Using splitters

Ludvig Renbo Olsen
Cognitive Science, Aarhus University
mail@ludvigolsen.dk | http://ludvigolsen.dk

10/29/2016

Contents

1 Including splitters.R in your session

```
source('splitters.R')
```

2 General information

splitters is a set of functions for easily splitting dataframes or vectors into multiple windows / subsets.

There are two groups of functions: greedy splitters and n (number of windows) splitters.

Each group contains 4 functions:

grouping_factor functions return a factor with window numbers.
This can be used to subset, aggregate, group_by, etc.

split functions are wrapper functions for the *dsplit* and *vsplit* functions (see below). It checks if the given data is a vector or a dataframe and calls the appropriate function. If you know the *split* functions you usually don't need to know the *dsplit* and *vsplit* functions, though they are included for special cases.

dsplit functions are used with dataframes. The dataframe is splitted and the new dataframes are returned in a list.

vsplit functions are used with vectors. The vector is splitted and the new vectors are returned in a list.

2.1 Greedy splitters

Greedy splitters uses window **size** for splitting the data.
Greedy means that each window grabs as many elements as possible (up to size), meaning that there might be less elements available to the last window

Example

> We have a vector with 57 values. We want to have window sizes of 10.
>
> The greedy splitters will return windows with this many values in them:
> 10, 10, 10, 10, 10, 7

By setting **force_equal** to TRUE, we discard the last window if it contains fewer values than the other windows.

Example

> We have a vector with 57 values. We want to have window sizes of 10.
>
> The greedy splitters with force_equal set to TRUE will return windows with this many values in them:
> 10, 10, 10, 10, 10
>
> meaning that 7 values have been discarded.

2.2 n splitters

n splitters use number of windows (n_windows) for splitting the data.

With *default settings*, they try to make the windows as equal as possible, but notice that the last window might contain fewer or more elements, if the length of the data is not divisible with the number of windows.

Example

> We have a vector with 57 values. We want to get back 5 windows.
>
> n splitters with default settings would return windows with this many values in them:
>
> 11, 11, 11, 11, 13

By setting **force_equal** to TRUE, n splitters will create the largest possible, equally sized windows by discarding excess data elements.

Example

> n splitters with **force_equal** set to TRUE would return windows with this many values in them:
>
> 11, 11, 11, 11, 11
>
> meaning that 2 values have been discarded.

Notice that n splitters will always return the given number of windows. They will never return a window with zero elements. For some situations that means that the last window will contain a lot of elements. Asked to split a vector with 57 elements into 20 windows, the first 19 windows will contain 2 elements, while the last window will itself contain 19 elements. Had we instead asked it to split the vector into 19 windows, we would have had 3 elements in all windows.

2.3 Arguments

2.3.1 data or v

The data to process.

data: dataframe or vector (function dependent)
Used in *split* functions where it can be either a dataframe or a vector.
Used in *dsplit* functions where it can **only** be a dataframe.

v: vector
Used in *grouping_factor* and *vsplit* functions

2.3.2 size or n_windows

size: whole number or percentage (Used by greedy splitters)

> Whole number: 1 or above. A size of 10 means that each window will contain 10 elements (possibly not the last window).
>
> Percentage: Numeric between 0-1. E.g. 0.1 is 10 percent. If your vector has a length of 100 and size is set to 0.2, each window will contain 20 elements.

n_windows: whole number or percentage (Used by n splitters)

> Whole number: 1 or above. A size of 10 means that the splitter will create exactly 10 windows / subsets / data splits.
>
> Percentage: Numeric between 0-1. E.g. 0.1 is 10 percent. If your vector has a length of 100 and n_windows is set to 0.2, the splitter will create exactly 20 windows / subsets / data splits.

2.3.3 force_equal

If you need windows with the exact same size, set force_equal to TRUE.
Implementation is different in the two groups of splitters. Read more in their sections above.
Be aware that this setting discards excess datapoints.

2.3.4 allow_zero

If you input 0 as size or n_windows (depending on the function), you get an error.
If you don't want this behavior, you can set allow_zero to TRUE, and (depending on the function) you will get the following output:

grouping_factor functions return the factor with NAs instead of numbers. It will be the same length as expected.

dsplit functions return the given dataset in the same list format as if it had been split.

vsplit functions return the given vector in the same list format as if it had been split.

3 Functions

3.1 gsplit_grouping_factor

Greedy split grouping factor

1. We create a dataframe

```
df = data.frame("x"=c(1:12),
                "species" = rep(c('cat','pig', 'human'), 4),
                "age" = sample(c(1:100), 12))
```

2. Using gsplit_grouping_factor()
   Notice that I only pass it 1 column from the dataframe

```
df$group = gsplit_grouping_factor(df[,1], 5)

df
```

```
##     x species age group
## 1   1     cat  40     1
## 2   2     pig  48     1
## 3   3   human  17     1
## 4   4     cat  61     1
## 5   5     pig  55     1
## 6   6   human  42     2
## 7   7     cat  27     2
## 8   8     pig  19     2
## 9   9   human  25     2
## 10 10     cat  78     2
## 11 11     pig  97     3
## 12 12   human  22     3
```

3. We could get the mean age of each group

```
aggregate(df[, 3], list(df$group), mean)
```

```
##   Group.1    x
## 1       1 44.2
## 2       2 38.2
## 3       3 59.5
```

3.1.1 force_equal

Getting an equal number of elements per window with gsplit_grouping_factor.

