CROSS VALIDATION

Why Do We Study Cross Validation?

In machine learning, we generally have a dataset consisting of rows and columns, where we have input and output columns in supervised learning. We train a model on this dataset, but how do we know if the trained model will perform well on a new dataset?

Suppose we are working at Ola and developing a machine learning model to detect whether a customer will leave the platform based on their activity. For example, if a user performs a certain action and does not leave but instead continues with other actions, our model should predict whether the user will stay or leave.

Now, can we directly deploy the model? How do we determine if the model we built performs well? Our manager may ask us how the model performs on new data. If we only evaluate the model after deployment, we might realize later that it performs poorly, which is not a good approach.

To avoid this, we must evaluate our model's performance before deployment so that we can select the best model. This is where cross-validation comes into play. In cross-validation, we train our model on different subsets of data and then evaluate it to ensure it generalizes well to new data. This helps us select the most reliable model before deploying it.

Hold-Out Approach

So far, we have used the hold-out approach in our machine learning workflow. This approach is also known as train-test split.

In this method, if we have a dataset with 1,000 customer records, we follow these steps:

1. Shuffle the data to ensure randomness.
2. Split the data into training and testing sets, typically in a 75:25 or 80:20 ratio.
3. Train the model using the training data.
4. Test the model by making predictions on the testing data.
5. Compare the predicted values (y_pred) with the actual values (y_true) to evaluate model performance.

This is the simplest approach for model evaluation. However, it is not considered the best method because the performance of the model depends heavily on how the data is split.

A better alternative to the hold-out approach is cross-validation, which provides a more reliable and improved way to evaluate models.

Problems with the Hold-Out Approach

- Variability in Performance
  The performance of a model can be highly sensitive to how the dataset is split into training and testing sets. If the split is not representative, the training set may not capture the overall distribution of the data, or the test set may contain disproportionately easy or difficult examples. This can result in significant variance in the model's performance evaluation.
- Data Inefficiency
  The hold-out method uses only a portion of the data for training and reserves a separate portion for testing. This means the model cannot learn from the entire dataset, which can be particularly problematic when the dataset is small, leading to suboptimal model training.
- Bias in Performance Estimation
  If certain classes or patterns are over- or under-represented in either the training or test set due to the random split, the model's performance evaluation may be biased. This can lead to an inaccurate assessment of how the model will perform on new, unseen data.
- Challenges with Hyperparameter Tuning
  When the hold-out method is used for hyperparameter tuning, there is a risk of overfitting to the test set. This occurs because information from the test set may indirectly influence the model's tuning process, leading to overly optimistic performance estimates on the test set that do not generalize well to unseen data.

Why is hold - out approach used then ?

- Simplicity : The holdout method is straightforward and easy to understand . You simply divide your data into two sets : a training set and a test set . This simplicity makes it appealing , especially for initial exploratory analysis or simple projects .
- Computational Efficiency : The holdout method is computationally less intensive than methods like k - fold cross - validation . In k - fold cross - validation , you need to train and test your model k times , which can be computationally expensive , especially for large datasets or complex models . With the holdout method , you only train the model once .
- Large Datasets : For very large datasets , even a small proportion of the data may be sufficient to form a representative test set . In these cases , the holdout method can work quite well .

CROSS VALIDATION :->

The idea of cross - validation is to divide the data into several subsets or " folds " . The model is
then trained on some of these subsets and tested , on the remaining ones . This process is
repeated multiple times , with different subsets used for training and validation each time . The results from each round are usually averaged to estimate the model's overall performance .

Cross-Validation in Resampling

Cross-validation is a technique that falls under the broader category of resampling methods. Resampling involves generating new samples from an existing set of data, which is then used to estimate population statistics. To clarify, sampling refers to the process of selecting observations (or data points) from a population, which are then used to make inferences about the entire population. Resampling refers to the process of creating new datasets by drawing observations from the observed sample.
For example, consider the Iris dataset used in machine learning. This dataset is a sample of data. What if we generate additional samples from this original dataset (e.g., 100 new datasets) and train a model on each of them? This process is called resampling.

