

# Experiment No:- 4

Name:- Dipesh  
Makdiya  
Class:- D15C  
Roll no:- 32

## EXPERIMENT: K-Nearest Neighbors (KNN) for Classification

---

### 1. Dataset Source

**Dataset Used:** Iris Dataset

**Type:** Classification

**Source:**

[https://scikit-learn.org/stable/datasets/toy\\_dataset.html#iris-dataset](https://scikit-learn.org/stable/datasets/toy_dataset.html#iris-dataset)

**Access Method:**

```
from sklearn.datasets import load_iris
```

**Justification:**

- Standard real-world dataset
  - Well suited for distance-based classifiers
  - Allows easy evaluation of KNN performance
- 

### 2. Dataset Description

#### 2.1 Iris Dataset

**Objective:**

Classify iris flowers into different species using the K-Nearest Neighbors algorithm.

**Number of Instances:** 150

**Number of Features:** 4

**Features:**

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

**Target Variable:**

- Species (Setosa, Versicolor, Virginica)

**Characteristics:**

- Numerical features
  - No missing values
  - Multiclass classification problem
- 

### 3. Mathematical Formulation of the Algorithm

#### 3.1 K-Nearest Neighbors (KNN)

KNN is a **non-parametric, instance-based learning algorithm** that classifies data points based on similarity.

**Distance Metric (Euclidean Distance):**

$$d(x,y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

**Classification Rule:**

- Identify the **K nearest neighbors**
- Assign the class by **majority voting**

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

```

● print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
print(confusion_matrix(y_test, y_pred_rf))
print(classification_report(y_test, y_pred_rf))

... Random Forest Accuracy: 1.0
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
      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

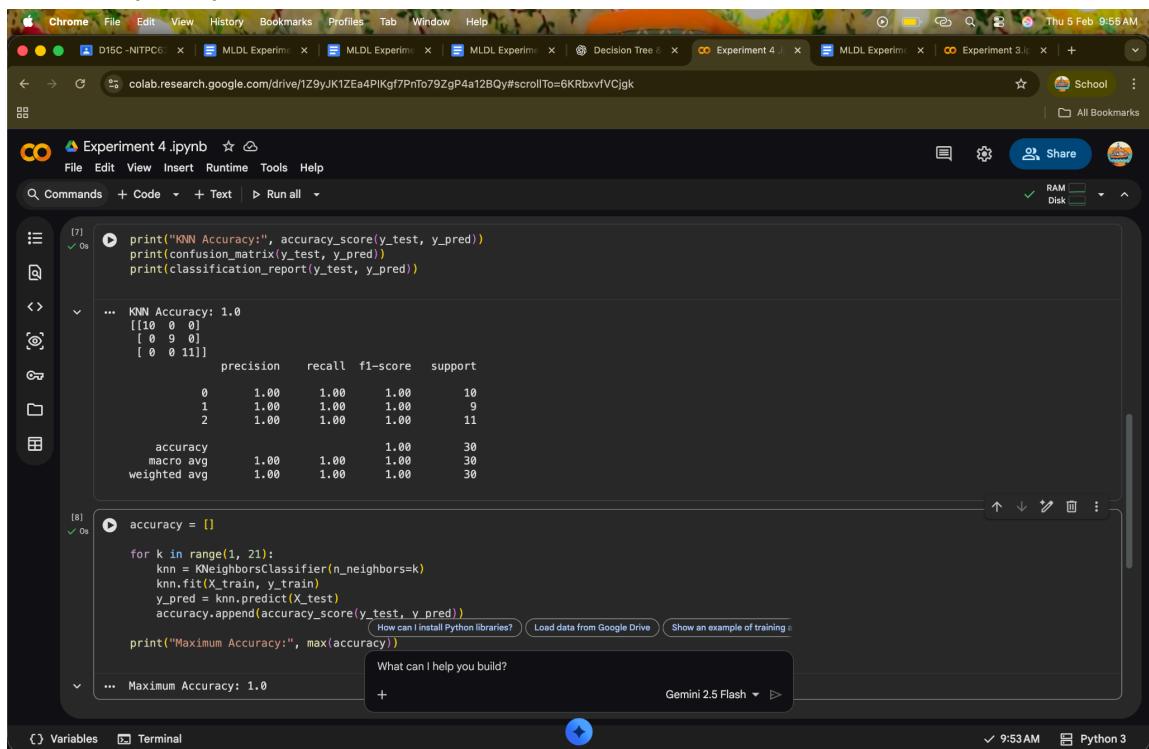
accuracy                           1.00
macro avg                           1.00     1.00     1.00      30
weighted avg                        1.00     1.00     1.00      30

