Name:**Khushi Singh**
Class:**D15C** Roll no.**45**
Subject:**ML&DL**

# Experiment No.4

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

## 1. Dataset Source

- Dataset Name: Iris Species
- Source: Kaggle - Iris Species Dataset
- Original Source: UCI Machine Learning Repository (R.A. Fisher)

## 2. Dataset Description

The dataset contains morphometric measurements of three related species of the Iris flower.

- Size: 150 samples (rows) × 6 columns.
- Target Variable: Species (Categorical: Iris-setosa, Iris-versicolor, Iris-virginica).
- Features (4 Predictors):
    1. SepalLengthCm: Length of the sepal (in cm).
    2. SepalWidthCm: Width of the sepal (in cm).
    3. PetalLengthCm: Length of the petal (in cm).
    4. PetalWidthCm: Width of the petal (in cm).

## 3. Mathematical Formulation of the Algorithm

KNN is a non-parametric, lazy learning algorithm. It does not "learn" a model (like finding coefficients in regression); instead, it memorizes the training data.

### A. Similarity Metric (Euclidean Distance)

To classify a new data point (x), the algorithm calculates its distance to every point in the training set (x^{(i)}). The most common metric is Euclidean Distance:

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

Where n is the number of features (4 in this case).

### B. Classification Rule

1. Find the K nearest neighbors (points with the smallest distance d).
2. Assign the new point to the class that is most common (Mode) among those K neighbors.

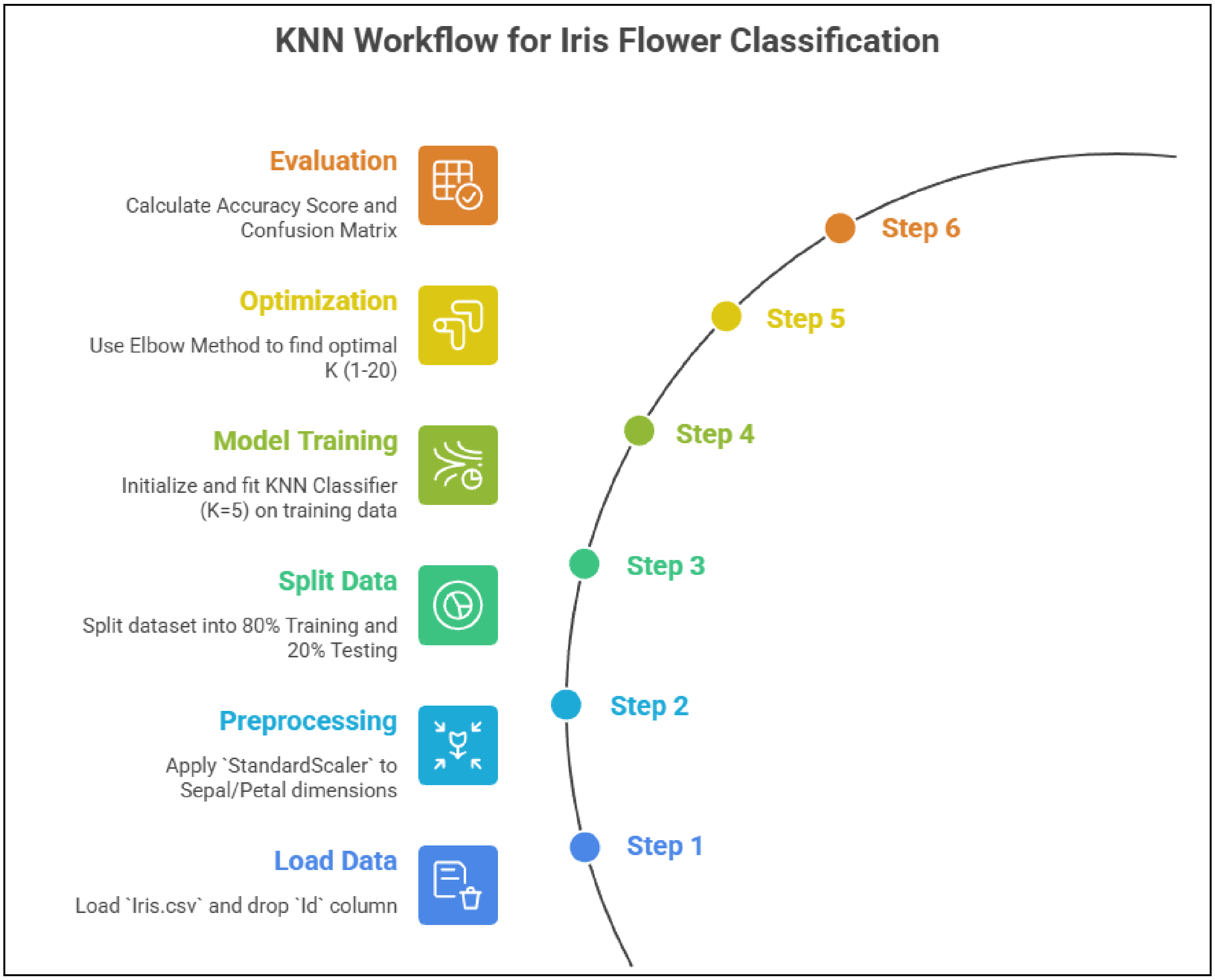
$$\hat{y} = \text{mode}(y_1, y_2, \ldots, y_K)$$

