Episode 6: Metrics, Pipelines & Hyperparameter Tuning

Building ML Workflows You Can Trust

AI & ML: From Theory to Practice

Matthew Care 2026

What You'll Learn

By the end of this episode, you will understand:

- Why **accuracy is misleading** for unbalanced datasets
- How to use the **confusion matrix** to understand model errors
- **Precision, Recall, F1, and MCC** — what each metric captures
- How **Pipelines** automate data preparation and prevent data leakage
- The difference between **parameters** and **hyperparameters**
- How **intelligent tuning** finds the best model configuration
- How **ensembles** combine models for better predictions

Why Accuracy Fails

The Accuracy Paradox

The Problem with Accuracy

Accuracy = percentage of correct predictions. *Sounds reasonable, right?*

But consider this scenario — a fraud detection system:

Model	Accuracy	What It Actually Does
Model A	**99%**	Predicts "no fraud" for everything. Catches **zero** frauds.
Model B	**95%**	Actually detects 80 out of 100 fraud cases.

The paradox: The model with *lower* accuracy is clearly more useful! Accuracy alone is misleading when one class dominates.

> *A fire alarm that never goes off is "correct" 99.99% of the time — but it's useless.*

When Can You Trust Accuracy?

Class Balance	Accuracy Reliable?
Balanced (50/50)	Yes
Slightly imbalanced (60/40)	Mostly
Moderately imbalanced (75/25)	**Caution**
Highly imbalanced (95/5)	**No**
Extremely imbalanced (99/1)	**Definitely not**

Rule of thumb: If the largest class makes up more than ~70% of the data, be sceptical of accuracy alone.

> *Most real-world problems have some degree of imbalance. We need better metrics.*

Better Ways to Measure

The Confusion Matrix and Beyond

The Confusion Matrix

A 2×2 table of prediction outcomes:

	Predicted: Positive	Predicted: Negative
Actual: Positive	True Positive (TP)	False Negative (FN)
Actual: Negative	False Positive (FP)	True Negative (TN)

TP: Correctly predicted positive

TN: Correctly predicted negative

FP: Incorrectly predicted positive (**Type I error**)

FN: Incorrectly predicted negative (**Type II error**)

> *All classification metrics are built from these four numbers.*

Understanding the Two Types of Error

False Positive (FP)

Type I Error — "False Alarm"

Predicted positive, but actually negative

- Flagging a legitimate transaction as fraud
- Diagnosing a disease in a healthy person
- Marking a real email as spam

False Negative (FN)

Type II Error — "Missed Detection"

Predicted negative, but actually positive

- Missing an actual fraud case
- Failing to detect a real disease
- Letting spam through to your inbox

The key question: In your application, which error is more costly?

In cancer screening, **missing a disease (FN)** is far worse than a false alarm (FP). In spam filtering, **deleting a real email (FP)** is worse than letting spam through (FN)

Precision: "How Trustworthy Are Positive Predictions?"

$$\text{Precision} = \frac{\text{True Positives}}{\text{All Positive Predictions}} = \frac{TP}{TP + FP}$$

"When the model says positive, how often is it right?"

- **High precision** → Few false alarms
- When you predict something is positive, you can trust that prediction

Precision matters when false alarms are costly:

- Don't delete real emails (spam filter)
- Don't freeze legitimate bank accounts (fraud detection)
- Don't raise unnecessary panic (alert systems)

> *A precise archer hits the target consistently — but might not choose to shoot at every target.*

Recall: "How Complete Is Our Detection?"

$$\text{Recall} = \frac{\text{True Positives}}{\text{All Actual Positives}} = \frac{TP}{TP + FN}$$

"Of all the actual positive cases, how many did we find?"

- **High recall** → Few missed detections
- We're catching most of the positive cases

Recall matters when missing positives is dangerous:

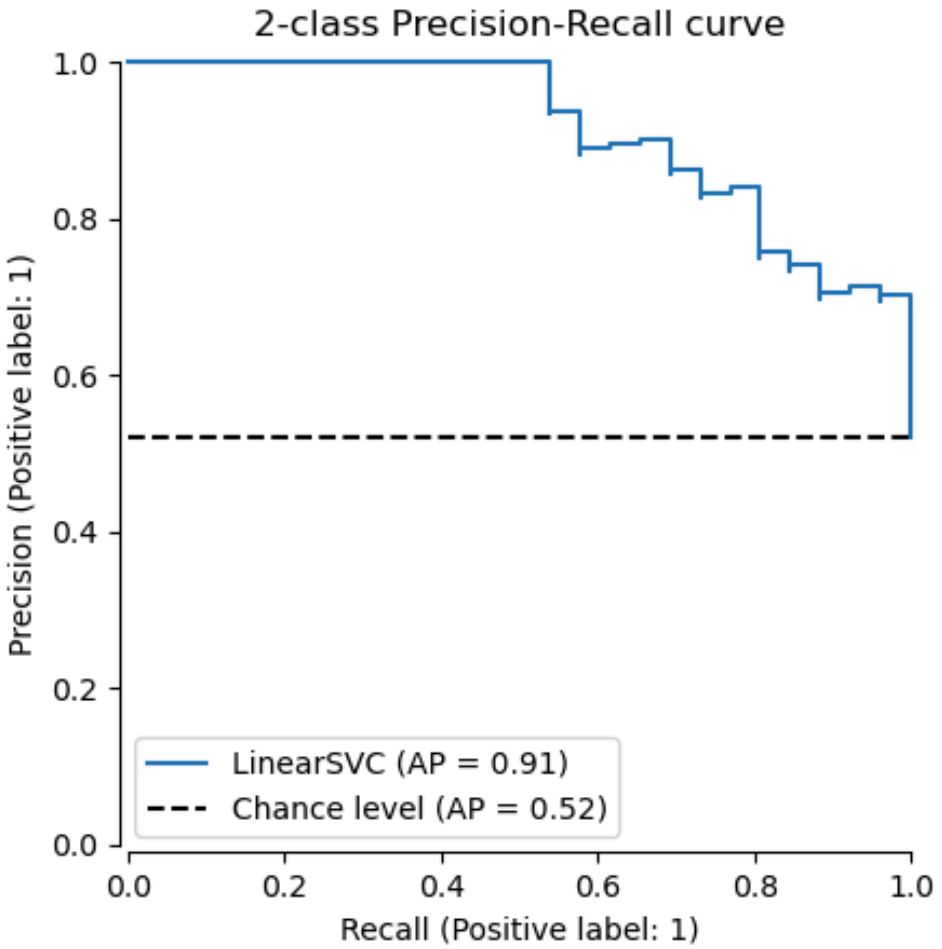
- Don't miss cancer in a screening test
- Don't fail to detect a security threat
- Don't miss a relevant legal document in discovery

