International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

A High-Level Ensemble Feature Selection Algorithm for Mitigating the Dimensionality in Stress Data

Mr. Prashant M. Suryavanshi*[1], Dr. Pradnya A. Vikhar [2]

Submitted: 15/11/2023 **Revised**: 27/12/2023 **Accepted**: 07/01/2024

Abstract: Stress is a common response to environmental and psychological factors, negatively impacting mental and physical health. Analyzing stress data with multiple features can reveal contributing factors and aid in developing effective stress management strategies. However, the large dimensionality poses challenges due to many features, leading to overfitting. Feature selection is crucial in mitigating this issue and improving machine learning model performance on stress data. This paper proposes a high-level ensemble feature selection (HLE-FS) algorithm for stress data. The algorithm aims to identify the most informative features relevant to stress classification, which can lead to a better understanding of the underlying factors contributing to stress and more accurate stress prediction. The proposed algorithm consists of several steps to preprocess the input stress data and apply different feature selection techniques. First, missing values in the data are imputed using hybrid imputation, and categorical variables are converted to numerical using categorical feature target encoding. The data is then normalized to ensure compatibility with machine learning algorithms. The algorithm applies three feature selection techniques in an ensemble approach, including filter-based, wrapper-based, and embedding-based methods. The filter-based feature selection technique uses information gain and ranker search to rank the features. The wrapper-based technique employs Naïve Bayes classifier and Greedy Stepwise search with ThreadPoolExecutor to search for the best feature subsets using a wrapper approach. Finally, the embedding-based technique uses Principal Component Analysis (PCA) to reduce the dimensionality of the data, and Ranker search to rank the PCA-derived features. The results of the three feature selection techniques are combined using a majority voting mechanism, and the top-k features are extracted from the combined results. The algorithm then evaluates the performance of the dataset with and without feature selection using a Random Forest classifier. Experimental results on stress data demonstrate that the proposed algorithm outperforms the existing system regarding the accuracy and computational efficiency. The algorithm effectively selects the most informative features from the input stress data, improving stress classification performance.

Keywords: *HLE-FS, Naïve Bayes, ThreadPoolExecutor, PCA.*

1. Introduction

Stress, a common response to environmental and psychological factors, negatively impacts mental and physical health [1]. Analyzing stress data, which often involves multiple features, can help reveal contributing factors and aid in the development of effective stress management strategies. However, the challenge of high dimensionality poses obstacles in stress data analysis due to the large number of features compared to instances, leading to overfitting. Feature selection is crucial in mitigating this issue and improving the performance of machine learning models on stress data [2].

Stress data capture physiological, psychological, and behavioural measures associated with stress, such as heart rate, cortisol levels, self-reported stress scales, and behavioural responses. Analyzing stress data can provide insights into the underlying factors contributing to stress, identify patterns and correlations, and facilitate the development of targeted interventions for stress management.

Analyzing stress data is important because it can help researchers and practitioners better understand stress's causes, consequences, and mechanisms [3]. Identifying the most informative features relevant to stress classification makes it possible to gain insights into the underlying factors contributing to stress, which can inform the development of more effective stress management strategies. Stress data analysis can also aid in identifying high-risk individuals or populations who may benefit from targeted interventions and can contribute to advancing stress research and clinical practice.

Feature selection is a critical step in mitigating the curse of dimensionality in stress data analysis [4]. The large dimensionality refers to the challenge of dealing with many features compared to instances in a dataset, which can lead to overfitting and decreased model performance. Feature selection involves identifying a subset of the most relevant features from the original feature set, which can lead to improved model performance, reduced

[1,2]Department of Computer Science and Engineering
[1] Research Scholar, Dr. A. P. J. Abdul Kalam University, Indore
[2] Research Supervisor, Dr. A. P. J. Abdul Kalam University, Indore, (M.P.), India.
E-mail Id: [1]sprashant1234@gmail.com, [2]pradnyav123@gmail.com
** Corresponding Author: Mr. Prashant M. Suryavanshi*
Email: sprashant1234@gmail.com

computational complexity, and enhanced interpretability of results. Selecting the most informative features can help improve the accuracy, robustness, and efficiency of machine learning models on stress data.

Existing feature selection systems for stress data typically employ individual feature selection techniques, such as filter-based, wrapper-based, or embedding-based methods, without taking advantage of the strengths of different techniques [5]. Filter-based methods rank features based on certain criteria, such as information gain or correlation, and select the top-k features. Wrapper-based methods use a search algorithm combined with a classifier to evaluate the performance of different feature subsets. Embedding-based methods, such as Principal Component Analysis (PCA), transform the original features into a lower-dimensional space while retaining the most important information. However, these existing systems may suffer limitations, such as limited accuracy, computational inefficiency, and the inability to handle missing values and categorical variables in stress data effectively.

To address these limitations, this paper proposes a high-level ensemble feature selection (HLE-FS) algorithm for stress data. The proposed algorithm aims to overcome the curse of dimensionality in stress data analysis by combining multiple feature selection techniques in an ensemble approach. The algorithm consists of several steps: data preprocessing, missing value imputation, categorical variable conversion, normalization, and application of filter-based, wrapper-based, and embedding-based techniques. The results of the three techniques are combined using a majority voting mechanism, and the top-k features are extracted from the combined results. The algorithm then evaluates the performance of the dataset with and without feature selection using Random Forest, a popular machine learning algorithm.

The proposed algorithm has several advantages over existing systems. First, it takes advantage of the strengths of different feature selection techniques in an ensemble approach, which can improve accuracy and robustness in selecting the most informative features from stress data. Second, it effectively handles missing values and categorical variables through imputation and conversion techniques. Finally, it utilizes a majority voting mechanism to combine the results of different techniques, which can lead to enhanced performance compared to individual techniques.

The paper is organized as follows. Section 2 provides an overview of related work on feature selection for stress data. Section 3 presents the proposed HLE-FS algorithm in detail, including the steps for data preprocessing and the ensemble feature selection techniques. Section 4 describes the experimental setup and presents the results and analysis of the algorithm's performance. Finally, Section 5 concludes the paper and highlights future research directions.

