode von Irizarry et al. [15], um die vorverarbeiteten Intensitäten der Genproben in Expressionsignale von Genen umzuwandeln. Diese Vorgehensweise ist in der Funktion `vsnrma` im Bioconductor [10]-Paket *vsn* [14] implementiert.

```
R > library(vsn)
```

```
R > normData <- vsnrma(rawData)
```

Nach der Vorverarbeitung erhalten wir ein Object `normData` der S4-Klasse *AffyBatch* von folgender Dimension und Größe

```
R > normData
```

```
ExpressionSet (storageMode: lockedEnvironment)
assayData: 22283 features, 60 samples
  element names: exprs
phenoData
  sampleNames: GSM106167.CEL, GSM106168.CEL, ..., GSM106226.CEL  (60 total)
  varLabels and varMetadata description:
    sample: arbitrary numbering
featureData
  featureNames: 1007_s_at, 1053_at, ..., AFFX-TrpnX-M_at  (22283 total)
  fvarLabels and fvarMetadata description: none
experimentData: use 'experimentData(object)'
Annotation: hgu133a
```

```
R > dim(normData)
```

```
Features  Samples
   22283       60
```

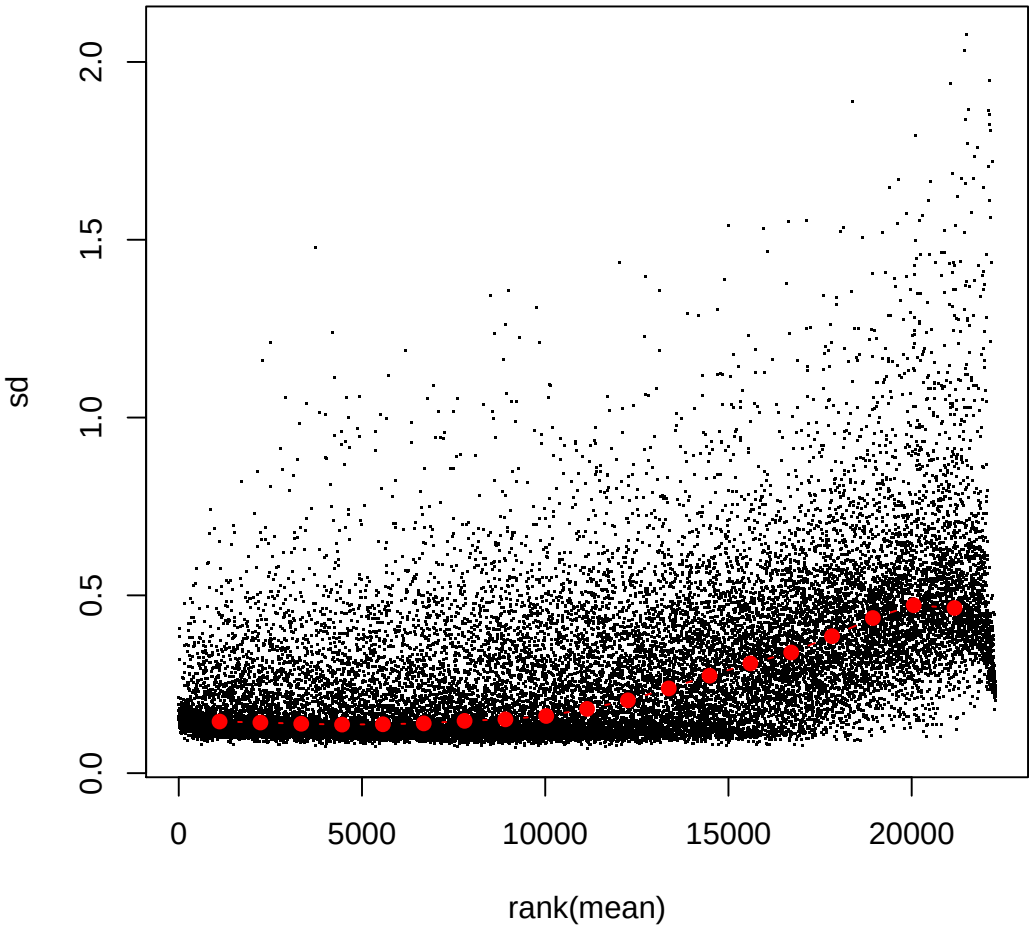
Da wir die Rohdaten nicht weiter benötigen, löschen wir diese, um den Arbeitsspeicher zu entlasten und räumen mittels der Funktion `gc` ("garbage collection") aus dem R-Paket *base* [21] den Arbeitsspeicher auf.

```
R > rm(rawData)
R > gc()
```

```
          used (Mb) gc trigger  (Mb) max used  (Mb)
Ncells  862780 23.1    2105982  56.3  2105982  56.3
Vcells 2644392 20.2   29729561 226.9 36463942 278.2
```

