Automated Sleep Stage Scoring with Deep Learning

Thuc Dinh
thucd2@illinois.edu
University of Illinois at Urbana-Champaign

ABSTRACT

In this paper, I construct two deep neural networks to score the sleep stage automatically. Both models are working on single-channel signals from Sleep Cassette study dataset in publicly available Sleep-EDF 2018 dataset. The first model comprises multiple convolutional neural networks (CNN) and bidirectional long short-term memory (Bi-LSTM) networks achieving overall accuracy of **75.4%**, an average macro F1-score of **67.8%** across all folds in 20-fold cross-validation and inter-rater reliability coefficient Cohen's kappa κ = **0.66**. The second model is based on convolutional neural networks predicting with an average accuracy of **79%** and average macro F1-score of **75%** and κ = **0.70** in the same cross-validation procedure**.** The result shows that my model is comparable to state-of-the-art methods with hand-engineered features.

Keywords

Sleep stage scoring, Convolutional neural network (CNN), Recurrent Neural Network (RNN), Bidirectional Long Short-term Memory (Bi-LSTM), Electroencephalography (EEG), Deep Learning

1. INTRODUCTION

Sleep plays an important role in human health. It enables the body to recharge, helps retain memory and leaves individual refreshed once wakes up. However, more than 50 million Americans already suffer from different sleep disorders and at least 25 million Americans suffer from sleep apnea [1]. It raises the need to diagnose sleep disorders and tracking the response to the treatment. One such way is monitoring sleep stages [2].

Overnight polysomnography (PSG), including brain monitoring using electroencephalography (EEG), is a central component of the diagnostics evaluation for sleep disorders. However, PSG is often performed by trained physicians which are often costly, time-consuming and limited in number. Recently, the advance of deep neural network learning algorithms along with publicly large datasets in healthcare has made the tasks of automated sleep scoring possible.

This paper aims to construct deep learning algorithms for automated sleep stage scoring using single-channel EEGs and try to achieve comparable performance with state-of-the-art methods using hand-engineered features.

2. LITERATURE SURVEY

The problem of annotating sleep stage can be formulated as followed [3]: The input data are multi-channels PSG signals segmented into 30-second epochs in the form of multivariate time series data, denoted as $X = \{X_1, X2, ..., X_N\}$, where X_i is 30 seconds long and contain physiological signals recorded. Each epoch X_i has a sleep stage label $y_i \in \{Wake, REM, N1, N2, N3\}$. Our task is to predict the sequence of sleep stages $S = \{s_1, s_2, ..., s_N\}$ based on X as close as possible to $Y = \{y_1, y_2, ..., y_N\}$.

In [3], Orestis et al. present a CNN architecture using only a single channel (Fpz-Cz) of EEG to solve the problem. Their CNN model comprises of two pairs of convolutional, and pooling layers, a single stacking layer placed between these two pairs, two fully-connected layers, and a 5-class soft-max layer. They achieve high mean F1-score 81%, the overall accuracy of 74% over all subjects.

Another work by Sajad. et al. [4] composed of deep CNN to extract time-invariant features, frequency information, and employ sequence to sequence model to capture the complex and long short-term context dependencies between sleep epochs and scores. The evaluation results demonstrate that the proposed method achieved the best annotation performance compared to current literature, with an overall accuracy of 84.26%, a macro F1-score of 79.66%, and inter-rater reliability coefficient Cohen's kappa κ = 0.79.

Focusing on interpretable of the model while trying to maintain accuracy as the black-box neural network models, a work by I. Al-Hussaini et al. [5] combining deep learning models (by using CNN) with expert-defined rules to generate simple interpretable models. The final models provide an intuitive understanding of the classifications. The SLEEPER method obtain overall accuracy in range of 77%-78.5%, about 85% ROC-AUC and κ = 0.7

3. METHOD

3.1 Data

In this study, I use the Physionet Sleep-EDF dataset [6] containing 197 whole-night PolySommoGraphic sleep recordings. The data comes from two studies: Sleep Cassette (153 recordings) and Sleep Telemetry (44 recordings).

