# Anomaly Detection Notes

Charlie Abela[1*]

[1*]Department of Artificial Intelligence, University of Malta.

Corresponding author(s). E-mail(s): charlie.abela@um.edu.mt;

**Abstract**

The anomaly-detection material has been recast as a self-paced workshop so that you learn the methods by doing rather than just hearing about them. Instead of watching me step through slides, you will work directly with three guided notebooks that mirror the typical analyst's workflow: laying statistical foundations, moving into modern unsupervised algorithms, and finally opening the "black box" with explainability tools. Because each checkpoint pairs concise theory with runnable code and immediate visual feedback, you can see how a tweak in a hyper-parameter changes a precision–recall curve or why SHAP singles out a specific feature, insights that seldom stick when encountered only in lecture form. The format also lets you proceed at your own pace (or collaborate in pairs), pause to investigate side questions, and revisit sections later when you tackle research or thesis projects. In short, the workshop turns a passive topic into an active skill-building session, equipping you not just to recall the algorithms but to apply, tune, and justify them in real data-science settings.

## Contents

# 1 Introduction to Anomaly Detection

Anomaly detection refers to the problem of finding patterns in data that do not conform to expected behavior. An anomaly (or outlier) is an observation that deviates significantly from other observations, arousing suspicion that it was generated by a different mechanism.

Anomaly detection is critical in many domains:

- **Fraud Detection:** Identifying unusual financial transactions that might indicate fraudulent activity
- **Health Monitoring:** Detecting irregular patterns in medical data that could indicate a disease
- **Network Security:** Identifying suspicious network traffic patterns that may indicate a cyber attack
- **Manufacturing:** Finding defects in production processes
- **Environmental Monitoring:** Detecting anomalous events in climate or ecological data

Real-world examples of anomalies include:

| Domain | Anomaly instance |
|---|---|
| Finance | Sudden $9,999 charge on dormant card |
| Health | Irregular heart-rate spike in ECG |
| Cybersecurity | 50 login attempts from a new IP in 10s |

**Table 1** Examples of anomalies in different domains

# 2 Types of Anomalies

Anomalies can be classified into three main categories:

## 2.1 Point Anomalies

A point anomaly is a single instance of data that is anomalous with respect to the rest of the data (see Figure 1. For example, a credit card purchase with an unusually large amount compared to the usual spending pattern would be a point anomaly.

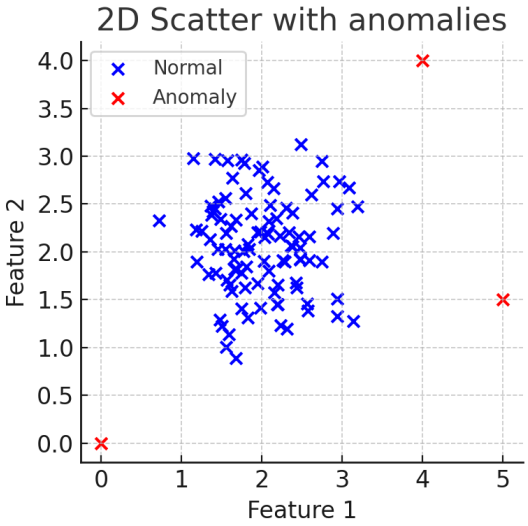


**Fig. 1** Example of **point anomalies** in a 2D feature space. Most observations (blue crosses) cluster around the center and represent normal behaviour. A few scattered observations (red crosses) deviate significantly from this distribution and are flagged as anomalies. These anomalies lie far from the high-density region and are typically detectable using distance-based or density-based methods.

Point anomalies are the simplest and most common type of anomalies. They're often referred to as outliers and are relatively easy to detect using statistical methods.

## 2.2 Contextual Anomalies

A contextual anomaly (also known as conditional anomaly) is an observation that is anomalous in a specific context but not otherwise. The context is typically defined by the structured nature of the dataset.

For example, 100°F temperature is normal in a desert afternoon but would be an anomaly in Antarctica. Or a high network traffic at 3 AM might be abnormal compared to typical nightly patterns.

## 2.3 Collective Anomalies

A collective anomaly occurs when a collection of related data points is anomalous with respect to the entire dataset, while individual data points may not be anomalous by themselves.

For example, a series of ECG measurements might individually appear normal, but together they may form an anomalous pattern indicating a heart condition. In network security, multiple login attempts across different locations might individually seem normal, but collectively form a suspicious pattern indicating a distributed attack.

# 3 Statistical Methods for Anomaly Detection

Statistical methods for anomaly detection involve defining a model of normality and then flagging observations that deviate from this model as potential anomalies. These methods are often simpler to implement and interpret compared to more complex machine learning approaches.

## 3.1 Z-Score Method (Univariate)

The Z-score measures how many standard deviations an observation is away from the mean. For a numeric feature, the z-score of a value $x$ is:

$$z = \frac{x - \mu}{\sigma} \tag{1}$$

where $\mu$ is the mean and $\sigma$ is the standard deviation of the population.

