

LECTURE 11

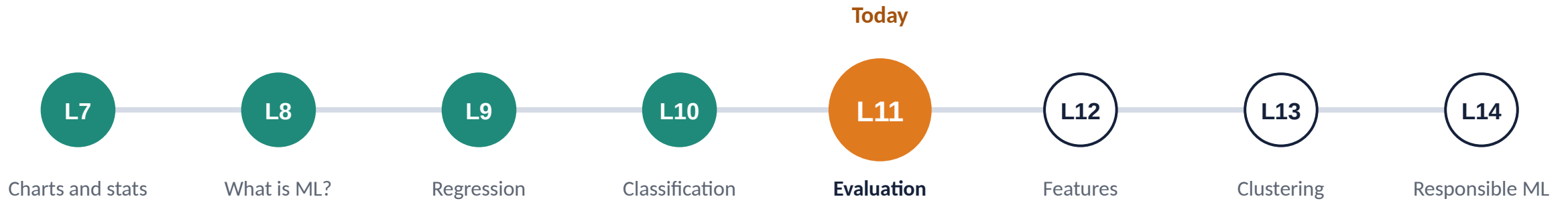
Evaluating models properly

Overfitting, cross-validation, precision and recall

Today's key line of code

```
cross_val_score(model, X, y, cv=5)  
precision_score(y_te, pred)
```

The road ahead: machine learning



Today you will

- 1 Recognise underfitting and overfitting
- 2 Use a validation set or cross-validation to choose settings
- 3 Compute precision, recall and F1 by hand
- 4 Pick the right metric for the problem
- 5 Spot the accuracy trap on imbalanced data

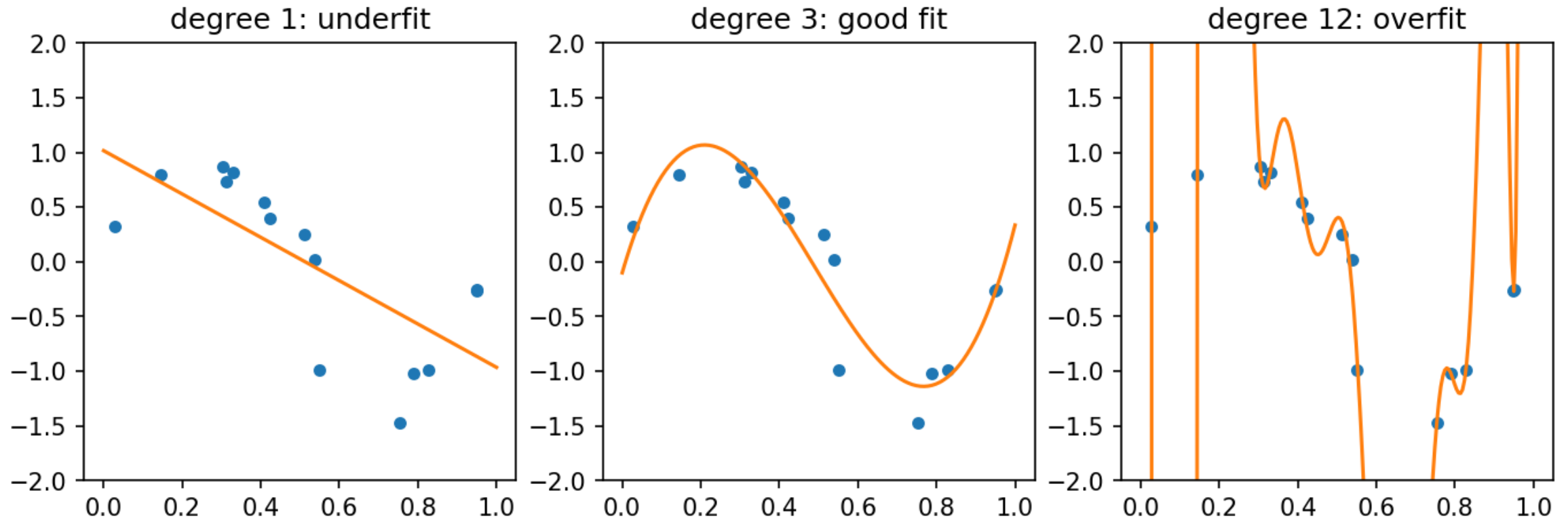
Too simple, just right, too complex

The same 15 noisy points (invented) fitted with curves of increasing flexibility. The right one follows the trend, not the noise.

