Logistic_Regression

May 29, 2026

1 LOGISTIC REGRESSION - Breast_Cancer dataset from SK Learn

1.0.1 Section 1: Imports

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report
```

1.0.2 Section 2: Load Data

[3]:
```
data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = data.target

print("Shape:", X.shape)
print("Target classes:", np.unique(y))
print("Target names:", data.target_names)
```

```
Shape: (569, 30)
Target classes: [0 1]
Target names: ['malignant' 'benign']
```

1.0.3 Section 3: Split and Scale

[4]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42, stratify=y)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

1.0.4 Section 4: Train Logistic Model

[5]:
```
log_model = LogisticRegression(max_iter=5000, random_state=42)
log_model.fit(X_train_scaled, y_train)
```

[5]:
```
LogisticRegression(max_iter=5000, random_state=42)
```

1.0.5 Section 5: Predict and Evaluate

[6]:
```
y_pred = log_model.predict(X_test_scaled)

y_prob = log_model.predict_proba(X_test_scaled)[:, 1]
acc = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)

print("Accuracy:", round(acc, 4))
print("Confusion Matrix:\n", cm)
print("Classification Report:\n", classification_report(y_test, y_pred,␣
 ↪target_names=data.target_names))
```

```
Accuracy: 0.9825
Confusion Matrix:
 [[41  1]
 [ 1 71]]
Classification Report:
              precision    recall  f1-score   support

   malignant       0.98      0.98      0.98        42
      benign       0.99      0.99      0.99        72

    accuracy                           0.98       114
   macro avg       0.98      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114
```

1.0.6 Section 6: Threshold Comparison

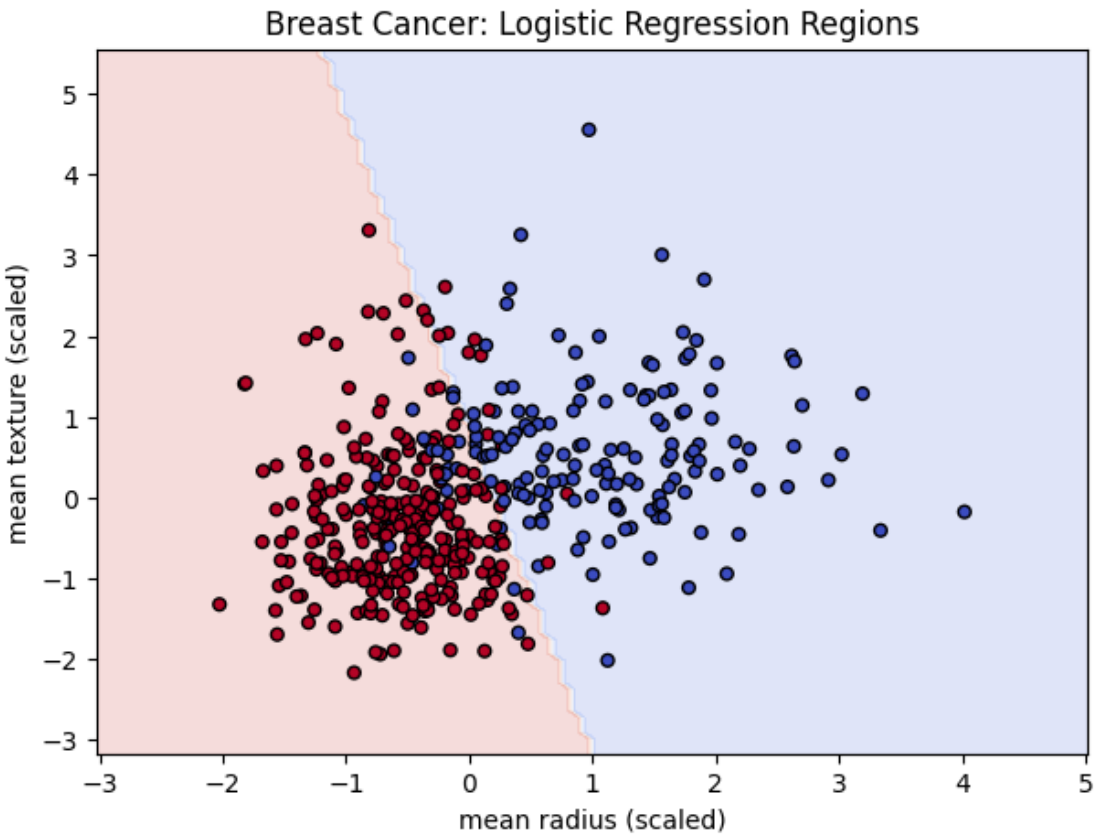
[7]:
```
for t in [0.30, 0.50, 0.70]:
    pred_t = (y_prob >= t).astype(int)
    print(f"threshold={t}: accuracy={accuracy_score(y_test, pred_t):.4f}")
    print(confusion_matrix(y_test, pred_t))
```

