Effort-To-Compress Ratio-Based Feature Extraction & Machine Learning Methods for EEG Signal Classification

Yojit Kumar
yojit.kumar@students.iiserpune.ac.in
Indian Institute of Science Education and Research Pune
Pune, Maharasthra, India

Nithin Nagaraj
nithin@nias.res.in
Complex Systems Programme,
National Institute of Advanced Studies
Bengaluru, India

Abstract

The classification of electroencephalogram (EEG) signals is critical for brain-computer interfaces and clinical diagnostics. This paper investigates the efficacy of a compression complexity-based feature derived from the Non-Sequential Recursive Pair Substitution (NSRPS) algorithm, the Effort-to-Compress (ETC) ratio, for robust EEG state classification. We extracted features from the publicly available PhysioNet Motor Movement/Imagery Dataset, focusing on the binary classification of Eyes-Open (EO) and Eyes-Closed (EC) states across 109 subjects. We assess features extracted directly from the signal and through an ordinal analysis approach, testing them with a suite of seven machine learning classifiers. The ordinal analysis method proved superior, with Logistic Regression and Support Vector Machine models achieving a classification accuracy of 88.6%. A key finding is the differential behaviour of ETC in filtered versus unfiltered data. In alpha-band filtered data, complexity was higher for the EO state, consistent with known alpha desynchronisation in the EC state. In unfiltered data, this trend was inverted, suggesting that ETC is also highly sensitive to signal artefacts. Our results validate ETC as a potent and discriminative feature for EEG analysis, with significant potential for robust, real-world applications.

CCS Concepts

• **Applied computing** → *Life and medical sciences*; • **Computing methodologies** → *Symbolic and algebraic algorithms.*

Keywords

effort-to-compress, machine learning, classification, permutation entropy

ACM Reference Format:
Yojit Kumar and Nithin Nagaraj. 2018. Effort-To-Compress Ratio-Based Feature Extraction & Machine Learning Methods for EEG Signal Classification. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

Conference acronym 'XX, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Electroencephalography (EEG) is a foundational technology in modern neuroscience and biomedical engineering. EEG signals capture the brain's electrical activity in a non-invasive and cost-effective manner. The applications of EEG are extensive and impactful, ranging from *brain-computer interface (BCI)* development to clinical diagnosis of neurological disorders like epilepsy, Alzheimer's, Dementia, and Parkinson's disease [22]. The introduction of computer-aided diagnosis (CAD) systems with artificial intelligence (AI) have significantly improved the diagnostic utility of EEG. By integrating machine learning (ML) and deep learning (DL) methodologies, these systems can identify often subtle patterns within complex EEG data [6].

A critical step in EEG classification is feature extraction. EEG recordings are typically high-dimensional (many channels, long time series) and highly nonstationary, so extracting informative features is necessary to reduce dimensionality and improve classifier performance. Dimensionality reduction through feature extraction compacts the raw EEG into a smaller set of descriptive measurements, which can greatly speed up learning and reduce overfitting. In practice, careful feature design and selection are known to reduce computational complexity and enhance the accuracy of machine learning models for EEG-based classification and diagnosis [21].

While traditional feature domains have proven effective, they are primarily based on linear assumptions and may not fully capture the rich, chaotic, and nonlinear dynamics inherent in EEG signals. Nonlinear features like complexity and entropy measures that are rooted in chaos theory and information theory aim to quantify the complexity of the signal. For example, measures such as *Lempel–Ziv complexity (LZC)* [15] and *Permutation Entropy (PE)* [2] have been used as compact features of EEG signals. Both metrics are conceptually simple, robust to noise, and suitable for short, nonstationary recordings. In particular, complexity indices like LZC and PE have been shown to be effective features in EEG analysis [8, 11].

PE inherently possesses several highly desirable properties for EEG analysis. It is conceptually simple, computationally very fast, and robust to noise and artefacts [2]. PE has been successfully applied in numerous EEG contexts, with various methodological improvements like *modified Permutation Entropy (mPE)* [5]. These improvements show us a trajectory of refinement, and limitations of one approach directly motivate the development of the next. This narrative of iterative improvement provides us with the ideal context for introducing and evaluating the *Effort-to-Compress (ETC)* [17] ratio as another novel approach in this ongoing search for the optimal complexity feature.

