

# **Machine Learning for Fault Detection in Water Distribution Systems**

## Summary

This project addresses the problem of fault and leak detection in modern water distribution systems using machine learning. Traditional manual monitoring techniques are slow, error-prone, and inadequate for identifying subtle system failures in real time. Using sensor and actuator data from the WADI dataset (iTrust, Singapore University of Technology and Design), I built supervised machine learning models to automatically classify system states as either normal or defective.

I trained and compared two classifiers, Random Forest and XGBoost, on precision, recall, F1-score, and ROC-AUC. Both models achieved near-perfect recall and precision, with Random Forest slightly outperforming XGBoost on false positives. The models were also analyzed for feature importance and sensor correlation to gain insights into fault indicators.

## Problem Statement

Modern Water Distribution Systems rely on computers, sensors and actuators for both monitoring and operational purposes. This combination of physical processes and embedded systems (cyber-physical systems, in short) improves the level of service of water distribution networks but exposes them to the potential threats of cyber attacks<sup>[1]</sup>. During the past decade, several water supply and distribution systems have been attacked, with the consequent creation of cyber-security agencies and international partnerships to defend water networks. Water distribution systems face operational and safety challenges due to undetected leaks or failures, leading to:

- Significant water loss
- Infrastructure degradation
- Operational inefficiencies

Manual inspection and rule-based alert systems are not fast or reliable enough for timely detection. I aim to solve this using supervised machine learning that can learn from labelled historical data and classify system state in real time based on sensor readings.

The goal is to classify each 1-second time step as:

- 0 → Normal
- 1 → Faulty/Attack (Leak, Pump failure, etc.)

## Existing Tools, Techniques, and Datasets for Water Infrastructure Fault Detection

### Tools and Techniques

Fault detection in water distribution systems is an active area of research and industrial automation. Some traditional and modern methods include:

- **SCADA-based monitoring systems**  
Widely used in industry for real-time data collection and rule-based alerting. However, these

systems are often limited to predefined thresholds and cannot adapt to new or subtle failure patterns.

- **Hydraulic simulation models (e.g., EPANET)**  
These simulate pressure, flow, and tank levels to detect inconsistencies, but require accurate calibration and can't handle real-time noise well.
- **Rule-based anomaly detection**  
Predefined rules (e.g., "if flow drops below X") are easy to implement but can't generalize or detect complex fault patterns.

## Datasets

There are many datasets available for faults in water distribution networks, some of the most popular include:

- **WADI (Water Distribution Dataset)** – *Used in this project*
  - Created by iTrust Labs, SUTD
  - Rich sensor and actuator data with real attacks
  - Ideal for supervised learning, anomaly detection, and ICS security research
- **SWaT (Secure Water Treatment Dataset)** – *Related ICS dataset*
  - Another iTrust dataset simulating a water treatment plant
  - Often used in anomaly detection and cyber-physical attack research
- **EPANET-based Synthetic Datasets**
  - Used for generating fault simulation data based on hydraulic models
  - Requires domain-specific tuning; lacks real-world noise and actuator data

## ML Models

This is not a new problem and a lot of previous work has been done on using ML for fault detection in water distribution systems. There even has been a competition, BATtle of the Attack Detection Algorithms (BATADAL) to objectively compare the performance of algorithms for the detection of cyber attacks in water distribution systems. Some of the techniques include:

- **Unsupervised anomaly detection:** Isolation Forests, Autoencoders, and PCA
- **Time-series forecasting:** LSTM, GRU for predicting expected behavior and comparing with actual readings
- **Supervised classification** (as done here): Decision Trees, SVMs, Random Forests, and Boosting models
- **Hybrid cyber-physical models:** Combine simulations with sensor data to improve fault diagnosis

## My Approach

While industrial systems rely heavily on SCADA and rule-based monitoring, the availability of labeled datasets like WADI has opened the door to data-driven, supervised machine learning approaches that offer better fault detection performance with minimal manual intervention.

