Lab 1- k nearest neighbors

Nguyen Thanh Phat
M01- B2005853

Table of Contents

1. Classifying the Test Set with kNN.....1
    1. Source Code:.....1
    2. Result :.....3
2. Implement kNN from scratch in Python.....3
    1. The Implementation:.....4
    2. The source code:.....5
    3. The Result :.....7
        1. iris dataset.....7
        2. Optics Dataset.....8
        3. Letter dataset.....9
        4. Faces dataset.....14
        5. Fp107 – the new datasets.....15
3. Proof of Cover-Hart's theorem:.....16

1. Classifying the Test Set with kNN

- Given a dataset as follows:

X1	X2	Class
0.376000	0.488000	0
0.312000	0.544000	0
0.298000	0.624000	0
0.394000	0.600000	0
0.506000	0.512000	0
0.488000	0.334000	1
0.478000	0.398000	1
0.606000	0.366000	1
0.428000	0.294000	1
0.542000	0.252000	1

- Classifying the testset with 1NN, 3NN:

1. Source Code:

```
import math
import numpy as np
import csv

# Load the training data
with open('training_data.csv', 'r') as csvfile:
    reader = csv.reader(csvfile)
    next(reader)
    train_data = np.array(list(reader))
```

```
# Load the test data
with open('test_data.csv', 'r') as csvfile:
    reader = csv.reader(csvfile)
    next(reader)
    test_data = np.array(list(reader))

# Print the loaded data
print("Training data features:\n", train_features)
print("Training data labels:\n", train_labels)
print("Test data features:\n", test_features)

k_list = [1, 3]
header = ['X1', 'X2', 'Class']

def euclidean_distance(point1, point2):
  squared_distances = 0
  for i in range(len(point1)):
    squared_distances += (float(point1[i]) - float(point2[i]))**2
  return math.sqrt(squared_distances)



def get_neighbors(train_features, train_labels, test_point, k):
  distances = []
  for i in range(len(train_features)):
    distances.append((euclidean_distance(train_features[i],
test_point), train_labels[i]))
  distances.sort(key=lambda x: x[0])
  return [label for distance, label in distances[:k]]


def knn_predict(train_features, train_labels, test_features, k):
  predictions = []
  for test_point in test_features:
    neighbors = get_neighbors(train_features, train_labels,
test_point, k)
    class_counts = {}
    for label in neighbors:
      if label not in class_counts:
        class_counts[label] = 0
      class_counts[label] += 1
    majority_vote = max(class_counts, key=class_counts.get)
    predictions.append(majority_vote)
  return predictions


for k in k_list:
  predictions = knn_predict(train_features, train_labels,
test_features, k)
  print(f"Predictions for k = {k}:\n", predictions)
```

2. Result :

```
Training data features:
 [['0.376' '0.488']
 ['0.312' '0.544']
 ['0.298' '0.624']
 ['0.394' '0.6']
 ['0.506' '0.512']
 ['0.488' '0.334']
 ['0.478' '0.398']
 ['0.606' '0.366']
 ['0.428' '0.294']
 ['0.542' '0.252']]
Training data labels:
 ['0' '0' '0' '0' '0' '1' '1' '1' '1' '1']
Test data features:
 [['0.55' '0.364']
 ['0.558' '0.47']
 ['0.456' '0.45']
 ['0.45' '0.57']]
Predictions for k = 1:
 ['1', '0', '1', '0']
Predictions for k = 3:
 ['1', '1', '0', '0']
```

Therefore,
K =1

X1	X2	Class
0.550000	0.364000	1
0.558000	0.470000	0
0.456000	0.450000	1
0.450000	0.570000	0

K= 3

X1	X2	Class
0.550000	0.364000	1
0.558000	0.470000	1
0.456000	0.450000	0
0.450000	0.570000	0

2. Implement kNN from scratch in Python.

The program requires 3 parameters:

- file name of trainset
- file name of testset
- number of nearest neighbors (k)

