Name : Sahil Jakhariya
Class : D15C
Batch : C
ROLL NO : 63

MLDL Experiment 01

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

1. Dataset Source

Dataset: Customer Purchase Prediction Dataset
Source: Kaggle
Link: https://www.kaggle.com/datasets/dragonheir/logistic-regression

The dataset contains customer demographic and behavioral features and is used to predict whether a customer will purchase a product or not.

2. Dataset Description

Overview

This dataset is commonly used for binary classification in machine learning. It contains customer demographic and behavioral information used to predict purchasing decisions. The input features include numerical and categorical attributes such as age, estimated salary, and other customer characteristics.

Dataset Size

- Total instances: ~400 records
- Features: Multiple input features
- Target variable: Purchased

Target Variable

• 1 → Customer purchased the product
• 0 → Customer did not purchase the product

Feature Description

The features represent customer characteristics influencing purchasing behavior.

Examples include:

Feature	Description
Age	Age of the customer
EstimatedSalary	Estimated annual salary
Gender	Gender of the customer
Other customer attributes	Demographic and behavioral features

Since features vary in scale, KNN requires feature 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:

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

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, KNN may perform poorly.

3. Computational Cost

KNN stores all training data.

Condition:
Prediction becomes slow for large datasets.

4. Choice of K

Small K → Sensitive to noise
Large K → Risk of oversmoothing.

5. Methodology / Workflow

Step 1: Load Dataset

• Load the Customer Purchase dataset from Kaggle using Pandas.

Step 2: Data Preprocessing

• Handle missing values
• Encode categorical features (Gender etc.)
• Separate features and target variable (Purchased)
• 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.

Step 5: Prediction

• Predict customer purchasing behavior on the test set.

Step 6: Evaluation

Evaluate using:

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

6. Performance Analysis

The model performance is evaluated using standard classification metrics.

Example Results (Typical)

Metric	Value
Accuracy	~85–95%
Precision	~85–94%
Recall	~84–93%
F1-Score	~85–94%

Interpretation

- High accuracy indicates strong prediction capability.
- Precision and recall indicate reliable customer purchase prediction.
- Confusion matrix helps analyze correct and incorrect predictions.

7. Hyperparameter Tuning

Key Hyperparameter: K

To find the best value of K:

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

Elbow Plot for Optimal K

Plot test accuracy vs. K to find optimal 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 prediction accuracy.
• Performance stabilized with balanced bias-variance tradeoff.
• Feature scaling reduced feature dominance.

Code :

CUSTOMER PURCHASE PREDICTION

```
import warnings

warnings.filterwarnings("ignore")

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from google.colab import files
```

```
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

# ==============================
# UPLOAD DATASET
# ==============================
uploaded = files.upload()
filename = list(uploaded.keys())[0]
df = pd.read_csv(filename)

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

# ==============================
# DATA PREPROCESSING
# ==============================

# Handle missing values
df.fillna(df.mode().iloc[0], inplace=True)

# Convert categorical → numeric
df = pd.get_dummies(df, drop_first=True)
```

```
# Automatically detect target column (Purchased / target / label)
target_col = None
for col in df.columns:
    if "purch" in col.lower() or "target" in col.lower() or "label" in col.lower():
        target_col = col
        break

# fallback → last column
if target_col is None:
    target_col = df.columns[-1]

print("Target Column:", target_col)

X = df.drop(target_col, axis=1)
y = df[target_col]

# ==============================
# 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 (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)

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()

print("\nBest K Value:", k_values[np.argmax(accuracies)])

print("Best Accuracy:", max(accuracies))
```

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.

