Cross Validation

What is Cross Validation and how it works ?

Data science and Machine learning Online Course

Machine Learning - Bahram Jannesar, Soroush Ghaderi

What is Cross-Validation:

Cross-validation is a technique for evaluating a machine learning model and testing its performance. CV is commonly used in applied ML tasks. It helps to compare and select an appropriate model for the specific predictive modeling problem.

CV is easy to understand, easy to implement, and it tends to have a lower bias than other methods used to count the model's efficiency scores. All this makes cross-validation a powerful tool for selecting the best model for the specific task.

There are a lot of different techniques that may be used to cross-validate a model. Still, all of them have a similar algorithm:

1. Divide the dataset into two parts: one for training, other for testing
2. Train the model on the training set
3. Validate the model on the test set
4. Repeat 1-3 steps a couple of times. This number depends on the CV method that you are using

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

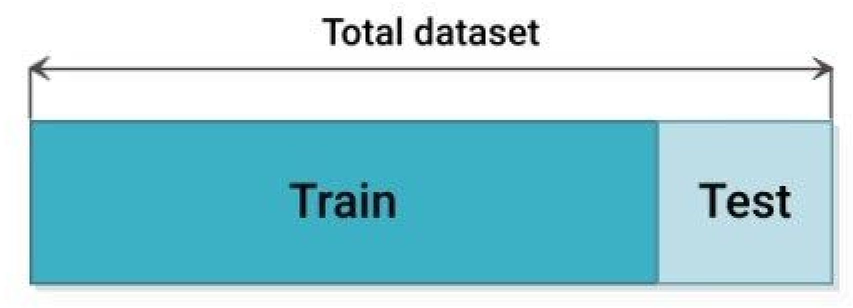
- *Hold-out*
- *K-folds*
- *Leave-one-out*
- *Leave-p-out*
- *Stratified K-folds*

Hold-out

Hold-out cross-validation is the simplest and most common technique. You might not know that it is a hold-out method but you certainly use it every day.

The algorithm of hold-out technique:

1. Divide the dataset into two parts: the training set and the test set. Usually, 80% of the dataset goes to the training set and 20% to the test set but you may choose any splitting that suits you better
2. Train the model on the training set
3. Validate on the test set
4. Save the result of the validation



That's it.

We usually use a hold-out method on large datasets as it requires training the model only once.

It is really easy to implement hold-out. For example, you may do it using `sklearn.model_selection.train_test_split`.

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

X, y = np.arange(10).reshape((5, 2)), range(5)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2,

random_state=111)
```

Still, hold-out has a major disadvantage.

For example, a dataset that is nor completely even distribution-wise. If so we may end up in a rough spot after the split. For example, the training set will not represent the test set. Both training and test sets may differ a lot, one of them might be easier or harder.

Moreover, the fact that we test our model only once might be a bottleneck for this method. Due to the reasons mentioned before, the result obtained by hold-out technique may be considered inaccurate.

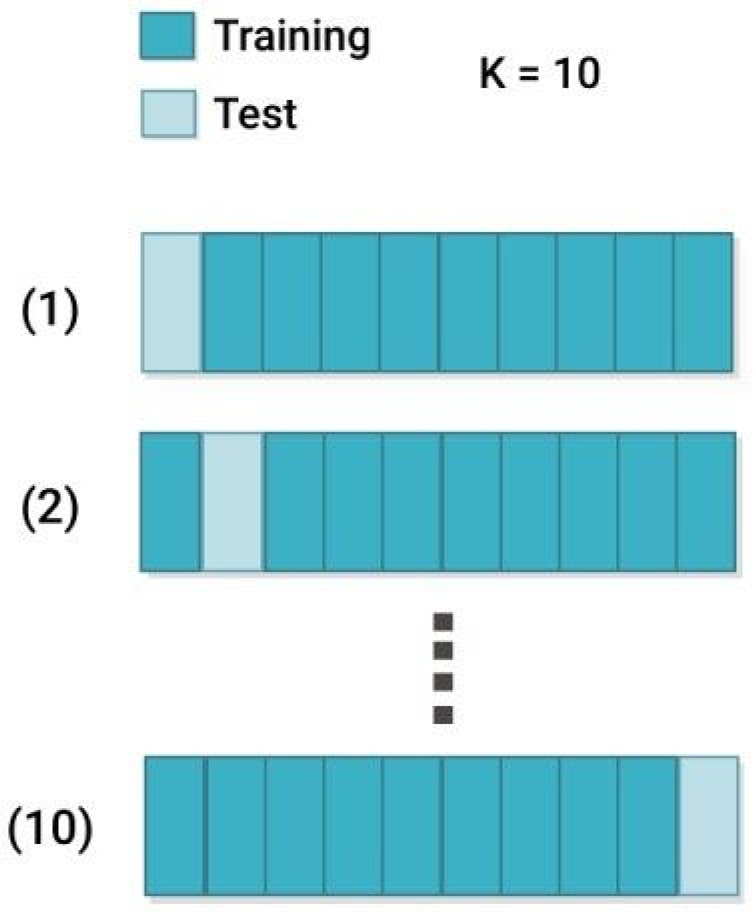
k-Fold

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

The algorithm of 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 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 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.



To perform k-Fold cross-validation you can use `sklearn.model_selection.KFold.`