2 Related Works:

Alghowinem et al. [6] propose a framework for interpreting depression detection models by analyzing the commonly selected features using various feature selection methods. They extract 902 behavioural cues from speech behaviour, speech prosody, eye movement, and head pose from three real-world depression datasets. They then use 38 feature selection algorithms to select the most promising features for modelling depression detection. The results of their framework show that speech behaviour features, such as pauses, are the most distinctive features of the depression detection model. They also identify other strong feature groups from different modalities, such as speech prosody, eye activity, and head movement. Their framework provides an interpretation of the model and improves the accuracy of depression detection by using a small number of selected features, which can reduce processing time.

Lin et al. [7] designed a neural network model based on Long Short-Term Memory (LSTM) to recognize stress using thermal and RGB imaging features. They experiment with different hyper parameters, activation functions, and optimizers to improve the model. They also apply feature selection and bimodal distribution removal techniques. Finally, they compare their results with another research paper focusing on the same problem and dataset and discuss the reasons for any differences.

Majid et al. [8] propose a framework for classifying perceived stress using multimodal data acquired from physiological sensors, including electroencephalography (EEG), galvanic skin response (GSR), and photoplethysmography (PPG). They extract time and frequency domain features from these signals and use a frequency band selection algorithm to select the optimum EEG frequency subband. They also use a wrapper-based method for optimal feature selection. They perform stress level classification using three different classifiers fed with a fusion of the selected features from three modalities. They achieve significant accuracy in classifying stress levels.

Parsi et al. [9] propose a feature selection technique based on the minimal redundancy-maximal relevance method to identify an optimal combination of heart rate variability and breathing rate metrics for detecting stress in drivers. First, they use galvanic skin response to measure ground truth stress levels. They then use a support vector machine algorithm with a radial basis function kernel and selected features to predict stress levels. The proposed method achieves high accuracy in predicting stress in the target dataset.

Reddy et al. [10] focus on stress prediction in working IT professionals and propose machine learning techniques. The authors may have conducted a study where they collected data from IT professionals, such as their stress levels, and then used machine learning algorithms or techniques to analyze the data and make predictions about stress levels. They use various features related to work-related stress and apply machine learning algorithms to predict stress levels. The aim is to identify stress-prone employees and take appropriate measures to manage their stress.

Jaiswal et al. [11] propose a novel approach that leverages a combination of observed facial behaviour and self-reported personality scores as powerful features for training deep neural networks, enabling accurate prediction of depression and anxiety scores. Furthermore, they argue that considering personality traits and behavioural features extracted from faces can improve prediction performance.

Rashid et al. [12] review recent neuroimaging-based approaches for predicting mental illness using features from different neuroimaging modalities such as structural, functional, and diffusion magnetic resonance imaging data. They introduce the concept of “predictome”, which involves incorporating multiple brain network-based features into a predictive model to jointly estimate features unique to a specific disorder and predict subjects accordingly.

Mousavian et al. [13] focus on feature selection and handling imbalanced data in the context of depression detection using machine learning. They investigate the correlation between regional volumes of the brain and depression and explore various feature selection techniques along with resampling methods to handle imbalanced data. They compare the performance of Random Forests (RF) and support vector machines (SVM) for depression detection.

Tadesse et al. [14] propose using natural language processing (NLP) techniques, feature selection and machine learning approaches to analyze posts from Reddit social media forums to detect depression attitudes of online users. They identify a lexicon of more common terms among depressed accounts and evaluate the efficiency of their proposed method. They compare the performance of different classifiers, such as Support Vector Machine (SVM) and Multilayer Perceptron (MLP), for depression detection.

Saeed et al. [15] present a pioneering approach for classifying long-term stress by harnessing the power of resting state EEG signal recordings and state-of-the-art machine learning algorithms. They meticulously investigate various methodologies for feature selection and labelling of the EEG signals and rigorously evaluate the performance of diverse classifiers to achieve accurate stress classification.

The disadvantages of the existing works mentioned include the following:

- **Limited interpretation of the model:** While Alghowinem et al. [6], Lin et al. [7], and other works propose various frameworks and algorithms for stress detection, they may lack a comprehensive interpretation of the selected features and their relevance to stress classification. It can make it difficult to understand the underlying factors contributing to stress and may hinder the interpretability of the model.
- **Suboptimal feature selection techniques:** Some existing works may use a single feature selection technique or a limited set of techniques, which may not capture the optimal set of features for stress classification. It can result in suboptimal performance and reduced accuracy in stress prediction.
- **Handling missing values and categorical variables:** Existing works may not effectively handle missing values in the stress data or categorical variables, which can affect the quality of feature selection and classification results. It can lead to biased or incomplete feature selection and inaccurate stress prediction.

The high-level ensemble feature selection (HLE-FS) algorithm aims to tackle these disadvantages by incorporating multiple feature selection techniques in an ensemble approach. First, the algorithm preprocesses the stress data by handling missing values, converting categorical variables, and normalizing the data. Then, it applies three different feature selection techniques in an ensemble approach, including filter-based, wrapper-based, and embedding-based methods. It ensures a more comprehensive and robust selection of informative features relevant to stress classification.

3. High-level ensemble feature selection (HLE-FS) algorithm

Ensemble feature selection techniques have gained increasing attention in machine learning and data mining research due to their ability to improve the accuracy and stability of feature selection. High-level ensemble feature selection (HLE-FS) is a novel approach that combines multiple feature selection methods to obtain a robust and informative feature set for machine learning models. HLE-FS leverages the strengths of different feature selection methods, such as filter-based, wrapper-based, and embedding-based, to overcome their limitations and enhance feature selection performance.

The HLE-FS algorithm is designed to handle stress data. The goal is to identify a reduced set of informative features that can accurately classify individuals’ stress levels (e.g., low, medium, high). The algorithm incorporates a series of steps, including data preprocessing, filter-based feature selection, wrapper-based feature selection, embedding-based feature selection, and ensemble feature selection. The final feature set obtained through HLE-FS is then evaluated using a machine learning model like Random Forest to assess its performance in stress data classification.

The key motivation behind the HLE-FS algorithm is to harness the complementary strengths of different feature selection methods to improve the accuracy and robustness of stress data classification. HLE-FS aims to select a reduced feature set that captures the most informative and relevant features for stress classification by combining filter-based, wrapper-based, and embedding-based methods in a majority voting ensemble approach. The algorithm also incorporates parallelization techniques to enhance computational efficiency and utilizes statistical analysis for performance evaluation and robustness assessment. Algorithm 1 shows the proposed HLE-FS algorithm in detail.

