Journal of Harbin Engineering University
ISSN: 1006-7043

Vol 44 No. 8
August 2023

OEFA: An Optimized Ensemble Fusion Algorithm for Stress Classification and Prediction

Suryavanshi Prashant Maharudra and Dr. Pradnya Ashish Vikhar
Department of Computer Science and Engineering
Dr. A. P. J. Abdul Kalam University, Indore (M. P.) – 452010

Abstract:-Stress is a prevalent issue in modern society, leading to various adverse effects on individuals' mental and physical well-being. Accurate classification and prediction of stress levels are crucial for effective intervention and support. However, existing stress classification and prediction techniques suffer from several disadvantages, including imbalanced datasets, missing values, categorical data, lack of feature selection, and limited model diversity. To address these challenges, this paper proposes an Optimized Ensemble Fusion Algorithm (OEFA) for stress classification and prediction. The OEFA algorithm combines multiple techniques to overcome the limitations of existing approaches. It employs the Cluster-Based Adaptive Synthetic Sampling (CBADAS) algorithm to balance imbalanced datasets, generating synthetic instances for the minority class. Missing values are handled by removing instances with missing data, ensuring only complete data is used for training and testing. Categorical data is transformed into numerical format using label encoding, enabling the use of traditional machine learning algorithms. Attribute selection is performed using the ReliefFAttributeEval algorithm with Ranker Search, reducing dimensionality and improving computational efficiency. Furthermore, OEFA leverages ensemble learning with the AdaBoostM1 algorithm, incorporating optimized versions of Logistic Regression, HoeffdingTree, LMT, REPTree, JRip, OneR, PART, and MultilayerPerceptron as base classifiers. Experimental results demonstrate the superiority of the OEFA algorithm, exhibiting the highest accuracy, precision, recall, and F1-score compared to existing techniques. The advantages of OEFA include improved accuracy through addressing imbalanced datasets, missing values, and categorical data. The algorithm enhances generalization capability, mitigates overfitting, and demonstrates robustness by combining multiple base classifiers. Efficient feature selection is achieved using ReliefFAttributeEval with Ranker Search, contributing to reduced dimensionality and computational requirements.

Keywords: Clustering, Classification, Prediction, Oversampling, Label encoding

1. Introduction

Stress is a prevalent issue affecting individuals across various domains, including healthcare, psychology, and human resources [1]. The ability to accurately classify and predict stress levels is crucial for early detection, intervention, and personalized treatment [2]. In recent years, machine learning techniques have shown promise in this area, enabling the development of automated systems to assist in stress assessment [3]. However, existing stress classification and prediction algorithms have certain limitations that hinder their effectiveness.

Existing stress classification and prediction techniques often face challenges such as imbalanced datasets, missing values, categorical data, lack of feature selection, and limited model diversity [4], [5]. Imbalanced datasets, where one class significantly outnumbers the other, can lead to biased models that perform poorly on the minority class. Additionally, stress datasets frequently contain missing values, creating challenges during data preprocessing and model training. Traditional machine learning algorithms require numerical inputs, necessitating the transformation of categorical data. Some algorithms lack a feature selection step, resulting in suboptimal performance and increased computational requirements. Furthermore, single models may not capture the full complexity of stress data, leading to lower predictive accuracy.

To address these challenges, this paper proposes an Optimized Ensemble Fusion Algorithm (OEFA) for stress classification and prediction. The primary objective of OEFA is to improve upon existing algorithms by leveraging ensemble learning techniques and addressing their inherent disadvantages. OEFA integrates multiple strategies

to overcome the limitations of imbalanced datasets, missing values, categorical data, lack of feature selection, and limited model diversity.

The key contributions of this paper are as follows:

- **Balanced dataset conversion:** OEFA employs the Cluster-Based Adaptive Synthetic Sampling (CBADAS) algorithm to generate synthetic instances and balance imbalanced datasets, ensuring improved representation of all classes.
- **Missing value handling:** OEFA includes a step to remove instances with missing values, allowing for the use of complete data during training and testing.
- **Categorical data transformation:** OEFA performs label encoding to convert categorical values into numerical representations, facilitating the utilization of traditional machine learning algorithms.
- **Attribute selection:** OEFA applies the ReliefFAttributeEval algorithm with Ranker Search to select the most informative features, reducing dimensionality and enhancing model efficiency.
- **Ensemble learning:** OEFA utilizes ensemble learning through the AdaBoostM1 algorithm, combining multiple optimized base classifiers to create a stronger and more robust stress classification model.
- **Experimental evaluation:** Extensive experiments are conducted to evaluate the performance of the proposed OEFA algorithm, comparing it with state-of-the-art methods on various stress classification and prediction datasets.

The aim of this study is to address the limitations of existing stress classification and prediction techniques and propose an optimized algorithm to improve classification accuracy, robustness, and generalization capability. The OEFA algorithm aims to enhance stress classification models' accuracy by addressing the challenges of imbalanced datasets, missing values, and categorical data. Additionally, the use of ensemble learning and attribute selection techniques reduces overfitting, improves generalization to unseen data, and enhances computational efficiency.

The remainder of this paper is organized as follows. Section 2 provides a comprehensive review of related works in stress classification and prediction. Section 3 presents the proposed Optimized Ensemble Fusion Algorithm (OEFA) in detail. Section 4 describes the experimental setup, including datasets, evaluation metrics, experimental results and performance comparison with existing methods. Finally, Section 5 concludes the paper and outlines potential future research directions in the field of stress classification and prediction.

2. Related Works

This section provides a comprehensive review of existing works in the field of stress classification and prediction. Stress assessment and prediction have garnered significant attention in recent years due to their potential impact on individuals' well-being and the effectiveness of interventions. Various machine learning and data mining techniques have been employed to develop models for stress classification and prediction.

Majid et al. [6] proposed a multimodal perceived stress classification framework using wearable physiological sensors. The approach leverages physiological data from wearable sensors to classify stress levels. The advantage of this approach lies in its ability to capture real-time physiological responses. However, a limitation of this method is the reliance on specialized sensors, which may not be easily accessible or feasible for widespread implementation.

Kumar et al. [7] presented an assessment of anxiety, depression, and stress using machine learning models. The study utilizes machine learning algorithms to classify stress levels based on self-reported data. The advantage of this approach is its simplicity and ease of data collection. However, self-report data may be subject to biases and inaccuracies, leading to potential limitations in accuracy and reliability.

Shafiee et al. [8] focused on predicting mental health problems among higher education students using machine learning. The approach utilizes various machine learning techniques to predict mental health problems based on academic and personal information. The advantage of this approach is its potential for early identification and intervention. However, the reliance on self-reported data and the limited scope of features may affect the generalizability of the model.

Chiong et al. [9] proposed a textual-based approach for depression detection using machine learning

classifiers and social media texts. The study leverages natural language processing techniques to analyze social media texts and classify individuals based on their depressive tendencies. The advantage of this approach is its potential for large-scale data analysis. However, the challenge lies in effectively extracting meaningful features from unstructured text data, and the reliance on social media data may introduce biases.

Shin et al. [10] presented a machine learning-based predictive modeling of postpartum depression. The approach utilizes machine learning algorithms to predict the risk of postpartum depression based on various factors. The advantage of this approach is its potential for early identification and targeted intervention for at-risk individuals. However, the limitation lies in the need for comprehensive and accurate data collection related to postpartum factors.

