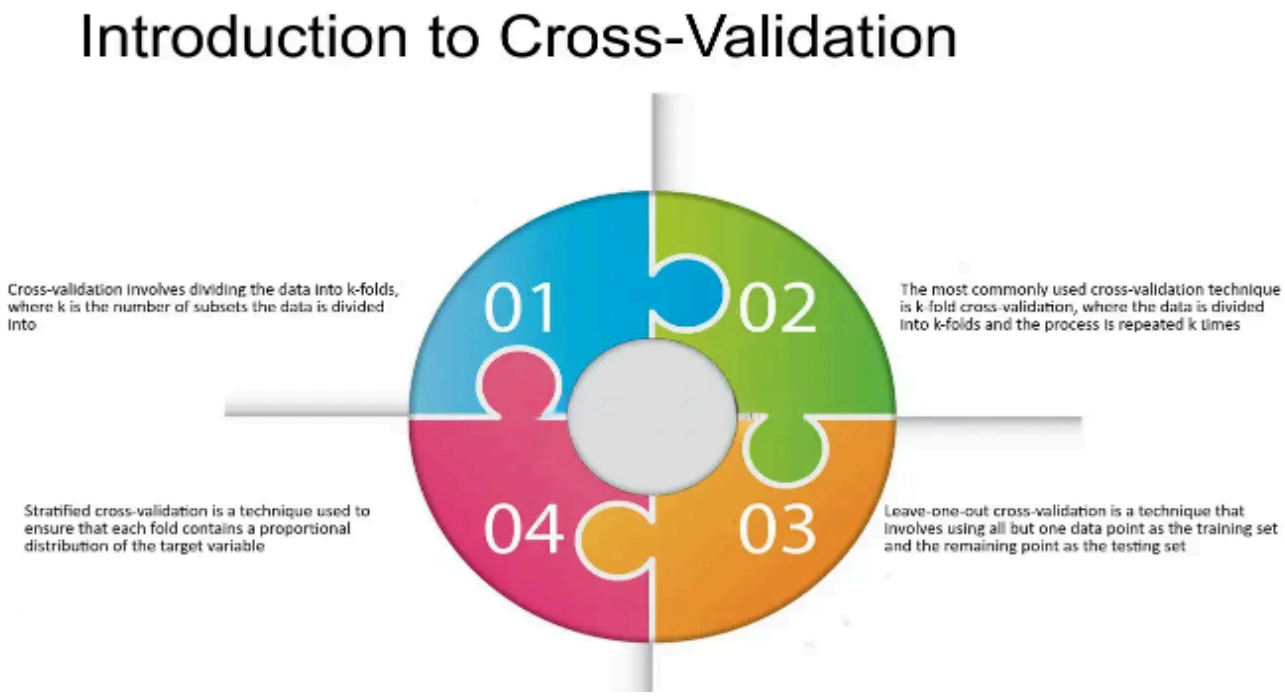
ARTICLE
Cross Validation Techniques



Introduction

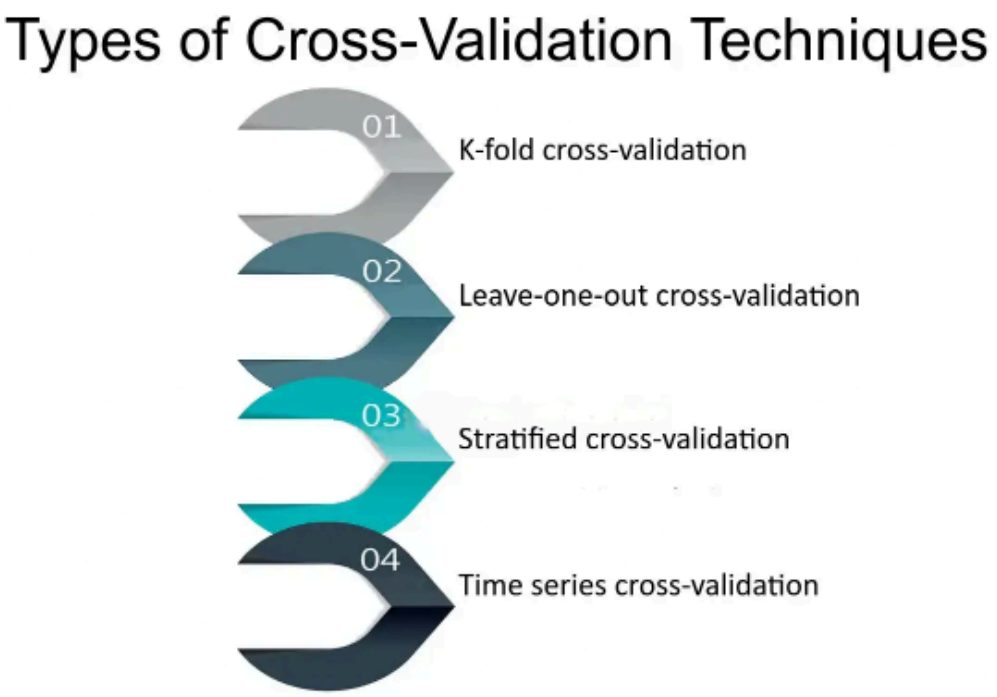
Today i am going to tell about an import technique in data science ,machine learning which reduce the over fitting in our model Because when data scientists create a model in machine learning the data should be split into train and test validation for better accuracy and gain the output or prediction values.But the main thing is when data scientist created a model it should be in generalized model . For being the generalized model cross validation techniques is very useful and it will reduce the overfitting problem from the model.

Let's discuss about cross validation:

The core mechanics of cross-validation involve splitting the dataset into multiple complementary subsets, with only some subsets used for model training and the remaining "unseen" data reserved for validation.

This process is repeated over multiple rounds, using different subsets each time for training versus testing. The performance metrics from each round are then aggregated into a final score reflecting the model's overall generalization capability.