Algorithm 1: High-level ensemble feature selection (HLE-FS) algorithm

Input : Stress data with n instances and m features

Target variable indicating the level of stress (e.g. low, medium, high)

Output : A reduced feature set that is informative for stress classification

Step 1 : Load the stress data set.

Step 2 : Check for missing values in the dataset.

Step 3 : If there are missing values, impute them using a hybrid imputation. // **Algorithm 2**

Step 4 : Convert any categorical features in the data to numerical ones using a categorical feature target encoding. // **Algorithm 3**

Step 5 : Normalize the data using a standardization technique:

For each feature:

a. Compute the mean (average) and standard deviation (SD) of the values in the feature.

b. Subtract the mean from each value in the feature.

c. Divide the result from step b by the standard deviation to obtain the standardized values.

d. Update the data by replacing the original values in the feature with the corresponding standardized values.

Step 6 : Apply filter-based feature selection using InfoGainAttributeEval and Ranker search:

- FS_result_1 = Apply_InfoGainAttributeEval(Normalized_data)
- FS_result_1 = Ranker_search(FS_result_1)

Step 7 : Apply wrapper-based feature selection using ClassifierSubsetEval with Naïve Bayes and GreedyStepwise with ThreadPoolExecutor: // **Algorithm 4**

- FS_result_2 = Apply_ClassifierSubsetEval_with_Naive_Bayes(Normalized_data)
- FS_result_2 = GreedyStepwise_search(FS_result_2, limit=10, parallel=True)

Step 8 : Apply embedding-based feature selection using PCA and Ranker Search:

- FS_result_3 = Apply_PCA(Normalized_data)
- FS_result_3 = Ranker_search(FS_result_3)

Step 9 : Perform majority voting among the three feature selection results: // **Algorithm 5**

- FS_result = Majority_Voting(FS_result_1, FS_result_2, FS_result_3)

Step 10 : Extract top-k features from the FS_result:

- Final_FS_result = Extract_top_k_features(FS_result, k)

Step 11 : Evaluate the performance of the dataset with and without feature selection using Random Forest:

- Evaluate_with_Random_Forest(Normalized_data, Final_FS_result)

The HLE-FS algorithm is a step-by-step process for reducing the feature set of stress data to select informative features for stress classification. The algorithm takes stress data with n instances and m features as input and aims to identify a reduced feature set that can be used for accurate stress classification.

The algorithm begins by loading the stress data set and checking for missing values in Step 1 and Step 2, respectively. If there are missing values, they are imputed using a hybrid imputation technique in Step 3. Then, in Step 4, any categorical variables in the data are converted to numerical using a categorical feature target encoding technique. Next, in Step 5, the data is normalized using a standardization technique to ensure all features are on a similar scale.

Filter-based feature selection is then applied in Step 6 using the InfoGainAttributeEval method, followed by a Ranker search to rank the features based on their importance.

In Step 7, wrapper-based feature selection is applied using the ClassifierSubsetEval method with Naïve Bayes as the classifier and a GreedyStepwise search with ThreadPoolExecutor for parallel processing. Finally, in Step 8, embedding-based feature selection is applied using Principal Component Analysis (PCA) followed by a Ranker search to rank the features.

In Step 9, a majority voting mechanism is applied among the three feature selection results obtained from Steps 6, 7, and 8 to determine the most important features. Then, in Step 10, the top-k features are extracted from the combined feature selection results in Step 9, where k is a predefined value. Finally, in Step 11, the performance of the dataset is evaluated using a Random Forest classifier with both the original oversampled data and the reduced feature set obtained from Step 10 to compare the performance of the dataset with and without feature selection.

3.1 Hybrid imputation algorithm:

The Hybrid Imputation algorithm imputes missing values in a dataset using a combination of mean and K-nearest neighbours (KNN) imputation. The Hybrid Imputation algorithm is needed to address the limitations of existing imputation methods, such as mean imputation and KNN imputation. For example, mean imputation may result in biased imputed values and not account for local patterns or trends in the data. In contrast, KNN imputation relies heavily on the choice of the K value and can be computationally expensive. The Hybrid Imputation algorithm combines the strengths of mean imputation and KNN imputation to overcome these limitations and provide a more accurate and flexible imputation approach.

3.1.1 Advantages of Hybrid Imputation:

Specifically, the Hybrid Imputation algorithm has the following advantages:

- The Hybrid Imputation algorithm combines the strengths of mean imputation and KNN imputation, addressing their limitations. Mean imputation provides a simple and quick initial imputation, while KNN imputation refines the imputed values based on the values of nearby data points.
- The Hybrid Imputation algorithm takes advantage of the local patterns and trends in the data through KNN imputation, which can lead to more accurate imputations than global mean imputation.
- The Hybrid Imputation algorithm can handle missing values in datasets with different characteristics, such as datasets with extreme values, sparse data points, or irregular data distributions, by combining the strengths of mean imputation and KNN imputation.
- The Hybrid Imputation algorithm balances computational efficiency and imputation accuracy by using mean imputation as an initial step, followed by KNN imputation for refinement, which can be computationally more efficient than applying KNN imputation to the entire dataset.

3.1.2 Implementation of Hybrid Imputation:

The input to the hybrid imputation algorithm is a dataset with missing values, parameters such as the K value for KNN imputation (i.e., the number of nearest neighbours to consider) and a threshold for defining “nearby” data points.

The Hybrid Imputation algorithm begins by identifying the features in the dataset that have missing values. Then, for each feature with missing values, the algorithm calculates the mean of the available values for that feature. The calculated mean is then used to replace the missing values in that feature.

Next, the algorithm applies KNN imputation to refine further the imputed values based on the values of nearby data points. Finally, the algorithm calculates the distance to all other data points in the dataset for each missing value using a metric such as Euclidean distance. Then, the nearest K data points, where K is the predefined value for KNN imputation, are selected based on the calculated distances.

The algorithm then takes the average of the values of the K nearest data points for each missing value. This average is used to refine the initial mean-imputed value. It refined the KNN-imputed value is then used to replace the initial mean-imputed value for each feature with missing values.

