Automatic Sleep Scoring with Multi-Signal Processing

V1.0.

2025-02-10

1. Introduction

Project Description and Objectives

The goal of this project is to develop an **automatic sleep-scoring system** using multiple **physiological signals**. Sleep scoring is a fundamental process in sleep medicine where continuous recordings are segmented into **30-second epochs** and classified into different sleep stages according to **AASM (American Academy of Sleep Medicine) standards**.

This project follows a phased approach, beginning with **EEG analysis** and progressively incorporating additional **biosignals** (EOG, EMG, ECG, respiration) to enhance classification accuracy. Students will:

- **Preprocess raw biosignal data** to remove artefacts.
- **Extract time, frequency, and time-frequency features**.
- **Statistically analyze feature significance across sleep stages**.
- **Develop and optimize machine learning models for classification**.
- (Optional!) **Detect sleep apnea episodes using SpO2 and respiration signals**.

2. Available Data

2.1 Data Files

The dataset includes **20 EDF (European Data Format) files**, each containing:

- **EEG (Electroencephalogram)** – Brain activity (2 channels)
- **EOG (Electrooculogram)** – Eye movements (2 channels: left and right eye)
- **EMG (Electromyogram)** – Muscle activity
- **ECG (Electrocardiogram)** – Heart activity
- **Body Position**
- **SpO2 (Oxygen Saturation)**
- **Thoracic and Abdominal Respiration**

2.2 Annotations

Each recording is accompanied by an **XML file** containing:

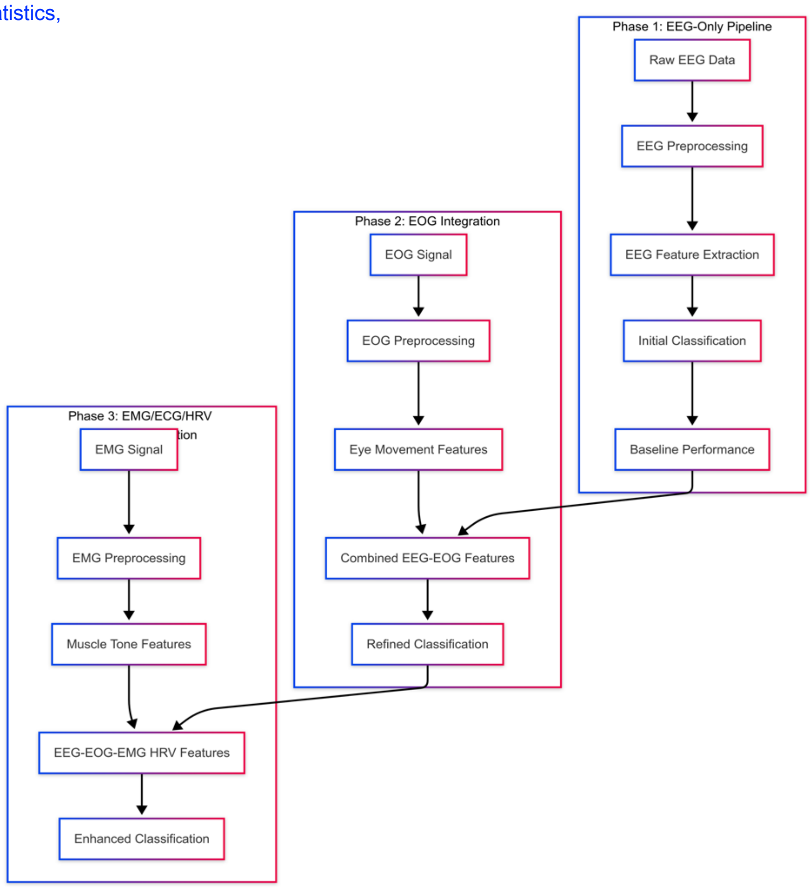
- **Sleep stage annotations** (30-second epochs but reported with start time and duration)
- **Signal quality markers**
- **Event annotations** (e.g., apnea episodes, desaturation events)
- **Sleep stage classifications** (Wake, N1, N2, N3, REM)

2.3 File Access

- **EDF files**: Use `edfread.m` for signal extraction (use the provided version and not the Matlab version).
- **XML files**: Use `xml_read.m` to parse annotations.
- **Recommendation**: Convert loaded data to `.mat` format as a cache for faster access.

