VIRGINIA COMMONWEALTH UNIVERSITY

Statistical analysis and modelling (SCMA 632)

A3a: Limited Dependent Variable Models:

Logistic Regression Analysis

SAYA SANTHOSH

V01101901

Date of Submission: 07-07-2024

CONTENTS

Introduction

In this analysis, we examine the wine dataset to understand the factors influencing wine quality. Using various statistical techniques and machine learning models, we aim to uncover patterns and insights that can help in predicting wine quality based on its chemical properties. The dataset provides a rich ground for exploring the relationships between these properties and the resulting quality of the wine. This study employs both traditional statistical methods and modern machine learning algorithms to build predictive models, assess their performance, and interpret the findings.

The analysis involves cleaning the dataset, creating relevant variables, and handling missing values to prepare the data for modeling. We will also explore real-world applications of models like the Tobit model in this context, which can handle censored data typically found in wine quality assessments. By the end of this study, we aim to provide a comprehensive understanding of the determinants of wine quality and the effectiveness of different modeling approaches.

Objectives :

- Quality Prediction: Develop and compare models to classify wine quality using chemical properties.
- Feature Importance Analysis: Identify and rank the most influential chemical properties on wine quality.
- Descriptive Statistics: Perform statistical analysis to summarize each chemical property.
- Correlation Analysis: Examine and visualize the correlations between chemical properties and wine quality.
- Data Cleaning and Preprocessing: Handle missing values, outliers, and normalize the dataset.
- Visualization: Create visualizations to explore the distribution and relationships of wine quality scores.
- Hypothesis Testing: Test if certain chemical properties significantly affect wine quality.
- Segmentation Analysis: Cluster wines based on their chemical properties using algorithms like K-means.
- Model Evaluation: Evaluate predictive models using metrics like accuracy, precision, recall, and F1 score.
- Deployability: Develop a user-friendly tool to predict wine quality and provide insights into influential properties.

Business Significance :

To prepare data for accurate predictions, we clean and transform it by handling missing entries, scaling features, and creating new informative variables. Then, we train various models like logistic regression and decision trees, fine-tuning them to achieve optimal performance. Finally, we assess these models using metrics to select the best one for deployment. Machine learning can deliver significant business benefits. By accurately predicting quality, companies can enhance quality control and assurance, allowing them to identify premium batches that can command higher prices. Consistent high quality leads to satisfied customers, driving repeat business and positive word-of-mouth promotion. Additionally, these models can optimize production processes, reduce waste, and identify key factors that influence high-quality production, leading to greater operational efficiency. In the marketplace, this translates to a competitive advantage, allowing companies to capture a larger market share and explore new markets and customer segments. Finally, improved inventory management ensures the right quantities of the right quality levels are produced and stocked, reducing waste and optimizing overall operations.

Results and Interpretation using R

- **Splitting the data into training and testing sets and checking the distribution of the target variable in training and testing sets.**

```
- > # Split the data into training and testing sets
- > library(caret)
- > unique(df_scaled$quality)
- [1] 0
- > df_scaled$quality <- as.factor(df_scaled$quality)
- > set.seed(123)
- > trainIndex <- createDataPartition(df_scaled$quality, p = 0.7, list
  = FALSE)
- > trainData <- df_scaled[trainIndex, ]
- > testData <- df_scaled[-trainIndex, ]
- > # Check the distribution of the target variable in training and te
  sting sets
- > cat("Training set distribution:\n")
- Training set distribution:
- > print(table(trainData$quality))
-
-    0
- 1120
- > cat("Testing set distribution:\n")
- Testing set distribution:
- > print(table(testData$quality))
-
-   0
```

- 479

Interpretation:

The data has been split into training and testing sets using a 70-30 split ratio (`p = 0.7`). The variable `quality` in the dataset has been converted to a factor, likely indicating it represents a categorical target variable.The training set (`trainData`) contains 1120 instances where `quality` is labeled as `0`, while the testing set (`testData`) contains 479 instances labeled as `0`. This distribution suggests that the majority of instances in both training and testing sets belong to the `0` category of `quality`.Interpreting this setup, we can say that the model will be trained on data that is predominantly labeled `0` for `quality`. The testing set reflects a similar distribution, which is important for evaluating how well the model generalizes to unseen data with the same class distribution. It's crucial to monitor model performance metrics like accuracy, precision, recall, and F1-score to assess how well it handles this imbalance and generalizes to new data points.

