Assign 1: k nearest neighbors

Student name: Nguyễn Văn Thuần

ID: B2112012

1) Given a dataset as follows:

X1	X2	Class
0.376	0.488	0
0.312	0.544	0
0.298	0.624	0
0.394	0.6	0
0.506	0.512	0
0.488	0.334	1
0.478	0.398	1
0.606	0.366	1
0.428	0.294	1
0.542	0.252	1

Classifying the testset with 1NN, 3NN:

X1	X2	Class
0.55	0.364	?
0.558	0.47	?
0.456	0.45	?
0.45	0.57	?

The solutions:

With 1NN (k= 1):

```
(myenv) D:\learning\hk2\ML\lab\lab1>python bai1.py

With k = 1
Using euclidean distance:
+--------+--------+--------+
| X1     | X2     | Class  |
+========+========+========+
| 0.550 | 0.364 |   1    |
+--------+--------+--------+
| 0.558 | 0.470 |   0    |
+--------+--------+--------+
| 0.456 | 0.450 |   1    |
+--------+--------+--------+
| 0.450 | 0.570 |   0    |
+--------+--------+--------+

With k = 1
Using manhattan distance:
+--------+--------+--------+
| X1     | X2     | Class  |
+========+========+========+
| 0.550 | 0.364 |   1    |
+--------+--------+--------+
| 0.558 | 0.470 |   0    |
+--------+--------+--------+
| 0.456 | 0.450 |   1    |
+--------+--------+--------+
| 0.450 | 0.570 |   0    |
+--------+--------+--------+
```

Result:

X1	X2	Class
0.55	0.364	1
0.558	0.47	0
0.456	0.45	1
0.45	0.57	0

With 3NN (k=3):

```
With k = 3
Using euclidean distance:
+--------+--------+--------+
| X1     | X2     | Class  |
+========+========+========+
| 0.550 | 0.364 |   1    |
+--------+--------+--------+
| 0.558 | 0.470 |   1    |
+--------+--------+--------+
| 0.456 | 0.450 |   0    |
+--------+--------+--------+
| 0.450 | 0.570 |   0    |
+--------+--------+--------+

With k = 3
Using manhattan distance:
+--------+--------+--------+
| X1     | X2     | Class  |
+========+========+========+
| 0.550 | 0.364 |   1    |
+--------+--------+--------+
| 0.558 | 0.470 |   1    |
+--------+--------+--------+
| 0.456 | 0.450 |   0    |
+--------+--------+--------+
| 0.450 | 0.570 |   0    |
+--------+--------+--------+
```

Result:

X1	X2	Class
0.55	0.364	1
0.558	0.47	1
0.456	0.45	0
0.45	0.57	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)

datasets: http://www.cit.ctu.edu.vn/~dtnghi/ml/data.tar.gz

The solution:

1. Iris:

With k = 1:

```
k = 1
Loading training data from data/iris/iris.trn...
Loading test data from data/iris/iris.tst...
Predicting...
Calculating metrics...
Confusion matrix:
[[17  0  0]
 [ 0 15  0]
 [ 0  3 15]]

Accuracy: 0.94
```

With k= 3:

```
k = 3
Loading training data from data/iris/iris.trn...
Loading test data from data/iris/iris.tst...
Predicting...
Calculating metrics...
Confusion matrix:
[[17  0  0]
 [ 0 15  0]
 [ 0  4 14]]

Accuracy: 0.92
```

2. Face

With k = 1:

```
k = 1
Loading training data from data/faces/data.trn...
Loading test data from data/faces/data.tst...
Predicting...
Calculating metrics...
Confusion matrix:
[[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]
 [ 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  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  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]]

Accuracy: 1.0
```

With k = 3:

```
k = 3
Loading training data from data/faces/data.trn...
Loading test data from data/faces/data.tst...
Predicting...
Calculating metrics...
Confusion matrix:
[[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]
 [ 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  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  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]]

Accuracy: 1.0
```