Choudhury et al. [11] focused on predicting depression in Bangladeshi undergraduates using machine learning. The study employed machine learning techniques to predict depression levels based on demographic and behavioral factors. The advantage lies in its potential for early identification and intervention in the university setting. However, the reliance on self-reported data and the limited demographic scope may affect the generalizability of the model.

Andersson et al. [12] proposed predictive models for women with depressive symptoms postpartum using machine learning methods. The study utilizes machine learning algorithms to predict depressive symptoms in women after childbirth based on various factors. The advantage lies in its potential for personalized intervention and support. However, the limitation lies in the need for accurate and comprehensive data collection related to postpartum factors.

Baek et al. [13] proposed a context deep neural network model for predicting depression risk using multiple regression. The study leverages deep learning techniques to predict depression risk based on contextual information. The advantage of this approach is its ability to capture complex contextual relationships. However, the limitation lies in the interpretability and potential bias of deep learning models.

Zakaria et al. [14] presented StressMon, a scalable detection system for perceived stress and depression using passive sensing of changes in work routines and group interactions. The approach utilizes passive sensing techniques to detect stress and depression based on changes in work routines and group interactions. The advantage lies in its unobtrusive nature and real-time monitoring. However, the limitation lies in the need for access to sensitive data and potential privacy concerns.

Mumu et al. [15] proposed a depressed people detection approach from Bangla social media status using LSTM and CNN. The study utilizes deep learning techniques to detect individuals with depressive tendencies based on their social media status. The advantage of this approach is its potential for analyzing large amounts of textual data. However, the limitation lies in the language specificity and potential bias of social media data.

While these approaches offer valuable insights into stress classification and prediction, they also have certain limitations. Challenges include imbalanced datasets, missing values, categorical data, lack of feature selection, and limited model diversity. Imbalanced datasets can lead to biased models with poor performance on minority classes, while missing values pose challenges during data preprocessing and model training. The transformation of categorical data into numerical inputs may be required for certain algorithms, and the lack of feature selection can result in suboptimal performance and increased computational requirements. Additionally, single models may struggle to capture the full complexity of stress data, leading to lower predictive accuracy.

To address these challenges, our proposed Optimized Ensemble Fusion Algorithm (OEFA) aims to overcome the limitations of existing approaches. OEFA leverages an ensemble of diverse models to capture a wider range of stress patterns and incorporates feature selection techniques to enhance model performance. By optimizing the fusion of multiple models, OEFA aims to improve the accuracy and robustness of stress classification and prediction. Furthermore, OEFA is designed to handle imbalanced datasets, missing values, and categorical data effectively, ensuring reliable and accurate stress assessment.

Through the integration of these advancements, OEFA aims to offer a promising solution to the challenges faced by existing stress classification and prediction techniques, ultimately contributing to improved mental health assessment and intervention.

3. An Optimized Ensemble Fusion Algorithm

This section designed an Optimized Ensemble Fusion Algorithm (OEFA) for stress classification and prediction tasks. It aims to improve upon the existing algorithms used for stress classification by addressing their disadvantages and leveraging ensemble learning techniques. The disadvantages commonly associated with existing stress classification and prediction algorithms:

- **Imbalanced datasets:** Many real-world stress datasets suffer from class imbalance, where the number of instances in one class is significantly higher than the other. This can lead to biased models that perform poorly on the minority class.
- **Missing values:** Stress datasets often contain missing values, which can create challenges during data preprocessing and model training.
- **Categorical data:** Traditional machine learning algorithms typically require numerical inputs, so categorical data needs to be transformed into a numerical format before training.
- **Lack of feature selection:** Some algorithms do not include a feature selection step, which can lead to suboptimal performance and increased computational requirements.
- **Limited model diversity:** Single models may not capture the full complexity of stress data, leading to lower predictive accuracy.

The reasons for developing the OEFA algorithm to address these disadvantages are:

- **Balanced dataset conversion:** OEFA employs the Cluster-Based Adaptive Synthetic Sampling (CBADAS) algorithm to address the issue of imbalanced datasets. CBADAS generates synthetic instances for the minority class to balance the data, improving the model's ability to capture patterns from both classes.
- **Missing value handling:** OEFA includes a step to remove instances with missing values, ensuring that only complete data is used for training and testing.
- **Categorical data transformation:** OEFA performs label encoding to convert categorical values into numerical representations, enabling the use of traditional machine learning algorithms.
- **Attribute selection:** OEFA applies the ReliefFAttributeEval algorithm with Ranker Search to select the most informative features, reducing dimensionality and enhancing model efficiency.
- **Ensemble learning:** OEFA employs ensemble learning using the AdaBoostM1 algorithm, combining multiple base classifiers to create a stronger and more robust model. The base classifiers used in OEFA include optimized version of Logistic Regression, HoeffdingTree, LMT, REPTree, JRip, OneR, PART, and MultilayerPerceptron.
- **Randomization and splitting:** OEFA randomizes the data to ensure the randomness of the split, and it divides the randomized data into training (70%) and testing (30%) datasets.

The key novelty of OEFA lies in its combination of techniques to address the specific challenges of stress classification and prediction. By integrating data balancing, missing value handling, categorical data transformation, feature selection, and ensemble learning, OEFA aims to improve classification accuracy, robustness, and generalization capability. Some of the advantages of the OEFA algorithm are:

- **Improved accuracy:** By addressing the challenges of imbalanced datasets, missing values, and categorical data, OEFA aims to enhance the accuracy of stress classification and prediction models.
- **Enhanced generalization:** The use of ensemble learning and attribute selection helps to reduce overfitting and improve the model's ability to generalize to unseen data.
- **Efficient feature selection:** The ReliefFAttributeEval algorithm with Ranker Search efficiently selects relevant features, reducing dimensionality and computational requirements.
- **Robustness:** By combining multiple base classifiers, OEFA increases the robustness of the model, making it less sensitive to variations in the data.
- **Applicability:** OEFA can be applied to various stress classification and prediction tasks, allowing for broader utilization across different domains and datasets.

Algorithm 1 discussed proposed OEFA algorithm.

Algorithm 1: Optimized Ensemble Fusion Algorithm (OEFA) for stress classification and prediction		
Input	**:**	Stress data (dataset with features and target variable)
Output	**:**	Predicted stress levels for the testing dataset
Step 1	**:**	Load the stress data.
Step 2	**:**	Remove instances with missing values.
Step 3	**:**	Convert categorical values to numerical using label encoding.
Step 4	**:**	Perform imbalanced to balanced dataset conversion using the Cluster-Based Adaptive Synthetic Sampling (CBADAS) algorithm.**// Algorithm 2**
Step 5	**:**	Normalize the data using Min-Max normalization.
Step 6	**:**	Perform attribute selection using the ReliefFAttributeEval algorithm with Ranker Search.
Step 7	**:**	Randomize the data to ensure randomness of the split.
Step 8	**:**	Split the randomized data into training (70%) and testing (30%) datasets.
Step 9	**:**	Create an AdaBoostM1 boosting classifier with 8 optimized classifiers as base classifiers (Logistic, HoeffdingTree, LMT, REPTree, JRip, OneR, PART, MultilayerPerceptron).
Step 10	**:**	Train the AdaBoostM1 classifier using the training dataset.
Step 11	**:**	Use the trained AdaBoostM1 classifier to predict stress levels in the testing dataset.

