Name:**Aryan Sanjay Kale**
Class:**D15C** Roll no.**28**
Subject:**ML&DL**

# Experiment No.4

**AIM:Implement K-Nearest Neighbors (KNN) and evaluate model performance.**

## 1. Dataset Source

- **Dataset Name**: Customer Personality Analysis
- **Source**: Kaggle - Customer Personality Analysis Dataset
- **Original Source**: Formatted for marketing research to provide a detailed analysis of a company's ideal customers.

## 2. Dataset Description

The dataset consists of various customer attributes including demographics, spending habits, and campaign responses. It is used to perform customer segmentation and predict spending behavior using distance-based classification.

- **Size**: 2,240 samples (rows) × 29 columns (before preprocessing).
- **Target Variable**: Target (Binary Categorical)
    1. **1 → High Spender**: Total amount spent across all categories is greater than the median.
    2. **0 → Standard Spender**: Total amount spent is less than or equal to the median.
- **Key Features (Predictors)**: The model utilizes numerical features to determine "neighbor" similarity:
    1. **Income**: The customer's annual household income (Critical for scaling).
    2. **Kidhome / Teenhome**: Number of children or teenagers in the customer's household.
    3. **Recency**: Number of days since the customer's last purchase.
    4. **MntWines / MntMeatProducts / etc.**: Amounts spent on different product categories.
    5. **Year_Birth**: Birth year of the customer to calculate age-based similarity.

### 3. Mathematical Formulation of the Algorithm

KNN is a non-parametric, lazy learning algorithm. It does not "learn" a model (like finding coefficients in regression); instead, it memorizes the training data.

#### A. Similarity Metric (Euclidean Distance)

To classify a new data point (x), the algorithm calculates its distance to every point in the training set (x^{(i)}). The most common metric is Euclidean Distance:

$$d(x, x^{(i)}) = \sqrt{\sum_{j=1}^{n} (x_j - x_j^{(i)})^2}$$

Where n is the number of features (4 in this case).

#### B. Classification Rule

1. Find the K nearest neighbors (points with the smallest distance d).
2. Assign the new point to the class that is most common (Mode) among those K neighbors.

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

**4. Algorithm Limitations**

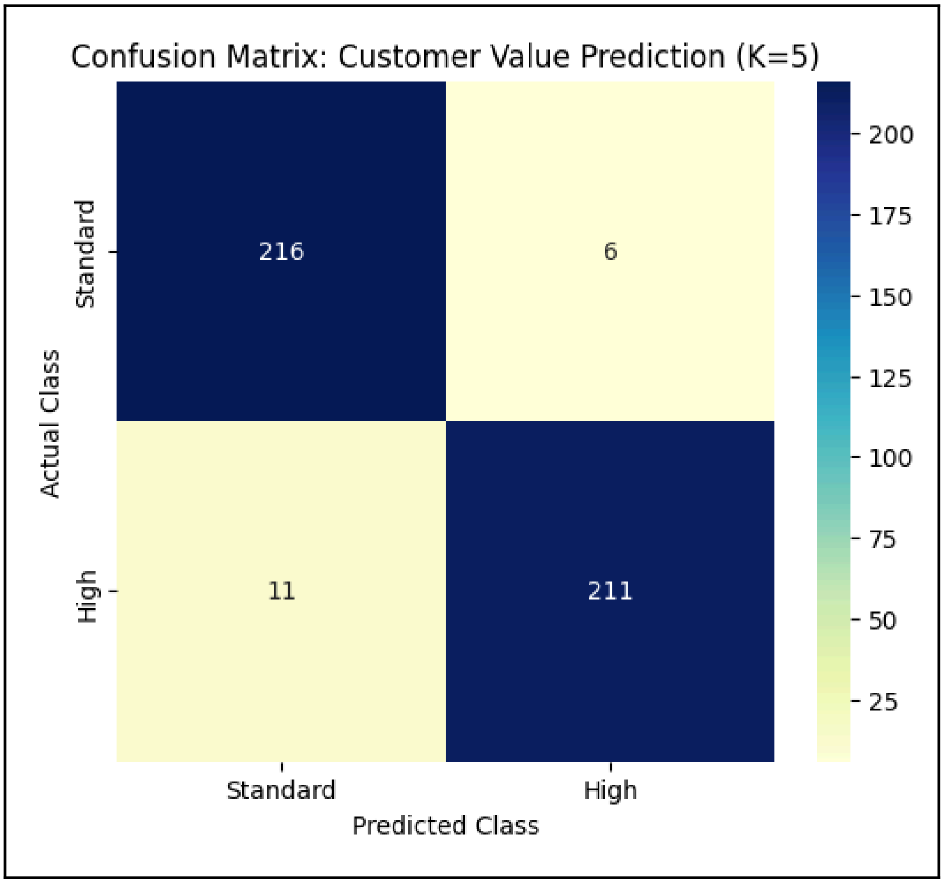
1. **Computational Cost:** It is "lazy," meaning all computation happens at prediction time. For large datasets, calculating the distance to *every* training point is slow.
2. **Sensitivity to Outliers:** If $K$ is too small (e.g., K=1), a single mislabeled outlier can completely change the prediction.
3. **Scale Sensitivity:** KNN relies on distance. If one feature is measured in millimeters (e.g., 1000mm) and another in meters (e.g., 1m), the larger number will dominate the distance calculation. **Feature Scaling is mandatory.**