These steps of applying mean imputation first and then refining with KNN imputation are repeated for all features with missing values in the dataset. Finally, the algorithm outputs the dataset with imputed values using the combined approach of mean imputation and KNN imputation, providing a more accurate imputation of missing values. Algorithm 2 discussed the proposed hybrid imputation.

Algorithm 2: Hybrid Imputation

Input	:	Dataset with missing values K value for KNN imputation (number of nearest neighbours to consider) The threshold for defining “nearby” data points
Output	:	Dataset with imputed values using combined mean imputation and KNN imputation
Step 1	:	Identify the features with missing values in the dataset.
Step 2	:	For each feature with missing values, calculate the mean of the available values for that feature.
Step 3	:	Replace the missing values with the calculated mean for each feature.
Step 4	:	For each feature with missing values, apply KNN imputation to refine further the imputed values based on the values of nearby data points.
Step 5	:	Calculate the distance to all other data points in the dataset for each missing value using a metric such as Euclidean distance.
Step 6	:	Select the K nearest data points based on the calculated distances, where K is the predefined K value for KNN imputation.
Step 7	:	Take the average of the K nearest data points for each missing value.
Step 8	:	Replace the initial mean-imputed value with the refined KNN-imputed value for each feature with missing values.
Step 9	:	Repeat steps 4-8 for all features with missing values.
Step 10	:	Output the dataset with imputed values using the combined mean and KNN imputations.

3.2 Categorical Feature Target Encoding:

Categorical feature target encoding, also known as target-based encoding, converts categorical features into numerical representations based on their relationship with the target feature in a supervised machine learning setting. It involves encoding categorical features using the mean of the target feature to create numerical labels for each category in the categorical feature.

The basic idea behind target encoding is to capture the relationship between the categorical and target features, which can be useful for predictive modelling. By incorporating target feature information into the encoding process, target encoding can improve model performance and provide more meaningful numerical representations of categorical features.

The steps involved in categorical feature target encoding are as follows:

- Group the data by the categorical feature.
- For each category in the categorical feature, calculate a mean of the target feature within that category.
- Assign the calculated statistical measure as the numerical label for each feature in the categorical feature.
- Replace the original categorical values with their corresponding numerical labels in the data.

The traditional label encoding technique assigns numerical labels to categories in a categorical feature based on their order or frequency of occurrence. However, this approach has some potential disadvantages:

- **Arbitrary numerical assignments:** Label encoding may introduce arbitrary numerical assignments to categories, which can lead to misinterpretation of relationships between categories. For example, assigning higher numerical values to categories with higher frequency may imply higher importance, which may not always be true.
- **Lack of capturing target feature information:** Label encoding does not consider the relationship between the categorical feature and the target feature, which may result in loss of information. The target feature contains valuable information useful for predictive modelling, and not utilizing this information can lead to suboptimal results.

On the other hand, target encoding has several advantages:

- **Incorporation of target variable information:** Target encoding utilizes the target feature information to encode categorical features, capturing the relationship between the categorical feature and the target feature. It can improve model performance as the encoded labels carry information about the target feature.
- **Handling of categorical features with high cardinality:** Target encoding can handle categorical features with high cardinality (i.e., many unique categories) better than label encoding. In label encoding, high cardinality categorical features may result in many numerical labels, leading to noisy or sparse representations. Target encoding can provide a more stable encoding even with high cardinality

features by using the mean of the target feature for each category.

- **Reduction of arbitrary numerical assignments:** Target encoding avoids arbitrary numerical assignments by encoding categories based on their relationship with the target feature. It can result in more meaningful and interpretable numerical labels, improving model interpretability.

Target encoding has the advantage of utilizing target variable information, handling high cardinality features better, and reducing arbitrary numerical assignments compared to traditional label encoding. Algorithm 3 discusses the categorical feature target encoding.

Algorithm 3: Categorical Feature Target Encoding

Input : Data with categorical features

List of categorical features to be converted

Output : Data with target-encoded numerical labels for categorical features

Step 1 : Load the data with categorical features.

Step 2 : Identify the categorical features to be converted to numerical labels.

Step 3 : For each categorical feature:

a. Compute the mean of the target variable for each category in the categorical feature.

b. Create a mapping of each category to its corresponding mean target value.

c. Update the data by replacing the original categorical values with the corresponding target-encoded numerical labels using the mapping created in Step 3b.

3.3 Filter-based feature selection using Information Gain and Ranker search:

In the HLE-FS algorithm, Filter-based feature selection using Information Gain and Ranker search is used to identify and select the most relevant features from a dataset based on their information gain and ranker search techniques.

Information gain is a measure used to quantify the information a feature provides about the target variable in a dataset. It is commonly used in decision tree algorithms to select the most informative features for splitting the data. Features with higher information gain are considered more important or relevant to the prediction task.

Ranker search is a technique used to rank features based on their importance or relevance to the target variable. Ranker search algorithms typically assign scores or rank to features based on certain criteria such as statistical measures, feature importance measures, or other domain-specific metrics.

Filter-based feature selection using Information Gain and Ranker search typically involves the following steps:

- **Compute Information Gain:** Information gain is calculated for each feature in the dataset using the entropy measure mentioned in Eq. (1). This quantifies each feature’s information about the target variable.

Information_Gain(feature) = E(tv) - WAE(feature) (1)

Where E(tv) is the entropy of the target variable, which measures the impurity or randomness of the target variable’s distribution in the dataset.

WAE(feature) is the weighted average entropy of the target variable after splitting the data based on the values of the feature. It is calculated by summing the target variable’s entropies for each feature value, weighted by the proportion of samples with that value.

- **Rank Features:** The features are then ranked based on their information gain scores, with higher scores indicating more informative features.
- **Apply Ranker Search:** A ranker search algorithm is applied to further rank the features based on additional criteria such as feature importance measures. This step helps to refine the feature ranking and identify the most relevant features for the prediction task.

Higher information gain values indicate more informative features, and features with higher information gain are generally considered more important or relevant for the prediction task.

