Weihao Liu, Karumanchi Samuel

Arun Jagota

Data mining

2023/06/08

Term project report

Heart disease remains one of the leading causes of mortality worldwide, and early detection is crucial for effective treatment and management. By applying supervised learning techniques to the UCI Heart Disease dataset, we can develop predictive models that help identify individuals at risk of heart disease. Such models not only contribute to advancing medical research but also have the potential to aid healthcare professionals in making more informed decisions, ultimately improving patient outcomes and saving lives. Given its prominence in the research community, it provides a robust framework for developing and testing predictive models aimed at classifying the presence or absence of heart disease based on various patient attributes.

The dataset was obtained from the UCI Machine Learning Repository. In detail, the Cleveland Heart Disease dataset subset. The dataset contains 14 attributes, including the target variable. The predictor attributes include both numeric and categorical data types:

- Numeric: age, trestbps (resting blood pressure), chol (serum cholesterol), thalach (maximum heart rate achieved), oldpeak (ST depression induced by exercise relative to rest).
- Categorical (nominal): sex, cp (chest pain type), fbs (fasting blood sugar), restecg (resting electrocardiographic results), exang (exercise-induced angina), slope (the slope of the peak exercise ST segment), ca (number of major vessels colored by fluoroscopy), thal (thalassemia).

The dataset in total contains 303 instances. One of its notable characteristics is the presence of missing values, marked by '?', which need to be handled appropriately during preprocessing.

To address the classification problem of predicting heart disease using the UCI Heart Disease dataset, we employed three distinct supervised learning algorithms: Logistic Regression, Random Forest, and Decision Tree. Logistic Regression is a statistical method for binary classification problems. It models the probability that a given input point belongs to a certain class. Random Forest is an ensemble learning method that constructs a multitude of decision trees during training and outputs the mode of the classes (classification) or mean prediction (regression) of the individual trees. A Decision Tree is a flowchart-like structure where an internal node represents a feature (or attribute), the branch represents a decision rule, and each leaf node represents the outcome. The paths from root to leaf represent classification rules. We used the usual hyperparameters that were necessary like the tree depth, C, 1/lambda to tune the closeness of the training data, etc.

To evaluate our models, we need to examine specific hyperparameters that significantly influence their performance. For the Decision Tree and Random Forest models, the depth of the tree is crucial. For Logistic Regression, the regularization parameter C is key. By varying the depth of the tree, we can balance the trade-off between bias and variance and by tuning the C value, we can find an optimal balance between fitting the training data well and maintaining generalizability to unseen data.

During model selection, we evaluated the performance of the various models using accuracy as the primary metric. Accuracy is straightforward to understand and interpret. It gives a clear indication of how well the model is performing in terms of overall correctness. The UCI Heart Disease dataset has a relatively balanced distribution of classes (presence and absence of heart disease). In cases where classes are balanced, accuracy is a reliable metric as it reflects the true performance of the model without being skewed by class imbalances. While accuracy is suitable for our case due to the balanced classes, it's important to consider other metrics in scenarios with class imbalance or specific business needs. Metrics such as precision, recall, F1-score, and ROC-AUC provide more nuanced insights, particularly in imbalanced datasets or when the cost of false positives and false negatives differs.

For the final evaluation of the selected model on the test set, we used the same metric: accuracy. Using the same metric (accuracy) for both model selection and final evaluation ensures consistency in our evaluation process. This allows us to directly compare the performance of different models throughout the development pipeline without introducing variability in the evaluation criteria. After selecting the best-performing model based on accuracy from the cross-validation results, we evaluated this model on the test set using the same accuracy metric to ensure that it generalizes well to unseen data.