In this work, we specifically study the use of ETC for the binary classification of a large EEG dataset [9, 19, 20], where we try to distinguish between Eyes Open or Eyes Closed of the subject using a very simple method of feature extraction from the EEG signals and testing the performance of different machine learning algorithms on it.

The paper is organised as follows. We first define ETC and the methods for feature extraction. We briefly present the different machine learning algorithms used. We detail the experimental results and discuss them. We finally present a summary and conclusion of the work.

2 Methodology

We are specifically using the Effort-To-Compress (ETC) Ratio to form our feature vector. ETC is a compression-based complexity measure. ETC uses the *Non-Sequential Recursive Pair Substitution (NSRPS)* algorithm [7] and is a compression-complexity measure applied to time series. We start by converting the given time series into a symbolic sequence by partitioning the data into the desired number of bins. We parse the sequence from left to right to find the most frequently occurring pair and replace the pair with a new symbol. This process is iteratively repeated until we are left with a single symbol. At this point, the *Shannon Entropy* of the sequence has reduced to zero. The number of steps it took for us to reach this stage is calculated as ETC. We use *normalised ETC* which is defined as:

$$\text{normalised ETC} = \frac{\text{ETC}}{N-1},$$

where N is the initial length of the symbolic sequence we created from the original input time series that we wish to analyse.

An example would be suited best to explain the algorithm. Let's take the initial symbolic sequence of '0010011001'. Here, ETC will transform this input symbolic sequence in the following manner: $0010011001 \rightarrow 2121121 \rightarrow 3313 \rightarrow 413 \rightarrow 53 \rightarrow 6$. It took us a total of five steps to reach this stage, and thus $ETC = 5$ for this input symbolic sequence. Thus, normalised ETC $5/10 = 0.5$ as 10 was the initial length of our sequence. In this example, for the first step, the pair '00' is clearly the most frequent one and thus got replaced. Similarly, the pair '21' was replaced by '3'. But in the next step, all the pairs have equal frequency, so we simply take the first one among them to do the substitution.

ETC offers a much robust and computationally inexpensive way to discern the complexity of a time-series, making it highly advantageous in real-world applications [16].

3 Implementation Details

3.1 Dataset

We have utilised the open-access EEG Motor Movement/Imagery Dataset from the PhysioNet database [9, 19, 20]. The dataset contains one and two-minute EEG recordings from 64 channels from over 109 volunteers. We have only taken the first two recordings for each volunteer, which corresponds to the *Eyes-Open (EO)* and *Eyes-Closed (EC)* state for each volunteer. Each recording is about 60 seconds and sampled at 160 Hz, providing 9600 data points for each instance. The position of the electrodes is as displayed in Figure 1. We tested the time series for both raw and filtered cases to compare their performance. An alpha band filter for the frequency range 8 Hz-12 Hz has been applied for the filtered case, using a function from the MNE Library on Python [10, 13]. The alpha band was chosen as studies have shown a significant difference in activity in the alpha-band for EO and EC states [3].

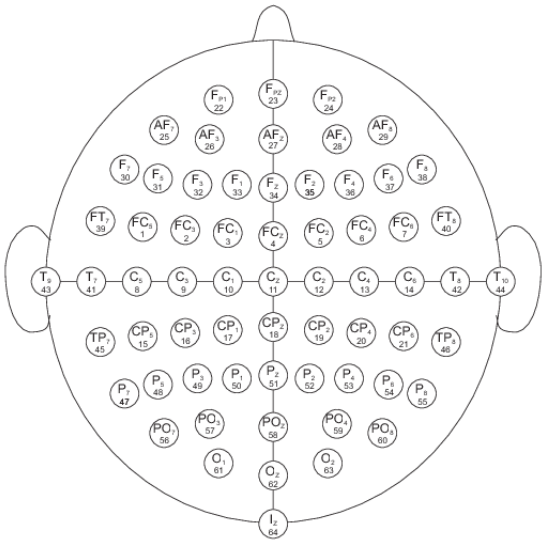


Figure 1: Position of electrodes on the scalp from Goldberger et al. [9].

3.2 Feature Extraction

There are two major ways to use ETC for feature extraction. Given a time series of length N, we take the whole time series (x) as input, and a given number of bins (b) to convert the series into a symbolic sequence with b different symbols. Then, the given symbolic sequence can be used to calculate ETC.

