Kyle Tranfaglia

COSC 311

Project 01

11 Apr. 2024

Wifi Localization Machine Learning Analysis

Code:

```
'''
Kyle Tranfaglia
COSC311 - Project01
Last updated 04/08/24
This program uses the "Wireless Indoor Localization Data Set" to perform a
Self-test for KNN and DT algorithms,
Independent-test for KNN and DT algorithms, and Classification model
finalization
'''
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,
classification_report, accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import StandardScaler

# Print the confusion matrix, classification report, accuracy score,
heatmap, and matrix display
def printResults(targetTest, prediction):
    confusionMatrix = confusion_matrix(targetTest, prediction)  # Create
confusion matrix

    # Display the confusion matrix, a classification report, and the
overall accuracy score of the prediction
    print("\nConfusion Matrix:\n", confusionMatrix)
    print("\nClassification Report:\n", classification_report(targetTest, prediction))
    print("Accuracy Score:", accuracy_score(targetTest, prediction))

    # Display a heatmap using matplotlib and the sklearn toolset to display a confusion matrix
    matrixDisplay = ConfusionMatrixDisplay(confusion_matrix = confusionMatrix)
    fig, ax = plt.subplots(figsize=(10, 8))  # Create layout and structure figure
    matrixDisplay.plot(ax = ax, cmap = 'Reds')  # Create Plot
    # Plot labels
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.title('Confusion Matrix')
    # plt.savefig('confusion_matrix.png')  # Save plot as png
    plt.show()  # Display plot

    # Display Heatmap using Seaborn
    # sb.heatmap(confusionMatrix, square = True, annot = True, fmt = 'd', cmap = 'Blues', cbar = False)  # Create heatmap with Seaborn
    # plt.savefig('heatmap.png')  # Save plot as png
    # plt.show()  # Display plot

# Main
# Read in data and label
wifi_data = pd.read_csv("wifi_localization.txt", sep="\t")
wifi_data.columns = ["column" + str(i + 1) for i in range(0, len(wifi_data.columns) - 1)] + ["Target"]  # Set up a list to store column labels (numbered columns)
print(wifi_data.info())

# Pre-test data analysis and set up

# Extract features and target labels
self_test_data = wifi_data.iloc[:, :-1].values
self_test_target = wifi_data.iloc[:, -1].values

# Scale the data
scaler = StandardScaler()
```

```
self_test_data_scaled = scaler.fit_transform(self_test_data)

wifi_data.plot()
# Plot labels
plt.figure(figsize=(10, 8))
plt.xlabel('Data Column')
plt.ylabel('Data Value')
plt.title('Wifi Localization Data Distribution')
plt.show()  # Display plot

# Iterate over each feature in the self_test_data to show data
distribution
for i in range(0, 7):
    plt.scatter(self_test_data[:, i], self_test_data[:, i+1 if i < 6 else
0], s=40)  # Create a scatter plot between the i and the i+1 feature
plt.figure(figsize=(10, 8))
plt.show()  # Plot scatter

# Declare variables for accuracy calculation and storage
neighbors = np.arange(1, 15)
accuracy = []

# Test different neighbor values to find the optimal number for the
dataset
for k in neighbors:
    knn = KNeighborsClassifier(n_neighbors=k)  # Create K Nearest
Neighbors classifier with k neighbors
    knn.fit(self_test_data_scaled, self_test_target)  # Fit the
classifiers to the self-test data
    accuracy.append(knn.score(self_test_data_scaled, self_test_target))  #
Append accuracy results to accuracy array

# Set up line graph
plt.figure(figsize=(10, 8))
plt.plot(neighbors, accuracy)
plt.title('KNN Self-test Accuracy by Neighbor Count')
plt.xlabel('Number of Neighbors')
plt.ylabel('Accuracy')
plt.show()
```

```
# Task 1 - Self-test

# Create K Nearest Neighbors and Decision Tree classifiers
# knn_classifier = KNeighborsClassifier(n_neighbors=5, algorithm="brute",
weights="uniform")
knn_classifier = KNeighborsClassifier(n_neighbors=4, algorithm="brute",
weights='distance')
# dt_classifier = DecisionTreeClassifier(criterion='entropy', max_depth=9,
min_samples_split=2)
dt_classifier = DecisionTreeClassifier(criterion='gini',
class_weight='balanced', max_features=3, max_depth=16,
min_samples_split=2)

# Fit the classifiers to the self-test data
knn_classifier.fit(self_test_data_scaled, self_test_target)
dt_classifier.fit(self_test_data_scaled, self_test_target)

# Get predictions
knn_self_test_predict = knn_classifier.predict(self_test_data_scaled)
dt_self_test_predict = dt_classifier.predict(self_test_data_scaled)

# Print results for self-test
printResults(self_test_target, knn_self_test_predict)  # knn results
printResults(self_test_target, dt_self_test_predict)  # dt results

'''
Varying numbers of neighbors, the algorithm type, and the weights for KNN
was tested. Also, the max_features, min_samples_split,
max_depth, criterion, and class_weight for the DT were tested in order to
determine the best combination of parameters to
produce the highest self test accuracy. From the data analysis, it was
determined that k=4 was the optimal amount of neighbors
for the KNN algorithm with "brute" as the algorithm and "distance" as the
weights. Relative to the neighbors, these parameters
had minor impacts on the accuracy, although they were able to bring the
accuracy to 100% for self-test. As for DT, max_depth
seemed to have the greatest impact on accuracy such that increasing the
max_depth value increased accuracy. It was found that
a value of 16 for max_depth yielded the greatest accuracy. Additionally,
it was found that the best value for the min_sample_splits
```

```
was 2, max_features was 3, class_weight was "balanced," and criterion was
"gini." It s important to note that the parameters beyond
max_depth had minor impact on accuracy, however, the combined parameters
brings the DT classifier to 100% accuracy for self-test.
'''

# Task 2 - Independent-test

# Split the data into training and testing sets
train_data, test_data, train_target, test_target =
train_test_split(wifi_data.iloc[:, :-1], wifi_data.iloc[:, -1],
test_size=0.3, random_state=7)

# Scale the training and testing data
train_data_scaled = scaler.fit_transform(train_data)
test_data_scaled = scaler.transform(test_data)

# Set up line graph
plt.figure(figsize=(10, 8))
plt.plot(neighbors, accuracy)
plt.title('KNN Self-test Accuracy by Neighbor Count')
plt.xlabel('Number of Neighbors')
plt.ylabel('Accuracy')
plt.show()

# Fit the classifiers to the independent-test data using same classifiers
from task 1
knn_classifier.fit(train_data_scaled, train_target)
dt_classifier.fit(train_data_scaled, train_target)

# Get predictions
knn_independent_test_predict = knn_classifier.predict(test_data_scaled)
dt_independent_test_predict = dt_classifier.predict(test_data_scaled)

# Print results for independent-test
printResults(test_target, knn_independent_test_predict)  # knn results
printResults(test_target, dt_independent_test_predict)  # dt results

'''
```

```
The same classifier models, as in the KNN and DT parameters were used for
the self test and the independent test. It was found
that this combination of parameters was optimal for both tests such that
the accuracy was the greatest for both the self-test
and independent test when using the current parameters. The same
parameters that were tested in task 1 were tested in task 2
as well, however, no changes to the model were able to surpass the
accuracy scores of the current model. Therefore, the model
was left unchanged moving from the self-test to the independent test as it
appears the model is optimized for both tests.

For KNN, the model had a final accuracy score of 98% with a 0.98 for
precision, recall, and f1-score, which is relatively high for
a machine learning algorithm. The Decision tree model was not far behind
with a final accuracy score of 96.67% with a 0.97 for
precision, recall, and f1-score, which is still relatively high, but not
not as good as the KNN model, despite model optimization.
Overall, both models performed well in the independent test, with KNN
being slightly more accurate, and neither algorithm could
be further optimized in transiton for the self-test to the independent
test in regards to the model parameters.
'''

# Task 3 - Classification model finalization

# Final optimization
# knn_classifier = KNeighborsClassifier(n_neighbors=4, algorithm="brute",
weights="distance")  # Create K Nearest Neighbors classifier

# # Split the data into training and testing sets
# train_data, test_data, train_target, test_target =
train_test_split(wifi_data.iloc[:, :-1], wifi_data.iloc[:, -1],
test_size=0.3, random_state=7)

# # Scale the training and testing data
# train_data_scaled = scaler.fit_transform(train_data)
# test_data_scaled = scaler.transform(test_data)

# # Fit the classifiers to the independent-test data
# knn_classifier.fit(train_data_scaled, train_target)
# dt_classifier.fit(train_data_scaled, train_target)

# # Get predictions and print results
# knn_predict = knn_classifier.predict(test_data_scaled)
# printResults(test_target, knn_predict)

'''
Final optimization: KNN is the best classifier for the dataset. Upon
previous testing and further parameter manipulation, the current KNN
classifier
with the given parameters is the most optimized version:
KNeighborsClassifier(n_neighbors=4, algorithm="brute", weights="distance")
This optimized classifier reaches 98% accuracy with random state of 7 for
train_test_split, which is the greatest accuracy achieved.
KNN was tested with all the algorithm options and weight options, and the
accuracy of 1-14 neighbors was tested to ensure the current
version was the most optimized. Given the 98% accuracy, relative to the
96.67% acuraccy for the Decision Tree classifier post optimization,
the KNN classifier will be used to conduct further testing with various
train and test data splits.
'''

# Lists to store test sizes and corresponding accuracies
test_sizes, accuracies = [0.1, 0.2, 0.3, 0.4, 0.5], []

# KNN classifier for all test sizes
for i in test_sizes:
    # Split the data into training and testing sets
    train_data, test_data, train_target, test_target =
train_test_split(wifi_data.iloc[:, :-1], wifi_data.iloc[:, -1],
test_size=i, random_state=7)

    # Scale the training and testing data
    scaler = StandardScaler()
    train_data_scaled = scaler.fit_transform(train_data)
    test_data_scaled = scaler.transform(test_data)

    knn_classifier.fit(train_data_scaled, train_target)  # Fit the
classifier to the training data
```

```
    knn_predict = knn_classifier.predict(test_data_scaled)  # Get predictions
    accuracies.append(accuracy_score(test_target, knn_predict))  # Calculate accuracy
    printResults(test_target, knn_predict)  # knn results

# Plot the accuracies
plt.figure(figsize=(10, 8))
plt.bar(test_sizes, accuracies, width=0.05)
# Add text labels for accuracy values at the top of each bar for closer analysis
for i, accuracy in enumerate(accuracies):
    plt.text(test_sizes[i], accuracy, f'{accuracy:.3f}', ha='center', va='bottom')
plt.xlabel('Test Size')
plt.ylabel('Accuracy')
plt.title('Accuracy per Test Size for KNN Classifier')
plt.xticks(test_sizes)
plt.grid(axis='y')
plt.show()
```

