Fortgeschrittene Datenanalyse mit R

Friedrich Leisch

Institut für Statistik
Ludwig-Maximilians-Universität München

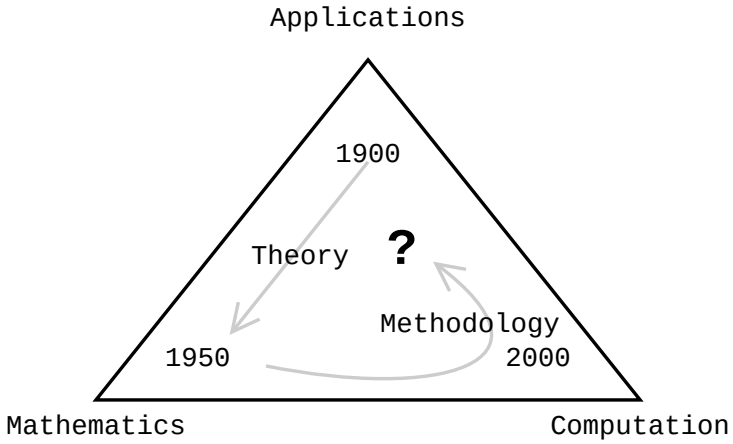
Kursziele

- Verwendung einer objekt-orientierten statistischen Rechenumgebung zur Umsetzung neuer Methoden
- Einsatz neuerer graphischer Verfahren
- Automatisiertes Verteilen und Testen von Software, Daten und Dokumentation
- John Chambers: "Turn ideas into software, quickly and faithfully."

→ Hilfe zur Selbsthilfe

Statistik und Computer

Efron (2001): Statistik im 20. Jahrhundert



Formeln und Code

Formeln sind der bei weitem effizienteste und eleganteste Weg, um mathematische Zusammenhänge auszudrücken:

$$1 + e^{i\pi} = 0$$

Ähnliches gilt für Programmiersprachen und Computer:

```
R> all.equal(exp(1i*pi)+1, 0+0i)
[1] TRUE
```

Das Schöne (und Schwierige?) an Statistik ist,

- daß wir mit Mathematik unsere Theorien beschreiben,
- und Computer verwenden, um Daten zu analysieren.

Was ist Statistiksoftware?

- Eine Programm, das Zahlen als Input nimmt, und daraus Tabellen (und Grafiken) erzeugt?
- Ein(e) (Sammlung von) Programm(en) für Exploration, Inferenz und Modellierung?
- Ein Werkzeug zur Verwaltung, Manipulation und Analyse von Daten?
- Ein Kommunikationsmedium mit CPU (Grafikarte, Drucker, . . .) des Rechners?

Kommunikationsformen

Zeichensprachen: Leicht zu lernen, oft universell verständlich, in Komplexität und Ausdrucksform limitiert. Bsp: österr. Verkehrszeichen, Apple-GUIs, etc.

Schriftsprachen: Schwer zu lernen, Festlegung auf Sprache, in Komplexität und Ausdrucksform nahezu unlimitiert. Bsp: österr. Deutsch, UNIX-Shell, etc.

Die Ilias und Odyssee von Homer haben bis heute überlebt, die Mythologie des antiken Ägypten war wohl ähnlich komplex wie die des alten Griechenland, aber codieren Sie einmal als Übung das erste Kapitel der Ilias ohne Informationsverlust in Icons!

OK, ist als reines Shell-Script auch nicht einfacher . . .

. . . außer das Programm ist gut kommentiert.

Relevanz für Datenanalysen

- Sehen Sie Statistiksoftware (auch) als Hilfsmittel, um dem Computer mitzuteilen, wie Sie Ihre Daten analysieren wollen.
- Für einfache Analysen reichen einfache Kommunikationsformen, aber selbst im komplexesten GUI haben Sie irgendwann das letzte Untermenü erreicht.
- Nur in programmierbaren Umgebungen stoßen Sie nicht irgendwann an die Decke des Möglichen.
- S hat den Oscar der Computersprachen gewonnen, darauf werden S·S oder S··S noch ein wenig warten müssen.

