**Performance Analysis and Comparison of Machine Learning Models with Optimized Feature Selection Techniques**

**Submitted by: Zhang.kaijie(Jack)**

**Student ID: 202118020413**

**Abstract**

**Background:** This coursework focuses on predicting chronic heart failure, a critical health issue with significant mortality rates. Traditional methods, such as clinical assessments and echocardiography, have been pivotal but often lack the precision required for early intervention [1]. Recent advancements in AI, particularly machine learning, have shown promise in enhancing diagnostic accuracy [2].

**Method:** I employed a rigorous approach involving a thorough data preprocessing phase to handle missing values and normalize data. Feature selection using Recursive Feature Elimination (RFE), Boruta, and Lasso. Eight machine learning classifiers were optimized and evaluated: Support Vector Machine (SVM), Logistic Regression (LR), K-Nearest Neighbors (KNN), Decision Tree (DT), Adaptive Boosting (Ada), Bagging Classifier (Bag), Stacking Classifier (Stack), and Voting Classifier. Performance metrics included accuracy, precision, recall, specificity, F1-score, AUC-ROC, confusion matrix, and training time.

**Result:** Feature Selected by the Optimized RFE, Leads the Bagging Classifier demonstrated superior performance with an accuracy of 90.63%, F1-Score with 89.60%, Specificity with 96.69%, AUC-ROC of 95.61%, Precision-Recall with 96.45%.

**Conclusion:** This coursework successfully applied and compared multiple machine learning models for predicting chronic heart failure. The optimized RFE and Bagging Classifier emerged as the most effective, demonstrating high accuracy and reliability. This study provides a solid foundation for future research aiming to refine heart failure prediction models.

1. **Introduction & Literature Review**

Chronic heart failure (CHF) is a prevalent condition with high mortality, underscoring the need for improved diagnostic methods [3]. Traditional diagnostics, reliant on clinical symptoms and imaging, provide valuable insights but are often subjective and lack sensitivity [4]. The advent of AI, especially machine learning, has introduced objective, data-driven approaches to enhance prediction accuracy [5]. However, existing machine learning models face challenges like overfitting and the curse of dimensionality [6], which can degrade performance [7]. This coursework aims to address these limitations by applying advanced feature selection techniques and an array of classifiers to predict CHF, contributing novel insights and a potentially more accurate diagnostic tool.

The major contributions of this ML coursework 1 on predicting chronic heart failure are as follows:

1. **Comprehensive Feature Selection**: The use of Recursive Feature Elimination (RFE), Boruta, and Lasso for feature selection ensured that the models were trained on the most relevant features, potentially improving their predictive power.
2. **Diverse Model Evaluation**: The coursework involved the optimization and evaluation of eight different machine learning classifiers, providing a thorough comparison of their performance on the task.
3. **Robust Performance Metrics**: A wide range of performance metrics were employed, including accuracy, precision, recall, specificity, F1-score, AUC-ROC, confusion matrix, and training time, offering a holistic view of the models' effectiveness.
4. **Optimization of Classifiers**: Each classifier was optimized to perform at its best, with the Bagging Classifier showing superior results, indicating the success of the optimization process.
5. **Practical Framework**: The coursework provides a practical framework for predicting chronic heart failure that can be built upon for future research and potentially integrated into clinical practice.
6. **In-depth Analysis**: The detailed analysis of each model's performance, including the training time, offers insights into not just the accuracy but also the efficiency of the models.
7. **Potential Clinical Impact**: The high performance of the models, especially the Bagging Classifier, suggests a potential clinical impact by improving the early diagnosis of chronic heart failure.

Table 1 Literature Review Summary

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Authors | Methodology | Data split | Models | Results (%) |
| Choi et al. [1](2017) | HER Data Utilization,  Medical Concept Vectors | Train: Validation: Test = 5:1:1 | RNN with GRUs,  Regularized logistic regression,  Multilayer Perceptron (MLP),  Support Vector Machine (SVM),  K-Nearest Neighbor (KNN) | RNN model AUC 77.70% |
| Chen et al. [2](2017) | Deletion of first and last interval.  Exclusion of RR intervals longer than 3 seconds.  Division into 5-minute segments and interpolation to equal length.  Sparse Auto-Encoder (SAE): Used for unsupervised feature learning from raw RR intervals. | Not explicitly mentioned, but data was randomly selected for training and testing to avoid over-training. | Deep Neural Network (DNN) | Acc 72.44% Sens 50.39% Spec 84.93% |
| Samuel et al. [8] (2017) | Excluded patient data with missing values, resulting in 297 input data samples | Not explicitly mentioned, but an online clinical dataset was used for evaluation. | Hybrid decision support method based on artificial neural networks and fuzzy analytic hierarchy process (Fuzzy\_AHP) techniques | Acc 91.10% |
| Reddy et al. [3] (2018) | Not explicitly mentioned, But H2FPEF Score: Created based on six predictive variables identified through univariable and multivariable logistic regression analysis. | Derivation Cohort: Included 414 consecutive patients (267 HFpEF cases and 147 controls).  Test Cohort: Included 100 consecutive patients (61 HFpEF cases and 39 controls). | LR | AUC 88.60% |
| Wang et al. [5](2019) | RR Interval Signals,  Long Short-Term Memory (LSTM) Network: Used for feature extraction from short-term RR interval data. | 10-fold cross-validation | Combination of the Long Short-Term Memory (LSTM) network and convolution net architecture | Acc 99.22% |
| Acharya et al. [7](2019) | - | The model was trained and tested using four different sets of data (A, B, C, and D). | Convolutional neural network (CNN) | Acc 98.97%  Spec 99.01%  Sens 98.87% |
| Ali et al. [6](2019) | L1&L2 Regularization | - | SVM | Acc 92.22%  Sens 100.00%  Spec 82.92% |
| Mohan et al. [9] (2019) | - | 70% Training, 30% Classification | Hybrid RF | Acc 88.40%  Sens 92.80%  Spec 82.60% |
| Lal et al. [10] (2020) | focuses on the extraction of various types of features from HRV signals. | - | SVM Gaussian,  K-NN, decision tree, SVM RBF, and SVM polynomial | SVM Gaussian  Acc 88.79%  Sens 93.06%  Spec 81.82%  AUC 95.00% |
| Potter et al. [4](2020) | Random Forest Classifier | Training Dataset: Consists of Australian subjects (n=254), with 135 (53%) having Stage B Heart Failure (SBHF).  External Test Dataset: Consists of subjects (n=65), with 27 (42%) having SBHF. | RF | AUC 76.00%  Sens 93.00%  Spec 61.00% |
| Ning et al. [11](2021) | - | The data is divided into segments of 5-minute ECG signals for analysis, with the last segment removed if it is not longer than 5 minutes | Hybrid DL algorithm that is composed of a CNN and a recursive NN | Acc 99.93%  Sens 99.85%  Spec 100% |

# Methodology

This section covers the data collection and preprocessing steps, the optimized feature selections, optimized eight ML classifiers, data-split ratio and evaluation metrics employed to checkmate the model performance including accuracy, precision, recall, specificity, f1-score, AUC-ROC, confusion matrix and training time in seconds.

**2.1.1 Dataset Description**

The dataset for this coursework is ‘Myocardial infarction complications’, from UCI, with a total of 124 variables, with 112 features and 12 targets. There is missing data. There are 1700 instances. I transferred all the data from data format files to csv format files. A partial description of the variables is listed below.

And a full description can be found on the ‘https://archive.ics.uci.edu/dataset/579/myocardial+Infarction

+complications’.

Table 2 Description some of the UCI Myocardial Infarction Complications Dataset

图形用户界面, 应用程序

描述已自动生成

图形用户界面, 应用程序

描述已自动生成

**2.1.2. Data Preprocessing**

In this section, I will explain the data processing steps such as loading and exploration of the data, handling missing data, rescaling and balancing the data.

