Jupyter Classification with Logistic Regression_Task4 Last Checkpoint: 18 hours ago

File Edit View Run Kernel Settings Help

Code

JupyterLab Python 3 (ipykernel) Trusted

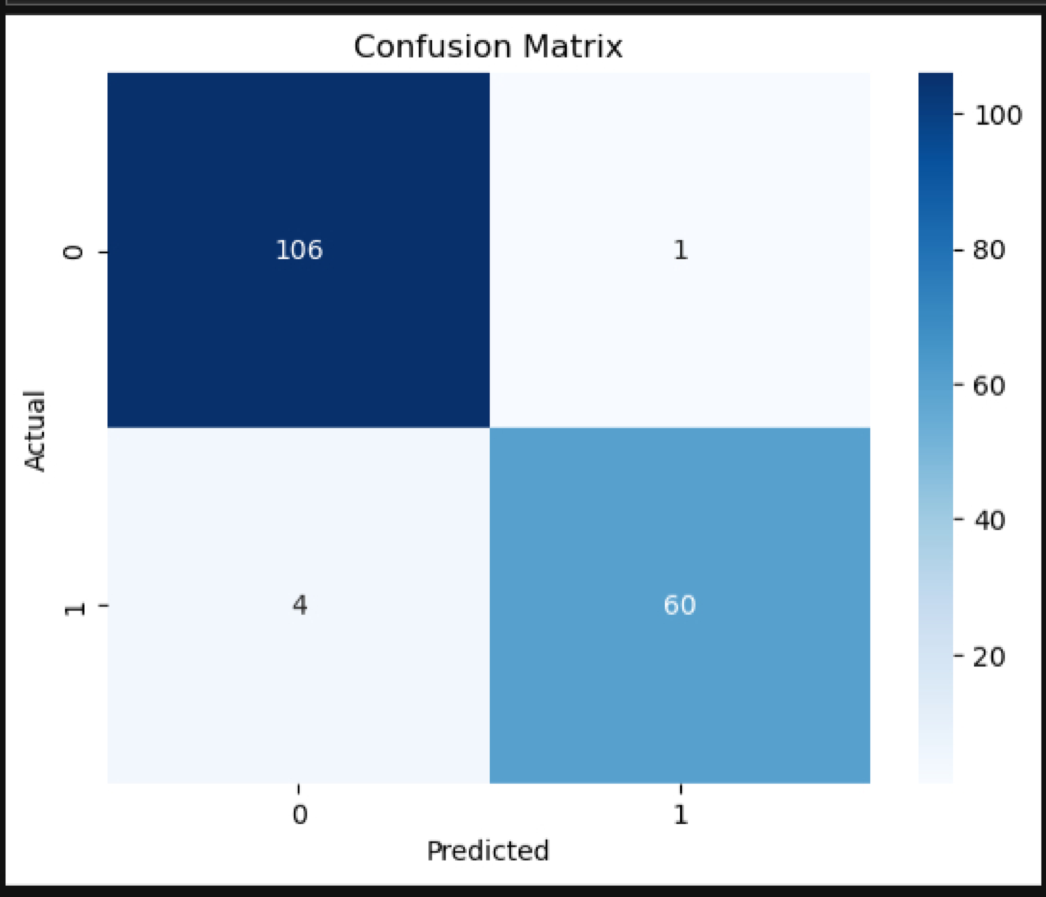
[28]:
```python
# logistic_regression_model.py

# 1️⃣ Import required libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    confusion_matrix,
    classification_report,
    roc_auc_score,
    roc_curve,
    accuracy_score,
    precision_score,
    recall_score
)
import matplotlib.pyplot as plt
import seaborn as sns

# 2️⃣ Load dataset
data = pd.read_csv(r"D:\mydata\Elevate Labs\task4\data.csv", sep=',', encoding='utf-8')

# Basic dataset info
print("\nMissing Values:\n", data.isnull().sum())
print("\nData Info:")
print(data.info())
print("\nDataset Shape:", data.shape)
print("\nFirst 5 rows:\n", data.head())

# 3️⃣ Drop unwanted columns and handle missing values
# Do NOT drop all rows blindly; only drop the NaN column
data = data.drop(['Unnamed: 32'], axis=1, errors='ignore')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
```

[29]:
```python
# Define features and labels
X = data.drop(['id', 'diagnosis'], axis=1, errors='ignore')
y = data['diagnosis'].map({'M': 1, 'B': 0})  # Encode malignant as 1, benign as 0

# Confirm shapes
print("\nX shape:", X.shape)
print("y shape:", y.shape)
print("\nValue Counts:\n", y.value_counts())

# Verify no missing values
print("\nAny NaN in X?", X.isna().sum().sum() > 0)
print("Any NaN in y?", y.isna().sum() > 0)

# 4️⃣ Split the data safely
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y   # stratify preserves class balance
)

print("\nTrain size:", X_train.shape, "Test size:", X_test.shape)
```