Each recording has file format *PSG.edf containing EEG, electrooculography (EOG), chin electromyography (EMG) and event markers. EEG, EOG, EMG signals were sampled at 100 Hz, the epoch duration is 30 seconds and the event marker at 1 Hz.

The dataset also provides annotations of the sleep patterns manually scored by well-trained technicians according to the Rechtschaffen and Kales manual. All hypnograms are in the file format *Hypnogram.edf, containing annotated of the sleep patterns that correspond to the PSGs. These patterns are W (wake), R (rapid-eye movement), N1, N2, N3, N4, M (movement time) and ? (not scored). Two classes N3 and N4 are merged into one cne class named N3. I also exclude M (movement) and ? (not scored) stages to have 5 sleep stages.

In this paper, I have worked with Sleep Cassette study containing 152 recordings (why 152 instead of 153 will be explained in Data Processing section), an equivalent of 1622 hours of data. Each epoch is annotated with a sleep stage by well-trained technicians. The distribution of each class is as below:

Table 1: Number of 30-s epoch for each sleep stage

Dataset\ Class	W	N1	N2	N3-N4	REM	Total
Sleep-EDF-18 Sleep Cassette	65,795	21,469	68,633	12,991	25,767	194,655

3.2 Data Preprocessing

Both recording and hypnogram are in edf, edf+ format, respectively. I employ script provided by DeepSleepNet [7] to prepare dataset.

3.2.1 *Extract Fpz-Cz, Pz-Oz channels*

The algorithm works with each channel separately, so that I extract each channel from the recording for the ease of using data later.

Each channel of each recording will then be stored in a single npz [8] format that can be loaded with numpy later. Therefore, there are 153 files total for each channel. Each file contains three information: channel signal, annotated label and sample rate. Due to some incompatibility between using libraries, there is an exception during processing 1 file, so in this study at this time, I work with 152 extracted files.

3.2.2 *Oversampling*

From class distribution as in [Table 1], the dataset is clearly imbalanced as W, N2 are the two largest classes and take up 68% sample of dataset. That may cause the model overfitting to the class that's represented more in our dataset if I train with the current dataset. Therefore, to overcome this, I used oversampling technique. Specifically, I duplicate the minority sleep stages in the original training set such that all sleep stages have the same number of samples.

3.2.3 *Transform five consecutive 30-s input to 150-s input*

In this study, one method I try to reproduce is Convolutional Neural Network as described in SleepEEG [4]. In that work, they follow the fact that the scoring of a particular epoch guided by AASM sleep scoring manual, can depend on the characteristics of the preceding or succeeding epochs so that they choose input data to be the signal of the current epoch to be classified together with the signals of the preceding two and succeeding two epochs. Therefore, I generate another training and validation dataset such that each new epoch consisting of five consecutive original epoch samples. I also try to train and compare models between the newly created dataset and the original dataset.

3.3 Model 1 - Convolutional neural network architecture and Bidirectional Long Short-term memory

The first model is inspired by DeepSleepNet [8] consisting of two parts. The first part is representation learning aiming to extract time-invariant features from each raw single-channel 30-s EEG sample. The main responsibility of the second part - sequence residual learning, is to encode temporal information such as stage transition rules from sequence of EEGs samples in the extracted features.