Dataset with m examples, n dimensions (attribute), c classes (0, 1, …, c-1), is in the format: val_i1_a1 val_i1_a2 … val_i1_an class_i1
val_i2_a1 val_i2_a2 … val_i2_an class_i2

…
val_im_a1 val_im_a2 … val_im_an class_im
The program reports the classification results (accuracy, confusion matrix) with different trials k=1, 3, etc for 5 datasets:

- Iris (.trn: trainset, .tst: testset)
- Optics (.trn: trainset, .tst: testset)
- Letter (.trn: trainset, .tst: testset)
- Face (.trn: trainset, .tst: testset)
- Fp (.trn: trainset, .tst: testset)

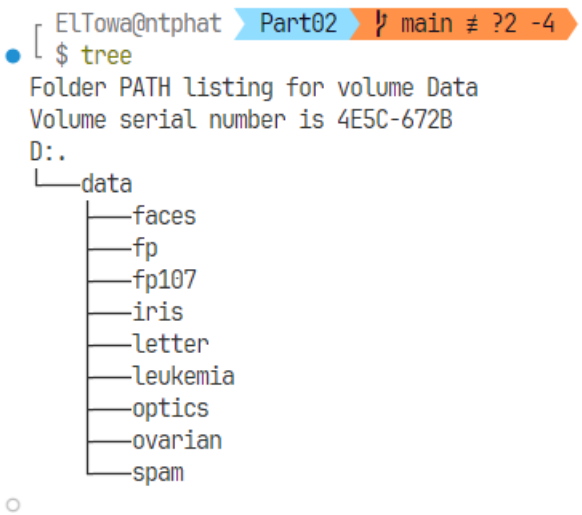
1. The Implementation:

Datasets: http://www.cit.ctu.edu.vn/~dtnghi/ml/data.tar.gz
I used the old data from the website and the new data set fp107 which will replace Iris. So we have:

- Fp107 (.trn: trainset, .tst: testset)
- Optics (.trn: trainset, .tst: testset)
- Letter (.trn: trainset, .tst: testset)
- Face (.trn: trainset, .tst: testset)
- Fp (.trn: trainset, .tst: testset)

I used K values: 1, 5, 7

Here the snapshot of my directory:



2. The source code:

```
import time
import numpy as np

# Function to calculate Euclidean distance between two sets of
points
def euclidean_distance(test, train):
    return np.sqrt(np.sum(np.square(test - train), axis=1))

# Function to perform k-nearest neighbors classification
def kNN(trainset_values, trainset_labels, testset_values, k,
metric=euclidean_distance):
    testset_predictions = []
    for test_value in testset_values:
        distances = metric(test_value, trainset_values)
        indices = np.argsort(distances)[:k]
        neighbors = trainset_labels[indices]

testset_predictions.append(np.argmax(np.bincount(neighbors)))
    return testset_predictions

# Function to create a confusion matrix from predictions and true
labels
def create_confusion_matrix(prediction, original):
    labels = np.unique(original)
    amount = len(labels)
    confusion_matrix = np.zeros((amount, amount))
    # Fill in the confusion matrix with the number of predictions
for each pair of true and predicted labels
    for i in range(amount):
        for j in range(amount):
```

```
            confusion_matrix[i, j] = np.sum((original == labels[i])
& (prediction == labels[j]))
    return confusion_matrix.astype(int)

# Function to calculate classification accuracy
def calculate_accuracy(testset_labels, testset_predictions):
    correct = 0
    for i in range(len(testset_labels)):
        if (testset_labels[i] == testset_predictions[i]):
            correct += 1
    accuracy = correct / len(testset_labels)
    return accuracy

# List of datasets containing training and testing data
list_datasets = [
    {'train_file': 'data//fp107//fp107.trn', 'test_file':
'data//fp107//fp107.tst'},
    {'train_file': 'data//optics//optics.trn', 'test_file':
'data//optics//optics.tst'},
    {'train_file': 'data//letter//letter.trn', 'test_file':
'data//letter//letter.tst'},
    {'train_file': 'data//faces//faces.trn', 'test_file':
'data//faces//faces.tst'},
    {'train_file': 'data//fp//fp.trn', 'test_file':
'data//fp//fp.tst'}
]


# List of k values to try
k_list = [1, 5, 7]


output_file = "kNN_results.txt"

# Open the file for writing
with open(output_file, 'w') as f_out:
    # Loop over all datasets and k values
    for dataset in list_datasets:
        try:
            # Load training and testing data from files
            train_data = np.loadtxt(dataset['train_file'],
delimiter=',', dtype=float)
            test_data = np.loadtxt(dataset['test_file'],
delimiter=',', dtype=float)
        except:
            train_data = np.loadtxt(dataset['train_file'],
delimiter=' ', dtype=float)
            test_data = np.loadtxt(dataset['test_file'],
delimiter=' ', dtype=float)

        # Split training and testing data into values and labels
        trainset_values, trainset_labels = train_data[:, :-1],
train_data[:, -1].astype(int)
```

```
        testset_values, testset_labels = test_data[:, :-1],
test_data[:, -1].astype(int)

        for k_value in k_list:
            start_time = time.time()

            # Call kNN function and store the predictions
            predictions = kNN(trainset_values, trainset_labels,
testset_values, k_value)

            # Calculate accuracy and create confusion matrix
            accuracy = calculate_accuracy(testset_labels,
predictions)
            confusion_matrix = create_confusion_matrix(predictions,
testset_labels)

            end_time = time.time()
            elapsed_time = end_time - start_time

            # Print the results to the console
            print(f"\nDataset: {dataset}\n k = {k_value}")
            print(f"Accuracy: {accuracy}")
            print(f'Confusion Matrix:{len(confusion_matrix)}\n')
            print(f"Elapsed Time: {elapsed_time:.4f} seconds")

            # Write the results to the output file
            f_out.write(f"\nDataset: {dataset}\n k = {k_value}\n")
            f_out.write(f"Accuracy: {accuracy}\n")
            f_out.write(f'Confusion Matrix:{len(confusion_matrix)}\
n')
            f_out.write(f"{confusion_matrix}")
            f_out.write(f"\nElapsed Time: {elapsed_time:.4f}
seconds\n")

print(f"\nResults have been written to {output_file}")
```

3. The Result :

The result written in the file : **kNN_results.txt**

1. iris dataset

```
Dataset: {'train_file': 'data//iris//iris.trn', 'test_file':
'data//iris//iris.tst'}
 k = 1
Accuracy: 0.94
Confusion Matrix: 3
[[17  0  0]
 [ 0 15  0]
 [ 0  3 15]]
Elapsed Time: 0.0011 seconds
```

```
Dataset: {'train_file': 'data//iris//iris.trn', 'test_file':
'data//iris//iris.tst'}
 k = 5
Accuracy: 0.94
Confusion Matrix: 3
[[17  0  0]
 [ 0 15  0]
 [ 0  3 15]]
Elapsed Time: 0.0011 seconds

Dataset: {'train_file': 'data//iris//iris.trn', 'test_file':
'data//iris//iris.tst'}
 k = 7
Accuracy: 0.94
Confusion Matrix: 3
[[17  0  0]
 [ 0 15  0]
 [ 0  3 15]]
Elapsed Time: 0.0010 seconds
```

