Simple k-Nearest Neighbor Classifier in Python

prepared by Hakkı Kokur / 150120033

1. Introduction

K-Nearest Neighbors is a supervised machine learning algorithm that classifies data points based on the majority class of their k nearest neighbors in the feature space.

This implementation demonstrates a complete K-Nearest Neighbors (KNN) classifier for predicting whether to play tennis based on weather conditions. The system processes weather data through several key stages: data preparation, distance calculation, neighbor selection, and prediction evaluation.

2. Methodology

The foundation begins with data loading and preprocessing, where categorical features are transformed using **one-hot encoding** to convert them into a numerical format suitable for distance calculations. This transformation is essential as KNN requires numerical values to compute distances between instances.

The core of the algorithm lies in the distance calculation between instances, supporting both Euclidean and Manhattan metrics. The Euclidean distance measures the straight-line distance between points in the feature space, while Manhattan distance calculates the sum of absolute differences along each dimension. This flexibility allows users to choose the most appropriate distance metric for their specific use case. **The algorithm doesn't root square the results because it increases performance. It compares the squared distance between instances.** And the default algorithm is euclidean.

The neighbor selection process involves calculating distances between a test instance and all training instances, sorting these distances, and selecting the k nearest neighbors. The value of k is user-defined and significantly influences the model's behavior - smaller values make the model more sensitive to local patterns, while larger values create smoother decision boundaries.

Prediction generation uses a majority voting mechanism among the k nearest neighbors. This democratic approach assigns the most common class label among the neighbors as the prediction for the test instance. The system maintains the original dataset's integrity while adding columns for predictions and evaluation metrics.

The evaluation script implements a comprehensive confusion matrix (True Positives, False Positives, True Negatives, False Negatives) and calculates key performance metrics: accuracy, precision, and recall. Accuracy measures overall correct predictions, precision indicates the reliability of positive predictions, and recall shows the model's ability to find all positive instances.

The implementation includes a practical interface allowing users to:

1. Train new models or load existing ones
2. Choose between distance metrics
3. Set the number of neighbors (k)
4. View detailed prediction results and performance metrics

This system embodies the lazy learning paradigm of KNN, where the model simply stores the training data and defers computation until prediction time. This approach offers flexibility in handling new data but requires storing the entire training dataset and computing distances during prediction.

The modular design separates core functionalities into distinct functions, promoting code reusability and maintenance. The system handles data persistence through CSV files, enabling model storage and reloading for future use without retraining

Core Functions:

```
def get_k_nearest_neighborhoods(model, instance2, main_column, k,
calculation_type):
```

```
def determine_prediction(neighbors, main_column):
```

```
def compute_distance(instance, instance2, main_column,
calculation_type):
```

3. Results

```
[3 rows x 12 columns]
Nearest Neighborhoods for 10:
    Play Tennis  Outlook_Overcast  Outlook_Rain  ...  Wind_Strong  Wind_Weak  Distance
10         Yes                 0             0  ...           1          0       0.0
1           No                 0             0  ...           1          0       4.0
6          Yes                 1             0  ...           1          0       4.0

[3 rows x 12 columns]
Nearest Neighborhoods for 11:
    Play Tennis  Outlook_Overcast  Outlook_Rain  ...  Wind_Strong  Wind_Weak  Distance
11         Yes                 1             0  ...           1          0       0.0
13          No                 0             1  ...           1          0       2.0
1           No                 0             0  ...           1          0       4.0

[3 rows x 12 columns]
Nearest Neighborhoods for 12:
    Play Tennis  Outlook_Overcast  Outlook_Rain  ...  Wind_Strong  Wind_Weak  Distance
12         Yes                 1             0  ...           0          1       0.0
2          Yes                 1             0  ...           0          1       2.0
6          Yes                 1             0  ...           1          0       4.0

[3 rows x 12 columns]
Nearest Neighborhoods for 13:
    Play Tennis  Outlook_Overcast  Outlook_Rain  ...  Wind_Strong  Wind_Weak  Distance
13          No                 0             1  ...           1          0       0.0
3          Yes                 0             1  ...           0          1       2.0
11         Yes                 1             0  ...           1          0       2.0
```

