Homework 1: Analysis of K-NN and PW-NN Algorithms

Introduction

In this paper, we examine the K-Nearest Neighbors (K-NN) algorithm and its improved variant, Proximity-Weighted Nearest Neighbors (PW-NN), applied to a dataset for classifying uppercase letters of the English alphabet. The objective is to analyze the effectiveness of both approaches using different distance metrics such as Euclidean and Manhattan, as well as to examine their advantages and disadvantages. Through comparative analysis of the results, we provide recommendations on which algorithm and metric are most suitable for this specific problem.

Dataset Overview

1. **Source:** The dataset used comes from the UCI Machine Learning Repository and was created for classification problems of uppercase letters of the English alphabet.
2. **Content:** The dataset consists of 20,000 examples of letters, each represented by 16 numerical features, including:
    - x-box: horizontal position of the letter.
    - y-box: vertical position of the letter.
    - width: width of the letter.
    - height: height of the letter.
    - onpix: number of black pixels that compose the letter.
    - and other characteristics that describe the shape and appearance of the letter.
3. **Objective:** The objective is to classify each example into one of the 26 letters of the alphabet, using machine learning algorithms to recognize the letters based on their features.

Algorithms Used

K-Nearest Neighbors (K-NN)

The K-NN algorithm is a supervised learning technique that classifies a new data point based on its K nearest neighbors in the training dataset. The basic idea is that nearby points, in terms of distance, tend to belong to the same class.

1. **Distance Calculation:** The classification of a new point is based on the distance between that point and the points in the dataset. The two distance metrics used are:
   - *Euclidean Distance:* measures the "normal" distance between two points in Euclidean space.

$$d_{\text{Euclidean}}(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

   - *Manhattan Distance:* measures the distance between two points by summing the differences along the orthogonal axes.

$$d_{\text{Manhattan}}(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

2. **Classification:** Once the distances are calculated, the point is assigned to the class of the majority of its K nearest neighbors.

Proximity-Weighted Nearest Neighbors (PW-NN)

PW-NN is an improved variant of K-NN, which takes into account the distances by weighting the neighbors. The central idea is that closer neighbors have a greater influence on the final decision compared to those that are farther away.

1. **Distance Calculation:** As in K-NN, distances are calculated from all points in the training dataset.
2. **Weight Assignment:** Each neighbor is weighted inversely proportional to its distance from the test point. The commonly used formula is:

$$w_i = \frac{1}{d(x_{\text{test}}, x_i)}$$

3. **Classification:** The label of the new point is determined by aggregating the votes of the weighted neighbors and choosing the class with the highest total weight.

Advantages and Disadvantages

Advantages of PW-NN:

- Better accuracy in datasets with non-uniformly distributed classes.
- Greater robustness against outliers and noise in the data.

Disadvantages of PW-NN:

- Higher computational intensity compared to K-NN.
- Performance may depend on the choice of the weighting function and distance metric.

Implementation

This section describes the implementation process used to compare the performance of the unweighted K-Nearest Neighbors (K-NN) algorithm and the weighted one. The code uses the letter recognition dataset, applying different distance metrics and values of K to evaluate accuracy.

Libraries Used:

The main libraries used for the implementation are as follows:

- `pandas`: for manipulating and managing the dataset.
- `matplotlib.pyplot`: for any data visualizations.
- `sklearn.neighbors.KNeighborsClassifier`: to implement the K-NN algorithm.
- `sklearn.model_selection.train_test_split`: to split the dataset into training and test sets.
- `sklearn.metrics.accuracy_score`: to calculate the accuracy of predictions.
- `sklearn.preprocessing.LabelEncoder`: to convert letter labels into numerical values.

Data Loading and Preprocessing

The dataset is loaded using the `pandas.read_csv()` function, and subsequently the features and labels are extracted. The labels, which represent the letters of the alphabet, are converted into numerical values using the `LabelEncoder` class, which facilitates the classification process.

The dataset is then divided into two parts: a training set and a test set. This is done using the `train_test_split` function from `sklearn`, maintaining a fixed value for the random seed (`random_state = 0`) to ensure reproducibility of the results.

Implementation of Unweighted K-NN

The `evaluate_knn()` function implements the unweighted K-Nearest Neighbors algorithm. This function takes as input the number of neighbors (K) and the distance metric (e.g., Euclidean or Manhattan). Using the `KNeighborsClassifier`, the model is trained with the training data (`X_train, y_train`) and then tested on unseen data (`X_test`), returning the model's accuracy.

Implementation of PW-NN (Weighted K-NN)

The `evaluate_weighted_knn()` function extends the K-NN algorithm by adding weights based on distance. This is done through the `weights='distance'` argument, which sets the weights so that closer neighbors have more influence than those that are farther away. Again, the classifier is trained and tested, and the accuracy is printed and returned.

Running Tests

The code iterates over different values of K and the two distance metrics (Euclidean and Manhattan) to evaluate both the unweighted K-NN algorithm and the weighted one. The results are printed with the accuracy for each combination of parameters. Here's an improved version of your results analysis:

Results Analysis

This analysis compares the accuracy of two K-NN algorithms—unweighted and weighted—based on different values of K and the chosen distance metrics. This comparison aims to identify the optimal combination of parameters for letter recognition tasks.

K-NN (Unweighted)

Observations:

- At $K = 1$, the Euclidean distance achieves the highest accuracy of 0.9580, indicating its effectiveness in leveraging the nearest neighbor.
- Starting from $K = 3$, the Manhattan distance begins to slightly outperform the Euclidean distance, suggesting its growing relevance with increasing neighbors.

K	Euclidean Accuracy	Manhattan Accuracy
1	0.9580	0.9514
3	0.9548	0.9528
5	0.9520	0.9522
7	0.9508	0.9528
9	0.9466	0.9484

Table 1: Accuracy comparison of unweighted K-NN algorithms

- As K increases, there is a general trend of decreasing accuracy for both distance metrics, indicating a potential over-smoothing effect.

Proximity-Weighted K-NN (Weighted)

Weighted K	Euclidean Accuracy	Manhattan Accuracy
1	0.9580	0.9514
3	0.9668	0.9640
5	0.9648	0.9624
7	0.9614	0.9572
9	0.9544	0.9516

Table 2: Accuracy comparison of weighted K-NN algorithms

Observations:

- For $K = 1$, the performance mirrors that of the unweighted case, indicating similar foundational effectiveness.
- At $K = 3$, the Proximity-Weighted K-NN achieves an impressive accuracy of 0.9668 using the Euclidean metric, demonstrating the advantage of weighting neighbors.
- The weighted algorithm consistently outperforms the unweighted version across various values of K, particularly for lower values, highlighting its robustness.

Conclusion

The analysis of K-NN and Proximity-Weighted K-NN (PW-NN) algorithms in the context of letter classification demonstrates that both methods are effective for this task. However, the PW-NN algorithm consistently outperforms the traditional K-NN across different metrics and values of K.

This study emphasizes the importance of utilizing a weighted approach with an optimal K value to achieve superior classification accuracy. It underscores

the significance of selecting the right distance metric and weighing mechanism within the K-NN family of algorithms, ultimately contributing to improved performance in letter recognition tasks.