2. Optics Dataset

```
Dataset: {'train_file': 'data//optics//optics.trn', 'test_file':
'data//optics//optics.tst'}
 k = 1
Accuracy: 0.9799666110183639
Confusion Matrix: 10
[[178   0   0   0   0   0   0   0   0   0]
 [  0 181   0   0   0   0   0   0   1   0]
 [  0   2 175   0   0   0   0   0   0   0]
 [  0   0   0 179   0   0   0   2   0   2]
 [  0   2   0   0 178   0   0   0   1   0]
 [  0   0   0   0   1 179   0   0   0   2]
 [  0   0   0   0   0   0 181   0   0   0]
 [  0   0   0   0   0   0   0 177   0   2]
 [  0   8   0   1   0   0   0   0 164   1]
 [  0   0   0   3   3   2   0   0   3 169]]
Elapsed Time: 2.6609 seconds

Dataset: {'train_file': 'data//optics//optics.trn', 'test_file':
'data//optics//optics.tst'}
 k = 5
Accuracy: 0.9788536449638287
Confusion Matrix: 10
[[178   0   0   0   0   0   0   0   0   0]
 [  0 181   0   0   0   0   1   0   0   0]
 [  0   3 174   0   0   0   0   0   0   0]
 [  0   1   1 178   0   1   0   1   1   0]
 [  0   1   0   0 179   0   0   0   1   0]
 [  0   0   0   0   1 180   0   0   0   1]
 [  0   0   0   0   0   1 180   0   0   0]
 [  0   0   0   0   0   0   0 173   1   5]
```

```
 [  0   8   0   2   0   1   0   0 162   1]
 [  0   0   0   2   1   1   0   0   2 174]]
Elapsed Time: 2.3126 seconds

Dataset: {'train_file': 'data//optics//optics.trn', 'test_file':
'data//optics//optics.tst'}
 k = 7
Accuracy: 0.9766277128547579
Confusion Matrix: 10
[[178   0   0   0   0   0   0   0   0   0]
 [  0 180   0   0   0   0   1   0   1   0]
 [  0   3 174   0   0   0   0   0   0   0]
 [  0   1   0 178   0   1   0   1   2   0]
 [  0   2   0   0 178   0   0   0   1   0]
 [  0   0   0   0   1 179   0   0   0   2]
 [  0   0   0   0   0   1 180   0   0   0]
 [  0   0   0   0   0   0   0 175   1   3]
 [  0  10   0   2   0   0   0   0 160   2]
 [  0   1   0   2   0   1   0   0   3 173]]
Elapsed Time: 2.3583 seconds
```

