Bout Analysis and Sedentary Patterns

Paul R. Hibbing

Introduction and Installation

This vignette will show you how to use `PBpatterns` for analyzing bouts (of any physical behavior) and sedentary patterns (specifically). The first step is making sure you have the `PBpatterns` package installed on your computer. Here's how:

```
## remotes is a package that makes it easy to install packages from GitHub, but
## in my experience it sometimes struggles to install the related packages (i.e.,
## dependencies) correctly. So first we'll do a 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 may have long installation times.

invisible(lapply(
  c(
    "DescTools", "ggplot2", "magrittr", "PAutilities", "purrr", "utils",
    "AGread", "PhysicalActivity", "tree", "randomForest", "knitr", "rmarkdown"
  ),
  function(x) if (!x %in% installed.packages()) install.packages(x)
))

## Once that's done, we can (hopefully) install from GitHub
remotes::install_github("paulhibbing/PBpatterns", dependencies = FALSE)
```

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

Preparation

After installation (and just like for the CRIB method), all you need is some activity data and the `analyze_bouts` function. For this demonstration, let's use some sample NHANES data. We'll also do some convenient pre-processing based on wear time analysis from the `PhysicalActivity` package.

```
## Load the data
data(example_data, package = "PBpatterns")

## Check that `PhysicalActivity` is installed
if (!"PhysicalActivity" %in% installed.packages()) install.packages(
  "PhysicalActivity"
)

## Add a timestamp variable to satisfy `PhysicalActivity` requirements
example_data$TimeStamp <- seq(
```

```r
  as.POSIXlt("2020-02-02", "UTC"),
  by = "1 min",
  length.out = nrow(example_data)
)

## Now label wear minutes
example_data <- PhysicalActivity::wearingMarking(
  example_data, perMinuteCts = 1, cts = "PAXINTEN",
  newcolname = "is_wear", getMinuteMarking = TRUE
)

## Format the `is_wear` variable as logical
example_data$is_wear <- example_data$is_wear == "w"

## And lastly, label valid indices (i.e., those that occur on days with at least
## 10 h of wear time). This is a bit of an abstract call -- you can think of it
## as a loop that labels each individual element of `PAXDAY` with the total wear
## time for that whole day.
example_data$valid_index <- sapply(
  example_data$PAXDAY,
  function(i, vec) vec[match(i, names(vec))],
  vec = tapply(example_data$is_wear, example_data$PAXDAY, sum)
) >= 600

## Here's a summary of what this tells us:
ftable(xtabs(~PAXDAY+valid_index+is_wear, example_data))
#>                    is_wear FALSE TRUE
#> PAXDAY valid_index
#> 1      TRUE                  657  783
#> 2      TRUE                  600  840
#> 3      TRUE                  666  774
#> 4      TRUE                  697  743
#> 5      TRUE                  823  617
#> 6      TRUE                  566  874
#> 7      TRUE                  715  725
```

All the days are valid based on our criteria, with wear time ranging from just over 10 hours (617 minutes on day 5) to around 14.5 hours (874 minutes on day 6).

Now, let's look at intensity classifications and subsequent analysis of intensity bouts. This dataset has activity counts that we can use to look at bouts of sedentary behavior (SB), light physical activity (LPA), and moderate-to-vigorous physical activity (MVPA). For illustration, let's say we initially coded our data as SB ($PAXINTEN \leq 100$), LPA ($PAXINTEN$ $101 - 759$), or MVPA ($PAXINTEN \geq 760$).

```r
## Determine minute-by-minute intensity
example_data$intensity <- cut(
  example_data$PAXINTEN,
  breaks = c(-Inf, 101, 760, Inf),
  labels = c("SB", "LPA", "MVPA"),
  right = FALSE
)
```

To see how we can use this in the `analyze_bouts` function, first it's a good idea to view the help page for that function.

```
?PBpatterns::analyze_bouts
```

Basic Usage and Available Methods

Any call to `analyze_bouts` will start with the same three elements: `x`, `target`, and `method`.

```
## This code is for illustration only. It will throw an informative error if you
## try to run it, but don't worry -- We will see how to use the rest of the code
## in a bit

PBpatterns::analyze_bouts(
  x = x,
  target = "MVPA",
  method = c(
    ## Choose from:
    "rle_standard",
    "CRIB",
    "Troiano_MVPA",
    "Ostendorf_MVPA",
    "SB_summary",
    "MVPA_summary"
  )
)
```

The first couple of arguments are pretty straightforward:

- **x** is the data you want to analyze. It needs to be a factor variable.
- **target** is the level of `x` for which you want the bout information.