Points with $|z|$ above a threshold (commonly 3) are considered anomalies. This assumes data is approximately normally distributed. Under normal distribution, only about 0.3% of values have a $|z|$ greater than 3.

**Example:** In a dataset of body temperatures ($\mu \approx 98.6°\text{F},\ \sigma \approx 0.7°\text{F}$), a reading of 102°F has $z \approx +4.86$, which is far beyond normal variation, a likely anomaly (fever).

**Advantages:**

- Simple to understand and implement
- Provides an interpretable "outlier score"

**Disadvantages:**

- Not reliable if distribution is not normal
- Only looks at one feature at a time (univariate)
- Sensitive to outliers in the data used to compute mean and standard deviation

### 3.1.1 Robust Z-Score

A more robust version of the Z-score uses median and Median Absolute Deviation (MAD) instead of mean and standard deviation:

$$z = \frac{|x - \tilde{x}|}{\text{MAD}} \tag{2}$$

where $\tilde{x}$ is the median and MAD = Median Absolute Deviation.

This approach is less sensitive to the presence of outliers in the dataset.

## 3.2 Mahalanobis Distance (Multivariate)

Mahalanobis distance extends the concept of z-score to multiple dimensions, taking into account the correlations between variables. It measures how many standard deviations away a point is from the center of a multi-dimensional distribution.

For a multivariate observation $x$, the Mahalanobis distance is:

$$\text{MD}(x) = \sqrt{(x - \mu)^T \Sigma^{-1} (x - \mu)} \tag{3}$$

where $\mu$ is the mean vector and $\Sigma$ is the covariance matrix.

Unlike Euclidean distance, Mahalanobis distance is scale-invariant and accounts for the correlation between features. It effectively "whitens" the data by the covariance matrix, so correlated features don't inflate the distance.

For approximately normal data, the squared Mahalanobis distance follows a chi-square distribution. We can flag x as an outlier if MD(x) is very large (corresponding to a p-value below a threshold, e.g., 0.01).

**Example:** When considering height vs. weight, Euclidean distance might label a very tall, heavy person as far from average. Mahalanobis distance would recognize that height and weight are correlated – that person might not be an anomaly if their weight is proportional to height.

**Advantages:**

- Accounts for feature correlations
- Scale-invariant (not affected by different measurement scales)
- Has a statistical basis for thresholding (chi-square distribution)

**Disadvantages:**

- Requires sufficient data for stable covariance estimation (n » p)
- Sensitive to ill-conditioned covariance matrices
- Assumes elliptical data distribution (multivariate normal)

## 3.3 Implementation Example

Here's how to implement Z-Score and Mahalanobis Distance in Python:

```
import numpy as np
from scipy import stats
from scipy.spatial.distance import mahalanobis

# Assuming data is in numpy array X of shape (n_samples, n_features)
# 1) Z-scores for each value in a single feature (e.g., column 0)
z_scores = stats.zscore(X[:, 0])
outliers_z = np.where(np.abs(z_scores) > 3)

# 2) Mahalanobis distance for multivariate data X
mu = X.mean(axis=0)
cov = np.cov(X, rowvar=False)
inv_cov = np.linalg.inv(cov)

def mahalanobis_distance(row):
    diff = row - mu
    return np.sqrt(diff.T.dot(inv_cov).dot(diff))

md = np.apply_along_axis(mahalanobis_distance, 1, X)
outliers_md = np.where(md > np.percentile(md, 99))  # flag top 1% distances
```

## 3.4 Robust Implementation

For a more robust implementation of Mahalanobis distance that is less sensitive to outliers:

```
from sklearn.covariance import LedoitWolf
from scipy.spatial.distance import mahalanobis

```

```python
# Fit robust covariance on scaled data
cov_est = LedoitWolf().fit(X_scaled)
inv_cov = cov_est.get_precision()
mean_vec = X_scaled.mean(axis=0)

# Calculate Mahalanobis distances
mahal_d = np.array([mahalanobis(row, mean_vec, inv_cov) for row in X_scaled])

# Threshold top 0.1%
thresh_m = np.quantile(mahal_d, 0.999)
pred_m = (mahal_d > thresh_m).astype(int)
```

# 4 Machine Learning Methods for Anomaly Detection

While statistical methods work well for simple cases, more complex anomaly patterns often require advanced machine learning techniques. These methods can identify complex patterns and relationships in the data that statistical methods might miss.

## 4.1 Isolation Forest (IF)

Isolation Forest is an ensemble method specifically designed for anomaly detection. It's based on the principle that anomalies are "few and different," making them easier to isolate than normal points.

### 4.1.1 Intuition

Isolation Forest builds an ensemble of isolation trees. Each tree partitions the data by randomly selecting a feature and a split value. Anomalies typically need fewer partitions to be isolated-they end up having shorter path lengths in the trees (see Figure 2).

