ANSWERS

U12.3 Generalisation

Learning, and the capstone · University · about 35 min

What this lesson is about

Held-out data, overfitting, and choosing a model by the error it makes on points it never saw.

- The claim a fit makes
- Overfitting, concretely
- The only honest test
- Fitting a polynomial in plain Python
- Model selection

Questions

7 marks in all

1. A degree 5 polynomial through six noisy speed measurements has zero training error. What has it learned? [1 mark]

- ☐ **A** Mainly the noise in those six points, so it cannot be trusted on new commands, least of all beyond the data
- ☐ **B** The true relationship exactly
- ☐ **C** The same as a straight line, since both fit the data
- ☐ **D** Nothing, because a zero error means the fit failed

> **Answer:** A. Six parameters can pass through any six points. The bends that hit every noisy point say nothing about new commands. Between the points they are small when the noise is small, but past the data nothing holds them down: in the lesson's task it misses command 95 by 18 cm/s.

2. A line is fitted to the three training points and scored by RMS error on both sets. What does this print? [1 mark]

```
train = [(30, 5.8), (50, 9.9), (70, 14.1)]
test = [(40, 8.2), (60, 11.7), (80, 16.3)]
n = len(train)
sx = sum(c for c, v in train)
sv = sum(v for c, v in train)
sxx = sum(c * c for c, v in train)
sxv = sum(c * v for c, v in train)
a = (n * sxv - sx * sv) / (n * sxx - sx * sx)
b = (sv - a * sx) / n

def rms(points):
    return (sum((a * c + b - v) ** 2 for c, v in points) / len(points)) ** 0.5

print(round(rms(train), 3), round(rms(test), 3))
```

> **Answer:**
>
> ```
> 0.024 0.278
> ```
>
> The line misses its own three points by 0.024 cm/s RMS and the three it never saw by 0.278 cm/s. Training error is an optimistic estimate of the error on new data.

3. A student compares five models by their test error, picks the best, and reports that same test error as the result. What is wrong? [1 mark]

- ☐ **A** The test set has been used to choose the model, so the reported error is optimistic; a separate validation set should do the choosing
- ☐ **B** Nothing, as long as the test points were never used to fit
- ☐ **C** Five models is too few to compare
- ☐ **D** Test error should be reported on the training data instead

> **Answer:** A. Choosing by the test error quietly fits to the test set. Train to fit, validate to choose, test to report, and open the test set once.

4. With very little data, what method splits the data into k parts, fits k times leaving each part out in turn, and averages the held-out errors? [1 mark]

> **Answer:** k-fold cross validation. Every point is used for fitting and for testing, and no fit ever sees the points it is scored on.

5. Which statements about bias and variance are right? [1 mark]

Tick every answer that is true.

- ☐ **A** A model with too few parameters is wrong the same way every time
- ☐ **B** A model with too many parameters is wrong a different way each time it is refitted on fresh data
- ☐ **C** The error on new data has a minimum somewhere in between
- ☐ **D** Adding parameters always lowers the error on new data
- ☐ **E** Training error measures variance

> **Answer:** A, B, C. Too simple is bias, too flexible is variance. Adding parameters always lowers the training error, not the error on new data.

6. Why scale commands, u = (c - 55) / 25, before fitting a high degree polynomial by the normal equations? [1 mark]

- ☐ **A** Raw commands raised to high powers differ by many orders of magnitude, and the solution comes back as noise
- ☐ **B** Scaling improves the model's held-out error
- ☐ **C** The normal equations only accept values between 0 and 1
- ☐ **D** It removes the dead band from the data

> **Answer:** A. A degree 5 fit needs sums of the tenth power: 80 to the tenth is about 10^19, and it sits in the same matrix as a count of 6, which makes Gaussian elimination numerically useless. Scaling is the difference between an answer and rubbish.

7. Two models score nearly the same held-out error, one degree 1 and one degree 3. Which should you choose, and why? [1 mark]

- ☐ **A** Degree 1, because the simpler model has less variance, so its held-out score is more trustworthy
- ☐ **B** Degree 3, because it fits the training data better
- ☐ **C** Degree 3, because it can represent more shapes
- ☐ **D** Either, because held-out error is the only thing that matters

> **Answer:** A. That is Occam's razor with a practical justification: when the scores are close, prefer the model whose score you can believe.

The task: choose the model by held-out error

Sample fourteen commands, six for training and eight for testing, fit a degree 1 and a degree 5 model to the training six, print all four errors, and print `train 5:` and `best degree:`. Three of the test commands, 25, 85 and 95, lie outside the training range, which is where an overfitted model is at its worst.

```
from bugbot import *
connect()

DT = 0.1
TRAIN = [30, 40, 50, 60, 70, 80]
TEST = [25, 35, 45, 55, 65, 75, 85, 95]
```

The hint students can ask for: Collect a speed at each of the fourteen commands, six for training and eight for testing, and never let the test eight touch the fit. Fit a straight line and a degree 5 polynomial to the training six, report each one's root mean square error on training and on test, and pick the winner on test. Scale the commands to about -1 to 1 before you raise them to the fifth power.

A solution

```
from bugbot import *
connect()

DT = 0.1
TRAIN = [30, 40, 50, 60, 70, 80]
TEST = [25, 35, 45, 55, 65, 75, 85, 95]

def turn_cmd():
    err = (imu()[0] + 180) % 360 - 180
    if abs(err) < 2.0:
        return 0
    cmd = max(18.0, min(30.0, abs(1.5 * err)))     # below 15 the drive does nothing at all
    return -cmd if err > 0 else cmd

def hold(cmd, ticks, collect=False):
    vs = []
    for i in range(ticks):
        drive(cmd, 0, turn_cmd())
        if collect:
            vs.append(flow()[1])
        wait(DT)
    return vs

def speed_at(cmd):
    hold(cmd, 12)
    vs = hold(cmd, 3, True)
    return sum(vs) / len(vs)

def sample(cmd):
    out = speed_at(cmd)
    back = speed_at(-cmd)
    return (out - back) / 2.0

train = [(c, sample(c)) for c in TRAIN]
test = [(c, sample(c)) for c in TEST]
stop()
wait(0.3)
```

```
def u(c):
    return (c - 55.0) / 25.0            # commands scaled to about -1 to 1

def fit(points, degree):
    """Least squares by the normal equations, solved with Gaussian elimination."""
    m = degree + 1
    a = []
    for i in range(m):
        row = [sum(u(c) ** (i + j) for c, v in points) for j in range(m)]
        row.append(sum(v * u(c) ** i for c, v in points))
        a.append(row)
    for col in range(m):
        p = max(range(col, m), key=lambda r: abs(a[r][col]))
        a[col], a[p] = a[p], a[col]
        for r in range(m):
            if r != col and a[col][col] != 0:
                f = a[r][col] / a[col][col]
                for k in range(col, m + 1):
                    a[r][k] -= f * a[col][k]
    return [a[i][m] / a[i][i] for i in range(m)]

def predict(w, c):
    return sum(wi * u(c) ** i for i, wi in enumerate(w))

def rms(w, points):
    return (sum((predict(w, c) - v) ** 2 for c, v in points) / len(points)) ** 0.5

line = fit(train, 1)
wiggle = fit(train, 5)
print("train 1:", round(rms(line, train), 3), " test 1:", round(rms(line, test), 3))
print("train 5:", round(rms(wiggle, train), 3))
print("test 5:", round(rms(wiggle, test), 3))
print("best degree:", 1 if rms(line, test) <= rms(wiggle, test) else 5)
```

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