Techniques in Cross-Validation

In cross-validation, there are two common techniques: Leave-One-Out Cross-Validation (LOOCV) and K-Fold Cross-Validation (KFCV). Both techniques have different variants and are used to assess the model’s performance by partitioning the dataset in various ways.

Leave-One-Out Cross-Validation (LOOCV) Technique

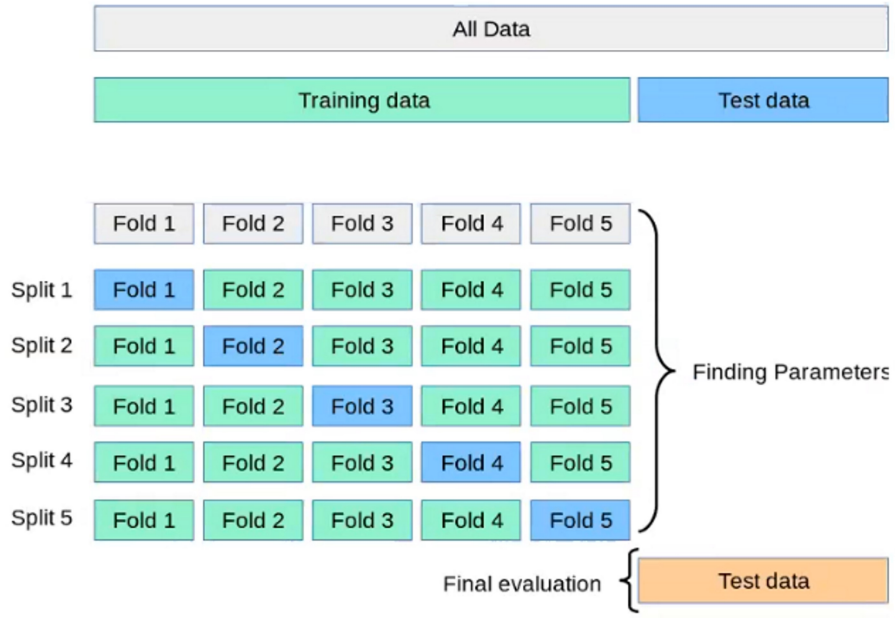
Leave-One-Out Cross-Validation (LOOCV) is a model evaluation technique where a model is trained multiple times, with each training session using a different subset of the data. This technique is particularly useful when we want to use every data point for both training and testing.
For example, suppose we have a dataset with 1,000 rows. In LOOCV, we will create 1,000 different models, each trained and tested on a slightly different version of the dataset.

Here’s how LOOCV works:

1. Initial Dataset:
    - Suppose our dataset is represented by rows: 1,2,3,4,5,…,1000
    - We will perform 1,000 iterations, each time removing one row for testing while using the remaining rows for training.
2. First Iteration:
    - In the first iteration, we remove the first row from the dataset. The remaining 999 rows are used for training, while the first row is used as the testing data.
    - We then train a model on the 999 training rows and test it on the first row.
3. Subsequent Iterations:
    - In the second iteration, we remove the second row and use the remaining 999 rows for training. The second row becomes the testing data, and we train the model on this subset.
    - This process continues, with each row being removed one at a time for testing, and the remaining rows being used for training. This continues until all 1,000 rows have been used for testing.
4. Final Iteration:
    - In the final iteration, the 1,000th row is removed, and the remaining 999 rows are used for training. The 1,000th row is then used as the testing data for this final model.
5. Model Evaluation:
    - After training all 1,000 models, each model will have an associated accuracy score (assuming it's a classification model).
    - The average accuracy of all 1,000 models is then calculated. This average gives us an overall performance metric for the model.

Summary: LOOCV ensures that each data point is used once as a test point, and the remaining points are used for training. The advantage of LOOCV is that it provides a very thorough evaluation of the model's performance, but it can be computationally expensive when dealing with large datasets.

K-Fold Cross-Validation



K-Fold Cross-Validation is one of the most commonly used techniques for model evaluation. It provides a good balance between computational efficiency and accuracy compared to the Leave-One-Out Cross-Validation (LOOCV) technique. While LOOCV can be very computationally expensive (especially for large datasets), K-Fold Cross-Validation helps overcome this by reducing the number of models that need to be trained.