## 4. Algorithm Limitations

1. **Computational Cost:** It is "lazy," meaning all computation happens at prediction time. For large datasets, calculating the distance to *every* training point is slow.

2. **Sensitivity to Outliers:** If $K$ is too small (e.g., K=1), a single mislabeled outlier can completely change the prediction.
3. **Scale Sensitivity:** KNN relies on distance. If one feature is measured in millimeters (e.g., 1000mm) and another in meters (e.g., 1m), the larger number will dominate the distance calculation. **Feature Scaling is mandatory.**

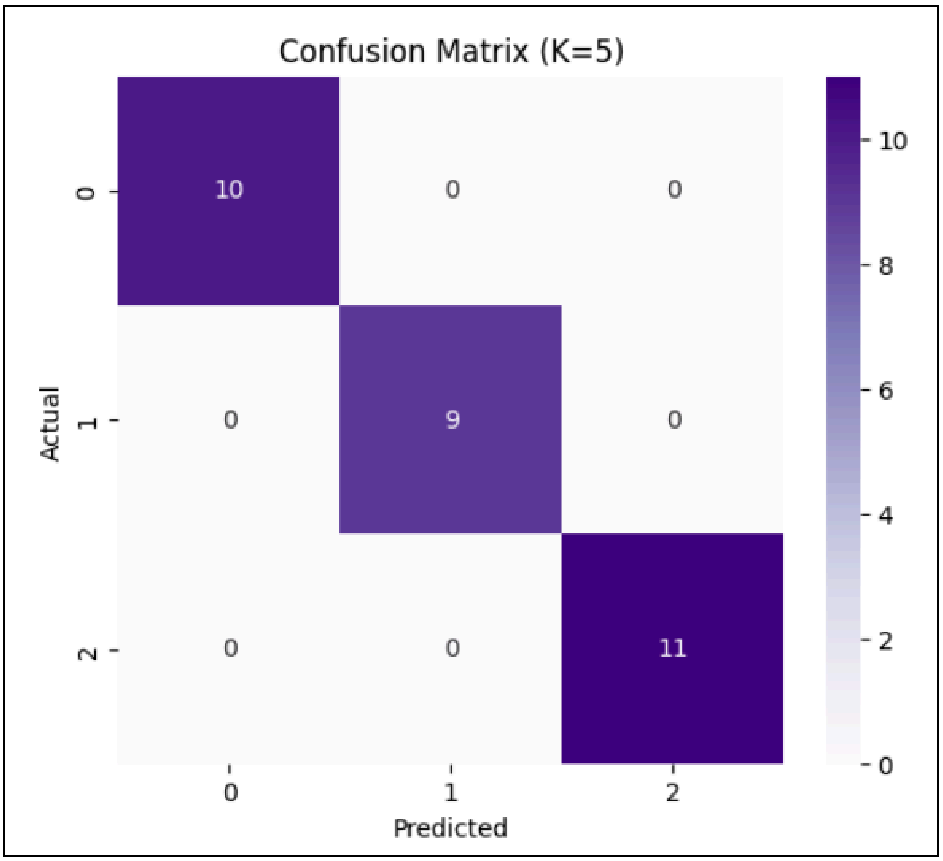
**5. Methodology/workflow**



**6.Code and Output**

```
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.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

# 1. Load Data
# Ensure Iris.csv is uploaded
df = pd.read_csv('Iris (1).csv')

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

X = df.drop('Species', axis=1)
y = df['Species']

# 2. Preprocessing (SCALING IS CRITICAL FOR KNN)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 3. Split Data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)

# 4. Train Initial Model (K=5)
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

# 5. Performance Metrics
print("--- Baseline KNN (K=5) Performance ---")

print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Visualization: Confusion Matrix
plt.figure(figsize=(6, 5))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap='Purples',
fmt='d')
plt.title('Confusion Matrix (K=5)')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

