Isolation Forest

Tree-based anomaly detection algorithm

Key Principle: Anomalies are easier to isolate (require fewer splits)

Algorithm Process

1. Build ensemble of random trees
2. Use random feature splits
3. Compute average path length
4. Calculate anomaly score

Anomaly Scoring

Anomalies

Normal Points

Key Parameters

n_estimators
Number of trees in the ensemble

contamination
Expected ratio of anomalies in dataset

Advantages

- ✓ No distribution assumptions
- ✓ Handles high dimensions
- ✓ Efficient and scalable
- ✓ Works on large datasets

Time Complexity

`O(n log n)`

Shorter paths
Easy to isolate

Longer paths
Hard to isolate

Scales to large datasets

Isolation Process: Path Length Comparison

Why anomalies are easier to isolate

Normal Data Point

Dense data around the point

Split 1

Split 2

Split 3

Split 4

Split 5

Split 6

Path Length

6

Anomaly Point

Few data points nearby

Split 1

Split 2

Path Length

2

Key Insight

Anomalies are isolated with **fewer node splits** → Shorter path length → **Higher anomaly score**

📐 Anomaly Score Formula

Normalizing path length to 0~1 range

Formula

$$s(x, n) = 2^{-E(h(x))/c(n)}$$

$$c(n) = 2H(n-1) - 2(n-1)/n$$

$$H(i) = \ln(i) + 0.5772 \text{ (Euler's constant)}$$

x
Specific data point

n
Total number of data points

h(x)
Path length of data point x

E[h(x)]
Average of h(x) across all trees

c(n)
Average path length of all data (normalization constant)

📊 Score Interpretation

⚠ Anomaly

$E[h(x)] \rightarrow 0$

✓ Normal Data

$E[h(x)] \rightarrow n-1$

$s(x, n) \to 2^{0}$

≈ 1.0

$s(x, n) \to 2^{-(n-1)}$

≈ 0.0

💡 **Note:** Score near 0.5 indicates ambiguous cases ($E[h(x)] \approx c(n)$)