Vorwort

Die erste Fassung dieses Skripts wurde von Matthias Brandt (Student im WS 2011/2012) erstellt. Überarbeitet und erweitert wurde es von Dr. Rainer Schmidt und Prof. Dr. Georg Füllen. Im Winter 2012/2013 wurde das Script von Melina Schellhorn (zu *Kürzesten Wegen, Maximum Likelihood und Single Nucleotide Polymorphismen)* erweitert. Im Winter 2014/2015 wurde Kapitel 5 zur Informationsbeschaffung neu eingefügt, im Winter 2015/2016 wurde dieses zu Web-Ressourcen ergänzt, es gab unzählige kleinere Verbesserungen im Skript und die Reihenfolge der Kapitel 1 und 4 wurde getauscht.

Das Skript ist angelehnt an die Vorlesung zur Bioinformatik/Medizininformatik von Prof. Füllen, die ein Teil der Vorlesung „Epidemiologie, Medizinische Biometrie und Medizinische Informatik“ ist.

Der erste Teil „Biomarker-Finden mit R“ basiert teilweise auf dem Buch „Applied Statistics for Bioinformatics using R“ von Wim P.Krijnen. In diesem Teil erfolgt auch eine Einführung in die Programmiersprache „R“, mit der nicht nur viele Rechnungen im Bereich der Bioinformatik durchgeführt werden, sondern auch die typischen statistischen Untersuchungen für medizinische Promotionen im Bereich Biostatistik/Biometrie möglich sind.
Der dritte Teil basiert teilweise auf dem Buch „Algorithmen und Datenstrukturen - Eine Einführung mit Java“ von Saake/Sattler und auf Materialien von R. König aus Heidelberg (mit freundlicher Genehmigung).

Auf den folgenden Seiten soll es um die Bioinformatik gehen. Angelehnt an die Vorlesung haben wir uns mit den Folien (siehe: http://www.ibima.med.uni-rostock.de/IBIMA/ unter „Teaching1“) auseinandergesetzt und versucht, jeweils zu erklären, worum es geht. Somit sollte es mit diesem Script möglich sein, den hier behandelten Teil der Vorlesung eigenständig nachzuvollziehen. Doch vorweg erst einmal die Antwort auf die zwei wichtigsten Fragen zu diesem Thema:

Was ist Bioinformatik?

Bioinformatik ist ein Teilbereich der Informatik, welcher mit Hilfe von Programmen Probleme und Fragestellungen der Biologie und Medizin zu lösen versucht.

Was geht mich das an?

Im Rahmen heutiger Fragestellungen, bei denen es nicht mehr nur darum geht, ob z. B. ein Medikament wirkt oder nicht, sondern auch danach gefragt wird, wo es wirkt (Rezeptoren, Carrier, …) und wie genau es dort angreift, kommt man an der Visualisierung und Berechnung derartiger Zusammenhänge einfach nicht mehr vorbei. In vielen Fällen sind die Datenmengen, die mit modernen Geräten gewonnen werden, so groß, dass sie ohne vorherige Sortierung und Organisierung gar nicht greifbar (und somit auswertbar) sind. Jede Information ist immer nur soviel wert wie der Zusammenhang, in den sie gesetzt wird.
Genau hier greift die Bioinformatik an und versucht mit Hilfe von Berechnungen Ordnung ins Chaos zu bringen. Dass uns dies noch nicht täglich in der Klinik begegnen wird ist klar. Wenn es jedoch darum geht, Forschung zu betreiben oder zu verstehen, woher die Daten von Genanalysen stammen und was damit gemacht werden kann, ist es recht hilfreich, einen Einblick bekommen zu haben.

In diesem Sinne hoffen wir, dass wir all jenen weiterhelfen können, die Probleme mit dem Themengebiet haben!

Am Ende der Kapitel befinden sich Übungsaufgaben. Hierbei handelt es sich um Multiple-Choice-Aufgaben, bei denen jeweils nur eine Antwort richtig ist. Die Lösungen zu den Übungsaufgaben befinden sich im Anhang.