The Optimized Ensemble Fusion Algorithm (OEFA) for stress classification and prediction is outlined as follows. Firstly, the stress data, consisting of features and the target variable, is loaded (Step 1). Instances with missing values are then removed from the dataset (Step 2). Categorical values are converted into numerical format using label encoding (Step 3). To address imbalanced datasets, the Cluster-Based Adaptive Synthetic Sampling (CBADAS) algorithm is employed for imbalanced to balanced dataset conversion (Step 4). Next, the data is normalized using Min-Max normalization (Step 5). Attribute selection is performed using the ReliefFAttributeEval algorithm with Ranker Search to identify the most informative features (Step 6). To ensure randomness, the data is randomized (Step 7), followed by splitting the randomized data into training (70%) and testing (30%) datasets (Step 8). An AdaBoostM1 boosting classifier is created with eight optimized classifiers as base classifiers, including Logistic Regression, HoeffdingTree, LMT, REPTree, JRip, OneR, PART, and MultilayerPerceptron (Step 9). The AdaBoostM1 classifier is trained using the training dataset (Step 10). Finally, the trained AdaBoostM1 classifier is utilized to predict stress levels in the testing dataset (Step 11). This algorithm aims to optimize stress classification and prediction by addressing data preprocessing, feature selection, imbalanced dataset handling, and leveraging ensemble learning techniques.

3.1 Cluster-Based Adaptive Synthetic Sampling (CBADAS) algorithm:

The Cluster-Based Adaptive Synthetic Sampling (CBADAS) algorithm is an approach used for converting imbalanced datasets into balanced datasets. It aims to address the issue of class imbalance, where one class has significantly fewer instances compared to the other class (es). Imbalanced datasets can pose challenges in various machine learning tasks, such as classification, as models tend to be biased towards the majority class and perform poorly on the minority class.

The CBADAS algorithm is designed to overcome the limitations of existing imbalanced data conversion techniques by leveraging cluster-based sampling and adaptive synthetic instance generation. It combines the concepts of clustering and synthetic sampling to create additional instances for the minority class, thus balancing the dataset. Algorithm 2 discusses the working process of CBADAS algorithm.

Algorithm 2: Cluster-Based Adaptive Synthetic Sampling (CBADAS) algorithm for imbalanced to balanced dataset conversion		
Input	**:**	Preprocessed Stress dataset file
Output	**:**	Oversampled data
Step 1	**:**	Read the Preprocessed Stress dataset

file and store the data in a List of String arrays.

Step 2 : Remove outlier instances based on the interquartile range:

a. For each attribute column in the data:

i. Sort the attribute values in ascending order.

ii. Calculate the first quartile (Q1) and third quartile (Q3) values.

iii. Calculate the interquartile range (IQR) as Q3 - Q1.

iv. Define the lower threshold as Q1 - k * IQR and the upper threshold as Q3 + k * IQR, where k is a user-defined parameter (typically set to 1.5 or 3).

v. Iterate through the instances:

- If the value of the attribute in the current instance is outside the lower and upper thresholds, remove the instance from the data.

Step 3 : Count the number of instances belonging to the majority class.

Step 4 : Initialize an empty list to store unique labels of the minority class.

Step 5 : Initialize an empty list to store instances corresponding to the unique labels of the minority class.

Step 6 : Iterate through the data and for each instance:

a. If the instance's label is not the majority label, and the label is not already present in the list:

i. Add the label to the list of unique labels.

ii. ii. Add the instance to the list of instances corresponding to the unique labels.

Step 7 : Create a new list and copy all instances from the remaining data into it.

Step 8 : For each unique label in the list of unique labels, perform the following steps:

a. Retrieve the corresponding instance.

b. Count the number of instances belonging to the current minority class.

c. If the current label is not the majority label and the minority count is less than the majority count:

i. Calculate the difference between the majority count and the minority count.

ii. Repeat the following steps countDifference times:

- Create an empty list, cluster1, to store the results of k-nearest neighbors.
- Create an empty list, cluster2, to store the results of similar instances.
- Find the k-nearest neighbors for the current instance and add them to cluster1.
- Find the similar instances for the current instance and add them to cluster2.
- Combine cluster1 and cluster2 into a new list, combinedCluster.
- Generate a synthetic instance as follows:
- Initialize variables sum = 0 and numericCount = 0.
- For each neighbor in combinedCluster:
- Retrieve the value of the attribute at index i from the neighbor.
- If the value is numeric:
- Parse the value as a double and add it to sum.
- Increment numericCount.
- Calculate the average by dividing sum by numericCount.
- Calculate the difference between the average and the value of the current instance at index i.
- Generate a random gap value using random.nextDouble().
- Compute the synthetic value using the formula: syntheticValue = currentValue + gap * difference.
- Convert the syntheticValue to a string and assign it to

syntheticInstance[i].
- Add the synthetic instance to the new list.

Step 9 : Return the new list as the Oversampled data.

The CBADAS algorithm follows a series of steps to convert an imbalanced dataset into a balanced one. Initially, the preprocessed stress dataset is read and stored in a list of string arrays. The algorithm then proceeds to remove outlier instances based on the interquartile range. For each attribute column, lower and upper thresholds are defined, and instances falling outside these thresholds are eliminated. The number of instances belonging to the majority class is counted, while an empty list is initialized to store the unique labels of the minority class. Another empty list is created to hold instances corresponding to these unique labels.

Subsequently, the algorithm iterates through the data, examining each instance. If an instance's label is neither the majority label nor already present in the list of unique labels, it is added to the respective lists. All remaining instances from the data are copied into a new list. The algorithm then focuses on each unique label in the list and performs the following steps: the corresponding instance is retrieved, the count of instances belonging to the current minority class is determined, and if the current label is not the majority label and the minority count is lower than the majority count, the difference between the two counts is calculated.

Within this context, the subsequent steps are repeated countDifference times. For each repetition, two empty lists, cluster1 and cluster2, are created to store the results of k-nearest neighbors and similar instances, respectively. The k-nearest neighbors for the current instance are identified and added to cluster1, while similar instances are added to cluster2. Cluster1 and cluster2 are then combined into a new list called combinedCluster. To generate synthetic instances, the algorithm calculates the average difference between the attribute values of the current instance and its neighbors in combinedCluster. These differences are used to create synthetic instances, which are added to the new list. Finally, the algorithm returns the new list, which contains the oversampled data, as the output.

The CBADAS algorithm can be used in scenarios where imbalanced datasets are encountered, and the goal is to improve the performance of machine learning models by balancing the class distribution. It can be applied in various domains, such as fraud detection, medical diagnosis, and text classification, where imbalanced data is prevalent.

One of the main advantages of the CBADAS algorithm is outlier removal. By incorporating the step of removing outlier instances based on the interquartile range, the CBADAS algorithm helps in improving the quality of the dataset before oversampling. Removing outliers can be beneficial as they can disproportionately influence the training process and the performance of machine learning models, especially in imbalanced datasets.

By eliminating outliers, the CBADAS algorithm enhances the overall data quality and mitigates the potential negative impact of outliers on the model's learning process. This step contributes to reducing the influence of noisy or erroneous data points, leading to more reliable and robust oversampling results.