**a. Data Loading and Exploration**

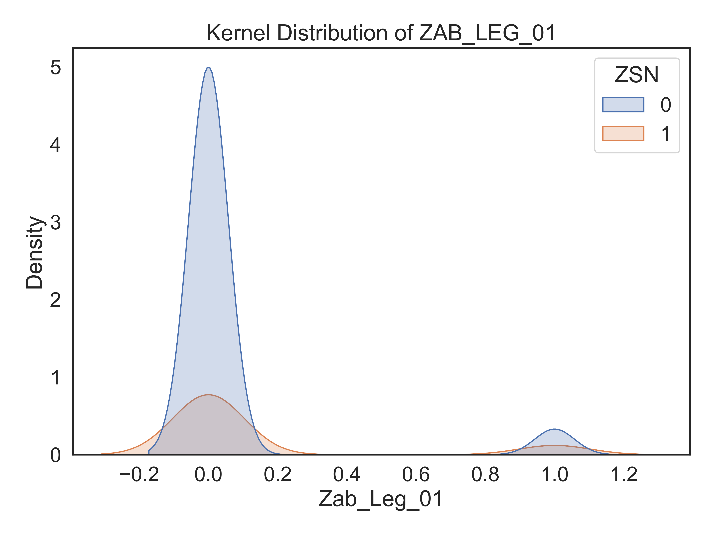
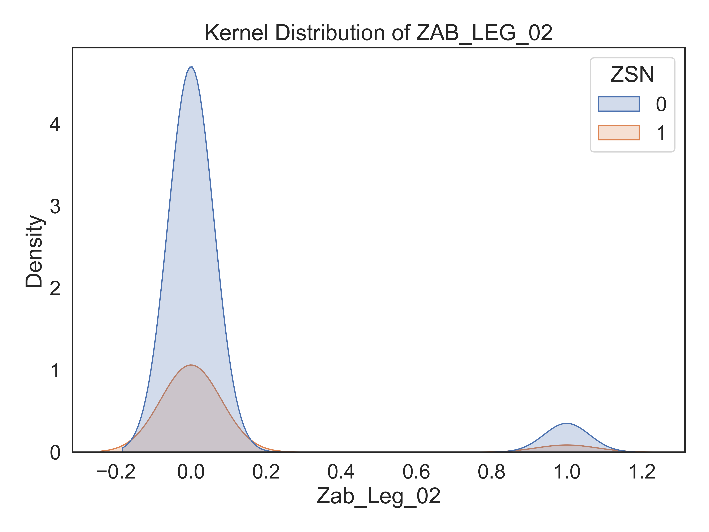
The dataset was loaded from a .data file using pandas, and the data was read into a DataFrame with specified column names. To make the data more accessible and shareable, I exported the data into a .csv format named 'MI.csv'. After loading the CSV data back into the environment, the 'ID' column was dropped as it was not essential for the analysis. After evaluating and processing the missing data, I finally selected 108 features, among which ‘ZSN’(chronic heart failure) is the target of my classification. I explored some of the data distribution and generated a plot to visualize the distribution of each feature.

图表, 直方图

描述已自动生成图表, 直方图

描述已自动生成图表

描述已自动生成图表, 直方图

描述已自动生成图表, 直方图

描述已自动生成图表, 直方图

描述已自动生成图表, 直方图

描述已自动生成图表

描述已自动生成图表, 直方图

描述已自动生成图表, 直方图

描述已自动生成

图表, 直方图

描述已自动生成 图表, 直方图

描述已自动生成图表, 直方图

描述已自动生成图表, 直方图

描述已自动生成图表, 直方图

描述已自动生成 图表, 直方图

描述已自动生成

Figure 1. Kernel distribution Plot

**b. Imputation of Missing Values**

In the imputation of missing values process, I first identified missing values across all columns in the dataset by calculating the percentage of missing data using the isnull() method and multiplying it by 100. 4 Columns with more than 50% missing data were considered too incomplete and were dropped from the dataset using the drop() method. For the remaining columns, I employed an IterativeImputer with a LinearRegression estimator to predict and fill in the missing values. This iterative approach models each feature with missing values as a function of other features, making it suitable for handling non-linear relationships. The imputation was controlled by setting parameters such as max\_iter to 100, tol to 1e-10, and imputation\_order to 'roman'. After the imputation, I transformed the imputed numerical data back into a DataFrame and combined it with the categorical data that had been preprocessed separately. The categorical data was converted to numeric and any remaining missing values were filled with the mode of each column. This comprehensive approach ensured that all missing values were addressed, resulting in a complete and clean dataset ready for further analysis or modeling.

**c. Rescaling of Dataset**

Given the varied scales of the features in the dataset, I decided to rescale the data to ensure that no single feature would unduly influence the model due to its scale. This was achieved using StandardScaler from scikit-learn, which standardizes features by removing the mean and scaling to unit variance. The rescaling process was applied to the numerical features of the dataset, ensuring that all features contribute equally to the distance metrics used in subsequent analyses.

**d. Balancing the Dataset**

The final step in my preprocessing pipeline was to balance the dataset. This was necessary due to the imbalanced class distribution, which could lead to biased model predictions. I employed SVMSMOTE, a technique that synthesizes new samples using a combination of SVM and SMOTE algorithms, to oversample the minority class. This not only increased the number of minority class samples but also preserved the geometric properties of the data. The effectiveness of the balancing was evaluated by comparing the class distributions before and after the application of SVMSMOTE, with the results visualized in a bar plot. The balanced dataset is then ready for model training, ensuring that all classes are equally represented and considered.

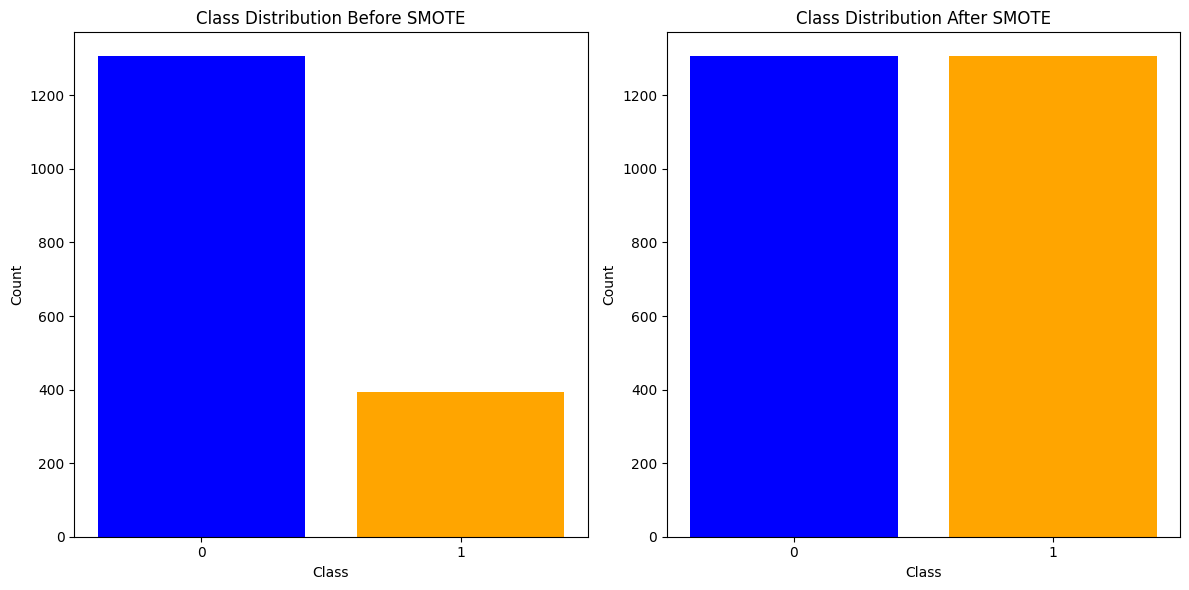


Figure 2. ‘ZSN’ Class Distribution Before and After SMOTE

**2.2 Optimized Feature selection for Chronic Heart Failure (ZSN) Prediction**

**2.2.1 Optimized Boruta**

Algorithm 1: Optimized boruta(Early stop)

**Inputs:**

* u\_scaled: The scaled feature dataset.
* y\_resample: The target variable dataset.
* feature\_names: A list of feature names.

**Output:**

* boruta\_selected\_features: A list of feature names selected by Boruta after the feature selection process.
* u\_boruta: The filtered feature dataset containing only the selected features.
* Plotting the importance of features as determined by Boruta.

**Terminal printed out:**

Early stopping: No significant change in feature selection after 1 iterations.

Boruta selected features:

['S\_AD\_ORIT', 'D\_AD\_ORIT', 'K\_BLOOD', 'NA\_BLOOD', 'ALT\_BLOOD', 'AST\_BLOOD', 'L\_BLOOD', 'ROE', 'AGE', 'SEX', 'GB', 'DLIT\_AG', 'ZSN\_A', 'ant\_im', 'TIME\_B\_S']

Boruta with early stop, selected 15 features out of 107

**Process:**

1. Initialize a robust random forest model (RandomForestClassifier) with 200 estimators, maximum depth of 10, using all available CPU cores (n\_jobs=-1), and a random state of 0 for reproducibility.
2. Create an instance of BorutaPy with the random forest model as the estimator, setting `n\_estimators` to 'auto', a random state of 0, and a maximum number of iterations of 100.
3. Define an early stopping mechanism with early\_stopping\_rounds set to 10, which stops the iteration if there is no significant change in feature selection for the specified number of rounds.
4. Initialize last\_selected\_features to store the features selected in the previous iteration.
5. Run Boruta feature selection in a loop with an early stopping condition:

Fit Boruta to the scaled features and target variable.

Retrieve the features selected by Boruta for the current iteration.

