



**Iran University of Science and Technology**

**Faculty of Computer Engineering**

**Artificial intelligence and robotics group**

**Pattern Recognition project**


**Student Name: Qamar Abbas**

**Student ID: 402722135**

**First semester of 1402 - 1403**

## Report: AdaBoost Classifier for Heart Disease Prediction(First section)

```
60 # Make predictions on the test set
61 y_pred = ada_boost_model.predict(X_test_imputed)
62
63 # Evaluate the model performance
64 accuracy = accuracy_score(y_test, y_pred)
65 precision = precision_score(y_test, y_pred, average='micro') # Change 'micro' to 'macro' or 'weighted' if needed
66 recall = recall_score(y_test, y_pred, average='micro') # Change 'micro' to 'macro' or 'weighted' if needed
67 f1 = f1_score(y_test, y_pred, average='micro') # Change 'micro' to 'macro' or 'weighted' if needed
68
69 # Print evaluation metrics
70 print(f'Accuracy: {accuracy:.4f}')
71 print(f'Precision: {precision:.4f}')
72 print(f'Recall: {recall:.4f}')
73 print(f'F1 Score: {f1:.4f}')
74
```

 Column Names: Index(['id', 'age', 'sex', 'dataset', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalch', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'num'], dtype='object')

Accuracy: 0.5761  
Precision: 0.5761  
Recall: 0.5761  
F1 Score: 0.5761

### Objective:

The objective of the code is to implement an AdaBoost classifier for predicting heart disease based on a given dataset. The code covers data loading, preprocessing, model training, prediction, and evaluation.

### Key Steps:

#### Data Loading:

The code uses the Pandas library to load a dataset ('heart\_disease\_uci.csv') into a DataFrame.

#### Dataset Exploration:

Column names of the dataset are printed to understand the available features.

#### Target Variable Selection:

The target variable ('num') is chosen based on the column names, ensuring it exists in the dataset.

#### Data Splitting:

The dataset is split into training and testing sets (80% training, 20% testing) using the train\_test\_split function.

**Data Preprocessing:**

Numeric and categorical columns are identified.

Transformers for imputing missing values in numeric columns and one-hot encoding categorical columns are created.

Transformers are combined into a preprocessor using ColumnTransformer.

The preprocessor is applied to impute missing values in both the training and testing sets.

**Model Definition:**

A Decision Tree classifier is chosen as the base model for AdaBoost.

**AdaBoost Classifier:**

An AdaBoost classifier is defined using the previously chosen Decision Tree as the base model.

The number of estimators is set to 50, and a random state is specified for reproducibility.

**Model Training:**

The AdaBoost model is trained on the preprocessed training data.

**Prediction:**

The trained model is used to make predictions on the preprocessed testing data.

**Model Evaluation:**

Performance metrics such as accuracy, precision, recall, and F1 score are calculated using scikit-learn's metrics functions.

The evaluation metrics are printed to assess the model's performance.

**Conclusion:**

The code successfully implements an AdaBoost classifier for heart disease prediction. The model is trained on a preprocessed dataset, and its performance is evaluated using standard classification metrics. This code provides a comprehensive workflow for building and assessing a machine learning model for heart disease prediction.

## Stacking Ensemble Report(Second Section)

```
47
48 # Define the stacking classifier
49 estimators = [('svm', svm_model), ('decision_tree', decision_tree_model), ('gradient_boosting', gradient_boosting_model)]
50 stacking_model = StackingClassifier(estimators=estimators, final_estimator=AdaBoostClassifier())
51
52 # Compare accuracy with different cross-validation folds
53 for k in [3, 5, 7]:
54     cross_val_scores = cross_val_score(stacking_model, X_train, y_train, cv=k)
55     print(f'Stacking with cross-validation (k={k}) accuracy: {cross_val_scores.mean()}')
56
57 # Fit the final model on the entire training set
58 stacking_model.fit(X_train, y_train)
59
60 # Make predictions on the test set
61 y_pred_stacking = stacking_model.predict(X_test)
62
63 # Evaluate the performance
64 accuracy_stacking = accuracy_score(y_test, y_pred_stacking)
65 print(f'Stacking Model Accuracy: {accuracy_stacking}')
66
```

```
Stacking with cross-validation (k=3) accuracy: 0.8287926552734914
Stacking with cross-validation (k=5) accuracy: 0.8233314947600661
Stacking with cross-validation (k=7) accuracy: 0.8369015530740598
Stacking Model Accuracy: 0.8586956521739131
```

