Detecting Sleep States

Mohammad Arjamand Ali

Nov 28, 2023

Abstract

This study addresses a Kaggle challenge to detect sleep transition events using machine learning. Different problem framing methods and models, including neural networks, XGBoost, and logistic regression, were explored, highlighting the importance of problem formulation and revealing neural networks' superior performance in this context.

Medium Abstract

This research focuses on a Kaggle competition hosted by the Child Mind Institute to detect sleep transition events using machine learning. The paper demonstrates how different approaches to problem framing can significantly impact the effectiveness of machine learning models. Three framing methods were iteratively developed and assessed, with the final approach focusing on predicting sleep status (awake/asleep) to infer transition events. The dataset comprised 277 wrist-worn accelerometer recordings, narrowed down to a subset for computational feasibility. Various machine learning models, including neural networks, XGBoost, and logistic regression, were trained and compared. An extensive hyperparameter search was conducted to obtain the best performing models..

This study underscores the role of problem framing in machine learning, especially for time series data. Initially, direct approaches to detect transition events encountered issues like class imbalance and complex patterns, leading to poor model performance. By reframing the problem to predict sleep status, the study overcame these challenges, achieving better results. Neural networks outperformed other models, indicating their suitability for this type of data. This paper ends with suggestions for future research, such as exploring advanced models and different data methods..

Introduction

The problem is a Kaggle competition hosted by the Child Mind Institute [1]. The goal is to detect transition events regarding sleep, namely onset and wakeup. This competition aims to utilize its findings to further conduct research as it pertains to sleep, with potential at a larger scale. This problem holds potential practical applications in improving sleep quality, diagnosing sleep disorders, and contributing to the broader understanding of human sleep behavior.

In this study, I outline three different versions of framing the problem, each iteratively improving on

the previous. I show how certain approaches to problems make it near impossible for models to learn relationships, while other non-direct approaches can yield significant results.

In the following sections, we begin with a description of the dataset, followed by a review of the relevant background in terms of machine learning. Then I describe in detail my approaches to solving the problem, followed by details of the models I used and their hyperparameters. In the final sections, I interpret and discuss the results and model performance. The paper then concludes with a summary of key findings and potential for future research.

Dataset

The dataset provided in the Kaggle competition consists of 277 wrist-worn accelerometer recordings, annotated for 2 ‘transition’ events: *onset*, beginning of sleep, and *wakeup*, end of sleep. Comprising approximately 127 million data points across these recordings, each data point includes a time step representing 5 seconds, a timestamp in ISO 8601 format, and two specific measurements: anglez and enmo. Anglez measures the angle of the arm compared to the vertical axis, which is indicative of the wearer’s orientation and relative position. Enmo (Euclidean Norm Minus One) represents the intensity of movement across all three axes.

Keeping in mind the size of the dataset and computational constraints, this project used a smaller subset of the dataset [2] which consists of 35 of the original recordings (~16 million data points). This subset was carefully selected to exclude recordings with any missing information. Additionally, this dataset contains a label for whether a person was awake or not, based on the annotated events.

For a visual representation of a typical recording from this dataset, refer to Fig 1 and 2. Fig 1 illustrates a full recording of the anglez and enmo measurements, and is labeled with when the transition events occur. Fig 2 is a zoomed version of Fig 1, showing the measurements over a span of 3 days.