As you may know, there are plenty of CV techniques. Some of them are commonly used, others work only in theory. Let’s see the cross-validation methods that will be covered in this article.



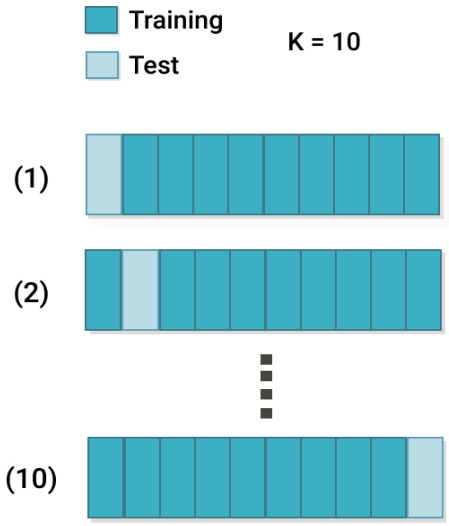
K-Fold Cross-Validation :

k-Fold cross-validation is a technique that minimizes the disadvantages of the hold-out method. k-Fold introduces a new way of splitting the dataset which helps to overcome the “test only once bottleneck”.

This is perhaps the most widely used cross-validation method. The dataset is randomly partitioned into K equally-sized subsets or "folds" with K-1 folds used for training and the remaining single fold held out for testing in each iteration.

This process repeats K times until all folds have been used exactly once for validation. In typical applications, K is often set to 5 or 10. While this can require K times as much computation as simpler train/test splits, it provides K times as many data samples to better estimate generalization performance.