Relevanz für Datenanalysen

- R-Befehle sind Ihre Anweisungen an den Computer.
- Kommandos, die Sie direkt am Prompt tippen, ähneln verbalen Zurufen: schnell kommuniziert und verarbeitet, schnell vergessen.
- Dinge, die man sich länger merken will, schreibt man sich auf → dasselbe gilt für Datenanlysen.
- Tipp 1: Schreiben Sie Ihre Analysen in einem Editor, sparen Sie nicht mit Kommentaren, beginnen Sie am Anfang (= Daten einlesen).

Administration von R

- R ist ein mächtiges, aber auch komplexes Werkzeug.
- Die Heimat von S ist auch die Heimat von Unix und C (New Jersey, nicht Redmond oder Palo Alto).
- Schon Installation und Administration von R haben für Windows-User ungewohnte Aspekte, seien Sie urban und erlauben ein wenig Multikulti am PC. Das Wiener Schnitzel kommt eigentlich auch aus Mailand.
- Nicht alles, was „anders“ ist, muß deshalb schlecht sein.
- Die absolute Mehrheit von R Core benutzt Unix/Linux als Computerplattform.

Administration von R

- Eine wichtiger Aspekt für einen laufenden R Prozess ist das derzeitige Arbeitsverzeichnis. Unter Unix wird R meist aus der Shell gestartet und nimmt das Startverzeichnis als Arbeitsverzeichnis, in einem GUI-Desktop muß (und sollte!) es meist explizit gesetzt werden.
- Umgebungsvariablen:

  LANGUAGE: für Statusmeldungen, z.B. `LANGUAGE=en`
  R_LIBS: Pfade zu Bibliotheken mit Erweiterungspaketen, z.B. `R_LIBS=c:/der/pfad/zu/meinen/paketen`

  Diese können entweder global gesetzt werden oder als Teil des Programmaufrufes. Pfade können auf allen Plattformen in Unix-Notation angegeben werden (wie in URLs).

Administration von R

Auch für den privaten PC gelten

- Tipp 2: Installieren Sie niemals Erweiterungspakete direkt in das versionsabhängige Installationsverzeichnis von R.
- Tipp 3: Verwenden Sie niemals das Installationsverzeichnis von R als Arbeitsverzeichnis.

Spätestens der nächste Update von R liefert einen guten Grund.

Texteditoren

- Damit Tipp 1 nicht mühsam wird, braucht man einen Editor, der R unterstützt (Syntax Markierung, Klammerpaare zählen, Einrückung, Verbindung zum R Prozess, . . .).
- R folgt der Unix-Philosophie, daß sich jedes Werkzeug auf seine Kernaufgaben konzentrieren sollte, diese möglichst gut erledigt, und für den Rest Schnittstellen zu anderen Werkzeugen benutzt.
- R ist kein Texteditor.
- Word ist kein Texteditor.
- Welcher Editor für wen optimal ist, hängt stark von persönlichen Präferenzen und vom Arbeitsstil ab → schwer Empfehlungen abzugeben.

Verwaltung von R Projekten

- Speicherung von Objekten
- Mehrere parallele Projekte
- Ausdruck von Ergebnissen
- Weiterverwendung von Resultaten

Speicherung von Objekten

Workspace: Wenn R gestartet wird lädt es die Datei `.RData`, beim Beenden kann die globale Umgebung (=Workspace) in diese Datei gespeichert werden (Zwischenspeicherung mit `save.image()`). Dies speichert alles außer zusätzlich geladener Pakete und entspricht dem `.Data` Verzeichnis von S-Plus.

Binäre Dateien: Der `save()` Befehl ermöglicht das explizite Speichern von Funktionen und Daten in eine binäre Datei. Diese kann durch `load()` in die globale Umgebung geladen oder durch `attach()` in die Suchliste aufgenommen werden.

Source code: Quellcode für Funktionen und die Reproduktion von Resultaten in einer Textdatei.

