

Green University of Bangladesh

Department of Computer Science and Engineering (CSE)

### Faculty of Sciences and Engineering Semester: Summer, Year: 2025, B.Sc. in CSE (Day)

**Lab Report:02**

### Code: CSE-412

**Section: 222 D3 Course Title: Algorithom**

**Lab Experiment Name:** K-Nearest Neighbors (KNN) algorithm

**Student Details**

|  |  |
| --- | --- |
| **Name** | **ID** |
| **Md.Jabed Mollah** | **222002167** |

### Lab Date : 20/07/2025

**Submission Date : 07/08/2025** **Course Teacher’s Name: Md. Sabbir Hosen Mamu**

**Lab Report Status Marks: ………………………………… Comments:..............................................**

**Signature:..................**

**... Date:...........................**

**Lab 01:**

## TITLE

KNN from Scratch: Flower and News Classification with Custom Evaluation Metrics

## OBJECTIVES

The main goal of this code is to help us learn the basics of machine learning by building the K-Nearest Neighbors (KNN) algorithm from scratch. By doing it ourselves, we get a clear idea of how it really works. We learn how to measure distance using the Euclidean formula, how to find the closest points (neighbors), and how to make predictions by seeing which class appears the most (majority vote). This hands-on practice gives us a better understanding than just using ready-made libraries.

### Ml Code:

import numpy as np

from collections import Counter

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

import matplotlib.pyplot as plt

class MyKNN:

    def \_\_init\_\_(self, k=3):

        self.k = k

    def fit(self, features, labels):

        self.train\_data = features

        self.train\_labels = labels

    def predict(self, test\_data):

        predictions = []

        for test\_point in test\_data:

            distances = [np.sqrt(np.sum((test\_point - train\_point) \*\* 2)) for train\_point in self.train\_data]

            nearest\_indices = np.argsort(distances)[:self.k]

            nearest\_labels = [self.train\_labels[i] for i in nearest\_indices]

            most\_common\_label = Counter(nearest\_labels).most\_common(1)[0][0]

            predictions.append(most\_common\_label)

        return np.array(predictions)

def calculate\_accuracy(true\_labels, predicted\_labels):

    return np.mean(true\_labels == predicted\_labels)

def create\_confusion\_matrix(true\_labels, predicted\_labels, num\_classes):

    matrix = np.zeros((num\_classes, num\_classes), dtype=int)

    for true, pred in zip(true\_labels, predicted\_labels):

        matrix[true][pred] += 1

    return matrix

def get\_metrics(true\_labels, predicted\_labels, num\_classes):

    matrix = create\_confusion\_matrix(true\_labels, predicted\_labels, num\_classes)

    precision = np.diag(matrix) / np.sum(matrix, axis=0)

    recall = np.diag(matrix) / np.sum(matrix, axis=1)

    f1\_score = 2 \* (precision \* recall) / (precision + recall)

    return np.nanmean(precision), np.nanmean(recall), np.nanmean(f1\_score)

data = load\_iris()

features, labels = data.data, data.target

best\_k = 1

highest\_accuracy = 0

accuracy\_list = []

k\_options = range(1, 16)

for k in k\_options:

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(features, labels, test\_size=0.3, random\_state=42)

    model = MyKNN(k)

    model.fit(X\_train, y\_train)

    predictions = model.predict(X\_test)

    acc = calculate\_accuracy(y\_test, predictions)

    accuracy\_list.append(acc)

    if acc > highest\_accuracy:

        highest\_accuracy = acc

        best\_k = k

print(f"Best K Value: {best\_k}, Accuracy: {highest\_accuracy:.3f}")

X\_train, X\_test, y\_train, y\_test = train\_test\_split(features, labels, test\_size=0.3, random\_state=42)

my\_knn = MyKNN(best\_k)

my\_knn.fit(X\_train, y\_train)

my\_predictions = my\_knn.predict(X\_test)

sklearn\_knn = KNeighborsClassifier(n\_neighbors=best\_k)

sklearn\_knn.fit(X\_train, y\_train)

sklearn\_predictions = sklearn\_knn.predict(X\_test)

my\_acc = calculate\_accuracy(y\_test, my\_predictions)

sklearn\_acc = calculate\_accuracy(y\_test, sklearn\_predictions)

my\_precision, my\_recall, my\_f1 = get\_metrics(y\_test, my\_predictions, 3)

sklearn\_precision, sklearn\_recall, sklearn\_f1 = get\_metrics(y\_test, sklearn\_predictions, 3)

print(f"\nCustom KNN - Accuracy: {my\_acc:.3f}, Precision: {my\_precision:.3f}, Recall: {my\_recall:.3f}, F1 Score: {my\_f1:.3f}")

print(f"Sklearn KNN - Accuracy: {sklearn\_acc:.3f}, Precision: {sklearn\_precision:.3f}, Recall: {sklearn\_recall:.3f}, F1 Score: {sklearn\_f1:.3f}")

