Cross-validation

It is a technique for evaluating the performance of a machine learning model. It involves dividing a dataset into a training set and a test set, and training the model on the training set. The model is then evaluated on the test set, and the performance measure is recorded.

- ***Cross-validation steps***:

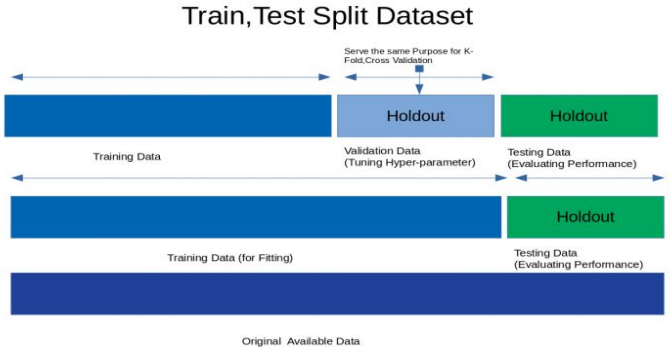
1. Divide data set at random into training and test sets.
2. Fit model on training set.
3. Test model on test set.
4. Compute and save fit statistic using test data (step 3).
5. Repeat 1 - 4 several times, then average results of all step 4.

- ***Advantages:***

1. It helps to avoid overfitting: By training the model on a subset of the data and evaluating it on a different subset, you can get a more accurate estimate of the model's performance on new data.
2. It allows you to use all of the data for training and evaluation: When you have a small dataset, you may not have enough data to split into separate training and test sets. Cross-validation allows you to use all of the data for both training and evaluation.
3. It provides a more robust estimate of the model's performance: By training and evaluating the model multiple times, you can get a better idea of how well the model will perform on new data.

- **Cross-validation types:**

1. **HoldOut Cross-validation or Train-Test Split:** In this technique of cross-validation, the whole dataset is randomly partitioned into a training set and validation set. Using a rule of thumb nearly 70% of the whole dataset is used as a training set and the remaining 30% is used as the validation set.




```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

iris=load_iris()
X=iris.data[li]
Y=iris.target[li]
print("Size of Dataset {}".format(len(X)))


logreg=LogisticRegression()

x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.3,random_state=42)
logreg.fit(x_train,y_train)
predict=logreg.predict(x_test)
```

- **pros**:
  Quick To Execute: As we have to split the dataset into training and validation set just once and the model will be built just once on the training set so gets executed quickly.

- **Cons:**
  Not Suitable for an imbalanced dataset: Suppose we have an imbalanced dataset that has class ‘0’ and class ‘1’. Let’s say 80% of data belongs to class ‘0’ and the remaining 20% data to class ‘1’.On doing train-test split with train set size as 80% and test data size as 20% of the dataset. It may happen that all 80% data of class ‘0’ may be in the training set and all data of class ‘1’ in the test set. So our model will not generalize well for our test data as it hasn’t seen data of class ‘1’ before

2. **Leave One Out cross-validation**: is an exhaustive cross-validation technique in which 1 sample point is used as a validation set and the remaining n-1 samples are used as the training set. Suppose we have 100 samples in the dataset. Then in each iteration 1 value will be used as a validation set and the remaining 99 samples as the training set. Thus the process is repeated till every sample of the dataset is used as a validation point.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import LeaveOneOut,cross_val_score

iris=load_iris()
X=iris.data[li]
Y=iris.target[li]

loo=LeaveOneOut()
tree=RandomForestClassifier(n_estimators=10,max_depth=5,n_jobs=-1)
score=cross_val_score(tree,X,Y,cv=loo)
```