3. *Letter dataset*

```
Dataset: {'train_file': 'data//letter//letter.trn', 'test_file':
'data//letter//letter.tst'}
 k = 1
Accuracy: 0.953045304530453
Confusion Matrix: 26
[[271   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0
    0   0   1   0   0   0   0   0]
 [  0 222   0   0   2   0   0   0   0   1   0   0   0   0   0   0
0   8
    1   0   0   5   0   1   0   0]
 [  0   0 218   0   2   0   1   0   0   0   0   0   0   0   1   0
1   0
    0   0   0   1   2   0   0   0]
 [  0   0   0 265   0   0   1   5   0   0   1   0   0   1   0   0
0   3
    1   0   0   0   0   0   0   0]
 [  0   1   3   0 239   1   4   0   0   0   1   2   0   0   0   1
0   0
    0   0   0   1   0   1   0   8]
 [  0   0   0   0   0 246   0   1   1   0   0   0   0   1   0  18
0   0
    0   1   0   1   0   0   0   0]
 [  0   1   1   0   3   0 249   1   0   0   1   0   0   0   2   0
3   0
    0   0   0   1   1   0   0   0]
 [  0   3   1   5   0   2   2 200   0   0   7   0   0   1   1   0
0   6
    1   0   0   0   0   1   0   0]
```

```
 [  0   0   0   0   0   3   0   0 257   8   0   0   0   0   0   0
0   0
    0   0   0   0   0   1   0   0]
 [  0   0   0   0   0   0   0   1  10 227   0   0   0   0   0   0
0   0
    0   0   1   0   0   0   0   0]
 [  0   1   0   0   0   0   0   7   0   0 225   1   0   0   0   0
0   3
    1   0   0   0   1   4   0   0]
 [  0   0   1   0   0   0   2   2   0   1   0 262   0   0   0   0
1   0
    0   0   0   0   0   1   0   0]
 [  0   0   0   0   0   0   1   0   0   0   0   0 239   1   0   0
0   0
    0   0   0   2   2   0   0   0]
 [  1   1   0   0   0   0   0   1   0   1   0   0   2 256   3   0
0   2
    0   0   0   4   1   0   0   0]
 [  0   0   0   1   0   0   0   0   0   0   0   0   0   0 231   0
4   0
    0   0   0   0   1   0   0   0]
 [  0   1   0   0   0  16   0   0   0   0   0   0   0   0   0 246
1   2
    0   0   0   0   0   0   0   0]
 [  0   0   0   2   0   0   1   0   0   0   0   0   0   0   6   0
270   0
    0   0   0   0   0   0   0   1]
 [  0   4   0   0   0   1   0   6   0   0   5   1   0   1   0   0
0 252
    1   0   0   0   0   0   0   0]
 [  0   2   0   2   1   0   0   0   0   0   0   1   0   0   0   0
1   0
  255   0   1   0   0   0   0   1]
 [  0   0   1   1   0   2   0   0   0   1   1   0   0   1   0   0
0   0
    0 234   0   0   0   0   3   0]
 [  0   1   0   0   0   0   0   0   0   0   0   0   1   0   1   0
0   0
    0   0 270   1   0   0   0   0]
 [  0   2   0   0   0   0   0   0   0   0   0   0   0   0   1   0
0   0
    0   0   0 234   1   0   0   0]
 [  0   0   0   0   0   0   0   1   0   0   0   0   1   1   0   0
0   0
    0   0   0   0 238   0   0   0]
 [  0   1   0   1   1   0   0   0   0   0   7   0   0   0   0   0
0   0
    0   2   0   0   0 249   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
2   0
    0   2   1   1   0   1 245   0]
 [  0   0   0   0   2   0   0   0   0   1   0   0   0   0   0   0
3   0
```

```
    0   0   0   0   0   0   0 253]]
Elapsed Time: 11.4303 seconds

Dataset: {'train_file': 'data//letter//letter.trn', 'test_file':
'data//letter//letter.tst'}
 k = 5
Accuracy: 0.941944194419442
Confusion Matrix: 26
[[272   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0
    0   0   0   0   0   0   1   0]
 [  0 229   0   1   1   0   0   0   0   0   0   0   0   0   0   0
0   6
    1   0   0   2   0   0   0   0]
 [  0   0 216   0   3   0   2   0   0   0   0   0   0   1   2   0
1   0
    0   0   0   0   1   0   0   0]
 [  0   3   0 264   0   0   0   7   0   0   1   0   0   1   0   0
0   1
    0   0   0   0   0   0   0   0]
 [  0   2   3   0 241   2   3   0   0   0   3   0   0   0   0   0
0   0
    0   0   0   0   0   0   0   8]
 [  0   0   0   0   2 245   0   3   2   1   0   0   0   1   0   9
0   0
    0   5   0   1   0   0   0   0]
 [  0   4   2   2   6   1 241   2   0   0   0   0   1   0   2   0
1   0
    0   0   0   1   0   0   0   0]
 [  0   6   0   5   2   0   2 201   0   0   6   1   0   0   1   0
0   4
    0   0   0   0   0   1   1   0]
 [  0   1   0   0   0   7   0   0 249  10   0   0   0   1   0   0
0   0
    0   0   0   0   0   1   0   0]
 [  0   0   0   0   1   0   0   0  12 222   0   0   0   0   1   0
2   0
    0   0   1   0   0   0   0   0]
 [  0   1   1   1   2   1   1  11   0   0 218   0   0   0   0   0
0   3
    0   0   0   0   0   4   0   0]
 [  0   1   0   0   0   0   2   2   0   2   0 260   0   0   0   0
1   2
    0   0   0   0   0   0   0   0]
 [  0   3   0   0   0   0   0   0   0   0   0   0 238   1   0   0
