Cross Validation in Machine Learning

Cross-validation is a technique used to check how well a machine learning model performs on unseen data while preventing overfitting. It works by:

- Splitting the dataset into several parts.
- Training the model on some parts and testing it on the remaining part.
- Repeating this resampling process multiple times by choosing different parts of the dataset.
- Averaging the results from each validation step to get the final performance.

Types of Cross-Validation

There are several types of cross-validation techniques which are as follows:

1. Holdout Validation

In Holdout Validation method typically 50% data is used for training and 50% for testing. Making it simple and quick to apply. The major drawback of this method is that only 50% data is used for training, the model may miss important patterns in the other half which leads to high bias.

2. LOOCV (Leave One Out Cross Validation)

In this method the model is trained on the entire dataset except for one data point which is used for testing. This process is repeated for each data point in the dataset.

- All data points are used for training, resulting in low bias.
- Testing on a single data point can cause high variance, especially if the point is an outlier.
- It can be very time-consuming for large datasets as it requires one iteration per data point.

3. Stratified Cross-Validation

It is a technique that ensures each fold of the cross-validation process has the same class distribution as the full dataset. This is useful for imbalanced datasets where some classes are underrepresented.

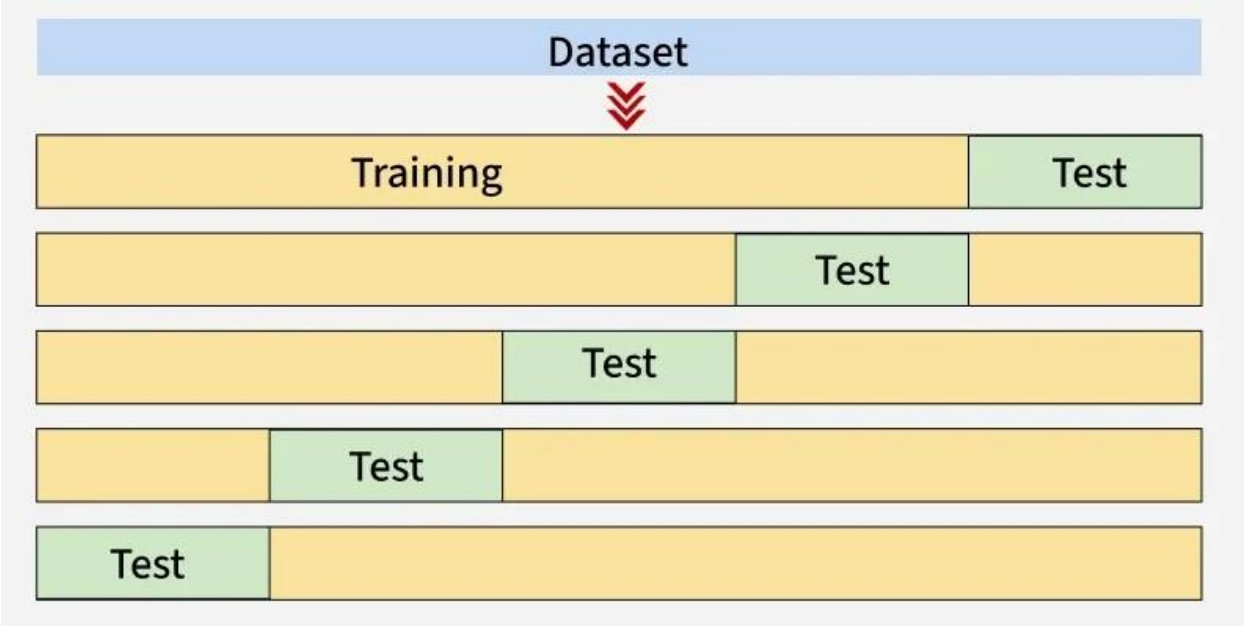
- The dataset is divided into k folds, keeping class proportions consistent in each fold.
- In each iteration, one fold is used for testing and the remaining folds for training.
- This process is repeated k times so that each fold is used once as the test set.
- It helps classification models generalize better by maintaining balanced class representation.

