Models and Optimisers Evaluation for Adolescent Depression via ML

Myat Pwint Phyu

Student ID: 33356017

Course : FIT5127

Supervisor : Dr Cheng Siong Lee

A thesis submitted for the degree of

Master of Artificial Intelligence

at Monash University in May 2025

Faculty: Information Technology,

Melbourne Australia

May 31, 2025

CONTENTS

LIST OF FIGURES

List of Tables

Models and Optimisers Evaluation for Adolescent Depression via ML

Myat Piwnt Phyu and Cheng Siong Lee

***Abstract*— Depression is a leading psychological condition challenge worldwide, and adolescents are especially at risk due to ongoing changes in brain development and social environment. Electroencephalography (EEG), which captures brain activity through non-invasive scalp electrodes, offers a practical and precise way to monitor neural patterns and has shown potential for identifying signs of depression.**

Aiming to facilitate early mental health intervention, this study proposes an EEG-based automated system targeting adolescents aged 18-19. A publicly available EEG dataset was filtered by age, and signals were preprocessed to extract four distinct feature sets: (i) temporal features via 1D-CNN, (ii) spatiotemporal features via 3D-CNN applied to spectrograms, (iii) spectral features from frequency-domain analysis, and (iv) beta-band power features associated with cognitive-affective function.

Comparative analysis was performed using multiple classification techniques, including XGBoost, Naive Bayes, Random Forest, Logistic Regression, K-Nearest Neighbours and Multilayer Perceptron, over all feature categories. XGBoost consistently outperformed others and was selected for further enhancement via hyperparameter optimisation. Five metaheuristic algorithms were employed: EJHO, ESAO, EIWO-SA, ECSO, and COIWSO-SA. Optimisation was guided by a custom multi-objective cost function incorporating accuracy, MAP, MCC, precision, FDR, and FPR.

Among all methods, EIWO-SA yielded the most stable convergence and highest performance, demonstrating its robustness for EEG-based depression classification. The proposed framework offers a reproducible, optimiser-agnostic pipeline for adolescent mental health screening using EEG and machine learning.

***Index Terms*— Electroencephalography (EEG), Depression Detection, Adolescent Mental Health, Beta Band, Spectral Features, XGBoost, Metaheuristic Optimisation, Hyperparameter Tuning, Multi-Objective Cost Function**

1. Introduction

A. Background

DEPRESSION is a primary global health concern, affecting approximately 280 million people worldwide [1]. According to projections, depression was expected to rank as the second most significant cause of disability globally by the year 2020 [2]. While traditionally associated with adults, it is increasingly recognised as a serious issue among adolescents. The World Health Organisation reports that approximately one in seven individuals between the ages of 10 and 19 is affected by a mental health condition, with depression impacting around 3.9% of adolescents worldwide [1].

Mental health issues during adolescence can significantly impact emotional development, academic performance, and long-term well-being [3]. Contributing factors include genetic vulnerability, hormonal changes, chronic stress, and adverse social environments [3]. Early detection is, therefore, crucial.

Conventional diagnostic methods frequently depend on individuals' self-reported symptoms, which can be subjective and susceptible to underreporting. Consequently, neurophysiological methods like electroencephalography (EEG), known for their non-invasiveness and high temporal accuracy, have gained increasing attention. EEG has gained recognition as a valuable method for detecting neural patterns associated with depression, supporting the development of automated and objective tools for diagnosing mental health conditions in adolescents.

B. Challenges in Diagnosing Adolescent Depression and Technological Approaches

Diagnosing depression during adolescence presents a unique clinical challenge due to the psychological and neurobiological fluctuations that occur during this developmental period [4]. The emotional and mental changes that occur during adolescence can make depressive symptoms harder to spot. Since no two adolescents are the same, treatment must be adapted to their personality and developmental needs.

Despite the importance of early detection, diagnostic procedures are limited by the lack of objective and reliable biomarkers. Adolescents at low risk often go undetected due to insufficient case representation in clinical datasets, increasing the likelihood of underdiagnosis or misdiagnosis [5]. Compounding this issue, traditional psychiatric evaluations rely heavily on subjective reports and clinician interpretation, both of which are susceptible to cognitive bias and situational variability.

Because of the challenges in current diagnostic approaches, researchers are exploring tools like EEG, MRI, fMRI, and ECG to find biological indicators of depression. The real-time tracking capability of EEG enhances its utility in observing dynamic brain processes. Compared to traditional methods, these techniques allow for faster and more accurate assessments with less dependence on subjective interpretation.

C. Research Aim and Objectives

This research aims to design an EEG-based binary classification framework for detecting Depression in adolescents aged 18–19, using BDI (Beck Depression Inventory) scores

to generate class labels. Participants scoring above a defined threshold were classified as " depressed " while those below were classified as "non-depressed."

The study proceeds in three stages. Initially, various machine learning algorithms, including NB, XGBoost, LR, RF, KNN, and MLP, were evaluated across four EEG feature sets to identify the most effective model-feature combination. XGBoost, which consistently achieved superior results, was selected for further optimisation.

In the second stage, the XGBoost classifier was trained using four distinct EEG feature extraction techniques: 1D-CNN-based temporal features, 3D-CNN-based spatiotemporal features derived from spectrograms, frequency-domain spectral features, and beta-band power features. Each feature set was evaluated in combination with several optimisation algorithms.

In the third stage, five metaheuristic optimisers: Elitist Jellyfish Optimisation (EJHO), Elitist Simulated Annealing Optimisation (ESAO), Elitist Invasive Weed Optimisation Simulated Annealing (EIWO-SA), Elitist Crisscross Mutation Optimisation (ECSO), and the hybrid Elitist COIWSO-SA were employed to optimise the hyperparameters of the XGBoost algorithm. Each optimiser-feature configuration was evaluated using a comprehensive cost function. The function integrated several key metrics, including classification accuracy, MAP, correlation strength via MCC, the correctness of positive predictions (precision), and error rates such as FDR and FPR.

2. Literature Review

A. Role of EEG in Depression Diagnosis

Electroencephalography (EEG) is a widely adopted neurophysiological method for evaluating brain activity. It enables the analysis of hyperactivity and hypoactivity in various cortical regions by capturing the brain's electrical signals through surface electrodes [6]. EEG signals are typically segmented into distinct frequency bands—delta (ranging from 0.5 to 4 Hz), theta (4 to 8 Hz), alpha (8 to 12 Hz), beta (12 to 30 Hz), and gamma (above 30 Hz)—each of which is associated with specific cognitive functions and brain activity patterns [7].

Among these, the beta band is particularly relevant to depression research. Beta activity is associated with alertness, cognitive control, and emotional regulation. Empirical evidence suggests that depressive disorders are associated with altered beta-band EEG activity when compared to control participants. Nofzinger et al. [8] reported that elevated beta EEG power during NREM sleep was linked to altered cerebral glucose metabolism in brain regions associated with dysfunctional arousal in depressive patients. Similarly, Grin-Yatsenko et al. [9] observed early-stage beta-band abnormalities in individuals with depression, further supporting the diagnostic value of this frequency range.

In addition to isolated beta-band analysis, spectral feature extraction methods such as power spectral density (PSD) and Short-Time Fourier Transform (STFT) provide a broader frequency-domain representation of EEG data. These methods allow for the decomposition of EEG signals across all frequency bands, aiding in identifying oscillatory patterns associated with depressive states. Dutta et al. [10] highlighted the role of spectral features in cognitive state classification, underscoring their relevance in mental health applications. Hinrikus et al. [11] further demonstrated that spectral features differ significantly between depressed and control subjects, particularly within the beta and alpha bands.

Beyond conventional signal processing techniques, deep learning models have emerged as powerful automated EEG feature extraction tools. One-dimensional convolutional neural networks (1D-CNNs) effectively capture local temporal patterns from raw EEG time series [12]. These models apply convolutional filters over the temporal domain, enabling them to learn depression-relevant features without manual intervention. In contrast, three-dimensional CNNs (3D-CNNs), often applied to EEG spectrograms generated via STFT, are capable of learning rich spatiotemporal representations by simultaneously capturing spatial, frequency, and temporal information [12], [13].

In summary, EEG offers multiple avenues for extracting informative features related to depression. Beta-band analysis provides a physiologically interpretable biomarker, spectral features quantify frequency-domain abnormalities, and deep learning architectures such as 1D-CNNs and 3D-CNNs enable the automated discovery of discriminative patterns. Collectively, these complementary approaches establish a strong foundation for EEG-based depression diagnosis.

B. A Comparative Study of EEG Characteristics in Different Age Populations

EEG signals are known to vary with both age and neurological conditions. As individuals age, they become more susceptible to brain-related disorders, which can lead to noticeable changes in their EEG activity. For instance, synchronisation likelihood has been applied in the detection of Alzheimer's disease [14]. Alyelyani et al. [15] also noted that age and illness both significantly influence EEG patterns. Prior studies have shown that developmental changes, including shifts in oscillatory behaviour, occur across different age groups [16]–[20]. This section reviews a selection of studies that examine the impact of age and disease on EEG signals, with a particular focus on adolescent and adult populations. A detailed comparison is provided in Table I.

C. Comparative Analysis of EEG-Based Depression Classification Models

Predicting depression using EEG data is a well-recognised challenge due to the complexity of brain signals and the variability across individuals. Several studies have investigated this by applying different machine-learning models and feature extraction methods. In this section, we compare a range of existing works that focus on EEG-based classification of depression. Table II outlines the models used, the features considered, and the accuracy or performance metrics reported in each study, helping to highlight what approaches have shown the most promise and where limitations remain.

TABLE I: Comparison of EEG Characteristics Across Age Groups and Neurological Conditions

Author	Age Range	Condition	EEG Frequency Domain Observations
Stam et al. (2005) [14]	–	Alzheimer's Disease	Reported an increase in synchronisation likelihood, indicating disrupted functional connectivity in patients
Barry et al. (2004) [21]	8–9 years	Healthy	Observed a slight decrease in EEG coherence in male participants
Barry et al. (2004) [21]	9–12 years	Healthy	Found a mild increase in coherence among older male children
Barry et al. (2004) [21]	–	Healthy	Noted age-related improvements in alpha band coherence
Benninger et al. (1984) [22]	4–17 years	Healthy	Identified increasing theta activity and decreasing delta power as children aged
Gasser et al. (1988) [23]	6–17 years	Healthy	Demonstrated that higher-frequency bands strengthened with age, while slower bands declined
Gasser et al. (1988) [23]	Adolescents	Healthy	Showed a general increase in absolute power across all bands except alpha-2
Kikuchi et al. (2000) [24]	–	Healthy	Found that younger individuals exhibited lower coherence in alpha-3, theta, beta-1, and beta-2 bands
Saletu et al. (1995) [25]	–	Multi-infarct Dementia	Reported enhanced theta activity and reduced alpha amplitude in occipital brain regions

D. Problem Statement

The detection of depression in adolescents remains a significant clinical challenge due to developmental variability and the limitations of subjective diagnostic approaches. Electroencephalography (EEG) has gained increasing attention as a non-invasive tool capable of identifying neural biomarkers associated with depressive states [1]. Recent advances have demonstrated the utility of deep learning models applied to EEG signals for mental health assessment. One such example, EEGDepressionNet, proposes a self-attention-based gated DenseNet architecture optimised through hybrid metaheuristic strategie [13] . While effective, the model's reliance on a deep end-to-end neural architecture constrains interpretability, and its feature representation is tightly coupled to a single model design, limiting modularity and generalizability.

Moreover, EEGDepressionNet does not explicitly target a tightly defined adolescent sub-cohort despite known neurodevelopmental differences that influence EEG signatures in this age group [16]–[20]. In contrast, our study focuses on adolescents aged 18–19 and proposes a transparent, optimiser-agnostic classification pipeline using four distinct EEG feature categories: temporal (via 1D-CNN), spatiotemporal (via 3D-CNN and spectrograms), frequency-domain spectral features, and beta-band power each shown to correlate with cognitive-affective functioning [6], [7]. We conduct a comparative evaluation across several conventional machine learning models and systematically optimise the top-performing classifier, XGBoost, using five elite metaheuristic algorithms. The model is guided by a custom multi-objective cost function that integrates clinical and statistical performance metrics, including accuracy, MCC, FDR, and FPR.

Unlike prior end-to-end deep models, our approach separates feature extraction and classification, enabling more control over model behaviour and easier adaptation across clinical settings. In contrast to prior deep learning approaches, this study targets a clearly defined adolescent group and builds a classification pipeline using distinct EEG feature types such as beta-band power and spectrogram-based spatiotemporal patterns combined with metaheuristic-based optimisation. This structure improves both interpretability and adaptability for future clinical use.

3. EEG-Based Depression Analysis Using Machine Learning Techniques

A. EEG Datasets Used for Depression Analysis

This study used EEG recordings sourced from the publicly accessible dataset "EEG: Probabilistic Selection and Depression" [37], which contains data from around 122 individuals of college age. Participants were stratified based on their Beck Depression Inventory (BDI) scores. To align with the age-specific focus of this research, only individuals aged 18 or 19 were included, yielding a final sample of 96 participants. Among them, 56 were identified as non-depressed. This dataset has been validated and employed in prior peer-reviewed studies. It contains only pre-processed EEG signals, as the original raw data is inaccessible. Representative EEG signal samples from this dataset are presented in Table III.

B. Proposed Depression Analysis with Machine Learning

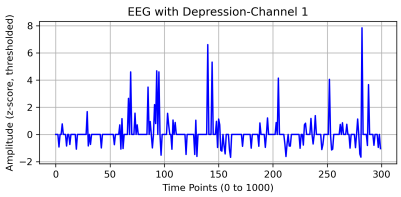
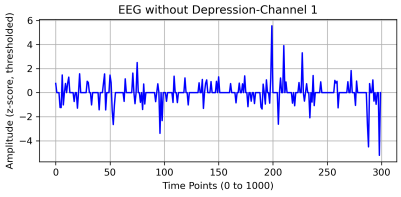
Depression is a multifactorial mental health disorder influenced by genetic, environmental, and psychosocial factors. Accurate diagnosis remains a significant clinical challenge, especially among adolescents, due to symptom variability and the subjective nature of emotional assessments. As Depression primarily affects cognitive and affective states, electroencephalogram (EEG) signals, which reflect neural electrical activity, have been widely adopted in recent research for automated classification. EEG techniques offer real-time, non-invasive brain monitoring and are well-suited for examining emotional and cognitive processes, owing to their satisfactory temporal resolution and data privacy advantages.

This research introduces a machine learning framework designed to perform binary classification of depression, distinguishing between depressed and non-depressed individuals using EEG data. The recommended model utilises the Extreme Gradient Boosting (XGBoost) algorithm, enhanced through metaheuristic-based hyperparameter tuning to improve classification accuracy. In figure 1, the proposed framework is illustrated with two optimisers based on their performance for various feature types.

TABLE II: Comparison for Training Models on EEG Data

Author	Classifier	Fold	Purpose	Accuracy	Sensitivity
Mumtaz et al. (2018) [26]	LR	10	Relationship between selected features and treatment outcomes	91.7%	86.66%
Mumtaz et al. (2018) [26]	SVM	10	Model comparison	98%	99.9%
Mumtaz et al. (2018) [26]	NB	10	Posterior probability modeling	93.6%	100%
Akbari et al. (2021) [27]	SVM	10	Identify normal vs. depression signals	93.6%	100%
Akbari et al. (2021) [27]	KNN	10	Identify normal vs. depression signals	98.79%	100%
Zhu et al. (2020) [28]	LR	10	EEG classification in resting state	85.18%	-
Zhu et al. (2020) [28]	RF	10	EEG classification in resting state	90.56%	-
Zhu et al. (2020) [28]	NB	10	EEG classification in resting state	81.21%	-
Zhu et al. (2020) [28]	KNN	10	EEG classification in resting state	78.37%	-
Kaur et al. (2021) [29]	RF	-	EEG classification with EMD-DFA-WPD	98.51%	-
Kaur et al. (2021) [29]	SVM	-	EEG classification with EMD-DFA-WPD	98.07%	-
Cai et al. (2018) [30]	SVM	10	Classification of resting state data	72.56%	-
Cai et al. (2018) [30]	KNN	10	EEG under audio stimulation (theta & beta)	79.27%	-
Bachmann et al. (2018) [31]	LR	-	Linear vs. non-linear depression classification	92%	-
Mahato et al. (2020) [32]	SVM	10	Using alpha-2 band (best accuracy)	88.33%	90.81%
Mahato et al. (2020) [32]	NB	10	Alpha-2 band	86.63%	86.63%
Mahato et al. (2020) [32]	LR	10	Alpha-2 band	85.82%	82.07%
Saeedi et al. (2020) [33]	SVM	10	Gamma wave classification	87.93%	-
Saeedi et al. (2020) [33]	KNN	10	Gamma wave classification	86.27%	-
Saeedi et al. (2020) [33]	MLP	10	Gamma wave classification	82.67%	-
Acharya et al. (2018) [34]	CNN	10	EEG from left hemisphere	93.5%	-
Acharya et al. (2018) [34]	CNN	10	EEG from right hemisphere	96.0%	-
Li et al. (2019) [35]	CNN	7	EEG using alpha wave	84.75%	-
Li et al. (2019) [35]	CNN	8	EEG using alpha wave	83.35%	-
Sarkar et al. (2022) [36]	CNN	-	EEG with 20% test set	59.25%	-
Sarkar et al. (2022) [36]	MLP	-	EEG with 20% test set	59.25%	-
Sarkar et al. (2022) [36]	LR	-	EEG with 20% test set	97.18%	-
Sarkar et al. (2022) [36]	SVM	-	EEG with 20% test set	97.65%	-

TABLE III: Comparison of EEG signals with and without Depression

Signal Description	EEG Signal
With Depression	
Without Depression	

1) EEG Signal Preprocessing via Bandpass Filtering: This study applies a bandpass filtering method to suppress low-frequency drift typically associated with sweat artifacts, slow movement, and high-frequency noise from muscle activity and electronic interference [38]. This preprocessing stage is crucial for increasing the signal-to-noise ratio in EEG data, which in turn enhances the accuracy and consistency of subsequent feature extraction and classification processes.