Check if there is no significant change in feature selection compared to the previous iteration. If no change is detected, print a message indicating early stopping and break the loop.

Update last\_selected\_features with the current iteration's selected features.

1. After the loop, obtain the Boolean values indicating the feature selection by Boruta (boruta\_support).
2. Extract the feature names selected by Boruta by masking the original list of feature names with boruta\_support.
3. Filter the scaled feature dataset (u\_scaled) based on the Boolean values of the features selected by Boruta to create u\_boruta.
4. Print the number of features selected by Boruta and the total number of features in the original dataset.
5. Plot the importance of features as determined by Boruta, with the x-axis representing the feature importance ranking and the y-axis representing the feature names. The plot is inverted to show the highest rank at the top.

图表, 条形图

描述已自动生成

Figure 3. Image of the Features selected for the optimized boruta

**2.2.2 Optimized RFE**

Algorithm 1: Optimized RFE(Ensemble)

**Inputs**:

* u\_scaled: The scaled feature dataset.
* y\_resample: The target variable dataset.
* feature\_names: A list of feature names from the dataset.

**Output**:

* u\_ensemble\_rfe: The filtered feature dataset containing only the common features selected by all three models.
* common\_selected\_features: A set of feature names that are commonly selected by the SVM, RandomForest, and Logistic Regression models.
* Cross-validation scores for different numbers of selected features.
* A plot showing the relationship between the number of features selected and the cross-validation score.
* A plot displaying the feature ranking based on their selection by the ensemble RFE.

**Terminal printed out:**

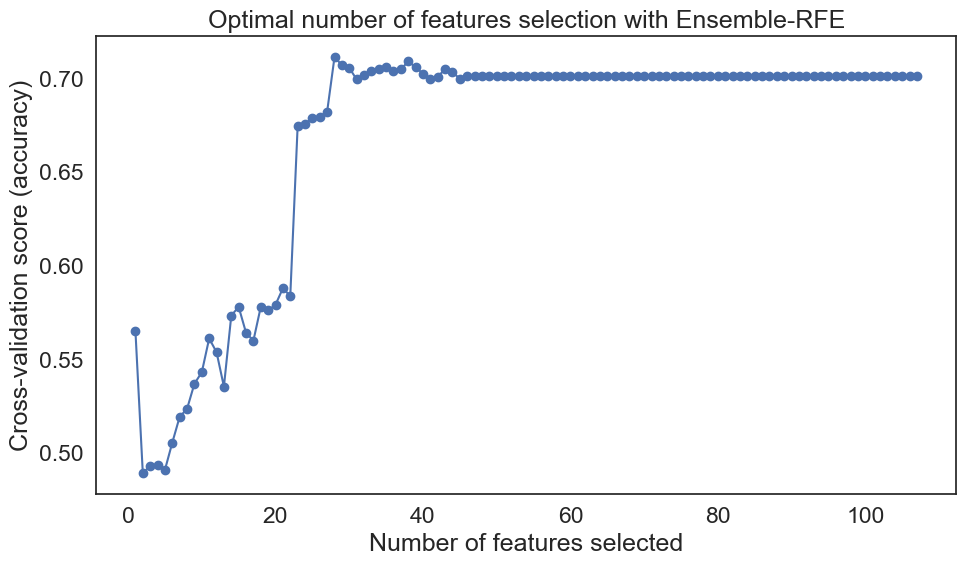
Common Selected Features from Ensemble RFE:

{'SEX', 'GB', 'NA\_R\_1\_n', 'IM\_PG\_P', 'O\_L\_POST', 'ALT\_BLOOD', 'R\_AB\_1\_n', 'L\_BLOOD', 'ANT\_CA\_S\_n', 'K\_SH\_POST', 'ZSN\_A', 'ritm\_ecg\_p\_01', 'TRENT\_S\_n', 'ASP\_S\_n', 'S\_AD\_ORIT', 'LID\_S\_n', 'zab\_leg\_02', 'n\_p\_ecg\_p\_07', 'FK\_STENOK', 'ant\_im', 'K\_BLOOD', 'D\_AD\_ORIT', 'AGE', 'ritm\_ecg\_p\_08', 'GIPO\_K', 'n\_p\_ecg\_p\_03', 'n\_r\_ecg\_p\_04', 'ritm\_ecg\_p\_07', 'zab\_leg\_03', 'MP\_TP\_POST', 'zab\_leg\_01', 'B\_BLOK\_S\_n', 'STENOK\_AN', 'LID\_KB', 'n\_r\_ecg\_p\_05', 'NA\_R\_3\_n', 'post\_im', 'INF\_ANAM', 'IBS\_POST', 'DLIT\_AG', 'NA\_R\_2\_n', 'n\_p\_ecg\_p\_10', 'GEPAR\_S\_n', 'n\_p\_ecg\_p\_12', 'R\_AB\_2\_n', 'ritm\_ecg\_p\_02'}

After Ensemble Feature Selection, selected 46 features out of 107

**Process**:

1. Initialize three different models: SVM with a linear kernel, RandomForestClassifier, and LogisticRegression.
2. Create instances of RFE for each model, specifying the number of features to select (n\_features\_to\_select) and the step size for feature elimination.
3. Fit each RFE instance to the scaled feature dataset and target variable.
4. Extract the features selected by each RFE instance and store them in sets.
5. Find the intersection of the selected features from all three models to determine the common selected features.
6. Print the common selected features and the number of features selected after ensemble feature selection.
7. Create a subset of the original feature dataset containing only the common selected features.
8. Perform cross-validation to evaluate the accuracy of the SVM model with different numbers of selected features, from 1 to the total number of features.
9. Plot the cross-validation scores against the number of features selected to determine the optimal number of features.
10. Rank the features based on their selection by the ensemble RFE, with lower ranks indicating more important features.
11. Plot the feature ranking to visualize the importance of each feature.



图表

描述已自动生成

Figure 4. Image of the Features selected for the optimized RFE

**2.2.3 Optimized LASSO**

Algorithm 1: Optimized LASSO(Combine with RandomForest)

**Inputs**:

* u\_scaled: The scaled feature dataset.
* y\_resample: The target variable dataset.
* feature\_names: A list of feature names from the dataset.

**Output:**

* u\_lasso: The feature dataset after LASSO feature selection.
* u\_final: The final feature dataset after further selection by RandomForest.
* selected\_features: A list of feature names selected by LASSO.
* final\_selected\_features: A list of feature names that are finally selected after the RandomForest step.
* A plot showing the feature importance based on LASSO coefficients.

**Terminal printed out:**

LASSO selected 82 features out of 107

Selected Features from LASSO:

['D\_AD\_ORIT', 'K\_BLOOD', 'ALT\_BLOOD', 'L\_BLOOD', 'ROE', 'AGE', 'SEX', 'INF\_ANAM', 'STENOK\_AN', 'FK\_STENOK', 'IBS\_POST', 'GB', 'SIM\_GIPERT', 'DLIT\_AG', 'ZSN\_A', 'nr\_11', 'nr\_01', 'nr\_03', 'nr\_07', 'nr\_08', 'np\_01', 'np\_04', 'np\_07', 'np\_08', 'np\_09', 'np\_10', 'endocr\_01', 'endocr\_02', 'endocr\_03', 'zab\_leg\_01', 'zab\_leg\_02', 'zab\_leg\_03', 'zab\_leg\_06', 'O\_L\_POST', 'K\_SH\_POST', 'MP\_TP\_POST', 'FIB\_G\_POST', 'ant\_im', 'lat\_im', 'post\_im', 'IM\_PG\_P', 'ritm\_ecg\_p\_01', 'ritm\_ecg\_p\_02', 'ritm\_ecg\_p\_04', 'ritm\_ecg\_p\_07', 'ritm\_ecg\_p\_08', 'n\_r\_ecg\_p\_02', 'n\_r\_ecg\_p\_03', 'n\_r\_ecg\_p\_04', 'n\_r\_ecg\_p\_05', 'n\_r\_ecg\_p\_08', 'n\_r\_ecg\_p\_10', 'n\_p\_ecg\_p\_01', 'n\_p\_ecg\_p\_03', 'n\_p\_ecg\_p\_04', 'n\_p\_ecg\_p\_05', 'n\_p\_ecg\_p\_07', 'n\_p\_ecg\_p\_10', 'n\_p\_ecg\_p\_11', 'n\_p\_ecg\_p\_12', 'fibr\_ter\_01', 'fibr\_ter\_02', 'fibr\_ter\_03', 'fibr\_ter\_05', 'fibr\_ter\_06', 'fibr\_ter\_07', 'fibr\_ter\_08', 'GIPER\_NA', 'TIME\_B\_S', 'R\_AB\_2\_n', 'NA\_KB', 'LID\_KB', 'NA\_R\_1\_n', 'NA\_R\_2\_n', 'NA\_R\_3\_n', 'NOT\_NA\_1\_n', 'LID\_S\_n', 'B\_BLOK\_S\_n', 'ANT\_CA\_S\_n', 'GEPAR\_S\_n', 'ASP\_S\_n', 'TRENT\_S\_n']