Code

```
import numpy as np
import matplotlib.pyplot as plt
rng = np.random.default_rng(1)
x = np.sort(rng.uniform(0, 1, 15))
y = np.sin(2 * np.pi * x) + rng.normal(0, 0.25, 15)
grid = np.linspace(0, 1, 200)
fig, axes = plt.subplots(1, 3, figsize=(11, 3.3))
for ax, deg, name in zip(axes, [1, 3, 12],
                          ["underfit", "good fit", "overfit"]):
    coefs = np.polyfit(x, y, deg)
    ax.scatter(x, y, s=18)
    ax.plot(grid, np.polyval(coefs, grid), color="tab:orange")
    ax.set_ylim(-2, 2)
    ax.set_title(f"degree {deg}: {name}")
plt.show()
```

Too simple, just right, too complex: the result



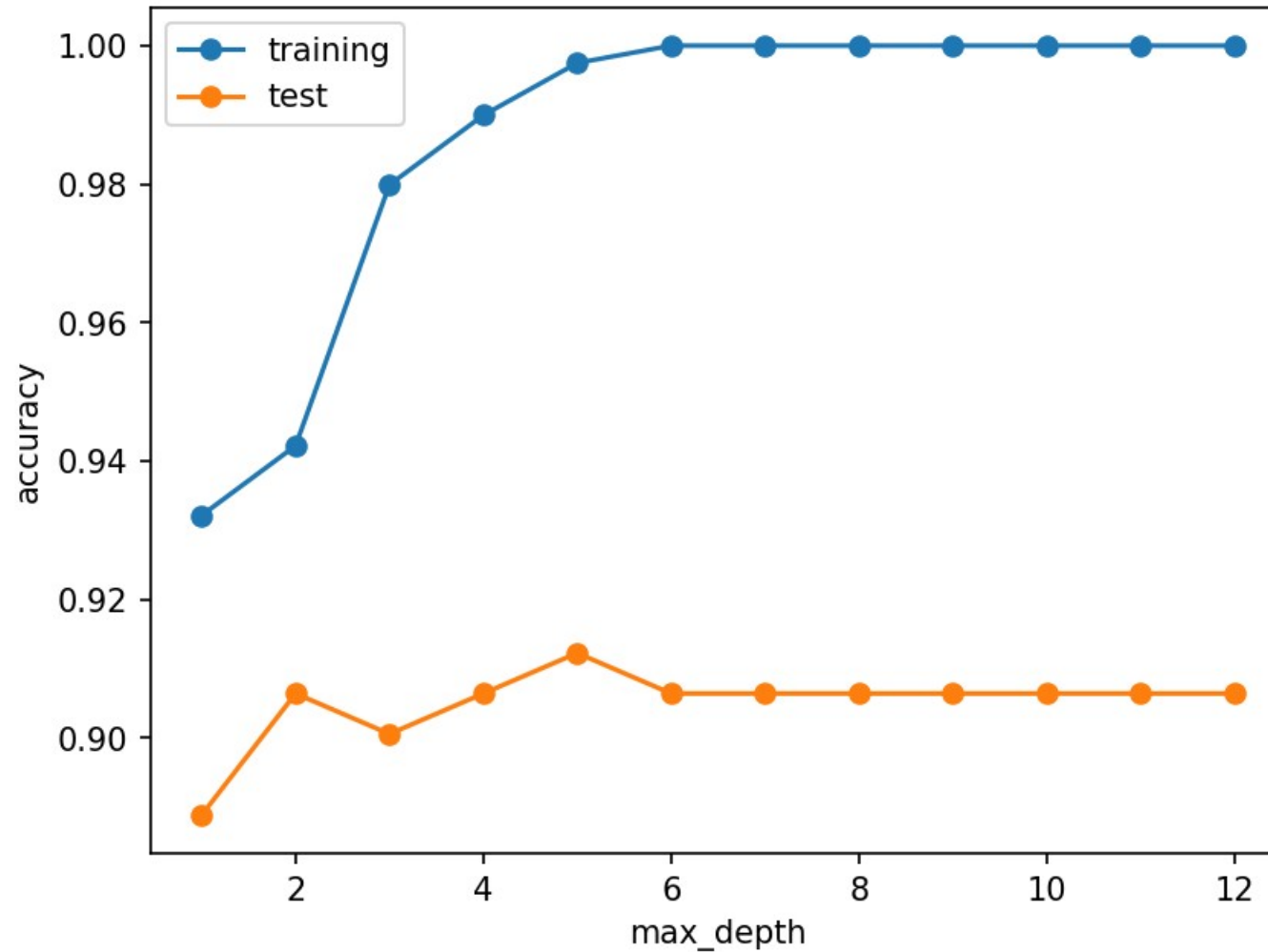
Overfitting in numbers: tree depth

Breast cancer data. As trees get deeper, training accuracy reaches 100% while test accuracy stops improving or drops.

Code

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
X, y = load_breast_cancer(return_X_y=True)
X_tr, X_te, y_tr, y_te = train_test_split(
    X, y, test_size=0.3, random_state=0, stratify=y)
depths = range(1, 13)
tr, te = [], []
for d in depths:
    t = DecisionTreeClassifier(max_depth=d, random_state=0)
    t.fit(X_tr, y_tr)
    tr.append(t.score(X_tr, y_tr))
    te.append(t.score(X_te, y_te))
plt.plot(list(depths), tr, marker="o", label="training")
plt.plot(list(depths), te, marker="o", label="test")
plt.xlabel("max_depth")
plt.ylabel("accuracy")
plt.legend()
plt.show()
```

Overfitting in numbers: tree depth: the result



Watch it: from underfitting to overfitting

WATCH FOR

- The curve gets more flexible frame by frame
- Training error keeps falling
- Test error falls, then rises again
- The best model sits at the bottom of the test curve

EVALUATION, STEP BY STEP

From underfitting to overfitting

The curve becomes more flexible step by step.
Watch the training error and the test error.

Click the video to play, click again to pause. Silent: talk the class through it.