Ein sehr wichtiger Schritt in der Analyse solch komplexer Daten ist eigentlich auch immer die Qualitätskontrolle der Daten. Hierauf werden wir jedoch im Rahmen dieser Demonstration nicht näher eingehen. Wir werden lediglich im Folgenden mittels eines sog. mean-sd-Plots kontrollieren, wie gut die Varianzstabilisierung der Daten funktioniert hat.

```
R > meanSdPlot(normData)
```



Wir sehen, dass die Streuung für einen gewissen Teil der gemessenen Intensitäten nahezu konstant ist, dass diese jedoch ab einer Intensität von ca. 10000 stetig zunimmt.

Vor der eigentlichen statistischen Analyse ist es üblich, die Dimension der Daten zu reduzieren, indem man gewisse Gene wegläßt. Ein Möglichkeit die uninteressanten Gene herauszufiltern ist mit Hilfe der Intensität und Variabilität. Wir verwenden dazu das Bioconductor-Paket *genefilter* [12]. Wir fordern, dass die Intensität in 25% der Fälle größer als 100 sein muss sowie, dass der IQR größer als 0.5 sein muss.

```
R > library(genefilter)
R > f1 <- pOverA(0.25, log2(100))
R > f2 <- function(x) (IQR(x) > 0.5)
R > ff <- filterfun(f1, f2)
```

```
R > auswahl <- genefilter(normData, ff)
R > normData.ausw <- normData[auswahl, ]
```

Es verbleiben also 5325 Gene für die weitere statistische Analyse.

3 Statistische Klassifikation

Aufgrund der großen Dimension der Daten, können die meisten Klassifikationsverfahren nicht direkt auf die Daten angewendet werden. Ein Grund dafür ist der sog. "Fluch der Dimensionen": mit wachsender Dimensionalität werden die Abstände zwischen den verschiedenen Klassen immer ähnlicher. Hinzukommt, dass verrauschte und irrelevante Variablen diese Effekt noch verstärken und es für ein Klassifikationsverfahren schwierig machen, Entscheidungsgrenzen festzulegen. Ein weiterer Grund warum Klassifikationsalgorithmen nicht direkt auf den vollen Daten anwendbar sind, ist der hohe Rechenaufwand, der nötig wäre. Folglich ist es nötig, geeignete Strategien für eine Vorauswahl von interessanten Variablen zu entwickeln. Ein Überblick über verschiedene Möglichkeiten geben Hall und Holmes (2003) [11]. Im allgemeinen wird zwischen sog. Filtern und Wrappern unterschieden.

Unter Filtern versteht man Ansätze, welche anhand irgendeines Kriteriums die Fähigkeit der Variablen zur Unterscheidung der Gruppen beurteilen. Innerhalb der Filteransätze kann man weiterhin zwischen Methoden unterscheiden, die auf Anordnungen basieren und solchen, die Teilmengen auswählen. Methoden, die auf Anordnungen basieren, bewerten die Nützlichkeit jeder Variablen unabhängig von den anderen Variablen. Als Ergebnis erhält man eine geordnete Liste. Die Verfahren, die auf Anordnungen basieren, sind sehr effizient, sie vernachlässigen jedoch Interaktionen und Korrelationen zwischen den Variablen.
Die zweite Variante der Filter, beurteilt die Nützlichkeit von Teilmengen. Somit bleibt im Prinzip die Information über die Interaktion der Variablen erhalten, jedoch vergrößert sich der Raum, in dem man sucht von Dimension d (Anzahl der Variablen) auf 2^d (Menge aller Teilmengen) bzw. zumindest $O(2^d)$. Folglich können für hoch-dimensionale Daten nur sehr einfache Strategien wie etwa die Vorwärtssuche zur Variablenselektion verwendet werden.

Die zweite Möglichkeit, um Variablen auszuwählen, sind die sog. Wrapper. In diesem Fall wird ein Klassifikator verwendet, um Teilmengen der Variablen zu bewerten. Mit Hilfe von Kreuzvalidierung wird die Klassifikationsgenauigkeit für eine konkrete Teilmenge bestimmt. Häufig stellt man fest, dass Wrapper zu einer höheren Klassifikationsgenauigkeit als Filter führen; vgl. Pochet et al. (2004) [20].

