Anomaly Detection: Spotting the Unusual in Data

From Intuition and Time Series to a Streaming Detector

Stephen Hailes

9 October 2025

Department of Computer Science, UCL

Session Plan

Introduction: What & Why

Foundations of Detection

Bioreactor Simulator Context

Getting Live Data via MQTT

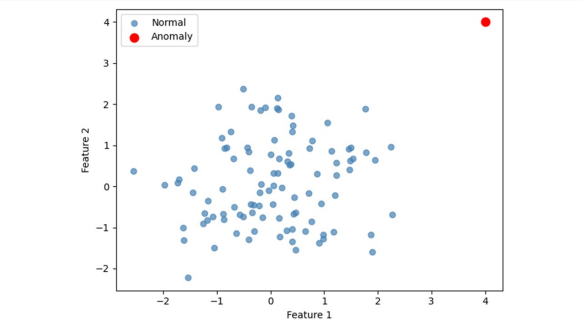
Wrap-Up

Introduction: What & Why

What is Anomaly Detection?

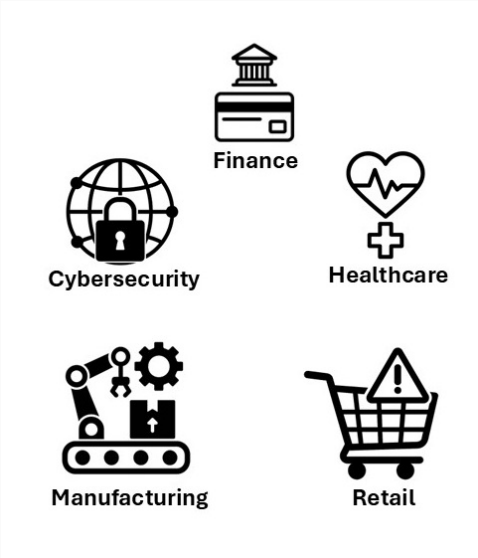
Simple definition

Anomaly detection is the identification of data points or patterns that deviate significantly from expected or typical behaviour.

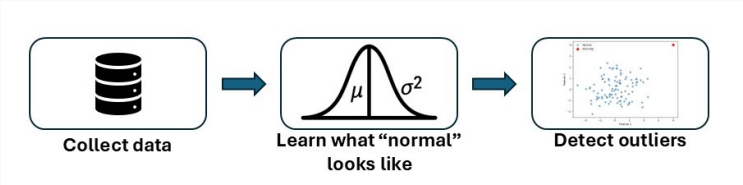


Where is Anomaly Detection Used?

- Finance — detect fraudulent activity
- Healthcare — rare diseases or abnormal readings
- Cybersecurity — intrusion and malware detection
- Manufacturing — predictive fault monitoring
- Retail — unusual purchasing behaviour



How Does It Work?



1. **Collect** representative data from sensors or logs (or MQTT in our case)
2. **Learn** what "normal" looks like — statistically or via models.
3. **Detect** deviations from this learned normality.

Popular Methods (Simplified)

- **Statistical:** use deviations from expected mean/variance (e.g. z-scores).
- **Model based time series:** ARMA/ARIMA, state-space or Kalman; predict $\hat{x}_t$ and flag large residuals.
- **Machine learning:** classifiers or clustering on features (supervised or unsupervised).
- **Deep learning:** sequence models and reconstruction methods (autoencoders, RNNs, 1D CNNs) that learn normal patterns.

Challenges in Anomaly Detection

- Defining “normal” accurately.
- Defining "anomalous" accurately: anomalous data may start to appear later in time than the start of (or even the end of) the fault
- Few anomaly examples $\rightarrow$ unbalanced data.
- False alarms vs missed detections.
- Concept drift — systems change over time.

Why Does It Matter?

- Safety — catch abnormal behaviour before accidents occur.
- Reliability — maintain stable operation and product quality.
- Preventative rather than reactive maintenance
- Cost — reduce downtime and material waste.
- Security — detect cyber-physical manipulation.

The Economics of Anomalies (2024–25 snapshot)

- Global 500 firms lose **$1.4 trillion / year** to unplanned downtime (≈ 11% of revenue). [Siemens 2024: https://tinyurl.com/bde5ndar]
- Large plants average **$253 million / year** in downtime loss. [Siemens 2024: https://tinyurl.com/bde5ndar]
- Cost per hour ranges: **$25 k average – $500 k +** [MaintainX 2024: https://tinyurl.com/3esdcfj8]; **$2.3 million** for automotive [Siemens 2024: https://tinyurl.com/bde5ndar]
- Anomaly-detection market ≈ **$5.5 – 6.2 B (2024–25)**, CAGR **12 – 18 %**, projected **$14 B by 2030**. [TBRC 2025: https://tinyurl.com/26qogq3z; GVR 2023: https://tinyurl.com/24rvl8ql]

Types of Anomalies

- **Point anomaly:** single measurement deviating strongly from baseline.
- **Contextual anomaly:** value abnormal only within its context (e.g. time of day).
- **Collective anomaly:** a subsequence of otherwise normal points forming an abnormal pattern.