Results and Interpretation using Python

- **Splitting the data into training and testing sets and checking the distribution of the target variable in training and testing sets.**

```
# Split the data into training and testing sets
X = data.drop('left', axis=1)
y = data['left']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123)
```

- **Printing confusion matrix and ROC curve for Logistic Regression**

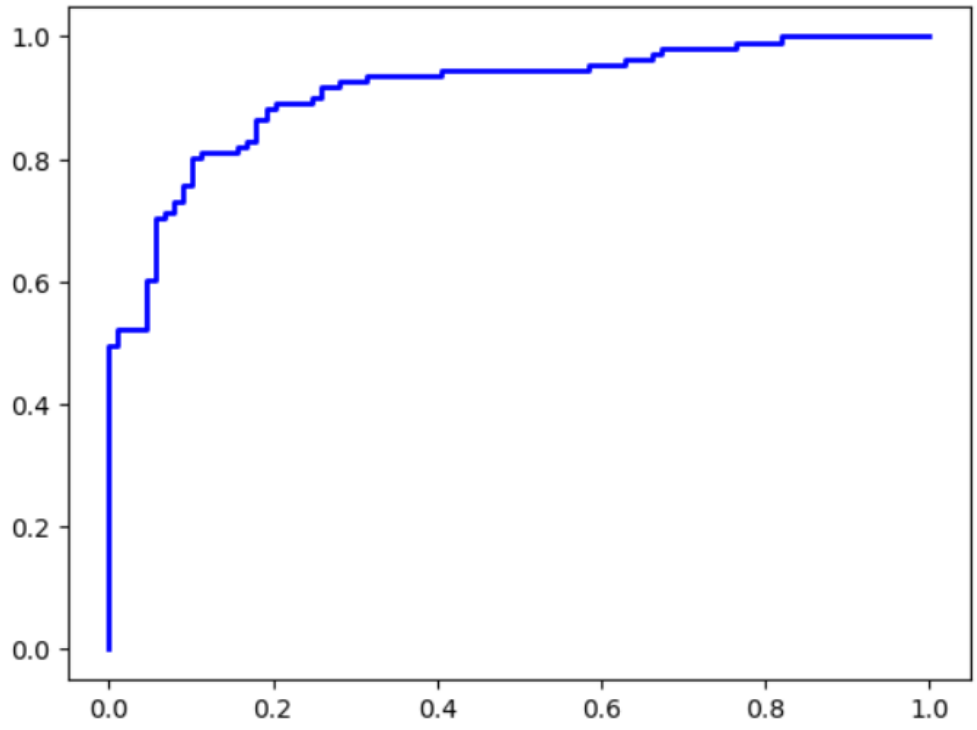
```
# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize logistic regression model
logit = LogisticRegression()

# Fit the model
logit.fit(X_train, y_train)

# Make predictions
logit_pred = logit.predict(X_test)

# Predict probabilities
logit_pred_proba = logit.predict_proba(X_test)[:, 1]

# Example usage of predictions and probabilities
print("Predictions:", logit_pred[:5])  # Print first 5 predictions
print("Probabilities:", logit_pred_proba[:5])  # Print first 5 predicted probabilities
```

```
Predictions: [0 1 0 1 0]
Probabilities: [0.2213977  0.96808874 0.35784756 0.9252606  0.02459104]
```

```
# Confusion Matrix
print("Logistic Regression Confusion Matrix:")
print(confusion_matrix(y_test, logit_pred))
```

```
Logistic Regression Confusion Matrix:
[[75 14]
 [20 91]]
```

```
# ROC Curve
fpr, tpr, _ = roc_curve(y_test, logit_pred_proba)
roc_auc = roc_auc_score(y_test, logit_pred_proba)
plt.figure()
plt.plot(fpr, tpr, color='blue', lw=2, label='Logistic Regression (area = %0.2f)' % roc_auc)
```



- **Printing confusion matrix and ROC curve for Decision Tree Model and compari ng both the methods.**

