Title

Portable, interpretable 5-class sleep staging from single-channel EEG using shared spectral features and multinomial logistic regression: models trained on Sleep-EDF Expanded and OpenNeuro BOAS

Authors

[Ryan Cameron]*, **[Divya Chander]**
[Lucidify Inc.]
*Corresponding author: **[ryan@lucidifyai.com]**

Abstract

Automated sleep staging is commonly performed with deep neural networks that can be difficult to deploy on lightweight or cross-platform applications. We describe a portable and interpretable sleep staging pipeline designed for integration into real-time visualization software. Two five-class (W, N1, N2, N3, REM) multinomial logistic regression models were trained using a shared set of spectral and signal-morphology features computed from 30-s single-channel EEG epochs. One model was trained on PhysioNet Sleep-EDF Expanded polysomnography (PSG) recordings and expert hypnograms, and a second model was trained on a BIDS-formatted OpenNeuro dataset (ds005555; “BOAS”) containing EEG with tabular events annotations. The pipeline standardizes features by training-set mean and standard deviation and exports the resulting linear models to a portable JSON format and an auto-generated C# implementation using numerically stable softmax. A companion evaluation script reproduces predictions from exported parameters and reports confusion matrices and classification metrics. This design enables consistent inference across Python, web, and Unity runtimes while retaining a transparent feature set suitable for scientific inspection.

Introduction

Sleep staging provides clinically and scientifically meaningful summaries of macro-architecture (wake, NREM stages, REM) and is foundational for sleep research and many applied neurotechnology products. While recent deep learning approaches have achieved strong performance, many are difficult to deploy in constrained environments (e.g., browsers, mobile devices, embedded compute) or to validate across heterogeneous runtimes. Interpretable linear models remain attractive when portability, auditability, and deterministic reproduction across platforms are primary constraints.

Here we describe a compact sleep staging system intended for cross-platform deployment. The system trains models from public datasets and exports parameters in a format that is consumed

directly by application code. Two models are trained on distinct source distributions—Sleep-EDF Expanded (PSG-grade) and OpenNeuro BOAS (wearable/headband vs PSG contexts)—to provide complementary inductive bias when applied to new EEG devices.

Materials and Methods

Source datasets

Sleep-EDF Expanded (PhysioNet).
Training data for the “Physio” model were derived from the Sleep-EDF Expanded database hosted on PhysioNet, which distributes EDF/EDF+ PSG signals and corresponding hypnogram annotations. (PhysioNet)

OpenNeuro BOAS (ds005555).
Training data for the “BOAS” model were derived from an OpenNeuro BIDS dataset (ds005555) providing EEG EDF files and associated events TSV annotations (including a human staging column used as labels). (OpenNeuro)

Rationale for using two datasets: the Sleep-EDF Expanded PSG distribution and the BOAS distribution differ in sensors, montage/channel naming, recording hardware, and likely noise/artifact statistics; training separate models allows users to select the model whose assumptions better match a target recording rather than relying on a single pooled model.

Manifest construction and record pairing

For reproducibility and automation, each dataset is first converted into a “manifest” CSV listing record identifiers and paths to EEG and annotation files.

- A Sleep-EDF manifest pairs PSG EDF files with hypnogram EDF files using filename conventions and prefix matching.
- A unified multi-dataset manifest generator also supports BOAS by locating EEG EDF files (by acquisition) and matching events TSV files (by folder, subject, or filename heuristics), storing dataset name, label column, and event source.

Epoching and label alignment

Epoch length.
Signals are segmented into contiguous 30-second epochs (configurable; default 30.0 s).

Sleep-EDF label extraction.
Sleep-EDF hypnogram annotations are read via MNE and mapped to 30-s epoch labels by assigning annotation descriptions to the epoch indices covered by each annotation interval. EEG is read from the PSG file using a single selected channel (default requested channel “EEG Fpz-Cz”; fallback selects the first channel matching preferred prefixes).

BOAS label extraction.
BOAS EEG is loaded from EDF, and labels are read from the events TSV (column configurable; typically `stage_hum`). The extractor supports alignment either via `begsample`/`endsample` or via `onset`/`duration` fields; unknown or missing codes are mapped to "UNK".

Sleep window definition.
To reduce training influence from long pre-sleep/post-sleep wake segments and from unknown-labeled regions, an "in_sleep_window" flag is computed as the epoch range from the first to last epoch whose label is neither W nor UNK.
This flag is stored per epoch for downstream filtering.

