Freie Universität Berlin

fu:stat

R-Pakete

Sebastian Warnholz
Statistische Beratungseinheit der FU Berlin

13. Februar 2015

Übersicht

Warum Pakete?

Ein paar allgemeine Gedanken...

- Wiederverwendbarer Code: Hierzu 'genügt' es Funktionen in einer R-Datei abzuspeichern
- Sicherstellung der Funktionalität: 'Dependencies' und 'Namespaces' vs. `library`, `require` und `packageName::functionName`
- Dokumentation von Funktionen: Formale Dokumentation (`F1`) vs. Kommentare im Kode
- Tests und Einhaltung von (Minimal-)Standards: `R CMD CHECK`

Trustworthy Software: The Prime Directive - Chambers (2008)

Übersicht

Methoden I

Wie kann "sinnvollerweise" ein Mittelwert für Daten vom Typ `character` definiert werden?

```
mean(1:10)
```

```
## [1] 5.5
```

```
mean(as.character(1:10))
```

```
## Warning in mean.default(as.character(1:10)): argument is
## logical: returning NA
```

```
## [1] NA
```

Methoden II


```
mean <- function(x) {
  if(is.character(x)) {
    base::mean(as.numeric(x))
  } else {
    base::mean(x)
  }
}
mean(as.character(1:10))
```

```
## [1] 5.5
```

```
rm("mean")
```

- Nicht optimal, da nur schwer erweiterbar
- Für jede Klasse muss eine eigene if-Bedingung hinzugefügt werden
- Besser: Eine einzelne ‘mean’ Funktion für jede Klasse (S3-Methode)

Methoden III

- Generische Funktion: `mean`. Hat nur die Aufgabe, die richtige Methode für die Klasse des ersten Arguments zu finden
- Eine Methode wird über den Namen definiert und hat folgende Konvention: `<generic>.<class>`

```
mean.character <- function(x, ...) {
  mean(as.numeric(x), ...)
}
mean(as.character(1:10))
```

```
## [1] 5.5
```

Generische Funktionen I

- Eine generische Funktion findet Methoden mit `UseMethod`:

```
mean <- function(x, ...) UseMethod("mean")
```

- Sollte keine passende Methode gefunden werden, wird eine "default" Methode aufgerufen:

```
mean.default <- function(x, trim = 0, na.rm = FALSE, ...) {
  ...
  }
```

Klassen I

- Das S3-Klassensystem besteht aus
  - generischen Funktionen
  - Methoden
  - Klassen
- S3-Klassen dienen einzig der Methodenauswahl
- Eine Klasse ist einfach ein Attribut eines Objektes

```
class(1:10)
```

```
## [1] "integer"
```

```
class("a")
```

```
## [1] "character"
```

Klassen II

- Klassen werden von Konstruktorfunktionen erzeugt (`numeric`, `list`, `lm`)

```
mylm <- function(y, X) {
  beta <- solve(crossprod(X, X)) %*% crossprod(X, y)
  class(beta) <- "mylm"
  beta
}
print.mylm <- function(object, ...) cat("Konsolen-Output\n")
mylm(rnorm(10), rnorm(10))
```

```
## Konsolen-Output
```

Diskussion

- Das S3 Klassensystem ist grundlegend für diverse Pakete in R
- Generische Funktionen haben eine einfache Aufgabe: Die Anzahl der Funktionsnamen, an die sich Nutzer erinnern müssen zu reduzieren (`plot`, `summary`, `print`, `predict`, `residuals`, etc.)
- Einfache Handhabung. Im wesentlichen eine Namenskonvention und die Funktion `class` ansonsten muss man nicht viel wissen
- Nachteil/Vorteil: es gibt keine formale Klassendefinition (wurde mit S4 implementiert)
- Nicht behandelt: Mehr Konventionen, Vererbung

Übersicht

Minimale Ordnerstruktur

- Die schnellste Möglichkeit ein Paket zu erstellen ist die Funktion: `package.skeleton()` oder `devtools::create()`
- Minimale Struktur (MUSS vorhanden sein!)
  - Ordner mit `*.R`-Dateien, muss mit 'R' benannt werden
  - `DESCRIPTION`-Datei
    - `Package`, `Version`, `License`, `Description`, `Title`, `Author`, `Maintainer`
  - `NAMESPACE`-Datei
    - `export`, `exportPattern`, `import`, etc.

