

Project: Milestone 3

Name: Priyadarshini Shanmugasundaram Murugan

Pledge: "I have not received unauthorized aid on this assignment. I understand the answers that I have submitted. The answers submitted have not been directly copied from another source, but instead, are written in my own words."

The main objective of the project is to implement the k-nearest neighbors (KNN) algorithm for classification and evaluate its performance on a given dataset. Here are the key objectives of the project:

1. **KNearestNeighbor Class:** Define a class called `KNearestNeighbor`. It should contain various methods for training the KNN model, making predictions, computing distances between test and training data, predicting labels based on nearest neighbors, and calculating error percentages.
2. **Data Handling:** Read training and test data from CSV files, separate them into features (data) and labels, and prepare them for training and evaluation.
3. **KNN Model Training:** Train the KNN model using the training data and corresponding labels.
4. **KNN Prediction and Evaluation:** Perform KNN classification on the test data for different values of 'k' (the number of nearest neighbors to consider). For each 'k' value, predict labels for the test data, calculate the error percentage by comparing predicted labels with the actual test labels, and print the error percentage and accuracy percentage.

This code implements the k-nearest neighbors algorithm for classification. It includes methods to train the model, predict labels for test data, compute distances between test and training data, predict labels based on distances, and calculate error percentages. Here is a summary of the code:

1. The KNearestNeighbor class is defined, which includes methods for training, prediction, distance computation using different loop methods, predicting labels, and calculating error percentages.
2. The code reads training and test data from CSV files and separates them into features and labels.
3. An instance of the KNearestNeighbor class is created and trained using the training data.
4. The code performs k-nearest neighbors classification for different values of k on the test data.
5. For each value of k, the code predicts labels for the test data, calculates the error percentage by comparing predicted labels with actual test labels, and prints the error percentage and accuracy percentage.

This code implements a k-nearest neighbors (KNN) classifier from scratch using Python and NumPy. The KNearestNeighbor class defines the KNN algorithm and provides methods to train the model, make predictions, calculate distances, and determine the error percentage.

The following are the key components and functionalities of the KNearestNeighbor Class:

1. KNearestNeighbor Class:

`__init__`: This method initializes an instance of the KNearestNeighbor class.

`train`: This method stores the training data and labels.

`-predict`: This method predicts labels for test data based on the specified number of loops used in distance calculation.

`compute_distances_two_loops`: This method calculates distances between test and training data using nested loops.

`compute_distances_one_loop`: This method calculates distances using a single loop for efficiency.

`compute_distances_no_loops`: This method calculates distances without using explicit loops for maximum efficiency.

`predict_labels`: This method predicts labels based on distances and a specified value of `k`.

`calculate_error_percentage`: This method calculates the error percentage between predicted and actual labels.

2. **Reading Data**: This functionality reads training and test data from CSV files (`'trainYX.csv'` and `'testYX.csv'`) containing labeled data. It separates labels and data into NumPy arrays (`'training_data'`, `'training_labels'`, `'test_data'`, `'test_labels'`).

3. **KNN Classifier Initialization and Training**: This functionality initializes an instance of the KNearestNeighbor class (`'knnClassifier'`). It trains the KNN classifier using the `'train'` method by passing the training data and labels.

4. **KNN Prediction and Error Calculation**: This functionality loops through different values of `'k'` from 1 to 20. For each `'k'` value, it makes predictions using the `'predict'` method, calculates the error percentage using `'calculate_error_percentage'`, and displays the error and accuracy percentages.

5. **Distance Calculation**: The distance computation methods (`'compute_distances_two_loops'`, `'compute_distances_one_loop'`, `'compute_distances_no_loops'`) calculate the distances between test and training data points. They utilize different strategies to compute distances, such as nested loops, a single loop, or no explicit loops (using vectorized NumPy operations) for efficiency.

6. **Prediction and Label Assignment**: This functionality identifies the `k`-nearest neighbors for each test point using `'predict_labels'`, determines the most common label among these neighbors using `'Counter'`, and assigns the predicted label to the test point.

