Name:Khushi S Singh
D15A / 26

Experiment 4

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

Theory:

1. Dataset Source

Dataset Name: **Iris Dataset**
Source: Kaggle / UCI Machine Learning Repository

Dataset Link:
https://www.kaggle.com/datasets/uciml/iris

The Iris dataset is a standard machine learning dataset used for classification tasks. It contains measurements of iris flowers belonging to three different species and is commonly used for testing classification algorithms.

2. Dataset Description

The Iris dataset consists of measurements of iris flowers which are used to classify them into three species. Each record represents a flower sample with four numerical features.

Dataset Features

Feature	Description
sepal length	Length of sepal in centimeters
sepal width	Width of sepal in centimeters
petal length	Length of petal in centimeters
petal width	Width of petal in centimeters
species	Type of iris flower (Target Variable)

Dataset Characteristics

- Dataset Type: **Multiclass Classification**
- Number of Records: **150**
- Number of Features: **4**
- Target Variable: **species**
- Number of Classes: **3**

Class Labels:

- 0 → Setosa
- 1 → Versicolor
- 2 → Virginica

All features are numerical and continuous. The dataset contains **no missing values**, so no data cleaning was required.

The dataset is balanced, meaning each class contains approximately equal number of samples.

3. Mathematical Formulation of the Algorithm

K-Nearest Neighbors (KNN) is a supervised learning algorithm that classifies data points based on the closest training samples in feature space. The classification is performed by measuring the distance between the test sample and all training samples.

Euclidean Distance Formula

$$d = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + (x_3 - y_3)^2 + (x_4 - y_4)^2}$$

Where:

- x = Test sample
- y = Training sample
- n = Number of features

The class label is determined using majority voting:

$$Predicted\ Class = Majority(K\ nearest\ neighbors)$$

If most of the nearest neighbors belong to a particular class, the new data point is assigned that class.

Feature scaling is required because distance calculations depend on feature magnitudes.

4. Algorithm Limitations

Although KNN is simple and effective, it has several limitations:

- Computationally expensive for large datasets because distances must be calculated for all training samples.
- Requires feature scaling since features with larger values dominate distance calculations.
- Sensitive to noise and outliers in the dataset.
- Choosing an inappropriate value of K may reduce classification accuracy.
- Requires storing the entire dataset in memory.

KNN works best on **small datasets with clearly separated classes**, such as the Iris dataset.

5. Methodology / Workflow

Step 1: Dataset Loading

The Iris dataset was loaded into the Python environment using the Pandas and Scikit-learn libraries.

Dataset inspection included:

- Viewing first few records
- Checking dataset structure
- Checking missing values

Step 2: Data Preprocessing

The dataset was prepared for model training.

The following steps were performed:

- Independent variables (features) and dependent variable (target) were separated.

Input Features:

- sepal length
- sepal width
- petal length
- petal width

Target Variable:

- species

Feature scaling was applied using **StandardScaler** to normalize feature values.

Scaling ensures that all features contribute equally to distance calculations.

Step 3: Train-Test Split

The dataset was divided into training and testing datasets.

- Training Data = 80%
- Testing Data = 20%

Training data was used to train the model and testing data was used to evaluate performance.

Step 4: Model Training

The KNN classifier was implemented with:
K = 5

The model learned from the training dataset by storing feature values and class labels.

Predictions were generated for the testing dataset.

Step 5: Data Visualization

Several visualizations were generated to understand the dataset and model performance.

Scatter Plot

Petal Length vs Petal Width scatter plot was created.

This visualization showed clear separation between the three iris species, indicating that classification is possible.

Confusion Matrix Heatmap

A confusion matrix heatmap was generated to visualize classification performance.

The heatmap showed that all samples were correctly classified.

Accuracy vs K Graph

Accuracy was plotted against different K values.

The graph showed that accuracy remained constant across K values from 1 to 10, indicating stable model performance.

6. Performance Analysis

Model Results

Accuracy:1.0
This means the model correctly classified **100% of test samples**.

Confusion Matrix

$$\begin{bmatrix} 10 & 0 & 0 \\ 0 & 9 & 0 \\ 0 & 0 & 11 \end{bmatrix}$$

Interpretation:

- 10 Setosa flowers correctly classified
- 9 Versicolor flowers correctly classified
- 11 Virginica flowers correctly classified

No misclassification occurred.

Classification Report

- Precision = **1.00**
- Recall = **1.00**
- F1 Score = **1.00**

These values indicate perfect classification performance.

The scatter plot confirms that iris species are well separated, which explains the high accuracy.

7. Hyperparameter Tuning

Hyperparameter tuning was performed by testing different values of K.

Tested values: K= 1 to 10

Accuracy was calculated for each value of K.