2) EEG Feature Extraction: To represent both temporal and frequency domain characteristics, EG signal processing involves four independent feature extraction strategies to capture various data characteristics:

1) **3D-CNN Features:** Raw EEG signals are segmented into overlapping epochs and reshaped into 3D tensors. The resulting data are subsequently processed using a 3D CNN, which extracts spatiotemporal features reflecting dynamic patterns across both time and EEG channels.
2) **1D-CNN Features:** Raw EEG epochs are processed using a 1D Convolutional Neural Network, enabling the model to learn temporal patterns and waveform variations across channels, which are then compressed into informative feature vectors via global average pooling.
3) **Spectral Features:** EEG data are divided into overlapping two-second segments, and the STFT is applied to each channel to calculate the mean spectral power across conventional frequency bands, thereby capturing the time-frequency characteristics relevant to depression classification.
4) **Beta-Band Features:** EEG data are partitioned into overlapping two-second windows, and STFT is employed to compute average spectral power in the beta band (12–30 Hz) across all channels, enabling the extraction of frequency-specific patterns linked to mental states.

3) Hyperparameter Optimisation: For optimal predictive performance, five state-of-the-art metaheuristic strategies are applied to optimise the key configuration settings of the XGBoost classifier.

- The step size used during training (commonly referred to as the learning rate)
- The deepest level a tree is permitted to grow during training
- The total number of trees (estimators) in the ensemble
- The proportion of data used per tree (row sampling)
- The fraction of features considered when splitting at each tree node (column sampling)

The following optimisation algorithms are employed:

- Elitist Jellyfish-Based Hyperparameter Optimiser (EJHO)
- Elitist Simulated Annealing-based Hyperparameter Optimiser (ESAO)
- Elitist Invasive Weed Optimisation with Simulated Annealing Logic (EIWO-SA)
- Elitist Crisscross Mutation Optimiser (ECSO)
- Elitist Hybrid COS-IWO-Based Optimiser (COIWSO-SA):

Each algorithm independently explores the hyperparameter space to identify optimal configurations. The final model selection is based on performance across standard classification metrics, ensuring the best trade-off between accuracy and generalisation.

C. Hyperparameter Optimisation Using Metaheuristic Algorithms for XGBoost-Based Depression Detection

This study optimises five critical hyperparameters of the XGBoost classifier to enhance EEG-based depression detection: learning rate, maximum tree depth, number of estimators, subsample ratio, and column sampling ratio by tree. These parameters significantly influence the model's convergence rate, learning capacity, and generalisation performance.

The optimisation process begins by defining lower and upper bounds for each hyperparameter, forming a constrained search space. An initial population of candidate configurations is generated via uniform random sampling. Each $\mathbf{x}_i$ in this population encodes a unique set of hyperparameter values.

For every candidate $\mathbf{x}_i$, an XGBoost model is trained on the training set $(X_{\text{train}}, y_{\text{train}})$ and validated on testing set $(X_{\text{test}}, y_{\text{test}})$. Classification performance is assessed using six key metrics: accuracy (Acc), MAP, MCC, precision (Pre), FDR, and FPR.

A custom cost function (CF) is utilised to evaluate solution quality:

$$\text{CF} = \frac{1}{\text{Acc} + 0.5 \cdot \text{MAP} + 0.3 \cdot \text{MCC} + 0.3 \cdot \text{Pre} + \epsilon} + 0.3 \cdot (\text{FDR} + \text{FPR}) \tag{1}$$

In (1), $\epsilon = 10^{-8}$ is included to prevent division by zero. The objective is to minimise CF, thus promoting solutions with high predictive performance and low error rates.

Each iteration involves selecting the top individual based on the cost metric, updating the archive, and evolving the population via metaheuristic-defined search dynamics. The algorithm proceeds through successive iterations until it meets the specified maximum iteration limit. Each candidate solution represents a distinct set of hyperparameters defined in Equation (2).

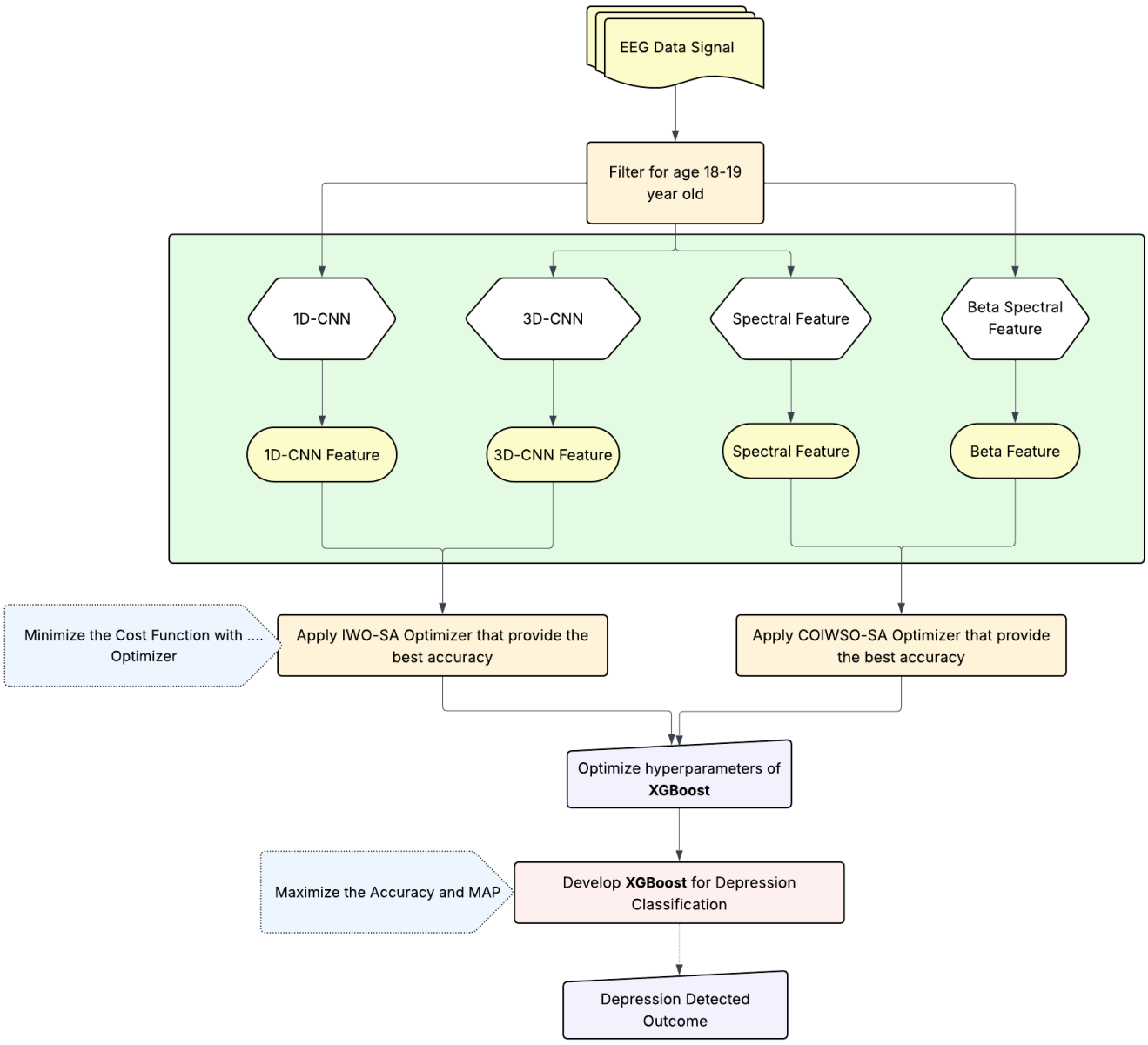


Fig. 1: Architecture of the Optimiser-Guided EEG-Based Depression Detection Framework

$$\mathbf{x}_i = [\text{learning\_rate}, \text{max\_depth}, \text{n\_estimators}, \text{subsample}, \text{colsample\_bytree}] \tag{2}$$

The final output includes the optimal hyperparameter configuration, the trained model instance, and a historical cost and accuracy evolution trace. The optimisation workflow is summarised in Fig. 2, ensuring a consistent and comparable evaluation across all metaheuristic algorithms.

1) Elitist Jellyfish-Based Hyperparameter Optimiser (EJHO): EHJO is an adapted variant of the Jellyfish Search algorithm, specifically designed to minimise CF and optimise hyperparameters in the XGBoost model. Inspired by the Modified Jellyfish Search (MJS) algorithm proposed by Yildizdan et al. [39], EHJO simplifies the original temperature-based and peer following mechanisms by favouring deterministic local search updates and strict elitism. This design choice enhances convergence stability and improves the effectiveness of hyperparameter tuning.

The complete optimisation procedure of EHJO is outlined in Algorithm 1. The algorithm begins with uniform sampling of an initial population $\mathcal{P}$ within the defined parameter bounds. The current best solution $\mathbf{x}^*$ is retained unchanged at each iteration. For all other individuals, localised updates are applied to each hyperparameter dimension based on its type.

For continuous parameters, the update is performed by adding Gaussian noise, promoting local exploration around the current solution in Equation 3:

$$x_i' = x_i + \mathcal{N}(0, \sigma^2) \tag{3}$$

In contrast, discrete hyperparameters like `max_depth` and

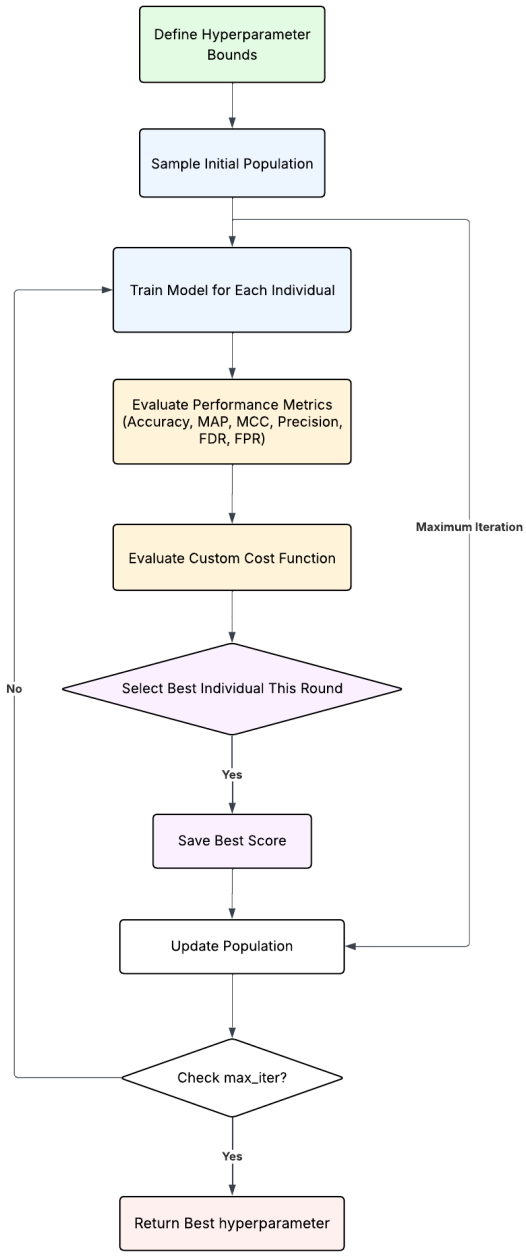


Fig. 2: Workflow of Metaheuristic Hyperparameter Optimisation for XGBoost-Based Depression Detection

`n_estimators` are updated using small integer shifts drawn from a predefined neighbourhood, as shown in Equation 4:

$$x_i' = x_i + \Delta, \quad \Delta \in \{-2, -1, +1, +2\} \tag{4}$$

After the mutation step, all updated values are clipped to remain within their specified parameter bounds in Equation 5:

$$x_i' = \min(\max(x_i', \text{lb}), \text{ub}) \tag{5}$$

Each perturbed solution $\mathbf{x}_i'$ is then evaluated using the objective function. If an improvement is observed, the new solution replaces the original. The global best $\mathbf{x}^*$ is updated if necessary. Unlike the original MJS, EHJO relies on a purely deterministic, elitist update scheme, which leads to faster convergence and more robust performance, especially in structured optimisation tasks such as EEG-based depression classification.

Algorithm 1 Elitist Jellyfish-Based Hyperparameter Optimiser (EJHO)

Input: Bounds $[\text{lb}, \text{ub}]$, population size N, max iterations T
Initialise population $\mathcal{P}$ of N individuals, evaluate fitness, set elite $\mathbf{x}^*$
for $t = 1$ to T **do**
  for each $\mathbf{x}_i \in \mathcal{P}$, excluding $\mathbf{x}^*$ **do**
    for each parameter $x_{i,k}$ **do**
      Apply mutation using Eq. (3) or Eq. (4) based on parameter type
      Clip $x'_{i,k} \leftarrow \min(\max(x'_{i,k}, \text{lb}), \text{ub})$ *(Eq. 5)*
    end for
    Evaluate $\mathbf{x}_i'$, replace if better: $\mathbf{x}_i \leftarrow \mathbf{x}_i'$
  end for
  Update $\mathbf{x}^*$ if a better solution is found
end for
Return: Best solution $\mathbf{x}^*$

2) Elitist Simulated Annealing-based Hyperparameter Optimiser (ESAO): ESAO is a lightweight variant of the Multi-Objective Simulated Annealing (MOSA) algorithm [40], designed for supervised model hyperparameter tuning. Unlike MOSA, ESAO simplifies optimisation by collapsing multiple evaluation metrics into a single scalar cost function, removing temperature-based acceptance logic, and enforcing strict elitism.

Each solution $\mathbf{x}_i$ represents a hyperparameter vector in Equation (2). As outlined in Algorithm 2, the optimiser first evaluates all individuals in the population, retains the best-performing solution $\mathbf{x}^*$, and applies local perturbations guided by $\mathbf{x}^*$ using the update rules defined in Equation (6). This formulation supports both continuous and discrete parameters through a combination of scaled drift and stochastic noise. Candidate solutions are accepted only if they yield improved performance compared to their predecessors.

By replacing probabilistic acceptance with deterministic elitism, ESAO achieves stable convergence and improved computational efficiency while preserving the fundamental principle of simulated annealing—localised, stochastic neighbourhood exploration.

$$x'_{i,k} = \begin{cases} \text{clip}\left(x_{i,k} + \text{int}(0.3(x^*_k - x_{i,k}) + \mathcal{U}(-3, 3))\right), & \text{if integer type} \\ \text{clip}\left(x_{i,k} + 0.3(x^*_k - x_{i,k}) + \mathcal{N}(0, 0.05)\right), & \text{otherwise} \end{cases} \tag{6}$$

This formulation supports both integer and continuous parameters, utilising a scaled drift toward the best-known solution combined with random noise. Mutated individuals are retained only if they improve performance, and $\mathbf{x}^*$ is updated accordingly.

Compared to MOSA, ESAO omits temperature scheduling and probabilistic acceptance of worse solutions. Instead, it

adopts a greedy, elitist strategy prioritising efficiency and convergence stability. This simplification makes ESAO more practical for structured model optimisation tasks, while preserving the core advantage of SA: directed stochastic neighbourhood search.

Algorithm 2 Elitist Simulated Annealing-Based Hyperparameter Optimiser (ESAO)

Initialise population $\mathcal{P}$ of size N with random hyperparameters
Evaluate all individuals and identify best $\mathbf{x}^*$
for each iteration **do**
  for each $\mathbf{x}_i \in \mathcal{P}$, excluding elite $\mathbf{x}^*$ **do**
    for each hyperparameter dimension k **do**
      Update $x'_{i,k}$ using Equation (6)
    end for
    Clip each $x'_{i,k}$ to bounds
    Replace $\mathbf{x}_i$ with $\mathbf{x}'_i$ if performance improves
  end for
  Update $\mathbf{x}^*$ if better solution is found
end for
Return best solution $\mathbf{x}^*$

3) Elitist Invasive Weed Optimisation with Simulated Annealing Logic (EIWO-SA): The Elitist Invasive Weed Optimisation with Simulated Annealing (EIWO-SA) is a hybrid metaheuristic developed for hyperparameter optimisation in structured learning models like XGBoost. This optimiser builds upon the classical Invasive Weed Optimisation (IWO) algorithm introduced by Xing and Gao [41]. However, it incorporates several enhancements: support for Continuous and discrete hyperparameters, elitist survival selection, and an optional Simulated Annealing (SA) acceptance mechanism to balance exploitation and exploration.

Two types of perturbations are utilised during offspring generation:

- **Continuous parameters:** A small Gaussian noise is added:
$$x'_{i,k} = x_{i,k} + \mathcal{N}(0, \sigma) \tag{7}$$
- **Integer parameters:** A discrete shift is applied:
$$x'_{i,k} = x_{i,k} + \delta, \quad \delta \in \{-1, +1\} \tag{8}$$

EIWO-SA includes an optional SA-style acceptance rule that permits occasional acceptance of worse solutions by using (9):

$$p = \exp\left(-\frac{f(\mathbf{x}'_i) - f(\mathbf{x}_i)}{T_{\text{sa}}}\right) \tag{9}$$

Compared to standard IWO, EIWO-SA emphasises deterministic elitism and modular adaptability. It streamlines population control while maintaining flexibility in neighborhood search via Gaussian and integer mutations.

4) Elitist Crisscross Mutation Optimiser (ECSO): ECSO is a population-based metaheuristic designed for hyperparameter optimisation in structured machine learning models, such as XGBoost. ECSO is inspired by the Crisscross Optimisation (CSO) algorithm introduced by Meng et al. [42], However, it simplifies the original framework by omitting crossover stages and focusing solely on mutation-based updates with elitist retention. This design prioritises computational efficiency and local search stability while maintaining strong exploratory behaviour.