Inspired by the idea of ordinal symbolic analysis, we have also tried to introduce ordinal patterns in our ETC implementation. Given a time series, we slide a time window of length t across the whole time series, and we replace each of these instances of ordinal patterns with a unique symbol. There is a possibility for $t!$ different symbols for a time window of length t. This is a theoretical possibility for random or simulated data; in the case of real-world data such as this, there are far fewer different symbols. So we further partition the sequence into a new symbolic sequence following the binning strategy described earlier. Thus, we obtain our final sequence, which we can use to calculate ETC.

To summarise and list, we have used the following methods for feature extraction from the dataset:

3.2.1 ETC w/o Ordinal Analysis. For each channel, we directly calculate *ETC* on the EEG signal after binning it with b bins. This gives us a value of ETC for the specific channel. We do the same thing for all the channels and get ourselves a 64-element feature vector.

3.2.2 ETC with Ordinal Analysis. For each channel, we slide a time window of length t across the time series and obtain a symbolic sequence for each ordinal pattern. We partition the sequence further into b bins and calculate ETC on this resulting symbolic sequence. This finally gives us a 64-element feature vector, one ETC value for each channel. We can also introduce a time delay d between the selection of the next time window when creating the ordinal symbolic sequence. All three of these work as hyperparameters that we can tune for better performance of our classification algorithm.

3.3 ML Models

In this paper, we have tried some of the popular machine learning models to find out which algorithm provides us with the best result. We have tested the data with Adaptive Boosting (AdaBoost), Decision Tree (DT), Gaussian-Naive-Bayes (GNB), k-Nearest Neighbour (kNN), Logistic Regression (LR), Random Forest (RF) and Support Vector Machine (SVM) classifiers.

3.3.1 Adaptive Boosting (AdaBoost). Sequentially trains weak learners (often shallow decision trees), reweighing misclassified samples at each iteration, and combines them into a weighted sum to form a strong classifier.

3.3.2 Decision Tree (DT). Splits the data by selecting features and thresholds that maximise information gain (or minimise Gini impurity) at each node, recursively partitioning the dataset into homogeneous subsets.

3.3.3 Gaussian-Naive-Bayes (GNB). Estimates the likelihood of each feature assuming Gaussian distributions and applies Bayes' theorem with the assumption of feature independence to compute class probabilities.

3.3.4 k-Nearest Neighbour (kNN). For a new data point, calculate distances to all training samples and assign the majority class among the k closest neighbours.

3.3.5 Logistic Regression (LR). Computes a linear combination of input features and applies the sigmoid function to estimate probabilities; class labels are assigned based on a decision threshold.

3.3.6 Random Forest. Constructs an ensemble of decision trees trained on random subsets of the data and features; final predictions are made by majority voting across the trees.

3.3.7 Support Vector Machine (SVM). Finds the hyperplane that maximises the margin between classes; uses kernel functions to project data into higher dimensions for nonlinear separation when necessary.

3.4 Performance Metrics

We optimise each algorithm for all of its hyperparameters, and compare the performance using the F1 Score. We use a standard 80/20 - training/testing data split, and perform five-fold cross-validation for all machine learning algorithms. All algorithms have been implemented using the scikit-learn library on Python [18].

We further calculate Accuracy, F1 Score, Precision, and Recall values for each test to compare the performance for different sets of feature extraction and machine learning algorithms used.

Table 1: Performance of ML Algorithms on using ETC w/o Ordinal Analysis. Best results are highlighted by bold font.

Algorithm	Accuracy	F1-Score	Precision	Recall
AdaBoost	0.8181	0.8181	0.8462	0.8462
Decision Tree	0.8409	0.8303	0.8471	0.8226
GNB	0.7184	0.7183	0.7217	0.7207
kNN	**0.8409**	**0.8408**	**0.8600**	**0.8654**
LR	0.6818	0.6812	0.7232	0.7134
Random Forest	0.7727	0.7723	0.8214	0.8077
SVM	0.7727	0.7723	0.8214	0.8077

3.4.1 Accuracy. The ratio of correctly predicted samples to the total number of samples:

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

where TP, TN, FP, FN are true positives, true negatives, false positives, and false negatives respectively.