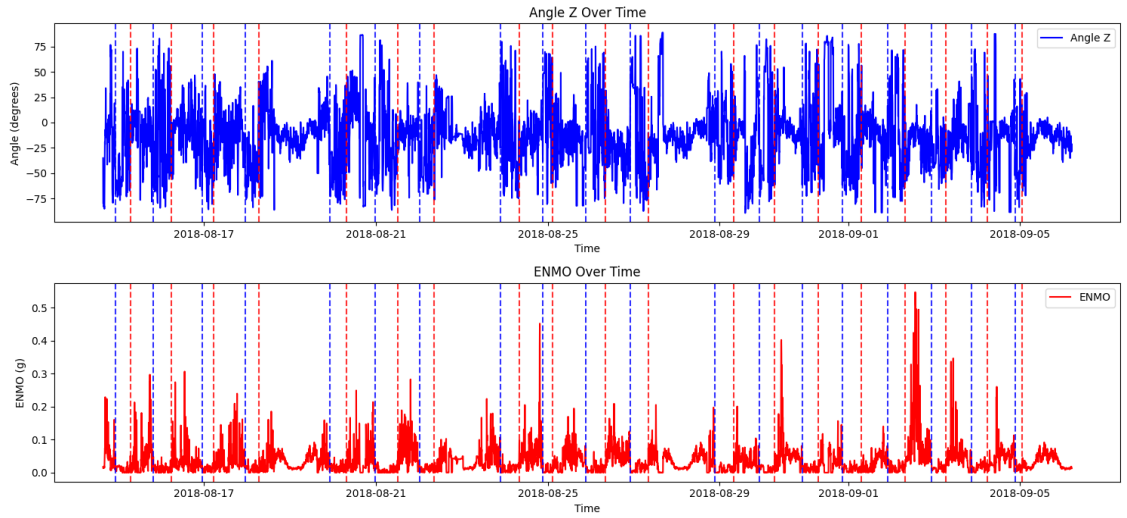


Fig 1. Showcasing the Angle Z and ENMO measurements of a full recording

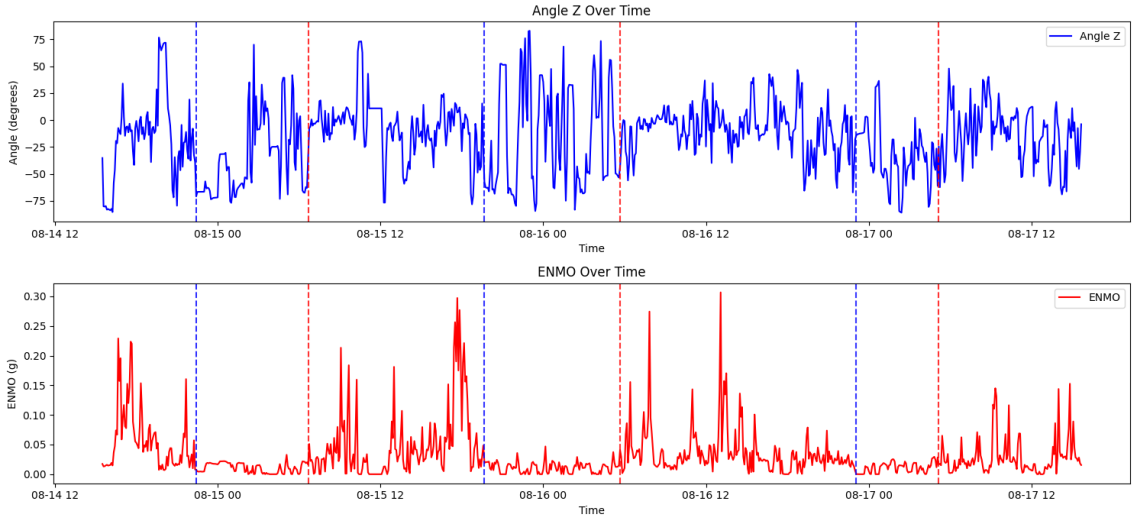


Fig 2. Showcasing a zoomed in, 3 day long version of the recording in Fig 1

Background

Machine Learning Overview

Conventional modeling methods define strict rules for the types of relationships between variables. Linear Regression defines a strictly linear relationship. Dynamical Systems define a recursive relationship based on the previous instance. In contrast, machine learning adopts a more ‘black-box’ approach to modeling, focusing less on predefined functions and instead on *approximating* the relationships from data.

This problem falls under the subset of machine learning called supervised learning. I supervised learning, the goal is to approximate a function that represents the relationship between input variables (X) and outputs (y). This begins with a model initialized with parameters, typically denoted as W, and uses some non-linear functions to process the inputs (X). This results in some predictions of the model, denoted by $\hat{y}$. These are compared against the actual outcomes (y), and a 'loss' (J) is calculated, which reflects the model's accuracy. Based on this loss, the model adjusts its parameters W in a way that its predictions are better next time. This process is performed iteratively for the whole dataset until the model learns a good relationship between the variables. The following equations represent supervised machine learning as a whole:

$$\hat{y} = Model_W(X)$$

$$J = Loss(\hat{y}, y)$$

Machine learning tends to be very powerful because of its ability to model very complex relationships with relative ease. This does come with a tradeoff, however. For machine learning models, especially advanced ones, it can become very difficult to understand how they work and how exactly they lead to their predictions. This often leads to discussions regarding performance vs interpretability for models.

Time Series Datasets/Problems

Time Series data is data that is tracked / measured over a discrete period of time. This consists of a label for time, along with measurements and readings for each point. Examples of time problems are predicting stock prices over time, or predicting temperature readings. Time series data tends to have unique characteristics, like cyclic behavior and trends, which aids in building models, but also adds a lot of complexity to problems.

