Detailed Explanation of the Weighted KNN Algorithm

Introduction

The k-Nearest Neighbors (KNN) algorithm is a classification method that determines the class of a new data point by observing the classes of its closest neighbors in the feature space. This project implements an advanced version of this algorithm, **Weighted KNN**, from scratch. Unlike standard KNN where every neighbor gets an equal vote, the weighted implementation gives more influence to closer neighbors, resulting in a more nuanced and often more accurate prediction. This report details the specific implementation found in `knn_weighted.py`.

Step-by-Step Implementation Breakdown

The classification of a single test data point involves two main functions: `get_neighbors` and `predict_classification`.

Step 1: Finding the Neighbors and Their Distances (`get_neighbors` function)

The first step is to identify the `k` data points from the training set that are closest to the new, unclassified data point.

1. **Distance Calculation**: The function iterates through every single row in the `train_dataset`. For each training row, it calculates the distance to the `test_row`.
2. **Flexible Distance Metrics**: A key feature of this design is its flexibility. The specific distance function (e.g., `euclidean_distance`, `manhattan_distance`, `hamming_distance`) is passed as a parameter, allowing the algorithm to be tailored to the type of data being analyzed.
3. **Sorting**: The calculated distances are stored along with their corresponding training rows as `(train_row, distance)` tuples. This entire list is then sorted in ascending order based on the distance.
4. **Selection**: The function returns the first `k` elements from the sorted list. The result is a list containing the `k` nearest neighbors and their precise distances, which is crucial for the weighting step.

Step 2: Making a Weighted Prediction (`predict_classification` function)

Once the nearest neighbors are identified, this function uses their distances to make a final prediction.

1. **The Weighting Scheme**: The core of this extension is the weighting method. The weight for each neighbor's vote is calculated as the inverse of its distance (`weight = 1 / distance`). This elegantly ensures that a smaller distance results in a larger weight, giving closer neighbors more influence.
2. **Weighted Voting**: The algorithm initializes a dictionary to store the cumulative weights for each potential class label. It then iterates through the `k` neighbors. For each neighbor, it adds its calculated weight to its class's total in the dictionary.
3. **Edge Case Handling**: The implementation robustly handles the case where a distance is exactly 0 (an identical match in the training data). In this scenario, that neighbor's class is returned immediately as the prediction, as no closer match is possible.
4. **Final Decision**: After summing the weights for all classes, the function finds the class label with the highest total weight and returns it as the final prediction.

This weighted approach provides a more sophisticated prediction than a simple majority vote and, as shown in the project's results, can significantly improve the model's performance.