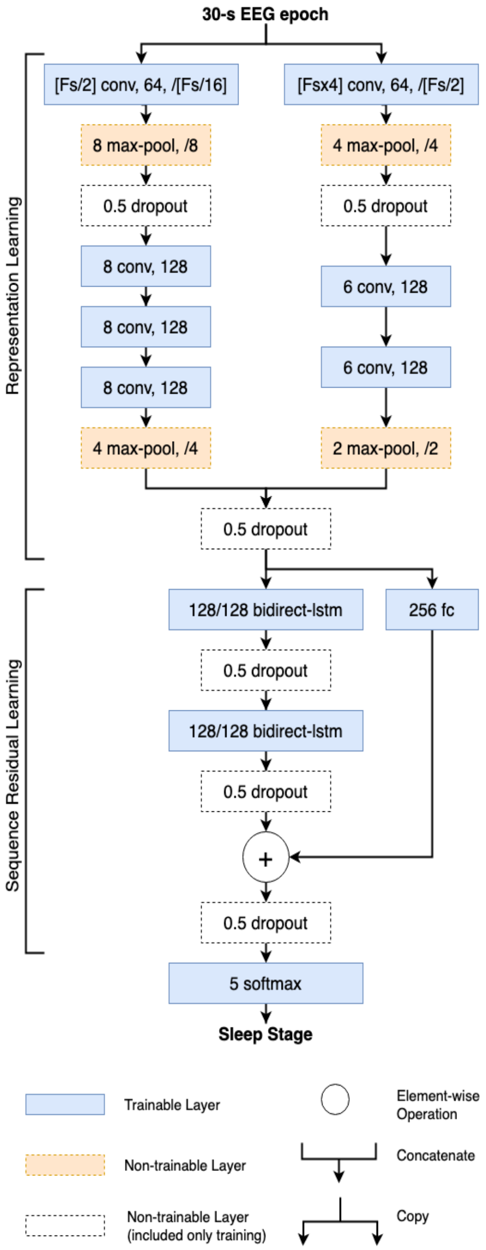

Figure 1: An overview architecture of the first model consisting of two parts: representation learning and sequence residual learning

3.3.1 *Representation learning*

According to signal processing experts [10], in analyzing neural time series data, one should control the trade-off between temporal and frequency precision in feature extraction algorithms. Therefore, I use two parallel CNNs with small and large filter sizes to extract these two different features from each raw single-channel 30-s EEG sample. The small filter can capture temporal information such as a certain EEG pattern, while the larger filter is good at understanding frequency information.

Each one of two CNNs generally consists of convolutional layers, two max-pooling layers, and a dropout layer. The specifications for the network can be seen in [Figure 1]. Each *conv* block shows a filter size, the number of filters, and stride size. *Fs* denotes sample rate, which is 100Hz in Sleep-EDF dataset. All convolutional layers are activated by the rectified linear unit (ReLU). Pooling layer down-samples the previous layers using *max* operation, and are demonstrated in *max-pool* block, featuring a pooling size and a stride size. Dropout layer is employed to prevent over-fitting during training and is removed during inference. Specifically, throughout this model, I use dropout layers with a probability of 0.5, meaning that during training input values have 50% chance of being set to 0.

Output of these two CNNs with small and large filter size are concatenated before being forwarded to the sequence residual learning.

3.3.2 Sequence Residual learning

This part consists of two main components: two bidirectional-LSTMs and a shortcut connection.

Two bidirectional-LSTMs are incorporated to learn temporal information such as stage transition rules which mimic the process sleep experts used to determine the next possible sleep stages as described in American Academy of Sleep Medicine (AASM) standard. Each bidirectional-LSTM having two LSTMs process backward and forward data independently so that the model can use information from next or previous samples to predict score of current sample.

Another component is a shortcut connection. It help the final model can make prediction using both features extract from previous representation learning and Bi-LSTMs. Specifically, I use a fully connected layer to transform extracted features from CNNs to a vector of same size with output of Bi-LSTMs. This fully connectied layer is also activated with ReLU.

Output of two components are then concatenated and go through dropout layer before a final 5-units fully connected layers activated with *softmax* to calculate the probability of sample belonging to each class.

3.3.3 Two-phase learning

The model is trained using two-phase learning to tackle the problem of imbalanced classes while also maintaining the order of samples as sequential input for training the sequence residual learning part. Both two phases are quantified with cross-entropy loss to measure the agreement of predicted and target sleep stages.

