gradient-boosting

January 23, 2025

[3]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier,␣
 ↪GradientBoostingRegressor
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report, mean_squared_error, r2_score
import seaborn as sns
import matplotlib.pyplot as plt
```

[4]:
```
# Load the Titanic dataset
# Replace 'path_to_titanic_data.csv' with your dataset's file path
data = pd.read_csv(r"C:\Users\91703\OneDrive\Desktop\TITANIC.csv")
```

[7]:
```
# Display the first few rows of the dataset
data.head()
```

[7]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th…	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

[9]:
```
# Data preprocessing
# Fill missing values and assign back to the DataFrame
data["Age"] = data["Age"].fillna(data["Age"].mean())
data["Embarked"] = data["Embarked"].fillna(data["Embarked"].mode()[0])
data["Fare"] = data["Fare"].fillna(data["Fare"].mean())

# Convert categorical columns to numeric
data["Sex"] = data["Sex"].map({"male": 0, "female": 1})
data["Embarked"] = data["Embarked"].map({"C": 0, "Q": 1, "S": 2})
```

[10]:
```
# Classification: Predicting survival
# Features and target variable
X_classification = data[["Pclass", "Sex", "Age", "Fare", "SibSp", "Parch",␣
 ↪"Embarked"]]
y_classification = data["Survived"]
```

[11]:
```
# Split data into training and testing sets
X_train_cls, X_test_cls, y_train_cls, y_test_cls = train_test_split(
    X_classification, y_classification, test_size=0.2, random_state=42
)
```

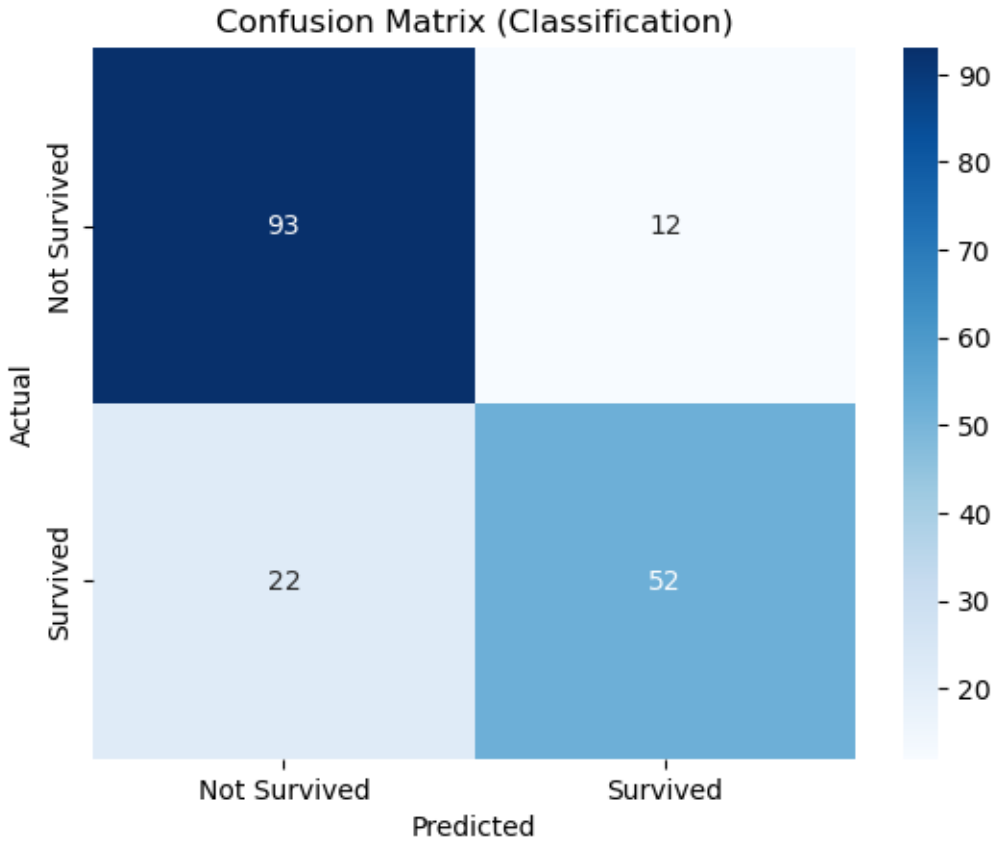
[12]:
```
# Initialize the Gradient Boosting Classifier
clf_model = GradientBoostingClassifier(random_state=42)
clf_model.fit(X_train_cls, y_train_cls)
```

[12]:
```
GradientBoostingClassifier(random_state=42)
```

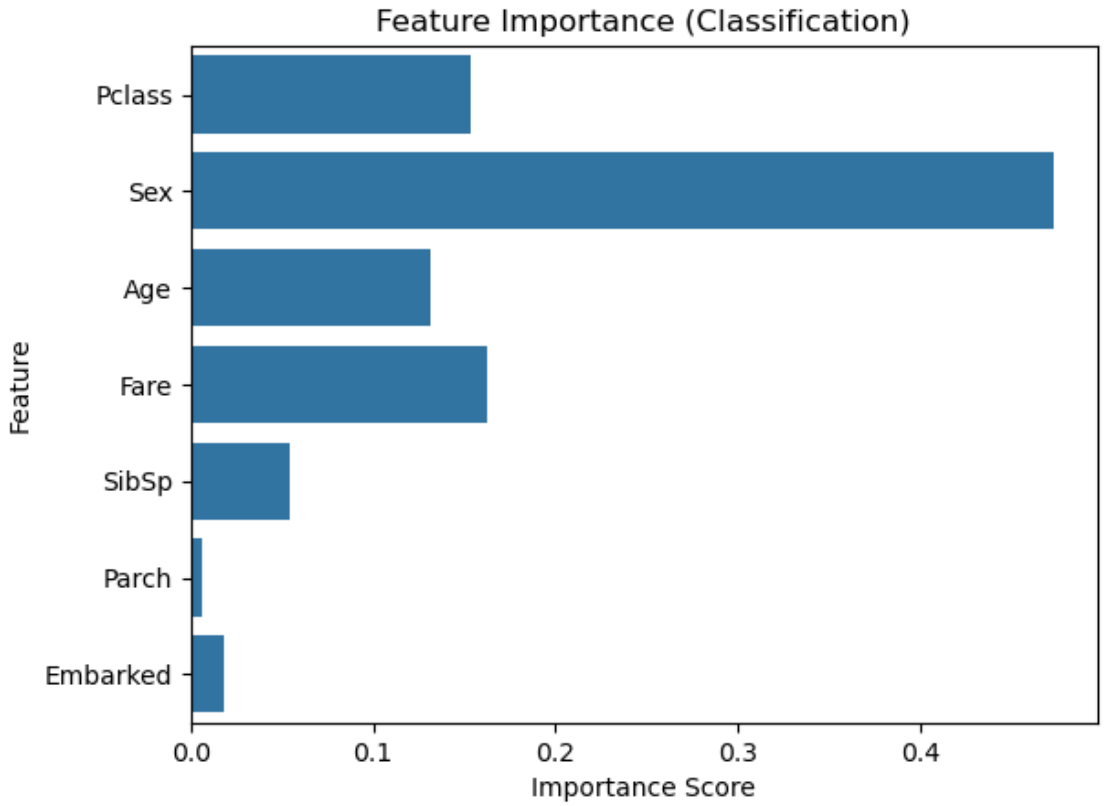
[13]:
```
# Predictions and Evaluation
y_pred_cls = clf_model.predict(X_test_cls)
clf_accuracy = accuracy_score(y_test_cls, y_pred_cls)
print("\nClassification Accuracy:", clf_accuracy)
print("\nClassification Report:\n", classification_report(y_test_cls,␣
 ↪y_pred_cls))
```

