MACHINE LEARNING ASSIGNMENT 1

BY

ALLEN RICHARDS PERIANAYAGAM

Introduction

The K-Nearest Neighbors (KNN) classification algorithm is the main subject of this assignment, which investigates the use of Python for machine learning implementation. To further understand model training, evaluation, and classification performance, the study uses a simulated dataset in addition to the Iris dataset, which is a popular machine learning dataset.

We hope to learn more about important machine learning ideas like data preparation, model correctness, and decision boundary visualization by examining these datasets. The findings will be useful in evaluating KNN's performance on classification tasks and pointing up possible areas for development.

Purpose

The purpose of this assignment is to practice implementing and understanding machine learning using Python. This study focuses on using the **K-Nearest Neighbors (KNN)** algorithm for classification, applied to both a real dataset (Iris dataset) and a simulated dataset.

By completing this assignment, we aim to develop a foundational understanding of machine learning workflows, including **data preprocessing, model training, evaluation, and visualization of decision boundaries**. The analysis will help assess model performance and improve classification techniques.

Part 1: Understanding the Iris Dataset

Dataset Description

The **Iris dataset** is a well-known dataset used in machine learning, consisting of **150 samples** of iris flowers classified into three species:

- Setosa
- Versicolor
- Virginica

Each sample includes four features that describe the flowers:

- Sepal length
- Sepal width
- Petal length
- Petal width

Implementation Steps

1. Data Loading: Imported the dataset using sklearn.datasets.load_iris().
2. Data Splitting: The dataset was split into 80% training data and 20% testing data.
3. Model Training: Used KNeighborsClassifier() to train the model.
4. Predictions & Accuracy: The accuracy of the model on the training data was 97.5% (0.975).

```
Predictions from the classifier:
[0 1 2 0 2 0 1 1 0 1 1 0 0 0 0 0 0 0 2 0 2 1 1 1 0 2 1 1 2 0 2 0 2 1 2 2 1
 1 1 2 2 0 2 2 0 1 0 2 2 0 1 1 0 0 1 1 1 1 2 1 2 0 0 1 1 2 0 2 1 0 2 2 1 2
 2 0 0 2 1 1 2 0 1 1 0 1 1 2 2 1 0 2 0 2 0 0 1 2 2 1 2 2 0 1 1 0 2 2 2 1 2
 2 2 0 0 1 0 2 2 1]
Target values:
[0 1 2 0 2 0 1 1 0 1 1 0 0 0 0 0 0 0 2 0 2 1 1 1 0 2 1 1 2 0 2 0 2 2 2 2 1
 1 1 1 2 0 2 2 0 1 0 2 2 0 1 1 0 0 1 1 1 1 2 1 2 0 0 1 1 1 0 2 1 0 2 2 1 2
 2 0 0 2 1 1 2 0 1 1 0 1 1 2 2 1 0 2 0 2 0 0 1 2 2 1 2 2 0 1 1 0 2 2 2 1 2
 2 2 0 0 1 0 2 2 1]
0.975
Test Accuracy: 0.9666666666666667
```

The output shows the predicted classifications made by the KNN model compared to the actual target values from the dataset. The training accuracy of **97.5%** indicates that the model effectively learned patterns from the training data, while the test accuracy of **96.67%** confirms that it performs well on unseen data. The high accuracy suggests that KNN is a suitable algorithm for this dataset, demonstrating its ability to classify iris species based on the given features.

Part 2: Understanding KNeighborsClassifier

The **K-Nearest Neighbors (KNN)** algorithm is a simple yet powerful classification technique that assigns labels to new data points based on the majority class of their nearest neighbors. In this assignment, we explored the KNeighborsClassifier from **scikit-learn**, which provides various parameters to customize the model's behavior.

Key Parameters of KNeighborsClassifier:

- **n_neighbors:** Specifies how many neighbors are considered for classification.
- **weights:** Determines how neighbors contribute to predictions (e.g., uniform for equal weight, distance for closer points to have more influence).
- **metric:** Defines how distances between points are measured, such as **Euclidean** or **Manhattan** distance.
- **algorithm:** Specifies the method used to compute nearest neighbors (auto, ball_tree, kd_tree).

