Name: Atharva Chaudhari

Roll No: 42

EXPERIMENT NO. 1

AIM:

Implementation of K-Nearest Neighbors (KNN)

Dataset Source:

Dataset Name: **Calories Burnt Prediction Dataset**

Source: Kaggle
Link: https://www.kaggle.com/datasets/fmendes/fmendes-datasets

Dataset Description:

The dataset contains information about individuals and the number of calories burned during exercise.

Features (Input Variables)

- **Gender** – Male/Female
- **Age** – Age of the person
- **Height** – Height in cm
- **Weight** – Weight in kg
- **Duration** – Exercise duration (minutes)
- **Heart Rate** – Heart beats per minute
- **Body Temp** – Body temperature during exercise

🔹 **Target Variable**

- **Calories** – Number of calories burned

In this experiment, we converted Calories into a classification problem:

- 0 → Low Calories
- 1 → High Calories

🔹 **Dataset Size**

- Approximately 15000 records
- 8 columns

The dataset contains both numerical and categorical features.

Mathematical Formulation of KNN:

K-Nearest Neighbors is a distance-based supervised learning algorithm.

It calculates the distance between a new data point and all training points.

The most common distance metric is Euclidean Distance:

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

Where:

- xix_ixi = feature of test point
- yiy_iyi = feature of training point
- n = number of features

Classification Rule:

1. Choose value of K
2. Find K nearest neighbors
3. Assign the class which appears most frequently among K neighbors.

Algorithm Limitations:

KNN has several limitations:

1. Computationally Expensive:
   It calculates distance for all training points.

2. Sensitive to Scaling:
   If features are not scaled, results may be incorrect.

3. **Curse of Dimensionality:**
   Performance decreases when dataset has many features.

4. **Memory Intensive:**
   It stores entire training dataset.

5. **Sensitive to Noise:**
   Outliers can affect prediction.

Methodology / Workflow:

Step 1: Data Collection

Dataset was loaded using pandas.

Step 2: Data Preprocessing

- Dropped unnecessary column (User_ID)
- Encoded categorical variable (Gender)
- Converted target into classification
- Applied StandardScaler

Step 3: Train-Test Split

- 80% Training Data
- 20% Testing Data

Step 4: Model Training

- Applied KNN with K = 5

Step 5: Prediction

- Predicted values on test data

Step 6: Evaluation

- Accuracy Score
- Classification Report
- Confusion Matrix

Performance Analysis:

The model was evaluated using:

a) Accuracy Score:

Measures overall correctness of model.

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

b) Confusion Matrix:

Shows:

- True Positives
- True Negatives
- False Positives
- False Negatives

c) Classification Report:

Includes:

- Precision
- Recall
- F1-Score

If accuracy is high (e.g., above 85%), model performs well.

The **Accuracy vs K graph** helps identify optimal K value.

Hyperparameter Tuning:

The main hyperparameter in KNN is:

K (Number of Neighbors):

We tested values from 1 to 20.

From the graph:

- Small K → High variance (overfitting)
- Large K → High bias (underfitting)

The best K is selected where accuracy is maximum.

Example:
If K = 7 gives highest accuracy, then K = 7 is optimal.

CODE:

```
# K-Nearest Neighbors (KNN) Classification

# Dataset: calories.csv

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.metrics import accuracy_score, confusion_matrix, classification_report

from sklearn.preprocessing import StandardScaler, LabelEncoder
```

```
# 1️⃣ Load Dataset
df = pd.read_csv("calories.csv")
print("First 5 Rows:")
print(df.head())
# 2️⃣ Data Preprocessing
# Drop User_ID if present
if 'User_ID' in df.columns:
    df = df.drop('User_ID', axis=1)
# 🔹 Encode Gender column (male/female → 0/1)
if 'Gender' in df.columns:
    le = LabelEncoder()
    df['Gender'] = le.fit_transform(df['Gender'])
# Convert Calories into classification problem
threshold = df['Calories'].median()
df['Calories_Class'] = np.where(df['Calories'] > threshold, 1, 0)
# Define Features and Target
X = df.drop(['Calories', 'Calories_Class'], axis=1)
y = df['Calories_Class']
# 3️⃣ Feature Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
# 4️⃣ Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42
)
# 5️⃣ Apply KNN
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
# 6️⃣ Predictions
y_pred = knn.predict(X_test)
# 7️⃣ Evaluation
accuracy = accuracy_score(y_test, y_pred)
print("\nModel Accuracy:", accuracy)
print("\nClassification Report:\n", classification_report(y_test, y_pred))
# 📊 Visualization 1: Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure()
sns.heatmap(cm, annot=True, fmt='d')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
# 📊 Visualization 2: Accuracy vs K Value

k_values = range(1, 21)

accuracy_list = []

for k in k_values:

    model = KNeighborsClassifier(n_neighbors=k)

    model.fit(X_train, y_train)

    pred = model.predict(X_test)

    accuracy_list.append(accuracy_score(y_test, pred))

plt.figure()

plt.plot(k_values, accuracy_list)

plt.title("Accuracy vs K Value")

plt.xlabel("K Value")

plt.ylabel("Accuracy")

plt.show()
```

OUTPUT:

```
First 5 Rows:
    User_ID  Gender  Age  Height  Weight  Duration  Heart_Rate  Body_Temp  \
0  14733363    male   68   190.0    94.0      29.0       105.0       40.8
1  14861698  female   20   166.0    60.0      14.0        94.0       40.3
2  11179863    male   69   179.0    79.0       5.0        88.0       38.7
3  16180408  female   34   179.0    71.0      13.0       100.0       40.5
4  17771927  female   27   154.0    58.0      10.0        81.0       39.8

   Calories
0     231.0
1      66.0
2      26.0
3      71.0
4      35.0

Model Accuracy: 0.9823333333333333
```

Classification Report:

	precision	recall	f1-score	support
0	0.98	0.98	0.98	1463
1	0.98	0.99	0.98	1537
accuracy			0.98	3000
macro avg	0.98	0.98	0.98	3000
weighted avg	0.98	0.98	0.98	3000