```
Classification Accuracy: 0.8100558659217877

Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.89      0.85       105
           1       0.81      0.70      0.75        74

    accuracy                           0.81       179
   macro avg       0.81      0.79      0.80       179
weighted avg       0.81      0.81      0.81       179
```

[14]:
```
# Confusion Matrix
cm = confusion_matrix(y_test_cls, y_pred_cls)
sns.heatmap(cm, annot=True, fmt='d', cmap="Blues", xticklabels=["Not Survived",␣
↪"Survived"], yticklabels=["Not Survived", "Survived"])
plt.title("Confusion Matrix (Classification)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



[15]:
```
# Feature Importance for Classification
clf_importances = clf_model.feature_importances_
sns.barplot(x=clf_importances, y=X_classification.columns)
plt.title("Feature Importance (Classification)")
plt.xlabel("Importance Score")
plt.ylabel("Feature")
plt.show()
```



[16]:
```
# Regression: Predicting fare
# Features and target variable
X_regression = data[["Pclass", "Sex", "Age", "SibSp", "Parch", "Embarked"]]
y_regression = data["Fare"]
```

[17]:
```
# Split data into training and testing sets
X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(
    X_regression, y_regression, test_size=0.2, random_state=42
)
```

[18]:
```
# Initialize the Gradient Boosting Regressor
reg_model = GradientBoostingRegressor(random_state=42)
reg_model.fit(X_train_reg, y_train_reg)
```

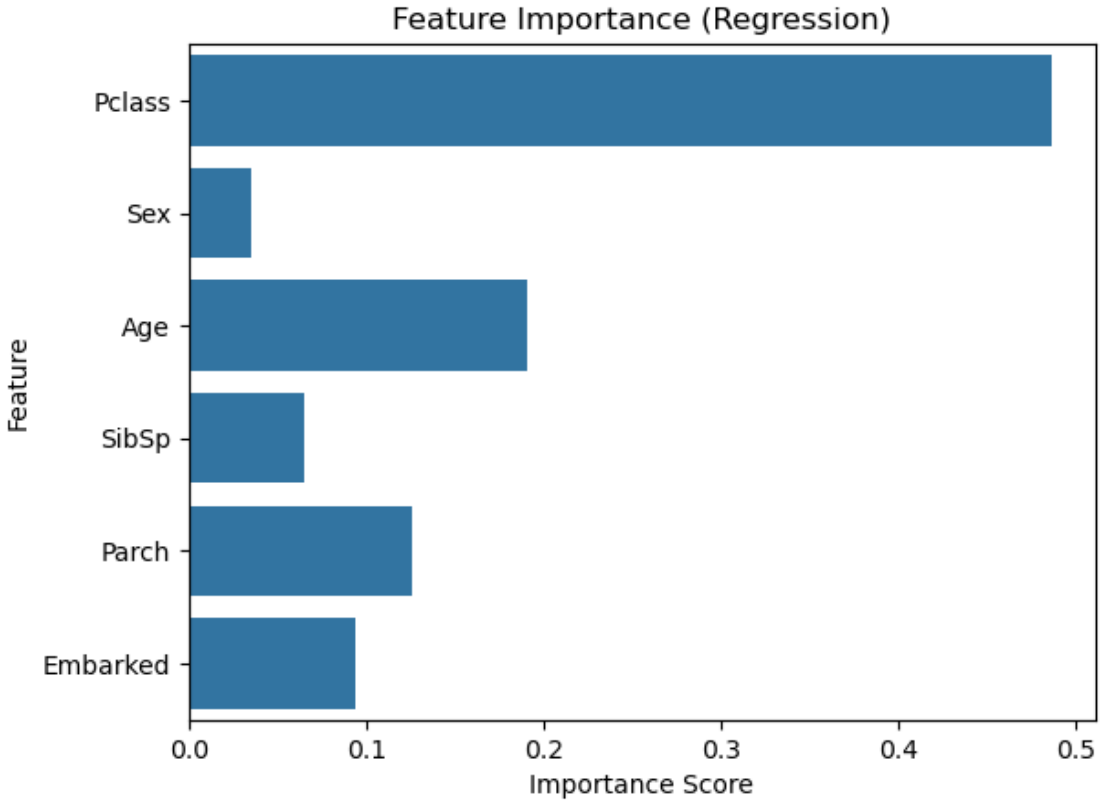
[18]:
```
GradientBoostingRegressor(random_state=42)
```