Mehrere Projekte

Es gibt zwei extreme Wege um mehrere Projekte zu verwalten:

- Speicherung jedes Projektes in einem anderen Arbeitsverzeichnis und Benutzung der jeweiligen `.RData` Datei für alle Objekte. Expliziter Import/Export wenn Funktionen oder Daten in mehreren Projekten verwendet werden. Die `.RData` Datei ist primär, Befehlsgeschichte und Quellcode von Funktionen sind Dokumentation.
- Speicherung aller Objekte im Quellcode. Für jeden Teil der Analyse gibt es eine Datei mit den zugehörigen R Kommandos (Daten einlesen, Berechnungen, Resultate). Der Quellcode ist primär, gespeicherte Resultate ersparen nur die Arbeit der Neuberechnung.

Diese Möglichkeiten bestehen sowohl bei Datenanalyse wie Programmierung.

Workspace ist primär

Diese Methode ist besonders bei neuen Benutzern von S-Plus weit verbreitet, aber auch bei Benutzern von R unter Windows sowie Umsteigern.

In R ist dieser Zugang riskanter als in S-Plus:

- In R wird der Workspace nur gespeichert, wenn der Benutzer dies explizit anfordert, in S-Plus automatisch.
- Eine fehlerhafte `.RData` Datei kann mit großer Wahrscheinlichkeit gar nicht gelesen werden, S-Plus legt für jedes Objekt eine einzelne Datei an.

Sinnvoll für kurzzeitige Projekte bei denen ein Verlust der Daten nicht kritisch ist, oder für extrem vorsichtige und selbstdisziplinierte Menschen.

Quellcode ist primär

Das Projektmanagement ist viel einfacher wenn alles aus Textdateien mit dem Quellcode der Analyse reproduziert werden kann. Die üblichen Werkzeuge des Betriebssystems zur Verwaltung von Dateien stehen zur Verfügung, inklusive Backups und Zeitstempel.

Emacs Speaks Statistics (ESS) ist besonders hilfreich für diesen Stil der Verwendung von R. Der R Prozess läuft direkt in einem Editor und die Ausgabe kann während der Analyse kommentiert werden. Tastenkombinationen senden R Kommandos aus dem Quellcode direkt an R.

Unter Windows kann auch der Inhalt der Konsole gespeichert oder gedruckt werden, manche Editoren unterstützen direkte Interaktion mit einem laufenden R Prozess. Auch einfaches Kopieren&Einfügen hilft in vielen Fällen.

Quellcode ist primär

Tipp 4: Sehen Sie den Quellcode als primär an, verwenden Sie `fix()` maximal zum Debuggen.

Verwendung von Resultaten

Das Abtippen von Ergebnissen aus R ist umständlich, kostet Zeit und ist eine potentielle Fehlerquelle. Leider unterstützt R das automatisierte Erzeugen von Berichten insbesonders unter Windows eher schlecht.

- Sweave erlaubt die vollständige Einbettung von R in LaTeX Dokumente, Paket `xtable` konvertiert manche R Objekte in LaTeX Code.
- Paket `R2HTML` kann viele R Klassen nach HTML konvertieren.
- Im Prinzip kann man R mittels DCOM in MS Word oder Excel einbetten . . .

Damit beschäftigen wir uns noch im weiteren Verlauf des Kurses.

R Grundlagen

Hilfe:	`help(topic)`	Zuweisungen:	`x = 5`
	`?topic`		`x <- 5`
	`help.start()`		`5 -> x`
Operatoren:	`+, -, *, /, ^`	Vergleiche:	`==, !=, >, >=, <,`
	`&, &&, |, ||`		`<=`
	siehe `help("+")`		siehe `help("==")`
Schleifen:	`for, while, if`	Kommentare:	alles nach `#`

Namen: Groß/Kleinschreibung wichtig, beginnen mit Buchstaben, Achtung bei "Underscore"

R Grundlagen

Elementare Datentypen	
logical:	`TRUE, FALSE, T, F`
integer:	`1, 100, 326, ...`
double:	`1.0, 3.1415297, 2.718282, NaN, Inf, -Inf`
complex:	`1+0i, 1i, 3+5i`
character:	`"Hallo", "Wie geht's?"`
fehlende Werte:	`NA`

Alle elementaren Datentypen sind ausschließlich als Vektoren verfügbar (es gibt keine Skalare), jeder Vektor enthält 0 oder mehr Elemente desselben Typs, eventuell gemischt mit `NA`.