Results showed that:

- Accuracy remained **1.0 for all values of K**
- Model performance was stable
- No overfitting observed

Best K value selected:K = 5

Impact of Hyperparameter Tuning

- Small K values can lead to overfitting.
- Large K values can reduce sensitivity.
- Moderate K values provide stable performance.

CODE and OUTPUT:

Step 1: Import Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

Step 2: Load Iris Dataset

```
iris = load_iris()

X = iris.data
y = iris.target

df = pd.DataFrame(X, columns=iris.feature_names)
df["species"] = y

print(df.head())
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   species
0        0
1        0
2        0
3        0
4        0
```

Step 3: Train–Test Split

```python
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

Step 4: Feature Scaling

```python
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Step 5: Train KNN Model

```python
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_scaled, y_train)

y_pred = knn.predict(X_test_scaled)
```

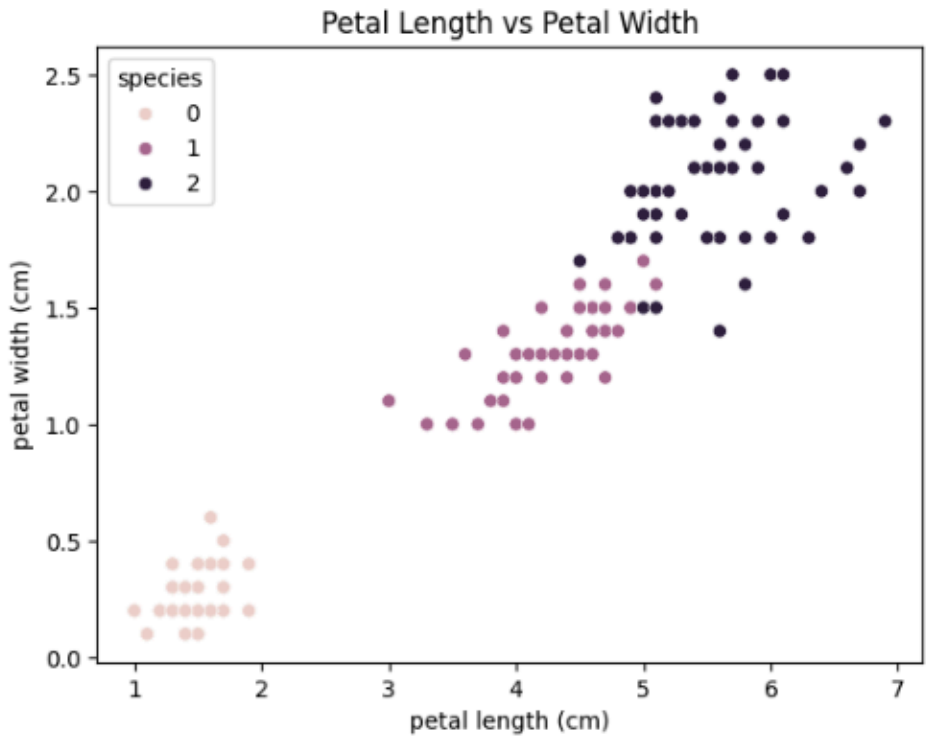
Step 6: Model Evaluation

```
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

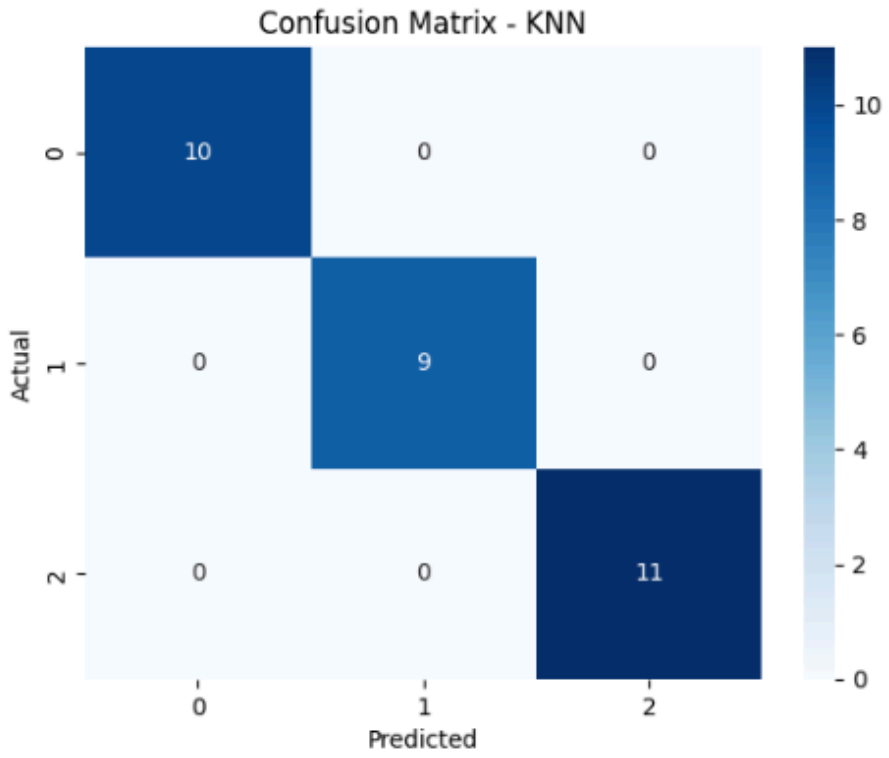
```
Accuracy: 1.0
Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       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
```

STEP 7: Visualization