Der Inhalt wurde grundsätzlich auf Korrektheit geprüft. Es kann jedoch hier und da Fehler geben, genauso wie in den Folien zur Vorlesung. Wir bitten um Hinweise.

Prof. Georg Füllen: fuellen@uni-rostock.de

Dr. Rainer Schmidt: rainer.schmidt@uni-rostock.de

1. Biomarker-Finden mit R

1.1 Einleitung

R ist eine Programmiersprache, insbesondere für die Bioinformatik. Die folgenden Beispiele mit R stammen aus dem Buch von Wim P. Krijnen: „Applied Statistics for Bioinformatics using R“. Dieses ist als PDF-Datei verfügbar: http://cran.r-project.org/doc/contrib/Krijnen-IntroBioInfStatistics.pdf.

Um Beispiele in R auszuprobieren, ist zunächst die Programmiersprache R auf dem Rechner zu installieren. Empfehlenswert ist die Installation einer R Umgebung (z.B. „RStudio“).
Zur Installation sollten die Hinweise aus Kapitel 1.1 des oben genannten Buchs von Krijen beachtet werden.

R ist eine freie Software, zu der verschiedene Nutzergruppen immer wieder neue Funktionen und Programme, sogenannte Pakete bzw. „packages“, bereitstellen können. Um diese zu nutzen, sind sie zu installieren.
Mit dem folgenden Befehl wird das Paket „TeachingDemos“ von der Internetseite http://cran.r-project.org heruntergeladen und in R installiert.

```
> install.packages(c("TeachingDemos"),repo="http://cran.r-project.org",dep=TRUE)
```

Die Installation kann auch manuell geschehen: Das gewünschte Paket aus dem Internet herunterladen, ggf. entpacken und in den Ordner „Library“ des R-Programms verschieben.

Dem zu erstellenden R Programm muss die Benutzung dieses Paketes bekannt gemacht werden:

```
> library(TeachingDemos)
```

Aufgrund der durch Nutzergruppen immer wieder neu erstellten Pakete stehen in R sehr viele Funktionen zur Verfügung. Daneben gibt es aber auch einfache Standardfunktionen, von denen einige hier zunächst vorgestellt werden.

Das Zeichen „>“ fragt den Benutzer nach dem nächsten Kommando. Es folgt etwas, das ausgeführt werden soll.
Nach einem Kommando und nach Funktionen kann man suchen, z.B. gibt das folgende Kommando die Antwort „sd“ als Funktionsnamen für die Standardabweichung zurück:

```
> help.search("standard deviation")
```

Es folgen ein paar Beispiele für einfache Kommandos/Funktionsaufrufe

Die Addition:
> 2+3

liefert die Antwort:
[1] 5

Die [1] bedeutet, dass der folgende Wert, der erste Wert des Ergebnisses ist.
Dies mag zunächst unsinnig erscheinen, der Sinn wird aber später bei umfangreicheren Ergebnissen deutlich.

Eine Liste von Zahlen, hier ganze Zahlen von 1 bis 5:
> 1:5
[1] 1 2 3 4 5

Die Summe, hier von 1 bis 5:
> sum(1:5)
[1] 15

Das Produkt von 1 bis 5, d.h. die Fakultät: 5!
> prod(1:5)
[1] 120

Die Konkatenation, d.h. die Zusammenfassung von Werten zu einer Liste mit der Funktion „c“:
> c(1.00, 1.50, 1.25)
[1] 1.00 1.50 1.25

Die Zuweisung eines Wertes an eine Variable:
> a <- 5
Die Variable „a“ erhält so den Wert 5.

