Detailed Explanation of the 10-Fold Cross-Validation Process

Introduction

To reliably measure a machine learning model's performance, it is essential to test it on data it has not seen before. **K-Fold Cross-Validation** is a robust technique for achieving this by systematically using the entire dataset for both training and testing. This project implements a 10-fold cross-validation harness from scratch to provide a reliable estimate of the KNN algorithm's accuracy. This report details the process as implemented in `cross_validation.py` and used by `knn_controller.py`.

Step-by-Step Implementation Breakdown

The cross-validation process involves splitting the data, looping through training and testing cycles, and measuring performance.

Step 1: Splitting the Dataset into Folds (`cross_validation_split` function)

The foundation of the process is the division of the data into 10 equal parts, or "folds."

1. **Inputs**: The function takes the entire `dataset` (as a list of lists) and the number of folds (`n_folds=10`) as input.
2. **Random Splitting**: The function works by creating 10 empty folds. It then randomly selects rows from a copy of the original dataset and adds them to a fold one by one until it reaches the calculated fold size. This process is repeated for all 10 folds.
3. **Output**: This method ensures that each data point is placed into exactly one fold randomly. The function returns a list containing these 10 folds, ready for the evaluation loop.

Step 2: The Evaluation Loop (in `knn_controller.py`)

The main experiment controller uses the generated folds to run 10 iterations of training and testing.

1. **Iteration**: The controller script loops 10 times, once for each fold. In each iteration `i`, `fold i` is designated as the hold-out **test set**.
2. **Training Set Creation**: The **training set** for that iteration is created by combining all the other 9 folds into a single, larger dataset.
3. **Model Evaluation**: The KNN model is then "trained" on this newly formed training set (by being given access to it) and is asked to make predictions for each row in the test set.

Step 3: Measuring Performance (`accuracy_metric` function)

For each of the 10 folds, an accuracy score is calculated to measure how well the model performed.

1. **Comparison**: After predictions are made for a test fold, the `accuracy_metric` function is called. It takes the list of `predicted` labels and the list of `actual` labels from the test fold as input.
2. **Calculation**: The function iterates through the lists and counts how many predictions were correct. It then calculates the percentage of correct predictions.
3. **Storing Scores**: This accuracy percentage is the score for that specific fold, and it is stored in a list of scores.

After the evaluation loop has completed all 10 iterations, the result is a list of 10 accuracy scores. The **mean** of these scores provides a robust estimate of the model's overall performance, while the **standard deviation** indicates the stability of its performance across different subsets of the data.