Algorithm 3 Elitist IWO-SA for Hyperparameter Optimisation

Input: Hyperparameter space $\mathcal{P}$,
Initialise population $\{\mathbf{x}_i\}_{i=1}^{N_0}$ within bounds
Set population cap $N_{\max}$ and maximum iterations $T_{\max}$
for $t = 1$ to $T_{\max}$ **do**
  for all individual $\mathbf{x}_i$ **do**
    Generate offspring $\mathbf{x}'_i$ using Eq. (7) or Eq. (8)
  end for
  Merge offspring with the parent population
  for all candidates $\mathbf{x}'_j$ **do**
    if SA logic enabled **then**
      Accept with probability from Eq. (9)
    else
      Accept if $f(\mathbf{x}'_j) < f(\mathbf{x}_j)$
    end if
  end for
  Retain top $N_{\max}$ individuals by fitness
  Update elite $\mathbf{x}^*$ if improved
end for
Return best solution $\mathbf{x}^*$

Each solution $\mathbf{x}_i \in \mathbb{R}^d \cup \mathbb{Z}^d$ represents a candidate hyperparameter vector in (2). The optimisation proceeds over a fixed number of iterations by applying localised perturbations termed *crisscross mutations* to everyone in the population, excluding the elite solution $\mathbf{x}^*$, which is preserved across generations.

Population Initialisation: Each parameter is initialised uniformly:

$$x_k^{(0)} \sim \begin{cases} \mathcal{U}(\text{lb}_k, \text{ub}_k), & \text{if continuous} \\ \text{randint}(\text{lb}_k, \text{ub}_k), & \text{if discrete} \end{cases} \tag{10}$$

Continuous Parameter Mutation:

$$x_k^{(t+1)} = \text{clip}\left(x_k^{(t)} + \mathcal{N}(0, \sigma^2),\ \text{lb}_k,\ \text{ub}_k\right) \tag{11}$$

Discrete Parameter Mutation:

$$x_k^{(t+1)} = \text{clip}\left(x_k^{(t)} + \Delta,\ \text{lb}_k,\ \text{ub}_k\right), \quad \Delta \in \{-2, -1, +1, +2\} \tag{12}$$

Unlike the original CSO, which performs horizontal and vertical crossover between individuals, ECSO relies solely on independent mutation operations. This mutation-only strategy improves efficiency, maintains population diversity, and ensures convergence stability. Strict elitism guarantees that the best solution is preserved across generations, making ECSO particularly well-suited.

5) Elitist Hybrid COS-IWO-Based Optimiser (COIWSO-SA): ECOIWSO-SA is a customised algorithm developed to improve hyperparameter optimisation in structured machine learning settings. COIWSO-SA integrates two complementary

Algorithm 4 Elitist Crisscross Mutation Optimiser (ECSO)

Input: Search space $\mathcal{S}$, model class $\mathcal{M}$, iterations T, population size N
Initialise population $\mathcal{P} = \{x_1, x_2, \ldots, x_N\}$ using Eq. (10)
Evaluate cost for each $x_i \in \mathcal{P}$
Set elite $x^* \leftarrow \arg\min_{x_i \in \mathcal{P}} \text{Cost}(x_i)$
for $t = 1$ to T **do**
  for each $x_i \in \mathcal{P}$ **do**
    if $x_i \neq x^*$ **then**
      for each parameter $p \in x_i$ **do**
        if p is continuous **then**
          Update p using Eq. (11)
        else
          Update p using Eq. (12)
        end if
      end for
    end if
  end for
  Re-evaluate population
  Update x^* if a better solution is found
  Record performance metrics
end for

return x^*

Algorithm 5 COIWSO-SA: Hybrid COS-IWO Optimiser with Simulated Annealing

Input: Search space $\mathcal{P}$
Initialise population $\mathcal{P} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$
Evaluate fitness $f(\mathbf{x}_i)$ and set elite $\mathbf{x}^*$
for $t = 1$ to $T_{\max}$ **do**
  for each non-elite $\mathbf{x}_i$ **do**
    Sample $D \sim \mathcal{U}(0, 1)$
    if $D < 0.5$ **then**
      Update $\mathbf{x}_i'$ via COS operator
    else
      UPDATE $\mathbf{x}_i'$ via IWO
    end if
    Clip $\mathbf{x}_i'$ to bounds and evaluate $f(\mathbf{x}_i')$
    if $f(\mathbf{x}_i') < f(\mathbf{x}_i)$ **then**
      Accept $\mathbf{x}_i'$
    else
      Accept with probability p from Eq. (13)
    end if
  end for
  Update elite $\mathbf{x}^*$ if improved
end for
Return best solution $\mathbf{x}^*$

update strategies: a Crisscross Optimisation-inspired (CSO) operator and an Invasive Weed Optimisation (IWO) mutation scheme. The hybrid design enables local refinement and global search within a unified elitist framework /citeabidi2024eegdepressionnet.

Each candidate solution $\mathbf{x}_i \in \mathbb{R}^d \cup \mathbb{Z}^d$ encodes five XGBoost hyperparameters set in (2). The initial population is uniformly sampled within parameter bounds. A composite cost function evaluates each solution's performance (Equation (1)). The best-performing solution $\mathbf{x}^*$ is retained without modification (elitism).

All mutated parameters are clipped to their valid bounds. Simulated annealing logic is optionally applied to accept slightly worse solutions using the probability:

$$p = \exp\left(-\frac{f(\mathbf{x}_i') - f(\mathbf{x}_i)}{T_{\text{sa}}}\right) \tag{13}$$

Unlike single-operator strategies such as ESAO or EIWO-SA, COIWSO-SA adaptively switches between recombination-based exploration and mutation-based exploitation. This hybrid structure balances diversity and convergence, making it well-suited for complex, high-dimensional optimisation landscapes.

4. Classification Models for EEG-Based Depression Detection

A. XGBoost as the Baseline Classifier for Hyperparameter Optimisation in Depression Detection

This study employs Extreme Gradient Boosting (XGBoost) as the core classifier for detecting Depression based on EEG-derived features. XGBoost is an ensemble learning technique based on gradient boosting, which constructs a powerful predictive model by incrementally adding decision trees that correct the errors of prior iterations. The internal mechanism of the algorithm, illustrated in Fig. 3, highlights the iterative process of training multiple trees, each optimised to reduce the classification error of its predecessor.

The process begins with an initial prediction, typically set to the log-odds of the target distribution. Residuals are then computed to quantify the errors between the predicted and actual values. These residuals drive the growth of the first regression tree. Each potential split within the tree is evaluated using a gain function. If the calculated gain exceeds a predefined threshold, the node splits; otherwise, it is pruned. The `max_depth` hyperparameter, marked in the diagram, constrains the maximum number of levels a tree can grow, thereby controlling model complexity and preventing overfitting.

The recursive splitting continues until the tree reaches its terminal nodes (leaves). The output values of these leaves are scaled using the learning rate parameter η, which regulates the contribution of each tree to the final prediction. This step in the diagram, indicated as *Scale with Learning Rate*, dampens overly confident predictions and helps prevent overfitting. The output is added to the cumulative prediction, and the next tree is trained using updated residuals.

This process repeats for a predefined number of boosting iterations, controlled by the `n_estimators` hyperparameter. The ensemble prediction is obtained by summing the outputs of all trees. Before classification, the aggregated output undergoes a logistic transformation and sigmoid activation to yield a probability score in the range $[0, 1]$. A classification threshold is then applied to label the instance as "Depression" or "Not Depression."

To enhance generalisation and reduce overfitting, XGBoost applies both row-wise and column-wise subsampling. The `colsample_bytree` hyperparameter determines the input features considered at each split, while `subsample` governs the training samples utilised per boosting round.

Formally, at iteration t, the regularised objective function is given by:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) \tag{14}$$

where l is the logistic loss function, and f_t represents the tree added at iteration t. The regularisation term is defined as:

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \tag{15}$$

Here, T denotes the number of leaves in the tree, and w_j is the weight of the j-th leaf. The prediction is updated as:

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + \eta \cdot f_t(x_i) \tag{16}$$

Where $\eta \in [0.05, 0.2]$ is the learning rate that controls the influence of each tree.

By integrating hyperparameters set in Equation (2) into the iterative learning loop, and optimising them via metaheuristic strategies (as discussed in Section 3.3), the proposed model achieves robust and high-accuracy predictions for complex EEG-based depression classification.

B. Baseline Classifier Comparison for EEG-Based Depression Detection

1) k-Nearest Neighbors (k-NN): The KNN algorithm is included as one of the baseline classifiers for EEG-based depression detection. Initially introduced by Dasarathy [43], as a non-parametric, instance-based model, KNN labels data points by majority voting among their k nearest neighbours. The classification of a new instance is determined by evaluating the labels of the k closest training samples, where proximity is measured using a distance metric defined in Equation (17). This study sets the number of neighbours to k = 2.

$$d(x, x_i) = \sqrt{\sum_{j=1}^{n} (x_j - x_{ij})^2} \tag{17}$$

2) Naïve Bayes (NB) Classifier: In this study, the NB classifier is included as one of the baseline models for EEG-based depression classification, evaluating comparative model performance across different learning algorithms. The classifier assumes that features follow a Gaussian (normal) distribution, as defined in Equation (18), and applies Bayes' theorem to compute posterior probabilities, as shown in Equation (19). The classification decision is made based on the maximum a posteriori (MAP) estimate, leveraging the likelihood and prior probability of each class [44].

$$P(x_i \mid C) = \frac{1}{\sqrt{2\pi\sigma_C^2}} \exp\left(-\frac{(x_i - \mu_C)^2}{2\sigma_C^2}\right) \tag{18}$$

$$P(C \mid X) = \frac{P(X \mid C) \cdot P(C)}{P(X)} \tag{19}$$

3) Random Forest Algorithm: RF is utilised in this study as one of the classification algorithms for EEG-based depression detection, leveraging its robust ensemble learning framework as introduced by Breiman [45]. This approach builds a forest of decision trees, with each tree trained on a randomly selected, bootstrapped subset of the original training data. By aggregating the outputs of all trees, the model improves its generalisation ability and reduces the likelihood of overfitting. In this work, the RF classifier available in the `sklearn.ensemble` library is used. The configuration includes 200 decision trees, a depth limit of 10 for each tree, and a requirement that at least two samples be present to perform a node split. To promote variation among trees, only a random portion of the available features is considered when deciding on each split, introducing variability among the trees and contributing to reduced overfitting and improved model performance.

4) Logistic Regression: LR is employed as one of the classification models in this study for analysing EEG-based depression data. It models the probability of the binary outcome (depressed vs. non-depressed) using the sigmoid (logistic) function in Figure 4, as defined in Equation (20), that converts input data into a normalized value bounded between 0 and 1 [46].

$$\mathrm{f}(z) = \frac{1}{1 + e^{-z}} \tag{20}$$

Model training is carried out by optimising the cross-entropy loss, which measures the divergence between predicted and actual class distributions, shown in Equation (21), using gradient-based optimisation over 500 iterations [46].

$$\mathcal{L}(\mathbf{w}) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right] \tag{21}$$

5) Multilayer Perceptron: Comprised of multiple interconnected layers of neurons, the Multilayer Perceptron is a type of feedforward artificial neural network [47]. This study uses the MLP as one of the classifiers for EEG-based depression detection. Its general architecture is illustrated in Figure 5. The model is implemented using the `MLPClassifier` from the `scikit-learn.neural_network` module, with training configured for up to 1000 iterations. The output layer is designed for binary classification, distinguishing between depressed and non-depressed states.

The activation function used in the network is ReLU, which is formally defined in Equation (22), and Adam optimiser is employed, offering the combined benefits of momentum and adaptive learning rate adjustment.

$$\mathrm{ReLU}(x) = \max(0, x) \tag{22}$$

Logistic loss, also known as log loss and represented by Equation (23), is used as the objective function for training.

$$\mathcal{L} = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right] \tag{23}$$

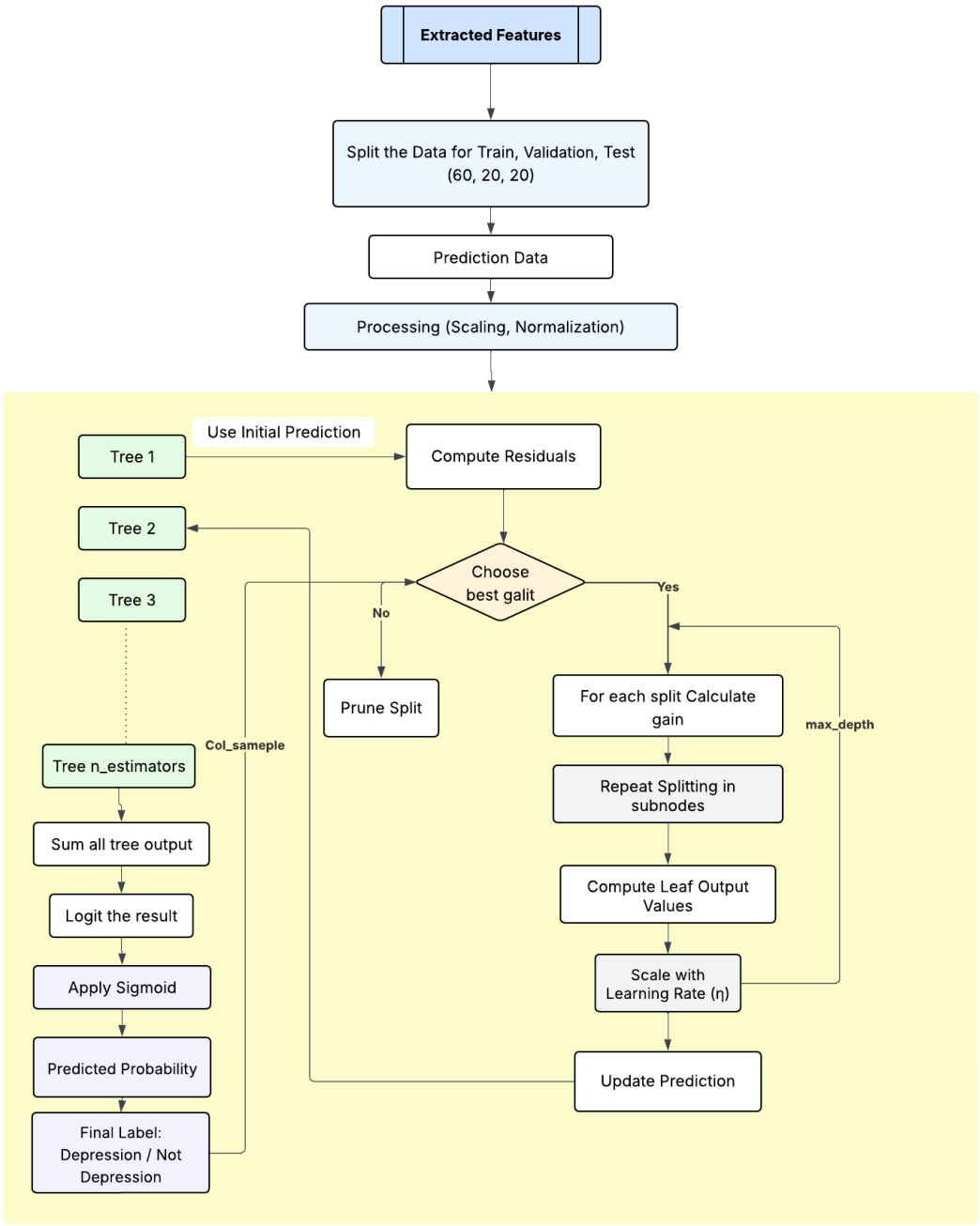


Fig. 3: Workflow of EEG-based depression detection using XGBoost, including feature preprocessing, residual-based tree training, and final label prediction via ensemble learning.

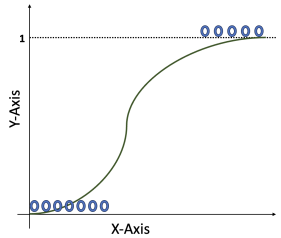


Fig. 4: Logistic regression curve: Predicting binary outcomes using a sigmoid function.

5. Multi-Representation Feature Extraction and Fusion for EEG-Based Depression Detection

A. Feature 1 : 1D-CNN-Based Temporal Feature Extraction

The pre-processed EEG signals are divided into overlapping 1-second epochs and processed using 1D-CNN to extract discriminative temporal features. Each epoch is structured as a time-series matrix with multiple channels, where each row represents a distinct EEG channel and each column corresponds to a specific time point.

1D-CNN architecture applies temporal convolutional filters independently across each EEG channel, enabling the network to learn localised temporal patterns indicative of depressive or non-depressive neural activity. The result produced by the b-th filter within the a-th convolutional layer is calculated as follows:

$$y_b^a = p\left(\sum_{c=1}^{N} y_c^{a-1} * l_{cb}^a + u_b^a\right) \tag{24}$$

In this expression, y_c^{a-1} refers to the output from the c-th channel of the preceding layer, l_{cb}^a denotes the 1D convolution kernel linking input channel c to output channel b, u_b^a is the bias associated with the b-th filter, and $p(\cdot)$ represents the activation function, implemented here as ReLU. The convolution operator is denoted by $*$, and N indicates the total number of input channels. Each epoch is structured as a time-series matrix with multiple channels, where each row represents a distinct EEG channel and each column corresponds to a specific time point.

Once the convolution layers are complete, a global average pooling (GAP) operation is applied to condense each feature map into a single scalar by computing the average along the temporal axis:

$$z_k = \frac{1}{T}\sum_{t=1}^{T} y_{k,t} \tag{25}$$

where $y_{k,t}$ is activation of the k-th feature at t, and T denotes total number of time steps. This produces a fixed-length vector $\text{FE}_{\text{1D-CNN}}$ that captures the temporal dynamics of the EEG epoch.

To mitigate inter-subject variability and ensure stable feature distributions, z-score normalisation is applied:

$$z_i = \frac{x_i - \mu}{\sigma} \tag{26}$$

In this process, x_i is the input feature before normalisation, and μ and σ represent the training data's central tendency and spread. The transformed feature vectors are then stored for use in both classification and multimodal fusion stages.