In the first phase, the representation learning part is appended with a 5-units fully connected layer activated with softmax before training. After this training phase was done, the newly added layer is then discarded. The original input is oversampled as described in [3.2.2] to make all classes have the same number of samples. The model is then trained with 60% of the class-balance training set using Adam optimizer with learning rate *lr1=0.001* in maximum *30 epochs*. The training can stop early before reaching maximum epochs if validation loss on 40% of data doesn't decrease in *2* consecutive epochs.

After the first phase is done, all learning parameters in two CNNs are frozen during the second training phase. The model is then training with original sequential input (not the over-sampled one) with the first 60% of dataset and and is validated against remaining 40% of dataset. The running time for each epoch is large, approximately 10 minutes for each epoch in my hardware setup - see section [6. Environment Specification], so the second phase is trained in only 3 epochs to make it feasible to run in 20-fold cross-validation later.

3.4 Model 2 - Convolutional neural network architecture

The second network architecture I try is CNN from SleepEEG [4] with some modifications to make it work in Keras and Tensorflow.

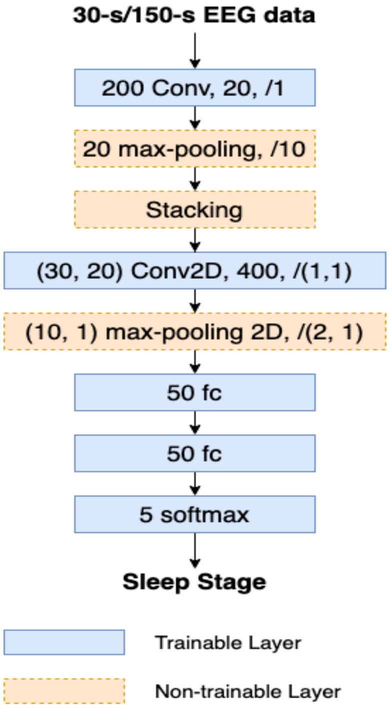

Figure 2: An overview of model 2 using CNNs

The CNN architecture as shown in [Figure 2] and detailed as in [Table 2], comprises two pairs of convolutional and pooling layers (C1-P1 and C2-P2), three fully connected layers (F1, F2, F3). Between layer P1 an C2, there is a reshape layer to technically make the output of P1 layer to be ready for C2 layer. But the meaning behind that is allow layer C2 to capture the relationship across 20 filtered signals produced by C1-P1, across a specific time window. I also note that, filter size of C1 is quite large (which is 200 for each filter, twice the sample rate) to help capture frequency information (i.e frequency components). This is inspired by the way signal processing experts control the trade-off between temporal and frequency precision in their feature extract algorithms [10]. Overall, this network demonstrates the idea of combining different frequency-specific activities over time and using that information to predict the class for each sample.

The cost function for the training is Sparse Categorical Cross-entropy with L2-regularization. Each convolutional layer (C1, C2) and two dense layers (F1, F2) are applied rectified linear unit

(ReLU) non-linearity. The last dense layer F3 is activated with softmax function to produce probability of each class among 5 classes. Each of the two dense layers F1, F2 have 50 units only (instead of 500 units as in original paper SleepEEG) to help us get faster as well as somehow evaluate the method performance.

Table 2: CNN architecture

Layer	**Type**	**# Unit**	**Acti-vation**	**Size**	**Stride**	**Output Shape**
Input						(B, 15000, 1)
C1	Conv1D	20	ReLU	200	1	(B, 14801, 20)
P1	MaxPool1D			20	10	(B, 1479, 20)
R	Reshape					(B, 1479, 20, 1)
C2	Conv2D	400	ReLU	(30,20)	(1,1)	(B, 1450, 1, 400)
P2	Max-Pool2D			(10,1)	(2,1)	(B, 721, 1, 400)
F	Flatten					(B, 288400)
F1	Dense	50	ReLU			(B, 50)
F2	Dense	50	ReLU			(B, 50)
F3	Dense	5	Soft-max			(B, 5)

**B: Batch size*

I train this CNN network for two different inputs. The first input is the original signal where each sample has 30.000 data-points (30 seconds at 100 Hz sampling rate). The second input is the transformed signal where each newly created epoch also consists of two preceding and two succeedings of original epochs as described in [2.2.3]. Each sample in the second input contains 15.000 data-points (150 seconds at 100 Hz sampling rate). The output shape column in Table 2, is demonstrated only for the second input, but its architecture remains the same for both cases.