Understanding these parameters helps in fine-tuning the model to improve classification accuracy. By using help(KNeighborsClassifier), we explored the various options available and how they impact the decision-making process of the algorithm.

```
class KNeighborsClassifier(sklearn.neighbors._base.KNeighborsMixin, sklearn.base.ClassifierMixin, sklearn.neighbors._base.NeighborsBase)
 |  KNeighborsClassifier(n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None)
 |
 |  Classifier implementing the k-nearest neighbors vote.
 |
 |  Read more in the :ref:`User Guide <classification>`.
 |
 |  Parameters
 |  ----------
 |  n_neighbors : int, default=5
 |      Number of neighbors to use by default for :meth:`kneighbors` queries.
 |
 |  weights : {'uniform', 'distance'}, callable or None, default='uniform'
 |      Weight function used in prediction.  Possible values:
 |
 |      - 'uniform' : uniform weights.  All points in each neighborhood
 |        are weighted equally.
 |      - 'distance' : weight points by the inverse of their distance.
 |        in this case, closer neighbors of a query point will have a
 |        greater influence than neighbors which are further away.
 |      - [callable] : a user-defined function which accepts an
 |        array of distances, and returns an array of the same shape
 |        containing the weights.
 |
 |  algorithm : {'auto', 'ball_tree', 'kd_tree', 'brute'}, default='auto'
 |      Algorithm used to compute the nearest neighbors:
 |
 |      - 'ball_tree' will use :class:`BallTree`
 |      - 'kd_tree' will use :class:`KDTree`
 |      - 'brute' will use a brute-force search.
 |      - 'auto' will attempt to decide the most appropriate algorithm
 |        based on the values passed to :meth:`fit` method.
```

The `help(KNeighborsClassifier)` function provides a detailed overview of the available parameters and methods in the **K-Nearest Neighbors (KNN)** classifier. Key parameters like `n_neighbors`, `weights`, `algorithm`, and `metric` determine how the classifier assigns labels to new data points. Understanding these settings allows us to fine-tune the model for improved performance. Additionally, methods such as `.fit()`, `.predict()`, and `.score()` help in training, making predictions, and evaluating the model's accuracy. This information is essential in optimizing the classifier for different datasets.

