Cross Validation

What is Cross-Validation?

Cross-validation (CV) is a technique used to **assess the performance** of a machine learning model by splitting the dataset into multiple subsets for training and testing. It helps to:

- Avoid **overfitting** (memorizing training data instead of generalizing).
- Improve **model performance estimation**.
- Select the best **hyperparameters**.

How Does Cross-Validation Work?

1. Split the dataset into **K folds** (subsets).
2. Train the model on **K-1 folds** and test it on the **remaining fold**.
3. Repeat this process **K times**, each time using a different fold as the test set.
4. Compute the **average performance** across all K iterations.

Common Cross-Validation Techniques

1. **K-Fold Cross-Validation**
2. **Stratified K-Fold Cross-Validation**
3. **Leave-One-Out Cross-Validation (LOOCV)**
4. Leave-P-Out Cross-Validation
5. Holdout (Simple Train-Test Split)
6. **Time Series Cross-Validation**

The Hold-out Approach (Train-Test-Split)

- Also known as **Train-Test-Split**
    1. Shuffle the data
    2. Divide it in ratio
    3. Train on X_test
    4. Test the model on test data
    5. Compare y_pred with y

Problem with *The Hold-out Approach*

- Variability
    - The accuracy/r2 score, etc. changes with data
- Data inefficiency
    - You only use 70-80 % data
- Bias in performance estimation
    - Bias will increase when data is reduced
- Less reliable for hyperparameter tuning:

Why is hold-out approach used then?

- Simplicity
- Computational Efficiency
- Large Datasets:

Resampling:

- This is a broad term referring to methods that repeatedly draw samples from a dataset.

- The goal is to gain insights into the properties of the data or a model, such as its variability or accuracy.

Types

1. Cross-Validation
    - Primarily used to estimate the performance of a predictive model.
    - It involves partitioning the dataset into subsets (folds).
    - The model is trained on some folds and tested on the remaining folds.
    - This process is repeated multiple times, with different folds used for testing each time.
2. Bootstrapping
    - Used to estimate the variability of a statistic (e.g., mean, standard deviation) or to build confidence intervals.
    - It involves repeatedly drawing samples from the original dataset *with replacement*.
    - This creates multiple "bootstrap samples," which are used to estimate the distribution of the statistic.

Purpose:

- **Cross-validation**: model evaluation.
- **Bootstrapping**: estimating uncertainty.

Leave-One-Out Cross-Validation (LOO-CV)

- Forms `n` models, where `n` is the number of rows.
- Uses **one** data point as a test set and the rest as training.
- Repeats the process for **each data point**.
- Average the performance across all iterations.
- Computationally expensive but **works well for small datasets**.
    - **Therefore, not used for big datasets.**

***Example**: For 100 data points, LOOCV trains and validates the model 100 times.*


```
from sklearn.model_selection import LeaveOneOut

loo = LeaveOneOut()
scores = cross_val_score(model, X, y, cv=loo)
print("Average Score:", scores.mean())
```

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import LeaveOneOut, cross_val_score

# Load the Boston Housing dataset
df = pd.read_csv('https://raw.githubusercontent.com/selva86/datasets/maste
r/BostonHousing.csv')
X = df.iloc[:,0:-1]
y = df.iloc[:,-1]

# Create a linear regression model
model = LinearRegression()

# Create a LeaveOneOut cross-validator
loo = LeaveOneOut()
```

```
# Use cross_val_score for the dataset with the model and LOOCV
# This will return the scores for each iteration of LOOCV
scores = cross_val_score(model, X, y, cv=loo, scoring='neg_mean_squared_er
ror')

