UCLouvain

École polytechnique de Louvain

Automatic classification of sleep stages for children using wearable device

Author: **Tom Przybylski**
Supervisors: **John Lee, Michel Verleysen**
Readers: **Dounia Mulders, Olivier Staquet**
Academic year 2021–2022
Master [120] in Biomedical Engineering

Abstract

Sleep related breathing disorders (SRBD) are known to have adverse effects on infants. Indeed, it was proven to be strongly linked with abnormal cognitive development and Sudden Infant Death Syndrome. Therefore, it is of prime importance to diagnose SRBD as early as possible to intervene and ensure a normal and healthy development of the patient.

The current golden standard for assessing the quality of sleep of an infant is Polysomnography (PSG). It consists in a very complete exam (EEG, ECG, EMG, Air flow measurement...) performed in a special facility or unit of the hospital overnight. PSG has a number of drawbacks : it is inconvenient for the patient and his family, it requires the presence of trained technicians to equip the patient, check that the procedure goes without failures and interpret the recordings. Moreover, the hospital environment may change the infant's behaviour and therefore have an impact on his sleep during the recording.

An alternative is at-home monitoring, using an armband. Gabi SmartCare has developed such a band as well as a whole processing system downstream in order to detect suspicious events and diagnose SRBD without the need of a PSG. A part of this process is the classification of the recording clips in sleep stages, as the sleep stage in which an event happens affects the interpretation that is made of it. The aim of this thesis is to implement that classifier.

We show that a classification can be performed using a pool of several patients as training and following the same process that Boe & al. used for a similar project on adults patients [3].

Acknowledgements

I would firstly like to thank John Lee and Michel Verleysen for their availablity and precious advices throughout the year.

I would also like to thank Olivier Staquet from Gabi SmartCare for his availablity, time and the interest he showed for my work. I also thank him for the data, insights and comments he provided me, as well as the warm welcome I got when visiting his office.

Finally, I would like to thank all those who supported me in their ways throughout year and the difficulties, and the people I discovered or rediscovered in the mean time.

Contents

Introduction

Organization of the report

This report is organized as follows :

Section 1 : Context provides the context in which this master thesis fits itself: the medical need, the current solution, the alternative that is considered and the state of the art on this new alternative, as well as the precise objective of this master thesis.

Section 2 : Data is a review of the data at our disposal. More precisely, we describe the patients as well as the different signals acquired and their characteristics.

Section 3 : Methods is itself divided in subsections : first, a summary of the preprocessing operations that have been performed on the data to make it useful and suitable for further processing. Then the construction of the initial features based on the raw signals is explicitated, as well as several further transformation to improve the performance of the classifier. Finally, a description of the splitting of the data and the structure of the models is given.

Section 4 : Results shows the results obtained from all the precedent steps, on the different variations of the model.

Section 5 : Discussion consists in the analysis of the results, several leads for further improvements and comments on the overall performance of the entire method.

Finally, the **Conclusion** is a summary of everything covered in this report.

1 Context

Sleep related breathing disorders (SRBD) are known to have adverse effects on infants. Indeed, Thach (2005) wrote that he could *'[...]conclude that there is strong, albeit indirect, evidence indicating that Obstructive Sleep Apnea (OSA) either predisposes an infant to Sudden Infant Death Syndrome (SIDS) or actually acts as the final critical stressor causing death.'*[1]
Moreover, it was proven that there is a strong link between SRBD and abnormal cognitive development : *'it was demonstrated that there was an unusually high prevalence of snoring and nocturnal gas exchange abnormalities in a cohort of children who were academically poor achievers. When successful therapeutic intervention was administered to those children in whom Sleep Associated Gas Exchange Abnormalities (SAGEA) was present, significant improvements in school grades occurred. It is therefore recommended that sleep- related symptoms should be actively sought in children with developmental or learning problems, and that referral for evaluation of sleep-disordered breathing should occur early rather than late'*[2].
Therefore, the early diagnosis of SBRD is crucial to intervene as soon as possible and ensure a normal and healthy development of the patient.

1.1 Golden standard

A polysomnography (PSG) (see Figure 1) is the concurrent recording of several physiological signals of the patient such as the respiratory volume, the EEG, ECG, EOG and sometimes EMGs. The 'somno' part means that it is performed while the patient is sleeping. A PSG requires a qualified staff to perform several tasks : apply and monitor the numerous sensors, verify the quality of the recorded data, score the sleep stages, detect important patterns in the signals or various events such as movements, arousals...

Figure 1: Polysomnography of a child

Despite being the current golden standard in terms of quality of sleep evaluation, PSG has a number of limitations : *'the financial costs and resource burden associated with PSG data acquisition, the subsequent scoring of sleep records, and the discomfort to patients can outweigh the benefit of this system's high accuracy and limit its potential for long-term sleep assessment.'*[3]
Indeed, a PSG requires a full overnight staff as well as cumbersome equipment, and a trained clinician to score the sleep stages after the recording. Furthermore, the patient is not familiar with the environment in which the PSG takes place, which can affect his behaviour during the evaluation. As a consequence, the evaluation may not be representative of the usual quality of sleep of the patient.
As a way to make the process much faster and to relieve the clinician from this tedious task, some have developed tools to automatically detect and classify events of interests in the recorded signals from the PSG. [4][5][6]

The automatic processing of the polysomnographic recordings is a a solution to half of the problem : the processing is much more effective but the recording is still cumbersome.

1.2 Alternatives - State of the art

To overcome that burden, prototypes of automatic event detection involving wearable armbands have been developed to replace PSG in the task of determining sleep stages in a significantly more ergonomic way. A first step towards this is to effectively determine the sleep stages of the patient. Indeed, it was shown that the sleep stage in which an event occurs is crucial to determine whether it is meaningful in the diagnosis process.[7]
As of today, such bands have proven to be quite effective on adults to distinguish between sleep and wake stages, as shown in the comparison made by Boe & al. in their article on the use of ActiWatch as an alternative to PSG ([3]). However, the differentiation between different sleep stages is still poor (see Figure 2). It is yet to be determined whether a similar method could be used on infants. Indeed, the physiological behaviours of infants and adults are significantly different, as shown in the review from Bue & al.[8]

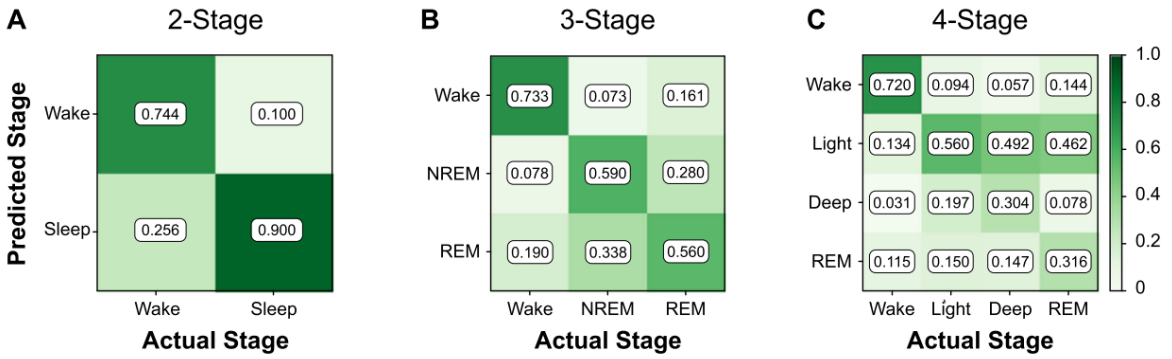


Figure 2: Sleep stages classification by Boe & al. [3]

1.3 Objective of the thesis

In this master thesis, we will implement a program that is able to classify sleep phases based on the signals provided by a band designed especially for that purpose : the Gabi Baby Band (Figure 3). It is an armband that has been developed by Gabi SmartCare, a Belgian MedTech company. The band is a part of a whole data managing structure : the band collects raw data, which is locally preprocessed then sent to the cloud for further processing. At the other end, an interface for the clinician to work directly on the data, remotely.
In that global framework, we want to include a module that automatically and almost instantly classifies the recordings into specific sleep phases, so that further processing (event detection, prediciton...) can follow right away.

Figure 3: Gabi Baby Band

2 Data

2.1 Patients

The data used in this work has been collected by Gabi Smartcare during a clinical trial performed on 43 patients trouhgout 2021. Those 43 patients have undergone polysomnography while wearing the Gabi armband, such that the data produced by the band could be confronted with the data from the PSG. Among those 43 recordings, 13 have been annotated by a clinician and therefore contain information about the sleep stages throughout the PSG. Their characteristics are summarized in Table 1, as well as the percentage of sleep stages they went trough during the recording.
'Other' refers to undetermined or not scored segments.

Table 1: Characteristics of the patients and recorded sleep stages

Patient #	**Age (months)**	**Gender**	**Total time (s)**	**Wake (%)**	**Light sleep (%)**	**Deep sleep (%)**	**REM (%)**	**Other (%)**
0	2	F	32970	25.11	22.57	16.47	35.85	0
1	4	M	38880	11.73	25.54	27.16	24.47	9.1
2	8	F	32220	16.2	49.53	19.74.	14.53	0
3	2	F	33390	28.48	0	0	0	71.52
4	45	M	26430	25.99	42.68	16.35	7.38	7.6
5	0	M	33480	43.19.	0	0	11.38	45.43
6	32	M	32400	9.44	34.35	25	22.96	8.24
7	3	F	25980	0.46	33.03	13.63	10.97	41.92
8	1	M	36630	21.46	0	0	0	78.54
9	44	F	30780	3.12	4.39	7.7	0	84.8
10	4	M	36660	15.22	0	0	0	84.78
11	41	M	30750	21.17	25.37	23.02	16	14.44
12	34	F	4467	26.87	17.39	36.27	12.09	7.39

2.2 Signals

The band effectively records accelerometry and temperature (at skin and accelerometer levels), and with a combination of LED's several other signals are computed : pulse rate, heart rate variability, respiratory rate and blood O2 saturation. Along with these measurements, the Gabi team also computed for each datapoint 4 Signal Quality Indices (SQI), which give insight on the quality of the data at the specific point in time. Table 2 summarizes the different recorded signals along with their units and sampling frequencies, and the file it comes from.

Table 2: Recorded signals

Signal Name	**Unit**	**Sampling frequency**
Pulse Rate	BPM	1 Hz
Oxygen Saturation (SpO2)	%	1 Hz
Heart Rate Variability	ms	1 Hz
Respiratory Rate	BrPM	1 Hz
Signal Quality Index (IBI)	%	1 Hz
Signal Quality Index (G)	%	1 Hz
Signal Quality Index (R)	%	1 Hz
Signal Quality Index (I)	%	1 Hz
X	G	10 Hz
Y	G	10 Hz
Z	G	10 Hz
Skin Temperature	°C	0.2 Hz
Accelerometer Temperature	°C	0.2 Hz

3 Methods

Because Boe & al.'s paper is aiming at the same goal than us with the only difference being the target public, it was taken as reference for the present work. However, we do not rigourously follow the article and some things are done differently here. Those differences are detailed hereunder, in the appropriate sections.

3.1 Preprocessing

3.1.1 Temporal alignment

The first step in preprocessing was to make the different signals correspond in terms of time. Indeed, the times at which the recordings started and ended did not exactly match depending on the sensor and as aforementioned, the sampling frequencies were also different (see example on Figure 4).