0   0
    0   0   0   2   1   0   0   0]
 [  0   2   0   1   0   0   0   5   0   0   0   1   3 249   1   0
1   5
    0   0   0   4   0   0   0   0]
 [  0   1   1   4   0   0   0   0   0   0   0   0   0   1 225   0
3   0
    0   0   0   1   1   0   0   0]
```

```
 [  0   2   0   2   1  18   0   3   0   0   0   0   0   0   0 239
0   1
    0   0   0   0   0   0   0   0]
 [  0   1   0   1   0   0   2   0   0   0   0   0   0   0   8   0
266   0
    0   0   0   0   0   0   2   0]
 [  0   5   0   2   0   2   0   6   0   0   6   0   0   3   0   0
0 246
    0   0   0   1   0   0   0   0]
 [  0   3   0   2   6   1   0   2   0   0   0   0   0   0   0   0
1   2
  244   0   1   0   0   1   0   1]
 [  0   1   0   2   0   1   0   0   0   0   0   0   0   0   0   0
1   0
    0 233   0   0   0   2   4   0]
 [  0   0   0   0   0   0   0   2   0   0   1   0   0   0   0   0
0   0
    0   0 271   0   0   0   0   0]
 [  0   5   1   0   0   0   0   0   0   0   0   0   0   0   1   0
0   0
    0   0   1 228   2   0   0   0]
 [  0   0   0   0   0   0   0   1   0   0   0   0   3   0   1   0
0   0
    0   0   0   0 236   0   0   0]
 [  2   1   0   1   1   0   0   0   0   0   6   0   0   0   0   0
0   0
    0   0   0   0   0 249   0   1]
 [  0   0   0   2   0   0   0   1   0   0   0   0   0   0   0   1
0   0
    0   1   1   1   0   1 244   0]
 [  0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0
4   0
    0   1   0   0   0   0   0 253]]
Elapsed Time: 13.4287 seconds

Dataset: {'train_file': 'data//letter//letter.trn', 'test_file':
'data//letter//letter.tst'}
 k = 7
Accuracy: 0.9402940294029403
Confusion Matrix: 26
[[271   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   1
    0   0   0   0   0   0   1   0]
 [  0 234   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   3
    1   0   0   2   0   0   0   0]
 [  0   0 217   0   2   0   3   0   0   0   0   0   1   0   1   0
1   0
    0   0   0   0   1   0   0   0]
 [  0   1   0 267   0   0   0   5   0   0   0   0   0   1   0   0
0   2
    1   0   0   0   0   0   0   0]
 [  0   1   6   0 241   1   2   0   0   0   2   0   0   0   0   0
```

```
0   0
    0   0   0   0   0   0   0   9]
 [  0   0   0   0   0 249   0   3   1   1   0   0   0   1   0   7
0   0
    0   6   0   1   0   0   0   0]
 [  1   4   1   2   4   1 242   2   0   0   0   0   0   0   2   0
2   0
    0   0   0   1   0   1   0   0]
 [  0   5   0   6   4   0   3 199   0   0   5   1   0   0   0   0
0   5
    0   0   0   0   0   1   0   1]
 [  0   1   0   2   0   8   0   0 245  11   0   0   0   0   0   1
0   0
    0   0   0   0   0   1   0   0]
 [  0   0   0   0   1   0   0   0  12 221   0   0   0   0   1   0
2   0
    0   0   1   0   0   1   0   0]
 [  0   1   0   2   2   0   1  13   0   0 215   0   0   0   0   0
0   5
    0   0   0   0   0   4   0   0]
 [  0   0   1   0   1   0   1   2   0   1   1 261   0   0   0   0
0   2
    0   0   0   0   0   0   0   0]
 [  0   4   0   0   0   0   2   0   0   0   0   0 235   2   0   0
0   0
    0   0   0   1   1   0   0   0]
 [  0   2   0   2   0   0   0   3   0   0   0   1   3 246   6   0
0   6
    0   0   0   3   0   0   0   0]
 [  0   1   0   3   0   0   0   0   0   0   0   0   0   1 230   0
1   0
    0   0   1   0   0   0   0   0]
 [  0   1   0   4   1  17   1   4   0   0   0   0   1   0   0 236
0   1
    0   0   0   0   0   0   0   0]
 [  1   0   0   2   0   0   2   0   0   0   0   0   0   0  10   0
264   0
    0   0   0   0   0   0   1   0]
 [  0   7   1   1   0   2   0   5   0   0   5   0   0   0   0   0
0 248
    0   1   0   1   0   0   0   0]
 [  0   3   0   2   3   0   0   2   0   0   0   1   0   0   1   0
1   3
  244   0   1   0   0   2   0   1]
 [  0   1   0   3   0   3   0   0   0   0   0   0   0   0   0   0
1   0
    0 230   0   0   0   2   3   1]
 [  0   0   0   1   0   0   0   2   0   0   1   0   1   0   0   0
0   0
    0   0 268   1   0   0   0   0]
 [  0   3   0   0   0   1   0   0   0   0   0   0   0   0   1   0
0   0
    0   0   0 232   1   0   0   0]
```

```
 [  0   1   0   0   0   0   0   0   0   0   1   0   2   0   1   0
0   0
    0   0   0   0 236   0   0   0]
 [  1   1   0   1   3   0   0   0   0   0   9   1   0   0   0   0
0   0
    0   2   1   0   0 241   0   1]
 [  0   0   0   2   0   0   0   0   0   0   0   0   0   0   0   0
0   0
    0   1   2   0   0   1 246   0]
 [  0   0   0   0   2   0   0   0   0   2   0   0   0   0   0   0
4   0
    1   0   0   0   0   0   0 250]]
Elapsed Time: 18.5220 seconds
```