3.4 Wrapper-based feature selection using Naïve Bayes, GreedyStepwise and ThreadPoolExecutor:

Wrapper-based feature selection is a type of feature selection method that evaluates the performance of a machine learning model using a subset of features and selects the best subset of features based on their performance. One commonly used technique for wrapper-based feature selection is using a specific classifier, such as Naïve Bayes, along with a search algorithm like GreedyStepwise for selecting subsets of features. In addition, a thread pool executor is a concurrent executor that can parallelize the computation and speed up the feature selection process.

Here’s an overview of the steps involved in this approach:

- **Load and preprocess the data:** Load the dataset and perform any necessary preprocessing steps, such as data normalization, handling missing values, and encoding categorical variables.
- **Choose a classifier:** Select a machine learning classifier to evaluate the subsets of features. In this case, Naïve Bayes is chosen as the classifier.
- **Define the evaluation measure:** Specify an evaluation measure or performance metric that will be used to evaluate the performance of the classifier on each subset of features. For example, accuracy, precision, recall, F1-score, etc.
- **Implement the search algorithm:** Choose a search algorithm, such as GreedyStepwise, that will be used to search for the best subset of features. This algorithm starts with an empty feature subset and iteratively adds or removes features based on their impact on the evaluation measure.
- **Implement parallel computation using ThreadPoolExecutor:** Use ThreadPoolExecutor to parallelize the evaluation of different subsets of features, which can speed up the feature selection process by evaluating multiple subsets concurrently.
- **Evaluate feature subsets:** For each subset of features, train the classifier on the subset of features using cross-validation or a holdout validation set, and evaluate its performance using the chosen evaluation measure.
- **Update the feature subset:** Update the feature subset based on the classifier's performance on the current subset of features, according to the search algorithm. For example, if adding a feature improves the performance, it is included in the subset; otherwise, it is removed.
- **Repeat the process:** Repeat steps 5-7 until a stopping criterion is met, such as reaching a desired subset size or no further improvement in the evaluation measure.
- **Output the selected feature subset:** Once the search algorithm converges, the final selected subset of features can be output as the reduced feature set for training a machine learning model.

Algorithm 4 discussed the proposed Wrapper-based feature selection.

Algorithm 4: Wrapper-based feature selection using Naïve Bayes, GreedyStepwise and ThreadPoolExecutor

Input	**:**	Stress data with n instances and m features
		Target variable indicating the level of stress (e.g. low, medium, high)
Output	**:**	A reduced feature set
Step 1	**:**	Load and preprocess the stress data
Step 2	**:**	Choose Naïve Bayes classifier
Step 3	**:**	Define the evaluation measure (e.g., accuracy)
Step 4	**:**	Implement the GreedyStepwise search algorithm.
Step 5	**:**	Implement parallel computation using ThreadPoolExecutor.
Step 6	**:**	Evaluate feature subsets:
Step 7	**:**	a. For each subset of features:
		i. Train the classifier on the subset of features
		ii. Evaluate its performance using a cross-validation set
Step 8	**:**	Update the feature subset based on the performance.
Step 9	**:**	Repeat the process until the stopping criterion is met.
Step 10	**:**	Output the selected feature subset

3.4.1 Naïve Bayes:

Naïve Bayes is a simple and widely used probabilistic classification algorithm based on Bayes' theorem. It is particularly well-suited for text classification and spam filtering tasks, but it can also be used for other classification tasks where the feature independence assumption holds.

Here's a detailed explanation of the Naïve Bayes algorithm:

1. **Probability and Bayes' Theorem:** Naïve Bayes algorithm is based on probability theory, specifically Bayes' theorem. Bayes' theorem is a mathematical formula that calculates an event's conditional probability, given that another event has already occurred. The formula is as follows:

$$\mathbf{P(A|B) = P(A) * P(B|A) / P(B)} \quad \mathbf{(2)}$$

Where: P(A|B) is the conditional probability of event A was given event B has occurred. P(A) is the prior probability of event A.P(B|A) is the probability of event B occurring, given that event, A has occurred. P(B) is the probability of event B occurring.

2. **Assumption of Feature Independence:** Naïve Bayes algorithm assumes that the features used for classification are independent, meaning that the presence or absence of one feature does not affect the

presence or absence of another feature. It is a strong assumption and may not always hold in real-world data, but it simplifies the calculation of probabilities and makes the algorithm computationally efficient.

3. **Training Phase:**

   a. **Input Data:** Naïve Bayes algorithm takes labelled training data as input, where each data point consists of a set of features (attributes) and a corresponding class label.

   b. **Feature and Class Probabilities:** The algorithm calculates the probabilities of each feature occurring in each class from the training data. For categorical features, it calculates the probability of each category occurring in each class. For continuous features, it models the distribution of feature values in each class using probability density functions.

   c. **Class Prior Probabilities:** The algorithm also calculates the prior probabilities of each class, which are the probabilities of each class occurring in the training data.

4. **Prediction Phase:**

   a. **Input Data:** Naïve Bayes algorithm takes a new, unlabeled data point as input, consisting of a set of features.

   b. **feature Probabilities:** The algorithm calculates the probabilities of the features occurring in the new data point using the probabilities calculated during the training phase.

   c. **Posterior Probabilities:** The algorithm calculates the conditional probabilities of each class given the observed feature values in the new data point using Bayes' theorem.

   d. **Prediction:** The algorithm assigns the class label with the highest posterior probability as the predicted class label for the new data point.

5. **Evaluation and Model Updating:** Naïve Bayes algorithm can be evaluated using performance metrics such as accuracy, precision, recall, F1-score, etc. The model can be updated with new training data if needed by re-calculating the feature and class probabilities.

Advantages of the Naïve Bayes algorithm*:*

- Simple and easy to implement.
- Computationally efficient, especially for large datasets.
- Performs well on text classification and spam filtering tasks.
- Handles categorical and continuous features.

3.4.2 GreedyStepwise:

GreedyStepwise search is a feature selection algorithm used in wrapper-based feature selection methods. A stepwise search algorithm combines a greedy approach with a stepwise selection strategy to select a subset of features from a larger set of features. The algorithm iteratively selects or removes features based on their impact on the performance of a chosen machine-learning model, typically using a validation set or cross-validation.