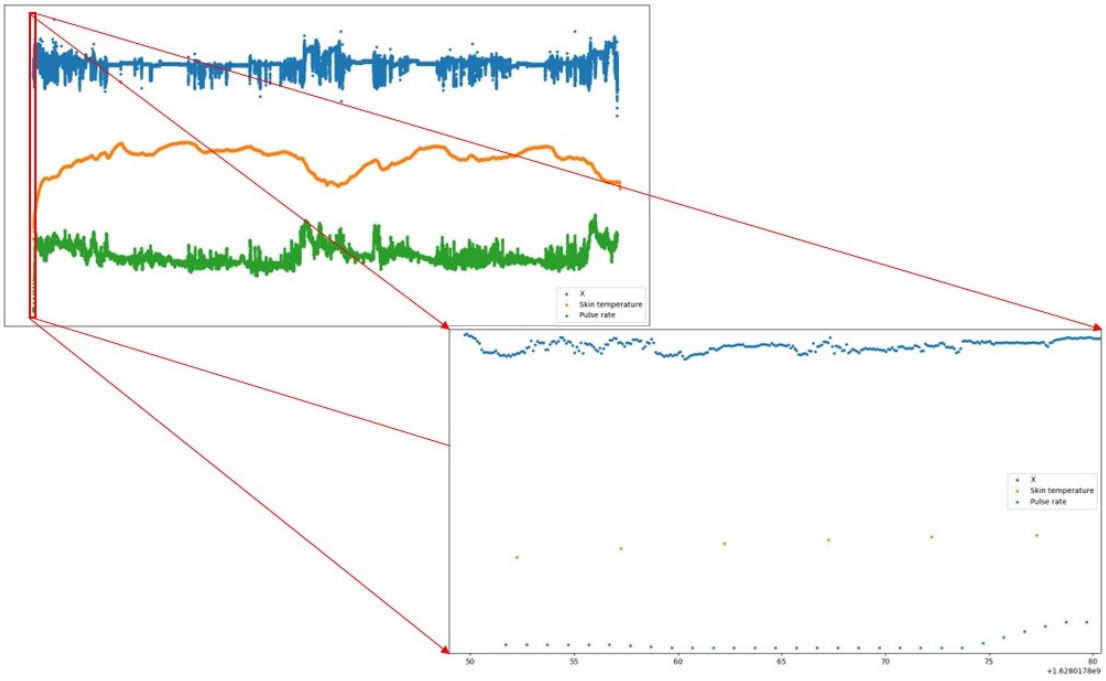


Figure 4: Different starting time and sampling frequency

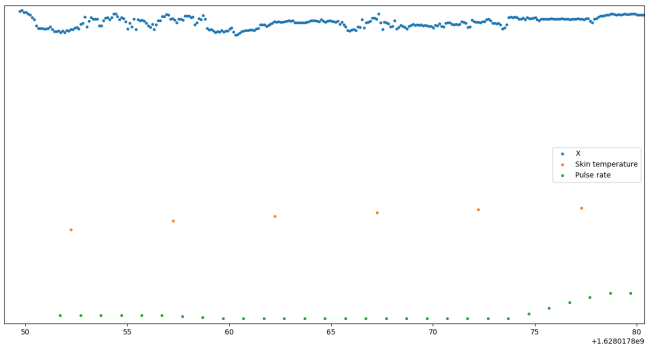
To do so, we grouped the signals in 3 dataframes according to the device that recorded them : DF_{plx} contains the signals sampled at 1 Hz, DF_{Accs} contains the accelerometer signals (10Hz) and DF_{temp} contains the temperature data (0.2Hz). Each dataframe contains the values of the recorded signals as well as the corresponding timestamps. The target vector, of which we also know the timstamps, was cropped along with the dataframes. The cropping was performed according to the following algorithm :

```
for p in n_patients:
    start_time[p] = max(DF_plx[p,timestamps[0]],DF_Accs[p,timestamps[0]],DF_temp[p,timestamps[0]],
     target[p,timestamps[0]])
    end_time[p] = min(DF_plx[p,timestamps[-1]],DF_Accs[p,timestamps[-1]],DF_temp[p,timestamps[-1]],
     target[p,timestamps[-1]])
    DF_plx[p] = DF_plx[p,start<timestamps<end]
    DF_Accs[p] = DF_Accs[p,start<timestamps<end]
    DF_temp[p] = DF_temp[p,start<timestamps<end]
    target[p] = target[p,start<timestamps<end]
```

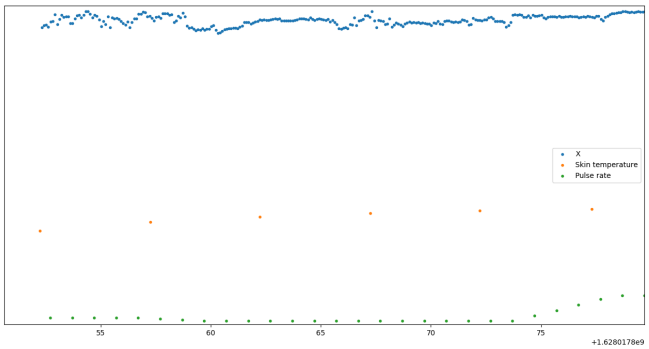
Then, we interpolated the dataframes and target with the lowest sampling frequencies (0.2 and 1 Hz) on every timestamp of DF_{Accs} to match the frequency of the signals with the highest temporal resolution (10 Hz) (see Figure 5). By doing so, we ensured that each point in time corresponded to a datapoint in each signal, as well as the annotations.

```
for p in n_patients:
    timestamps[p] = DF_Accs[p,timestamps]
    DF_plx[p] = interpolate(X_init = DF_plx[p,timestamps], f_init = DF_plx[p,data], x_interp =
     timestamps[p])
    DF_Accs[p] = interpolate(X_init = DF_Accs[p,timestamps], f_init = DF_Accs[p,data], x_interp =
     timestamps[p])
    DF_temp[p] = interpolate(X_init = DF_temp[p,timestamps], f_init = DF_temp[p,data], x_interp =
     timestamps[p])
    target[p] = interpolate(X_init = target[p,timestamps], f_init = target[p,values], x_interp =
     timestamps[p])
```

Original signals



Cropped signals



Interpolated signals

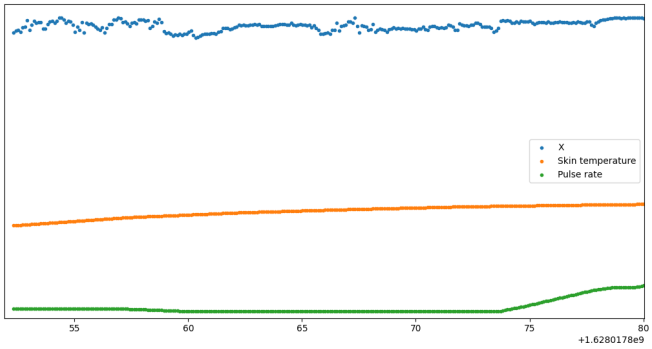


Figure 5: Cropping and interpolation of the signals

Because of an apparent problem in the recording, the timestamps of several signals from patient 11 did not overlap at all and the data was therefore entirely discarded by the cropping. We decided to remove him from the patient pool entirely.

3.1.2 Segmentation

As was done in [3], we decided to segment the signals into 30 seconds clips.

3.1.3 Removal of useless segments

Certain segments were annotated as 'Undetermined' or 'Not scored' (see Section 2.1). As we wanted to perform supervised learning, those segments were useless and therefore discarded.

3.1.4 Segments after preprocessing

Table 3: Characteristics of the patients and recorded sleep stages

Patient #	Age (months)	Gender	Number of segments	Wake	Light sleep	Deep sleep	REM
0	2	F	1116	291	249	182	394
1	4	M	1260	227	336	351	346
2	8	F	896	172	425	160	139
3	2	F	313	312	0	0	1
4	45	M	668	90	366	144	68
5	0	M	380	328	4	3	45
6	32	M	172	2	80	26	64
7	3	F	522	40	280	117	85
8	1	M	10	7	2	1	0
9	44	F	159	60	28	70	1
10	4	M	220	218	2	0	0
12	34	F	1395	320	327	427	321

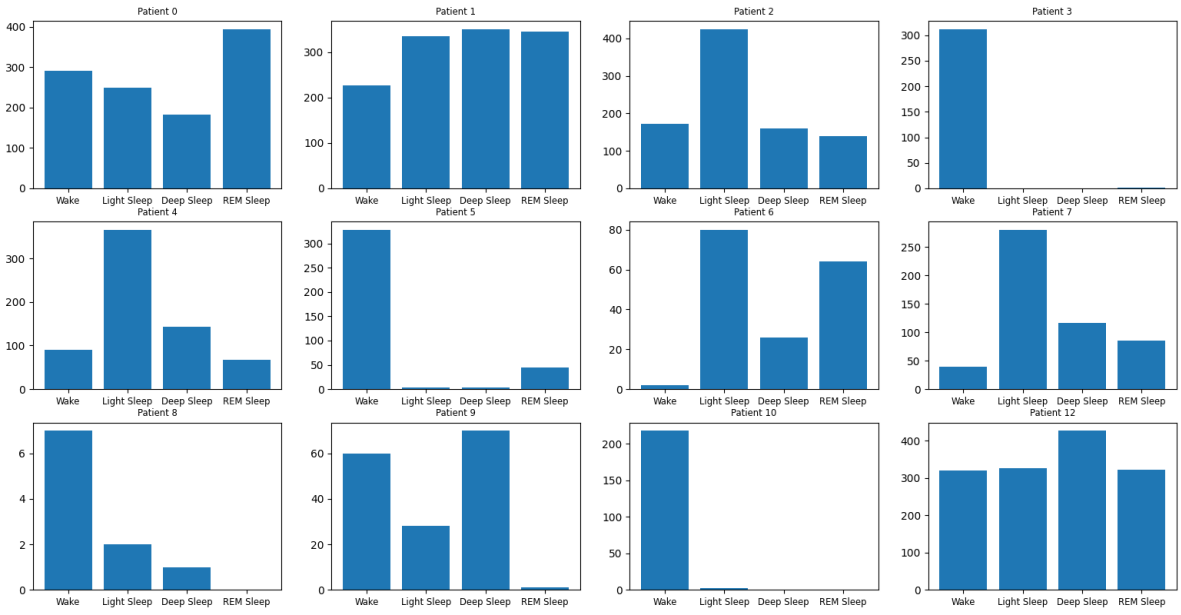


Figure 6: Distribution of sleep stages in each patient

Figure 6 shows imbalance in almost all patients, except patients 0, 1 and 12. Patients 3 and 10 show very

severe imbalance with almost only wakefulness.

The programmes used for importing and building the target from the annotation file, data loading, cropping and interpolating, and segmenting can be found in appendices 2 to 5 respectively.

3.2 Feature construction

3.2.1 Initial features

The choice of features was based on [3], which used the features displayed in Figure 7.

Table 4. Features extracted from the sensor data.

Sensor modality	Sampling frequency (Hz)	No. of features	Features
Accelerometer	62.5	33	Mean (x,y,z) Minimum (x,y,z) Maximum (x,y,z) Range (x,y,z) Interquartile range (x,y,z) Standard deviation (x,y,z) Kurtosis (x,y,z) Root mean squared (x,y,z) Variance (x,y,z) Pearson's coefficient (x,y,z) Pearson's *p* value (x,y,z)
ECG	1000	14	Mean R-R interval Minimum R-R interval Maximum R-R interval Standard deviation R-R interval RMSSD NN50, PNN50 NN20, PNN20 VLF, LF, HF LF/HF Ratio
Skin temperature	0.0167	4	Mean DPG Minimum DPG Maximum DPG Range DPG

RMSSD root mean square of successive differences; NN*X* number of successive R-R intervals that differ by more than *X* ms, PNN*X* ratio of NN*X* to total number of R-R intervals, *VLF* very low frequency power (activity in the 0.003–0.04 Hz frequency band); *LF* low frequency power (activity in the 0.04–0.15 Hz frequency band), *HF* high frequency power (activity in the 0.15–0.40 Hz frequency band); *DPG* distal-to-proximal gradient[35]

Figure 7: Features used by Boe & al.[3]

Although we decided to start with these features, some changes were brought :

Accelerometer We realized that using each direction could be misleading in our case. Indeed, it was surprising that the most significant features to differentiate wakefulness from sleep were related to the z-axis specifically. Olivier Staquet brought insight on this : in the process of the PSG, the bed can be lifted if the patient wakes up to take care of him (eg. to feed him). Therefore, it is a very strong correlate which is completely irrelevant in the perspective of generalizing to a monitoring outside of the hospital evaluation room. It was therefore decided to use actigraphy instead, which is computed as follows :

$$A(t) = \sqrt{(x(t)^2 + y(t)^2 + z(t)^2)} - 1 \tag{1}$$

The 9 first accelerometer features were then computed on the actigraphy instead of the independant components. The 2 last features (Pearson's coefficient and p-value) refer to correlation between the x-y-z components and where not used in consequence.

ECG Although we do not have ECG data, we have got similar information through the band's LED's and therefore we can compute similar features. We started with the mean, minimum, maximum and standard deviation for both pulse rate and heart rate variability, as well as for the inter-pulse interval (inverse of the rate).

Temperature We had access to both the skin and the accelerometer temperatures, which allowed to compute (almost) the same features as in the reference article. Indeed, we decided to split skin and accelerometer temperatures to have more flexibility. The gradient is nothing but a linear combination of the two.

