Learning K-Fold Cross-Validation: A Practical Guide with Python

Authored by

Mohammed looti

November 6, 2025

To accurately assess the predictive capability of any statistical or machine learning model, it is essential to measure how effectively its predictions align with unseen data. If we evaluate a model solely on the data used for training, we risk severe overfitting, leading to unreliable performance in real-world applications. Therefore, robust validation techniques are paramount for achieving generalizable results.

One of the most widely accepted and reliable methods for model evaluation is K-Fold Cross-Validation. This technique is designed to maximize the use of the available data while providing a stable and unbiased estimate of the model's test error rate. By systematically partitioning the dataset, K-Fold CV ensures that every observation is used for both training and validation at least once, dramatically reducing variance in the performance metric.

Understanding K-Fold Cross-Validation

K-Fold Cross-Validation is a powerful resampling procedure used to evaluate machine learning models on a limited data sample. The core idea is to divide the entire dataset into K subsets, or "folds," of approximately equal size. The process is then iterated K times, with each iteration serving a different fold as the holdout or validation set, and the remaining K-1 folds used for training the model.

This systematic rotation ensures a comprehensive evaluation, minimizing the bias associated with random train-test splits. The procedure for K-Fold Cross-Validation follows a standardized sequence of steps:

The original dataset is randomly partitioned into **k** groups, or folds, which are roughly equal in size.

One fold is designated as the validation set (the holdout set). The model is trained exclusively on the observations contained within the remaining **k-1** folds.

The trained model is then used to calculate a test error metric, such as the Mean Squared Error (MSE), on the observations in the single holdout fold.

Steps 2 and 3 are repeated **k** times, ensuring that each of the **k** folds serves as the validation set exactly once.

The overall test error estimate is calculated as the average of the **k** individual error metrics recorded during the iterations.

This tutorial provides a detailed, step-by-step implementation of K-Fold Cross-Validation for a simple linear model using the Python programming language and the versatile Scikit-learn library.

Step 1: Loading Essential Libraries and Modules

Before proceeding with the implementation, it is necessary to import all the required functions and modules. We rely heavily on the Scikit-learn library, specifically its `model_selection` module, which provides the tools necessary for managing cross-validation procedures like `KFold` and `cross_val_score`. We also utilize `LinearRegression` for our model, and `numpy` and `pandas` for efficient data handling and mathematical operations.

Ensure that these libraries are installed in your environment before executing the following code block, which sets up the computational environment:

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from numpy import mean
from numpy import absolute
from numpy import sqrt
import pandas as pd
```

These imports provide the foundation for defining our dataset, specifying the cross-validation strategy, training the regression model, and finally calculating the resulting error metrics.

Step 2: Creating the Sample Data Structure

For demonstration purposes, we will construct a small synthetic dataset. This dataset will be managed using a Pandas DataFrame, which is the standard structure for handling tabular data in Python. Our dataset includes a single response variable, `y`, that we aim to predict, based on two predictor variables, `x1` and `x2`. This simple structure is sufficient for illustrating the mechanics of K-Fold CV.

The code snippet below initializes the DataFrame with ten observations, defining the relationship we intend to model using linear regression. The features `x1` and `x2` will serve as the independent variables (predictors), and `y` will be the dependent variable (response).