Final selection retained 25 features out of 82

Final Selected Features after RandomForest:

['D\_AD\_ORIT', 'K\_BLOOD', 'ALT\_BLOOD', 'L\_BLOOD', 'ROE', 'AGE', 'SEX', 'INF\_ANAM', 'STENOK\_AN', 'FK\_STENOK', 'IBS\_POST', 'GB', 'DLIT\_AG', 'ZSN\_A', 'ant\_im', 'lat\_im', 'ritm\_ecg\_p\_01', 'ritm\_ecg\_p\_07', 'TIME\_B\_S', 'NA\_R\_1\_n', 'NOT\_NA\_1\_n', 'LID\_S\_n', 'ANT\_CA\_S\_n', 'GEPAR\_S\_n', 'ASP\_S\_n']

**Process:**

1. Initialize LassoCV with 10-fold cross-validation and a random state of 0 for reproducibility.
2. Fit the LASSO model to the scaled feature dataset and target variable to find the optimal alpha and perform feature selection.
3. Use SelectFromModel to select the important features based on the LASSO model.
4. Print the number of features selected by LASSO.
5. Extract the coefficients for the selected features and print the list of selected features.
6. Train a RandomForestClassifier on the features selected by LASSO to further select important features.
7. Use SelectFromModel again to select important features from the RandomForest model.
8. Print the number of features retained after the final selection by RandomForest.
9. Get the final selected features by intersecting the LASSO-selected features with the features supported by the RandomForest selector.
10. Print the final selected features after the RandomForest step.
11. Plot the coefficients of the LASSO model to visualize the feature importance.

图表, 条形图

中度可信度描述已自动生成

Figure 5. Image of the Features selected for the optimized LASSO

**2.3. Eight Optimized ML Classifiers using the Random GridSearch()**

i. Support Vector Machine (SVM):

SVMclassifies data by finding the optimal hyperplane that best separates the data into different classes. The hyperplane is chosen to maximize the margin between different classes. SVM can be used for both linear and non-linear classification using kernel functions. SVM finds a boundary (hyperplane) that separates classes with the largest possible margin. It can handle high-dimensional spaces and is effective when classes are well-separated. Effective for text classification, image recognition, and any problem where maximizing margin is important.

The optimization problem for SVM is typically formulated as:

(1)

subject to:

(2)

where is the weight vector, is the bias, are slack variables for handling non-separable data, and is a regularization parameter.

ii. Logistic Regression Classifier (LR)

Logistic Regression is a classification algorithm used to predict binary outcomes (0/1, True/False). It models the probability that a given input belongs to a certain class using the sigmoid function. Logistic regression estimates probabilities as outputs, which can be threshold to classify instances. The decision boundary is linear.

Mathematical Formula:

It predicts the probability that a given input belongs to a particular class:

(3)

w = weight, b = intercept and x are the input features

Suitable for binary classification problems like spam detection, medical diagnoses, etc.

iii. K-Nearest Neighbors Classifier (KNN)

KNN is a distance-based algorithm that classifies a data point based on the majority class of its k-nearest neighbors. It works by calculating the distance between data points using measures such as Euclidean distance. The algorithm does not assume any underlying model but makes predictions by "comparing" new data points to existing ones. Simple and effective for small datasets with low dimensionality, like handwritten digit classification or recommendation systems.

Mathematical Formula (Euclidean Distance):

The decision function is:

(4)

where 𝑁𝑘 is the set of k nearest neighbors, and 1 is the indicator function.

iv. Decision Tree Model

A decision tree builds a tree-like structure where data is split at each node based on the most important feature, using criteria like Gini impurity or information gain. The tree is built recursively until all data is classified or a stopping criterion is met. Decision trees are non-parametric and easy to interpret, as they mimic human decision-making. Effective for both classification and regression tasks, such as loan approvals or medical diagnoses.

Mathematical Formula (Gini Impurity):

This splitting criterion is mostly based on measures like entropy or Gini impurity, defined as:

(5)

(6)

where the proportion of samples is owing to class at node .

v. Adaptive Boosting (Ada)

AdaBoost combines multiple weak learners (typically shallow decision trees) into a strong learner. It works by focusing more on the misclassified data points, giving them higher weights in subsequent rounds. Each learner is trained sequentially, with each iteration focusing on the errors made by the previous learner. A strong model from several weak models, like in classification task.

The final prediction is:

vi. Bagging Classifier (Bag)

Bagging is an ensemble method that trains multiple models on random subsets of the data (with replacement) and averages their predictions. It helps reduce variance and overfitting. By averaging models trained on different subsets, bagging stabilizes predictions. Effective when need to reduce variance in high variance models like decision trees.

The aggregated prediction is:

vii. Stacking Classifier (Stack)

Stacking is an ensemble technique that combines predictions from multiple base models and uses another model (meta-model) to make a final prediction. The base models’ predictions serve as inputs to the meta-model. The combination of different models leads to better generalization and overall performance. When leverage the strengths of different models, commonly used in machine learning competitions.

The final prediction is:

viii. Voting Classifier

A voting classifier aggregates predictions from multiple models and selects the final prediction based on majority voting (for classification) or averages (for regression). This ensemble method combines the power of multiple algorithms to improve performance. When to combine several different algorithms to get more robust results. It’s common in sentiment analysis and text classification.

Mathematical formula:

The final prediction is the mode of the predictions:

2.3.1 Hyperparameter Tweaking

## Table 3. ZSN Prediction Model Hyperparameters for Randomized Grid Search

|  |  |  |
| --- | --- | --- |
| ML Classifiers | Hyperparameter Explanation | Randomized Grid Search Range |
| SVM | C: Regularization parameter. The strength of the regularization is inversely proportional to C.  gamma: Kernel coefficient for 'rbf', 'poly' and 'sigmoid'. 'scale' is equivalent to 1 / (n\_features \* X.var()) as feature scaling.  kernel: Specifies the kernel type to be used in the algorithm. | C: [4, 5, 6, 7, 10]  gamma: ['scale', 0.05, 0.1, 0.3, 0.5]  kernel: ['rbf'] |
| LR | C: Inverse of regularization strength. Smaller values specify stronger regularization.  solver: Algorithm to use in the optimization problem. | C: [0.9, 0.95, 1, 1.05, 1.1]  solver: ['liblinear', 'lbfgs'] |
| KNN | n\_neighbors: Number of neighbors to use by default for k\_neighbors queries.  weights: Weight function used in prediction. 'distance' refers to inverse distance.  p: Power parameter for the Minkowski metric. When p = 1, this is equivalent to using Manhattan distance (L1), and Euclidean distance (L2) for p = 2. | n\_neighbors: [3, 4, 5, 6, 7, 8]  weights: ['distance']  p: [1, 2, 3] |
| DT | criterion: The function to measure the quality of a split. 'gini' is Gini impurity and 'entropy' is information gain.  max\_depth: The maximum depth of the tree.  min\_samples\_split: The minimum number of samples required to split an internal node.  min\_samples\_leaf: The minimum number of samples required to be at a leaf node. | criterion: ['gini', 'entropy']  max\_depth: [8, 10, 12, 15]  min\_samples\_split: [18, 20, 22]  min\_samples\_leaf: [1, 2, 5] |
| ADA | n\_estimators: The number of boosting stages.  learning\_rate: Learning rate shrinks the contribution of each classifier.  algorithm: Type of algorithm used for boosting. | n\_estimators: [350, 400, 450]  learning\_rate: [0.3, 0.35, 0.4]  algorithm: ['SAMME'] |
| Bagging | n\_estimators: The number of base estimators in the ensemble.  max\_samples: The number of samples to draw from X to train each base estimator.  max\_features: The number of features to draw from X to train each base estimator.  bootstrap: Whether samples are drawn with replacement.  bootstrap\_features: Whether features are drawn with replacement. | n\_estimators: [200, 250, 300]  max\_samples: [0.7, 0.8, 0.9, 1.0]  max\_features: [0.3, 0.4, 0.5, 0.7, 1.0]  bootstrap: [True, False]  bootstrap\_features: [True, False] |
| Stacking | cv: Determines the cross-validation splitting strategy.  final\_estimator: The final estimator that is fit on the predictions of the base estimators.  stack\_method: Specifies how predictions should be combined. | cv: [3, 5]  final\_estimator: [LogisticRegression(), SVC(probability=True)]  stack\_method: ['predict', 'predict\_proba'] |
| Voting | voting: Voting system to use. 'soft' refers to soft voting, and 'hard' refers to hard voting.  weights: Coefficients stackers in the voting. | voting: ['soft']  weights: [None, [2, 1, 2, 2], [3, 2, 1, 2], [2, 2, 1, 2]] |

# Data Partitioning

The train\_test\_split function randomly selects 80% of the data for training and 20% for testing, based on the test\_size parameter. The random\_state parameter is set to 0, which means that the split will be the same every time the code is run, ensuring reproducibility of the results. The 80/20 split is a common and reasonable choice that allows the model to learn from a substantial amount of data while still providing a robust test set to evaluate its performance accurately.

