

# ECG-based Seizure Detection Agent

## Explanation of the Training Pipeline Implemented in `agent_ecg.py`

Safwen Gharbi

November 25, 2025

## 1 Introduction

This report describes the machine learning pipeline implemented in the Python script `agent_ecg.py`. The goal is to train a seizure detection agent using **ECG (electrocardiogram)** features extracted from the federated project dataset `ml_ready_balanced.csv`. The agent predicts, for each short ECG window, whether it corresponds to a seizure or not.

Only the ECG modality is used. Each row in the CSV corresponds to a short time window (epoch) of ECG data with pre-computed features and a binary label:

$$\text{Label} = \begin{cases} 1 & \text{seizure window} \\ 0 & \text{non-seizure window} \end{cases}$$

The pipeline is designed to be:

- **Realistic:** subject-wise cross-validation to avoid patient leakage.
- **Robust:** feature selection and hyperparameter search for XGBoost.
- **Deployable:** all preprocessing and model artefacts are saved and can be reused in an inference service.

The main outputs of `agent_ecg.py` are:

- a trained XGBoost model;
- a fitted StandardScaler;
- a JSON configuration file, `model_info.json`, summarising performance metrics, selected features, and decision threshold.

## 2 Data and Preprocessing

### 2.1 Input data

The CSV file `ml_ready_balanced.csv` contains, among others, the following columns:

- **Subject\_ID:** patient identifier (string, e.g. `sub-005`).
- **Modality:** signal type (ECG, EEG, EMG, ...).
- **Label:** binary seizure annotation (0 / 1).
- **Onset\_Seconds, Duration\_Seconds, Epoch\_Index:** time information for the window.
- **Sex:** categorical demographic feature.

- $\approx 900$  numerical features: wavelet, time-domain, and spectral descriptors for each ECG channel, for example:  
`Channel_0.Wavelet_Detail3.Kurtosis,`  
`Channel_0.Freq_SpectralEntropy,` etc.

The script first filters the data to keep **ECG-only** rows:

$$\text{Modality} = \text{ECG}.$$

## 2.2 Feature matrix, labels and groups

The function that prepares the data (in `agent_ecg.py`) performs the following steps:

1. Filter the DataFrame to keep rows where `Modality = ECG`.
2. Build:
  - $X$ : the feature matrix obtained by dropping `Label`, `Subject_ID`, `Modality`, `split`, `Onset_Seconds`, `Duration_Seconds`, `Epoch_Index`.
  - $y$ : the binary label column `Label`.
  - `groups`: the `Subject_ID` column, used for grouped cross-validation.
3. Apply one-hot encoding to the categorical column `Sex` using `pandas.get_dummies`, so that sex is represented as a binary feature.

The tuple  $(X, y, \text{groups})$  is then passed to a grouped cross-validation scheme to respect subject boundaries.

## 3 Feature Selection and Scaling

Because  $X$  initially contains close to 900 features, the script performs feature selection to reduce dimensionality and remove noisy or redundant features.

### 3.1 Mutual information feature selection

The script uses the function `mutual_info_classif` from `sklearn.feature_selection` to compute the **mutual information** between each feature  $X_j$  and the label  $y$ :

$$I(X_j; y) = \sum_{x_j} \sum_y p(x_j, y) \log \frac{p(x_j, y)}{p(x_j)p(y)}.$$

Features are ranked by their mutual information scores, and the top `TOP_K_FEATURES` (set to 200) are selected:

$$\mathcal{S} = \{j \mid j \text{ is among the 200 features with highest } I(X_j; y)\}.$$

The indices of these selected features are stored in `selected_feature_indices`, and their names in the `feature_names` array inside `model_info.json`.

### 3.2 Standardisation

On the selected subset of features, a **StandardScaler** is fitted:

$$X_{ij}^{\text{scaled}} = \frac{X_{ij} - \mu_j}{\sigma_j},$$

where  $\mu_j$  and  $\sigma_j$  are the mean and standard deviation of feature  $j$  on the training data.

This scaler is later applied to any new data before passing it to the model. The fitted scaler is saved in `scaler.joblib`.

