Sleep stage classification – thinking process

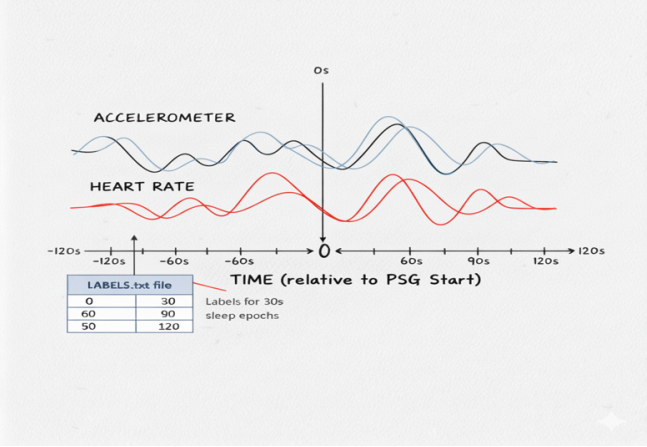
1. Assumptions

- The model will work in an 'autoregressive' way. What I mean by this is that given N timestamps of heart beats and accelerometer measurements, we will predict the sleep stage at the current timestamp. So, in other words, to predict the stage at the timestamp N, we will use timestamps N-L where L is a lookback window size (the past measurements).
- Given the low subject number, the model will not classify between Wake, N1, N2, N3, REM. We will simplify this by simplifying the labels to Wake, NREM, REM (like the paper).
- I will discard the step count, which only represents the number of steps in previous days. I could use that to estimate the time of day based on the activity of the user (assuming that the highest number of steps was achieved mid-day) but to be a robust feature I would need 3-7 days before the PSG start.

2. Problem description

31 subjects wore an Apple Watch for several days before the experiment. The data is structured such that for each feature (accelerometer, heart rate) we have a single .txt file per subject. Inside the .txt file we have the timestamp and the measurement value. We have also a .txt file with the labels. The labels start from timestamp 0 and the sampling rate is for 30s sleep epochs (so labels look like 0, 30, 60, … and so on).

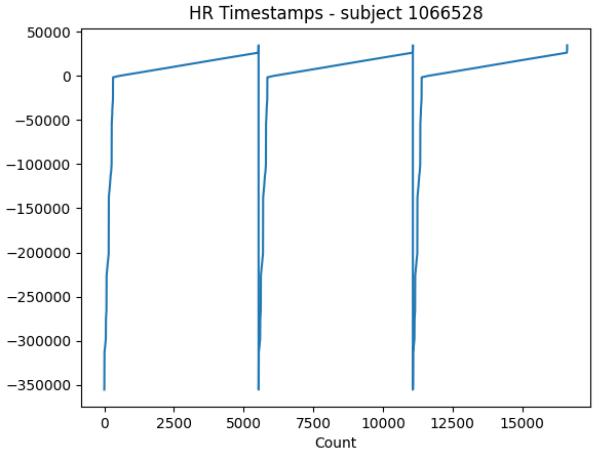
Timestamp 0 from the labels denote the PSG (experiment) start. In this experiment all the users slept in a laboratory for close monitoring of the sleep. The timestamps for the heart rate and accelerometer are relative to this start of the PSG, so we will have negative and positive timestamps as well.



3. EDA

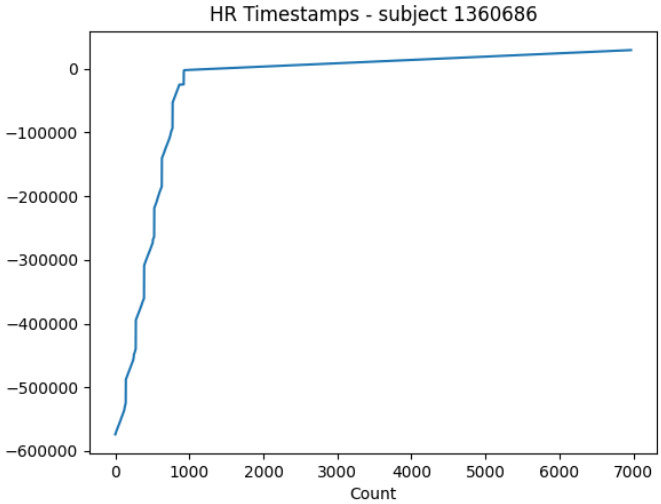
The first step I am taking is to examine the data. What I mean by this is to simply plot the accelerometer data, heart rate data and the labels. All timestamps are relative to the PSG start (the PSG represents the night when subjects slept in the laboratory)

During this first stage, I encountered a problem in data. For the heart rate measurements, for some users, there is a strange pattern in the timestamps.