How K-Fold Cross-Validation Works:

1. Selecting the Value of K:
    - First, we decide on the value of K, which typically is set to 5 or 10. These values have been empirically proven to provide reliable results. The choice of K determines how many equal-sized subsets (or folds) the data will be divided into.
2. Splitting the Data:
    - Suppose we have a dataset with 100 rows. In this case, with K = 5, we divide the data into 5 equal-sized folds (each fold contains 20 rows). The data is randomly shuffled to ensure randomness in each fold.
3. Training and Testing:
    - For each fold, we train the model on K-1 folds (all but one fold), and use the remaining fold for testing.
    - For example, in the first iteration (Model 1):
        - We train the model on folds 2, 3, 4, and 5.
        - We test it on fold 1.

    - In the second iteration (Model 2):
        - We train the model on folds 1, 3, 4, and 5.
        - We test it on fold 2.
    - This process is repeated for each fold, meaning we train and test 5 different models (one for each fold). Each fold will serve as the test set exactly once.
4. Evaluating the Models:
    - After training and testing all models, each model will have an associated accuracy score.
    - The final average accuracy score is calculated by taking the average of the accuracy scores from all K models.
5. Final Testing:
    - Once the model's performance is evaluated using cross-validation, we can proceed to test the final model on a separate, unseen test dataset that was held out at the start for this purpose.

Summary:

K-Fold Cross-Validation helps in balancing the need for thorough evaluation (by using different parts of the dataset for testing) while reducing the computational burden compared to LOOCV. This technique is efficient and helps in understanding the model's ability to generalize to unseen data.

Stratified Cross-Validation

One of the main drawbacks of K-Fold Cross-Validation is that it can be unreliable when dealing with imbalanced datasets in classification problems. Specifically, if the classes in the dataset are imbalanced, there's a risk that some folds may not contain samples from the minority classes at all. In such cases, a machine learning model might end up predicting only the majority class for every test instance, leading to misleading results, such as achieving 100% accuracy by predicting the majority class for every instance.

The Problem with K-Fold in Imbalanced Datasets:

- Suppose you have a dataset with three classes, where the distribution of the classes is highly imbalanced. For example:
    - Class 1: 2 instances
    - Class 2: 4 instances
    - Class 3: 3 instances

When you apply K-Fold Cross-Validation with, say, 5 folds, it's possible that in some folds, there could be no instances of the minority classes (Class 1 or Class 3), and only instances of the majority class (Class 2) might be included. In this scenario, the model would learn to predict only the majority class, resulting in a misleading accuracy score of 100%. This is because the model never has to predict the minority classes and always predicts the majority class correctly.

The Solution: Stratified K-Fold Cross-Validation

Stratified K-Fold Cross-Validation solves this issue by ensuring that each fold contains the same class distribution as the original dataset. This means that each fold will have the same proportions of each class as seen in the entire dataset.

Here's how it works:

1. Class Proportions in the Original Dataset:
    - Suppose we have a dataset where the ratio of the classes is as follows:
        - Class 1: 2 instances
        - Class 2: 4 instances
        - Class 3: 3 instances
        - This gives us a class distribution ratio of 2:4:3.
2. Stratified Folds Creation:
    - When applying Stratified K-Fold, the algorithm ensures that each fold contains the same ratio of classes as the original dataset. So, if we have 5 folds, each fold will contain:
        - Class 1: 2/9 of the instances
        - Class 2: 4/9 of the instances
        - Class 3: 3/9 of the instances

3. Balanced Folds:
    - This ensures that all classes are represented in each fold, and their proportions in each fold reflect the proportions in the original dataset. Therefore, each fold will be a representative sample of the entire dataset.
4. Improved Accuracy Evaluation:
    - As a result, each model trained during cross-validation will be evaluated on a dataset that contains all classes in the correct proportions. This prevents the issue of the model being biased toward the majority class, and it ensures that the accuracy score is more reliable and realistic.

Summary:

Stratified K-Fold Cross-Validation addresses the problem of imbalanced datasets in classification tasks by maintaining the class distribution in each fold, ensuring that each fold is representative of the entire dataset. This leads to more realistic model evaluation, where the accuracy is not inflated by the model predicting the majority class and allows for better generalization to unseen data.