Responses:

Task 1:
Varying numbers of neighbors, the algorithm type, and the weights for KNN were tested. Also, the max_features, min_samples_split, max_depth, criterion, and class_weight for the DT were tested in order to determine the best combination of parameters to produce the highest self-test accuracy. From the data analysis, it was determined that k=4 was the optimal amount of neighbors for the KNN algorithm with "brute" as the algorithm and "distance" as the weights. Relative to the neighbors, these parameters had minor impacts on the accuracy, although they were able to bring the accuracy to 100% for self-test. As for DT, max_depth seemed to have the greatest impact on accuracy such that increasing the max_depth value increased accuracy. It was found that a value of 16 for max_depth yielded the greatest accuracy. Additionally, it was found that the best value for the min_sample_splits was 2, max_features was 3, class_weight was "balanced," and the criterion was "gini." It's important to note that the parameters beyond max_depth had a minor impact on accuracy, however, the combined parameters bring the DT classifier to 100% accuracy for self-test.

Task 2:
The same classifier models, as in the KNN and DT parameters were used for the self-test and the independent test. It was found that this combination of parameters was optimal for both tests

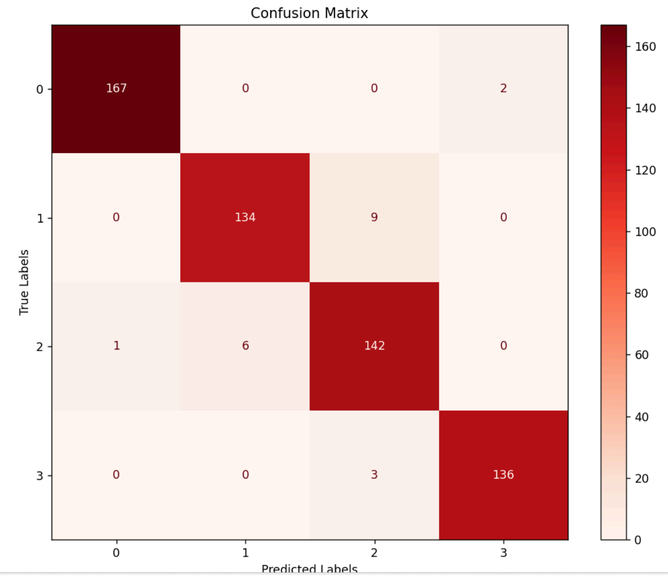
such that the accuracy was the greatest for both the self-test and independent test when using the current parameters. The same parameters that were tested in Task 1 were tested in Task 2 as well, however, no changes to the model were able to surpass the accuracy scores of the current model. Therefore, the model was left unchanged moving from the self-test to the independent test as it appears the model is optimized for both tests.

For KNN, the model had a final accuracy score of 98% with a 0.98 for precision, recall, and f1-score, which is relatively high for a machine learning algorithm. The Decision tree model was not far behind with a final accuracy score of 96.67% with a 0.97 for precision, recall, and f1-score, which is still relatively high, but not as good as the KNN model, despite model optimization. Overall, both models performed well in the independent test, with KNN being slightly more accurate, and neither algorithm could be further optimized in the transition from the self-test to the independent test in regard to the model parameters.

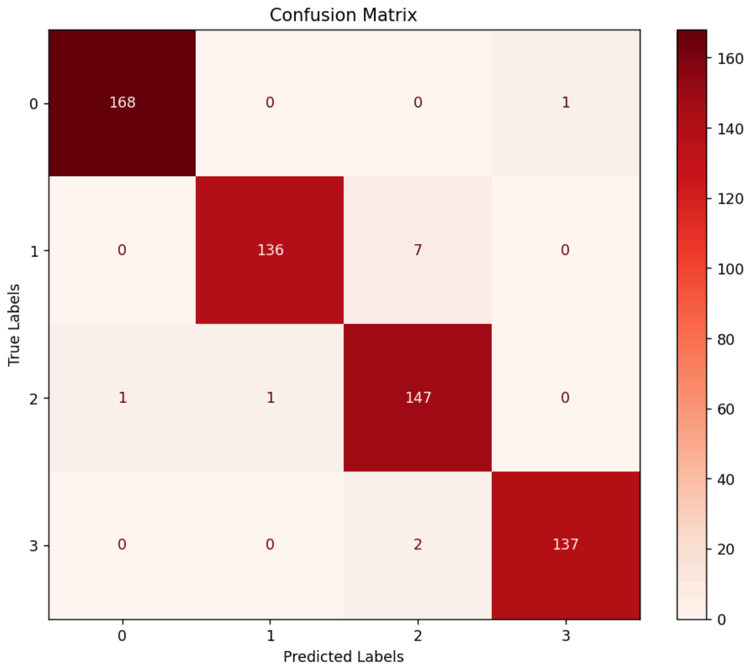
Task 3:
Final optimization: KNN is the best classifier for the dataset. Upon previous testing and further parameter manipulation, the current KNN classifier with the given parameters is the most optimized version: KNeighborsClassifier(n_neighbors=4, algorithm="brute", weights="distance") This optimized classifier reaches 98% accuracy with a random state of 7 for train_test_split, which is the greatest accuracy achieved. KNN was tested with all the algorithm options and weight options, and the accuracy of 1-14 neighbors was tested to ensure the current version was the most optimized. Given the 98% accuracy, relative to the 96.67% accuracy for the Decision Tree classifier post optimization, the KNN classifier will be used to conduct further testing with various train and test data splits.