Für die Klassifikation wir das Bioconductor-Paket *MCRestimate* [22], welches leider unter der aktuellen Bioconductor-Version 2.2 nicht zur Verfügung steht, sondern entweder aus der alten Version 2.1 oder der Entwicklerversion 2.3 bezogen werden muss. Ähnlich verhält es sich mit dem Bioconductor-Paket *arrayMagic* [2], welches für die Installation von *MCRestimate* benötigt wird und welches weder unter 2.2 noch 2.3 zur Verfügung steht, sondern nur über die alte Version 2.1 bezogen werden kann. Ein möglicher Installationsbefehl ist somit

```
R > install.packages("MCRestimate",
+      repos = c("http://cran.at.r-project.org",
+                "http://www.bioconductor.org/packages/2.3/bioc/",
+                "http://bioconductor.org/packages/2.2/data/experiment/",
+                "http://www.bioconductor.org/packages/2.1/bioc/"))
```

Um die Funktion `MCRestimate` zu verwenden, müssen wir noch ein paar Vorbereitungen vornehmen.

```
R > library(MCRestimate)
R > group <- Patienten[,"Time.of.relapse"]
R > levels(group) <- c("very early", "early", "late")
R > pD0 <- data.frame(group = group,
+                     row.names = sampleNames(normData.ausw))
R > pD <- new("AnnotatedDataFrame", data = pD0,
+           varMetadata = data.frame(labelDescription = "very early/early/late"))
R > phenoData(normData.ausw) <- pD
```

Wir berechnen nun einen Klassifikator für die Daten unter Verwendung der Funktion `MCRestimate`. Zur Selektion der Variablen verwenden wir die Funktion `varSel.highest.var`, welche die Variablen auswählt, welche die größte Variabilität zeigen. Wir geben vor, dass zur Klassifikation 20 Variablen (Gene) verwendet werden sollen. Als Klassifikationsverfahren verwenden wir random forests [1], welches innerhalb von `MCRestimate` mittels der Funktion `RF.wrap` aufgerufen wird. Wir schätzen die Klassifikationsgenauigkeit mittels 20 Wiederholungen einer 5-fach Kreuzvalidierung.

```
R > rf.cv <- MCRestimate(eset = normData.ausw,
+                        class.column = "group",
+                        classification.fun = "RF.wrap",
+                        poss.parameters = list(var.numbers = 20),
+                        variableSel.fun = "varSel.highest.var",
+                        cross.outer = 5, cross.inner = 1,
+                        cross.repeat = 20,
+                        plot.label = "group", rand = 111)
```

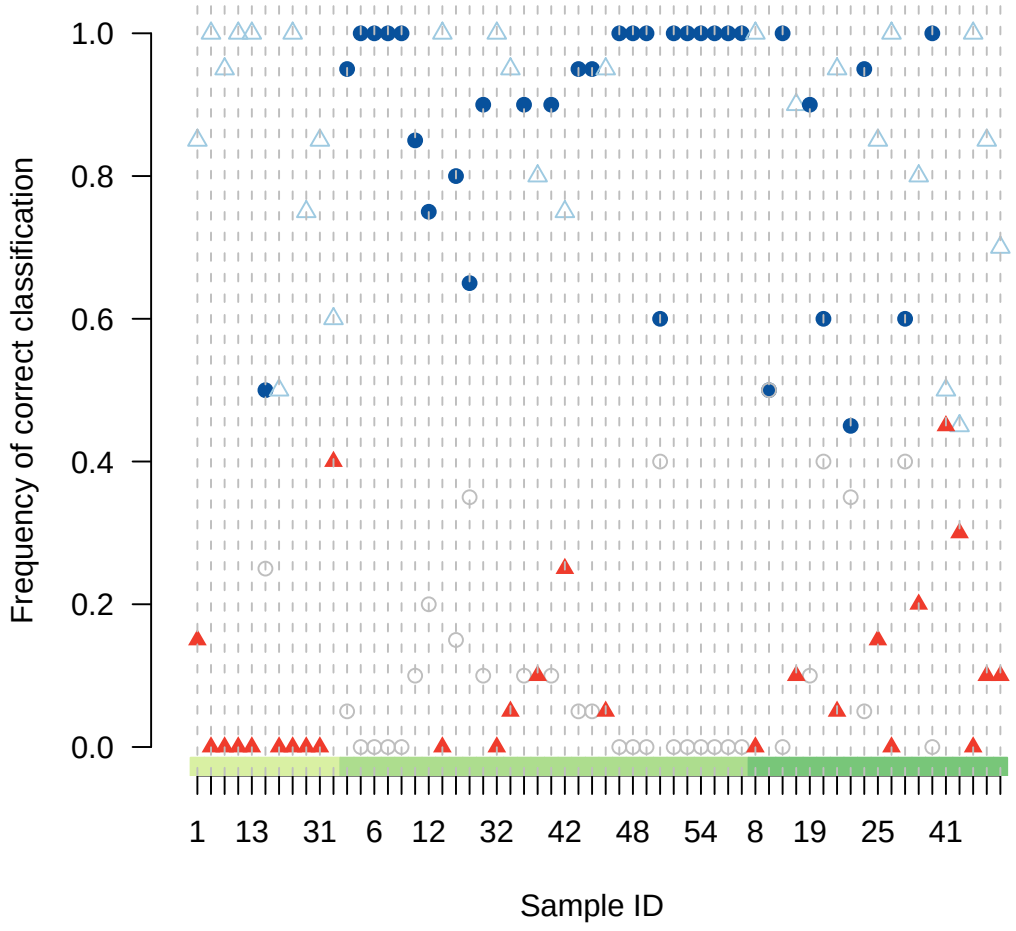
Wir erhalten das folgende Ergebnis.

```
R > rf.cv


Result of MCRestimate with 20 repetitions of 5-fold cross-validation

Preprocessing function 1 : varSel.highest.var
Preprocessing function 2 : varSel.identity
Classification function  : RF.wrap
```

```
The confusion table:
           early late very early class error
early         24    6          0       0.200
late          11    8          0       0.579
very early     7    3          1       0.909
```