Other features In addition to those features, it was decided to compute the mean, minimum value, maximum value and range of the SpO2 and Respiratory rate over each segment. Age was also added to complete our feature set.

3.2.2 Signal Quality Index correction

The Signal Quality Indices (SQI) could be used to determine whether each datapoint from a specific signal was reliable or not. A set of rules was provided by Gabi SmartCare to determine said reliability :

$$\begin{aligned} SQI_{IBI} > 50\% &\longrightarrow \text{Suitable for Pulse Rate related features} \\ 0.3 \times SQI_G + 0.4 \times SQI_R + 0.3 \times SQI_I > 20\% &\longrightarrow \text{Suitable for SpO2 related features} \\ SQI_G > 20\% &\longrightarrow \text{Suitable for Respiratory Rate related features} \\ \text{Always} &\longrightarrow \text{Suitable for Actigraphy related features} \end{aligned}$$

These were used to keep the bad quality datapoints from being taken into account while constructing the features (see section 3.2), signal by signal. For example, a datapoint that does not satisfy the quality rule for the respiratory rate will not be used to compute the mean respiratory rate on the segment to which it belongs, but it will still be used to compute the mean actigraphy as the quality is always ensured for this signal.

3.2.3 Removal of outliers

Once all features are computed, we have to take care of the eventual outliers. To do so, we find outlier segments according to the following rule :

for p in $0 < p < 12$ (patient) , for f in $0 < f < 32$ (feature), for i in $0 < i <$ Number of segments for patient p:

$$\text{if } \left| S_{p,f,i} - \overline{S_{p,f}} \right| > 10\sigma_{S_{p,f}} \longrightarrow \text{Discard } S_{p,i} \tag{2}$$

In other words, if the value of any feature from a segment is far from the patient's mean by more than 10σ, the whole segment is discarded.

3.2.4 Feature transformations

The last part of the preprocessing is the transformation of several features. Indeed, some distributions seemed very unbalanced upon visual examination, typically standard deviations. It was thus decided to apply a transformation to several of them. An example is given in Figure 8 (Standard deviation of the pulse rate passed through the logarithmic function), and Appendix 1 summarizes those transformations. Finally, the data are scaled in the interval $[0, 1]$.

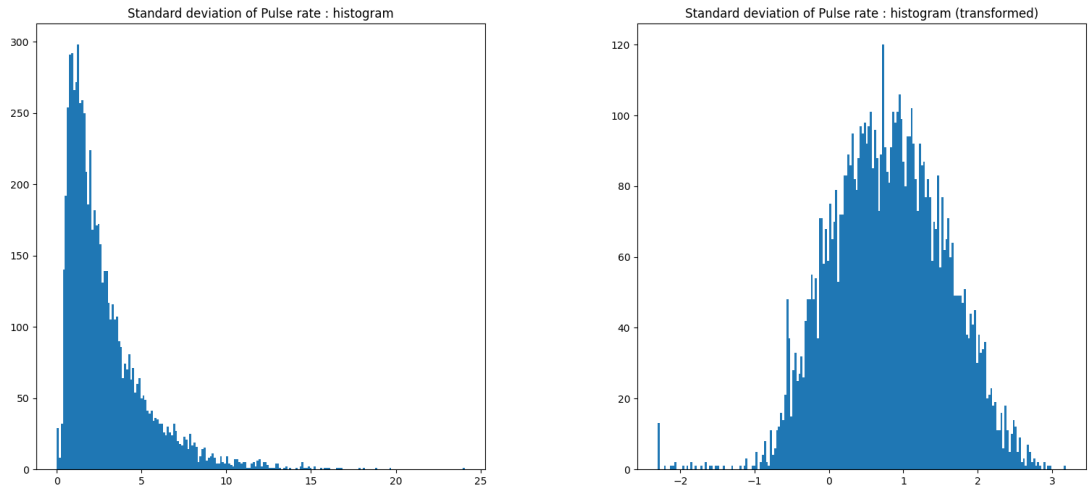


Figure 8: Feature histogram before and after transformation

3.2.5 Feature selection

In order to reduce redundant information, decrease model complexity and therefore improve generalizability and computational efficiency, we perform feature selection. Table 4 is a summary of all the features used, as well as the p-value of the χ^2 test, which determines whether the target is dependant with each feature. It was decided to keep features with a p-value under .05 (significatively dependant), which leaves us with 24 out of our 37 initial features.

Table 4: Features

Feature #	Signal	Operation	χ^2 p-value
0	Accelerometry (Actigraphy, see X)	Mean	4,6E-15
1		Min	2,9E-17
2		Max	6,7E-26
3		Range	7,9E-10
4		Inter-Quartile-Range	5,9E-20
5		Standard Deviation	4,4E-13
6		Kurtosis	5,0E-24
7		Variance	5,2E-09
8	Skin Temperature	Mean	4,8E-01
9		Min	4,8E-01
10		Max	4,6E-01
11		Standard Deviation	9,3E-02
12	Accelerometer Temperature	Mean	2,3E-01
13		Min	2,8E-01
14		Max	2,8E-01
15		Standard Deviation	8,6E-01
16	Heart Rate Variability	Mean	3,7E-03
17		Min	3,6E-04
18		Max	5,8E-02
19		Standard Deviation	8,0E-01
20	Respiratory Rate	Mean	1,1E-23
21		Min	8,5E-26
22		Max	9,1E-21
23		Standard Deviation	1,5E-03
24	Pulse Rate	Mean	1,1E-23
25		Min	2,0E-17
26		Max	8,2E-24
27		Standard Deviation	1,4E-04
28	Inter-Pulse Interval	Mean	5,2E-21
29		Min	4,0E-21
30		Max	6,5E-16
31		Standard Deviation	3,5E-01
32	SpO2	Mean	6,5E-02
33		Min	8,4E-02
34		Max	4,6E-02
35		Standard Deviation	3,8E-03
36	Age	/	1,4E-26

3.2.6 Dimensionality reduction

To further decrease the complexity of the model, we performed Principal Component Analysis. We decided to choose a number of components that explained 99.5% of the variance. The transformation left us with 18

new components (see Figure 9).

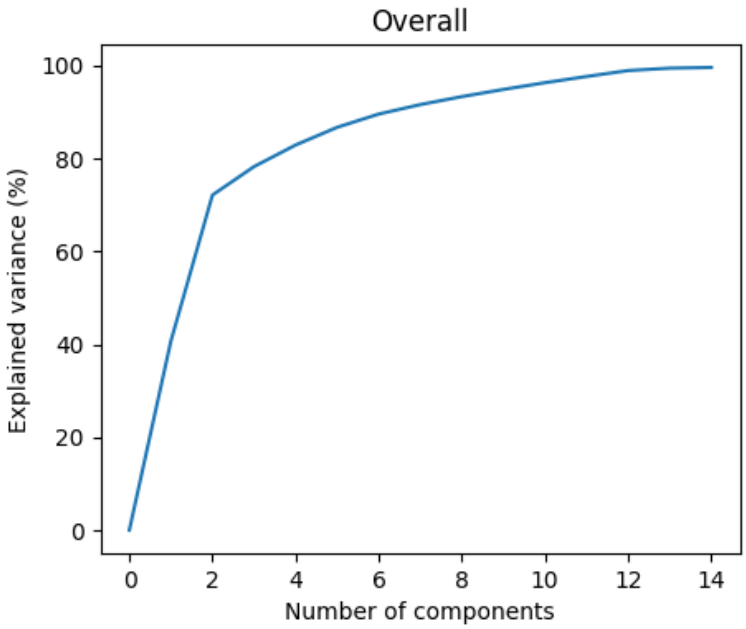


Figure 9: Cumulative explained variance of the components

3.3 Processing

3.3.1 Training and test sets

The aim of this study is to build models that can effectively classify sleep stages in **new patients**. It is therefore crucial to train and evaluate our models on different patients. Boe used a leave-one-out testing scheme, that is train the model on all patients but one, use this last patient as the test set and repeat for every patient, then reports the results (Figure 10).
This configuration is indeed the closest possible to the actual case of the model being built on our entire dataset and then used on a new patient.
Internal 3-fold is used within the training set to choose optimal meta-parameters for our models.

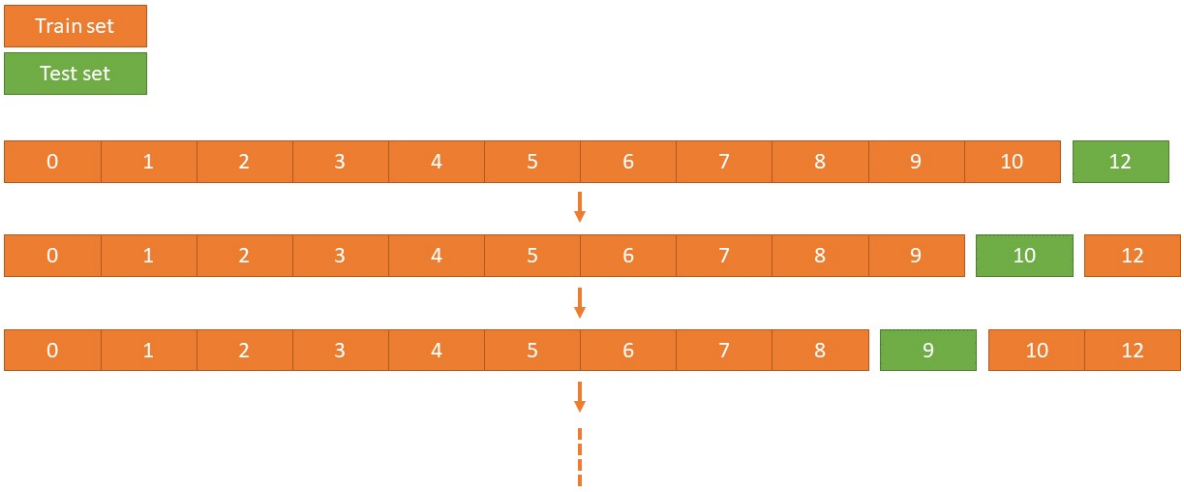


Figure 10: Leave-one-out cross-validation

3.3.2 Models

As in Boe's work ([3]), we explored the classification of sleep stages at 3 different levels :

1. wake vs. sleep
2. wake vs. non-REM sleep vs. REM sleep
3. wake vs. light sleep vs. deep sleep vs. REM sleep

Several models were built, and their parameters optimized through 5-fold cross-validation. The structures used were logistic regression, decision tree, support vector machine and random forest, mainly because of the low amount of meta-parameters involved.

1-stage vs 2-stages classifier It was suspected that the wake segments would be marginal compared to all 3 kinds of sleep segments. Therefore, a composite 2-level classifier was considered. The first level is a wake vs. sleep, one-versus-rest classifier (C1), whereas the second differentiates between sleep stages (C2). Then, the results of the second layer replace the segments labeled as wakefulness (Figure 11). This can be useful in the case we want to use 2 different structures successively. In the results section, we review how that composite structure affects the performance.

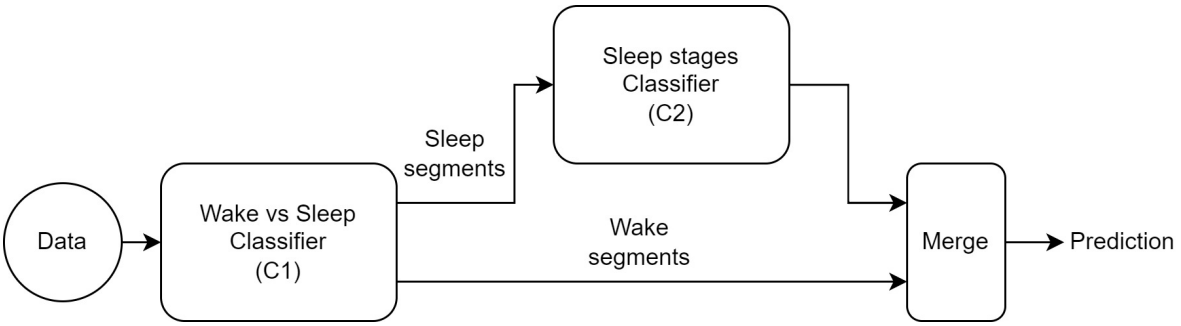


Figure 11: 2-level classifier

In order to optimize our 2-level classifier, we restart the feature selection with respect to the binarized, wake vs. sleep target on the one hand and the target that contains only sleep segments on the other hand (see Table 5). We also re-run Principal Components Analysis on those significant features, still with the 95% variance threshold. By doing so, we extract 12 components for the binary case and 10 for the asleep case (see Figure 12).

