Assignment No. 5

Problem Statement: Develop and evaluate the K-Nearest Neighbors (KNN) algorithm for both classification and regression tasks..

Objective: Gain a deep understanding of KNN, implement it effectively, and analyze how different parameters influence its performance in classification and regression.

Prerequisite :

1. A Python setup with essential libraries like NumPy, Pandas, Matplotlib, Seaborn, and Scikit-learn.

2. Internet access (for dataset retrieval if necessary).

3. Fundamental knowledge of machine learning and the KNN algorithm.

Theory :

K-Nearest Neighbors (KNN) Algorithm

KNN is a supervised learning technique used for classification and regression. It predicts outcomes based on the similarity between a test data point and its closest training examples.

How KNN Works

1. Choose the number of neighbors (K).
2. Compute the distance (e.g., Euclidean, Manhattan) between the test instance and all training data.
3. Identify the K nearest neighbors.
4. For classification: Assign the most frequent class among the neighbors.
5. For regression: Compute the average (or weighted average) of the neighbors' values.

Choosing the Right K Value

- Small K (e.g., 1) → Sensitive to noise, may lead to overfitting.
- Large K (e.g., 20) → Smoothens decision boundary but may lose local data patterns.

Common Distance Metrics

- Euclidean Distance: Measures direct straight-line distance.
- Manhattan Distance: Calculates distance based on grid-based movements.

Advantages of KNN

Easy to understand and implement.
Works well for small datasets with fewer features.

Disadvantages of KNN

Computationally expensive for large datasets.
Performance is affected by irrelevant features and feature scaling.

Implementation Steps

1. Loading the Dataset
    - Use Pandas to import the dataset.
    - Check dataset properties (.shape, .info(), .isnull().sum()).
2. Data Preprocessing
    - Handle missing values (impute or remove).
    - Encode categorical variables (LabelEncoder, OneHotEncoder).
    - Normalize numerical data (Min-Max Scaling, Standardization).
3. Train-Test Split
    - Use train_test_split to split data (e.g., 80% training, 20% testing).
4. KNN for Classification
    - Use KNeighborsClassifier from `sklearn.neighbors`.
    - Train model and evaluate using accuracy, precision, recall, and confusion matrix.
5. KNN for Regression
    - Use KNeighborsRegressor from `sklearn.neighbors`.
    - Evaluate using Mean Squared Error (MSE) and R-squared Score.
6. Hyperparameter Tuning
    - Experiment with different K values.
    - Compare performance with different distance metrics (Euclidean, Manhattan, Minkowski).
7. Data Visualization
    - Plot decision boundaries for classification.
    - Show the impact of K value on accuracy.
    - Compare actual vs. predicted values in regression.