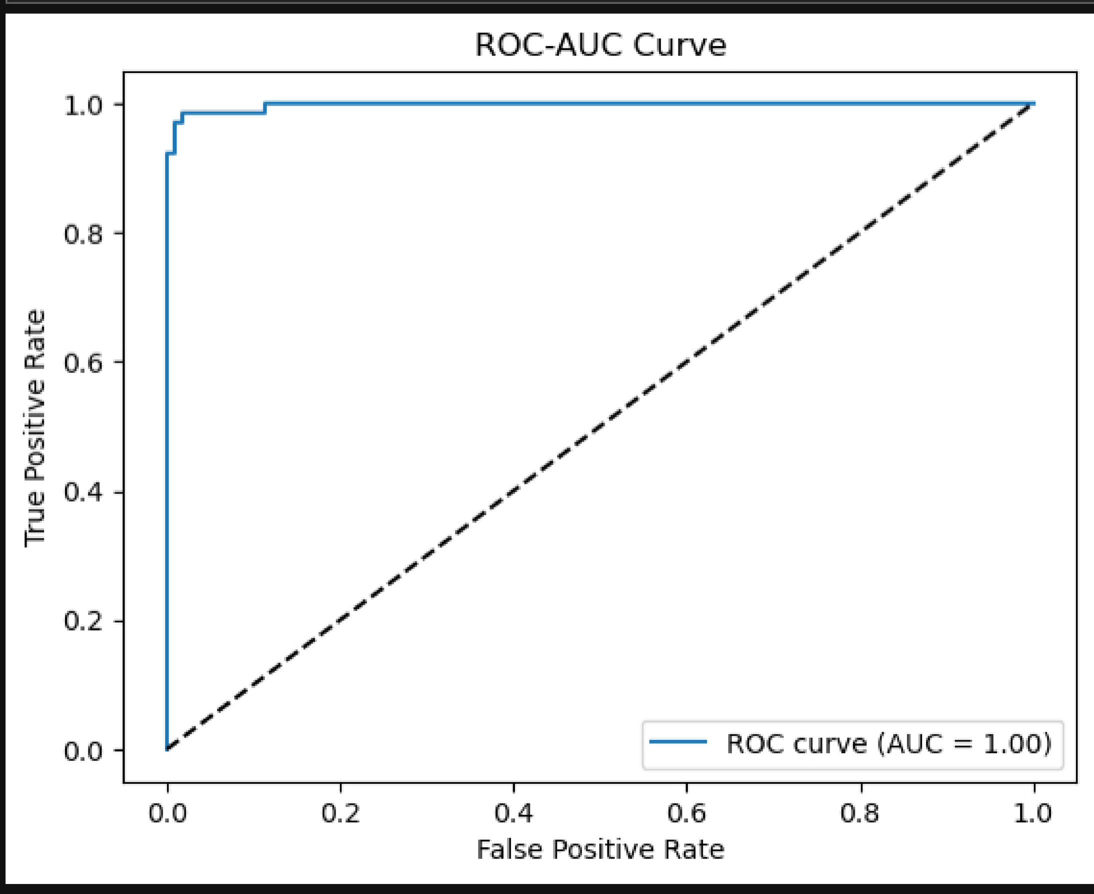
```
X shape: (569, 30)
y shape: (569,)

Value Counts:
 diagnosis
0    357
1    212
Name: count, dtype: int64

Any NaN in X? False
Any NaN in y? False

Train size: (398, 30) Test size: (171, 30)
```

[30]:
```python
# 5️⃣ Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 6️⃣ Train the Logistic Regression model
model = LogisticRegression(random_state=42, max_iter=1000)
model.fit(X_train_scaled, y_train)

# 7️⃣ Model predictions
y_pred = model.predict(X_test_scaled)
y_pred_prob = model.predict_proba(X_test_scaled)[:, 1]

# 8️⃣ Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# 9️⃣ Classification Report
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```




```
Classification Report:
               precision    recall  f1-score   support

           0       0.96      0.99      0.98       107
           1       0.98      0.94      0.96        64

    accuracy                           0.97       171
   macro avg       0.97      0.96      0.97       171
weighted avg       0.97      0.97      0.97       171
```

[31]:
```python
# 1🔟 ROC-AUC Curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
roc_auc = roc_auc_score(y_test, y_pred_prob)

plt.plot(fpr, tpr, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC-AUC Curve")
plt.legend()
plt.show()

# 1️⃣1️⃣ Threshold tuning example
threshold = 0.6
y_pred_custom = (y_pred_prob >= threshold).astype(int)
print(f"\nPerformance at threshold {threshold}:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_custom):.3f}")
print(f"Precision: {precision_score(y_test, y_pred_custom):.3f}")
print(f"Recall: {recall_score(y_test, y_pred_custom):.3f}")
```




```
Performance at threshold 0.6:
Accuracy: 0.971
Precision: 1.000
Recall: 0.922
```