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(k\_options, accuracy\_list, marker='o')

plt.axvline(x=best\_k, color='red', linestyle='--', label=f'Best k = {best\_k}')

plt.xlabel('K Value')

plt.ylabel('Accuracy')

plt.title('Accuracy for Different K Values')

plt.legend()

plt.grid(True)

plt.subplot(1, 2, 2)

cm = create\_confusion\_matrix(y\_test, my\_predictions, 3)

plt.imshow(cm, cmap='Blues')

plt.colorbar()

plt.title('Confusion Matrix (Custom KNN)')

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

for i in range(3):

    for j in range(3):

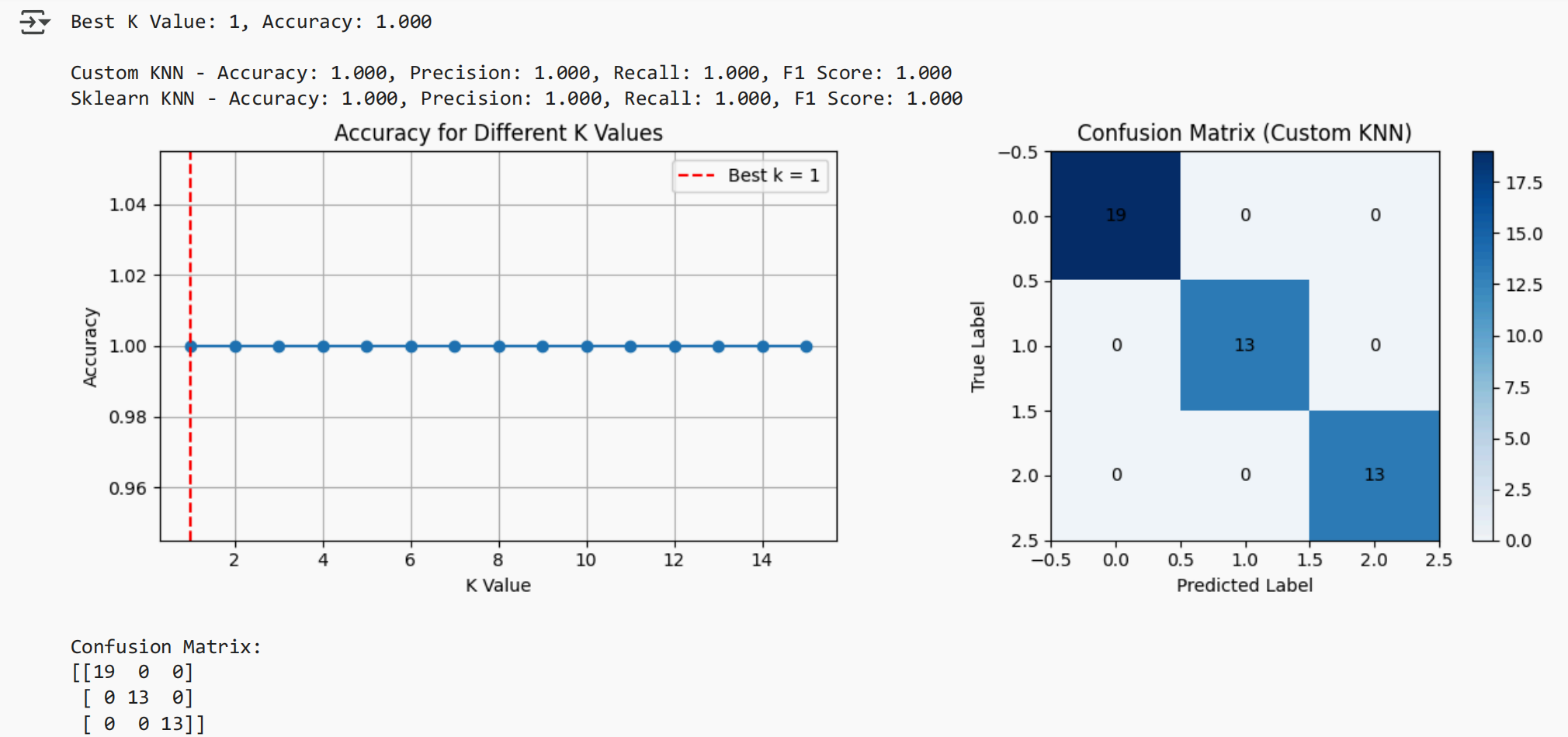
        plt.text(j, i, cm[i, j], ha='center', va='center', color='black')

plt.tight\_layout()

plt.show()

print(f"\nConfusion Matrix:\n{cm}")

## OUTPUT:



# DISCUSSION

This code demonstrates the K-Nearest Neighbors (KNN) algorithm using the Iris dataset. It creates a custom KNN model using the Euclidean formula, closest neighbors, and majority voting. The model is trained and tested with different k values, and compared with Scikit-Learn's built-in model. Evaluation metrics like accuracy, precision, recall, and F1 score are checked. The results are presented in a chart and confusion matrix.

# 8.Reference:

<https://github.com/mdjabedmollah/ml-learning/blob/main/ID_222002167_CSE412_222D3_LabReport02_knnFromScratch_.ipynb>

**Date and Time: 05/08/2025 08.30pm**