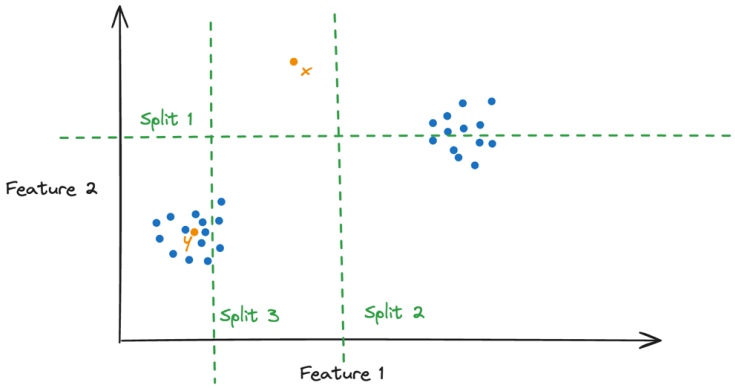


**Fig. 2** Illustration of the splitting mechanism in **Isolation Forest**. The green dashed lines represent random axis-aligned splits used to build isolation trees. Normal points (blue) are typically located in dense clusters and require more splits to isolate. In contrast, anomalous points (orange) lie in sparse regions of the feature space and are isolated in fewer steps. This difference in path length forms the basis of the anomaly score in Isolation Forest.

**Key insight:** If we randomly select features and split values to build a tree, anomalous points will typically be isolated with fewer splits than normal points.

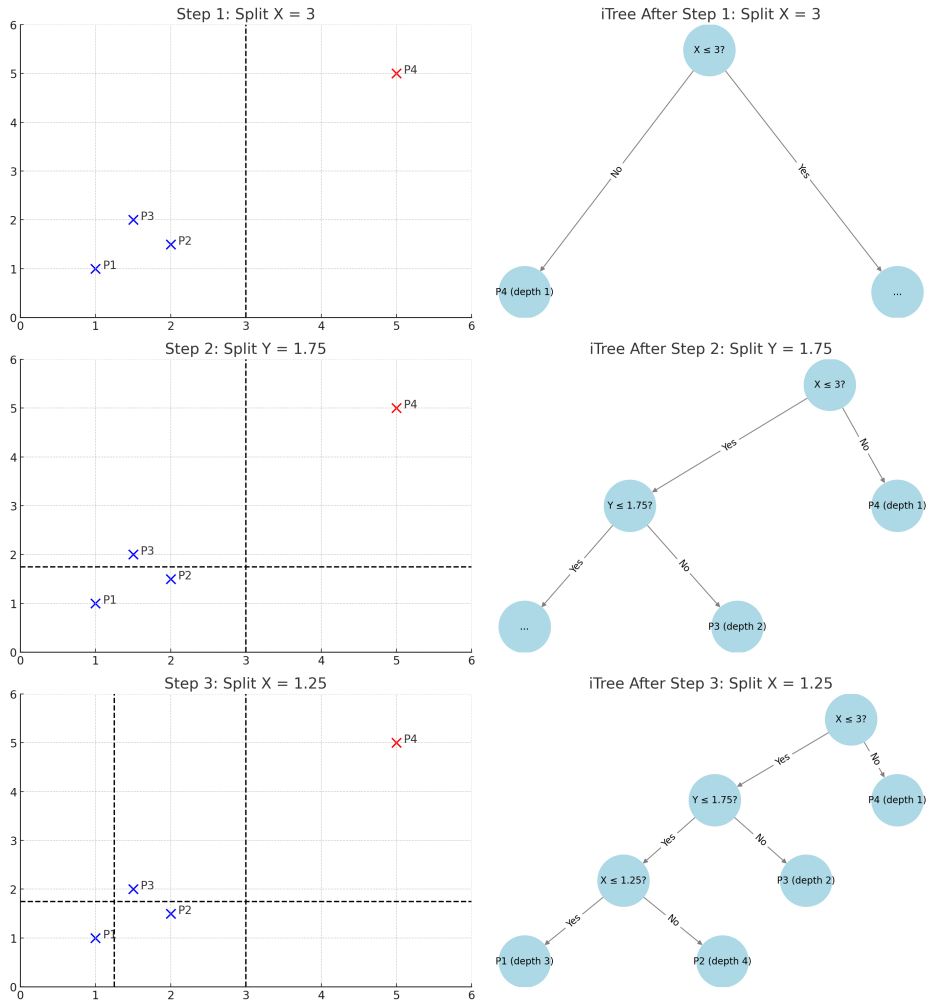


**Fig. 3** A combined 2D and tree-based view of how **Isolation Forest** isolates points through recursive random splits. On the left, splits are visualised in 2D feature space over three steps. Each vertical or horizontal line represents a randomly chosen split along the X or Y axis. On the right, the corresponding isolation tree (iTree) grows step-by-step, reflecting how each point is partitioned. Normal points (blue) require more splits to be isolated, leading to longer paths (e.g., P1 at depth 3, P2 at depth 4), while anomalies like P4 (red) are isolated early (depth 1), which is the key signal used in the anomaly score.