```
df = pd.DataFrame({'y': ,
'x1': ,
'x2': })
```

Step 3: Implementing K-Fold CV and Calculating Mean Absolute Error (MAE)

With the data prepared, the next crucial step is to define the model and the cross-validation strategy. We will fit a multiple linear regression model to predict `y` using `x1` and `x2`. We initialize the `KFold` object, specifying `n_splits=10`, which means we will use 10 folds for validation. This is equivalent to Leave-One-Out Cross-Validation (LOOCV) since our dataset only contains 10 observations. We also set a `random_state` for reproducibility and ensure `shuffle=True` to randomize the fold assignments.

The `cross_val_score` function handles the iterative training and scoring process automatically. We specify the scoring metric as `neg_mean_absolute_error`. Scikit-learn returns negative error scores by convention, so we use `mean(absolute(scores))` to obtain the positive Mean Absolute Error (MAE).

#define predictor and response variables
X = df]
y = df

#define cross-validation method to use (k=10)
cv = KFold(n_splits=10, random_state=1, shuffle=True)

#build multiple linear regression model
model = LinearRegression()

#use k-fold CV to evaluate model
scores = cross_val_score(model, X, y, scoring='neg_mean_absolute_error',
cv=cv, n_jobs=-1)

#view mean absolute error
mean(absolute(scores))

3.6141267491803646

The resulting Mean Absolute Error (MAE) is approximately **3.614**. This value signifies the average magnitude of the errors in a set of predictions, without considering their direction. In practical terms, this means that, on average, our model's predictions deviate from the actual observed values by 3.614 units of `y`.

Step 4: Evaluating Performance using Root Mean Squared Error (RMSE)

While MAE is useful for understanding average error magnitude, another common and highly

sensitive metric for evaluating regression models is the Root Mean Squared Error (RMSE). RMSE is particularly sensitive to large errors because it squares the difference between predicted and actual values before averaging, thereby penalizing larger deviations more heavily. This metric is often preferred when large errors are disproportionately undesirable.

To calculate RMSE using K-Fold CV, we change the scoring parameter in `cross_val_score` to `neg_mean_squared_error`. In this example, we will also demonstrate flexibility by setting the number of folds `n_splits` to 5 instead of 10. Once the negative MSE scores are returned, we apply the square root function to the absolute mean of those scores to derive the final Root Mean Squared Error (RMSE).

#define predictor and response variables
X = df]
y = df

#define cross-validation method to use (k=5)
cv = KFold(n_splits=5, random_state=1, shuffle=True)

#build multiple linear regression model
model = LinearRegression()

#use k-fold CV to evaluate model
scores = cross_val_score(model, X, y, scoring='neg_mean_squared_error',
cv=cv, n_jobs=-1)

#view RMSE
sqrt(mean(absolute(scores)))

4.284373111711816

The calculated Root Mean Squared Error (RMSE) for the model, using 5-fold cross-validation on this data, is **4.284**. A fundamental principle in model evaluation is that lower values for both MAE and RMSE indicate superior model performance, as they demonstrate a closer fit between predicted values and observed data points.

Step 5: Selecting the Optimal Number of Folds (k)

The choice of *k*, the number of folds, is a critical hyperparameter in K-Fold Cross-Validation, influencing both the bias and variance of the error estimate. While we used k=10 in the first example and k=5 in the second, the ideal choice balances computational cost with statistical reliability.

A very small *k* (e.g., k=2) means the training sets are large, leading to less biased models but a higher variance in the test error estimate, as the validation set is large but sampled fewer times. Conversely, a very large *k* (such as k=N, which is LOOCV) results in highly biased models (training sets are very similar to the full dataset) but a low variance in the error estimate.

In standard data science practice, the value of *k* is typically chosen to be between 5 and 10. This range has been empirically shown to offer the best compromise, providing stable and reliable test error rates without excessive computational overhead. For most moderate to large datasets, setting *k* to 10 is considered the standard benchmark.

By using K-Fold Cross-Validation and comparing performance metrics like MAE and RMSE across different candidate models (e.g., Linear Regression vs. Ridge Regression), data scientists can confidently select the model that produces the lowest expected test error rates, ensuring its robustness when encountering new, unseen data.

Additional Resources

For those interested in exploring related validation techniques or deepening their understanding of the underlying models, the following resources are recommended:

An Introduction to K-Fold Cross-Validation

A Complete Guide to Linear Regression in Python

Leave-One-Out Cross-Validation in Python

An Introduction to K-Fold Cross-Validation
A Complete Guide to Linear Regression in Python
Leave-One-Out Cross-Validation in Python