ANSWERS

8.3 Nearest neighbour

Learning · Robot club · about 20 min

What this lesson is about

The simplest classifier, in a dozen lines.

- How alike are two views?
- The classifier
- Where it goes wrong

Questions

7 marks in all

1. What does this program print? [1 mark]

```
def difference(a, b):
    return sum((p - q) ** 2 for p, q in zip(a, b))

print(difference([10, 20, 30], [12, 20, 27]))
```

Answer:

```
13
```

The differences are -2, 0 and 3. Squared they are 4, 0 and 9, which add to 13.

2. What does this program print? [1 mark]

```
def nearest(sample, data):
    best_label, best_d = None, 1e18
    for feats, label in data:
        d = sum((a - b) ** 2 for a, b in zip(feats, sample))
        if d < best_d:
            best_label, best_d = label, d
    return best_label

DATA = [([50, 50, 50], "open"), ([16, 16, 16], "wall-ahead"), ([45, 45, 16], "gap-left")]
print(nearest([40, 42, 20], DATA))
```

Answer:

```
gap-left
```

The differences are 1064 from open, 1268 from wall-ahead and only 50 from gap-left, so gap-left is the nearest.

3. Why are the differences squared before adding them up? [1 mark]

- ☐ **A** So a negative difference counts as much as a positive one instead of cancelling it out
- ☐ **B** To make the numbers smaller
- ☐ **C** Because the depth grid is square
- ☐ **D** So that the answer is always a whole number

Answer: A. Without squaring, -30 in one column and +30 in another would add to 0 and two very different views would look identical.

4. Around 30 cm from the wall, the classifier says `gap-left` when the truth is a wall ahead. What is the cure? [1 mark]

- ☐ **A** Record more samples, including views from in-between places
- ☐ **B** Write a cleverer `nearest` function
- ☐ **C** Delete the gap-left sample
- ☐ **D** Use rows 6 and 7 instead

> **Answer:** A. There was no sample from that distance. The cure is more data, not cleverer code.

5. The dataset has no sample labelled `corner`. The robot drives into a corner. What can nearest neighbour say? [1 mark]

- ☐ **A** One of the labels it has, whichever sample looks most like the corner
- ☐ **B** corner
- ☐ **C** unsure
- ☐ **D** Nothing: it raises an error

> **Answer:** A. Nearest neighbour can only answer with a label it has seen, from a place it has seen it.

6. A student tests `nearest` by classifying each sample in `DATA` and gets every one right. What does that show? [1 mark]

- ☐ **A** Very little: each sample is at difference 0 from itself, so it always wins. Test on views it has not stored
- ☐ **B** The classifier is perfect and ready for the mat
- ☐ **C** The data has no wrong labels
- ☐ **D** The samples were recorded in enough places

> **Answer:** A. Testing on the data it learned from always looks perfect. Only new views, like the task's spots, show whether it works.

7. Two views of 16 readings differ by exactly 1 cm in every column. What is their difference, using the sum of squared differences? [1 mark]

> **Answer:** 16. Each column adds 1 squared, which is 1, and there are 16 columns.

The task: nearest neighbour

Drive to (0, 15), (0, 40), (-38, 40) and (38, 40) in turn, and print `spot <n>: <label>` at each, using `nearest` against `DATA`.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()
# maths: atan2, hypot, sin, cos, radians
import math

def wrapped(h):
    return (h + 180) % 360 - 180

def go_to(x, y, speed=60):
    # where am I?
    px, py = position()
    a = math.radians(wrapped(math.degrees(math.atan2(x - px, y - py)) - heading()))
    # forward, sideways, rotation: -100 to 100 each, until the next command
    drive(speed * math.cos(a), speed * math.sin(a), wrapped(0 - heading()) * 3)

def near(x, y, cm=4):
    # where am I?
    px, py = position()
    return math.hypot(x - px, y - py) < cm

def drive_to(x, y):
    while not near(x, y):
        go_to(x, y)
        # pause 0.1 s (the robot keeps doing what it was told)
        wait(0.1)
    # all motors off
    stop()
    # pause 0.4 s (the robot keeps doing what it was told)
    wait(0.4)
def level_rows():
    # rows 2 and 3: the 16 readings that look straight ahead
    return tof_grid()[16:32]

def nearest(sample, data):
    # the label of the recorded sample most like this one (smallest sum of squared
differences)
    best_label, best_d = None, 1e18
    for feats, label in data:
        d = sum((a - b) ** 2 for a, b in zip(feats, sample))
        if d < best_d:
            best_label, best_d = label, d
    return best_label
DATA = [
    ([54, 52, 51, 51, 51, 51, 52, 54, 54, 52, 51, 51, 51, 51, 52, 54], 'open'),
    ([16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16], 'wall-ahead'),
    ([36, 47, 46, 46, 46, 16, 16, 16, 36, 47, 46, 46, 46, 16, 16, 16], 'gap-left'),
    ([16, 16, 16, 46, 46, 46, 47, 36, 16, 16, 16, 46, 46, 46, 47, 36], 'gap-right'),
]
drive_to(0, 15)
print('spot 1:', nearest(level_rows(), DATA))
```

The hint students can ask for: Drive to the four spots (0, 15), (0, 40), (-38, 40), (38, 40) in turn and print `spot <n>: <label>` using nearest neighbour against DATA.

A solution

```
from bugbot import *
connect()
import math

def wrapped(h):
    return (h + 180) % 360 - 180

def go_to(x, y, speed=60):
    px, py = position()
    a = math.radians(wrapped(math.degrees(math.atan2(x - px, y - py)) - heading()))
    drive(speed * math.cos(a), speed * math.sin(a), wrapped(0 - heading()) * 3)

def near(x, y, cm=4):
    px, py = position()
    return math.hypot(x - px, y - py) < cm

def drive_to(x, y):
    while not near(x, y):
        go_to(x, y)
        wait(0.1)
    stop()
    wait(0.4)
def level_rows():
    return tof_grid()[16:32]              # rows 2 and 3: the 16 readings that look
straight ahead

def nearest(sample, data):
    # the label of the recorded sample most like this one (smallest sum of squared
differences)
    best_label, best_d = None, 1e18
    for feats, label in data:
        d = sum((a - b) ** 2 for a, b in zip(feats, sample))
        if d < best_d:
            best_label, best_d = label, d
    return best_label
DATA = [
    ([54, 52, 51, 51, 51, 51, 52, 54, 54, 52, 51, 51, 51, 51, 52, 54], 'open'),
    ([16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16], 'wall-ahead'),
    ([36, 47, 46, 46, 46, 16, 16, 16, 36, 47, 46, 46, 46, 16, 16, 16], 'gap-left'),
    ([16, 16, 16, 46, 46, 46, 47, 36, 16, 16, 16, 46, 46, 46, 47, 36], 'gap-right'),
]
spots = [(0, 15), (0, 40), (-38, 40), (38, 40)]
for i, (x, y) in enumerate(spots):
    drive_to(x, y)
    print(f"spot {i + 1}: {nearest(level_rows(), DATA)}")
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.