3.4.2 Precision. The proportion of correctly predicted positive samples to all predicted positives:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

3.4.3 Recall. The proportion of correctly predicted positive samples to all actual positives:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

3.4.4 F1-Score. The harmonic mean of precision and recall, balancing both metrics:

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

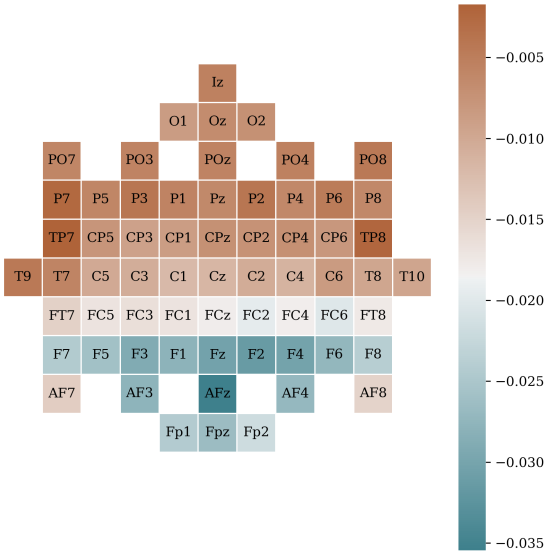
3.5 Code Availability

In order to ensure full reproducibility of our results, the complete source code, including Jupyter notebooks for data preprocessing, feature extraction, model training, and figure generation, has been made publicly available on a GitHub repository [12].

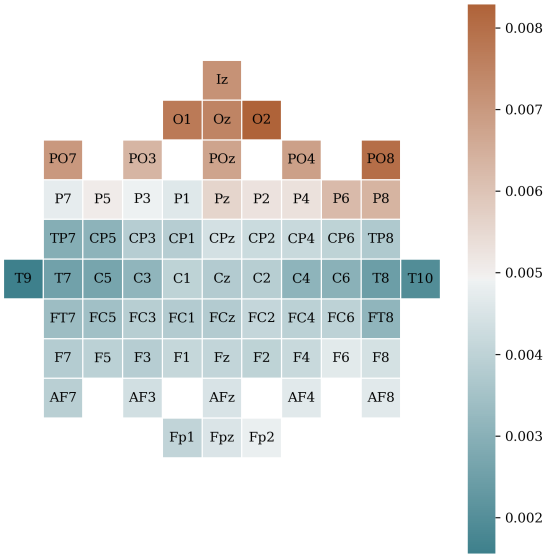
4 Experimental Results and Discussion

We start by plotting heatmaps for the average difference in ETC for EO and EC states for the unfiltered and filtered data. Figure 2b shows us the difference between the ETC measure for both states across all the channels measured through the EEG setup. We see a trend of larger ETC values for the EO state than the EC state, which conforms to the observations of studies showing *alpha synchronisation* for the EO state and *desynchronisation* for the EC state [1, 4, 14]. On average, PE also shows larger values for EO states than EC states [8]. Interestingly, in our testing, as seen in Figure 2a, the unfiltered data shows an opposite trend and has, on average, a larger value for EC state than EO state.

This trend can be better seen in the Figure 3. We do the same to compare the trends for the method ETC with Ordinal Analysis, as shown in Figure 4.



(a) $ETC_{EO} - ETC_{EC}$ for unfiltered data.



(b) $ETC_{EO} - ETC_{EC}$ for filtered data.

Figure 2: Difference in ETC across channels for Eyes-Open and Eyes-Closed states.

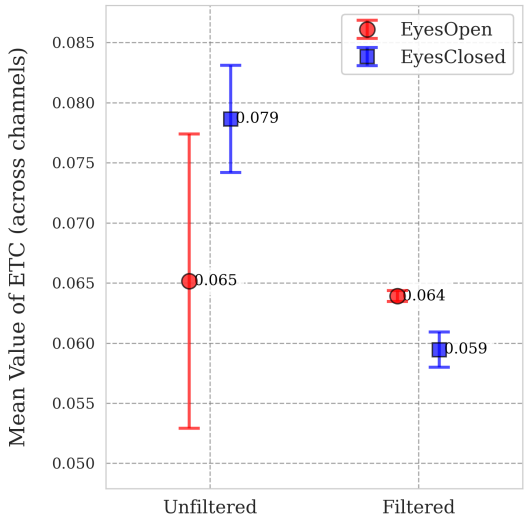


Figure 3: ETC obtained without ordinal analysis, for unfiltered and filtered data, for EO and EC states.