Table 5: Selected features for the 1-level classifier, and for the binary and asleep levels of the 2-level classifier

Feature #	**Signal**	**Operation**	χ^2 p-value (Overall)	χ^2 p-value (Binary)	χ^2 p-value (Asleep)
0	Accelerometry (Actigraphy)	Mean	4,6E-15	3,01E-07	7,91E-11
1		Min	2,9E-17	1,64E-04	1,47E-15
2		Max	6,7E-26	2,57E-27	9,53E-02
3		Range	7,9E-10	5,64E-03	2,04E-09
4		Inter-Quartile-Range	5,9E-20	8,34E-22	7,05E-01
5		Standard Deviation	4,4E-13	5,54E-02	1,57E-13
6		Kurtosis	5,0E-24	9,19E-11	5,25E-14
7		Variance	5,2E-09	9,54E-01	9,68E-10
8	Skin Temperature	Mean	4,8E-01	6,88E-01	3,12E-01
9		Min	4,8E-01	6,63E-01	3,18E-01
10		Max	4,6E-01	7,10E-01	2,98E-01
11		Standard Deviation	9,3E-02	1,19E-02	9,51E-01
12	Accelerometer Temperature	Mean	2,3E-01	7,10E-01	1,22E-01
13		Min	2,8E-01	6,67E-01	1,60E-01
14		Max	2,8E-01	5,84E-01	1,71E-01
15		Standard Deviation	8,6E-01	5,61E-01	8,07E-01
16	Heart Rate Variability	Mean	3,7E-03	4,55E-01	1,70E-03
17		Min	3,6E-04	1,82E-01	2,94E-04
18		Max	5,8E-02	9,62E-01	2,38E-02
19		Standard Deviation	8,0E-01	3,19E-01	9,99E-01
20	Respiratory Rate	Mean	1,1E-23	4,98E-21	5,42E-06
21		Min	8,5E-26	5,51E-23	2,93E-06
22		Max	9,1E-21	1,72E-18	2,00E-05
23		Standard Deviation	1,5E-03	1,96E-04	4,69E-01
24	Pulse Rate	Mean	1,1E-23	4,78E-23	9,58E-04
25		Min	2,0E-17	3,39E-17	4,28E-03
26		Max	8,2E-24	3,43E-23	9,23E-04
27		Standard Deviation	1,4E-04	3,85E-05	1,60E-01
28	Inter-Pulse Interval	Mean	5,2E-21	1,14E-19	1,03E-03
29		Min	4,0E-21	8,32E-20	1,05E-03
30		Max	6,5E-16	4,72E-15	3,38E-03
31		Standard Deviation	3,5E-01	7,73E-02	9,14E-01
32	SpO2	Mean	6,5E-02	7,40E-03	9,79E-01
33		Min	8,4E-02	1,04E-02	9,68E-01
34		Max	4,6E-02	4,76E-03	9,84E-01
35		Standard Deviation	3,8E-03	1,14E-03	2,79E-01
36					
37	Age	/	1,4E-26	6,27E-20	1,79E-08

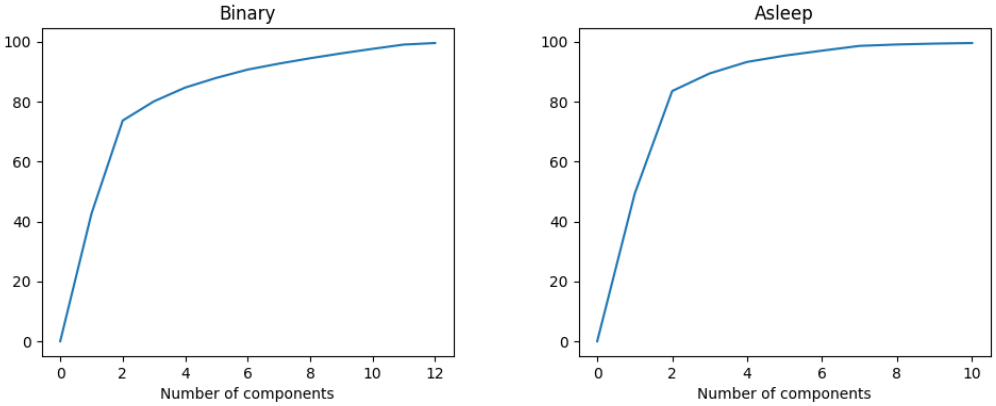


Figure 12: Cumulative explained variance of the components determined by the PCA (%)

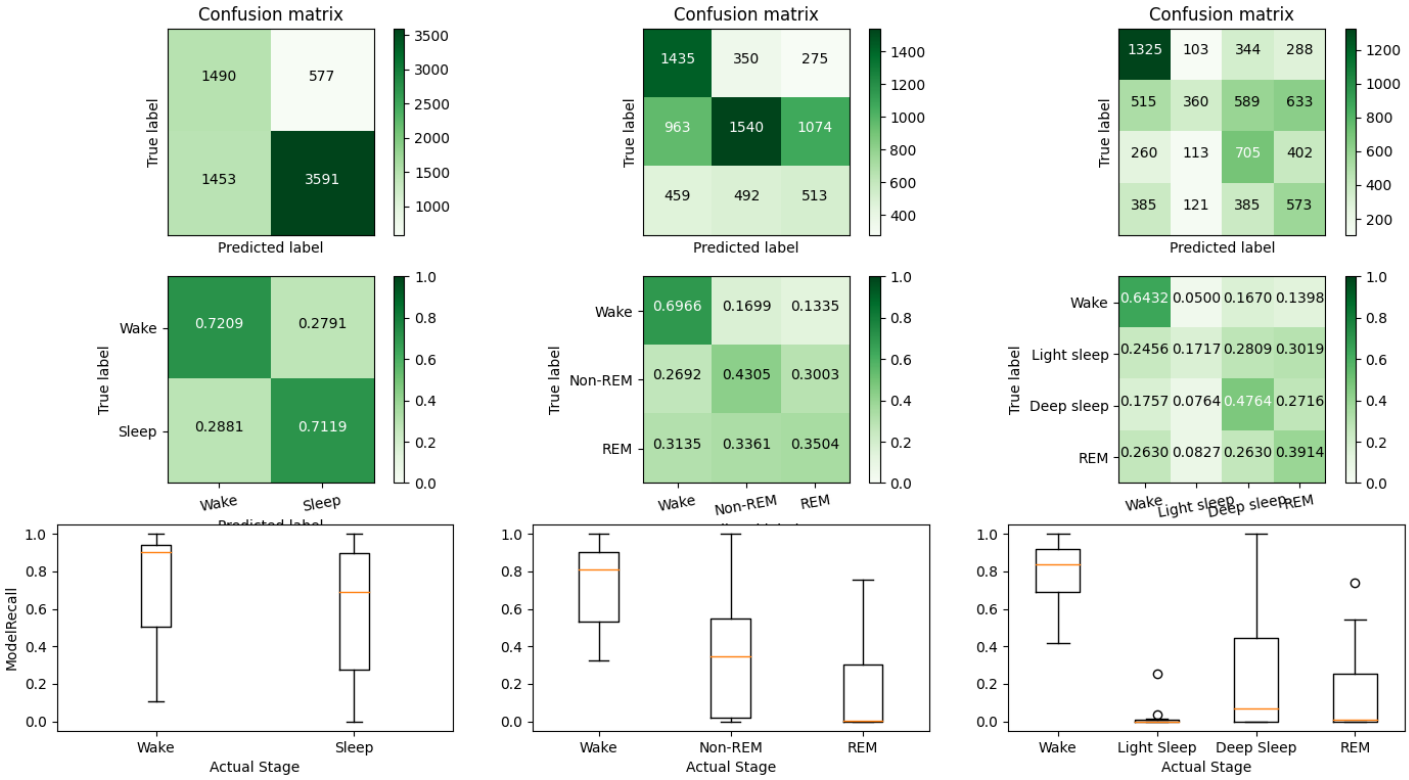
4 Results

This section will present the obtained results for the one-level classifier first, then for the two-level classifier. For each of them, the most interesting results will be displayed as well as the corresponding learning structure and their meta-parameters. The results displayed consist in :

- The confusion matrix for each level of classification
- The confusion normalized according to the true labels (each row sums up to 1)
- The boxplots of the recall across the folds of the cross-validation, for each label

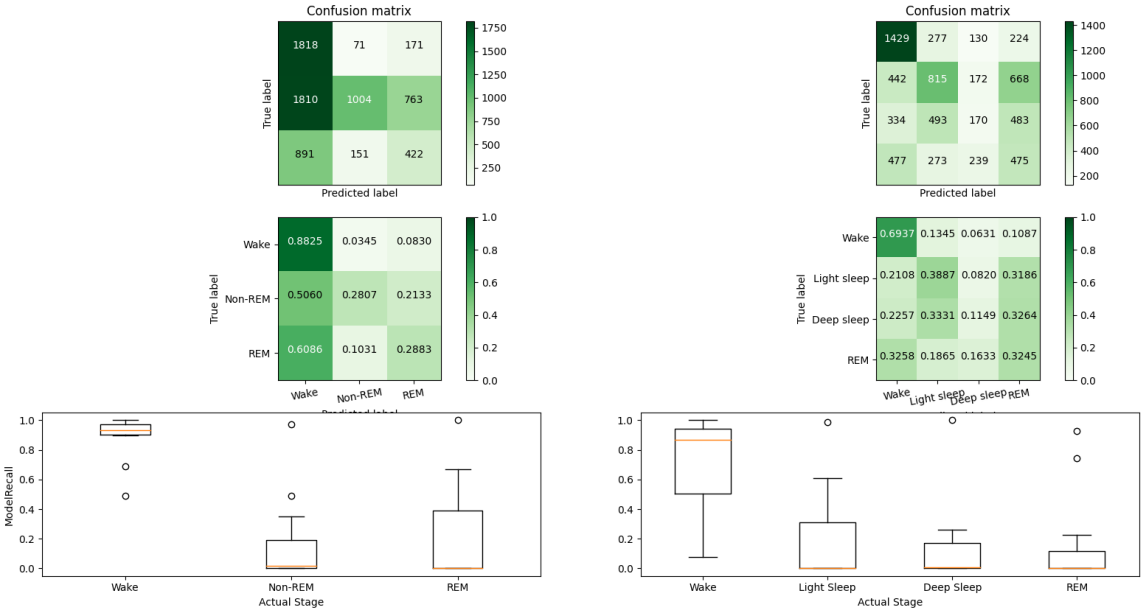
4.1 One-level classifier

Classifier	Structure	Meta-parameter	Value (2-level)	Value (3-level)	Value (4-level)
C	Bagging Classifier	Evaluator	SVM	SVM	SVM
		C	0.05	0.01	0.01
		n_features in subsample	5	5	5



4.2 Two-level classifier

Classifier	Structure	Meta-parameter	Value (3-level)	Value (4-level)
C1	Bagging Classifier	Evaluator	SVM	SVM
		C	0.05	0.05
		n_features in subsample	5	5
C2	SVM	C	0.01	0.01
		class weights	$1/120 \times [40.5, 39, 40.5]$	$120 \times [29, 31, 29, 31]$



5 Discussion

5.1 Comments on the results

5.1.1 One-level classifier

The confusion matrices are quite straightforward to read : in everyone of them, the wake segments are relatively well differentiated from the sleep segments. However, the discrimination between sleep phases is poor. The graphs for the recall are clear : except for wake, the recall is very low on average and varies a lot from a patient to another. As a result, this classifier is not trustworthy to do anything else than discriminate wake segments from the others.

5.1.2 Two-level classifier

As seen on the results, the division into 2 levels allows to discriminate the wake segments even more. It only makes the results for the sleep segments worse, however.

5.2 Generalizability of the results

Generalizability based on our results As seen in the previous section, the results of the classification vary a lot across patients. An explaination could be that the activity of the infant during sleep changes dramatically as he grows. Therefore, it could be considered to build different models by age groups. However, our dataset did not allow to do so because the groups were so small, and the inter-patient variability was too important.