Graphs and Heatmaps:

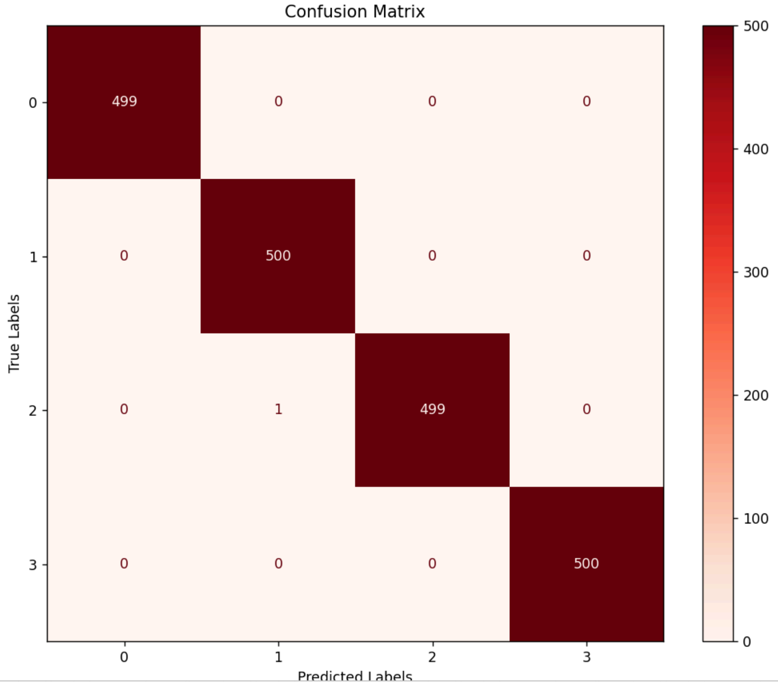
Independent test: DT



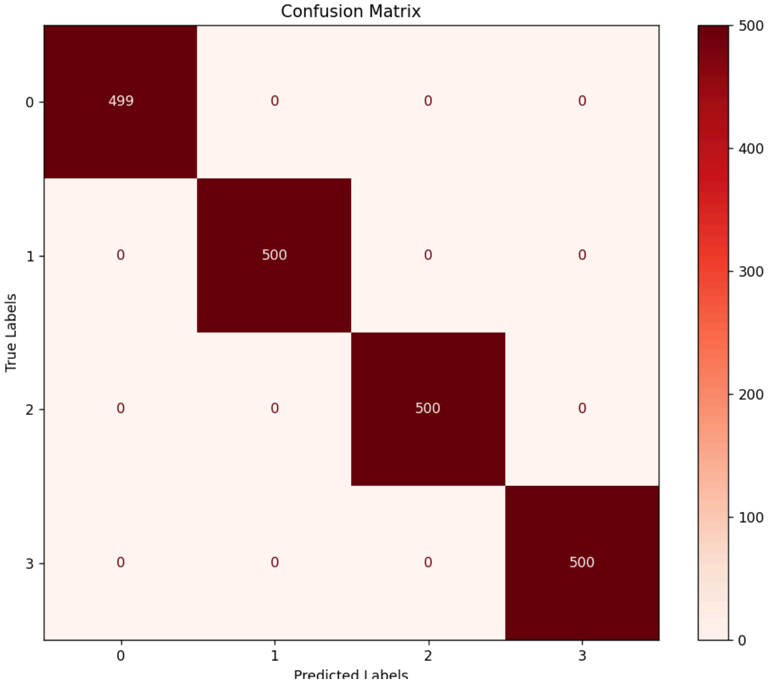
Independent test: KNN



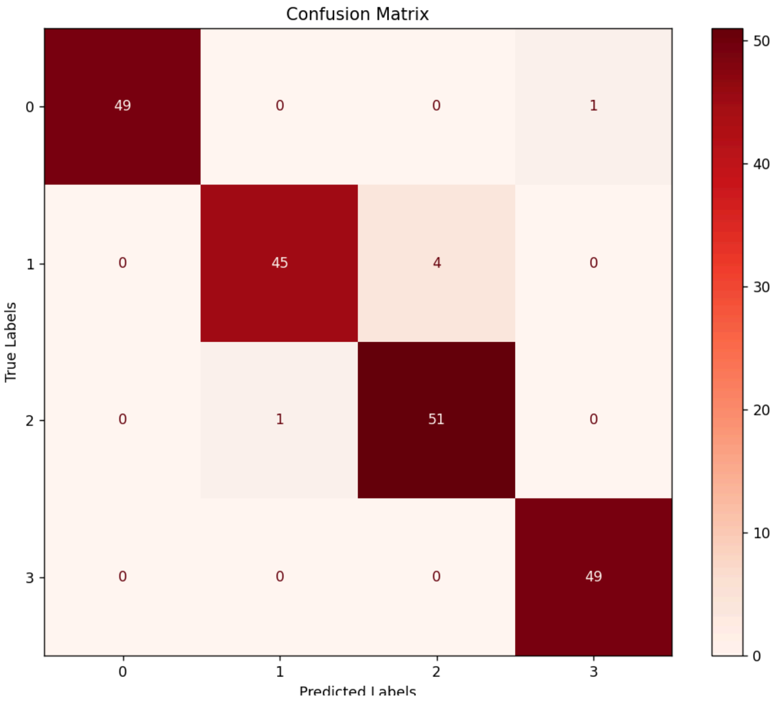
Self-test: DT



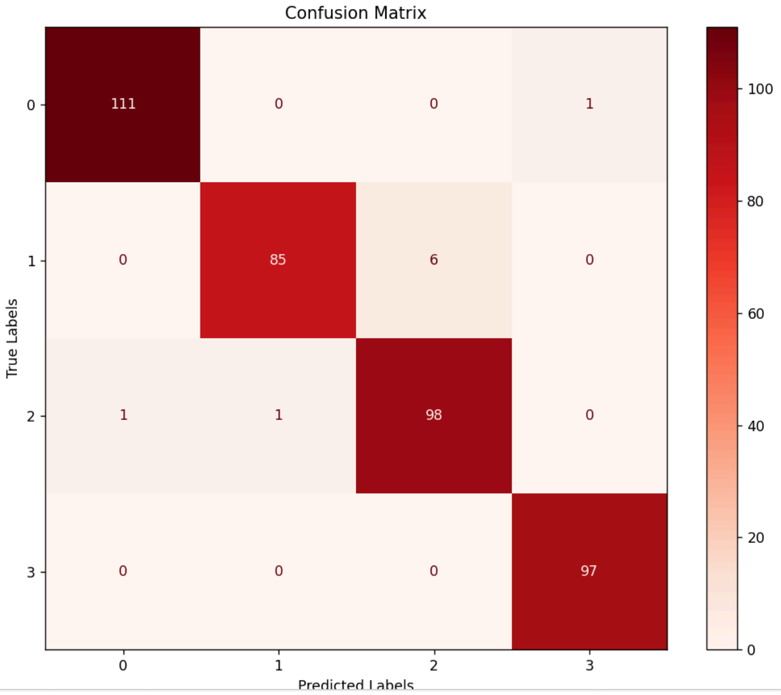
Self-test KNN



