CardioRisk: Unraveling Patterns for Early Diagnosis Using Machine Learning

Shruti Badrinarayanan

Department of Applied Data Science, San Jose State University

DATA 270: Data Analytics Process

Dr. Linsey Pang

December 4, 2023

4. Model Development

4.1 Model Proposal

Extreme Gradient Boosting, or XGBoost, is an efficient and scalable implementation of gradient boosted decision trees, widely used for structured data classification and regression problems. XGBoost is designed for speed and performance and operates by constructing new models to predict the residuals of previous models, then combining them for final prediction.

It is excellent for its crucial classification tasks such as predicting and diagnosing heart disease effectively. It starts with a straightforward model and refines it through cycles of assessment. The model learns by addressing previous errors, each time adding new decision trees (base learners) that correct them.

Figure 1

Pseudo code for XGBoost Algorithm

Algorithm 1 XGBoost Algorithm

Require: Training set $\{(x_i, y_i)\}_{i=1}^{N}$, a differentiable loss function $L(y, F(x))$, a number of weak learners M and a learning rate α.

Initialize model with a constant value:

$\hat{f}_0(x) = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, \theta)$.

for $m = 1$ to M **do**

Compute the 'gradients' and 'hessians':

$\hat{g}_m(x_i) = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x)=\hat{f}_{(m-1)}(x)}$.

$\hat{h}_m(x_i) = \left[\frac{\partial^2 L(y_i, f(x_i))}{\partial f(x_i)^2}\right]_{f(x)=\hat{f}_{(m-1)}(x)}$.

Fit a base learner (or weak learner, e.g., tree) using the training set $\{x_i, \frac{\hat{g}_m(x_i)}{\hat{h}_m(x_i)}\}_{i=1}^{N}$ by solving the optimization problem below:

$\phi_m = \underset{\phi \in \Phi}{\operatorname{argmin}} \sum_{i=1}^{N} \frac{1}{2} \left[\phi(x_i) - \frac{\hat{g}_m(x_i)}{\hat{h}_m(x_i)}\right]^2$.

$\hat{f}_m(x) = \alpha \phi_m(x)$.

Update the model:

$\hat{f}_m(x) = \hat{f}_{(m-1)}(x) + \hat{f}_m(x)$.

end for

Output $\hat{f}(x) = \hat{f}_M(x) = \sum_{m=0}^{M} \hat{f}_m(x)$.

Figure 1 goes into detail about the detailed pseudo code for the XGBoost algorithm. This pseudocode was obtained from studying the XGBoost algorithm for the problem from various sources such as the XGBoost library from sklearn, and tutorials. In practice, XGBoost

uses more sophisticated techniques for handling various aspects of the training process, such as regularization and handling of missing values.

A crucial element of XGBoost is the learning rate, which dictates how fast the model learns. Too fast, and the model might be overfit to the training data, failing to generalize to new data. Too slow, and the model might not learn enough to be effective. The learning continues until the model no longer shows substantial improvement or meets the iteration limit. The final model is an aggregation of all the iterative changes, combining the knowledge gained for the most accurate predictions.

XGBoost's parameters are categorized into general, booster, and task parameters. General parameters define the booster type, booster parameters depend on the chosen booster, and task parameters specify the learning objective.

In tuning the XGBoost algorithm, the chosen hyperparameters are max_depth, learning_rate, n_estimators, and subsample. The max_depth parameter is crucial for regulating the complexity of the trees, dictating how deep the trees can grow and therefore the level of detail they can represent. The learning_rate influences the contribution of each tree to the final outcome, with a smaller value typically requiring more trees but potentially yielding better generalization performance. n_estimators is the number of trees in the ensemble; too few may not capture the data structure well, while too many can lead to overfitting. Finally, subsample determines the fraction of the total dataset that is used to train each tree, providing a means to control overfitting by adding some randomness to the model training process. Selecting the right combination of these hyperparameters through a systematic grid search can lead to a more accurate and robust model.