Three sets, three jobs

1

Training set

The model learns from it with `fit()`.

2

Validation set

We compare settings (k, depth) on it and pick the best.

3

Test set

Used once, at the very end, to report the final score.

5-fold cross-validation, pictured

Split the training data into 5 parts. Each round, one part validates and four train. Average the 5 scores.

Code

```
# Every row is used for validation exactly once.  
# 5 scores instead of 1:  
#   - a more reliable average  
#   - and a sense of how much it varies
```

	part 1	part 2	part 3	part 4	part 5
round 1	VALID	train	train	train	train
round 2	train	VALID	train	train	train
round 3	train	train	VALID	train	train
round 4	train	train	train	VALID	train
round 5	train	train	train	train	VALID

Click to reveal the next row. Predict it first!

Cross-validation in one line

`cross_val_score` runs all 5 rounds · Use only the training data · Report the mean and the spread · Choose depth by mean CV score, not by test score

Code

```
from sklearn.model_selection import cross_val_score
for d in [2, 3, 5, 10]:
    t = DecisionTreeClassifier(max_depth=d, random_state=0)
    scores = cross_val_score(t, X_tr, y_tr, cv=5)
    print(d, scores.round(3), round(scores.mean(), 3))
```

Output

```
2 [0.95  0.95  0.9   0.899 0.962] 0.932
3 [0.962 0.912 0.888 0.899 0.962] 0.925
5 [0.95  0.9   0.912 0.911 0.975] 0.93
10 [0.95  0.888 0.912 0.899 0.975] 0.925
```

Then, and only then, the test set

- Cross-validation picked `max_depth = 2`
- Fit it on all training data
- Score the test set once
- That number is what you report

Code

```
best = DecisionTreeClassifier(max_depth=2, random_state=0)
best.fit(X_tr, y_tr)
print(round(best.score(X_te, y_te), 3))
```

Output

0.906

By hand: precision, recall and F1 for our spam model

Our Lecture 6 results: TP = 117 spam caught, FP = 5 real messages flagged, FN = 14 spam missed.

Code

