D15A / 25

EXPERIMENT 4

AIM:

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

THEORY

1. Dataset Source

Dataset Name: **Iris Dataset**
Source: Kaggle
Link: https://www.kaggle.com/datasets/uciml/iris

Originally introduced by Ronald A. Fisher (1936).
This dataset is commonly used for classification problems in machine learning.

2. Dataset Description

The Iris dataset contains measurements of iris flowers belonging to three different species.

Dataset Characteristics

- Total Samples: 150
- Number of Features: 4
- Number of Classes: 3
- Samples per Class: 50
- Type: Multiclass Classification
- Balanced Dataset

Features (Independent Variables)

1. Sepal Length (cm)
2. Sepal Width (cm)
3. Petal Length (cm)
4. Petal Width (cm)

All features are numerical and continuous.

Target Variable (Dependent Variable)

- Species of flower:
    - Iris-setosa
    - Iris-versicolor

- Iris-virginica

Relevant Characteristics

- Iris-setosa is linearly separable from the other two classes.
- Iris-versicolor and Iris-virginica show some overlap.
- Dataset is clean and contains no missing values.
- Suitable for demonstrating classification algorithms.

3. Mathematical Formulation of the Algorithm

K-Nearest Neighbors (KNN) is a non-parametric, instance-based learning algorithm used for classification and regression.

Step 1: Choose K

Select the number of nearest neighbors (K).

Step 2: Distance Calculation

The Euclidean distance between two points x and x_i is calculated as:

$$d(x, x_i) = \sqrt{\sum_{j=1}^{n} (x_j - x_{ij})^2}$$

Where:

- x = test sample
- x_i = training sample
- n = number of features

Step 3: Identify K Nearest Neighbors

Sort distances in ascending order and select the first K nearest samples.

Step 4: Majority Voting

For classification, the predicted class is determined by:

$$\hat{y} = \text{mode}(y_1, y_2, ..., y_k)$$

Where:

- $y_1, y_2, ..., y_k$ are class labels of K nearest neighbors.

Decision Rule

$$\hat{y} = \arg\max_c \sum_{i \in N_k(x)} I(y_i = c)$$

Where:

- $N_k(x)$ = set of K nearest neighbors
- I = indicator function

4. Algorithm Limitations

Although KNN performs well on small datasets, it has several limitations:

1. Computationally expensive for large datasets (lazy learning).
2. Requires feature scaling (distance-based algorithm).
3. Sensitive to noisy data and outliers.
4. Performance decreases in high-dimensional datasets (curse of dimensionality).
5. Choice of K significantly affects performance.
6. Poor performance on imbalanced datasets.

5. Methodology / Workflow

The following steps were performed:

Step 1: Data Collection

The Iris dataset was downloaded from Kaggle.

Step 2: Data Preprocessing

- Removed unnecessary Id column.
- Separated features (X) and target (y).
- Applied StandardScaler to normalize feature values.

Step 3: Train-Test Split

- 80% data used for training.
- 20% data used for testing.
- Stratified splitting was used to maintain class balance.

Step 4: Model Training

- KNN classifier implemented using sklearn.
- Initial K value chosen as 5.

Step 5: Model Evaluation

- Accuracy score calculated.
- Confusion matrix generated.
- Classification report analyzed.
- Accuracy vs K graph plotted.

6. Performance Analysis

Model Accuracy

$$Accuracy = \frac{CorrectPredictions}{TotalPredictions}$$

Obtained Accuracy:

$$Accuracy = 93.33\%$$

Out of 30 test samples:

- 28 correctly classified
- 2 misclassified

Confusion Matrix Interpretation

- Iris-setosa: 100% correctly classified
- Iris-versicolor: 100% correctly classified
- Iris-virginica: 80% recall (2 misclassified as versicolor)

The model performs very well overall.
Minor confusion occurs between versicolor and virginica due to overlapping feature values.

Precision, Recall, F1-score

- Macro Average F1-score ≈ 0.93
- Weighted Average F1-score ≈ 0.93

This indicates balanced performance across classes.

7. Hyperparameter Tuning

Hyperparameter: K (Number of Neighbors)

K values from 1 to 20 were tested.

An Accuracy vs K graph was plotted.

Observations

- Small K (e.g., 1) → Higher variance (overfitting).
- Large K (e.g., 15+) → Lower variance but may underfit.
- Accuracy remained high and stable between K = 3 to K = 9.

Final Selected Value

K=5
Reason:

- Provided high and stable accuracy (~93–96%).
- Avoids tie situations (odd number).
- Balances bias and variance.