Therefore, the advantage of outlier removal in the CBADAS algorithm is that it promotes data integrity and prepares a cleaner dataset for subsequent oversampling, ensuring the synthetic instances generated are based on a more accurate representation of the underlying data distribution.

Other advantages of the CBADAS algorithm include:

- Ability to handle class imbalance effectively by generating synthetic instances for the minority class.
- Utilization of cluster-based sampling to capture the underlying structure of the data, resulting in more representative synthetic instances.
- Adaptive synthetic instance generation that takes into account the specific characteristics of each minority class, leading to improved performance.

Overall, the CBADAS algorithm is a valuable technique for converting imbalanced datasets into balanced datasets. It addresses the limitations of existing approaches by leveraging clustering and adaptive synthetic instance generation. The algorithm is versatile and can be applied in various domains to enhance the performance of machine learning models when faced with imbalanced data.

The CBADAS algorithm combines outlier removal, label identification, and adaptive synthetic instance generation to convert imbalanced datasets into balanced ones. By following these steps, the algorithm can effectively address the class imbalance issue, produce representative synthetic instances, and enhance the performance of machine learning models in scenarios where imbalanced data is encountered. Its advantages lie in its ability to effectively handle class imbalance, capture data structure through cluster-based sampling, and generate representative synthetic instances.

3.2 Optimized Logistic Regression:

Logistic Regression is a popular algorithm used for binary classification problems. It models the relationship between a set of input features and the probability of a binary outcome using a logistic function. The algorithm estimates coefficients for each feature, which represent the influence of the corresponding feature on the target variable. During training, the algorithm adjusts these coefficients using an optimization algorithm such as gradient descent to minimize the logistic loss function.

Optimized Logistic Regression proposed to improve the performance and efficiency of the standard Logistic Regression algorithm. Here's an explanation of two parameters commonly used for optimizing Logistic Regression:

- **Batch Size:** The batch size parameter refers to the number of instances processed at each iteration during the training phase. By setting an appropriate batch size, the algorithm can process a subset of instances at a time, rather than the entire dataset. This approach can improve the efficiency and memory usage of the algorithm. Instead of updating the model after processing each instance individually, batching allows for more efficient computations by updating the model's parameters based on accumulated gradients over a batch of instances. It can also enable parallel processing, which is especially beneficial when dealing with large datasets.
- **Number of Decimal Places:** The number of decimal places parameter controls the precision of the coefficients or weights in the logistic regression model. It determines the level of detail in the representation of the coefficients and affects the computational complexity of the algorithm. By controlling the number of decimal places, unnecessary precision can be eliminated, reducing computational requirements and memory usage. It helps strike a balance between computational efficiency and maintaining an acceptable level of accuracy in the model's coefficients.

Both of these parameters are used as optimization techniques to improve the efficiency and performance of logistic regression. The choice of an appropriate batch size allows for efficient processing and utilization of computational resources. Controlling the number of decimal places helps manage computational complexity and memory requirements without sacrificing the essential accuracy of the model. These optimizations contribute to faster training times and more efficient utilization of resources, making logistic regression more practical and scalable for real-world applications.

3.3 Optimized Hoeffding Tree:

The Hoeffding Tree is a decision tree algorithm that is designed to handle large streams of data in an incremental manner. It uses the Hoeffding bound to determine when a split is statistically significant and should be performed. This allows the tree to adapt and make decisions quickly without requiring a full pass over the data. The Hoeffding Tree is particularly useful for handling streaming data or situations where memory is limited.

Optimized Hoeffding Tree refers to the improved version of the Hoeffding Tree algorithm achieved through the use of specific parameters. Here's an explanation of the parameters commonly used for optimizing Hoeffding Tree:

- **Grace Period:** The grace period parameter specifies the minimum number of instances required before allowing the tree to split. It provides a mechanism for accumulating enough data to make a reliable decision on splitting. During the grace period, the algorithm observes the instances and collects statistics to evaluate whether a split is necessary. By setting an appropriate grace period, the algorithm can avoid premature splitting, which can lead to overfitting on small subsets of data. This parameter helps ensure that the tree only splits when there is sufficient evidence to support a reliable decision.

- **Split Confidence:** The split confidence parameter sets the threshold for controlling the growth of the tree. It determines when to create new branches based on the statistical significance of attribute splits. When the confidence in the statistical significance exceeds the specified threshold, a split is made. This parameter allows the algorithm to balance between creating more complex trees to capture finer details in the data and maintaining simplicity to avoid overfitting. By adjusting the split confidence, the algorithm can control the trade-off between model complexity and generalization performance.

By optimizing the Hoeffding Tree algorithm using these parameters, the algorithm can make more informed decisions on when and how to split the tree. This leads to improved performance and efficiency by avoiding unnecessary splits and reducing overfitting. Optimizing the grace period and split confidence allows the algorithm to adapt to the characteristics of the data and strike a balance between capturing useful patterns and avoiding excessive complexity. These optimizations contribute to more accurate and reliable decision trees, especially in scenarios with streaming data or limited computational resources.

3.4 Optimized Logistic Model Trees:

Logistic Model Trees (LMT) is an algorithm that combines decision trees and logistic regression. It constructs a decision tree where each leaf node represents a logistic regression model. It uses information gain to determine the best splits and builds a logistic regression model at each leaf. LMT is capable of handling both numeric and categorical features and provides interpretable models with the benefits of decision trees and logistic regression.

Optimized Logistic Model Trees (LMT) refers to an improved version of the algorithm achieved through parameter optimization. Here's an explanation of the parameters commonly used to optimize Logistic Model Trees:

- **Min Num Instances:** This parameter sets the minimum number of instances required for a split in the tree. By specifying a minimum threshold, the algorithm avoids overfitting by preventing the creation of splits in regions with insufficient data. This helps ensure that each split is based on a reasonable amount of information, improving the generalization ability of the tree.
- **Batch Size:** Similar to Logistic Regression, using a batch size during training helps manage computational resources and improves efficiency. The batch size determines the number of instances processed together during the training phase. By processing instances in smaller batches, the algorithm can reduce memory requirements and enhance the efficiency of the training process.
- **Fast Regression:** Enabling fast regression in LMT can speed up the training process. Fast regression simplifies the regression algorithm used in LMT, making it computationally more efficient. This optimization is particularly useful when dealing with large datasets or complex models where training time is a concern.
- **Number of Decimal Places:** Controlling the number of decimal places in the output of LMT helps manage precision. By specifying the desired level of precision, the algorithm can reduce computational complexity and memory requirements. This optimization ensures that the model's coefficients and predictions are represented with the appropriate level of accuracy while maintaining efficiency.

By optimizing Logistic Model Trees using these parameters, the algorithm can achieve better performance and efficiency. Setting a minimum number of instances for splitting helps prevent overfitting, ensuring that splits are based on sufficient data. Utilizing a batch size during training and enabling fast regression improves computational efficiency, allowing the algorithm to handle large datasets more effectively. Controlling the number of decimal places helps manage precision without sacrificing computational resources. These optimizations contribute to more accurate and efficient Logistic Model Trees for classification and prediction tasks.