```
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]
    y_train, y_test = y[train_index], y[test_index]
```

In general, it is always better to use k-Fold technique instead of hold-out. In a head to head, comparison k-Fold gives a more stable and trustworthy result since training and testing is performed on several different parts of the dataset. We can make the overall score even more robust if we increase the number of folds to test the model on many different sub-datasets.

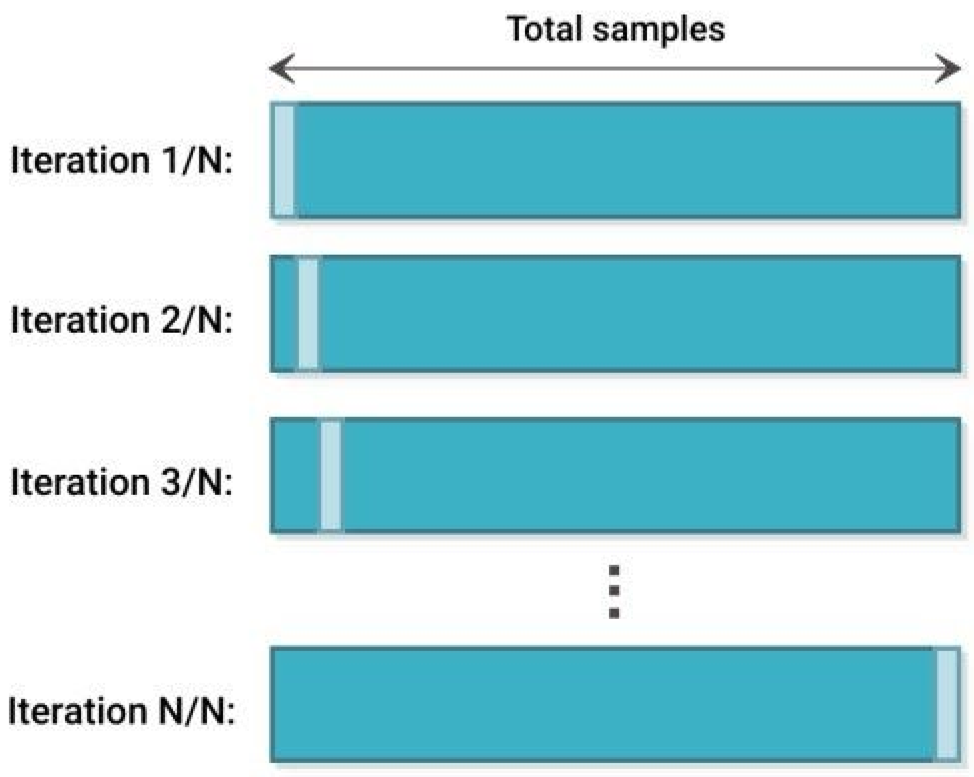
Still, k-Fold method has a disadvantage. Increasing k results in training more models and the training process might be really expensive and time-consuming.

Leave-one-out

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 k-Fold case is equivalent to Leave-one-out technique.

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.



For LOOCV sklearn also has a built-in method. It can be found in the model_selection library – `sklearn.model_selection.LeaveOneOut`.

```
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 researches, which suggests that 5- or 10-fold cross-validation should be preferred over LOOCV.

Leave-p-out

Leave-p-out cross-validation (LpOC) is similar to Leave-one-out CV as it creates all the possible training and test sets by using p samples as the test set. All mentioned about LOOCV is true and for LpOC.