```
print("Cross Validation Scores are {}".format(score))
print("Average Cross Validation score :{}".format(score.mean()))
```

- **Pros:**
  Simple, easy to understand, and implement.

- **Cons:**
  - The computation time required is high.

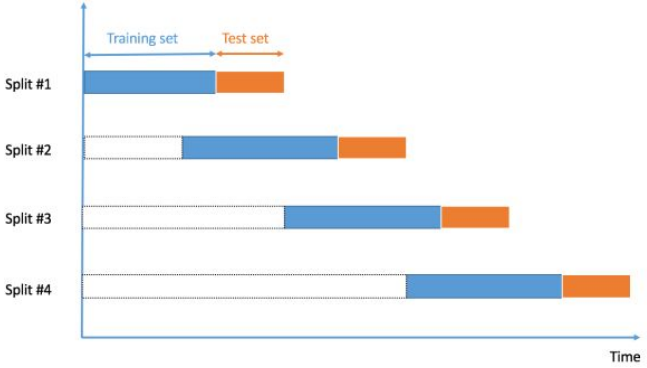
3. **Leave P Out cross-validation:** LeavePOut cross-validation is an exhaustive cross-validation technique, in which p-samples are used as the validation set and remaining n-p samples are used as the training set. Suppose we have 100 samples in the dataset. If we use p=10 then in each iteration 10 values will be used as a validation set and the remaining 90 samples as the training set. This process is repeated till the whole dataset gets divided on the validation set of p-samples and n-p training samples.

It is the same as LeaveOneOut cross-validation with p=1.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import LeaveOneOut,cross_val_score

iris=load_iris()
X=iris.data[li]
Y=iris.target[li]

loo=LeaveOneOut()
tree=RandomForestClassifier(n_estimators=10,max_depth=5,n_jobs=-1)
score=cross_val_score(tree,X,Y,cv=loo)
```

```
print("Cross Validation Scores are {}".format(score))
print("Average Cross Validation score :{}".format(score.mean()))
```

4. **Time Series Cross-Validation:** In the case of time-series data, we cannot choose random samples and assign them to either training or validation set as it makes no sense in using the values from the future data to predict values of the past data. As the order of the data is very important for time series related problems, so we split the data into training and validation set according to time, also called as “Forward chaining” method or rolling cross-validation. We start with a small subset of data as the training set. Based on that set we predict later data points and then check the accuracy. The Predicted samples are then included as part of the next training dataset and subsequent samples are forecasted.



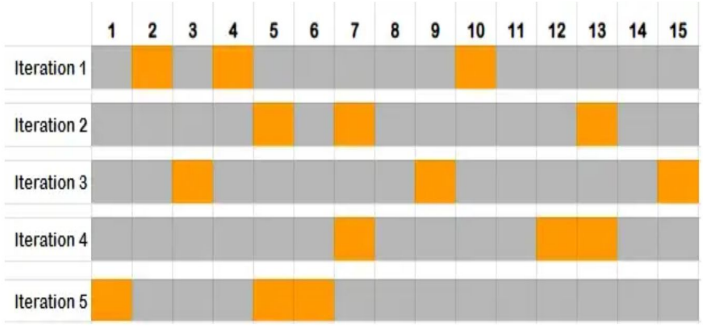

```
import numpy as np
from sklearn.model_selection import TimeSeriesSplit

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

time_series = TimeSeriesSplit()

print(time_series)

for train_index, test_index in time_series.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]
```

```
TimeSeriesSplit(gap=0, max_train_size=None, n_splits=5, test_size=None)
TRAIN: [0] TEST: [1]
TRAIN: [0 1] TEST: [2]
TRAIN: [0 1 2] TEST: [3]
TRAIN: [0 1 2 3] TEST: [4]
TRAIN: [0 1 2 3 4] TEST: [5]
```

- **Pros**:
  One of the finest techniques .

- **Cons:**
  Not suitable for validation of other data types: As in other techniques we choose random samples as training or validation set, but in this technique order of data is very important.

5. Monte Carlo Cross-Validation(Shuffle Split): Monte Carlo cross-validation, also known as Shuffle Split cross-validation, is a very flexible strategy of cross-validation. In this technique, the datasets get randomly partitioned into training and validation sets. We have decided upon the percentage of the dataset we want to be used as a training set and the percentage to be used as a validation set. If the added percentage of training and validation set size is not sum up to 100 then the remaining dataset is not used in either training or validation set. Let's say we have 100 samples and 60% of samples to be used as training set and 20% of the sample to be used as validation set then the remaining 20%(100-(60+20)) is not to be used.



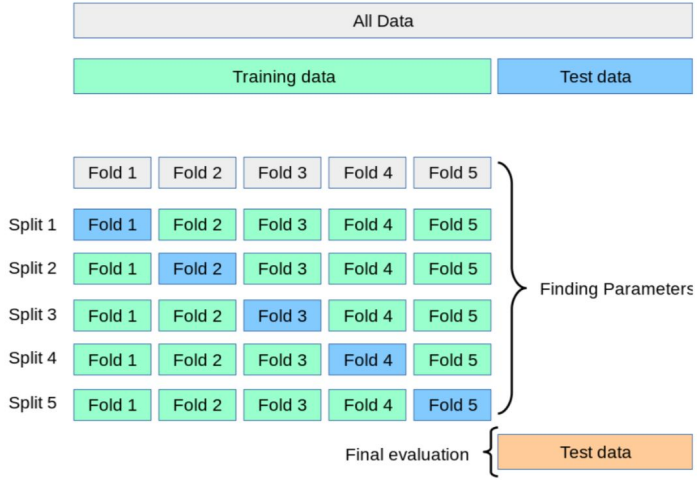

```
from sklearn.model_selection import ShuffleSplit,cross_val_score
from sklearn.linear_model import LogisticRegression

logreg=LogisticRegression()

shuffle_split=ShuffleSplit(test_size=0.3,train_size=0.5,n_splits=10)

scores=cross_val_score(logreg,iris.data[li],iris.target[li],cv=shuffle_split)
```

```
print("cross Validation scores:n {}".format(scores))
print("Average Cross Validation score :{}".format(scores.mean()))
```