The algorithm of the k-Fold technique:



1. Pick a number of folds – k. Usually, k is 5 or 10 but you can choose any number which is less than the dataset’s length.
2. Split the dataset into k equal (if possible) parts (they are called folds)
3. Choose k – 1 folds as the training set. The remaining fold will be the test set
4. Train the model on the training set. On each iteration of cross-validation, you must train a new model independently of the model trained on the previous iteration
5. Validate on the test set
6. Save the result of the validation
7. Repeat steps 3 – 6 k times. Each time use the remaining fold as the test set. In the end, you should have validated the model on every fold that you have.
8. To get the final score average the results that you got on step 6.

```
import numpy as np
from sklearn.model_selection import KFold

X = np.array([[1, 2], [3, 4], [1, 2], [3, 4]])
y = np.array([1, 2, 3, 4])
kf = KFold(n_splits=2)

for train_index, test_index in kf.split(X):
   print("TRAIN:", train_index, "TEST:", test_index)
   X_train, X_test = X[train_index], X[test_index]
```

Leave-one-out cross-validation (LOOCV) :

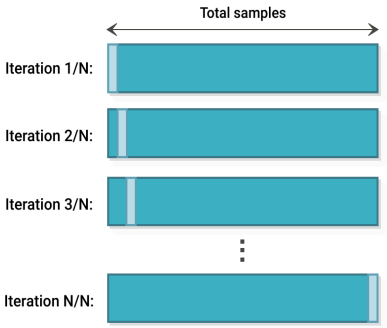
Leave-one-out cross-validation (LOOCV) is an extreme case of k-Fold CV. Imagine if k is equal to n where n is the number of samples in the dataset. Such a k-Fold case is equivalent to the Leave-one-out technique.

This approach takes K-fold to the extreme by using a single observation from the original dataset as the test set, with the remaining data used for training. Like K-fold, this entire procedure iterates over all observations once.

LOOCV fully utilizes all available data and provides very reliable estimates with low bias. However, it is computationally expensive for large datasets, requiring runs on N separate training models. It can also overfit on datasets with noisy observations or outliers.

The algorithm of LOOCV technique:

1. Choose one sample from the dataset which will be the test set
2. The remaining n – 1 samples will be the training set
3. Train the model on the training set. On each iteration, a new model must be trained
4. Validate on the test set
5. Save the result of the validation
6. Repeat steps 1 – 5 n times as for n samples we have n different training and test sets
7. To get the final score average the results that you got on step 5



```
import numpy as np
from sklearn.model_selection import LeaveOneOut

X = np.array([[1, 2], [3, 4]])
y = np.array([1, 2])
loo = LeaveOneOut()

for train_index, test_index in loo.split(X):
    print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

The greatest advantage of Leave-one-out cross-validation is that it doesn't waste much data. We use only one sample from the whole dataset as a test set, whereas the rest is the training set. But when compared with k-Fold CV, LOOCV requires building n models instead of k models, when we know that n which stands for the number of samples in the dataset is much higher than k. It means LOOCV is more computationally expensive than k-Fold, it may take plenty of time to cross-validate the model using LOOCV.

Thus, the Data Science community has a general rule based on empirical evidence and different research, which suggests that 5- or 10-fold cross-validation should be preferred over LOOCV.

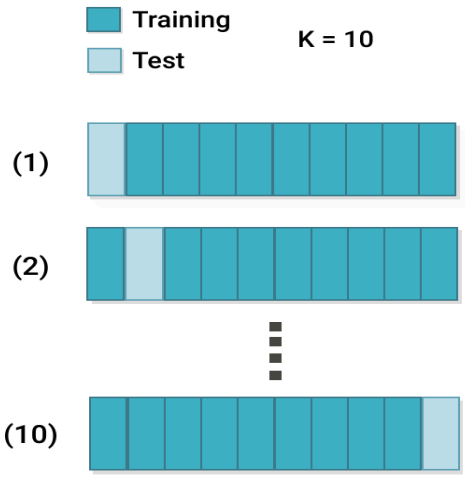
Stratified Cross-Validation :

Stratified k-Fold is a variation of the standard k-Fold CV technique which is designed to be effective in such cases of target imbalance.

It works as follows. Stratified k-Fold splits the dataset on k folds such that each fold contains approximately the same percentage of samples of each target class as the complete set. In the case of regression, Stratified k-Fold makes sure that the mean target value is approximately equal in all the folds.

The algorithm of Stratified k-Fold technique:

1. Pick a number of folds – k
2. Split the dataset into k folds. Each fold must contain approximately the same percentage of samples of each target class as the complete set
3. Choose k – 1 folds which will be the training set. The remaining fold will be the test set
4. Train the model on the training set. On each iteration a new model must be trained
5. Validate on the test set
6. Save the result of the validation
7. Repeat steps 3 – 6 k times. Each time use the remaining fold as the test set. In the end, you should have validated the model on every fold that you have.
8. To get the final score average the results that you got on step 6.



As you may have noticed, the algorithm for Stratified k-Fold technique is similar to the standard k-Folds. You don't need to code something additionally as the method will do everything necessary for you.

Stratified k-Fold also has a built-in method in sklearn – **sklearn.model_selection.StratifiedKFold**.