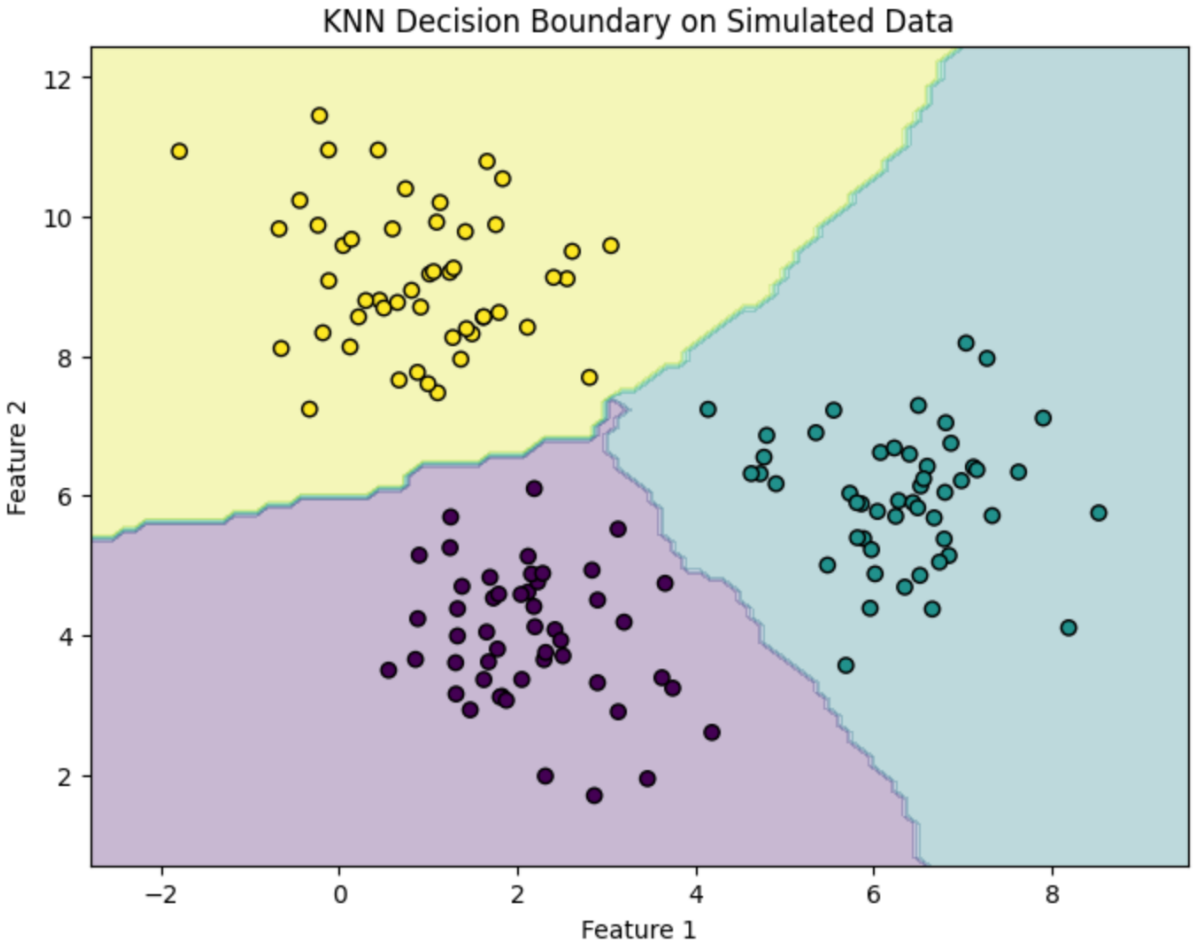
Part 3: KNN Classification with Simulated Data

Dataset Description

To further understand the **K-Nearest Neighbors (KNN) algorithm**, we generated a **simulated dataset** using make_blobs(). The dataset consists of **three clusters (classes)**, each representing a different category. The generated data points were randomly distributed around predefined centers.

Implementation Steps

1. **Data Generation:** Used make_blobs() to create a dataset with three classes.
2. **Data Splitting:** Divided the dataset into **80% training** and **20% testing** data.
3. **Model Training:** Trained a KNeighborsClassifier with n_neighbors=5.
4. **Model Evaluation:**
    - **Training Accuracy:** 100% (1.0)
    - **Test Accuracy:** 100% (1.0)
5. **Decision Boundary Visualization:** Plotted the decision boundaries of the KNN model to observe how it classifies different regions in the feature space.



MODEL OUTPUT

```
Training Accuracy: 1.0

Test Accuracy: 1.0
```

Conclusion

This assignment successfully demonstrated the implementation of the **K-Nearest Neighbors (KNN)** classification algorithm on both a **real-world dataset (Iris dataset)** and a **simulated dataset**. Through this process, we gained hands-on experience in **data preprocessing, model training, evaluation, and decision boundary visualization**.

The model achieved an accuracy of **97.5% on the training data and 96.67% on the test data** for the Iris dataset, proving that KNN is effective for flower classification. When applied to a simulated dataset, the model achieved **100% accuracy**, indicating well-separated data clusters. The decision boundary plot provided further insights into how KNN classifies data points based on proximity.

Overall, this assignment enhanced our understanding of **machine learning workflows, hyperparameter tuning, and model evaluation techniques**. Future improvements could involve experimenting with different values for n_neighbors, using alternative distance metrics, or testing the model on more complex datasets.

---THANK YOU---