Advance Machine Learning

Assignment-1

Jayanth Yadlapalli
Department of Information Systems and Business Analytics,
Kent State University

Chaojiang (CJ) Wu, Ph.D.

02-11-2025

INTRODUCTION

The results of a K-Nearest Neighbors (KNN) classification experiment are presented in this report. There are two main tasks involved in the project:

1. Using the Iris dataset and KNN.
2. Replicating the study with a make_blobs function-generated simulated dataset.

Using Python to Understand KNN on IRIS Dataset

Loading and splitting the data:

- Datasets.load_iris() was used to import the Iris dataset.
- Using train_test_split, the data was split between training and testing sets in an 80-20 ratio. Randomness was managed by setting random_state=12.

Model Training and Evaluation:

- The training data was used to build a default KNN classifier.
- The model's accuracy was 96.67% on the test set and 97.5% on the training set.
- When hyperparameters like n_neighbors=5 and metric='minkowski' were applied to a re-trained model, the outcomes stayed consistent, demonstrating the usefulness of the default configurations.

Results:

- **Training Accuracy:** 97.5%
- **Testing Accuracy:** 96.67%

Implementation on BLOB dataset

Loading and splitting dataset:

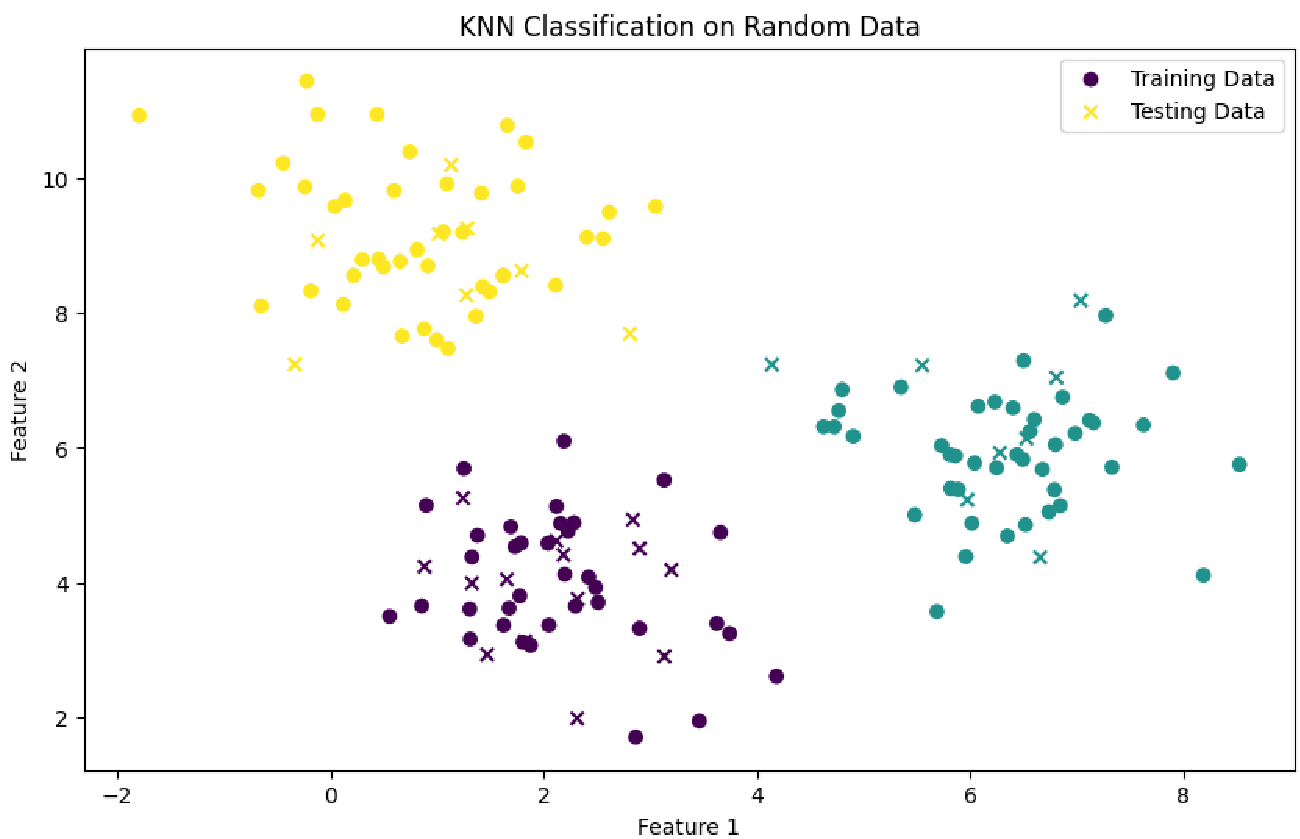
- Make_blobs was used to construct a dataset with three different centers.
- Train_test_split was used to separate the data into training (80%) and testing (20%) groups.