3. Letter

With k = 1:

```
k = 1
Loading training data from data/letter/let.trn...
Loading test data from data/letter/let.tst...
Predicting...
Calculating metrics...
Confusion matrix:
[[271   3   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0]
 [  0 222   0   0   1   0   0   2   0   1   0   0   0   0   0   0   0   7
    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   0 237   1   4   1   0   0   1   2   0   0   0   1   0   0
    0   0   0   1   0   2   0   8]
 [  0   0   0   0   1 246   0   1   1   0   0   0   0   1   0  18   0   0
    0   1   0   0   0   0   0   0]
 [  0   1   1   0   3   0 248   1   0   0   1   0   0   0   3   0   3   0
    0   0   0   1   1   0   0   0]
 [  0   2   1   4   1   2   2 203   0   0   5   0   0   1   2   0   0   5
    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   0 227   0   0   0   0   1   0   3
    0   0   0   0   0   4   0   0]
 [  0   0   1   0   0   0   2   1   0   1   0 263   0   0   0   0   1   0
    0   0   0   0   0   1   0   0]
 [  0   1   0   1   0   0   1   0   0   0   0   0 237   1   0   0   0   0
    0   0   0   2   2   0   0   0]
 [  1   1   0   1   0   0   0   1   0   1   0   0   2 256   2   0   0   2
    0   0   0   4   1   0   0   0]
 [  0   0   1   1   0   0   0   0   0   0   0   0   0   0 230   0   3   0
    0   0   1   0   1   0   0   0]
 [  0   1   0   1   1  16   0   0   0   0   0   0   0   0   0 243   1   2
    0   0   0   1   0   0   0   0]
 [  0   0   0   1   0   0   1   0   0   0   0   0   0   0   7   0 270   1
    0   0   0   0   0   0   0   0]
 [  0   4   0   0   0   1   0   5   0   0   5   0   0   1   0   0   0 253
    0   1   0   1   0   0   0   0]
 [  0   2   0   2   1   0   0   0   0   0   0   0   1   0   0   0   0   1   2
  253   0   1   0   0   0   0   1]
 [  0   0   1   1   0   2   0   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   0   1   0   1   0   0   0
    0   0 270   1   0   0   0   0]
 [  0   3   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0
    0   0   0 232   2   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   0 238   0   0   0]
 [  0   1   0   1   1   0   0   0   0   0   0   6   0   0   0   0   0   0   0
    0   2   0   0   0 250   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   2   0
    0   3   1   1   0   1 244   0]
 [  0   0   0   0   2   0   0   0   0   0   1   0   0   0   0   0   3   0
    0   0   0   0   0   0   0 253]]

Accuracy: 0.9521452145214522

k = 3
```

