Overfitting and Evaluating a Model

Overfitting is memorising instead of learning. Train score vs test score, the confusion

Introduction

Here is the most important lesson in this whole series. A model can score perfectly on its training data and still be useless — because it memorised the answers instead of learning the pattern. This failure has a name, **overfitting**, and every later topic in machine learning is, in one way or another, a defence against it.

Think of a student who memorises last year's exam paper word for word. On that exact paper they score 100%. On this year's paper, with new questions on the same topics, they fail — they never learned the subject, only the paper. An overfitted model is that student. It learned every quirk, typo and coincidence of the training data, including noise that will never repeat. A good model is the student who studied the topics: slightly lower on the practise paper, much better on the new one.

How do you tell the two apart? Compare two scores: the **train score** (how the model does on data it trained on) and the **test score** (how it does on unseen data). A big gap — high train score, low test score — is the signature of overfitting. And for classification, one table makes the model's mistakes visible instead of hiding them behind accuracy: the **confusion matrix**.

It helps to picture overfitting as a curve, not a single mistake. Imagine models from very simple to very complex, left to right. On the left, a too-simple model misses even the real pattern: both train and test scores are low. That is underfitting. In the middle, the model catches the real pattern without memorising noise: both scores are decent and close. That is the sweet spot. On the right, a very complex model fits every quirk: the train score climbs towards 100% while the test score falls away. That is overfitting. Your job is not to maximise the train score — that road leads right, off the sweet spot — but to find the middle, where the test score is highest.

One more defence deserves a plain explanation: **regularisation**. It is a penalty for complexity added to the model's training score, so a model that needs a wildly complicated shape to fit the data gets punished for it. In plain words, regularisation tells the model "you may fit the data, but keep it simple unless complexity really earns its keep". It is the same instinct as limiting a tree's depth, applied to many model types, and it is one of the first tools professionals reach for when the train-test gap appears.

(Roman Urdu: Overfitting ka matlab hai ratta lagana, seekhna nahi — training par number poora, naye data par fail. Is ko pehchanne ka tareeqa train score aur test score ka farq dekhna hai.)

Key points

- **Overfitting = memorising the training data, including its noise.** The model fits quirks that will not appear in new data.
- **Underfitting is the opposite failure.** A model too simple to catch even the real pattern — it scores badly on training data too.

- **Train score vs test score is the key comparison.** High train + low test = overfitting. Low train + low test = underfitting. Both decent and close together = a healthy model.
- **A confusion matrix shows the four kinds of classification result:** true positives (predicted sick, actually sick), true negatives (predicted healthy, actually healthy), false positives (predicted sick, actually healthy) and false negatives (predicted healthy, actually sick).
- **False negatives and false positives hurt differently.** In illness testing, missing a sick patient (false negative) is usually worse than a false alarm. The matrix lets you see which mistake your model makes most.
- **Accuracy hides the matrix.** Two models can share 90% accuracy while making completely different mistakes. Always look at the matrix before trusting a score.
- **Cross-check habit: never trust one split.** Check the model on a different test split, or repeat the split a few times with different shuffles. If the score jumps wildly, the model is not stable yet.
- **Picture the sweet spot, not just the scores.** Too simple = underfitting (both scores low). Just right = both scores decent and close. Too complex = overfitting (train high, test low).
- **Regularisation penalises complexity.** It adds a cost for complicated models during training, pushing the model back towards the simple, general shape.
- **More real data is the best cure for overfitting.** With more varied examples, quirks stop looking like patterns, and the model is forced to learn what actually repeats.

Examples

Example 1 — Spotting overfitting from two numbers

Model A: train score 99%, test score 98%. Model B: train score 100%, test score 71%. Model A is the better model, even though B looks perfect on training data. B memorised; A learned. The gap between train and test is the warning sign — a gap of nearly 30 points means the model will disappoint on real data.

Example 2 — A confusion matrix in plain words

A clinic model tests 100 people. The results, in a simple table:

	Actually sick	Actually healthy
Predicted sick	8 (right)	12 (false alarm)
Predicted healthy	2 (missed sick)	78 (right)

Accuracy is (8 + 78) ÷ 100 = 86% — sounds fine. But the matrix shows the real story: the model missed 2 sick patients and raised 12 false alarms. If missing a sick patient is dangerous, this model needs work despite its 86%. One number hid all of that; the table revealed it.

Example 3 — A simple cross-check in code (scikit-learn)

This example needs scikit-learn installed first: `pip install scikit-learn`. It trains a decision tree and compares its training score with its test score using only standard calls:

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

X = [[1], [2], [3], [4], [5], [6], [7], [8], [9], [10]]
y = [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]  # 0 = Fail, 1 = Pass