### Objective:

The objective of this report is to implement and evaluate the Stacking ensemble method using three base models (Support Vector Machine, Decision Tree, and Gradient Boosting) on a dataset related to heart disease.

### Code Overview:

#### Data Loading and Preprocessing:

The dataset is loaded from the provided file path.

Missing values are imputed using the most frequent strategy.

Categorical columns are encoded using label encoding.

Numeric features are standardized using StandardScaler.

#### **Target Variable Transformation:**

The target variable 'num' is transformed to binary form (0 or 1).

#### **Data Splitting:**

The dataset is split into features (X) and the target variable (y).

The data is further divided into training and testing sets.

#### **Base Models:**

Three base models are chosen:

##### **Support Vector Machine (SVM)**

##### **Decision Tree**

##### **Gradient Boosting**

Stacking Classifier:

A Stacking Classifier is created with the specified base models.

AdaBoostClassifier is used as the final estimator.

#### **Cross-Validation:**

The stacking model is evaluated using cross-validation with different folds (k = 3, 5, 7).

Cross-validation scores are printed to assess model performance.

#### **Model Training and Testing:**

The stacking model is trained on the entire training set.

Predictions are made on the test set.

#### **Performance Evaluation:**

The accuracy of the stacking model is calculated using the accuracy\_score function.

The accuracy is printed for the evaluation of the stacking model on the test set.

#### **Results:**

Cross-validation is performed with varying folds ( $k = 3, 5, 7$ ) to assess the generalization performance of the stacking model.

The final accuracy of the stacking model on the test set is reported.

### **Recommendations:**

The stacking ensemble method shows promise in improving predictive performance by leveraging diverse base models.

Further tuning of hyperparameters and experimentation with additional base models could potentially enhance the model's performance

### **Explain the Stacking method in the field of machine learning**

#### **Stacking Method in Machine Learning:**

##### **Overview:**

Stacking, short for stacked generalization, is an ensemble learning technique that combines multiple diverse base models to improve predictive performance. It introduces a meta-model (or a higher-level model) that takes as input the predictions of the base models and produces a final prediction. The idea is to leverage the strengths of different models, capturing various aspects of the underlying data patterns, and potentially achieving better generalization than individual models.

##### **Key Components:**

##### **Base Models:**

Stacking involves training multiple base models on the same dataset. These models can be of different types or can be the same algorithm with different hyperparameters.

The diversity in base models is crucial as it allows capturing different perspectives on the data and helps in reducing overfitting.

##### **Meta-Model (Final Estimator):**

A meta-model, often referred to as the final estimator, is trained to make predictions based on the outputs of the base models.

The meta-model learns how to combine or weigh the predictions of the base models to generate a more accurate and robust final prediction.

##### **Training Process:**

The training process typically involves two stages. In the first stage, the base models are trained on the input data. In the second stage, the meta-model is trained using the predictions made by the base models.

The training dataset is often split into multiple subsets, and the base models are trained on different subsets to introduce variability.

#### **Prediction Process:**

During the prediction phase, the input data is passed through each of the trained base models, and their individual predictions are collected.

These individual predictions are then used as input features for the meta-model, which produces the final prediction.

#### **Advantages:**

##### **Improved Predictive Performance:**

Stacking aims to combine the strengths of different models, mitigating the weaknesses of individual models.