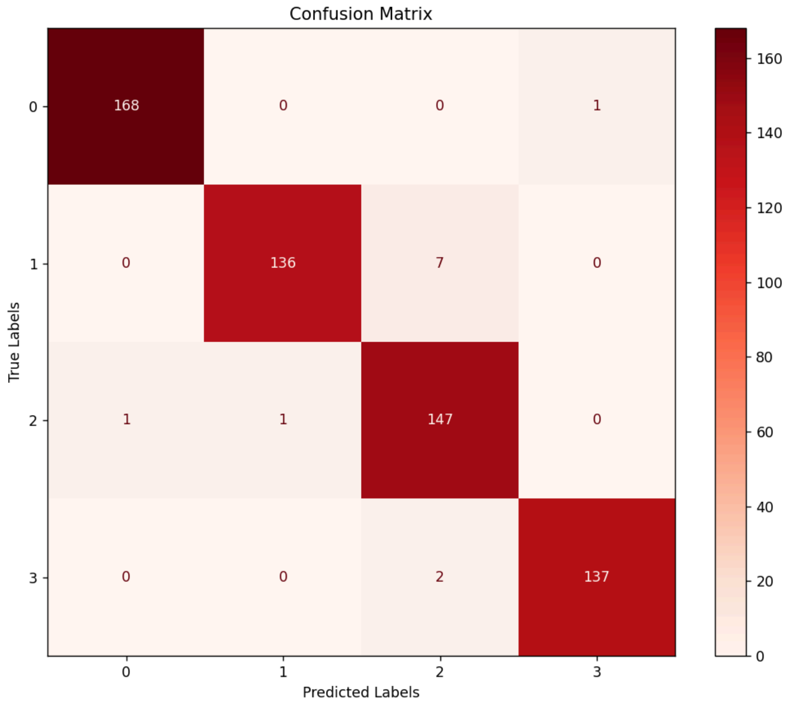
Independent test: KNN 0.1



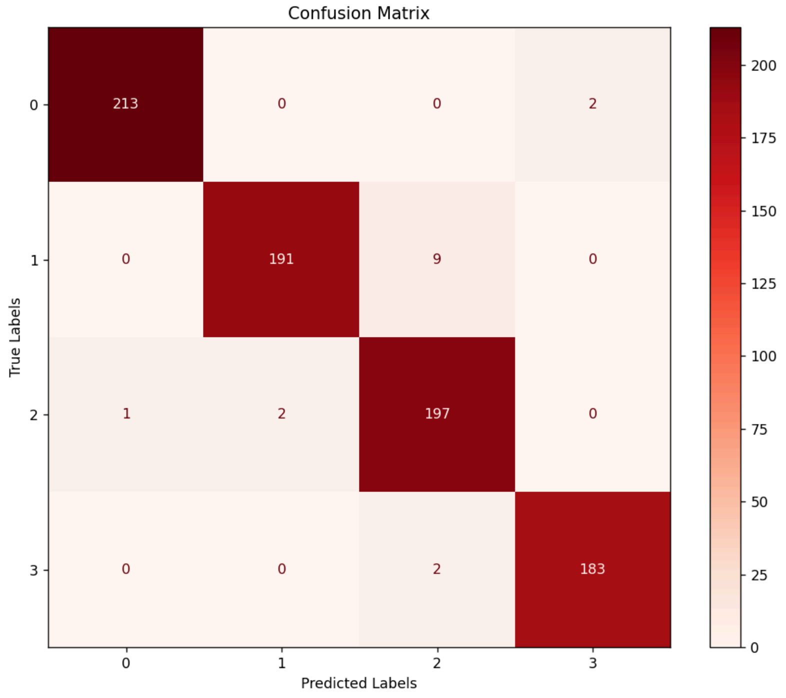
Independent test: KNN 0.2



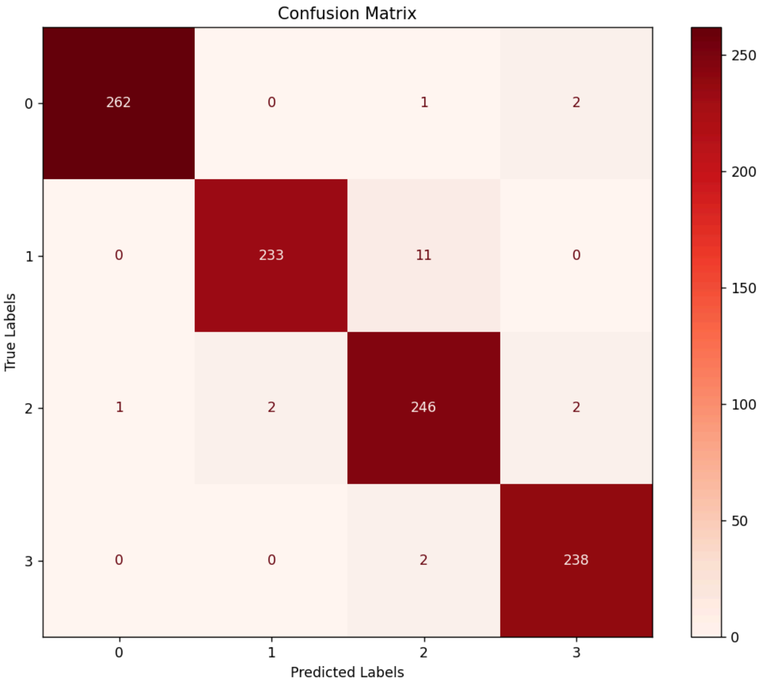
Independent test: KNN 0.3

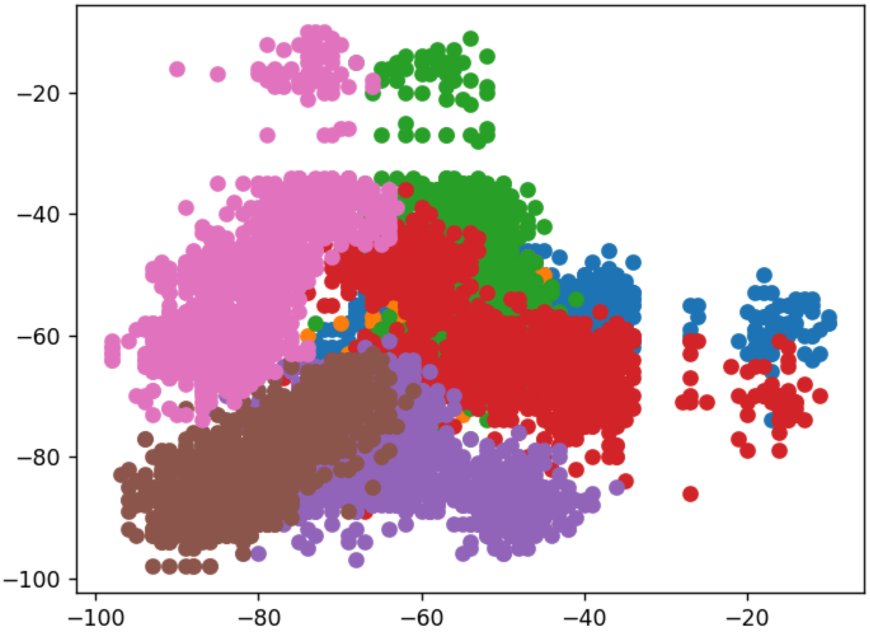


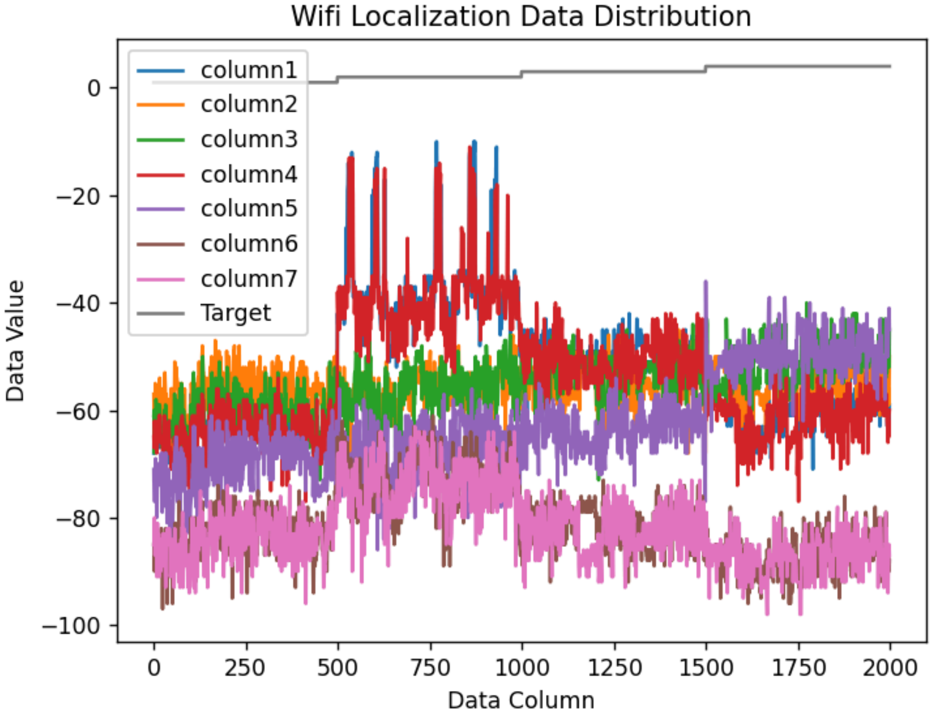