This project builds upon such modern approaches by combining real-world sensor data with ensemble learning techniques (Random Forest, XGBoost) to achieve highly accurate fault classification. I will be using pre-existing Python libraries, scikit-learn for the Random Forest classifier implementation and xgboost for the XGBoost classifier implementation respectively.

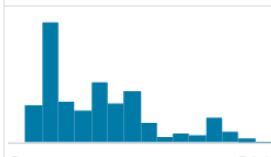
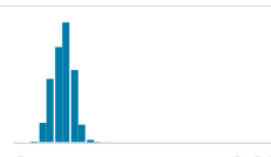
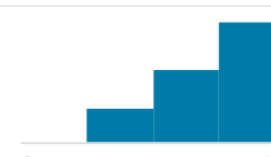
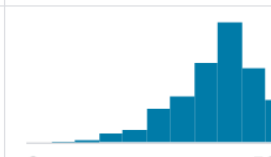
## Data Preprocessing

I got the WADI dataset from Kaggle<sup>[2]</sup>. Kaggle provides tools to visualize the data in the browser itself, and I used these to manually inspect and understand the dataset. A snippet is shown below:

### WADI\_data



Data Card   Code (1)   Discussion (0)   Suggestions (0)

Detail   Compact   Column				10 of 130 columns ▾	
# 1_AIT_001_PV ▮	# 1_AIT_002_PV ▮	# 1_AIT_003_PV ▮	# 1_AIT_004_PV ▮		
					
0	0	0	0		
214	2.06	12	527		
171.155	0.619473	11.5759	504.645		
171.155	0.619473	11.5759	504.645		
171.155	0.619473	11.5759	504.645		
171.155	0.607477	11.5725	504.673		
171.155	0.607477	11.5725	504.673		
171.155	0.607477	11.5725	504.673		
171.155	0.607477	11.5725	504.673		

The WADI dataset required significant cleaning and preprocessing to prepare it for supervised learning. Below is a summary of each preprocessing step and its rationale.

## File Handling and Label Creation

- Loaded two separate files:
  - WADI\_14days\_new.csv (normal operation) — labeled 0
  - WADI\_attackdataLABEL.csv (with attack periods) — extracted from "Attack LABEL" column and mapped:  
-1 → 1 (Fault), 1 → 0 (Normal)
- Stripped whitespaces from all column names to prevent mismatches ('Date ' vs 'Date')

## Column Cleanup

- Dropped metadata columns not useful for ML ('Row', 'Date', 'Time')
- Concatenated both datasets (normal and attack) into a unified dataframe with a Label column indicating the system state.

## Handling Missing Data

- Applied forward-fill (ffill()) to fill missing values using previous time steps
- Dropped:
  - Columns where all values were NaN
  - Rows where more than 10% of values were NaN
- Aligned labels (y) with the cleaned feature matrix (X)

## Train/Test Split

The data is split into train (80% of full dataset) and test (20%) sets. The train/test split is stratified to ensure that the proportion of normal vs. faulty samples in the training and test sets is the same as in the original dataset.

This is important since the original data is highly imbalanced (~1% faulty, ~99% normal). Without stratification, the test set might contain too few or too many faulty samples, leading to misleading evaluation metrics. Stratification ensures both train and test sets are representative of the real-world class distribution.

## Final Result

- **Total rows after preprocessing:** 957,374
- **Remaining features:** 123
- **Class distribution:**
  - Normal: 947,397 samples
  - Fault: 9,977 samples (~1.04%)
- **Data split:**
  - 80% training (stratified), 20% test set

## Modelling Approach

The fault classification task is framed as a binary supervised classification problem where each second of sensor readings is labelled as either:

- 0 → Normal
- 1 → Fault (leak, pump failure, actuator issue, etc.)

