Mapping levels of a factor

The gdata package

by Gregor Gorjanc

Introduction

Factors use levels attribute to store information on mapping between internal integer codes and character values i.e. levels. First level is mapped to internal integer code 1 and so on. Although some users do not like factors, their use is more efficient in terms of storage than for character vectors. Additionally, there are many functions in base R that provide additional value for factors. Sometimes users need to work with internal integer codes and mapping them back to factor, especially when interfacing external programs. Mapping information is also of interest if there are many factors that should have the same set of levels. This note describes `mapLevels` function, which is an utility function for mapping the levels of a factor in **gdata** [1] package (Warnes, 2006).

Description with examples

Function `mapLevels()` is an (S3) generic function and works on `factor` and `character` atomic classes. It also works on `list` and `data.frame` objects with previously mentioned atomic classes. Function `mapLevels` produces a so called "map" with names and values. Names are levels, while values can be internal integer codes or (possibly other) levels. This will be clarified later on. Class of this "map" is `levelsMap`, if `x` in `mapLevels()` was atomic or `listLevelsMap` otherwise - for `list` and `data.frame` classes. The following example shows the creation and printout of such a "map".

```
> library(gdata)
> (fac <- factor(c("B", "A", "Z", "D")))

[1] B A Z D
Levels: A B D Z

> (map <- mapLevels(x=fac))

A B D Z
1 2 3 4
```

If we have to work with internal integer codes, we can transform factor to integer and still get "back the original factor" with "map" used as argument in `mapLevels<-` function as shown bellow. `mapLevels<-` is also an (S3) generic function and works on same classes as `mapLevels` plus `integer` atomic class.

```
> (int <- as.integer(fac))

[1] 2 1 4 3

> mapLevels(x=int) <- map
> int

[1] B A Z D
Levels: A B D Z

> identical(fac, int)

[1] TRUE
```

Internally "map" (`levelsMap` class) is a `list` (see bellow), but its print method unlists it for ease of inspection. "Map" from example has all components of length 1. This is not mandatory as `mapLevels<-` function is only a wrapper around workhorse function `levels<-` and the later can accept `list` with components of various lengths.

```
> str(map)

List of 4
 $ A: int 1
 $ B: int 2
 $ D: int 3
 $ Z: int 4
 - attr(*, "class")= chr "levelsMap"
```

Although not of primary importance, this "map" can also be used to remap factor levels as shown bellow. Components "later" in the map take over the "previous" ones. Since this is not optimal I would rather recommend other approaches for "remapping" the levels of a `factor`, say `recode` in **car** package (Fox, 2006).

```
> map[[2]] <- as.integer(c(1, 2))
> map

A B B D Z
1 1 2 3 4

> int <- as.integer(fac)
> mapLevels(x=int) <- map
> int

[1] B B Z D
Levels: A B D Z
```

Up to now examples showed "map" with internal integer codes for values and levels for names. I call this integer "map". On the other hand character "map" uses levels for values and (possibly other) levels for names. This feature is a bit odd at first sight, but can be used to easily unify levels and internal integer codes across several factors. Imagine you have a factor that is for some reason split into two factors `f1` and `f2` and that each factor does not have all levels. This is not uncommon situation.

[1] from version 2.3.1

```
> (f1 <- factor(c("A", "D", "C")))

[1] A D C
Levels: A C D

> (f2 <- factor(c("B", "D", "C")))

[1] B D C
Levels: B C D
```

If we work with this factors, we need to be careful as they do not have the same set of levels. This can be solved with appropriately specifying `levels` argument in creation of factors i.e. `levels=c("A", "B", "C", "D")` or with proper use of `levels<-` function. I say proper as it is very tempting to use:

```
> fTest <- f1
> levels(fTest) <- c("A", "B", "C", "D")
> fTest

[1] A C B
Levels: A B C D
```

Above example extends set of levels, but also changes level of 2nd and 3rd element in `fTest`! Proper use of `levels<-` (as shown in `levels` help page) would be:

```
> fTest <- f1
> levels(fTest) <- list(A="A", B="B",
+                       C="C", D="D")
> fTest

[1] A D C
Levels: A B C D
```

Function `mapLevels` with character "map" can help us in such scenarios to unify levels and internal integer codes across several factors. Again the workhorse under this process is `levels<-` function from base R! Function `mapLevels<-` just controls the assignment of (integer or character) "map" to `x`. Levels in `x` that match "map" values (internal integer codes or levels) are changed to "map" names (possibly other levels) as shown in `levels` help page. Levels that do not match are converted to `NA`. Integer "map" can be applied to `integer` or `factor`, while character "map" can be applied to `character` or `factor`. Result of `mapLevels<-` is always a `factor` with possibly "remapped" levels.

To get one joint character "map" for several factors, we need to put factors in a `list` or `data.frame` and use arguments `codes=FALSE` and `combine=TRUE`. Such map can then be used to unify levels and internal integer codes.

```
> (bigMap <- mapLevels(x=list(f1, f2),
+                      codes=FALSE,
+                      combine=TRUE))

  A   B   C   D
"A" "B" "C" "D"

> mapLevels(f1) <- bigMap
> mapLevels(f2) <- bigMap
> f1


[1] A D C
Levels: A B C D


> f2


[1] B D C
Levels: A B C D


> cbind(as.character(f1), as.integer(f1),
+       as.character(f2), as.integer(f2))


     [,1] [,2] [,3] [,4]
[1,] "A"  "1"  "B"  "2"
[2,] "D"  "4"  "D"  "4"
[3,] "C"  "3"  "C"  "3"
```

If we do not specify `combine=TRUE` (which is the default behaviour) and `x` is a `list` or `data.frame`, `mapLevels` returns "map" of class `listLevelsMap`. This is internally a `list` of "maps" (`levelsMap` objects). Both `listLevelsMap` and `levelsMap` objects can be passed to `mapLevels<-` for `list/data.frame`. Recycling occurs when length of `listLevelsMap` is not the same as number of components/columns of a `list/data.frame`.

Additional convenience methods are also implemented to ease the work with "maps":

- `is.levelsMap`, `is.listLevelsMap`, `as.levelsMap` and `as.listLevelsMap` for testing and coercion of user defined "maps",
- `"["` for subsetting,
- `c` for combining `levelsMap` or `listLevelsMap` objects; argument `recursive=TRUE` can be used to coerce `listLevelsMap` to `levelsMap`, for example `c(llm1, llm2, recursive=TRUE)` and
- `unique` and `sort` for `levelsMap`.

Summary

Functions `mapLevels` and `mapLevels<-` can help users to map internal integer codes to factor levels and unify levels as well as internal integer codes among several factors. I welcome any comments or suggestions.

Bibliography

J. Fox. *car: Companion to Applied Regression*, 2006. URL `http://socserv.socsci.mcmaster.ca/jfox/`. R package version 1.1-1.

G. R. Warnes. *gdata: Various R programming tools for data manipulation*, 2006. URL `http://cran.r-project.org/src/contrib/Descriptions/gdata.html`. R package version 2.3.1. Includes R source code and/or documentation contributed by Ben Bolker, Gregor Gorjanc and Thomas Lumley.

Gregor Gorjanc
University of Ljubljana, Slovenia
`gregor.gorjanc@bfro.uni-lj.si`