### 4.1.2 Algorithm

1. Randomly select a subset of data for each tree
2. For each tree:

- Start with all points
- Randomly select a feature
- Randomly select a split value between min and max of that feature
- Split the node into two branches (<= split, > split)
- Recursively continue until all points are isolated or max depth is reached

3. Compute the path length for each point (number of splits needed to isolate it)
4. Anomaly score is based on average path length-shorter paths indicate anomalies

A visual example of the algorithmic process is shown in Figure 3.
The anomaly score is defined as:

$$s(x) = 2^{-\frac{E[h(x)]}{c(n)}} \tag{4}$$

where $E[h(x)]$ is the average path length for point x across all trees, and $c(n)$ is the average path length of unsuccessful search in a binary search tree.

**Advantages:**

- Handles high-dimensional data well
- No distributional assumptions
- Linear time complexity O(n log n)
- Resilient to irrelevant features

**Disadvantages:**

- Performance depends on forest parameters (number of trees, subsample size)
- May produce false positives if data is highly imbalanced
- Requires setting contamination parameter (expected fraction of outliers)

```python
from sklearn.ensemble import IsolationForest
import pandas as pd
import numpy as np

# Load data (e.g., credit card transactions dataset)
data = pd.read_csv("creditcard.csv")
X = data.drop(columns=["Class"])  # features

# Create and fit the model
model = IsolationForest(contamination=0.001, random_state=42)
model.fit(X)

# Compute anomaly scores and predictions
scores = model.decision_function(X)
anomaly_scores = -scores  # invert: higher -> more anomalous
predictions = model.predict(X)  # +1 = normal, -1 = anomaly

# Get indices of flagged anomalies
anomalies_idx = np.where(predictions == -1)[0]
```

### 4.1.3 Support Vector Machine Basics

Support Vector Machines (SVMs) are powerful supervised machine learning algorithms that excel in classification and regression tasks. Developed at AT&T Bell Laboratories, SVMs are one of the most studied models in machine learning.

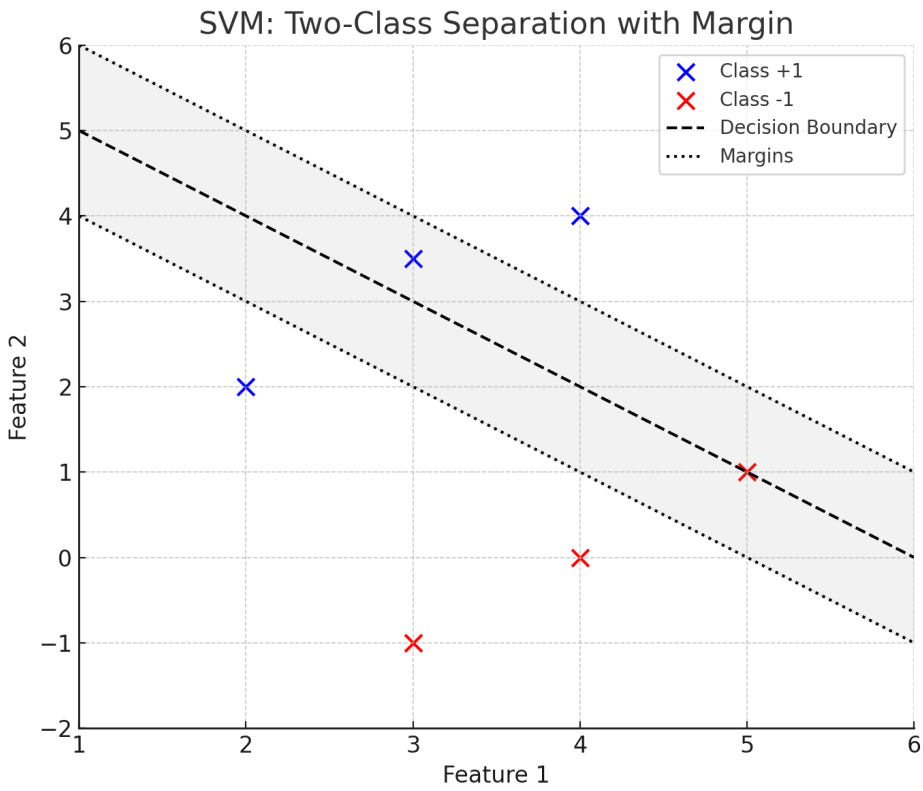


**Fig. 4** The plot demonstrates the fundamental concept of SVM by showing how it creates an optimal decision boundary (dashed line) between two classes (blue +1 and red -1) in a two-dimensional feature space. The dotted lines represent the margins, which SVM maximizes during training. This maximum-margin approach helps create a more robust classifier that generalizes better to unseen data. The data points closest to the margins (potential support vectors) are the most influential in determining the final position of the decision boundary.

Traditional SVMs work by finding an optimal hyperplane that separates different classes with maximum margin (as shown in Figure 4. The "support vectors" are those data points that lie closest to the decision boundary and directly influence its position. The algorithm aims to maximize the distance between the hyperplane and these support vectors.