Code & Output

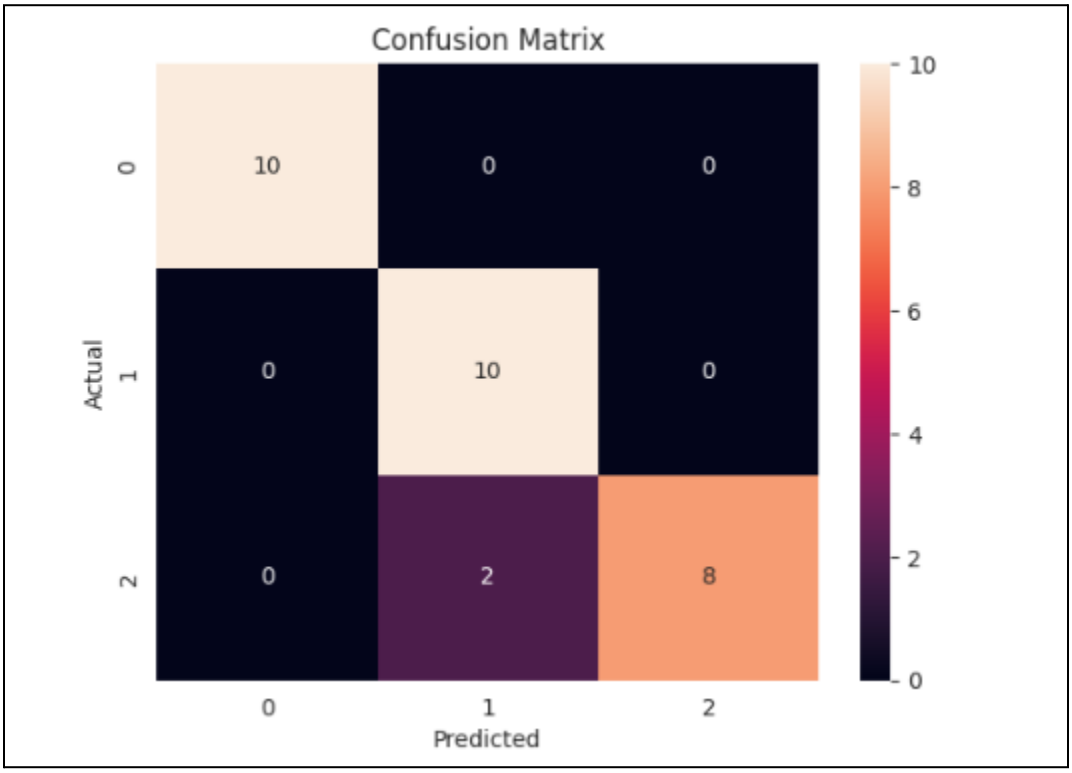
Step 1: Import Required Libraries

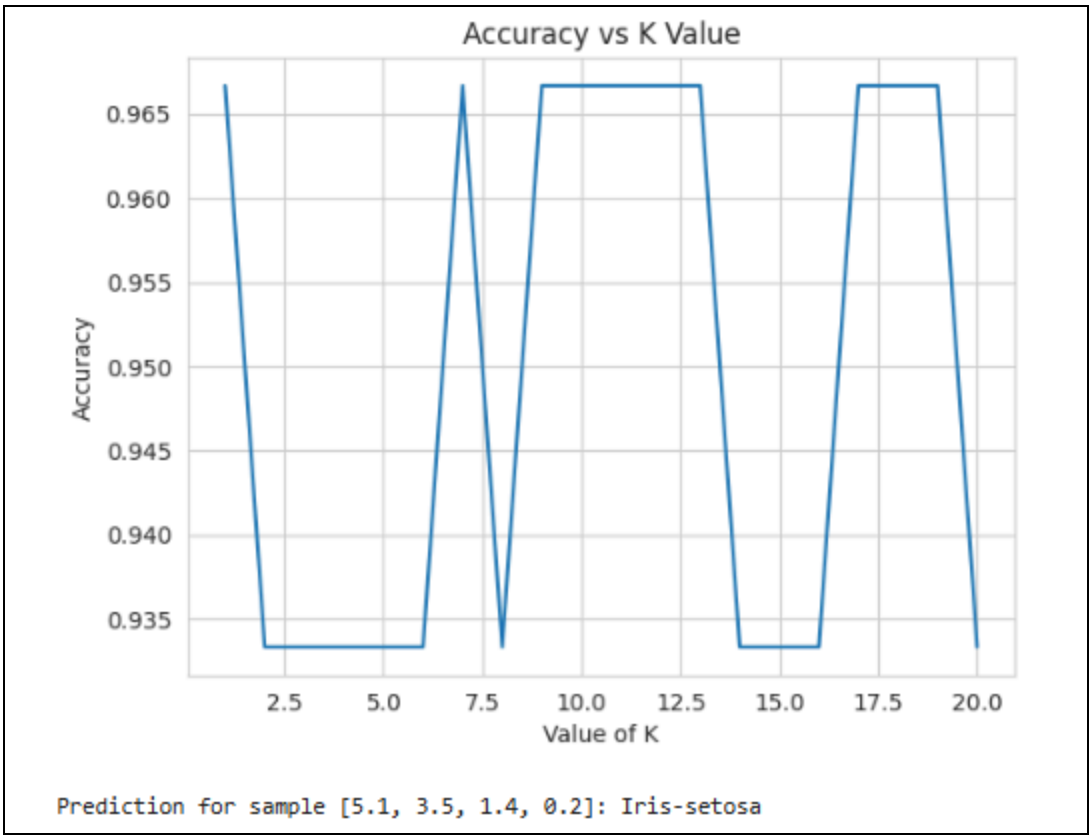
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