```

## 4. Algorithm Limitations

### KNN Limitations

- Computationally expensive for large datasets
- Sensitive to feature scaling
- Performance depends heavily on choice of K
- Affected by noisy and irrelevant features



The screenshot shows a Google Colab notebook titled "Experiment 4.ipynb". The code cell at the top prints the accuracy, confusion matrix, and classification report for a KNN model. The output shows 100% accuracy with a confusion matrix of [[10, 0, 0], [0, 9, 0], [0, 0, 11]] and a classification report with precision, recall, f1-score, and support for each class (0, 1, 2) and overall metrics (accuracy, macro avg, weighted avg). Below this, another code cell defines an 'accuracy' list and loops through K values from 1 to 21 to calculate accuracy for each. The final output shows the maximum accuracy achieved.

```

[7] 0s
● print("KNN Accuracy:", accuracy_score(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

... KNN Accuracy: 1.0
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
      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
macro avg                           1.00     1.00     1.00      30
weighted avg                        1.00     1.00     1.00      30

[8] 0s
● accuracy = []

for k in range(1, 21):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy.append(accuracy_score(y_test, y_pred))

print("Maximum Accuracy:", max(accuracy))

... Maximum Accuracy: 1.0

```

## 5. Methodology / Workflow

### Step-by-Step Workflow

1. Import required libraries
2. Load dataset
3. Data exploration
4. Feature scaling
5. Train-test split
6. Train KNN model
7. Prediction
8. Performance evaluation
9. Hyperparameter tuning (K value)

### Workflow Diagram (Conceptual):

Dataset → Scaling → Train/Test Split  
→ Model Training → Prediction → Evaluation

---

## 6. Performance Analysis

### Metrics Used

- Accuracy
- Confusion Matrix
- Classification Report (Precision, Recall, F1-Score)

### Interpretation:

- Higher accuracy indicates better classification
- Proper K selection improves model performance

The screenshot shows a Google Colab notebook titled "Experiment 4.ipynb". The code cell contains Python code for a K-Nearest Neighbors classifier. The output shows a confusion matrix and classification report. The confusion matrix is:

	0	1	2	3
0	1.00	1.00	1.00	1.00
1	1.00	1.00	1.00	9
2	1.00	1.00	1.00	11
3				

The classification report is:

accuracy	macro avg	weighted avg
1.00	1.00	1.00
30	30	30

The code in the cell is:accuracy = []
for k in range(1, 21):
 knn = KNeighborsClassifier(n\_neighbors=k)
 knn.fit(X\_train, y\_train)
 y\_pred = knn.predict(X\_test)
 accuracy.append(accuracy\_score(y\_test, y\_pred))

print("Maximum Accuracy:", max(accuracy))

... Maximum Accuracy: 1.0

At the bottom, there is a Gemini 2.5 Flash interface with a text input field asking "What can I help you build?".

## Result and Conclusion

### Result:

K-Nearest Neighbors achieved high classification accuracy on the Iris dataset. Performance varied with different values of K, with moderate values providing optimal results.

### Conclusion:

The KNN algorithm was successfully implemented for multiclass classification. Proper feature scaling and optimal K selection significantly improved model performance, demonstrating the effectiveness of KNN for small-to-medium sized datasets.