Independent test: KNN 0.4

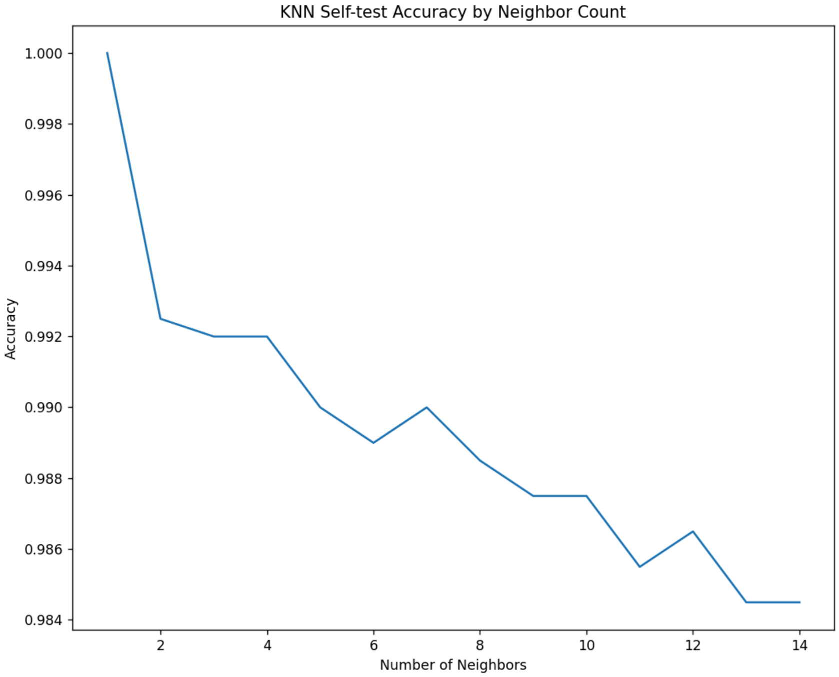


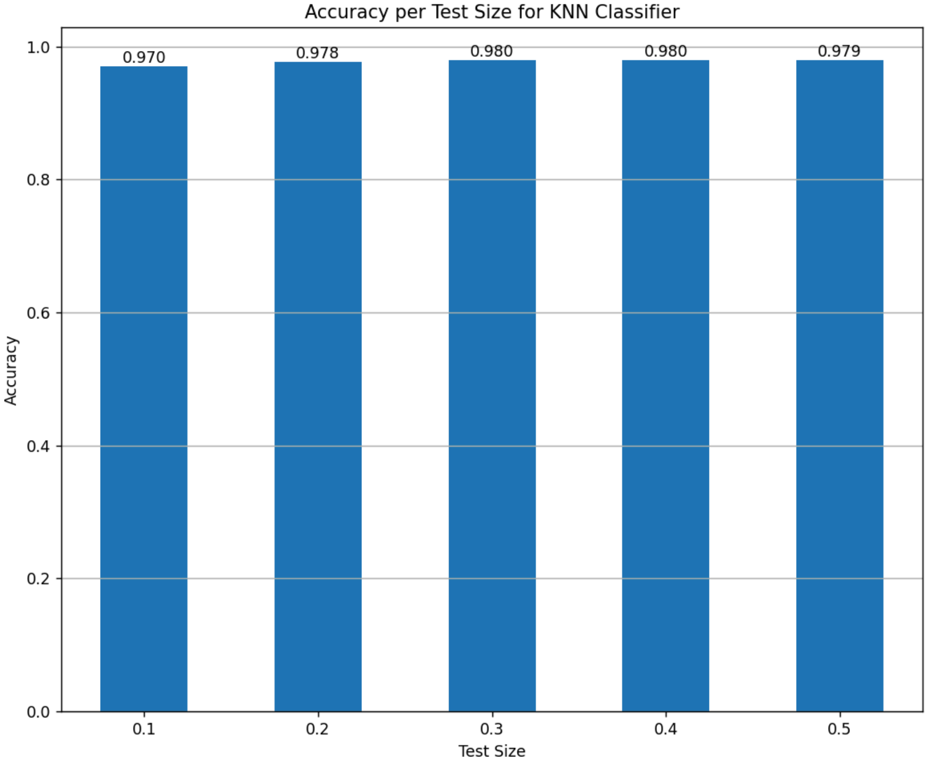
Independent test: KNN 0.5


Confusion Matrix
True Labels
Predicted Labels
0
1
2
3
262
0
1
2
0
233
11
0
1
2
246
2
0
0
2
238
0
50
100
150
200
250