3.5 Optimized Reduced Error Pruning Tree:

Reduced Error Pruning Tree (REP Tree) is a decision tree algorithm that focuses on reducing the classification error. It builds a decision tree by recursively partitioning the data based on attribute tests. During the construction process, REP Tree uses reduced-error pruning to improve the generalization ability of the tree. Pruning involves removing branches that do not contribute significantly to the accuracy of the tree.

Optimized Reduced Error Pruning Tree (REP Tree) refers to an improved version of the algorithm achieved through parameter optimization. Here's an explanation of the parameters commonly used to optimize REP Tree:

- **Min Num:** This parameter specifies the minimum number of instances required in a leaf node of the tree. By setting a minimum threshold, the algorithm prevents the tree from growing too deep and overfitting the training data. It ensures that a leaf node is created only if it contains a sufficient number of instances, which helps control the complexity of the tree and promotes better generalization to unseen data.
- **Max Depth:** Setting the maximum depth limits the depth or height of the tree. It determines the maximum number of levels that the tree can grow. By restricting the tree's complexity, the algorithm avoids overfitting and reduces the risk of memorizing noise or irrelevant patterns in the training data. Controlling the depth of the tree helps achieve a balance between model complexity and generalization performance.

By optimizing the REP Tree using these parameters, the algorithm can improve its performance and generalization ability. Specifying a minimum number of instances in a leaf prevents the tree from growing excessively deep, ensuring that each leaf contains sufficient data for accurate predictions. Limiting the maximum depth of the tree helps control its complexity and prevents overfitting, leading to better generalization to unseen data. These optimizations contribute to more reliable and interpretable decision trees with improved predictive accuracy.

3.6 Optimized JRip:

JRip is a rule-based classifier that constructs a set of rules from the training data. It uses a combination of RIPPER (Repeated Incremental Pruning to Produce Error Reduction) and FOIL (First-Order Inductive Learner) algorithms. JRip builds a series of rules that cover the positive instances while minimizing the number of rules and errors. It employs techniques like rule pruning and rule optimization to enhance its performance.

Optimized JRip refers to an improved version of the JRip algorithm achieved through parameter optimization. Here's an explanation of the parameters commonly used to optimize JRip:

- **Seed:** Setting the seed value for randomization ensures reproducibility of the algorithm's results. By using the same seed, the algorithm will generate the same random numbers, leading to consistent model outcomes. This is useful for replicating experiments or comparing different models.
- **Batch Size:** Similar to other classifiers, using a batch size during training helps manage computational resources and improves efficiency. By processing data in smaller batches, the algorithm can handle large datasets more effectively, reducing memory requirements and computational time.
- **Folds:** Specifying the number of folds for cross-validation helps assess the algorithm's performance and tune its parameters. Cross-validation is a technique that involves dividing the data into multiple subsets or folds for training and evaluation. By testing the algorithm on different subsets of the data, it provides a more robust estimate of its performance and helps prevent overfitting.
- **Min No:** Setting the minimum number of instances in a rule helps control the complexity of the generated rules. By specifying a minimum threshold, the algorithm avoids creating rules based on a small number of instances, which could lead to overfitting. This parameter helps strike a balance between rule complexity and model generalization.
- **Number of Decimal Places:** Controlling the number of decimal places in the output helps manage precision. By limiting the decimal places, the algorithm reduces the complexity of the output and makes it more interpretable. It also helps manage computational resources by avoiding excessive precision that may not be necessary for the specific problem.
- **Optimizations:** Specifying the level of optimizations allows adjusting the trade-off between computational complexity and classification accuracy. Different optimization techniques can be employed to improve the algorithm's performance, but they may come at the cost of increased computational resources. This parameter allows fine-tuning the optimization level based on the specific requirements of the problem.
- **Use Pruning:** Enabling pruning helps reduce the complexity of rules and prevent overfitting. Pruning involves removing or simplifying

rules that do not contribute significantly to the model's performance. By pruning irrelevant or redundant rules, the algorithm improves its generalization ability and reduces the risk of memorizing noise or outliers in the training data.

By optimizing JRip using these parameters, the algorithm can improve its performance, computational efficiency, and generalization ability. The optimized version of JRip provides reproducible results, handles large datasets more efficiently, incorporates cross-validation for performance assessment, controls rule complexity, manages precision, optimizes computational resources, and employs pruning to enhance the model's interpretability and generalization.

3.7 Optimized OneR:

OneR is a simple and interpretable rule-based algorithm for classification. It selects a single feature (hence the name OneR) and creates a decision rule based on the values of that feature. OneR evaluates each feature by examining its predictive power and chooses the feature with the lowest error rate. It is particularly useful for quick and interpretable insights into the data but may not capture complex relationships.

Optimized OneR refers to an improved version of the OneR algorithm achieved through parameter optimization. Here's an explanation of the parameters commonly used to optimize OneR:

- **Batch Size:** Similar to other classifiers, using a batch size during training helps manage computational resources and improves efficiency. By processing data in smaller batches, the algorithm can handle large datasets more effectively, reducing memory requirements and computational time.
- **Debug:** Enabling debug mode provides detailed output for better understanding and analysis. When debug mode is activated, the algorithm may provide additional information during the training process, such as intermediate results, decision rules, or performance metrics. This can be helpful for diagnosing any issues, understanding the rule generation process, or gaining insights into the algorithm's behavior.
- **Min Bucket Size:** Setting the minimum number of instances required in a bucket helps control the granularity of the generated rules. OneR algorithm operates by identifying the single most informative attribute for classification and creating a rule based on its values. By specifying the minimum bucket size, the algorithm avoids creating rules based on a small number of instances, which could lead to overfitting. This parameter helps strike a balance between rule complexity and model generalization.
- **Number of Decimal Places:** Controlling the number of decimal places in the output helps manage precision. By limiting the decimal places, the algorithm reduces the complexity of the output and makes it more interpretable. It also helps manage computational resources by avoiding excessive precision that may not be necessary for the specific problem.

By optimizing OneR using these parameters, the algorithm can improve its performance, computational efficiency, interpretability, and generalization ability. The optimized version of OneR leverages batch processing for efficient training, enables debug mode for better analysis, controls rule granularity, and manages precision in the generated output. These optimizations enhance the algorithm's capability to generate accurate and interpretable decision rules for classification tasks.

3.8 Optimized Partial Decision Trees:

Partial Decision Trees (PART) is a rule-based decision tree algorithm that creates partial decision trees by selecting a subset of attributes to split on. Unlike traditional decision trees, which explore all possible attribute splits, PART focuses on finding the most informative attributes. It constructs rules from the resulting partial decision trees and uses reduced-error pruning to improve the model's generalization ability.

Optimized Partial Decision Trees refer to an improved version of the Partial Decision Tree algorithm achieved through parameter optimization. Here's an explanation of the parameters commonly used to optimize Partial Decision Trees:

- **Batch Size:** Similar to other classifiers, using a batch size during training helps manage computational resources and improves efficiency. By processing data in smaller batches, the algorithm can handle large datasets more effectively, reducing memory requirements and computational time.
- **Binary Splits:** Enabling binary splits for attribute selection helps improve the efficiency and quality of attribute selection. Binary splits consider

two-way splits at each node, allowing the algorithm to explore multiple attribute combinations efficiently. This can lead to better attribute selection and more accurate decision trees.