4. Faces dataset

```
Dataset: {'train_file': 'data//faces//faces.trn', 'test_file':
'data//faces//faces.tst'}
 k = 1
Accuracy: 1.0
Confusion Matrix: 20
[[17  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0 10  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  7  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  6  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0 10  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  8  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  4  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  8  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0 19  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  9  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  8  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0 11  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0 12  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  8  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  5  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 21  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  7  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 10]]
Elapsed Time: 2.3784 seconds

Dataset: {'train_file': 'data//faces//faces.trn', 'test_file':
'data//faces//faces.tst'}
 k = 5
Accuracy: 0.9947916666666666
Confusion Matrix: 20
[[17  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0 10  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  7  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
```

```
 [ 0  0  0  0  6  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0 10  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  8  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  4  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  8  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0 19  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  9  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  8  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0 11  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0 12  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  8  0  0  0  0]
 [ 0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  4  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 21  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  7]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 10]]
Elapsed Time: 2.4773 seconds

Dataset: {'train_file': 'data//faces//faces.trn', 'test_file':
'data//faces//faces.tst'}
 k = 7
Accuracy: 0.984375
Confusion Matrix: 20
[[17  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0 10  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  7  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  6  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0 10  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  8  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  4  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  8  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0 19  0  0  0  0  0  0  0  0  0]
 [ 1  0  1  0  0  0  0  0  0  0  7  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  8  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0 11  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0 12  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  8  0  0  0  0]
 [ 0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  4  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 21  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  7]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 10]]
Elapsed Time: 2.4550 seconds
```