3. Implementation Phases

1. Preprocess signals in MATLAB
2. Usual stats pipelines for statistics, correlations etc



Phase 1: EEG-Only Pipeline

Preprocessing

- Baseline drift removal
- Muscle noise filtering
- Power line interference removal
- Any other artefact detection and handling

Feature Extraction

- **Time-domain, for instance**:
  - Mean, variance, skewness, kurtosis
  - Zero-crossing rate
  - Hjorth parameters
- **Frequency-domain, for instance**::
  - Power in standard EEG bands (delta, theta, alpha, beta, gamma)

- Spectral edge frequency
- Relative band powers

- **Time frequency for instance**:
  - Wavelet coefficients
  - Spectral entropy

Initial Classification

- **k-NN classifier** for baseline performance
- **Confusion matrix evaluation**
- **Limitations documentation**

Phase 2: EEG + EOG Integration

EOG Preprocessing

- Baseline correction — high-pass Butterworth filter 0.5Hz
- Noise filtering — low-pass Butterworth filter 30Hz

EOG Feature Extraction

- Sleep Eye movement detection (optional)
- Blink Detection (optional)
- Movement density
- Blink rate and characteristics
- Slow/rapid eye movement patterns

Combined Features & Performance Evaluation

- Merge EEG and EOG features
- Feature selection
- Performance comparison with Phase 1

Phase 3: EEG + EOG + other signals (EMG or ECG or Respiration) Integration

EMG Preprocessing

- Bandpass filtering
- Envelope extraction
- Artifact removal

EMG Feature Extraction, for instance

- RMS amplitude
- Frequency content
- Burst detection
- Muscle tone analysis

Complete Integration & Optimization

1. Better Wavelet Tuning

The wavelet type (db4, sym3, etc.) and decomposition level (4, 5, 6, …) can be optimized based on your data.

•You could also define more sub-bands (e.g., separate 3–10 Hz from 10–20 Hz, 20–30 Hz, etc.) if that helps your classification goals.

2. Additional EOG Markers

Some pipelines track left vs. right EOG for certain advanced REM or micro-saccade analyses. If you have two EOG channels, you might want to do difference signals or ICA to isolate pure ocular signals from other EEG contamination.

3.Artifact Rejection

If EOG or EEG contain big saturations or clipped segments, wavelet or other artifact rejection might be essential. Your code does a basic wavelet denoise, but more advanced or domain-specific methods could help.

4.Integration with EEG

Typically, full sleep scoring uses EEG (delta power for N3, spindles for N2, alpha for wake, etc.) combined with EOG (REM detection). You can unify these features into a single classification.
The pipeline would be the same, but you'd gather EEG-based features and EOG-based features for each epoch, then train a multi-modal classifier.

3. EOG Feature Extraction
From each 30-second EOG epoch, you extracted:

1. Blink Detection
Adaptive threshold is computed to detect peaks in the absolute EOG signal.
peakStdFactor = 3, plus a cap at 0.9×max amplitude prevents "Invalid MinPeakHeight" warnings.
Blink Rate (blinks per minute) is then computed from the detected blink peaks.Also counts the number of blinks (numBlinks) in that epoch.

2. Movement Density
The EOG signal is split into 5-second mini-windows.
Standard deviation in each 5-second window is computed and averaged.
This yields a single movementDensityMean per epoch (a measure of general EOG fluctuations).
3. Slow vs. Rapid Eye Movement Power
Using Welch's PSD (pwelch), you integrate power in two frequency bands:
Slow Band ~ 0.5–3 HzRapid Band ~ 3–30 Hz
A ratio of slow-to-rapid power (powerRatio) is stored. Large ratio values might indicate predominantly slow EOG activity.

4.Wavelet-Based Features (Optional)
Decompose the EOG epoch into wavelet packet nodes.
Summate power in a "slow" node set (<3 Hz) vs. a "fast" node set (3–30 Hz).
You store waveletSlowPower, waveletFastPower, and a wavelet-based ratio.
Thus, you have five or six main EOG features:
1 blinkRatePerMin (blinks/min)
2. movementDensityMean (std-based movement measure)3.powerRatio (slow vs. rapid from PSD)
4.waveletSlowPower, waveletFastPower, and waveletRatio (optional wavelet sub-bands)

1.Labeling
You used the sleep-stage labels from the XML (if available). Some epochs might be unknown (-1).
2.Feature Standardization
For each fold in cross-validation, you computed mean/std on the training set and applied z-score scaling to both training and test sets.
3. 5-Fold Cross Validation
Random Forest (50 trees) with a uniform prior if doBalanceRF=true.
Multiclass SVM (ECOC with RBF kernel) with a cost matrix if doBalanceSVM=true.
4. Confusion Matrices & Accuracy
You plotted confusion matrices for the Random Forest and SVM.
In your example run, Random Forest had ~82% accuracy, while SVM had ~87% accuracy.