```
# precision: of the messages flagged as spam,  
#           how many really were spam?  
# recall:   of all real spam,  
#           how much did we catch?  
# F1: one number balancing both
```

metric	formula	numbers	value
precision	$TP / (TP + FP)$	12 / 22	0.959
recall	$TP / (TP + FN)$	12 / 22	0.893
F1	$2PR / (P + R)$	$2 \cdot P \cdot R / (P + R)$	0.925

Click to reveal the next row. Predict it first!

Which metric matters?

Problem	Worse mistake	Watch especially
Spam filter	Losing a real message (false positive)	Precision
Disease screening	Missing a sick patient (false negative)	Recall
Fraud alerts	Both are costly	F1, then discuss
Balanced, low-stakes task	Neither stands out	Accuracy is fine

The accuracy trap

Predict first

Code

```
from sklearn.metrics import accuracy_score, recall_score
y_true = [0] * 95 + [1] * 5
y_pred = [0] * 100
print(accuracy_score(y_true, y_pred))
print(recall_score(y_true, y_pred))
```



95 healthy people, 5 sick. A 'model' always answers healthy. What are its accuracy and recall?

Answer: The accuracy trap

Code

```
from sklearn.metrics import accuracy_score, recall_score
y_true = [0] * 95 + [1] * 5
y_pred = [0] * 100
print(accuracy_score(y_true, y_pred))
print(recall_score(y_true, y_pred))
```

Output

```
0.95
0.0
```

- Accuracy 95%: sounds excellent
- Recall 0: it finds no sick person at all
- On imbalanced data, accuracy hides failure
- Always check recall and precision for the rare class

All the metrics at once

- `classification_report` gives precision, recall and F1 per class
- support = how many test rows of each class
- In this dataset, class 0 = malignant
- Read the row of the class you care most about

Code

```
from sklearn.metrics import classification_report
pred = best.predict(X_te)
print(classification_report(y_te, pred,
                           target_names=["malignant", "benign"], digits=3))
```

Output

	precision	recall	f1-score	support
malignant	0.843	0.922	0.881	64
benign	0.950	0.897	0.923	107
accuracy			0.906	171
macro avg	0.897	0.910	0.902	171
weighted avg	0.910	0.906	0.907	171

Your turn

10 minutes

- 1 Use `cross_val_score` to choose `k` for `k`-NN on the breast cancer training data (`k = 1, 5, 15, 31`)
- 2 Fit the best `k` on all training data
- 3 Print its `classification_report` on the test set
- 4 What is the recall for malignant tumours?

Code

