Model Evaluation: Evaluation Metrics and Cross-Validation Methods

Agenda

- Exploring common evaluation metrics
- Cross-Validation Techniques
- Real-time scenarios to determine the best practices for model evaluation.

Introduction

- Model evaluation is a critical step in the machine learning pipeline to assess the performance of a model and ensure its reliability for deployment
- Choosing the right evaluation metric and cross-validation method depends on the type of problem being addressed, the data distribution, and the project's objectives

Evaluation Metrics

1. Classification Metrics

- Accuracy
  - Definition: Ratio of Correctly Predicted instances to the total instances
  - When to Use: When dataset is balanced [Equal number of samples/records/rows for each class] -> Balanced(Unbiased) Vs Biased Dataset
  - Example:

Core Concept

```
from sklearn.metrics import accuracy_score #
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

# Generating Synthetic Data with 1000 samples and 10 features using
random function
X, y = make_classification(n_samples=1000, n_features=10,
random_state=42)

# Dividing the dataset into Training set (X_train, y_train) and
Testing set (X_test, y_test)
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size =
0.2, random_state=42)

# Model Training
# Step-1 : Create Classifier Instance
model = RandomForestClassifier(random_state=42)
```

```
# Step-2 : Fit the model using X_train, y_train
model.fit(X_train, y_train)

# Step-3 : Predict the output of the model using X_test and store into
y_pred
y_pred = model.predict(X_test)

# Calculate Accuracy using Accuracy Metrics
accuracy = accuracy_score(y_test, y_pred)
#accuracy
print(f"Accuracy: {accuracy:.2f}")

Accuracy: 0.88
```

Actual Positives - positives in y_test dataset True Positives - model has predicted postive and it's positive in y_test Predicted Positives - predicted by model

TP, FP, TN, FN

Precision -> TP/TP + FP

- Definition: Focuses on the Propotion of True Positives among all Predicted Positives[40]

```
from sklearn.metrics import precision_score
# Calculate Precision Score
precision = precision_score(y_test, y_pred)
# precision
print(f"Precision: {precision:.2f}")

Precision: 0.91
```

Recall -> TP/TP + FP ?

- Definition: Focuses on the Propotion of True Positives captured among all actual Positives

```
from sklearn.metrics import recall_score
# Calculate Recall Score
recall = recall_score(y_test, y_pred)
# precision
print(f"Recall: {recall:.2f}")

Recall: 0.86
```

F1-Score -> TP/TP + FP ?

- Definition: Harmonic Mean of Precision and Recall

```
from sklearn.metrics import f1_score
f1 = f1_score(y_test, y_pred)
# f1
print(f"f1: {f1:.2f}")
```

```
f1: 0.89
```

Confusion Matrix

Confusion Matrix

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

!

Accuracy ->The ratio of correctly predicted instances to the total instances. Accuracy = TP+TN/TP+FP+TN+FN

Precision -> Focuses on the Propotion of True Positives among all Predicted Positives Precision = TP/TP+FP

Recall -> Focuses on the Propotion of True Positives captured among all actual Positives Recall = TP/TP+FN

F1 Score = 2 * (Precision * Recall/Precision + Recall)

ROC-AUC

- Definition: Measures the trade-off between true positive rate (TPR) and false positive rate (FPR) at various thresholds.

```python
from sklearn.metrics import roc_auc_score

# Probability Predictions
y_prob = model.predict_proba(X_test)[:, 1]
roc_auc = roc_auc_score(y_test, y_prob)
print(f"ROC-AUC: {roc_auc:.2f}")
```

```
ROC-AUC: 0.95
```

Regression Metrics

Mean Absolute Error (MAE)

- Measures the average magnitude of errors without considering their direction.

```
from sklearn.metrics import mean_absolute_error
from sklearn.linear_model import LinearRegression
from sklearn.datasets import make_regression

# Synthetic Data
X, y = make_regression(n_samples=1000, n_features=10, noise=0.1,
random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Model Training
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)

#mae
print(f"Mean Absolute Error: {mae:.2f}")

Mean Absolute Error: 0.08

from sklearn.metrics import mean_squared_error
import numpy as np

# MSE and RMSE
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

print(f"Mean Squared Error: {mse:.2f}")
print(f"Root Mean Squared Error: {rmse:.2f}")

Mean Squared Error: 0.01
Root Mean Squared Error: 0.10

from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred)
print(f"r2: {r2:.2f}")

r2: 1.00
```

Cross-Validation Method

K-Fold Cross Validation

```
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification

# Synthetic Data
X, y = make_classification(n_samples=1000, n_features=10,
random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Model Training
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 0.88
```

```
from sklearn.model_selection import cross_val_score

# K-Fold Cross-Validation
scores = cross_val_score(model, X, y, cv=7, scoring='accuracy')
print(f"K-Fold Accuracy Scores: {scores}")
print(f"Mean Accuracy: {scores.mean():.2f}")
```

```
K-Fold Accuracy Scores: [0.90909091 0.94405594 0.92307692 0.8951049
0.90909091 0.88111888
 0.92253521]
Mean Accuracy: 0.91
```