B. Feature 2 : 3D-CNN-Based Spatiotemporal Feature Extraction from Spectrograms

3D-CNN is employed to model complex spatiotemporal dependencies in EEG data, extracting deep feature representations from overlapping 2-second EEG epochs. Each epoch is reshaped into a five-dimensional tensor of shape $(1, C, T, 1)$, where C and T represent number of EEG channels and time samples, respectively. This structure maintains spatial and temporal integrity throughout the receptive regions processed by the network.

The full input to the network follows the format $(N, 1, C, T, 1)$, and the network input consists of N training epochs, C EEG channels, and T time steps. The singleton dimensions (the first and last 1s) are utilised to maintain compatibility with 3D convolution operations. The architecture features three convolutional stages with increasing filter depths, each followed by batch normalisation and dropout to maintain training consistency and minimise overfitting.

Convolutional Processing:

Each 3D convolutional layer applies localised spatiotemporal filters to the input tensor. The output of the b-th filter at layer a is given by:

$$y_b^a = p\left(\sum_{c=1}^{N}\sum_{i=1}^{k_h}\sum_{j=1}^{k_w} x_{c,i,j}^{a-1} \cdot l_{cbij}^a + u_b^a\right) \tag{27}$$

where $x_{c,i,j}^{a-1}$ is the input at spatial position (i, j) from channel c in the previous layer, l_{cbij}^a is the weight of the convolutional kernel, and u_b^a is the corresponding bias term. The function $p(\cdot)$ denotes the activation function, implemented as ReLU.

The first two convolutional blocks use kernel sizes of $(1, 3, 3)$ to preserve high temporal resolution while capturing local spatial dependencies. Batch normalisation and a dropout rate of 0.3 stabilise and regularise training.

Global Average Pooling:

A global average pooling (GAP) layer is applied to condense the feature maps by averaging values over all spatial dimensions:

$$z_k = \frac{1}{H \cdot W \cdot D}\sum_{h=1}^{H}\sum_{w=1}^{W}\sum_{d=1}^{D} y_{k,h,w,d} \tag{28}$$

Where $y_{k,h,w,d}$ is the activation at spatial position (h, w, d) in the k-th feature map, and H, W, D represent the height, width, and depth, respectively. The resulting fixed-length vector $\text{FE}_{\text{3D-CNN}} \in \mathbb{R}^d$ summarises spatiotemporal features of the EEG epoch.

Feature Normalisation:

To ensure inter-subject comparability and reduce amplitude bias, z-score normalisation is applied:

$$z_i = \frac{x_i - \mu}{\sigma} \tag{29}$$

Here, x_i denotes the original feature value, while μ and σ represent the mean and standard deviation calculated over the training dataset.

The resulting normalised vectors are stored alongside binary class labels (depressed vs. non-depressed) in `.npy` format for downstream classification and multimodal fusion.

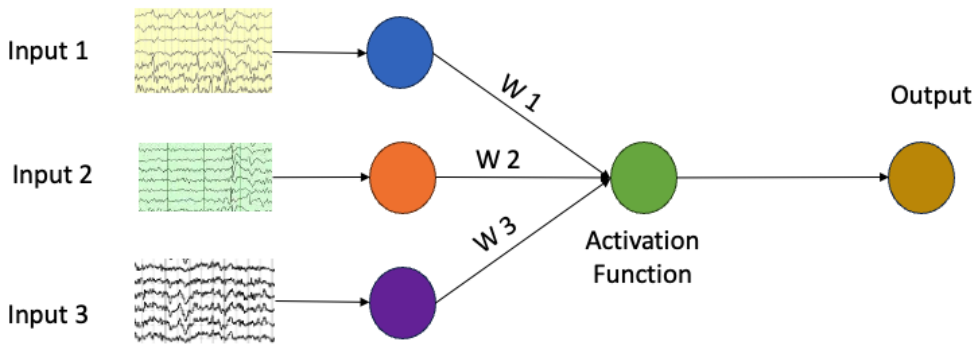


Fig. 5: Single perceptron model: Computing the weighted sum of EEG inputs and passing through an activation function to generate the output.

C. Feature 3 : Spectral Feature Extraction Using Frequency-Domain Analysis

A spectral feature extraction procedure based on STFT was implemented to extract interpretable frequency-domain representations from EEG signals. This method provides a localised time-frequency decomposition, allowing for the capture of oscillatory patterns associated with depressive neural activity.

Before transformation, each EEG recording was bandpass filtered between 1–50 Hz to suppress physiological and environmental noise and resampled to 256 Hz for consistency with STFT parameters. The EEG signal for each channel was segmented into overlapping 1-second windows with a 50% overlap (i.e., 128-sample Hamming window and 64-sample overlap). For each segment, the STFT was computed as:

$$Z_x(f,t) = \sum_{n=0}^{N-1} x(n) \cdot w(n-t) \cdot e^{-j2\pi fn/N}$$

where $Z_x(f,t)$ denotes the complex STFT coefficient at frequency f and time t, $x(n)$ is the discrete EEG signal, $w(\cdot)$ is the Hamming window, and N is the window length.

The power spectrogram was obtained by computing the squared magnitude of the STFT coefficients:

$$P(f,t) = |Z_x(f,t)|^2$$

To reduce dimensionality and enhance interpretability, average power was computed for each canonical EEG frequency band. For a given frequency band b bounded by f_l and f_h, the average band power $\bar{P}_b$ was defined as:

$$\bar{P}_b = \frac{1}{|F_b| \cdot T} \sum_{f \in F_b} \sum_{t=1}^{T} P(f,t)$$

where $F_b = \{f \mid f_l \leq f \leq f_h\}$ is the set of frequency bins for band b, and T is the number of time frames.

This process was repeated for five standard EEG bands: Delta, Theta, Alpha, Beta, and Gamma. The average power across these bands and all channels was concatenated into a feature vector:

$$\text{FE}_{\text{STFT}} \in \mathbb{R}^{C \times B}$$

In this context, C denotes the total number of EEG channels, while $B = 5$ refers to the number of frequency bands used.

To standardise the features and reduce inter-subject/session variability, z-score normalisation was applied:

$$z_i = \frac{x_i - \mu}{\sigma}$$

Where x_i is the raw feature, and μ, σ are the mean and standard deviation computed across training epochs.

The final output is a normalised spectral feature matrix per EEG epoch, utilised in subsequent classification and fusion stages.

D. Feature 4 : Beta Band Feature Extraction from EEG Signals

Beta-band power was computed using STFT to extract frequency domain features associated with cognitive and emotional processing. This time–frequency method enables localised spectral decomposition and is well-suited for capturing neural oscillations relevant to depressive states.

EEG signals were segmented into overlapping 2-second epochs with a 1-second overlap, ensuring sufficient temporal resolution. Each segment was processed channel-wise using STFT to obtain the time–frequency representation.

For a discrete EEG signal $x(t)$, the STFT is given by:

$$Z_x(f,t) = \sum_{n=0}^{N-1} x(n) \cdot w(n-t) \cdot e^{-j2\pi fn/N} \tag{30}$$

where $w(\cdot)$ is a 128-sample Hamming window, N is the window length, and $Z_x(f,t)$ denotes the complex-valued STFT coefficient at frequency f and time t.

The power spectrogram is computed as:

$$P(f,t) = |Z_x(f,t)|^2 \tag{31}$$

Spectral components within 12–30 Hz were retained to isolate beta-band activity. For each channel c, the average beta-band power $\bar{P}_{\beta,c}$ was calculated as:

$$\bar{P}_{\beta,c} = \frac{1}{|F_\beta| \cdot T} \sum_{f \in F_\beta} \sum_{t=1}^{T} P_c(f,t) \tag{32}$$

where $F_\beta = \{f \mid 12 \leq f \leq 30\}$ is the beta frequency range, T is the total number of time frames, and $P_c(f,t)$ is the power for channel c at frequency f and time t.

The result is a $\mathbb{R}^C$-dimensional feature vector for each epoch, where C is the number of EEG channels. To reduce inter-subject variability and ensure normalisation, each feature is z-score normalised as:

$$z_i = \frac{x_i - \mu}{\sigma} \tag{33}$$

The raw beta power is denoted as x_i, and the mean (μ) and standard deviation (σ) are computed from training examples. These normalised vectors, labelled according to BDI-based depression status, were stored in downstream classification and fusion modules.

6. RESULTS AND DISCUSSIONS

A. Experimental Configuration

All experiments for the proposed EEG-based depression detection framework were implemented in Python. An initial benchmarking phase was conducted using six widely recognised classification algorithms: KNN, LR, NB, MLP, RF, and XGBoost. These classifiers were evaluated across four distinct EEG feature representations: (a) temporally encoded features from 1D-CNN, (b) spatiotemporal features derived from 3D-CNN, (c) spectral power features obtained via STFT, and (d) frequency-specific beta-band features. Performance was quantified using two primary metrics: classification accuracy and mean average precision (MAP) to assess both predictive accuracy and probabilistic ranking capability.

Among all classifiers, XGBoost consistently demonstrated superior performance across feature sets and evaluation metrics. Due to its robustness and high predictive capacity, XGBoost was selected as the target model for hyperparameter optimisation.

Five custom elitist metaheuristic algorithms were employed to enhance its performance further and tune key XGBoost hyperparameters: EJHO, ESAO, EIWO-SA, ECSO, and HCOIWSO-SA. The optimisation process was conducted with a population size of 10 over 30 iterations to ensure sufficient convergence.

B. Evaluation Metrics and Visualisation Methods

The performance of the proposed depression detection model was evaluated using a comprehensive set of standard classification metrics, including accuracy, precision, recall, MCC, FDR, FPR, and MAP. These metrics were implemented using the `scikit-learn` library and were selected to provide insights into both predictive accuracy and error behaviour across various experimental settings.

Multiple visualisation techniques were applied to enhance interpretability. Line charts illustrated the convergence behaviour of each optimiser based on the custom cost function across iterations. Bar charts and tables summarised classification results for each feature set and optimisation method, facilitating direct performance comparisons. Additionally, bubble charts visually integrate multiple performance metrics in a single plot, highlighting trade-offs and optimiser strengths.

To understand model discrimination spatially, a Principal Component Analysis (PCA) plot was generated to project high-dimensional EEG features into two dimensions. Specifically, the PCA visualisation factualised on the XGBoost classifier's performance to showcase class separability and feature distribution within the reduced space.

C. Model Comparison and Selection for Optimisation

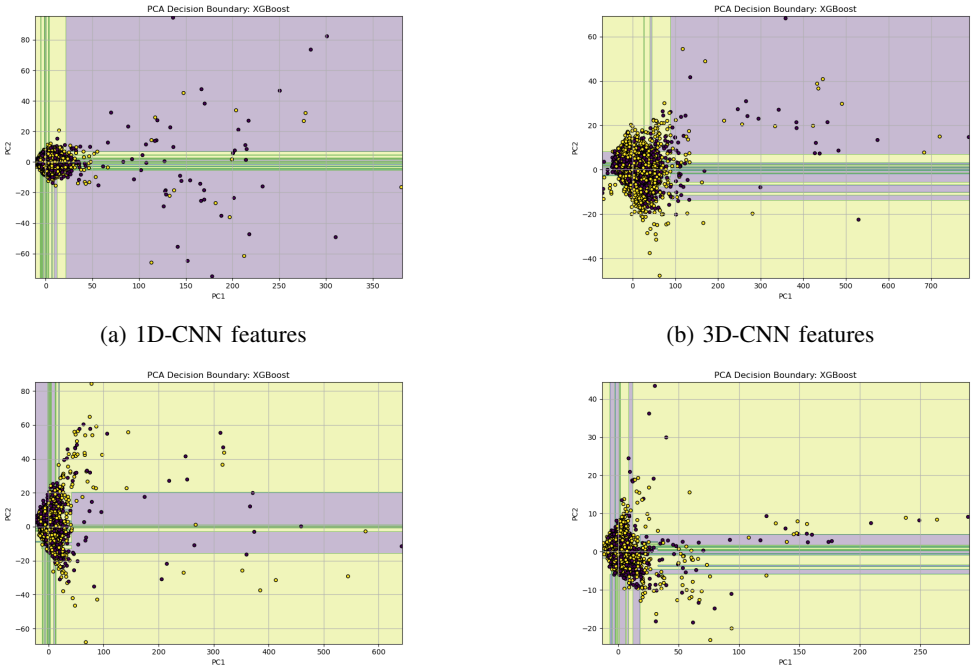
In this section, six classification algorithms: XGBoost, RF, KNN, MLP, LR, and NB are evaluated across four EEG-derived feature sets: (a) features extracted using 1D-CNN, (b) 3D-CNN features, (c) spectral features obtained via STFT, and (d) beta-band power features. Model performance is assessed using two key metrics: classification accuracy and mean average precision (MAP), which capture both predictive correctness and probabilistic ranking quality.

Figure 7 presents the results of this comparative evaluation. The bar charts represent model accuracy, while the overlaid line plots illustrate corresponding MAP values. Across all feature representations, XGBoost consistently achieves the highest performance, with accuracy exceeding 96% and MAP values above 0.99. Furthermore, its performance remains relatively stable across different feature sets, indicating robustness to varying input representations.

These findings demonstrate the superior generalisation capability of XGBoost in EEG-based depression classification. Based on this performance, XGBoost is selected as the target model for further refinement. In the subsequent section, it is optimised using five custom metaheuristic strategies: the Elitist Jellyfish Optimiser (EJHO), Elitist Simulated Annealing Optimiser (ESAO), Elitist Invasive Weed Optimisation with Simulated Annealing (EIWO-SA), Elitist Crisscross Mutation Optimiser (ECSO), and the hybrid COS-IWO-based optimiser (COIWSO-SA). The following section discusses optimisation results and comparative performance improvements in detail.

D. Optimisation Behavior and Cost Function Convergence

Figure 8 illustrates the convergence behaviour of the proposed custom cost function, as defined in Equation (1), across five optimisation algorithms: Elitist Jellyfish Optimiser (EJHO), Elitist Simulated Annealing Optimiser (ESAO), Elitist Invasive Weed Optimisation with Simulated Annealing (EIWO-SA), Elitist Crisscross Mutation Optimiser (ECSO), and the hybrid COS-IWO-based Optimiser (COIWSO-SA). Each optimiser was evaluated using four different EEG feature sets: (a) 1D-CNN features, (b) 3D-CNN features, (c) spectral features, and (d) beta-band features.



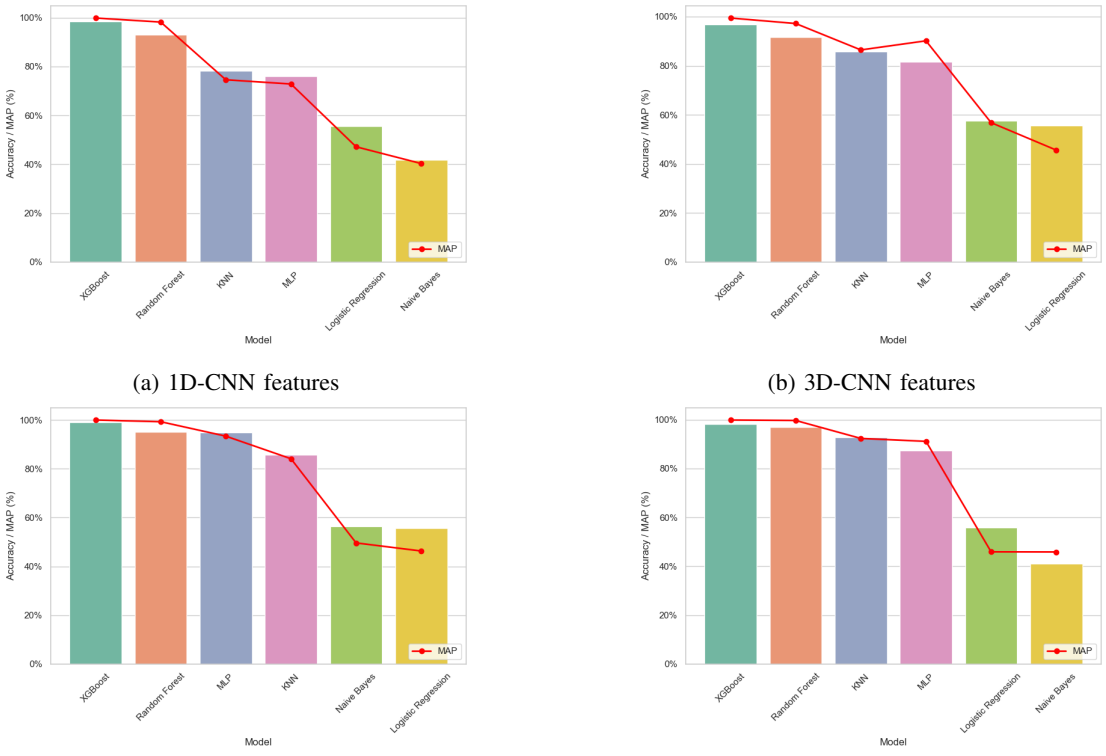
(a) 1D-CNN features

(b) 3D-CNN features

(c) Spectral features

(d) Beta Band Features

Fig. 6: Two-dimensional PCA projection showing XGBoost decision boundaries across four EEG feature types: (a) 1D-CNN, (b) 3D-CNN, (c) spectral features, and (d) beta-band power features. Each plot illustrates the classifier's separability and clustering behaviour within the reduced feature space.



(a) 1D-CNN features

(b) 3D-CNN features

(c) Spectral features

(d) Beta Band Features

Fig. 7: Performance comparison of six classification models (XGBoost, RF, KNN, MLP, LR, NB) on (a) 1D-CNN, (b) 3D-CNN, (c) Spectral Features and (d) Beta Band Feature . Both Accuracy and Mean Average Precision (MAP) are presented as percentages.

Among all strategies, EIWO-SA consistently demonstrated the most effective convergence across all feature types. Specifically, it achieved the lowest final cost values 0.4915, 0.5060, 0.4890, and 0.4945 for 1D-CNN, 3D-CNN, spectral, and beta-band features, respectively. These correspond to relative improvements of 2.25%, 0.4%, 2.7%, and 6.5% when compared to the next best optimiser in each setting.