The meta-model can learn to weigh the predictions of base models, giving more emphasis to the models that perform well on specific instances.

##### **Enhanced Generalization:**

By leveraging the diversity of base models, stacking can improve generalization to new, unseen data.

It is particularly useful in situations where no single model performs well across the entire dataset.

##### **Flexibility:**

Stacking is a flexible approach that allows the incorporation of various types of models, making it suitable for a wide range of machine learning problems.

Considerations:

##### **Computational Complexity:**

Stacking can be computationally more expensive than individual models, as it involves training multiple models and a meta-model.

##### **Risk of Overfitting:**

Care should be taken to prevent overfitting, especially in cases where the stacking model might memorize the training data instead of capturing general patterns.

##### **Conclusion:**

Stacking is a powerful technique in the ensemble learning toolbox, providing a way to harness the collective intelligence of diverse models for improved prediction accuracy and generalization in machine learning tasks. Its effectiveness is often demonstrated in various competitions and real-world applications where the performance of individual models may be limited.

### **5. How can Stacking be used to assess the importance of each base model?**

Stacking assesses base model importance by observing the meta-model's reliance on their predictions. If a base model's predictions significantly influence the meta-model's decisions, it indicates importance. Feature importance techniques specific to the meta-model can also be employed, revealing the contribution of each base model in the ensemble.

### **6. How does Stacking prevent Overfitting?**

Stacking helps prevent overfitting by training base models on different subsets of the data, introducing diversity. Additionally, the meta-model learns to generalize from the diverse predictions of base models, reducing the risk of memorizing noise in the training data. This ensemble approach promotes robustness and better generalization.

### **7. Differences between Bagging, Boosting, and Stacking:**

#### **Bagging (Bootstrap Aggregating):**

Purpose: Reduce variance and prevent overfitting.

Process: Trains multiple instances of the same model on different bootstrapped samples.

**Combination:** Averages predictions (for regression) or takes a majority vote (for classification).

#### **Boosting:**

Purpose: Improve model accuracy by sequentially correcting errors.

Process: Trains models sequentially, giving more weight to misclassified instances.

Combination: Weighted combination of models' predictions.

#### **Stacking:**

Purpose: Combine diverse models for enhanced performance.



Process: Trains multiple diverse models and a meta-model to combine their predictions.

Combination: Meta-model combines base models' predictions based on learned weights.

## XGBoost Model Report (Third Section )

```
28
29 # Replace 'num' with the correct target variable column name
30 target_column = 'num'
31 df[target_column] = df[target_column].apply(lambda x: 0 if x == 0 else 1)
32
33 # Split the data into features (X) and target variable (y)
34 X = df.drop(target_column, axis=1)
35 y = df[target_column]
36
37 # Split the data into training and testing sets
38 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
39
40 # Define the XGBoost classifier
41 xgb_model = XGBClassifier()
42
43 # Fit the model on the training data
44 xgb_model.fit(X_train, y_train)
45
46 # Make predictions on the test set
47 y_pred = xgb_model.predict(X_test)
48
49 # Evaluate the performance
50 accuracy = accuracy_score(y_test, y_pred)
51 print(f'XGBoost Model Accuracy: {accuracy}')
52
```

XGBoost Model Accuracy: 0.8695652173913043

### Dataset

The dataset used for this analysis is the UCI Heart Disease dataset.

The dataset contains both categorical and numerical features.

### Data Preprocessing

Imputation: Missing values in the dataset were handled using the most frequent strategy.

Categorical Encoding: Categorical columns were encoded using label encoding.

Feature Scaling: Numerical features were standardized using StandardScaler.

Target Variable Transformation: The target variable 'num' was transformed into a binary format, where '0' indicates absence and '1' indicates presence of heart disease.

## **Data Splitting**

The dataset was split into training and testing sets using a 80-20 split ratio.

