Experiment 6 : Implementation of the K-Nearest Neighbors Algorithm from Scratch

Lab Experiment:

Total Marks: 100

1. Learning Objectives

Upon successful completion of this assignment, students will be able to:

- Implement a complete machine learning algorithm (K-Nearest Neighbors) using Python and the NumPy library.
- Develop and structure a modular machine learning project, separating concerns such as data handling, utility functions, and model logic.
- Implement fundamental machine learning pipeline components, including data loading, preprocessing, and a custom data splitting function.
- Conduct Exploratory Data Analysis (**EDA**) using `matplotlib` to derive insights about data separability.
- Evaluate model performance using an accuracy metric and analyze the impact of the key **hyperparameter**, 'k'.
- Generalize a custom-built classifier by applying it to a novel dataset.

---

2. Introduction

The **K-Nearest Neighbors (KNN)** algorithm is a non-parametric, instance-based learning algorithm used for classification and regression. For a classification task, its fundamental principle is to assign a class to a new data point based on the majority class of its 'k' nearest neighbors in the feature space. Proximity is typically determined using a distance metric, such as the Euclidean distance.

This assignment will guide you through building a complete KNN classifier from scratch to classify species of Iris flowers from the well-known **Iris dataset**. You will then validate your implementation on the **Wine dataset**.

---

3. Prerequisites

Before beginning, ensure you have a Python environment with the following libraries installed. You can install them using pip:

```
pip install numpy pandas matplotlib scikit-learn ucimlrepo
```

---

4. Experiment Tasks

You are required to build a machine learning project from scratch. Follow the structured tasks below.

Task 1: Project Scaffolding and Data Handling (15 Marks)

The first step is to establish a clean project structure and create a script for data acquisition and preparation.

1. **Create the Project Directory:** In your working environment, create a new folder for this project.
2. **Create Python Files:** Inside the project folder, create the following five empty Python files:
   - `data.py`: For all data loading and **preprocessing** logic.
   - `utils.py`: For general utility functions.
   - `knn_classifier.py`: For the implementation of the KNN model.
   - `eda.py`: For scripts related to exploratory data analysis.
   - `main.py`: The main script to execute the full training and evaluation pipeline.
3. **Implement the Data Loader (`data.py`):**
   - Import necessary libraries (`ucimlrepo`, `pandas`).
   - Use the `fetch_ucirepo` function to load the **Iris dataset** (ID: 53).
   - Extract the features into a pandas **DataFrame** named `X` and the targets into a **DataFrame** named `y`.
   - The target labels in the original dataset are formatted as "Iris-species". Preprocess the `y` DataFrame to remove the "Iris-" prefix, leaving only the species name (e.g., 'setosa').

- Convert the final `X` and `y` **DataFrames** into NumPy arrays to be used by the classifier.

Task 2: Utility Function Implementation (15 Marks)

A crucial step in evaluating any machine learning model is to split the data into separate sets for training and testing. You will implement this function in `utils.py`.

1. Implement train_test_split: Define a function with the following signature:

   `def train_test_split(X, y, test_size=0.2, random_state=None):`

   - The function should take features `X`, labels `y`, a `test_size` proportion, and an optional `random_state` for reproducibility.
   - If `random_state` is set, use it to seed NumPy's random number generator (`np.random.seed`).
   - Generate a randomly shuffled sequence of indices corresponding to the number of samples in the dataset.
   - Partition the shuffled indices into training and testing sets based on the `test_size`.
   - Use the partitioned indices to slice the `X` and `y` arrays.
   - Return the four resulting NumPy arrays in the specified order: `X_train`, `X_test`, `y_train`, `y_test`.

Task 3: K-Nearest Neighbors Classifier Implementation (30 Marks)

This is the central task of the assignment. In `knn_classifier.py`, you will build the classifier. The implementation should be contained within a class.