Key strengths of SVMs include:

- Effectiveness in high-dimensional spaces
- Versatility through different kernel functions
- Memory efficiency (only a subset of training points are used as support vectors)
- Strong performance even with relatively small datasets
- Resilience to noise in the training data

For non-linear classification, SVMs employ the "kernel trick" to implicitly map input data into high-dimensional feature spaces where linear classification becomes

possible. This allows SVMs to create complex decision boundaries without explicitly calculating the transformation.

For anomaly detection purposes, we need a specialized variant of SVM that can work with mostly or exclusively normal data, which is where One-Class SVM becomes valuable.

## 4.2 One-Class SVM (OC-SVM)

One-Class SVM is an unsupervised algorithm that learns a decision boundary that encloses "normal" data points. It's a variant of the Support Vector Machine algorithm adapted for outlier detection.

### 4.2.1 Concept

One-Class SVM finds a hyperplane in a transformed feature space that separates most of the data points from the origin with maximum margin. The goal is to create a boundary that encloses most of the normal data in a "small" region, treating points outside this boundary as anomalies.

### 4.2.2 Algorithm

1. Map data points to a high-dimensional feature space using a kernel function (commonly RBF kernel)
2. Find a hyperplane that separates most points from the origin with maximum margin
3. The parameter $\nu$ controls the fraction of training examples allowed to be outside the boundary (outliers)
4. Points falling outside the learned boundary are flagged as anomalies

**Advantages:**

- Can capture complex decision boundaries through kernel trick
- Works well when data has a clear, compact boundary
- Better than density estimation when data doesn't fit a specific distribution

**Disadvantages:**

- Sensitive to parameter selection (especially kernel parameters)
- Doesn't scale well to large datasets (quadratic complexity)
- Limited effectiveness for very high-dimensional data
- Requires careful hyperparameter tuning

```
from sklearn.svm import OneClassSVM

# Using a subset of data as normal data (e.g., only setosa class)
normal_data = iris[iris['species'] == 'setosa'].drop(columns='species')
ocsvm = OneClassSVM(kernel="rbf", nu=0.05, gamma='scale')
ocsvm.fit(normal_data)

# Predict on new data (which may contain anomalies)
test_X = iris.drop(columns='species')
pred = ocsvm.predict(test_X)  # +1 = inlier, -1 = outlier
```

```
scores = ocsvm.decision_function(test_X)

# Evaluate: e.g., how many from other species are flagged as -1
outliers_idx = np.where(pred == -1)[0]
```

## 4.3 Autoencoders for Anomaly Detection

Autoencoders are neural networks designed to learn efficient representations of data (encoding) and then reconstruct the original data from this encoding. They can be used for anomaly detection by measuring reconstruction error.

### 4.3.1 Architecture

An autoencoder consists of (see Figure 5):

- **Encoder:** Compresses input data into a lower-dimensional representation
- **Bottleneck:** The compressed representation (latent space)
- **Decoder:** Reconstructs the input data from the compressed representation

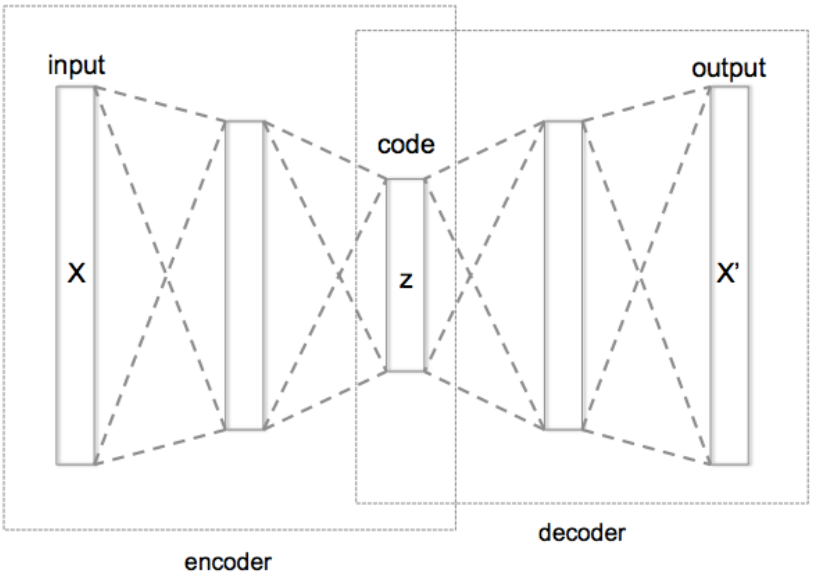


**Fig. 5** The diagram illustrates the symmetric hourglass structure of an autoencoder neural network with three key components: the encoder (left), the bottleneck layer labeled "code" or "z" (center), and the decoder (right). The encoder progressively compresses the high-dimensional input data X into a lower-dimensional latent representation z, while the decoder attempts to reconstruct the original input as X' from this compressed representation. In anomaly detection applications, autoencoders are trained on normal data only, learning efficient representations of normal patterns. When presented with anomalous data, the reconstruction error between X and X' becomes elevated, serving as an effective anomaly score. This unsupervised approach is particularly valuable when labeled anomaly examples are scarce or unknown.