- **Pros**:
  The proportion of train and validation splits is not dependent on the number of iterations or partitions.

- **Cons**:
  - Some samples may not be selected for either training or validation.
  - Not suitable for an imbalanced dataset.

6. K-Fold Cross-Validation: In this technique of K-Fold cross-validation, the whole dataset is partitioned into K parts of equal size. Each partition is called a “Fold“.So as we have K parts we call it K-Folds. One Fold is used as a validation set and the remaining K-1 folds are used as the training set.
The technique is repeated K times until each fold is used as a validation set and the remaining folds as the training set.
The final accuracy of the model is computed by taking the mean accuracy of the k-models validation data.



$$\mathrm{acc_{cv}} = \sum_{i=1}^{k} \frac{\mathrm{acc_i}}{k}$$

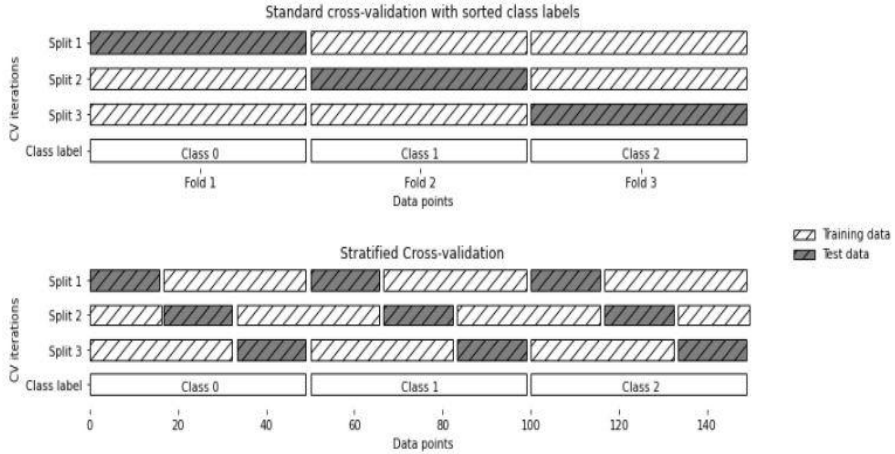
```
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score,KFold
from sklearn.linear_model import LogisticRegression

iris=load_iris()

X=iris.data[li]
Y=iris.target[li]

logreg=LogisticRegression()
kf=KFold(n_splits=5)
score=cross_val_score(logreg,X,Y,cv=kf)
```

```
print("Cross Validation Scores are {}".format(score))
print("Average Cross Validation score :{}".format(score.mean()))
```

- **Pros:**
  - Low time complexity.
  - The entire dataset is utilized for both training and validation.

- **Cons:**
  - Not to be used for imbalanced datasets.
  - Not suitable for Time Series data.

7. **Stratified k-fold cross-validation:** Stratified K-Fold is an enhanced version of K-Fold cross-validation which is mainly used for imbalanced datasets. Just like K-fold, the whole dataset is divided into K-folds of equal size. But in this technique, each fold will have the same ratio of instances of target variable as in the whole datasets.




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

iris=load_iris()
X=iris.data[li]
Y=iris.target[li]

logreg=LogisticRegression()
stratifiedkf=StratifiedKFold(n_splits=5)

score=cross_val_score(logreg,X,Y,cv=stratifiedkf)
```

```
print("Cross Validation Scores are {}".format(score))
print("Average Cross Validation score :{}".format(score.mean()))
```

- **Pros:**
  Works perfectly well for Imbalanced Data: Each fold in stratified cross-validation will have a representation of data of all classes in the same ratio as in the whole dataset.

- **Cons:**
  - Not suitable for Time Series data: for Time Series data the order of the samples matter. But in Stratified Cross-Validation, samples are selected in random order.