One of the main challenges with time series data is their large and variant sizes. Typical models in machine learning have fixed input sizes for the problem they try to solve. Time series data is sequential data, so its size usually varies across examples, which means you can't use a fixed model size. And since the datasets are very big in size, it becomes almost impossible to fit a model with such a big input size.

So for time series data, you usually have to resort to sequence models that are built to retain patterns over time. Another way of approaching time series problems are clever techniques like sliding windows, which allow you to use standard machine learning models in an effective way.

Neural Networks

Neural networks are the cornerstone of machine learning; they were originally inspired by the biological nervous system. The basic idea behind these is to mimic how biological neurons process information. A neural network model consists of neurons/nodes, which represent functions with parameters W. These functions are nonlinear functions, Fig 3 shows the three most commonly used ones, ReLU, Sigmoid, and TanH. Multiple neurons can be stacked one after the other, resulting in a complex model consisting of functions of functions.

The model learns through an algorithm called backpropagation, where it recursively calculates derivatives to represent how much the loss (J) was affected by each parameter W. It then uses these derivatives to adjust W such that the loss is better next time.

This ability to learn complex relations makes neural networks particularly well-suited for the problems presented in this study.

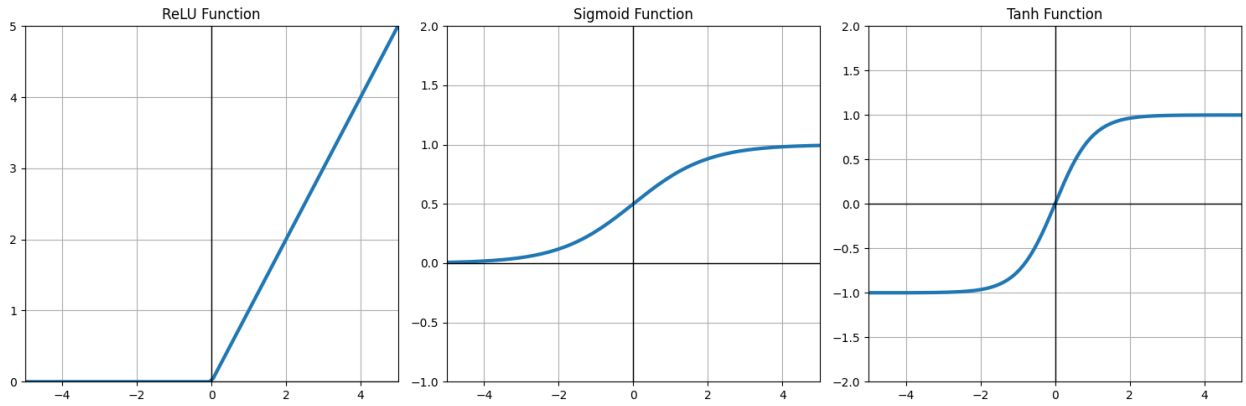


Fig 3. Most common non-linearity functions used in neural networks, ReLU, Sigmoid and TanH

XGBoost

XGboost, which stands for eXtreme Gradient Boosting, is a highly efficient and scalable implementation of gradient boosting. Gradient boosting is a machine learning method that uses decision trees, which are flowchart-like structures that make decisions based on the input data.

The ‘boosting’ refers to how XGBoost builds the trees in a sequential manner; each new tree created tries to correct errors made by its predecessors. This is done by focusing on where the previous trees performed poorly. XGBoost also incorporates a loss (J). Each new tree created aims to reduce this loss, thereby improving the model’s accuracy.

XGBoost models are known for their speed and efficiency. This, paired with their ability to handle large datasets, make them particularly good for the problems presented in this study.

Logistic Regression

Logistic regression can be thought of as a simplified neural network model with a single node that employs a sigmoid function (see Fig 3). This model, despite its simplicity, can be very effective for binary classification tasks, that is, predicting if a certain condition is true or false.

Due to the simplicity of logistic regression, its parameters can be inspected directly in order to understand the reasoning behind its predictions. This reason makes logistic regression a good candidate for this problem; in that *if* it does fit our data well, we may gain some insight into the underlying relations that were learned, which is something that is lacking in the neural network and XGBoost models.

Methodology

Initial Approach 1

For the first approach, I reshaped the dataset so that each data point includes measurements from a ‘window’ of its surrounding data points. I chose a window size of 10 steps, which is equivalent to a 50s time window. Fig 4 shows an example of this; for the time step 75, we would also include readings for time steps 70-80.

The objective for the model was then to classify each data point as one of {onset, wakeup, none}.