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

sns.set_style("whitegrid")

# ------------------------------
# Step 2: Load Dataset
# ------------------------------

# Upload Iris.csv file (downloaded from Kaggle)
from google.colab import files
uploaded = files.upload()

df = pd.read_csv("Iris.csv")

print("First 5 Rows:\n")
print(df.head())

print("\nDataset Shape:", df.shape)

# ------------------------------
# Step 3: Data Preprocessing
# ------------------------------

# Drop Id column if present
if "Id" in df.columns:
    df.drop("Id", axis=1, inplace=True)

# Separate Features and Target
X = df.drop("Species", axis=1)
y = df["Species"]

print("\nFeature Columns:", X.columns.tolist())
print("Target Classes:", y.unique())

# ------------------------------
# Step 4: Feature Scaling (Important for KNN)
# ------------------------------

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# ------------------------------
# Step 5: Train-Test Split
```

```
# ------------------------------

X_train, X_test, y_train, y_test = train_test_split(
    X_scaled,
    y,
    test_size=0.2,
    random_state=42,
    stratify=y
)

print("\nTraining Samples:", X_train.shape[0])
print("Testing Samples:", X_test.shape[0])

# ------------------------------
# Step 6: Train KNN Model
# ------------------------------

k = 5
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)

# ------------------------------
# Step 7: Predictions
# ------------------------------

y_pred = knn.predict(X_test)

# ------------------------------
# Step 8: Model Evaluation
# ------------------------------

accuracy = accuracy_score(y_test, y_pred)
print("\nModel Accuracy:", accuracy)

print("\nConfusion Matrix:")
cm = confusion_matrix(y_test, y_pred)
print(cm)

print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))

# ------------------------------
# Step 9: Confusion Matrix Visualization
# ------------------------------

plt.figure()
sns.heatmap(cm, annot=True)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
# ------------------------------
# Step 10: Accuracy vs K Graph
# ------------------------------

k_values = range(1, 21)
accuracy_scores = []

for k in k_values:
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train, y_train)
    pred = model.predict(X_test)
    accuracy_scores.append(accuracy_score(y_test, pred))

plt.figure()
plt.plot(k_values, accuracy_scores)
plt.xlabel("Value of K")
plt.ylabel("Accuracy")
plt.title("Accuracy vs K Value")
plt.show()

# ------------------------------
# Step 11: Custom Sample Prediction (Warning-Free)
# ------------------------------

# Create sample as DataFrame (IMPORTANT to avoid warning)
sample_df = pd.DataFrame(
    [[5.1, 3.5, 1.4, 0.2]],
    columns=X.columns
)

# Scale sample
sample_scaled = scaler.transform(sample_df)

prediction = knn.predict(sample_scaled)

print("\nPrediction for sample [5.1, 3.5, 1.4, 0.2]:", prediction[0])
```

```
Choose Files  Iris (1).csv
Iris (1).csv(text/csv) - 5107 bytes, last modified: 2/21/2026 - 100% done
Saving Iris (1).csv to Iris (1) (4).csv
First 5 Rows:

   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa

Dataset Shape: (150, 6)

Feature Columns: ['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']
Target Classes: ['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']

Training Samples: 120
Testing Samples: 30

Model Accuracy: 0.9333333333333333

Confusion Matrix:
[[10  0  0]
 [ 0 10  0]
 [ 0  2  8]]

Classification Report:

                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       0.83      1.00      0.91        10
 Iris-virginica       1.00      0.80      0.89        10

       accuracy                           0.93        30
      macro avg       0.94      0.93      0.93        30
   weighted avg       0.94      0.93      0.93        30
```





Conclusion

The K-Nearest Neighbors algorithm was successfully implemented on the Iris dataset. The model achieved an accuracy of 93.33%.

The algorithm performed exceptionally well for Iris-setosa and Iris-versicolor, with minor misclassification between Iris-versicolor and Iris-virginica due to feature overlap.

Hyperparameter tuning confirmed that K = 5 provides an optimal balance between overfitting and underfitting.

Thus, KNN proves to be an effective classification algorithm for small, balanced datasets.