Name Surname : Muhammed Enes Gündüz
Student ID: 150120038

K-Nearest Neighbors (KNN) Classifier Implementation

1. Introduction

The **K-Nearest Neighbors (K-NN)** algorithm is a simple and widely-used machine learning technique used for both classification and regression tasks. The basic principle of K-NN is that for a given test instance, it finds the `k` closest training instances based on a distance metric (usually Euclidean or Manhattan distance). The class label of the test instance is assigned based on the majority class of these `k` nearest neighbors. This makes K-NN a **non-parametric** method, meaning it does not make any assumptions about the underlying distribution of the data.

Objectives of the Assignment

The objective of this assignment was to implement a K-Nearest Neighbors classifier for predicting whether a game of tennis will be played based on weather conditions. The goal was to:

- Implement K-NN with both Euclidean and Manhattan distance metrics.
- Evaluate the performance of the classifier using accuracy and confusion matrix metrics.
- Analyze the results and identify any misclassifications.

---

2. Methodology

Data Preparation

The data used in this project consisted of two main files:

1. **Training Data (`play_tennis_data.json`)**: A set of instances containing weather conditions (Outlook, Temperature, Humidity, Wind) and a label (PlayTennis) indicating whether tennis was played on that day.
2. **Test Data (`test_data.json`)**: A set of instances with the same format used to evaluate the trained model.

The data was pre-processed as follows:

- **Encoding Categorical Features**: The categorical features (Outlook, Temperature, Humidity, Wind) were encoded using **one-hot encoding**, which creates binary columns for each category, transforming the categorical data into numerical format.

- **Handling Missing Values**: Any missing or misaligned data was handled appropriately by filling or removing rows, ensuring that the dataset was ready for training and evaluation.
- **Splitting Features and Labels**: The target variable (`PlayTennis`) was separated from the feature columns, and only the features were used for training.

K-Nearest Neighbors (K-NN) Implementation

The K-NN classifier was implemented in Python using the following steps:

1. **Distance Calculation**:
    - The Euclidean distance was computed using the formula:

$$\text{distance} = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

    - The Manhattan distance was computed using the formula:

$$\text{distance} = \sum_{i=1}^{n}|x_i - y_i|$$

2. **Model Training**:
    - The training data was encoded and stored in a JSON file for persistence.
    - The K-NN classifier was trained on the training data, with the number of neighbors (`k`) and the distance metric (`Euclidean` or `Manhattan`) being specified by the user.
3. **Model Prediction**:
    - For each test instance, the classifier calculated the distance to all training instances, sorted them, and selected the top `k` nearest neighbors. The predicted class was determined by the majority class among these neighbors.
4. **Evaluation**:
    - Accuracy was calculated as the proportion of correct predictions out of the total predictions.
    - The **confusion matrix** was generated, which showed the number of true positives, false positives, true negatives, and false negatives.

Handling Challenges

One challenge encountered during the implementation was ensuring that the dimensions of the test instance matched those of the training data. Any mismatch in the number of features or categorical encoding could cause issues. This was addressed by ensuring the proper

alignment of features between the training and test datasets using reference columns during encoding.

---

3. Results

The model was evaluated on a test dataset after training. The user was prompted to choose the value for `k` (number of neighbors) and the distance metric (Euclidean or Manhattan). Below are the performance metrics based on the test data:

Accuracy: The proportion of correctly classified instances.
Example output:
`Accuracy on test data: 0.80`
Confusion Matrix: The confusion matrix was calculated as follows:This resulted in:

- **True Positives (TP)** = 3
- **False Positives (FP)** = 1
- **True Negatives (TN)** = 1
- **False Negatives (FN)** = 0

These values were logged in a `classification_log.txt` file for later review.

---

4. Discussion

Analysis of Results

The model demonstrated a high level of accuracy (e.g., 80%) on the test data, indicating that the classifier is effective for this particular dataset. The confusion matrix showed that the model made few misclassifications, particularly for the "Yes" class.

Misclassifications

- **False Positives (FP)**: The model predicted that tennis would be played when it was not. This could be due to an overlap in the feature space between classes.
- **False Negatives (FN)**: The model predicted that tennis would not be played when it actually was. This may be due to the lack of sufficient distinct features for clear separation.

Limitations

- **Curse of Dimensionality**: K-NN can perform poorly when the number of features (dimensions) is too high, as the distance between points becomes less meaningful. The dataset used here was small, so this did not significantly impact the performance.
- **Choice of `k`**: The choice of `k` is critical. A small value of `k` might lead to overfitting, while a large value might smooth out the model too much.

---

5. Conclusion

This project demonstrated the implementation and evaluation of a K-Nearest Neighbors classifier. By selecting the number of neighbors (k) and the distance metric (Euclidean or Manhattan), the model was able to predict whether tennis would be played based on weather conditions. The model's accuracy was high, and the confusion matrix provided insights into the model's strengths and weaknesses.

6. References

- Slides on Classroom
- K-Nearest Neighbors (KNN) in Python
- ChatGPT for bug fix