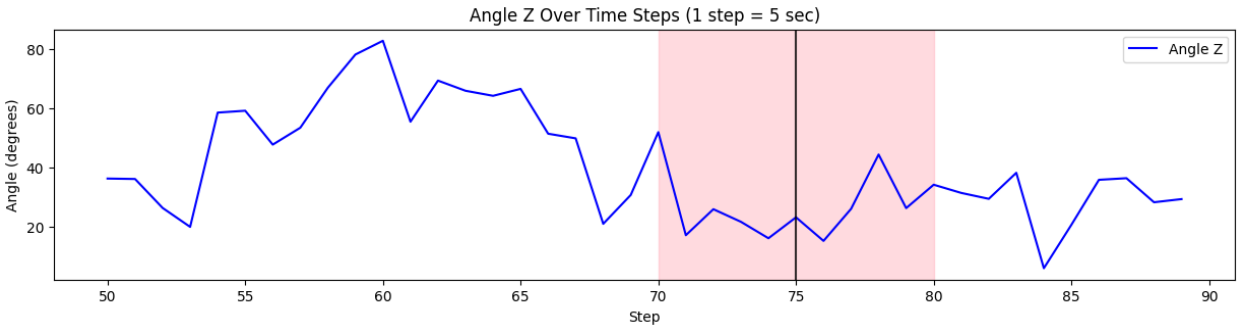


Fig 4. Example of a window in Approach 1 for time step = 75

Initial Approach 2

For the second approach, I used a simplified version of approach 1. I grouped the dataset to windows of surrounding measurements. I chose a window size of 400 steps, which is equivalent to a 33min window. The objective for the model was then to identify *if* a transition event occurred during that time window, making our classes = {transition, no transition}. We would later try to further classify them as onset or wakeup.

Fig 5 shows part of the dataset separated into 5 windows. The red-dotted line denotes a wakeup event. The labels for the green window would then be 'transition', while the rest of the 4 would be labeled as 'no transition'.

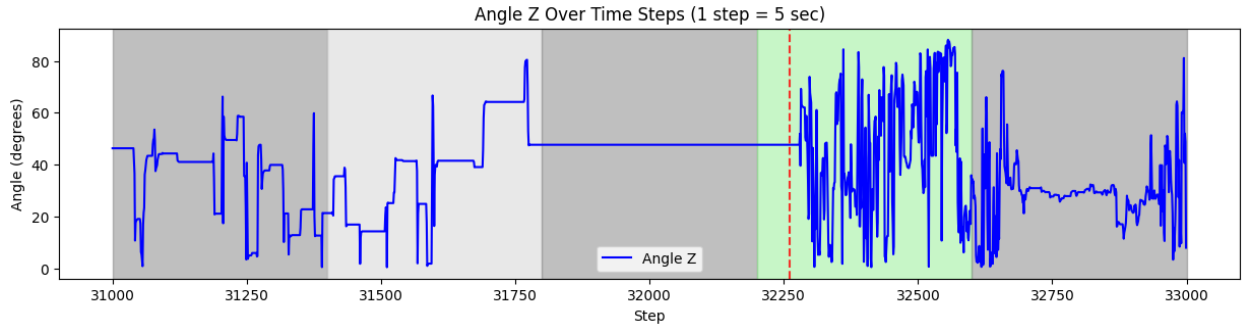


Fig 5. Example of windows in Approach 2 of size 400. Green shaded region indicates a window with a transition event. Any gray shaded region indicates otherwise.

Problems with Initial Approaches

Framing the problems like I did initially presented some significant challenges. The first being that the data was heavily skewed. In the first approach, the ratio of classes in the dataset was 1:1:17,000, and for the second, simplified approach, the ratio was 1:21. The second challenge was the fact that the model would have to detect very minor changes in the windows. And due to the size and diversity of this dataset, erroneous examples are common, and could confuse the model.

Because of these challenges, no model I tried to fit yielded any significant results. In fact, the model would always overfit to one class; as in, it would always predict the same class. The model would always predict the 'non-transition event' class.

An attempted solution involved adding a penalty in the loss function that added more weight to the rarer classes. But this either had no effect, or led to the opposite extreme: overfitting to the rare cases to the point that it would only predict them.

Given these persistent issues and overall lack of performance, this approach was abandoned in favor of other methods.

Revised Approach

I then came up with a revised approach. Instead of trying to predict the transition events directly, the model should try to predict if the person is awake or asleep. The model predictions could then be used to pinpoint when the transition events occur with relative accuracy.

