K-Nearest Neighbors (k-NN) Algorithm Implementation Report:

Rahul Jha | rahuljha@umd.edu | University of Maryland College Park

I implemented a few functionalities in the k-Nearest Neighbors (k-NN) algorithm in Python. The following tasks were successfully completed:

Accomplishments Description

The goal of this research was to apply cross-validation to the k-Nearest Neighbors (k-NN) algorithm in order to find the ideal number of neighbors (k). The principal achievements consist of:
• **Using k-fold cross-validation**: To ensure robustness in model evaluation, the training data was divided into 5 folds for the purpose of training and validating the model across a number of iterations.
• **Accuracy calculations for various values of k**: By evaluating k values between 1 and 100, the algorithm was able to determine which value of k was best for the dataset.
• **Creation of distance computation techniques**: Euclidean distance was calculated without the need for explicit loops, streamlining the procedure for quicker k-NN algorithm assessment.

Key tasks:

- Numpy.array split was used to split the data, producing balanced folds for the training and validation sets.
- To hold the accuracy values for various values of k, a dictionary called k_to_accuracies was introduced.
- Four folds were used to train the model, while the remaining fold served as the validation set for calculating accuracy. For every fold and every value of k, this was repeated.
- To display the model's performance during the cross-validation procedure, accuracy values were printed.

Missed Points:

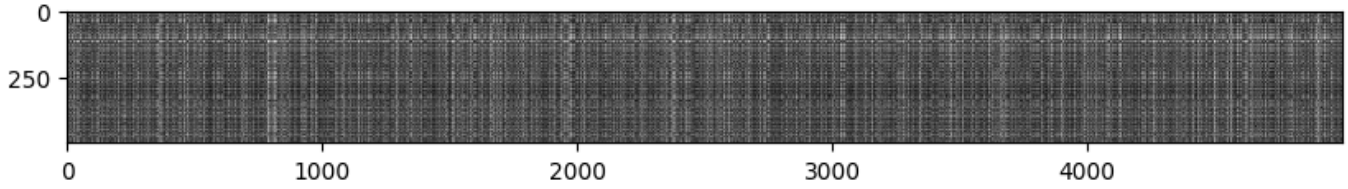
- The lack of any graphical or visual depiction of the data may have made it more difficult to understand how k and accuracy relate to one another.
- A thorough analysis of computational complexity might have shown ways to increase efficiency.

Explanation of Implementation Decisions

Euclidean Distance: Euclidean distance is the most often used distance measure in k-NN and performs well in low-dimensional feature spaces, it was selected as the metric for calculating the distances between data points. It offers a simple way to calculate the geometric proximity between two places.

Three Distance Calculation Methods:

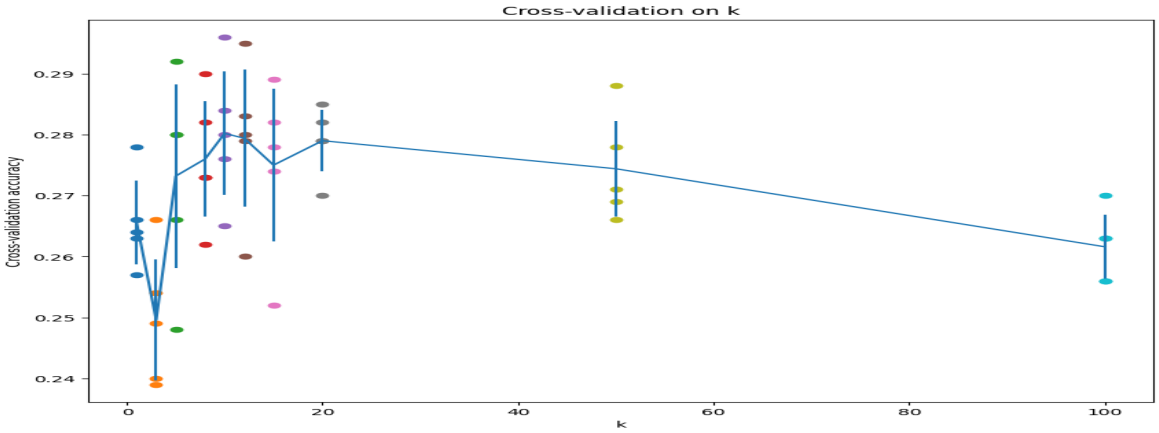
- The two-loop method was used to illustrate how to compute distances in a simple yet ineffective manner.
- The one-loop approach simplifies things and maximizes NumPy's broadcasting features.
- The no-loop technique, which is more effective for big datasets, maximizes performance by using matrix operations to compute distances instead of explicit loops.
- Two loop version took 38.592718 seconds, one loop version took 47.866453 seconds, No loop version took 0.423912 seconds.



Cross-Validation: It divides the data into numerous training and validation sets, k-Fold cross-validation provides for a more robust estimation of model performance, which is why it was selected. By making sure the

model is assessed on every data point, this method helps prevent overfitting and yields a more accurate estimate of the ideal k value.

Parameter k Selection: Using cross-validation, different values of k (1, 3, 5, 8, 10, 12, 15, 20, 50, 100) were evaluated. This aids in determining the optimal k for this particular dataset in order to balance variance and bias. Larger numbers result in underfitting of the training data, while smaller values typically overfit the data. The best value for K in this dataset is 10.



Critical Thinking and Analysis

Performance Trade-offs: Vectorized operations in the no-loop method greatly increase the speed of distance computation, which is essential when working with huge datasets. On the other hand, the performance gain might not be as apparent for smaller datasets, and more straightforward implementations might be adequate.

Effect of k on Accuracy: Smaller values of k can, as predicted, cause overfitting, in which case the model becomes more susceptible to noise in the data. On the other hand, higher values of k smooth out predictions by taking into account more neighbors, but they may also cause underfitting, which occurs when the model becomes overly generalized. A sweet spot between these extremes was identified with the aid of the cross-validation data.

Possible Improvements: Future work could involve significantly optimizing the algorithm, particularly for higher-dimensional data, by including quicker search techniques like KD-trees or Ball Trees. Furthermore, grid search-based hyperparameter tuning could improve the selection of k and other parameters.

Issues with Memory Consumption: Overloading RAM caused enormous matrix sizes and quadratic memory expansion, which resulted in high RAM utilization and Google Colab crashes.The way without a loop:

dists = np.sqrt(np.sum(np.square(X), axis=1).reshape(-1, 1) + np.sum(np.square(self.X_train), axis=1) - 2 * np.dot(X, self.X_train.T))

The Reason Behind These Outcomes:
1. Low k overfitting: The model is overly susceptible to anomalies.
2. Underfitting with High k: Too many neighbors averaged together smoothes the bounds of decisions.

Next Actions:
1. Examine Additional Metrics: Examine the Minkowski or Manhattan lengths.
2. Apply PCA or KD-Trees to increase query speed and decrease memory usage: Cut down on dimensionality to improve productivity.

Supporting References

Scikit-learn-Documentation-on-k-NN, NumPy-array_split-Documentation , Euclidean-distance-explained, Vectorization-in-numpy, Choosing-Optimal-k-in-k-NN, Cross-Validation