Experiment 4

Aim

Implement K-Nearest Neighbors (KNN) and evaluate model performance

Theory:

1. Dataset Source

- **Source:** UCI Machine Learning Repository
- **Dataset Link:** https://archive.ics.uci.edu/ml/datasets/iris

2. Dataset Description

The Iris dataset is a classic dataset used for **multiclass classification** problems. It contains measurements of iris flowers from three different species.

Features:

- **SepalLengthCm** – Sepal length in centimeters
- **SepalWidthCm** – Sepal width in centimeters
- **PetalLengthCm** – Petal length in centimeters
- **PetalWidthCm** – Petal width in centimeters
- **Species** – Target variable with three classes:
    - Iris-setosa
    - Iris-versicolor
    - Iris-virginica

Dataset Size:

- **Total Instances:** 150
- **Number of Features:** 4 (plus 1 target variable)
- **Samples per Class:** 50

3. Mathematical Formulation of the Algorithm

K-Nearest Neighbors (KNN) Classifier

KNN is a **non-parametric, instance-based learning algorithm**.
For a given test sample, it finds the **k nearest neighbors** in the feature space and assigns the class that appears most frequently among them.

Distance Metric (Euclidean Distance)

$d(x,xi)=\sum j=1n(xj-xij)2$d(x,xi)=j=1∑n(xj−xij)2

Where:

- xx = Test sample
- xixi = Training sample
- nn = Number of features

Prediction Rule

$y^=mode(yi1,yi2,...,yik)$y^=mode(yi1,yi2,...,yik)

Where:

- yijyij = Labels of the k nearest neighbors
- $y^$y^ = Predicted class

4. Algorithm Limitations

- Sensitive to the choice of **k (number of neighbors)**
- Sensitive to **feature scaling** and irrelevant features
- Computationally expensive for large datasets (distance calculation required for all training samples)
- Poor performance on **imbalanced or noisy datasets**

5. Methodology / Workflow

Steps Followed:

1. **Data Loading:**
   Load the Iris dataset into a pandas DataFrame.
2. **Data Preprocessing:**
   - Drop the `Id` column (if present)
   - Separate features (X) and target (y)
   - Split data into training and testing sets
3. **Model Training:**
   Train the KNN classifier with k=5k=5.
4. **Prediction:**
   Predict species labels for the test dataset.
5. **Evaluation:**
   - Compute Accuracy
   - Generate Classification Report
   - Create Confusion Matrix

6. **Visualization:**
   Plot and save the confusion matrix heatmap.

6. Performance Analysis

- **Accuracy:** 1.0 (100%) on the test dataset
- **Classification Report:**
  - Precision = 1.0
  - Recall = 1.0
  - F1-Score = 1.0 for all three classes
- **Confusion Matrix:**
  - No misclassifications observed

Interpretation

KNN performs exceptionally well on the Iris dataset because:

- The classes are well-separated.
- The dataset is balanced.
- The feature space is low-dimensional.

Perfect classification is common for this dataset using KNN with a proper train-test split.

7. Hyperparameter Tuning

Tuning Process:

- Perform cross-validation for different values of **k**.
- Select the value of **k** that provides the highest validation accuracy.

Impact of k:

- **Small k (e.g., 1):** May lead to overfitting (high variance).
- **Large k:** May lead to underfitting (high bias).
- **Cross-validation:** Helps find the optimal balance for better generalization.

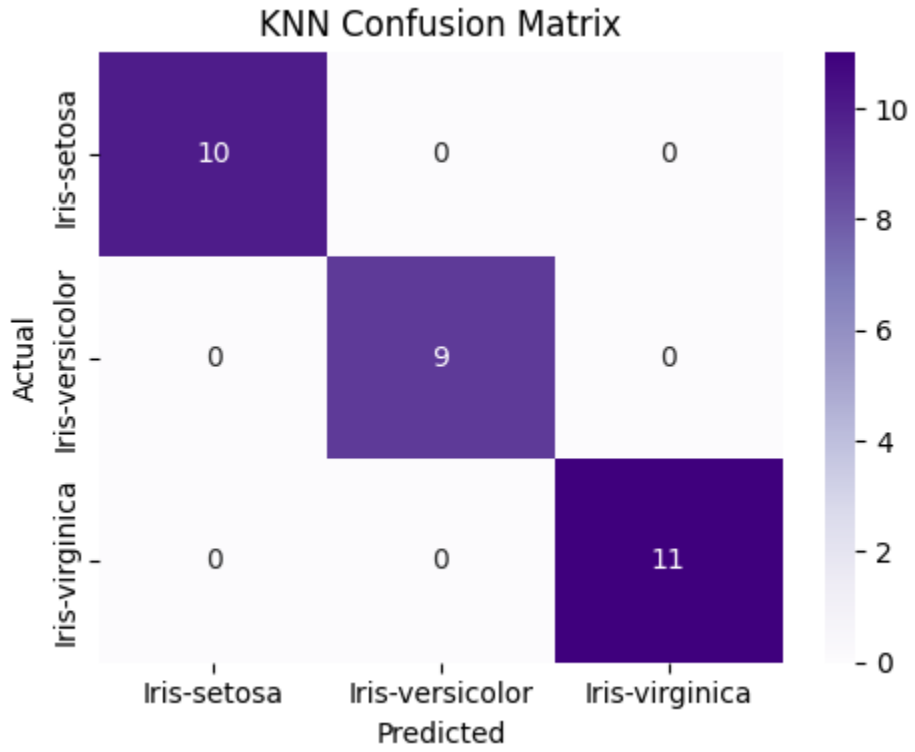
Code:

```
import os
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, accuracy_score,
confusion_matrix
```

```
import matplotlib.pyplot as plt
import seaborn as sns

# Dynamically get the path to the CSV file relative to this script
script_dir = os.path.dirname(os.path.abspath(__file__))
csv_path = os.path.join(script_dir, 'archive (3)', 'Iris.csv')

# Load dataset
# The Iris dataset usually has an 'Id' column, which we will drop

data = pd.read_csv(csv_path)
if 'Id' in data.columns:
   data = data.drop('Id', axis=1)

# Assume the last column is the target
y = data.iloc[:, -1]
X = data.iloc[:, :-1]

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# K-Nearest Neighbors Classifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

print('KNN Results:')
print('Accuracy:', accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Purples', xticklabels=knn.classes_,
yticklabels=knn.classes_)
plt.title('KNN Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.tight_layout()
plt.savefig(os.path.join(script_dir, 'knn_confusion_matrix.png'))
plt.close()
```

Output:

```
(.venv) [chirapython -u "/home/chirag/Desktop/ML_EXP/exp4/knn_classifier.py"
KNN Results:
Accuracy: 1.0
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00         9
 Iris-virginica       1.00      1.00      1.00        11

       accuracy                           1.00        30
      macro avg       1.00      1.00      1.00        30
   weighted avg       1.00      1.00      1.00        30
```



Conclusion:

The K-Nearest Neighbors (KNN) algorithm performs exceptionally well on the Iris dataset, achieving perfect classification accuracy due to the dataset's balanced structure and clearly separable classes. By selecting an appropriate value of *k* and applying proper preprocessing, KNN proves to be a simple yet highly effective method for multiclass classification problems like Iris.