This approach heavily unskewed our dataset, since now the ratio of labels (awake:asleep) was now 2:1. This approach also simplifies the problem, as detecting sleeping status is much simpler than very minor transition events themselves.

For this approach, I also reshaped the data differently. The following method was adapted from [3]. For a given data point and a window, instead of including the surrounding data, I calculated running statistics for the window. First, I'd select a window for each data point similar to the one shown in Fig 4, but this time with window size = 20. Then, I'd calculate the following statistics for the window for both anglez and enmo: diff, rolling mean, rolling max, rolling std, diff rolling mean, diff rolling max [Appendix A]. These statistics provide ample information regarding the window, and at the same time also keep our input to the model relatively small since we don't have to give it the whole of the window.

With the aforementioned statistics, we also include the hour of the timestamp (instead of the whole timestamp like in the previous approaches). Of all the timestamp markers, the hour mark is the most significant in terms of indicating when a person might sleep.

After a model is trained, it can be used to predict if a person was awake or asleep. After we obtain these predictions, a 'smoothing function', also adapted from [3], would be run on the outputs in order to remove any outliers in the model prediction. This smoothing function would simply calculate the mean of the probabilities predicted by the model in a set window of size 460, which is equivalent to 38 min.

The values would then be rounded to indicate awake or asleep. Using these smoothed values, the transition events were determined to be whenever this smoothed value changed from one to another.

Models

The subsequent sections provide an in-depth description of the machine learning models utilized in this study. These sections also include the hyperparameters I considered for each model. Hyperparameters, different from the parameters (W) of a model, are configurations that guide how a model learns. They can have a significant impact on a model's performance, and are thus very important considerations when building machine learning models.

Neural Network

The neural network architecture is shown below in Fig 6. It comprises 3 major layers with [128, 64,

32] nodes respectively, followed by an output layer that projects to 2 dimensions to match our 2 classes. Each major layer further consists of a linear layer that contains the main ‘trainable’ parameters (W), a batch normalization layer that normalizes the inputs in order to speed up the training process, an activation layer that adds non-linearity to the model, and a dropout layer that helps prevent the model from overfitting the training data.

Smaller and bigger neural networks were considered ([32] and [256, 256, 128, 128, 64, 64, 32, 32] nodes respectively). The smaller neural networks severely underperformed, and the larger models took significantly longer in training to produce any viable results. So I ended up not considering these models.

A few hyperparameters were considered for this model.

- Batch Size: due to the size of the dataset, data cannot be processed through the model all at once. It instead needs to be ‘batched’ into smaller parts to be processed. The size of this batch can considerably affect the performance of the model, so batch sizes chosen were = {4096, 16384, 65536}
- Epochs: epochs represents how many passes through the dataset the model is trained on. Training it on less passes might cause the model to underfit, while training it for too long might cause overfitting. Due to the size of the dataset, the epochs chosen were = {1, 3}
- Activations: as mentioned earlier, neural networks require a non-linearity, often called the activation function. Two common choices used in machine learning were chosen = {ReLU, TanH} (see Fig 3)
- Penalty: since the data is slightly skewed (1:2 ratio between asleep and awake), adding a small penalty of x1.5 for the less asleep class might result in slightly better predictions. So the values for this are = {Penalty, No Penalty}

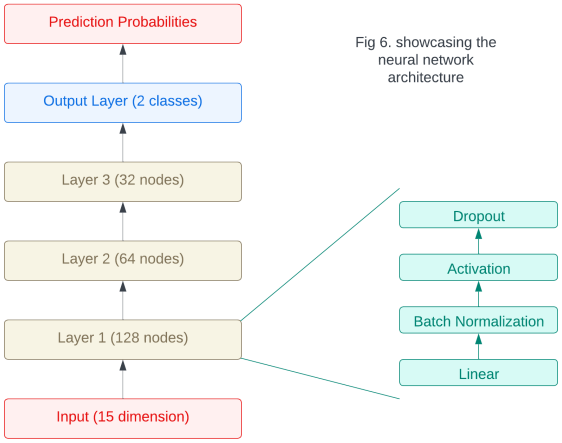


Fig 6. showcasing the neural network architecture

Fig 6. Neural Network Architecture used for model. It comprises of an input layer with size = 15, an output layer with size =2, and 3 main layers in the middle with sizes = [128, 64, 32] respectively.