The `method` argument is ever so slighltly more involved. As you can see, there are currently five available methods:

- **rle_standard** This is the traditional method based on run-length encoding. It simply returns information (start index, end index, and duration) about every distinct occurrence.
- **CRIB** See the CRIB vignette
- **Troiano_MVPA** This is the method of Troiano et al. (2008) for assessing bouts of MVPA.
- **Ostendorf_MVPA** This is the method of Ostendorf et al. (2018) for assessing bouts of MVPA.
- **SB_summary** This is the option to select if you want to analyze sedentary patterns. It's also the driving function behind the `profile_describe_sb` described in the sedentary profiles vignette.
- **MVPA_summary** This method is similar to `SB_summary`, but simpler and focused on MVPA instead of SB.

The first three methods return a data frame with one row per bout. The `*_summary` methods return a one-row data frame that summarizes all of the bouts.

Completing the Call and Understanding the Output

Each method requires one more argument (`epoch_length_sec`) to run properly. Additionally, you can provide values for arguments called `is_wear` (wear time indicator), `valid_indices` (valid day indicator), and `minimum_bout_duration_minutes` (the shortest allowable bout length) to them all. Apart from that, there are specialized settings you can feed into each method. This is where the help file (see `?PBpatterns::analyze_bouts`) is so important, as noted above. In that file, you can see what the relevant arguments are for each method. In most cases, there are well-defined default values, so you probably won't need to provide any extra information. But it's still good to know what's possible. In the case of `CRIB`, there are some arguments for which a default value can't be defined. So you can expect to see informative errors if you don't specify them all. Let's look at some code now.

Run-Length Encoding Standard Method

```
standard_bouts <- PBpatterns::analyze_bouts(
  example_data$intensity, "SB", "rle_standard", epoch_length_sec = 60
)

head(standard_bouts)
#>   start_index end_index values duration_minutes
#> 1           1       574     SB              574
#> 2         586       587     SB                2
#> 3         603       603     SB                1
#> 4         606       606     SB                1
#> 5         609       611     SB                3
#> 6         616       617     SB                2
```

CRIB

See the CRIB vignette.

Troiano MVPA

```
troiano_bouts <- PBpatterns::analyze_bouts(
  example_data$intensity, "MVPA", "Troiano_MVPA", epoch_length_sec = 60
)

head(troiano_bouts)
#>   start_index end_index values mvpa_min
#> 1         912       932   MVPA       17
#> 2         998      1011   MVPA       12
#> 3        1017      1063   MVPA       41
#> 4        2066      2083   MVPA       13
#> 5        2126      2145   MVPA       16
#> 6        2203      2226   MVPA       20
```

Ostendorf MVPA

```r
ostendorf_bouts <- PBpatterns::analyze_bouts(
  example_data$intensity, "MVPA", "Ostendorf_MVPA", epoch_length_sec = 60
)

head(ostendorf_bouts)
#>   start_index end_index values mvpa_min  mvpa_pct
#> 1         912       932   MVPA       17 0.8095238
#> 2         998      1011   MVPA       12 0.8571429
#> 3        1017      1063   MVPA       41 0.8723404
#> 4        2203      2226   MVPA       20 0.8333333
#> 5        3910      3922   MVPA       12 0.9230769
#> 6        4166      4188   MVPA       21 0.9130435
```