R Grundlagen

Wir unterscheiden zwischen Modus und Speichermodus eines Vektors:

```
> mode(1:10)
[1] "numeric"
> storage.mode(1:10)
[1] "integer"
> mode(pi)
[1] "numeric"
> storage.mode(pi)
[1] "double"
> mode(TRUE)
[1] "logical"
> mode("Hello")
[1] "character"
```

R Grundlagen

Komplexere Datentypen	
array:	Vektor mit Dimensionsattribut, Anzahl der Dimensionen ist de facto nur durch den verfügbaren Hauptspeicher beschränkt.
matrix:	2-dimensionaler Array, Spezialfall um übliche Matrixoperationen (Multiplikation, ...) zu ermöglichen.
factor:	spezieller Vektor für kategorielle Merkmale.
list:	kombiniert Elemente verschiedener Typen (inkl. Listen → rekursive Strukturen).
data frame:	"Mischung" aus Matrix und Liste.

Konstruktion & Umwandlung

- Funktionen zum Erzeugen von Datentypen haben üblicherweise denselben Namen wie der betreffende Datentyp (ein generelles Prinzip in S), Methoden zur Konversion zwischen verschiedenen Typen haben Namen wie `as.xxx()`, Überprüfung des Typs erfolgt mit `is.xxx()`.

```
R> integer(5)
R> double(0)
R> x = matrix(1, nrow=5, ncol=2)
R> is.matrix(x)
R> as.vector(x)
R> x = list(a="Hallo", b=1:10, pi=3.1415927)
```

- Konstruktion einfacher Vektoren

```
R> c(1,2,7)
R> c("Hallo", "Welt")
```

Konstruktion & Umwandlung

- S wandelt den Typ eines Objektes automatisch um, wenn dies notwendig und möglich ist:

```
> TRUE + 2
[1] 3
> c("Hello", 100)
[1] "Hello" "100"
```

- Beispiele für häufige Konversionen sind

logisch	→numerisch
logisch, numerisch	→komplex
logisch, numerisch, komplex	→Text
numerisch, komplex	→logisch

Konstruktion & Umwandlung

- Logische Werte können in jeden anderen Modus umgewandelt werden, daher wurde die Konstante `NA` als eine logische Variable definiert:

```
> mode(NA)
[1] "logical"
```

- In den meisten Fällen wird der Modus von `NA` jedoch durch die restlichen Terme eines Ausdruckes bestimmt:

```
> 1 + NA
[1] NA
> mode(1 + NA)
[1] "numeric"
```

- Unzulässige Konversionen liefern `NA`:

```
> as.numeric(c("1", "10e2", , "text"))
[1]    1 1000   NA
```

`NA`, `NaN` und `NULL`

Alle drei können als Darstellung von "Nichts" angesehen werden, allerdings ist die Ursache für das Fehlen der Information verschieden:

NA: fehlende Werte in der Stichprobe, unzulässige Konversionen, ...

NaN: ähnlich zu `NA`, aber die Ursache des Fehlens ist bekannt: eine mathematische Operation liefert kein Ergebnis, z.B. weil eine Funktion außerhalb ihres Definitionsbereiches verwendet wurde ($\log(-1)$, ...). Üblicherweise begleitet von Warnung.

NULL: Zeiger auf Nichts im informatischen Sinn

```
> c(3, NA)
[1]  3 NA
> c(3, NaN)
[1]   3 NaN
> c(3, NULL)
[1] 3
```

Unendlich

Die beiden speziellen Symbole `Inf` und `-Inf` können wie Zahlen in vielen mathematischen Rechenoperationen verwendet werden:

```
> max(3, NA)
[1] NA
> min(3, NA)
[1] NA
> max(3, Inf)
[1] Inf
> min(3, Inf)
[1] 3
```

Achtung: manche numerische Verfahren können nicht mit Unendlich rechnen und liefern Fehlermeldungen.