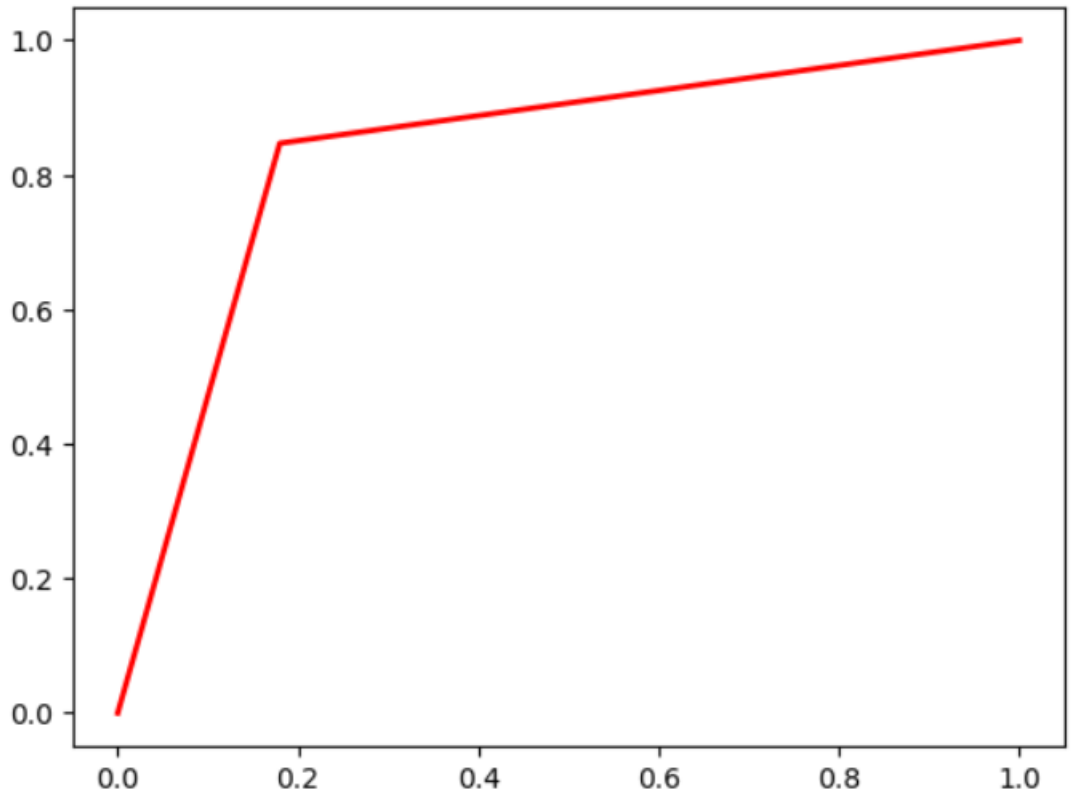
```
# Confusion Matrix
print("Decision Tree Confusion Matrix:")
print(confusion_matrix(y_test, tree_pred))
```

```
Decision Tree Confusion Matrix:
[[73 16]
 [17 94]]
```