4. K-Fold Cross Validation

K-Fold Cross Validation splits the dataset into *k* equal-sized folds. The model is trained on *k-1* folds and tested on the remaining fold. This process is repeated *k* times each time using a different fold for testing.

Example of K Fold Cross Validation

The diagram below shows an example of the training subsets and evaluation subsets generated in k-fold cross-validation. Here we have total 25 instances.



K Fold Cross Validation

- Here we will take k as 5.
- **1st iteration:** The first 20% of data [1–5] is used for testing and the remaining 80% [6–25] is used for training.
- **2nd iteration:** The second 20% [6–10] is used for testing and the remaining data [1–5] and [11–25] is used for training.
- This process continues until each fold has been used once as the test set.

Iteration	Training Set Observations	Testing Set Observations
1	[5-24]	[0-4]
2	[0-4, 10-24]	[5-9]
3	[0-9, 15-24]	[10-14]
4	[0-14, 20-24]	[15-19]
5	[0-19]	[20-24]

Each iteration uses different subsets for testing and training, ensuring that all data points are used for both training and testing.

Example Dataset
Suppose we have a dataset with 6 observations (data points) and a single feature (e.g., a data point is a pair of features and a target value, but for simplicity, we'll just number the observations):
Data = [Observation 1, Observation 2, Observation 3, Observation 4, Observation 5, Observation 6]
We will use **k=3** folds for this example. The dataset is randomly shuffled and divided into 3 equal parts (folds), each containing 2 observations.
Machine Learning Mastery

- **Fold 1:** [Observation 1, Observation 2]
- **Fold 2:** [Observation 3, Observation 4]
- **Fold 3:** [Observation 5, Observation 6]

The 3-Fold Cross-Validation Process
The algorithm iterates k (3) times. In each iteration, one fold is used as the **test set** (or validation set), and the remaining k-1 (2) folds are combined to form the **training set**.
Machine Learning Mastery +2

Iteration	Training Data	Test Data	Model Trained	Evaluation Metric
1	Folds 2 & 3	Fold 1	Model 1	Score 1
2	Folds 1 & 3	Fold 2	Model 2	Score 2
3	Folds 1 & 2	Fold 3	Model 3	Score 3

Detailed Steps

1. **Iteration 1:**
    0. The model is trained on [Observation 3, Observation 4, Observation 5, Observation 6].
    1. The trained Model 1 is then used to make predictions on the test set: [Observation 1, Observation 2].
    2. A performance metric (e.g., accuracy for classification, mean squared error for regression) is calculated and recorded as **Score 1**.
2. **Iteration 2:**
    0. The model is trained on a different training set: [Observation 1, Observation 2, Observation 5, Observation 6].

1. The trained Model 2 makes predictions on its test set: [Observation 3, Observation 4].
2. The performance metric is calculated and recorded as **Score 2**.

3. Iteration 3:

0. The model is trained on [Observation 1, Observation 2, Observation 3, Observation 4].
1. The trained Model 3 makes predictions on its test set: [Observation 5, Observation 6].
2. The performance metric is calculated and recorded as **Score 3**.

2. Final Result
   After all k iterations are complete, the model's overall performance is estimated by taking the **average** of all the recorded scores.

$$\text{Average Performance Score} = \frac{\text{Score 1} + \text{Score 2} + \text{Score 3}}{3}$$

This average score provides a more reliable estimate of the model's generalization ability than a single train-test split, as every data point has been used for both training and testing exactly once.

Advantages and disadvantages

K-fold cross validation is straightforward to implement: once we have a routine for training a predictive model, we just run it times on the different partitions of the data. The only real disadvantage is the computational cost.

As a reward for facing an increased computational cost, we have two main advantages:

- our final model (the ensemble average) has been trained on all the data (no data have been "wasted") and it enjoys some benefits from ensembling;
- although we no longer have an unbiased estimate of the expected loss of the final model, is an estimate of an upper bound to it, which uses all the data in the sample and should therefore be pretty precise.