4. RESULT & ANALYSIS

4.1 Evaluation & Metrics

To evaluate the generalizability of the algorithms, I obtained results using 20-fold cross-validation. Specifically, all recordings are sorted in increasing order of patient id and night id which are encoded in filename. After that, I split the dataset into 20 equal parts - to be exact, there are 8 parts each contain 7 recordings, and the remaining 12 parts contain 8 recordings each. For each fold, one part is used for testing while 19 other parts for training (60%) and validation (40%).

I evaluate the performance of different models using mean of macro F1-score (MF1), mean of overall accuracy (ACC) across all folds and Cohen's Kappa coefficient (κ). The MF1 and ACC for each fold are calculated as follows:

$$ACC = \frac{\sum_{c=1}^{C} TP_c}{N}$$

$$MF1 = \frac{\sum_{c=1}^{C} F1_c}{C}$$

where TP_C is the true positive of class c, $F1_C$ is per-class F1-score of class c, C is the number of sleep stages, and N is the total number of test samples. Cohen's Kappa coefficient is used to measure inter-rater reliability. In this situation it quantifies the agreement between the model's prediction and the actual value scored by human.

4.2 Model 1: CNNs + Bi-LSTMS

The running time for each epoch in first phase of learning is about 25 seconds/epoch, while that number for second phase is about 5 minutes/epoch. That makes the total time for each fold approximate 35 minutes, and it take more than 10 hours to finish all 20 folds on my current setup.

In term of performance, model 1 achieves average accuracy of **75.4%**, average macro F1-score of **67.8%** and Cohen's Kappa **κ = 0.66**. This kappa coefficient value falls in range of 0.61 - 0.80 suggest substantial agreement between model result and human according to Natioanl Institutes of Health [13]. Figure 3 visualizes accuracy for each fold during benchmarking procedure.

Below is confusion matrix after 20 folds. It is constructed as sum of all confusion matrix after each fold, and is normalized to the percentage of an actual class. From this table, we can see that *91%, 39%, 73%, 80%, 68%* of actual *W, N1, N2, N3, REM* classes are correctly identified, respectively. This sensitivity value for class *N1* is the lowest among all classes as it is normally misclassified as *W* (20%), *N2* (22%), or *REM* (19%).

Table 3: Model 1's confusion matrix in the percentage of actual class

		PREDICTED				
	Stage	W	N1	N2	N3	REM
	W	**91**	5	1	0	3
ACTUAL	N1	20	**39**	22	0	19
	N2	1	14	**73**	3	8
	N3	1	1	18	**80**	0
	REM	7	17	8	0	**68**

4.3 Model 2: CNNs

In this architecture, I perform training using different inputs: 30-s EEG data and 150-s EEG data on the first fold to check if they may produce comparable results. After that I perform 20-fold cross validation on the chosen input to check performance of the model.

For the first fold, with 30-s EEG data, on the test set, the accuracy is **80.9%** and macro F1-score is 76.9%, after 10 epochs with a run-time of approximately 1 minute for each epoch. Meanwhile, with 150-s EEG data:

- After 15 epochs, the training accuracy is 76.64%, the accuracy on the test set is 79.7%,

- After 24 epochs: the training accuracy is 78.26%, the accuracy on the test set reaches **79.1%**

However, the running time for each epoch of 150-s EEG data is about 7 minutes - 6 times higher than the one from 30-s EEG data. From both accuracy and running time perspective, the model seems to work better with 30-s EEG data. Therefore, I decide to perform 20-fold cross-validation on 30-s EEG data.