Das Paket *MCRestimate* beinhaltet auch eine Möglichkeit der graphischen Darstellung für das Ergebnis – einen sog. vote-Plot.

```
R > plot(rf.cv)
```



Wir verwenden ein anderes Klassifikationsverfahren, nämlich support vector machines mit den Default-Parametern.

```
R > svm.cv <- MCRestimate(eset = normData.ausw,
+                         class.column = "group",
+                         classification.fun = "SVM.wrap",
+                         poss.parameters = list(var.numbers = 20),
+                         variableSel.fun = "varSel.highest.var",
+                         cross.outer = 5, cross.inner = 1,
+                         cross.repeat = 20,
+                         plot.label = "group", rand = 111)
```

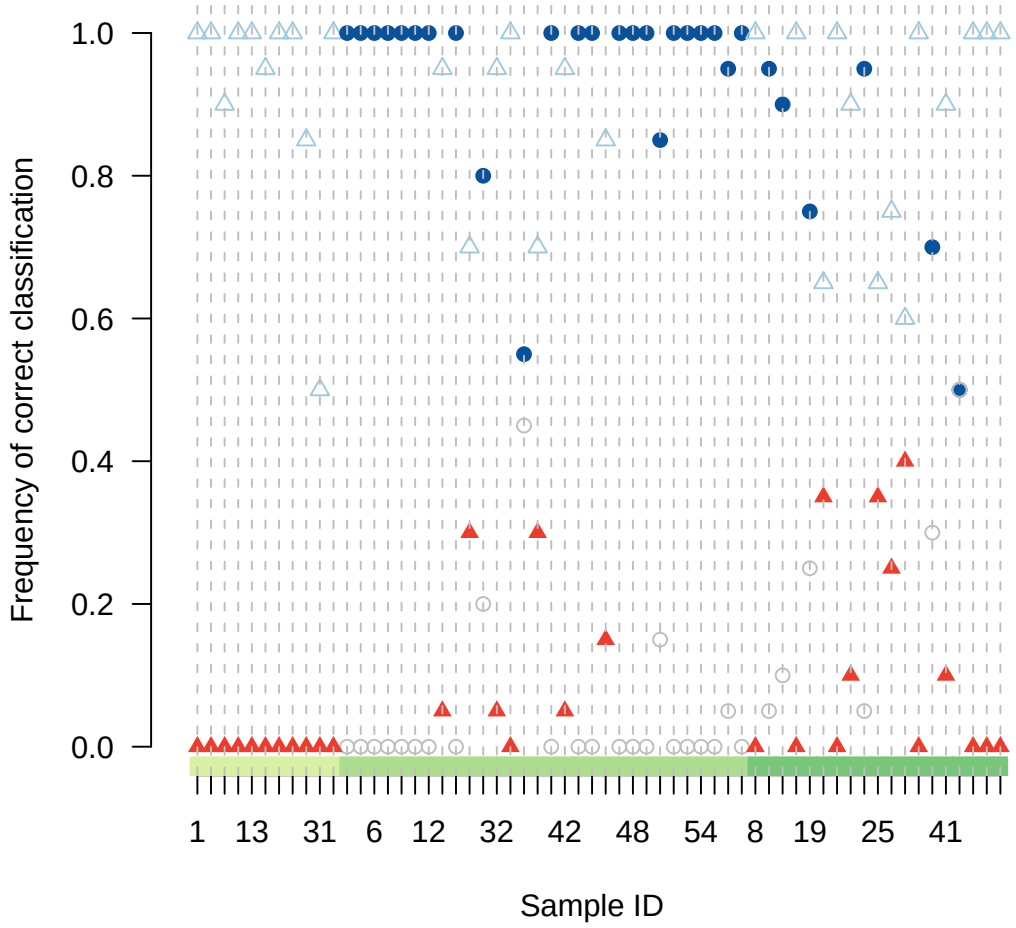
Wir erhalten das folgende Ergebnis.

```
R > svm.cv
```

```
Result of MCRestimate with 20 repetitions of 5-fold cross-validation

Preprocessing function 1 : varSel.highest.var
Preprocessing function 2 : varSel.identity
Classification function  : SVM.wrap

The confusion table:
           early late very early class error
early         23    7          0       0.233
late          13    6          0       0.684
very early     9    2          0       1.000
```