1. Implement the Distance Metric: Outside the class, create a standalone function to calculate the Euclidean distance between two data points:

   `def euclidean_distance(x1, x2):`

   This function should accept two NumPy arrays $(x1, x2)$ and return their scalar distance. The formula is: $d(x_1, x_2) = \sqrt{\sum_{i=1}^{n}(x_{1i} - x_{2i})^2}$

2. **Implement the `KNNClassifier` Class:**

   - **`__init__(self, k=3)`:** The constructor should accept an integer `k` and store it as an instance attribute.
   - **`fit(self, X_train, y_train)`:** This method should accept the training features and labels and store them within the instance (e.g., `self.X_train` and `self.y_train`).

- `predict(self, X_test)`: This method will generate predictions for a set of test data. It should iterate through each sample in `X_test` and use a private helper method (detailed next) to determine its class. It must return a NumPy array containing the predictions.
- _predict(self, x): This private helper method will contain the core **KNN** logic for a single data point x. It must perform the following steps:

  a. Calculate the distance from x to every point in `self.X_train`.

  b. Identify the indices of the k training samples with the smallest distances. (Hint: `np.argsort` is highly effective for this).

  c. Retrieve the labels corresponding to these k indices from `self.y_train`.

  d. Determine the most frequently occurring label (majority vote) among these k neighbors and return it as the prediction. (Hint: collections.Counter can simplify this step).

Task 4: Exploratory Data Analysis (10 Marks)

Before finalizing the model, it is critical to understand the data's underlying structure. In `eda.py`, you will write a script to visualize the **Iris dataset**.

1. **Implement the Visualization Script:**
   - Import `matplotlib.pyplot` and the data variables from your `data.py` file.
   - Generate a series of scatter plots to visualize the relationship between every unique pair of features in the dataset.
   - In each plot, ensure that data points are colored according to their species. This will help visualize the class separability.
   - Properly label the axes and provide a title for each subplot. Include a legend to identify the species.
   - Use `plt.show()` to display the final figure.

Task 5: Model Integration and Evaluation (30 Marks)

In `main.py`, you will integrate all the components you have built to train, evaluate, and analyze your **KNN** classifier.

1. **Assemble the Pipeline:**
   - Import your `KNNClassifier`, the data from `data.py`, and the `train_test_split` function from `utils.py`. Also, import `numpy`.

- Split the Iris data into training and testing sets using your `train_test_split` function (`test_size=0.2`, `random_state=42`).
- Instantiate `KNNClassifier` with `k=3`.
- Train the classifier using the `.fit()` method on the training data.
- Generate predictions for the test set using the `.predict()` method.

2. **Calculate Accuracy:**
   - Implement logic to compare the predicted labels against the true test labels (`y_test`).
   - Calculate the classification accuracy using the formula:

     $$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

   - Print the final accuracy to the console, formatted clearly.
3. **Hyperparameter Tuning and Analysis:**
   - Modify your script to train and evaluate the model iteratively for a range of `k` values: `[1, 3, 5, 7, 9, 11, 15]`.
   - Store the accuracy for each value of `k`.
   - After the loop, generate a line plot of "Accuracy vs. k-value" using `matplotlib`. Label the axes appropriately.
4. **Generalization to a New Dataset:**
   - Modify your `data.py` script to load the **Wine dataset** (ID: 109). Note that this dataset may not require the same pre-processing for its target variable; inspect the data and adjust accordingly.
   - Re-run your `main.py` script using the best value of `k` you identified from the previous step.

---

5. Submission Guidelines

Submit a single `.zip` archive containing the following:

1. **Source Code:** All five completed Python files (`main.py`, `data.py`, `eda.py`, `knn_classifier.py`, `utils.py`).
2. **PDF Report:** A formal report named `StudentID_Lab_Report.pdf` that includes:
   - A brief analysis of the EDA plots from Task 4, answering: Which feature pair provides the best class separation? Is any class inherently easier to distinguish?

- The final classification accuracy achieved on the Iris dataset (with k=3).
- The "Accuracy vs. k-value" plot generated in Task 5.
- An analysis of the plot: which value of `k` yielded the best performance? Provide a brief explanation for why very small or very large values of `k` can be sub optimal.
- The final accuracy achieved on the Wine dataset.
- A conclusion summarizing your key learning and any challenges encountered during the implementation.