With k = 3:

```
k = 3
Loading training data from data/letter/let.trn...
Loading test data from data/letter/let.tst...
Predicting...
Calculating metrics...
Confusion matrix:
[[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]
 [  0 226   0   1   0   0   0   0   0   1   0   0   0   0   0   0   0   5
    1   0   0   4   0   2   0   0]
 [  0   0 217   0   2   0   1   0   0   0   0   0   0   1   0   2   0
    0   0   0   1   2   0   0   0]
 [  0   1   0 265   0   0   1   6   0   0   1   0   0   1   0   0   0   1
    1   0   0   0   0   0   0   0]
 [  0   1   3   0 237   1   4   0   0   0   4   2   0   0   0   0   0   0
    0   0   0   1   0   1   0   8]
 [  0   0   0   0   1 244   0   1   1   0   0   0   0   2   0  16   0   0
    0   3   0   1   0   0   0   0]
 [  0   2   1   2   4   0 248   0   0   0   0   0   1   0   2   0   1   0
    0   0   0   1   1   0   0   0]
 [  0   4   1   6   1   0   2 203   0   0   6   0   0   0   0   0   0   4
    1   0   0   0   0   1   1   0]
 [  0   0   0   0   0   3   0   0 251  14   0   0   0   0   0   0   0   0
    0   0   0   0   0   1   0   0]
 [  0   0   0   0   0   0   0   1   9 227   0   0   0   0   0   0   1   0
    0   0   1   0   0   0   0   0]
 [  0   1   0   0   1   0   0   9   0   0 223   0   0   0   0   0   0   5
    0   0   0   0   0   4   0   0]
 [  0   0   1   0   0   0   1   2   0   2   0 263   0   0   0   0   0   0
    0   0   0   0   0   1   0   0]
 [  0   2   0   0   0   0   1   0   0   0   0   0 235   3   0   0   0   0
    0   0   0   2   2   0   0   0]
 [  1   0   0   0   0   0   0   3   0   0   0   0   4 256   1   0   1   1
    0   0   0   5   0   0   0   0]
 [  0   1   1   1   0   0   0   0   0   0   0   0   0   0 233   0   1   0
    0   0   0   5   0   0   0   0]
 [  0   1   1   1   0   0   0   0   0   0   0   0   0   0 233   0   1   0
    0   0   0   0   0   0   0   0]
 [  0   1   0   1   1  19   0   0   0   0   0   0   0   0   0 240   1   2
    0   0   0   1   0   0   0   0]
 [  0   0   0   1   0   0   1   0   0   0   0   0   0   0   7   0 270   0
    0   0   0   0   0   0   1   0]
 [  0   2   0   0   0   1   0   5   0   0   3   0   0   1   0   0   0 257
    0   0   0   2   0   0   0   0]
 [  0   1   0   2   2   1   0   0   0   0   0   0   0   0   0   0   1   0
  255   0   1   0   0   0   0   1]
 [  0   0   1   2   0   0   0   0   0   1   1   0   0   1   0   0   1   0
    0 232   0   0   0   0   5   0]
 [  0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0
    0   0 272   1   0   0   0   0]
 [  0   4   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   1 232   1   0   0   0]
 [  0   0   0   0   0   0   0   1   0   0   0   0   2   0   1   0   0   0
    0   0   0   0 237   0   0   0]
 [  0   2   0   1   2   0   0   0   0   0   5   0   0   0   0   0   0   0
    0   1   0   0   0 250   0   0]
 [  0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   1   0
    0   3   1   2   0   1 243   0]
 [  0   0   0   0   1   0   0   0   0   2   0   0   0   0   0   0   2   0
    1   0   0   0   0   0   0 253]]

Accuracy: 0.9512451245124512
```

4. Optic

With k = 1:

```
k = 1
Loading training data from data/optics/opt.trn...
Loading test data from data/optics/opt.tst...
Predicting...
Calculating metrics...
Confusion matrix:
[[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]]

Accuracy: 0.9799666110183639
```

With k = 3:

```
k = 3
Loading training data from data/optics/opt.trn...
Loading test data from data/optics/opt.tst...
Predicting...
Calculating metrics...
Confusion matrix:
[[178   0   0   0   0   0   0   0   0   0]
 [  0 180   0   0   0   0   1   0   1   0]
 [  0   4 173   0   0   0   0   0   0   0]
 [  0   0   0 180   0   0   0   2   1   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   0 181   0   0   0]
 [  0   0   0   0   0   0   0 172   1   6]
 [  0   9   0   1   0   0   0   0 162   2]
 [  0   0   0   2   0   1   0   0   1 176]]

Accuracy: 0.9788536449638287
```