```
threshold=0.3: accuracy=0.9825
[[40  2]
 [ 0 72]]
threshold=0.5: accuracy=0.9825
[[41  1]
 [ 1 71]]
threshold=0.7: accuracy=0.9474
```

```
[[41  1]
 [ 5 67]]
```

1.0.7 Section 7: Visualizing Classification Regions (2 Features)

[15]:
```
from sklearn.inspection import DecisionBoundaryDisplay

feature_names = list(data.feature_names)

# Select two informative features by name (safe matching)
def find_feature_index(name, all_names):
    key = name.strip().lower().replace("_", " ")
    norm = [n.strip().lower().replace("_", " ") for n in all_names]
    if key in norm:
        return norm.index(key)
    raise ValueError(f"Feature '{name}' not found.")

ix = find_feature_index("mean radius", feature_names)
iy = find_feature_index("mean texture", feature_names)

X2 = data.data[:, [ix, iy]]
y2 = data.target

X2_train, X2_test, y2_train, y2_test = train_test_split(
    X2, y2, test_size=0.2, random_state=42, stratify=y2
)

scaler2 = StandardScaler()
X2_train_scaled = scaler2.fit_transform(X2_train)

log2 = LogisticRegression(max_iter=5000, random_state=42)
log2.fit(X2_train_scaled, y2_train)

fig, ax = plt.subplots(figsize=(7, 5))
DecisionBoundaryDisplay.from_estimator(
    log2, X2_train_scaled, response_method="predict", cmap="coolwarm", alpha=0.
 ↪18, ax=ax
)
ax.scatter(X2_train_scaled[:, 0], X2_train_scaled[:, 1], c=y2_train,␣
 ↪cmap="coolwarm", s=24, edgecolor="k")
ax.set_xlabel(feature_names[ix] + " (scaled)")
ax.set_ylabel(feature_names[iy] + " (scaled)")
ax.set_title("Breast Cancer: Logistic Regression Regions")
plt.show()
```

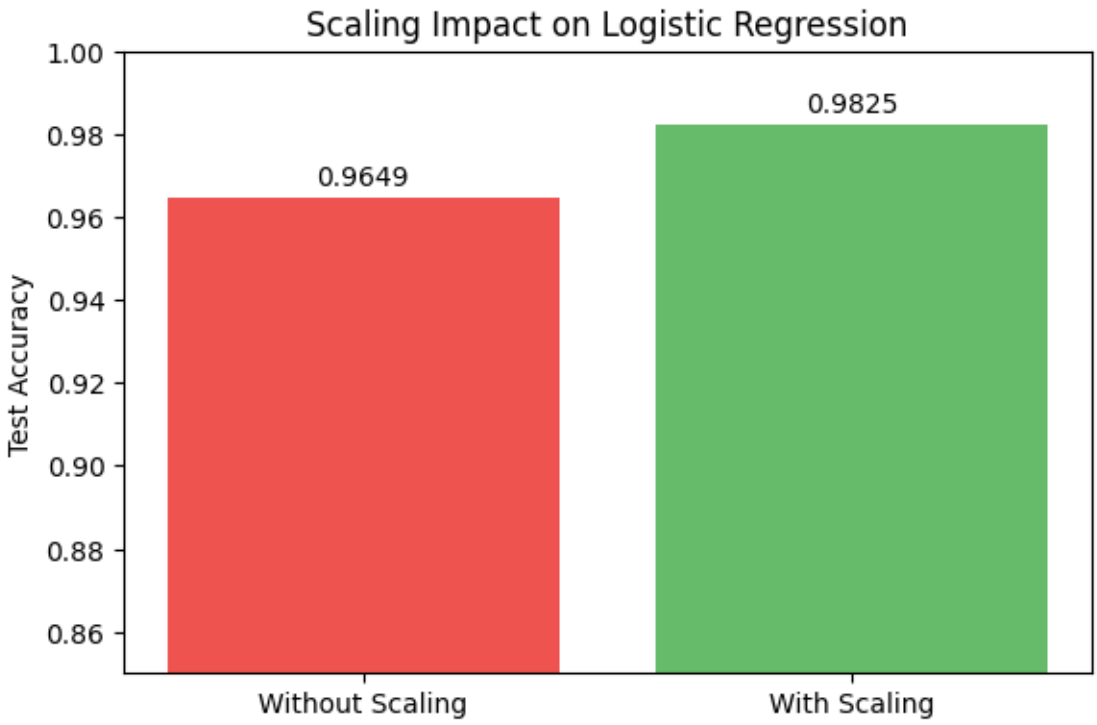


2 GRAPHS

2.0.1 Graph 1: Why Scaling Matters (Logistic Regression)

[16]:
```
from sklearn.pipeline import make_pipeline