## 4 Model: XGBoost Classifier

### 4.1 Choice of model

The main model is an **XGBoost** classifier (`XGBClassifier`). This choice is motivated by:

- Good performance on structured/tabular data.
- Ability to capture non-linear interactions between features.
- Robustness to heterogeneous feature scales (in combination with scaling).
- Extensive use in the literature and good tooling support.

### 4.2 Hyperparameter search

The script performs a **randomized hyperparameter search** using `RandomizedSearchCV`. A parameter distribution is defined (for example):

- `n_estimators`  $\in \{100, 200, 300, 500, 700\}$ ,
- `max_depth`  $\in \{3, 4, 5, 6, 8, 10\}$ ,
- `learning_rate`  $\in \{0.01, 0.03, 0.05, 0.1, 0.2\}$ ,
- `subsample`  $\in \{0.6, 0.7, 0.8, 0.9, 1.0\}$ ,
- `colsample_bytree`  $\in \{0.6, 0.7, 0.8, 0.9, 1.0\}$ ,
- `gamma`  $\in \{0, 0.1, 0.2, 0.5\}$ ,
- `min_child_weight`  $\in \{1, 3, 5, 7\}$ .

`RandomizedSearchCV` samples a fixed number of random combinations (e.g. 20) and performs an internal cross-validation for each, using F1-score as the optimisation metric. The best configuration is kept for each outer fold.

## 5 Cross-Validation and Threshold Tuning

### 5.1 Stratified grouped cross-validation

The script uses a grouped, stratified cross-validation scheme with  $K = 5$  folds (`N_FOLDS = 5`):

- **Grouped:** samples from the same subject (`Subject_ID`) never appear in both training and validation of the same fold.
- **Stratified:** the proportion of seizure vs non-seizure windows is approximately preserved across folds.

For each fold:

1. Split  $(X, y)$  into training and validation sets, respecting subject groups.
2. Select the top 200 features by mutual information on the training set.
3. Fit the `StandardScaler` on the selected training features and transform both training and validation.
4. Run `RandomizedSearchCV` on the training fold to find the best XGBoost configuration.
5. Fit the best XGBoost model on the scaled training data.
6. Obtain predicted probabilities  $\{p_i\}$  on the validation data.

## 5.2 Decision threshold optimisation

Instead of fixing the decision threshold at 0.5, the script searches over a grid of thresholds:

$$\theta \in \{0.10, 0.11, \dots, 0.89, 0.90\}.$$

For each threshold  $\theta$ , the predicted label is

$$\hat{y}_i = \begin{cases} 1 & \text{if } p_i \geq \theta, \\ 0 & \text{otherwise.} \end{cases}$$

For each  $\theta$ , the following metrics are computed on the validation data:

- Accuracy:

$$\text{Acc} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}},$$

- Precision:

$$\text{Prec} = \frac{\text{TP}}{\text{TP} + \text{FP}},$$

- Recall (sensitivity):

$$\text{Rec} = \frac{\text{TP}}{\text{TP} + \text{FN}},$$

- F1-score:

$$\text{F1} = \frac{2 \cdot \text{Prec} \cdot \text{Rec}}{\text{Prec} + \text{Rec}}.$$

The threshold that maximises the F1-score on the validation fold is kept. At the end of cross-validation, the thresholds from all folds are averaged to obtain a *global* estimate, stored as `final_threshold_estimated`.

## 5.3 Mean cross-validation metrics

The script aggregates the validation metrics over the 5 folds and computes the mean performance. The `model_info.json` file contains the following section (`cv_metrics_mean`):

```
"cv_metrics_mean": {
  "accuracy": 0.7243708425293571,
  "precision": 0.7028677589489518,
  "recall": 0.8102393115574957,
  "f1": 0.7491299446157702,
  "roc_auc": 0.7907878688703714,
  "pr_auc": 0.793431973344753,
  "threshold": 0.396
},
"final_threshold_estimated": 0.396,
```

This means that, on average across folds:

- Accuracy  $\approx 0.72$ ,
- Precision  $\approx 0.70$ ,
- Recall (sensitivity)  $\approx 0.81$ ,
- F1-score  $\approx 0.75$ ,

- ROC-AUC  $\approx 0.79$ ,
- PR-AUC  $\approx 0.79$ ,
- Best decision threshold  $\theta \approx 0.396$ .