The average overall accuracy of this model is 79%, ranging from 72% at fold 5 to 84% at fold 9. Details of accuracy for each fold of this model can be seen in figure 3. The average macro F1-score is 75% and Cohen's Kappa κ = 0.70**.** The kappa score also suggests a substantial agreement between this model result and sleep stage well-trained graders. We can also see that this model is better than the previous in all terms: average of overall accuracy, average macro F1-score, Cohen's Kappa coefficient, and time.

I also compile a confusion matrix as the same as the previous model and can be seen in table 4. This time, the sensitivity for class N1 gets much better as this value increases greatly from 39% from last model to 71% of the current model. But it still suffers from mis-classification between N1 and N2. Specifically, 12% of samples from class N1 are falsely classified as N2, and 20% of samples from class N2 are misclassified as N1.

Table 4: Model 2's confusion matrix in the percentage of actual class

		PREDICTED				
	Stage	W	N1	N2	N3	REM
ACTUAL	W	**88**	9	0	0	2
	N1	7	**71**	12	0	11
	N2	1	20	**68**	2	9
	N3	0	0	3	**97**	0
	REM	3	22	4	0	**72**

4.4 Comparison with hand-engineered features

In [12], Tsinalis et al. using fine-tuned time-frequency analysis-based method to extract sleep stage-specific signal features. In this, they use complex Morelet wavelets to capture the mixture of frequencies and their interrelations at different points in times. They then use ensemble learning with an ensemble of stacked sparse auto-encoders for classifying the sleep stages.

The comparison between methods using hand-engineered features and raw input can be seen in table 5. There are 2 models are using hand-engineered features in this table, one using and ensemble of stacked sparse auto-encoder, the other one using CNNs. The best model achieves result of average accuracy at 78% and average macro F1-score at 84%. My model produces a similar accuracy score at 79%, but get a lower F1-score than these two models, at 75%.

Table 5: Comparison of my model with other model using hand-engineered features

Features	**Method**	**Average accuracy**	**F1**
Hand-engineered (Morlet wavelets)	Ensemble learning (described in [12])	78%	**84%**
Hand-engineered (Morlet wavelets)	CNN method (describe in [3])	73%	81%
Raw input	CNNs in this paper (model 2)	**79%**	75%

