How to Use Sedentary Profiles

Paul R. Hibbing

Introduction

The main purpose of this vignette is to show you how to use sedentary profiles. I have tried to write it as non-technically as possible, in recognition that most people are not R programmers, and many are not coders at all. That said, the profiles were created in R, and they need to be implemented in R. So I will not be able to avoid technical talk altogether. If you feel anything could be clearer, you are probably not alone. Please reach out on the Issues page (https://github.com/paulhibbing/PBpatterns/issues) and let me know what doesn't work or doesn't make sense. Others will thank you, and so will I!

Before you proceed with the vignette, you will need to install the following free programs, if you haven't already:

1. R (https://www.r-project.org/)
    - This is the R programming language
2. RStudio (https://rstudio.com/products/rstudio/download/#download)
    - This is technically optional, but **strongly** recommended. It is a program that enhances R by allowing you to work more interactively (type, point, click etc). It makes R into more of a "program" in the familiar sense.

You can also set up a profile on GitHub (https://github.com/join). This is optional unless you want to communicate on the PBpatterns web page (https://github.com/paulhibbing/PBpatterns), e.g., by posting an Issue. **Note:** For this vignette, the focus is on the sedentary profiles of Hibbing et al. (2021). Eventually, new profiles may be created and additional support added to this package. The plan is for all such methods to be accessible via the same `PBpatterns::sb_profile` parent function.

Setting up the Code

Once you have R, you need to get a copy of the PBpatterns code. You can do that in several ways, but I am going to focus on the easiest one. Simply open RStudio, paste the below code into your console, and press enter. If the code doesn't work, let me know. The most likely causes are 1) failure of dependencies to install, 2) problems with building this vignette, or 3) poor cross-platform coding. (The package was written using Windows 10 and may run into issues on other operating systems.)

```
if (!"remotes" %in% installed.packages()) install.packages("remotes")

remotes::install_github("paulhibbing/PBpatterns")
```

```
## If the above doesn't work, try the below.
## One or both of these extra arguments might help

remotes::install_github(
  repo = "paulhibbing/PBpatterns",
  dependencies = FALSE,
  build_vignettes = FALSE
)
```

Using the Code

To use the code, you need some accelerometer data. For now, let's start with the built in example data. Use this code to load it:

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

## If you want a sense of how the data are structured,
## un-comment and run the below. It will show you a few rows of data.

# head(example_data)
```

Before we go further, you may want to familiarize yourself with a few relevant functions. Use the below code to look at the help pages. Don't worry if you find them unhelpful right now – It's just good to know they are there. If you get stuck in the future, you can come back to them, and they may make more sense over time.

```
?PBpatterns::sb_profile
?PBpatterns::sb_profile_Hibbing2021
?PBpatterns::profile_describe_sb
```

From here, let's see how we would determine the sedentary profile for the data we loaded earlier. It ends up being fairly easy. All we have to do is run the command below. Note that the program tells us it's calculating non-wear using the Choi algorithm. This will be done automatically unless you tell it you have already run a non-wear algorithm. To do that, just tell it where the information is stored by passing a value for the 'wear' argument. (See the commented line below. In this case, that line would throw an error because we have not created a variable called 'is_wear'.) If you want to run the Choi algorithm yourself, you can do so via the `PhysicalActivity` package. (That's the same approach `PBpatterns` uses for running the algorithm – but `PBpatterns` does it in a specialized way, which is why the code isn't exported.) You can see some sample code for doing this in the section on managing your own accelerometer data.

```
PBpatterns::sb_profile(
  object = example_data, ## Give it the data you want to evaluate
  model = "both", ## Can be 'decisionTree', 'randomForest', or 'both'
  id = NULL, ## This could name a stratifying variable, if applicable
  counts = "PAXINTEN", ## Name the activity counts variable
  #wear = "is_wear", ## Name the wear time variable (TRUE for wear time, else FALSE)
  sb = 100, ## Provide the SB cut point
  minimum_bout_length = 1, ## Minimum bout length. Must be 1 or 5
```

```
  valid_indices = NULL ## Optional vector of indices that meet wear time criteria
)
#> Applying Choi non-wear algorithm (separately for each chunk specified by `id`, if applicable)
#>   decisionTree randomForest
#> 1 Intermediate Intermediate
```

Let's change the settings to illustrate how else this could work.