Here's a detailed explanation of the GreedyStepwise search algorithm:

1. **Input Data:** GreedyStepwise search takes a dataset with features and corresponding class labels as input and a machine learning model for performance evaluation.
2. **Initialization:** The algorithm starts with an empty set of selected features and an initial set of candidate features, typically including all available features.
3. **Feature Evaluation:** The algorithm evaluates the machine learning model's performance using the selected set of features, usually through a validation set or cross-validation. This evaluation is used as the initial performance score.
4. **Greedy Approach:**

   a. Forward Selection: The algorithm iteratively adds one candidate feature at a time to the selected set of features and evaluates the machine learning model's performance with the expanded set of features. The feature that results in the highest performance improvement, as measured by a predefined performance metric, is selected and added to the selected set of features.

   b. Backward Elimination: The algorithm iteratively removes one feature at a time from the selected set of features and evaluates the machine learning model's performance with the reduced set of features. The feature that results in the highest performance improvement, as measured by the predefined performance metric, is removed from the selected set of features.

5. **Stepwise Selection:**

   a. Stopping Criteria: The algorithm continues the forward selection and backward elimination steps until a stopping criterion is met. This criterion can be a predefined number of iterations, a threshold for performance improvement, or any other condition the user specifies.

   b. Best Subset Selection: The algorithm keeps track of the best subset of features that resulted in the highest performance score so far at each iteration. This best subset is updated whenever a feature is added or removed from the selected set of features.

6. **Final Subset of Features:** Once the stopping criterion is met, the algorithm returns the final subset of selected features that resulted in the highest performance score during the iterations.
7. **Model Evaluation and Updating:** The selected subset of features can be used to train a machine learning model, and its performance can be evaluated using performance metrics such as accuracy, precision, recall, F1-score, etc. The model can be updated with new training data, and the feature selection process can be repeated if needed.

Advantages of GreedyStepwise search*:*

- Combines the advantages of the greedy approach and stepwise selection strategy.
- Can handle large feature spaces by iteratively selecting or removing features based on their impact on performance.
- It can be used with any machine learning model for performance evaluation.
- Allows for fine-grained control over the feature selection process with customizable stopping criteria.

3.4.3 ThreadPoolExecutor:

ThreadPoolExecutor is a concurrent programming technique to perform feature subset evaluation in parallel. It allows for faster and more efficient computation of the performance of different feature subsets using cross-validation.

Here's a detailed explanation of how parallel computation using ThreadPoolExecutor can be implemented in Algorithm 4:

1. **ThreadPoolExecutor:** ThreadPoolExecutor is a class in Java that provides an easy way to create a pool of worker threads that can perform tasks concurrently. It can be used to parallelize this algorithm's evaluation of feature subsets.
2. **Feature subset evaluation:** In Algorithm 4, for each subset of features, the classifier is trained on that subset and its performance is evaluated using cross-validation. This step can be computationally expensive, especially if the dataset or the number of features is large.
3. **Parallel computation:** To speed up the feature subset evaluation process, ThreadPoolExecutor can create a pool of worker threads that can evaluate different feature subsets concurrently. Each worker thread can train the classifier and evaluate its performance for a specific subset of features.
4. **Efficiency and speed:** Using ThreadPoolExecutor, multiple feature subsets can be evaluated in parallel, significantly reducing the computation time compared to sequential evaluation. It can lead to a more efficient and faster feature selection process, allowing quicker identification of the optimal feature subset.
5. **Implementation:** The algorithm can create an instance of ThreadPoolExecutor with a specified number of threads, depending on the available hardware resources, to implement parallel computation using ThreadPoolExecutor. Then, the feature subsets can be distributed among the worker threads, and each thread can independently train the classifier and evaluate its performance. The results can be collected and updated in the algorithm accordingly.

3.5 Embedding-based feature selection using PCA and Ranker Search: Embedding-based feature selection using PCA (Principal Component Analysis) and a Ranker Search algorithm is a method used to reduce the dimensionality of a dataset by transforming the original features into a lower-dimensional space using PCA and then ranking the transformed features based on their importance using a Ranker Search algorithm. Here's a detailed explanation of each step:

1. **Principal Component Analysis (PCA):** PCA is a statistical technique commonly used for dimensionality reduction in machine learning. It works by finding the principal components of the data, which are linear combinations of the original features that capture the most important patterns or variations in the data. These principal components are orthogonal and are sorted by importance, with the first principal component capturing the most variance in the data.
2. **Embedding-based feature selection:** In this method, PCA is used to embed the original features of the dataset into a lower-dimensional space. It is done by calculating the principal components of the dataset and then using them as the new features for the subsequent feature selection step.
3. **Ranker Search algorithm:** Once the features are transformed into a lower-dimensional space using PCA, a Ranker Search algorithm is used to rank the importance of the transformed features. The Ranker Search algorithm evaluates the importance of each feature based on the feature importance score. The features are ranked in descending order based on their importance, with the most important feature ranked first.
4. **Feature selection:** The transformed features are then selected based on their rankings. The top-ranked features considered the most important in the lower-dimensional space, are selected as the reduced feature set for further analysis or modelling. The number of top-ranked features selected can be determined based on a predefined threshold.
5. **Benefits of embedding-based feature selection using PCA and Ranker Search algorithm:** This method has several benefits. First, PCA can effectively reduce the dimensionality of the dataset by transforming the original features into a lower-dimensional space while

retaining the most important patterns or variations in the data. It can help to overcome the curse of dimensionality and improve the performance of subsequent modelling algorithms. Second, the Ranker Search algorithm provides a systematic and data-driven way to rank the importance of the transformed features, allowing for efficient feature selection. Finally, using a combination of embedding-based feature selection using PCA and Ranker Search algorithm, it is possible to identify the most important features in the lower-dimensional space, leading to a more interpretable and efficient feature subset for further analysis or modelling.

6. **Implementation:** To implement embedding-based feature selection using PCA and Ranker Search algorithm, the following steps can be followed:
   a. Perform PCA on the original feature matrix to obtain the principal components.
   b. Rank the principal components based on their importance using a Ranker Search algorithm, such as a feature importance score.
   c. Select the top-ranked principal components as the reduced feature set for further analysis or modelling.
   d. Optionally, the number of top-ranked principal components can be determined based on a predefined threshold.