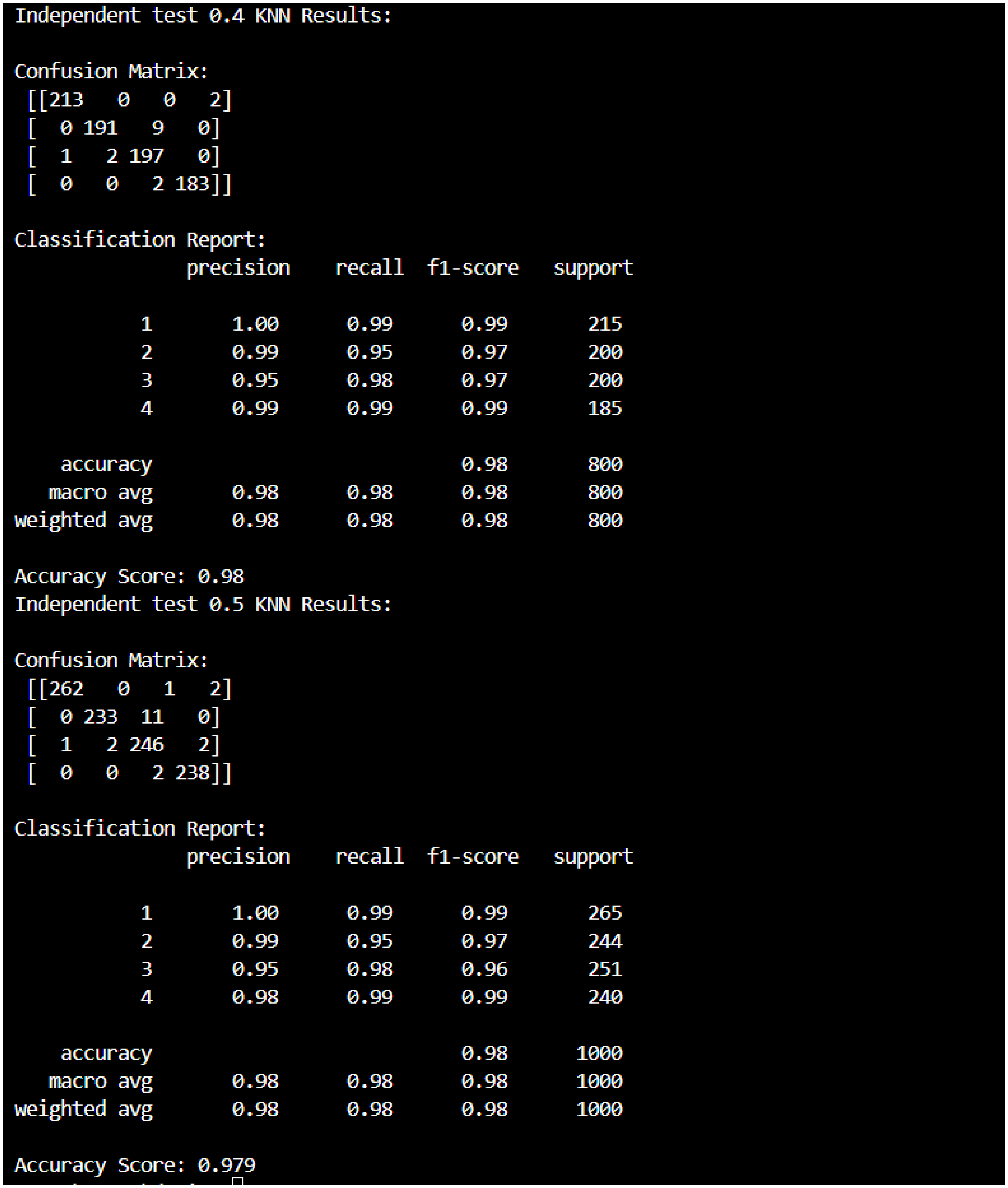

−20
−40
−60
−80
−100
−100
−80
−60
−40
−20


Wifi Localization Data Distribution
column1
column2
column3
column4
column5
column6
column7
Target
0
−20
−40
−60
−80
−100
Data Value
0
250
500
750
1000
1250
1500
1750
2000
Data Column

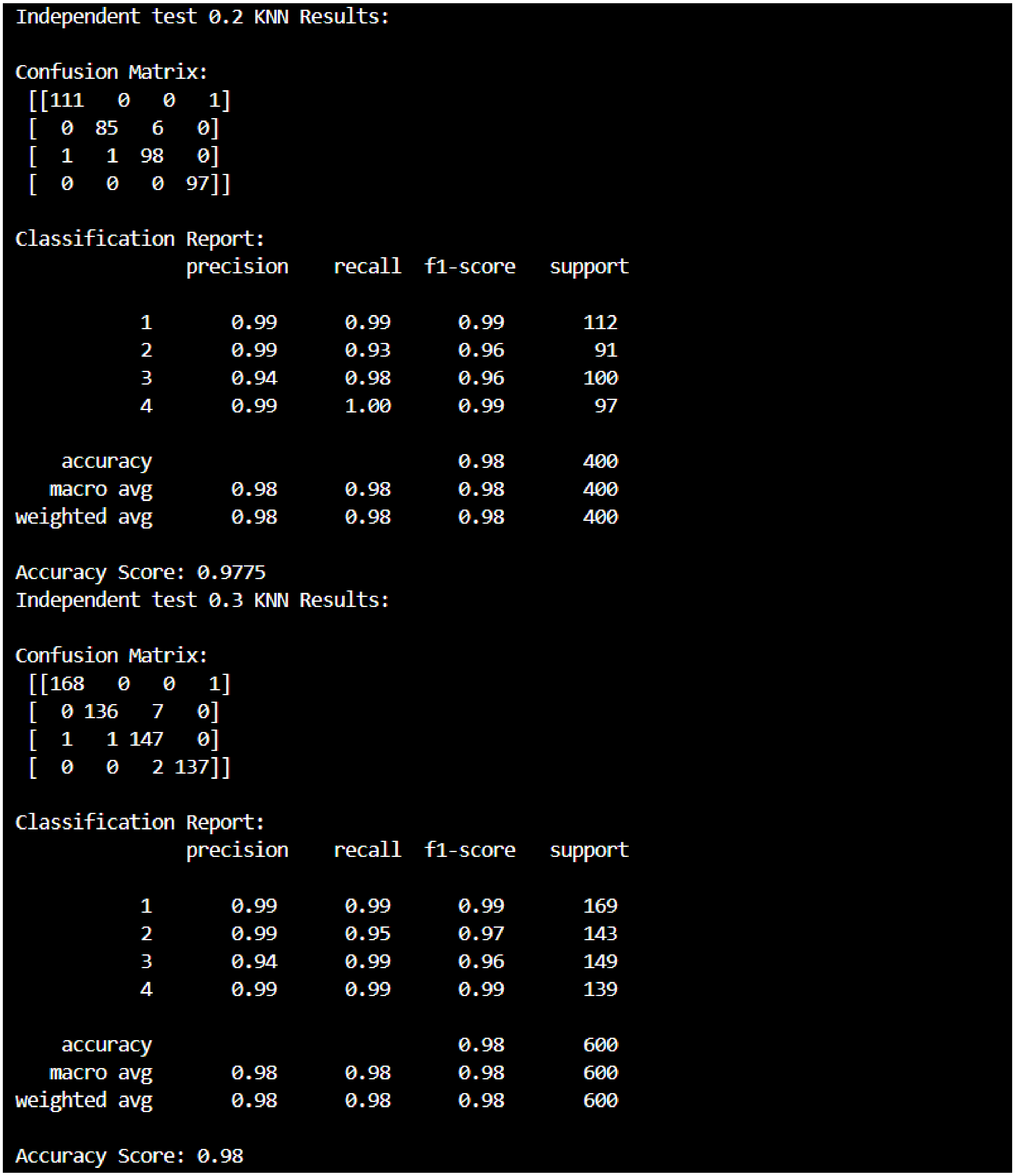

KNN Self-test Accuracy by Neighbor Count
Accuracy
1.000
0.998
0.996
0.994
0.992
0.990
0.988
0.986
0.984
2
4
6
8
10
12
14
Number of Neighbors


Accuracy per Test Size for KNN Classifier
Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
0.970
0.978
0.980
0.980
0.979
0.1
0.2
0.3
0.4
0.5
Test Size