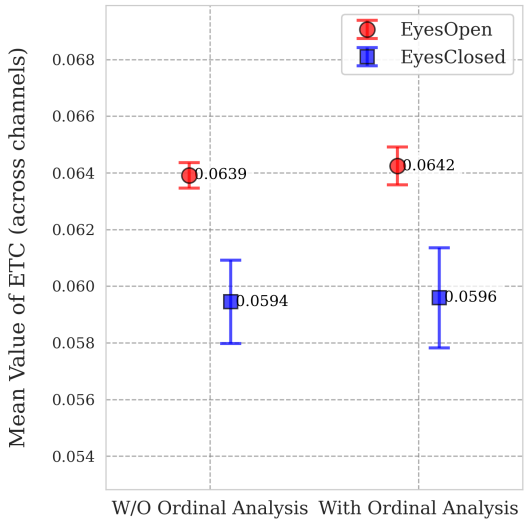


Figure 4: ETC obtained without ordinal analysis and with ordinal analysis for filtered data, for EO and EC states.

Next, using these features from the filtered data, we ran different machine learning algorithms to classify the two states. Table 1 shows us the results for different performance measures for the first method (ETC w/o Ordinal Analysis), and Table 2 shows us the results from testing the second method (ETC with Ordinal Analysis). The best results have been highlighted in bold. K-Nearest Neighbours classifier yielded the best results, while the Decision Trees classifier also outperformed other algorithms (other than kNN).

Table 3 shows us how our results compare to feature extraction attempted through Permutation Entropy in a recent study [8].

5 Discussion and Conclusions

In this work, we investigated the use of the Effort-to-Compress (ETC) ratio as a feature for classifying EEG signals between Eyes-Open (EO) and Eyes-Closed (EC) states. Our findings indicate that ETC is a highly discriminative measure, with the ordinal analysis approach to delivering the best performance. Under this configuration, Logistic Regression and Support Vector Machine classifiers

Table 2: Performance of ML Algorithms using ETC with Ordinal Analysis with $t = 3$, $b = 6$ and $d = 1$. Best results are highlighted by bold font.

Algorithm	Accuracy	F1-Score	Precision	Recall
AdaBoost	0.8409	0.8408	0.8600	0.8654
Decision Tree	0.8636	0.8611	0.8583	0.8675
GNB	0.7126	0.7096	0.7333	0.7189
kNN	0.7954	0.7953	0.8333	0.8269
LR	**0.8863**	**0.8858**	**0.8913**	**0.9038**
Random Forest	0.8181	0.8181	0.8462	0.8462
SVM	**0.8863**	**0.8858**	**0.8913**	**0.9038**

Table 3: Comparison of ETC-based Classification (this study) with Permutation Entropy Results from Gancio et al. [8].

Method	Accuracy	F1-Score	Precision	Recall
PE based [8]	0.76	0.72	0.86	0.71
ETC based	0.88	0.88	0.89	0.90

both achieved an accuracy and F1-score of approximately 88.6% (Table 2), surpassing the no ordinal analysis approach, where the best-performing model (kNN) achieved 84.1% accuracy (Table 1).

A notable observation is the contrasting behaviour of ETC on filtered versus unfiltered EEG data (Figure 2; Figure 3). For alpha-band filtered data (8–12 Hz), EO states showed higher complexity (greater ETC) than EC states. This aligns with established neurophysiological understanding: EC resting states are dominated by regular alpha oscillations, which are less complex, while EO states involve active visual processing and desynchronised neural activity, increasing signal complexity [1, 4, 14]. These results support ETC's sensitivity to biologically meaningful features of EEG.

Interestingly, this trend reversed for unfiltered data, where EC states displayed higher ETC values. This suggests that, without filtering, ETC may also respond to non-neural components such as artefacts and noise. Thus, the measure not only captures neural complexity in clean signals but may also act as an indicator of contamination in raw data.

Compared to other complexity-based approaches, ETC achieved promising results. A recent study [8] using Permutation Entropy (PE) on the same dataset reported an accuracy of about 75%. In contrast, our highest accuracy of 88.6% suggests that ETC, particularly with ordinal analysis, may capture richer discriminative information than PE for this classification task.