Feature extraction

Each 30-s epoch is featurized into a fixed 12-dimensional vector shared across datasets, enabling identical downstream training and inference.

Feature set.
The exported training script defines the default feature order as:
log bandpowers: `log_delta`, `log_theta`, `log_alpha`, `log_sigma`, `log_beta`; ratios: `log_delta_over_beta`, `log_sigma_over_theta`, `log_thetaalpha_over_beta`; plus `sef95`, `spec_entropy`, `rms`, `zero_cross_rate`.

Power spectral density estimation and derived metrics.
PSD is computed via Welch's method (SciPy) and used to compute bandpower summaries and spectral measures. The implementation uses constant detrending and density scaling and then computes log bandpowers and the ratio features, spectral edge frequency (95%), spectral entropy, RMS amplitude, and a zero-crossing-rate estimate.
(Reference: Welch PSD as implemented in SciPy.)

Model training procedure (supervised learning; multinomial logistic regression)

Training objective. Each 30 s EEG epoch is assigned a discrete sleep-stage label (*W, N1, N2, N3, REM*) and the model is trained to predict the labeled class from per-epoch features using supervised multi-class classification.[1]

Epoch definition and labels. EEG is segmented into fixed-length epochs with a default `epoch_len = 30.0` seconds.[2] Per epoch, the pipeline stores both a string label (e.g., `"N2"`) and an integer `label_id` using the fixed mapping {0:W, 1:N1, 2:N2, 3:N3, 4:REM}.[3] Epochs also store an `in_sleep_window` boolean indicating whether they fall between the first and last non-W/non-UNK epoch within a recording (used as a training filter).[4][5]

Per-epoch feature extraction. For each epoch, 12 engineered features are computed, including log-bandpowers (delta/theta/alpha/sigma/beta), several log-ratio features, and summary spectral/amplitude measures (`sef95`, `spec_entropy`, `rms`, `zero_cross_rate`).[1]

These are computed from a Welch PSD estimate (Hann window, detrend constant, density scaling), then combined into the final feature vector.[6]

Training-set filtering. Before training:

- Epochs with unknown labels are removed: `label_id >= 0`.[7]
- By default, training is restricted to the sleep window: `in_sleep_window == True` (unless explicitly overridden).[7]

Standardization. Features are z-scored using training-set statistics:

- `means = X.mean(axis=0)`
- `stds = X.std(axis=0)` with zero std values clamped to 1.0
- `Xz = (X - means) / stds`[8]

The exported model stores `means` and `stds` to ensure identical preprocessing at inference time.[9]

Classifier. The learning algorithm is **multinomial logistic regression** implemented via scikit-learn's `LogisticRegression`, configured explicitly with:

- `solver = "lbfgs"`
- `max_iter = 5000`
  All other hyperparameters are left at library defaults.[10]
  The model is fit via `clf.fit(Xz, y)` using the integer stage labels `y = label_id`.[10]

Learned parameters. Training produces:

- weight matrix `W = clf.coef_` with shape `[class, feature]`
- bias vector `b = clf.intercept_` with shape `[class]`[10]

Inference computation (as implemented/exported). For a standardized feature vector `x`:

- `logits = W x + b`

- `probs = softmax(logits)` (stable softmax is also implemented in the generated C# export).[11]

Export artifacts (reproducibility across runtimes).

- `model.json` includes `feature_order`, `means`, `stds`, `W`, `b`, and provenance notes including the filters used.[9]

- A generated Unity/C# classifier embeds the same parameters and applies the same standardization and stable softmax.[12]

- `golden_vectors.json` samples epochs per class and stores raw features, standardized features, logits, probabilities, and predicted labels for cross-runtime verification.[13]

Separate models per source domain.
The same pipeline is run independently to produce:

1. a “Physio” model trained on Sleep-EDF-derived epochs, and
2. a “BOAS” model trained on BOAS-derived epochs, using the same label space and feature schema.

Model export and cross-runtime reproducibility

Portable model JSON.
Trained parameters are exported to `model.json` containing: format identifier, label list, feature order, means/stds, weight matrix `W` and bias vector `b`, and training provenance notes including filtering choices.

Unity/C# export.
A C# source file is generated embedding identical arrays and implementing standardized linear logits and a numerically stable softmax for inference.