Notice that we discard the last window that would have contained less elements than the other groups. Since the grouping factor is shorter than the dataframe, we can't combine them as they are. A way to do so would be to shorten the dataframe to be the same length as the grouping_factor.

1. We create a dataframe

```
df = data.frame("x"=c(1:12),
                "species" = rep(c('cat','pig', 'human'), 4),
                "age" = sample(c(1:100), 12))
```

2. Using gsplit_grouping_factor() with force_equal

```
group = gsplit_grouping_factor(df[,1], 5, force_equal = TRUE)

group
```

```
##  [1] 1 1 1 1 1 2 2 2 2 2
## Levels: 1 2
```

3. Combining dataframe and grouping factor

First we make the dataframe the same size as the grouping factor. Then we add the grouping factor to the dataframe.

```
df = head(df, length(group))

df$group = group

df
```

```
##     x species age group
## 1   1     cat  47     1
## 2   2     pig  76     1
## 3   3   human  18     1
## 4   4     cat  20     1
## 5   5     pig   6     1
## 6   6   human  23     2
## 7   7     cat  24     2
## 8   8     pig  63     2
## 9   9   human  17     2
## 10 10     cat  45     2
```

3.2 gdsplit

Greedy dataframe split

1. We create a dataframe

```
df = data.frame("x"=c(1:12),
                "species" = rep(c('cat','pig', 'human'), 4),
                "age" = sample(c(1:100), 12))
```

2. Using gdsplit()

```
df_list = gdsplit(df, 5)

df_list
```

```
## $`1`
##   x species age
## 1 1     cat  23
## 2 2     pig  95
## 3 3   human  80
## 4 4     cat  56
## 5 5     pig  33
##
## $`2`
##     x species age
## 6   6   human  94
## 7   7     cat  46
## 8   8     pig  75
## 9   9   human  67
## 10 10     cat  90
##
## $`3`
##     x species age
## 11 11     pig  87
## 12 12   human  88
```

3. We can get a specific dataframe

```
df_list[[2]]
```

```
##     x species age
## 6   6   human  94
## 7   7     cat  46
## 8   8     pig  75
## 9   9   human  67
## 10 10     cat  90
```

3. We could get the mean of age for that particular dataframe

```
mean(df_list[[2]]$age)
```

```
## [1] 74.4
```

3.2.1 force_equal

Getting an equal number of elements per window with gdsplit.

Notice that we discard the last dataframe that would have contained fewer rows than the others.

1. We create a dataframe

```
df = data.frame("x"=c(1:12),
                "species" = rep(c('cat','pig', 'human'), 4),
                "age" = sample(c(1:100), 12))
```

2. Using gdsplit() with force_equal

```
df_list = gdsplit(df, 5, force_equal = TRUE)

df_list
```

```
## $`1`
##   x species age
## 1 1     cat  70
## 2 2     pig  96
## 3 3   human  60
## 4 4     cat  57
## 5 5     pig  97
##
## $`2`
##     x species age
## 6   6   human  17
## 7   7     cat  71
## 8   8     pig  73
## 9   9   human  12
## 10 10     cat  94
```

3.3 gvsplit

Greedy vector split

1. We create a vector

```
vec = c(1:12)
```

2. Using gvsplit()

```
vec_list = gvsplit(vec, 5)

vec_list
```

```
## $`1`
## [1] 1 2 3 4 5
##
## $`2`
## [1]  6  7  8  9 10
##
## $`3`
## [1] 11 12
```

3. We can get a specific vector

```
vec_list[[2]]
```

```
## [1]  6  7  8  9 10
```

4. We could get the mean of that particular vector

```
mean(vec_list[[2]])
```

```
## [1] 8
```

3.3.1 force_equal

Getting an equal number of elements per window with gvsplit.

Notice that we discard the last vector that would have contained fewer elements than the others.

1. We create a vector

```
vec = c(1:12)
```

2. Using nvsplit() with force_equal

```
vec_list = gvsplit(vec, 5, force_equal = TRUE)

vec_list
```

```
## $`1`
## [1] 1 2 3 4 5
##
## $`2`
## [1]  6  7  8  9 10
```

3.4 nsplit_grouping_factor

Number of windows split grouping factor

1. We create a dataframe

```
df = data.frame("x"=c(1:12),
                "species" = rep(c('cat','pig', 'human'), 4),
                "age" = sample(c(1:100), 12))
```

2. Using nsplit_grouping_factor()
   Notice that I only pass it 1 column from the dataframe

```
df$group = nsplit_grouping_factor(df[,1], 5)

df
```

```
##     x species age group
## 1   1     cat  91     1
## 2   2     pig  15     1
## 3   3   human   6     2
## 4   4     cat  39     2
## 5   5     pig  65     3
## 6   6   human  42     3
## 7   7     cat  92     4
## 8   8     pig  57     4
## 9   9   human  37     5
## 10 10     cat  53     5
## 11 11     pig  46     5
## 12 12   human  31     5
```

3. We could get the mean age of each group

```
aggregate(df[, 3], list(df$group), mean)
```

```
##   Group.1     x
## 1       1 53.00
## 2       2 22.50
## 3       3 53.50
## 4       4 74.50
## 5       5 41.75
```

3.4.1 force_equal

Getting an equal number of elements per window with nsplit_grouping_factor.