Overall, this study demonstrates that the Effort-to-Compress ratio is a robust and computationally efficient feature for EEG signal classification. Beyond achieving strong classification performance, our findings highlight ETC's dual sensitivity to neurophysiological dynamics and signal quality. Future work may explore its application in more complex brain–computer interface (BCI) paradigms and clinical diagnostic settings, where efficient and interpretable features remain essential.

References

[1] Edgar Douglas Adrian and Brian HC Matthews. 1934. The Berger rhythm: potential changes from the occipital lobes in man. *Brain* 57, 4 (1934), 355–385.

[2] Christoph Bandt and Bernd Pompe. 2002. Permutation entropy: a natural complexity measure for time series. *Physical review letters* 88, 17 (2002), 174102.

[3] Robert J Barry, Adam R Clarke, Stuart J Johnstone, Christopher A Magee, and Jacqueline A Rushby. 2007. EEG differences between eyes-closed and eyes-open resting conditions. *Clinical neurophysiology* 118, 12 (2007), 2765–2773.

[4] Hans Berger. 1929. Über das elektroenkephalogramm des menschen. *Archiv für psychiatrie und nervenkrankheiten* 87, 1 (1929), 527–570.

[5] Chunhua Bian, Chang Qin, Qianli DY Ma, and Qinghong Shen. 2012. Modified permutation-entropy analysis of heartbeat dynamics. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* 85, 2 (2012), 021906.

[6] Zehong Cao. 2020. A review of artificial intelligence for EEG-based brain- computer interfaces and applications. *Brain Science Advances* 6, 3 (2020), 162–170.

[7] Werner Ebeling and Miguel A Jiménez-Montaño. 1980. On grammars, complexity, and information measures of biological macromolecules. *Mathematical Biosciences* 52, 1 (1980), 53–71.

[8] Juan Gancio, Cristina Masoller, and Giulio Tirabassi. 2024. Permutation entropy analysis of EEG signals for distinguishing eyes-open and eyes-closed brain states: Comparison of different approaches. *Chaos: An Interdisciplinary Journal of Nonlinear Science* 34, 4 (2024), 043130.

[9] Ary L Goldberger, Luis AN Amaral, Leon Glass, Jeffrey M Hausdorff, Plamen Ch Ivanov, Roger G Mark, Joseph E Mietus, George B Moody, Chung-Kang Peng, and H Eugene Stanley. 2000. PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals. *circulation* 101, 23 (2000), e215–e220.

[10] Alexandre Gramfort, Martin Luessi, Eric Larson, Denis A. Engemann, Daniel Strohmeier, Christian Brodbeck, Roman Goj, Mainak Jas, Teon Brooks, Lauri Parkkonen, and Matti S. Hämäläinen. 2013. MEG and EEG Data Analysis with MNE-Python. *Frontiers in Neuroscience* 7, 267 (2013), 1–13. doi:10.3389/fnins.2013.00267

[11] Jing Hu, Jianbo Gao, and Jose C Principe. 2006. Analysis of biomedical signals by the Lempel-Ziv complexity: the effect of finite data size. *IEEE Transactions on Biomedical Engineering* 53, 12 (2006), 2606–2609.

[12] Yojit Kumar. 2025. *EEG Classification Using ETC.* https://github.com/yojit-kumar/eeg-classification-using-etc

[13] Eric Larson, Alexandre Gramfort, et al. 2025. *MNE-Python.* doi:10.5281/zenodo.15928841

[14] H Legewie, O Simonova, and OD Creutzfeldt. 1969. EEG changes during performance of various tasks under open-and closed-eyed conditions. *Electroencephalography and Clinical Neurophysiology* 27, 5 (1969), 470–479.

[15] Abraham Lempel and Jacob Ziv. 2003. On the complexity of finite sequences. *IEEE Transactions on information theory* 22, 1 (2003), 75–81.

[16] Nithin Nagaraj and Karthi Balasubramanian. 2017. Dynamical complexity of short and noisy time series: Compression-Complexity vs. Shannon entropy. *The European physical journal special topics* 226, 10 (2017), 2191–2204.