After evaluating multiple models, we concluded that **Logistic Regression** performed the best in predicting heart disease from the UCI Heart Disease dataset. It consistently achieved higher accuracy compared to the Decision Tree and Random Forest models. Logistic Regression, despite its simplicity, proved to be highly effective for this classification problem. Its linear decision boundary was sufficient to capture the relationships in the data. Adjusting hyperparameters such as the depth of trees for Decision Trees and Random Forests, and the regularization parameter C for Logistic Regression, is essential for optimizing model performance. Our experiments highlighted how sensitive models can be to these parameters. Our analysis demonstrated that Logistic Regression is the most effective model for predicting heart disease using the UCI Heart Disease dataset. It balances accuracy with interpretability, making it an excellent choice for this medical classification task. The project underscored the importance of careful model selection, hyperparameter tuning, and the consideration of dataset characteristics in achieving reliable and meaningful predictive performance.

Below we have the detailed code with results and graph respect to each method.

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score,␣
↪roc_auc_score, roc_curve, auc

#load dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/
↪processed.cleveland.data"
columns = ["age", "sex", "cp", "trestbps", "chol", "fbs", "restecg", "thalach",
           "exang", "oldpeak", "slope", "ca", "thal", "target"]

df = pd.read_csv(url, names=columns)
df.replace('?', pd.NA, inplace=True)
df = df.apply(pd.to_numeric)
print(df.info())
print(df.describe())
print(df['target'].value_counts())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    float64
 1   sex       303 non-null    float64
 2   cp        303 non-null    float64
 3   trestbps  303 non-null    float64
 4   chol      303 non-null    float64
 5   fbs       303 non-null    float64
 6   restecg   303 non-null    float64
 7   thalach   303 non-null    float64
```

```
 8   exang     303 non-null    float64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    float64
 11  ca        299 non-null    float64
 12  thal      301 non-null    float64
 13  target    303 non-null    int64
dtypes: float64(13), int64(1)
memory usage: 33.3 KB
None
              age         sex          cp    trestbps        chol         fbs  \
count  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000
mean    54.438944    0.679868    3.158416  131.689769  246.693069    0.148515
std      9.038662    0.467299    0.960126   17.599748   51.776918    0.356198
min     29.000000    0.000000    1.000000   94.000000  126.000000    0.000000
25%     48.000000    0.000000    3.000000  120.000000  211.000000    0.000000
50%     56.000000    1.000000    3.000000  130.000000  241.000000    0.000000
75%     61.000000    1.000000    4.000000  140.000000  275.000000    0.000000
max     77.000000    1.000000    4.000000  200.000000  564.000000    1.000000

          restecg     thalach       exang     oldpeak       slope          ca  \
count  303.000000  303.000000  303.000000  303.000000  303.000000  299.000000
mean     0.990099  149.607261    0.326733    1.039604    1.600660    0.672241
std      0.994971   22.875003    0.469794    1.161075    0.616226    0.937438
min      0.000000   71.000000    0.000000    0.000000    1.000000    0.000000
25%      0.000000  133.500000    0.000000    0.000000    1.000000    0.000000
50%      1.000000  153.000000    0.000000    0.800000    2.000000    0.000000
75%      2.000000  166.000000    1.000000    1.600000    2.000000    1.000000
max      2.000000  202.000000    1.000000    6.200000    3.000000    3.000000

             thal      target
count  301.000000  303.000000
mean     4.734219    0.937294
std      1.939706    1.228536
min      3.000000    0.000000
25%      3.000000    0.000000
50%      3.000000    0.000000
75%      7.000000    2.000000
max      7.000000    4.000000
target
0    164
1     55
2     36
3     35
4     13
Name: count, dtype: int64
```

```
X = df.drop('target', axis=1)
y = df['target'].apply(lambda x: 1 if x > 0 else 0)

X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val,␣
 ↪test_size=0.25, random_state=42)
numeric_features = ["age", "trestbps", "chol", "thalach", "oldpeak"]
categorical_features = ["sex", "cp", "fbs", "restecg", "exang", "slope", "ca",␣
 ↪"thal"]

numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)])

X_train_preprocessed = preprocessor.fit_transform(X_train)
X_val_preprocessed = preprocessor.transform(X_val)
X_test_preprocessed = preprocessor.transform(X_test)
```