The algorithm employs gradient descent to minimize loss, working in a forward stage-wise fashion. It sequentially introduces weak learners to compensate for existing ones' deficiencies.

The objective function for XGBoost is defined as:

$$L(\theta) = \sum [l(y_i, \hat{y}_i)] + \sum \Omega(f_i), \quad (1)$$

where l is the loss function quantifying the difference between prediction $\hat{y}$ and target y, and Ω imposes a penalty on model complexity. XGBoost's regularization and column subsampling features effectively prevent overfitting.

The boosting process builds predictors sequentially to correct previous errors, calculates gradients, constructs a tree to predict gradients, updates the model, and outputs the final ensemble model.

A Simpler Pseudo code for the XGBoost algorithm is as follows:

1. Initialize the model with a constant value: $\hat{y}_i^{(0)}$.
2. For $m = 1$ to M (total number of trees):
    a. Compute the gradients: $g_i = \frac{\partial L}{\partial \hat{y}_i^{(m-1)}}$.
    b. Fit a new model (tree) $f_m(x)$ to predict the gradients.
    c. Find the optimal output value for the new model: $\omega = \underset{\omega}{\operatorname{argmin}} \sum L(y_i, \hat{y}_i^{(m-1)} + \omega f_m(x_i))$.
    d. Update the model: $\hat{y}_i^{(m)} = \hat{y}_i^{(m-1)} + \eta \omega f_m(x_i)$.
3. Output the final model: $\hat{y}_i^{(M)}$.

XGBoost is robust to missing data, can scale and handle diverse predictive modeling problems such as the diagnosis of heart disease.

4.2 Model Supports

Environment, Platform, and Tools

The model is trained with optimal resource allocation using high-performance processing. Data is stored in CSV files, and Python serves as the primary platform, excelling

across various domains. Python's extensive library ecosystem for machine learning, combined with the versatility of the Jupyter Notebook IDE with Python 3.9, was employed for implementing, training, validating, testing, and evaluating, visualizing metrics for the various models. All the libraries used in this implementation along with the purpose are detailed in Table 1.

Table 1

Libraries Used for Model Development

Purpose	**Libraries Used**	**Description**
Data Manipulation	pandas, numpy	They offer efficient data manipulation tools for analysis and data manipulation.
Data Visualization	matplotlib, seaborn	Plotting various visualizations
Data Preprocessing	pandas, numpy,	To effectively prepare and explore datasets before model training.
Model Building and Training	train_test_split, scikit-learn, xgboost	Split the training data for training and validation, building the xgboost models
Model Evaluation	from sklearn.metrics : accuracy_score, confusion_matrix, classification_report, roc_curve, roc_auc_score, mean_squared_error,log_loss, precision_score, recall_score, f1_score	A comprehensive set of tools for assessing the performance of models.
Hyperparameter Tuning	scikit-learn	For hyperparameter tuning, enabling systematic exploration and optimization of model parameters to enhance model performance.