3.6 Majority voting-based feature selection:

The Majority Voting based Feature Selection algorithm is a method that combines the results of three different feature selection methods: Filter-based, Wrapper-based, and Embedding-based, to select relevant features from a dataset. The algorithm takes the results of these three methods as input, which are represented as lists of selected features (FS_result_1, FS_result_2, FS_result_3).

First, the algorithm concatenates these three lists of features to create a single list called all_features. Then, it creates an empty dictionary called vote_count to track each feature's vote count.

Next, the algorithm iterates through each feature in the list of all_features. For each feature, it checks if it is already present in the vote_count dictionary. If not, it adds the feature to the dictionary with an initial vote count of 1. If the feature is already in the dictionary, it increments its vote count by 1.

After counting the votes for all features, the algorithm creates an empty list called FS_result, which will store the final list of selected features. It then iterates through each feature and its vote count in the vote_count dictionary. If the vote count of a feature is greater than or equal to 2, the feature is appended to the FS_result list.

Finally, after iterating through all the features and their vote counts, the algorithm returns the FS_result list as the final list of selected features. The majority voting algorithm is discussed in Algorithm 5.

Algorithm 5: Majority voting-based feature selection

Input	:	Filter-based feature selection results (FS_result_1), Wrapper-based feature selection results (FS_result_2), Embedding-based feature selection results (FS_result_3)
Output	:	FS_result
Step 1	:	all_features = concatenate(FS_result_1, FS_result_2, FS_result_3)
Step 2	:	vote_count = create_empty_dictionary()
Step 3	:	FOR EACH feature IN all_features:
Step 4	:	IF feature NOT IN vote_count:
Step 5	:	vote_count[feature] = 1
Step 6	:	ELSE:
Step 7	:	vote_count[feature] += 1
Step 8	:	FS_result = create_empty_list()
Step 9	:	FOR EACH feature, count IN vote_count:
Step 10	:	IF count >= 2:
Step 11	:	append feature TO FS_result
Step 12	:	RETURN FS_result

In essence, the Majority Voting based Feature Selection algorithm aims to improve the robustness and stability of feature selection by considering the consensus of multiple feature selection methods. Features that receive votes from at least two of the three methods are selected as the final set of relevant features, potentially leading to improved model performance and generalization.

4. Experimental Results and Discussions:

This section presents the experimental results and discussions of a high-level ensemble feature selection (HLE-FS) algorithm for stress data. The algorithm is implemented in Java and utilizes two datasets, namely the Swell-EDA and WESAD-EDA datasets. The Swell-EDA dataset consists of 9849 rows and 57 features, while the WESAD-EDA dataset contains 3395 rows and 49 features. Using the Random Forest classifier, the algorithm's performance is evaluated by comparing the dataset's accuracy, precision, recall, and F1-score with and without feature selection. Finally, this section provides a detailed analysis of the experimental results and discusses the findings, highlighting the effectiveness of the HLE-FS algorithm for stress data feature selection.

In the context of a classifier, accuracy, precision, recall, and F1-score are commonly used performance metrics to evaluate the performance of a classifier in machine learning.

Accuracy measures how well a classifier correctly predicts the overall number of instances. It is the ratio of correctly predicted instances to the total number of instances in the dataset. The formula for accuracy is:

Accuracy = (Number of Correct Predictions) / (Total Number of Predictions) (3)

Precision measures how well a classifier correctly predicts the positive instances out of the instances it predicted as positive. It is the ratio of the number of true positive predictions to the sum of true positive and false positive predictions. The formula for precision is:

Precision = (Number of True Positives) / (Number of True Positives + Number of False Positives) (4)

Recall, also known as sensitivity or true positive rate, measures how well a classifier identifies all the positive instances in the dataset. It is the ratio of the number of true positive predictions to the sum of true positive and false negative predictions. The formula for the recall is:

Recall = (Number of True Positives) / (Number of True Positives + Number of False Negatives) (5)

F1-score is a measure of the trade-off between precision and recall. It is the harmonic mean of precision and recall and provides a balanced measure of the classifier's performance. The formula for F1-score is:

F1-score = 2 * (Precision * Recall) / (Precision + Recall) (6)

These performance metrics are commonly used in evaluating the effectiveness of a classifier in terms of its accuracy, precision, recall, and F1-score and can provide valuable insights into the performance of the HLE-FS algorithm for stress data feature selection.

Table 1 compares the Swell-EDA dataset's performance before and after HLE-FS feature selection using the Random Forest classifier.

Table 1: Performance comparison of the Swell-EDA dataset before and after HLE-FS feature selection using Random Forest classifier

Metrics	Swell-EDA dataset before HLE-FS	Swell-EDA dataset after HLE-FS
Accuracy	51.00	88.00
Precision	51.86	87.80
Recall	51.00	88.00
F1-score	51.24	87.58

Figure 1 visually represents the performance comparison of the Swell-EDA dataset before and after HLE-FS feature selection using the Random Forest classifier.

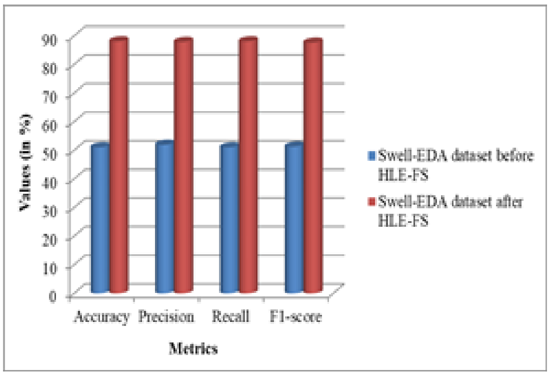

Fig 1: Performance comparison of the Swell-EDA dataset before and after HLE-FS feature selection using Random Forest classifier

From Figure 1, we can see that applying the HLE-FS technique to the Swell-EDA dataset has significantly improved the performance of the machine learning model, as evidenced by the higher values of accuracy, precision, recall, and F1-score after applying the HLE-FS technique compared to the original dataset without applying the technique. It suggests that the HLE-FS technique has helped improve the predictive accuracy and performance of the machine learning model on the Swell-EDA dataset.

Furthermore, Table 2 compares the WESAD-EDA dataset's performance before and after HLE-FS feature selection using the Random Forest classifier.