```
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier()
}

param_grid = {
    'Logistic Regression': {'C': [0.1, 1, 10]},
    'Decision Tree': {'max_depth': [None, 10, 20, 30]},
    'Random Forest': {'n_estimators': [10, 50, 100]}
}

best_models = {}
for name, model in models.items():
    grid_search = GridSearchCV(model, param_grid[name], cv=5, scoring='accuracy')
    grid_search.fit(X_train_preprocessed, y_train)
    best_models[name] = grid_search.best_estimator_
    print(f"{name}: Best Parameters -> {grid_search.best_params_}")
```

```
Logistic Regression: Best Parameters -> {'C': 0.1}
```

```
Decision Tree: Best Parameters -> {'max_depth': None}
Random Forest: Best Parameters -> {'n_estimators': 100}
```

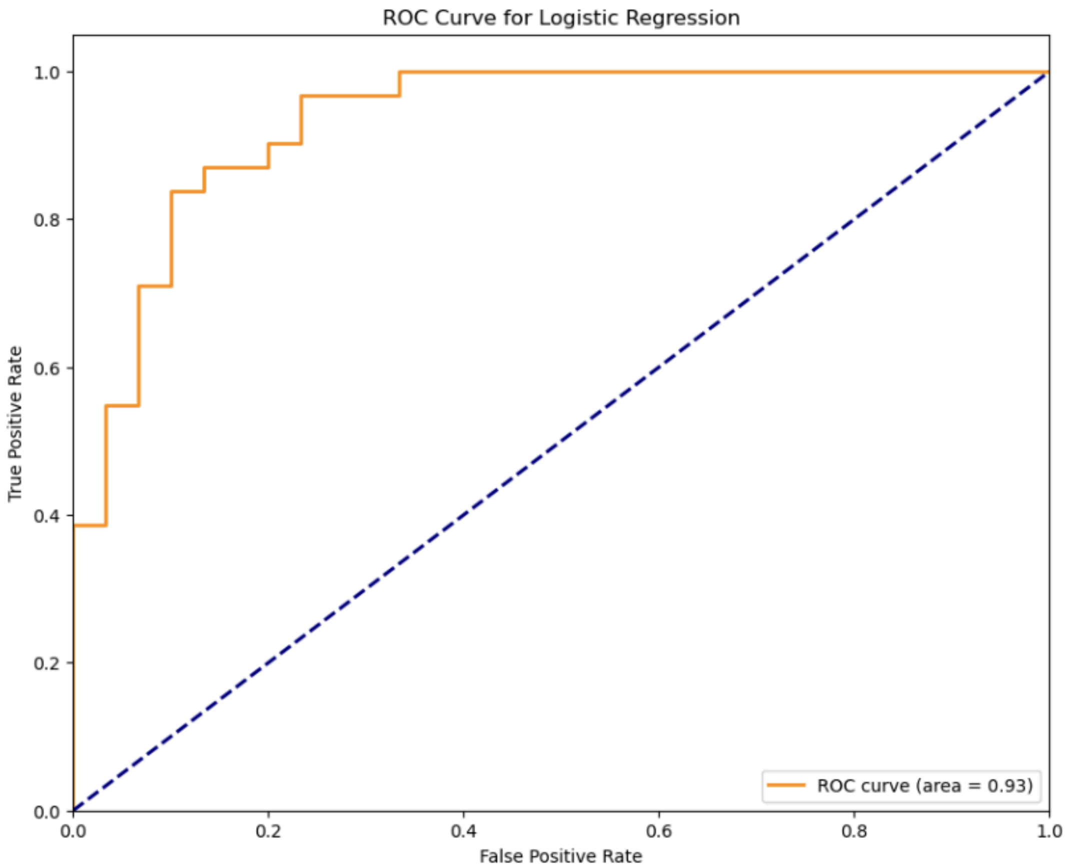
```
for name, model in best_models.items():
    y_val_pred = model.predict(X_val_preprocessed)
    print(f"{name} Validation Accuracy: {accuracy_score(y_val, y_val_pred)}")

best_model_name = max(best_models, key=lambda name: accuracy_score(y_val,␣
 ↪best_models[name].predict(X_val_preprocessed)))
best_model = best_models[best_model_name]

y_test_pred = best_model.predict(X_test_preprocessed)
print(f"Best Model: {best_model_name}")
print("Test Set Performance:")
print(f"Accuracy: {accuracy_score(y_test, y_test_pred)}")
print(f"ROC AUC: {roc_auc_score(y_test, y_test_pred)}")
print(classification_report(y_test, y_test_pred))
```