### 4.3.2 Anomaly Detection Approach

1. Train the autoencoder on normal data
2. The autoencoder learns to reconstruct normal patterns efficiently
3. For new data points:
   - Calculate reconstruction error (difference between input and output)
   - High reconstruction error indicates anomaly

The reconstruction error is typically calculated using mean squared error:

$$E(x) = \frac{1}{d}\sum_{i=1}^{d}(x_i - x_i')^2 \tag{5}$$

where $d$ is the number of features, $x$ is the original input, and $x'$ is the reconstruction.

**Advantages:**

- Can capture complex non-linear relationships
- Adaptable to various data types (images, time series, tabular data)
- Can be combined with other deep learning techniques

**Disadvantages:**

- Requires sufficient data for training
- More complex to implement and tune than traditional methods
- Computationally intensive
- Black-box nature limits interpretability

```python
import numpy as np
from tensorflow import keras

# Define a simple autoencoder for tabular data
input_dim = X_train.shape[1]
encoder = keras.models.Sequential([
    keras.layers.Dense(16, activation='relu', input_shape=[input_dim]),
    keras.layers.Dense(3, activation='relu')  # bottleneck of size 3
])
decoder = keras.models.Sequential([
    keras.layers.Dense(16, activation='relu', input_shape=[3]),
    keras.layers.Dense(input_dim)
])
autoencoder = keras.models.Sequential([encoder, decoder])
autoencoder.compile(loss='mse', optimizer='adam')

# Train on normal data
autoencoder.fit(X_train_normal, X_train_normal, epochs=20, batch_size=32, verbose
    =0)

# Compute reconstruction errors on new data
X_pred = autoencoder.predict(X_test)
mse_errors = np.mean(np.square(X_test - X_pred), axis=1)
anomaly_flags = mse_errors > np.percentile(mse_errors, 95)  # flag top 5% errors
```

# 5 Evaluation Metrics for Anomaly Detection

Evaluating anomaly detection models presents unique challenges due to the inherent imbalance in the data (anomalies are rare). Standard accuracy metrics can be misleading.

## 5.1 Precision, Recall, and F1-Score

- **Precision** (Positive Predictive Value): Among points flagged as anomalies, how many are truly anomalies?

$$\text{Precision} = \frac{TP}{TP + FP} \tag{6}$$

  High precision means few false alarms.
- **Recall** (Detection Rate): Among all true anomalies, how many did we catch?

$$\text{Recall} = \frac{TP}{TP + FN} \tag{7}$$

High recall means catching most anomalies, but could include more false positives.

- **F1-Score**: Harmonic mean of Precision and Recall

$$\text{F1} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{8}$$

A balanced measure when we want a single metric to optimize.

## 5.2 ROC and PR Curves

Figure 6

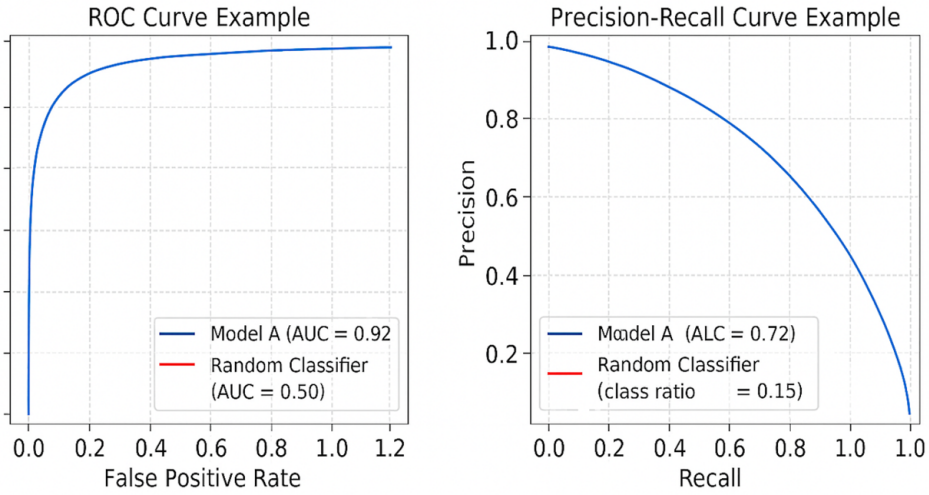


**Fig. 6** ROC and Precision-Recall curves for anomaly detection model evaluation. The left panel displays the Receiver Operating Characteristic (ROC) curve showing True Positive Rate versus False Positive Rate, with Model A achieving an excellent AUC score of 0.92 (blue line), substantially outperforming the random classifier baseline of 0.50 (red line). The right panel shows the corresponding Precision-Recall curve for the same model, with an AUC of 0.72 (blue line) compared to the random classifier baseline at the class ratio of 0.15 (red line). While both visualizations assess model performance across different thresholds, the Precision-Recall curve is particularly valuable for anomaly detection tasks with class imbalance, as it focuses directly on the trade-off between precision (minimizing false alarms) and recall (catching true anomalies). These complementary visualizations help data scientists select optimal decision thresholds based on application-specific requirements for anomaly detection systems.

