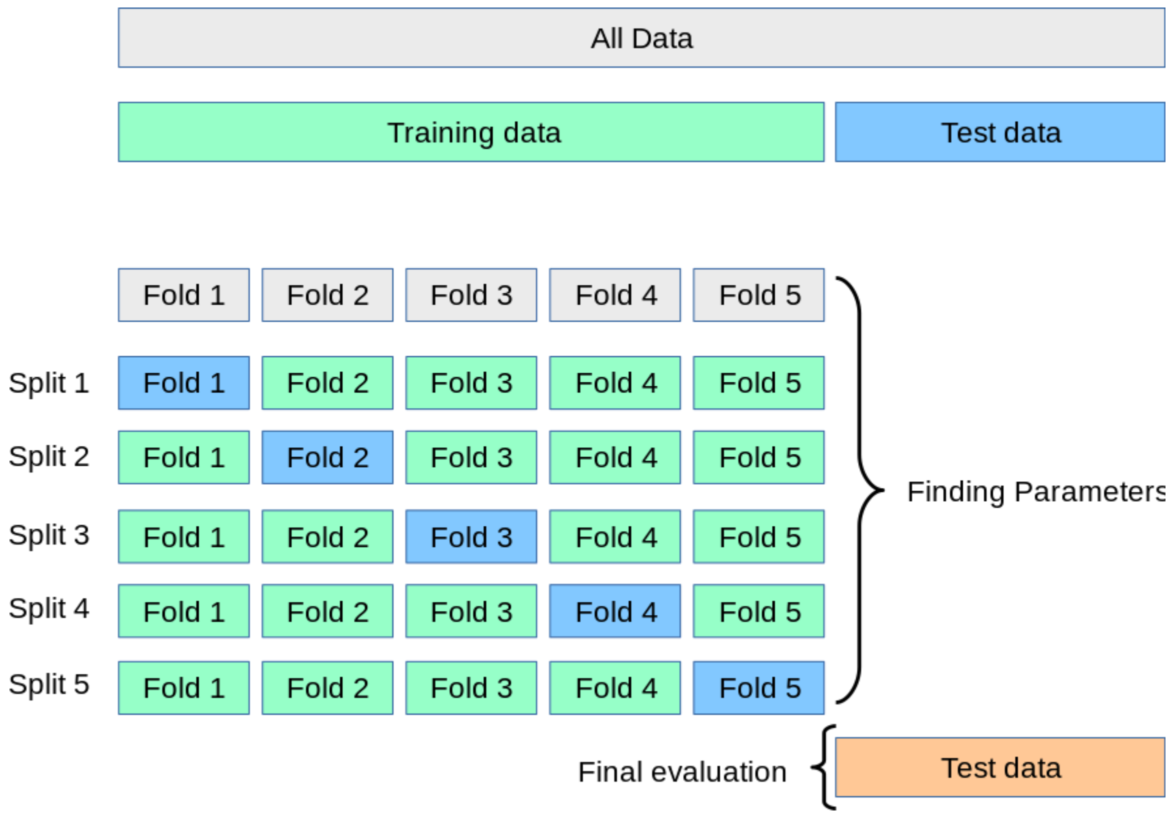
Cross-Validation

1. What is Cross-Validation?

- **Purpose:** Cross-validation is a resampling technique used to evaluate machine learning models on a limited data sample. Its primary goals are:
    - To get a more **reliable estimate of model performance** on unseen data (generalization performance).
    - To **avoid overfitting** during model evaluation and selection. A model might perform well on the specific data it was trained on (training data), but CV helps estimate how it performs on data it hasn't seen before.
    - To **tune model hyperparameters** (like the α in Ridge/Lasso/Elastic Net, or the depth of a decision tree) by selecting the parameters that yield the best average performance across CV folds.
- **Basic Idea:** Instead of a single split into training and testing sets (which can be sensitive to *which* data points end up in the test set), CV involves splitting the data into multiple "folds" or subsets. The model is repeatedly trained on some folds and tested on the remaining fold, and the results are averaged.



2. Why Use Cross-Validation?

- **Robust Performance Estimate:** Averaging results over multiple splits gives a less biased and lower variance estimate of how the model is likely to perform on new, independent data compared to a single train-test split.
- **Efficient Data Usage:** Especially important for smaller datasets where holding out a large test set isn't feasible. CV allows virtually all data points to be used for both training and validation across different iterations.
- **Hyperparameter Tuning:** Provides a systematic way to compare models with different hyperparameter settings and choose the best ones based on their CV performance.