> *A thorough detective finds every clue — but might also investigate some false leads.*

The Precision–Recall Trade-off

You usually can't maximise both. Improving one tends to worsen the other. The right balance depends on the costs of each type of error in your application.

> *An airport security system set very sensitively (high recall) catches every threat but also flags many innocent passengers (low precision). Dialled down, it's faster but might miss something...*



Which Should You Prioritise?

Application	Priority	Reason
Email spam filter	**Precision**	Don't delete important emails
Cancer screening	**Recall**	Don't miss cancer cases
Search engine results	**Precision**	Top results should be relevant
Legal document review	**Recall**	Must find all relevant documents
Fraud detection	**Depends**	Balance false alarms vs missed fraud

Ask yourself: What's the real-world cost of a false positive compared to a false negative?

F1 Score: Balancing Precision and Recall

When you need a single number that balances both:

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Key property: F1 is high **only when both** precision and recall are high.

Precision	Recall	F1 Score
0.90	0.90	**0.90**
0.90	0.50	**0.64**
0.90	0.10	**0.18**

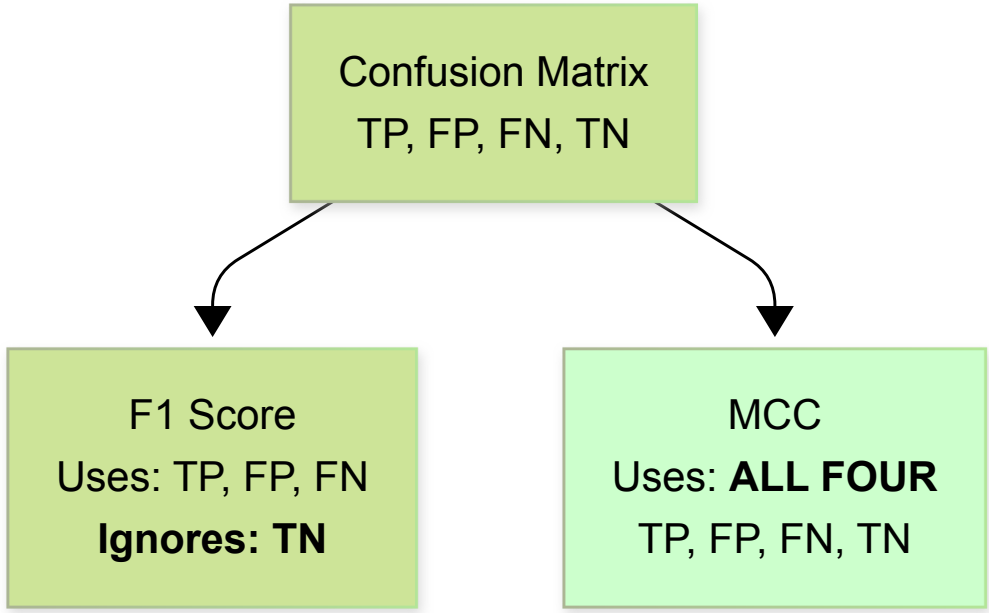
> *F1 **penalises imbalance**. You can't get a good F1 score by being great at one thing but terrible at the other.*

F1's Limitation: It Ignores True Negatives

F1 only uses TP, FP, and FN — it completely ignores TN.

This means F1 **doesn't reward the model for correctly identifying negatives**, e.g. a patient is disease free, a transaction isn't fraudulent etc.

> *For a truly balanced view of model performance, we need a metric that considers all four cells.*



MCC: The Gold Standard for Imbalanced Data

Matthews Correlation Coefficient uses all four confusion matrix cells:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Score	Meaning
+1	Perfect predictions
0	No better than random guessing
-1	Perfectly wrong predictions

Critical advantage: A model that always predicts the majority class gets **MCC = 0**, while accuracy makes it look respectable (e.g., 76%). MCC correctly reveals it has no predictive power.

Comparing Metrics Side by Side

Same census income data, same useless "always predict majority" model:

Metric	"Always Predict ≤$50K"	Actual Trained Model
Accuracy	76% *(looks okay!)*	83%
F1 (minority class)	0.0	0.58
MCC	**0.00** *(correctly shows no value)*	**0.52**

Recommendation: Use **MCC** as your primary metric for imbalanced classification. Supplement with precision/recall for interpretability.