# Without scaling
raw_model = LogisticRegression(max_iter=5000, random_state=42)
raw_model.fit(X_train, y_train)
raw_pred = raw_model.predict(X_test)
raw_acc = accuracy_score(y_test, raw_pred)

# With scaling
scaled_model = make_pipeline(StandardScaler(),␣
 ↪LogisticRegression(max_iter=5000, random_state=42))
scaled_model.fit(X_train, y_train)
scaled_pred = scaled_model.predict(X_test)
scaled_acc = accuracy_score(y_test, scaled_pred)

plt.figure(figsize=(6.5, 4.2))
```

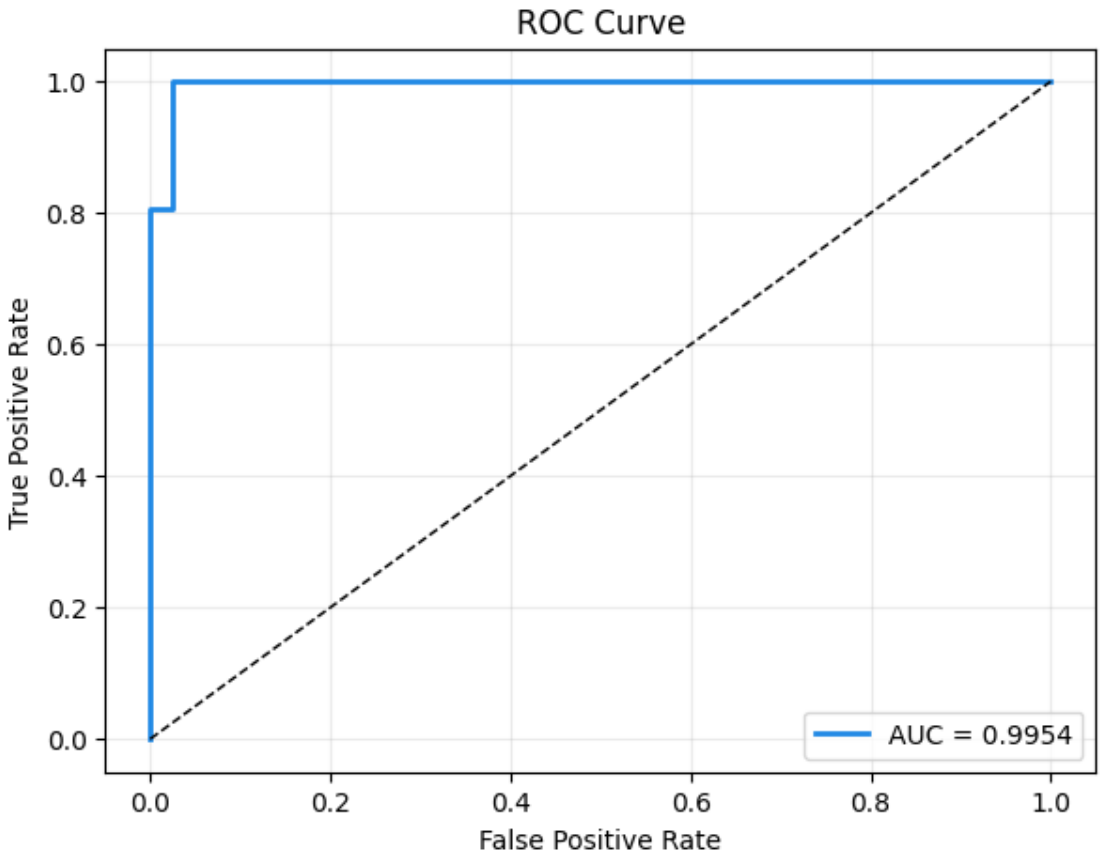
```
plt.bar(["Without Scaling", "With Scaling"], [raw_acc, scaled_acc],␣
 ↪color=["#ef5350", "#66bb6a"])
plt.ylim(0.85, 1.00)
plt.ylabel("Test Accuracy")
plt.title("Scaling Impact on Logistic Regression")
for i, v in enumerate([raw_acc, scaled_acc]):
    plt.text(i, v + 0.003, f"{v:.4f}", ha="center")
plt.show()
```



2.0.2 Graph 2: ROC Curve and AUC

[10]:
```
from sklearn.metrics import roc_curve, auc

fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6.8, 5))
plt.plot(fpr, tpr, color="#1e88e5", linewidth=2, label=f"AUC = {roc_auc:.4f}")
plt.plot([0, 1], [0, 1], "k--", linewidth=1)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend(loc="lower right")
```

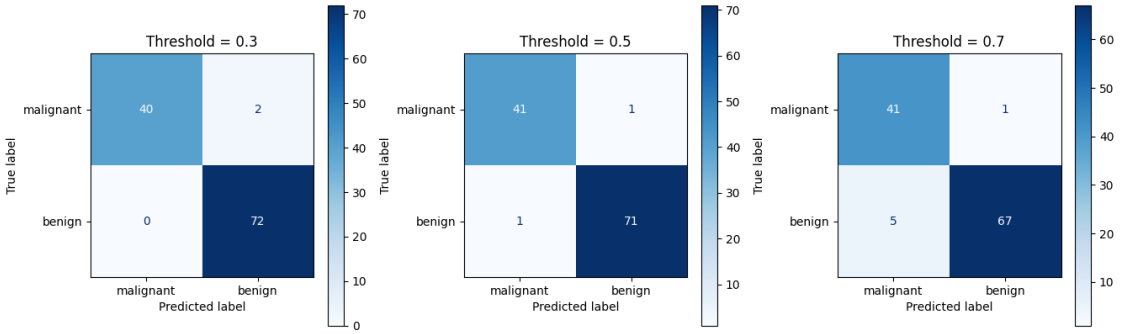
```
plt.grid(alpha=0.25)
plt.show()
```



2.0.3 Graph 3: Confusion Matrices at Different Thresholds

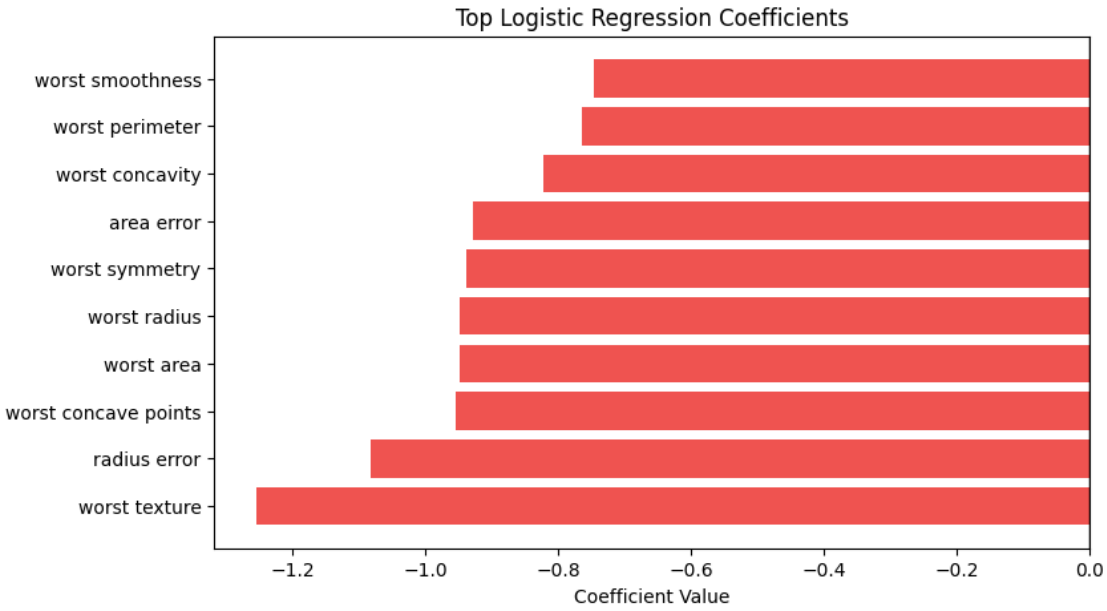
[11]:
```
from sklearn.metrics import ConfusionMatrixDisplay

thresholds_to_plot = [0.30, 0.50, 0.70]
fig, axes = plt.subplots(1, 3, figsize=(13.5, 4.2))

for i, t in enumerate(thresholds_to_plot):
    pred_t = (y_prob >= t).astype(int)
    ConfusionMatrixDisplay.from_predictions(
        y_test, pred_t, display_labels=data.target_names, cmap="Blues",␣
 ↪values_format="d", ax=axes[i]
    )
    axes[i].set_title(f"Threshold = {t}")