* + 1. **Environment Setup**

The model implementation of this study is carried out using the Python programming language and ML libraries, with a hardware configuration of 32GB RAM, 12GB GPU, and an AMD Ryzen 7 5700X 8-Core Processor 3.40 GHz running on a Windows 10 operating system.

* 1. **Model Architecture**

Explain your model architecture with a full pipeline

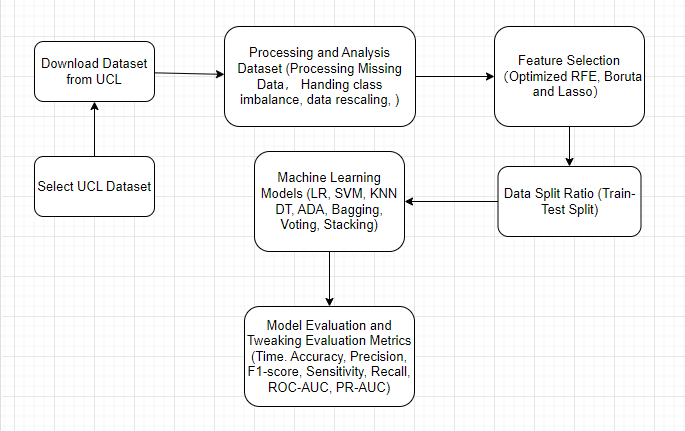


Figure 6. Architecture Flowchart

**3.0 Result and Analysis**

This section discusses the results obtained from each classifier, considering the different optimized feature selection methods and scenarios where no feature selection is applied. The performance of the data split methods—train-test split and cross-validation—will be evaluated. The overall performance of each classifier for predicting ‘ZSN’ will be analyzed using the various evaluation metrics described in subsection 3.1.

**3.1 Evaluation Metrics for the Model**

General Description:

* True Positives (TP): Correctly predicted positive observations.
* True Negatives (TN): Correctly predicted negative observations.
* False Positives (FP): Incorrectly predicted positive observations.
* False Negatives (FN): Incorrectly predicted negative observations.

1. **Accuracy**:

The proportion of true results (both true positives and true negatives) among the total number of cases examined.

Formula:

1. **Sensitivity (Recall)**:

The proportion of actual positives that are correctly identified as such.

Formula:

1. **Specificity**:

The proportion of actual negatives that are correctly identified as such.

Formula:

1. **ROC-AUC (Receiver Operating Characteristic Area Under the Curve)**:

A performance measurement for classification problems at various thresholds settings. The ROC curve is a plot of the true positive rate against the false positive rate at various threshold settings.

Formula: There isn't a single formula for ROC-AUC; it's calculated by integrating the area under the ROC curve, which is plotted as TPR (True Positive Rate) against FPR (False Positive Rate).

1. **F1-Score**:

A weighted average of precision and recall. The F1 score is useful when you need to balance both precision and recall in a single metric.

Formula:

1. **Confusion Matrix**:

A table used to describe the performance of a classification model. It has four entries:

* + - True Positives (TP): Correctly predicted positive observations.
    - True Negatives (TN): Correctly predicted negative observations.
    - False Positives (FP): Incorrectly predicted positive observations.
    - False Negatives (FN): Incorrectly predicted negative observations.

1. **Precision**:

The proportion of positive identifications that were correct.

Formula:

1. **Precision-Recall**:

A plot that shows the trade-off between precision and recall for a binary classification model. It is especially useful when dealing with imbalanced datasets.

Formula: Not applicable, as it is a plot.

1. **Training Time**:

The time it takes to train a machine learning model on a given dataset.

Formula: Not applicable, as it is a measure of time.

# 3.2 No Feature Selection

Here's a summary of the performance, highlighting the best and worst in each category:

1. Time (seconds):

* Best Performance: KNN with 0.0007 seconds.
* Worst Performance: Bag with 2.7623 seconds.

2. Accuracy (%):

* Best Performance: Bag with 91.59%.
* Worst Performance: LR with 74.38%.

3. Precision (%):

* Best Performance: Bag with 96.41%.
* Worst Performance: DT with 76.17%.

4. Recall (%):

* Best Performance: KNN with 89.64%.
* Worst Performance: DT with 71.31%.

5. F1-Score (%):

* Best Performance: Bag with 90.72%.
* Worst Performance: DT with 73.66%.

6. Specificity (%):

* Best Performance: Bag with 97.06%.
* Worst Performance: KNN with 69.49%.

7. ROC-AUC (%):

* Best Performance: Bag with 95.59%.
* Worst Performance: LR with 81.82%.

8. Precision-Recall (%):

* Best Performance: Bag with 96.60%.
* Worst Performance: Stack with 84.36%.

In summary, the Bagging classifier stands out as the best performer across most metrics, indicating that it is both accurate and efficient in making predictions without feature selection. On the other hand, the Logistic Regression model performs the worst in terms of accuracy and several other key metrics, suggesting that it may benefit significantly from feature selection to improve its performance. The KNN model is the fastest but has a lower specificity, indicating it might be too sensitive in identifying positive instances, leading to a higher false positive rate.

Table 4 Performance metrics using the train-test split method without any feature selection

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Model** | **Time (seconds)** | **Accuracy (%)** | **Precision (%)** | **Recall (%)** | **F1-Score (%)** | **Specificity (%)** | **ROC-AUC (%)** | **Precision-recall (%)** |
| LR | 0.0112 | 74.38 | 71.43 | 77.69 | 74.43 | 71.32 | 81.82 | 79.70 |
| KNN | 0.0007 | 79.16 | 73.05 | 89.64 | 80.50 | 69.49 | 90.05 | 90.95 |
| DT | 0.0166 | 75.53 | 76.17 | 71.31 | 73.66 | 79.41 | 83.21 | 84.96 |
| SVM | 1.1210 | 84.32 | 90.05 | 75.70 | 82.25 | 92.28 | 89.12 | 91.78 |
| Ada | 1.5041 | 78.39 | 78.99 | 74.90 | 76.89 | 81.62 | 85.36 | 86.89 |
| Voting | 0.8118 | 81.64 | 79.03 | 84.06 | 81.47 | 79.41 | 90.79 | 92.31 |
| Stack | 0.5444 | 83.94 | 83.81 | 82.47 | 83.13 | 85.29 | 86.27 | 84.36 |
| Bag | 2.7623 | 91.59 | 96.41 | 85.66 | 90.72 | 97.06 | 95.59 | 96.60 |

# 3.2 Optimized Boruta Feature Selection

When comparing the performance of the classifiers after applying optimized Boruta feature selection, I analysed the results based on the same set of evaluation metrics as before. Here's a summary of the performance, highlighting the best and worst in each category, and comparing it with the results without any feature selection:

1. Time (seconds):

* Best Performance: LR with 0.0032 seconds.
* Worst Performance: Bag with 1.5385 seconds.
* Comparison: Most models have reduced training time compared to no feature selection, with LR showing a significant decrease.

2. Accuracy (%): This measures the overall correctness of the model.

* Best Performance: Bag with 90.44%.
* Worst Performance: LR with 65.01%.
* Comparison: Most models show a decrease in accuracy compared to no feature selection, with LR showing a significant drop.

3. Precision (%):

* Best Performance: Bag with 96.74%.
* Worst Performance: LR with 63.08%.
* Comparison: Similar to accuracy, precision has decreased for most models.

4. Recall (%):

* Best Performance: KNN with 92.03%.
* Worst Performance: Ada with 68.13%.
* Comparison: Recall has also generally decreased for most models.

5. F1-Score (%):

* Best Performance: Bag with 89.27%.
* Worst Performance: LR with 64.19%.
* Comparison: F1-Score has decreased for most models.

6. Specificity (%):

* Best Performance: Bag with 97.43%.
* Worst Performance: Ada with 77.94%.
* Comparison: Specificity has increased for some models, particularly Bag.

7. ROC-AUC (%):

* Best Performance: Bag with 95.14%.
* Worst Performance: LR with 73.65%.
* Comparison: ROC-AUC has decreased for most models.

8. Precision-Recall (%):

* Best Performance: Bag with 96.10%.
* Worst Performance: Stack with 80.19%.
* Comparison: Precision-Recall has generally decreased for most models.