Zuweisung und Konkatenation:
Zunächst wird wie oben mit Hilfe der Funktion „c“ konkateniert. Anschließend wird das Ergebnis der Variablen „gene1“ zugewiesen.
> gene1 <- c(1.00, 1.50, 1.25)

Die Anfrage:
> gene1

Liefert dann:
[1] 1.00 1.50 1.25

Mit Variablen als Argumente lassen sich Funktionen ausführen. Die Variable „gene1“ enthält eine Folge von Zahlen, sodass die folgenden Funktionsaufrufe möglich sind:

```
> sum(gene1)
[1] 3.75
```

Der Mittelwert:

```
> sum(gene1)/3
[1] 1.25
```

oder so:

```
> mean(gene1)
[1] 1.25
```

Die Standardabweichung:

```
> sqrt(sum((gene1-mean(gene1)) ^ 2)/2)
[1] 0.25
```

oder einfacher so:

```
> sd(gene1)
[1] 0.25
```

Die Funktion „sqrt“ steht für „square root“ und die Funktion „sd“ für „standard deviation“.

Schrittweise Erstellung einer Matrix:

Zunächst werden weitere Variablen erzeugt und es werden Werte zugewiesen. Jede Variable entspricht einer Zeile der zu erzeugenden Matrix. Jeder Variablen werden per Konkatenation die Zahlenwerte zugewiesen.

```
> gene2  <-  c(1.35,1.55,1.00)
> gene3  <-  c(-1.10,-1.50,-1.25)
> gene4  <-  c(-1.20,-1.30,-1.00)
```

Die Zeilen (rows) und Spalten(columns) der Matrix sollen Namen erhalten. Dies geschieht mit Hilfe der Funktion „list“, die zwei Listen verbindet und der Variablen „rowcolnames“ zuweist. In der ersten Liste werden die Zeilennamen (“gene1“,“gene2“,“gene3“,“gene4“) und in der zweiten Liste die Spaltennamen (die Namen der Patienten) benannt.

```
> rowcolnames  <- list(c(“gene1“,“gene2“,“gene3“,“gene4“),c(“Eric“,“Peter“,“Anna“))
```

Der Wert der Variablen „rowcolnames“ wird im nächsten Schritt dem Parameter „dimnames“ der Funktion „matrix“ gleich gesetzt. Die Funktion „matrix“ ist so definiert, dass der Parameter „dimnames“ zwei Listen erhält, die erste für die Zeilennamen, die zweite für die

Spaltennamen. Für die Zeilennamen werden dabei die Strings (Zeichenketten) “gene1“ bis “gene4“ konkateniert.
(Achtung: In “ “ stehen immer Strings, die nicht mit Variablen zu verwechseln sind. “gene1“ ist also ein String und nicht zu verwechseln mit der Variablen gene1.)

Jetzt können wir die Matrix konstruieren:

```
> gendat  <- matrix(c(gene1,gene2,gene3,gene4),nrow=4,ncol=3,byrow=TRUE,
dimnames=rowcolnames)
```

Die Matrix besteht aus den Zeilen gene1, gene2, gene3 und gene4. Sie wird also zeilenweise aufgebaut (byrow=TRUE) und besteht aus vier Zeilen (nrow=4) und drei Spalten (ncol=3).

```
>gendat
```
liefert dann diese Matrix:

```
       Eric Peter  Anna
gene1  1.00  1.50  1.25
gene2  1.35  1.55  1.00
gene3 -1.10 -1.50 -1.25
gene4 -1.20 -1.30 -1.00
```

Mit der allgemeinen Funktion „apply“ können auf die Matrix „gendat“ Funktionen angewendet werden, z.B. die Berechnung des Durchschnitts mittels „mean“ als Parameter von „apply“.