```
import numpy as np
from sklearn.model_selection import StratifiedKFold

X = np.array([[1, 2], [3, 4], [1, 2], [3, 4]])
y = np.array([0, 0, 1, 1])
skf = StratifiedKFold(n_splits=2)

for train_index, test_index in skf.split(X, y):
    print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

Time Series Cross-Validation :

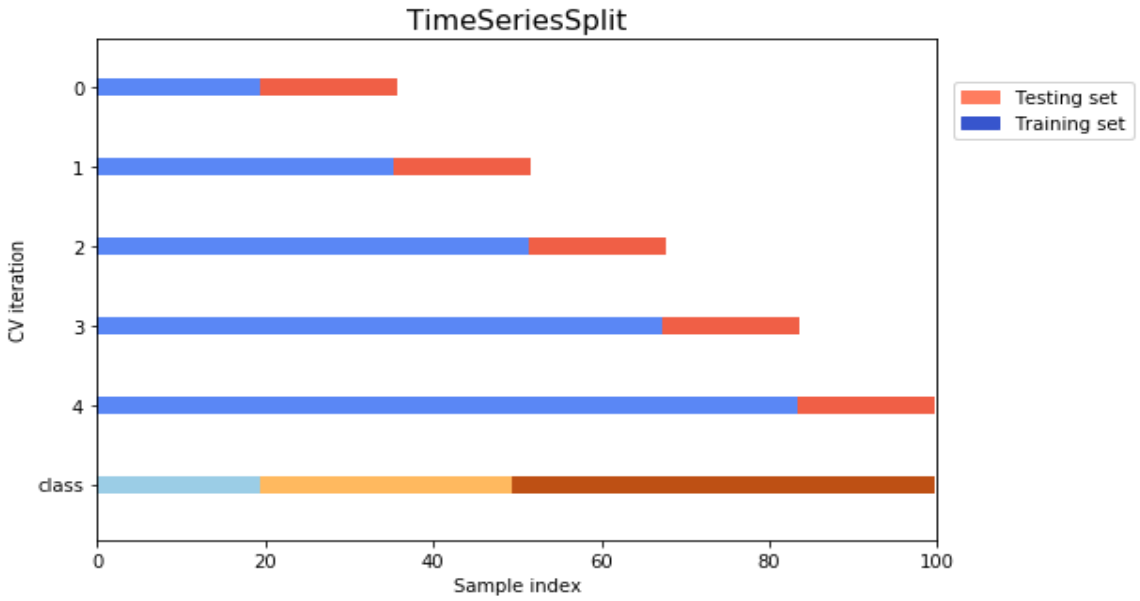
For sequences of temporal data like time series, random shuffling during cross-validation violates the inherent chronological ordering dependencies in the data. Standard methods will leak future information into the training sets.

Time series cross-validation overcomes this by creating contiguous folds respecting the order of observations. Common methods include forward chaining with a fixed window sliding one step ahead or fixed units like years, months, days as folds.