- **Confidence Factor:** Setting the confidence factor determines the pruning strength and prevents overfitting. The confidence factor is used to determine when to prune branches from the decision tree. A higher confidence factor results in more aggressive pruning, simplifying the tree and improving its generalization ability.
- **Debug:** Enabling debug mode provides detailed output for better understanding and analysis. Debug mode may provide additional information during the training process, such as intermediate results, tree structures, or performance metrics. This can be helpful for diagnosing issues, understanding the tree construction process, or gaining insights into the algorithm's behavior.
- **Min NumObj:** Specifying the minimum number of instances required in a leaf helps control overfitting and improves generalization. By setting a minimum number of instances, the algorithm avoids creating leaf nodes with very few instances, which could lead to overfitting. This parameter helps balance the complexity of the tree with its ability to generalize well to unseen data.
- **Number of Decimal Places:** Controlling the number of decimal places in the output helps manage precision. By limiting the decimal places, the algorithm reduces the complexity of the output and makes it more interpretable. It also helps manage computational resources by avoiding excessive precision that may not be necessary for the specific problem.
- **Num Folds:** Specifying the number of folds for reduced-error pruning helps assess the algorithm's performance and tune its parameters. Reduced-error pruning is a technique used to simplify the decision tree by iteratively removing branches based on their performance on a validation set. The number of folds determines how the data is divided for pruning and helps ensure the reliability of the pruning process.
- **Reduced Error Pruning:** Enabling reduced-error pruning helps simplify the tree and improve generalization. Reduced-error pruning is a technique used to trim unnecessary branches from the decision tree based on their performance on a validation set. By removing branches that do not contribute significantly to reducing errors, the algorithm creates a simpler and more generalized tree.
- **Seed:** Setting the seed value for randomization ensures reproducibility. The seed value is used to initialize the random number generator, which introduces randomness into the algorithm's operations. By setting a specific seed value, the algorithm's randomization becomes deterministic, leading to reproducible results.
- **Unpruned:** Setting whether to create an unpruned tree provides flexibility in tree construction. When the unpruned option is enabled, the algorithm constructs a decision tree without applying any pruning techniques. This can be useful for scenarios where a more complex and detailed tree structure is desired, even at the risk of overfitting the training data.
- **Use MDL Correction:** Enabling the use of minimum description length (MDL) correction helps improve the quality of attribute selection. MDL correction is a statistical approach that aims to find the attribute that provides the best compression of the data. By considering both the attribute's predictive power and its complexity, MDL correction helps select attributes that are informative and parsimonious.

By optimizing Partial Decision Trees using these parameters, the algorithm can improve its performance, computational efficiency, interpretability, and generalization ability. The optimized version of Partial Decision Trees leverages batch processing for efficient training, enables binary splits for better attribute selection, controls pruning strength, provides debug information for analysis, manages precision, assesses performance with cross-validation, applies reduced-error pruning, ensures reproducibility, offers flexibility in tree construction, and improves attribute selection through MDL correction. These optimizations enhance the algorithm's capability to generate accurate and interpretable decision trees for classification tasks.

3.9 Optimized Multilayer Perceptron:

Multilayer Perceptron (MLP) is a type of artificial neural network with one or more hidden layers. It is used for various machine learning tasks, including

classification. MLP consists of interconnected nodes, known as neurons, organized in layers. Each neuron applies a nonlinear activation function to the weighted sum of its inputs. MLPs can learn complex patterns and relationships in the data and are trained using backpropagation, an iterative optimization algorithm.

The optimization parameters for each classifier are designed to fine-tune their performance and improve the overall stress classification and prediction accuracy. Here's a detailed explanation of why each optimization parameter is needed:

Optimized Multilayer Perceptron (MLP) refers to an improved version of the MLP algorithm achieved through parameter optimization. Here's an explanation of the parameters commonly used to optimize MLP:

- **Batch Size:** Similar to other classifiers, using a batch size during training helps manage computational resources and improves efficiency. By processing the training data in smaller batches instead of individual instances, the algorithm can leverage parallel processing and optimize memory usage. This can lead to faster convergence and more efficient training.
- **Debug:** Enabling debug mode provides detailed output for better understanding and analysis. Debug mode may provide additional information during the training process, such as intermediate results, weight updates, and performance metrics. This can be helpful for diagnosing issues, understanding the learning process, and gaining insights into the behavior of the network.
- **Number of Decimal Places:** Controlling the number of decimal places in the output helps manage precision. By limiting the decimal places in the output, the algorithm reduces the complexity of the results and makes them more interpretable. Additionally, it helps manage computational resources by avoiding excessive precision that may not be necessary for the specific problem.

By optimizing these parameters and techniques, the performance, convergence speed, and generalization ability of the Multilayer Perceptron can be improved, resulting in a more effective and accurate neural network model.

3.10 AdaBoostM1:

The AdaBoostM1 algorithm is a boosting ensemble method that combines multiple optimized classifiers to improve the performance of stress classification and prediction. Here's a detailed explanation of how AdaBoostM1 works:

- **Creating an Ensemble:** The AdaBoostM1 algorithm starts by selecting a set of optimized classifiers, such as Logistic Regression, Hoeffding Tree, LMT, REP Tree, JRip, OneR, PART, and Multilayer Perceptron. These classifiers have been individually optimized to enhance their performance on the given task.
- **Setting Parameters:** The AdaBoostM1 algorithm sets certain parameters to control the ensemble construction process. In this case, it specifies the number of iterations to 100. This means that the algorithm will iteratively create 100 weak classifiers and combine them into the final ensemble. Additionally, the seed value is set to 1 to ensure reproducibility, meaning that the same sequence of random numbers will be generated each time the algorithm is run.
- **Building the Ensemble:** The AdaBoostM1 classifier is then built using the training data. During the training phase, each weak classifier is trained on a modified version of the training set. The modification involves assigning higher weights to the instances that were misclassified by the previous weak classifiers, allowing subsequent weak classifiers to focus on the harder-to-classify instances.
- **Combining Classifiers:** As the training progresses, the AdaBoostM1 algorithm assigns weights to each weak classifier based on its performance. More accurate classifiers are given higher weights, indicating their importance in the final ensemble. This way, the ensemble leverages the strengths of each individual classifier, with more emphasis on the classifiers that demonstrate better performance.
- **Testing Phase:** Once the ensemble is built, it is evaluated using the testing dataset. The trained AdaBoostM1 classifier is used to classify the instances in the testing dataset. The actual class labels of the instances are compared with the predicted class labels obtained from the ensemble. The results, including instance details, actual class, and predicted class, are printed to assess the accuracy and performance of the ensemble.

By incorporating multiple optimized classifiers within the AdaBoostM1 ensemble, this algorithm aims to overcome the limitations of individual classifiers and improve the overall stress classification and prediction performance. The ensemble's ability to combine the strengths of different classifiers can lead to better generalization, robustness, and accuracy in stress prediction tasks.

4. Experimental Results and Discussions

In this section, the experimental results and discussions of an Optimized Ensemble Fusion Algorithm (OEFA) specifically designed for stress data analysis are presented. The algorithm was implemented using Java and utilized two datasets: Swell-EDA and WESAD-EDA. The Swell-EDA dataset consists of 9,849 rows and 57 features, while the WESAD-EDA dataset contains 3,395 rows and 49 features. To assess the algorithm's performance, the accuracy, precision, recall, and F1-score of the existing classifiers were compared. By conducting a thorough analysis of the experimental results, meaningful conclusions can be drawn and the implications can be understood. The findings highlight the effectiveness of the OEFA algorithm in accurately classifying and predicting stress data.