Zugriff auf einzelne Elemente

Vektoren: durch Nummer, Name oder logischen Vektor

```
R> x = c(5,3,7)
R> names(x) = c("apple","banana","orange")
R> x["apple"]
R> x[1:2]
R> x[c(T,T,F)]
```

Arrays: wie Vektoren, Dimensionen durch Komma getrennt

```
R> x = matrix(1:10, ncol=2);  colnames(x) = c("Eins", "Zwei")
R> x[1:2,]               # liefert eine Matrix
R> x[,"Zwei"]            # liefert einen Vektor
R> x[,"Zwei",drop=F]     # liefert eine Matrix
R> x[-3,]                # alles außer Zeile 3
```

Zugriff auf einzelne Elemente

Listen: Dollarzeichen oder Nummer/Name

```
R> x = list(a="Hallo", b=1:10, pi=3.1415927)
R> x$a
R> x[["a"]]

R> x[[1]]   ## erstes Element der Liste
R> x[1]     ## Liste mit einem Element

R> x[2:3]   ## Liste mit zwei Elementen
R> x[[2:3]] ## FEHLER!
```

Data frames

- Duale Sichtweise:
  - Matrix in der jede Spalte einen anderen Datentyp haben kann
  - Liste in der jedes Element dieselbe Länge haben muß
- Zugriff auf Elemente wie bei Matrizen oder Listen:

```
R> data(iris)
R> iris[1:10,]         ## data frame mit 10 Zeilen
R> iris[,1]            ## numerischer Vektor
R> iris$Sepal.Length   ## dasselbe
R> iris[,1,drop=FALSE] ## data frame mit einer Spalte
```

Attribute

Sind wie „Pickerl auf einem Marmeladeglas“ und ermöglichen die detailliertere Beschreibung des Inhaltes. So ist z.B. ein Array nur ein Vektor mit einem Dimensionsattribut. Weitere häufige Attribute beinhalten Namen für die Zeilen oder Spalten einer Matrix, Elemente einer Liste, . . .

```
> x = matrix(1:10, ncol = 5)
> x
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
> attributes(x)
$dim
[1] 2 5
```

Attribute

```
> rownames(x) = c("Eins", "Zwei")
> x
     [,1] [,2] [,3] [,4] [,5]
Eins    1    3    5    7    9
Zwei    2    4    6    8   10
> attributes(x)
$dim
[1] 2 5

$dimnames
$dimnames[[1]]
[1] "Eins" "Zwei"

$dimnames[[2]]
NULL
```

Attribute

Die `as.xxx()` Funktionen löschen alle Attribute, inklusive der Dimensionalität:

```
> x = matrix(1:10, nrow = 2)
> x
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
> as.character(x)
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10"
> storage.mode(x) = "character"
> x
     [,1] [,2] [,3] [,4] [,5]
[1,] "1"  "3"  "5"  "7"  "9"
[2,] "2"  "4"  "6"  "8"  "10"
```

Klassen

- Unglücklicherweise hat S gleich zwei Klassensysteme: S3 und S4. Wir werden uns nur mit dem viel einfacheren S3 System beschäftigen.
- Ein spezielles Attribut jedes Objektes beschreibt seine „Klasse“, dies ermöglicht einfaches objekt-orientiertes Programmieren und das Überladen von Funktionen.
- Setzen und Überprüfen der Klasse eines Objektes:
  ```
  R> class(y)
  R> class(y) = "newclass"
  ```
- Generische Funktionen verhalten sich unterschiedlich für Objekte verschiedener Klassen.

Klassen

```
> x = rep(0:1, c(10, 20))
> x
 [1] 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
> class(x)
[1] "integer"
> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.0000  1.0000  0.6667  1.0000  1.0000
> y = as.factor(x)
> class(y)
[1] "factor"
> summary(y)
 0  1
10 20
```

Data import and export

(following the R Data Import/Export manual).

