Cluster-Based Bouts

Paul R. Hibbing

Introduction and Installation

This vignette will show you how to run the cluster-based bout identifier. The first step is making sure you have the `PAutilities` package (version 1.1.0 or greater) installed on your computer. To get the most current code, you can install from GitHub rather than CRAN. Here's how to do that:

```r
## devtools is a package that makes it easy to install packages from GitHub, but
## I find that it sometimes struggles to install the related packages (i.e.,
## dependencies) correctly. So here is my manual workaround. All it's doing is
## looking through a list of required packages, and installing any of them that
## haven't already been installed (they'll be skipped if they have).
## Be aware: Some of these packages have long installation times.

invisible(lapply(
  c(
    "dplyr", "equivalence", "ggplot2", "graphics", "lazyeval",
    "lubridate", "magrittr", "methods", "reshape2", "rlang",
    "stats", "Rcpp"
  ),
  function(x) if (!x %in% installed.packages()) install.packages(x)
))

devtools::install_github("paulhibbing/PAutilities", dependencies = FALSE)
```

Copy and paste the above into your R console, then hit enter to run it.

Preparation

Once you have the package installed, all you need is some activity data and the `get_bouts` function. For this demonstration, we will use the example data available within `PAutilities` itself.

```r
data(ex_data, package = "PAutilities")
```

This dataset has energy expenditure values that we can use to look at bouts of moderate-to-vigorous physical activity. Before we get there, though, we need to do a little bit of pre-processing as follows:

```r
## First, extract an intensity vector from the energy expenditure data
intensity <- PAutilities::get_intensity(ex_data$METs)

## The above command returns a factor variable -- the `get_bouts`
## function will complain about this and tell you it needs a
## character or numeric variable, so let's cast the intensity
## vector to character
intensity <- as.character(intensity)
```

Running the Code

Once we have our datastream of interest (`intensity` in this case), we can plug it into `get_bouts`. Let's see the code first, then go over what it means.

```
mvpa_bouts <- PAutilities::get_bouts(
  x = intensity,
  method = "cluster-based",
  target = "MVPA",
  target_buffer = 30,
  longest_allowable_interruption = 2,
  required_percent = 80,
  max_n_interruptions = Inf
)
```

Here is what each piece means:

- **mvpa_bouts <-** Store the function results in an object called `mvpa_bouts`
- **PAutilities::get_bouts** This tells R to find the `get_bouts` function in the `PAutilities` package. In fact, if you run `PAutilities::get_bouts` in your console, R will print the source code.
- **x = intensity** Here we specify that our input datastream is `intensity`, as defined in the earlier code.
- **method = "cluster-based"** Here we specify that R should run the cluster-based bout identification method. Right now, this is the only option, but in the future it will be possible to add others.
- **target = "MVPA"** Here we specify which behavior we are interested in. The input data (`intensity`) has values in the set `{SB, LPA, MVPA}`, and we would like to look specifically at bouts of `MVPA`, with the other behaviors being lumped together in a single group called `other`.
- **target_buffer = 30** Here we specify how our data should be stratified/partitioned. In this case, `intensity` is a minute-by-minute variable, so our setting of `30` means the data will be stratified/partitioned anytime we see 30+ consecutive minutes of `other` behavior. *It is crucial to understand that the meaning of this setting depends on the epoch length of the input data. If `intensity` was a second-by-second variable, we would need to set `target_buffer = 1800` to achieve the same 30-min threshold we are using for this example.*
- **longest_allowable_interruption = 2** Here we specify that a valid bout should not include any single interruption lasting longer than 2 minutes. *(Again, this is dependent on epoch length; a setting of 120 would be needed to achieve the same threshold for a second-by-second input variable)*
- **required_percent = 80** Here we specify that a valid bout should be interrupted for no more than 20% of its full duration.
- **max_n_interruptions = Inf** Here we specify that a valid bout can have unlimited interruptions as long as the criteria for `longest_allowable_interruption` and `required_percent` are met.

The above elements are set up to allow flexible bout criteria depending on the research question and the variable of interest. In our example, we set `max_n_interruptions = Inf` to avoid a restriction in that area – similar approaches can be taken for other settings as well, by setting them to `0` or `Inf` as appropriate. (For `required_percent`, 100 is the upper limit rather than `Inf`.)