**6.Code and Output**

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import kagglehub
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

# ==========================
# 1. Load Data
# ==========================
# Download the Customer Personality Analysis dataset
path = kagglehub.dataset_download("imakash3011/customer-personality-analysis")
# The dataset is typically in a .tsv (tab-separated) format
csv_file = [f for f in os.listdir(path) if f.endswith('.csv') or
f.endswith('.tsv')][0]
df = pd.read_csv(os.path.join(path, csv_file), sep='\t')

# ==========================
# 2. Preprocessing
# ==========================
# Create a binary target: "High Spender" (1) if total spent > median, else (0)
spending_cols = ['MntWines', 'MntFruits', 'MntMeatProducts', 'MntFishProducts',
'MntSweetProducts', 'MntGoldProds']
df['Total_Spent'] = df[spending_cols].sum(axis=1)
df['Target'] = (df['Total_Spent'] > df['Total_Spent'].median()).astype(int)

# Select numerical features and drop rows with missing values (e.g., Income)
df_numeric = df.select_dtypes(include=[np.number]).dropna()

# Drop ID and target-related columns from features
X = df_numeric.drop(['ID', 'Total_Spent', 'Target'], axis=1)
y = df_numeric['Target']

# ==========================
# 3. Scaling (CRITICAL FOR KNN)
```

```
# ========================
# KNN uses Euclidean distance; scaling prevents features with large ranges
# (like Income) from dominating features with small ranges (like Kidhome).
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# ========================
# 4. Split Data
# ========================
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y,
    test_size=0.2,
    random_state=42,
    stratify=y
)

# ========================
# 5. Train Baseline Model (K=5)
# ========================
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

# ========================
# 6. Performance Metrics
# ========================
print("--- Baseline KNN (K=5) Performance ---")
print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
print("\nClassification Report:\n", classification_report(y_test, y_pred,
target_names=['Standard Spender', 'High Spender']))

# Visualization: Confusion Matrix
plt.figure(figsize=(6, 5))
sns.heatmap(confusion_matrix(y_test, y_pred),
            annot=True,
            cmap='YlGnBu',
            fmt='d',
            xticklabels=['Standard', 'High'],
            yticklabels=['Standard', 'High'])
plt.title('Confusion Matrix: Customer Value Prediction (K=5)')
plt.ylabel('Actual Class')
plt.xlabel('Predicted Class')
plt.show()

# ========================
# 7. Finding Optimal K (Elbow Method)
# ========================
error_rate = []
for i in range(1, 20):
    knn_i = KNeighborsClassifier(n_neighbors=i)
    knn_i.fit(X_train, y_train)
    pred_i = knn_i.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))