- Merge EEG, EOG, and EMG features
- Feature selection
- Final classification tuning

4. Technical Requirements

4.2 Performance Evaluation

Classification Tool

To simplify model training and evaluation, you can use MATLAB's **Classification Learner** app, which provides an interactive environment for training, validating, and comparing different machine learning models. More information can be found at MATLAB Classification Learner.

Metrics

- **Overall accuracy**
- **Per-stage accuracy**
- **F-score**
- **Confusion matrix**

Cross-Validation Methods

- **Leave-one-subject-out** cross-validation
- **K-fold cross-validation** for parameter tuning

5. Implementation Guidelines

Iterative Development

- Start with a simple model and progressively add complexity.
- Document performance at each phase.
- Validate each modification quantitatively.

Code Organization

- Modular functions for each component
- Clear documentation
- Version control
- Parameter configuration files

Optimization Strategies

- Feature selection methods
- Hyperparameter tuning
- Classification algorithm comparison
- Performance vs. complexity trade-offs

6. Deliverable

Code Repository

- Preprocessing functions
- Feature extraction code
- Classification implementations
- Utility functions

Final Report

- Methodology description
- Results and analysis
- Comparative Performance
- Future improvement

8. Timeline Recommendations

- **Week 1-3**: Phase 1 implementation
- **Week 4-6**: Phase 2 implementation
- **Week 7-8**: Phase 3 implementation
- **Week 9-10**: Optimization and Report

9. Report Guidelines

The final report should be concise, focusing on key findings and improvements rather than a step-by-step history of the project. The goal is to document:

- The **final methodology** with a brief summary of the approach taken.
- **Improvements made** across phases of implementation.
- **Final results** with relevant figures and tables.
- **Lessons learned** and potential future improvements.

Report Constraints

- **Maximum Length**: **15 pages**
- **Figures and Tables**: **Up to 12 combined**
- **Structure**:
    - **Introduction** (1 page): Brief background and objectives.
    - **Methods** (2-3 pages): Summary of preprocessing, feature extraction, and classification.
    - **Results & Discussion** (4-5 pages): Key findings, improvements over iterations, and comparison of models.
    - **Conclusion** (1 page): Summary of results and future work suggestions.
    - **References** (not counted towards page limit).

Key Focus Areas

- **Comparative Analysis**: Instead of listing all methods tried, emphasize **how modifications improved performance**.
- **Performance Metrics**: Report **accuracy, F-score, and confusion matrix** comparisons between phases.
- **Visualization**: Use clear **figures and tables** to highlight improvements.
- **Lessons Learned**: Document unexpected challenges and insights gained.

10. Iterative Development vs. Waterfall Development

Development Approaches

In software and data science projects, two primary development methodologies are commonly used: **Waterfall Development** and **Iterative Development**.

- **Waterfall Development** follows a sequential, linear process where each stage (e.g., preprocessing, feature extraction, model training) is completed before moving to the next. While this ensures thorough documentation and clear requirements, it lacks flexibility when encountering unexpected issues.
- **Iterative Development** is an agile approach that emphasizes repeated cycles of improvements, allowing teams to quickly adapt to new insights, optimize performance, and adjust based on real-time evaluation.

Advantages of Iterative Development

- **Early Detection of Issues**: Errors in preprocessing or feature extraction can be addressed without waiting for the final model.
- **Improved Adaptability**: If a classifier performs poorly, feature selection can be reworked without restarting the entire process.
- **Incremental Performance Gains**: Each iteration can be optimized for better accuracy, reducing risk in later stages.

Examples of Iteration Planning

In this project, an **iterative full-pipeline approach** is recommended. Each iteration covers the entire workflow—preprocessing, feature extraction, feature selection, and classification—ensuring that refinements in one stage are immediately evaluated within the full system. This allows for rapid feedback and adjustments, leading to a more optimized sleep scoring model.

Iteration Plan

- **Iteration 1**: Implement a basic pipeline with minimal preprocessing and a simple classifier (e.g., k-NN). Evaluate baseline performance.
- **Iteration 2**: Refine preprocessing steps, introducing artifact removal techniques (e.g., notch filtering, baseline drift correction). Compare the impact on feature stability.

- **Iteration 3**: Optimize feature selection by reducing dimensionality using methods like Principal Component Analysis (PCA) or feature importance ranking.
- **Iteration 4**: Fine-tune classification models, experimenting with different algorithms such as Support Vector Machine (SVM), Random Forest, and Neural Networks. Adjust hyperparameters for optimal performance.
- **Iteration 5+**: Continue refining based on validation results, adjusting preprocessing, feature engineering, or classification parameters as needed.

Each iteration results in a **fully functional sleep scoring system**, ensuring that improvements are systematically integrated rather than being addressed in isolated steps. This approach reduces risk, enhances adaptability, and improves final classification accuracy.