> *MCC is the truth-teller among metrics. It can't be fooled by class imbalance.*

Choosing the Right Metric: Summary

Situation	Recommended Metrics
Balanced classes	Accuracy, F1, MCC *(all similar)*
Moderate imbalance	**MCC**, Balanced Accuracy, F1
Severe imbalance	**MCC**, per-class F1, PR AUC
False positives costly	Precision, Specificity
False negatives costly	Recall, Sensitivity
Need to rank predictions	ROC AUC, PR AUC

> *When in doubt, **MCC is the safest default**. It's honest regardless of class distribution.*

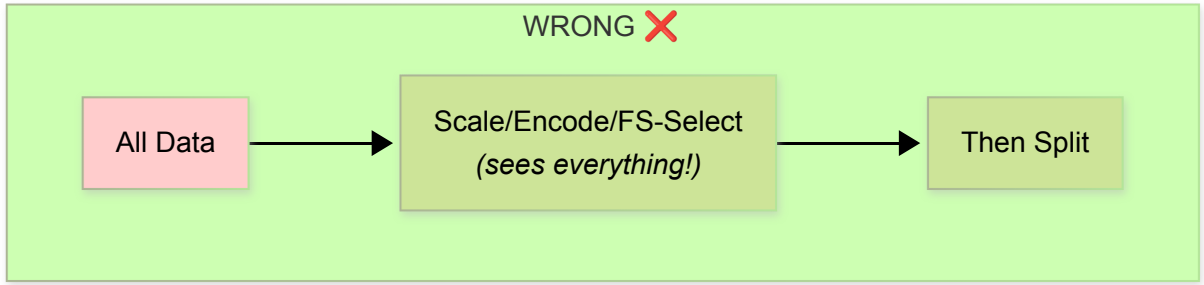
Building Reliable Workflows

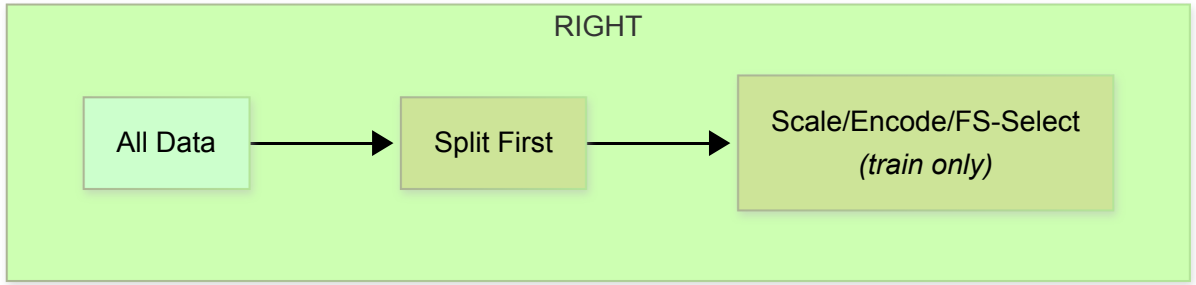
Pipelines: Automating and Protecting Your Process

The Data Leakage Problem (Revisited)

From Episode 5: Never let test data influence training.

But this is surprisingly easy to get wrong:



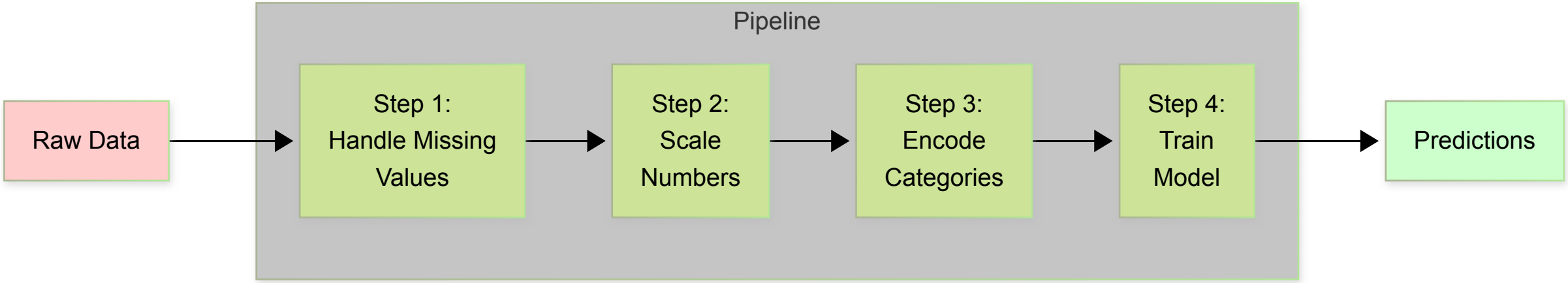


Doing this correctly during cross-validation (where you split multiple times) is **tedious and error-prone** if done manually.

> *This is where Pipelines come in — they automate correct methodology.*
>
> Easy to add additional functionality (e.g. over/under-sampling, feature-subsetting etc.)

What is a Pipeline?

A Pipeline chains all steps — preparation and modelling — into a single automated workflow:



The Pipeline automatically ensures:

- Each step is fitted on **training data only**
- The same transformations are applied consistently to new data (not learned again)
- No data leakage — even during cross-validation