XGBoost

There isn't much variety in terms of the architecture of an XGBoost model like there is for neural networks. A typical XGBoost model is shown below in Fig 7. A few hyperparameter were considered for this model:

- No. of Estimators/Trees: specifies the number of decision trees. The values considered were = {50, 100, 200}
- Max Depth: specifies the maximum depth of a tree. This allows more complex relations to be modeled, but comes at a high computation cost. The values considered were = {3, 6, 10}
- Min Child Weight: this sets the minimum sum of weights required by a child node, and is mainly used to control overfitting. The values considered were = {1, 100, 300}

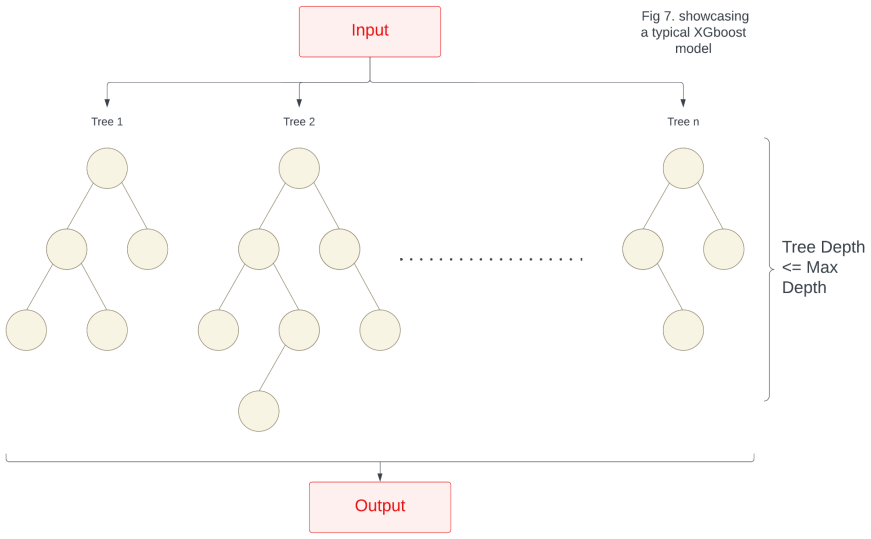


Fig 7. Typical XGBoost model

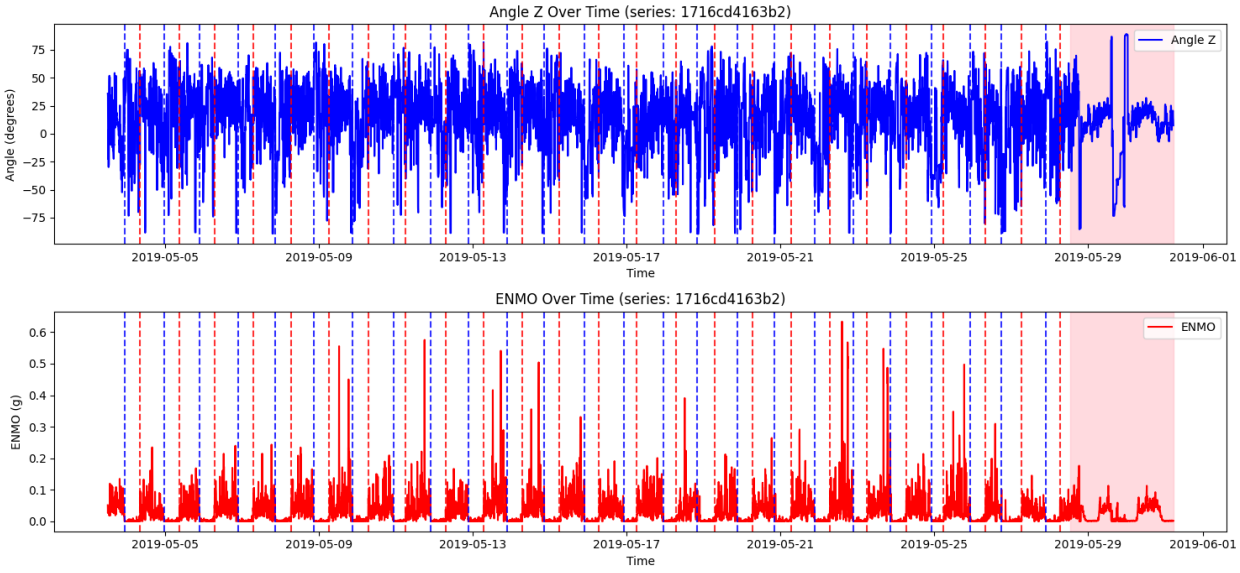
Logistic Regression

A logistic regression is simplistic in nature, and since its purpose in this study is not for performance, a model with a batch size of 16384 was considered with epochs = {1, 3}.