Furthermore, EIWO-SA exhibited rapid convergence, stabilising the cost function within the first five iterations in all cases. For example, in the 1D-CNN setting, the cost decreased sharply from approximately 0.502 to 0.4915 within five iterations. This convergence efficiency, combined with its superior final performance, highlights the robustness and effectiveness of IWO-SA for optimising XGBoost hyperparameters in EEG-based depression classification.

E. Feature Set Impact on Classification Performance

Figure 9 presents a comprehensive comparative analysis of four EEG-derived feature sets across seven key evaluation metrics: (a) Accuracy, (b) MAP, (c) MAR, (d) MCC, (e) FDR, (f) FPR, and (g) the proposed Cost Function. The spectral feature set demonstrates the most robust and consistent performance among the evaluated feature types. It achieves the highest classification scores, including an accuracy exceeding 99%, a MAP of 99.9%, a MAR of 99.3%, and an MCC of 98.2%, underscoring its superior discriminative capacity.

Moreover, the spectral features exhibit the lowest values in error-sensitive metrics, with a cost function value below 0.5 and FDR and FPR maintained at approximately 0.02. These results indicate that the model is highly accurate and well-calibrated, exhibiting conservative behaviour in reducing false alarms, an essential consideration in clinical screening applications.

Together, these findings establish the spectral feature set as the most informative and reliable input representation for EEG-based depression classification in this study. Its superior balance between sensitivity and specificity suggests strong potential for real-world deployment in automated diagnostic systems.

F. Comparative Evaluation of Metaheuristic Optimisers for XGBoost Hyperparameter Tuning

In this section, the performance of five metaheuristic optimisers—EJHO, ESAO, EIWO-SA, ECSO, and COIWSO-SA—is systematically evaluated across four EEG-derived feature sets, as illustrated in Fig. 10: (a) 1D-CNN features, (b) 3D-CNN features, (c) spectral features, and (d) beta-band features. While Section 3.2.3 focuses on these algorithms' internal mechanisms and convergence behaviours, the present analysis compares their outcomes regarding classification accuracy, mean average precision (MAP), and the proposed cost function.

For 1D-CNN features, EIWO-SA yields the best performance, attaining an accuracy of 0.5890, a MAP score of 0.999, and the lowest cost function value at 0.4919. This trend persists with 3D-CNN features, where IWO-SA again outperforms its counterparts, achieving an accuracy of 0.9735, a MAP of 0.996, and a cost of 0.5600, highlighting its effectiveness in optimising convolutional representations.

In contrast, COIWSO-SA is the leading optimiser when considering frequency domain features. Regarding spectral features, it achieves the highest accuracy of 0.5906 and a perfect MAP of 1.000, with a slightly elevated cost value of approximately 0.4780. A similar pattern is observed for beta-band features, where COIWSO-SA achieves an accuracy of 0.5900, a MAP of 0.999, and a cost function value of approximately 0.4950.

These findings suggest that EIWO-SA is well-suited for optimising deep temporal-spatial representations such as CNN-based features. At the same time, COIWSO-SA provides better generalisation for spectral-domain inputs. Overall, the results underscore the importance of aligning the optimiser strategy with the feature representation to achieve optimal performance in EEG-based depression classification.

G. Model Selection Based on Optimiser-Driven Performance Evaluation

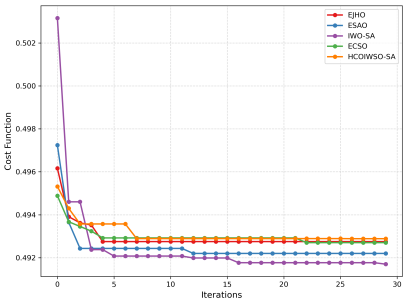
Based on the comparative performance results presented in Table IV, the final recommended model is the XGBoost classifier optimised using the COIWSO-SA algorithm with spectral feature inputs. This configuration achieved the highest classification accuracy of 99.06% and consistently performed well across all evaluation metrics. These findings confirm that integrating spectral feature extraction with COIWSO-SA-based hyperparameter tuning provides the most effective and reliable framework for EEG-based depression detection within the proposed system.

7. Conclusion

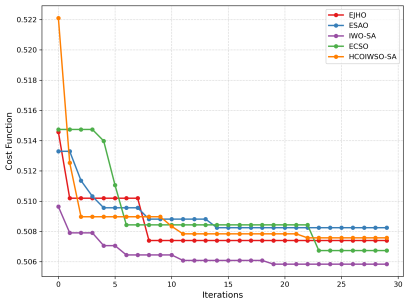
This study introduces an optimiser-guided EEG-based depression detection framework that leverages multiple feature representations and metaheuristic hyperparameter tuning to enhance classification accuracy. EEG signals were pre-processed and decomposed into four distinct feature sets: 1D-CNN-based temporal features, 3D-CNN-based spatiotemporal features, spectral power features using STFT, and frequency-specific beta-band features. These features were inputs for six machine learning classifiers, among which XGBoost demonstrated the most consistent and robust performance.

To further improve model accuracy and reliability, five custom elitist metaheuristic algorithms: EJHO, ESAO, EIWO-SA, ECSO, and COIWSO-SA were employed for hyperparameter tuning of the XGBoost model. Among these, COIWSO-SA combined with spectral features yielded the highest classification performance, achieving 99.06% accuracy and a perfect MAP score of 1.000. EIWO-SA also demonstrated competitive results, particularly on CNN-derived features, with fast convergence and strong generalisation.

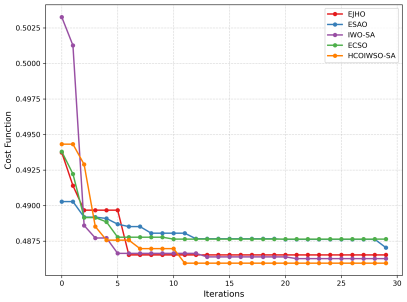
While the proposed system offers promising performance in EEG-based depression classification, it is essential to note that the dataset used in this study is limited to subjects aged 18–19, due to the scarcity of open-source data. The current model is best suited for use within this specific age group. For individuals under 18, additional validation or retraining may



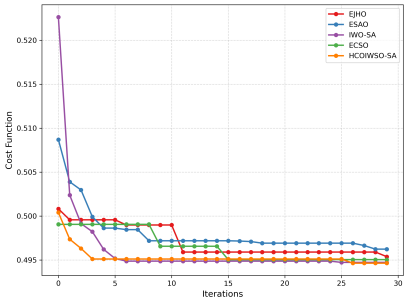
(a) 1D-CNN features



(b) 3D-CNN features

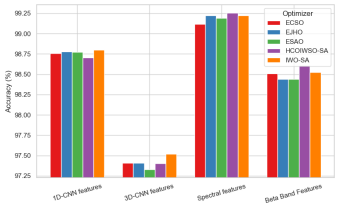


(c) Spectral features

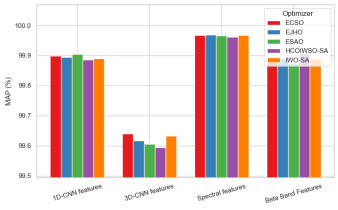


(d) Beta Band Features

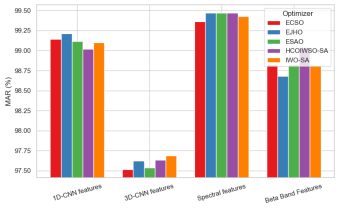
Fig. 8: Cost function for five optimisers - EJHO, ESAO, EIWO-SA, ECSO, and HCOIWSO-SA applied to different EEG feature sets. a) 1D-CNN-based features, (b) 3D-CNN-based features, (c) spectral features, and (d) beta band features



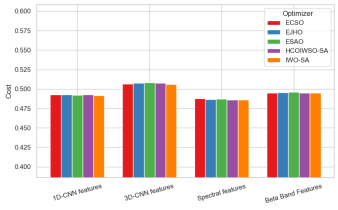
(a) Accuracy Comparison Across Feature Sets and Optimisers



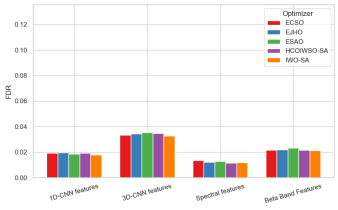
(b) MAP Comparison Across Feature Sets and Optimisers



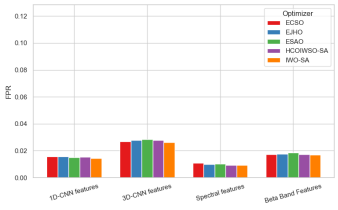
(c) MAR Comparison Across Feature Sets and Optimisers



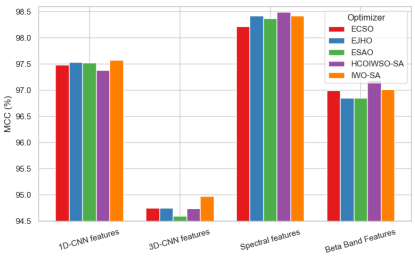
(d) Cost Function Comparison Across Feature Sets and Optimisers



(e) FDR Comparison Across Feature Sets and Optimisers

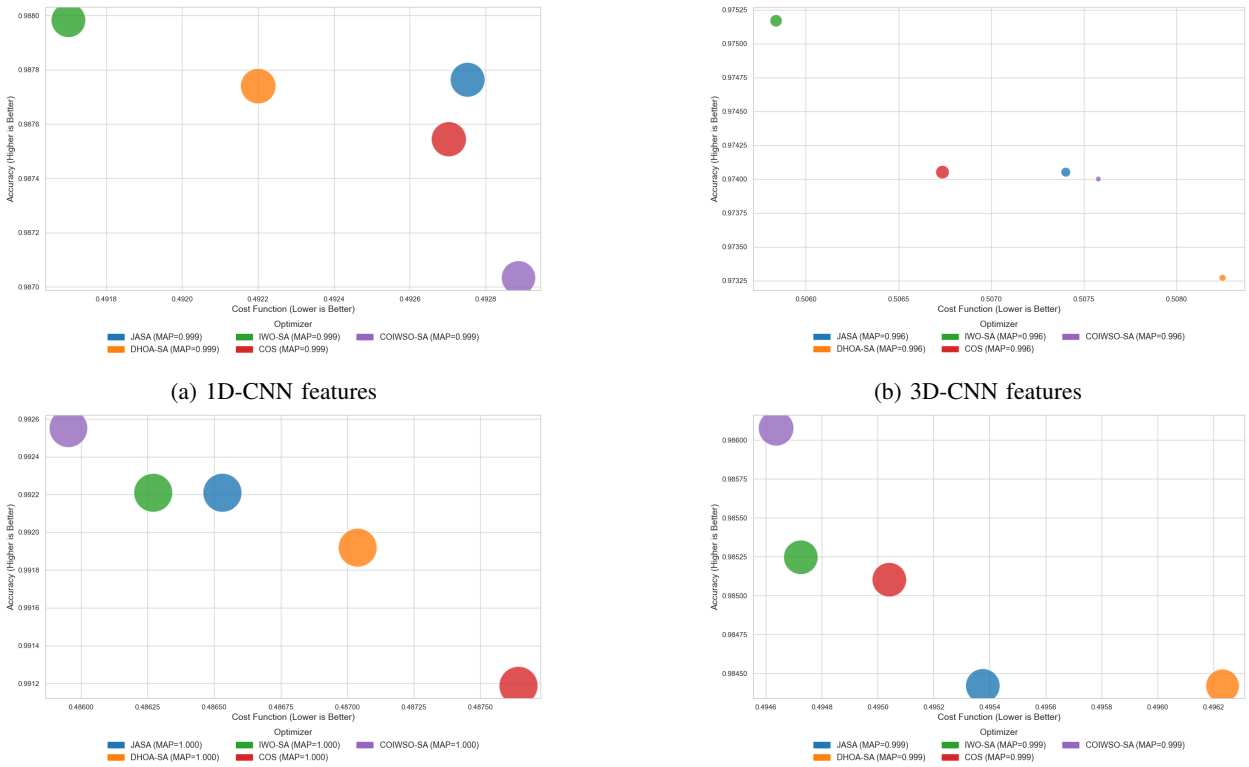


(f) FPR Comparison Across Feature Sets and Optimisers



(g) MCC Comparison Across Feature Sets and Optimisers

Fig. 9: Cost function trends across EEG feature sets using five optimisers: EJHO, ESAO, EIWO-SA, ECSO, and HCOIWSO-SA over ”(a) 1D-CNN features, (b) 3D-CNN features, (c) Spectral features and (d) Beta Band Features”



(a) 1D-CNN features

(b) 3D-CNN features

(c) Spectral features

(d) Beta Band Features

Fig. 10: Accuracy, Cost, MAP Trade off Across Five Optimisers, EJHO, ESAO, EIWO-SA, ECSO, and COIWSO-SA and Applied to Different EEG Feature Sets: (a) 1D-CNN-Based Features, (b) 3D-CNN-Based Features, (c) Spectral Features, and (d) Beta Band Features

TABLE IV: Performance Comparison of XGBoost and Optimised Models with their recommended EEG Feature Sets

Terms	**1D-CNN Features**		**3D-CNN Features**		**Spectral Features**		**Beta Band Features**	
	XGBoost	**+ EIWO-SA**	**XGBoost**	**+ EIWO-SA**	**XGBoost**	**+ COIWSO-SA**	**XGBoost**	**+ COIWSO-SA**
Accuracy	0.9848	0.9879	0.9677	0.9752	0.9896	0.9906	0.9803	0.9861
MAP	0.9983	0.9989	0.9941	0.9963	0.9993	0.9996	0.9981	0.9991
MAR	0.9875	0.9909	0.9658	0.9769	0.9914	0.9947	0.9823	0.9904
FDR	0.0215	0.0179	0.0386	0.0326	0.0148	0.0115	0.0265	0.0216
FPR	0.0172	0.0143	0.0307	0.0261	0.0118	0.0092	0.0212	0.0173
MCC	0.9693	0.9756	0.9346	0.9497	0.9790	0.9849	0.9602	0.9718

be necessary to ensure clinical reliability and developmental appropriateness.

The results confirm that aligning feature representations with tailored optimisation strategies enhances detection performance. This framework has strong potential for deployment in real-time, mobile, or clinical environments for early-stage depression screening. Future work will expand the dataset to include more diverse age groups, improve model interpretability, and extend the framework to support multi-class emotional state analysis.

In addition to this thesis submission, the core findings and methodology presented have been submitted for peer-reviewed publication in the IEEE Journal of Biomedical and Health Informatics. The submitted manuscript includes the optimiser-guided EEG classification framework, multi-representation feature extraction techniques, and metaheuristic tuning strategies introduced in this study. This submission aims to add to existing research on machine learning in mental health and to provide further evidence supporting the reliability of the proposed method.

8. ACKNOWLEDGMENT

The author would like to express sincere gratitude to Assoc. Prof. Vincent Lee, Associate Professor in the Department of Data Science & AI, for his invaluable guidance and continuous support throughout this research, including during the summer break. His mentorship and expertise played a crucial role in completing this work.

This research also made use of the dataset *"EEG: Probabilistic Selection and Depression"* by James F. Cavanagh (2021), available via OpenNeuro [DOI: https://doi.org/10.18112/openneuro.ds003474.v1.1.0], accessed in May 2024. The dataset is shared under the Creative Commons CC0 (Public Domain Dedication) license and includes EEG recordings collected during a probabilistic selection task, aimed at exploring neural correlates of depression.

REFERENCES

[1] World Health Organization, "Adolescent mental health," 2021, accessed: 2024-09-03. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/adolescent-mental-health

[2] WHO, "What on earth is world health day?" *Depression*, no. 3, 2017.

[3] KidsHealth, "Why am i so depressed?" 2023, accessed: 2024-09-03. [Online]. Available: https://kidshealth.org/en/teens/why-depressed.html

[4] D. Beirão, H. Monte, M. Amaral, A. Longras, C. Matos, and F. Villas-Boas, "Depression in adolescence: a review," *Middle East current psychiatry*, vol. 27, no. 1, p. 50, 2020.

[5] C. Kieling, A. Adewuya, H. L. Fisher, R. Karmacharya, B. A. Kohrt, J. R. Swartz, and V. Mondelli, "Identifying depression early in adolescence," *The Lancet. Child & adolescent health*, vol. 3, no. 4, p. 211, 2019.

[6] D. P. X. Kan and P. F. Lee, "Decrease alpha waves in depression: An electroencephalogram (eeg) study," in *2015 International Conference on BioSignal Analysis, Processing and Systems (ICBAPS)*. IEEE, 2015, pp. 156–161.

[7] A. Baskaran, R. Milev, and R. S. McIntyre, "The neurobiology of the eeg biomarker as a predictor of treatment response in depression," *Neuropharmacology*, vol. 63, no. 4, pp. 507–513, 2012.

[8] E. A. Nofzinger, J. C. Price, C. C. Meltzer, D. J. Buysse, V. L. Villemagne, J. M. Miewald, R. C. Sembrat, D. A. Steppe, and D. J. Kupfer, "Towards a neurobiology of dysfunctional arousal in depression: the relationship between beta eeg power and regional cerebral glucose metabolism during nrem sleep," *Psychiatry Research: Neuroimaging*, vol. 98, no. 2, pp. 71–91, 2000.

[9] V. A. Grin-Yatsenko, I. Baas, V. A. Ponomarev, and J. D. Kropotov, "Independent component approach to the analysis of eeg recordings at early stages of depressive disorders," *Clinical Neurophysiology*, vol. 121, no. 3, pp. 281–289, 2010.

[10] S. Dutta, S. Hazra, and A. Nandy, "Human cognitive state classification through ambulatory eeg signal analysis," in *Proceedings of the 5th International Conference on Soft Computing for Problem Solving (SocProS 2015)*, ser. Lecture Notes in Computer Science, vol. 11884. Springer, 2019, pp. 183–194.