Interpreting the Output

Now let's take a look at the output and go over what it means:

```
head(mvpa_bouts)
#>   start_index end_index values n_total_events n_value_events
#> 1         654       654   MVPA              1              1
#> 2         661       662   MVPA              1              1
#> 3         665       665   MVPA              1              1
#> 4         673       673   MVPA              1              1
#> 5         706       708   MVPA              1              1
#> 6         730       731   MVPA              1              1
#>   n_interruption_events length_total length_value length_interruption
#> 1                     0            1            1                   0
#> 2                     0            2            2                   0
#> 3                     0            1            1                   0
#> 4                     0            1            1                   0
#> 5                     0            3            3                   0
#> 6                     0            2            2                   0
#>   longest_interruption_event percent_time_engaged
#> 1                          0                  100
#> 2                          0                  100
#> 3                          0                  100
#> 4                          0                  100
#> 5                          0                  100
#> 6                          0                  100
```

This is a data frame with one row per bout. The variables are:

- **start_index** The starting point of the bout (i.e., `intensity[654]` for the first bout in this example)
- **end_index** The ending point of the bout (i.e., `intensity[654]` for the first bout in this example)
- **values** A meaningless constant (equal to the setting of `target`), left over from run length encoding
- **n_total_events** The number of distinct behavior events occurring between `start_index` and `end_index`
- **n_value_events** The number of distinct target behavior events occurring between `start_index` and `end_index` (referred to as `value` events in reference to the `values` column)
- **n_interruption_events** The number of distinct interruption events occurring between `start_index` and `end_index`
- **length_total** The combined duration of all `value` and `interruption` events
- **length_value** The combined duration of all `value` events
- **length_interruption** The combined duration of all `interruption` events
- **longest_interruption_event** The duration of the single longest interruption event
- **percent_time_engaged** Percentage of `length_total` comprised by `length_value`

As before, epoch length influences the interpretation of length variables. Keep that in mind.

Filtering the Output

In some cases, you may wish to look only at bouts that last a certain amount of time. Let's restrict to only bouts lasting at least 10 minutes (recalling again that `length` conveniently equates to `minutes` because our input was minute-by-minute data).

```
mvpa_bouts <- mvpa_bouts[mvpa_bouts$length_value >= 10, ]
head(mvpa_bouts)
#>     start_index end_index values n_total_events n_value_events
#> 40          1167      1182   MVPA              1              1
#> 47          1286      1296   MVPA              1              1
#> 80          2288      2297   MVPA              1              1
#> 96          2490      2502   MVPA              1              1
#> 97          2508      2534   MVPA              1              1
#> 156         5262      5271   MVPA              1              1
#>     n_interruption_events length_total length_value length_interruption
#> 40                      0           16           16                   0
#> 47                      0           11           11                   0
#> 80                      0           10           10                   0
#> 96                      0           13           13                   0
#> 97                      0           27           27                   0
#> 156                     0           10           10                   0
#>     longest_interruption_event percent_time_engaged
#> 40                           0                  100
#> 47                           0                  100
#> 80                           0                  100
#> 96                           0                  100
#> 97                           0                  100
#> 156                          0                  100
```

Expanding the Output

Lastly, in some cases we may want to convert our bout information back to the original length of the input (i.e., `intensity`). We can use the `bout_expand` function to accomplish that.

```
expanded <- PAutilities::bout_expand(mvpa_bouts)
str(expanded)
#>  chr [1:10080] "other" "other" "other" "other" "other" "other" "other" ...
```

And we can also append that new variable into our original dataset as well.

```
ex_data$intensity <- expanded
head(ex_data)
#>                                      FileID      Date     Time
#> 1 30200-MOS2C42140323-(2015-01-15)1sec.csv 1/15/2015 00:00:00
#> 2 30200-MOS2C42140323-(2015-01-15)1sec.csv 1/15/2015 00:01:00
#> 3 30200-MOS2C42140323-(2015-01-15)1sec.csv 1/15/2015 00:02:00
#> 4 30200-MOS2C42140323-(2015-01-15)1sec.csv 1/15/2015 00:03:00
#> 5 30200-MOS2C42140323-(2015-01-15)1sec.csv 1/15/2015 00:04:00
#> 6 30200-MOS2C42140323-(2015-01-15)1sec.csv 1/15/2015 00:05:00
#>              DateTime dayofyear minofday Axis1 Axis2 Axis3 Steps
#> 1 2015-01-15 00:00:00        15        0    60   284    13     0
#> 2 2015-01-15 00:01:00        15        1     0     0     0     0
#> 3 2015-01-15 00:02:00        15        2     0     0     0     0
#> 4 2015-01-15 00:03:00        15        3     0     0     0     0
#> 5 2015-01-15 00:04:00        15        4     0     0     0     0
#> 6 2015-01-15 00:05:00        15        5     0     0     0     0
#>   Vector.Magnitude     METs intensity
#> 1           290.56 1.324039     other
#> 2             0.00 1.000000     other
#> 3             0.00 1.000000     other
#> 4             0.00 1.000000     other
#> 5             0.00 1.000000     other
#> 6             0.00 1.000000     other
```

Conclusion

This should give you a broad sense of how to use the cluster-based bout identification method and what else you can do with it. Feel free to post an issue on the GitHub page if any of the above gives you trouble. Good luck!