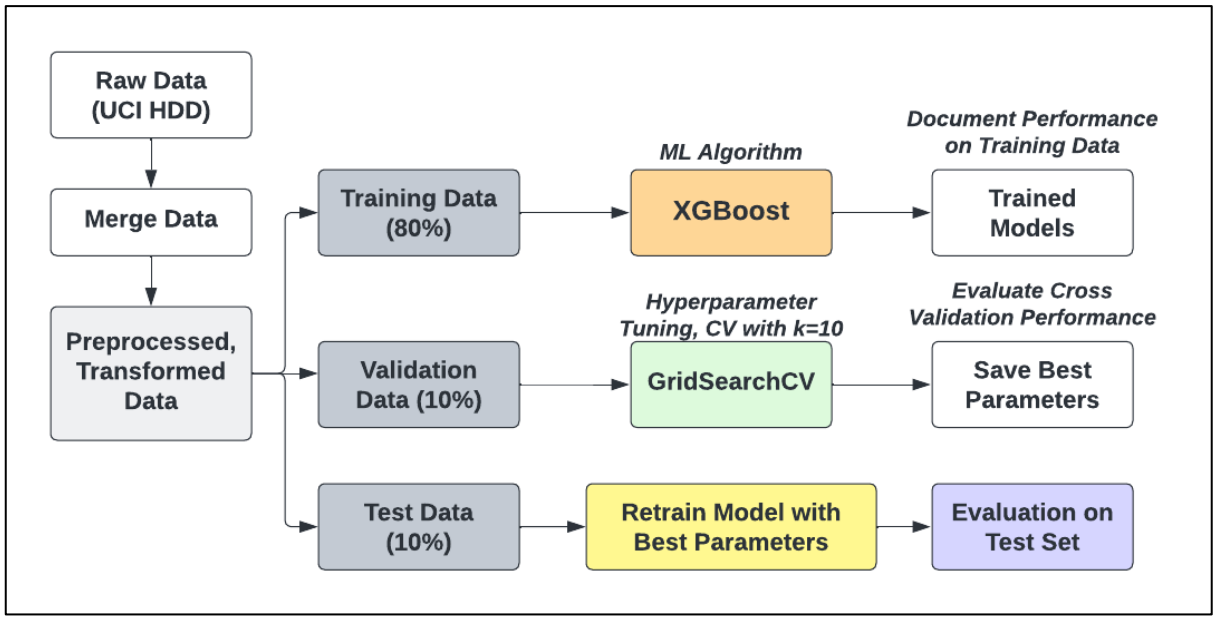
Model Architecture and Dataflow

Figure 2 demonstrates a general data flow and system architecture diagram for the implementation and evaluation of the XGBoost model. The raw data is obtained from UCI Machine Learning Dataset which is available in 4 distinct files. These files are merged and then preprocessed to handle missing values and outliers. Post this stage, the data undergoes one hot encoding for certain features.

Once this is complete, PCA dimensionality technique is used to transform the data. The resultant dataset undergoes a split into train, validation, and test sets with an 80:10:10 ratio. To facilitate this split, the data is separately saved as CSV files. XGboost models are implemented using xgboost library.

Figure 2

Data Flow and System Architecture



Hyperparameter tuning is employed to enhance performance and execution time for machine learning models. GridSearchSV is employed to search for optimal parameters by exploring different instances of each parameter, estimating results, and returning the parameter set with the best outcomes. Once the best hyperparameters are determined, the

models are optimized using these values. 10-fold Cross Validation is also done and the cross validation accuracy is evaluated.

The saved parameters are used to build the model on the test set and the performance is evaluated using various metrics. Any general-purpose tasks are facilitated through the utilization of Pandas and NumPy libraries.

4.3 Model Comparison and Justification

In the study by Yang. J et al. (2020), XGBoost was employed to predict the occurrence of Major Adverse Cardiac and Cerebrovascular Events (MACCE) in heart disease patients. XGBoost excels in this context due to its ability to sequentially train models using residuals, effectively reducing error over numerous iterations. The algorithm optimizes the objective function through a second-order Taylor expansion and controls complexity with a regularization term, thus mitigating overfitting risks. This approach contributes to XGBoost's high accuracy and efficiency, making it well-suited for predictive modeling in complex medical scenarios.

However, the application of XGBoost in predicting MACCE is not without its challenges. The algorithm's computational intensity can be a limitation, particularly in large datasets or scenarios requiring rapid processing. Additionally, the complexity of XGBoost and the necessity for hyperparameter tuning might present difficulties, especially for practitioners who are not deeply versed in machine learning optimization. Despite these challenges, the balance of XGBoost's predictive accuracy and efficiency makes it a valuable tool in the medical field, particularly in predicting critical outcomes like MACCE in heart disease patients.

In the research conducted by Budholiya, Shrivastava, & Sharma (2022), XGBoost was leveraged for predicting heart disease, showcasing its effectiveness in diagnostic systems. The key strength of XGBoost in this study lies in its use of ensemble tree methods

and gradient descent architecture to enhance weak learners. This is achieved through system optimization and algorithmic improvements, which include managing multiple tree-related hyperparameters like subsample and max_depth to mitigate overfitting and improve model performance. Moreover, the application of Bayesian optimization for hyperparameter tuning significantly contributed to the model's efficiency, allowing for refined adjustments that directly impact its predictive power.