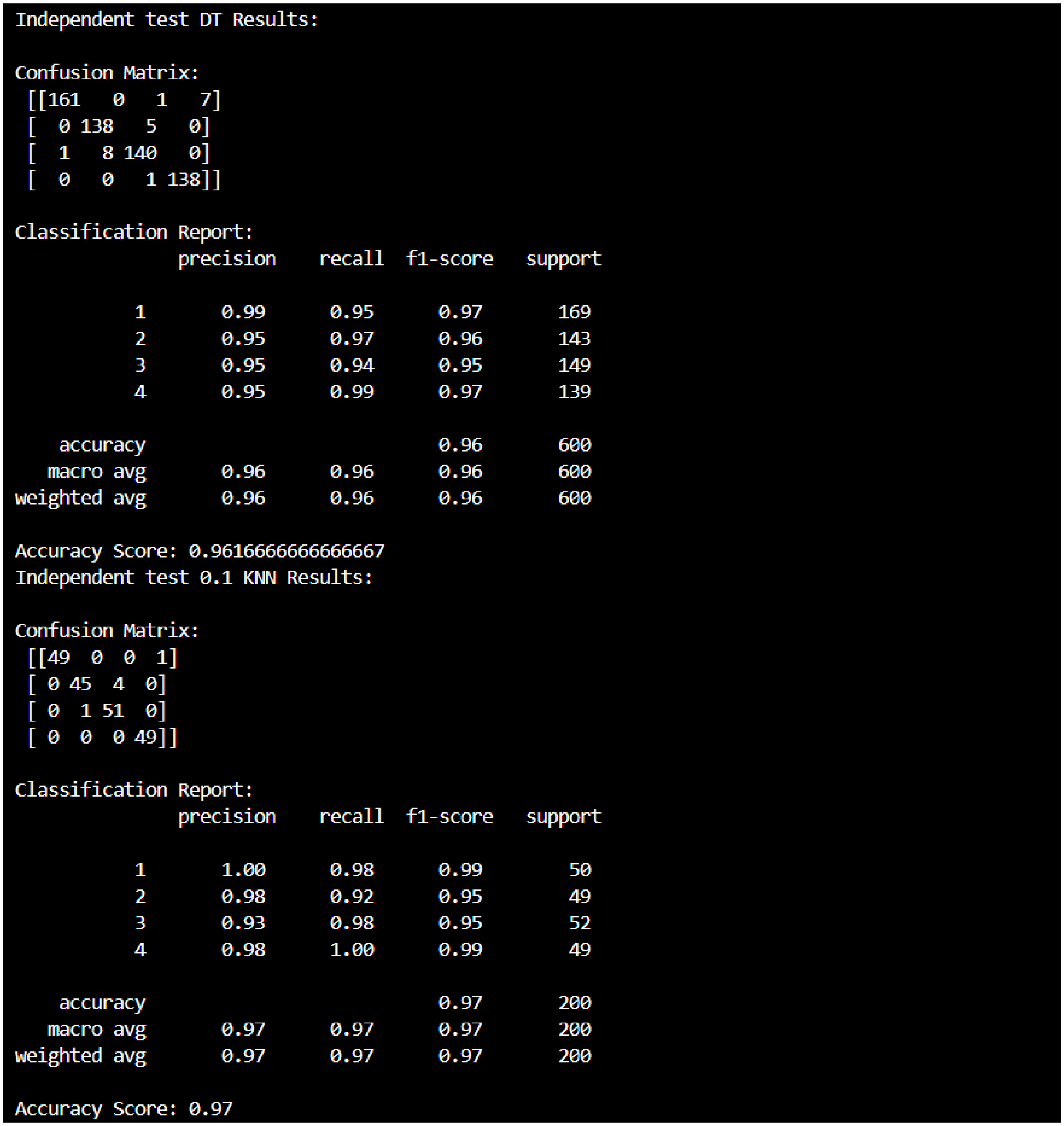
Results:



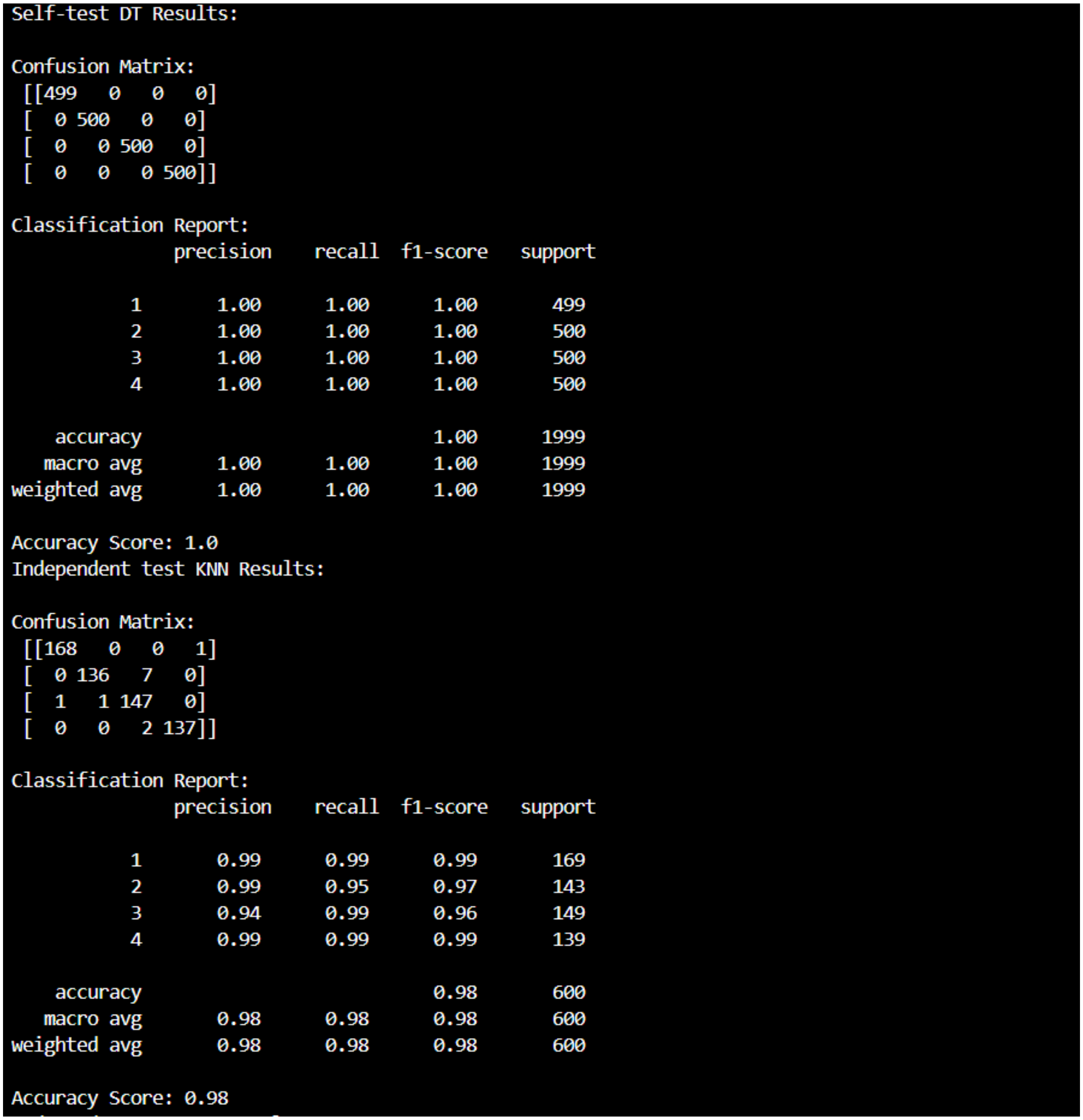
```
Independent test 0.4 KNN Results:

Confusion Matrix:
 [[213   0   0   2]
 [  0 191   9   0]
 [  1   2 197   0]
 [  0   0   2 183]]

Classification Report:
               precision    recall  f1-score   support

           1       1.00      0.99      0.99       215
           2       0.99      0.95      0.97       200
           3       0.95      0.98      0.97       200
           4       0.99      0.99      0.99       185

    accuracy                           0.98       800
   macro avg       0.98      0.98      0.98       800
weighted avg       0.98      0.98      0.98       800

Accuracy Score: 0.98
Independent test 0.5 KNN Results:

Confusion Matrix:
 [[262   0   1   2]
 [  0 233  11   0]
 [  1   2 246   2]
 [  0   0   2 238]]

Classification Report:
               precision    recall  f1-score   support

           1       1.00      0.99      0.99       265
           2       0.99      0.95      0.97       244
           3       0.95      0.98      0.96       251
           4       0.98      0.99      0.99       240

    accuracy                           0.98      1000
   macro avg       0.98      0.98      0.98      1000
weighted avg       0.98      0.98      0.98      1000

Accuracy Score: 0.979
```