```
## Randomly sample 8000 row numbers to use as "valid indices". In a real
## analysis, you might use this approach to specify which rows occurred on days
## that have 10+ hours of wear time. (Notably, the `sb_profile` function does
## test for wear time internally, but it does not check for the extra criteria
## like daily wear requirements -- that's up to you to take care of beforehand)

valid_indices <- sample(
  seq(nrow(example_data)), 8000
)

PBpatterns::sb_profile(
  object = example_data,
  model = "decisionTree",
  id = "PAXDAY", ## Stratifying by day, just for illustration
  counts = "PAXINTEN",
  sb = 100,
  minimum_bout_length = 5,
  valid_indices = valid_indices
)
#> Applying Choi non-wear algorithm (separately for each chunk specified by `id`, if applicable)
#>   PAXDAY decisionTree
#> 1      1  Interrupted
#> 2      2  Interrupted
#> 3      3  Interrupted
#> 4      4  Interrupted
#> 5      5  Interrupted
#> 6      6  Interrupted
#> 7      7  Interrupted
```

So there you go! You now know how to determine the sedentary profile for one participant (or several, if you cleverly use the `id` argument). You can close this vignette if that's all you need. However, there are a few other topics you may find useful for supplementary analysis and work. That's what the rest of the vignette will cover.

Retrieving the bout distribution

When we used the `sb_profile` function, R took care of the whole profiling process for us under the hood. That means it pulled out the participant's bout distribution and analyzed it. If we want to see the distribution for ourselves, we can use the `profile_describe_sb` function in one of two ways:

1) By directly providing information for one person

```
PBpatterns::profile_describe_sb(
  is_sb = example_data$PAXINTEN <= 100, ## SB cut point
  is_wear = rep(TRUE, nrow(example_data)) ## Assume all epochs are wear time
)
#>   total_weartime_min n_SB_bouts minimum_bout_length_threshold total_SB_min
#> 1              10080        165                             5         6225
#>   Q10_bout Q20_bout Q25_bout Q30_bout Q40_bout Q50_bout Q60_bout Q70_bout
#> 1        5        5        6        6        6        8        9       11
#>   Q75_bout Q80_bout Q90_bout IQR  IDR   SB_perc bout_frequency
#> 1       12       13     23.2   6 18.2 0.6175595      0.9821429
```

2) By providing data frame input and setting up a stratified analysis (much like we did for `sb_profile`)

```
example_data$is_wear <- TRUE

PBpatterns::profile_describe_sb(
  df = example_data,
  minimum_bout_length = 1,
  id = "PAXDAY",
  wear = "is_wear",
  counts = "PAXINTEN",
  sb = 100
)
#>   PAXDAY total_weartime_min n_SB_bouts minimum_bout_length_threshold
#> 1      1               1440         62                             1
#> 2      2               1440        107                             1
#> 3      3               1440         91                             1
#> 4      4               1440         80                             1
#> 5      5               1440         69                             1
#> 6      6               1440        120                             1
#> 7      7               1440         86                             1
#>   total_SB_min Q10_bout Q20_bout Q25_bout Q30_bout Q40_bout Q50_bout Q60_bout
#> 1          839        1        1        1        1      1.4      2.0        3
#> 2          950        1        1        1        1      2.0      2.0        3
#> 3         1083        1        1        1        2      3.0      3.0        4
#> 4         1153        1        1        1        2      2.0      3.5        5
#> 5         1174        1        1        1        1      2.0      3.0        4
#> 6          933        1        1        1        1      2.0      2.0        3
#> 7          937        1        1        1        1      1.0      2.0        2
#>   Q70_bout Q75_bout Q80_bout Q90_bout  IQR  IDR   SB_perc bout_frequency
#> 1      4.0     4.75      5.0      6.0 3.75  5.0 0.5826389       2.583333
#> 2      3.0     4.00      4.0      8.0 3.00  7.0 0.6597222       4.458333
#> 3      5.0     6.00      7.0     11.0 5.00 10.0 0.7520833       3.791667
#> 4      6.3     8.25     10.2     20.0 7.25 19.0 0.8006944       3.333333
#> 5      5.6     6.00      7.0     12.0 5.00 11.0 0.8152778       2.875000
```

```
#> 6      3.0     4.00      5.0      8.0 3.00  7.0 0.6479167       5.000000
#> 7      3.0     3.75      4.0      6.5 2.75  5.5 0.6506944       3.583333
```

After retrieving the bout distribution

If we manually run `profile_describe_sb`, we can feed the output directly into `sb_profile`. This allows us to look at both the bout distribution and the sedentary profile, although it does mean we have to do two steps instead of one. Using our example data, we could implement this two-step process like so:

Method 1 above