Wir stellen das Ergebnis wieder graphisch dar.

```
R > plot(svm.cv)
```



In beiden Fällen erhalten wir einen hohen Klassifikationsfehler. Wir gehen über zu einem anderen Filter. Wir verwenden den paarweisen Mann-Whitney Test für die Auswahl der "besten" Gene.

```
R > library(MCRestimate)
R > my.wtest <- function(x, classfactor){
+     res <- pairwise.wilcox.test(x, classfactor)[["p.value"]]
+     res <- res[lower.tri(res, diag = TRUE)]
+     return(res)
+ }
R > varSel.wilcox <- function (sample.gene.matrix, classfactor,
+                               theParameter = NULL, var.numbers = 18, ...){
```

```
+     if (is.null(theParameter)) {
+         genes.sd <- apply(sample.gene.matrix, 1, sd)
+         genes.mean <- rowMeans(sample.gene.matrix)
+         genes.select <- order(genes.sd/genes.mean, decreasing = TRUE)[1:1000]
+         scores <- apply(sample.gene.matrix[genes.select, ], 1,
+                         my.wtest, classfactor)
+         anz <- var.numbers %/% 3
+         selection1 <- order(scores[1,])[1:anz]
+         selection2 <- order(scores[2,])[1:anz]
+         selection3 <- order(scores[3,])[1:anz]
+         selection <- union(selection1, selection2)
+         selection <- union(selection, selection3)
+         theParameter <- rep(TRUE, nrow(sample.gene.matrix))
+         theParameter[genes.select][selection] <- FALSE
+     }
+     train.matrix <- sample.gene.matrix[!theParameter, , drop = FALSE]
+     return(list(matrix = train.matrix, parameter = theParameter))
+ }
```

Wir verwenden zuerst wieder random forests für die Klassifikation.