Dies geht für die Zeilen der Matrix:

```
> apply(gendat,1,mean)
```

```
gene1       gene2       gene3       gene4
1.250000    1.300000   -1.283333   -1.166667
```

und für die Spalten der Matrix:

```
> apply(gendat,2,mean)
```

```
Eric    Peter   Anna
0.0125 0.0625 0.0000
```

Der Unterschied zwischen beiden Aufrufen der Funktionen “apply” ist der Wert des zweiten Parameters. Die Funktion „apply“ ist so definiert, dass als erster Parameter eine Liste oder eine Matrix anzugeben ist. Wenn es sich dabei speziell um eine Matrix handelt, dann verlangt (laut Definition der Funktion „apply“) eine 1 im zweiten Parameter eine zeilenweise Betrachtung und eine 2 eine spaltenweise Betrachtung. Im dritten Parameter steht die

anzuwendende Funktion. Optional können weitere Parameter folgen, in denen ggf. Argumente der anzuwendenden Funktion stehen.

Die Ergebnisse eines Aufrufs von „apply“ können wiederum einer Variablen zugewiesen werden:

```
> meanexprsval  <- apply(gendat,1,mean)

 gene1      gene2      gene3      gene4
1.250000  1.300000 -1.283333 -1.166667
```

Mit Hilfe der Funktion „order“ kann sortiert werden. Durch „decreasing=TRUE“ wird festgelegt, dass nach absteigender Reihenfolge sortiert wird:

```
> od  <- order(meanexprsval, decreasing=TRUE)
> od

[1] 2 1 4 3
```

Dieses Ergebnis gibt die Positionen in absteigenden Reihenfolge sortiert nach Mittelwerten für “gene1“,“gene2“,“gene3“,“gene4“ an. An erster Stelle steht die Position „2“, an der in der Matrix “gene2“ steht, weil dieses Gen mit 1.30 den höchsten Mittelwert besitzt.

Matrizen lassen sich zeilen- und spaltenweise sortieren. Mit [,] lassen sich quasi zwei durch Komma getrennte Sortiermöglichkeiten durchführen, zeilenweise und spaltenweise. Die Matrix „gendat“ lässt sich wie folgt zeilenweise nach absteigendem Mittelwert (in der Variablen „od“) sortieren, wobei die zweite Möglichkeit (für die Spalten) nicht genutzt wird:

```
> gendat[od,]

      Eric  Peter  Anna
gene2  1.35   1.55  1.00
gene1  1.00   1.50  1.25
gene4 -1.20 -1.30 -1.00
gene3 -1.10 -1.50 -1.25
```

Eine explizite Reihenfolge der Spalten sieht z.B. so aus

```
> gendat [,c(3,2,1)]
      Anna Peter  Eric
gene1  1.25   1.50  1.00
gene2  1.00   1.55  1.35
gene3 -1.25 -1.50 -1.10
gene4 -1.00 -1.30 -1.20
```

Beides ist auch kombinierbar:

> gendat [od,c(3,2,1)]

```
      Anna Peter  Eric
gene2 1.00  1.55  1.35
gene1 1.25  1.50  1.00
gene4 -1.00 -1.30 -1.20
gene3 -1.25 -1.50 -1.10
```

Möglich sind auch Abfragen mit Vergleichsoperatoren.

Zur Erinnerung:
> meanexprsval <- apply(gendat,1,mean)
> meanexprsval

```
   gene1       gene2       gene3       gene4
1.250000  1.300000 -1.283333 -1.166667
```

Es kann z.B. abgefragt werden, welche Werte der Variablen „meanexprsval“ größer Null sind:

> meanexprsval > 0

```
gene1   gene2 gene3   gene4
TRUE  TRUE FALSE FALSE
```

Dabei erfolgt der Vergleich einer Zahlenfolge mit einem Wert elementweise.
Das erste Element, meanexprsval[1], ist gleich 1.25 und somit > 0, sodass der Vergleich TRUE ergibt.

Es lassen sich so auch Teile der Matrix explizit selektieren:

> gendat [meanexprsval > 0,]

```
      Eric Peter Anna
gene1 1.00  1.50 1.25
gene2 1.35  1.55 1.00
```

1.2 Die Golub-Daten

Die Daten zur molekularen Klassifikation von Krebspatienten stammen von Golub et al.:

Golub et al. (1999): Molecular classification of cancer: class discovery and class prediction by gene expression monitoring, *Science*, Vol. 286:531-537.
http://science.sciencemag.org/content/286/5439/531.long

Das Paper ist mit einer google-Suche erhältlich.