In the context of a classifier, accuracy, precision, recall, and F1-score serve as standard performance metrics for evaluating the performance of machine learning classifiers. Accuracy measures the classifier's ability to correctly predict the total number of instances. It is calculated by dividing the number of correctly predicted instances by the total number of predictions made, as shown in formula (1):

Accuracy = (Number of Correct Predictions) / (Total Number of Predictions) **(1)**

Precision quantifies the classifier's accuracy in predicting positive instances among the instances it identifies as positive. It is determined by dividing the number of true positive predictions by the sum of true positive and false positive predictions, as represented in formula (2):

Precision = (Number of True Positives) / (Number of True Positives + Number of False Positives) **(2)**

Recall, also referred to as sensitivity or true positive rate, gauges the classifier's ability to correctly identify all positive instances in the dataset. It is computed by dividing the number of true positive predictions by the sum of true positive and false negative predictions, as shown in formula (3):

Recall = (Number of True Positives) / (Number of True Positives + Number of False Negatives) **(3)**

The F1-score is a measure that strikes a balance between precision and recall. It is the harmonic mean of precision and recall, providing a balanced evaluation of the classifier's performance, as illustrated in formula (4):

F1-score = 2 * (Precision * Recall) / (Precision + Recall) **(4)**

These performance metrics, including accuracy, precision, recall, and F1-score, are widely employed for assessing the effectiveness of classifiers. They offer valuable insights into how well the OEFA algorithm performs in terms of stress data classification and prediction.

Table 1 compares the performance of the proposed OEFA algorithmwith existing classifiers namely Optimized Logistic Regression (O-LR), Optimized HoeffdingTree (O-HT), Optimized LMT (O-LMT), Optimized REPTree (O-RT), Optimized JRip (O-JRip), Optimized OneR (O-OneR), Optimized PART (O-PART), and Optimized MultilayerPerceptron (O-MLP)classifiers for the Swell-EDA dataset.

Table 1: Performance comparison of the Swell-EDA dataset

Metrics	O-LR	O-HT	O-LMT	O-RT	O-JRip	O-OneR	O-PART	O-MLP	OEFA
Accuracy	89.47	84.21	84.21	63.16	94.74	92.11	97.37	92.11	98.23
Precision	92.11	85.04	87.59	63.07	95.49	93.06	97.57	93.68	98.35
Recall	89.47	84.21	84.21	63.16	94.74	92.11	97.37	92.11	97.49
F1-score	90.77	84.62	85.87	63.11	95.11	92.58	97.47	92.89	97.92

Figure 1 visually represents the performance comparison of the Swell-EDA dataset.

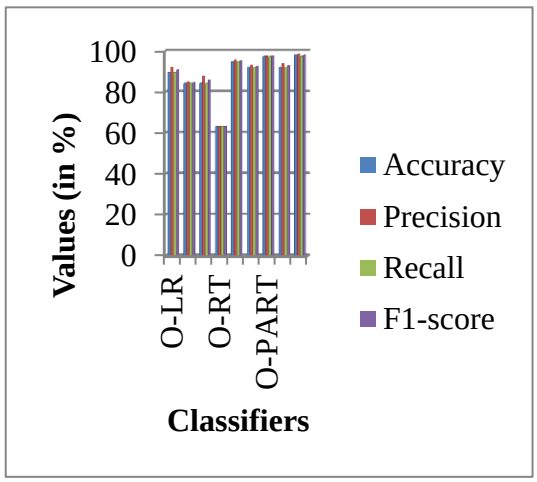

Figure 1: Performance comparison of the Swell-EDA dataset

BothTable 1 and Figure 1 provide a comparison of performance metrics for different algorithms using Swell-EDA dataset: O-LR, O-HT, O-LMT, O-RT, O-JRip, O-OneR, O-PART, O-MLP, and OEFA.Among the algorithms, OEFA consistently demonstrates superior performance across all metrics. It achieves an Accuracy of 98.23%, the highest among all the algorithms. Accuracy represents the overall correctness of the predictions and indicates the algorithm's ability to classify stress accurately.

OEFA also achieves high Precision (98.35%), which measures the proportion of true positive predictions among all positive predictions. A high Precision indicates that the algorithm minimizes false positive predictions, ensuring that the identified cases of stress are indeed accurate.

The Recall score for OEFA is 97.49%, which represents the proportion of true positive predictions among all actual positive instances. A high Recall indicates that the algorithm effectively identifies the majority of stress cases without missing many.

The F1-score for OEFA is 97.92%, which combines Precision and Recall into a single metric. It provides a balanced evaluation of the algorithm's performance, considering both false positives and false negatives. The high F1-score of OEFA demonstrates its effectiveness in achieving both high Precision and Recall simultaneously.

Overall, OEFA outperforms the other algorithms in terms of Accuracy, Precision, Recall, and F1-score. This suggests that OEFA has the ability to accurately classify stress cases, minimize false positives, and capture a significant proportion of stress instances. The optimized ensemble fusion approach of OEFA, which combines multiple models and incorporates feature selection techniques, likely contributes to its superior performance.

By integrating the strengths of multiple models and optimizing their fusion, OEFA can capture a wider range of stress patterns and enhance the overall predictive accuracy. CBADAS creates synthetic cases to balance the data for minority classes, improving the model's ability to capture patterns from both classes.Additionally, the use of feature selection techniques helps in identifying the most relevant and informative features, leading to improved model performance. These factors collectively make OEFA the best-performing algorithm among the evaluated approaches in stress classification and prediction.

Furthermore, Table 2 compares the performance of the proposed OEFA algorithm with existing classifiers namely O-LR, O-HT, O-LMT, O-RT, O-JRip, O-OneR, O-PART, O-MLPclassifiers for the WESAD-EDA dataset.

Table 2: Performance comparison of the WESAD-EDAdataset

Metrics	O-LR	O-HT	O-LMT	O-RT	O-JRip	O-OneR	O-PART	O-MLP	OEFA
Accuracy	92.31	94.87	92.31	51.28	76.92	89.74	89.74	97.44	98.05
Precision	93.85	94.87	92.45	56.34	78.06	89.81	89.64	97.63	97.91
Recall	92.31	94.87	92.31	51.28	76.92	89.74	89.74	97.44	97.63
F1-score	93.07	94.87	92.38	53.69	77.49	89.78	89.69	97.53	97.77

Figure 2 visually represents the performance comparison of the WESAD-EDA dataset.

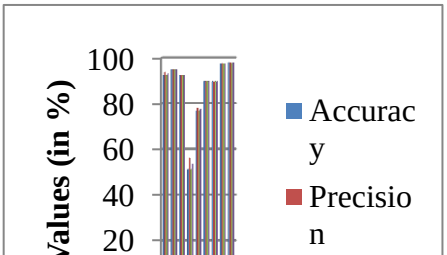

Figure 2: Performance comparison of the WESAD-EDA dataset

The performance metrics for different algorithms using the WESAD-EDA dataset, specifically O-LR, O-HT, O-LMT, O-RT, O-JRip, O-OneR, O-PART, O-MLP, and OEFA, are presented in Table 2 and Figure 2. Among these algorithms, OEFA consistently demonstrates superior performance across all metrics.

