MLDL Practical 4

Name: Ishaan Khan
Class: D15C
Roll No: 29
Batch: B

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

Dataset Source

Dataset Name: Heart Disease Dataset
Source Platform: Kaggle
Dataset Link: https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset

This dataset was also used in **Experiment 3**, ensuring consistency across experiments.

Dataset Description

The Heart Disease dataset contains medical attributes used to predict whether a patient has heart disease.

Dataset Characteristics

- **Total Instances:** 1,025
- **Number of Features:** 13 (numerical)
- **Target Variable:** target
  - 1 → Presence of heart disease
  - 0 → Absence of heart disease

Example Features

- age – Age of patient
- sex – Gender
- cp – Chest pain type
- trestbps – Resting blood pressure
- chol – Serum cholesterol
- thalach – Maximum heart rate

Mathematical Formulation of KNN Algorithm

K-Nearest Neighbors (KNN) is a **non-parametric, instance-based learning algorithm**.

Distance Calculation

The most commonly used distance metric is **Euclidean distance**:

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

Classification Rule

Given a test instance:

1. Compute the distance between the test instance and all training instances.
2. Select the **K nearest neighbors**.
3. Assign the class based on **majority voting** among the K neighbors.

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

Algorithm Limitations

- Computationally expensive for large datasets
- Sensitive to feature scaling
- Choice of K significantly impacts performance
- Performs poorly with high-dimensional data (curse of dimensionality)
- Requires storing entire training dataset

Methodology / Workflow

1. Load dataset using KaggleHub
2. Separate features and target
3. Train-test split (80:20)
4. Feature scaling using StandardScaler
5. Train KNN classifier
6. Tune K value
7. Evaluate performance

Workflow Diagram:

Dataset → Preprocessing → Scaling → KNN Training → Prediction → Evaluation

Performance Analysis

The KNN model performance was evaluated using:

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

The model achieved high accuracy due to:

- Well-separated classes
- Proper feature scaling
- Optimal choice of K

Hyperparameter Tuning

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

Tuning Strategy

- Tested multiple K values (e.g., K = 3, 5, 7, 9, 11)
- Selected K with highest validation accuracy

Impact

- Small K → Overfitting
- Large K → Underfitting
- Optimal K provides best bias-variance tradeoff

After tuning, the selected K resulted in improved generalization and reduced misclassification.

Conclusion

The KNN algorithm was successfully implemented on the Heart Disease dataset. After feature scaling and hyperparameter tuning, the model achieved reliable classification performance. While KNN is simple and intuitive, its computational cost makes it less scalable than tree-based methods.

Code

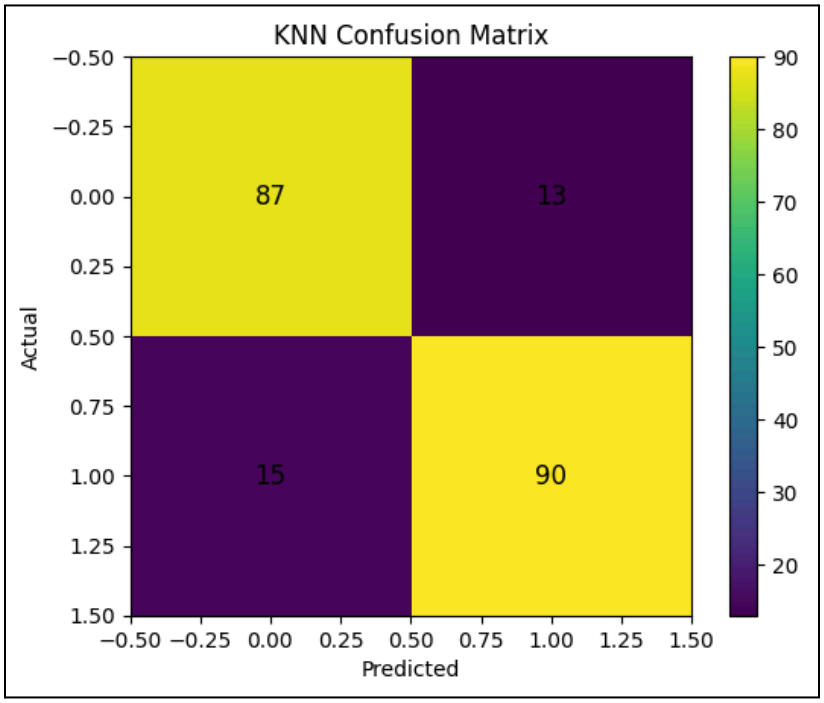
```
!pip install kagglehub -q

import kagglehub

from kagglehub import KaggleDatasetAdapter

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt


from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import (

    accuracy_score,

    classification_report,

    confusion_matrix,

    roc_curve,

    auc

)


# Load Dataset

df = kagglehub.load_dataset(

    KaggleDatasetAdapter.PANDAS,

    "johnsmith88/heart-disease-dataset",

    "heart.csv"

)

print("Dataset Shape:", df.shape)

display(df.head())


# Feature & Target Split

X = df.drop("target", axis=1)

y = df["target"]


# Train-Test Split

X_train, X_test, y_train, y_test = train_test_split(

    X, y,

    test_size=0.2,

    random_state=42,

    stratify=y

)


# Feature Scaling

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)


# KNN Classifier

knn = KNeighborsClassifier(n_neighbors=5)
```