Zunächst sind die Daten aus dem Internet herunterzuladen. Da die Daten im Paket „multtest“ enthalten sind, genügt die Installation des Pakets. Zu finden ist dieses unter:

http://www.bioconductor.org/packages/release/bioc/html/multtest.html

In der Umgebung Rstudio können Packages auch über eine Auswahl im „Packages Menu“ installiert werden, sofern die Liste sie führt. Ansonsten ist das Package „Multtest“ wie folgt zu installieren:

```
> install.packages(c("Multtest"),repo="http://cran.r-project.org",dep=TRUE)
```

Im R-Programm ist dann die Library einzuladen und der Datensatz einzulesen:

```
> library(multtest)
> data(golub)
```

Die zugehörige R-Datei enthält eine Matrix. Diese kann man sich kaum ganz ansehen (zu groß), aber man kann einzelne Informationen durch Abfragen erhalten.

```
> nrow(golub)
[1] 3051
```

und

```
> ncol(golub)
[1] 38
```

geben an, dass die Datei 3051 Zeilen (jede Zeile für jeweils ein Gen) und 38 Spalten hat (jede Spalte für jeweils einen Patienten).

Die 1042ste Zeile erhält man so:

```
> golub[1042,]
```

```
 [1]  2.10892  1.52405  1.96403  2.33597  1.85111  1.99391  2.06597
 [8]  1.81649  2.17622  1.80861  2.44562  1.90496  2.76610  1.32551
[15]  2.59385  1.92776  1.10546  1.27645  1.83051  1.78352  0.45827
[22]  2.18119  2.31428  1.99927  1.36844  2.37351  1.83485  0.88941
[29]  1.45014  0.42904  0.82667  0.63637  1.02250  0.12758 -0.74333
[36]  0.73784  0.49470  1.12058
```

Hier sieht man endlich den Sinn der in "[]" im Output stehenden Zahlen, die die einzelnen Ergebniszeilen einleiten. Die "[8]" zu Anfang der zweiten Zeile gibt an, dass der nachfolgende Wert an der achten Stelle des Ergebnisses steht.

Der erste Wert in der 1042sten Zeile:

> golub[1042,1]
[1] 2.10892

Der zweite Wert in der 1042sten Zeile:

> golub[1042,2]
[1] 1.52405

Die genaue Bezeichnung des Gens der Zeile 1042, zu dem die obigen Werte gehören, kann wie folgt aus der Matrix „golub.gnames" aller Gennamen erhalten werden:

> golub.gnames[1042,2]
```
[1] "CCND3 Cyclin D3"
```

Ein anderer Genname ist z.B. der des Gens der Zeile 829:

> golub.gnames[829,2]
```
[1] "CST3 Cystatin C (amyloid angiopathy and cerebral hemorrhage)"
```

Die zugehörigen Werte (2.10892, 1.52405 etc.) für das Gen aus Zeile 1042 sind Genexpressionswerte. Sie bezeichnen gemessene Mengen an Gentranskripten (mRNA) für die einzelnen Gene (Zeilen), jeweils für bestimmte Patienten (Spalten).

Jedem Patienten (jeder Spalte) ist ein „Status" (bzw. eine Klasse) zugeordnet. Die Golub Daten enthalten Daten zu Leukämie-Patienten. Dabei handelt es sich um zwei Arten (Klassen) von Leukämie, akute myeloische Leukämie (AML) und akute lymphoblastische Leukämie (ALL). Für die Golub Daten sind die Klassen ALL und AML in der Liste „golub.cl" durch 0 und 1 kodiert.

> golub.cl
```
[1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1
[35] 1 1 1 1
```