Der Ordner 'R'

- Im Verzeichnis des Paketes muss es einen Ordner mit dem Namen 'R' geben
- Alle Dateien, die in diesem Ordner liegen, müssen `*.R`-Dateien sein
- Alle Dateien werden in alphanumerischer Reihenfolge in das Paket geladen. So wie mit der Funktion `source` Funktionen in `R` geladen werden können
- Alle `R`-Objekte, die mit `<-()` zugewiesen werden und in diesen Dateien definiert werden, stehen danach in der Umgebung ('Environment') des Paketes zur Verfügung

```
mylm <- function(y, X) {
  solve(crossprod(X, X)) %*% crossprod(X, y)
}
```

DESCRIPTION


```
Package: mylm
Version: 0.1.1
License: GPL-2
Description: Test-Package
Title: mylm Test-Package
Author: Sebastian Warnholz
Maintainer: <Sebastian.Warnholz@fu-berlin.de>
```

NAMESPACE & Dependencies

- Definiert den Namensraum eines Paketes
- `Depends`/`Imports`: R-Pakete die innerhalb des Paketes verwendet werden
  - Ohne diese kann das Paket nicht installiert werden
  - Pakete unter `Depends` werden beim laden des Paketes dem Suchpfad hinzugefügt, was nach Möglichkeit vermieden werden sollte
- `import`: Einzelne Objekte oder Pakete die innerhalb eines Paketes zur Verfügung stehen. Unabhängig vom Suchpfad. Sehr wichtig um die Funktionalität des Paketes zu garantieren
- `export`: Welche `R`-Objekte stehen dem Anwender zur Verfügung

NAMESPACE

Mögliche Aufrufe:

```
export(mylm)
exportPattern("^[^\\.]")
import()
importFrom()
S3method()
useDynLib()
```

Zusatz für S4 Klassen und Methoden:

```
exportClasses()
exportMethods()
```

Die `NAMESPACE`-Datei kann mit dem Paket `roxygen2` automatisch erstellt werden! Mehr dazu später.

S3 Klassen und Methoden

- S3 Klassen und Methoden können grundsätzlich genauso definiert werden, wie außerhalb des Pakets
- Beim Suchen von Methoden kann es allerdings zu Fehlern kommen, wenn die generische Funktion außerhalb des Paketes definiert ist. Daher die Option `S3method` für die `NAMESPACE`-Datei

```
mylm <- function(y, X) {
  beta <- solve(crossprod(X, X)) %*% crossprod(X, y)
  class(beta) <- "mylm"
  beta

}
```

S3 Klassen und Methoden


```
print.mylm <- function(x, ...) {
  cat("Geschätzte Koeffizienten:\n")
  cat(x, "\n")
}
mylm(rnorm(10), rnorm(10))
```

```
## Geschätzte Koeffizienten:
## -0.2865683
```

Erweiterung für `NAMESPACE`:

```
export(mylm)
S3method(print, mylm)
```

S3 Klassen und Methoden

Neue generische Funktion bzw. Methode:

```
myGeneric <- function(x, ...) UseMethod("myGeneric")
myGeneric.mylm <- function(x, ...) cat("test")
```

Erweiterung für `NAMESPACE`:

```
export(myGeneric)
S3method(myGeneric, mylm)
```

Dokumentation

- In Paketen sollten Funktionen, Methoden, S4-Klassen und Datensätze dokumentiert werden
- Dokumentations-Dateien sind `*.Rd`-Dateien und befinden sich in dem Ordner 'man'
- Aus diesen Dateien wird eine HTML- und eine PDF-Dokumentation erstellt
- Das Paket `roxygen2` vereinfacht die Paketerstellung dramatisch:
  - Die `*.Rd`-Dateien müssen nicht manuell erstellt werden
  - Zusätzlich kann auch die `NAMESPACE` automatisch von `roxygen2` gepfegt werden

Dokumentation für mylm


```
#' Mein lineares Modell
#'
#' @description Berechnet die Koeffizienten eines linearen M
#' @param y abhängige Variable
#' @param X Design-Matrix
#'
#' @return Object der Klasse \code{mylm}
#' @details Mehr Details
#' @export
#' @examples mylm(rnorm(10), rnorm(10))
mylm <- function(y, X) {
  beta <- solve(crossprod(X, X)) %*% crossprod(X, y)
  class(beta) <- "mylm"
  beta
}
```

Dokumentation für S3-Methoden

- Methoden für generische Funktionen aus anderen Paketen werden dokumentiert, wenn sie wesentlich vom verhalten der generischen Funktion abweichen
- Ansonsten reicht `@export`

```
#' @export
print.mylm <- function(x, ...) {
  cat("Geschätzte Koeffizienten:\n")
  cat(x, "\n")
}
```

Dokumentation für S3-Generics

- Neue generische Funktionen werden genauso dokumentiert wie Funktionen
- Typischerweise gibt es keinen Grund dafür, dass der Anwender weiß oder sich kümmert oder verstehen muss, was eine generische Funktion ist

```
#' Neue generische Funktion
#'
#' @description Test
#' @param x ein object
#' @param ... zusätzliche Argumente, die an Methoden weiterg
#'
#' @export
myGeneric <- function(x, ...) UseMethod("myGeneric")

#' @export
myGeneric.mylm <- function(x, ...) "test"
```

Übersicht

Diskussion

- Best Practice: `R CMD CHECK`, Testen, Vignette, Versions-Kontrolle, Kontinuierliche Integration, Umgang mit CRAN
- Hilfreiche Pakete: `roxygen2`, `devtools`, `testthat`
- Material: r-pkgs, Tutorial