Still, the complexity of XGBoost and the need for careful hyperparameter tuning can be seen as potential drawbacks. While Bayesian optimization aids in this process, the necessity of such fine-tuning can be a hurdle, especially in settings where rapid model deployment is required. Despite these challenges, the study's results demonstrated that the optimized XGBoost model outperformed other models like Random Forest (RF) and Extra Tree (ET) classifiers in various metrics, including accuracy and Area Under the ROC Curve (AUC). This indicates that, although XGBoost requires careful configuration and has a complex architecture, its ability to deliver high predictive accuracy (91.8%) makes it a valuable tool in clinical settings for heart disease prediction.

In the study conducted by Rajliwall, Davey, & Chetty (2018), XGBoost was utilized for cardiovascular risk prediction within a novel predictive modeling framework. The study highlighted the effectiveness of XGBoost in handling large and complex datasets like the NHANES dataset and the Framingham Heart Study (FHS) dataset. XGBoost's ability to combine many simpler models through boosting, a technique that involves penalizing individual trees to manage the bias-variance tradeoff, was pivotal in achieving high predictive accuracy. This scalability and efficient memory usage, driven by parallel and distributed computing, allow XGBoost to perform well in terms of both prediction accuracy and model building time.

But, the study also noted the trade-offs necessary when using XGBoost, particularly between model accuracy and building time. While XGBoost demonstrated superior performance in terms of accuracy (97.7% on the NHANES dataset and 89.9% on the FHS dataset) and had one of the shortest model building times, it is essential to balance these aspects for real-time data analysis. The flexibility and efficiency of XGBoost make it a good candidate for predictive modeling in dynamic settings where quick model generation and high accuracy are crucial. This balance of speed and accuracy makes XGBoost a valuable tool in the healthcare domain, especially for tasks like cardiovascular risk prediction where timely and accurate predictions can significantly impact patient outcomes.

The advantages and disadvantages of the XGboost algorithm are given in Table 2.

Table 2

XGBoost's Advantages and Disadvantages

Advantages	**Disadvantages**
• Excels in processing complex data, crucial for medical diagnostics. • Manages missing values and diverse data types effectively. • Regularization to combat overfitting in small datasets. • Model Interpretability: Offers insights into feature contributions using SHAP values.	• Can overfit if regularization isn't managed properly. • Tuning numerous hyperparameters requires significant compute power. • Limited size of dataset raises overfitting risks. • Interpretability-Accuracy Balance: While XGBoost can be made interpretable, the most accurate models may use complex ensembles of trees that are harder to interpret than simpler models.

In summary, across diverse studies, XGBoost has established itself as a formidable tool in the realm of medical predictive modeling, particularly for cardiovascular diseases. Its

sophisticated ensemble tree methods and boosting techniques, along with its ability to optimize through regularization and handle large datasets efficiently, render it highly accurate and swift in model training. The integration of advanced methods like Bayesian optimization for hyperparameter tuning further enhances its performance, contributing to its robust predictive power. However, the need for meticulous hyperparameter adjustment and computational demands remain as challenges that practitioners must navigate. Despite these hurdles, XGBoost's exceptional predictive accuracy and its adaptability in balancing speed and accuracy underline its potential as a pivotal asset in clinical decision-making and risk assessment, significantly impacting patient care and management.

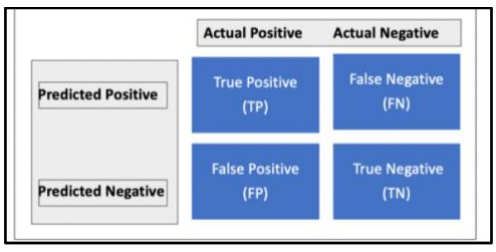
4.4 Model Evaluation Methods

Confusion Matrix

A confusion matrix is a foundational tool for evaluating the performance of a classification model. It provides a comprehensive breakdown of predicted and actual class memberships, with four key components: True Positive (correctly predicted positive instances), True Negative (correctly predicted negative instances), False Positive (incorrectly predicted as positive), and False Negative (incorrectly predicted as negative). From the confusion matrix, several other metrics can be derived, such as accuracy, precision, recall, specificity, and the F1-Score. Figure 3 shows a confusion matrix for a binary classification problem. These metrics offer nuanced insights into different aspects of model performance, making the confusion matrix a versatile and widely used evaluation tool.