Real use Because we used leave-one-out cross-validation, we can be confident that the prediction made by our model on a new patient whose data have been collected in the same conditions, that is in a PSG environment, will be in the range of what has been reported in Section 4. However, the eventual purpose of the band is to be used at home, meaning that the patient's behaviour could change, and his physiological signals as well. As a result, we anticipate that the classifier may be less performant in its intended use.

5.3 Further improvements

Undetermined label based on regression The results presented in Section 4 are not showing an important metric that could be used in further improvements : for each segment, the probability of it belonging to each class is computed, then the class with the highest probability is selected as the final label for the segment. In that situation, a label selected with probabilities 97%-1%-1%-1% is represented the same way as another label selected upon 28%-24%-24%-24%. It could be useful in the case of further processing to have an index of how confident the classification is.

Temporal context Our work focused on classifying segments, independently from their temporal context. We do not make use of the labels of the preceding segments, which is a consequent loss of information. Indeed, it is clear that a segment that follows 10 minutes of deep sleep is likely to be deep sleep too, and much less likely to be wake. Such information could be very helpful to differentiate between 2 labels that seem equivalently likely for a given segment. More specifically, we could think of a Gated Recurrent Unit like done recently by Feng & al. [9], or use a Hidden Markov Model to perform a correction based on transitions between classes in the training set, as in Ghimatgar's paper ([10]).

6 Conclusion

In this report, we have shown how, based on several raw signals collected with an armband, we have been able to build coherent features that were relevant to the classification of sleep stages. Besides, we have explored several ways to process them and extract models that are relatively working. More globally, in this master thesis, we have shown how we could classify, to a certain extent, the sleep stages in infants using a wearable device as the mean to collect data. We have followed the paper by Boe & al. and adapted it to our data, when possible.
We have made the same observation : discriminating wake from sleep is feasible and relatively reliable when using that method. However, discrimination between sleep stages is yet to be investigated for the results are not satisfying, and some perspectives have been provided.

References

[1] Grigg-Damberger M, Ralls F. Cognitive dysfunction and obstructive sleep apnea: From cradle to tomb. Curr Opin Pulm Med. 2012;18(6):580-587. doi:10.1097/MCP.0b013e328358be18

[2] Gozal D. Sleep-disordered breathing and school performance in children. Pediatrics. 1998;102(3 I):616-620. doi:10.1542/peds.102.3.616

[3] Boe AJ, McGee Koch LL, O'Brien MK, et al. Automating sleep stage classification using wireless, wearable sensors. npj Digit Med. 2019;2(1):1-9. doi:10.1038/s41746-019-0210-1

[4] Chambon S, Galtier MN, Arnal PJ, Wainrib G, Gramfort A. A Deep Learning Architecture for Temporal Sleep Stage Classification Using Multivariate and Multimodal Time Series. IEEE Trans Neural Syst Rehabil Eng. 2018;26(4):758-769. doi:10.1109/TNSRE.2018.2813138

[5] Estévez PA, Held CM, Holzmann CA, et al. Polysomnographic pattern recognition for automated classification of sleep-waking states in infants. Med Biol Eng Comput. 2002;40(1):105-113. doi:10.1007/BF02347703

[6] Isler JR, Thai T, Myers MM, Fifer WP. An automated method for coding sleep states in human infants based on respiratory rate variability. Dev Psychobiol. 2016;58(8):1108-1115. doi:10.1002/dev.21482

[7] Romero D, Jane R. Relationship between Sleep Stages and HRV response in Obstructive Sleep Apnea Patients. Annu Int Conf IEEE Eng Med Biol Soc IEEE Eng Med Biol Soc Annu Int Conf. 2021;2021:5535-5538. doi:10.1109/EMBC46164.2021.9630148

[8] Bue A Lo, Salvaggio A, Insalaco G. Obstructive sleep apnea in developmental age . A narrative review International Classification of Sleep Disorders. Eur J Pediatr. 2020;179(3):357-365.

[9] Feng L-X, Li X, Wang H-Y, et al. Automatic Sleep Staging Algorithm Based on Time Attention Mechanism. Front Hum Neurosci. 2021;15:692054. doi:10.3389/fnhum.2021.692054

[10] Ghimatgar H, Kazemi K, Helfroush MS, et al. Neonatal EEG sleep stage classification based on deep learning and HMM. J Neural Eng. 2020;17(3):36031. doi:10.1088/1741-2552/ab965a

Appendices

Appendix 1 : Complete table of features and transformations

Feature #	Signal	Operation	Transformation
0	Accelerometry (Actigraphy, see X)	Mean	/
1		Min	/
2		Max	/
3		Range	/
4		Inter-Quartile-Range	/
5		Standard Deviation	/
6		Kurtosis	/
7		Variance	/
8	Skin Temperature	Mean	/
9		Min	/
10		Max	/
11		Standard Deviation	log
12	Accelerometer Temperature	Mean	/
13		Min	/
14		Max	
15		Standard Deviation	log
16	Heart Rate Variability	Mean	/
17		Min	/
18		Max	/
19		Standard Deviation	log
20	Respiratory Rate	Mean	/
21		Min	/
22		Max	/
23		Standard Deviation	log
24	Pulse Rate	Mean	/
25		Min	/
26		Max	/
27		Standard Deviation	log
28	SpO2	Mean	/
29		Min	/
30		Max	/
31		Standard Deviation	log
32	Age	/	/

Appendix 2 : Programme used to convert annotation file into readable csv

```
import numpy as np
import glob
import pandas as pd

path = 'C:/Users/tompr/Documents/TFE/WAGSC1/psg/H7RN/*.rml'
files = glob.glob(path)
#print('N   of files : ',len(files))
for j,file in enumerate(files):
    # Data extraction
    opened_file = open(file)
    text = opened_file.read()
    subtext = text[text.index("<UserStaging>"):text.index("</UserStaging>")].splitlines()
    subtext = subtext[2:-7]
    opened_file.close()

    stage_array = []

    def switch(argument):
        switcher = {
            'NotScored': 0,
            'IndeterminateSleep': 0,
            'Stage1': 1,
            'NonREM1': 1,
            'Stage2': 2,
            'NonREM2': 2,
            'Stage3': 3,
            'NonREM3': 3,
            'REM': 4,
            'ActiveSleep': 4,
            'Wake': 5,
            'QuietSleep': 6
        }
        return switcher.get(argument, "Invalid Stage")

    for i in range(len(subtext)):
        stage = subtext[i][subtext[i].find("Type=")+6:subtext[i].find("Start")-2]
        stage_code = switch(stage)
        stage_time = int(subtext[i][subtext[i].find("Start=")+7:subtext[i].find(" /")-1])
        stage_array.append([stage_code,stage_time])
    stage_array = np.array(stage_array)

    # Writing
    csv_name = file[:file.find("/H")+5]+file[file.find("/H"):file.find("/H")+5]+'_Human.csv'
    #print(csv_name)
    print('File ',j,' ok!','( '+file+')')
    pd.DataFrame(stage_array).to_csv(csv_name)
```

Appendix 3 : Programme used to load data

```
import glob
import pandas as pd

def load(path):

    files = glob.glob(path)
    li = []
    f_list = []
    for i,f in enumerate(files):
        # read in csv
        temp_df = pd.read_csv(f)
        f_list.append(f)
        # append df to list
        li.append(temp_df)
        #print(f'Successfully created dataframe for {f} with shape {temp_df.shape}')
    print('Number of subjects : '+str(len(li)))
    return li, f_list

def load_target(path_csv, path_rml):

    files_csv = glob.glob(path_csv)
    li = []
    f_list = []
    for i,f in enumerate(files_csv):
        # read in csv
        temp_df = pd.read_csv(f)
        f_list.append(f)
        # append df to list
        li.append(temp_df)
        #print(f'Successfully created dataframe for {f} with shape {temp_df.shape}')
    print('Number of subjects : '+str(len(li)))

    files_rml = glob.glob(path_rml)
    start_times = []
    durations = []
    for j,file in enumerate(files_rml):
        opened_file = open(file)
        text = opened_file.read()
        subtext_start = text[text.index("<RecordingStart>"):text.index("</RecordingStart>")]
        subtext_start = subtext_start[16:]
        subtext_duration = text[text.index("<Duration>"):text.index("</Duration>")]
        subtext_duration = int(subtext_duration[10:])
        opened_file.close()
        start_times.append(subtext_start)
        durations.append(subtext_duration)

    return li, f_list, start_times, durations

def load_xlsx(path_xlsx):
    return pd.read_excel(path_xlsx, engine = 'openpyxl')
```