- Spreadsheet-like data
- Data preprocessing and regular expressions

Data import and export

Reading data into a statistical system for analysis and exporting the results to some other system for report writing

- can be a frustrating task
- can take far more time than the statistical analysis itself
- in theory, should be work of the client (deliver data in specified format)
- in practice, it is often easier to import badly formatted data than explain what a „good“ format is

Data import and export

- Statistical systems like R are not particularly well suited to manipulations of large-scale data
- Other tools may be better suited for soma tasks (follows Unix tradition of re-usable tools): perl or awk for data manipulation, databases for storage, ...
- Traditional Unix tools are now much more widely available, including on Windows

Data import

To read spreadsheet-like data you probably have already used function `read.table()` and variations like `read.csv()`, ...

Header: names of variables (columns) and observations (rows)
Separator: field separator vs. record separator
Quoting: protecting separators appearing in strings
Missing values: which string (or number) represents missing values?
Unfilled lines: are lines with missing trailing fields OK?
Classes for the variables: which type of variable is contained in which column?

Re-shaping date

Sometimes spreadsheet data is in a compact format that gives the covariates for each subject followed by all the observations on that subject, but R's modeling functions need observations in a single column.

```
Status   Age    V1     V2     V3    V4
     P 23646 45190  50333  55166 56271
    CC 26174 35535  38227  37911 41184
    CC 27723 25691  25712  26144 26398
    CC 27193 30949  29693  29754 30772
    CC 24370 50542  51966  54341 54273
...
```

Re-shaping date

```
> reshape(zz, idvar="id",timevar="var",
  varying=list(c("V1","V2","V3","V4")),direction="long")
    Status   Age var    V1 id
1.1      P 23646   1 45190  1
2.1     CC 26174   1 35535  2
3.1     CC 27723   1 25691  3
4.1     CC 27193   1 30949  4
5.1     CC 24370   1 50542  5
1.2      P 23646   2 50333  1
2.2     CC 26174   2 38227  2
...
```

Exporting data

- Exporting results from R is usually a less contentious task than importing
- normally a text file will be the most convenient interchange vehicle
- most common task is to write a matrix or data frame to file as a rectangular grid of numbers
- `cat()`, `write()`. `write.table()`

Exporting data

- **Precision:** governed by the current setting of `options(digits)`. Export for report writing or further analysis?
- **Separator:** CSV (comma separated values) files can be read by many applications. Note different decimal points in different countries → using a semicolon or tab are probably the safest options.
- **Header, missing values, quoting:** depend on target application . . .

Flat contingency tables

```
> data(UCBAdmissions)
> ftable(UCBAdmissions)
                Dept   A   B   C   D   E   F
Admit    Gender
Admitted Male        512 353 120 138  53  22
         Female       89  17 202 131  94  24
Rejected Male        313 207 205 279 138 351
         Female       19   8 391 244 299 317
```

Use `read.ftable()` to import.

Data Preprocessing

Data as obtained from practitioners usually not ready to load into statistics software:

- Wrong format (field and record separators, . . .)
- In huge data sets we must not need all observations or variables → reduce memory requirements by extraction of relevant information.
- Variables not directly usable.

Need efficient tools for easy data transformation.

Example: `tcpdump`

Output of `tcpdump`, 800kB/min

```
17:08:05.422121 40 > 4 at-lap#103 37
17:08:06.302121 40 > 4 at-lap#103 37
17:08:06.302121 0:0:c:19:60:c2 > 1:80:c2:0:0:0 802.1d ui/C len=43
                        0000 0000 0000 0000 001d 01ee 3e00 0000
                        6480 00aa 0004 00d3 cd80 0501 0014 0002
                        000f 0000 0000 0000 0000 00
17:08:06.332121 fav.ud.tuwien.ac.at.59883 > fgidec1.tuwien.ac.at.6000:
                P 1261791 002:1261791042(40) ack 660933177 win 8760 (DF)
17:08:06.372121 fgidec1.tuwien.ac.at.6000 > fav.ud.tuwien.ac.at.59883:
                . ack 40 win 16384
17:08:06.462121 galadriel.ci.tuwien.ac.at.614 > fangorn.ci.tuwien.ac.at.711: udp
17:08:06.462121 fangorn.ci.tuwien.ac.at.711 > galadriel.ci.tuwien.ac.at.614: udp
17:08:06.462121 galadriel.ci.tuwien.ac.at.1622 >
                tutimea.tuwien.ac.at.domain: 64 854+ PTR?
                23.170.130.128.in-addr.arpa. (45)
```