Figure 3

Confusion Matrix for Classification



Accuracy Score

Accuracy is a fundamental metric that gauges the overall correctness of predictions made by a model. It is calculated by dividing the sum of correctly predicted instances (both true positives and true negatives) by the total number of instances as shown in Figure 4. While accuracy provides a quick and easy-to-understand measure of performance, it may not be suitable for imbalanced datasets where one class significantly outnumbers the other. In such cases, accuracy can be misleading, and it's crucial to consider additional metrics, such as precision, recall, and the F1-Score, for a more comprehensive evaluation.

Figure 4

Formula for Accuracy Score

$$Accuracy = \frac{TrueNegatives + TruePositive}{TruePositive + FalsePositive + TrueNegative + FalseNegative}$$

Precision

Precision is a metric that emphasizes the accuracy of positive predictions made by a model. It assesses the ratio of true positive predictions to the total number of predicted positive instances as shown in Figure 5. Precision is particularly relevant in scenarios where the cost of false positives is high.

Figure 5

Formula for Precision in terms of 'True positives' and 'False Positives'

$$Precision = \frac{True\ positive}{True\ Positive + False\ Positive}$$

For example, in medical diagnoses, a high precision indicates a low rate of false positives, meaning that when the model predicts a positive result, it is likely to be correct. However, precision should be considered alongside other metrics, as optimizing for high

precision might lead to lower recall, and striking the right balance is crucial for a well-rounded evaluation.

Recall (Sensitivity or True Positive Rate)

Recall, also known as sensitivity or the true positive rate, measures a model's ability to correctly identify all relevant instances of a particular class. It is calculated by dividing the number of true positive predictions by the sum of true positives and false negatives as shown in Figure 6.

Figure 6

Formula for Recall

$$Recall = \frac{True\ Positive}{True\ Positive + True\ Negative}$$

Recall is crucial in scenarios where missing positive instances is more critical than erroneously identifying negative instances. For instance, in spam email detection, high recall ensures that the model captures most of the spam emails, even if it means tolerating some false positives.

F1-Score

The F1-Score is a harmonic mean of precision and recall as shown in Figure 7, offering a balanced assessment of a model's performance. It considers both false positives and false negatives, providing a single metric that reflects the trade-off between precision and recall. The F1-Score is particularly useful when there is an uneven class distribution or when both false positives and false negatives need to be minimized simultaneously. It becomes especially relevant in situations where precision and recall alone may not provide a clear picture of the model's effectiveness, making it a valuable metric for comprehensive model evaluation.

Figure 7

Formula for F1-Score

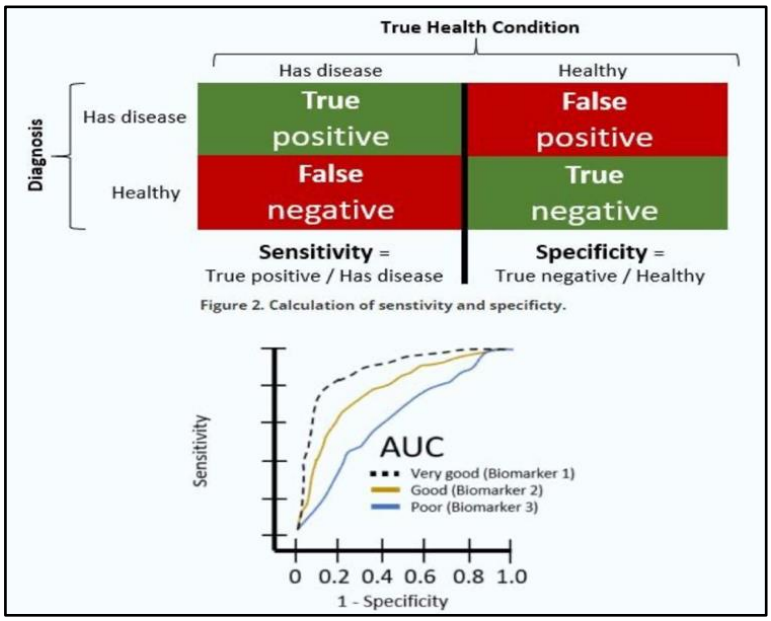
$$F = 2 * \frac{Precision * Recall}{Precison + Recall}$$