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)
print("Train score:", model.score(X_train, y_train))
print("Test score:", model.score(X_test, y_test))
```

What the result means: `.score` on the training set shows how well the model fits data it has seen; `.score` on the test set shows how well it generalises to data it has not. If the two are close, the model is healthy. If the train score is much higher, suspect overfitting — and try a simpler model or a depth limit (lesson ML.6).

Example 4 — Worked example: three models, one winner, by the numbers

Three models trained on the same student data, judged on the same test set:

| Model | Train score | Test score | Gap | Verdict | |---|---|---|---|---| | Simple rule (hours ≥ 5 → Pass) | 82% | 80% | 2 points | Healthy, slightly underfit | | Decision tree, depth 2 | 90% | 88% | 2 points | Healthy — the sweet spot | | Decision tree, no depth limit | 100% | 63% | 37 points | Overfitted — memorised |

The unlimited tree looks best if you only read the train column, and worst if you read the test column. The depth-2 tree wins because its test score is highest and its gap is tiny. Notice also that the simple rule is not a bad model — it is only 8 points behind the winner, and it is the easiest to explain. In a real project, you would report all three rows, not just the winner, so a reader can see that the choice was close and why you made it.

Example 5 — Worked example: cross-checking properly, in code (scikit-learn)

This example needs scikit-learn installed first: `pip install scikit-learn`. Instead of one lucky or unlucky split, it checks the model on five different splits and reports every score:

```
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier

X = [[i] for i in range(1, 21)]
y = [0] * 10 + [1] * 10  # 0 = Fail, 1 = Pass

model = DecisionTreeClassifier(max_depth=2, random_state=42)
scores = cross_val_score(model, X, y, cv=5)
print("Scores on 5 splits:", scores)
print("Average score:", scores.mean())
```

Run it and look at the spread, not just the average. If the five scores are close together, the model is stable and you can trust the average. If they jump around — say from 60% to 100% — the model's score depends on which rows happened to land in the test set, and you need more data or a simpler model before claiming anything. This habit, checking several splits, is what turns a single number into evidence.

Real-world case — A hospital model that failed in a new hospital

A team trained a model to spot patients at risk, using data from one large hospital. On that hospital's test set it scored 94% — an excellent result. When a second hospital tried the same model, the score fell to 71%. What happened? The first hospital's data had quirks the model had memorised: its own coding habits, its own equipment, even the way its forms were filled in. Those quirks did not exist in the second hospital, so the memorised patterns failed. The model had not learned "risk" in general; it had learned "risk at hospital one". The fix was to train on data from several hospitals and to test on a hospital the model had never seen — the same train/test discipline from lesson ML.3, applied at the level of whole hospitals. Overfitting is not just a classroom problem; it is how real models fail in real deployments.

Quick recap

- Overfitting is memorising training data, including noise; underfitting is missing even the real pattern.
- Compare train score and test score: a big gap means overfitting.
- The confusion matrix shows all four outcomes: two kinds of right, two kinds of wrong.
- Accuracy alone hides which mistakes a model makes.
- Cross-check on different splits before trusting a score.

Common mistakes

1. **Celebrating a 100% training score** — it usually means memorisation, not mastery. Fix: always report the test score beside it.
2. **Judging a classifier by accuracy alone** — a model can be 95% accurate and catch nothing. Fix: read the confusion matrix, especially the false negatives.
3. **Trusting a single lucky split** — one test set can flatter or punish a model by chance. Fix: repeat with different shuffles and look at the range of scores, not one number.

1. **Tuning on the test set until the score looks good** — each tweak guided by the test score leaks test information into the model. Fix: tune on a validation set, and touch the test set once, at the end.
2. **Reporting only the best split's score** — cherry-picking the luckiest split is the same as inventing a better model. Fix: report the average across splits and the range, not the best single number.

Practice

1. Two models: A scores train 95%, test 93%; B scores train 100%, test 60%. Which would you ship, and why? Answer in two sentences.

2. Draw the confusion matrix for a spam filter that, out of 50 emails (10 spam, 40 not spam), correctly caught 8 spam, missed 2 spam, and wrongly flagged 5 good emails. What is its accuracy?
3. Run the code in Example 3, then remove `random_state=42` and run it three times. Do the test scores stay the same? Write one line about what that tells you.

1. A model scores train 88%, test 87% on split one, and train 99%, test 65% on split two (same model, different data). Which split worries you more, and what would you try first to fix it?
2. Build a confusion matrix for a model that tested 200 emails (40 spam, 160 not spam) and caught 30 spam, missed 10 spam, and wrongly flagged 20 good emails. What is its accuracy, and why does that number mislead?
3. Run the code in Example 5, then change `max_depth=2` to no depth limit and run it again. Compare the spread of the five scores and write one line about what changed.

Practice quiz

1. What is overfitting?

- Fitting too few points
- Memorising the training data so well that the model fails on new data
- Using too little data
- A model that is too simple

Answer: Memorising the training data so well that the model fails on new data

2. What does a big gap between train score and test score tell you?

- The model is perfect
- Likely overfitting — it learned the training set, not the pattern
- The test set is broken
- More training is always the fix

Answer: Likely overfitting — it learned the training set, not the pattern

3. What does a confusion matrix show?

- How confused the user is
- A table of right and wrong predictions per class
- The model's code
- The training time

Answer: A table of right and wrong predictions per class

Watch the video lesson: https://youtu.be/1UrTwhPA-iw

Read online: https://scopetreasure.com/notes/ml-overfitting-evaluation · More free tools and notes at https://scopetreasure.com