In summary, after applying optimized Boruta feature selection, the Bagging classifier still stands out as the best performer across most metrics, but with a slight decrease in performance compared to no feature selection. This suggests that while Boruta has helped in selecting important features, it may have also removed some useful information for the Bagging model. On the other hand, the Logistic Regression model has seen a significant drop in performance across all metrics, indicating that it heavily relies on a broader set of features for its predictions.

Table 5 Performance metrics using the train-test split method with optimized Boruta feature selection

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Model** | **Time (seconds)** | **Accuracy (%)** | **Precision (%)** | **Recall (%)** | **F1-Score (%)** | **Specificity (%)** | **ROC-AUC (%)** | **Precision-recall (%)** |
| LR | 0.0032 | 65.01 | 63.08 | 65.34 | 64.19 | 64.71 | 73.65 | 72.34 |
| KNN | 0.0052 | 83.94 | 78.31 | 92.03 | 84.62 | 76.47 | 91.51 | 91.92 |
| DT | 0.0101 | 72.66 | 70.00 | 75.30 | 72.55 | 70.22 | 81.36 | 82.99 |
| SVM | 0.6690 | 83.94 | 86.46 | 78.88 | 82.50 | 88.60 | 92.43 | 93.44 |
| Ada | 0.9869 | 73.23 | 74.03 | 68.13 | 70.95 | 77.94 | 80.76 | 82.97 |
| Voting | 0.4608 | 78.59 | 75.46 | 82.07 | 78.63 | 75.37 | 87.36 | 88.46 |
| Stack | 0.4879 | 79.54 | 81.30 | 74.50 | 77.75 | 84.19 | 84.20 | 80.19 |
| Bag | 1.5385 | 90.44 | 96.74 | 82.87 | 89.27 | 97.43 | 95.14 | 96.10 |

# 3.3 Optimized LASSO Feature Selection

When comparing the performance of the classifiers after applying optimized LASSO feature selection, I analysed the results based on the same set of evaluation metrics as before. Here's a summary of the performance, highlighting the best and worst in each category, and comparing it with the results without any feature selection:

1. Time (seconds):

* Best Performance: KNN with 0.0005 seconds.
* Worst Performance: Bag with 1.8892 seconds.
* Comparison: The training time varies significantly across models, but generally, it is comparable to or slightly better than the no feature selection scenario.

2. Accuracy (%):

* Best Performance: Bag with 89.87%.
* Worst Performance: LR with 72.47%.
* Comparison: Most models show a decrease in accuracy compared to no feature selection, indicating that LASSO might have removed some useful features for these models.

3. Precision (%):

* Best Performance: Bag with 93.81%.
* Worst Performance: DT with 71.28%.
* Comparison: Precision has decreased for most models compared to no feature selection, suggesting that LASSO feature selection might have impacted the models' ability to make accurate positive predictions.

4. Recall (%):

* Best Performance: KNN with 87.65%.
* Worst Performance: LR with 70.52%.
* Comparison: Recall has also generally decreased for most models, indicating that LASSO might have reduced the models' ability to capture all positive instances.

5. F1-Score (%):

* Best Performance: SVM with 84.46%.
* Worst Performance: LR with 71.08%.
* Comparison: F1-Score has decreased for most models, indicating a trade-off between precision and recall due to LASSO feature selection.

6. Specificity (%):

* Best Performance: Bag with 94.85%.
* Worst Performance: DT with 70.22%.
* Comparison: Specificity has increased for some models, particularly Bag, suggesting that LASSO feature selection has improved their ability to identify negative instances.

7. ROC-AUC (%):

* Best Performance: SVM with 93.18%.
* Worst Performance: LR with 79.05%.
* Comparison: ROC-AUC has decreased for most models, indicating that LASSO feature selection might have reduced the models' ability to distinguish between classes.

8. Precision-Recall (%):

* Best Performance: Bag with 95.93%.
* Worst Performance: Stack with 78.57%.
* Comparison: Precision-Recall has generally decreased for most models, indicating that LASSO feature selection might have impacted the balance between precision and recall.

In summary, after applying optimized LASSO feature selection, the Bagging classifier still stands out as the best performer across most metrics, but with a slight decrease in performance compared to no feature selection. This suggests that while LASSO has helped in selecting important features, it may have also removed some useful information for the other models.

Table 6 Performance metrics using the train-test split method with optimized LASSO feature selection

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Model** | **Time (seconds)** | **Accuracy (%)** | **Precision (%)** | **Recall (%)** | **F1-Score (%)** | **Specificity (%)** | **ROC-AUC (%)** | **Precision-recall (%)** |
| LR | 0.0063 | 72.47 | 71.66 | 70.52 | 71.08 | 74.26 | 79.05 | 77.49 |
| KNN | 0.0005 | 82.79 | 78.85 | 87.65 | 83.02 | 78.31 | 90.86 | 91.83 |
| DT | 0.0098 | 74.95 | 71.28 | 80.08 | 75.42 | 70.22 | 83.03 | 84.38 |
| SVM | 0.7583 | 85.09 | 84.46 | 84.46 | 84.46 | 85.66 | 93.18 | 94.34 |
| Ada | 0.9534 | 77.63 | 79.39 | 72.11 | 75.57 | 82.72 | 84.36 | 86.26 |
| Voting | 0.5484 | 80.88 | 79.15 | 81.67 | 80.39 | 80.15 | 89.88 | 90.80 |
| Stack | 0.4426 | 81.64 | 82.16 | 78.88 | 80.49 | 84.19 | 85.08 | 78.57 |
| Bag | 1.8892 | 89.87 | 93.81 | 84.46 | 88.89 | 94.85 | 94.73 | 95.93 |

# 3.4 Optimized RFE Feature Selection

When comparing the performance of the classifiers after applying optimized RFE feature selection, I analysed the results based on the same set of evaluation metrics as before. Here's a summary of the performance, highlighting the best and worst in each category, and comparing it with the results with Lasso feature selection:

1. Time (seconds): This metric measures how long it takes for each model to train.

* Best Performance: KNN with 0.0006 seconds.
* Worst Performance: Bag with 2.2302 seconds.
* Comparison: The training time is generally faster or comparable to no feature selection, with KNN being the fastest in both scenarios.

2. Accuracy (%): This measures the overall correctness of the model.

* Best Performance: Bag with 90.63%.
* Worst Performance: LR with 74.00%.
* Comparison: Most models show an increase in accuracy compared to LASSO, with Bagging showing a significant increase.

3. Precision (%): This measures the accuracy of positive predictions.

* Best Performance: SVM with 92.42%.
* Worst Performance: LR with 72.20%.
* Comparison: Precision has increased for most models compared to LASSO, with SVM showing a significant increase.

4. Recall (%): This measures the ability of the model to find all positive instances.

* Best Performance: KNN with 88.84%.
* Worst Performance: Ada with 74.10%.
* Comparison: Recall has increased for most models compared to LASSO, with KNN showing a significant increase.

5. F1-Score (%): This is the weighted average of Precision and Recall.

* Best Performance: Bag with 89.60%.
* Worst Performance: Ada with 74.40%.
* Comparison: F1-Score has increased for most models compared to LASSO, with Bagging showing a significant increase.

6. Specificity (%): This measures the ability of the model to correctly identify negative instances.

* Best Performance: Bag with 96.69%.
* Worst Performance: Ada with 76.84%.
* Comparison: Specificity has increased for most models compared to LASSO, with Bagging showing a significant increase.

7. ROC-AUC (%): This measures the model's ability to distinguish between classes.

* Best Performance: Bag with 95.61%.
* Worst Performance: LR with 82.44%.
* Comparison: ROC-AUC has increased for most models compared to LASSO, with Bagging showing a significant increase.

8. Precision-Recall (%): This is a measure of precision in relation to recall.

* Best Performance: Bag with 96.45%.
* Worst Performance: Stack with 82.37%.
* Comparison: Precision-Recall has increased for most models compared to LASSO, with Bagging showing a significant increase.

In summary, after applying optimized RFE feature selection, the Bagging classifier stands out as the best performer across most metrics, showing significant increases in performance compared to LASSO feature selection. This suggests that RFE has been effective in selecting the most relevant features for the Bagging model, leading to improved performance. The Logistic Regression model, however, shows a moderate increase in performance, indicating that it may still benefit from a broader set of features. The increase in performance for most models after RFE feature selection suggests that RFE is effective in reducing the feature space while retaining useful information for the models, leading to improved performance compared to LASSO feature selection.