So, as you can see in the figure above, the same timestamp gets repeated 3 times, with a reset-like pattern. I find this to be strange because, normally, a timestamp should not be able to do this, because the time cannot go backwards.

To validate this observation, let's visualize how the HR timestamps look for other subjects:



So, everything looks normal here, no reset in HR timestamps.

I am assuming that this is a measurement error. To test this, I will segment the data at the timestamp resets and check if the heart rate values are identical across segments.

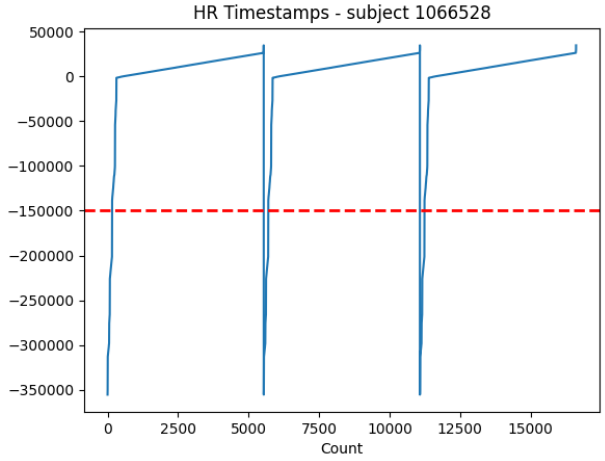
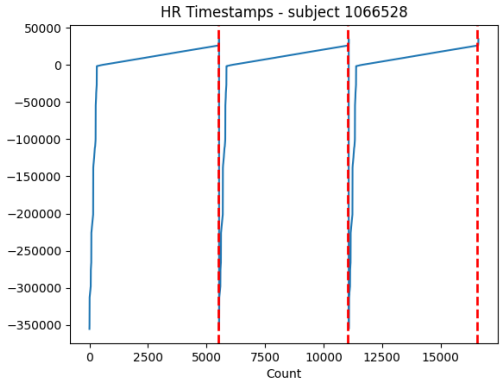


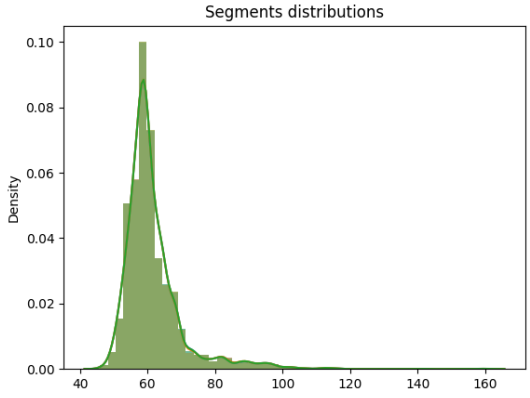
Figure 1 Experiment ilustration

We have the following values:

```
# this is for subject 1066528
hr_timestamp = np.array(hr_timestamp)
mask = hr_timestamp == hr_timestamp[159]
mask.sum() # is 3, so we have the exact same timestamp 3
times
hr[mask] # 72, 72, 72
```

To test this further, I will split each section where the timestamps reset, I will gather the heart rate data for each segment, and I'll plot their distributions. If the data repeats, the distributions should be the same.



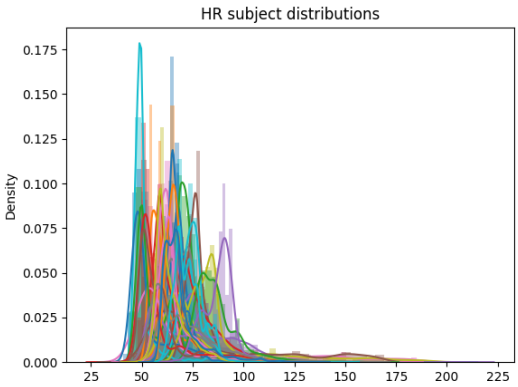


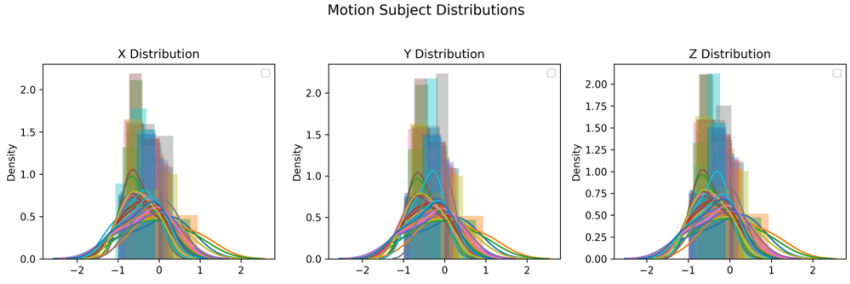
The distributions of the heart rates bpm in the segments are the same, so it's safe to say that this confirms a data collection error.