[19]:
```
# Predictions and Evaluation
y_pred_reg = reg_model.predict(X_test_reg)
mse = mean_squared_error(y_test_reg, y_pred_reg)
r2 = r2_score(y_test_reg, y_pred_reg)
print("\nRegression Mean Squared Error:", mse)
```

```
print("\nRegression R-squared Score:", r2)
```

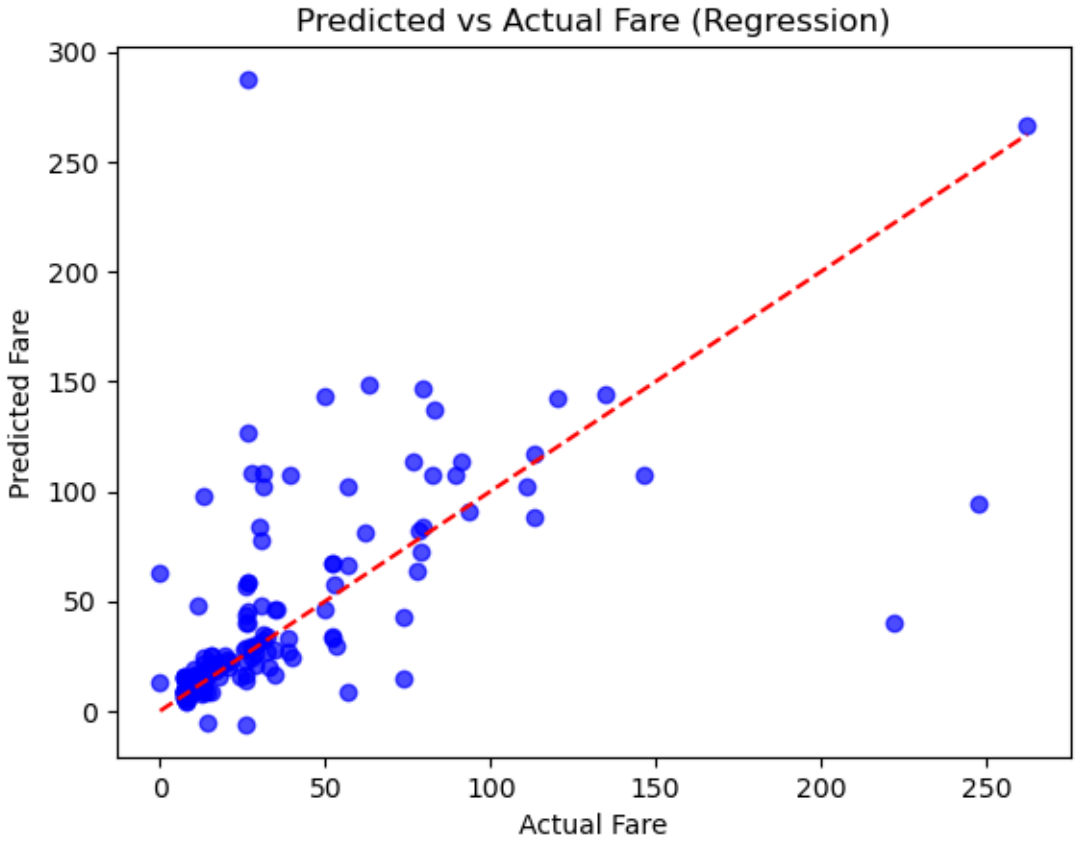
```
Regression Mean Squared Error: 1250.7616592332615

Regression R-squared Score: 0.19171660957802783
```

[20]:
```
# Feature Importance for Regression
reg_importances = reg_model.feature_importances_
sns.barplot(x=reg_importances, y=X_regression.columns)
plt.title("Feature Importance (Regression)")
plt.xlabel("Importance Score")
plt.ylabel("Feature")
plt.show()
```