Table 2: Performance comparison of the WESAD-EDA dataset before and after HLE-FS feature selection using Random Forest classifier

Metrics	WESAD-EDA dataset before HLE-FS	WESAD-EDA dataset after HLE-FS
Accuracy	69.00	91.00

Precision	69.04	92.05
Recall	69.00	91.00
F1-score	68.88	91.01

Figure 2 visually represents the performance comparison of the WESAD-EDA dataset before and after HLE-FS feature selection using the Random Forest classifier.

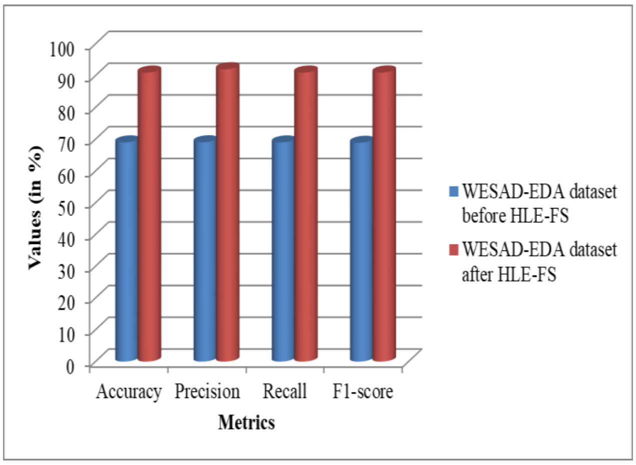

Fig 2: Performance comparison of the WESAD-EDA dataset before and after HLE-FS feature selection using Random Forest classifier

From Figure 2, we can see that applying the HLE-FS technique to the WESAD-EDA dataset has significantly improved the performance of the machine learning model, as evidenced by the higher values of accuracy, precision, recall, and F1-score after applying the HLE-FS technique compared to the original dataset without applying the technique. It suggests that the HLE-FS technique has helped improve the machine learning model's predictive accuracy and performance on the WESAD-EDA dataset.

5. Conclusion:

In conclusion, this paper proposed a high-level ensemble feature selection (HLE-FS) algorithm for stress data to mitigate the challenges of large dimensionality and improve machine learning model performance in stress classification. The algorithm incorporated three feature selection techniques - filter-based, wrapper-based, and embedding-based methods - in an ensemble approach, utilizing information gain, Naïve Bayes classifier, GreedyStepwise search, Principal Component Analysis (PCA), and Ranker search. The results of these techniques were combined using a majority voting mechanism to select the most informative features. The proposed algorithm demonstrated superior accuracy and computational efficiency performance compared to the existing system, effectively identifying relevant features and improving stress classification performance. Future work can be carried out to enhance the proposed algorithm further. One potential direction is to explore further other feature selection techniques and algorithms, such as genetic algorithms or recursive feature elimination, to improve the selection of relevant features for stress data. Further investigation can also be conducted to evaluate the performance of the proposed algorithm on different types of stress data, as well as in real-world stress prediction scenarios.

Author contributions

Mr. Prashant M. Suryavanshi: Conceptualization, Methodology, Software, Field study, Data curation, Writing-Original draft preparation, Software, Validation., Field study. **Dr. Pradnya A Vikhar:** Visualization, Investigation, Writing-Reviewing and Editing.

Conflicts of interest

The authors declare no conflicts of interest.

References

[1] Daniel, C. O. (2019). Effects of job stress on employee's performance. International Journal of Business, Management and Social Research, 6(2), 375-382.

[2] Khaire, U. M., & Dhanalakshmi, R. (2022). Stability of feature selection algorithm: A review. Journal of King Saud University-Computer and Information Sciences, 34(4), 1060-1073.

[3] Asif, A., Majid, M., & Anwar, S. M. (2019). Human stress classification using EEG signals in response to music tracks. Computers in biology and medicine, 107, 182-196.

[4] Hwangbo, H., Sharma, V., Arndt, C., & TerMaath, S. (2023). A Randomized Subspace-based Approach for Dimensionality Reduction and Important Variable Selection. Journal of Machine Learning Research, 24, 1-30.

[5] Yang, P., Huang, H., & Liu, C. (2021). Feature selection revisited in the single-cell era. Genome Biology, 22, 1-17.

[6] Alghowinem, S. M., Gedeon, T., Goecke, R., Cohn, J., & Parker, G. (2020). Interpretation of depression detection models via feature selection methods. IEEE Transactions on affective computing.

[7] Lin, S. Stress Recognition Using LSTM-Based Neural Network Model with Feature Selection and Bimodal Distribution Removal.

[8] Majid, M., Arsalan, A., & Anwar, S. M. (2022). A Multimodal Perceived Stress Classification Framework using Wearable Physiological Sensors. arXiv preprint arXiv:2206.10846.

[9] Parsi, A., O'Callaghan, D., & Lemley, J. (2023). A Feature Selection Method for Driver Stress Detection Using Heart Rate Variability and Breathing Rate. arXiv preprint arXiv:2302.01602.

[10] Reddy, U. S., Thota, A. V., & Dharun, A. (2018, December). Machine learning techniques for stress prediction in working employees. In 2018 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC) (pp. 1-4). IEEE.

[11] Jaiswal, S., Song, S., & Valstar, M. (2019, September). Automatic prediction of depression and anxiety from behaviour and personality attributes. In 2019 8th international conference on affective computing and intelligent interaction (acii) (pp. 1-7). IEEE.

[12] Rashid, B., & Calhoun, V. (2020). Towards a brain-based predictome of mental illness. Human brain mapping, 41(12), 3468-3535.

[13] Mousavian, M., Chen, J., & Greening, S. (2018). Feature selection and imbalanced data handling for depression detection. In Brain Informatics: International Conference, BI 2018, Arlington, TX, USA, December 7–9, 2018, Proceedings 11 (pp. 349-358). Springer International Publishing.

[14] Tadesse, M. M., Lin, H., Xu, B., & Yang, L. (2019). Detection of depression-related posts in Reddit social media forum. IEEE Access, 7, 44883-44893.

[15] Saeed, S. M. U., Anwar, S. M., Khalid, H., Majid, M., & Bagci, U. (2020). EEG-based classification of long-term stress using psychological labelling. Sensors, 20(7), 1886.