Green University of Bangladesh
Department of Computer Science and Engineering (CSE)
Faculty of Sciences and Engineering
Semester: (Spring, Year:2025), B.Sc. in CSE (Day)

Report Title: KNN from Scratch: Flower and News Classifcation with Custom Evaluation Metrics.

CourseTitle: Machine Learning Lab
Course Code: CSE-412 Section:221-D13

Student Details

Name	ID
Abu Bakkar Siddik	221902265

Course Teacher's Name : Md. Sabbir Hosen Mamun
Submission Date: 07.08.2025

[For Teachers use only: Don't Write Anything inside this box]

Marks: …………………………………	Signature:.....................
Comments:..	Date:..............................

1.TITLE OF THE LAB REPORT

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

2. OBJECTIVES

The objective of this experiment is to implement the K-Nearest Neighbors (KNN) classification algorithm from scratch in Python and evaluate its performance on two datasets: the Iris flower dataset and a custom News classification dataset. The tasks include:

1. Understanding the working principle of the KNN algorithm.
2. Developing custom implementations for accuracy, confusion matrix, precision, recall, and F1-score.
3. Training and testing the custom KNN classifier on both datasets.
4. Determining the optimal value of k (number of neighbors) and best train-test split ratio for maximum accuracy.
5. Comparing the performance of the custom KNN implementation with scikit-learn’s built-in KNN.

.

3. IMPLEMENTATION

Code:

```
import numpy as np

import pandas as pd

from collections import Counter
```

```
from sklearn.datasets import load_iris

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder from

sklearn.neighbors import KNeighborsClassifier
```

```
dataset = pd.read_csv('/content/drive/MyDrive/news_dataset.csv')

dataset.head()


class CustomKNN:

    def __init__(self, k=3):

        self.k = k

    def fit(self, X_train, y_train):

        self.X_train = X_train

        self.y_train = y_train

    def predict(self, X_test):

        predictions = []

        for x in X_test:

            distances = np.linalg.norm(self.X_train - x, axis=1)

            k_indices = np.argsort(distances)[:self.k]

            k_nearest_labels = self.y_train[k_indices]

            most_common =

Counter(k_nearest_labels).most_common(1)[0][0]

            predictions.append(most_common)

        return np.array(predictions)
```

```
def accuracy(y_true, y_pred):

    return np.sum(y_true == y_pred) / len(y_true)



def confusion_matrix(y_true, y_pred):

    unique_labels = np.unique(np.concatenate((y_true, y_pred)))

    cm = np.zeros((len(unique_labels), len(unique_labels)), dtype=int)

    for i, true_label in enumerate(unique_labels):

        for j, pred_label in enumerate(unique_labels):

            cm[i, j] = np.sum((y_true == true_label) & (y_pred ==
pred_label))

    return cm, unique_labels



def precision_recall_f1(y_true, y_pred):

    cm, labels = confusion_matrix(y_true, y_pred)

    precisions, recalls, f1s = [], [], []

    for i in range(len(labels)):

        TP = cm[i, i]

        FP = np.sum(cm[:, i]) - TP

        FN = np.sum(cm[i, :]) - TP

        precision = TP / (TP + FP) if (TP + FP) > 0 else

        0 recall = TP / (TP + FN) if (TP + FN) > 0 else 0

        f1 = 2 * precision * recall / (precision + recall) if
(precision + recall) > 0 else 0

        precisions.append(precision)

        recalls.append(recall)

        f1s.append(f1)
```

```
    return np.mean(precisions), np.mean(recalls), np.mean(f1s)


y_true = np.array([0, 1, 2, 2, 0, 1, 2])

y_pred = np.array([0, 2, 1, 2, 0, 0, 2])


acc = accuracy(y_true, y_pred)

cm, labels = confusion_matrix(y_true, y_pred)

precision, recall, f1 = precision_recall_f1(y_true, y_pred)


print("Accuracy:", acc)

print("Confusion Matrix:\n", cm)

print("Labels:", labels)

print(f"Precision: {precision:.2f}, Recall: {recall:.2f}, F1-score:
{f1:.2f}")




def find_best_k_split(X, y, k_values, split_ratios):

    best_score = 0

    best_k = None

    best_split = None

    for k in k_values:

        for split in split_ratios:

            X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=split, random_state=42)

            model = CustomKNN(k=k)

            model.fit(X_train, y_train)

            y_pred = model.predict(X_test)

            score = accuracy(y_test, y_pred)
```

```
            if score > best_score:

                best_score = score

                best_k = k

                best_split = split

    return best_k, best_split, best_score



iris = load_iris()

X_iris = iris.data

y_iris = iris.target


k_values = range(1, 16)

split_ratios = [0.2, 0.25, 0.3]


best_k_iris, best_split_iris, best_score_iris =
find_best_k_split(X_iris, y_iris, k_values, split_ratios)

print(f"Iris Best k: {best_k_iris}, Best split: {best_split_iris},
Accuracy: {best_score_iris:.2f}")


X_train, X_test, y_train, y_test = train_test_split(X_iris,
y_iris, test_size=best_split_iris, random_state=42)

custom_knn = CustomKNN(k=best_k_iris)

custom_knn.fit(X_train, y_train)

y_pred = custom_knn.predict(X_test)


acc = accuracy(y_test, y_pred)

cm, labels = confusion_matrix(y_test, y_pred)

precision, recall, f1 = precision_recall_f1(y_test, y_pred)
```

```
print("\nCustom KNN Iris Metrics:")
print(f"Accuracy: {acc:.2f}")
print(f"Confusion Matrix:\n{cm}")
print(f"Precision: {precision:.2f}, Recall: {recall:.2f}, F1-Score: {f1:.2f}")

data = {
    'text': [
        'Team wins championship', 'Election results announced', 'Player scores hat-trick', 'New policy debate',
        'Match postponed due to rain', 'Government passes new bill', 'Coach resigns after loss', 'Parliament in session',
        'Fans celebrate victory', 'Budget discussion heats up'
    ]*10,
    'category': ['sports', 'politics', 'sports', 'politics', 'sports', 'politics', 'sports', 'politics', 'sports', 'politics'] * 10
}
news_df = pd.DataFrame(data)

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
news_df['category_encoded'] = label_encoder.fit_transform(news_df['category'])

tfidf_vectorizer = TfidfVectorizer(max_features=1000)
X_news = tfidf_vectorizer.fit_transform(news_df['text']).toarray()
```

```
y_news = news_df['category_encoded'].values


k_values_news = range(1, 16)

split_ratios_news = [0.2, 0.25, 0.3]


best_k_news, best_split_news, best_score_news =
find_best_k_split(X_news, y_news, k_values_news, split_ratios_news)

print(f"News Best k: {best_k_news}, Best split: {best_split_news},
Accuracy: {best_score_news:.2f}")


from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier


X_train_news, X_test_news, y_train_news, y_test_news =
train_test_split(X_news, y_news, test_size=best_split_news,
random_state=42)


sklearn_knn_news = KNeighborsClassifier(n_neighbors=best_k_news)

sklearn_knn_news.fit(X_train_news, y_train_news)

y_pred_sklearn_news = sklearn_knn_news.predict(X_test_news)


acc_sk_news = accuracy(y_test_news, y_pred_sklearn_news)

precision_sk_news, recall_sk_news, f1_sk_news =
precision_recall_f1(y_test_news, y_pred_sklearn_news)


print("\nScikit-learn KNN News Metrics:")

print(f"Accuracy: {acc_sk_news:.2f}")

print(f"Precision: {precision_sk_news:.2f}, Recall:
{recall_sk_news:.2f}, F1-Score: {f1_sk_news:.2f}")
```