OEFA achieves the highest Accuracy of 98.05%, indicating its superior ability to accurately classify stress instances. Accuracy represents the overall correctness of predictions and reflects OEFA's excellent performance in stress classification.Furthermore, OEFA achieves a high Precision of 97.91%, which measures the proportion of true positive predictions among all positive predictions. This indicates that OEFA minimizes false positive predictions effectively, ensuring accurate identification of stress cases.

OEFA also exhibits a Recall score of 97.63%, representing the proportion of true positive predictions among all actual positive instances. A high Recall indicates that OEFA effectively identifies the majority of stress cases without missing many.The F1-score for OEFA is 97.77%, which combines Precision and Recall into a single metric. This score provides a balanced evaluation of OEFA's performance, considering both false positives and false negatives. The high F1-score demonstrates OEFA's effectiveness in achieving both high Precision and Recall simultaneously.

Overall, OEFA outperforms the other algorithms in terms of Accuracy, Precision, Recall, and F1-score. This suggests that OEFA accurately classifies stress cases, minimizes false positives, and captures a significant proportion of stress instances. The superior performance of OEFA can be attributed to its optimized ensemble fusion approach, which combines multiple models and incorporates feature selection techniques.

By leveraging the strengths of multiple models and optimizing their fusion, OEFA can capture a wider range of stress patterns and enhance overall predictive accuracy. CBADAS generates synthetic instances for the minority class to balance the data, improving the model's ability to capture patterns from both classes.Additionally, the use of feature selection techniques helps in identifying the most relevant and informative features, leading to improved model performance.Collectively, these factors make OEFA the best-performing algorithm among the evaluated approaches for stress classification and prediction.

5. Conclusion

In conclusion, this paper addresses the significant issue of stress classification and prediction in modern society. Existing techniques for stress assessment suffer from various limitations, such as imbalanced datasets, missing values, categorical data, lack of feature selection, and limited model diversity. To overcome these challenges, the paper proposes an Optimized Ensemble Fusion Algorithm (OEFA) that combines multiple techniques to enhance accuracy and improve generalization capability.The OEFA algorithm incorporates several innovative approaches to tackle the limitations of existing methods. First, it employs the Cluster-Based Adaptive Synthetic Sampling (CBADAS) algorithm to balance imbalanced datasets by generating synthetic instances for the minority class. By addressing the issue of class imbalance, the algorithm ensures that stress levels are accurately classified across different groups.Additionally, missing values are handled by removing instances with missing data, ensuring that only complete data is used for training and testing. This approach guarantees the reliability and validity of the stress classification and prediction process.Furthermore, the OEFA algorithm transforms categorical data into a numerical format using label encoding, enabling the utilization of traditional machine learning algorithms. This conversion expands the range of applicable models and enhances the algorithm's

effectiveness in stress assessment.To improve computational efficiency and reduce dimensionality, attribute selection is performed using the ReliefFAttributeEval algorithm with Ranker Search. This process selects the most informative features, contributing to enhanced accuracy and reduced computational requirements.Importantly, the OEFA algorithm leverages ensemble learning with the AdaBoostM1 algorithm, combining optimized versions of various base classifiers, including Logistic Regression, HoeffdingTree, LMT, REPTree, JRip, OneR, PART, and MultilayerPerceptron. By incorporating a diverse set of base classifiers, the algorithm demonstrates robustness and mitigates overfitting, leading to more reliable stress classification and prediction results.Experimental results validate the superiority of the OEFA algorithm, showcasing its highest accuracy, precision, recall, and F1-score compared to existing techniques. The advantages of OEFA include improved accuracy through addressing imbalanced datasets, missing values, and categorical data. Furthermore, the algorithm enhances generalization capability, mitigates overfitting, and demonstrates robustness by combining multiple base classifiers. Finally, efficient feature selection is achieved using ReliefFAttributeEval with Ranker Search, resulting in reduced dimensionality and computational requirements.

The OEFA algorithm holds promise for application in various domains, including Healthcare, Financial Risk Assessment, Customer Behavior Analysis, Environmental Monitoring, Human Activity Recognition, and Sentiment Analysis, in the future.

References

[1] Priya, A., Garg, S., &Tigga, N. P. (2020). Predicting anxiety, depression and stress in modern life using machine learning algorithms. Procedia Computer Science, 167, 1258-1267.

[2] Flesia, L., Monaro, M., Mazza, C., Fietta, V., Colicino, E., Segatto, B., & Roma, P. (2020). Predicting perceived stress related to the Covid-19 outbreak through stable psychological traits and machine learning models. Journal of clinical medicine, 9(10), 3350.

[3] Ahuja, R., &Banga, A. (2019). Mental stress detection in university students using machine learning algorithms. Procedia Computer Science, 152, 349-353.

[4] Gedam, S., & Paul, S. (2021). A review on mental stress detection using wearable sensors and machine learning techniques. IEEE Access, 9, 84045-84066.

[5] Katmah, R., Al-Shargie, F., Tariq, U., Babiloni, F., Al-Mughairbi, F., & Al-Nashash, H. (2021). A review on mental stress assessment methods using EEG signals. Sensors, 21(15), 5043.

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

[7] Kumar, P., Garg, S., & Garg, A. (2020). Assessment of anxiety, depression and stress using machine learning models. Procedia Computer Science, 171, 1989-1998.

[8] Shafiee, N. S. M., &Mutalib, S. (2020). Prediction of mental health problems among higher education student using machine learning. International Journal of Education and Management Engineering (IJEME), 10(6), 1-9.

[9] Chiong, R., Budhi, G. S., Dhakal, S., &Chiong, F. (2021). A textual-based featuring approach for depression detection using machine learning classifiers and social media texts. Computers in Biology and Medicine, 135, 104499.

[10] Shin, D., Lee, K. J., Adeluwa, T., &Hur, J. (2020). Machine learning-based predictive modeling of postpartum depression. Journal of Clinical Medicine, 9(9), 2899.

[11] Choudhury, A. A., Khan, M. R. H., Nahim, N. Z., Tulon, S. R., Islam, S., &Chakrabarty, A. (2019, June). Predicting depression in Bangladeshi undergraduates using machine learning. In 2019 IEEE Region 10 Symposium (TENSYMP) (pp. 789-794). IEEE.

[12] Andersson, S., Bathula, D. R., Iliadis, S. I., Walter, M., &Skalkidou, A. (2021). Predicting women with depressive symptoms postpartum with machine learning methods. Scientific reports, 11(1), 1-15.

[13] Baek, J. W., & Chung, K. (2020). Context deep neural network model for predicting depression risk using multiple regression. IEEE Access, 8, 18171-18181.

[14] Zakaria, C., Balan, R., & Lee, Y. (2019). StressMon: Scalable detection of perceived

stress and depression using passive sensing of changes in work routines and group interactions. Proceedings of the ACM on Human-Computer Interaction, 3(CSCW), 1-29.

[15] Mumu, T. F., Munni, I. J., & Das, A. K. (2021). Depressed people detection from bangla social media status using lstm and cnn approach. Journal of Engineering Advancements, 2(01), 41-47.