- **ROC Curve** (Receiver Operating Characteristic): Plots True Positive Rate vs False Positive Rate at different threshold settings. The area under the ROC curve (ROC AUC) measures the model's ability to distinguish between classes.
- **Precision-Recall (PR) Curve**: Plots Precision vs Recall for varying thresholds. PR AUC focuses on the positive class (anomalies) and is more informative when classes are imbalanced.

**When to use which:**

- Use **ROC curve** when you care equally about positive and negative classes
- Use **PR curve** when positives (anomalies) are rare and of particular interest
- With high class imbalance (common in anomaly detection), PR curves are generally more informative

### 5.3 Choosing Thresholds

The choice of threshold is critical in anomaly detection-it determines the trade-off between false positives and false negatives:

- **Statistical approach:** Set threshold based on assumed distribution (e.g., $3\sigma$ for normal distribution)
- **Quantile-based:** Flag top n% of anomaly scores as outliers
- **Business-driven:** Set threshold based on cost/impact of false positives vs. false negatives
- **Validation-based:** If labeled data is available, choose threshold that maximizes F1-score or other metrics

## 6 Explainability in Anomaly Detection

As we deploy complex anomaly detection models, understanding *why* a point was flagged becomes crucial. This is where explainability techniques come in.

### 6.1 SHAP Values for Model Explainability

SHAP (SHapley Additive exPlanations) is a unified framework for explaining model outputs based on game theory. It treats each feature as a "player" in a game where the prediction is the payout.

#### 6.1.1 Key Properties of SHAP

- **Additivity:** Feature contributions sum to the difference between the model output and a baseline expectation
- **Fair credit allocation:** Features' influence is fairly considered regardless of the order they're introduced to the model

#### 6.1.2 SHAP for Anomaly Detection Models

Even for unsupervised anomaly detection models, SHAP values can be computed to explain why a specific instance was flagged:

- **Isolation Forest + SHAP:** Since IF is tree-based, TreeExplainer can compute exact Shapley values that show how each feature contributed to the anomaly score
- **Autoencoder + SHAP:** We can treat the reconstruction error as the output and use KernelSHAP to understand which features contributed most to the high reconstruction error

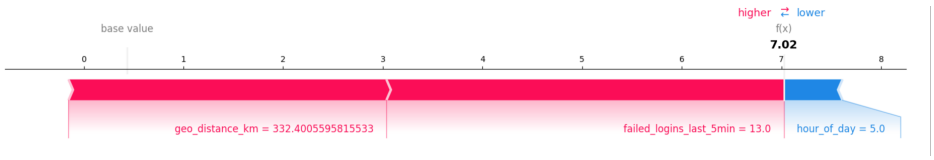


**Fig. 7** SHAP force plot visualizing feature contributions to an anomaly detection score. This plot shows how individual features influence the final anomaly score (f(x) = 7.02) relative to the base value. Two key factors strongly push the score higher (red): a substantial geographic distance (332.4 km) and multiple failed login attempts (13 in the last 5 minutes), suggesting potentially suspicious account access. The early morning hour (5:00) slightly reduces the score (blue), but its effect is minimal compared to the risk factors. This visualization effectively explains why the system flagged this particular event as anomalous, providing transparent reasoning for security analysts reviewing the alert.

### 6.1.3 Visualization Tools

- **Waterfall Plot:** Shows how each feature's SHAP value takes the model output from the baseline to the final prediction
- **Force Plot:** Visual "push-pull" format showing how features increase or decrease the anomaly score (as shown in Figure 7
- **Summary Plot:** Combines feature importance with feature effects across all instances
- **Bar Chart:** Shows average absolute SHAP value per feature to rank feature importance

```
import shap

# Example for Isolation Forest
explainer = shap.TreeExplainer(isolation_forest_model)
shap_values = explainer.shap_values(X_sample)
base_val = explainer.expected_value

# Display local explanation
shap.initjs()
shap.force_plot(base_val, shap_values, X_sample)

# For autoencoder, use KernelExplainer on anomaly score function
def anomaly_score(x):
    recon = autoencoder.predict(x)
    err = np.mean(np.square(x - recon), axis=1)
    return err

explainer_ae = shap.KernelExplainer(anomaly_score, X_background)
shap_vals_ae = explainer_ae.shap_values(X_sample)
```

# 7 Comparison of Methods

Different anomaly detection methods have different strengths and weaknesses. Here's a comparative analysis to help choose the right method for specific use cases:

**When to use which method:**

- **Z-Score:** When working with univariate data or when explainability is critical
- **Mahalanobis Distance:** When data is multivariate and approximately normal, and correlation structure matters

| Method | Interpretability | Scalability | Non-linear patterns | Multivariate |
|---|---|---|---|---|
| Z-Score | High | High | No | No |
| Mahalanobis | Medium | Medium | No | Yes |
| Isolation Forest | Medium | High | Yes | Yes |
| One-Class SVM | Low | Low | Yes | Yes |
| Autoencoder | Low | Medium | Yes | Yes |