Receiver Operating Curve (ROC) and Area Under the Curve (AUC)

The Receiver Operating Characteristic (ROC) curve is a graphical representation of a model's ability to distinguish between classes at various classification thresholds, plotting the true positive rate against the false positive rate. The Area Under the Curve (AUC) is a numerical measure derived from the ROC curve, offering a comprehensive assessment of a model's discriminatory performance. A perfect model has an AUC of 1, indicating optimal discrimination, while random guessing corresponds to an AUC of 0.5.

Figure 8

AUC-ROC Curve



The ROC AUC is particularly valuable in binary classification tasks, providing a model with the ability to correctly identify. It is instrumental in scenarios with imbalanced class distributions or when the consequences of false positives and false negatives differ. The

ROC AUC complements other evaluation metrics, offering a holistic view of a model's performance in distinguishing between classes across varying decision thresholds. Figure 8 shows the specifics of the AUC-ROC curve.

4.5 Model Validation and Evaluation

Baseline Model. As depicted in Figure 9, the baseline XGBoost model exhibits remarkable performance metrics during the training phase. The model boasts a training accuracy of 1.0000, along with an equal score for balanced accuracy, precision, recall, and F1 score. Such uniform perfection across these metrics is rarely observed in real-world scenarios and suggests that the model is exceptionally well-fitted to the training data. Furthermore, the model achieves an ideal training ROC AUC score of 1.0000, and a training log loss of a mere 0.0045, underscoring the model's adeptness at classification without any observable error.

The confusion matrix within Figure 9 reinforces the prowess of the baseline model, indicating no false negatives or false positives among the predictions. All 315 instances of one class and 404 instances of the other class were classified with 100% accuracy. The accompanying ROC curve, plateauing at the top-right corner of the graph, visually confirms the model's flawless discriminatory ability during training.

However, while the results in Figure 9 are impressive, they also raise concerns regarding the potential for overfitting. A model that perfectly predicts training data can be indicative of memorization rather than generalization. It is imperative to assess how the model performs on unseen data to ensure that these training metrics translate into equally robust real-world performance. This will be crucial for validating the model's practical utility in clinical environments, where the ability to generalize is paramount for accurate diagnosis and treatment planning.

Figure 9

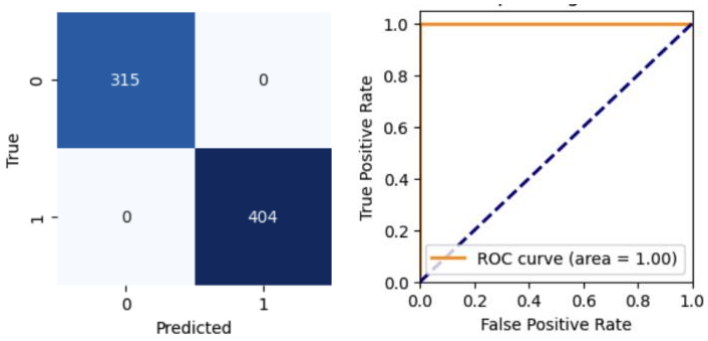
Metrics for XGBoost Baseline Model

```
Training Accuracy: 1.0000
Training Balanced Accuracy: 1.0000
Training Precision: 1.0000
Training Recall: 1.0000
Training F1 Score: 1.0000
Training ROC AUC: 1.0000
Training Log Loss: 0.0045

Training Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       315
         1.0       1.00      1.00      1.00       404

    accuracy                           1.00       719
   macro avg       1.00      1.00      1.00       719
weighted avg       1.00      1.00      1.00       719
```



