CROSS VALIDATION

MACHINE LEARNING

CROSS-VALIDATION

Cross-Validation (CV) is a **model validation technique** used to check how well your machine learning model will generalize on **unseen data.**

Instead of training on one fixed train-test split, CV **splits the data multiple times** and checks performance in each split.

Purpose	Explanation
1. Reduce overfitting	Ensures model performs well on unseen data.
2. Better model evaluation	Uses multiple splits instead of one.
3. Reliable performance score	Gives average accuracy/MAE/R^2 etc.
4. Helps in hyperparameter tuning	Used in GridSearchCV, RandomizedSearchCV.

WHY CROSS-VALIDATION IS USED?

Because a single train-test split is not reliable.

Example:

Train on 80%

Test on 20%

If the 20% test set is **too easy or too difficult**, accuracy becomes misleading.

Cross-Validation solves this by:

- Splitting the dataset into *k* equal parts (folds)
- Training multiple times
- Averaging the score

This gives **stable and unbiased** performance.

TYPES OF CROSS-VALIDATION

10 are the standard techniques used in ML:

1. K-Fold
2. Stratified K-Fold
3. LOOCV
4. Leave-P-Out
5. ShuffleSplit
6. Repeated K-Fold
7. TimeSeriesSplit
8. Group K-Fold
9. Stratified Group K-Fold
10. Nested Cross-Validation

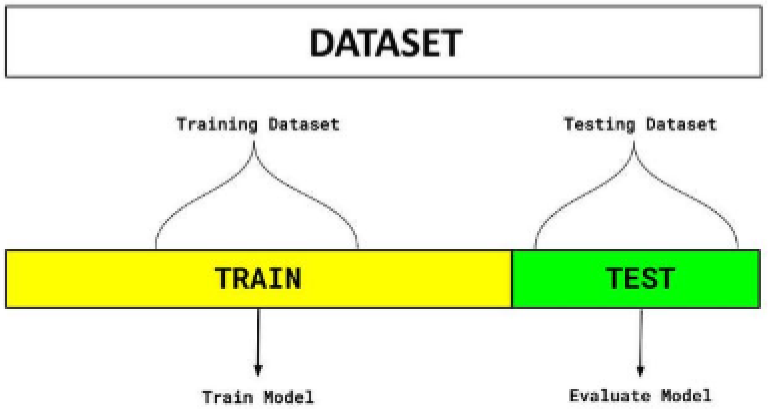
CROSS-VALIDATION IS USEFUL IN **ANY DOMAIN WHERE YOU TRAIN MODELS AND NEED TO EVALUATE THEM**, INCLUDING:

AI Domain	Used?	Why
Machine Learning	YES	Evaluate models reliably
Deep Learning	Sometimes	Usually replaced by train/val/test splits due to huge data
NLP	Optional	Depends on dataset size
Computer Vision	Optional	Big datasets → single split is enough
Time Series Forecasting (AI)	YES	Uses TimeSeriesSplit
Reinforcement Learning	Rare	RL uses episodic evaluation
Statistics & Data Science	YES	Model reliability

HOLD OUT CROSS VALIDATION

- The Holdout Method is a **fundamental validation technique** in machine learning used to evaluate the performance of a predictive model.
- The dataset is commonly divided into training set and test set.
- Typical **split ratios** include **70:30, 80:20 or 60:40** depending on dataset size.
- It is **most effective** when the **dataset is large enough** to allow meaningful splitting.
- **Random shuffling** before splitting is often applied to reduce bias.
- Simple and Faster

DRAWBACK: single random split, Not ideal for small datasets



CODE:

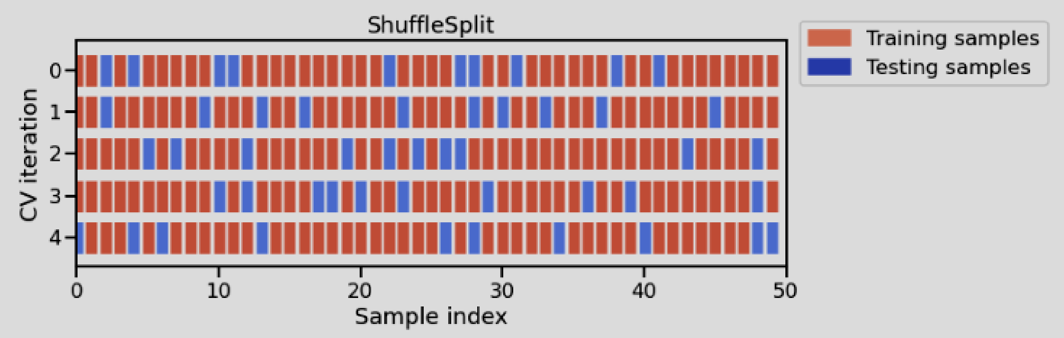
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
model = LogisticRegression()
model.fit(X_train, y_train)
```

SHUFFLE SPLIT CROSS VALIDATION

- ShuffleSplit **randomly shuffles** the dataset.
- Then it splits it into **train (e.g., 70%)** and **test (e.g., 30%)**.
- It does this **multiple times** (e.g., 10 iterations).
- Every iteration gives a **different random 70-30 split.** (eg: 65 – 35, 80-20, 73-7 ..)
- The model is trained and tested on each different split.
- Finally, we take the **average performance** over all splits.

<u>Benefits</u>: works for small dataset, fast and stable model accuracy

<u>Drawback</u>: time consuming



<u>CODE</u>:

```
from sklearn.model_selection import ShuffleSplit
from sklearn.linear_model import LogisticRegression
#dataset
X, y = make_classification(n_samples=100, n_features=4,
n_informative=2,
n_redundant=0, n_repeated=0, n_classes=2,
random_state=42)
ss = ShuffleSplit(n_splits=5, test_size=0.3, random_state=0)
model = LogisticRegression(random_state=42)
scores = []
for train_index, test_index in ss.split(X):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        model.fit(X_train, y_train)
```

LOOCV (LEAVE ONE OUT CROSS VALIDATION)

- In this method the model is **trained on the entire dataset except for <u>one data point which is used for testing</u>**. This process is repeated for each data point in the dataset.
- All data points used for training – low bias
- **very time-consuming for large datasets** as it requires one iteration per data point.

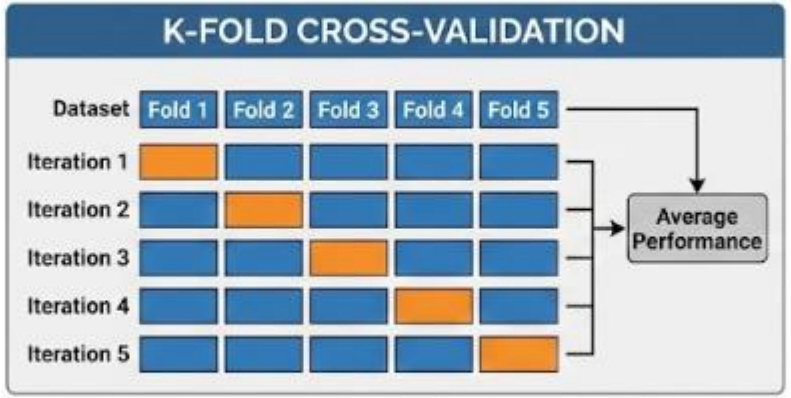
<u>DRAWBACK</u>: Computationally expensive for large datasets.

<u>CODE</u>:

```
from sklearn.model_selection import LeaveOneOut
# sample dataset
X = np.array([[1], [2], [3], [4], [5], [6], [7], [8], [9], [10]])
y = np.array([2, 4, 5, 4, 6, 7, 8, 9, 10, 12])
# Initialize the LeaveOneOut cross-validator
loo = LeaveOneOut()
model = LinearRegression()
# Perform LOOCV
for train_index, test_index in loo.split(dataset):
        # Split data into training and testing sets for the current fold
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        # Fit the model on the training data
        model.fit(X_train, y_train)
```

K-FOLD CROSS-VALIDATION

- The dataset is **divided 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**, with each fold serving as the test set exactly once.
- The **results** are then **averaged.**
- Best use case: Small to medium datasets
- Drawback: slower for large dataset, can't handle imbalanced dataset



CODE:

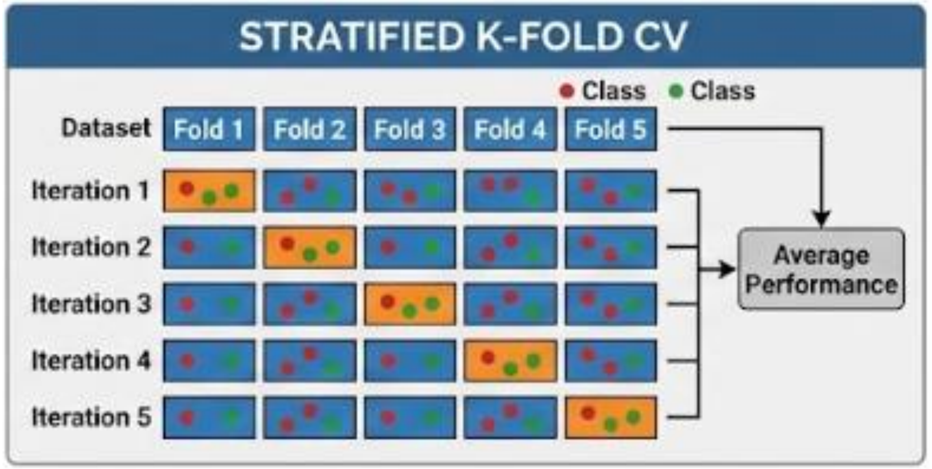
```
from sklearn.model_selection import KFold, cross_val_score
from sklearn.ensemble import RandomForestClassifier
kf = KFold(n_splits=5, shuffle=True, random_state=42)
model = RandomForestClassifier(random_state=42)
for fold_num, (train_index, test_index) in enumerate(kf.split(X, y)):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        model.fit(X_train, y_train)
```

STRATIFIED K-FOLD CROSS-VALIDATION

- Similar to k-fold, but ensures that **each fold maintains the same proportion of target variable classes** as the original dataset.
- This is crucial for imbalanced datasets.
- Best for **imbalanced classification problems**
- Key benefits: **Reliable Metrics, Generalization**

CODE:

```
from sklearn.model_selection import StratifiedKFold
from sklearn.linear_model import LogisticRegression
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
model = LogisticRegression(random_state=42)
for fold_idx, (train_index, test_index) in enumerate(skf.split(X, y)):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        model.fit(X_train, y_train)
```



TIME SERIES CROSS-VALIDATION (ROLLING CROSS-VALIDATION)

- Used for **time-dependent data**. Instead of random splits, it respects **the order of data**, using past events to predict future events.
- It's essential for time series forecasting.
- drawback: Can't use future data to predict the past

CODE:

```
from sklearn.model_selection import TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=5)
for fold, (train_index, test_index) in enumerate(tscv.split(X)):
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]
        model = LinearRegression()
        model.fit(X_train, y_train)
```