[11] H. Hinrikus, A. Suhhova, M. Bachmann, K. Aadamsoo, Ü. Võhma, H. Pehlak, and J. Lass, "Spectral features of eeg in depression," 2010.

[12] P. Penava and R. Buettner, "Early-stage non-severe depression detection using a novel convolutional neural network approach based on resting-state eeg data," *IEEE Access*, 2024.

[13] M. H. Abidi, K. Moiduddin, R. Ayub, M. K. Mohammed, A. Shankar, and S. Shiaeles, "Eegdepressionnet: A novel self attention-based gated densenet with hybrid heuristic adopted mental depression detection model using eeg signals," *IEEE Journal of Biomedical and Health Informatics*, 2024.

[14] C. J. Stam, T. Montez, B. Jones, S. Rombouts, Y. Van Der Made, Y. A. Pijnenburg, and P. Scheltens, "Disturbed fluctuations of resting state eeg synchronization in alzheimer's disease," *Clinical neurophysiology*, vol. 116, no. 3, pp. 708–715, 2005.

[15] S. Alelyani, J. Tang, and H. Liu, "Feature selection for clustering: A review," *Data Clustering*, pp. 29–60, 2018.

[16] J. Ashburner, "A fast diffeomorphic image registration algorithm," *Neuroimage*, vol. 38, no. 1, pp. 95–113, 2007.

[17] A. R. Clarke, R. J. Barry, R. McCarthy, and M. Selikowitz, "Age and sex effects in the eeg: development of the normal child," *Clinical neurophysiology*, vol. 112, no. 5, pp. 806–814, 2001.

[18] L. Cragg, N. Kovacevic, A. R. McIntosh, C. Poulsen, K. Martinu, G. Leonard, and T. Paus, "Maturation of eeg power spectra in early adolescence: a longitudinal study," *Developmental science*, vol. 14, no. 5, pp. 935–943, 2011.

[19] P. J. Marshall, Y. Bar-Haim, and N. A. Fox, "Development of the eeg from 5 months to 4 years of age," *Clinical neurophysiology*, vol. 113, no. 8, pp. 1199–1208, 2002.

[20] P. Matthis, D. Scheffner, C. Benninger, C. Lipinski, and L. Stolzis, "Changes in the background activity of the electroencephalogram according to age," *Electroencephalography and clinical neurophysiology*, vol. 49, no. 5-6, pp. 626–635, 1980.

[21] R. J. Barry, A. R. Clarke, R. McCarthy, M. Selikowitz, S. J. Johnstone, and J. A. Rushby, "Age and gender effects in eeg coherence: I. developmental trends in normal children," *Clinical neurophysiology*, vol. 115, no. 10, pp. 2252–2258, 2004.

[22] C. Benninger, P. Matthis, and D. Scheffner, "Eeg development of healthy boys and girls. results of a longitudinal study," *Electroencephalography and clinical neurophysiology*, vol. 57, no. 1, pp. 1–12, 1984.

[23] T. Gasser, R. Verleger, P. Bächer, and L. Sroka, "Development of the eeg of school-age children and adolescents. i. analysis of band power," *Electroencephalography and clinical neurophysiology*, vol. 69, no. 2, pp. 91–99, 1988.

[24] M. Kikuchi, Y. Wada, Y. Koshino, Y. Nanbu, and T. Hashimoto, "Effect of normal aging upon interhemispheric eeg coherence: analysis during rest and photic stimulation," *Clinical Electroencephalography*, vol. 31, no. 4, pp. 170–174, 2000.

[25] B. Saletu, E. Paulus, L. Linzmayer, P. Anderer, H. V. Semlitsch, J. Grünberger, L. Wicke, A. Neuhold, and I. Podreka, "Nicergoline in senile dementia of alzheimer type and multi-infarct dementia: a double-blind, placebo-controlled, clinical and eeg/erp mapping study," *Psychopharmacology*, vol. 117, pp. 385–395, 1995.

[26] W. Mumtaz, S. S. A. Ali, M. A. M. Yasin, and A. S. Malik, "A machine learning framework involving eeg-based functional connectivity to diagnose major depressive disorder (mdd)," *Medical & biological engineering & computing*, vol. 56, pp. 233–246, 2018.

[27] H. Akbari, M. T. Sadiq, M. Payan, S. S. Esmaili, H. Baghri, and H. Bagheri, "Depression detection based on geometrical features extracted from sodp shape of eeg signals and binary pso." *Traitement du Signal*, vol. 38, no. 1, 2021.

[28] J. Zhu, Z. Wang, T. Gong, S. Zeng, X. Li, B. Hu, J. Li, S. Sun, and L. Zhang, "An improved classification model for depression detection using eeg and eye tracking data," *IEEE transactions on nanobioscience*, vol. 19, no. 3, pp. 527–537, 2020.

[29] C. Kaur, A. Bisht, P. Singh, and G. Joshi, "Eeg signal denoising using hybrid approach of variational mode decomposition and wavelets for depression," *Biomedical Signal Processing and Control*, vol. 65, p. 102337, 2021.

[30] H. Cai, J. Han, Y. Chen, X. Sha, Z. Wang, B. Hu, J. Yang, L. Feng, Z. Ding, Y. Chen *et al.*, "A pervasive approach to eeg-based depression detection," *Complexity*, vol. 2018, no. 1, p. 5238028, 2018.

[31] M. Bachmann, L. Päeske, K. Kalev, K. Aarma, A. Lehtmets, P. Ööpik, J. Lass, and H. Hinrikus, "Methods for classifying depression in single channel eeg using linear and nonlinear signal analysis," *Computer methods and programs in biomedicine*, vol. 155, pp. 11–17, 2018.

[32] S. Mahato and S. Paul, "Classification of depression patients and normal subjects based on electroencephalogram (eeg) signal using alpha power and theta asymmetry," *Journal of medical systems*, vol. 44, pp. 1–8, 2020.

[33] M. Saeedi, A. Saeedi, and A. Maghsoudi, "Major depressive disorder assessment via enhanced k-nearest neighbor method and eeg signals," *Physical and Engineering Sciences in Medicine*, vol. 43, pp. 1007–1018, 2020.

[34] U. R. Acharya, S. L. Oh, Y. Hagiwara, J. H. Tan, H. Adeli, and D. P. Subha, "Automated eeg-based screening of depression using deep convolutional neural network," *Computer methods and programs in biomedicine*, vol. 161, pp. 103–113, 2018.

[35] X. Li, X. Zhang, J. Zhu, W. Mao, S. Sun, Z. Wang, C. Xia, and B. Hu, "Depression recognition using machine learning methods with different feature generation strategies," *Artificial intelligence in medicine*, vol. 99, p. 101696, 2019.

[36] A. Sarkar, A. Singh, and R. Chakraborty, "A deep learning-based comparative study to track mental depression from eeg data," *Neuroscience Informatics*, vol. 2, no. 4, p. 100039, 2022.

[37] J. F. Cavanagh, "EEG: Probabilistic Selection and Depression," https://doi.org/10.18112/openneuro.ds003474.v1.1.0, 2021, openNeuro dataset, accessed May 2024.

[38] W. Liu, K. Jia, and Z. Wang, "Graph-based eeg approach for depression prediction: integrating time-frequency complexity and spatial topology," *Frontiers in Neuroscience*, vol. 18, p. 1367212, 2024.

[39] G. Yildizdan, "Mjs: A modified artificial jellyfish search algorithm for continuous optimization problems," *Neural Computing and Applications*, vol. 35, no. 4, pp. 3483–3519, 2023.
[40] A. Gülcü and Z. Kuş, "Multi-objective simulated annealing for hyper-parameter optimization in convolutional neural networks," *PeerJ Computer Science*, vol. 7, p. e338, 2021.
[41] B. Xing and W.-J. Gao, *Innovative computational intelligence: a rough guide to 134 clever algorithms*. Springer, 2014, vol. 62.
[42] A.-b. Meng, Y.-c. Chen, H. Yin, and S.-z. Chen, "Crisscross optimization algorithm and its application," *Knowledge-Based Systems*, vol. 67, pp. 218–229, 2014.
[43] B. V. Dasarathy, "Nearest neighbor (nn) norms: Nn pattern classification techniques," *IEEE Computer Society Tutorial*, 1991.
[44] W. I. D. Mining, "Data mining: Concepts and techniques," *Morgan Kaufinann*, vol. 10, no. 559-569, p. 4, 2006.
[45] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.
[46] M. H. Kutner, C. J. Nachtsheim, J. Neter, and W. Li, *Applied linear statistical models*. McGraw-hill, 2005.
[47] Sejal Jaiswal, "Multilayer perceptrons in machine learning," 2021, accessed: September 3, 2024. [Online]. Available: https://www.datacamp.com/tutorial/multilayer-perceptrons-in-machine-learning

APPENDIX I
FULL PYTHON CODE USED IN THE STUDY

A. Main Pipeline Code (Python)

Listing 1: Emphasizing dataset paths and setup

```
    data_dir = "/Users/myatpwintphyu/Desktop/EEG␣Feature␣Extract␣Result"
    result_root_dir = "/Users/myatpwintphyu/Desktop/Results"

    # === List of EEG Feature Files ===
    dataset_files = [
        "eeg_1dcnn_features",
        "eeg_3dcnn_features",
        "eeg_stft_beta_features",
        "eeg_stft_features"
    ]
```

Listing 2: Python Implementation of Multi-Model Evaluation Across EEG Datasets

```
models = {
    "Naive␣Bayes": GaussianNB(),
    "Logistic␣Regression": LogisticRegression(max_iter=500),
    "KNN": KNeighborsClassifier(n_neighbors=2),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric='mlogloss'),
    "MLP": MLPClassifier(max_iter=1000),
    "Random␣Forest": RandomForestClassifier(),
} # === Loop through datasets ===
all_results = []
all_report_rows = []

for folder_name in dataset_files:
    dataset_name = folder_name.replace("eeg_", "")
    dataset_path = os.path.join(data_dir, folder_name)

    # Output folders
    dataset_result_dir = os.path.join(result_root_dir, dataset_name)
    os.makedirs(dataset_result_dir, exist_ok=True)
    pca_dir = os.path.join(dataset_result_dir, "PCA")
    prc_dir = os.path.join(dataset_result_dir, "PRC")
    os.makedirs(pca_dir, exist_ok=True)
    os.makedirs(prc_dir, exist_ok=True)

    X_chunks, y_chunks = [], []

    for fname in os.listdir(dataset_path):
        if fname.startswith("X_feats_"):
            chunk_id = fname.split("_")[-1].replace(".npy", "")
            X = np.load(os.path.join(dataset_path, f"X_feats_{chunk_id}.npy"))
            y = np.load(os.path.join(dataset_path, f"y_labels_{chunk_id}.npy"))
            X_chunks.append(X)
            y_chunks.append(y)

    target_dim = min([x.shape[1] for x in X_chunks])
    # print(f" Aligning all feature chunks to {target_dim} dimensions...")

    X_aligned = []
    for x in X_chunks:
        if x.shape[1] > target_dim:
            x_trimmed = x[:, :target_dim]  # truncate extra columns
        elif x.shape[1] < target_dim:
            x_trimmed = np.pad(x, ((0, 0), (0, target_dim - x.shape[1])), mode='constant')  # pad with
    zeros
        else:
            x_trimmed = x
        X_aligned.append(x_trimmed)

    X_all = np.vstack(X_aligned)
    y_all = np.concatenate(y_chunks)

    # === Step 3: Train/Validation/Test split ===
    X_train, X_test, y_train, y_test = train_test_split(X_all, y_all, test_size=0.4, stratify=y_all,
    random_state=42)
```

```
    # === Step 4: Normalise features ===
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # === Model evaluation ===
    results = []
    report_rows = []

    for model_name, model in models.items():
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        y_proba = model.predict_proba(X_test)[:, 1] if hasattr(model, "predict_proba") else None

        acc = accuracy_score(y_test, y_pred)
        map_score = average_precision_score(y_test, y_proba) if y_proba is not None else None

        results.append({
            'Feature Set': dataset_name,
            'Model': model_name,
            'Accuracy': acc,
            'MAP': map_score
        })

        print(f"'Feature Set': {dataset_name} | 'Model': {model_name} | 'Accuracy': {acc:.4f} | 'MAP':
    {map_score:.4f}")

        report = classification_report(y_test, y_pred, output_dict=True)
        for cls, metrics in report.items():
            if isinstance(metrics, dict):
                report_rows.append({
                    'Feature Set': dataset_name,
                    'Model': model_name,
                    'Class': cls,
                    'Precision': metrics.get('precision'),
                    'Recall': metrics.get('recall'),
                    'F1-Score': metrics.get('f1-score'),
                    'Support': metrics.get('support')
                })

        try:
            plot_pca_decision_boundary(X_test, y_test, model, model_name, pca_dir)
        except Exception as e:
            print(f" PCA plot error for {model_name} on {dataset_name}: {e}")

        if y_proba is not None:
            try:
                plot_precision_recall(y_test, y_proba, model_name, prc_dir)
            except Exception as e:
                print(f" PRC plot error for {model_name} on {dataset_name}: {e}")

    all_results.extend(results)
    all_report_rows.extend(report_rows)
```

Listing 3: Full Pipeline for Hyperparameter Optimisation and Evaluation Across EEG Feature Sets

```
optimiser_summary = []
final_optimiser_results = []
# === Loop Through Datasets ===
for folder_name in dataset_files:
    dataset_name = folder_name.replace("eeg_", "")
    dataset_path = os.path.join(data_dir, folder_name)

    # Output folders
    dataset_result_dir = os.path.join(result_root_dir, dataset_name)
    os.makedirs(dataset_result_dir, exist_ok=True)
    pca_dir = os.path.join(dataset_result_dir, "PCA")
    prc_dir = os.path.join(dataset_result_dir, "PRC")
    os.makedirs(pca_dir, exist_ok=True)
    os.makedirs(prc_dir, exist_ok=True)

    X_chunks, y_chunks = [], []

    for fname in os.listdir(dataset_path):
```

```
        if fname.startswith("X_feats_"):
            chunk_id = fname.split("_")[-1].replace(".npy", "")
            X = np.load(os.path.join(dataset_path, f"X_feats_{chunk_id}.npy"))
            y = np.load(os.path.join(dataset_path, f"y_labels_{chunk_id}.npy"))
            X_chunks.append(X)
            y_chunks.append(y)

    target_dim = min([x.shape[1] for x in X_chunks])
    # print(f" Aligning all feature chunks to {target_dim} dimensions...")

    X_aligned = []
    for x in X_chunks:
        if x.shape[1] > target_dim:
            x_trimmed = x[:, :target_dim]  # truncate extra columns
        elif x.shape[1] < target_dim:
            x_trimmed = np.pad(x, ((0, 0), (0, target_dim - x.shape[1])), mode='constant')  # pad with
    zeros
        else:
            x_trimmed = x
        X_aligned.append(x_trimmed)

    # Model config
    model_builder, param_space = model_configs["XGBoost"]

    for opt_name, OptClass in optimisers.items():
        print(f"\n Optimiser: {opt_name} on {dataset_name}")

        optimiser = build_optimiser(OptClass, model_builder, param_space)
        optimiser.fit(X_train, y_train, X_test, y_test)

        df_full = pd.DataFrame(optimiser.history)

        best_rows = []
        for iter_id, group in df_full.groupby("iteration"):
            min_cost = group["Cost Function"].min()
            lowest_cost_rows = group[group["Cost Function"] == min_cost]
            max_acc = lowest_cost_rows["accuracy"].max()
            best_acc_rows = lowest_cost_rows[lowest_cost_rows["accuracy"] == max_acc]
            best_row = best_acc_rows.loc[best_acc_rows["MAP"].idxmax()]
            best_row_dict = best_row.to_dict()
            best_row_dict["iteration"] = iter_id
            best_rows.append(best_row_dict)

        df = pd.DataFrame(best_rows).reset_index(drop=True)
        df.insert(0, "Feature Set", dataset_name)
        df.insert(1, "Optimiser", opt_name)

        keep_cols = [
            "Feature Set", "Optimiser", "iteration", "learning_rate", "max_depth", "n_estimators",
            "subsample", "colsample_bytree", "accuracy", "MAR", "FDR", "FPR", "MCC",
            "precision", "MAP", "Cost Function"
        ]
        df = df[[col for col in keep_cols if col in df.columns]]
        optimiser_summary.append(df)

        best_cost = df["Cost Function"].min()
        lowest_cost_df = df[df["Cost Function"] == best_cost]
        max_acc = lowest_cost_df["accuracy"].max()
        highest_acc_df = lowest_cost_df[lowest_cost_df["accuracy"] == max_acc]
        final_best_row = highest_acc_df.loc[highest_acc_df["MAP"].idxmax()]
        final_optimiser_results.append(final_best_row)

        fig_acc, ax_acc = plt.subplots()
        ax_acc.plot(optimiser.best_accuracy_per_iteration)
        ax_acc.set_title(f"{opt_name} Accuracy on {dataset_name}")
        ax_acc.set_xlabel("Iteration")
        ax_acc.set_ylabel("Accuracy")
        ax_acc.grid(True)
        fig_acc.tight_layout()
        fig_acc.savefig(os.path.join(dataset_result_dir, f"{opt_name}_accuracy_plot.png"))
        plt.close(fig_acc)

        fig_cost, ax_cost = plt.subplots()
        ax_cost.plot(optimiser.best_cost_per_iteration)
        ax_cost.set_title(f"{opt_name} Cost Function on {dataset_name}")
        ax_cost.set_xlabel("Iteration")
        ax_cost.set_ylabel("Cost")
```

```
        ax_cost.grid(True)
        fig_cost.tight_layout()
        fig_cost.savefig(os.path.join(dataset_result_dir, f"{opt_name}_cost_plot.png"))
        plt.close(fig_cost)

# Save results
summary_df = pd.concat(optimiser_summary, ignore_index=True)
summary_df.to_csv(os.path.join(result_root_dir, "all_optimiser_summary.csv"), index=False)

final_df = pd.DataFrame(final_optimiser_results)
final_df.to_csv(os.path.join(result_root_dir, "final_optimiser_results.csv"), index=False)

print("␣All␣optimiser␣results␣saved:")
print("␣-␣all_optimiser_summary.csv␣(best␣per␣iteration)")
print("␣-␣final_optimiser_results.csv␣(final␣best␣only)")
```

