Lab 1- k nearest neighbors

**Nguyen Thanh Phat**

M01- B2005853

**Table of Contents**

[1. Classifying the Test Set with kNN 1](#__RefHeading___Toc13972_3159053761)

[1. Source Code: 1](#__RefHeading___Toc13974_3159053761)

[2. Result : 3](#__RefHeading___Toc13976_3159053761)

[2. Implement kNN from scratch in Python. 3](#__RefHeading___Toc13980_3159053761)

[1. The Implementation: 4](#__RefHeading___Toc13982_3159053761)

[2. The source code: 5](#__RefHeading___Toc13984_3159053761)

[3. The Result : 7](#__RefHeading___Toc13986_3159053761)

[1. iris dataset 7](#__RefHeading___Toc13988_3159053761)

[2. Optics Dataset 8](#__RefHeading___Toc13990_3159053761)

[3. Letter dataset 9](#__RefHeading___Toc13992_3159053761)

[4. Faces dataset 14](#__RefHeading___Toc13994_3159053761)

[5. Fp107 – the new datasets 15](#__RefHeading___Toc13996_3159053761)

[3. Proof of Cover-Hart’s theorem: 16](#__RefHeading___Toc13998_3159053761)

## 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:

### 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) |

### 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 |

## 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)

### 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:

### 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}") |

### The Result :

The result written in the file : **kNN\_results.txt**

#### 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 |

#### 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 |

#### 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 |

#### 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 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 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 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: 2.4773 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]  [ 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]  [ 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 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 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 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: 2.4550 seconds |

#### 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 |

## 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.
* **:** The class predicted by the 1-NN classifier for data point 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:

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

**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 be a training point and its class label. The error of the 1-NN classifier on x is:

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 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,

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

Setting and , we get:

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:

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.