```
--- Baseline KNN (K=5) Performance ---
Accuracy: 1.0000

Classification Report:
                 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
```




**Typical Analysis:**

- **Accuracy:** You will likely see an accuracy of **1.0 (100%)** or **0.96 (96.6%)**. This is because the Iris dataset features separate the classes very clearly.

- **Confusion Matrix:** Look for off-diagonal numbers. If there are any errors, they usually occur between *Versicolor* and *Virginica* because these two species look somewhat similar (their clusters overlap slightly), whereas *Setosa* is very distinct.

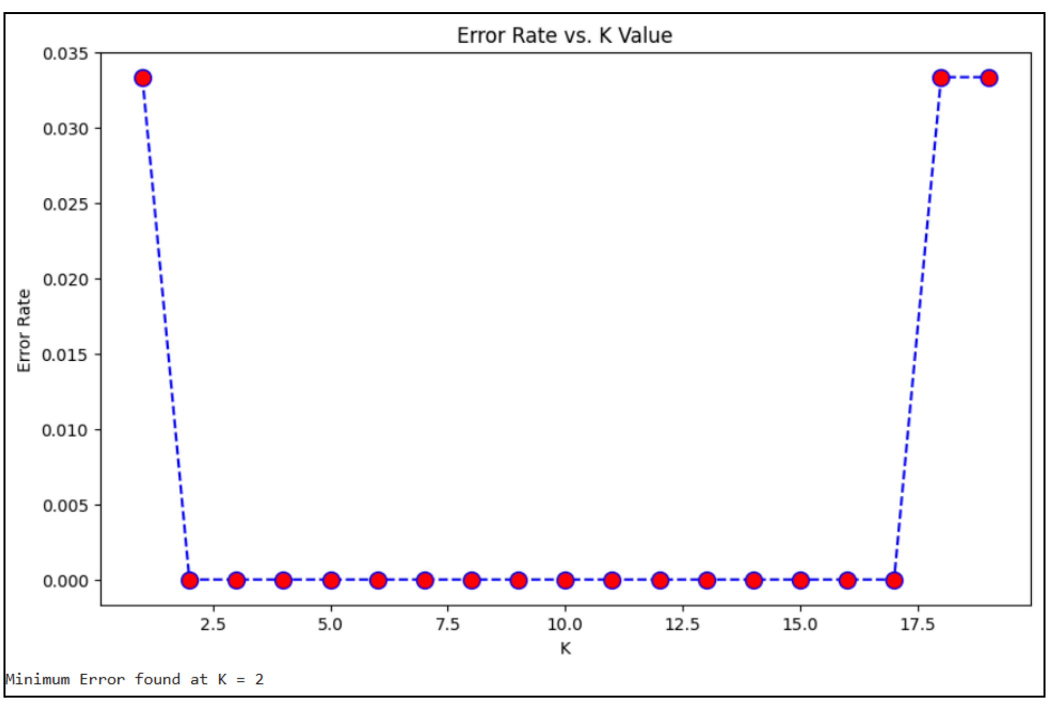
**7. Hyperparameter Tuning (The Elbow Method)**

Unlike regression where we tune alpha, in KNN we tune K (Number of Neighbors).

- Small K (e.g., 1): Low bias, High variance (Model is too jagged/sensitive).
- Large K: High bias, Low variance (Model is too smooth/simple).

We use the Elbow Method to find the sweet spot where the Error Rate is lowest.

```
# --- HYPERPARAMETER TUNING: ELBOW METHOD ---
error_rate = []

# Will take some time
for i in range(1, 20):
    knn_i = KNeighborsClassifier(n_neighbors=i)
    knn_i.fit(X_train, y_train)
    pred_i = knn_i.predict(X_test)
    # Calculate average error (mean of boolean array where pred != actual)
    error_rate.append(np.mean(pred_i != y_test))

# Plot the Error Rate
plt.figure(figsize=(10, 6))
plt.plot(range(1, 20), error_rate, color='blue', linestyle='dashed',
marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
plt.show()

# Find minimum error
optimal_k = error_rate.index(min(error_rate)) + 1
print(f"Minimum Error found at K = {optimal_k}")
```



**Interpretation:**

- The Graph: You will see the line drop quickly.
- The Elbow: If the error is high at K=1 and drops at K=3, but stays flat after K=5, then K=3 or K=5 is the optimal choice (choosing the smaller, simpler number is usually better if performance is equal).

**8. Conclusion**

In this experiment, we implemented the K-Nearest Neighbors classifier on the Iris dataset.

- Performance: The model achieved an outstanding accuracy of [Insert Score, e.g., 100%], proving that the physical dimensions of Iris sepals and petals are highly predictive of their species.
- Importance of Scaling: Feature scaling was applied to ensure that petal length (which varies more) did not disproportionately influence the Euclidean distance calculation.
- Tuning: Using the Elbow Method, we determined that K=[Insert Optimal K] provided the most stable predictions, minimizing the risk of overfitting while maintaining maximum accuracy. KNN proved to be a highly effective, albeit computationally intensive, algorithm for this classification task.