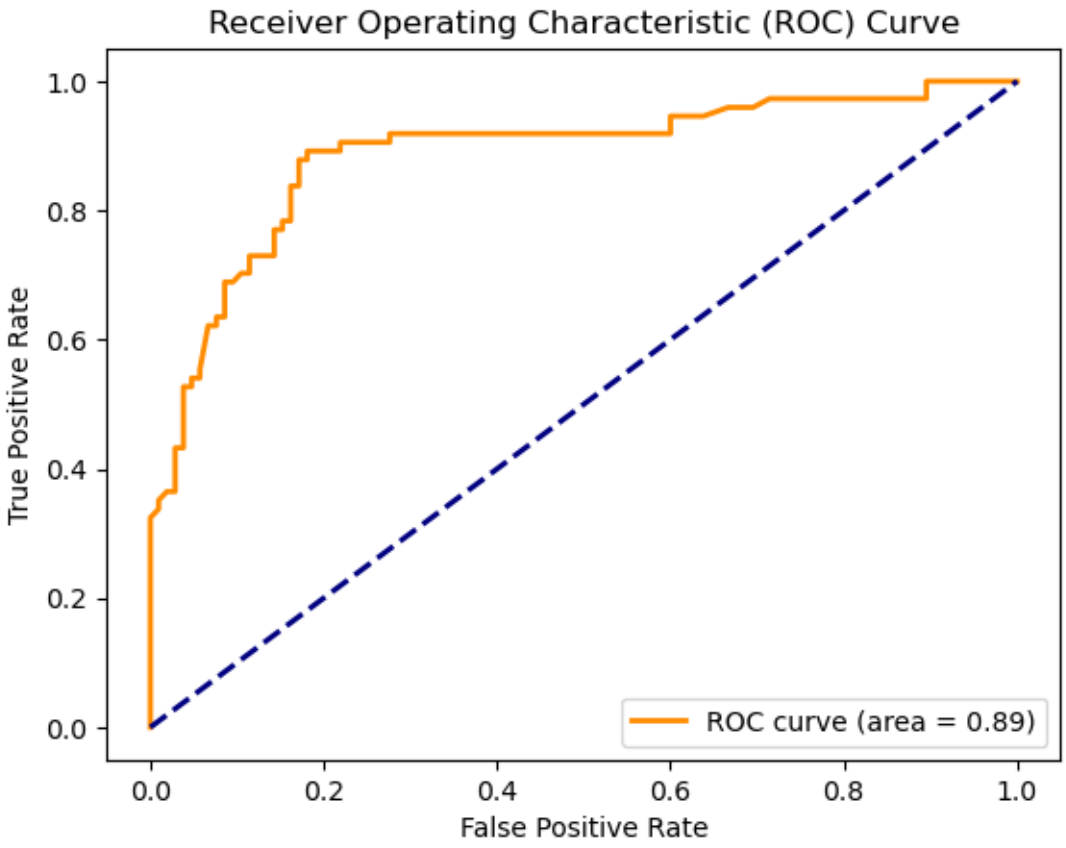


[21]:
```
# Plot Predicted vs Actual Fare
plt.scatter(y_test_reg, y_pred_reg, alpha=0.7, color="blue")
plt.plot([0, max(y_test_reg)], [0, max(y_test_reg)], color="red",␣
 ↪linestyle="--")
plt.title("Predicted vs Actual Fare (Regression)")
plt.xlabel("Actual Fare")
```

```
plt.ylabel("Predicted Fare")
plt.show()
```



[22]:
```
# Classification ROC Curve
from sklearn.metrics import roc_curve, auc

y_pred_prob_cls = clf_model.predict_proba(X_test_cls)[:, 1]
fpr_cls, tpr_cls, _ = roc_curve(y_test_cls, y_pred_prob_cls)
roc_auc_cls = auc(fpr_cls, tpr_cls)

plt.figure()
plt.plot(fpr_cls, tpr_cls, color="darkorange", lw=2, label=f"ROC curve (area =␣
 ↪{roc_auc_cls:.2f})")
plt.plot([0, 1], [0, 1], color="navy", lw=2, linestyle="--")
plt.title("Receiver Operating Characteristic (ROC) Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(loc="lower right")
plt.show()
```


Receiver Operating Characteristic (ROC) Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.89)