Test Set

In order to test the models, I collected 4 recordings from that dataset that were not included in the training set. These recordings were chosen to be ones that had an ample number of transition events (>30) and had minimal missing information.

These recordings like the rest of the remaining data, however, did still have missing information regarding transition events. An example of this is shown in Fig 8, where it can be seen that no transition events were recorded for the last few days. In the case where this was true for the start or the end of the recording, I dropped the data points with no corresponding labels. To better understand this, it is equivalent to dropping all the data points in the pink shaded area in Fig 8.



Evaluation

One metric used when evaluating models is the F1 Score. This is a metric better suited for classification tasks as opposed to accuracy, as this model takes into consideration dataset skews. This metric can be used to track how our model is progressing throughout training and how it performs on our modified problem, that is detecting if a person is awake or asleep.

$$F_1 = 2 * (precision * recall)/(precision + recall)$$

$$precision = (TruePositive)/(TruePositive + FalsePositive)$$

$$recall = (TruePositive)/(TruePositive + FalseNegative)$$

The main evaluation metric is the Average Precision of the detected events. This is an evaluation metric provided by the Kaggle competition hosts [4]. The details of the implementation are too lengthy in the context of this paper, but in short it is the average precision of detected events, averaged over error tolerances for time steps, averaged over transition event classes.

Important Note on Evaluation: when comparing the Score to those in the Kaggle leaderboard, there are a few limitations to consider. The major being that the dataset they evaluate on is not accessible to the public, so I was not able to evaluate on that. Another limitation is the fact that the test recordings still

contain some missing information on transition events. Although I removed the corresponding data points from the starts and ends, there are some in the middle that are difficult to remove without disrupting the flow of the data. These missing values can have a significant impact on the score since the models will predict transition events there, and the dataset has none.

Hyperparameter Search

An extensive hyperparameter search was conducted with the aforementioned hyperparameters. The model details, along with the performance results are given below.

Model	***Batch Size***	***Epochs***	***Activation***	***Penalty***	***F1 (asleep)***	***F1 (awake)***	***Average Precision***
Neural Network	4096	1	ReLU	Yes	0.9273	0.9672	*0.2894*
				No	0.9316	0.9698	0.2287
			TanH	Yes	0.9177	0.9621	0.2312
				No	0.9220	0.9655	0.2410
		3	ReLU	Yes	0.9353	0.9714	0.2693
				No	0.9356	0.9717	0.2220
			TanH	Yes	0.9266	0.9674	0.2401
				No	0.9290	0.9691	0.2182
	16384	1	ReLU	Yes	0.9223	0.9648	0.2411
				No	0.9233	0.9655	0.2561
			TanH	Yes	0.9084	0.9563	0.2226
				No	0.9132	0.9604	0.1950
		3	ReLU	Yes	0.9282	0.9678	0.2640
				No	0.9308	0.9692	*0.2939*
			TanH	Yes	0.9166	0.9614	0.2718
				No	0.9209	0.9645	0.2575
	65536	1	ReLU	Yes	0.8911	0.9439	0.2510
				No	0.8917	0.9463	0.2390
			TanH	Yes	0.8717	0.9314	0.1614
				No	0.8719	0.9331	0.1426
		3	ReLU	Yes	0.9082	0.9554	0.2552
				No	0.9116	0.9580	*0.2790*
			TanH	Yes	0.8878	0.9428	0.2086
				No	0.8953	0.9488	0.2160
Logistic Regression	16384	1	-	-	0.9042654	0.9042654	0.0593
		3	-	-	0.91992701	0.91992701	0.0789

Model	No. of Estimators	Max Depth	Min Child Weight	F1 (asleep)	F1 (awake)	Average Precision
XGBoost	50	3	1	0.9407	0.9683	0.2275
			100	0.9409	0.9684	0.2106
			300	0.9407	0.9684	0.2122
		6	1	0.9436	0.9697	*0.2621*
			100	0.9435	0.9696	0.2464
			300	0.9439	0.9699	0.2473
		10	1	0.9426	0.9692	0.2019
			100	0.9440	0.9698	0.2297
			300	0.9442	0.9699	0.2254
	100	3	1	0.9425	0.9692	0.2024
			100	0.9423	0.9691	0.2203
			300	0.9422	0.9691	0.2202
		6	1	0.9439	0.9698	0.2528
			100	0.9437	0.9697	0.2417
			300	0.9446	0.9702	*0.2610*
		10	1	0.9417	0.9687	0.2059
			100	0.9434	0.9696	0.2379
			300	0.9440	0.9699	0.2406
	200	3	1	0.9436	0.9697	0.2214
			100	0.9436	0.9697	0.2383
			300	0.9437	0.9698	0.2473
		6	1	0.9444	0.9701	0.2354
			100	0.9442	0.9700	0.2339
			300	0.9441	0.9700	0.2328
		10	1	0.9405	0.9681	0.2310
			100	0.9424	0.9690	0.2402
			300	0.9436	0.9696	0.2321