SB Summary

```r
## Note that here, we are providing extra information about wear time and valid
## days. This is optional, but it is a really good idea. In this case, if we
## don't provide this information, we will get a warning about failure to
## calculate the predicted usual bout duration.

SB_patterns <- PBpatterns::analyze_bouts(
  example_data$intensity, "SB", "SB_summary",
  is_wear = example_data$is_wear,
  valid_indices = example_data$valid_index,
  epoch_length_sec = 60
)

SB_patterns
#>   epoch_length total_weartime_min SB_bout_exclusion_threshold_minutes
#> 1           60               5356                                   0
#>   n_SB_bouts total_SB_min Q10_bout Q20_bout Q25_bout Q30_bout Q40_bout Q50_bout
#> 1        601         2346        1        1        1        1        2        2
#>   Q60_bout Q70_bout Q75_bout Q80_bout Q90_bout IQR IDR   SB_perc bout_frequency
#> 1        3        4        5        5        9   4   8 0.4380134       6.732636
#>   mean_SB_bout_min sb_0_14_hr sb_15_29_hr sb_30_Inf_hr ubd_empirical
#> 1         3.903494   31.53333    4.983333     2.583333             6
#>   ubd_predicted fragmentation_index      gini    alpha   alpha_se
#> 1      5.797587            15.37084 0.5062546 2.082909 0.04417279
```

For this method, the output yields many variables. Some are self-explanatory, but others may be more cryptic (particularly those used for sedentary profiles). Here are explanations for the ones that need it:

- **SB__bout__exclusion__threshold__minutes** reflects the setting that was provided for `minimum_bout_duration_minutes` – It is renamed in the output to avoid misinterpretation. (The latter term is clear when making the function call, but not necessarily when interpreting the output)
- The **Q*__bout** variables are bout length percentiles, in minutes.
- **IQR** and **IDR** are the interquartile and interdecile ranges, respectively, in minutes

- **SB_perc** is the percentage of total time that was spent sedentary
- **bout_frequency** is given in bouts per hour of wear time
- **sb_0_14**, **sb_15_29**, and **sb_30_Inf** give total sedentary time (minutes) in bouts of < 15 minutes, $15 - 29.9$ minutes, and ≥ 30 minutes, respectively
- **ubd_empirical** is the usual bout duration (minutes), calculated from the observed data
- **ubd_predicted** is the usual bout duration (minutes), calculated using a lonlinear modeling method
- **fragmentation_index** is given as sedentary breaks per sedentary hour
- **gini** is the Gini index
- **alpha** is alpha from the power law distribution (see Chastin & Granat (2010))
- **alpha_se** is the standard error for `alpha`

MVPA Summary

```
MVPA_patterns <- PBpatterns::analyze_bouts(
  example_data$intensity, "MVPA", "MVPA_summary", epoch_length_sec = 60
)

MVPA_patterns
#>   epoch_length total_weartime_min MVPA_bout_exclusion_threshold_minutes
#> 1           60              10080                                     0
#>   n_MVPA_bouts total_MVPA_min  MVPA_perc
#> 1          496            939 0.09315476
```

Interpretation of this output is pretty straightforward. The `MVPA_bout_exclusion_threshold_minutes` variable is interpreted the same way as its counterpart discussed in SB Summary above.

Further Tools for Sedentary Pattern Analysis

There are a couple more tools in `PBpatterns` that can be leveraged for research focused on sedentary patterns. The first is `summarize_weartime`, and its general usage looks like this:

```
PBpatterns::summarize_weartime(example_data, "is_wear", "TimeStamp")
#>   epoch_length total_weartime_min n_days weartime_hr_day
#> 1           60               5356      7        12.75238
```