To solve this problem, we have two options: (1) skip the problematic subjects, but this would lose valuable data in an already small dataset, or (2) keep only one segment per subject. I chose the second option.

Another problem that can appear is how different the subjects can be. For example, for a person that is more athletic, a 54 RHR (resting heart rate) would be normal, while for a person that is not so active, a higher RHR is also normal. Basically, the bottom line is that each metrics is relative to the subject baselines.

Here is the HR (heart rate) and accelerometer measurements distributions per subject.



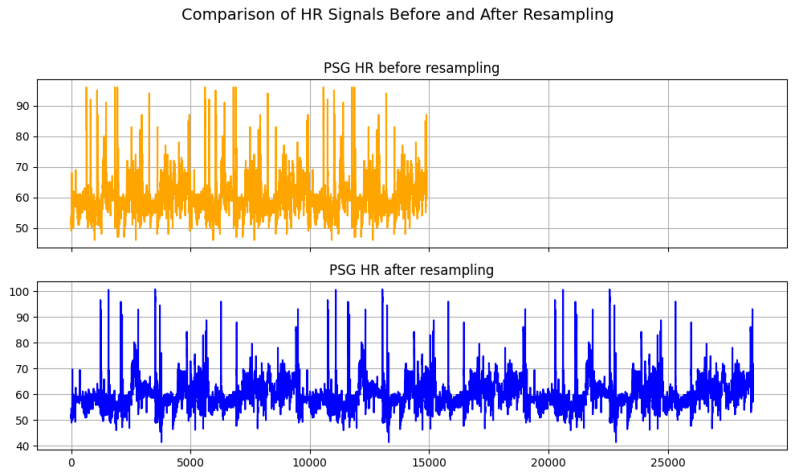


As you can see in the plots above, each subject is different, and this will pose a challenge in our data-limited approach. Why am I saying this? Basically, if we train on 90% of the data and test on the 10%, the odds of those 10% being out of the training data distribution are not negligible.

This challenge primarily affects heart rate, as motion patterns are more consistent across individuals.

4. Feature engineering

As noted in the referenced paper, the most important features for sleep stage classification are HRV (heart rate variability) and the motion. The first problem that I've encountered is the frequency of the measurements. To be able to create meaningful features for the model, I would need to bring both measurements to the same temporal scale, so I've resampled HR data and accelerometer data to 1Hz.



We cannot compute HRV because we do not have data from an ECG (we need consecutive R-waves on the ECG measurements) since we only have heart rate in BPM from the Apple Watch sampled at varying intervals. To overcome this, I decided to follow the paper's approach and compute standard deviation for different window sizes as a 'proxy' to HRV.

Before computing the rolling std dev, we normalize the HR for each user individually.

```
subject_hr_mean = hr_resampled.mean()
subject_hr_std = hr_resampled.std()
hr_normalized = (hr_resampled - subject_hr_mean) / (subject_hr_std + 1e-8)
```

Here is how the rolling std dev looks for 30, 60 and 120 s windows for a smaller interval.

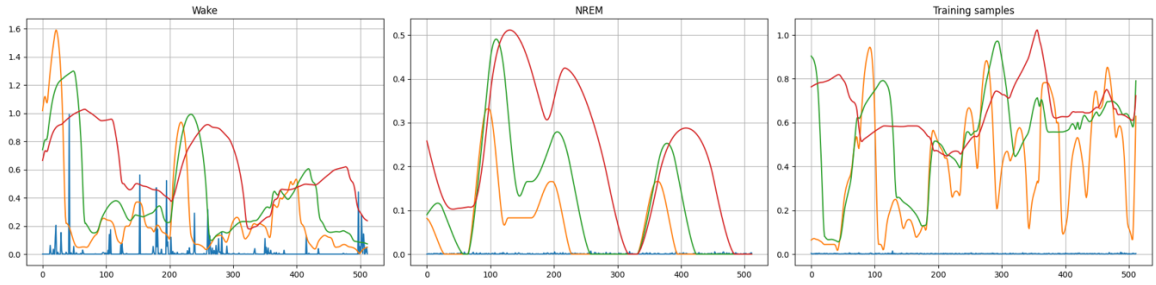
By choosing these windows, we can capture short, medium, and long-term variability of the heart rate which is helpful for detecting REM stage.

