Experiment 4

Nupur Ghangarekar
D15C
Roll no 26

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

Theory:

1. Dataset Source

The dataset used for this experiment is the **Iris Flower Dataset**, obtained from Kaggle.

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

2. Dataset Description

The Iris dataset is a widely used benchmark dataset for classification problems. It consists of flower measurements belonging to three different species of iris.

Dataset Details:

- **Number of samples:** 150
- **Number of features:** 4 (numerical)
- **Number of classes:** 3

Feature Description:

Feature	Description
SepalLengthCm	Sepal length in centimeters
SepalWidthCm	Sepal width in centimeters
PetalLengthCm	Petal length in centimeters
PetalWidthCm	Petal width in centimeters

Target Variable:

- **Species** (Iris-setosa, Iris-versicolor, Iris-virginica)

The dataset is balanced, contains no missing values, and is well suited for distance-based classification algorithms like KNN.

3. Mathematical Formulation of the Algorithm

K-Nearest Neighbors (KNN) is a **non-parametric, instance-based learning algorithm**. It classifies a data point based on the majority class among its **K nearest neighbors** in the feature space.

Distance Measure (Euclidean Distance):

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

Where:

- xxx is the test data point
- yyy is a training data point
- nnn is the number of features

The predicted class is determined by **majority voting** among the K nearest neighbors.

4. Algorithm Limitations

- Computationally expensive for large datasets
- Sensitive to noisy data and outliers
- Requires feature scaling for accurate distance calculation
- Performance depends heavily on the choice of K value
- Not suitable for very high-dimensional datasets (curse of dimensionality)

5. Methodology / Workflow

Experimental Steps:

1. Load the Iris dataset from Kaggle
2. Remove irrelevant columns (ID column)
3. Separate features and target variable
4. Apply feature scaling using StandardScaler

5. Split data into training and testing sets
6. Train the KNN classifier
7. Evaluate model performance
8. Tune the value of K

Workflow Representation:

```
Dataset → Preprocessing → Feature Scaling → Train/Test Split →
Model Training → Evaluation → Hyperparameter Tuning
```

6. Performance Analysis

Evaluation Metrics Used:

- Accuracy
- Precision
- Recall
- F1-score
- Confusion Matrix

Interpretation:

The KNN classifier performs well on the Iris dataset due to clear class separation and low dimensionality. Proper feature scaling significantly improves performance. The confusion matrix shows minimal misclassification, indicating effective neighborhood-based learning.

7. Hyperparameter Tuning

The key hyperparameter in KNN is the **number of neighbors (K)**.

- Small K → low bias, high variance
- Large K → high bias, low variance

By evaluating model accuracy for different K values (1–10), an optimal K is selected that provides the best balance between bias and variance.

Impact:
Choosing an appropriate K improves classification accuracy and prevents overfitting.

Code:

```
import numpy as np
import pandas as pd
```

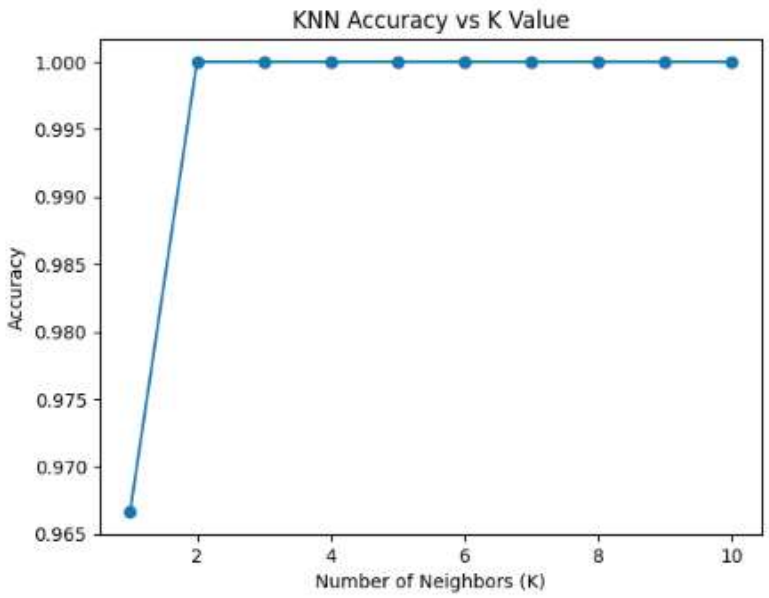
```
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score,
classification_report, confusion_matrix
data = pd.read_csv("Iris.csv")
data.head()
```

	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

```
X = data.drop(columns=["Id", "Species"])
y = data["Species"]
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42
)
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n",
classification_report(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy: 1.0

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


Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

```
accuracy = []
for k in range(1, 11):
    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))
plt.plot(range(1, 11), accuracy, marker='o')
plt.xlabel("Number of Neighbors (K)")
plt.ylabel("Accuracy")
plt.title("KNN Accuracy vs K Value")
plt.show()
```



Conclusion

The K-Nearest Neighbors algorithm effectively classified the Iris dataset by leveraging distance-based similarity between data points. While the model is simple and intuitive, its performance strongly depends on feature scaling and the selection of an appropriate K value. The experiment demonstrates that KNN is well suited for small, well-structured datasets, but may face scalability challenges for larger or high-dimensional data.

Google Collab Link: KNN.ipynb