How to Use The Sedentary Profiles

Introduction

The main purpose of this vignette is to show you how to use the sedentary behavior (SB) profiles. We recognize most people are not R programmers, and many are not coders at all. As such, this has been written to get the point across as plainly as possible. That said, the profiles were created in R, and they need to be implemented in R. So we will not be able to avoid technical spots entirely. If you feel our guidance could be improved, please reach out on the Issues page (https://github.com/paulhibbing/SBprofiles/issues).

You will get the most out of this vignette if you install the following free programs first:

1. R (https://www.r-project.org/)
   - This is the R programming language
2. RStudio (https://rstudio.com/products/rstudio/download/#download)
   - This 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.
3. Git (https://git-scm.com/downloads)
   - This is a program that tracks how code changes over time. It is widely used for software development, and R/RStudio have some very convenient interfaces with Git, which will make it easy to access `SBprofiles` from its online repository (GitHub).

You can also set up a profile on GitHub (https://github.com/join). This is not essential, but it will make it easier to communicate on the `SBprofiles` web page (https://github.com/paulhibbing/SBprofiles) if things go wrong.

Setting up the Code

Once you have R running, you need to get a copy of the `SBprofiles` code. You can do that in several ways, but we are going to focus on the easiest one. Simply open RStudio, paste the below code into your console, and press enter.

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

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

Using the Code

To use the code, you need some accelerometer data. We will get to that, but for now we will start with the built in example data. Use this code to load it:

```
data(example_data, package = "SBprofiles")
print(head(example_data))
#>        SEQN PAXSTAT PAXCAL PAXDAY PAXN PAXHOUR PAXMINUT PAXINTEN
#> 50401 21010       1      1      7    1       0        0        1
#> 50402 21010       1      1      7    2       0        1        0
#> 50403 21010       1      1      7    3       0        2        0
#> 50404 21010       1      1      7    4       0        3        0
#> 50405 21010       1      1      7    5       0        4        0
#> 50406 21010       1      1      7    6       0        5        0
```

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 SB profile for the data we loaded earlier. It ends up being fairly easy. All we have to do is the following:

```
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
)
#>   decisionTree randomForest
#> 1      Limiter      Limiter
```

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

```
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 = sample(
    seq(nrow(example_data)), 8000
  ) ## Randomly designate 8000 minutes as 'wear time' minutes
    ## Note: get_profile does test for wear time internally, but
    ##       it does not check for criteria like 10+ hours on 4+ days.
    ##       That's where this variable would come into play
)
#>   PAXDAY decisionTree
#> 1      1      Avoider
#> 2      2      Avoider
#> 3      3      Avoider
#> 4      4      Avoider
#> 5      5      Avoider
#> 6      6      Avoider
#> 7      7      Avoider
```

So there you go! You have now determined the SB 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 this vignette will cover.

Retrieving the bout distribution

When we used the `get_profile`, R took care of the whole profiling process for us under the hood. One of the things it did was pull out the participant's bout distribution using the function `sb_bout_dist`. If we want to see the distribution for ourselves, we can use that function directly in one of two ways:

1) By directly providing information for a single stratum

```
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
)
#>   Q10_bout Q20_bout Q25_bout Q30_bout Q40_bout Q50_bout Q60_bout Q70_bout
#> 1        5        5        6        6        6        7        8       10
#>   Q75_bout Q80_bout Q90_bout IQR IDR total_SB_raw n_bouts total_wear_min
#> 1       11       12       16   5  11         1502     157           5356
#>     SB_perc bouts_weartime min_bout_threshold
#> 1 0.2804332     0.02931292                  5
```

2) By providing data frame input for stratified analysis (much like `get_profile`)

```
SBprofiles::sb_bout_dist(
  df = example_data,
  min_bout = 1,
  id = "PAXDAY",
  counts = "PAXINTEN",
  sb = 100
)
#>   PAXDAY Q10_bout Q20_bout Q25_bout Q30_bout Q40_bout Q50_bout Q60_bout
#> 1      1        1        1        1      1.0      1.0        2      3.0
#> 2      2        1        1        1      1.0      2.0        2      3.0
#> 3      3        1        1        1      2.0      2.2        3      4.0
#> 4      4         1         1         1       1.8       2.0         3      5.0
#> 5      5         1         1         1       1.0       2.0         3      4.0
#> 6      6         1         1         1       1.0       2.0         2      2.2
#> 7      7         1         1         1       1.0       1.0         2      2.0
#>   Q70_bout Q75_bout Q80_bout Q90_bout  IQR  IDR total_SB_raw n_bouts
#> 1      3.3     4.00      5.0      6.0 3.00  5.0          182      60
#> 2      3.0     3.75      4.0      7.0 2.75  6.0          350     106
#> 3      5.0     5.00      6.4     10.2 4.00  9.2          417      89
#> 4      6.0     8.00      9.0     15.2 7.00 14.2          457      77
#> 5      5.0     6.00      6.0     10.4 5.00  9.4          351      67
#> 6      3.0     4.00      4.0      7.3 3.00  6.3          367     118
#> 7      3.0     3.00      4.0      5.7 2.00  4.7          222      84
#>   total_wear_min   SB_perc bouts_weartime min_bout_threshold
#> 1            783 0.2324393     0.07662835                  1
#> 2            840 0.4166667     0.12619048                  1
#> 3            774 0.5387597     0.11498708                  1
#> 4            743 0.6150740     0.10363392                  1
#> 5            617 0.5688817     0.10858995                  1
#> 6            874 0.4199085     0.13501144                  1
#> 7            725 0.3062069     0.11586207                  1
```

After retrieving the bout distribution

We can feed the distribution information directly into `get_profile`. This has the drawback of making the profiling process two steps instead of one, but it has the advantage of making it easy for us to look at both the bout distribution and the profile. The two-step way would look like this:

```
bout_info <- 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
)

profile <- SBprofiles::get_profile(
  bout_info
) ## Output can get more sophisticated if you add extra
  ## settings like we did before

print(profile)
#> $decisionTree
#> [1] Avoider
#> Levels: Avoider Limiter Prolonger
#>
#> $randomForest
#> [1] Avoider
#> Levels: Avoider Limiter Prolonger
```

Managing your own accelerometer data

Non-Wear Algorithm

The Choi algorithm is available elsewhere, but `SBprofiles` does offer a wrapper through `SBprofiles::nhanes_wear`. The only advantage of this wrapper is that it is able to account for the lack of formatting in the timestamps of NHANES accelerometer files. The original function (`PhysicalActivity::wearingMarking`) has other features that may make it better for most applications.

Reading and Formatting Data

There are packages to help with this too.