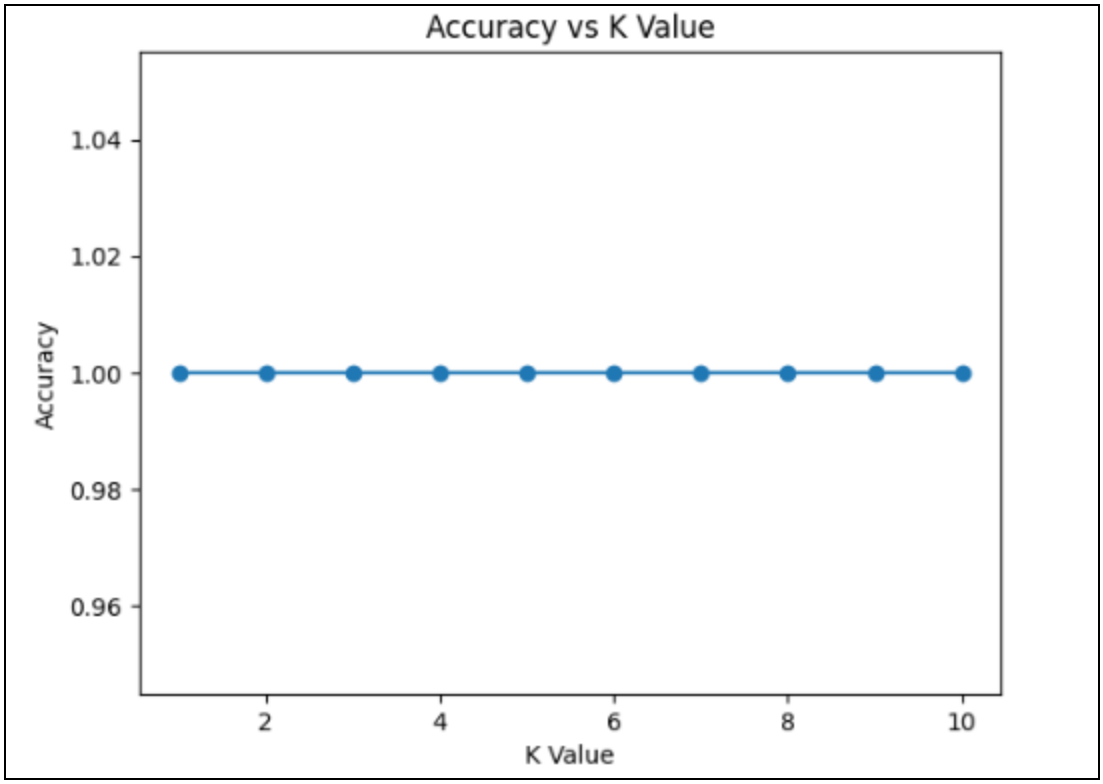
```
plt.figure()
sns.scatterplot(
    x=df["petal length (cm)"],
    y=df["petal width (cm)"],
    hue=df["species"]
)
plt.title("Petal Length vs Petal Width")
plt.show()
```



```
plt.figure()
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - KNN")
plt.show()
```




```
accuracy_list = []

for k in range(1, 11):
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train_scaled, y_train)
    pred = model.predict(X_test_scaled)
    accuracy_list.append(accuracy_score(y_test, pred))

plt.figure()
plt.plot(range(1, 11), accuracy_list, marker='o')
plt.xlabel("K Value")
plt.ylabel("Accuracy")
plt.title("Accuracy vs K Value")
plt.show()
```



Conclusion:

The experiment successfully demonstrated the implementation of the **K-Nearest Neighbors (KNN) algorithm** for classifying iris flower species. The model achieved **100% accuracy**, indicating excellent classification performance on the Iris dataset. The confusion matrix and classification report showed that all test samples were correctly classified.

The scatter plot showed clear separation between iris species, which helped the KNN model perform accurately. Hyperparameter tuning with different K values showed stable performance across all values. Overall, the experiment demonstrated that KNN is an effective algorithm for classification problems with well-separated data.