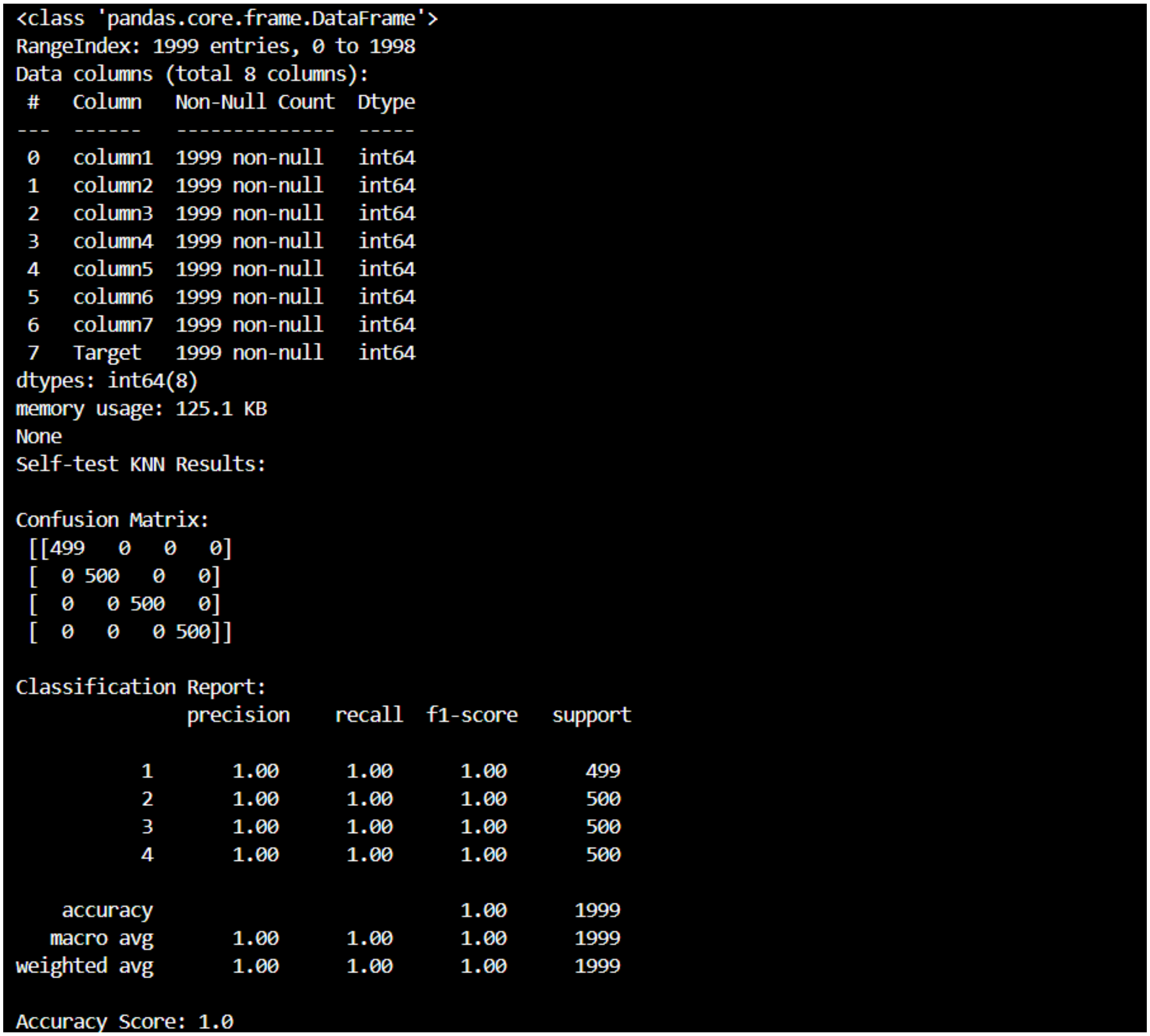
```
Independent test 0.2 KNN Results:

Confusion Matrix:
 [[111   0   0   1]
 [  0  85   6   0]
 [  1   1  98   0]
 [  0   0   0  97]]

Classification Report:
               precision    recall  f1-score   support

           1       0.99      0.99      0.99       112
           2       0.99      0.93      0.96        91
           3       0.94      0.98      0.96       100
           4       0.99      1.00      0.99        97

    accuracy                           0.98       400
   macro avg       0.98      0.98      0.98       400
weighted avg       0.98      0.98      0.98       400

Accuracy Score: 0.9775
Independent test 0.3 KNN Results:

Confusion Matrix:
 [[168   0   0   1]
 [  0 136   7   0]
 [  1   1 147   0]
 [  0   0   2 137]]

Classification Report:
               precision    recall  f1-score   support

           1       0.99      0.99      0.99       169
           2       0.99      0.95      0.97       143
           3       0.94      0.99      0.96       149
           4       0.99      0.99      0.99       139

    accuracy                           0.98       600
   macro avg       0.98      0.98      0.98       600
weighted avg       0.98      0.98      0.98       600

Accuracy Score: 0.98
```



```
Independent test DT Results:

Confusion Matrix:
 [[161   0   1   7]
 [  0 138   5   0]
 [  1   8 140   0]
 [  0   0   1 138]]

Classification Report:
               precision    recall  f1-score   support

           1       0.99      0.95      0.97       169
           2       0.95      0.97      0.96       143
           3       0.95      0.94      0.95       149
           4       0.95      0.99      0.97       139

    accuracy                           0.96       600
   macro avg       0.96      0.96      0.96       600
weighted avg       0.96      0.96      0.96       600

Accuracy Score: 0.9616666666666667
Independent test 0.1 KNN Results:

Confusion Matrix:
 [[49  0  0  1]
 [ 0 45  4  0]
 [ 0  1 51  0]
 [ 0  0  0 49]]

Classification Report:
               precision    recall  f1-score   support

           1       1.00      0.98      0.99        50
           2       0.98      0.92      0.95        49
           3       0.93      0.98      0.95        52
           4       0.98      1.00      0.99        49

    accuracy                           0.97       200
   macro avg       0.97      0.97      0.97       200
weighted avg       0.97      0.97      0.97       200

Accuracy Score: 0.97
```



```
Self-test DT Results:

Confusion Matrix:
 [[499   0   0   0]
 [  0 500   0   0]
 [  0   0 500   0]
 [  0   0   0 500]]

Classification Report:
               precision    recall  f1-score   support

           1       1.00      1.00      1.00       499
           2       1.00      1.00      1.00       500
           3       1.00      1.00      1.00       500
           4       1.00      1.00      1.00       500

    accuracy                           1.00      1999
   macro avg       1.00      1.00      1.00      1999
weighted avg       1.00      1.00      1.00      1999

Accuracy Score: 1.0
Independent test KNN Results:

Confusion Matrix:
 [[168   0   0   1]
 [  0 136   7   0]
 [  1   1 147   0]
 [  0   0   2 137]]

Classification Report:
               precision    recall  f1-score   support

           1       0.99      0.99      0.99       169
           2       0.99      0.95      0.97       143
           3       0.94      0.99      0.96       149
           4       0.99      0.99      0.99       139

    accuracy                           0.98       600
   macro avg       0.98      0.98      0.98       600
weighted avg       0.98      0.98      0.98       600

Accuracy Score: 0.98
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1999 entries, 0 to 1998
Data columns (total 8 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   column1  1999 non-null   int64
 1   column2  1999 non-null   int64
 2   column3  1999 non-null   int64
 3   column4  1999 non-null   int64
 4   column5  1999 non-null   int64
 5   column6  1999 non-null   int64
 6   column7  1999 non-null   int64
 7   Target   1999 non-null   int64
dtypes: int64(8)
memory usage: 125.1 KB
None
Self-test KNN Results:

Confusion Matrix:
 [[499   0   0   0]
 [  0 500   0   0]
 [  0   0 500   0]
 [  0   0   0 500]]

Classification Report:
               precision    recall  f1-score   support

           1       1.00      1.00      1.00       499
           2       1.00      1.00      1.00       500
           3       1.00      1.00      1.00       500
           4       1.00      1.00      1.00       500

    accuracy                           1.00      1999
   macro avg       1.00      1.00      1.00      1999
weighted avg       1.00      1.00      1.00      1999

Accuracy Score: 1.0
```