Hyperparameter Tuned Model. The hyperparameter tuning phase for the model was conducted using GridSearchCV. The search spanned a range of hyperparameters, including max_depth values of 3, 5, and 7, learning_rate options of 0.01, 0.1, and 0.2, n_estimators set to 100, 200, and 300, and subsample ratios of 0.8 and 1. Upon executing the grid search, the best parameter combination was identified as a learning_rate of 0.1, max_depth of 5, n_estimators of 100, and a subsample of 1. This configuration achieved a cross-validation accuracy of 0.9000 on the validation data as shown in Figure 10.

Figure 10

Best Parameters, Validation Accuracy of XGBoost Hyperparameter Tuned Model

```
Best Parameters: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 100, 'subsample': 1}
Best Cross Validation Accuracy: 0.9000
```

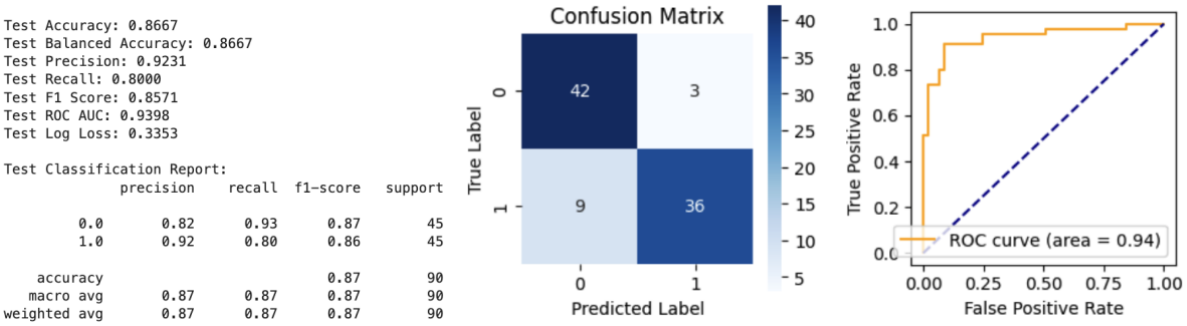
The results suggest that these hyperparameters provide a strong balance between model complexity and generalization capability. With a relatively modest depth and number of estimators, the model is complex enough to capture underlying patterns in the data without overfitting, as evidenced by the high cross-validation accuracy. Following this, the optimal parameters will be applied to the final model, which is expected to enhance its predictive performance while maintaining robustness against overfitting. This step is crucial in ensuring that the model can generalize well to unseen data, which is critical for diagnosing heart disease.

Test Data. Upon evaluation with test data, the hyperparameter-tuned XGBoost model, as showcased in Figure 11, demonstrates robust predictive performance. The model achieves a test accuracy of 86.67%, which, while not perfect like the training phase, is still notably high for practical applications. The balance between precision and recall is evident, with scores of 92.31% and 80.00%, respectively. This balance is further highlighted by the F1 score of 85.71%, which is a harmonic mean of precision and recall, indicating a strong model performance when both aspects are considered. The Receiver Operating Characteristic (ROC) curve, with an area under the curve (AUC) of 93.98%, illustrates the model's strong capability to discriminate between the classes at various threshold settings.

The confusion matrix within Figure 11 presents a more granular view of the model's performance on the test data. It reveals a small number of false positives (3) and false negatives (9), which are acceptable given the model's high number of true positives (36) and true negatives (42). This indicates a high true positive rate and a low false positive rate, which are critical metrics for clinical decision-making tools where the cost of a false negative can be significant. Some key code snippets and implementation of XGBoost for this project is available in Appendix A.