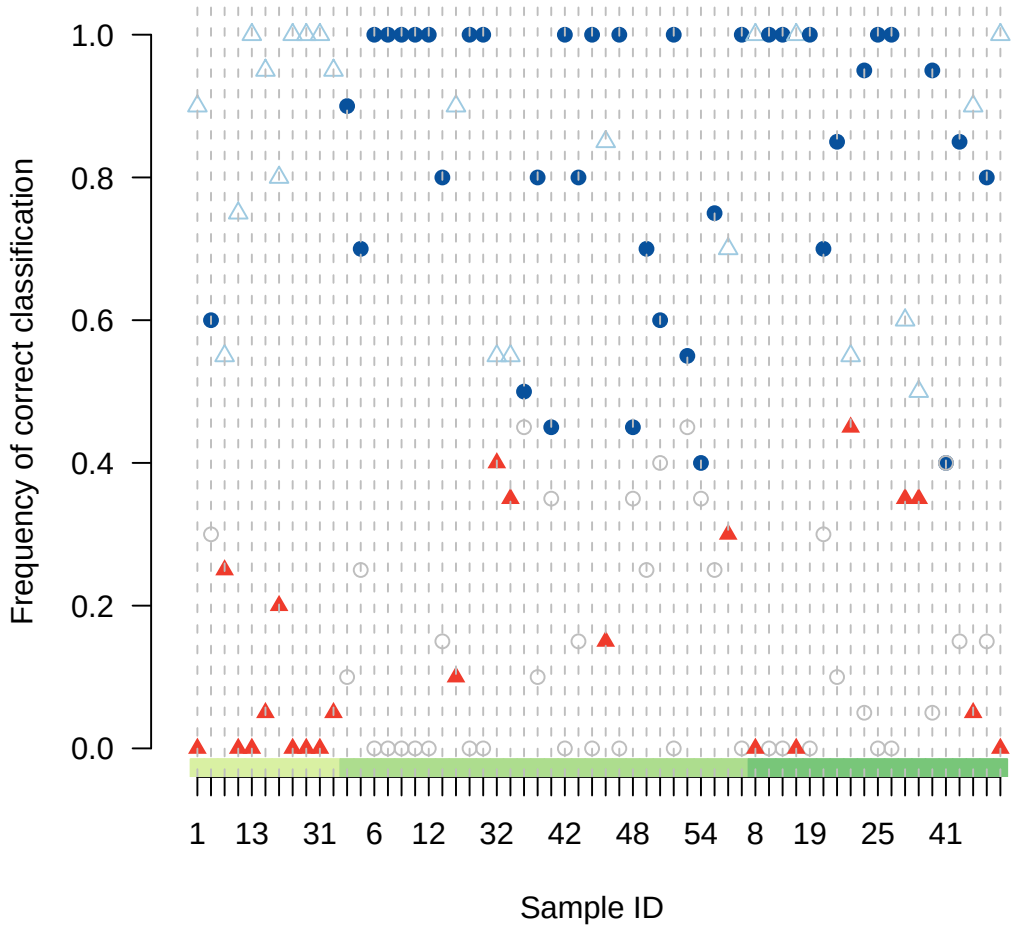
```
R > rf.cv1 <- MCRestimate(eset = normData.ausw,
+                         class.column = "group",
+                         classification.fun = "RF.wrap",
+                         poss.parameters = list(var.numbers = 21),
+                         variableSel.fun = "varSel.wilcox",
+                         cross.outer = 5, cross.inner = 1,
+                         cross.repeat = 20,
+                         plot.label = "group", rand = 111)
```

Wir erhalten das folgende Ergebnis.

```
R > rf.cv1

Result of MCRestimate with 20 repetitions of 5-fold cross-validation

Preprocessing function 1 : varSel.wilcox
Preprocessing function 2 : varSel.identity
Classification function  : RF.wrap

The confusion table:
           early late very early class error
early         25    3          2       0.167
late           7   12          0       0.368
very early     8    2          1       0.909
```

Wir plotten das Ergebnis.

```
R > plot(rf.cv1)
```



Im zweiten Schritt verwenden wir wieder support vector machines für die Klassifikation.

```
R > svm.cv1 <- MCRestimate(eset = normData.ausw,
+                          class.column = "group",
+                          classification.fun = "SVM.wrap",
+                          poss.parameters = list(var.numbers = 21),
+                          variableSel.fun = "varSel.wilcox",
+                          cross.outer = 5, cross.inner = 1,
```