Appendix 4 : Programme used to crop and interpolate the signals, with additional utility functions

```
"""
Imports
"""
from my_load import load, load_target
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from datetime import datetime

def ts_to_float(timestamps):
    try:
        dt=datetime.strptime(timestamps,"%Y-%m-%d %H:%M:%S.%f"+" +0200 CEST")
    except ValueError:
        dt=datetime.strptime(timestamps, "%Y-%m-%d %H:%M:%S"+" +0200 CEST")
    dt = datetime.timestamp(dt)
    return dt

def ts_to_float_array(array):
    y = np.zeros_like(array)
    for i in range(len(array)):
        y[i] = ts_to_float(array[i])
    return y

def ts_to_float_target(start_times):
    dt = []
    for i in range(len(start_times)):
        dt.append(datetime.strptime(start_times[i],"%Y-%m-%dT%H:%M:%S"))
    dt = np.array(list(map(datetime.timestamp,dt)))
    return dt

def max_start_times(Daccs,Dphys,Dtemp,Dresults,target_timestamps,nsubjects=12):
    y = np.zeros(nsubjects)
    for i in range(nsubjects):
        y[i] = np.max((ts_to_float(Daccs[i]['Timestamp'].values[0]),
                       ts_to_float(Dphys[i]['Timestamp'].values[0]),
                       ts_to_float(Dtemp[i]['Timestamp'].values[0]),
                       ts_to_float(Dresults[i]['Timestamp'].values[0]),
                       target_timestamps[i][0]))
    return y

def min_end_times(Daccs,Dphys,Dtemp,Dresults,target_timestamps,nsubjects=12):
    y = np.zeros(nsubjects)
    for i in range(nsubjects):
        y[i] = np.min((ts_to_float(Daccs[i]['Timestamp'].values[-1]),
                       ts_to_float(Dphys[i]['Timestamp'].values[-1]),
                       ts_to_float(Dtemp[i]['Timestamp'].values[-1]),
                       ts_to_float(Dresults[i]['Timestamp'].values[-1]),
                       target_timestamps[i][-1]))
    return y

def make_target_and_timestamps(start_times,end_times,durations,Dtarget):
    timestamps = []
    target = []
    for i in range(len(start_times)):
        timestamps.append(np.arange(start_times[i],end_times[i]))
        t = []
        a = np.vstack((Dtarget[i].values,[0,0,durations[i]]))
        for j in range(1,len(Dtarget[i])+1):
            t.append([a[j-1][1]]*(a[j][2]-a[j-1][2]))
        t = [item for sublist in t for item in sublist]
        target.append(np.array(t))
    return timestamps,target

def make_patient(target):
    patient = []
    for i in range(len(target)):
        p = pd.DataFrame(i*np.ones_like(target[i]), columns=['Patient'])
        patient.append(p)
    return patient

def discard_off_limits(Daccs,Dphys,Dtemp,Dresults,target_timestamps,target,nsubjects=12):
    new_Daccs = []
    new_Dphys = []
    new_Dtemp = []
```

```
    new_Dresults = []
    new_target = []
    new_target_timestamps = []
    for i in range(nsubjects):
        start = max_start_times(Daccs,Dphys,Dtemp,Dresults,target_timestamps)[i]
        end = min_end_times(Daccs,Dphys,Dtemp,Dresults,target_timestamps)[i]
        new_Daccs.append(Daccs[i][(ts_to_float_array(Daccs[i]['Timestamp'])>=start) & (ts_to_float_array(Daccs[i]['Timestamp'])<=end)])
        new_Dphys.append(Dphys[i][(ts_to_float_array(Dphys[i]['Timestamp'])>=start) & (ts_to_float_array(Dphys[i]['Timestamp'])<=end)])
        new_Dtemp.append(Dtemp[i][(ts_to_float_array(Dtemp[i]['Timestamp'])>=start) & (ts_to_float_array(Dtemp[i]['Timestamp'])<=end)])
        new_Dresults.append(Dresults[i][(ts_to_float_array(Dresults[i]['Timestamp'])>=start) & (ts_to_float_array(Dresults[i]['Timestamp'])<=end)])
        new_target.append(target[i][(target_timestamps[i]>=start) & (target_timestamps[i]<=end)])
        new_target_timestamps.append(target_timestamps[i][(target_timestamps[i]>=start) & (target_timestamps[i]<=end)])
    return new_Daccs,new_Dphys,new_Dtemp,new_Dresults,new_target,new_target_timestamps

def interpolate_signals(Daccs,Dphys,Dtemp,Dresults,target,target_timestamps,nsubjects=12):
    new_Dphys = []
    new_Dtemp = []
    new_Dresults = []
    new_target = []
    for i in range(nsubjects):
        x = tuple(ts_to_float_array(Daccs[i]['Timestamp'].values))

        xp_phys = tuple(ts_to_float_array(Dphys[i]['Timestamp'].values))
        new_pulse_rate = pd.Series(np.interp(x, xp_phys, tuple(Dphys[i]['Pulse_rate_(BPM)'].values)),name='Pulse_rate_(BPM)')
        new_spo2 = pd.Series(np.interp(x, xp_phys, tuple(Dphys[i]['SpO2_(%)'].values)),name='SpO2_(%)')
        new_sqi_ibi = pd.Series(np.interp(x, xp_phys, tuple(Dphys[i]['SQI_IBI_(%)'].values)),name='SQI_IBI_(%)')
        new_sqi_g = pd.Series(np.interp(x, xp_phys, tuple(Dphys[i]['SQI_G_(%)'].values)),name='SQI_G_(%)')
        new_hrv = pd.Series(np.interp(x, xp_phys, tuple(Dphys[i]['HRV_(ms)'].values)),name='HRV_(ms)')
        new_rr = pd.Series(np.interp(x, xp_phys, tuple(Dphys[i]['Respiratory_rate_(BrPM)'].values)),name='Respiratory_rate_(BrPM)')
        new_sqi_r = pd.Series(np.interp(x, xp_phys, tuple(Dphys[i]['SQI_R_(%)'].values)),name='SQI_R_(%)')
        new_sqi_i = pd.Series(np.interp(x, xp_phys, tuple(Dphys[i]['SQI_I_(%)'].values)),name='SQI_I_(%)')
        phys = pd.DataFrame(np.array((x,new_pulse_rate,new_spo2,new_sqi_ibi,new_sqi_g, new_hrv, new_rr, new_sqi_r, new_sqi_i)).T)
        new_Dphys.append(phys)

        xp_temp = tuple(ts_to_float_array(Dtemp[i]['Timestamp'].values))
        new_skin_temp = pd.Series(np.interp(x, xp_temp, tuple(Dtemp[i]['Skin_temperature_( C )'].values)),name='Skin_temperature_( C )')
        new_accs_temp = pd.Series(np.interp(x, xp_temp, tuple(Dtemp[i]['ACCS_temperature_( C )'].values)),name='ACCS_temperature_( C )')
        temp = pd.DataFrame(np.array((x,new_skin_temp, new_accs_temp)).T)
        new_Dtemp.append(temp)

        xp_results = tuple(ts_to_float_array(Dresults[i]['Timestamp'].values))
        new_pulse_rate_r = pd.Series(np.interp(x, xp_results, tuple(Dresults[i]['Pulse_rate_(BPM)'].values)),name='Pulse_rate_(BPM)_r')
        new_spo2_r = pd.Series(np.interp(x, xp_results, tuple(Dresults[i]['SpO2_(%)'].values)),name='SpO2_(%)_r')
        new_sqi_ibi_r = pd.Series(np.interp(x, xp_results, tuple(Dresults[i]['SQI_IBI_(%)'].values)),name='SQI_IBI_(%)_r')
        new_sqi_g_r = pd.Series(np.interp(x, xp_results, tuple(Dresults[i]['SQI_G_(%)'].values)),name='SQI_G_(%)_r')
        new_hrv_r = pd.Series(np.interp(x, xp_results, tuple(Dresults[i]['HRV_(ms)'].values)),name='HRV_(ms)_r')
        new_rr_r = pd.Series(np.interp(x, xp_results, tuple(Dresults[i]['Respiratory_rate_(BrPM)'].values)),name='Respiratory_rate_(BrPM)_r')
        new_sqi_r_r = pd.Series(np.interp(x, xp_results, tuple(Dresults[i]['SQI_R_(%)'].values)),name='SQI_R_(%)_r')
        new_sqi_i_r = pd.Series(np.interp(x, xp_results, tuple(Dresults[i]['SQI_I_(%)'].values)),name='SQI_I_(%)_r')
        results = pd.DataFrame(np.array((x,new_pulse_rate_r,new_spo2_r,new_sqi_ibi_r,new_sqi_g_r, new_hrv_r, new_rr_r, new_sqi_r_r, new_sqi_i_r)).T)
        new_Dresults.append(results)

        xp_target = tuple(target_timestamps[i])
        new_t = pd.Series(np.interp(x, xp_target, tuple(target[i])),name='Target')
```

```
130         new_t_df = pd.DataFrame(new_t.T)
131         new_target.append(new_t_df)
132 
133     return Daccs,new_Dphys,new_Dtemp,new_Dresults,new_target
```

Appendix 5 : Programme used to segment the data

```
def segment(frame,segment_size):
    segments = []
    for i in range(len(frame)):
        segments_from_subject = []
        for j in range(int(len(frame[i])/segment_size)):
            segments_from_subject.append(frame[i][segment_size*j:segment_size*(j+1)])
        print('Data for subject '+str(i+1)+'/'+str(len(frame))+' segmented')
        segments.append(segments_from_subject)
    flattened_segments = [item for sublist in segments for item in sublist]
    return segments, flattened_segments
```