5. Fp107 – the new datasets

Although the entries of the confusion matrix are not explicitly written on file, it is evident from the statement "Confusion Matrix: 105" that the matrix dimensions are consistent at 105x105 for all evaluated cases. This uniformity indicates a balanced multiclass classification setting with 105 distinct classes.

```
Dataset: {'train_file': 'data//fp107//fp107.trn',
```

```
'test_file': 'data//fp107//fp107.tst'}
 k = 1
Accuracy: 0.9812734082397003
Confusion Matrix: 105
[[5 0 0 ... 0 0 0]
 [0 3 0 ... 0 0 0]
 [0 0 3 ... 0 0 0]
 ...
 [0 0 0 ... 5 0 0]
 [0 0 0 ... 0 8 0]
 [0 0 0 ... 0 0 7]]
Elapsed Time: 7.6341 seconds

Dataset: {'train_file': 'data//fp107//fp107.trn',
'test_file': 'data//fp107//fp107.tst'}
 k = 5
Accuracy: 0.9794007490636704
Confusion Matrix: 105
[[5 0 0 ... 0 0 0]
 [0 3 0 ... 0 0 0]
 [0 0 3 ... 0 0 0]
 ...
 [0 0 0 ... 5 0 0]
 [0 0 0 ... 0 8 0]
 [0 0 0 ... 0 0 7]]
Elapsed Time: 7.6603 seconds

Dataset: {'train_file': 'data//fp107//fp107.trn',
'test_file': 'data//fp107//fp107.tst'}
 k = 7
Accuracy: 0.9812734082397003
Confusion Matrix: 105
[[5 0 0 ... 0 0 0]
 [0 3 0 ... 0 0 0]
 [0 0 3 ... 0 0 0]
 ...
 [0 0 0 ... 5 0 0]
 [0 0 0 ... 0 9 0]
 [0 0 0 ... 0 0 7]]
Elapsed Time: 7.6783 seconds
```

3. Proof of Cover-Hart's theorem:

For sufficiently large training set size m, the error rate of the 1NN classifier is less than twice the Bayes error rate.

1. Define the key terms:

- X: The feature space where data points reside.

- **y:** The class label space.
- **P(x, y):** The joint probability distribution of X and Y.
- **P(y|x):** The conditional probability of class y given a data point x.
- **R(c):** The risk (expected error) of a classifier c.
- $c_{1\,NN}(x)$**:** The class predicted by the 1-NN classifier for data point x.
- $c_{1\,NN}(x)$**:** The Bayes error rate, the minimum achievable risk.

2. Consider the Bayes rule:

The Bayes rule minimizes the risk by predicting the class with the highest posterior probability for a given data point:

$$c_{Bayes}(x) = arg\,max_y\, P(y \vee x)$$

The Bayes error rate is then the minimum risk achievable with this rule:

$$\rho_B = min_c\, R(c)$$

3. Analyze the 1-NN classifier:

The 1-NN classifier predicts the class of the nearest neighbor in the training set for a given data point. Let $x_i$be a training point and y_i its class label. The error of the 1-NN classifier on x is:

$$e_{1\,NN}(x) = 1 - \delta\left(c_{1\,NN}(x), y_{\square}(x)\right)$$

where δ is the Kronecker delta function, which is 1 if its arguments are equal and 0 otherwise.

4. Introduce the covering numbers:

The covering number $N(\lambda, X, P)$ represents the minimum number of balls with radius λ needed to cover the entire space X according to the probability distribution P. It measures the complexity of the data distribution.

5. State the main theorem:

Cover and Hart (1967) proved the following theorem:

Theorem: For any probability distribution P on X, any λ > 0, and any c,

$$R(c) \leq 2\,N(\lambda, X, P)\, inf_x\, P\left(e_c(x) > \lambda\right)$$

6. Apply the theorem to the 1-NN classifier:

Setting $\lambda = \rho_B$ and $c = c_{1\,NN}$, we get:

$$R\left(c_{1\,NN}\right) \leq 2\,N\left(\rho_B, X, P\right) inf_x\, P\left(e_{1\,NN}(x) > \rho_B\right)$$

This tells us that the risk of the 1-NN classifier is bounded by twice the covering number at the Bayes error rate multiplied by the probability of the 1-NN classifier making an error greater than the Bayes error rate.

7. Analyze the bound:

As the training set size grows, the covering number typically decreases. This means that the first term in the bound becomes smaller. Additionally, with a large enough training set, the probability of the 1-NN classifier making a significantly larger error than the Bayes error rate approaches 0.

Therefore, for sufficiently large training set size m, the bound on the 1-NN error rate becomes:

$$R\left(c_{1NN}\right)<2\,\rho_B$$

This proves the claim that the error rate of the 1-NN classifier is less than twice the Bayes error rate for sufficiently large training sets.