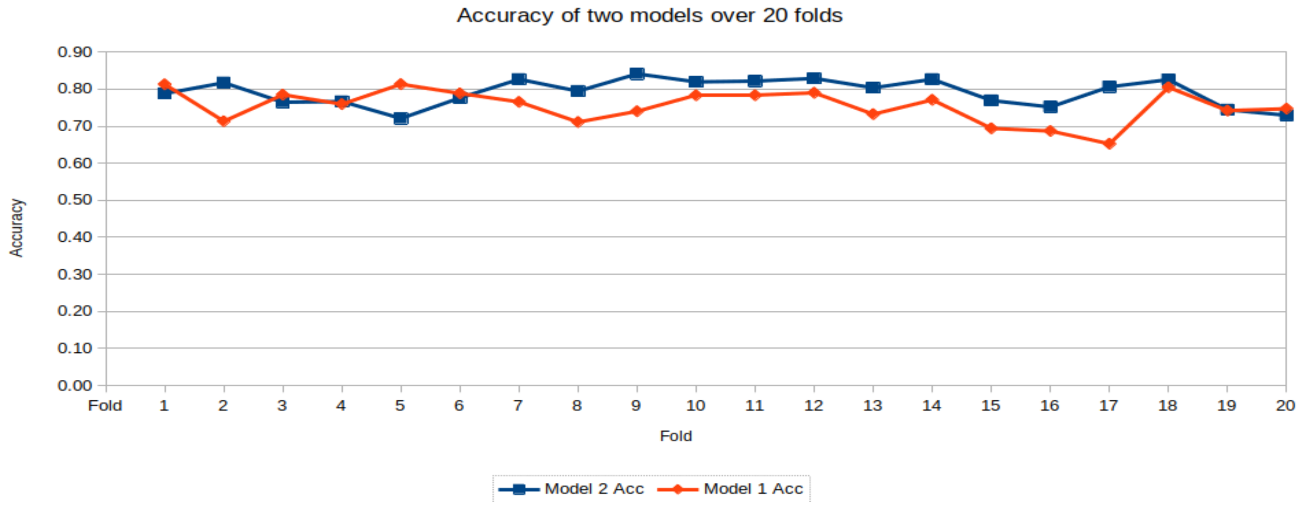

Figure 3: Comparison of accuracy between two models over 20 folds

5. CONCLUSION

In this paper, I have constructed and experimented with two deep learning algorithms to automatically annotate sleep stages.

One model incorporates two CNNs and two Bi-LSTMs into its architecture. By doing this, the model aims to first capture temporal information and frequency information in parallel, then learn temporal information such as stage transition rules to predict sleep stages. This idea is promising, but in my experiment, the result is not good enough as already described in section 4.2. The reason may be due to a change in the number of units in Bi-LSTMs layers. In the original paper of DeepSleepNet, there are 512 units in each Bi-LSTMS. However, it takes about 60 minutes for each epoch with this parameter, making it impracticable to run all 20 folds on my hardware. Therefore, in this model, I trade accuracy for performance as each Bi-LSTMs is now having only 128 units. That's also the reason why the first model's result is not as good as the second model.

Another model is using just CNNs in its network. This network extracts and combines different frequency-specific activities over time to predict the class for each sample. Although simple, the model gives a good result and is comparable with other methods using hand-engineered features.

6. ENVIRONMENT SPECIFICATION

The experiment is conducted on my local computer. Specifically, the hardware's specification: AMD (R) Ryzen 7 3700x 8 cores 16 threads, 32 GB RAM, GPU GeForce GTX 1660 6GB, 1TB SSD.

The software environment is as follows: python 3.7.10, tensorflow 2.4.1, tensorflow-gpu 2.4.1, numpy 1.19.2, pandas 1.2.4, scikit-learn 0.24.1, mne 0.23.0

7. REFERENCES

[1] https://www.sleephealth.org/sleep-health/the-state-of-sleephealth-in-america/.

[2] Carskadon, Mary A and Rechtschaffen, Allan. Monitoring and staging human sleep. Principles and practice of sleep medicine, 2000.

[3] O. Tsinalis, P. M. Matthews, Y. Guo, S. Zafeiriou. Automatic Sleep Stage Scoring with Single-Channel EEG Using Convolutional Networks

[4] S. Mousavi, F. Afghah, U. Rajendra Achrya. SleepEEGNet: Automated Sleep Stage Scoring with Sequence to Sequence Deep Learning Approach

[5] I. Al-Hussaini, C. Xiao, M. B. Westover, and J. Sun. Sleeper: inter-pretable sleep staging via prototypes from expert rules. In Machine Learning for Healthcare Conference, pages 721–739, 2019

[6] Sleep EDFX. https://www.physionet.org/content/sleep-edfx/1.0.0/.

[7] DeepSleepNet github repository. https://github.com/akaraspt/deepsleepnet

[8] Npz file format. https://numpy.org/doc/stable/reference/generated/numpy.savez.html

[9] A. Supratak, H. Dong, C. Wu, Y.Guo. DeepSleepNet: a Model for Automatic Sleep Stage Scoring based on Raw Single-Channel EEG

[10] M. X. Cohen, Analyzing Neural Time Series Data: Theory and Practice. MIT Press. 2014

[11] Anthony J Viera, Joanne M Garrett, et al. Understanding interobserver agreement: the kappa statistic. Fam med, 37(5):360–363, 2005

[12] Tsinalis, O., P. M. Matthews, and Y. Guo. Automatic sleep stage scoring using time-frequency analysis and stacked sparse auto encoders. *Annals of Biomedical Engineering. 2015*

[13] Interrater reliability: the kappa statistic. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3900052/