Still, it is worth mentioning that unlike LOOCV and k-Fold test sets will overlap for LpOC if p is higher than 1.

The algorithm of LpOC technique:

1. Choose p samples from the dataset which will be the test set
2. The remaining n – p 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 2 – 5 Cpn times
7. To get the final score average the results that you got on step 5

You can perform a Leave-p-out CV using sklearn – `sklearn.model_selection.LeavePOut`.

```python
import numpy as np
from sklearn.model_selection import LeavePOut

X = np.array([[1, 2], [3, 4], [5, 6], [7, 8]])
y = np.array([1, 2, 3, 4])
lpo = LeavePOut(2)

for train_index, test_index in lpo.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]
```

LpOC has all the disadvantages of the LOOCV, but, nevertheless, it's as robust as LOOCV.

Stratified k-Fold

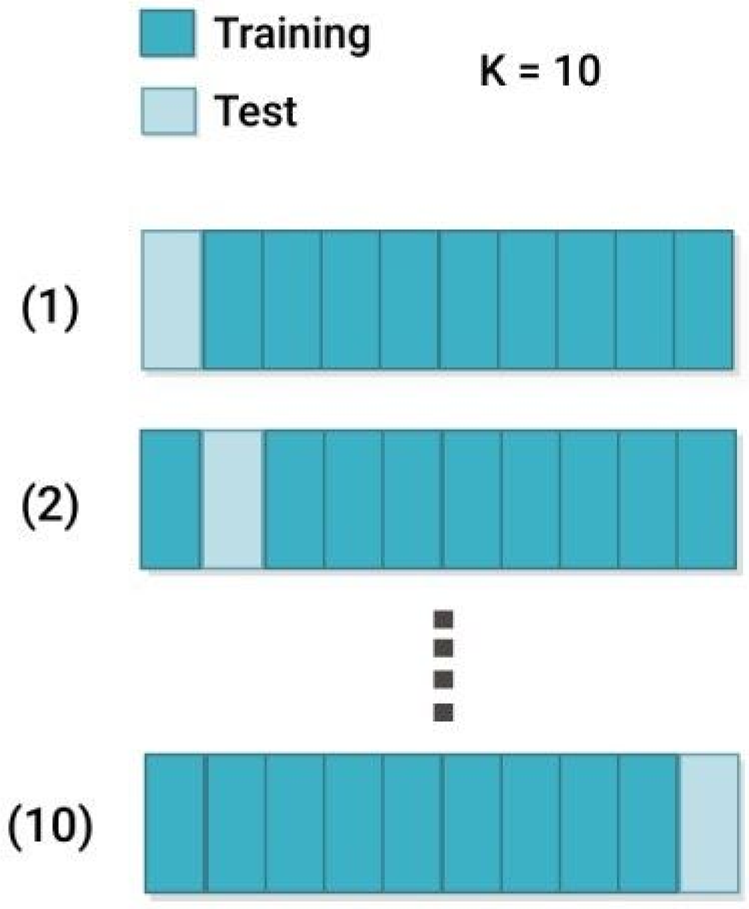
Sometimes we may face a large imbalance of the target value in the dataset. For example, in a dataset concerning wristwatch prices, there might be a larger number of wristwatches having a high price. In the case of classification, in cats and dogs dataset there might be a large shift towards the dog class.

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]
```

All mentioned above about k-Fold CV is true for Stratified k-Fold technique. When choosing between different CV methods, make sure you are using the proper one. For example, you might think that your model performs badly simply because you are using k-Fold CV to validate the model which was trained on the dataset with a class imbalance. To avoid that you should always do a proper exploratory data analysis on your data.

Best practices and tips

It's worth mentioning that sometimes performing cross-validation might be a little tricky.

For example, it's quite easy to make a logical mistake when splitting the dataset which may lead to an untrustworthy CV result.

You may find some tips that you need to keep in mind when cross-validating a model below:

1. Be logical when splitting the data (does the splitting method make sense)
2. Use the proper CV method (is this method viable for my use-case)
3. When working with time series don't validate on the past (see the first tip)
4. When working with medical or financial data remember to split by person. Avoid having data for one person both in the training and the test set as it may be considered as data leak
5. When cropping patches from larger images remember to split by the large image Id

Of course, tips differ from task to task and it's almost impossible to cover all of them. That's why performing a solid exploratory data analysis before starting to cross-validate a model is always the best practice.