plt.tight_layout()
```

```
plt.show()
```



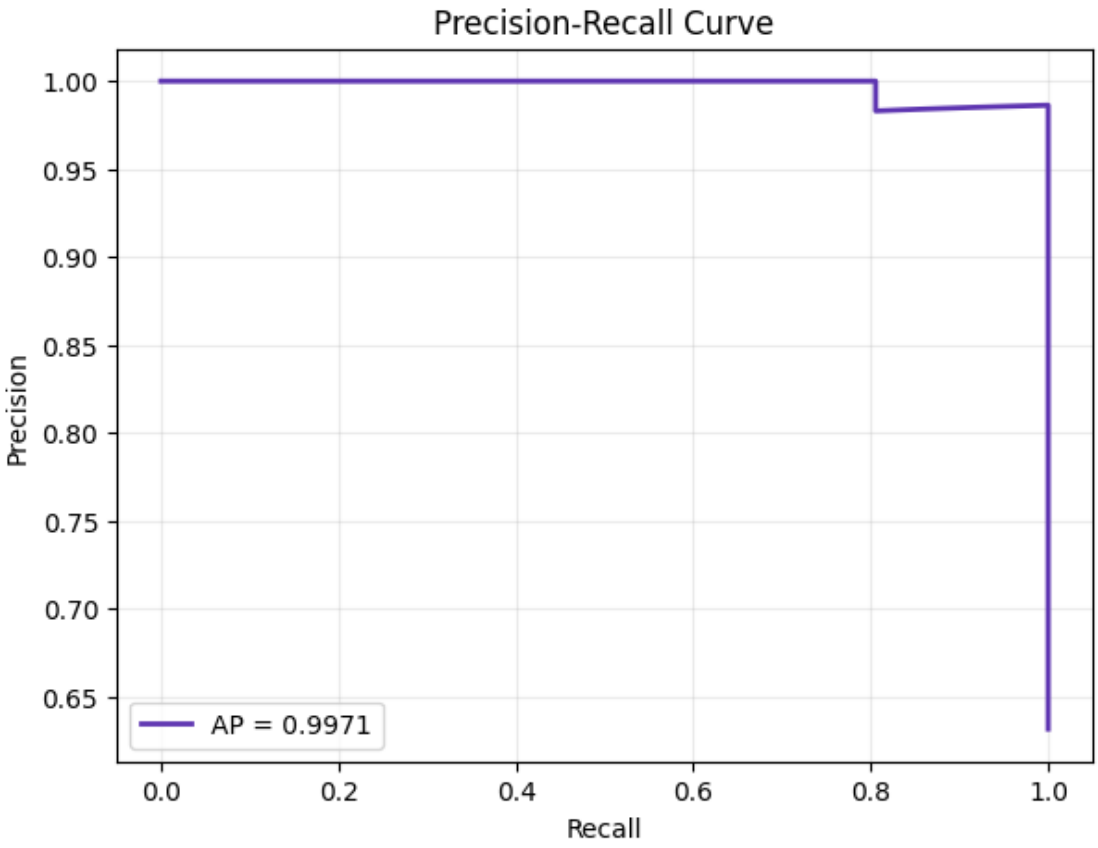
2.0.4 Graph 4: Coefficient Importance (Signed)

[12]:
```
coef_df = pd.DataFrame({
    "feature": X.columns,
    "coef": log_model.coef_[0]
})
coef_df["abs_coef"] = coef_df["coef"].abs()
top = coef_df.sort_values("abs_coef", ascending=False).head(10).
 ↪sort_values("coef")

colors = ["#ef5350" if v < 0 else "#66bb6a" for v in top["coef"]]

plt.figure(figsize=(8.5, 5))
plt.barh(top["feature"], top["coef"], color=colors)
plt.axvline(0, color="black", linewidth=1)
plt.xlabel("Coefficient Value")
plt.title("Top Logistic Regression Coefficients")
plt.show()
```



2.1 Graph 5: Precision-Recall Curve and Average Precision

[13]:
```
from sklearn.metrics import precision_recall_curve, average_precision_score

precision, recall, pr_thresholds = precision_recall_curve(y_test, y_prob)
ap = average_precision_score(y_test, y_prob)

plt.figure(figsize=(6.8, 5))
plt.plot(recall, precision, color="#5e35b1", linewidth=2, label=f"AP = {ap:.
↪4f}")
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve")
plt.legend(loc="lower left")
plt.grid(alpha=0.25)
plt.show()
```



2.2 Graph 6: Predicted Probability Distribution by True Class

[14]:
```
prob_df = pd.DataFrame({
    "y_true": y_test,
    "y_prob": y_prob
})

plt.figure(figsize=(7.2, 4.8))
plt.hist(
    prob_df.loc[prob_df["y_true"] == 0, "y_prob"],
    bins=18,
    alpha=0.65,
    color="#ef5350",
    label=f"True class 0 ({data.target_names[0]})"
)
plt.hist(
    prob_df.loc[prob_df["y_true"] == 1, "y_prob"],
    bins=18,
    alpha=0.65,
    color="#66bb6a",
```

```
    label=f"True class 1 ({data.target_names[1]})"
)
plt.axvline(0.5, color="black", linestyle="--", linewidth=1.2, label="Threshold␣
↪0.50")
plt.xlabel("Predicted Probability for Class 1")
plt.ylabel("Count")
plt.title("Probability Separation by True Class")
plt.legend()
plt.tight_layout()
plt.show()
```