Results and Discussion

Overall from the F1 Score, we can see that the majority of models were able to perform well on predicting whether a person was awake or asleep, as the majority of them were able to achieve >0.9 F1 scores on both classes.

As for the main problem metric, the Average Precision, the best performing model achieved a precision of 0.2939. In contrast, the highest metric on the Kaggle leaderboard (as of writing this paper) is 0.797. There are various reasons mentioned in the Evaluation section why my precision is expected to be low, but these values suggest that there is still room for improvement.

The neural network models performed the best across all models, with 4 of them having precision >0.27 and one reaching as high as 0.2939. Despite having lower F1 scores than XGBoost models, it seems that neural networks were better able to capture the relation across examples. One significant note here is that the ReLU function seemed to consistently outperform the TanH function.

The XGBoost models showed relatively decent performance as well. Although they didn't perform as well as the best neural networks (the highest precision being 0.2621), they seemed to be relatively consistent across the board, with no model having precision <0.2.

The logistic regression models, although were able to gain an F1 Score > 0.9, significantly underperformed on the average precision, failing to even pass the 0.1 mark. Due to this, no further interpretation attempt was pursued.

Conclusion

This study underscored the importance of framing problems in machine learning, especially for time series datasets. Initially, our approach to detecting transition events directly posed very significant problems like dataset skew and hard-to-learn patterns that resulted in no significant performing model. By changing our approach to solve a much simpler problem like detecting awake status, and then using further techniques to pinpoint the transition event, we were able to circumvent the difficulties posed by the original framing and achieve better results.

Furthermore, our analysis of different models revealed something noteworthy: neural networks outperformed other models, specifically XGBoost. The best neural networks were better able to learn the patterns across data points and generalize. Logistic regression turned out to be too simple of a model, and thus yielded nothing of significance.

It is important to note, however, that the neural network performance may not generalize to larger models when compared to XGBoost. Larger neural networks may underperform compared to larger XGBoost models if computation is considered as they tend to be way more expensive in that regard.

Looking ahead, there are a few areas for further research and improvement. Exploring more advanced models like sequence models or sliding transformers may yield better results. Additionally, using convolutions instead of sliding windows may also be a viable way to extract window information.

References

[1] Kaggle. (2023). Child Mind Institute - Detect Sleep States. Retrieved from https://www.kaggle.com/competitions/child-mind-institute-detect-sleep-states/

[2] Carle McBirde Ellis. (2023). Zzzs: Lightweight training dataset + target. Retrieved from https://www.kaggle.com/datasets/carlmcbrideellis/zzzs-lightweight-training-dataset-target/data

[3] Carle McBirde Ellis. (2023). Zzzs: Random Forest model starter. Retrieved from https://www.kaggle.com/code/carlmcbrideellis/zzzs-random-forest-model-starter

[4] Kaggle Competition Metrics. (2023). Event Detection AP. Retrieved from https://www.kaggle.com/code/metric/event-detection-ap/notebook

Appendix A: Definitions of Statistical Measures

For any measurement m_n at a given time step n, assuming a window size = 2w, the definitions follow for m:

$$diff = m_{n+w} - m_{n-w}$$

$$rollin\_mean = avg(m_{n-w}, \ldots\ldots, m_{n+w})$$

$$rolling\_max = max(m_{n-w}, \ldots\ldots, m_{n+w})$$

$$rolling\_std = std(m_{n-w}, \ldots\ldots, m_{n+w})$$

$$diff\_rolling\_mean = avg(diff_{n-w}, \ldots\ldots, diff_{n+w})$$

$$diff\_rolling\_max = max(diff_{n-w}, \ldots\ldots, diff_{n+w})$$

Appendix B: Training Details

All XGBoost models were trained on Kaggle Notebooks on a CPU. All neural network & logistic regression models were trained on Google Colab with a CPU. For all Dropout layers in the neural network, dropout_rate = 0.5. The learning rate for all these models followed the same pattern, where e denotes the epoch number:

$$learning\_rate(e) = 0.001 * 0.5^{e-1}$$