Foundations of Detection

Time Series Basics

A time series is a sequence $\{x_t\}_{t=1}^{T}$ sampled in time.

- Level and spread: $\mu = \mathbb{E}[x], \quad \sigma^2 = \mathrm{Var}(x)$
- Dependence: $\rho(k) = \mathrm{corr}(x_t, x_{t-k})$
- Stationarity: mean and variance stable over time

What Is a Decision Rule?

- A **decision rule** is the rule that turns a continuous-valued score into a final decision.
- It compares the score to a chosen **threshold**:

  At each time t : if $s(x_t) \geq \tau$ then label as anomaly; otherwise normal.

- The threshold τ controls how strict the detector is:
  - High $\tau \rightarrow$ fewer alarms, miss more anomalies.
  - Low $\tau \rightarrow$ more alarms, suffer more false alarms.
- Choosing τ defines the operating point of the detector.

Detecting an anomaly

Single feature z-score

$$z_{t,i} = \frac{x_{t,i} - \mu_{t,i}}{\sigma_{t,i}}$$

Pooled score across d features

$$s_t(x_t) = \sqrt{\sum_{i=1}^{d} \left(\frac{x_{t,i} - \mu_{t,i}}{\sigma_{t,i}} \right)^2}$$

where $\mu_{t,i}$ and $\sigma_{t,i}$ are time-varying baselines estimated from recent history or a seasonal model.

Decision rule

Anomaly $\Leftrightarrow s(x_t) \geq \tau$

Robustness & Redundancy

- Use robust statistics when data vary over time or contain outliers:

$$
\begin{aligned}
m_t &= \mathrm{median}(x_t) \\
\mathrm{MAD}_t &= \mathrm{median}(|x_t - m_t|) \\
\sigma_t &\approx 1.4826 \times \mathrm{MAD}_t \\
\text{modified-Z}_{t,i} &= \frac{0.6745\,(x_{t,i} - m_t)}{\mathrm{MAD}_t}
\end{aligned}
$$

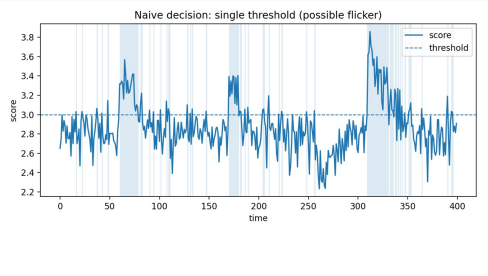
- Avoid redundant or correlated features.
- Handle missing values gracefully.

Naive Thresholding

- A simple rule:

  If $s(x_t) \geq \tau$ then alarm, else normal.

- But... real data are noisy – scores often fluctuate around the decision threshold τ.
- Problem: when $s(x)$ hovers near τ, the alarm can flicker on and off.
- This instability makes alarms unreliable and distracting in practice.



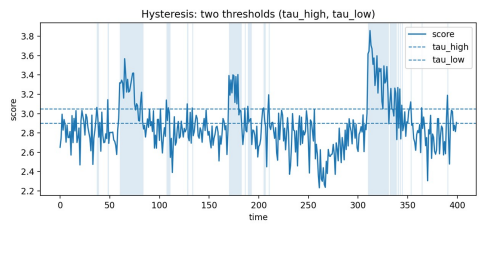
Single threshold – alarm flickers as score crosses τ repeatedly.

Hysteresis: Two Thresholds

- Adds stability by defining two thresholds:

$$\text{Raise alarm if } s(x_t) \geq \tau_{\text{high}},$$
$$\text{Clear alarm if } s(x_t) \leq \tau_{\text{low}}.$$

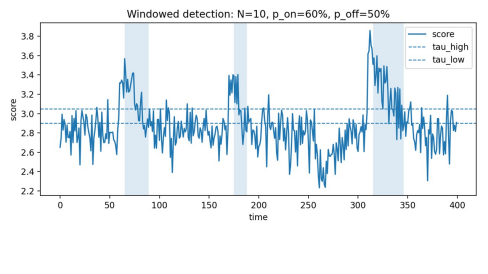
- The gap $\tau_{\text{high}} - \tau_{\text{low}}$ is the hysteresis band.
- Prevents frequent switching when the score fluctuates near the boundary.