CODE & OUTPUT :

```
[1]: import numpy as np
     import pandas as pd
     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
     from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, precision_score, recall_score
```

```
•[4]: df = pd.read_csv('/Users/pranavashokdivekar/this_mac/Machine Learning/heart.csv')
```

```
[5]: print(df.head(5))
```

```
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   52    1   0       125   212    0        1      168      0      1.0      2
1   53    1   0       140   203    1        0      155      1      3.1      0
2   70    1   0       145   174    0        1      125      1      2.6      0
3   61    1   0       148   203    0        1      161      0      0.0      2
4   62    0   0       138   294    1        1      106      0      1.9      1

   ca  thal  target
0   2     3       0
1   0     3       0
2   0     3       0
3   1     3       0
4   3     2       0
```

```
[6]: print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1025 non-null   int64
 1   sex       1025 non-null   int64
 2   cp        1025 non-null   int64
 3   trestbps  1025 non-null   int64
 4   chol      1025 non-null   int64
 5   fbs       1025 non-null   int64
 6   restecg   1025 non-null   int64
 7   thalach   1025 non-null   int64
 8   exang     1025 non-null   int64
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64
 11  ca        1025 non-null   int64
 12  thal      1025 non-null   int64
 13  target    1025 non-null   int64
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
None
```

```
[7]: print(df.describe())
```

```
               age          sex           cp     trestbps        chol  \
count  1025.000000  1025.000000  1025.000000  1025.000000  1025.00000
mean     54.434146     0.695610     0.942439   131.611707   246.00000
std       9.072290     0.460373     1.029641    17.516718    51.59251
min      29.000000     0.000000     0.000000    94.000000   126.00000
25%      48.000000     0.000000     0.000000   120.000000   211.00000
50%      56.000000     1.000000     1.000000   130.000000   240.00000
75%      61.000000     1.000000     2.000000   140.000000   275.00000
max      77.000000     1.000000     3.000000   200.000000   564.00000

               fbs      restecg      thalach        exang      oldpeak  \
count  1025.000000  1025.000000  1025.000000  1025.000000  1025.000000
mean      0.149268     0.529756   149.114146     0.336585     1.071512
std       0.356527     0.527878    23.005724     0.472772     1.175053
min       0.000000     0.000000    71.000000     0.000000     0.000000
25%       0.000000     0.000000   132.000000     0.000000     0.000000
50%       0.000000     1.000000   152.000000     0.000000     0.800000
75%       0.000000     1.000000   166.000000     1.000000     1.800000
max       1.000000     2.000000   202.000000     1.000000     6.200000

             slope           ca         thal       target
count  1025.000000  1025.000000  1025.000000  1025.000000
mean      1.385366     0.754146     2.323902     0.513171
std       0.617755     1.030798     0.620660     0.500070
min       0.000000     0.000000     0.000000     0.000000
25%       1.000000     0.000000     2.000000     0.000000
50%       1.000000     0.000000     2.000000     1.000000
75%       2.000000     1.000000     3.000000     1.000000
max       2.000000     4.000000     3.000000     1.000000
```

```
[8]: print(df.isnull().sum())
```

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

[10]:
```python
# Check if the 'id' column exists in the dataset and drop it (to remove unnecessary identifiers)
if 'id' in df.columns:
    df.drop(columns=['id'], inplace=True)

# Check if the 'Unnamed: 32' column exists in the dataset and drop it (likely an extra unnamed column)
if 'Unnamed: 32' in df.columns:
    df.drop(columns=['Unnamed: 32'], inplace=True)
```

[11]:
```python
# Separate the features (X) and target variable (y)
X = df.iloc[:, 1:]  # Select all columns except the first one as features
y = df.iloc[:, 0]   # Select the first column as the target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2)
```

[12]:
```python
# Create a K-Nearest Neighbors (KNN) classifier with k=3 (3 neighbors)
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
```

[12]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

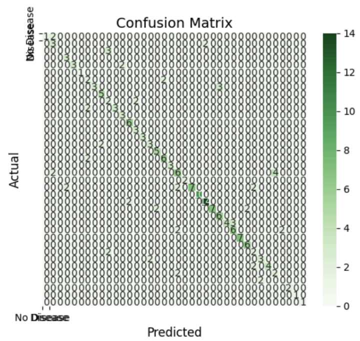
[13]:
```python
# Predict the target values
y_pred = knn.predict(X_test)
```

[14]:
```python
# Calculate the accuracy
accuracy = accuracy_score(y_test, y_pred)
# Calculate the precision (ratio of correctly predicted positive observations to total predicted positives)
precision = precision_score(y_test, y_pred, average='macro', zero_division=1)
# Calculate recall
recall = recall_score(y_test, y_pred, average='macro', zero_division=1)
# Compute the confusion matrix
cm = confusion_matrix(y_test, y_pred)
# Compute specificity
specificity = cm[0,0] / (cm[0,0] + cm[0,1])
```

[15…
```python
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("Specificity:", specificity)
print("Confusion Matrix:\n", cm)
print("Classification Report:\n", classification_report(y_test, y_pred))
```