## **XGBoost Model**

The XGBoost classifier was chosen for this task

## **Model Training**

The XGBoost model was trained on the training set.

## **Model Evaluation**

The trained model was evaluated on the testing set using accuracy as the performance metric.

The accuracy of the XGBoost model on the test set is reported.

## **Results**

The XGBoost Model Accuracy: [insert accuracy value here]

## **Conclusion**

The XGBoost model, trained on the UCI Heart Disease dataset, achieved an accuracy of [insert accuracy value here] on the test set. This indicates the effectiveness of the model in predicting the presence or absence of heart disease based on the given features.

### **a) In machine learning, what is "ensemble learning," and how does XGBoost fit into this concept?**

#### **Ensemble Learning and XGBoost:**

Ensemble learning is a machine learning paradigm where multiple models are combined to improve overall performance. XGBoost fits into this concept by being an ensemble learning algorithm, specifically designed for gradient boosting. In gradient boosting, weak learners (usually decision trees) are sequentially added to the model, each one correcting errors made by the combination of existing models. XGBoost, or eXtreme Gradient Boosting, enhances traditional gradient boosting by employing a regularization term in the objective function and utilizing a more robust tree learning algorithm.

### **b) Explain the concept of "sparsity-aware" learning within the framework of the XGBoost algorithm. How does the algorithm handle sparse data differently from dense data?**

#### **Sparsity-Aware Learning in XGBoost:**

XGBoost incorporates a sparsity-aware algorithm to handle sparse data efficiently. Sparse data refers to datasets where a majority of the features have zero values. XGBoost leverages a novel data structure to represent the sparse feature matrix and uses an efficient algorithm to handle sparsity during tree construction. This allows XGBoost to perform better on datasets with a large number of zero-valued features compared to algorithms that don't explicitly consider sparsity.

**c) How does XGBoost handle missing values in input data? Explain the role of a missing value in the optimization process.**

**Handling Missing Values in XGBoost:**

XGBoost has a built-in capability to handle missing values in the input data. During the tree construction process, XGBoost automatically learns the best direction to take when a missing value is encountered for a particular feature. This is done by including missing values as a separate value in the split evaluation process, and the algorithm determines the optimal direction based on the reduction in the objective function.

**d) Compare the main goals of XGBoost and AdaBoost algorithms. What are the similarities and differences in their objectives?**

**Goals of XGBoost vs. AdaBoost:**

Both XGBoost and AdaBoost are ensemble learning techniques, but their main goals differ. XGBoost aims to improve the overall model performance by optimizing a differentiable loss function, making it suitable for various tasks. AdaBoost, on the other hand, focuses on combining weak learners to create a strong learner by assigning different weights to data points. While both aim to improve accuracy, XGBoost focuses on optimizing the model directly, whereas AdaBoost adjusts the weights of misclassified samples.

**e) In terms of system design, optimization techniques, and experimental results, it describes the main advantages of XGBoost over AdaBoost. Are there specific scenarios where AdaBoost still has an advantage?**

**Advantages of XGBoost over AdaBoost:**

XGBoost has several advantages over AdaBoost. It handles both linear and non-linear relationships efficiently, supports parallel and distributed computing, and includes regularization techniques to prevent overfitting. XGBoost also performs well with large datasets and is less prone to overfitting compared to AdaBoost. However, AdaBoost may have an advantage in scenarios where interpretability of the model is crucial, as its models are usually simpler.

**f) Implement XGBoost using the dataset from the first section and report the results. Use evaluation metrics such as accuracy, precision, recall, and F1-score. Compare the results with the first section**

**Implementing XGBoost and Reporting Results:**

Unfortunately, the provided text doesn't contain the details necessary to implement XGBoost on a specific dataset. To proceed with this task, I would need the dataset mentioned in the first section. Once the data is available, I can guide you through implementing XGBoost and evaluating its performance using metrics like accuracy, precision, recall, and F1-score.