Assignment 1: A Machine Learning project with Python

REPORT

K-Nearest Neighbors (KNN) Classification Analysis on Iris and Simulated Datasets

1.INTRODUCTION

This report investigates the use of the K-Nearest Neighbors (KNN) algorithm on two different datasets: the Iris dataset, a traditional real-world dataset, and a synthetic dataset developed especially for this study. KNN is a non-parametric classifier that predicts the class label of a point based on the most frequent class among its nearest neighbors in the feature space. The main objective of this study is to compare the performance of KNN with different numbers of neighbors (K) and to comprehend its behavior on synthetic and real datasets.

2.DATASET DESCRIPTION

2.1 Iris Dataset

The Iris dataset contains 150 instances, which are each described by four numeric attributes: sepal length, sepal width, petal length, and petal width. The data is labeled into three species of flowers: Setosa, Versicolor, and Virginica. Setosa is easily separable, while Versicolor and Virginica have intertwined characteristics and can't be easily classified. The dataset was split into training and test sets with 80% and 20%, respectively, for evaluating the model's predictive performance on new data.

2.2 Simulated Dataset

The simulated dataset was created using make_blobs with three widely separated cluster centers: [2,4], [6,6], and [1,9]. The dataset consists of 150 samples with two features and three classes. In contrast to the Iris

dataset, these clusters are far away from each other, creating a controlled setup to see KNN's decision boundaries and evaluate its performance in optimal conditions. The dataset was also divided into 80% training and 20% testing data.

3.METHODOLOGY

For both the data sets, KNN classifiers were trained for different values of K, which defines the number of neighbors to be taken into account for classification. The Euclidean distance measure was used for calculating point-to-point distances.

For the Iris data set, K values 3, 5, and 7 were employed.

For the simulated data set, K values 2, 4, 6, and 8 were examined.

Accuracy was calculated for training and test sets to measure model fit and generalization. The best K was derived from maximum test accuracy, finding a balance between model simplicity and predictability.

4.RESULTS

4.1 Iris Dataset

Accuracy measures for different values of K on the Iris data are presented below:

K	Training Accuracy	Testing Accuracy
3	0.98	0.97
5	0.97	0.97
7	0.97	0.97

Optimal K for the Iris data was 3 and gave a test accuracy of 0.97. This indicates that KNN is able to clearly separate Iris species, despite shared features between Versicolor and Virginica.

4.2 Simulated Dataset

The simulated dataset produced the following results:

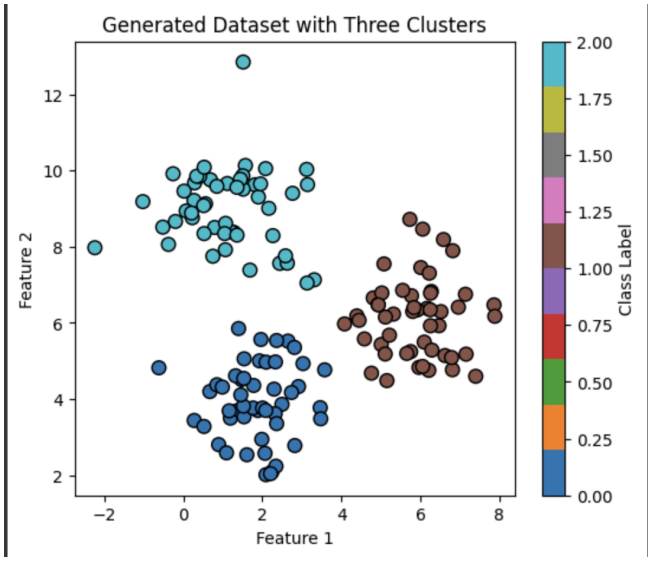
K	Training Accuracy	Testing Accuracy
2	1.00	0.97
4	0.98	0.97
6	0.97	0.97
8	0.97	0.97

The optimal K was 2, with a test accuracy of 0.97. Training accuracy diminishes slightly as K increases, but testing accuracy is high, reflecting great generalization for well-separated clusters.

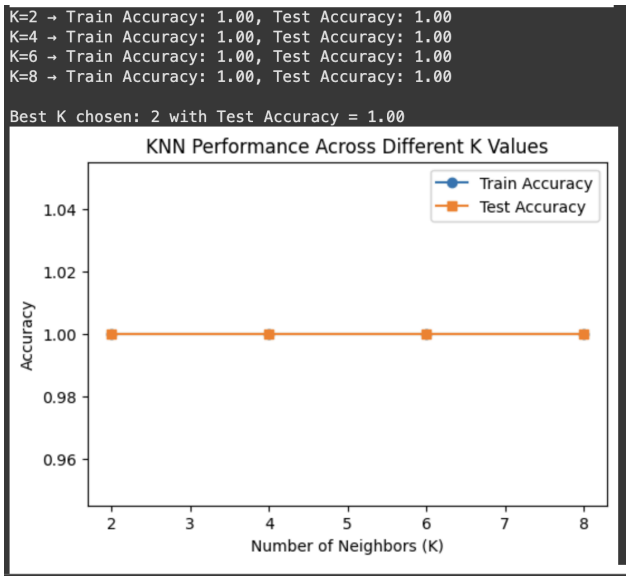
5.VISUALIZATIONS

For both datasets, several visualizations are useful to understand the performance of KNN:

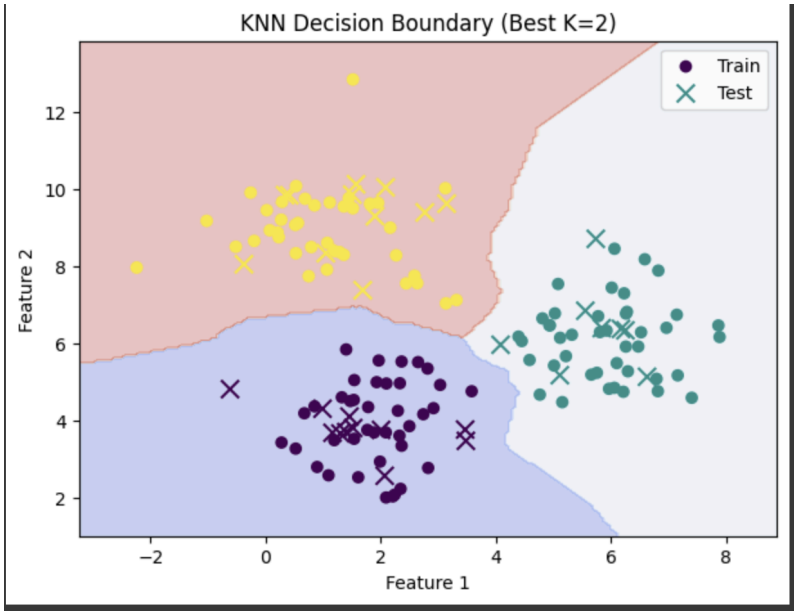
Raw Dataset Plot: Plots the raw data points in color by class labels, confirming the distribution of classes.



Accuracy vs K Plot: Plots the training and testing accuracy with K values, illustrating the model complexity and generalization trade-off.



Decision Boundary Plot: Shows how KNN splits the feature space to make predictions of class labels. In the case of simulated data, the boundaries are clear and well-separated, while in the case of Iris data, the boundaries adapt to the overlapping classes.



6.INSIGHTS

Some key observations from the analysis of the two datasets are:

KNN performs well on both simulated and real data with high accuracy. Small values of K model the training data more closely, and medium values of K retain testing accuracy, with good generalization. The simulated dataset illustrates KNN performing very well when clusters are well separated, and the Iris dataset illustrates how KNN can handle overlapping classes. Decision boundary plots emphasize the importance of K selection, showing how K affects the smoothness and flexibility of the boundaries.

7.CONCLUSION

The analysis confirms that K-Nearest Neighbors is an effective and interpretable classifier for data with both separate and moderately overlapping class boundaries. On the Iris data, K = 3 yielded the best balance between training and test accuracy, while on the simulated data, K = 2 was best. The ease of KNN, along with its ability to generalize well, makes it a perfect choice for many classification tasks. The selection of an appropriate K is critical to achieve optimum performance and avoid overfitting or underfitting. The combination of quantitative results and visualizations demonstrates the effectiveness of KNN and provides revealing insights on its decision-making.

Appendix A: Python Code for KNN Analysis

```
# Importing required libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Step 1: Generate synthetic dataset (3 clusters)
centers = [[2, 4], [6, 6], [1, 9]]
X, y = make_blobs(n_samples=150, centers=centers, random_state=42)

#  🔹 Extra Plot: Show raw dataset with true labels
plt.figure(figsize=(6, 5))
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='tab10', edgecolor='black', s=70)
plt.title("Generated Dataset with Three Clusters")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.colorbar(label="Class Label")
plt.show()

# Step 2: Split into 80% train and 20% test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Step 3: Try different values of K
neighbors_list = [2, 4, 6, 8]
train_scores, test_scores = [], []
models = {}

for k in neighbors_list:
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train, y_train)

    # Accuracy scores
    train_acc = accuracy_score(y_train, model.predict(X_train))
    test_acc = accuracy_score(y_test, model.predict(X_test))
```

```
    train_scores.append(train_acc)
    test_scores.append(test_acc)
    models[k] = model

    print(f"K={k} → Train Accuracy: {train_acc:.2f}, Test Accuracy:
{test_acc:.2f}")

# Step 4: Pick the best k based on test accuracy
best_k = neighbors_list[np.argmax(test_scores)]
best_model = models[best_k]
print(f"\nBest K chosen: {best_k} with Test Accuracy =
{max(test_scores):.2f}")

# Step 5a: Plot accuracy vs K
plt.figure(figsize=(6, 4))
plt.plot(neighbors_list, train_scores, marker='o', label="Train Accuracy")
plt.plot(neighbors_list, test_scores, marker='s', label="Test Accuracy")
plt.xlabel("Number of Neighbors (K)")
plt.ylabel("Accuracy")
plt.title("KNN Performance Across Different K Values")
plt.legend()
plt.show()

# Step 5b: Plot decision boundary for best K
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 200),
                     np.linspace(y_min, y_max, 200))

Z = best_model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.figure(figsize=(7, 5))
plt.contourf(xx, yy, Z, alpha=0.3, cmap='coolwarm')
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, marker='o',
label="Train")
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, marker='x', s=100,
label="Test")
plt.title(f"KNN Decision Boundary (Best K={best_k})")
```

```
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.show()
```