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.

```
data(example_data, package = "PBpatterns")
```

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
x <- 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.

```r
?PBpatterns::analyze_bouts
```

Basic Usage and Available Methods

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

```r
## 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",
    "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.
- **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(
  x, "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(
  x, "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
```

SB Summary

```
## Note the warning this gives about returning NA for the predicted usual bout
## duration

SB_patterns <- PBpatterns::analyze_bouts(
  x, "SB", "SB_summary", epoch_length_sec = 60
)
#> Warning: Error fitting model for predicted usual bout duration -- returning NA

SB_patterns
#>   epoch_length total_weartime_min SB_bout_exclusion_threshold_minutes
#> 1           60              10080                                    0
#>   n_SB_bouts total_SB_min Q10_bout Q20_bout Q25_bout Q30_bout Q40_bout Q50_bout
#> 1        609         7069        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        6       10   4   9 0.7012897          3.625
#>   mean_SB_bout_min sb_0_14_hr sb_15_29_hr sb_30_Inf_hr ubd_empirical
#> 1         11.60755   31.53333    4.716667     81.56667           574
#>   ubd_predicted fragmentation_index      gini    alpha   alpha_se
#> 1            NA             5.16056 0.8231683 2.004841 0.04071822
```

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(
  x, "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:

```
## First, add a timestamp variable to example_data
example_data$timestamp <- seq(
  as.POSIXlt(Sys.Date()),
  by = "1 min",
  length.out = nrow(example_data)
)

## Then, add a random wear time indicator to `example_data` (in real life, you
## might use the `PhysicalActivity` package for this)
set.seed(610)
example_data$is_wear <- sample(c(FALSE, TRUE), nrow(example_data), TRUE)

## Now, run the function
PBpatterns::summarize_weartime(example_data, "is_wear", "timestamp")
#>   epoch_length total_weartime_min n_days weartime_hr_day
#> 1           60               5003      7         11.9119
```

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, 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.)

```r
## Save intensity as a variable in the dataset
example_data$intensity <- x

## For simplicity, label each row of data as valid
example_data$valid_index <- TRUE

## Extract information about wear time, SB patterns, and MVPA (a common covariate)
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.

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.

```r
## 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                699      1        11.65000
#> 2           60                701      1        11.68333
#> 3           60                704      1        11.73333
#> 4           60                716      1        11.93333
#> 5           60                723      1        12.05000
#> 6           60                729      1        12.15000
```

```
#> 7           60                731      1        12.18333
#>   SB_bout_exclusion_threshold_minutes n_SB_bouts total_SB_min SB_hr_day
#> 1                                   0        316          557  9.283333
#> 2                                   0        251          478  7.966667
#> 3                                   0        258          471  7.850000
#> 4                                   0        293          542  9.033333
#> 5                                   0        233          424  7.066667
#> 6                                   0        327          610 10.166667
#> 7                                   0        259          464  7.733333
#>   Q10_bout Q20_bout Q25_bout Q30_bout Q40_bout Q50_bout Q60_bout Q70_bout
#> 1        1        1        1        1        1        1        2        2
#> 2        1        1        1        1        1        1        2        2
#> 3        1        1        1        1        1        1        2        2
#> 4        1        1        1        1        1        1        2        2
#> 5        1        1        1        1        1        1        2        2
#> 6        1        1        1        1        1        1        2        2
#> 7        1        1        1        1        1        1        2        2
#>   Q75_bout Q80_bout Q90_bout IQR IDR   SB_perc bout_frequency mean_SB_bout_min
#> 1        2        3        3   1   2 0.7968526       27.12446         1.762658
#> 2        2        3        4   1   3 0.6818830       21.48359         1.904382
#> 3        2        3        4   1   3 0.6690341       21.98864         1.825581
#> 4        2        3        3   1   2 0.7569832       24.55307         1.849829
#> 5        2        3        3   1   2 0.5864454       19.33610         1.819742
#> 6        2        2        4   1   3 0.8367627       26.91358         1.865443
#> 7        2        2        3   1   2 0.6347469       21.25855         1.791506
#>   sb_0_14_hr sb_0_14_hr_day sb_15_29_hr sb_15_29_hr_day sb_30_Inf_hr
#> 1   9.283333       9.283333           0               0           0
#> 2   7.966667       7.966667           0               0           0
#> 3   7.850000       7.850000           0               0           0
#> 4   9.033333       9.033333           0               0           0
#> 5   7.066667       7.066667           0               0           0
#> 6  10.166667      10.166667           0               0           0
#> 7   7.733333       7.733333           0               0           0
#>   sb_30_Inf_hr_day ubd_empirical ubd_predicted fragmentation_index      gini
#> 1                0             2      1.538130            33.93178 0.3031946
#> 2                0             2      1.809873            31.38075 0.3478494
#> 3                0             2      1.646776            32.73885 0.3153568
#> 4                0             2      1.675924            32.43542 0.3170020
#> 5                0             2      1.642600            32.97170 0.3112191
#> 6                0             2      1.721839            32.06557 0.3315297
#> 7                0             2      1.581936            33.49138 0.3210706
#>      alpha  alpha_se MVPA_bout_exclusion_threshold_minutes n_MVPA_bouts
#> 1 3.434455 0.1369488                                   0          46
#> 2 3.273723 0.1435161                                   0          76
#> 3 3.297111 0.1430119                                   0          57
#> 4 3.238046 0.1307481                                   0          38
#> 5 3.292173 0.1501652                                   0          84
#> 6 3.276806 0.1259077                                   0          14
#> 7 3.450758 0.1522827                                   0          44
#>   total_MVPA_min MVPA_min_day  MVPA_perc adj_total_SB adj_mean_SB_bout
#> 1             56           56 0.08011445     9.313659         1.761294
#> 2            100          100 0.14265335     7.993133         1.903192
#> 3             87           87 0.12357955     7.870676         1.824651
```

```
#> 4             44           44 0.06145251     9.030852         1.849941
#> 5            103          103 0.14246196     7.050677         1.820462
#> 6             15           15 0.02057613    10.139098         1.866684
#> 7             52           52 0.07113543     7.701905         1.792920
#>   adj_sb_0_14 adj_sb_15_29 adj_sb_30_Inf adj_median_sb_bout  adj_MVPA
#> 1    9.313659            0             0                   1  38.51152
#> 2    7.993133            0             0                   1  84.73733
#> 3    7.870676            0             0                   1  75.07604
#> 4    9.030852            0             0                   1  45.43088
#> 5    7.050677            0             0                   1 112.22120
#> 6   10.139098            0             0                   1  30.89862
#> 7    7.701905            0             0                   1  70.12442
```

This 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 is a big effort, and a definite work in progress, so suggestions and tips are appreciated!