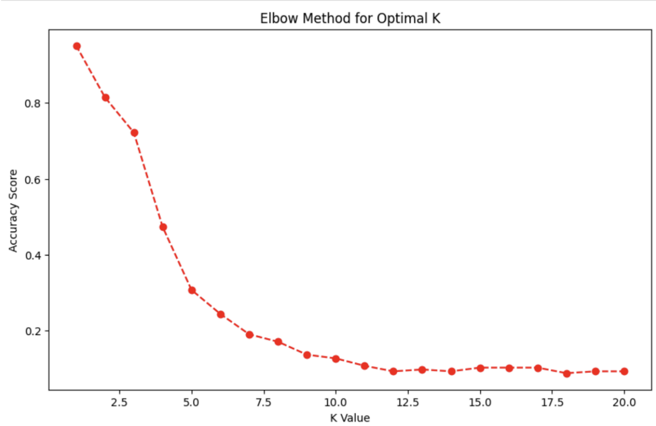
```
Accuracy: 0.7219512195121951
Precision: 0.7658208020050126
Recall: 0.7360856249014144
Specificity: 0.3333333333333333
Confusion Matrix:
 [[1 2 0 ... 0 0 0]
 [0 3 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 2 0 0]
 [0 0 0 ... 0 1 0]
 [0 0 0 ... 0 0 1]]
Classification Report:
               precision    recall  f1-score   support

          34       1.00      0.33      0.50         3
          35       0.43      0.60      0.50         5
          37       0.00      0.00      0.00         3
          38       0.60      1.00      0.75         3
          39       1.00      0.60      0.75         5
          40       1.00      1.00      1.00         1
          41       0.33      0.50      0.40         4
          42       1.00      0.50      0.67         6
          43       0.71      1.00      0.83         5
          44       0.29      0.50      0.36         4
          45       1.00      0.60      0.75         5
          46       0.60      1.00      0.75         3
          47       0.75      1.00      0.86         6
          48       0.60      1.00      0.75         3
          49       1.00      1.00      1.00         3
          50       0.60      0.60      0.60         5
          51       0.71      1.00      0.83         5
          52       0.75      0.75      0.75         8
          53       1.00      1.00      1.00         3
          54       0.50      0.50      0.50        12
          55       1.00      1.00      1.00         2
          56       1.00      0.54      0.70        13
          57       1.00      1.00      1.00        10
          58       0.88      0.88      0.88        16
          59       1.00      0.64      0.78        11
          60       0.67      1.00      0.80         6
          61       1.00      0.57      0.73         7
          62       0.67      0.60      0.63        10
          63       1.00      1.00      1.00         7
          64       0.75      1.00      0.86         6
          65       0.33      0.50      0.40         4
          66       0.60      0.60      0.60         5
          67       1.00      0.67      0.80         6
          68       0.33      0.50      0.40         4
          69       0.00      0.00      0.00         2
          70       1.00      1.00      1.00         2
          76       1.00      1.00      1.00         1
          77       1.00      1.00      1.00         1

    accuracy                           0.72       205
   macro avg       0.74      0.74      0.71       205
weighted avg       0.77      0.72      0.72       205
```

```python
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

# Plot
plt.figure(figsize=(6, 5))  # Set figure size
sns.heatmap(cm, annot=True, fmt="d", cmap="Greens", annot_kws={"size": 10})  # Reduce text size
plt.xlabel("Predicted", fontsize=12)
plt.ylabel("Actual", fontsize=12)
plt.title("Confusion Matrix", fontsize=14)
plt.xticks(ticks=[0, 1], labels=["No Disease", "Disease"], rotation=0, fontsize=10)
plt.yticks(ticks=[0, 1], labels=["No Disease", "Disease"], rotation=90, fontsize=10)
plt.show()
```




```python
scores = []
k_range = range(1, 21)  # Testing k from 1 to 20

for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred_k = knn.predict(X_test)
    scores.append(accuracy_score(y_test, y_pred_k))
```

```python
plt.figure(figsize=(10, 6))
plt.plot(k_range, scores, marker='o', linestyle='dashed', color='red')
plt.xlabel('K Value')
plt.ylabel('Accuracy Score')
plt.title('Elbow Method for Optimal K')
plt.show()
```



Github :- https://github.com/Pranav-Divekar/Machine-learning-

Conclusion:

The KNN model's performance depended on K value selection and feature scaling. Lower K led to overfitting, while higher K caused underfitting. Proper preprocessing and tuning improved accuracy, making KNN effective for classification and regression while highlighting its strengths and limitations.