Figure 11

Metrics for XGBoost Hyperparameter Tuned with Test Data



These test results provide a realistic assessment of the model's capability to generalize to new, unseen data. A test ROC AUC of 93.98% is indicative of a model that is not only

well-tuned but also capable of maintaining high performance when applied to real-world scenarios. This confirms the model's utility as a reliable tool for diagnosing heart disease, balancing accuracy with speed and demonstrating the effectiveness of the hyperparameter tuning process.

References

Budholiya, K., Shrivastava, S. K., & Sharma, V. (2022). *An optimized XGBoost based diagnostic system for effective prediction of heart disease.* Journal of King Saud University-Computer and Information Sciences, 34(7), 4514-4523. https://www.sciencedirect.com/science/article/pii/S1319157820304936

Rajliwall, N. S., Davey, R., & Chetty, G. (2018, December). *Cardiovascular risk prediction based on XGBoost.* In 2018 5th Asia-Pacific World Congress on Computer Science and Engineering (APWC on CSE) (pp. 246-252). IEEE. https://ieeexplore.ieee.org/abstract/document/8853798

Yang, J., & Guan, J. (2022). *A heart disease prediction model based on feature optimization and smote-Xgboost algorithm.* Information, 13(10), 475. https://www.mdpi.com/2078-2489/13/10/475

Appendix A

Code for Implementing XGBoost

This appendix has sample code from the modeling stage of the XGBoost Algorithm for cardiovascular disease prediction. The code for Model Validation is shown in Figure A1 and Figure A2 has the code for Model Evaluation with best hyperparameters on test data.

```python
from sklearn.model_selection import GridSearchCV

# Define the hyperparameter grid
param_grid = {
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
    'n_estimators': [100, 200, 300],
    'subsample': [0.8, 1]
}

# Initialize GridSearchCV with the XGBoost classifier
xgb_grid_search = GridSearchCV(
    estimator=xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss'),
    param_grid=param_grid,
    scoring='accuracy',
    cv=10
)

# Perform the grid search on the validation data
xgb_grid_search.fit(X_val, y_val)

# Output the best parameters and the highest cross validation accuracy
best_params = xgb_grid_search.best_params_
best_score = xgb_grid_search.best_score_
print(f"Best Parameters: {best_params}")
print(f"Best Cross Validation Accuracy: {best_score:.4f}")
```

Figure A1. Code for XGBoost Model Validation

```python
# Use the best estimator from the grid search to make predictions on the test data
y_test_pred = xgb_grid_search.best_estimator_.predict(X_test)
y_test_proba = xgb_grid_search.best_estimator_.predict_proba(X_test)[:, 1]

# Compute various evaluation metrics for the test data
test_accuracy = accuracy_score(y_test, y_test_pred)
test_balanced_accuracy = balanced_accuracy_score(y_test, y_test_pred)
test_precision = precision_score(y_test, y_test_pred)
test_recall = recall_score(y_test, y_test_pred)
test_f1 = f1_score(y_test, y_test_pred)
test_roc_auc = roc_auc_score(y_test, y_test_proba)
test_log_loss = log_loss(y_test, y_test_proba)
test_conf_matrix = confusion_matrix(y_test, y_test_pred)
test_classification_report = classification_report(y_test, y_test_pred)

# Display the evaluation metrics for the test set
print(f"Test Accuracy: {test_accuracy:.4f}")
print(f"Test Balanced Accuracy: {test_balanced_accuracy:.4f}")
print(f"Test Precision: {test_precision:.4f}")
print(f"Test Recall: {test_recall:.4f}")
print(f"Test F1 Score: {test_f1:.4f}")
print(f"Test ROC AUC: {test_roc_auc:.4f}")
print(f"Test Log Loss: {test_log_loss:.4f}")
print("\nTest Classification Report:")
print(test_classification_report)

# Plot the confusion matrix
print("\nTest Confusion Matrix:")
plt.figure(figsize=(3, 3))
sns.heatmap(test_conf_matrix, annot=True, fmt='d', cmap='Blues', square=True)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

Figure A2. Code for XGBoost Model Evaluation