mse_scores = -scores  # Invert the sign of the scores

# Print the mean MSE over all LOOCV iterations
print("Mean MSE:", mse_scores.mean())

Output:
Mean MSE: 23.725745519476153
```

- 👆Boston Housing dataset
- We didn't do train-test-split. We sent the entire data.
- **We cannot find out R2 score as you cannot calculate an R2 score for a single row.**

k-Fold Cross-Validation

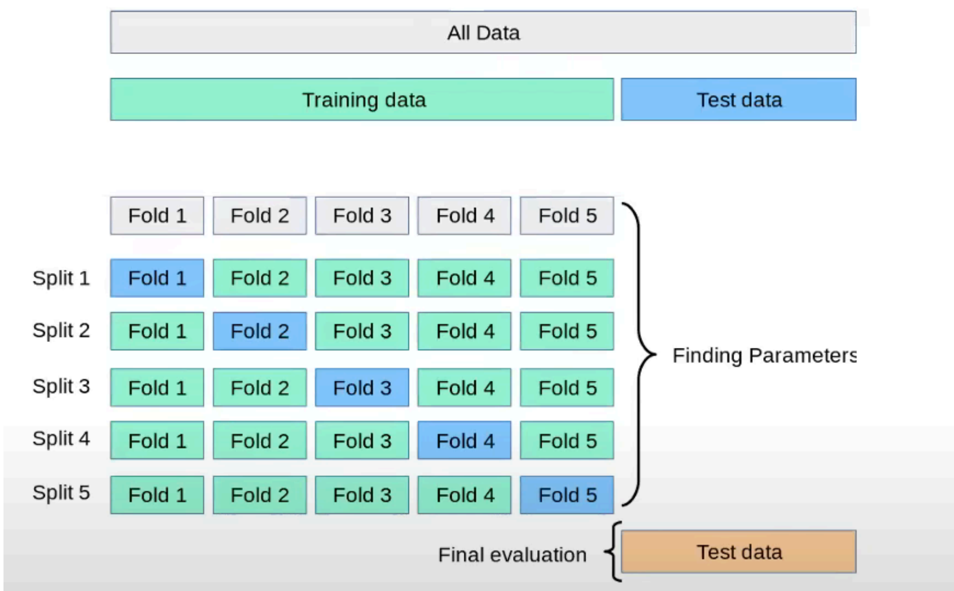
Most used technique.

> 💡 By default, `cross_val_score` **uses k-fold. You just need to provide** `cv=` **(default → 5)**

1. Split the data into k equal parts (folds).
2. Train the model on $k-1$ folds and validate on the remaining fold.
3. Repeat this process k times, each time using a different fold as the validation set.
4. Average the performance across all k folds.

💡 Generally, **k=5** or **10**

- **Example**: 5-fold cross-validation splits the data into 5 parts and uses each part once as the validation set.




```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import KFold
import pandas as pd

# Load the Boston Housing dataset
df = pd.read_csv('https://raw.githubusercontent.com/selva86/datasets/maste
```

```
r/BostonHousing.csv')
X = df.iloc[:,0:-1]
y = df.iloc[:,-1]

# Initialize a Linear Regression model
model = LinearRegression()

# Initialize the KFold parameters
kfold = KFold(n_splits=10, shuffle=True, random_state=42)

# Use cross_val_score on the model and dataset
scores = cross_val_score(model, X, y, cv=kfold, scoring='r2')

print("R2 scores for each fold:", scores)
print("Mean R2 score across all folds:", scores.mean())
```

Output:

R2 scores for each fold: [0.75981355 0.60908125 0.76975858 0.71639463
0.61663293 0.79789535
0.76682601 0.79453027 0.74066667 0.59908146]

Mean R2 score across all folds: **0.7170680714871457**

```
from sklearn.model_selection import KFold, cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.datasets import load_diabetes

# Load dataset
X, y = load_diabetes(return_X_y=True)

# Define model
model = LinearRegression()

# Apply 5-Fold Cross Validation
```

```
kf = KFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_val_score(model, X, y, cv=kf, scoring='r2')

print("R² Scores for each fold:", scores)
print("Average R² Score:", scores.mean())
```

Output:

R^2 Scores for each fold: [0.45260276 0.57320015 0.39144785 0.58428888 0.39081186]
Average R^2 Score: 0.47847030225778475

`return_X_y=True` is a shortcut that directly gives you the feature matrix (`X`) and target vector (`y`) as NumPy arrays,

Advantages of K-Fold Cross Validation:

- Reduction of Variance
- Computationally Inexpensive

Disadvantages of K-Fold Cross Validation:

- Potential for **High Bias**
- May not work well with Imbalanced Classes:

When to use?

- When you have a sufficiently **large dataset**
- When your data is **evenly distributed**

Stratified K-Fold

- Similar to k-fold, but ensures **each fold has the same proportion of classes as the original dataset.**
- **Useful for imbalanced datasets.**

- Mostly used for **Classification** problems.


```
from sklearn.model_selection import StratifiedKFold

skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

```
from sklearn.datasets import load_iris
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.linear_model import LogisticRegression

# Load iris dataset
data = load_iris()
X, y = data.data, data.target

# Create a Logistic Regression model
model = LogisticRegression(max_iter=10000, random_state=42)

# Create StratifiedKFold object
```

```
skf = StratifiedKFold(n_splits=5, random_state=42, shuffle=True)

# Perform stratified cross validation
scores = cross_val_score(model, X, y, cv=skf, scoring='accuracy')

# Print the accuracy for each fold
print("Accuracies for each fold: ", scores)
print("Mean accuracy across all folds: ", scores.mean())
```

Output:

Accuracies for each fold: [1.   0.96666667 0.93333333 1.   0.93333333]
Mean accuracy across all folds:
0.9666666666666668