Appendix 6 : Main programme

```
1
2  # """
3  # Imports
4  # """
5  # from my_load import load, load_target, load_xlsx
6  # from sklearn.neighbors import KNeighborsClassifier
7  # from segments import segment
8  # import numpy as np
9  # import matplotlib.pyplot as plt
10 # import pandas as pd
11 # from sklearn.decomposition import PCA, FastICA
12 from sklearn.preprocessing import StandardScaler, minmax_scale, QuantileTransformer
13 # from scipy import stats
14 # from sklearn.model_selection import KFold, StratifiedGroupKFold, GroupKFold, StratifiedKFold
15 # from sklearn.tree import DecisionTreeClassifier
16 from sklearn.metrics import confusion_matrix, accuracy_score, balanced_accuracy_score,recall_score
17 # from sklearn.svm import SVC
18 # from sklearn.linear_model import LogisticRegressionCV
19 # from sklearn.ensemble import RandomForestClassifier, BaggingClassifier
20 # from sklearn.neural_network import MLPClassifier
21 # from sklearn.impute import SimpleImputer
22 # import scipy.signal
23 # import time
24 # from rect_timestamps import ts_to_float, ts_to_float_array, ts_to_float_target, max_start_times,
       min_end_times, discard_off_limits, interpolate_signals, make_target_and_timestamps,
       make_patient
25 # import random
26 # from sklearn.utils import resample
27 # from sklearn.feature_selection import mutual_info_classif, SequentialFeatureSelector, chi2,
       SelectKBest
28
29 # # """
30 # # CSV data load, and segmentation in 30s segments
31 # # """
32 # individual_scaling = False
33 # total_scaling = True
34
35 n_classes = 3
36 # folds = 3
37
38 # data_path_accs = 'C:/Users/tompr/Documents/TFE/WAGSC1/csv/H*/*raw.bin.accs.csv'
39 # data_path_phys = 'C:/Users/tompr/Documents/TFE/WAGSC1/csv/H*/plx.bin.csv'
40 # data_path_temp = 'C:/Users/tompr/Documents/TFE/WAGSC1/csv/H*/temperature.bin.csv'
41 # data_path_results = 'C:/Users/tompr/Documents/TFE/WAGSC1/csv/H*/results_Thursday.csv'
42 # data_path = 'C:/Users/tompr/Documents/TFE/WAGSC1/patients_data.xlsx'
43 # target_path_csv = 'C:/Users/tompr/Documents/TFE/WAGSC1/psg/H*/H*_Human.csv'
44 # target_path_rml = 'C:/Users/tompr/Documents/TFE/WAGSC1/psg/H*/*.rml'
45
46
47 # Ddata = load_xlsx(data_path)
48 # Valid = np.array(Ddata['Valid'])
49 # Age = np.array(Ddata['AgeInMonth'])[Valid==1]
50 # Gender = np.array(Ddata['Gender'])[Valid==1]
51 # Gender = np.array(Gender=='MALE')
52 # Pulse_rate_alarm = np.array(Ddata['Pulse_rate_alarm'])[Valid==1]
53 # RR_alarm = np.array(Ddata['RR_alarm'])[Valid==1]
54
55 # Daccs,_ = load(data_path_accs)    # loads the data in the chosen path(s) as a pd dataframe
56 # Dphys,_ = load(data_path_phys)
57 # Dtemp,_ = load(data_path_temp)
58 # Dresults,_ = load(data_path_results)
59 # Dtarget,f_list,start_times,durations = load_target(target_path_csv, target_path_rml)
60 # start_times = ts_to_float_target(start_times)
61 # end_times = start_times + durations
62 # target_timestamps, target = make_target_and_timestamps(start_times,end_times,durations,Dtarget)
63
64 # Daccs, Dphys, Dtemp, Dresults, target, target_timestamps = discard_off_limits(Daccs,Dphys,Dtemp,
       Dresults,target_timestamps,target,nsubjects=12)
65 # print('Data cropped')
66
67 # Daccs, Dphys, Dtemp, Dresults, target = interpolate_signals(Daccs,Dphys,Dtemp,Dresults,target,
       target_timestamps,nsubjects=12)
68 # print('Data interpolated')
69
70 # for i in range(len(Dphys)):
71 #     Dphys[i].columns = ['Timestamps','Pulse_rate_(BPM)','SpO2_(%)','SQI_IBI_(%)','SQI_G_(%)','
       HRV_(ms)','Respiratory_rate_(BrPM)','SQI_R_(%)','SQI_I_(%)']
```

```
#     Dtemp[i].columns = ['Timestamps','Skin_temperature_( C )','ACCS_temperature_( C )']
#     Dresults[i].columns = ['Timestamps','Pulse_rate_(BPM)_r','SpO2_(%)_r','SQI_IBI_(%)_r','SQI_G_(%)_r','HRV_(ms)_r','Respiratory_rate_(BrPM)_r','SQI_R_(%)_r','SQI_I_(%)_r']

# patient = make_patient(target)

# target_all = pd.DataFrame(pd.concat((pd.DataFrame(i,columns=['Target']) for i in target),axis=0, ignore_index=True))
# accs_all = pd.DataFrame(pd.concat((i for i in Daccs),axis=0, ignore_index=True))
# phys_all = pd.DataFrame(pd.concat((i for i in Dphys),axis=0, ignore_index=True))
# temp_all = pd.DataFrame(pd.concat((i for i in Dtemp),axis=0, ignore_index=True))
# results_all = pd.DataFrame(pd.concat((i for i in Dresults),axis=0, ignore_index=True))
# patient_all = pd.DataFrame(pd.concat((i for i in patient),axis=0, ignore_index=True))

# list_all = [accs_all,phys_all,temp_all,results_all,patient_all,target_all]
# D = pd.concat(list_all,axis=1)



# Valid_IBI = D['SQI_IBI_(%)_r']>50
# Valid_SpO2 = 0.3*D['SQI_G_(%)_r']+0.4*D['SQI_G_(%)_r']+0.3*D['SQI_I_(%)_r']>20
# Valid_RR = D['SQI_G_(%)_r']>20


# segment_size = 300
# segments, flattened_segments = segment([D],segment_size) # segments the data in sub-df of segment_size timesteps each

# """
# Feature building
# """

# def atan(x,n):
#     y = x
#     for i in range(n):
#         #y = (1 - np.exp(-x)) / (1 + np.exp(-x))
#         y = np.arctan(x)
#         x = y
#     return y

# def log(x):
#     return np.log(x)

# def find_outliers(x):
#     m = np.mean(x)
#     s = np.std(x)
#     y = np.argwhere(np.abs(x-m)>10*s)
#     return y

# def remove_outliers(x):
#     outliers_indices = find_outliers(x)
#     y = np.copy(x)
#     y[outliers_indices] = np.nan
#     return y

# def find_outliers_matrix(x):
#     y = []
#     for i in range(np.shape(x)[1]):
#         outliers_indices = list(np.ravel(find_outliers(x[:,i])))
#         for item in outliers_indices:
#             y.append(item)
#     return np.array(list(set(y)))

# def remove_outliers_matrix(x,target,patient):
#     outliers_indices = find_outliers_matrix(x)
#     y = np.delete(x,outliers_indices, axis=0)
#     t = np.delete(target,outliers_indices, axis=0)
#     p = np.delete(patient,outliers_indices, axis=0)
#     return y,t,p

# def myfft(x, fs):
#     freqs = np.fft.fftshift(np.fft.fftfreq(np.shape(x)[0],1/fs))
#     fx = np.abs(np.real(np.fft.fftshift(np.fft.fft(x))))
#     return freqs, fx

# n_features = 37
# features = np.zeros((len(flattened_segments),n_features))
# count = 0
# target = np.zeros(len(flattened_segments))
```

```
# patient = np.zeros(len(flattened_segments))

# for j in range(len(flattened_segments)):
#     flattened_segments[j]['AG'] = (flattened_segments[j]['X_(G)']**2+flattened_segments[j]['Y_(G)']**2+flattened_segments[j]['Z_(G)']**2)**0.5-1
#     AG = flattened_segments[j]['AG']

#     features[j,0] = np.mean(AG)
#     features[j,1] = np.min(AG)
#     features[j,2] = np.max(AG)
#     features[j,3] = np.ptp(AG)
#     features[j,4] = stats.iqr(AG)
#     features[j,5] = np.std(AG)
#     features[j,6] = stats.kurtosis(AG)
#     features[j,7] = np.var(AG)

#     ST = flattened_segments[j]['Skin_temperature_( C )']
#     features[j,8] = np.mean(ST)
#     features[j,9] = np.min(ST)
#     features[j,10] = np.max(ST)
#     features[j,11] = np.std(ST)

#     AT = flattened_segments[j]['ACCS_temperature_( C )']
#     features[j,12] = np.mean(AT)
#     features[j,13] = np.min(AT)
#     features[j,14] = np.max(AT)
#     features[j,15] = np.std(AT)

#     HRR = flattened_segments[j]['HRV_(ms)_r'][Valid_RR]
#     features[j,16] = np.mean(HRR)
#     features[j,17] = np.min(HRR)
#     features[j,18] = np.max(HRR)
#     features[j,19] = np.std(HRR)

#     RR = flattened_segments[j]['Respiratory_rate_(BrPM)_r'][Valid_RR]
#     features[j,20] = np.mean(RR)
#     features[j,21] = np.min(RR)
#     features[j,22] = np.max(RR)
#     features[j,23] = np.std(RR)

#     PRR = flattened_segments[j]['Pulse_rate_(BPM)_r'][Valid_IBI]
#     features[j,24] = np.mean(PRR)
#     features[j,25] = np.min(PRR)
#     features[j,26] = np.max(PRR)
#     features[j,27] = np.std(PRR)

#     PP = 1/PRR
#     features[j,28] = np.mean(PP)
#     features[j,29] = np.min(PP)
#     features[j,30] = np.max(PP)
#     features[j,31] = np.std(PP)

#     SP = flattened_segments[j]['SpO2_(%)_r'][Valid_SpO2]
#     features[j,32] = np.mean(SP)
#     features[j,33] = np.min(SP)
#     features[j,34] = np.max(SP)
#     features[j,35] = np.std(SP)


#     target[j] = int(np.mean(flattened_segments[j]['Target']))
#     patient[j] = int(np.mean(flattened_segments[j]['Patient']))

# for i in range(int(patient[-1])+1):
#       features[patient==i,36] = Age[i]

# feature_names = np.array(['Actigraphy : Mean',
# 'Actigraphy : Min',
# 'Actigraphy : Max',
# 'Actigraphy : PTP',
# 'Actigraphy : IQR',
# 'Actigraphy : StD',
# 'Actigraphy : Kurtosis',
# 'Actigraphy : Variance',
# 'Skin Temp. : Mean', 'Skin Temp. : Min', 'Skin Temp. : Max', 'Skin Temp. : StD',
# 'Accs Temp. : Mean', 'Accs Temp. : Min', 'Accs Temp. : Max', 'Accs Temp. : StD',
# 'Heart Rate Variability : Mean', 'Heart Rate Variability : Min', 'Heart Rate Variability : Max', 'Heart Rate Variability : StD',
# 'Respiratory Rate : Mean', 'Respiratory Rate :  Min', 'Respiratory Rate : Max', 'Respiratory Rate : StD',
```

```
# 'Pulse Rate : Mean', 'Pulse Rate :  Min', 'Pulse Rate : Max','Pulse Rate : StD',
# 'Pulse Period : Mean', 'Pulse Period :  Min', 'Pulse Period : Max','Pulse Period : StD',
# 'SpO2 : Mean', 'SpO2 :  Min', 'SpO2 : Max','SpO2 : StD',
# 'Age'])

# features = np.delete(features,np.where((target==0)|(target==6)),axis=0)
# patient = np.delete(patient,np.where((target==0)|(target==6)),axis=0)
# target = np.delete(target,np.where((target==0)|(target==6)),axis=0)


# target = np.delete(target,np.nonzero(np.isnan(features))[0],axis=0)
# patient = np.delete(patient,np.nonzero(np.isnan(features))[0],axis=0)
# features = np.delete(features,np.nonzero(np.isnan(features))[0],axis=0)


# features,target,patient = remove_outliers_matrix(features,target,patient)


# for i in [11,15,19,23,27,31]:
#     features[:,i] = log(features[:,i]+0.1)


# """
# Preprocessing (scaling, feature selection, PCA)
# """

# # deleting NotScored/Undeterminate segments

# def binary(target):
#     # 0: wake, 1: asleep
#     target = np.where(target == 2, 1, target)
#     target = np.where(target == 3, 1, target)
#     target = np.where(target == 4, 1, target)
#     target = np.where(target == 5, 0, target)
#     return target

def asleep(target, n_classes):
    if n_classes == 4:
        # 0: wake, 1: light sleep (NREM1&2), 2: deep sleep (NREM3), 3: REM
        target = np.where(target == 2, 1, target)
        target = np.where(target == 3, 2, target)
        target = np.where(target == 4, 3, target)
        target = np.where(target == 5, 0, target)
        return target
    elif n_classes == 3:
        # 0: wake, 1: NREM, 2:REM
        target = np.where(target == 2, 1, target)
        target = np.where(target == 3, 1, target)
        target = np.where(target == 4, 2, target)
        target = np.where(target == 5, 0, target)
        return target
    elif n_classes == 2:
        return binary(target)


usable_features = features[(patient!=8)]
usable_target = target[(patient!=8)]
usable_patient = patient[(patient!=8)]

def countOccurrence(a):
    k = {}
    for j in a:
      if j in k:
        k[j] +=1
      else:
        k[j] =1
    return k

# def upsample_classes(features, patient, target):
#     features = np.hstack((features,patient.reshape((len(target),1)) ,target.reshape((len(target)
    ,1))))
#     resampled_features = []
#     for i in (set(patient)):
#         resampled_features_patient = []
#         for j in (set(target)):
#             M = np.max(list(countOccurrence(asleep(target[patient==i],4)).values()))
#             if len(features[(patient==i)&(target==j)])>0:
#                 if (j==1)|(j==2):
#                     A = resample(features[(patient==i)&(target==j)],n_samples = int(M*len(
```

```
    features[(patient==i)&(target==j)])/(len(features[(patient==i)&((target==1)|(target==2))])))))
#                     resampled_features_patient.append(A)
#                 else:
#                     A = resample(features[(patient==i)&(target==j)],n_samples = M)
#                     resampled_features_patient.append(A)
#         resampled_features_patient = [item for sublist in resampled_features_patient for item in sublist]
#         resampled_features.append(resampled_features_patient)
#     resampled_features = np.array([item for sublist in resampled_features for item in sublist])
#     resampled_target = resampled_features[:,-1]
#     resampled_patient = resampled_features[:,-2]
#     resampled_features = resampled_features[:,:-2]
#     return resampled_features, resampled_patient, resampled_target

def upsample_classes(features, patient, target):
    features = np.hstack((features,patient.reshape((len(target),1)) ,target.reshape((len(target),1))))
    resampled_features = []
    M = max(countOccurrence(patient).values())
    for i in (set(patient)):
        A = resample(features[(patient==i)],n_samples = M)
        resampled_features.append(A)
    resampled_features = np.array([item for sublist in resampled_features for item in sublist])
    resampled_target = resampled_features[:,-1]
    resampled_patient = resampled_features[:,-2]
    resampled_features = resampled_features[:,:-2]
    return resampled_features, resampled_patient, resampled_target


# resampled_features, resampled_patient, resampled_target = upsample_classes(usable_features, usable_patient, usable_target)
# resampled_features, resampled_patient, resampled_target = upsample_classes(features, patient, target)



# Class = [np.copy(usable_features), np.copy(usable_target), np.copy(usable_patient)]
Class = [np.copy(features), np.copy(target), np.copy(patient)]
# Class = [np.copy(resampled_features), np.copy(resampled_target), np.copy(resampled_patient)]
Classes = [Class]

for Class in Classes:
    Class_features, Class_target, Class_patient = Class
    if individual_scaling:
        for i in set(Class_patient):
            p_features = Class_features[Class_patient==i][:,:-1]
            scaled_p_features = StandardScaler().fit_transform(p_features)
            Class_features[Class_patient==i][:,:-1] = scaled_p_features
    if total_scaling:
        Class_features = minmax_scale(Class_features)
    scaled_features = Class_features

    scaled_features_all = scaled_features[:,chi2(scaled_features,asleep(Class_target,4))[1]<0.05]

    pca = PCA(14)
    pca.fit(scaled_features_all)
    scaled_features_all = pca.transform(scaled_features_all)

    scaled_features_bin = scaled_features
    scaled_features_bin = scaled_features_bin[:,chi2(scaled_features,asleep(Class_target,2))[1]<0.05]

    pca_bin = PCA(12)
    pca_bin.fit(scaled_features_bin)
    scaled_features_bin = pca_bin.transform(scaled_features_bin)