**Table 2** Comparison of anomaly detection methods

- **Isolation Forest:** General-purpose method that works well on many datasets, especially higher-dimensional data
- **One-Class SVM:** When data has a clear boundary and you have enough data for proper parameter tuning
- **Autoencoders:** For complex data types (images, sequences) or when relationships between features are highly non-linear

# 8 Practical Implementation Example

Let's walk through a complete anomaly detection workflow using a credit card fraud detection example:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import IsolationForest
from sklearn.metrics import precision_recall_curve, auc
import matplotlib.pyplot as plt

# 1. Load and preprocess data
url = "https://storage.googleapis.com/download.tensorflow.org/data/creditcard.csv"
df = pd.read_csv(url)
print("Dataset shape:", df.shape)
print(df['Class'].value_counts(normalize=True).rename({0:'Legit',1:'Fraud'}))

# 2. Separate features/labels and scale data
X = df.drop(columns=['Class'])
y = df['Class']  # Target: 1 for fraud, 0 for normal
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 3. Apply Robust Z-score
from scipy.stats import median_abs_deviation

med = X.median()
mad = X.apply(median_abs_deviation).replace(0, 1e-6)
z_scores = ((X - med) / mad).abs()
df['z_sum'] = z_scores.sum(axis=1)

# 4. Threshold top 0.1%
thresh = np.quantile(df['z_sum'], 0.999)
pred_z = (df['z_sum'] > thresh).astype(int)

# 5. Evaluate Z-score method
precision, recall, _ = precision_recall_curve(y, df['z_sum'])
pr_auc_z = auc(recall, precision)
print(f"Robust Z-score PR-AUC: {pr_auc_z:.4f}")

```

```python
# 6. Apply Mahalanobis distance
from sklearn.covariance import LedoitWolf
from scipy.spatial.distance import mahalanobis

# Fit robust covariance on scaled data
cov_est = LedoitWolf().fit(X_scaled)
inv_cov = cov_est.get_precision()
mean_vec = X_scaled.mean(axis=0)

# Calculate Mahalanobis distances
mahal_d = np.array([mahalanobis(row, mean_vec, inv_cov) for row in X_scaled])
df['mahal'] = mahal_d

# 7. Evaluate Mahalanobis method
precision_m, recall_m, _ = precision_recall_curve(y, df['mahal'])
pr_auc_m = auc(recall_m, precision_m)
print(f"Mahalanobis PR-AUC: {pr_auc_m:.4f}")

# 8. Apply Isolation Forest
model_if = IsolationForest(contamination=0.001, random_state=42)
model_if.fit(X_scaled)
scores_if = -model_if.decision_function(X_scaled)  # higher = more anomalous

# 9. Evaluate Isolation Forest
precision_if, recall_if, _ = precision_recall_curve(y, scores_if)
pr_auc_if = auc(recall_if, precision_if)
print(f"Isolation Forest PR-AUC: {pr_auc_if:.4f}")

# 10. Plot PR curves for comparison
plt.figure(figsize=(10, 6))
plt.plot(recall, precision, label=f"Z-score (AUC={pr_auc_z:.4f})")
plt.plot(recall_m, precision_m, label=f"Mahalanobis (AUC={pr_auc_m:.4f})")
plt.plot(recall_if, precision_if, label=f"Isolation Forest (AUC={pr_auc_if:.4f})")
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curves")
plt.legend()
plt.grid(True)
plt.show()
```

# 9 Conclusion

Anomaly detection remains a challenging problem due to the evolving nature of anomalies, class imbalance, and the need for high-precision detectors in many real-world applications. The methods covered in these notes provide a foundation for addressing these challenges:

- **Statistical methods** like Z-Score and Mahalanobis Distance provide interpretable baseline approaches.
- **Machine learning methods** like Isolation Forest, One-Class SVM, and Autoencoders can capture more complex patterns.
- **Proper evaluation** is critical, with PR curves and PR-AUC often more informative than traditional metrics for imbalanced anomaly detection problems.
- **Explainability techniques** like SHAP help make black-box models more interpretable for end users.

The best approach often depends on the specific application context, data characteristics, and whether the priority is high recall (catching all anomalies) or high precision (minimizing false alarms).

# 10 Further Reading


- Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. ACM computing surveys, 41(3), 1-58.
- Liu, F. T., Ting, K. M., & Zhou, Z. H. (2008). Isolation forest. In 2008 IEEE International Conference on Data Mining.
- Schölkopf, B., Platt, J., Shawe-Taylor, J., Smola, A. J., & Williamson, R. C. (2001). Estimating the support of a high-dimensional distribution. Neural computation, 13(7), 1443-1471.
- Lundberg, S. M., & Lee, S. I. (2017). A unified approach to interpreting model predictions. Advances in neural information processing systems, 30.