Golden vectors for verification.
A small set of per-class “golden” epochs is sampled, and their raw features, standardized features, logits, probabilities, and predicted labels are exported to `golden_vectors.json` to validate byte-for-byte equivalence across runtimes.

Embedding into web runtime.
A helper script embeds both model JSON objects into a single JavaScript file that assigns `window.LucidifySleepStageModels = { physio, boas }`, enabling direct use in browser applications without additional IO.

Evaluation procedure

A standalone evaluation script loads an epochs parquet and a model JSON, applies the same standardization, computes logits and softmax probabilities, and reports confusion matrix and per-class metrics using scikit-learn.

Planned reporting (fill with your actual runs):

- Dataset splits: [train/val/test protocol; subject-wise or record-wise]
- Metrics: overall accuracy, macro-F1, per-class precision/recall/F1, confusion matrices
- Optional: per-record aggregation and Cohen's κ (if desired)

Results

[Insert quantitative results here.]
Example structure:

- Sleep-EDF model performance on Sleep-EDF held-out records: Accuracy = **XX.X%**, Macro-F1 = **X.XX**, κ = **X.XX**.
- BOAS model performance on BOAS held-out records: Accuracy = **XX.X%**, Macro-F1 = **X.XX**, κ = **X.XX**.
- Cross-domain evaluation (optional): Sleep-EDF model on BOAS and BOAS model on Sleep-EDF.

Discussion

This work emphasizes *portability* and *auditability* over architectural complexity. Logistic regression over physiologically meaningful spectral summaries yields a compact model that is easily exported, inspected, and re-implemented. The explicit feature schema and golden-vector strategy provide a practical method to guarantee consistent inference across Python and Unity/web environments.

Training separate models on Sleep-EDF and BOAS reflects an assumption that staging priors and artifact statistics differ materially across PSG and wearable contexts. In deployment, selecting the model closer to the target acquisition domain may yield more plausible staging outputs on novel devices (e.g., forehead patches), though this should be empirically validated.

Limitations

- Single-channel EEG features do not incorporate EOG/EMG, which may reduce REM/N1 separability relative to multimodal PSG pipelines.
- No explicit temporal model is included in the classifier itself (e.g., HMM smoothing); predicted stage sequences may exhibit higher epoch-to-epoch variance.
- Performance will depend on montage differences, filtering, and sampling rates, as features are computed directly from the provided waveform without enforced resampling or artifact rejection in the core pipeline.

Data and code availability

- Sleep-EDF Expanded is available via PhysioNet under its posted terms. (PhysioNet)
- BOAS dataset is available via OpenNeuro (ds005555). (OpenNeuro)
- Code and exported model artifacts: **[insert your repository link / OSF / DOI]**.

References

1. PhysioNet Sleep-EDF Expanded dataset landing page and citation guidance. (PhysioNet)
2. Goldberger AL, et al. *PhysioBank, PhysioToolkit, and PhysioNet* (dataset platform paper). (OpenNeuro)
3. SciPy `signal.welch` documentation (Welch PSD).
4. scikit-learn `LogisticRegression` documentation.
5. MNE-Python EDF reading/annotations (used for EDF IO in the pipeline).

References (local files)

1. `train_and_export.py` (feature list, labels, softmax, training entry point)
   train_and_export
2. `extract_epochs_multi.py` (default epoch length)
   extract_epochs_multi
3. `train_and_export.py` (label mapping)
   train_and_export
4. `extract_sleepedf_epochs.py` (sleep window definition)
   extract_sleepedf_epochs
5. `extract_sleepedf_epochs.py` (epoch rows include `in_sleep_window`)
   extract_sleepedf_epochs
6. `extract_epochs_multi.py` (Welch PSD + feature computation)
   extract_epochs_multi
7. `train_and_export.py` (training filters `label_id>=0`, `in_sleep_window`)
   train_and_export

8. `train_and_export.py` (means/stds and z-scoring)
   train_and_export
9. `train_and_export.py` (`model.json` export + provenance notes)
   train_and_export
10. `train_and_export.py` (LogisticRegression config and fit; W/b extraction)
    train_and_export
11. `train_and_export.py` (logits/probs computation used for golden vectors)
    train_and_export
12. `train_and_export.py` (generated C# standardization/logits/softmax)
    train_and_export

    train_and_export
13. `train_and_export.py` (golden vector contents exported)
    train_and_export