DAY 20 **[17 - JULY - 2025]**

Model Evaluation and Validation

Cross-Validation Techniques

Introduction

Cross-validation is a robust statistical technique used in machine learning to evaluate the performance and generalizability of a model by partitioning the dataset into multiple subsets. It helps prevent overfitting and provides a more reliable estimate of model performance compared to a single train-test split. This document focuses on two common cross-validation techniques: **k-fold cross-validation** and **stratified cross-validation**, including their definitions, use cases, and implementation in Python.

1. K-Fold Cross-Validation

Definition

K-fold cross-validation involves dividing the dataset into **k** equally sized (or nearly equal) subsets, or "folds." The model is trained on **k-1** folds and tested on the remaining fold. This process is repeated **k** times, with each fold serving as the test set exactly once. The final performance metric is typically the average of the metrics computed for each fold.

Key Features

- **Number of Folds (k)**: Common values are 5 or 10, balancing computational cost and reliability.
- **Advantages**:
  - Utilizes the entire dataset for both training and testing.
  - Reduces variance in performance estimates compared to a single train-test split.
- **Limitations**:
  - May not preserve class distribution in imbalanced datasets.
  - Computationally expensive for large datasets or complex models.

Python Example

Below is an example of implementing k-fold cross-validation using scikit-learn.

```
from sklearn.model_selection import KFold
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import numpy as np

# Sample data (replace X, y with your dataset)
X = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10], [11, 12]])
y = np.array([0, 1, 0, 1, 0, 1])

# Initialize k-fold cross-validation with k=3
kfold = KFold(n_splits=3, shuffle=True, random_state=42)
```

```
# Initialize model
model = LogisticRegression()

# Store accuracy scores
scores = []

# Perform k-fold cross-validation
for fold, (train_index, test_index) in enumerate(kfold.split(X)):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Train the model
    model.fit(X_train, y_train)

    # Predict and evaluate
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    scores.append(accuracy)
    print(f"Fold {fold + 1} Accuracy: {accuracy:.2f}")

# Compute average accuracy
print("Mean Accuracy:", np.mean(scores))
```

Output:

```
Fold 1 Accuracy: 0.50
Fold 2 Accuracy: 0.50
Fold 3 Accuracy: 1.00
Mean Accuracy: 0.6666666666666666
```

2. Stratified Cross-Validation

Definition

Stratified cross-validation is a variation of k-fold cross-validation that ensures each fold maintains the same class distribution as the original dataset. This is particularly important for imbalanced datasets, where one class may be underrepresented.

Key Features

- **Class Proportion Preservation**: Each fold has approximately the same proportion of classes as the full dataset.
- **Advantages**:
    - More reliable for imbalanced datasets, as it prevents folds with no instances of the minority class.
    - Provides better performance estimates for classification tasks.
- **Limitations**:
    - Not applicable to regression tasks, as it relies on class labels.
    - Slightly more complex to implement than standard k-fold.

Python Example

Below is an example of implementing stratified k-fold cross-validation using scikit-learn.

```
from sklearn.model_selection import StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import numpy as np

# Sample data (replace X, y with your dataset)
X = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10], [11, 12], [13, 14],
[15, 16]])
y = np.array([0, 0, 0, 0, 1, 1, 1, 1])  # Imbalanced dataset

# Initialize stratified k-fold cross-validation with k=4
strat_kfold = StratifiedKFold(n_splits=4, shuffle=True, random_state=42)

# Initialize model
model = LogisticRegression()

# Store accuracy scores
scores = []

# Perform stratified k-fold cross-validation
for fold, (train_index, test_index) in enumerate(strat_kfold.split(X, y)):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Train the model
    model.fit(X_train, y_train)

    # Predict and evaluate
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    scores.append(accuracy)
    print(f"Fold {fold + 1} Accuracy: {accuracy:.2f}")
    print(f"Fold {fold + 1} Class Distribution in Test Set:",
np.bincount(y_test))

# Compute average accuracy
print("Mean Accuracy:", np.mean(scores))
```

Output:

```
Fold 1 Accuracy: 0.50
Fold 1 Class Distribution in Test Set: [1 1]
Fold 2 Accuracy: 0.50
Fold 2 Class Distribution in Test Set: [1 1]
Fold 3 Accuracy: 1.00
Fold 3 Class Distribution in Test Set: [1 1]
Fold 4 Accuracy: 1.00
Fold 4 Class Distribution in Test Set: [1 1]
Mean Accuracy: 0.75
```