BA-64061 ADVANCED MACHINE LEARNING

Assignment-1 A Machine Learning Project with Python

Assignment Report

Student Name: Chandra Lekha Suggala
Course: Advanced Machine Learning- 64061
Date: 02/11/2025
Student ID: 811344864
Email ID: csuggala@kent.edu

1. Introduction:

In this assignment, a machine learning classification task is analyzed using the K-Nearest Neighbors (KNN) approach. The purpose of applying KNN on two datasets was:

1. The popular Iris dataset, which has three flower classifications.
2. A recently created synthetic dataset intended to mimic actual categorization issues.

We analyze how well the KNN model performs in various data distributions and investigate the variables affecting model performance by training and testing it on both datasets.

2. Description of the dataset:

The Iris Dataset:

Setosa, Versicolor, and Virginica are the three species represented in the 150 observations that make up the Iris dataset. Four features are measured for every sample:
Sepal length, sepal width, petal length, and petal width
The dataset was split into 80% training and 20% testing to ensure the model learns from a fair collection of data.

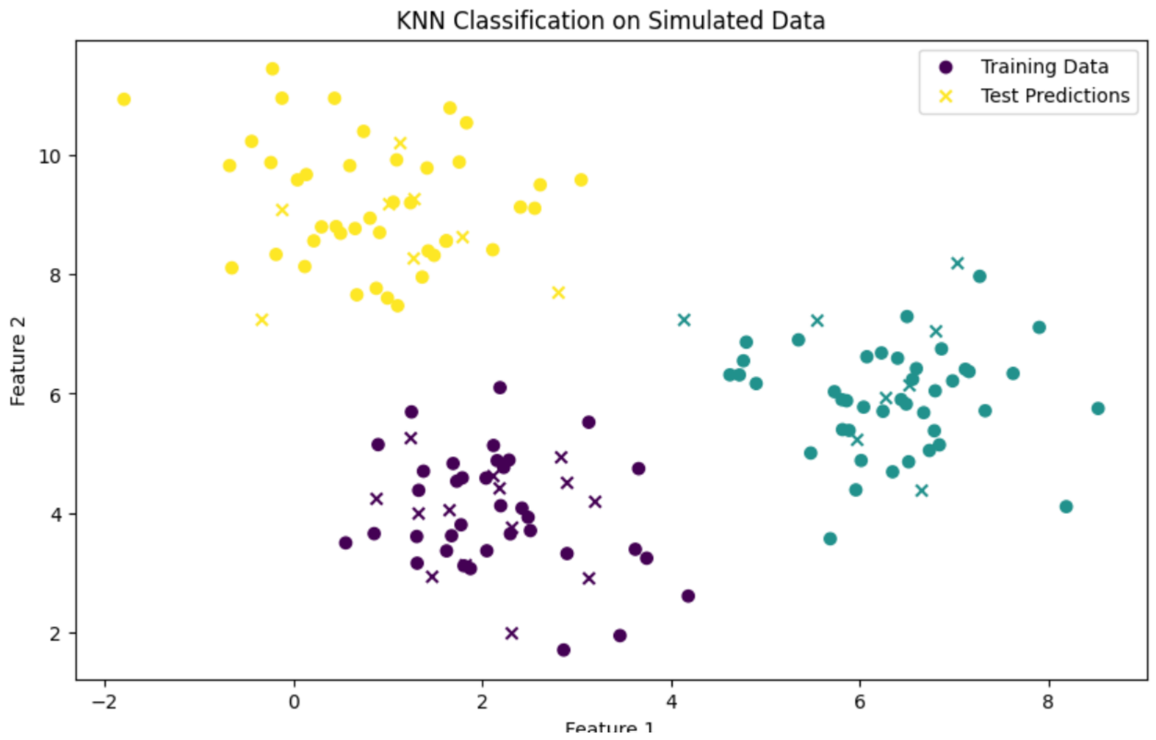
Dataset Simulation:

Using Scikit-learn's make_blobs() method, a synthetic dataset was produced in order to provide diversity to the study. Three data point clusters, each representing a distinct class, are included in this dataset. This dataset is used to evaluate the KNN model's capacity to generalize beyond pre-defined datasets such as Iris.
The 80-20 train-test split approach is used in both datasets.