[17] Nithin Nagaraj, Karthi Balasubramanian, and Sutirth Dey. 2013. A new complexity measure for time series analysis and classification. *The European Physical Journal Special Topics* 222, 3 (2013), 847–860.

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[19] Gerwin Schalk, Dennis J McFarland, Thilo Hinterberger, Niels Birbaumer, and Jonathan R Wolpaw. 2004. BCI2000: a general-purpose brain-computer interface (BCI) system. *IEEE Transactions on biomedical engineering* 51, 6 (2004), 1034–1043.

[20] Gerwin Schalk, Dennis J McFarland, Thilo Hinterberger, Niels Birbaumer, and Jonathan R Wolpaw. 2009. *EEG Motor Movement/Imagery Dataset.* https://physionet.org/content/eegmmidb/1.0.0/

[21] Anupreet Kaur Singh and Sridhar Krishnan. 2023. Trends in EEG signal feature extraction applications. *Frontiers in artificial intelligence* 5 (2023), 1072801.

[22] Joshi Vandana and Nanavati Nirali. 2021. A review of EEG signal analysis for diagnosis of neurological disorders using machine learning. *Journal of Biomedical Photonics & Engineering* 7, 4 (2021), 40201.

A Hyperparameter Settings

In this appendix, we report the hyperparameter settings that were obtained for each of the ML classifiers via a 5-fold crossvalidation strategy for ETC with and without ordinal analysis.

Table 4: Optimal hyperparameter settings for this study.

Component	Parameter	Description	Search Space	Optimal value
Feature Extraction				
ETC w/o Ordinal Analysis	`BINS` b	Number of discrete bins for quantizing signal amplitudes before computing ETC. Higher b yields finer resolution.	{2,3,4,5}	4
Classifiers				
AdaBoost	`n_estimators`	Number of weak learners sequentially added to the ensemble.	{50,100,200,300}	50
Decision Tree	`msl_values`	Minimum number of training samples required at a leaf node.	{2,5,10,15}	5
	`md_values`	Maximum depth (levels) of the decision tree.	{2,3,4,5}	3
kNN	`k_values`	Number of nearest neighbours used for classification.	{3,5,7,9,11}	3
LR	`C_values`	Inverse of regularization strength; smaller values mean stronger regularization.	{0.001,0.01,0.1,10,100,1000}	10
	`penalty`	Regularization norm applied to model coefficients.	l1	l1
	`solver`	Algorithm for optimization (e.g., `liblinear`).	liblinear	liblinear
Random Forest	`n_estimators`	Total number of decision trees in the ensemble.	{100,200,300,500}	100
	`msl_values`	Minimum number of training samples required at a leaf node.	{3,5,10,None}	3
	`md_values`	Maximum depth (levels) of the decision trees.	{1,2,4}	2
SVM	`C_values`	Regularization parameter; controls margin-width trade-off.	{0.1,1,10,100}	10
Feature Extraction				
ETC with Ordinal Analysis	`TIME_WINDOW` t	Length of the time window over which ETC is computed.	{3,4,5,6,7}	3
	`DELAY` d	Embedding time lag between points for ordinal patterns.	{1,2}	1
	`BINS` b	Number of discrete bins for signal quantization.	{2,3,4,5}	4
Classifiers				
AdaBoost	`n_estimators`	Number of weak learners sequentially added to the ensemble.	{50,100,200,300}	100
Decision Tree	`msl_values`	Minimum number of training samples required at a leaf node.	{2,5,10,15}	10
	`md_values`	Maximum depth (levels) of the decision tree.	{2,3,4,5}	4
kNN	`k_values`	Number of nearest neighbours used for classification.	{3,5,7,9,11}	5
LR	`C_values`	Inverse of regularization strength; smaller values mean stronger regularization.	{0.001,0.01,0.1,10,100,1000}	10
	`penalty`	Regularization norm applied to model coefficients.	l1	l1
	`solver`	Algorithm for optimization (e.g., `liblinear`).	liblinear	liblinear
Random Forest	`n_estimators`	Total number of decision trees in the ensemble.	{100,200,300,500}	200
	`msl_values`	Minimum number of training samples required at a leaf node.	{3,5,10,None}	5
	`md_values`	Maximum depth (levels) of the decision trees.	{1,2,4}	2
SVM	`C_values`	Regularization parameter; controls margin-width trade-off.	{0.1,1,10,100}	1