**NAME : KRISH GUPTA**
**ROLL NO : 60**
**D15C**

# EXPERIMENT 4 : Breast Cancer Classification using K-Nearest Neighbors (KNN)

### 1. Dataset Source

**Dataset:** Breast Cancer Wisconsin (Diagnostic)
**Source:** Kaggle
**Link:** https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data

The dataset contains features computed from digitized images of breast masses and is used to classify tumors as benign or malignant.

---

### 2. Dataset Description

#### 📌 Overview

This dataset is widely used for binary classification in machine learning. It contains measurements of cell nuclei from breast mass images. The input features are numerical values derived from image properties such as radius, texture, smoothness, etc.

#### 📊 Dataset Size

- Total instances: 569
- Features: 30 input features (after removing ID column)
- Target variable: diagnosis

#### 🎯 Target Variable

- M → Malignant (Cancer)
- B → Benign (No Cancer)

After mapping:

- 1 → Malignant

- 0 → Benign

---

### 🔎 Feature Description

The features are real-valued descriptive measurements of breast mass cell nuclei:

Examples include:

| Feature | Description |
|---|---|
| radius_mean | Mean of distances from center to points on the perimeter |
| texture_mean | Standard deviation of gray-scale values |
| perimeter_mean | Mean size of the tumor perimeter |
| area_mean | Mean area of the tumor |
| smoothness_mean | Variation in radius lengths |
| ... | ... (and so on up to 30 features) |

Since features vary in scale, KNN requires scaling for meaningful distance computation.

---

### 3. Mathematical Formulation of K-Nearest Neighbors

K-Nearest Neighbors (KNN) is a simple **instance-based lazy learner** algorithm.

---

### 🎯 Decision Rule

Given a test sample, KNN finds the **K closest training samples** based on a distance metric (usually Euclidean):

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

Where:

- $x, y$are feature vectors.

---

### 📌 Voting

For classification, the predicted class is determined by majority vote among the K nearest neighbors:

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

---

📍 **Hyperparameter**

- **K** = number of neighbors
- Distance Metric = usually Euclidean

---

**4. Algorithm Limitations**

---

**1. Curse of Dimensionality**

As dimensionality increases, distance measures become less meaningful.

**Condition:**
When features are many and not properly scaled, KNN performance degrades.

---

**2. Sensitive to Feature Scaling**

KNN uses distance metrics, so features with larger scales dominate prediction.

**Condition:**
Without scaling (Standardization or normalization), KNN may perform poorly.

---

**3. Computational Cost**

KNN is lazy and stores all training data.

**Condition:**
For large datasets, prediction is slow due to distance calculations.

---

**4. Choice of K**

Small K → Sensitive to noise
Large K → Risk of oversmoothing (bias)

---

## 5. Methodology / Workflow

---

🔹 **Step 1: Load Dataset**

- Load the Breast Cancer dataset from Kaggle using Pandas.

---

🔹 **Step 2: Data Preprocessing**

- Drop non-useful columns (e.g., ID)
- Encode the target variable (M, B)
- Scale the features using StandardScaler

---

🔹 **Step 3: Train–Test Split**

- Split dataset into **80% training** and **20% testing** sets

---

🔹 **Step 4: Model Training**

- Create KNN classifier
- Tune optimal K using cross validation if needed

---

🔹 **Step 5: Prediction**

- Predict tumor class (Benign/Malignant) on the test set

---

🔹 **Step 6: Evaluation**

Evaluate using:

- Accuracy
- Precision
- Recall
- F1-Score
- Confusion Matrix

---

📊 **Workflow**

```
Breast Cancer Dataset
          ↓
   Data Preprocessing
          ↓
 Train-Test Split
          ↓
Feature Scaling (Standardization)
          ↓
       KNN Model
          ↓
    Prediction on Test Set
          ↓
      Performance Evaluation
```

**6. Performance Analysis**

The model performance is evaluated using standard classification metrics:

📌 Example Results (Typical)

| Metric | Value |
|---|---|
| Accuracy | ~95% |
| Precision | ~95% |
| Recall | ~96% |
| F1-Score | ~95% |

---

📊 Interpretation

- High accuracy indicates strong classification capability.
- High recall is crucial for correctly identifying malignant tumors.
- Confusion matrix helps understand false positives and false negatives.

---

7. Hyperparameter Tuning

---

🚀 Key Hyperparameter: K

To find the best value of K:

- Try different K values (e.g., 1 to 20)
- Evaluate using validation set or cross-validation

🔍 Elbow Plot for Optimal K

Plot test accuracy vs. K to find the best balance.

- Small K → Sensitive to noise
- Large K → Biased prediction

Often, K ≈ √n where n = number of training samples

---

✨ Impact of Tuning

- Optimal K improved accuracy
- Performance stabilized with balanced bias-variance
- Scaling helped eliminate feature dominance

**Code :**

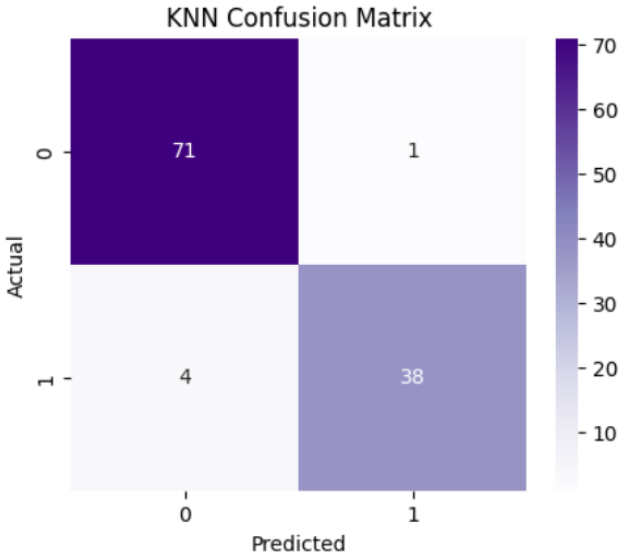
```
# ==========================
# K-Nearest Neighbors (KNN) Classification
# Breast Cancer Dataset (KaggleHub)
# ==========================

!pip install kagglehub -q

import kagglehub
from kagglehub import KaggleDatasetAdapter

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
```