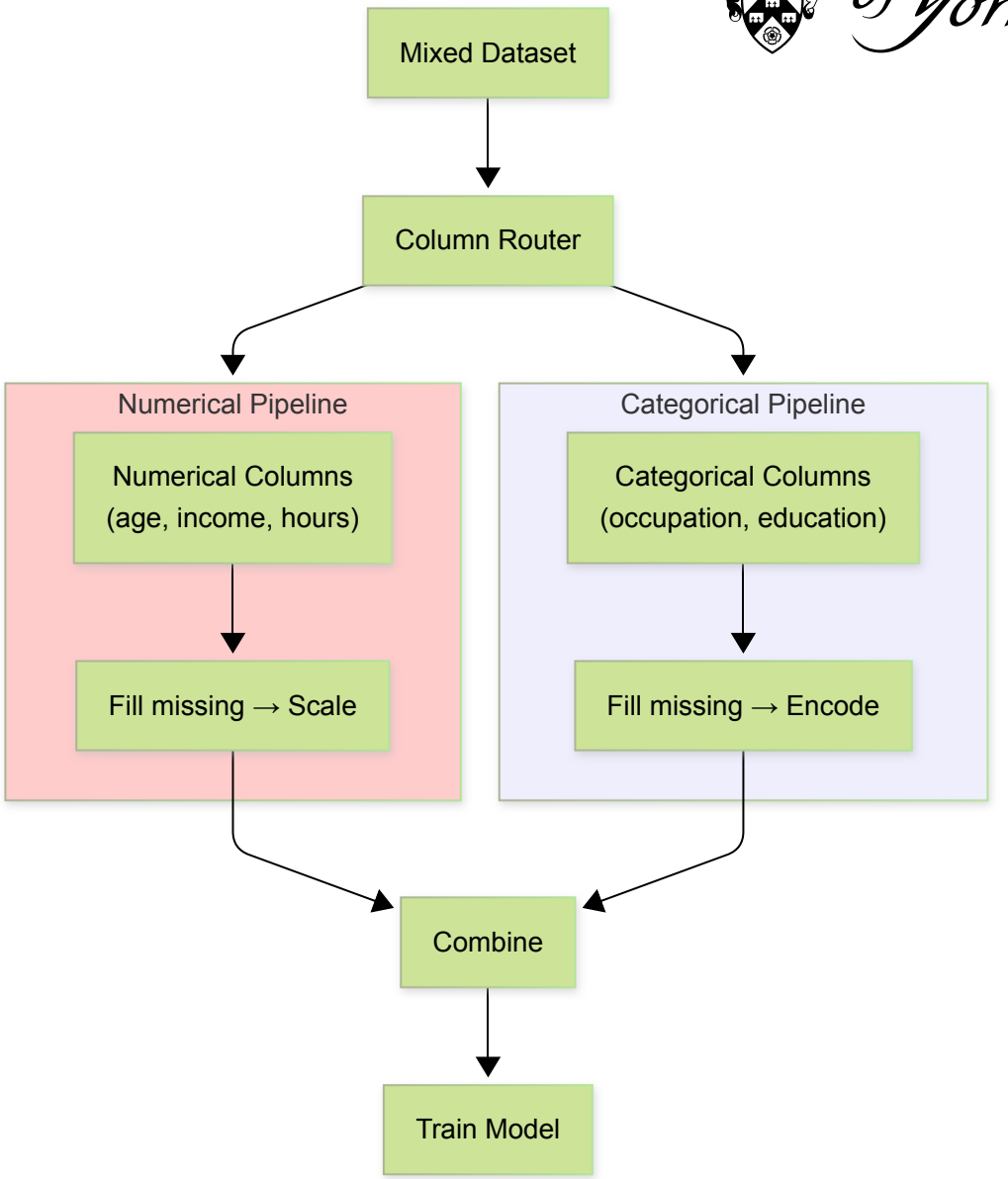
> *Think of it as an assembly line where each station does its job in the right order, every time, without mistakes.*

Different Columns Need Different Treatment

Real datasets have mixed types — each needs its own preparation:

Pipelines handle this routing automatically — you specify which columns get which treatment, and the system manages the rest.

> Handled via Sklearn ColumnTransformer



Pipeline Benefits

Correctness

- **No data leakage** — automatic isolation
- Consistent preprocessing every time
- Proper fit/transform separation

Convenience

- Single object to save and deploy
- Works with all evaluation tools
- Easy to experiment with different configurations

Best practice: Always use pipelines. They prevent subtle bugs that can silently invalidate your results.

> *A pipeline is like a standard operating procedure in a lab — it ensures every experiment follows the same rigorous protocol, regardless of who runs it.*

Making Models Better

Hyperparameter Tuning

Parameters vs Hyperparameters

Parameters

Learned automatically during training

- Model discovers these from data
- Examples: coefficients, weights, split points

> *Like the specific skills a student develops through studying*

Hyperparameters

Set by the user before training (external to learning)

- Control how the model learns
- Examples: learning speed, model complexity, number of trees

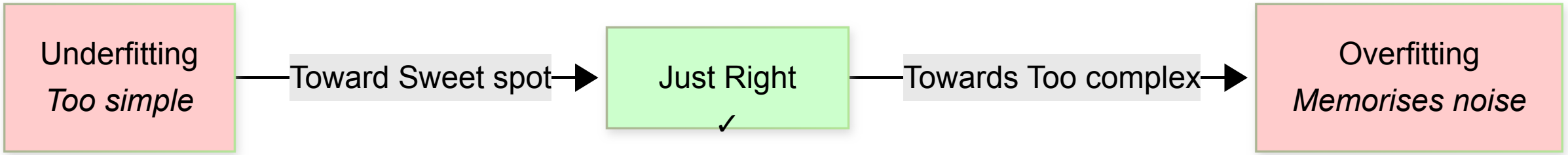
> *Like choosing class size, study hours, and teaching method*

Hyperparameter tuning = systematically finding the best configuration settings for your model and data.

Why Tune Hyperparameters?