3. General Cross-Validation Procedure (Illustrative Example: K-Fold)

1. **Split:** Divide the entire dataset randomly into *K* non-overlapping subsets of roughly equal size. These are called "folds".
2. **Iterate:** Loop *K* times. In each iteration `i` (from 1 to K):
   - **Hold-out:** Use fold `i` as the **validation set**.
   - **Train:** Use the remaining `K-1` folds as the **training set**.
   - **Evaluate:** Train the model on the training set and evaluate its performance (e.g., calculate accuracy, MSE) on the validation set (fold `i`). Store this performance score.
3. **Aggregate:** Combine the `K` performance scores obtained (e.g., by averaging them) to get the overall cross-validation performance metric. This aggregated metric provides a more robust estimate of the model's generalization ability.

4. Types of Cross-Validation

A. Leave-One-Out Cross-Validation (LOOCV)

- **How it works:** A special case of K-Fold where the number of folds `K` is equal to the number of data points `N`. In each iteration, one data point is held out as the validation set, and the model is trained on the remaining `N-1` data points. This is repeated `N` times.
- **Pros:**
  - Uses the maximum possible amount of data for training in each iteration, leading to a potentially less biased estimate of model performance.
  - No randomness involved in the splitting process; the result is fully deterministic and reproducible.
- **Cons:**
  - **Computationally very expensive:** Requires training the model `N` times, which is prohibitive for large datasets.
  - Can have high variance: Because the training sets in each iteration are almost identical (differing by only one point), the performance estimates from each fold are highly correlated, leading to potentially high variance in the final averaged estimate.

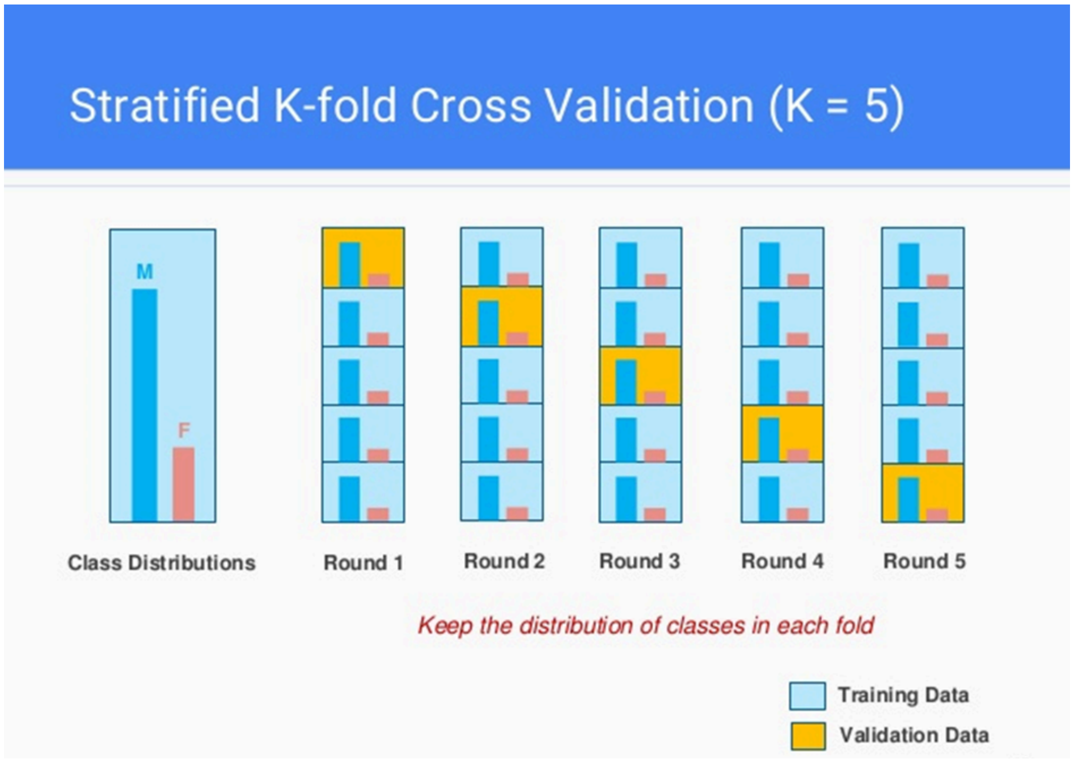
- **When to consider:** Primarily useful for very small datasets where maximizing training data in each fold is critical, and the high computational cost is manageable.