Fp

With k = 1:

```
k = 1
Loading training data from data/fp/fp.trn...
Loading test data from data/fp/fp.tst...
Predicting...
Calculating metrics...
Confusion matrix:
[[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0 29  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  1  0 10  0  0  0  0  0  0  0  0  1  0  0  0]
 [ 0  0  0  0  5  0  0  1  0  0  0  0  1  0  0  0]
 [ 0  0  0  0  0  9  0  0  0  0  0  0  0  0  0  0]
 [ 0  1  0  0  0  0 13  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  1  0  0  0  0  0  0 10  0  0  0  0  0  0  0]
 [ 0  1  0  0  0  0  0  0  0  6  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  1  0  0  3  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  9  0  0  0]
 [ 0  0  0  0  0  0  0  3  0  0  0  0  0  7  0  0]
 [ 0  1  0  0  0  0  0  3  0  0  0  0  0  0  6  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 14]]

Accuracy: 0.90625
```

With k = 3:

```
k = 3
Loading training data from data/fp/fp.trn...
Loading test data from data/fp/fp.tst...
Predicting...
Calculating metrics...
Confusion matrix:
[[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0 29  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  1  0 10  0  0  0  0  0  0  0  0  1  0  0  0]
 [ 0  0  0  0  5  0  0  1  0  0  0  0  1  0  0  0]
 [ 0  0  0  0  0  9  0  0  0  0  0  0  0  0  0  0]
 [ 0  2  0  0  0  0 12  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  1  0  0  0  0  0  0 10  0  0  0  0  0  0  0]
 [ 0  1  0  0  0  0  0  0  0  6  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  1  0  0  3  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  9  0  0  0]
 [ 0  0  0  0  0  0  0  4  0  0  0  0  0  6  0  0]
 [ 0  3  0  0  0  0  0  2  0  0  0  0  0  0  5  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 14]]

Accuracy: 0.8875
```

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.

Solution:

Cover-Hart's theorem states that for a given classification problem, the error rate of the 1-Nearest Neighbor (1NN) classifier is no more than twice the Bayes error rate, as long as the training set size is sufficiently large.

Prerequisites:

Let X be a random variable representing the input data, and Y be a random variable representing the corresponding class labels. Let P(x,y) be the joint probability distribution over X and Y, and let P(x|y) and P(y|x) be the conditional probability distributions.

Key Assumptions:

- The sample space is a metric space
- The training dataset is sufficiently large and well-distributed
- The distance function is appropriate for the problem

Mathematical Proof:

The Bayes error rate is defined as the minimum possible error rate that any classifier can achieve. It is the error rate obtained by a classifier that always chooses the class with the highest posterior probability, given the input data:

$$Bayes\_error\_rate = 1 - \max_y P(y|x)$$

The 1NN classifier works by finding the closest training example to the input data and assigning the same class label. The error rate of the 1NN classifier is defined as:

$$1NN\_error\_rate = E[1\{y' \neq y\}]$$

where y' is the class label assigned by the 1NN classifier, y is the true class label, and $1\{y' \neq y\}$ is the indicator function that equals 1 if $y' \neq y$ and 0 otherwise.

To prove Cover-Hart's theorem, we need to show that:

$$1NN\_error\_rate \leq 2Bayes\_error\_rate$$

The Bayes error rate equals the probability that the 1NN classifier assigns the wrong label:

$$Bayes\_error\_rate = P(y' \neq y \mid x)$$

Using the law of total probability:

$$1NN\_error\_rate = E[P(y' \neq y \mid d(x,x') = \min\_d(x,x'))]$$

where d(x,x') is the distance between input data x and the closest training example x'.

For any fixed distance value r:

$$P(y' \neq y \mid d(x,x') = r) \leq P(y' \neq y \mid x, d(x,x') = r) \leq 1 - P(y' = y \mid x, d(x,x') = r)$$

As the training set size approaches infinity:

- The nearest neighbor x' approaches x
- $P(y' \neq y \mid x)$ approaches the Bayes error rate
- Therefore: $1NN\_error\_rate \leq 2Bayes\_error\_rate$

Conclusion:

Cover-Hart's theorem provides a theoretical guarantee that the 1NN classifier's error rate is bounded by twice the Bayes error rate when the training set is sufficiently large. This theoretical foundation helps explain why KNN algorithms are effective in practice and continue to be widely used in machine learning applications.