B. Signal Extraction and Feature Preparation from EEG Data Using Python

Listing 4: Python Implementation of 1D-CNN-Based Temporal Feature Extraction

```
# === Settings ===
save_dir = "/Users/myatpwintphyu/Desktop/eeg_1dcnn_features"  # where to save
os.makedirs(save_dir, exist_ok=True)

# === Step 1: Load labels ===
xlsx_path = "/Users/myatpwintphyu/Desktop/Monash/Master␣
    Thesis/Test_and_do_18_19_20/Data/ds003474-download/participants.xlsx"
df = pd.read_excel(xlsx_path)
df = df[df['age'] < 20]
df['label'] = (df['BDI'] > 10).astype(int)
subject_label_map = dict(zip(df['participant_id'], df['label']))

# === Step 2: Define 1D CNN feature extractor (built after shape is known) ===
def build_1dcnn_extractor(input_shape):
    inp = Input(shape=input_shape)
    x = Conv1D(64, kernel_size=5, activation='relu')(inp)
    x = Conv1D(128, kernel_size=5, activation='relu')(x)
    x = GlobalAveragePooling1D()(x)
    model = Model(inputs=inp, outputs=x)
    return model

# === Step 3: Load EEG and extract features subject-by-subject ===
root_dir = "/Users/myatpwintphyu/Desktop/Monash/Master␣
    Thesis/Test_and_do_18_19_20/Data/ds003474-download/Data"
subject_ids = list(subject_label_map.keys())
chunk_id = 0

for subject in subject_ids:
    set_path = os.path.join(root_dir, subject, "eeg", f"{subject}_task-ProbabilisticSelection_eeg.set")
    if not os.path.exists(set_path):
        print(f"␣Skipping␣missing:␣{subject}")
        continue

    try:
        print(f"␣Processing␣{subject}")
        raw = mne.io.read_raw_eeglab(set_path, preload=True)
        raw.filter(1., 50.)
        epochs = mne.make_fixed_length_epochs(raw, duration=1.0, overlap=0.5, preload=True)
        data = epochs.get_data()  # shape: (n_epochs, n_channels, n_times)
        label = subject_label_map[subject]

        if data.shape[0] == 0:
            print(f"␣No␣epochs␣for␣{subject}")
            continue

        # Build CNN on first run
        if chunk_id == 0:
            input_shape = data.shape[1:]  # (channels, time)
            cnn_model = build_1dcnn_extractor(input_shape)
```

```
        # Extract features
        feats = cnn_model.predict(data, batch_size=16)
        labels = np.full(len(feats), label)

        # Optionally Normalise features
        scaler = StandardScaler()
        feats = scaler.fit_transform(feats)

        # Save to disk
        np.save(os.path.join(save_dir, f"X_feats_{chunk_id}.npy"), feats)
        np.save(os.path.join(save_dir, f"y_labels_{chunk_id}.npy"), labels)
        print(f"␣Saved␣features␣for␣chunk␣{chunk_id}:␣{feats.shape}")
        chunk_id += 1

    except Exception as e:
        print(f"␣Error␣processing␣{subject}:␣{e}")
```

Listing 5: Python Code for 3D-CNN-Based Spatiotemporal Feature Extraction from Spectrograms

```
# === Save location for 3D CNN features ===
save_dir = "/Users/myatpwintphyu/Desktop/eeg_3dcnn_features"
os.makedirs(save_dir, exist_ok=True)

# === Load participant labels ===
xlsx_path = "/Users/myatpwintphyu/Desktop/Monash/Master␣
    Thesis/Test_and_do_18_19_20/Data/ds003474-download/participants.xlsx"
df = pd.read_excel(xlsx_path)
df = df[df['age'] < 20]
df['label'] = (df['BDI'] > 10).astype(int)
subject_label_map = dict(zip(df['participant_id'], df['label']))

# === Define 3D CNN Feature Extractor (Improved) ===
def build_3dcnn_extractor(input_shape):
    inp = Input(shape=input_shape)  # shape: (1, channels, time, 1)
    x = Conv3D(32, kernel_size=(1, 3, 3), activation='relu')(inp)
    x = BatchNormalization()(x)
    x = Conv3D(64, kernel_size=(1, 3, 3), activation='relu')(x)
    x = BatchNormalization()(x)
    x = Dropout(0.3)(x)
    x = Conv3D(128, kernel_size=(1, 3, 3), activation='relu')(x)
    x = GlobalAveragePooling3D()(x)
    model = Model(inputs=inp, outputs=x)
    return model

# === Loop over subjects and extract features ===
root_dir = "/Users/myatpwintphyu/Desktop/Monash/Master␣
    Thesis/Test_and_do_18_19_20/Data/ds003474-download/Data"
subject_ids = list(subject_label_map.keys())
chunk_id = 0

for subject in subject_ids:
    set_path = os.path.join(root_dir, subject, "eeg", f"{subject}_task-ProbabilisticSelection_eeg.set")
    if not os.path.exists(set_path):
        print(f"␣Missing␣file␣for:␣{subject}")
        continue

    try:
        print(f"␣Processing␣{subject}")
        raw = mne.io.read_raw_eeglab(set_path, preload=True)
        raw.filter(1., 50.)

        epochs = mne.make_fixed_length_epochs(raw, duration=2.0, overlap=1.0, preload=True)
        data = epochs.get_data()  # shape: (n_epochs, n_channels, n_times)
        label = subject_label_map[subject]

        if data.shape[0] == 0:
            print(f"␣No␣valid␣epochs␣for:␣{subject}")
            continue

        # Reshape for 3D CNN: (samples, 1, channels, time, 1)
        data = data[:, np.newaxis, :, :, np.newaxis]

        # Build model on first run
        if chunk_id == 0:
            input_shape = data.shape[1:]  # (1, channels, time, 1)
```

```
            cnn_model = build_3dcnn_extractor(input_shape)

        # Extract features
        feats = cnn_model.predict(data, batch_size=16, verbose=0)
        labels = np.full(len(feats), label)

        # Normalise features
        scaler = StandardScaler()
        feats = scaler.fit_transform(feats)

        # Save features and labels
        np.save(os.path.join(save_dir, f"X_feats_{chunk_id}.npy"), feats)
        np.save(os.path.join(save_dir, f"y_labels_{chunk_id}.npy"), labels)

        print(f"Saved␣chunk␣{chunk_id}:␣{feats.shape}")
        chunk_id += 1

    except Exception as e:
        print(f"Error␣processing␣{subject}:␣{e}")
```

Listing 6: Python-Based Spectral Feature Extraction Using Frequency-Domain Analysis

```
# === Settings ===
save_dir = "/Users/myatpwintphyu/Desktop/eeg_stft_features"
os.makedirs(save_dir, exist_ok=True)

fs = 256            # Sampling frequency (Hz)
win_size = 128      # Window size (samples)
overlap = 64        # Overlap (samples)

# EEG band limits (Hz)
bands = {
    "delta": (1, 4),
    "theta": (4, 8),
    "alpha": (8, 12),
    "beta": (12, 30),
    "gamma": (30, 50)
}

# === Load participant labels ===
xlsx_path = "/Users/myatpwintphyu/Desktop/Monash/Master␣
    Thesis/Test_and_do_18_19_20/Data/ds003474-download/participants.xlsx"
df = pd.read_excel(xlsx_path)
df = df[df['age'] < 20]
df['label'] = (df['BDI'] > 10).astype(int)
subject_label_map = dict(zip(df['participant_id'], df['label']))

# === STFT-based power feature extractor ===
def extract_stft_features(epoch_data, fs):
    n_channels, n_samples = epoch_data.shape
    features = []
    for ch in range(n_channels):
        f, t, Zxx = stft(epoch_data[ch], fs=fs, window='hamming', nperseg=win_size, noverlap=overlap)
        power = np.abs(Zxx) ** 2  # Power spectrogram

        # Average power per frequency band
        band_powers = []
        for name, (low, high) in bands.items():
            band_mask = (f >= low) & (f <= high)
            band_power = power[band_mask, :].mean()
            band_powers.append(band_power)
        features.extend(band_powers)
    return np.array(features)

# === Loop over subjects and extract STFT features ===
root_dir = "/Users/myatpwintphyu/Desktop/Monash/Master␣
    Thesis/Test_and_do_18_19_20/Data/ds003474-download/Data"
subject_ids = list(subject_label_map.keys())
chunk_id = 0

for subject in subject_ids:
    set_path = os.path.join(root_dir, subject, "eeg", f"{subject}_task-ProbabilisticSelection_eeg.set")
    if not os.path.exists(set_path):
        print(f"␣Missing␣file␣for:␣{subject}")
```

```
        continue

    try:
        print(f"␣Processing␣{subject}")
        raw = mne.io.read_raw_eeglab(set_path, preload=True)
        raw.filter(1., 50.)
        raw.resample(fs)  # Downsample to match STFT frequency

        # Epoch duration = 2s
        epochs = mne.make_fixed_length_epochs(raw, duration=2.0, overlap=1.0, preload=True)
        data = epochs.get_data()  # shape: (n_epochs, n_channels, n_samples)
        label = subject_label_map[subject]

        if data.shape[0] == 0:
            print(f"␣No␣valid␣epochs␣for:␣{subject}")
            continue

        feats = []
        for i in range(data.shape[0]):
            fvec = extract_stft_features(data[i], fs)
            feats.append(fvec)
        feats = np.array(feats)
        labels = np.full(len(feats), label)

        # Normalise features
        scaler = StandardScaler()
        feats = scaler.fit_transform(feats)

        # Save to disk
        np.save(os.path.join(save_dir, f"X_feats_{chunk_id}.npy"), feats)
        np.save(os.path.join(save_dir, f"y_labels_{chunk_id}.npy"), labels)
        print(f"␣Saved␣STFT␣features:␣{feats.shape}")

        chunk_id += 1

    except Exception as e:
        print(f"␣Error␣processing␣{subject}:␣{e}")
```

Listing 7: Python Script for Beta Band Feature Extraction from EEG Signals

```
# === Settings ===
save_dir = "/Users/myatpwintphyu/Desktop/eeg_stft_beta_features"
os.makedirs(save_dir, exist_ok=True)

# === Label file ===
xlsx_path = "/Users/myatpwintphyu/Desktop/Monash/Master␣
    Thesis/Test_and_do_18_19_20/Data/ds003474-download/participants.xlsx"
df = pd.read_excel(xlsx_path)
df = df[df['age'] < 20]
df['label'] = (df['BDI'] > 10).astype(int)
subject_label_map = dict(zip(df['participant_id'], df['label']))

# === STFT settings ===
fs = 250  # sampling frequency (Hz)
win_size = 128  # window size (samples)
overlap = 64  # overlap (samples)
beta_range = (12, 30)

def extract_beta_power(data_epoch, fs):
    n_channels, n_times = data_epoch.shape
    beta_power = []

    for ch in range(n_channels):
        f, t, Zxx = stft(data_epoch[ch], fs=fs, nperseg=win_size, noverlap=overlap, window='hamming')
        power = np.abs(Zxx) ** 2
        # Filter frequencies in Beta range
        beta_mask = (f >= beta_range[0]) & (f <= beta_range[1])
        beta_band_power = power[beta_mask].mean(axis=0)  # average over beta freqs
        beta_power.append(beta_band_power.mean())  # mean over time

    return np.array(beta_power)

# === Main extraction loop ===
data_root = "/Users/myatpwintphyu/Desktop/Monash/Master␣
    Thesis/Test_and_do_18_19_20/Data/ds003474-download/Data"
```

```
subject_ids = list(subject_label_map.keys())
chunk_id = 0

for subject in subject_ids:
    set_path = os.path.join(data_root, subject, "eeg", f"{subject}_task-ProbabilisticSelection_eeg.set")
    if not os.path.exists(set_path):
        print(f"␣Missing␣file␣for:␣{subject}")
        continue

    try:
        print(f"␣Processing␣{subject}")
        raw = mne.io.read_raw_eeglab(set_path, preload=True)
        raw.filter(1., 50.)
        epochs = mne.make_fixed_length_epochs(raw, duration=2.0, overlap=1.0, preload=True)
        data = epochs.get_data()  # (n_epochs, n_channels, n_times)
        label = subject_label_map[subject]

        if data.shape[0] == 0:
            print(f"␣No␣valid␣epochs␣for␣{subject}")
            continue

        features = []
        for epoch in data:
            beta_feat = extract_beta_power(epoch, fs)
            features.append(beta_feat)

        feats = np.array(features)
        labels = np.full(len(feats), label)

        # Normalise
        scaler = StandardScaler()
        feats = scaler.fit_transform(feats)

        # Save
        np.save(os.path.join(save_dir, f"X_feats_{chunk_id}.npy"), feats)
        np.save(os.path.join(save_dir, f"y_labels_{chunk_id}.npy"), labels)
        print(f"␣Saved␣chunk␣{chunk_id}:␣{feats.shape}")
        chunk_id += 1

    except Exception as e:
        print(f"␣Error␣processing␣{subject}:␣{e}")
```

C. Hyperparameter Optimiser Code (Python)

Listing 8: Elitist Jellyfish-Based Hyperparameter Optimiser Python Implementation

```
def compute_metrics(y_true, y_pred, y_proba):
    acc = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    mcc = matthews_corrcoef(y_true, y_pred)
    tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
    fdr = fp / (fp + tp) if (fp + tp) > 0 else 0
    fpr = fp / (fp + tn) if (fp + tn) > 0 else 0
    logloss = log_loss(y_true, y_proba)
    map_score = average_precision_score(y_true, y_proba)
    return acc, recall, fdr, fpr, mcc, precision, logloss, map_score

class JASAOptimiser:
    def __init__(
        self,
        model_class,
        param_space,
        max_iter=25,
        population_size=5,
        temperature=1.0,
        metric_to_optimise='logloss'
    ):
        self.model_class = model_class
        self.param_space = param_space
        self.max_iter = max_iter
```

```
        self.population_size = population_size
        self.temperature = temperature
        self.metric_to_optimise  = metric_to_optimise
        self.best_params = None
        self.best_score = np.inf  # for logloss
        self.history = []
        self.best_accuracy_per_iteration = []
        self.best_cost_per_iteration = []

    def _sample_individual(self):
        return {
            'learning_rate': np.random.uniform(*self.param_space['learning_rate']),
            'max_depth': np.random.randint(*self.param_space['max_depth']),
            'n_estimators': np.random.randint(*self.param_space['n_estimators']),
            'subsample': np.random.uniform(*self.param_space['subsample']),
            'colsample_bytree': np.random.uniform(*self.param_space['colsample_bytree']),
        }

    def _sample_population(self):
        return [self._sample_individual() for _ in range(self.population_size)]

    def _jellyfish_move(self, individual):
        new_individual = {}
        for key in individual:
            if key in ['max_depth', 'n_estimators']:
                move = np.random.choice([-1, 1]) * np.random.randint(1, 3)
                new_value = int(individual[key] + move)
                low, high = self.param_space[key]
                new_individual[key] = int(np.clip(new_value, low, high))
            else:
                move = np.random.normal(0, 0.02)
                new_value = individual[key] + move
                low, high = self.param_space[key]
                new_individual[key] = float(np.clip(new_value, low, high))
        return new_individual

    def fit(self, X_train, y_train, X_test, y_test):
        population = self._sample_population()
        best_individual = None
        best_score = np.inf

        for i in range(self.max_iter):
            print(f"\n␣Iteration␣{i+1}/{self.max_iter}")
            scores = []

            for j, individual in enumerate(population):
                model = self.model_class(**individual)
                model.fit(X_train, y_train)
                y_pred = model.predict(X_test)
                y_proba = model.predict_proba(X_test)[:, 1]

                acc, recall, fdr, fpr, mcc, precision, logloss, map_score = compute_metrics(y_test,
y_pred, y_proba)
                score = (1.0 / (acc + 0.5 * map_score + 0.3 * mcc + 0.3 * precision + 1e-8)) + (0.3 * (fdr
+ fpr))

                print(f"[{j}]␣|␣Accuracy␣=␣{acc:.4f}␣|␣Cost␣Function␣=␣{score:.4f}␣|␣MAP␣=␣{map_score:.4f}␣
")

                scores.append((individual, acc, score))

                if score < best_score:
                    best_score = score
                    best_individual = individual

                self.history.append({
                    'iteration': i,
                    **individual,
                    'accuracy': acc,
                    'MAR': recall,
                    'FDR': fdr,
                    'FPR': fpr,
                    'MCC': mcc,
                    'precision': precision,
                    'logloss': logloss,
                    'MAP': map_score,
                    'Cost␣Function': score
```

```
                })

            # Apply strict elitism
            new_population = []
            for individual, acc, score in scores:
                if individual == best_individual:
                    new_population.append(individual)  # preserve elite
                else:
                    new_population.append(self._jellyfish_move(individual))
            population = new_population

            model_best = self.model_class(**best_individual)
            model_best.fit(X_train, y_train)
            y_pred_best = model_best.predict(X_test)
            y_proba_best = model_best.predict_proba(X_test)[:, 1]
            acc_best, recall, fdr, fpr, mcc, precision, logloss, map_score = compute_metrics(y_test,
    y_pred_best, y_proba_best)

            score_best = (1.0 / (acc_best + 0.5 * map_score + 0.3 * mcc + 0.3 * precision + 1e-8)) + (0.3
    * (fdr + fpr))

            self.best_accuracy_per_iteration.append(acc_best)
            self.best_cost_per_iteration.append(score_best)

        self.best_params = best_individual
        self.best_score = best_score

    def get_best_model(self):
        return self.model_class(**self.best_params)
```