Traditional cross-validation techniques don't work on sequential data such as time-series because we cannot choose random data points and assign them to either the test set or the train set as it makes no sense to use the values from the future to forecast values in the past. There are mainly two ways to go about this:

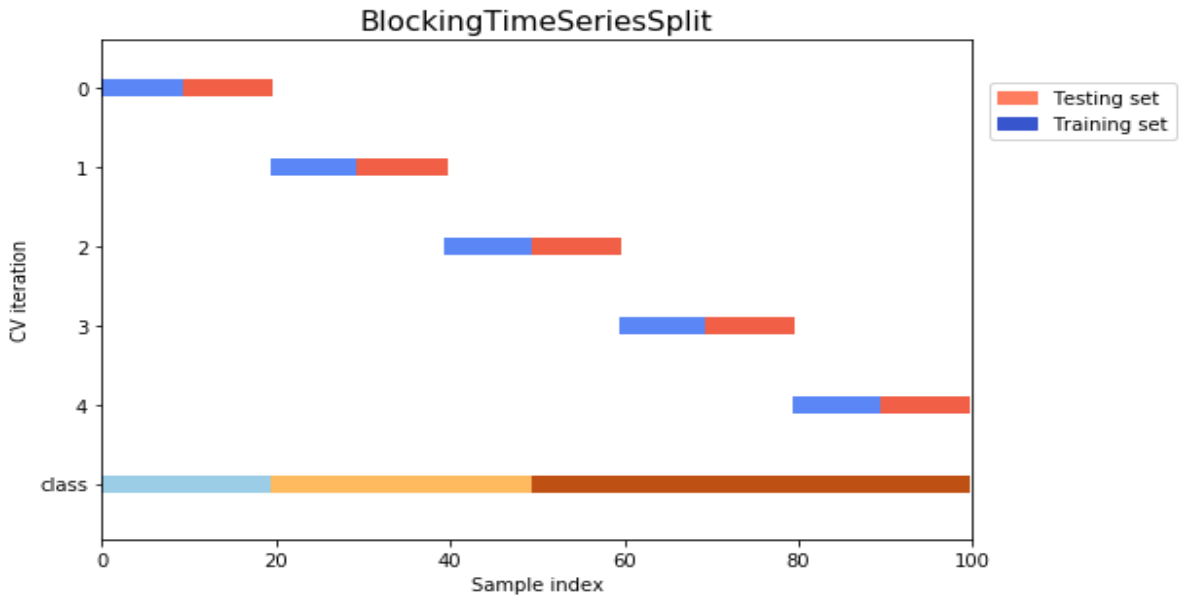
1. **Rolling cross-validation :**

   Cross-validation is done on a rolling basis i.e. starting with a small subset of data for training purposes, predicting the future values, and then checking the accuracy on the forecasted data points. The following image can help you get the intuition behind this approach.



2. **Blocked cross-validation :**

   The first technique may introduce leakage from future data to the model. The model will observe future patterns to forecast and try to memorize them. That's why blocked cross-validation was introduced.



It works by adding margins at two positions. The first is between the training and validation folds in order to prevent the model from observing lag values which are used twice, once as a regressor and another as a response. The second is between the folds used at each iteration in order to prevent the model from memorizing patterns from one iteration to the next.

Best Practices :

To get reliable model evaluations, follow these key cross-validation best practices:

- Maintain true data splitting - avoid data leakage or contamination between datasets
- Use stratification or grouping splits to account for class imbalance or other heterogeneity
- Use sufficiently many folds (e.g. 5, 10) for less biased estimates
- For time series, respect temporal ordering with forward-chaining techniques
- Prefer simple validation metrics and statistical tests over implicitly complex ones

- Nest cross-validation within outer loops for hyperparameter search
- Reserve a true held-out test set beyond validation for final evaluation
- Report validation scores across multiple runs with confidence intervals

While cross-validation adds overhead and computation, it's a crucial step to maximize a machine learning model's ability to generalize to new scenarios.

From comparing multiple algorithms to hyperparameter tuning, these resampling methods play an instrumental role across the entire model building lifecycle. With principled practices, cross-validation helps deliver robust, reliable, and trustworthy models that translate experimental results into real-world impact.