Example: `tcpdump`

Suppose we want to extract a table containing information about senders and recipients of Ethernet packages (how many packets were sent):

	fangorn.ci	galadriel.ci	sungold1.at.ibm.net	thorin.ci
aragorn.ci	2	0	0	0
elendil.ci	4	0	0	0
fangorn.ci	0	157	0	53
galadriel.ci	152	0	0	0
sungold1.at.ibm.net	0	0	0	149
thorin.ci	25	0	236	0
tutimea	0	27	0	0

Example: Changing Notation

Evgenia Dimitriadou	→	Dimitriadou, Evgenia
Kurt Hornik		Hornik, Kurt
Friedrich Leisch		Leisch, Friedrich
Andreas Weingessel		Weingessel, Andreas

amounts to

seq1 space seq2	→	seq2 comma space seq1

where seq stands for a sequence of letters.

Regular expressions

- Simple known example: wildcards for file names.
- POSIX regular expressions are available for processing strings in arbitrary C (C++, Java, ...) programs.
- Also directly available in many (Unix) programs: Emacs, vi, perl, awk, sed, R, ...

Regex Syntax

Each character matches itself, unless it is a special character. Special characters must be escaped using a "\".

`.`	matches arbitrary characters
`(...)`	groups a series of patterns to a single element
`^`	beginning of target
`$`	end of target
`[...]`	class of characters (match any of them)
`[^...]`	negates the class (none of them)
`(...|...|...)`	matches one of the alternatives

Regex Syntax

Repeating Patterns:

`*`	match 0 or more times
`+`	match 1 or more times
`?`	match 0 or 1 times
`{n}`	match exactly n times
`{m,n}`	match a minimum of m and at maximum n times

References:

`\1`, `\2`, ... refer to matched sub-expressions grouped with `()`.

Regex: Examples

`Ma.er`	"Maier", "Mayer", "Mader", ...
`Ma[iy]er`	only "Maier" and "Mayer"
`M..er`	"Maier", "Mayer", "Meyer", "Meier", ...
`M.{2}er`	same
`M[ae][iy]e?r`	"Maier", "Mayer", "Meier", "Meyer", "Mair", "Mayr", "Meir", "Meyr"
`M.*er`	"Maier", "Mayer", "Meyer", "Meier", "Meierhofer", "May Walter", ...

Note: regexps always return longest match

Solution in R

```
### ``Important recipients''
imp.r <- names(tab.r)[tab.r>100]

### make a table using only important recipients
paste(imp.r, collapse="|")
ok <- grep(paste(imp.r, collapse="|"), recp2)
table(sender2[ok], recp2[ok])
```

Solution in R

```
### read data into a vector holding one line per element
x <- scan("tcpdump-output1", sep="\n", what="")

### extract sender fields
sender <- sub(".* ([^ ]*) > .*", "\\1", x)

### remove protocol
sender1 <- sub("(.*\\.(at|net|1))\\..*", "\\1", sender)

### extract recipient field
recp <- sub(".* > ([^ ]*).*", "\\1", x)

### remove protocol in a different way
recp1 <- sub("(.*)\\.[^.]*:$", "\\1", recp)

### TU is the default domain
recp2 <- sub(".tuwien.ac.at", "", recp1)
sender2 <- sub(".tuwien.ac.at", "", sender1)

### Who sent and received how many packets?
tab.r <- table(recp2)
tab.s <- table(sender2)
```