Notice that the last group in the factor now contains the same number of elements as other groups. Since the grouping factor is shorter than the dataframe, we can't combine them as they are. We could though shorten the dataframe to be the same length as the grouping_factor.

1. We create a dataframe

```
df = data.frame("x"=c(1:12),
                "species" = rep(c('cat','pig', 'human'), 4),
                "age" = sample(c(1:100), 12))
```

2. Using nsplit_grouping_factor() with force_equal

```
group = nsplit_grouping_factor(df[,1], 5, force_equal = TRUE)

group
```

```
##  [1] 1 1 2 2 3 3 4 4 5 5
## Levels: 1 2 3 4 5
```

3. Combining dataframe and grouping factor

First we make the dataframe the same size as the grouping factor. Then we add the grouping factor to the dataframe.

```
df = head(df, length(group))

df$group = group

df
```

```
##     x species age group
## 1   1     cat  40     1
## 2   2     pig  11     1
## 3   3   human  75     2
## 4   4     cat  41     2
## 5   5     pig  30     3
## 6   6   human  73     3
## 7   7     cat  88     4
## 8   8     pig  72     4
## 9   9   human  47     5
## 10 10     cat  78     5
```

3.5 ndsplit

Number of windows dataframe split

1. We create a dataframe

```
df = data.frame("x"=c(1:12),
                "species" = rep(c('cat','pig', 'human'), 4),
                "age" = sample(c(1:100), 12))
```

2. Using ndsplit()

```
df_list = ndsplit(df, 5)

df_list
```

```
## $`1`
##   x species age
## 1 1     cat  28
## 2 2     pig  93
##
## $`2`
##   x species age
## 3 3   human  56
## 4 4     cat  29
##
## $`3`
##   x species age
## 5 5     pig  35
## 6 6   human  42
```

```
##
## $`4`
##   x species age
## 7 7     cat  99
## 8 8     pig  15
##
## $`5`
##     x species age
## 9   9   human  41
## 10 10     cat  89
## 11 11     pig  38
## 12 12   human  32
```

3. We can get a specific dataframe

```
df_list[[2]]
```

```
##   x species age
## 3 3   human  56
## 4 4     cat  29
```

3. We could get the mean of age for that particular dataframe

```
mean(df_list[[2]]$age)
```

```
## [1] 42.5
```

3.5.1 force_equal

Getting an equal number of elements per window with ndsplit.

Notice that the last dataframe now contains the same number of rows as the others.

1. We create a dataframe

```
df = data.frame("x"=c(1:12),
                "species" = rep(c('cat','pig', 'human'), 4),
                "age" = sample(c(1:100), 12))
```

2. Using ndsplit() with force_equal

```
df_list = ndsplit(df, 5, force_equal = TRUE)

df_list
```

```
## $`1`
##   x species age
## 1 1     cat  33
## 2 2     pig  82
##
```

```
## $`2`
##   x species age
## 3 3   human  92
## 4 4     cat  47
##
## $`3`
##   x species age
## 5 5     pig  67
## 6 6   human  14
##
## $`4`
##   x species age
## 7 7     cat  75
## 8 8     pig  46
##
## $`5`
##     x species age
## 9   9   human   7
## 10 10     cat  90
```

3.6 nvsplit

Number of windows vector split

1. We create a vector

```
vec = c(1:12)
```

2. Using nvsplit()

```
vec_list = nvsplit(vec, 5)

vec_list
```

```
## $`1`
## [1] 1 2
##
## $`2`
## [1] 3 4
##
## $`3`
## [1] 5 6
##
## $`4`
## [1] 7 8
##
## $`5`
## [1]  9 10 11 12
```

3. We can get a specific vector

```r
vec_list[[2]]
```

```
## [1] 3 4
```

4. We could get the mean of that particular vector

```r
mean(vec_list[[2]])
```

```
## [1] 3.5
```

3.6.1 force_equal

Getting an equal number of elements per window with nvsplit.

Notice that the last vector now contains the same number of elements as the others.

1. We create a vector

```r
vec = c(1:12)
```

2. Using nvsplit() with force_equal

```r
vec_list = nvsplit(vec, 5, force_equal = TRUE)

vec_list
```

```
## $`1`
## [1] 1 2
##
## $`2`
## [1] 3 4
##
## $`3`
## [1] 5 6
##
## $`4`
## [1] 7 8
##
## $`5`
## [1]  9 10
```