```
from sklearn.neighbors import KNeighborsClassifier
for k in [1, 5, 15, 31]:
    s = cross_val_score(KNeighborsClassifier(n_neighbors=k),
                        X_tr, y_tr, cv=5)
    print(k, round(s.mean(), 3))
...
```

Starter code: replace the ... with your own code

Common evaluation mistakes

Read the last line of an error first. It names the problem.

What you typed	What Python says	What it means
<pre>cross_val_score(best, X_tr, y_tr, scoring="acuracy")</pre>	<pre>InvalidParameterError: The 'scoring' parameter of cross_val_score must be a str among {'recall_micro', 'roc_auc', 'preci ...</pre>	Misspelt metric name. The message lists valid names.
<pre>recall_score(["a", "b"], ["a", "a"])</pre>	<pre>ValueError: pos_label=1 is not a valid label. It should be one of ['a', 'b']</pre>	With text labels, say which one is positive: pos_label="b".
<pre>cross_val_score(best, X_tr, y_tr[:100], cv=5)</pre>	<pre>ValueError: Found input variables with inconsistent numbers of samples: [398, 100]</pre>	X and y must have the same number of rows.

Interactive lab: Evaluation Lab

In the lab you can

1 Underfit and overfit

Raise the polynomial degree; watch error on new points.

2 Choose depth honestly

5-fold CV picks the depth; the test set is scored once.

3 Precision and recall

The spam model's threshold, matrix and PR curve.

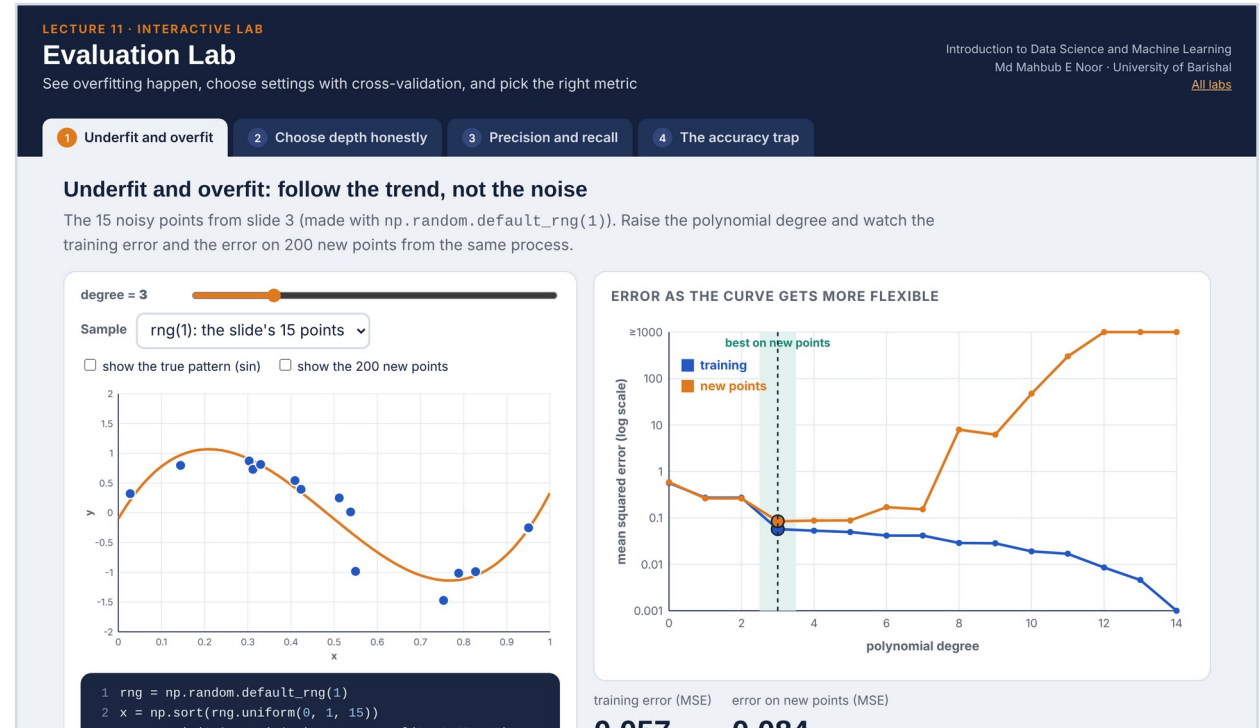
4 The accuracy trap

Rare disease, lazy model: 95% accuracy, 0 recall.

Open the lab ►

Works offline in any browser

Labs_open_in_browser/
L11_Evaluation_Lab.html



Today's toolkit

underfit / overfit	Too simple / memorises noise
cross_val_score(m, X, y, cv=5)	5 validation scores
precision	$TP / (TP + FP)$
recall	$TP / (TP + FN)$

F1	Balance of precision and recall
classification_report	All metrics per class
test set	Used once, at the end

Homework 11

Due before Lecture 12 • [date]

1 By hand

A model has $TP = 40$, $FP = 10$, $FN = 20$, $TN = 130$. Compute accuracy, precision, recall and F1.

2 Choose depth honestly

On wine, choose `max_depth` with 5-fold cross-validation on the training data, then report the test accuracy once.

3 Metric choice

For three problems of your choice, say which mistake is worse and which metric you would watch.

4 Variation

Print the 5 CV scores for one model. What are the smallest and largest? Why do they differ?

5 Explain

In 3 sentences: why is choosing settings on the test set a form of cheating?

Submit on [LMS / Google Classroom]: your notebook file `HW11_<your ID>.ipynb`, or a Colab link set to “Anyone with the link can view”.

Recap: Lecture 11 in 90 seconds

RECAP

Lecture 11 in 90 seconds

A short recap of today's lecture. Pause any time.

Click to play or pause. Watch it again at home to revise.