```
Logistic Regression Validation Accuracy: 0.8524590163934426
Decision Tree Validation Accuracy: 0.6885245901639344
Random Forest Validation Accuracy: 0.8032786885245902
Best Model: Logistic Regression
Test Set Performance:
Accuracy: 0.8360655737704918
ROC AUC: 0.8372844827586207
              precision    recall  f1-score   support

           0       0.81      0.86      0.83        29
           1       0.87      0.81      0.84        32

    accuracy                           0.84        61
   macro avg       0.84      0.84      0.84        61
weighted avg       0.84      0.84      0.84        61
```

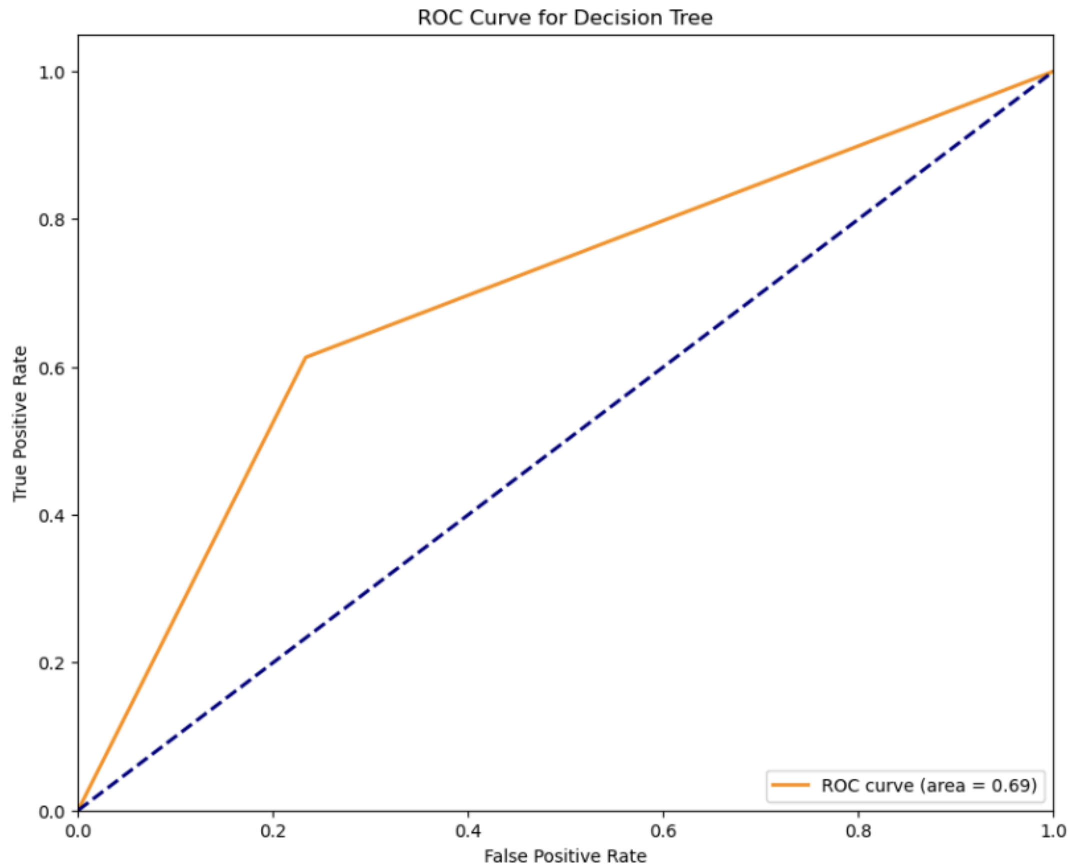
```
def plot_roc_curve(fpr, tpr, roc_auc, label=None):
    plt.figure(figsize=(10, 8))
    plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area =␣
 ↪{roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend(loc='lower right')
    if label:
        plt.title(f'ROC Curve for {label}')
```