Two-threshold hysteresis – alarm changes state only outside the band.

Windowed Detection

- Applies a **time-windowed rule** rather than relying on individual points.
- Evaluate the last N samples in a sliding window.
- Raise or clear the alarm based on the **proportion** of points crossing the thresholds:
  - Raise alarm if a large enough fraction of recent scores exceed τ_{high}.
  - Clear alarm if most recent scores fall below τ_{low}.
- Smooths the decision signal and ignores brief transients or outliers.



Windowed detection – alarm depends on the proportion of high scores within a time window.

Model-Based Detection: ARMA and ARIMA Methods (Advanced)

Basic thresholding works for steady signals, but more complex systems require **models** of normal (i.e. expected) dynamics.

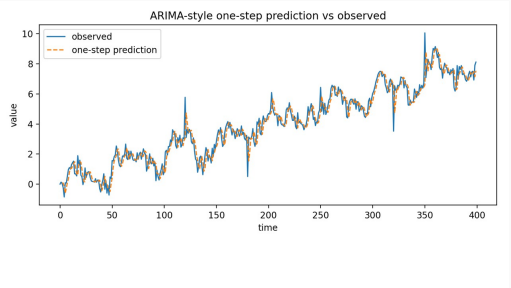
- ARMA or ARIMA models capture temporal dependencies in data[1].
- The detector predicts the next value $\hat{x}_t$ and compares it to observed value x_t.

$$e_t = x_t - \hat{x}_t$$

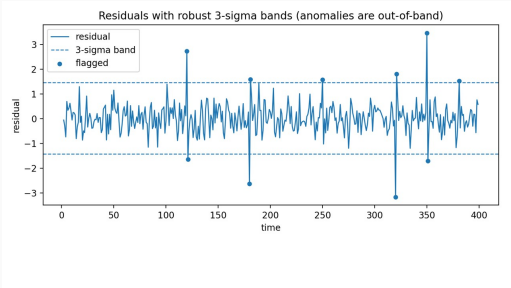
- Large residuals $|e_t|$ indicate behaviour that deviates from the model – potential anomalies.
- These models adapt to trends, noise, and seasonality, reducing false alarms in drifting systems.

[1] ARMA = AutoRegressive Moving Average; ARIMA = AutoRegressive Integrated Moving Average

ARIMA-style Detection: Prediction and Residuals



One-step prediction vs observed



Residuals with robust 3-sigma bands

Assessing Detector Performance

- We need objective measures of how well a detector distinguishes normal from faulty behaviour.
- Good detectors should:
  - Raise alarms when faults occur.
  - Stay quiet when systems behave normally.
- Metrics must capture both correctness and completeness of detection.

- Note: detectors should also raise alarms *quickly* but that's for another day

Precision: How Reliable Are Our Alarms?

Precision: "When I raise an alarm, how often am I right?"

- High precision means few false alarms.
- It measures the **reliability** or **purity** of detected events.
- Example: 10 alarms raised, 8 genuine faults $\rightarrow$ precision $= 0.8$.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- TP – true positives (correct alarms)
- FP – false positives (false alarms)

Recall: How Complete Is Our Detection?

Recall: "Of all real faults, how many did I detect?"

- High recall means few missed faults.
- It measures the **completeness** or **sensitivity** of the detector.
- Example: 12 faults occurred, 8 detected $\rightarrow$ recall $= 0.67$.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

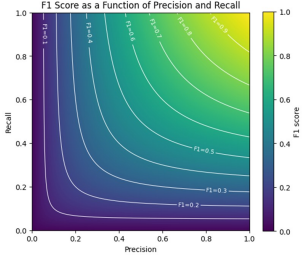
- TP – true positives (correct alarms)
- FN – false negatives (missed faults)

F_1 Score: Balancing Precision and Recall

F_1 combines precision and recall into a single performance measure

- High only when both are high. If either precision or recall is low, F_1 falls sharply.
- $$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$
- Example values:

$$P = 0.8,\ R = 0.8 \Rightarrow F_1 = 0.8$$
$$P = 0.9,\ R = 0.3 \Rightarrow F_1 = 0.45$$
$$P = 0.4,\ R = 0.9 \Rightarrow F_1 \approx 0.55$$