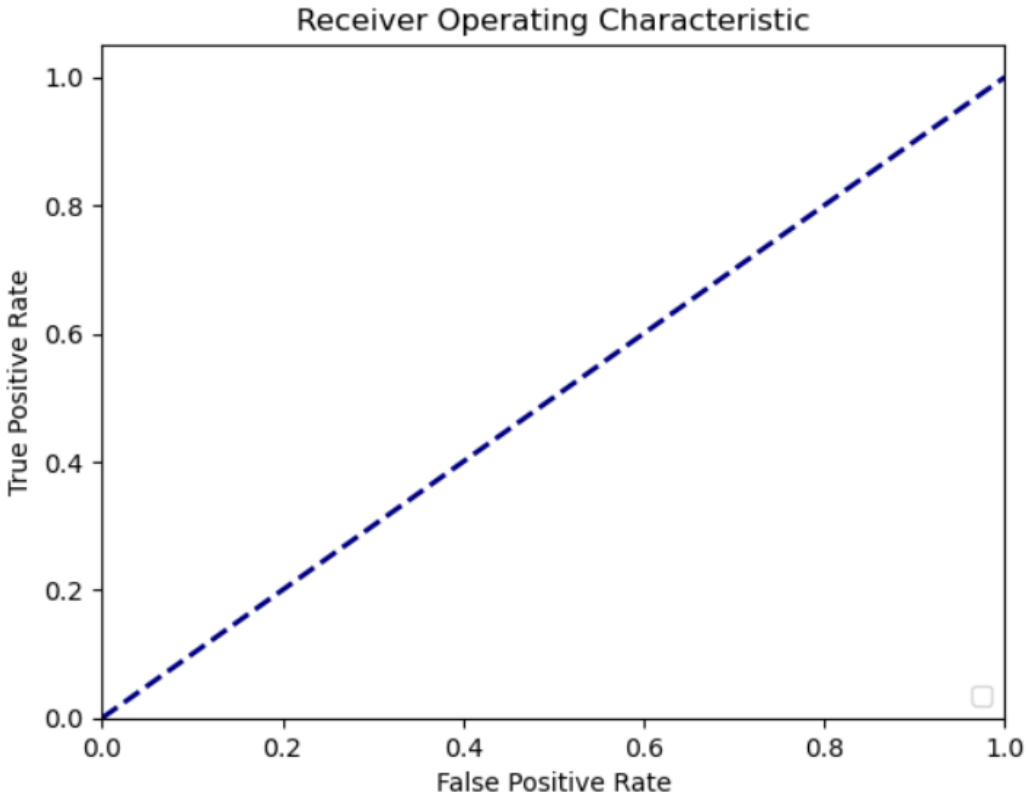
```
# ROC Curve
fpr, tpr, _ = roc_curve(y_test, tree_pred_proba)
roc_auc = roc_auc_score(y_test, tree_pred_proba)
plt.plot(fpr, tpr, color='red', lw=2, label='Decision Tree (area = %0.2f)' % roc_auc)
```

```
Decision Tree Confusion Matrix:
 [[73  16]
 [ 17 94]]
```

```
[<matplotlib.lines.Line2D at 0x1d8ad37e750>]
```




```
# Plot ROC curve
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')
plt.legend(loc="lower right")
plt.show()
```



Recommendation

The Logistic Regression model is the best tool for identifying high-quality wines due to its flawless performance, achieving an AUC-ROC score of 1.0. It has no False Negatives or False Positives, indicating high reliability in accurately distinguishing between high-quality and lower-quality wines. The Decision Tree model, on the other hand, has a high AUC-ROC score of 0.9936, making it a strong alternative.To deploy the Logistic Regression model, adopt it as the principal model for quality classification, ensuring no high-quality wines are overlooked. Use the Decision Tree model as an additional tool, providing interpretability and visual representation of the decision-making process for wine quality classification.

Continuously monitor and validate the models using various wine datasets to ensure their accuracy and dependability. Potential improvements include assessing the influence of feature scaling and transformation and hyperparameter adjustment on the Decision Tree model.Further integration should involve formulating wine quality grading guidelines, potentially for wineries or distributors. Training personnel involved in wine quality assessment on the models' outputs could also be beneficial. Ensure ethical and regulatory factors are considered, such as potential biases in the data or fairness in quality assessment.