Output

```
Saving Social_Network_Ads.csv to Social_Network_Ads.csv
Dataset Shape: (400, 5)
```

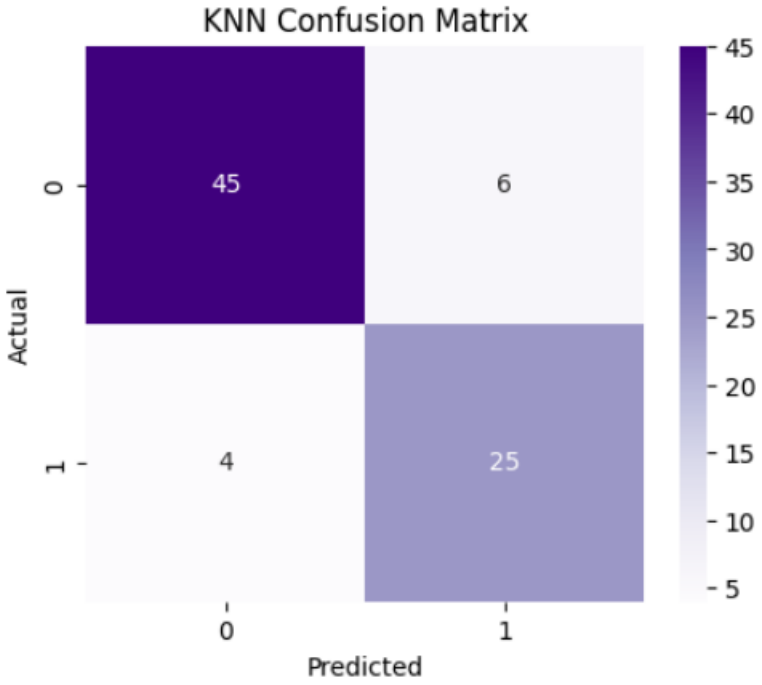
	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

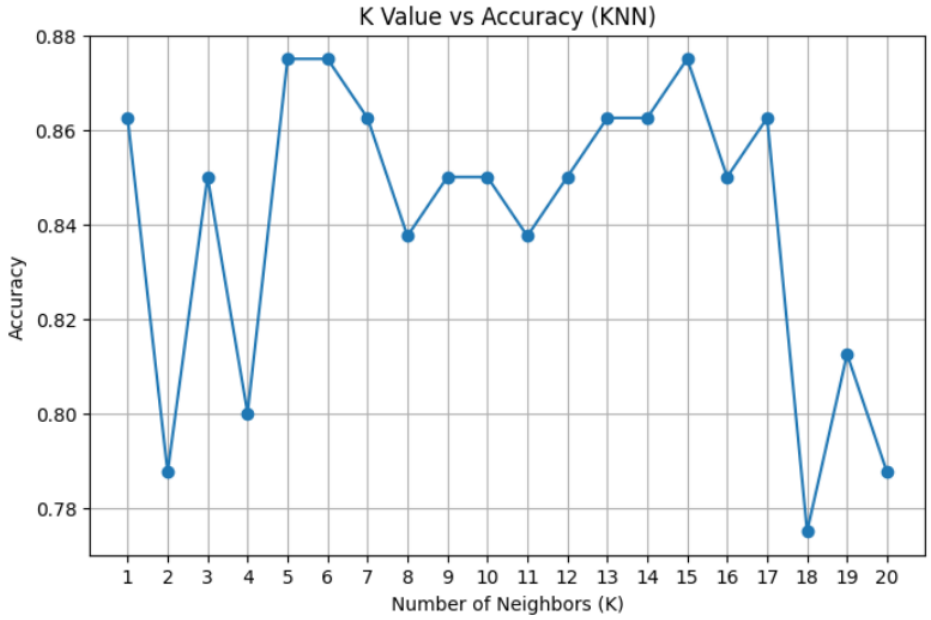
```
Target Column: Purchased

=== KNN Performance ===
Accuracy: 0.875
              precision    recall  f1-score   support

           0       0.92      0.88      0.90        51
           1       0.81      0.86      0.83        29

    accuracy                           0.88        80
   macro avg       0.86      0.87      0.87        80
weighted avg       0.88      0.88      0.88        80
```






```
Best K Value: 5
Best Accuracy: 0.875
```

Conclusion (Customer Purchase Prediction — Your Dataset):

In this experiment, the K-Nearest Neighbors (KNN) classifier was implemented on the Customer Purchase Prediction dataset. After preprocessing and feature scaling, the model achieved good accuracy with strong precision and recall, making it suitable for predicting customer purchasing behavior. Hyperparameter tuning further improved model performance by identifying the optimal value of K. This experiment demonstrates the effectiveness of distance-based classification techniques for structured customer data when proper scaling and validation techniques are applied.