How to Use The Sedentary Profiles

Paul R. Hibbing

Introduction

The main purpose of this vignette is to show you how to use the 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/SBprofiles/issues) and let me know what doesn't work or doesn't make sense. Others will thank you, and I will too!

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 SBprofiles web page (https://github.com/paulhibbing/SBprofiles), e.g., by posting an Issue.

Setting up the Code

Once you have R, you need to get a copy of the SBprofiles 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, or 2) poor cross-platform coding. (The package was written using Windows 10 and may run into issues on other operating systems.) There's also a chance your system won't like this vignette, which gets lumped into the installation with all the code and everything else. Regardless, please let me know so I can find a solution.

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

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

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 = "SBprofiles")

# print(head(example_data)) #<-- Use this to view a few rows of the data
```

Before we go further, you may want to familiarize yourself with the main functions available in the SBprofiles package. 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.

```
?SBprofiles::sb_bout_dist
?SBprofiles::get_profile
?SBprofiles::nhanes_wear
```

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 following command:

```
SBprofiles::get_profile(
  object = example_data, ## Give it the data you want to evaluate
  method = "both", ## Can be 'decisionTree', 'randomForest', or 'both'
  id = NULL, ## This could name a stratifying variable if applicable
  counts = "PAXINTEN", ## Name the activity counts variable
  sb = 100, ## Provide the SB cut point
  min_bout = 1, ## Minimum bout length. Must be 1 or 5
  valid_indices = NULL ## Optional vector of indices that meet wear time criteria
)
#> Registered S3 methods overwritten by 'tibble':
#>   method     from
#>   format.tbl pillar
#>   print.tbl  pillar
#>   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 `get_profile` function does
## test for wear time internally, but it does not check for the extra criteria
## like daily wear requirements)

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

SBprofiles::get_profile(
  object = example_data,
  method = "decisionTree",
  id = "PAXDAY", ## Stratifying by day, just for illustration
  counts = "PAXINTEN",
  sb = 100,
  min_bout = 5,
  valid_indices = valid_indices
)
#>   PAXDAY decisionTree
#> 1      1  Interrupted
#> 2      2  Interrupted
#> 3      3  Interrupted
#> 4      4  Interrupted
#> 5      5  Interrupted
#> 6      6  Interrupted
#> 7      7 Intermediate
```

So there you go! You have now determined the sedentary profile for one participant (or several, if you have cleverly used 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 `get_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 `sb_bout_dist` function in one of two ways:

1) By directly providing information for one person

```
SBprofiles::sb_bout_dist(
  is_sb = example_data$PAXINTEN <= 100, ## SB cut point
  is_wear = SBprofiles::nhanes_wear(
    example_data$PAXINTEN
  ) ## Choi non-wear algorithm
)
#>   total_weartime_min n_SB_bouts min_bout_threshold total_SB_min Q10_bout
#> 1               5356        157                  5         1502        5
#>   Q20_bout Q25_bout Q30_bout Q40_bout Q50_bout Q60_bout Q70_bout Q75_bout
#> 1        5        6        6        6        7        8       10       11
#>   Q80_bout Q90_bout IQR IDR   SB_perc bout_frequency
#> 1       12       16   5  11 0.2804332       1.758775
```

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

```
SBprofiles::sb_bout_dist(
  df = example_data,
  min_bout = 1,
  id = "PAXDAY",
  counts = "PAXINTEN",
  sb = 100
)
#>   PAXDAY total_weartime_min n_SB_bouts min_bout_threshold total_SB_min Q10_bout
#> 1      1                783         60                  1          182        1
#> 2      2                840        106                  1          350        1
#> 3      3                774         89                  1          417        1
#> 4      4                743         77                  1          457        1
#> 5      5                617         67                  1          351        1
#> 6      6                874        118                  1          367        1
#> 7      7                725         84                  1          222        1
#>   Q20_bout Q25_bout Q30_bout Q40_bout Q50_bout Q60_bout Q70_bout Q75_bout
#> 1        1        1      1.0      1.0        2      3.0      3.3     4.00
#> 2        1        1      1.0      2.0        2      3.0      3.0     3.75
#> 3        1        1      2.0      2.2        3      4.0      5.0     5.00
#> 4        1        1      1.8      2.0        3      5.0      6.0     8.00
#> 5        1        1      1.0      2.0        3      4.0      5.0     6.00
#> 6        1        1      1.0      2.0        2      2.2      3.0     4.00
#> 7        1        1      1.0      1.0        2      2.0      3.0     3.00
#>   Q80_bout Q90_bout  IQR  IDR   SB_perc bout_frequency
#> 1      5.0      6.0 3.00  5.0 0.2324393       4.597701
#> 2      4.0      7.0 2.75  6.0 0.4166667       7.571429
#> 3      6.4     10.2 4.00  9.2 0.5387597       6.899225
#> 4      9.0     15.2 7.00 14.2 0.6150740       6.218035
#> 5      6.0     10.4 5.00  9.4 0.5688817       6.515397
#> 6      4.0      7.3 3.00  6.3 0.4199085       8.100686
#> 7      4.0      5.7 2.00  4.7 0.3062069       6.951724
```

After retrieving the bout distribution

If we manually run `sb_bout_dist`, we can feed the output directly into `get_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. For right now, this two-step process only works if `sb_bout_dist` is used according to option #1 above. Implementing it with our example data would look like this:

```
bout_info <- SBprofiles::sb_bout_dist(
  is_sb = example_data$PAXINTEN <= 100,
  is_wear = SBprofiles::nhanes_wear(
    example_data$PAXINTEN
  )
)

profile <- SBprofiles::get_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] Interrupted
#> Levels: Interrupted Intermediate Prolonged
#>
#> $randomForest
#> [1] Interrupted
#> Levels: Interrupted Intermediate Prolonged
```

Managing your own accelerometer data

Before we wrap up, we should address how to get your own data into R, and how to pre-process it so you can apply the SBprofiles 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
    ) %>%

    ## Get the sedentary profile
    cbind(., SBprofiles::get_profile(
      ., counts = "Axis1", method = "decisionTree"
    )) %>%

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

    ## Convert the `is_wear` variable to logical
    within({
      is_wear = ifelse(
        as.character(is_wear) == "nw", FALSE, TRUE
      )
    }) %>%

    ## 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
```

Thank you for following along. Again, let me know on GitHub if there are improvements we can make.