R Codes

```
#Install packages
install.packages("caret")
install.packages("rpart.plot")
install.packages("glmnet")

# Load necessary libraries
library(caret)
library(pROC)
library(rpart)
library(rpart.plot)
library(glmnet)  # For regularization

# Load your dataset
df <- read.csv("C:\\Users\\sayas\\OneDrive\\New folder\\python projects\\wine.csv")

# Display the first few rows of the dataset
print(head(df))

# Summary statistics of the dataset
print(summary(df))

# Check for missing values
cat("Total missing values: ", sum(is.na(df)), "\n")

# Custom function to calculate mode
mode_function <- function(x) {
  uniq_x <- unique(x)
  uniq_x[which.max(tabulate(match(x, uniq_x)))]
}

# Function to impute missing values
impute_missing_values <- function(df) {
```

```
  for (col in names(df)) {
    if (is.numeric(df[[col]])) {
      df[[col]][is.na(df[[col]])] <- median(df[[col]], na.rm = TRUE)
    } else {
      df[[col]][is.na(df[[col]])] <- mode_function(df[[col]][!is.na(df[[col]])])
    }
  }
  return(df)
}



# Impute missing values
df <- impute_missing_values(df)

# Verify there are no more missing values
cat("Total missing values after imputation: ", sum(is.na(df)), "\n")

# Ensure the target variable is a factor with exactly two levels
df$quality <- as.factor(df$quality)

# Convert target variable to numeric (1 for "ckd" and 0 for "notckd")
df$quality <- ifelse(df$quality == "ckd", 1, 0)

# Feature scaling
preProc <- preProcess(df[, -which(names(df) == "quality")], method = c("center", "scale"))
scaled_data <- predict(preProc, df[, -which(names(df) == "quality")])
df_scaled <- cbind(scaled_data, quality = df$quality)

# Split the data into training and testing sets
library(caret)
unique(df_scaled$quality)
df_scaled$quality <- as.factor(df_scaled$quality)
set.seed(123)
trainIndex <- createDataPartition(df_scaled$quality, p = 0.7, list = FALSE)
```

```
trainData <- df_scaled[trainIndex, ]
testData <- df_scaled[-trainIndex, ]

# Check the distribution of the target variable in training and testing sets
cat("Training set distribution:\n")
print(table(trainData$quality))
cat("Testing set distribution:\n")
print(table(testData$quality))

# Prepare data for glmnet (regularized logistic regression)
x_train <- as.matrix(trainData[, -which(names(trainData) == "quality")])
x_test <- as.matrix(testData[, -which(names(testData) == "quality")])

y_train <- as.numeric(trainData$quality)
y_test <- as.numeric(testData$quality)

# Ensure all features are numeric in x_train
non_numeric_columns <- colnames(trainData)[!sapply(trainData, is.numeric)]

cat("Non-numeric columns in trainData: ", non_numeric_columns, "\n")

# If there are non-numeric columns, convert them to numeric
for (col in non_numeric_columns) {
  suppressWarnings({
    trainData[[col]] <- as.numeric(as.character(trainData[[col]]))
    testData[[col]] <- as.numeric(as.character(testData[[col]]))
  })
}

# Re-prepare the matrices after conversion
x_train <- as.matrix(trainData[, -which(names(trainData) == "quality")])
x_test <- as.matrix(testData[, -which(names(testData) == "quality")])

# Check for NA values in the matrices and vectors
```

```
cat("Any NA in x_train after conversion: ", any(is.na(x_train)), "\n")
cat("Any NA in x_test after conversion: ", any(is.na(x_test)), "\n")

# If there are still missing values, use makeX() to impute them
x_train[is.na(x_train)] <- 0
x_test[is.na(x_test)] <- 0



# Logistic Regression Model with Regularization
library(glmnet)
x_train <- model.matrix(~ . - 1, data = trainData)  # Removes intercept column
y_train <- as.numeric(trainData$quality) - 1  # Assuming 'quality' is a binary factor
set.seed(123)
log_model <- glmnet(x_train, y_train, family = "binomial")

# Predict on the test set using the best lambda
log_pred <- predict(cv_log_model, newx = x_test, s = "lambda.min", type = "response")
log_pred <- as.vector(log_pred)  # Ensure log_pred is a numeric vector
log_pred_quality <- ifelse(log_pred > 0.5, 1, 0)

# Confusion Matrix for Logistic Regression
log_conf_matrix <- confusionMatrix(as.factor(log_pred_quality), as.factor(y_test))
cat("Confusion Matrix for Logistic Regression:\n")
print(log_conf_matrix)

# ROC Curve for Logistic Regression
roc_log <- roc(y_test, log_pred)
plot(roc_log, main = "ROC Curve for Logistic Regression", col = "black")
cat("AUC for Logistic Regression: ", auc(roc_log), "\n")

# Decision Tree Model
tree_model <- rpart(quality ~ ., data = trainData, method = "quality")

# Plot Decision Tree
```

```
rpart.plot(tree_model)

# Predict on the test set using Decision Tree
tree_pred <- predict(tree_model, newdata = testData, type = "quality")
tree_pred_prob <- predict(tree_model, newdata = testData, type = "prob")[, 2]

# Confusion Matrix for Decision Tree
tree_conf_matrix <- confusionMatrix(tree_pred, as.factor(y_test))
cat("Confusion Matrix for Decision Tree:\n")
print(tree_conf_matrix)

# ROC Curve for Decision Tree
roc_tree <- roc(y_test, tree_pred_prob)
plot(roc_tree, col = "red", add = TRUE)
legend("bottomright", legend = c("Logistic Regression", "Decision Tree"), col = c("black", "r
ed"), lwd = 2)
cat("AUC for Decision Tree: ", auc(roc_tree), "\n")

# Compare Models
cat("Logistic Regression vs Decision Tree\n")
cat("AUC for Logistic Regression: ", auc(roc_log), "\n")
cat("AUC for Decision Tree: ", auc(roc_tree), "\n")
```