B. K-Fold Cross-Validation

- **How it works:** As described in the general procedure. The dataset is split into `K` folds. The model is trained `K` times, each time using a different fold as the validation set and the rest for training.
- Lets if we have 100 data points for training then if k=5 then in each 20 records will be used for validation and remaining for training and this happens in cycle.
- **Choosing K:** Common values are 5 or 10. The choice involves a trade-off:
    - *Higher K:* Less bias (training sets are larger), higher variance (training sets overlap more), higher computational cost. LOOCV is the extreme (`K=N`).
    - *Lower K:* More bias (training sets are smaller), lower variance (training sets overlap less), lower computational cost.
- **Pros:**
    - Provides a good balance between computational cost and obtaining a reliable performance estimate.
    - Generally considered a standard and robust method for model evaluation and tuning.
    - Lower variance in the performance estimate compared to LOOCV.
- **Cons:**
    - Performance estimates can have slightly more bias than LOOCV (since less data is used for training in each fold).
    - The specific split into folds involves randomness (unless seeded), so results might vary slightly across different runs.
    - Can be problematic for imbalanced datasets (see Stratified K-Fold).
- **When to use:** A good default choice for most regression and classification problems when the dataset is not extremely small or time-dependent.

C. Stratified K-Fold Cross-Validation

- **How it works:** A variation of K-Fold designed specifically for **classification** problems, especially useful when dealing with **imbalanced datasets** (where some classes have far fewer samples than others). It ensures that each fold maintains approximately the same percentage of samples for each target class as present in the original dataset.


Stratified K-fold Cross Validation (K = 5)
M
F
Class Distributions
Round 1
Round 2
Round 3
Round 4
Round 5
Keep the distribution of classes in each fold
Training Data
Validation Data

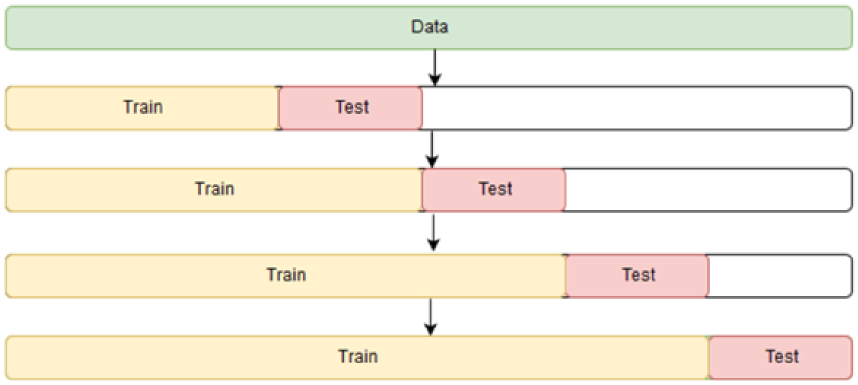
- **Pros:**
  - Ensures that all classes are represented in both training and validation sets within each fold, preventing situations where a fold might contain only samples from the majority class.
  - Leads to more reliable, less biased estimates of model performance (especially for metrics sensitive to class distribution like accuracy, precision, recall, F1-score) on imbalanced data.
- **Cons:**
  - Primarily applicable to classification tasks (though can sometimes be adapted for regression by stratifying based on binned target values).
- **When to use:** Essential for classification problems with imbalanced class distributions. Often recommended as the default K-Fold approach for classification.

D. Time Series Cross-Validation (e.g., Forward Chaining, Rolling Forecast Origin)

- **How it works:** Standard CV methods shuffle data randomly, which breaks the temporal order crucial in time series data (you shouldn't train on future data to predict the past). Time Series CV respects this order. Common approaches include:
  - **Expanding Window (Forward Chaining):**
    1. Train on data from time 1 to `t`, test on `t+1`.
    2. Train on data from time 1 to `t+1`, test on `t+2`.
    3. ...and so on, expanding the training window.
  - **Sliding Window (Rolling Forecast Origin):**

    1. Train on data from time `t-k` to `t`, test on `t+1`.
    2. Train on data from time `t-k+1` to `t+1`, test on `t+2`.
    3. ...and so on, using a fixed-size training window `k` that slides forward.

- **Pros:**
    - Provides a realistic estimate of how the model will perform when deployed sequentially in time.
    - Prevents data leakage from the future into the training set.
- **Cons:**
    - Generally uses less data for training compared to standard K-Fold (especially in early folds of the expanding window method).
    - Can be computationally intensive if models need frequent retraining.
    - Performance can depend on the initial training size or window size chosen.
- **When to use:** Absolutely necessary for any problem involving time-ordered data where the goal is to predict future values based on past values (e.g., forecasting sales, stock prices, weather).


Data
Train
Test
Train
Test
Train
Test
Train
Test