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 split functions: greedy split and n (number of windows) split.

Each group contains 2 functions:

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

split functions split the given data (dataframe or vector) into the specified windows and return them in a list.

2.1 Greedy split

Greedy split 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 splitter 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 splitter 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 split

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

With *default settings*, it tries 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 splitter with default settings would return windows with this many values in them:
>
> 11, 11, 11, 11, 13

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

Example

> n splitter 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 the n splitter will always return the given number of windows. It 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
Used in *split* functions

v: vector
Used in *grouping_factor*

2.3.2 size or n_windows

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

> 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 splitter)

> 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 kinds 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.

split functions will return the given data (dataframe or 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  57     1
## 2   2     pig  96     1
## 3   3   human  46     1
## 4   4     cat  59     1
## 5   5     pig  70     1
## 6   6   human  75     2
## 7   7     cat  35     2
## 8   8     pig  64     2
## 9   9   human  36     2
## 10 10     cat  43     2
## 11 11     pig  93     3
## 12 12   human  45     3
```

3. We could get the mean age of each group

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

```
##   Group.1    x
## 1       1 65.6
## 2       2 50.6
## 3       3 69.0
```

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  84     1
## 2   2     pig  95     1
## 3   3   human  49     1
## 4   4     cat   5     1
## 5   5     pig  15     1
## 6   6   human  85     2
## 7   7     cat  61     2
## 8   8     pig  30     2
## 9   9   human  87     2
## 10 10     cat  98     2
```

3.2 gsplit

Greedy splitter for both vectors and dataframes

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() for dataframes

```
df_list = gsplit(df, 5)

df_list
```

```
## $`1`
##   x species age
## 1 1     cat  88
## 2 2     pig  44
## 3 3   human  30
## 4 4     cat  73
```

```
## 5 5     pig  64
##
## $`2`
##     x species age
## 6   6   human  26
## 7   7     cat  28
## 8   8     pig  33
## 9   9   human  74
## 10 10     cat  13
##
## $`3`
##     x species age
## 11 11     pig  77
## 12 12   human  75
```

3. We can get a specific dataframe

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

```
##     x species age
## 6   6   human  26
## 7   7     cat  28
## 8   8     pig  33
## 9   9   human  74
## 10 10     cat  13
```

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

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

```
## [1] 34.8
```

5. Using gsplit() for vectors

```
# Note that I only pass it one column!
vec_list = gsplit(df[,1], 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
```

3.2.1 force_equal

Getting an equal number of elements per window with gsplit.

Notice that we discard the last dataframe/vector that would have contained fewer rows/elements 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 gsplit() with force_equal on a dataframe

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

df_list
```

```
## $`1`
##   x species age
## 1 1     cat  45
## 2 2     pig  71
## 3 3   human   3
## 4 4     cat  17
## 5 5     pig  73
##
## $`2`
##     x species age
## 6   6   human  93
## 7   7     cat  53
## 8   8     pig  18
## 9   9   human  28
## 10 10     cat  20
```

3. Using gsplit() with force_equal on a vector

```
# Note that I only pass it one column!
vec_list = gsplit(df[,1], 5, force_equal = TRUE)

vec_list
```

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

3.3 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  44     1
## 2   2     pig  15     1
## 3   3   human  24     2
## 4   4     cat  60     2
## 5   5     pig  29     3
## 6   6   human  85     3
## 7   7     cat  43     4
## 8   8     pig  51     4
## 9   9   human  45     5
## 10 10     cat  97     5
## 11 11     pig  71     5
## 12 12   human  50     5
```

3. We could get the mean age of each group

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

```
##   Group.1     x
## 1       1 29.50
## 2       2 42.00
## 3       3 57.00
## 4       4 47.00
## 5       5 65.75
```

3.3.1 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  11     1
## 2   2     pig  37     1
## 3   3   human  57     2
## 4   4     cat  74     2
## 5   5     pig  91     3
## 6   6   human  33     3
## 7   7     cat   6     4
## 8   8     pig  99     4
## 9   9   human   4     5
## 10 10     cat  16     5
```

3.4 nsplit

Number of windows splitter for both vectors and dataframes.

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() for dataframes

```
df_list = nsplit(df, 5)

df_list
```

```
## $`1`
##   x species age
## 1 1     cat  74
## 2 2     pig  86
##
## $`2`
##   x species age
## 3 3   human  80
## 4 4     cat  34
##
## $`3`
##   x species age
## 5 5     pig  92
## 6 6   human  52
##
## $`4`
##   x species age
## 7 7     cat   3
## 8 8     pig  18
##
## $`5`
##     x species age
## 9   9   human  16
## 10 10     cat  10
## 11 11     pig  69
## 12 12   human  35
```

3. We can get a specific dataframe

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

```
##   x species age
## 3 3   human  80
## 4 4     cat  34
```

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

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

```
## [1] 57
```

5. Using nsplit() for vectors

```
# Note that I only pass it one column!
vec_list = nsplit(df[,1], 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

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

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

3.4.1 force_equal

Getting an equal number of elements per window with nsplit.

Notice that the last dataframe/vector now contains the same number of rows/elements 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 nsplit() with force_equal on a dataframe

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

df_list
```

```
## $`1`
##   x species age
## 1 1     cat  39
## 2 2     pig  89
##
## $`2`
##   x species age
## 3 3   human   7
## 4 4     cat  24
##
## $`3`
##   x species age
```

```
## 5 5     pig  63
## 6 6   human  61
##
## $`4`
##   x species age
## 7 7     cat  25
## 8 8     pig  95
##
## $`5`
##     x species age
## 9   9   human  59
## 10 10     cat  19
```

3. Using nsplit() with force_equal on a vector

```
# Note that I only pass it one column!
vec_list = nsplit(df[,1], 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
```