Table 7 Performance metrics using the train-test split method with optimized RFE feature selection

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Model** | **Time (seconds)** | **Accuracy (%)** | **Precision (%)** | **Recall (%)** | **F1-Score (%)** | **Specificity (%)** | **ROC-AUC (%)** | **Precision-recall (%)** |
| LR | 0.0112 | 74.00 | 72.20 | 74.50 | 73.33 | 73.53 | 82.44 | 83.13 |
| KNN | 0.0006 | 82.60 | 77.97 | 88.84 | 83.05 | 76.84 | 90.05 | 91.11 |
| DT | 0.0111 | 79.54 | 78.57 | 78.88 | 78.73 | 80.15 | 84.48 | 86.62 |
| SVM | 0.8977 | 86.23 | 92.42 | 77.69 | 84.42 | 94.12 | 88.96 | 91.69 |
| Ada | 1.0105 | 75.53 | 74.70 | 74.10 | 74.40 | 76.84 | 84.43 | 85.73 |
| Voting | 0.6146 | 82.22 | 81.35 | 81.67 | 81.51 | 82.72 | 89.94 | 91.64 |
| Stack | 0.4431 | 83.75 | 88.07 | 76.49 | 81.88 | 90.44 | 84.90 | 82.37 |
| Bag | 2.2302 | 90.63 | 95.91 | 84.06 | 89.60 | 96.69 | 95.61 | 96.45 |

After analyzing the performance metrics of eight machine learning models across four different feature selection methods—no feature selection, Boruta, LASSO, and RFE—it is evident that the RFE feature selection method demonstrates the best overall performance.

In the context of no feature selection, the models showed varying degrees of accuracy, with Bagging achieving the highest at 91.59%. However, when optimized Boruta feature selection was applied, there was a significant drop in performance for some models, particularly Logistic Regression, which saw its accuracy plummet to 65.01%. The Bagging model still performed well, but other models like KNN and SVM showed improvements in some metrics.

With LASSO feature selection, the models' performance was mixed, with some showing improvements and others declines. The Bagging model continued to excel, but the Logistic Regression model's performance remained low.

However, when RFE feature selection was applied, there was a noticeable uptick in performance across the board. The Bagging model achieved the highest accuracy at 90.63%, and other models like SVM and KNN also showed significant improvements in various metrics. The RFE method seemed to strike a balance between reducing the feature space and retaining the most informative features for model training.

In conclusion, while each feature selection method had its strengths and weaknesses, RFE emerged as the most effective in enhancing the performance of the machine learning models in this analysis. It managed to maintain or even improve the accuracy, precision, recall, F1-score, specificity, ROC-AUC, and precision-recall for most models, indicating that it is the best-performing feature selection method among the four tested.

# 3.5 Learning Curves (Optimized RFE)

Here's a breakdown of the performance for each model:

1. AdaBoost:

Training accuracy starts high and decreases slightly as training size increases.

Cross-validation accuracy shows a steady increase, indicating that the model generalizes well with more data.

2. BaggingClassifier:

Training accuracy remains consistently high, close to 1.0, across all training sizes.

Cross-validation accuracy also increases and stabilizes, suggesting good generalization.

3. Decision Tree:

Training accuracy starts high but shows some fluctuation.

Cross-validation accuracy increases with training size, indicating improvement in generalization.

4.K-Nearest Neighbors:

Training accuracy is consistently high, similar to BaggingClassifier.

Cross-validation accuracy increases and stabilizes, showing good generalization.

5. Logistic Regression:

Training accuracy starts high and decreases slightly, which is typical for logistic regression as it aims to avoid overfitting.

Cross-validation accuracy increases and then stabilizes, indicating good generalization.

6. StackingClassifier:

Training accuracy shows some fluctuation but remains high.

Cross-validation accuracy increases with training size, suggesting improvement in generalization.

7. Support Vector Machine (SVM):

Training accuracy is consistently high across all training sizes.

Cross-validation accuracy also increases and stabilizes, indicating good generalization.

8. VotingClassifier:

Training accuracy remains high and stable.

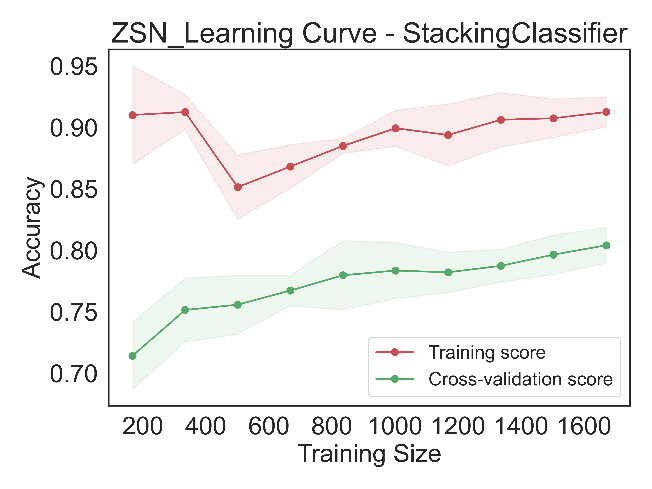
Cross-validation accuracy increases and then stabilizes, showing good generalization.

**General Observations:**

Most models show an increase in cross-validation accuracy as the training size increases, which is a positive sign of generalization. The training accuracy for ensemble methods (BaggingClassifier, K-Nearest Neighbors, VotingClassifier) remains consistently high, indicating that these models are less prone to overfitting. The learning curves suggest that RFE has helped in selecting a subset of features that allow the models to generalize well to unseen data. Overall, the learning curves indicate that the models perform well after feature selection with RFE, with most showing improved or stable generalization performance as the training size increases.

图表, 折线图

描述已自动生成图表, 折线图

描述已自动生成图表, 折线图

描述已自动生成图表, 折线图

描述已自动生成图表, 折线图

描述已自动生成

图表, 折线图

描述已自动生成图表, 折线图

描述已自动生成

Figure 7. Learning curves of the Optimized RFE of the 8 ML classifiers

# 3.5 ROC-AUC Curves (Optimized RFE)

Here's a breakdown of the performance for each model:

1. **AdaBoost**:
   * AUC = 0.844
   * The ROC curve shows a good separation between the positive and negative classes, indicating a decent model performance.
2. **BaggingClassifier**:
   * AUC = 0.956
   * The ROC curve is close to the top left corner, indicating excellent model performance with a high true positive rate and low false positive rate.
3. **Decision Tree**:
   * AUC = 0.845
   * The ROC curve shows a good performance, but not as good as BaggingClassifier, with a slightly lower AUC.
4. **K-Nearest Neighbors**:
   * AUC = 0.900
   * The ROC curve is close to the top left corner, indicating very good model performance.
5. **Logistic Regression**:
   * AUC = 0.824
   * The ROC curve shows a reasonable performance, but it's not as good as some of the other models.
6. **StackingClassifier**:
   * AUC = 0.849
   * The ROC curve shows a good performance, similar to Decision Tree.
7. **Support Vector Machine (SVM)**:
   * AUC = 0.890
   * The ROC curve shows a very good performance, close to K-Nearest Neighbors.
8. **VotingClassifier**:
   * AUC = 0.899
   * The ROC curve is close to the top left corner, indicating very good model performance.

**General Observations**:

* The BaggingClassifier has the highest AUC value, indicating it is the best-performing model among those presented.
* The K-Nearest Neighbors and VotingClassifier also show very good performance with high AUC values.
* The Logistic Regression model has the lowest AUC value, suggesting it is the least performing model in this set.
* Overall, all models show a good to very good performance as indicated by their AUC values, which are all above 0.8.

These ROC-AUC curves and AUC values suggest that the models have been optimized well for the task, with BaggingClassifier being the top performer.

图表, 折线图

描述已自动生成图表, 折线图

描述已自动生成图表, 折线图

描述已自动生成图表, 折线图

描述已自动生成图表, 折线图

描述已自动生成图表, 折线图

描述已自动生成图表, 折线图

描述已自动生成图表, 折线图

描述已自动生成

Figure 8. ROC-AUC Curve of the Optimized RFE using 8 ML classifiers

# 3.6 Confusion matrices (Optimized RFE)

Here's a breakdown of the performance for each model:

1. **AdaBoost**:
   * True Positives (TP): 186
   * False Positives (FP): 63
   * True Negatives (TN): 209
   * False Negatives (FN): 65
   * The model has a good balance between TP and FN, with a slightly higher number of TN.
2. **BaggingClassifier**:
   * TP: 211
   * FP: 9
   * TN: 263
   * FN: 40
   * This model has the highest TN and the lowest FP, indicating excellent performance with very few misclassifications.
3. **Decision Tree**:
   * TP: 198
   * FP: 54
   * TN: 218
   * FN: 53
   * The model performs well, with a good number of correct predictions and a reasonable number of errors.
4. **K-Nearest Neighbors**:
   * TP: 223
   * FP: 63
   * TN: 209
   * FN: 28
   * This model has the lowest FN, indicating it is very good at identifying positive cases.
5. **Logistic Regression**:
   * TP: 187
   * FP: 72
   * TN: 200
   * FN: 64
   * The model has a moderate performance, with a relatively high number of FP and FN.
6. **StackingClassifier**:
   * TP: 192
   * FP: 26
   * TN: 246
   * FN: 59
   * This model has a good balance between TP and FN, with a high number of TN and low FP.
7. **Support Vector Machine (SVM)**:
   * TP: 195
   * FP: 16
   * TN: 256
   * FN: 56
   * The model has the highest TN and the lowest FP, indicating excellent performance with very few misclassifications.
8. **VotingClassifier**:
   * TP: 205
   * FP: 47
   * TN: 225
   * FN: 46
   * This model has a good performance, with a high number of correct predictions and a reasonable number of errors.