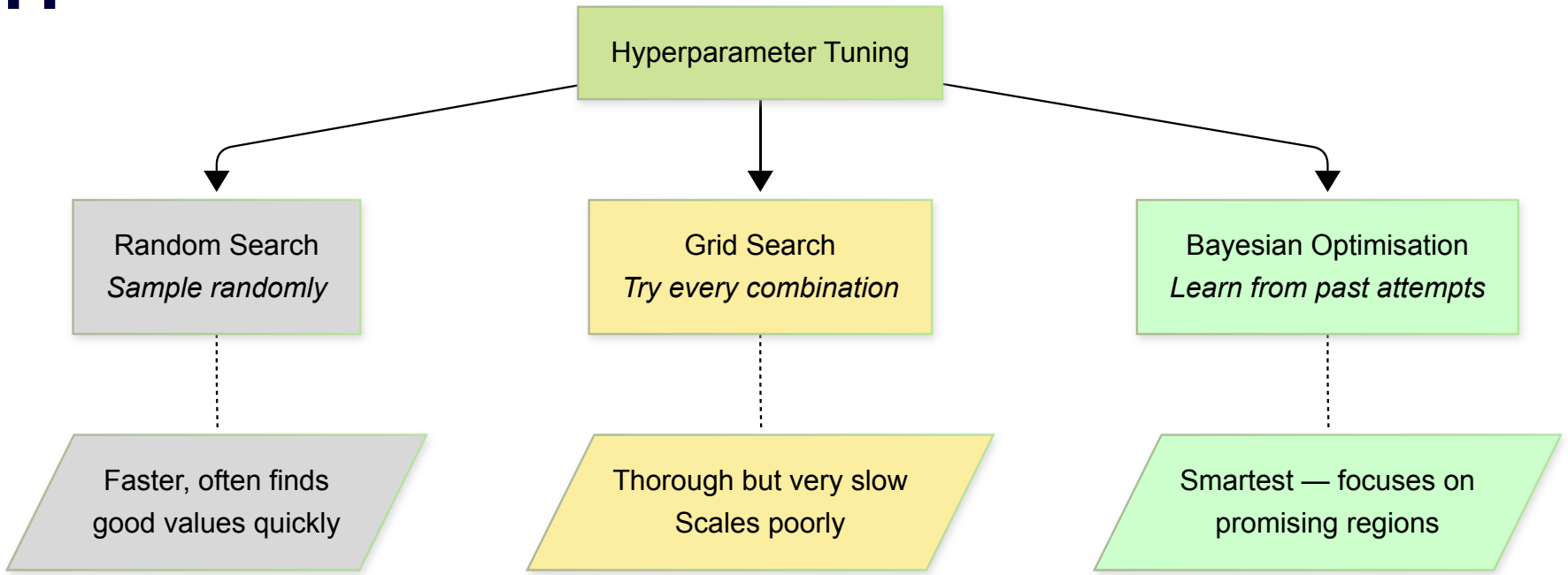
The same algorithm with different settings can produce very different results:

Setting	Effect
Model too simple	**Underfitting** — misses important patterns
Model too complex	**Overfitting** — memorises noise
Learning too fast	Unstable, jumps over good solutions
Learning too slow	Takes forever, may get stuck



> *Tuning is finding the Goldilocks zone — not too simple, not too complex, just right for your data.*

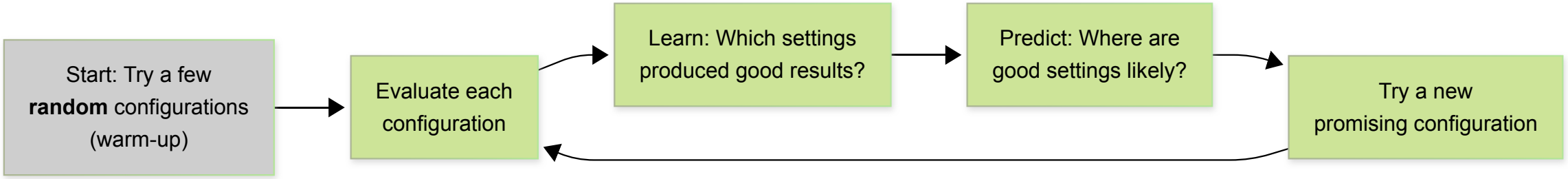
Tuning Approaches



Method	Trials Needed (4 settings, 5 options each)	Uses Past Results?
Grid Search	**625** (tries all)	No
Random Search	~60–100	No
Bayesian (e.g. Optuna)	~30–50	**Yes**

Intelligent Optimisation: Learning From Experience

Bayesian optimisation (used by tools like Optuna) works like a smart scientist:



1. **Explore** randomly at first to understand the landscape
2. **Learn** which regions of the search space work well
3. **Exploit** that knowledge to focus on promising areas
4. **Repeat** — getting smarter with each attempt

> *Rather than blindly searching everywhere, it's like a prospector who learns where gold is likely and focuses their digging there.*