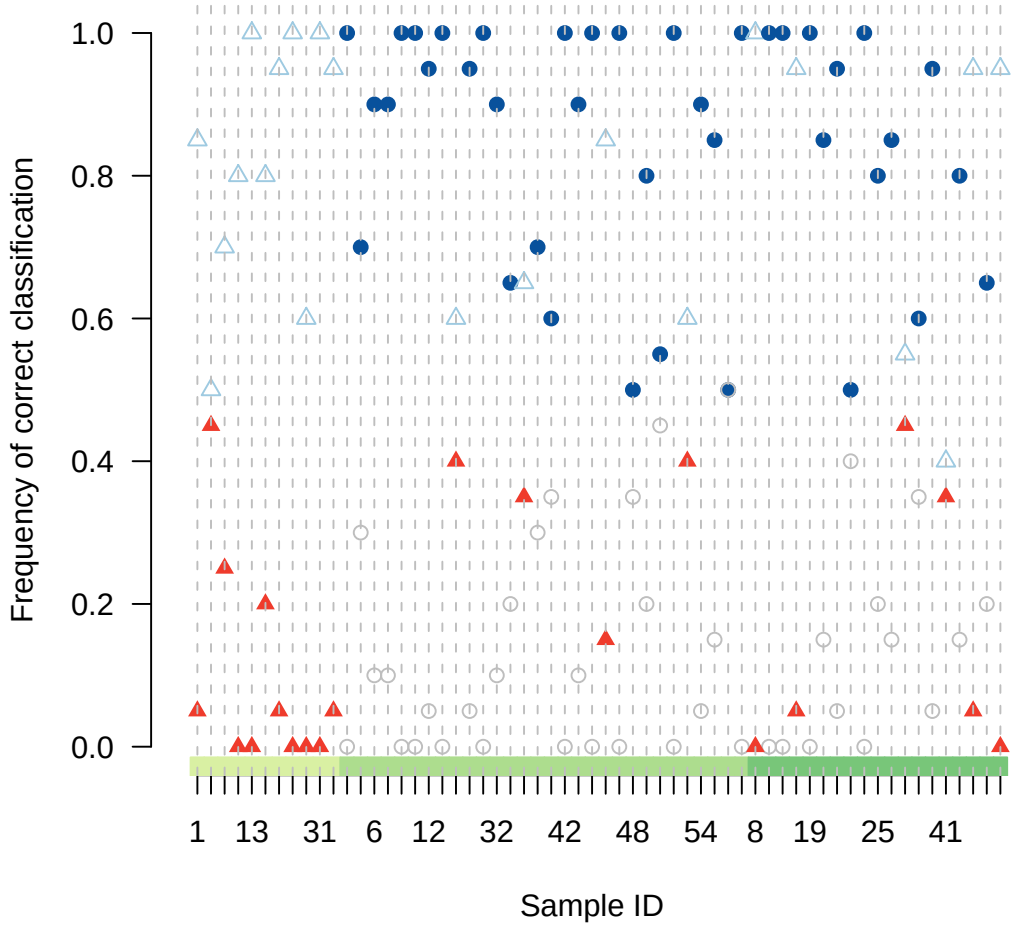
```
+                       cross.repeat = 20,
+                       plot.label = "group", rand = 111)
```

Wir erhalten das folgende Ergebnis.

```
R > svm.cv1


Result of MCRestimate with 20 repetitions of 5-fold cross-validation

Preprocessing function 1 : varSel.wilcox
Preprocessing function 2 : varSel.identity
Classification function  : SVM.wrap

The confusion table:
           early late very early class error
early         26    2          2       0.133
late           5   13          1       0.316
very early     9    2          0       1.000
```

Wir stellen das Ergebnis wieder graphisch dar.

```
R > plot(svm.cv1)
```



Wir erreichen durch den neuen Filter einen etwas kleineren Klassifikationsfehler. **Abschließende Bemerkungen:**

1. Im Fall derart komplexer Daten gibt es eine nahzu unbegrenzte Anzahl von Möglichkeiten, eine Klassifikation durchzuführen. Wir haben die Auswahl über verschiedene Filter oder Wrapper, die Anzahl der Variablen, das Klassifikationsverfahren (inkl. entsprechender Tuningparameter) und nicht zu vergessen, die verschiedenen Möglichkeiten, die Daten vorzuverarbeiten.

2. Im vorliegenden Fallbeispiel haben wir einen Bias in Kauf genommen, dadurch, dass wir die Vorverarbeitung der Daten simultan – die Arrays wurden nicht einzeln, sondern als Batch vorverarbeitet – für den gesamten Datensatz vorgenommen

haben. Eigentlich müßte man in die Kreuzvalidierung auch die Vorverarbeitung der Daten einschließen, da sonst die Trainingsdaten über das Vorverarbeiten Einfluss auf die Testdaten nehmen.

Literatur

[1] Breiman, L. (2001). Random Forests. Machine Learning 45(1), 5-32. 7

[2] Andreas Buness, Wolfgang Huber, Klaus Steiner, Holger Sueltmann, and Annemarie Poustka (2004). arrayMagic: two-colour cDNA microarray quality control and preprocessing. Bioinformatics Advance Access published on September 28, 2004. doi:10.1093/bioinformatics/bti052 6

[3] Vince Carey and Henning Redestig for C++ language enhancements (2008). ROC: utilities for ROC, with uarray focus. R package version 1.14.0. http://www.bioconductor.org

[4] Evgenia Dimitriadou, Kurt Hornik, Friedrich Leisch, David Meyer and Andreas Weingessel (2008). e1071: Misc Functions of the Department of Statistics (e1071), TU Wien. R package version 1.5-18.