    scaled_features_asleep = scaled_features[binary(Class_target)==1]
    target_asleep = Class_target[binary(Class_target)==1]
    scaled_features_asleep = scaled_features_asleep[:,chi2(scaled_features_asleep,asleep(target_asleep,4))[1]<0.05]

    pca_asleep = PCA(10)
    pca_asleep.fit(scaled_features_asleep)
    scaled_features_asleep = pca_asleep.transform(scaled_features_asleep)

    labels = [['Wake','Sleep'],['Wake','Non-REM','REM'],['Wake','Light sleep','Deep sleep','REM']]

    """
```

```
    Learning
    """
    def plot_confusion_matrix(cm,
                              target_names,
                              n_classes,
                              title='Confusion matrix',
                              cmap=None,
                              normalize=True,
                              new_fig=True):
        import itertools

        accuracy = np.trace(cm) / np.sum(cm).astype('float')
        misclass = 1 - accuracy

        if cmap is None:
            cmap = plt.get_cmap('Blues')

        if new_fig:
            plt.figure(figsize=(8, 8))
        plt.imshow(cm, interpolation='nearest', cmap=cmap)
        plt.title(title)
        plt.colorbar()

        target_names = target_names[n_classes-2]
        if target_names is not None:
            tick_marks = np.arange(len(target_names))
            plt.xticks(tick_marks, target_names, rotation=45)
            plt.yticks(tick_marks, target_names)

        if normalize:
            cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]


        thresh = cm.max() / 1.5 if normalize else cm.max() / 2
        for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
            if normalize:
                plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                         horizontalalignment="center",
                         color="white" if cm[i, j] > thresh else "black")
            else:
                plt.text(j, i, "{:0.0f}".format(cm[i, j]),
                         horizontalalignment="center",
                         color="white" if cm[i, j] > thresh else "black")

        if normalize:
            plt.clim(0,1)
        # plt.tight_layout()
        plt.ylabel('True label')
        plt.xlabel('Predicted label')
        plt.show()

    def model_train(training_data, training_target, classes, model):
        if model == 'svm1':
            # c = SVC(class_weight = 'balanced', C = 0.005, gamma = 'auto')
            # c = SVC(class_weight = 'balanced')
            class_weight = len(training_data)/(2*np.bincount(training_target.astype(int)))*((2/100)
    *np.array([45,55]))
            d = dict(enumerate(class_weight.flatten()))
            c = SVC(gamma = 'auto', C = 0.05, class_weight = d)
        elif model == 'svm2':
            if classes == 3:
                class_weight = len(training_data)/(3*np.bincount(training_target.astype(int)))
    [1:]*((3/120)*np.array([40.5,39,40.5]))
                d = dict(enumerate(class_weight.flatten(), 1))
                c = SVC(class_weight = d, C = 0.01, gamma = 'scale')
            else:
                # class_weight = len(training_data)/(2*np.bincount(training_target.astype(int)))
    [1:]*((2/80)*np.array([40,40]))
                # d = dict(enumerate(class_weight.flatten(), 1))
                # c = SVC(class_weight = d, C = 0.01, gamma = 'scale')
                c = SVC(class_weight = 'balanced',C = 0.05)
        elif model == 'svm3':
            class_weight = len(training_data)/(4*np.bincount(training_target.astype(int)))
    [:4]*((4/120)*np.array([29,31,29,31]))
            d = dict(enumerate(class_weight.flatten(), 0))
            c = SVC(class_weight = d, C = 0.01, gamma = 'scale')
            # c = SVC(class_weight = 'balanced',C = 0.01)
        elif model == 'svm4':
            class_weight = len(training_data)/(4*np.bincount(training_target.astype(int)))
```

```
    [:4]*((4/120)*np.array([30,30,30,30]))
            d = dict(enumerate(class_weight.flatten(), 0))
            c = SVC(class_weight = d, C=0.01, gamma = 'auto')
            # c = SVC(probability = True, class_weight = d, C = 0.005)
        elif model == 'dt':
            c = DecisionTreeClassifier(class_weight = 'balanced', criterion='entropy')
        elif model == 'log':
            c = LogisticRegressionCV(max_iter = 10000, solver = 'sag', Cs = 10)
        elif model == 'bag':
            c = BaggingClassifier(SVC(class_weight = 'balanced', C=0.1), max_features=5)
            # c = BaggingClassifier(DecisionTreeClassifier(class_weight = 'balanced'))
        elif model == 'mlp':
            c = MLPClassifier(max_iter = 1000, hidden_layer_sizes = (20,100,100,100,100,20),
    verbose = False, learning_rate_init = 0.1, learning_rate='adaptive', solver='sgd',
    early_stopping=True, alpha = 0.001)
        else:
            # class_weight = len(training_data)/(2*np.bincount(training_target.astype(int)))
    [:2]*((2/120)*np.array([100,20]))
            # d = dict(enumerate(class_weight.flatten(), 0))
            c = RandomForestClassifier(class_weight='balanced_subsample', bootstrap=(True),
    max_depth=8, max_samples = None, n_estimators = 100)
            # c = RandomForestClassifier(bootstrap=(False), max_depth=(None))
        c.fit(training_data, np.ravel(training_target))
        return c

    def my_kfold_double_classifier(scaled_features_bin, scaled_features_asleep, target, patient,
    n_classes, k):
        recall=[]
        total_cm = np.zeros((n_classes,n_classes))
        # kfold = StratifiedGroupKFold(k, shuffle = False)
        # for trn_idx, val_idx in kfold.split(scaled_features_bin, target, patient):
        # kfold = KFold(k, shuffle = False)
        # for trn_idx, val_idx in kfold.split(scaled_features_bin):
        kfold = GroupKFold(k)
        for trn_idx, val_idx in kfold.split(scaled_features_bin, groups=patient):
            print("")
            print('trn : ',np.shape(trn_idx),set(patient[trn_idx]))
            print('val :', np.shape(val_idx),set(patient[val_idx]))

            training_data_binary = scaled_features_bin[trn_idx]
            training_target_asleep = asleep(target,n_classes)[trn_idx]
            training_target_binary = binary(target)[trn_idx]
            print('target trn:', countOccurrence(target[trn_idx]))
            print('target test:', countOccurrence(target[val_idx]))
            c_bin = model_train(training_data_binary, training_target_binary, 2, 'bag')

            train_data_asleep = training_data_binary[training_target_binary==1]
            train_target_asleep = training_target_asleep[training_target_binary==1]

            if n_classes>2:
                c_asleep = model_train(train_data_asleep, train_target_asleep, n_classes-1, 'bag')

            validation_data_bin = scaled_features_bin[val_idx]
            prediction_bin = c_bin.predict(validation_data_bin)
            validation_target_bin = binary(target)[val_idx]
            bin_score =  balanced_accuracy_score(validation_target_bin, prediction_bin, adjusted=(
    True))
            validation_target = validation_target_bin
            print('binary :', bin_score)
            if n_classes>2:
                validation_data_asleep = validation_data_bin[prediction_bin==1]
                if len(validation_data_asleep!=0):
                    prediction_asleep = c_asleep.predict(validation_data_asleep)
                    prediction_bin[np.array(np.where(prediction_bin==1))[0]] = prediction_asleep
                validation_target = asleep(target, n_classes)[val_idx]
                total_score = balanced_accuracy_score(validation_target, prediction_bin, adjusted=(
    True))
                print('final :', total_score)
            recall.append(recall_score(validation_target, prediction_bin, average =None))
            cm = np.array(confusion_matrix(validation_target, prediction_bin))
            # cm = np.array([cm[i]/np.sum(cm[i]) for i in range(n_classes)])
            plot_confusion_matrix(cm, labels, n_classes, cmap='Greens', normalize=False)
            total_cm += cm
        return total_cm, validation_target, prediction_bin, recall

    def my_kfold(scaled_features_all, target, patient, n_classes, k, confidence=False):
        scores = []
        recall = []
        total_cm = np.zeros((n_classes,n_classes))
```

```
        if confidence:
            total_cm = np.zeros((n_classes,n_classes+1))
        # kfold = StratifiedGroupKFold(k, shuffle = True)
        # for trn_idx, val_idx in kfold.split(scaled_features_all, patient, patient):
        # kfold = StratifiedGroupKFold(k, shuffle = False)
        # for trn_idx, val_idx in kfold.split(scaled_features_all, target, patient):
        # kfold = KFold(k, shuffle = True)
        # for trn_idx, val_idx in kfold.split(scaled_features_all):
        kfold = GroupKFold(k)
        for trn_idx, val_idx in kfold.split(scaled_features_all, groups=patient):
            print("")
            print('trn : ',np.shape(trn_idx),set(patient[trn_idx]))
            print('val :', np.shape(val_idx),set(patient[val_idx]))

            training_data = scaled_features_all[trn_idx]
            training_target = asleep(target,n_classes)[trn_idx]
            print('target trn:', countOccurrence(target[trn_idx]))
            print('target test:', countOccurrence(target[val_idx]))
            c = model_train(training_data, training_target, n_classes, 'bag')

            validation_data = scaled_features_all[val_idx]
            prediction = c.predict(validation_data)
            if confidence:
                df = c.decision_function(validation_data)
                ordered = np.flip(np.sort(df),axis=1)
                threshold = 1.03
                confident_idx = (ordered[:,0]-ordered[:,1])<threshold
                prediction[confident_idx] = 4
            validation_target = asleep(target,n_classes)[val_idx]
            score = balanced_accuracy_score(validation_target, prediction, adjusted=(True))
            recall.append(recall_score(validation_target, prediction, average =None))
            print('total :', score)
            scores.append(score)
            cm = np.array(confusion_matrix(validation_target, prediction))
            # cm = np.array([cm[i]/np.sum(cm[i]) for i in range(n_classes)])
            plot_confusion_matrix(cm, labels, n_classes, cmap='Greens', normalize=False)
            total_cm += cm
        return total_cm, validation_target, prediction, scores, recall


    total_cm,  validation_target, prediction, scores, recall = my_kfold(scaled_features_all,
    Class_target, Class_patient, n_classes, len(set(Class_patient)), confidence=False)
    # total_cm,  validation_target, prediction, scores, recall = my_kfold(scaled_features_all,
    Class_target, Class_patient, n_classes, 3, confidence=False)
    # total_cm, validation_target, prediction_bin, recall = my_kfold_double_classifier(
    scaled_features_bin, scaled_features_asleep, Class_target, Class_patient, n_classes, len(set(
    Class_patient)))
    # total_cm, validation_target, prediction_bin, recall = my_kfold_double_classifier(
    scaled_features_bin, scaled_features_asleep, Class_target, Class_patient, n_classes, 2)

    # def f(scaled_features_all, target_all):
    #     test_idx = random.choices(np.arange(len(scaled_features_all)))
    #     test_set = scaled_features_all[test_idx]
    #     trn_idx = np.arange(len(scaled_features_all))!=test_idx
    #     train_set = scaled_features_all[trn_idx]

    #     target_test = asleep(target_all,4)[test_idx]
    #     target_train = asleep(target_all,4)[trn_idx]
    #     c = model_train(train_set, target_train, 4, 'svm4')
    #     df = c.decision_function(train_set)
    #     ordered = np.flip(np.sort(df),axis=1)
    #     threshold = 1.05
    #     confident = np.sum((ordered[:,0]-ordered[:,1])>threshold)
    #     total = len(trn_idx)
    #     return confident, total


    # print(f(scaled_features_all, Class_target))
    # '''
    # print('(score, specificity, sensitivity) : ',total_score, total_specificity,
    total_sensitivity)
    # print('Used features :', best_comp)
    # '''
    norm_cm = np.array([total_cm[i]/np.sum(total_cm[i]) for i in range(n_classes)])




    # """
```

```
    # Plots
    # """

    plot_confusion_matrix(total_cm, labels, n_classes, cmap='Greens', normalize = False)
    plot_confusion_matrix(norm_cm, labels, n_classes, cmap='Greens', normalize = True)

# # colors = ['black','silver','lightcoral','red','saddlebrown','darkorange','yellow','chartreuse','green','turquoise','deepskyblue','blue','darkviolet']
# # p = Class_0[2].astype(np.int32)
# # c = [colors[int(x)] for x in p]

# # feature_names = ['Actigraphy : Mean', 'Actigraphy : Mean (sigmoid)',
# # 'Actigraphy : Min', 'Actigraphy : Min (sigmoid)',
# # 'Actigraphy : Max', 'Actigraphy : Max (sigmoid)',
# # 'Actigraphy : PTP', 'Actigraphy : PTP (sigmoid)',
# # 'Actigraphy : IQR', 'Actigraphy : IQR (sigmoid)',
# # 'Actigraphy : StD', 'Actigraphy : StD (sigmoid)',
# # 'Actigraphy : Kurtosis', 'Actigraphy : Kurtosis (sigmoid)',
# # 'Actigraphy : Variance', 'Actigraphy : Variance (sigmoid)',
# # 'Skin Temp. : Mean', 'Skin Temp. : Min', 'Skin Temp. : Max', 'Skin Temp. : StD',
# # 'Accs Temp. : Mean', 'Accs Temp. : Min', 'Accs Temp. : Max', 'Accs Temp. : StD',
# # 'Heart Rate Variability : Mean', 'Heart Rate Variability : Min', 'Heart Rate Variability : Max', 'Heart Rate Variability : StD',
# # 'Respiratory Rate : Mean', 'Respiratory Rate :  Min', 'Respiratory Rate : Max', 'Respiratory Rate : StD',
# # 'Pulse Rate : Mean', 'Pulse Rate :  Min', 'Pulse Rate : Max',' Pulse Rate : StD', 'Age']

# # for i in range(len(features)):
# #     # plt.subplot(7,7,i+1)
# #     plt.figure()
# #     plt.title(feature_names[i], fontsize = 7)
# #     plt.hist(features[:,i],bins=100)
# #     plt.tick_params(axis='both', labelsize=7)
# #     # plt.boxplot(features[:,i])

# # points = []
# # for C in [0.001,0.005,0.01,0.05,0.1,0.5,1,5,10,15,20,25,50,100,500,1000]:
# #     total_cm,  validation_target, prediction, scores = my_kfold(scaled_features_all, Class_target, Class_patient, n_classes, 3, confidence=False)
# #     points.append(scores)
# # plt.plot(np.mean(points,axis=1))

# points = []
# for C in [5,15,25]:
#     total_cm,  validation_target, prediction, recall = my_kfold_double_classifier(scaled_features_bin, scaled_features_asleep, Class_target, Class_patient, n_classes, len(set(Class_patient)))
#     points.append(recall)
# plt.plot(np.mean(points,axis=1))
```

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**