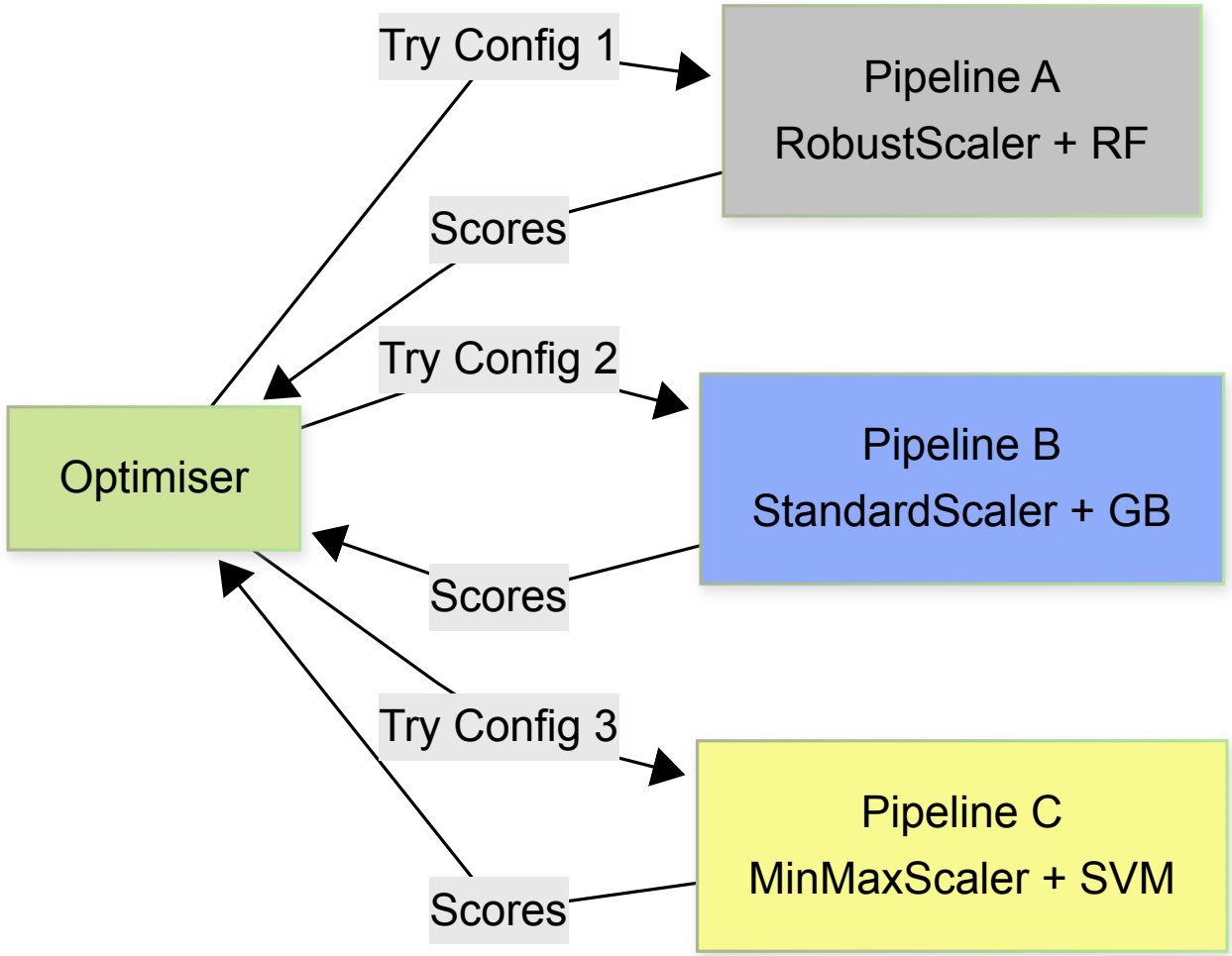
Tuning the Complete Pipeline

The real power: tuning everything together

Intelligent optimisation can simultaneously search for:

- The **best preprocessing** choices (which scaling method? Over/under-sampling, how to handle missing values?)
- The **best algorithm** (Random Forest? Gradient Boosting? SVM?)
- The **best hyperparameters** for the chosen algorithm

> *You're not just tuning one dial — you're tuning the entire instrument panel to find the best combination.*



Fine-Tuning Results

Threshold Optimisation

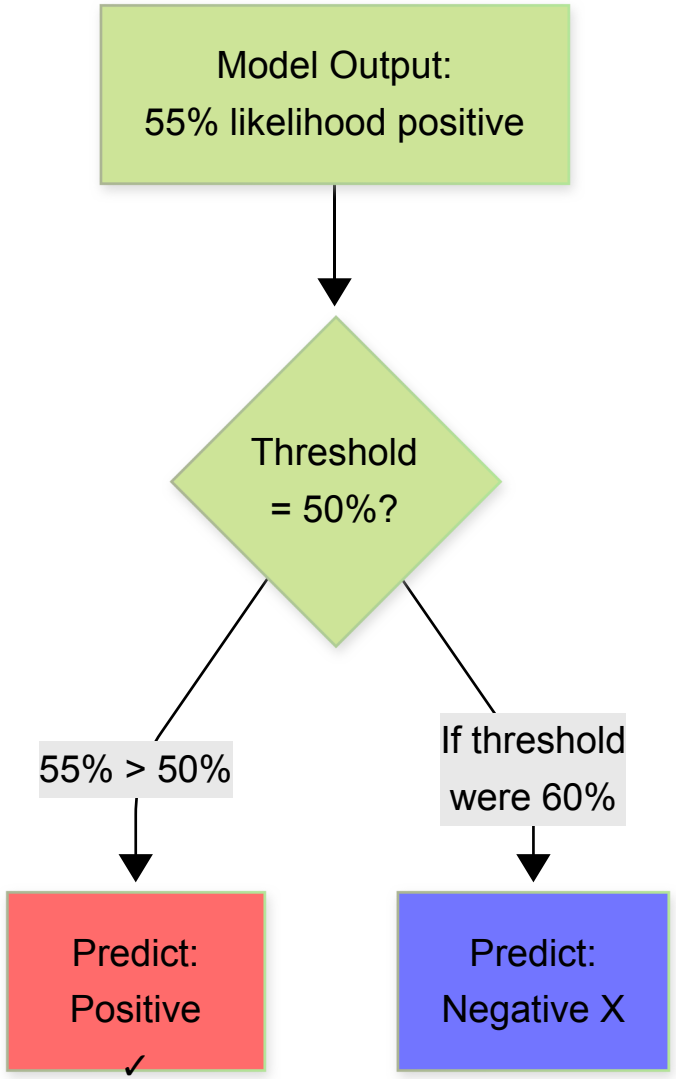
The Decision Threshold

Most models output a probability, then convert it to a prediction using a threshold:

The default threshold is 50%, but this assumes:

- Equal costs for both types of error
- Balanced class distribution
- Well-calibrated probabilities

> *These assumptions rarely hold in practice.*



Adjusting the Threshold

Moving the threshold trades precision for recall:

Threshold	Effect
Lower (e.g., 30%)	More positive predictions → **higher recall**, but more false alarms
Default (50%)	Balanced starting point
Higher (e.g., 70%)	Fewer positive predictions → **higher precision**, but more missed cases

Optimise the threshold on validation data (not test data!) by searching for the value that maximises your chosen metric (e.g., MCC or F1).