For the accelerometer, I computed the magnitude of the movement and extracted Earth gravity.

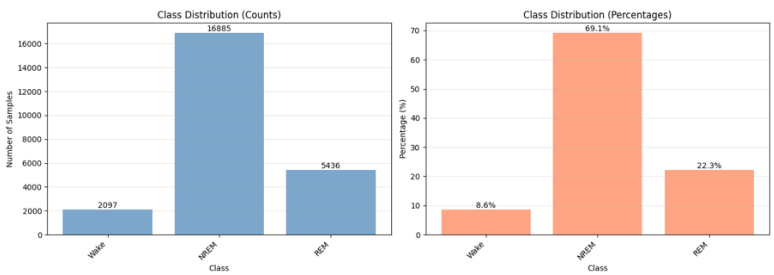
$$magnitude = \sum \sqrt{x^2 + y^2 + z^2} - 1$$

I chose only these features because the paper stated that raw HR is useless and it does not help.

After doing all the preprocessing mentioned above, here are some training windows. For the Wake state, we should see high movement and high HR variability, for NREM we should see low HR variability and sometimes motion and for REM we should NOT see any motion (muscle lock) and high HR variability.



After I have the data, the next step would be to look at the class distributions of the dataset.



Data is dominated by the NREM class, so we should take this into account when training the model.

5. Model selection and training

For training the model, as stated in the assumptions section, I will use the past L timestamps. I chose L to be 512 in this case, so approximately 8 minutes before the current timestamp for which we will predict the stage. This is different from the paper, because they are using 10-minute windows centered around the timestamp for which they will predict the stage. Their model is non-causal as it uses future timestamps, so it cannot be used for real time prediction.

First, to set a baseline, I would train a simple logistic regression. Initially, training on 90% of the data and validating on 10% in a single run yields the following results.

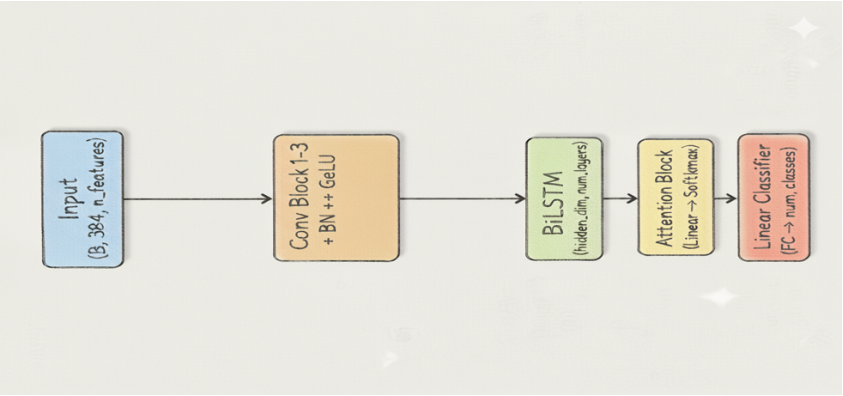
Per-class Accuracy	
Wake	0.43
NREM	0.758
REM	0.559

And for 5 different runs on different train-val splits:

	Per-class Accuracy	Per-class std dev
Wake	40.92	0.051
NREM	68.03	0.07
REM	57.03	0.16

Based on my experience with time-series data, a Conv + LSTM + Bahdanau attention model should converge faster and achieve better results. I also tested a transformer encoder with bidirectional attention

The LSTM architecture is shown below:



For model training, I will use AdamW optimizer with a cosine scheduler with warmup. Due to the class imbalance, I will use Focal Loss combined with class weights. In my

experience, class imbalanced models benefit from a low learning rate, small batch size (32-64), and gradient norm clipping.

After training the LSTMs on 5 different train-val splits:

	Per-class Accuracy	Per-class std dev
Wake	25.48	0.06
NREM	80.44	0.07
REM	48.70	0.043

After training the transformer on 5 different train-val splits:

	Per-class Accuracy	Per-class std dev
Wake	28.21	0.07
NREM	74.94	0.064
REM	37.6	0.02

The models performance could be suboptimal because no hyperparameters tuning was performed. Also, the model suffered from overfit. To tackle overfitting, I added dropout, clipping of the grad norm, weight decay, early stopping and making the networks smaller, but it wasn't enough.

6. Future improvements

Basically. I could train a VAE unsupervised using all the data, not just the data from the PSG, and then, to take the encoder of the VAE to instantiate the weights for the classification model, so the model has a better starting point.

Also, we could use augmentations to create synthetic data for under-represented classes.