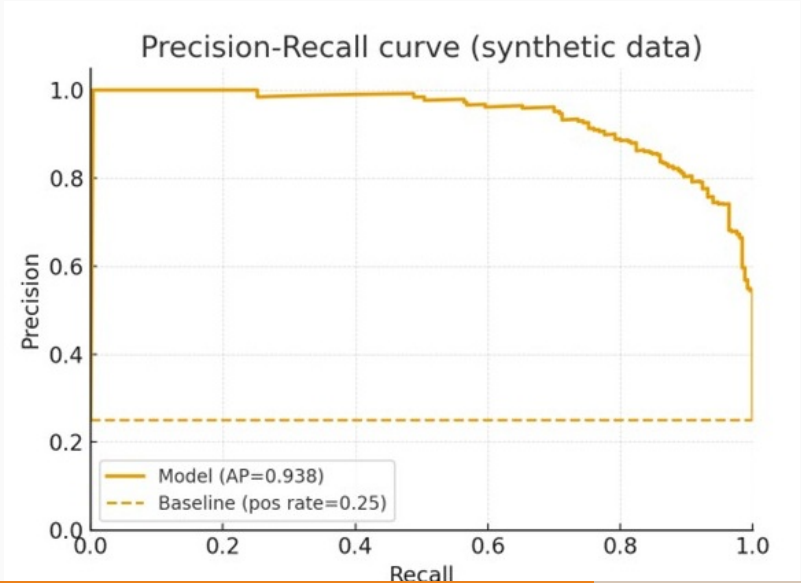
F_1 score surface – colour indicates F_1 for combinations of precision and recall.

Creating a Precision–Recall Curve

A Precision–Recall curve shows how precision and recall change as we vary the decision threshold.

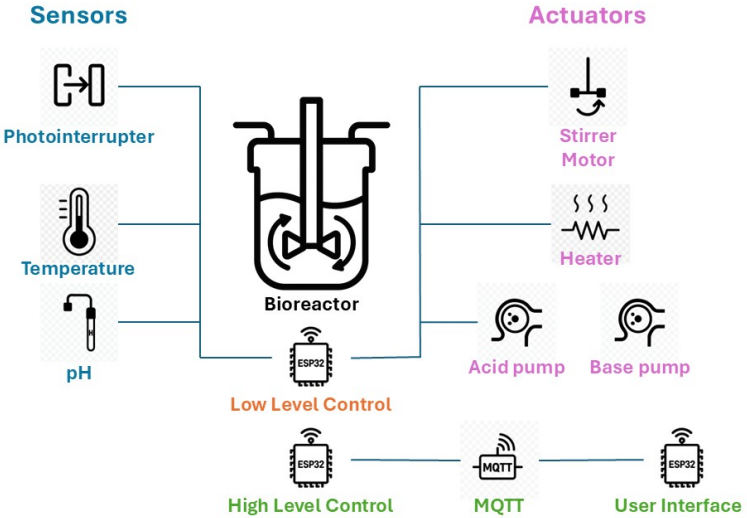
- To build the curve:
  1. Sort all samples by their predicted score (highest first).
  2. For each possible threshold:
     - Label points above the threshold as *anomalies*.
     - Compute TP, FP, FN, then precision and recall.
  3. Plot precision (vertical axis) against recall (horizontal axis).
- The **area under the curve (AUC/AP)** summarises overall performance.
- A good detector maintains high precision even as recall increases.

Precision–Recall Curve

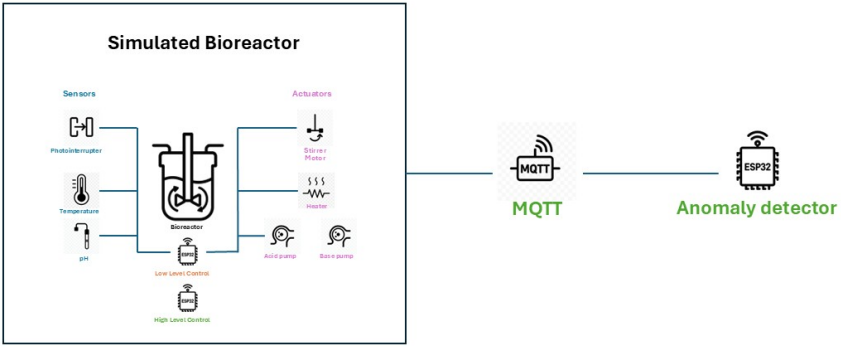


Bioreactor Simulator Context

The bioreactor



The simulator



The simulator

- The simulator is a black box as far as you are concerned.
- It is code that simulates your bioreactor[2] and has:
  - Sensors: temperature, pH, RPM
  - Actuators: heater, stirrer motor, acid/base dosing
  - A PID controller, whose parameters have been tuned
- The simulator publishes **summary JSON** over MQTT once per second.
- These messages summarise the conditions of the bioreactor over the last second
- In the simulator:
  - We inject noise
  - Faults of different types can be injected, at different points in time.
  - The setpoints can be changed - either on a schedule, or at random.
  - Fault labels can be reported ... or not.