### Why Supervised Learning?

- Labels are available from WADI\_attackdataLABEL.csv, and the other csv has only normal data
- Faults are real and manually injected, providing reliable ground truth
- High interpretability and evaluation capability compared to unsupervised methods

### Models

#### Random Forest Classifier

- Robust to noise and nonlinearity
- Naturally handles unscaled features and missing data
- Interpretable via feature importance
- Class imbalance handled using
  - `class_weight='balanced'`

#### XGBoost Classifier

- Gradient boosting model known for high accuracy
- Handles class imbalance via
  - `scale_pos_weight = (normal samples) / (faulty samples)`
- Tuned with
  - `eval_metric='logloss'`
  - `random_state=42`
  - `n_jobs=-1`

Both models were trained on the same stratified dataset, predictions generated, and performance was then evaluated using precision, recall, F1-score, confusion matrix, and ROC-AUC.

## Metrics for Model Evaluation

Given the extreme class imbalance in the dataset (faults represent only ~1% of all cases), traditional accuracy is not a sufficient measure, since a model could achieve 99% accuracy just by predicting everything as normal while completely missing all actual faults. Instead, I focused on the following metrics.

### Key Metrics Used

- **Precision** – How many predicted faults were actually faults? Prevents false alarms.
- **Recall** – How many real faults did we correctly detect? Critical to ensure no real faults are missed.
- **F1-Score** – Harmonic mean of precision and recall. Best overall indicator for imbalanced problems.
- **Confusion Matrix** – Shows exact counts of true positives, false positives, etc.
- **ROC-AUC Score (area under the receiver operating characteristic curve)** – Indicates the classifier's ability to distinguish between classes across thresholds.
- **Precision-Recall Curve** – Especially valuable for imbalanced datasets like this, showing how recall drops off as precision improves.

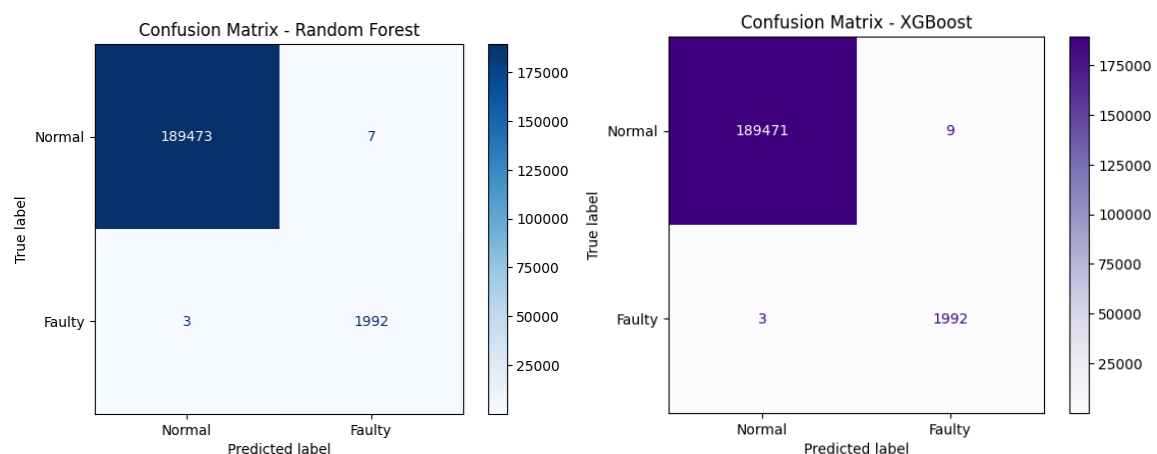
Metric	Definition	Goal
Accuracy	$(TP + TN) / (TP + FP + TN + FN)$	Overall correctness
Precision	$TP / (TP + FP)$	Avoid false alarms
Recall	$TP / (TP + FN)$	Detect all actual faults
F1-Score	$2 \times (Precision \times Recall) / (P + R)$	Balance of precision & recall
ROC-AUC	Area under ROC curve	Distinguish between classes

## Results & Model Comparison

I evaluated both models on the same test set of ~191,000 rows (with balanced class proportions via stratification). Below is a summary of the results:

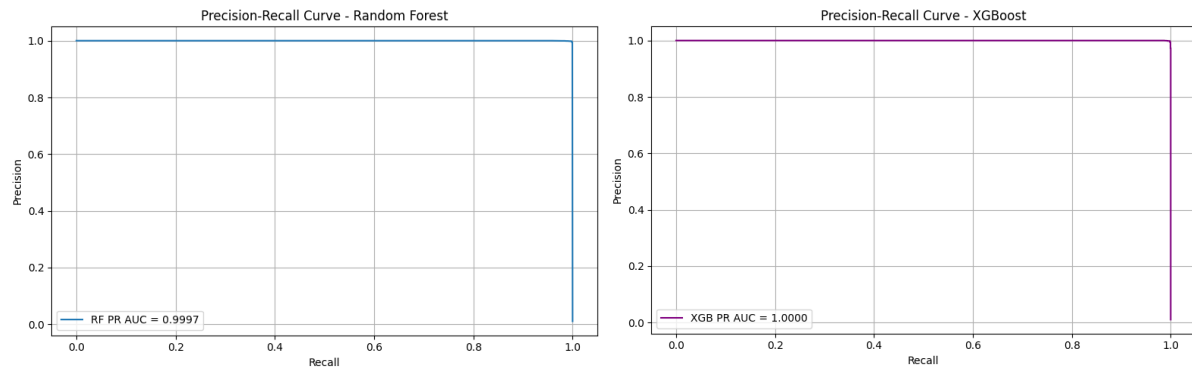
Metric	Random Forest Classifier	XGBoost Classifier
Precision	<b>0.9965</b>	0.9955
Recall	0.9985	0.9985
F1-Score	<b>0.9975</b>	0.9970
ROC-AUC	0.9992	0.9992
False Alarms	<b>7</b>	9
Missed Faults	3	3

While the two models perform well and have very similar recall of 0.9985, the Random Forest classifier beats out the XGBoost model with slightly higher precision and F1-score, due to a lower false positive rate. ROC-AUC scores of 0.9992 for both models indicate excellent separability. Further statistics on classification are present in the Python Notebook under “Classification Report” for both the models. Some visualizations of model performance:



**Confusion Matrices**





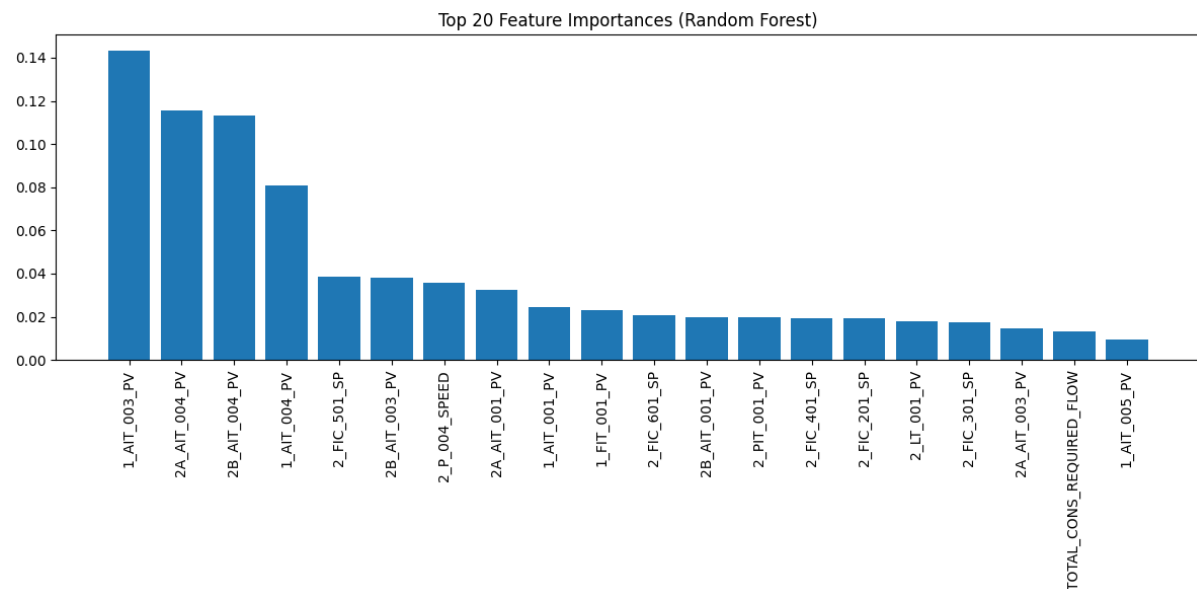
*Precision Recall Curves*

## Feature Analysis

I analysed feature (sensors or actuators) importances from both models, and also visualized correlations among top features.

### Top Features by Importance

#### Random Forest

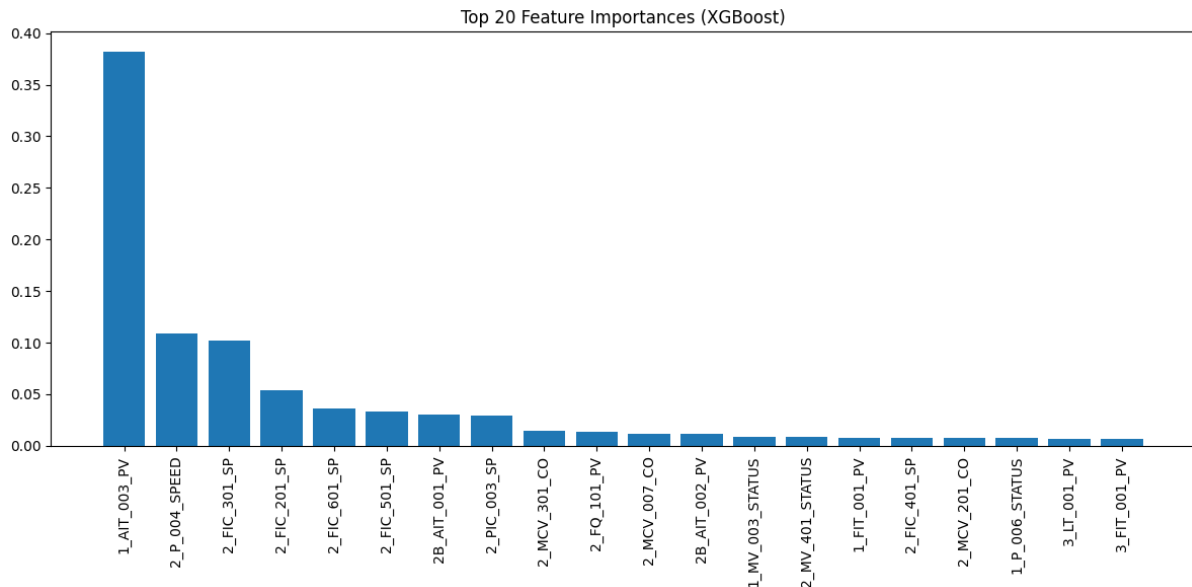


The 20 most important features (based on the model's Gini impurity reductions) were primarily from:

- Flow sensors (FIT)
- Pressure indicators (PIT, PIC)
- Valve statuses (MV\_001\_STATUS, etc.)
- Derived pressure metrics like LEAK\_DIFF\_PRESSURE

These features often showed abrupt changes during fault periods and were thus reliable indicators for the model.