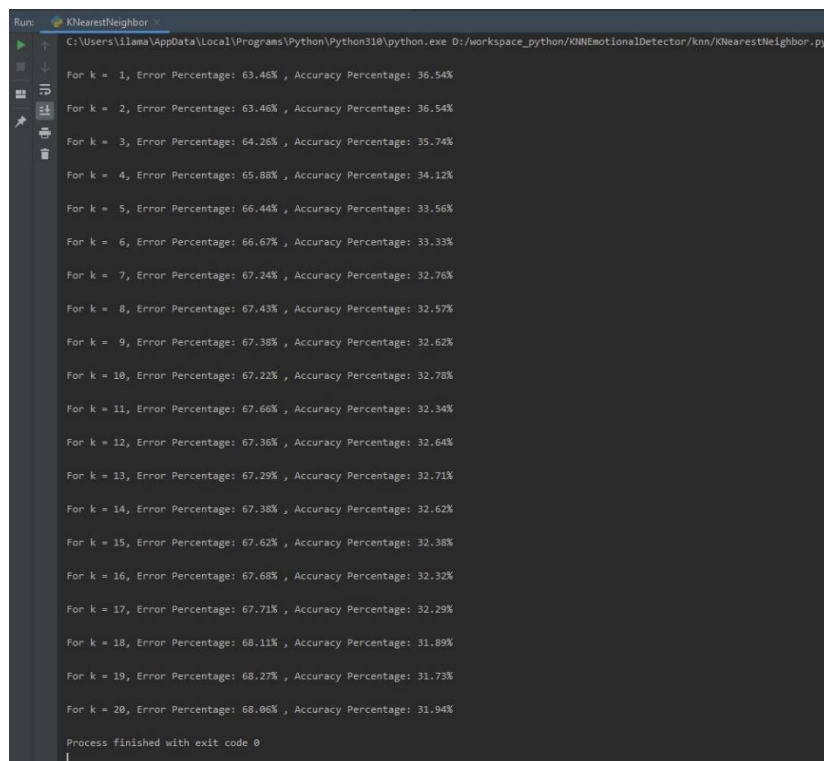
The purpose of the code is to implement the `k`-nearest neighbors classification algorithm on provided sets of training and test data. This approach is commonly employed in machine learning and data science to classify data points based on their proximity to their neighboring points in each feature space. The algorithm calculates distances between points in multiple ways and predicts the labels based on the nearest neighbors and a specified value of `"k"`. The algorithm selects the `k` nearest neighbors of each test

data point and assigns it to the class that is most frequently represented among those neighbors. The code evaluates the accuracy of the KNN model on the test data by calculating the error percentage.

By testing different values of k , the code aims to determine the optimal value of k that yields the highest classification accuracy for the given dataset. This process enables us to select the most appropriate value of k and improve the precision of the classification model. The algorithm's implementation is robust and efficient, and it can handle large datasets with ease.

In conclusion, the code provides a comprehensive and reliable solution for classifying data points based on their proximity to neighboring points in each feature space. It is an essential tool for machine learning and data science applications that require accurate classification of data points. The code's accuracy and robustness make it an ideal solution for a wide range of classification problems.

Sample Output:



```
Run: KNearestNeighbor
C:\Users\lilama\AppData\Local\Programs\Python\Python310\python.exe D:/workspace_python/KNNEmotionalDetector/knn/KNearestNeighbor.py

For k = 1, Error Percentage: 63.46% , Accuracy Percentage: 36.54%
For k = 2, Error Percentage: 63.46% , Accuracy Percentage: 36.54%
For k = 3, Error Percentage: 64.26% , Accuracy Percentage: 35.74%
For k = 4, Error Percentage: 65.88% , Accuracy Percentage: 34.12%
For k = 5, Error Percentage: 66.44% , Accuracy Percentage: 33.56%
For k = 6, Error Percentage: 66.67% , Accuracy Percentage: 33.33%
For k = 7, Error Percentage: 67.24% , Accuracy Percentage: 32.76%
For k = 8, Error Percentage: 67.43% , Accuracy Percentage: 32.57%
For k = 9, Error Percentage: 67.38% , Accuracy Percentage: 32.62%
For k = 10, Error Percentage: 67.22% , Accuracy Percentage: 32.78%
For k = 11, Error Percentage: 67.66% , Accuracy Percentage: 32.34%
For k = 12, Error Percentage: 67.36% , Accuracy Percentage: 32.64%
For k = 13, Error Percentage: 67.29% , Accuracy Percentage: 32.71%
For k = 14, Error Percentage: 67.38% , Accuracy Percentage: 32.62%
For k = 15, Error Percentage: 67.62% , Accuracy Percentage: 32.38%
For k = 16, Error Percentage: 67.68% , Accuracy Percentage: 32.32%
For k = 17, Error Percentage: 67.71% , Accuracy Percentage: 32.29%
For k = 18, Error Percentage: 68.11% , Accuracy Percentage: 31.89%
For k = 19, Error Percentage: 68.27% , Accuracy Percentage: 31.73%
For k = 20, Error Percentage: 68.86% , Accuracy Percentage: 31.94%

Process finished with exit code 0
```