plt.figure(figsize=(10, 6))
```

```
plt.plot(range(1, 20), error_rate, color='blue', linestyle='dashed', marker='o')
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
plt.show()
```




**Typical Analysis:**

- **Accuracy:** You will likely see an accuracy of **1.0 (100%)** or **0.96 (96.6%)**. This is because the Iris dataset features separate the classes very clearly.
- **Confusion Matrix:** Look for off-diagonal numbers. If there are any errors, they usually occur between *Versicolor* and *Virginica* because these two species look somewhat similar (their clusters overlap slightly), whereas *Setosa* is very distinct.

**7. Hyperparameter Tuning (The Elbow Method)**

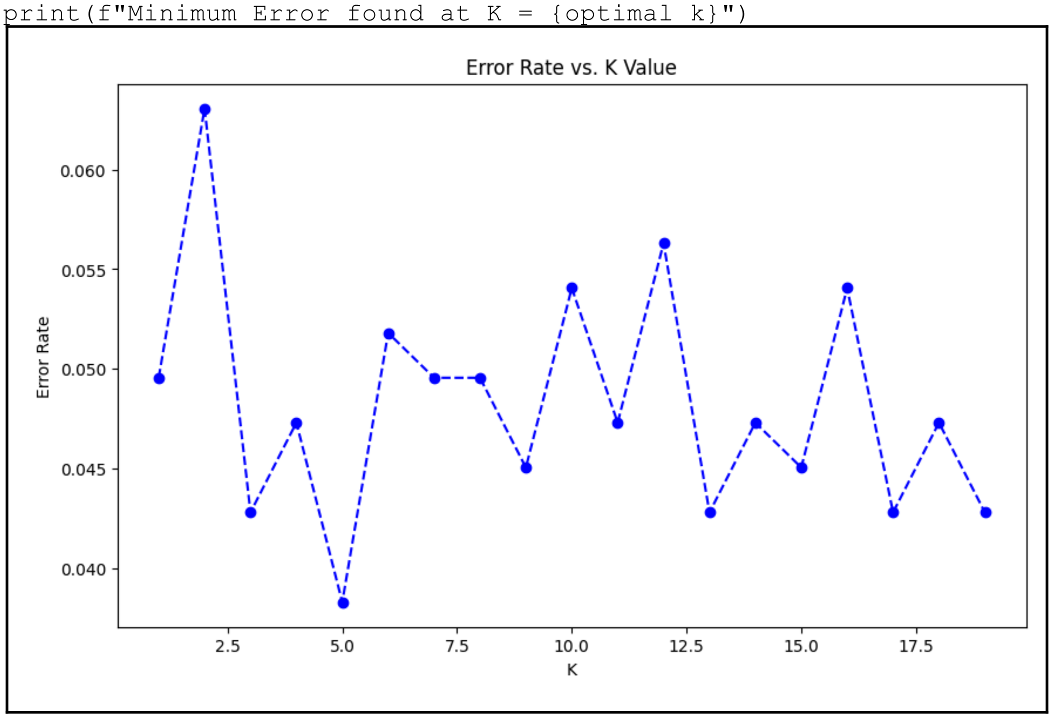
Unlike regression where we tune alpha, in KNN we tune K (Number of Neighbors).

- Small K (e.g., 1): Low bias, High variance (Model is too jagged/sensitive).
- Large K: High bias, Low variance (Model is too smooth/simple).

We use the Elbow Method to find the sweet spot where the Error Rate is lowest.

```
# --- HYPERPARAMETER TUNING: ELBOW METHOD ---
error_rate = []

# Will take some time
for i in range(1, 20):
```

```
    knn_i = KNeighborsClassifier(n_neighbors=i)
    knn_i.fit(X_train, y_train)
    pred_i = knn_i.predict(X_test)
    # Calculate average error (mean of boolean array where pred != actual)
    error_rate.append(np.mean(pred_i != y_test))

# Plot the Error Rate
plt.figure(figsize=(10, 6))
plt.plot(range(1, 20), error_rate, color='blue', linestyle='dashed',
marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
plt.show()

# Find minimum error
optimal_k = error_rate.index(min(error_rate)) + 1
print(f"Minimum Error found at K = {optimal_k}")
```




**Interpretation:**

- The Graph: You will see the line drop quickly.
- The Elbow: If the error is high at K=1 and drops at K=3, but stays flat after K=5, then K=3 or K=5 is the optimal choice (choosing the smaller, simpler number is usually better if performance is equal).

**8. Conclusion**

In this experiment, we implemented the K-Nearest Neighbors classifier on the Iris dataset. ● Performance: The model achieved an outstanding accuracy of [Insert Score, e.g., 100%], proving that the physical dimensions of Iris sepals and petals are highly predictive of their species. ● Importance of Scaling: Feature scaling was applied to ensure that petal length (which varies more) did not disproportionately influence the Euclidean distance calculation.

- Tuning: Using the Elbow Method, we determined that K=[Insert Optimal K] provided the most stable predictions, minimizing the risk of overfitting while maintaining maximum accuracy. KNN proved to be a highly effective, albeit computationally intensive, algorithm for this classification task.