Note: this needs an additional validation split (**not** done in CE1/CE2)

Combining Models

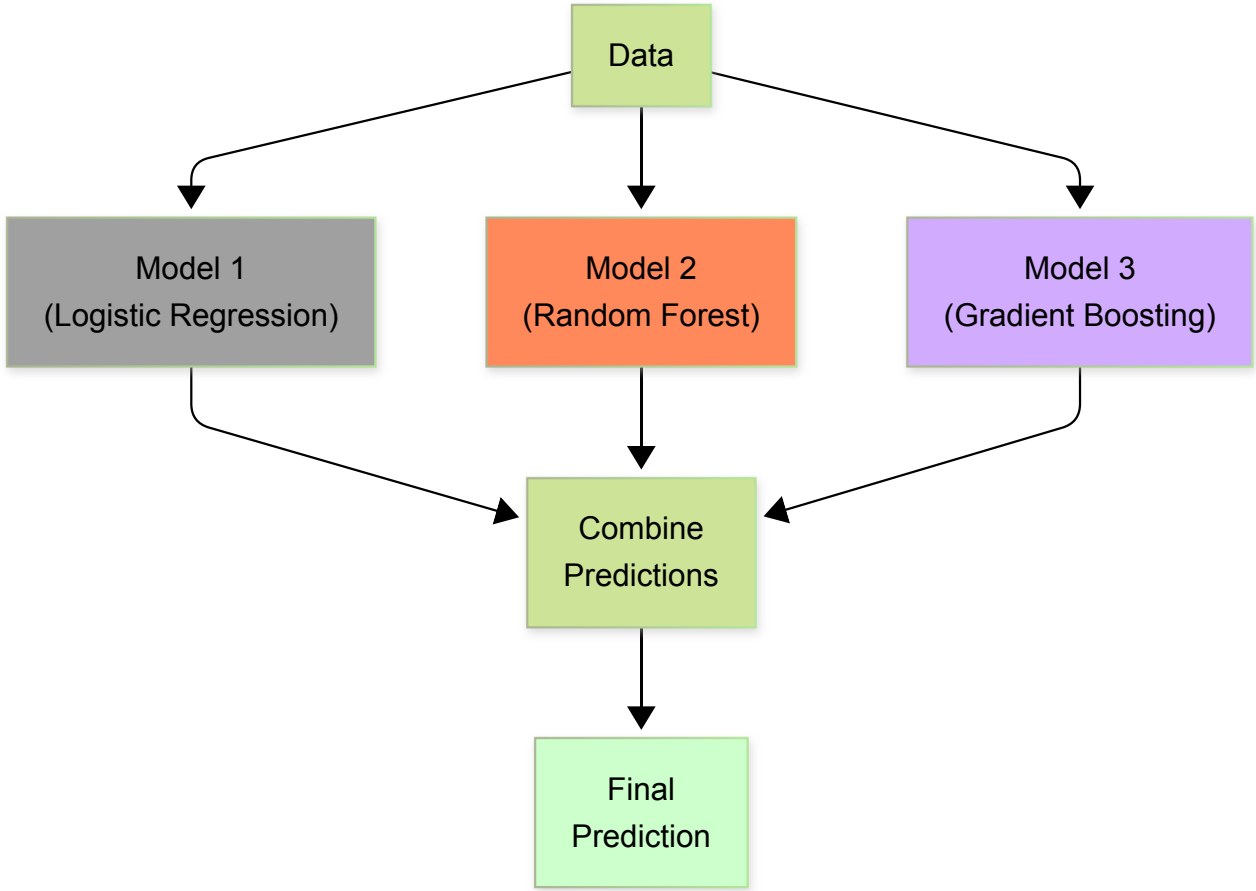
Ensemble Methods: The Wisdom of Crowds

Ensemble Methods: Why Combine Models?

Different models make different errors. Combining them can average out mistakes.

> *A panel of doctors diagnosing a patient is likely to be more accurate than any single doctor. Each brings different expertise and perspectives.*

Key requirement: models must be **diverse** — if they all make the same mistakes, combining them doesn't help.



Two Main Ensemble Approaches

Voting

Simple combination

- **Hard voting:** majority wins
- **Soft voting:** average the confidence scores

> *Like a committee voting on a decision*

Simple, effective, easy to implement

Stacking

Learned combination

- Train a "meta-model" that learns *how* to best combine the base models
- The meta-model learns which base models to trust in which situations

> like a senior consultant that knows which specialist is best for a particular problem, and combines their opinions into a final diagnosis

More powerful, but more complex

When Ensembles Help (and Don't)

Ensembles Help When:

- Base models are **strong but diverse**
- Models make **different types of errors**
- Sufficient data to train multiple models
- Slight accuracy improvement justifies the added complexity

Ensembles Don't Help When:

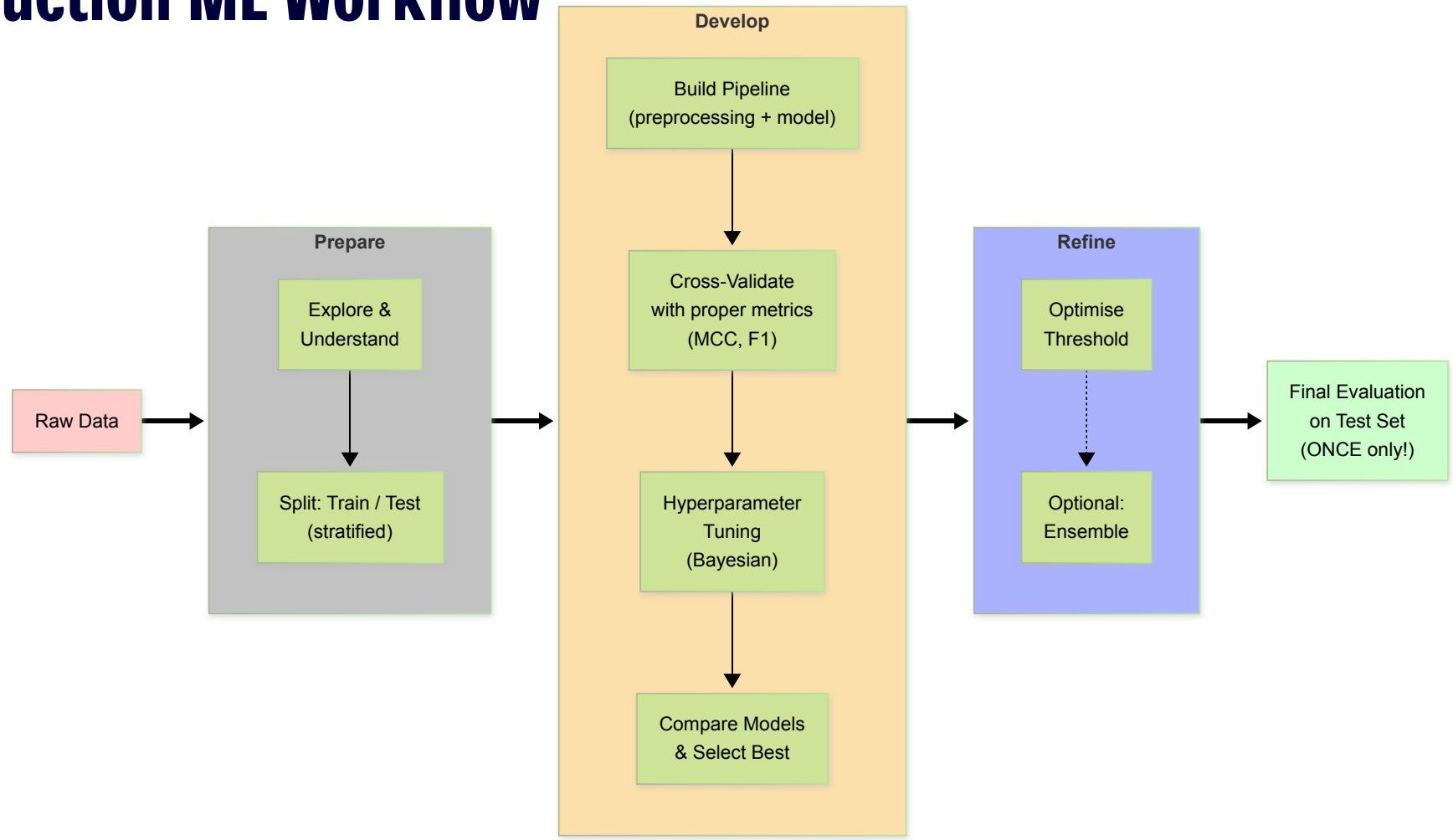
- All models are similar
- One model clearly dominates
- Interpretability is required
- Limited computation budget

Remember: Combining several weak models won't produce a strong one. Ensembles amplify diversity, not weakness.

Putting It All Together

The Complete ML Pipeline

The Production ML Workflow



> *Each step builds on the previous. Pipelines ensure correctness. Proper metrics ensure honesty. Tuning ensures optimal performance.*

From CE1 to CE2

Aspect	Episode 5 (Basics)	Episode 6 (Production)
Data preparation	Manual, step-by-step	Automated Pipelines
Primary metric	Accuracy	MCC
Tuning	None	Bayesian optimisation (*Optuna*)
Class imbalance	Acknowledged	Addressed with metrics + resampling
Threshold	Default (0.5)	Optimised
Models	Single models	Ensembles of best models

Episode 5 teaches the fundamentals. Episode 6 teaches how to do them *properly and reliably*.

Key Takeaways

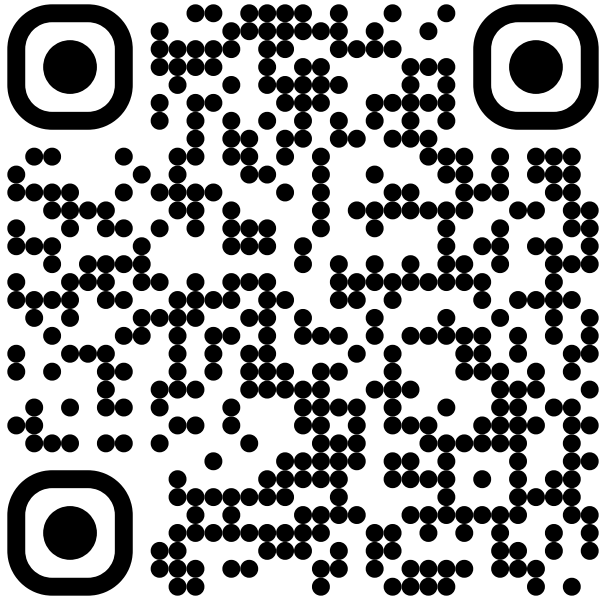
1. **Accuracy is misleading** for imbalanced data — it can make a useless model look good
2. The **confusion matrix** reveals what accuracy hides
3. **Precision** measures trustworthiness; **Recall** measures completeness
4. **MCC** is the most reliable single metric for imbalanced classification
5. **Pipelines** automate correct methodology and prevent data leakage
6. **Hyperparameters** control how models learn — tuning them matters
7. **Bayesian optimisation** finds good settings far more efficiently than brute force
8. **Threshold optimisation** and **ensembles** can further improve results

Questions Before CE2?

Up Next: Coding Exercise 2

You'll build a complete, production-quality ML pipeline!

- Expect it to take longer than CE1
- Read the detailed documentation in the notebook
- Experiment with different configurations
- Compare metrics to understand their behaviour



There are 3 versions of CE2: simple, advanced, expert - they build complexity from a functional starting point.

> **Remember:** This is what real ML engineering looks like. Master these techniques and you can tackle any tabular ML problem.

Resources and Documentation

Metrics:

- sklearn Metrics Guide
- MCC Wikipedia

Pipelines:

- sklearn Pipeline User Guide
- ColumnTransformer

Optuna:

- Optuna Documentation
- Optuna Tutorial