On its own, this function is somewhat unremarkable. The real power comes into play when we combine it with other package code. For the next chunk (see following page), we will use the `purrr` package to apply functions separately for each day in the `example_data` object, then combine the results. This is a concise approach, but might be tough to follow – Don't worry too much about the specifics. This is just for illustration, and in the real world you can accomplish the same thing using a for loop or any other approach you're comfortable with. (Be aware of some looping limitations and alternatives, though. In R, I prefer to use loops for saving an output data file in each iteration, rather than appending the iteration's result to an existing object.)

```
## Extract information about wear time, SB patterns, and MVPA (a common
## covariate in sedentary pattern analysis)
weartime_info <- purrr::map_df(
  split(example_data, example_data$PAXDAY),
  ~ summarize_weartime(.x, "is_wear", "TimeStamp", .x$valid_index)
)

sb_bouts <- purrr::map_df(
  split(example_data, example_data$PAXDAY),
   ~ analyze_bouts(
     .x$intensity, "SB", "SB_summary",
     is_wear = .x$is_wear,
     valid_indices = .x$valid_index,
     epoch_length_sec = 60
    )
)


mvpa_bouts <- purrr::map_df(
  split(example_data, example_data$PAXDAY),
  ~ analyze_bouts(
      .x$intensity, "MVPA", "MVPA_summary",
      is_wear = .x$is_wear,
      valid_indices = .x$valid_index,
      epoch_length_sec = 60
    )
)

## Now combine all the above pieces of information (This works because all the
## objects have matching and unique `epoch_length` and `total_weartime_min`
## columns). In real life, you wouldn't have a guarantee of this. Thus, you
## would need to set up the merge using additional indicators (participant IDs
## and potentially dates as well).

d <- merge(weartime_info, sb_bouts)
d <- merge(d, mvpa_bouts)
```