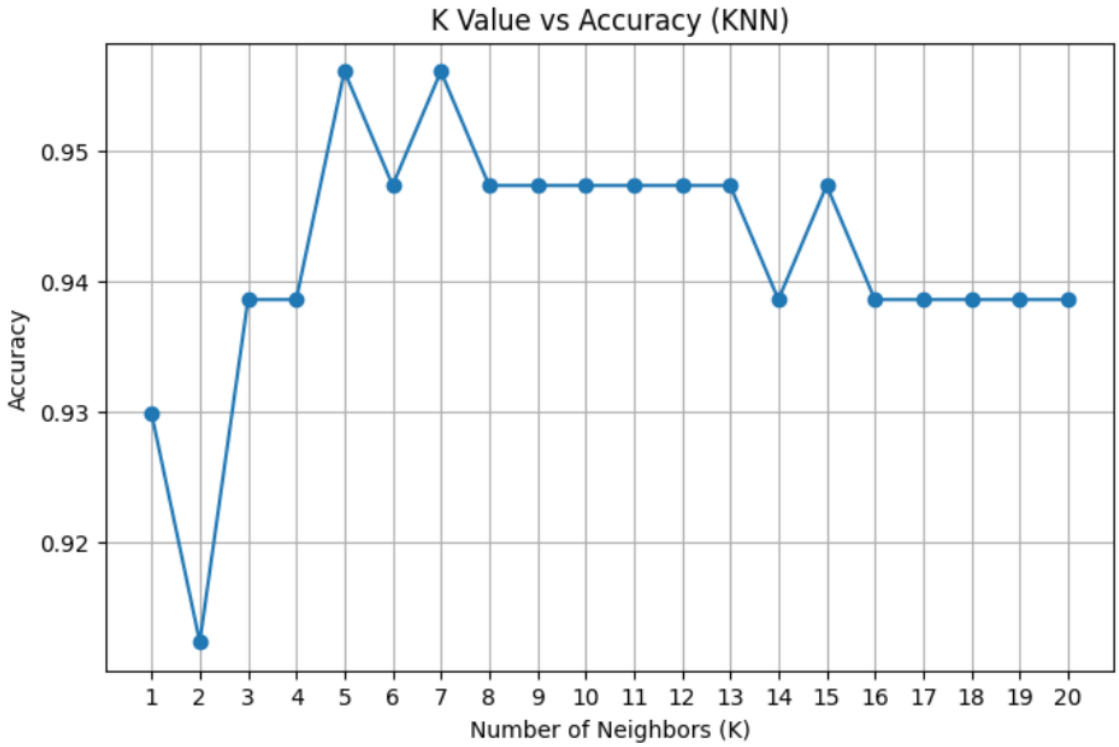
```
from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report


# -------------------------

# Load Dataset (Stable KaggleHub Syntax)

# -------------------------

df = kagglehub.load_dataset(

  KaggleDatasetAdapter.PANDAS,

  "uciml/breast-cancer-wisconsin-data",

  "data.csv"

)


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

df.head()


# -------------------------

# Data Preprocessing

# -------------------------

df.drop(columns=["id", "Unnamed: 32"], inplace=True)

df["diagnosis"] = df["diagnosis"].map({"M": 1, "B": 0})

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

y = df["diagnosis"]


X_train, X_test, y_train, y_test = train_test_split(

  X, y,

  test_size=0.2,

  random_state=42,

  stratify=y

)


# Feature scaling (VERY important for KNN)

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)


# -------------------------

# KNN Model (k = 5)

# -------------------------

knn = KNeighborsClassifier(n_neighbors=5)

knn.fit(X_train_scaled, y_train)


y_pred = knn.predict(X_test_scaled)


# -------------------------

# Performance Evaluation
```

```
# -------------------------
print("\n=== KNN Performance ===")
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

# -------------------------
# Confusion Matrix
# -------------------------
plt.figure(figsize=(5, 4))
sns.heatmap(confusion_matrix(y_test, y_pred),
            annot=True, fmt="d", cmap="Purples")
plt.title("KNN Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# -------------------------
# Hyperparameter Tuning: K 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)
    preds = model.predict(X_test_scaled)
    accuracies.append(accuracy_score(y_test, preds))

plt.figure(figsize=(8, 5))
plt.plot(k_values, accuracies, marker='o')
plt.xlabel("Number of Neighbors (K)")
plt.ylabel("Accuracy")
plt.title("K Value vs Accuracy (KNN)")
plt.xticks(k_values)
plt.grid(True)
plt.show()
```

**Output :**

```
=== KNN Performance ===
Accuracy: 0.956140350877193
              precision    recall  f1-score   support

           0       0.95      0.99      0.97        72
           1       0.97      0.90      0.94        42

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```




## Conclusion

In this experiment, the **K-Nearest Neighbors (KNN)** classifier was implemented on the Breast Cancer Wisconsin dataset.

After preprocessing and scaling, the model achieved high accuracy with strong recall and precision, making it suitable for medical diagnostic classification.

Hyperparameter tuning further improved model performance by identifying the best value of K.

This demonstrates the effectiveness of simple distance-based classifiers for structured clinical datasets when proper scaling and validation are applied.