```
bout_info <- PBpatterns::profile_describe_sb(
  is_sb = example_data$PAXINTEN <= 100,
  is_wear = example_data$is_wear
)

profile <- PBpatterns::sb_profile(bout_info)
## (We can get more sophisticated output if we add extra settings
## like we did before, but this is fine for now)

print(profile)
#> $decisionTree
#> [1] Intermediate
#> Levels: Interrupted Intermediate Prolonged
#>
#> $randomForest
#> [1] Interrupted
#> Levels: Interrupted Intermediate Prolonged
```

Method 2 above

```
bout_info <- PBpatterns::profile_describe_sb(
  df = example_data,
  minimum_bout_length = 1,
  id = "PAXDAY",
  wear = "is_wear",
  counts = "PAXINTEN",
  sb = 100,
  simplify = FALSE ##<-- This is critical
)

profile <- PBpatterns::sb_profile(bout_info, model = "randomForest")

print(profile)
#> [1] Intermediate Interrupted  Intermediate Prolonged    Intermediate
#> [6] Interrupted  Interrupted
#> Levels: Interrupted Intermediate Prolonged
```

Managing your own accelerometer data

Before wrapping up, let's address how you can get your own data into R, and how you can pre-process it so you can apply the PBpatterns code. That will ultimately depend on what type of monitor data you are using. I can't be exhaustive here, but I will give an example that will hopefully help. In the code below, I'll show how you might handle data from an ActiGraph data file. If you use a different monitor, the trick will be finding tools that allow you to use these same concepts on your specific data. In many cases, the tools are probably out there. If not, you might end up being the one to provide them by the time you're finished!

```
## First, make sure you have the right packages installed

  packages <- c("AGread", "PhysicalActivity", "magrittr")
  invisible(lapply(
    packages, function(x) if (!x %in% installed.packages()) install.packages(x)
  ))

## Attach the magrittr package (makes code more readable via pipe operators like %>%)

  library(magrittr)

## Find an example data file

  ag_file <-
    system.file("extdata/example1sec.csv", package = "AGread") %T>%
    {stopifnot(file.exists(.))}

## Process the file using various packages, and
## store the result in an object called AG

  AG <-

    ## Read and reintegrate
    AGread::read_AG_counts(ag_file) %>%
    AGread::reintegrate(60, direction = "forwards") %>%

    ## The next part is a hack I've needed in the past in order to get the
    ## non-wear algorithm to work
    within({
      TimeStamp = as.character(Timestamp)
    }) %>%

    ## Now run the algorithm
    PhysicalActivity::wearingMarking(
      perMinuteCts = 1, TS = "TimeStamp", cts = "Axis1",
      newcolname = "is_wear", getMinuteMarking = TRUE
    ) %>%

    ## Format the wear time variable to a logical vector
    within({
      is_wear = is_wear %in% "w" ## use %in% rather than == because of how it handles NA
    }) %>%

    ## Get the sedentary profile
```

```
    cbind(., PBpatterns::sb_profile(
      ., counts = "Axis1", wear = "is_wear", model = "decisionTree"
    )) %>%

    ## Take the `TimeStamp` variable back out
    .[ ,names(.) != "TimeStamp"] %>%

    ## Rename the sedentary profile variable from 'decisionTree' to 'SB_profile'
    stats::setNames(., gsub("^decisionTree$", "SB_profile", names(.)))

## View the data

  AG
#>             Timestamp      Date     Time Axis1 Axis2 Axis3 Steps Lux
#> 1 2019-02-14 08:58:00 2/14/2019 08:58:00     0     0     0     0   0
#> 2 2019-02-14 08:59:00 2/14/2019 08:59:00  1594  1529  1041     6   0
#> 3 2019-02-14 09:00:00 2/14/2019 09:00:00  9379  6709 11298    30   0
#>   Inclinometer.Off Inclinometer.Standing Inclinometer.Sitting
#> 1               60                     0                    0
#> 2               16                    24                    0
#> 3                0                    59                    1
#>   Inclinometer.Lying Vector.Magnitude is_wear  weekday days   SB_profile
#> 1                  0             0.00   FALSE Thursday    1 Intermediate
#> 2                 20          2441.79   FALSE Thursday    1 Intermediate
#> 3                  0         16143.76   FALSE Thursday    1 Intermediate
```

That's it! Thanks for following along. Again, let me know on GitHub if there are improvements I can make.