Now that we have our combined weartime/SB/MVPA dataset (the object called `d`), we can use the `adjust_bout_summaries` function to calculate residualized variables suitable for modeling.

```
## Set `verbose` to TRUE if you want console updates about what's happening
adjust_bout_summaries(d, verbose = FALSE)
#>   epoch_length total_weartime_min n_days weartime_hr_day
#> 1           60                617      1        10.28333
#> 2           60                725      1        12.08333
#> 3           60                743      1        12.38333
#> 4           60                774      1        12.90000
#> 5           60                783      1        13.05000
#> 6           60                840      1        14.00000
#> 7           60                874      1        14.56667
#>   SB_bout_exclusion_threshold_minutes n_SB_bouts total_SB_min SB_hr_day
#> 1                                   0         67          351  5.850000
#> 2                                   0         84          222  3.700000
#> 3                                   0         77          457  7.616667
#> 4                                   0         89          417  6.950000
#> 5                                   0         60          182  3.033333
#> 6                                   0        106          350  5.833333
#> 7                                   0        118          367  6.116667
#>   Q10_bout Q20_bout Q25_bout Q30_bout Q40_bout Q50_bout Q60_bout Q70_bout
#> 1        1        1        1      1.0      2.0        3      4.0      5.0
#> 2        1        1        1      1.0      1.0        2      2.0      3.0
#> 3        1        1        1      1.8      2.0        3      5.0      6.0
#> 4        1        1        1      2.0      2.2        3      4.0      5.0
#> 5        1        1        1      1.0      1.0        2      3.0      3.3
#> 6        1        1        1      1.0      2.0        2      3.0      3.0
#> 7        1        1        1      1.0      2.0        2      2.2      3.0
#>   Q75_bout Q80_bout Q90_bout  IQR  IDR   SB_perc bout_frequency
#> 1     6.00      6.0     10.4 5.00  9.4 0.5688817       6.515397
#> 2     3.00      4.0      5.7 2.00  4.7 0.3062069       6.951724
#> 3     8.00      9.0     15.2 7.00 14.2 0.6150740       6.218035
#> 4     5.00      6.4     10.2 4.00  9.2 0.5387597       6.899225
#> 5     4.00      5.0      6.0 3.00  5.0 0.2324393       4.597701
#> 6     3.75      4.0      7.0 2.75  6.0 0.4166667       7.571429
#> 7     4.00      4.0      7.3 3.00  6.3 0.4199085       8.100686
#>   mean_SB_bout_min sb_0_14_hr sb_0_14_hr_day sb_15_29_hr sb_15_29_hr_day
#> 1         5.238806   3.700000       3.700000   0.7333333       0.7333333
#> 2         2.642857   3.700000       3.700000   0.0000000       0.0000000
#> 3         5.935065   4.633333       4.633333   2.4166667       2.4166667
#> 4         4.685393   5.650000       5.650000   0.7000000       0.7000000
#> 5         3.033333   3.033333       3.033333   0.0000000       0.0000000
#> 6         3.301887   5.016667       5.016667   0.8166667       0.8166667
#> 7         3.110169   5.800000       5.800000   0.3166667       0.3166667
#>   sb_30_Inf_hr sb_30_Inf_hr_day ubd_empirical ubd_predicted fragmentation_index
#> 1    1.4166667        1.4166667            10      8.987124            11.45299
#> 2    0.0000000        0.0000000             3      3.099579            22.70270
#> 3    0.5666667        0.5666667            11     10.053425            10.10941
#> 4    0.6000000        0.6000000             7      6.554283            12.80576
#> 5    0.0000000        0.0000000             4      3.813573            19.78022
#> 6    0.0000000        0.0000000             5      4.498950            18.17143
#> 7    0.0000000        0.0000000             4      3.958991            19.29155
```

```
#>        gini    alpha   alpha_se MVPA_bout_exclusion_threshold_minutes
#> 1 0.5910386 1.959570 0.11723015                                     0
#> 2 0.4070335 2.423440 0.15531001                                     0
#> 3 0.5457791 1.809017 0.09219601                                     0
#> 4 0.4868651 1.883886 0.09369168                                     0
#> 5 0.4341591 2.259012 0.16253775                                     0
#> 6 0.4781497 2.222688 0.11875792                                     0
#> 7 0.4428375 2.230039 0.11323429                                     0
#>   n_MVPA_bouts total_MVPA_min MVPA_min_day  MVPA_perc adj_total_SB
#> 1           30             45           45 0.07293355     5.971679
#> 2           89            204          204 0.28137931     3.732972
#> 3           55             96           96 0.12920592     7.634854
#> 4           65            102          102 0.13178295     6.942725
#> 5          112            217          217 0.27713921     3.018666
#> 6           78            176          176 0.20952381     5.771848
#> 7           66             98           98 0.11212815     6.027255
#>   adj_mean_SB_bout adj_sb_0_14 adj_sb_15_29 adj_sb_30_Inf adj_median_sb_bout
#> 1         4.064227    4.802658   0.51996914    0.64905224           2.429233
#> 2         2.324577    3.998792  -0.05781614   -0.20800352           1.845337
#> 3         5.759501    4.798147   2.38477520    0.45193163           2.914688
#> 4         4.755619    5.584074   0.71275659    0.64589402           3.034125
#> 5         3.174917    2.900419   0.02571892    0.09252826           2.068800
#> 6         3.895406    4.459489   0.92448040    0.38787845           2.288411
#> 7         3.973264    4.989754   0.47344923    0.56405226           2.419407
#>    adj_MVPA
#> 1  86.15584
#> 2 215.15216
#> 3 102.15155
#> 4  99.53938
#> 5 212.03907
#> 6 155.20380
#> 7  67.75819
```

The preceding code added several variables:

- **SB_hr_day** is daily SB time (hours/day)
- **sb_0_14_hr_day**, **sb_15_29_hr_day**, and **sb_30_Inf_hr_day** are sedentary time (hours/day) in bouts of < 15 minutes, $15 - 29.9$ minutes, and ≥ 30 minutes, respectively
- **MVPA_min_day** is daily MVPA time (minutes/day) – it's equivalent to `total_MVPA_min` because of the way we set up this illustration
- **adj_total_SB** is adjusted total SB (hours/day)
- **adj_mean_SB_bout** is adjusted mean SB bout length (minutes)
- **adj_sb_0_14**, **adj_sb_15_29**, and **adj_sb_30_Inf** are adjusted SB time (hours/day) in bouts of < 15 minutes, $15 - 29.9$ minutes, and ≥ 30 minutes, respectively
- **adj_median_sb_bout** is the adjusted median bout duration (minutes)
- **adj_MVPA** is adjusted MVPA time (minutes)

Wrapping Up

This should get you on your way to using `PBpatterns` for your analyses. As always, feel free to post an issue if something can be improved. This definitely a work in progress, so suggestions and tips are appreciated!