This visual shows which instances in the model were found to be closest to it, according to the index order in the test data. Since the same data set is used to train and test the model, the first instance actually refers to the same weather conditions as the tested and learned instance.

```
(ml_projects) yangwenli@hyperion:~/base/a5Projects/school/ml-projects/simple_knn$ python main.py
Train new model(y/n):
Enter the calculation type(euclidean/manhattan):
Enter the k number: 3
Model with prediction and confusion matrix:
    Play Tennis  Outlook_Overcast  Outlook_Rain  ...  Wind_Weak  Play Tennis Prediction  Confusion Matrix
0           No                 0             0  ...          1                      No                TN
1           No                 0             0  ...          0                      No                TN
2          Yes                 1             0  ...          1                     Yes                TP
3          Yes                 0             1  ...          1                      No                FP
4          Yes                 0             1  ...          1                     Yes                TP
5           No                 0             1  ...          0                     Yes                FN
6          Yes                 1             0  ...          0                     Yes                TP
7           No                 0             0  ...          1                      No                TN
8          Yes                 0             0  ...          1                     Yes                TP
9          Yes                 0             1  ...          1                     Yes                TP
10         Yes                 0             0  ...          0                     Yes                TP
11         Yes                 1             0  ...          0                      No                FP
12         Yes                 1             0  ...          1                     Yes                TP
13          No                 0             1  ...          0                     Yes                FN

[14 rows x 13 columns]
Accuracy:  0.7142857142857143
Precision:  0.7777777777777778
Recall:  0.7777777777777778
```

In the table above, we see metrics such as Confusion Matrix, Accuracy, Precision and Recall that show the model's mood. Since our data is limited, the rates are very similar. As the test dataset, I used the same dataset that I used to train the model.

4. Discussion

The accuracy of 71.43% suggests that the model correctly predicts the outcome in approximately 7 out of 10 cases. While this is a good starting point, there is potential for enhancement.

Both precision and recall are at 77.78%, indicating a balanced performance in terms of predicting positive instances (i.e., predicting "Yes" for playing tennis). Precision shows that when the model predicts "Yes," it is correct 77.78% of the time. Recall indicates that the model identifies 77.78% of all actual "Yes" instances.

The confusion matrix provides a detailed breakdown of the model's performance:
True Positives (TP): 7
True Negatives (TN): 3
False Positives (FP): 2
False Negatives (FN): 2
The model makes an equal number of false positive and false negative errors, which suggests a balanced but not perfect performance.

The current implementation uses one-hot encoding for categorical features, which is appropriate. However, **additional feature engineering techniques could be explored. For example, creating interaction features or using polynomial features might capture more complex relationships in the data.**

The value of k (number of neighbors) is a critical hyperparameter in KNN. The current implementation uses k=3, but this value might not be optimal. Performing hyperparameter tuning using techniques like cross-validation can help identify the best k value for the dataset.

Distance-based algorithms like KNN are sensitive to the scale of the features. Ensuring that all features are on a similar scale through normalization or standardization can improve the model's performance. Techniques like Min-Max scaling or Z-score normalization can be applied to the dataset.

KNN is a lazy learning algorithm, meaning it defers computation until prediction time. This can be computationally expensive, especially with large datasets. Implementing efficient data structures like KD-Trees or Ball Trees can speed up the nearest neighbor search process.

5. Conclusion

The KNN model for predicting whether to play tennis based on weather conditions shows reasonable performance with an accuracy of 71.43% and balanced precision and recall. However, there are several areas for improvement, including increasing the dataset size, feature engineering, hyperparameter tuning, scaling, and handling imbalanced data. By addressing these performance issues, the model's accuracy and reliability can be significantly enhanced, making it a more robust tool for prediction tasks.

References

- *Introduction to Machine Learning* by Ethem Alpaydin
- *Pattern Recognition and Machine Learning* by Christopher Bishop
- https://www.geeksforgeeks.org/k-nearest-neighbours/
- https://medium.com/@madhuri15/knn-classifier-implementation-best-practices-and-tips-part-i-6288181c8eed