4.OUTPUT

```python
y_true = np.array([0, 1, 2, 2, 0, 1, 2])
y_pred = np.array([0, 2, 1, 2, 0, 0, 2])

acc = accuracy(y_true, y_pred)
cm, labels = confusion_matrix(y_true, y_pred)
precision, recall, f1 = precision_recall_f1(y_true, y_pred)

print("Accuracy:", acc)
print("Confusion Matrix:\n", cm)
print("Labels:", labels)
print(f"Precision: {precision:.2f}, Recall: {recall:.2f}, F1-score: {f1:.2f}")
```

```
Accuracy: 0.5714285714285714
Confusion Matrix:
 [[2 0 0]
 [1 0 1]
 [0 1 2]]
Labels: [0 1 2]
Precision: 0.44, Recall: 0.56, F1-score: 0.49
```


```python
y_true = np.array([0, 1, 2, 2, 0, 1, 2])
y_pred = np.array([0, 2, 1, 2, 0, 0, 2])

acc = accuracy(y_true, y_pred)
cm, labels = confusion_matrix(y_true, y_pred)
precision, recall, f1 = precision_recall_f1(y_true, y_pred)

print("Accuracy:", acc)
print("Confusion Matrix:\n", cm)
print("Labels:", labels)
print(f"Precision: {precision:.2f}, Recall: {recall:.2f}, F1-score: {f1:.2f}")
```

```
Accuracy: 0.5714285714285714
Confusion Matrix:
 [[2 0 0]
 [1 0 1]
 [0 1 2]]
Labels: [0 1 2]
Precision: 0.44, Recall: 0.56, F1-score: 0.49
```

```python
k_values_news = range(1, 16)
split_ratios_news = [0.2, 0.25, 0.3]

best_k_news, best_split_news, best_score_news = find_best_k_split(X_news, y_news, k_values_news, split_ratios_
print(f"News Best k: {best_k_news}, Best split: {best_split_news}, Accuracy: {best_score_news:.2f}")
```

```
News Best k: 1, Best split: 0.2, Accuracy: 1.00
```

```python
[36] from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

X_train_news, X_test_news, y_train_news, y_test_news = train_test_split(X_news, y_news, test_size=best_split_n

sklearn_knn_news = KNeighborsClassifier(n_neighbors=best_k_news)
sklearn_knn_news.fit(X_train_news, y_train_news)
y_pred_sklearn_news = sklearn_knn_news.predict(X_test_news)

acc_sk_news = accuracy(y_test_news, y_pred_sklearn_news)
precision_sk_news, recall_sk_news, f1_sk_news = precision_recall_f1(y_test_news, y_pred_sklearn_news)

print("\nScikit-learn KNN News Metrics:")
print(f"Accuracy: {acc_sk_news:.2f}")
print(f"Precision: {precision_sk_news:.2f}, Recall: {recall_sk_news:.2f}, F1-Score: {f1_sk_news:.2f}")
```

```
Scikit-learn KNN News Metrics:
Accuracy: 1.00
Precision: 1.00, Recall: 1.00, F1-Score: 1.00
```

5. Analysis and Discussion

The implemented KNN model correctly classified both the Iris and News datasets with high accuracy. The optimal k and split ratios were determined through experimentation, showing that accuracy depends on neighbor count and training size. Custom metrics confirmed perfect or near-perfect precision, recall, and F1-scores, indicating balanced predictions without bias toward any class. The News dataset, having simpler and repeated text samples, was easier to classify, while the Iris dataset required careful tuning of k for best performance. Overall, the custom KNN implementation performed comparably to scikit-learn's version.

GitHub Link:

https://github.com/ShawonTech/Machine-Learning-Lab.