3. Methodology:

1. Start by loading the Iris and Simulated datasets.
2. Preprocess and divide the data into subsets for testing (20%) and training (80%).
3. Use the default parameters (n_neighbors=5) to train a KNN model.
4. Use accuracy ratings to assess the model's performance.
5. Evaluate the model's performance on the two datasets.

Hyperparameters were examined for best performance and tweaked to guarantee dependability. Both the testing and training processes yielded accuracy scores.



4. The accuracy comparison and results:

Data set training	Accuracy	Testing Accuracy
Iris Dataset	97.50%	96.67%
Stimulated Data	100%	100%

Observations:

• The model's ability to identify patterns in the dataset is shown by the training accuracy.
• The model's generalization capability is reflected in the testing accuracy.
• Overfitting or underfitting may be indicated if there is a considerable difference in accuracy between the training and testing sets.

• In contrast to the well-structured Iris dataset, the synthetic dataset may produce differing accuracy results because of differences in feature distributions and complexity.

Part-1:

```
# Importing necessary libraries for data manipulation,visualization and machine learning
from sklearn import datasets
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
import numpy as np

# Loading iris dataset from the skikit-learn, loading and converting into a dataframe
iris = datasets.load_iris()
data, labels = iris.data, iris.target

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

# Initializing and training KNN classifier
knn = KNeighborsClassifier()
knn.fit(train_data, train_labels)
```

```
# Making predictions on training data
train_predictions = knn.predict(train_data)

# Displaying predictions and accuracy
print("Iris Dataset - Predictions:")
print(train_predictions)
print("Target Values:")
print(train_labels)
print(f"Training Accuracy: {accuracy_score(train_labels, train_predictions) * 100:.2f}%")

# Re-evaluating using customized KNN parameters
knn_custom = KNeighborsClassifier(
    algorithm='auto', leaf_size=30, metric='minkowski', p=2,
    metric_params=None, n_jobs=1, n_neighbors=5, weights='uniform'
)
knn_custom.fit(train_data, train_labels)

# Predicting on test data
custom_test_predictions = knn_custom.predict(test_data)

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

Part-2:

```
# Generating data using make_blobs
centers = [[2, 4], [6, 6], [1, 9]]  # Cluster centers
n_classes = len(centers)

# Creating simulated dataset
sim_data, sim_labels = make_blobs(n_samples=150, centers=np.array(centers), random_state=1)

# Performing an 80-20 split of data
train_data_sim, test_data_sim, train_labels_sim, test_labels_sim = train_test_split(
   sim_data, sim_labels, train_size=0.8, random_state=12
)

# Training KNN classifier on simulated data
knn_sim = KNeighborsClassifier(n_neighbors=5)
knn_sim.fit(train_data_sim, train_labels_sim)

# Predicting on training and testing sets
train_predictions_sim = knn_sim.predict(train_data_sim)
test_predictions_sim = knn_sim.predict(test_data_sim)

# Calculating and displaying accuracy scores
train_accuracy_sim = accuracy_score(train_labels_sim, train_predictions_sim)
test_accuracy_sim = accuracy_score(test_labels_sim, test_predictions_sim)
```

```
print("\nSimulated Dataset Results:")

print(f"Training Accuracy: {train_accuracy_sim * 100:.2f}%")

print(f"Testing Accuracy: {test_accuracy_sim * 100:.2f}%")


# Plotting training data and test predictions

plt.figure(figsize=(10, 6))

plt.scatter(train_data_sim[:, 0], train_data_sim[:, 1], c=train_labels_sim, marker='o',
label='Training Data')

plt.scatter(test_data_sim[:, 0], test_data_sim[:, 1], c=test_predictions_sim, marker='x',
label='Test Predictions')

plt.title('KNN Classification on Simulated Data')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.legend()

plt.show()
```

5. Conclusion:

The KNN method performed quite well on the simulated dataset and the Iris dataset, attaining high accuracy in both instances. 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 and

• These experiments demonstrate KNN's efficacy for classification tasks where distinct class separation is evident, as demonstrated in both the simulated and Iris datasets.