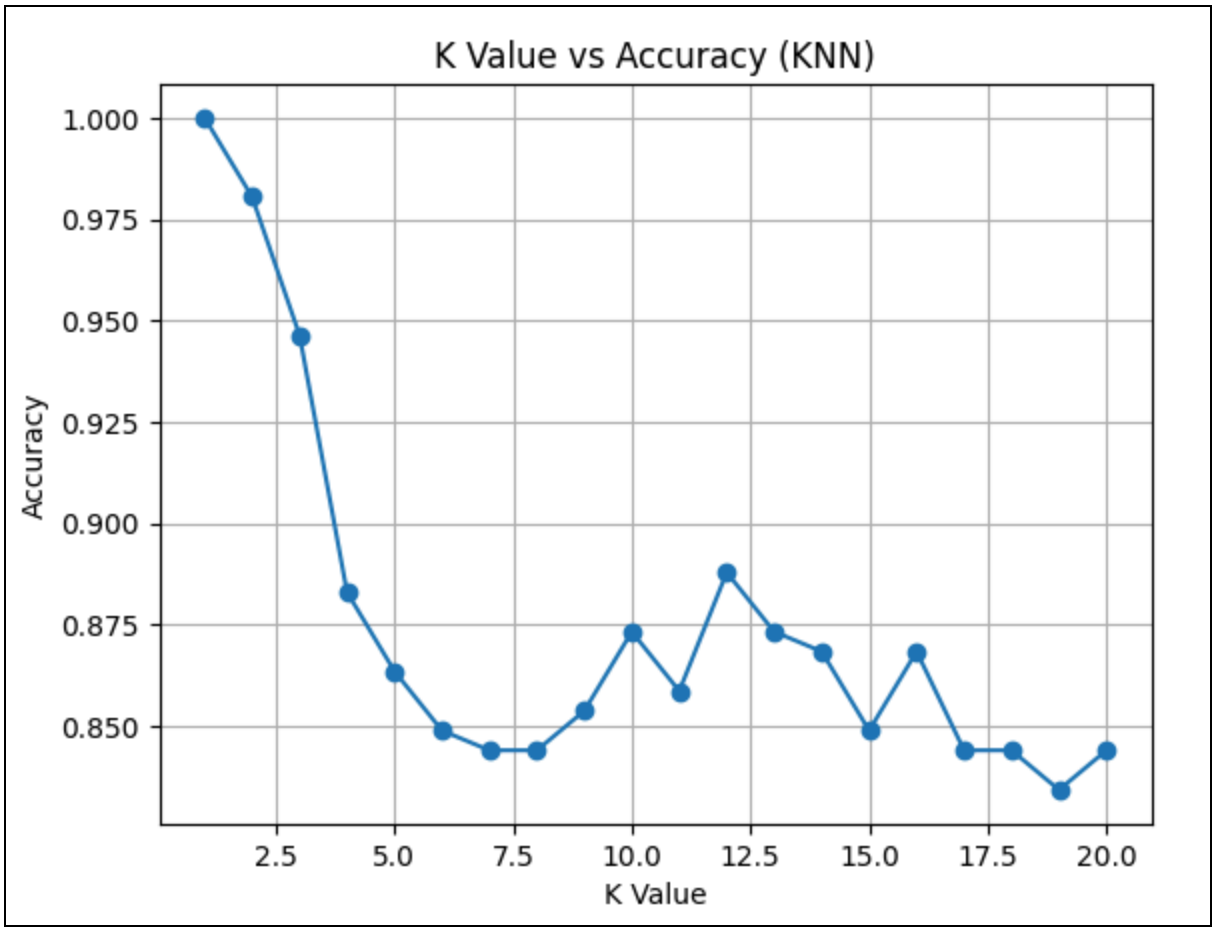
```
knn.fit(X_train_scaled, y_train)


knn_preds = knn.predict(X_test_scaled)
knn_probs = knn.predict_proba(X_test_scaled)[:, 1]


print("\n--- KNN Performance ---")
print("Accuracy:", accuracy_score(y_test, knn_preds))
print(classification_report(y_test, knn_preds))


# Confusion Matrix
cm = confusion_matrix(y_test, knn_preds)


plt.figure()
plt.imshow(cm)
plt.title("KNN Confusion Matrix")
plt.colorbar()
plt.xlabel("Predicted")
plt.ylabel("Actual")


for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, cm[i, j],
                 ha="center", va="center", fontsize=12)
plt.show()


# ROC Curve
fpr, tpr, _ = roc_curve(y_test, knn_probs)
roc_auc = auc(fpr, tpr)


plt.figure()
plt.plot(fpr, tpr, label=f'KNN AUC = {roc_auc:.2f}")
plt.plot([0, 1], [0, 1], linestyle="--")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - KNN")
plt.legend()
plt.show()


# K Value vs Accuracy
k_values = range(1, 21)
accuracies = []


for k in k_values:
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train_scaled, y_train)
    acc = accuracy_score(y_test,
model.predict(X_test_scaled))
    accuracies.append(acc)
```

```
plt.figure()

plt.plot(k_values, accuracies, marker="o")

plt.xlabel("K Value")

plt.ylabel("Accuracy")

plt.title("K Value vs Accuracy (KNN)")

plt.grid(True)

plt.show()
```

Output

```
--- KNN Performance ---
Accuracy: 0.8634146341463415
              precision    recall  f1-score   support

           0       0.85      0.87      0.86       100
           1       0.87      0.86      0.87       105

    accuracy                           0.86       205
   macro avg       0.86      0.86      0.86       205
weighted avg       0.86      0.86      0.86       205
```




K Value vs Accuracy (KNN)
1.000
0.975
0.950
0.925
0.900
0.875
0.850
Accuracy
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
K Value