Model Training and Evaluation:

- The simulated dataset was used to train a KNN classifier with n_neighbors=5.
- The model scored 100% on both the training and testing sets, achieving perfect accuracy.

Visualization:

- To graphically depict the data points and their labels, a scatter plot was created. The dataset's potential for KNN classification was demonstrated by the plot, which verified distinct cluster separation.



Results Summary for Simulated Dataset:

- **Training Accuracy:** 100%
- **Testing Accuracy:** 100%

PYTHON CODE

```
# Load the libraries
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = datasets.load_iris()
data, labels = iris.data, iris.target

# Split the dataset into training (80%) and testing (20%) sets
train_data, test_data, train_labels, test_labels = train_test_split(
    data, labels, train_size=0.8, test_size=0.2, random_state=12
)

# Initialize and train the default KNN classifier
knn = KNeighborsClassifier()
knn.fit(train_data, train_labels)

# Predict labels on training set
train_predictions = knn.predict(train_data)
```

```
# Print training predictions and accuracy
print("Predictions from the classifier:")
print(train_predictions)
print("Target values:")
print(train_labels)
print(f"Training Accuracy: {accuracy_score(train_labels, train_predictions) * 100:.2f}%")

# Initialize and train customized KNN classifier with specific parameters
custom_knn = KNeighborsClassifier(
    algorithm='auto',
    leaf_size=30,
    metric='minkowski',
    p=2,  # Equivalent to Euclidean distance
    n_jobs=1,
    n_neighbors=5,
    weights='uniform'
)
custom_knn.fit(train_data, train_labels)

# Predict labels on test set
test_predictions = custom_knn.predict(test_data)

# Print test accuracy
print(f"Testing Accuracy with Custom KNN: {accuracy_score(test_labels, test_predictions) * 100:.2f}%")
```

```
# Define cluster centers for data generation
cluster_centers = [[2, 4], [6, 6], [1, 9]]  # Locations of clusters
num_clusters = len(cluster_centers)

# Generate synthetic dataset
random_data, random_labels = make_blobs(n_samples=150, centers=np.array(cluster_centers),
random_state=1)

# Split dataset into 80% training and 20% testing
X_train_sim, X_test_sim, y_train_sim, y_test_sim = train_test_split(
    random_data, random_labels, train_size=0.8, random_state=12
)

# Initialize and train KNN classifier
knn_random = KNeighborsClassifier(n_neighbors=5)
knn_random.fit(X_train_sim, y_train_sim)

# Predict labels for training and testing sets
train_preds_sim = knn_random.predict(X_train_sim)
test_preds_sim = knn_random.predict(X_test_sim)

# Compute and display accuracy scores
train_acc_sim = accuracy_score(y_train_sim, train_preds_sim)
test_acc_sim = accuracy_score(y_test_sim, test_preds_sim)

print("\nResults on Random Dataset:")
```

```
print(f"Training Accuracy: {train_acc_sim * 100:.2f}%")
print(f"Testing Accuracy: {test_acc_sim * 100:.2f}%")


# Visualizing training data and test predictions
plt.figure(figsize=(10, 6))
plt.scatter(X_train_sim[:, 0], X_train_sim[:, 1], c=y_train_sim, marker='o', label='Training Data')
plt.scatter(X_test_sim[:, 0], X_test_sim[:, 1], c=test_preds_sim, marker='x', label='Testing Data')
plt.title('KNN Classification on Random Data')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```

CONCLUSION

- On both the simulated and Iris datasets, the KNN method performed quite well, attaining high accuracy in both situations.
- Key findings include:
    - The simulated dataset was especially well-suited for KNN, with clearly separable data points leading to perfect classification.
    - The model demonstrated strong generalization capabilities, achieving high testing accuracy on both datasets.
    - These experiments demonstrate KNN's efficacy for classification tasks where distinct class separation is evident, as demonstrated in both the simulated and Iris datasets.