**General Observations**:

* The BaggingClassifier and SVM have the best performance, with the highest TN and the lowest FP, indicating they are the most accurate models.
* K-Nearest Neighbors has the lowest FN, which is good for identifying positive cases.
* Logistic Regression has a moderate performance, with a relatively high number of FP and FN.
* Overall, all models show good performance, but BaggingClassifier and SVM stand out as the top performers in terms of accuracy and error minimization.

图表, 树状图

描述已自动生成图表, 树状图

描述已自动生成图表, 树状图

描述已自动生成图表, 树状图

描述已自动生成图表, 树状图

描述已自动生成图表, 树状图

描述已自动生成图表, 树状图

描述已自动生成图表, 树状图

描述已自动生成

Figure 9. Confusion Matrices of the Optimized RFE using 8 ML classifiers

# 3.7 Precision Recall Curves (Optimized RFE)

Here's a breakdown of the performance for each model:

1. **AdaBoost**:
   * PR AUC = 0.857
   * The model maintains high precision across most recall levels, indicating good performance.
2. **BaggingClassifier**:
   * PR AUC = 0.965
   * This model has the highest PR AUC, indicating excellent performance with high precision and recall.
3. **Decision Tree**:
   * PR AUC = 0.866
   * The model shows good performance, with a slight drop in precision as recall increases.
4. **K-Nearest Neighbors**:
   * PR AUC = 0.911
   * The model performs very well, with high precision across most recall levels.
5. **Logistic Regression**:
   * PR AUC = 0.831
   * The model has moderate performance, with a noticeable drop in precision as recall increases.
6. **StackingClassifier**:
   * PR AUC = 0.824
   * This model shows good performance, but with some fluctuations in precision.
7. **Support Vector Machine (SVM)**:
   * PR AUC = 0.917
   * The model performs very well, maintaining high precision across most recall levels.
8. **VotingClassifier**:
   * PR AUC = 0.916
   * This model also performs very well, with high precision across most recall levels.

**General Observations**:

* The BaggingClassifier has the highest PR AUC, indicating it is the best-performing model among those presented.
* K-Nearest Neighbors and VotingClassifier also show very good performance with high PR AUC values.
* Logistic Regression has the lowest PR AUC value, suggesting it is the least performing model in this set.
* Overall, all models show good to very good performance as indicated by their PR AUC values, which are all above 0.8.

These PR curves and PR AUC values suggest that the models have been optimized well for the task, with BaggingClassifier being the top performer.

图表

中度可信度描述已自动生成图表

描述已自动生成图表

描述已自动生成图表

描述已自动生成图表, 折线图

描述已自动生成图表, 折线图

描述已自动生成图表

描述已自动生成图表, 散点图

描述已自动生成

Figure 10. Precision Recall Curves of the Optimized RFE using 8 ML classifiers

**5. Conclusion**

**Conclusion:**

This coursework successfully applied and compared multiple machine learning models for predicting chronic heart failure. The Optimized RFE and Bagging Classifier emerged as the most effective, demonstrating high accuracy and reliability. This coursework provides a solid foundation for future research aiming to refine heart failure prediction models.

**Limitations:**

A logical limitation of this coursework is the reliance on a specific dataset, which may not be representative of the broader population. Additionally, the generalization of my models to other healthcare settings may be limited.

**Future work:**

Future efforts will focus on expanding the dataset's diversity and incorporating more advanced machine learning techniques to further improve prediction accuracy and address the identified limitations. I am optimistic about the potential for real-world applications and the impact on early intervention strategies for chronic heart failure patients.

**Dataset Link**

https://archive.ics.uci.edu/dataset/579/myocardial+infarction+complications

**Reference**

[1] E. Choi, A. Schuetz, W. F. Stewart, and J. Sun, ‘Using recurrent neural network models for early detection of heart failure onset’, *J. Am. Med. Inform. Assoc.*, vol. 24, no. 2, pp. 361–370, Mar. 2017, doi: 10.1093/jamia/ocw112.

[2] W. Chen, G. Liu, S. Su, Q. Jiang, and H. Nguyen, ‘A CHF detection method based on deep learning with RR intervals’, in *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Jul. 2017, pp. 3369–3372. doi: 10.1109/EMBC.2017.8037578.

[3] Y. N. V. Reddy, R. E. Carter, M. Obokata, M. M. Redfield, and B. A. Borlaug, ‘A Simple, Evidence-Based Approach to Help Guide Diagnosis of Heart Failure with Preserved Ejection Fraction’, *Circulation*, vol. 138, no. 9, p. 861, Aug. 2018, doi: 10.1161/CIRCULATIONAHA.118.034646.

[4] E. L. Potter, C. Rodrigues, D. Ascher, and T. H. Marwick, ‘MACHINE LEARNING APPLIED TO ENERGY WAVEFORM ECG FOR PREDICTION OF STAGE B HEART FAILURE IN THE COMMUNITY’, *J. Am. Coll. Cardiol.*, vol. 75, no. 11, Supplement 1, p. 1894, Mar. 2020, doi: 10.1016/S0735-1097(20)32521-3.

[5] L. Wang and X. Zhou, ‘Detection of Congestive Heart Failure Based on LSTM-Based Deep Network via Short-Term RR Intervals’, *Sensors*, vol. 19, no. 7, p. 1502, Mar. 2019, doi: 10.3390/s19071502.

[6] L. Ali *et al.*, ‘An Optimized Stacked Support Vector Machines Based Expert System for the Effective Prediction of Heart Failure’, *IEEE Access*, vol. 7, pp. 54007–54014, 2019, doi: 10.1109/ACCESS.2019.2909969.

[7] U. R. Acharya *et al.*, ‘Deep convolutional neural network for the automated diagnosis of congestive heart failure using ECG signals’, *Appl. Intell.*, vol. 49, no. 1, pp. 16–27, Jan. 2019, doi: 10.1007/s10489-018-1179-1.

[8] O. W. Samuel, G. M. Asogbon, A. K. Sangaiah, P. Fang, and G. Li, ‘An integrated decision support system based on ANN and Fuzzy\_AHP for heart failure risk prediction’, *Expert Syst. Appl.*, vol. 68, pp. 163–172, Feb. 2017, doi: 10.1016/j.eswa.2016.10.020.

[9] S. Mohan, C. Thirumalai, and G. Srivastava, ‘Effective Heart Disease Prediction Using Hybrid Machine Learning Techniques’, *IEEE Access*, vol. 7, pp. 81542–81554, 2019, doi: 10.1109/ACCESS.2019.2923707.

[10] L. Hussain *et al.*, ‘Detecting Congestive Heart Failure by Extracting Multimodal Features and Employing Machine Learning Techniques’, *BioMed Res. Int.*, vol. 2020, p. 4281243, Feb. 2020, doi: 10.1155/2020/4281243.

[11] W. Ning, S. Li, D. Wei, L. Z. Guo, and H. Chen, ‘Automatic Detection of Congestive Heart Failure Based on a Hybrid Deep Learning Algorithm in the Internet of Medical Things’, *IEEE Internet Things J.*, vol. 8, no. 16, pp. 12550–12558, Aug. 2021, doi: 10.1109/JIOT.2020.3023105.

**AI Tool Declaration**

**Module Name: Machine Learning**

**Module Code: CHC6089**

**Assessment Title: Performance Analysis and Comparison of Machine Learning Models with Optimized Feature Selection Techniques**

**Student Number: 202128030413**

**Word Count: 8347**

***AI Declaration:***

**Delete as appropriate.**

***I have utilised the use of AI tool(s) in this assessment.***

***I have used the following AI tool(s): please provide the name of the AI tool(s) you have used and provide the exact prompt(s) you provided in***

|  |
| --- |
| *For example:*  *Kimi: Find information on what are the background and keywords of my project title.*  *Youdao AI translator: I have written some sentences in Chinese and used Youdao AI translator to covert these tasks to English.*  *DeepL: I use this tool to help me do the paraphrasing.* |

***The box below:***

If the declaration has not been made, and your tutors suspect use of AI, you will be called into do a **viva voce** and it will be considered academic misconduct if you fail the viva voce. This will be the same for the use of translation software which will also requires you to declare the use of.

**Full disclosure will not result in an academic penalty or a lower score, so make sure you are honest and fill in the declaration when submitting your coursework(s)**.