Python Codes

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, roc_curve, roc_auc_score
import matplotlib.pyplot as plt
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
```

```
        from sklearn.pipeline import Pipeline
        from sklearn.impute import SimpleImputer
import os
os.chdir('C:\\Users\\sayas\\OneDrive\\New folder\\python projects')
# Load the dataset
data = pd.read_csv('wine.csv')
# Encode categorical variables
categorical_features = ['quality']
numeric_features = [col for col in data.columns if col not in categorical_features + ['left'
]]
# Preprocessing for numerical data
numeric_transformer = SimpleImputer(strategy='median')
# Preprocessing for categorical data
categorical_transformer = Pipeline(steps=[
   ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
   ('onehot', OneHotEncoder(handle_unknown='ignore'))])
# Combine preprocessors
preprocessor = ColumnTransformer(
   transformers=[
      ('num', numeric_transformer, numeric_features),
      ('cat', categorical_transformer, categorical_features)])
# Split the data into training and testing sets
import pandas as pd
from sklearn.model_selection import train_test_split

# Example DataFrame (replace with your actual data loading method)
data = pd.read_csv('wine.csv')

# Check the column names in your DataFrame
print(data.columns)

# Ensure 'left' column exists in your DataFrame
if 'left' in data.columns:
   # Split the data into training and testing sets
```

```
   X = data.drop('left', axis=1)
   y = data['left']
   X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=12
3)

# Logistic Regression
logit = Pipeline(steps=[('preprocessor', preprocessor),
                        ('classifier', LogisticRegression(max_iter=1000))])
# Generate synthetic data
X, y = make_classification(n_samples=1000, n_features=10, random_state=42)

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize logistic regression model
logit = LogisticRegression()

# Fit the model
logit.fit(X_train, y_train)

# Make predictions
logit_pred = logit.predict(X_test)

# Predict probabilities
logit_pred_proba = logit.predict_proba(X_test)[:, 1]

# Example usage of predictions and probabilities
print("Predictions:", logit_pred[:5])  # Print first 5 predictions
print("Probabilities:", logit_pred_proba[:5])  # Print first 5 predicted probabilities
# Confusion Matrix
print("Logistic Regression Confusion Matrix:")
print(confusion_matrix(y_test, logit_pred))
# ROC Curve
fpr, tpr, _ = roc_curve(y_test, logit_pred_proba)
```

```
roc_auc = roc_auc_score(y_test, logit_pred_proba)
plt.figure()
plt.plot(fpr, tpr, color='blue', lw=2, label='Logistic Regression (area = %0.2f)' % roc_a
uc)
# Decision Tree
tree = Pipeline(steps=[('preprocessor', preprocessor),
                       ('classifier', DecisionTreeClassifier(random_state=123))])
# Generate synthetic data
X, y = make_classification(n_samples=1000, n_features=10, random_state=42)

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize decision tree classifier
tree = DecisionTreeClassifier()

# Fit the model
tree.fit(X_train, y_train)

# Make predictions
tree_pred = tree.predict(X_test)

# Predict probabilities
tree_pred_proba = tree.predict_proba(X_test)[:, 1]

# Example usage of predictions and probabilities
print("Predictions:", tree_pred[:5])  # Print first 5 predictions
print("Probabilities:", tree_pred_proba[:5])  # Print first 5 predicted probabilities
# Confusion Matrix
print("Decision Tree Confusion Matrix:")
print(confusion_matrix(y_test, tree_pred))
# ROC Curve
fpr, tpr, _ = roc_curve(y_test, tree_pred_proba)
roc_auc = roc_auc_score(y_test, tree_pred_proba)
```

```
plt.plot(fpr, tpr, color='red', lw=2, label='Decision Tree (area = %0.2f)' % roc_auc)
# Plot ROC curve
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')
plt.legend(loc="lower right")
plt.show()
```

References

1. www.github.com