Listing 9: Elitist Simulated Annealing-based Hyperparameter Optimiser Python Implementation

```
def compute_metrics(y_true, y_pred, y_proba):
    acc = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    mcc = matthews_corrcoef(y_true, y_pred)
    tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
    fdr = fp / (fp + tp) if (fp + tp) > 0 else 0
    fpr = fp / (fp + tn) if (fp + tn) > 0 else 0
    logloss = log_loss(y_true, y_proba)
    map_score = average_precision_score(y_true, y_proba)
    return acc, recall, fdr, fpr, mcc, precision, logloss, map_score

class DHOA_SA_Optimiser:
    def __init__(self, model_class, param_space, max_iter=30, population_size=5, temperature=1.0):
        self.model_class = model_class
        self.param_space = param_space
        self.max_iter = max_iter
        self.population_size = population_size
        self.temperature = temperature
        self.best_params = None
        self.best_score = np.inf
        self.history = []
        self.best_accuracy_per_iteration = []
        self.best_cost_per_iteration = []

    def _sample_individual(self):
        return {
            'learning_rate': np.random.uniform(*self.param_space['learning_rate']),
            'max_depth': np.random.randint(*self.param_space['max_depth']),
            'n_estimators': np.random.randint(*self.param_space['n_estimators']),
            'subsample': np.random.uniform(*self.param_space['subsample']),
            'colsample_bytree': np.random.uniform(*self.param_space['colsample_bytree']),
        }

    def _explore_exploit_sa(self, individual, global_best):
        new_individual = {}
        for key in individual:
            if key in ['max_depth', 'n_estimators']:
                delta = int(0.3 * (global_best[key] - individual[key]) + np.random.randint(-3, 4))
                new_test = int(individual[key] + delta)
                low, high = self.param_space[key]
                new_individual[key] = int(np.clip(new_test, low, high))
            else:
```

```
            delta = 0.3 * (global_best[key] - individual[key]) + np.random.normal(0, 0.05)
            new_test = individual[key] + delta
            low, high = self.param_space[key]
            new_individual[key] = float(np.clip(new_test, low, high))
    return new_individual

def fit(self, X_train, y_train, X_test, y_test):
    population = [self._sample_individual() for _ in range(self.population_size)]
    best_individual = None

    for i in range(self.max_iter):
        print(f"\n␣DHOA-SA␣Iteration␣{i+1}/{self.max_iter}")

        scores = []

        for j, individual in enumerate(population):
            model = self.model_class(**individual)
            model.fit(X_train, y_train)
            y_pred = model.predict(X_test)
            y_proba = model.predict_proba(X_test)[:, 1]

            acc, recall, fdr, fpr, mcc, precision, logloss, map_score = compute_metrics(y_test,
y_pred, y_proba)
            score = (1.0 / (acc + 0.5 * map_score + 0.3 * mcc + 0.3 * precision + 1e-8)) + (0.3 * (fdr
+ fpr))

            print(f"[{j}]␣|␣Accuracy␣=␣{acc:.4f}␣|␣Score␣=␣{score:.6f}␣|␣MAP␣=␣{map_score:.4f}")

            scores.append((individual, acc, score))

            if score < self.best_score:
                self.best_score = score
                self.best_params = individual
                best_individual = individual

            self.history.append({
                'iteration': i,
                **individual,
                'accuracy': acc,
                'MAR': recall,
                'FDR': fdr,
                'FPR': fpr,
                'MCC': mcc,
                'precision': precision,
                'logloss': logloss,
                'MAP': map_score,
                'Cost␣Function': score
            })

        # === Strong elitism ===
        new_population = []
        for individual, acc, score in scores:
            if individual == best_individual:
                new_population.append(individual)
            else:
                new_population.append(self._explore_exploit_sa(individual, best_individual))
        population = new_population

        model_best = self.model_class(**best_individual)
        model_best.fit(X_train, y_train)
        y_pred_best = model_best.predict(X_test)
        y_proba_best = model_best.predict_proba(X_test)[:, 1]

        acc_best, recall, fdr, fpr, mcc, precision, logloss, map_score = compute_metrics(y_test,
y_pred_best, y_proba_best)
        score_best = (1.0 / (acc_best + 0.5 * map_score + 0.3 * mcc + 0.3 * precision + 1e-8)) + (0.3
* (fdr + fpr))

        self.best_accuracy_per_iteration.append(acc_best)
        self.best_cost_per_iteration.append(score_best)

def get_best_model(self):
    return self.model_class(**self.best_params)
```

Listing 10: Elitist Invasive Weed Optimisation with Simulated Annealing Logic Optimiser Python Implementation

```
def compute_metrics(y_true, y_pred, y_proba):
    acc = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    mcc = matthews_corrcoef(y_true, y_pred)
    tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
    fdr = fp / (fp + tp) if (fp + tp) > 0 else 0
    fpr = fp / (fp + tn) if (fp + tn) > 0 else 0
    logloss = log_loss(y_true, y_proba)
    map_score = average_precision_score(y_true, y_proba)
    return acc, recall, fdr, fpr, mcc, precision, logloss, map_score

class IWO_SA_Optimiser:
    def __init__(self, model_class, param_space, max_iter=30, initial_pop=5, max_pop=20, temperature=1.0):
        self.model_class = model_class
        self.param_space = param_space
        self.max_iter = max_iter
        self.initial_pop = initial_pop
        self.max_pop = max_pop
        self.temperature = temperature
        self.best_params = None
        self.best_score = np.inf  # minimizing custom cost
        self.history = []
        self.best_accuracy_per_iteration = []
        self.best_cost_per_iteration = []

    def _sample_individual(self):
        return {
            'learning_rate': np.random.uniform(*self.param_space['learning_rate']),
            'max_depth': np.random.randint(*self.param_space['max_depth']),
            'n_estimators': np.random.randint(*self.param_space['n_estimators']),
            'subsample': np.random.uniform(*self.param_space['subsample']),
            'colsample_bytree': np.random.uniform(*self.param_space['colsample_bytree']),
        }

    def _mutate(self, individual, step_size=0.1):
        mutant = individual.copy()
        for key in mutant:
            if key in ['max_depth', 'n_estimators']:
                mutation = int(np.random.choice([-1, 1]) * np.random.randint(1, 3))
                mutant[key] = int(np.clip(mutant[key] + mutation, *self.param_space[key]))
            else:
                mutation = np.random.normal(0, step_size)
                mutant[key] = float(np.clip(mutant[key] + mutation, *self.param_space[key]))
        return mutant

    def fit(self, X_train, y_train, X_test, y_test):
        population = [self._sample_individual() for _ in range(self.initial_pop)]
        best_individual = None

        for i in range(self.max_iter):
            print(f"\n IWO-SA Iteration {i+1}/{self.max_iter}")

            evaluated = []

            for individual in population:
                model = self.model_class(**individual)
                model.fit(X_train, y_train)
                y_pred = model.predict(X_tes)
                y_proba = model.predict_proba(X_tes)[:, 1]

                acc, recall, fdr, fpr, mcc, precision, logloss, map_score = compute_metrics(y_tes, y_pred,
    y_proba)
                score = (1.0 / (acc + 0.5 * map_score + 0.3 * mcc + 0.3 * precision + 1e-8)) + (0.3 * (fdr
    + fpr))

                print(f"Accuracy = {acc:.4f} | Cost Function = {score:.6f} | MAP = {map_score:.4f}")

                if score < self.best_score:
                    self.best_score = score
                    self.best_params = individual
                    best_individual = individual

                self.history.append({
                    'iteration': i,
                    **individual,
                    'accuracy': acc,
```

```
                    'MAR': recall,
                    'FDR': fdr,
                    'FPR': fpr,
                    'MCC': mcc,
                    'precision': precision,
                    'logloss': logloss,
                    'MAP': map_score,
                    'Cost␣Function': score
                })

                num_offspring = np.random.randint(1, 4)
                for _ in range(num_offspring):
                    child = self._mutate(individual, step_size=0.1)
                    evaluated.append(child)

            # Add parent individuals too
            evaluated += population

            # Evaluate all candidates
            evaluated_with_scores = []
            for candidate in evaluated:
                model = self.model_class(**candidate)
                model.fit(X_train, y_train)
                y_pred = model.predict(X_test)
                y_proba = model.predict_proba(X_test)[:, 1]
                acc, recall, fdr, fpr, mcc, precision, logloss, map_score = compute_metrics(y_test,
    y_pred, y_proba)
                score = (1.0 / (acc + 0.5 * map_score + 0.3 * mcc + 0.3 * precision + 1e-8)) + (0.3 * (fdr
    + fpr))

                evaluated_with_scores.append((score, acc, candidate))

            evaluated_with_scores.sort(key=lambda x: (x[0], -x[1]))  # prioritize lowest cost, then
    highest accuracy
            population = [x[2] for x in evaluated_with_scores[:self.max_pop]]

            #Save best of this generation
            best_score_gen, best_acc_gen, _ = evaluated_with_scores[0]
            self.best_accuracy_per_iteration.append(best_acc_gen)
            self.best_cost_per_iteration.append(best_score_gen)

    def get_best_model(self):
        return self.model_class(**self.best_params)
```

Listing 11: Elitist Crisscross Mutation Optimiser Python Implementation

```
def compute_metrics(y_true, y_pred, y_proba):
    acc = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    mcc = matthews_corrcoef(y_true, y_pred)
    tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
    fdr = fp / (fp + tp) if (fp + tp) > 0 else 0
    fpr = fp / (fp + tn) if (fp + tn) > 0 else 0
    logloss = log_loss(y_true, y_proba)
    map_score = average_precision_score(y_true, y_proba)
    return acc, recall, fdr, fpr, mcc, precision, logloss, map_score

class COSOptimiser:
    def __init__(self, model_class, param_space, max_iter=30, population_size=10, temperature=1.0):
        self.model_class = model_class
        self.param_space = param_space
        self.max_iter = max_iter
        self.population_size = population_size
        self.temperature = temperature
        self.best_params = None
        self.best_score = np.inf
        self.history = []
        self.best_accuracy_per_iteration = []
        self.best_cost_per_iteration = []

    def _sample_individual(self):
        return {
            'learning_rate': np.random.uniform(*self.param_space['learning_rate']),
            'max_depth': np.random.randint(*self.param_space['max_depth']),
```

```
            'n_estimators': np.random.randint(*self.param_space['n_estimators']),
            'subsample': np.random.uniform(*self.param_space['subsample']),
            'colsample_bytree': np.random.uniform(*self.param_space['colsample_bytree']),
        }

    def _crisscross_move(self, individual):
        new_individual = {}
        for key in individual:
            if key in ['max_depth', 'n_estimators']:
                move = np.random.choice([-2, -1, 1, 2])
                new_value = int(individual[key] + move)
                low, high = self.param_space[key]
                new_individual[key] = int(np.clip(new_value, low, high))
            else:
                move = np.random.normal(0, 0.05)
                new_value = individual[key] + move
                low, high = self.param_space[key]
                new_individual[key] = float(np.clip(new_value, low, high))
        return new_individual

    def fit(self, X_train, y_train, X_test, y_test):
        population = [self._sample_individual() for _ in range(self.population_size)]
        best_individual = None

        for i in range(self.max_iter):
            print(f"\nCOS␣Iteration␣{i+1}/{self.max_iter}")
            scores = []

            for j, individual in enumerate(population):
                model = self.model_class(**individual)
                model.fit(X_train, y_train)
                y_pred = model.predict(X_test)
                y_proba = model.predict_proba(X_test)[:, 1]

                acc, recall, fdr, fpr, mcc, precision, logloss, map_score = compute_metrics(y_test,
y_pred, y_proba)
                score = (1.0 / (acc + 0.5 * map_score + 0.3 * mcc + 0.3 * precision + 1e-8)) + (0.3 * (fdr
+ fpr))

                print(f"[{j}]␣|␣Accuracy␣=␣{acc:.4f}␣|␣Cost␣Function␣=␣{score:.6f}␣|␣MAP␣=␣
{map_score:.4f}")

                scores.append((score, acc, individual))

                if score < self.best_score:
                    self.best_score = score
                    self.best_params = individual
                    best_individual = individual

                self.history.append({
                    'iteration': i,
                    **individual,
                    'accuracy': acc,
                    'MAR': recall,
                    'FDR': fdr,
                    'FPR': fpr,
                    'MCC': mcc,
                    'precision': precision,
                    'logloss': logloss,
                    'MAP': map_score,
                    'Cost␣Function': score
                })

            # Apply elitism: keep best individual
            new_population = []
            for individual in population:
                if individual == best_individual:
                    new_population.append(individual)
                else:
                    new_population.append(self._crisscross_move(individual))

            population = new_population

            # Save best accuracy and best cost for plotting
            best_score_iter, best_acc_iter, _ = min(scores, key=lambda x: (x[0], -x[1]))
            self.best_accuracy_per_iteration.append(best_acc_iter)
            self.best_cost_per_iteration.append(best_score_iter)
```

```
    def get_best_model(self):
        return self.model_class(**self.best_params)
```

Listing 12: litist Hybrid COS-IWO-Based Optimiser Python Implementation

```
def compute_metrics(y_true, y_pred, y_proba):
    acc = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    mcc = matthews_corrcoef(y_true, y_pred)
    tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
    fdr = fp / (fp + tp) if (fp + tp) > 0 else 0
    fpr = fp / (fp + tn) if (fp + tn) > 0 else 0
    logloss = log_loss(y_true, y_proba)
    map_score = average_precision_score(y_true, y_proba)
    return acc, recall, fdr, fpr, mcc, precision, logloss, map_score

class COIWSO_SA_Optimiser:
    def __init__(self, model_class, param_space, max_iter=30, population_size=5, temperature=1.0,
    metric_to_optimise ='logloss'):
        self.model_class = model_class
        self.param_space = param_space
        self.max_iter = max_iter
        self.population_size = population_size
        self.temperature = temperature
        self.metric_to_optimise  = metric_to_optimise
        self.best_params = None
        self.best_score = np.inf
        self.history = []
        self.best_accuracy_per_iteration = []
        self.best_cost_per_iteration = []

    def _sample_individual(self):
        return {
            'learning_rate': np.random.uniform(*self.param_space['learning_rate']),
            'max_depth': np.random.randint(*self.param_space['max_depth']),
            'n_estimators': np.random.randint(*self.param_space['n_estimators']),
            'subsample': np.random.uniform(*self.param_space['subsample']),
            'colsample_bytree': np.random.uniform(*self.param_space['colsample_bytree']),
        }

    def _cos_operator(self, p1, p2):
        child = {}
        for key in p1:
            child[key] = p1[key] if np.random.rand() < 0.5 else p2[key]
        for key in child:
            if key in ['max_depth', 'n_estimators']:
                mutation = int(np.random.choice([-1, 1]) * np.random.randint(1, 3))
                child[key] = int(np.clip(child[key] + mutation, *self.param_space[key]))
            else:
                mutation = np.random.normal(0, 0.05)
                child[key] = float(np.clip(child[key] + mutation, *self.param_space[key]))
        return child

    def _iwo_operator(self, individual):
        mutant = individual.copy()
        for key in mutant:
            if key in ['max_depth', 'n_estimators']:
                mutation = int(np.random.choice([-1, 1]) * np.random.randint(1, 4))
                mutant[key] = int(np.clip(mutant[key] + mutation, *self.param_space[key]))
            else:
                mutation = np.random.normal(0, 0.1)
                mutant[key] = float(np.clip(mutant[key] + mutation, *self.param_space[key]))
        return mutant

    def fit(self, X_train, y_train, X_test, y_test):
        population = [self._sample_individual() for _ in range(self.population_size)]

        for i in range(self.max_iter):
            print(f"\n COIWSO-SA Iteration {i+1}/{self.max_iter}")
            new_population = []
            evaluated = []

            for individual in population:
```

```
            model = self.model_class(**individual)
            model.fit(X_train, y_train)

            y_pred = model.predict(X_test)
            y_proba = model.predict_proba(X_test)[:, 1]

            acc, recall, fdr, fpr, mcc, precision, logloss, map_score = compute_metrics(y_test,
y_pred, y_proba)
            cost = (1.0 / (acc + 0.5 * map_score + 0.3 * mcc + 0.3 * precision + 1e-8)) + (0.3 * (fdr
+ fpr))

            print(f"␣Accuracy␣=␣{acc:.4f}␣|␣Cost␣Function␣=␣{cost:.6f}␣|␣MAP␣=␣{map_score:.4f}")

            evaluated.append((cost, acc, individual))

            if cost < self.best_score:
                self.best_score = cost
                self.best_params = individual

            self.history.append({
                'iteration': i,
                **individual,
                'accuracy': acc,
                'MAR': recall,
                'FDR': fdr,
                'FPR': fpr,
                'MCC': mcc,
                'precision': precision,
                'logloss': logloss,
                'MAP': map_score,
                'Cost␣Function': cost
            })

        # Elitism: keep best individual
        evaluated.sort(key=lambda x: (x[0]))  # lower cost is better
        elite_individual = evaluated[0][2]
        elite_accuracy = evaluated[0][1]
        elite_cost = evaluated[0][0]

        # Renew population using COS or IWO
        for _ in range(self.population_size - 1):
            D = np.random.rand()
            if D < 0.5:
                p1, p2 = np.random.choice(population, size=2, replace=False)
                candidate = self._cos_operator(p1, p2)
            else:
                parent = population[np.random.randint(0, len(population))]
                candidate = self._iwo_operator(parent)

            new_population.append(candidate)

        # Update population
        population = [elite_individual] + new_population

        # Save best per iteration
        self.best_accuracy_per_iteration.append(elite_accuracy)
        self.best_cost_per_iteration.append(elite_cost)

def get_best_model(self):
    return self.model_class(**self.best_params)
```

APPENDIX II
DATASETS USED IN THE STUDY

Dataset Title: EEG: Probabilistic Selection and Depression
Author: James F. Cavanagh
Year: 2021
Source: OpenNeuro
DOI: 10.18112/openneuro.ds003474.v1.1.0
Accessed: May 2024
License: Creative Commons CC0 (Public Domain Dedication)
Description: This dataset comprises EEG recordings collected during a probabilistic selection task designed to investigate the

neural correlates associated with depression.
Access Link: https://openneuro.org/datasets/ds003474