[2] This is done carefully; the simulator uses principled physical and chemical models of the real world

Your (your team's) task

- Detect the faults!
- This is progressive - I will publish several streams
  - No faults
  - Labelled faults
  - Unlabelled faults
- In several levels of difficulty[3]
  - Fixed setpoints, no fault
  - Fixed setpoints, one fault
  - Fixed setpoints, multiple faults
  - Varying setpoints, no faults
  - Varying setpoints, one fault
  - Varying setpoints, multiple faults

[3] Details will be on Moodle when Challenge 2 starts

Detector Workflow

1. **Train:** use fault-free data to compute baseline μ_i, σ_i.
2. **Test (labelled):** evaluate predictions vs true labels to tune τ.
3. **Test (unlabelled):** deploy detector; interpret alerts without ground truth.

Getting Live Data via MQTT

ESP32 MQTT Client with PicoMQTT

- Demonstration of a minimal ESP32 MQTT client using:
  - `WiFi.h` - to connect to the wireless network
  - `PicoMQTT.h` - lightweight MQTT client library
- Connects to `broker.hivemq.com`
- Subscribes to a telemetry topic and prints messages
- You should install the PicoMQTT library before compiling the code

Minimal PicoMQTT Client

```
#include <WiFi.h>
#include <PicoMQTT.h>

const char* WIFI_SSID = "YOUR_SSID";
const char* WIFI_PASSWORD = "YOUR_PASS";
const char* MQTT_HOST = "broker.hivemq.com";
const uint16_t MQTT_PORT = 1883;
const char* TOPIC = "bioreactor_sim/train/telemetry/summary";

PicoMQTT::Client mqtt(MQTT_HOST, MQTT_PORT);
```

Message handler

```
void onMessage(const char* topic, const char* payload) {
  Serial.print("[MQTT] ");
  Serial.print(topic);
  Serial.print(": ");
  if (payload) {
    Serial.println(payload);
  } else {
    Serial.println("(null)");
  }
}
```

Setup and subscription

```
void setup() {
  Serial.begin(115200);

  // Connect Wi-Fi
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  while (WiFi.status() != WL_CONNECTED) {
    delay(250);
    Serial.print('.');
  }
  Serial.println("\nWi-Fi connected");

  // Subscribe once - PicoMQTT auto re-subscribes after reconnects
  mqtt.subscribe(TOPIC, onMessage);
  mqtt.begin(); // start the MQTT client
}
```

Main Loop

```
1 void loop() {
2   mqtt.loop(); // manages reconnections and messages
3 }
```

Summary

- PicoMQTT simplifies ESP32 MQTT code drastically.
- Automatic reconnects and re-subscriptions reduce complexity.
- No payload size limit - suitable for JSON telemetry.
- Ideal for quick prototypes.

Understanding the MQTT Stream

Telemetry summary message (JSON):

```
{"window": {"start": 1760286536, "end": 1760286537, "seconds": 1, "samples": 11},

"temperature_C": {"mean": 22.37, "min": 22.35, "max": 22.40},

"pH": {"mean": 7.07, "min": 6.90, "max": 7.14},

"rpm": {"mean": 291.31, "min": 274.17, "max": 307.03},

"actuators_avg": {"heater_pwm": 0.99, "motor_pwm": 0.32, "acid_pwm": 0.0050, "base_pwm": 0.00038}, "dosing_L": {"acid": 3.35e-05, "base": 1.79e-05}, "heater_energy_Wh": 0.039, "photoevents": 11,

"setpoints": {"temperature_C": 35.0, "pH": 6.9, "rpm": 300.0},

"faults": {"last_active": [], "counts": {}}}
```

Key Points:

- One JSON message per second.
- `window`: time range (start, end, seconds).
- `temperature_C`, `pH`, `rpm`: mean/min/max values.
- `actuators_avg`: PWM duty cycles.
- `faults`: appears only when labels are enabled.

Wrap-Up

Key Takeaways

- Defined anomalies and their importance across domains.
- Learned simple statistical tools: mean, variance, z-scores, thresholds.
- Explored robustness, drift, and evaluation metrics.
- Connected theory to practice via the bioreactor and MQTT streaming.

Questions?