```
    plt.show()

for name, model in best_models.items():
    y_val_pred_prob = model.predict_proba(X_val_preprocessed)[:, 1]
    fpr, tpr, _ = roc_curve(y_val, y_val_pred_prob)
    roc_auc = auc(fpr, tpr)
    print(f"{name} Validation ROC AUC: {roc_auc:.2f}")
    plot_roc_curve(fpr, tpr, roc_auc, label=name)

best_model_name = max(best_models, key=lambda name: roc_auc_score(y_val,␣
 ↪best_models[name].predict_proba(X_val_preprocessed)[:, 1]))
best_model = best_models[best_model_name]

y_test_pred_prob = best_model.predict_proba(X_test_preprocessed)[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_test_pred_prob)
roc_auc = roc_auc_score(y_test, y_test_pred_prob)
print(f"Best Model: {best_model_name}")
print("Test Set Performance:")
print(f"Accuracy: {accuracy_score(y_test, best_model.
 ↪predict(X_test_preprocessed))}")
print(f"ROC AUC: {roc_auc:.2f}")
print(classification_report(y_test, best_model.predict(X_test_preprocessed)))

plot_roc_curve(fpr, tpr, roc_auc, label=f'Best Model: {best_model_name}')
```

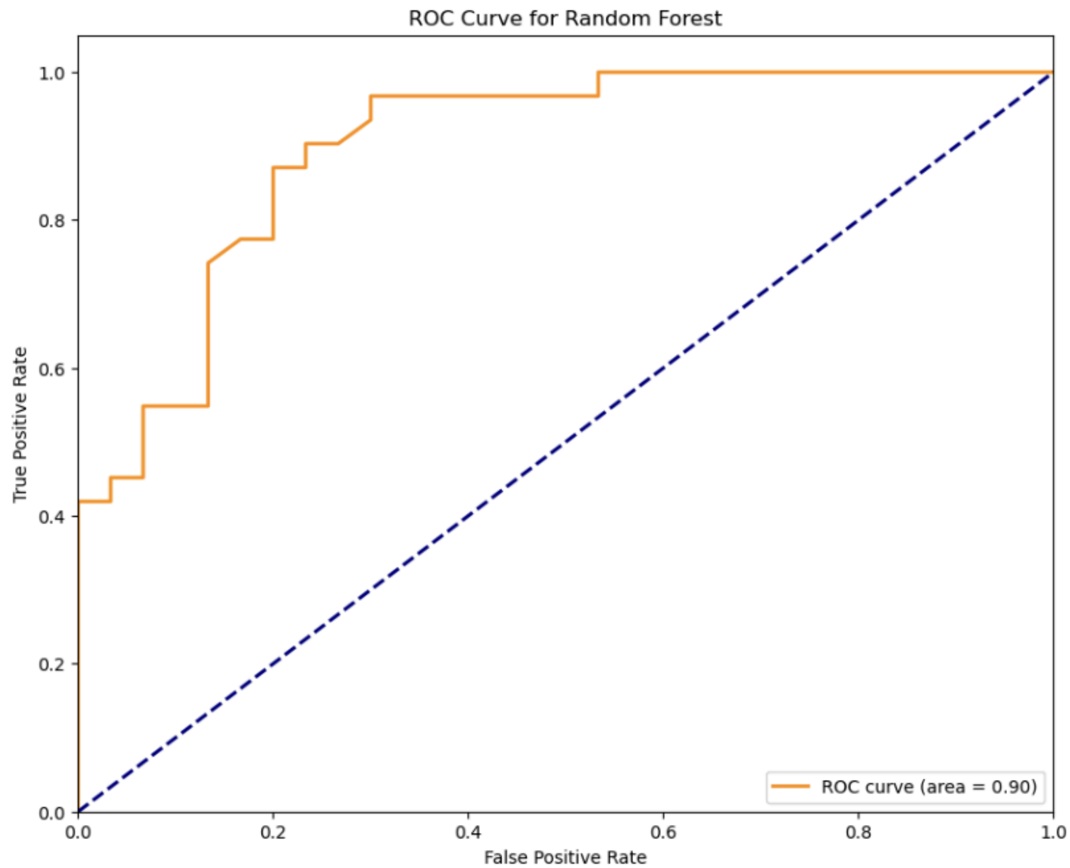
```
Logistic Regression Validation ROC AUC: 0.93
```



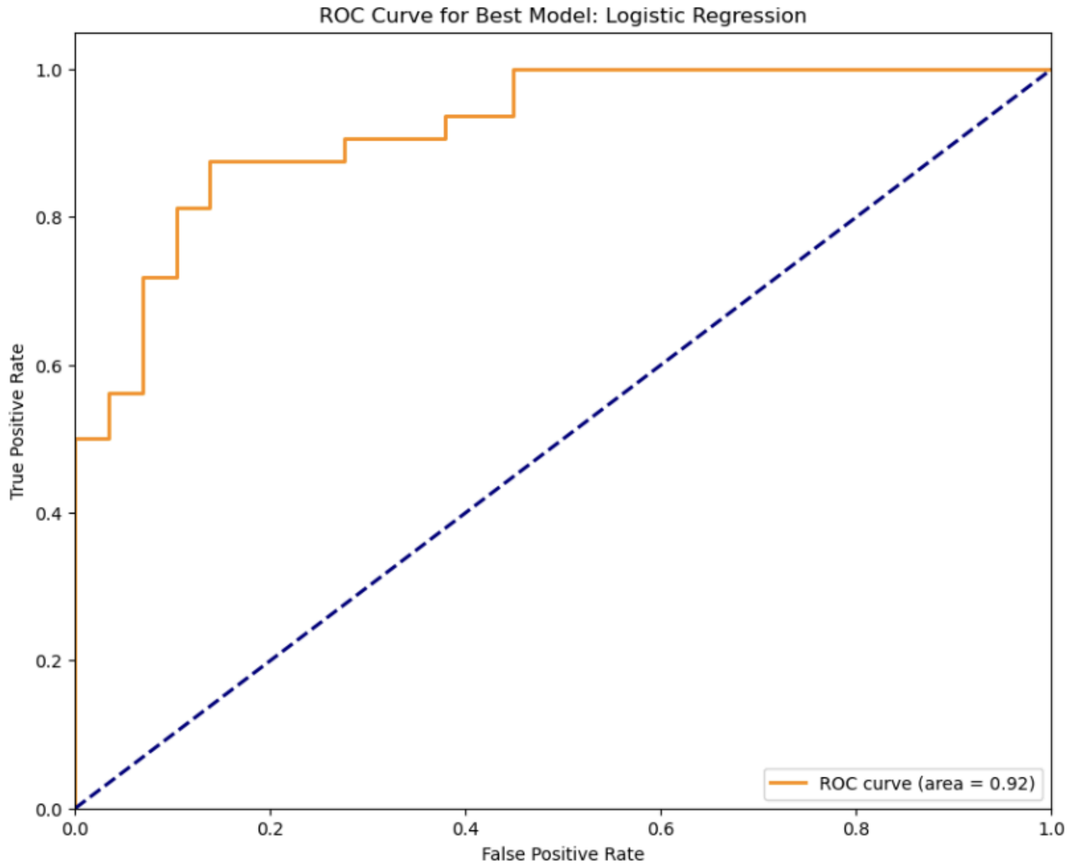

```
Decision Tree Validation ROC AUC: 0.69
```




```
Random Forest Validation ROC AUC: 0.90
```




```
Best Model: Logistic Regression
Test Set Performance:
Accuracy: 0.8360655737704918
ROC AUC: 0.92
              precision    recall  f1-score   support

           0       0.81      0.86      0.83        29
           1       0.87      0.81      0.84        32

    accuracy                           0.84        61
   macro avg       0.84      0.84      0.84        61
weighted avg       0.84      0.84      0.84        61
```


ROC Curve for Best Model: Logistic Regression
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.92)