The threshold is intentionally lower than 0.5 in order to favour seizure detection (high recall) at the cost of some false positives, which is appropriate for a screening or decision-support agent.

## 6 Final Model and Saved Artefacts

### 6.1 Final training on full data

After cross-validation, the script retrains a final model on the **full** dataset:

1. Select the top 200 features by mutual information on all samples.
2. Fit a new StandardScaler on these 200 features.
3. Run RandomizedSearchCV on the full data to choose the best hyperparameters.
4. Train the XGBoost model with these hyperparameters on all scaled samples.

This final model is then saved to disk, together with the scaler and metadata, so that the same preprocessing and prediction logic can be applied at inference time.

### 6.2 Artefacts in model\_info.json

The JSON file `model_info.json` contains:

- `cv_metrics_mean`: the cross-validation metrics shown above.
- `final_threshold_estimated`: the decision threshold  $\theta \approx 0.396$ .
- `selected_feature_indices`: the indices of the selected 200 features in the original encoded feature matrix.
- `feature_names`: the names of these 200 features, in the correct order expected by the scaler and the model, for example:
  - `Channel_0_Wavelet_Detail3_Kurtosis`,
  - `Channel_0_Wavelet_Detail2_Kurtosis`,
  - `Channel_0_Time_LineLength`,
  - `Channel_0_Freq_SpectralEntropy`,
  - ...
- `config`: configuration parameters, e.g. `TOP_K_FEATURES = 200`, `N_FOLDS = 5`, `RANDOM_STATE = 42`.
- `timestamp`: the date and time of training.

The model and scaler are saved in:

- `xgb_ecg_seizure_model.joblib`,
- `scaler.joblib`.

### 6.3 Using the model at inference time

To use the model on new data:

1. Compute the same ECG features for a new window (at least those listed in `feature_names`).
2. Build a one-row vector in the exact order of `feature_names`.
3. Apply the saved StandardScaler (`scaler.joblib`).
4. Pass the scaled vector to the XGBoost model to obtain a seizure probability  $p$ .
5. Predict seizure if  $p \geq 0.396$ , non-seizure otherwise.

This guarantees that inference is consistent with the training pipeline.

## 7 Summary of the Code Structure

Table 1 summarises the main responsibilities inside `agent_ecg.py`.

Component / Function	Role
Data loading	Load <code>ml_ready_balanced.csv</code> , filter ECG rows, extract $X$ , $y$ , and subject groups.
One-hot encoding	Encode <code>Sex</code> as binary variables.
Feature selection	Use mutual information to select the top 200 most informative features.
Scaling	Fit a StandardScaler on selected training features, apply to all splits.
Hyperparameter tuning	Apply <code>RandomizedSearchCV</code> on XGBoost using F1-score as objective.
Cross-validation	Use grouped, stratified 5-fold CV over subjects; compute mean metrics.
Threshold tuning	Scan thresholds and select the one that maximises F1 on validation data.
Final training	Retrain XGBoost on all data with selected features and best hyperparameters.
Saving artefacts	Save model, scaler, selected feature names, threshold, and metrics in <code>artifacts/</code> .

Table 1: High-level summary of the main components in `agent_ecg.py`.

## 8 Conclusion

The script `agent_ecg.py` implements a complete training pipeline for an ECG-based seizure detection agent. Its main characteristics are:

- subject-wise cross-validation to avoid data leakage between patients;
- mutual information feature selection and standardisation;
- XGBoost classifier with randomised hyperparameter search;
- decision threshold tuned to maximise F1-score and favour high sensitivity;

- all necessary artefacts stored for later deployment in an inference service.

The cross-validation results (accuracy  $\approx 0.72$ , F1-score  $\approx 0.75$ , recall  $\approx 0.81$ ) indicate that the agent captures meaningful seizure-related patterns in ECG features, under a realistic evaluation protocol where subjects do not leak across folds. The saved artefacts (`model_info.json`, model and scaler) provide a solid basis for deploying this agent as a component in a larger clinical decision-support system.