## XGBoost

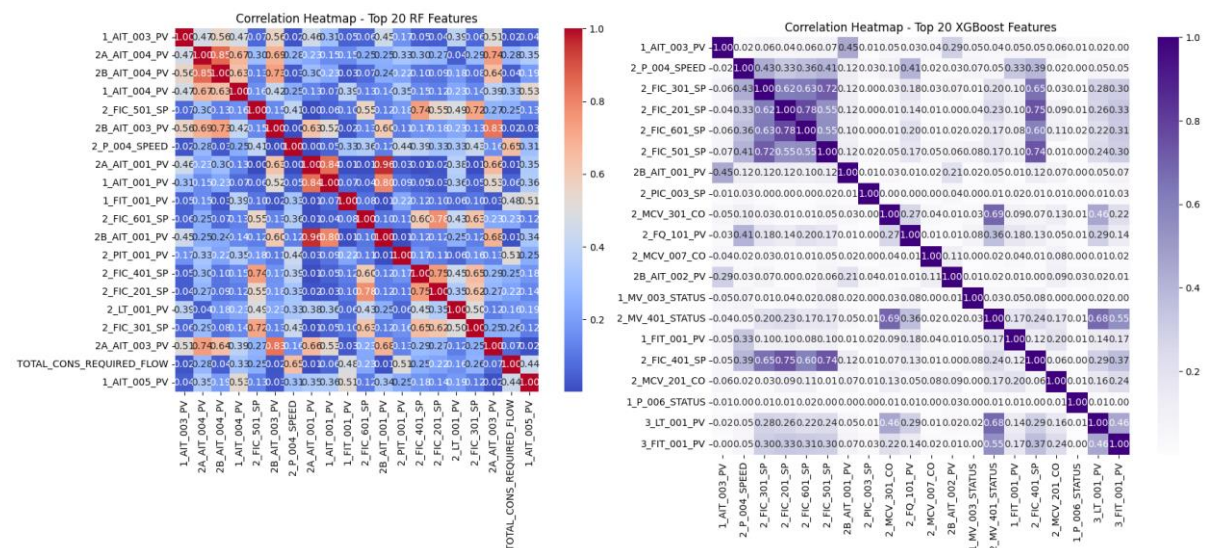


XGBoost's top features largely overlapped with those from Random Forest, confirming consistency across models. However, XGBoost placed slightly higher weight on:

- Level sensors (LIT, LT)
- Motor speeds (P\_SPEED)
- Actuator control values (MV, P\_STATUS)

This suggests that XGBoost might be capturing pattern shifts in addition to raw sensor deviations.

## Sensor Redundancy and Correlation



Observations from the correlation heatmaps for the two models:

- Several flow sensors and valve statuses were highly correlated, likely because they monitor the same physical subsystem or operate together.
- Some pressure and level sensors showed low or no correlation, making them more informative individually.
- Redundant features (those with 0.95+ correlation) can potentially be pruned or combined in future models without significant performance loss.

## Conclusion and Future Work

This project demonstrated that supervised machine learning, specifically Random Forest and XGBoost, can accurately detect faults in water distribution systems using real-time sensor and actuator data. Both models achieved near-perfect precision and recall, with Random Forest slightly outperforming XGBoost on false positives. Key features driving model decisions included flow rates, pressure readings, and actuator states.

To further improve performance and generalization, future efforts can focus on:

- Incorporating temporal patterns using time-windowed features or sequence models (e.g., LSTM)
- Reducing redundancy through feature selection or dimensionality reduction
- Collecting more diverse labelled data, especially edge cases, early-stage faults, and recovery periods
- Engineering new features like sensor deltas, trends, or inter-sensor ratios

These steps will help make the model more robust, interpretable, and ready for real-world deployment.

## Code and Data Availability

The code and data can be found on:

- GitHub (only zip of data): <https://github.com/hridaik/wadi-fault-detection>
- Google Drive (inflated CSVs included):  
<https://drive.google.com/drive/folders/1UBarffwmWhFChHmrzXOhz3i1OZEIKZsG?usp=sharing>

## References

1. BATtle of the Attack Detection ALgorithms (BATADAL), <https://www.batadal.net>
2. WADI Data, Giovanni Monco, <https://www.kaggle.com/datasets/giovannimonco/wadi-data>