[5] Ding, B.Y. and Gentleman, R. (2004). Classification using generalized partial least squares. Bioconductor Project Working Papers. Paper 5. http://www.begress.com/bioconductor/paper5

[6] S. Dudoit, J. Fridlyand, and T. P. Speed (2000). Comparison of Discrimination Methods for the Classification of Tumors Using Gene Expression Data. June 2000. (Statistics, UC Berkeley, Tech Report #576).

[7] Edgar R, Domrachev M, Lash AE (2002). Gene Expression Omnibus: NCBI gene expression and hybridization array data repository Nucleic Acids Res. 2002 Jan 1;30(1):207-10. 2

[8] Seth Falcon (2008). weaver: Tools and extensions for processing Sweave documents. R package version 1.6.0. 2

[9] Gautier, L., Cope, L., Bolstad, B. M., and Irizarry, R. A. (2004). affy – analysis of Affymetrix GeneChip data at the probe level. Bioinformatics 20, 3 (Feb. 2004), 307-315. 2

[10] Robert C. Gentleman and Vincent J. Carey and Douglas M. Bates and others (2004). Bioconductor: Open software development for computational biology and bioinformatics *Genome Biology*, 5:R80. http://genomebiology.com/2004/5/10/R80 2, 3

[11] M. A. Hall and G. Holmes (2003). Benchmarking attribute selection techniques for discrete class data mining. IEEE Transactions on Knowledge and Data Engineering, 15(6): 1437-1447, 2003. http://www.cs.waikato.ac.nz/~mhall/HallHolmesTKDE.pdf 6

[12] R. Gentleman, V. Carey, W. Huber and F. Hahne (2008). genefilter: methods for filtering genes from microarray experiments. R package version 1.20.0. 5

[13] T. Hastie, R. Tibshirani, J. Friedman (2001). The Elements of Statistical Learning: Data Mining, Inference and Prediction. Springer.

[14] Huber W, von Heydebreck A, Sueltmann H, Poustka A, Vingron M (2002). Variance Stabilization Applied to Microarray Data Calibration and to the Quantification of Differential Expression. Bioinformatics 18, S96-S104 (2002). 3

[15] Irizarry RA, Bolstad BM, Collin F, Cope LM, Hobbs B, Speed TP (2003). Summaries of Affymetrix GeneChip probe level data. Nucleic Acids Res 2003; 31:15. 3

[16] R. Kirschner-Schwabe R, C. Lottaz C et al. (2006). Expression of late cell cycle genes and an increased proliferative capacity characterize very early relapse of childhood acute lymphoblastic leukemia. Clin Cancer Res. 2006 Aug 1; 12(15):4553-61. http://clincancerres.aacrjournals.org/cgi/content/full/12/15/4553 http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE4698 2, 3

[17] Friedrich Leisch (2002). Dynamic generation of statistical reports using literate data analysis. In W. Haerdle and B. Roenz, editors, Compstat 2002 - Proceedings in Computational Statistics, pages 575-580. Physika Verlag, Heidelberg, Germany. ISBN 3-7908-1517-9. 2

[18] Erich Neuwirth (2005). RColorBrewer: ColorBrewer palettes. R package version 0.2-3.

[19] Andrea Peters and Torsten Hothorn (2007). ipred: Improved Predictors. R package version 0.8-5.

[20] Pochet, N., De Smet, F., Suykens, J.A., and De Moor, B.L. (2004). Systematic benchmarking of microarray data classification: assessing the role of nonlinearity and dimensionality reduction. Bioinformatics, 20(17):3185-95 (2004). http://bioinformatics.oxfordjournals.org/cgi/reprint/bth383v1.pdf 6

[21] R Development Core Team (2008). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org 4

[22] Markus Ruschhaupt, Ulrich Mansmann, Patrick Warnat, Wolfgang Huber and Axel Benner (2008). MCRestimate: Misclassification error estimation with cross-validation. R package version 1.11.7. 6

[23] Tobias Sing, Oliver Sander, Niko Beerenwinkel and Thomas Lengauer (2007). ROCR: Visualizing the performance of scoring classifiers. R package version 1.0-2. http://rocr.bioinf.mpi-sb.mpg.de/

[24] V. Vapnik (1999). The Nature of Statistical Learning Theory. Springer, New York.

[25] Venables, W. N. and Ripley, B. D. (2002). Modern Applied Statistics with S. Fourth Edition. Springer, New York. ISBN 0-387-95457-0.

[26] Sanford Weisberg, (2007). dr: Methods for dimension reduction for regression. R package version 3.0.2.